Search-Based Information Systems Migration: Case Studies on Refactoring Model Transformations

by

Bader Alkhazi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Information Systems Engineering)
in the University of Michigan-Dearborn
2019

Doctoral Committee:

Associate Professor Marouane Kessentini, Chair
Professor Yubao Chen
Professor William Grosky
Professor Bruce Maxim
Professor Qiang Zhu

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

ATL Atlas Transformation Language

DSL Domain-Specific Language

EMO Multi-objective Optimization

ETL Epsilon Transformation Language

HOT High-order Transformation

IGD Inverted Generational Distance

IHV Hypervolume

JTL Janus Transformation Language

KML Keyhole Markup Language

MDE Model-Driven Engineering

MOP Multi-objective Optimization Problem

MTBE Model Transformation by Example

OCL Object Constraint Language

QVT Query/View/Transformation

R2ML REWERSE I1 Markup Language

SBSE Search-Based Software Engineering

SOP Single-objective Optimization Problem

SWRL Semantic Web Rule Language

UML Unified Modeling Language

VIATRA VIsual Automated model TRAnsformations

ABSTRACT

Information systems are built to last for decades; however, the reality suggests otherwise. Companies are often pushed to migrate or modernize their systems to reduce costs, meet new policies, improve the security, or to be competitive in the marketplace. Model-driven engineering (MDE) approaches are used in several successful projects to modernize or migrate systems. MDE raises the level of abstraction for complex systems by relying on models as first-class entities. These models are maintained and transformed using model transformations (MT), which are expressed by means of transformation rules to transform models from source to target meta-models.

The migration and modernization process for information systems may take years for large systems. Thus, many changes are going to be introduced to the transformations to reflect the new business requirements, fix bugs, or to meet the updated metamodel versions. Therefore, the quality of MT should be continually checked and improved during the evolution process to avoid future technical debts.

Most model transformation programs are written as one large module due to the lack of refactoring/modularization and regression testing tools support. In object-oriented systems, composition and modularization are used to tackle the issues of maintainability and testability. Moreover, refactoring is used to improve the non-functional attributes of the software, making it easier and faster for developers to work and manipulate the code. Thus, we proposed an intelligent computational search approach to automatically modularize model transformations. Furthermore, we took inspiration from a well-defined quality assessment

model for object-oriented design (QMOOD) to propose a quality assessment model for MT in particular. The results showed a 45% improvement in the developer's speed to detect or fix bugs, and developers made 40% less errors when performing a task with the modularized and optimized version.

Since refactoring operations changes the transformation, it is important to apply regression testing to check their correctness and robustness. Thus, we proposed a multi-objective test case selection technique to find the best trade-off between coverage and computational cost. Results showed a drastic speed-up of the testing process while still showing a good testing performance. The survey with practitioners highlighted the need of such maintenance and evolution framework to improve the quality and efficiency of the existing migration process.

CHAPTER I

Introduction

1.1 Research context: Model-Driven Engineering

Traditionally, models have been used in software development for documentation purposes Völter et al. (2013).The use of models was emphasized by the rise of Unified Modeling Language (UML) in the mid 90s. Such modeling languages helped create common guidelines for constructing and visualizing models. Models are effective and widely used in software engineering because they are providing abstraction for a real system or its environment, making it easier for different stakeholders (i.e. developers, investors, lawyers, upper management etc.) to comprehend the system and its internal workings from multiple points of view and at different levels of abstraction. Also, it is important to anticipate the consequences of any addition of a new feature or bug fixing at the model level before being propagated at the code level. It will, in the long run, save companies a huge amount of time and cost.

Model-Driven Engineering (MDE) is supporting the area of modeling since models are not only used for documentation purposes; they are also used to both simplify the design process and increase the productivity as they can be automatically implemented. Thus, models are as important as code in traditional object-oriented programming. In short, MDE is meant to decrease the accidental complexity Brooks and Kugler (1987) and increase the productivity by maximizing compatibility between systems, simplifying the process

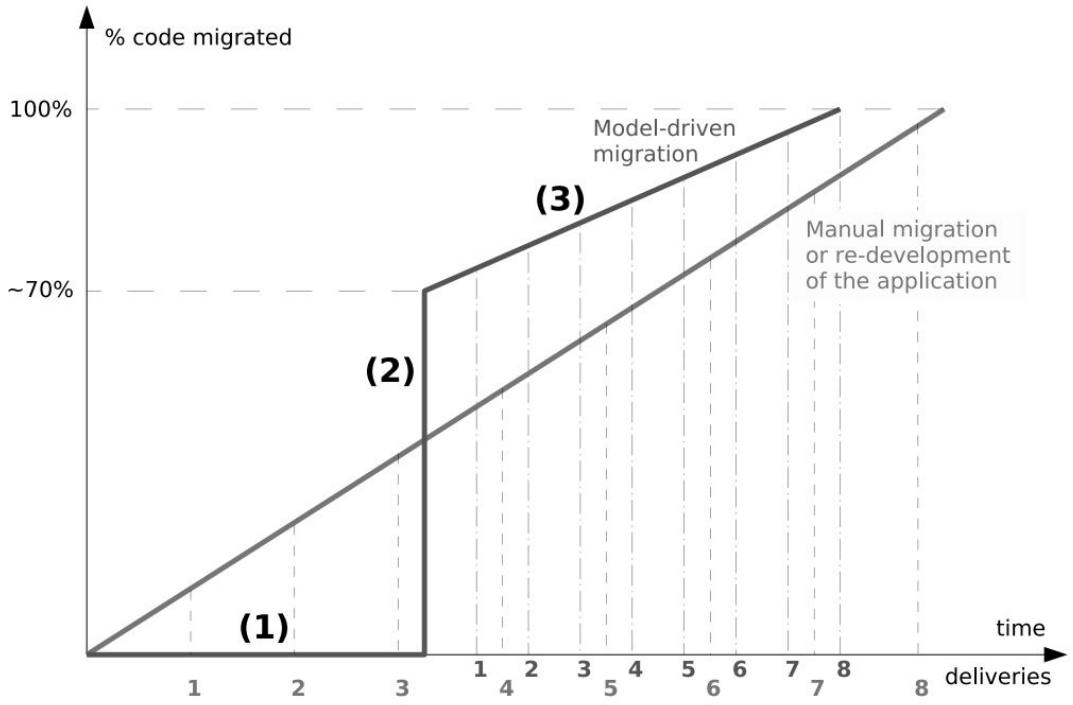


Figure 1.1: Migrated code percentage in function of time. Figure from Fleurey et al. (2007)

.

of design, and promoting communication between individuals and teams working on the system by performing model transformations. These model transformations are, in general, defined as a set of rules to transform models between a source and target languages.

Model transformation technologies, in practice, provide an efficient segue to automate the migration of legacy information systems to more modernized ones Fleurey et al. (2007); Reus et al. (2006); Bordbar et al. (2005); Mooij et al. (2015). In fact, the continuous evolution of software technologies requires an equivalent effort to keep the systems up-to-date in order to avoid security breaches and reduce future technical debt Kruchten et al. (2012), which may force companies to perform a full re-implementation of the system. Figure 1.1 Fleurey et al. (2007) shows a comparison between the time and deliverables rate for model-driven migration versus manual re-development of the whole application. For model-driven migration, the first phase is dedicated to planning and tool development or adjustment. Thus, no deliverables are expected during this phase; however, once the wheels

start to spin, the migration rate is accelerating rapidly.

What makes model-driven migration profitable, especially for substantially huge systems, is that a large portion of the legacy system is being transformed automatically using tools that can be reused with small adjustments for the different parts of the project or future ones. To do so, a set of rules of the transformation program needs to be modified (or added) to reflect the new specifications or requirements. Over the lifetime of a project, model transformation programs slowly become more complex, less readable, less comprehensible, and less maintainable, leading to a possible increase in the maintenance and testing activities in terms of time and cost Mohamed et al. (2009).

1.2 Problem Statement

The complexity and the maintainability of a software project are two important quality concepts of modern software projects, to keep the complexity low and the maintainability high, we need to continuously perform refactoring operations Brown et al. (1998) to improve the quality of the code without messing up their behavior.

Model transformation programs are designed differently Gniesser (2012), thus they can not be directly compared to object-oriented ones. Therefore, most of the existing literature for the latter needs to be slightly tweaked to accommodate the special properties of model transformations. This introduces the first problem one might face when working with model transformations or ATL in particular.

- Problem 1: Lack of methods and tools to refactor model transformations.

Many different model transformation languages emerged in the last decade such as Henshin Arendt et al. (2010), AGG Taentzer (2003), AToM3 De Lara and Vangheluwe (2002), e-Motions Rivera et al. (2009), VIATRA Csertán et al. (2002), QVT Greenyer and Kindler (2010), Kermeta Jézéquel et al. (2009), JTL Cicchetti et al. (2010), and ATL Jouault and Kurtev (2005). However, there is a gap in the literature to define refactoring methods and

tools for these model transformation programs. For instance, there are few studies to define some quality metrics or refactoring operations for model transformation languages, such as ATL van Amstel and van den Brand (2010, 2011), but it is up to the developer to locate the refactoring opportunities and apply them manually making most of the existing ATL transformations difficult to evolve, test and maintain.

- Problem 2: Lack of formal definition of ATL quality models.

ATL is relatively new language, formal definition of quality model is still missing. Therefore, it is hard to know what is the best quality metric or attribute to improve and what are the defects that could be found in an ATL program Wimmer et al. (2012).

- Problem 3: Prioritization of the refactoring solutions

When few design defects are detected then it is not a big deal. However, when the developer is working on big projects, the number of detected defects might be large. Thus, it is hard to fix them all especially when there is a time or resources constraint. For that reason, it is useful to find a way to prioritize the defects based on their importance.

- Problem 4: Refactoring recommendation based on the trade-off between cost and benefit.

It is important to optimize the process of recommending refactoring solutions for model transformation programs. Programmers are not interested to fix all the defects in the program. They are more interested to fix a maximum of relevant defects in a minimum amount of time or effort. However, the existing literature do not provide support to find a trade-off between maximizing the benefits of refactorings and minimizing its cost such the additional required testing effort.

- Problem 5: Performing an efficient regression testing

After making changes to a transformation, it is important to validate the changes by performing regression testing. However, it is not efficient to simply run the entire test suite, which in some cases could take weeks Elbaum et al. (2000). Since developers, in practice, have limited time and resources, they have to use other means to perform decent testing while keeping in mind the aforementioned limitations.

These observations were the main motivations of this thesis. In the following section, we give an overview of research directions to solve the problems mentioned above.

1.3 Contributions

To address the problems mentioned above, we propose the following solutions which are organized into three main contributions.

- Contribution 1: Modularization of Model Transformations

Modular design is a desirable property for model transformations as it can significantly improve their evolution, comprehensibility, maintainability, reusability, and thus, their overall quality. We introduced a new automated search-based software engineering approach based on NSGA-III to tackle the challenge of model transformation modularization. Specifically, we formulate the problem as a many-objective problem and use search-based algorithms to calculate a set of Pareto-optimal solutions based on four quality objectives: the number of modules in the transformation, the difference between the lowest and highest numbers of responsibilities in a module, the cohesion ratio and the coupling ratio.

- Contribution 2: Automatic Refactoring of ATL Model Transformations

We proposed an automated approach for refactoring ATL programs that find a trade-off between three different objectives. Our automated approach allows developers to benefit from search-based refactoring tools without manually identifying refactoring opportunities. To evaluate the effectiveness of our tool, we conducted a human study on a set of software

developers who evaluated the tool and compared it with random search and mono-objective formulation. Our evaluation results provide strong evidence that our tool improves the applicability and automation of existing ATL refactoring techniques.

- Contribution 3: Test case selection approach for ATL Model transformations

It is important to check the correctness of model transformation programs. Several approaches have been proposed to generate test cases for model transformations based on different coverage criteria (e.g., statements, rules, metamodel elements, etc.). However, the execution of a large number of test cases during the evolution of transformation programs is time-consuming and may include a lot of overlap between the test cases. Therefore, we propose a test cases selection approach for model transformations based on multi-objective search. We use the non-dominated sorting genetic algorithm (NSGA-II) to find the best trade-offs between two conflicting objectives: (1) maximize the coverage of rules and (2) minimize the execution time of the selected test cases. We validated our approach on several evolution cases of medium and large ATL programs.

1.4 Organization of the Dissertation

This thesis is organized as follows. Chapter II introduces the current state of the art and related work. Chapter III presents our approach to automatically modularize an ATL program. Chapter IV discusses our proposed approach to refactor ATL programs to improve certain quality attributes. Chapter V, describes our test case selection contribution. A summary and future research directions are presented in VI.

CHAPTER II

State of the Art

2.1 Introduction

In this chapter, we cover the necessary background information related to our work followed by an overview of existing work that intersects with ours.

2.2 Background

2.2.1 Migration of Information Systems

Information systems are crucial for the information flow within modern enterprises. The accumulative knowledge acquired over several years resides within these systems. Thus, stopping or freezing one of these systems could influence the business of the organization entirely. That could be a good reason for most enterprises to avoid making modifications to their legacy systems, unless they have no other options. Over the life time of the organization, these information systems slowly become very sensitive to changes, costly, and too difficult to replace Bisbal et al. (1999); Yeo (2002). However, being competitive in the global market requires continuous evolution and optimization of existing systems. Another motivation is the skill shortage Hainaut et al. (2008). Since systems in some cases last for decades, finding experts in an obsolete programming language or system might be a challenge. Moreover, systems will become a liability when vendors

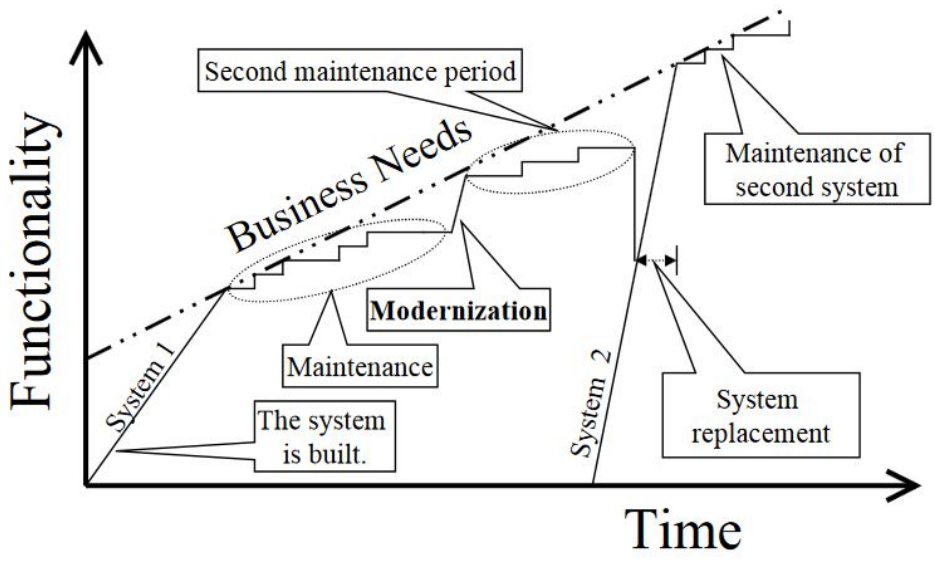


Figure 2.1: Information System Life Cycle. Figure from Comella-Dorda et al. (2000).

no longer support them or their underlying technology, making companies vulnerable to security breaches and countless performance issues. To overcome these challenges, there are mainly three system evolution activities Weiderman et al. (1997): maintenance, modernization, and replacement. Figure 2.1 Comella-Dorda et al. (2000) illustrates how these three activities are carried out at different times during the life time of the system.

The migration process is not a straightforward task, and many migration efforts fail Bisbal et al. (1997) along the way because of the complexity of the interconnected components. For instance, a migration project Bisbal et al. (1997) includes some or all of the following tasks: reverse engineering, business re-engineering, schema mapping, data transformation, application development, human computer-interface, testing, documentation, and training. For that reason, many migration projects are carried out in an incremental fashion over a long period of time which typically lasts five to ten years Brodie and Stonebraker (1995). The modernization tasks can be broadly categorized into black-box and white-box techniques Comella-Dorda et al. (2000). The former requires knowledge of the external interfaces of the legacy system and solutions are often based on wrapping, while the latter requires extensive understanding of the internals of the legacy system before re-engineering

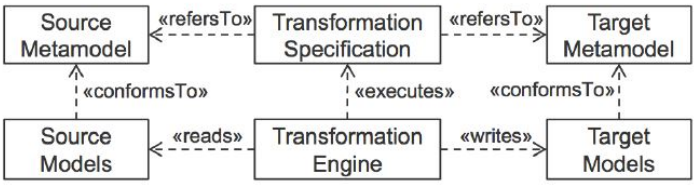


Figure 2.2: Model transformation pattern. Figure from Czarnecki and Helsen (2006).

it.

MDE techniques proved to be helpful in simplifying the migrating tasks Fleurey et al. (2007); Mooij et al. (2015); Selim et al. (2012) by raising the level of abstraction, automating many time consuming tasks, and reusing the generated transformation tools across different areas of the project, which are key advantages over traditional migration techniques.

In our work, we focus on refactoring and optimizing the transformation programs/tools used for migration to extend their reuse lifetime and to reduce the learning time required before using them. Also, simplifying the transformations reduces the errors that are highly probable with complex tasks like this.

2.2.2 Model Transformations

Model transformations are key techniques to automate migration tasks in MDE Harman (2007); Brambilla et al. (2012), by providing the essential mechanisms for manipulating models. In fact, these mechanisms allow to transform models into other models or into code, and are essential for synthesizing systems in MDE. In Czarnecki and Helsen (2006); Mens and Van Gorp (2006), an overview of transformation language concepts as well as a classification of different transformation types are presented. In this thesis, we focus on model-to-model (M2M) transformations. Generally speaking, a M2M transformation is a program executed by a transformation engine which takes one or more models as input to produce one or more models as output as is illustrated by the model transformation pattern in Figure 2.2. One important aspect is that model transformations are developed on the meta-model level and are thus reusable for all valid models.

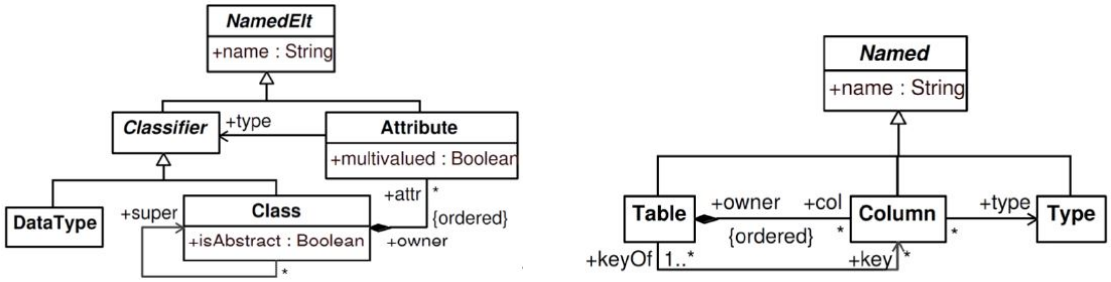
In our work, we focus on Atlas Transformation Language (ATL) since it has come to prominence in the MDE community. This success is due to the ATL's flexibility, support of the main metamodeling standards, usability that relies on tool integration with Eclipse, and a supportive development community Sendall and Kozaczynski (2003).

2.2.3 Meta-models

A meta-model is a model that specifies the concepts of a language, their relationships, and the structural rules to build valid models. As an example, the meta-model for UML is a model that contains the elements to describe UML models, such as Package, Class, Operation, Association, etc. In this way, each model is described in the language defined by its meta-model, so there should hold a conformance relation between a model and its meta-model. A meta-model is itself a model, and consequently, it is written in the language defined by its meta-metamodel. Meta-models allow to specify general-purpose languages as well as domain-specific languages (DSLs) Mernik et al. (2005). For realizing model transformations, there exist dedicated DSLs which are explained next.

2.2.4 Atlas Transformation Language (ATL)

ATL Jouault et al. (2008) is a hybrid model transformation language containing a mixture of declarative and imperative constructs. Listing II.1 shows an excerpt of an ATL transformation (from the ATL Zoo) that generates a relational schema from a class diagram. The input and output meta-models of this transformation are depicted in Figure 2.3. In this excerpt, we have included two declarative rules (so-called "matched rules"). The first rule, ClassAttribute2Column, takes elements of type Attribute whose type is a Class and whose are single-valued. These elements are transformed into elements of type Column. The value assigned to the name attribute is the same as the name of the Attribute element concatenated with "Id". The element referenced by the type relationship is retrieved by a helper function.



(a) Source Meta-model (b) Target Meta-model

Figure 2.3: Meta-models of the Class2Relational transformation

Listing II.1: Excerpt of the Class2Relational Transformation

```
module Class2Relation;
create OUT : Relational from IN : Class;
helper def : objectIdType : Relational!Type =
    Class!DataType.allInstances()
      -> select(e | e.name = 'Integer') -> first();
rule ClassAttribute2Column {
    from
      a : Class!Attribute (a.type.oclIsKindOf(Class!Class)
        and not a.multiValued)
    to
      foreignKey : Relational!Column ( name <- a.name + 'Id', type
        <- thisModule.objectIdType) }
rule Class2Table {
    from
      c : Class!Class
    to
      out : Relational!Table (
       name <- c.name,
       col <- Sequence {key}->union(c.attr
        ->select(e | not e.multiValued)),
       key <- Set {key}),
      key : Relational!Column ( name <- 'objectId',
        type <- thisModule.objectIdType) }
```

The second rule, Class2Table, takes an element of type Class as input and creates two elements: one of type Table and one of type Column. The name given to the Column is "objectId", and its type is also assigned with the helper. Regarding the Table, its key points to the new Column created. As for its col reference, it also points to the Column and to other elements. In order to retrieve these other elements, ATL performs a transparent lookup of output model elements for given input model elements. Thus, since such elements are of type Class!Attribute, it automatically retrieves the corresponding Relational!Column

elements that are created from the former elements.

The transparent lookup is performed in ATL by using an internal tracing mechanism. Thereby, every time a rule is executed, it creates a new trace and stores it in the internal trace model. This is graphically illustrated in Figure 2.4. In the left-hand side of the figure there is a sample input model, where elements are given an identifier (e.g., at1 and c1), that conforms to the meta-model shown in Figure 2.3. The right- hand side shows the model produced by the Class2Relation transformation and that conforms to the meta-model in Figure 2.3. In the central part of the figure contains the traces that have been produced from the execution of the two rules described. The traces keep track of which output elements are created from which input ones and by which rule. Thus, rule ClassAttribute2Column creates creates Trace 1 and rule Class2Table creates Trace 2. In order to properly set the col reference of the element t1, the engine searches in the trace model for the traces where c1.attr is the input element. It selects those traces of type Trace 1 and retrieves the elements created from such traces, co1 in our example, so they are selected as target for t1.col.

The elements created by rule Class2Table depend on the elements created by rule ClassAttribute2Column. For this reason, we say that the former rule has a dependency with the latter. Furthermore, both rules have a dependency with helper objectIdType. These dependencies are crucial for the approach we present in Chapter III. In fact, from any ATL transformation, we can obtain a dependency graph showing the dependencies among rules, between rules and helpers, and among helpers. For the given example, such graph is visualized in Figure 2.5.

2.2.5 Search Based Software Engineering (SBSE)

Search-based software engineering Harman (2007) is a field that applies search-based optimization techniques to software engineering problems. Search-based optimization techniques can be categorized as metaheuristic approaches that deal with large or even infinite search spaces in an efficient manner. These metaheuristic approaches are divided

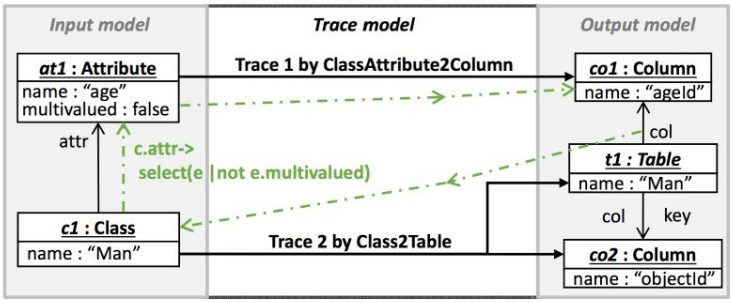


Figure 2.4: Representation of a sample transformation execution

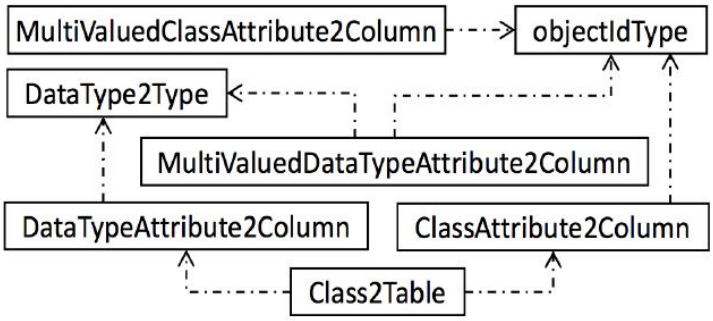


Figure 2.5: Class2Relational transformation elements dependencies

in two groups, namely local search methods and evolutionary algorithms. The aim of the former is to improve one single solution at a time, examples of algorithms of this type are Tabu Search Glover (1986) or Simulated Annealing Kirkpatrick et al. (1983). On the other hand, evolutionary algorithms Holland (1992) manage a set of solutions, referred to as population, at once. Some widely used evolutionary algorithms include NSGA-II Deb et al. (2002) and NSGA-III Deb and Jain (2014). For many real-world problems, multiple partially conflicting objectives need to be considered in order to find a set of desirable solutions. In fact, the field of Evolutionary Multi-objective Optimization (EMO) is considered one of the most active research areas in evolutionary computation Deb and Jain (2012). Especially in recent years, SBSE has also been applied successfully in the area of model and program transformations. Examples include the generation of model transformations from examples, the optimization of regression tests for model transformations, the detection of high-level model changes or the enhancement of the readability of source code for given

metrics.

Let us briefly discuss the need for applying metaheuristic search for the given problem. We can categorize the modularization problem as a problem related to the partitioning of a set of labeled elements into non-empty modules so that every element is included in exactly one module. The number of possible partitions, i.e., modules, is given by the Bell number (cf. Equation 2.1). The n_{th} of these numbers, B_n, counts the number of different ways a given set of n elements can be divided into modules. If there are no elements given (B_0), we can in theory produce exactly one partition (the empty set, $\emptyset$). The order of the modules as well as the order of the elements within a module does not need to be considered as this is not a semantic concern.

$$\begin{aligned} B_{n+1} &= \sum_{k=0}^{n} \binom{n}{k} B_k \\ B_0 &= 1 \end{aligned} \tag{2.1}$$

Considering the first Bell numbers (cf. sequence A0001101 in the OEIS online database for Integer sequences), we can see that the number of partition possibilities grows exponentially and is already quite high for a low number of elements. For example, an instance where you need to assign 15 elements to an unknown amount of modules already yields 1382958545 different possibilities.

2.2.5.1 Multi/Many-objective Problem

Formally, a multi-objective problem can be stated as follows Mkaouer et al. (2015):

$$\begin{cases} Min\, f(x) = [f_1(x), f_1(x), \ldots, f_M(x)]^T & \\ g_j(x) \geq 0 & j = 1, \ldots, P; \\ h_k(x) = 0 & k = 1, \ldots, Q; \\ x_i^L \leq x_i \leq x_i^U & i = 1, \ldots, n; \end{cases}$$

In this formulation, M is the number of objective functions, P is the number of inequality constraints, Q is the number of equality constraints, and x_i^L and x_i^U correspond to the lower and upper bounds of the decision variable x_i. A solution x consists of a set of decision variables which are optimized by the metaheuristic algorithm. A solution satisfying the $(P+Q)$ constraints is said to be feasible and the set of all feasible solutions defines the feasible search space denoted by Ω. The objective value for a specific solution is calculated by the provided objective function f_i and the aggregation of all objective functions defines the *fitness function* f. In this formulation, all objectives need to be minimized. Any objective that needs to be maximized can easily be turned into an objective that needs to be minimized by taking its negative value.

Recently, due to the limits of how many objectives different algorithms can handle, a distinction is made between multi-objective problems and many-objective problems. A many-objective problem, as opposed to a multi-objective problem, is a problem with at least four objectives, i.e., $M > 3$.

2.2.5.2 Pareto-optimal Solutions

Each of the objective functions defined for a multi-objective problem is evaluated for a concrete solution of the problem. By comparing the objective vectors of two solutions, we can determine whether one solution is 'better' than another with respect to these objectives. A common way to do this comparison is to aggregate all objective values of one solution and compare it with the the aggregated value of another solution. However, this is only

possible if all values are on the same scale. Alternatively, in SBSE, we often use the notion of Pareto optimality. As defined in (2.2) and (2.3) for strict inequality Harman (2007), under Pareto optimality, one solution is considered better than another if it is better according to at least one of the individual objective functions and no worse according to all the others. Using this definition, we can determine whether one solution is better than another, but not measure by 'how much' it is better.

$$F(\overline{x_1}) \geq F(\overline{x_2}) \Leftrightarrow \forall i \, f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \tag{2.2}$$

$$F(\overline{x_1}) > F(\overline{x_2}) \Leftrightarrow \forall i \, f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \wedge \exists i \, f_i(\overline{x_1}) > f_i(\overline{x_2}) \tag{2.3}$$

The algorithms used in SBSE apply the notion of Pareto optimality during the search to yield a set of non-dominated solutions. Each non-dominated solution can be viewed as an optimal trade-off between all objective functions where no solution in the set is better or worse than another solution in the set. It should be noted that in SBSE we assume that the 'true' Pareto front of a problem, i.e., the subset of values which are all Pareto optimal, is impossible to derive analytically and impractical to produce through exhaustive search Harman and Tratt (2007). Therefore, each set produced using metaheuristic search is an approximation to this, often unknown, 'true' Pareto front. Additional runs of such an algorithm may improve the approximation. In the remaining part of the thesis, we always refer to the Pareto front approximation.

2.2.5.3 NSGA-II

Most real world optimization problems encountered in practice involve multiple criteria to be considered simultaneously. These criteria, also called objectives, are often conflicting. Usually, there is no single solution that is optimal with respect to all these objectives at the same time, but rather many different designs exist which are incomparable per se. Conse-

quently, contrary to Single-objective Optimization Problems (SOP) where we look for the solution presenting the best performance, the resolution of a multi-objective optimization (MOP) yields a set of compromise solutions presenting the optimal trade-offs between the different objectives. When plotted in the objective space, the set of compromise solutions is called the Pareto front.

The resolution of a MOP yields a set of trade-off solutions, called Pareto optimal solutions or non-dominated solutions, and the image of this set in the objective space is called the Pareto front. Hence, the resolution of a MOP consists in approximating the whole Pareto front. We use one of the widely used multi-objective algorithms called NSGA-II Deb et al. (2002). NSGA-II is a powerful search method stimulated by natural selection that is inspired from the theory of Darwin. Hence, the basic idea of NSGA-II is to make a population of candidate solutions evolve toward the near-optimal solution in order to solve a multi-objective optimization problem. NSGA-II is designed to find a set of optimal solutions, called non-dominated solutions, also Pareto set. A non-dominated solution is the one which provides a suitable compromise between all objectives without degrading any of them. As described in Figure 2.6, the first step in NSGA-II is to create randomly a population P_0 of individuals encoded using a specific representation (line 1). Then, a child population Q_0 is generated from the population of parents P_0 using genetic operators such as crossover and mutation (line 2). Both populations are merged into an initial population R_0 of size N (line 5). As a consequence, NSGA-II starts by generating an initial population based on a specific representation that will be discussed later, using the exhaustive list of possible solutions than can be represented in a sequence or a vector.

To summarize, the main NSGA-II loop goal is to make a population of candidate solutions evolve toward the best sequence of solutions. During each iteration t, an offspring population Q_t is generated from a parent population P_t using genetic operators (selection, crossover and mutation). Then, Q_t and P_t are assembled in order to create a global population R_t. Then, each solution S_i in the population R_t is evaluated using our four fitness

1: Create an initial population P_0
2: Generate an offspring population Q_0
3: $t = 0$
4: **while** *stopping criteria not reached* **do**
5: $R_t = P_t \cup Q_i$
6: F = fast-non-dominated-sort (R_t)
7: $P_{t+1} = \phi$ and i=1;
8: **while** $|\mathrm{P}_{t+1}|+|\mathrm{F}_i| \leq N$ **do**
9: Apply crowding-distance-assignment(F_i);
10: $P_{t+1} = \mathrm{P}_{t+1} \cup F_i$;
11: $i = i + 1$
12: **end**
13: Sort($F_i, < n$)
14: $P_{t+1} = P_{t+1} \cup F_i[1 : (N\text{-}|\mathrm{P}_{t+1}|)]$;
15: Q_{t+1} = create-new-pop(P_{t+1});
16: $t = t + 1$;
17: **end**

Figure 2.6: NSGA-II overview

functions.

2.2.5.4 NSGA-III

NSGA-III is a very recent many-objective algorithm proposed by Deb et al. Deb and Jain (2014). The basic framework remains similar to the original NSGA-II algorithm with significant changes in its selection mechanism. Figure 2.8 gives the pseudo-code of the NSGA-III procedure for a particular generation t. First, the parent population P_t (of size N) is randomly initialized in the specified domain, and then the binary tournament selection, crossover and mutation operators are applied to create an offspring population Q_t. Thereafter, both populations are combined and sorted according to their domination level and the best N members are selected from the combined population to form the parent population for the next generation.

The fundamental difference between NSGA-II and NSGA-III lies in the way the niche preservation operation is performed. Unlike NSGA-II, NSGA-III starts with a set of reference points Z^r. After non-dominated sorting, all acceptable front members and the last

front F_l that could not be completely accepted are saved in a set S_t. Members in S_t/F_l are selected right away for the next generation. However, the remaining members are selected from F_l such that a desired diversity is maintained in the population. Original NSGA-II uses the crowding distance measure for selecting well-distributed set of points, however, in NSGA-III the supplied reference points (Z^r) are used to select these remaining members as described in Figure 2.7. To accomplish this, objective values and reference points are first normalized so that they have an identical range. Thereafter, orthogonal distance between a member in S_t and each of the reference lines (joining the ideal point and a reference point) is computed. The member is then associated with the reference point having the smallest orthogonal distance. Next, the niche count p for each reference point, defined as the number of members in S_t/F_l that are associated with the reference point, is computed for further processing. The reference point having the minimum niche count is identified and the member from the last front F_l that is associated with it is included in the final population. The niche count of the identified reference point is increased by one and the procedure is repeated to fill up population P_{t+1}.

It is worth noting that a reference point may have one or more population members associated with it or need not have any population member associated with it. Let us denote this niche count as p_j for the j-th reference point. We now devise a new niche-preserving

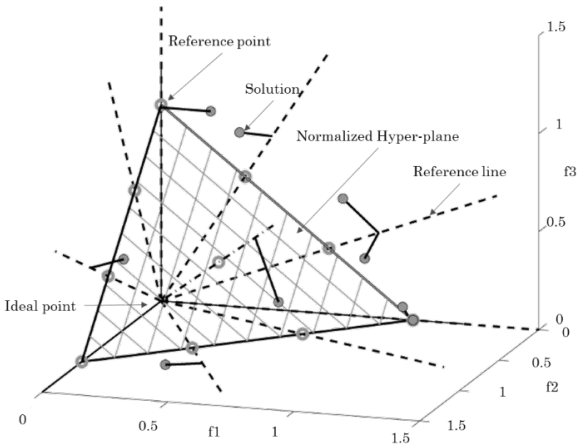


Figure 2.7: Normalized reference plane for a three-objective case

Input: H structured reference points Z_s, population P_t
Output: population P_{t+1}

$S_t \leftarrow \emptyset, i \leftarrow 1$
$Q_t \leftarrow$ VARIATION(P_t)
$R_t \leftarrow P_t \cup Q_t$
$(F_1, F_2, \ldots) \leftarrow$ NONDOMINATED_SORT(R_t)
repeat
    $S_t \leftarrow S_t \cup F_i$
    $i \leftarrow i + 1$
until $|S_t| \geq N$
$F_l \leftarrow F_i$ *// last front to be included*
if $|S_t| = N$ **then**
    $P_{t+1} \leftarrow S_t$
else
    $P_{t+1} \leftarrow \bigcup_{j=1}^{l-1} F_j$
    $K \leftarrow N - |P_{t+1}|$ *// number of points chosen from F_l*
    // normalize objectives and create reference set Z^r
    NORMALIZE(F^M, S_t, Z^r, Z^s)
    // Associate each member s of S_t with a reference point
    // $\pi(s)$: closest reference point
    // $d(s)$: distance between s and $\pi(s)$
    $[\pi(s), d(s)] \leftarrow$ ASSOCIATE(S_t, Z^r)
    // Compute niche count of a reference point $j \in Z^r$
    $p_j \leftarrow \sum_{s \in S_t / F_l} ((\pi(s) = j)\ ?\ 1 : 0)$
    // Choose K members one by one from F_l to construct P_{t+1}
    NICHING$(K, p_j, \pi(s), d(s), Z^r, F_l, P_{t+1})$
end if

Figure 2.8: NSGA-III procedure at generation t

operation as follows. First, we identify the reference point set $J_{\min} = \{j : \text{argmin}_j(p_j)\}$ having minimum p_j. In case of multiple such reference points, one ($j^* \in J_{\min}$) is chosen at random. If $p_{j*} = 0$ (meaning that there is no associated P_{t+1} member to the reference point j^*), two scenarios can occur. First, there exists one or more members in front F_l that are already associated with the reference point j^*. In this case, the one having the shortest perpendicular distance from the reference line is added to P_{t+1}. The count p_{j*} is then incremented by one. Second, the front F_l does not have any member associated with the reference point j^*. In this case, the reference point is excluded from further consideration for the current generation. In the event of $p_{j*} \geq 1$ (meaning that already one member associated with the reference point exists), a randomly chosen member, if exists, from front F_l that is associated with the reference point F_l is added to P_{t+1}. If such a member exists, the count p_{j*} is incremented by one. After p_j counts are updated, the procedure is repeated for a total of K times to increase the population size of P_{t+1} to N.

2.3 Related Work

2.3.1 Modularization of Model Transformations

Concerning the contribution of this area, we discuss main threads of related work. First, we summarize works considering modularization in the general field of software engineering. then, we discuss modularization support in different transformation languages.

2.3.1.1 Modularization in Software Engineering

In the last two decades, a large number of research has been proposed to support (semi-)automatic approaches to help software engineers maintain and extend existing systems. In fact, several studies addressed the problem of clustering to find the best decomposition of a system in terms of modules and not improving existing modularizations.

Wiggerts (1997) provides the theoretical background for the application of cluster analysis in systems remodularization. He discusses on how to establish similarity criteria between the entities to cluster and provide the summary of possible clustering algorithms to use in system remodularization. Later, Anquetil and Lethbridge (1999) use cohesion and coupling of modules within a decomposition to evaluate its quality. They tested some of the algorithms proposed by Wiggerts and compared their strengths and weaknesses when applied to system remodularization. Maqbool and Babri (2007) focus on the application of hierarchical clustering in the context of software architecture recovery and modularization. They investigate the measures to use in this domain, categorizing various similarity and distance measures into families according to their characteristics. A more recent work by Shtern and Tzerpos (2009) introduced a formal description template for software clustering algorithms. Based on this template, they proposed a novel method for the selection of a software clustering algorithm for specific needs, as well as a method for software clustering algorithm improvement.

There have been several developments in search-based approaches to support the automation of software modularization Kessentini et al. (2010, 2011); Ouni et al. (2015); Mkaouer et al. (2016); Kessentini et al. (2011); ben Fadhel et al. (2012); Kessentini et al. (2010); Boussaa et al. (2013); Kessentini et al. (2013); Ghannem et al. (2013); Ouni et al. (2017). Mancoridis et al. (1998) presented the first search-based approach to address the problem of software modularization using a single-objective approach. Their idea to identify the modularization of a software system is based on the use of the hill-climbing search heuristic to maximize cohesion and minimize coupling. The same technique has been also used by Mitchell and Mancoridis (2006, 2008) where the authors present Bunch, a tool supporting automatic system decomposition. Subsystem decomposition is performed by Bunch by partitioning a graph of entities and relations in a given source code. To evaluate the quality of the graph partition, a fitness function is used to find the trade-off between interconnectivity (i.e., dependencies between the modules of two distinct subsystems) and

intra-connectivity (i.e., dependencies between the modules of the same subsystem), to find out a satisfactory solution. Harman et al. (2002) use a genetic algorithm to improve the subsystem decomposition of a software system. The fitness function to maximize is defined using a combination of quality metrics, e.g., coupling, cohesion, and complexity. Similarly, Seng et al. (2005) treated the remodularization task as a single-objective optimization problem using genetic algorithm. The goal is to develop a methodology for object-oriented systems that, starting from an existing subsystem decomposition, determines a decomposition with better metric values and fewer violations of design principles. Abdeen et al. (2009) proposed a heuristic search-based approach for automatically optimizing (i.e., reducing) the dependencies between packages of a software system using simulated annealing. Their optimization technique is based on moving classes between packages. Mkaouer et al. (2015) proposed to remodularize object oriented software systems using many objective optimization with seven objectives based on structural metrics and history of changes at the code level. In this work, we are addressing a different problem since transformation programs are a set of rules and thus the used objectives are different from those that can be applied to JAVA programs. Praditwong et al. (2011) have recently formulated the software clustering problem as a multi-objective optimization problem. Their work aims at maximizing the modularization quality measurement, minimizing the inter-package dependencies, increasing intra-package dependencies, maximizing the number of clusters having similar sizes and minimizing the number of isolated clusters.

2.3.1.2 Modularization in Transformation Languages

The introduction of an explicit module concept going beyond rules as modularization concept Kurtev et al. (2007) has been considered in numerous transformation languages besides ATL to split up transformations into manageable size and scope. In the following, we shortly summarize module support in the imperative transformation language QVT-O OMG (2005), the declarative transformation languages TGGs Klar et al. (2007) and

QVT-R OMG (2005), and the hybrid transformation languages ETL Kolovos et al. (2008) and RubyTL Cuadrado and García Molina (2008); Cuadrado and Molina (2009). All these languages allow to import transformation definitions *statically* by means of explicit keywords. In QVT-O the keyword `extends` is provided, in order to base a new transformation on an existing one. In TGGs, it is possible to *merge* the rule types, i.e., the high-level correspondences from one transformation with those of a new one. In QVT-R it is possible to *import* a dependent transformation file and to *extend* a certain transformation of this file. ETL allows to *import* rules from a different transformation definition and so does RubyTL. Going one step further, in Cuadrado et al. (2014a) the authors propose transformation components which may be considered as transformation modules providing a more systematic description of their usage context such as required metamodel elements and configurations of a transformation's variability.

As for ATL, we are not aware of any automatic modularization support for transformation written in the aforementioned languages. In general, our proposed approach may be also applicable for other transformation languages providing a module concept. The only requirement is to find a transformation from the language to our modularization metamodel.

2.3.2 Automatic Refactoring of ATL Model Transformations

In this subsection, we initially discuss the different kinds of work regarding the evaluation of the quality of model transformations. Followed by discussing work specifically dealing with the refactoring of model transformations.

2.3.2.1 Quality of Model Transformations

Certainly, there is a substantial work regarding the quality of software, thus, we will only discuss the most closely related work especially those focusing on the quality of model transformations. The authors in Mohagheghi and Dehlen (2008) defined the characteristics of a quality framework for model-driven engineering (MDE). In Syriani and Gray (2012),

the authors discussed the various challenges that affects the quality of model transformations and proposed design patterns as well as quantitative metrics to assess the quality of transformations.

There is a decent amount of work evolving around the definition of metrics to assess the quality of model transformations in general or for a particular transformation language. Both Tolosa et al. (2011); Vignaga (2009) defined metrics for ATL, the latter though categorized ATL metrics into three main groups; rule, unit, and helper metrics. Similarly, in van Mf Marcel Amstel (2012) metrics were divided into four groups; rule, helper, dependency, and miscellaneous metrics. In van Amstel et al. (2009), however, the authors defined 27 quality metrics to measure six quality attributes: understandability, modularity, modifiability, reusability, completeness, and consistency. An emphasis of the need to relate metrics to quality attributes for ATL is detailed in van Amstel and van den Brand (2011) and the relation between performance and the size and complexity of input model was put under examination in van Amstel et al. (2011) in addition to a comparison between the performance of execution engines for three transformation languages: ATL, QVTr, and QVTo.

In Saeki and Kaiya (2007), the authors evaluated the external quality of transformation by applying metrics to both source and target models and evaluate the impact of the transformation on the model's quality. In Vieira and Ramalho (2014), a set of metrics were proposed to measure the change impact of ATL model transformations. Other contributions focused on a particular quality attribute; Rahimi and Lano (2011) identified the differences between transformation languages in terms of comprehensibility, whereas a set of metrics to measure the maintainability of QVT relational transformations has been proposed in Kapová et al. (2010). Finally, in Lano and Alfraihi (2018) the authors discuss the concept of technical depth for transformation languages by adapting quality flaws based on metrics for program code for different model transformation languages. Bad smells based on metrics for transformations written in the Epsilon Transformation Language (ETL) are

reported in Bonet et al. (2018).

2.3.2.2 Refactoring in Model Driven Engineering

With respect to the automatic exploration of model transformation refactorings opportunities, we discuss in this section related approaches. Compared to refactorings for different modeling languages, e.g., cf. Mohamed et al. (2009); Zhang et al. (2005); Misbhauddin and Alshayeb (2015); Mansoor et al. (2017, 2015); Ghannem et al. (2014), to mention just a few approaches and surveys, only few dedicated approaches have been developed for refactoring model transformations.

Dedicated refactoring operators for graph transformations have been presented in Taentzer et al. (2012) with a concentration on certain quality aspects such as changeability, conciseness, and comprehensibility. Henshin-specific model transformation bad smells which have an impact on the performance have been discussed in Tichy et al. (2013). The authors in Strüber et al. (2016) proposed clone detection and a merge-based rule refactoring approach for graph transformations which is related to inheritance-based ATL refactorings. However, the study focusses on the correctness of the merge-based rule refactorings, while we focus on the application of inheritance-based ATL refactorings with respect to quality metrics. Recently, Strüber et al. proposed variability-based model transformation approach, in order to tackle two issues; the maintainability and the performance of model transformations Strüber et al. (2018).

In Wimmer et al. (2012), the first refactoring catalogue for model transformations is presented which has been implemented for ATL. In our contributions, we build on the refactoring operations presented but go beyond the automation support initially proposed by Wimmer et al. While in the previous work, the refactoring process is semi-automated, meaning that the refactoring operations have to be explicitly triggered by the user, in our work we provide a fully automated approach for searching the refactoring space of a model transformation.

2.3.3 Test case selection for ATL Model transformations

We discuss main two threads of related work: (i) test case selection and (ii) testing in MDE.

2.3.3.1 Test Case Selection

There are three main test case management directions in the literature; test case prioritization, reduction, and selection. In this section, we consider test case selection work.

Early test case selection approaches using Integer Programming technique are presented in Fischer (1977); Fischer et al. (1981); Lee and He (1990). Fischer's algorithm was extended in Hartmann and Robson (1989, 1990) to be applied for C programs. Several test case selection techniques have been proposed afterwards including symbolic execution Yau and Kishimoto (1987), program slicing Agrawal et al. (1993); Bates and Horwitz (1993), data-flow analysis Gupta et al. (1992); Harrold and Souffa (1988); Taha et al. (1989), path analysis Benedusi et al. (1988), dependence and flow graphs Rothermel and Harrold (1993, 1994, 1997); Laski and Szermer (1992); Ball (1998). There are works that used heuristics to select test cases; In Biswas et al. (2009), the authors used genetic algorithms. In Mirarab et al. (2012); Kumar et al. (2012); Panichella et al. (2015); de Souza et al. (2014); Yoo and Harman (2007), the authors used multi-objective optimization techniques to select the appropriate cases. The following surveys discussed test case selection techniques in a broader manner Yoo and Harman (2012); Biswas et al. (2011); Rosero et al. (2016); Kazmi et al. (2017).

We are inline with test case selection approaches using multi-objective optimization techniques, but we apply them to a new kind of software artefact, namely model transformations.

2.3.3.2 Testing and Model Driven Engineering

Model transformation testing is considered as one of the main challenges in MDE and several papers discussed the need for a systematic validation of model transformations Bryant et al. (2011); Baudry et al. (2006, 2010); France and Rumpe (2007); Van Der Straeten et al. (2008); Fleurey et al. (2004). In Brottier et al. (2006), the authors presented an automatic approach to generate test model to satisfy certain criteria. Several papers took this direction afterwards such as the works of Fleurey et al. (2009); Lamari (2007); Ehrig et al. (2009); Almendros-Jiménez and Becerra-Terón (2016), while others used GA techniques to make the test case generation more efficient or relevant Jilani et al. (2014); Shelburg et al. (2013); Wang et al. (2013); Gomez et al. (2012); Sahin et al. (2015). In Finot et al. (2013), the authors proposed an approach to partially validate the output using expected target models. A black-box approach was proposed in Vallecillo et al. (2012), where Tracts are used to certify that test models works for the transformation. The authors in Rose and Poulding (2013) worked on producing smaller test suits by using probabilistic distributions for generating model samples, while the authors of Kessentini et al. (2011) discussed the definition of oracle function, and the automatic derivation of well-formedness rules is presented in Faunes et al. (2013a).

In the context of Model-Based Testing (MBT), a number of contributions was proposed to manage test suites. In Hemmati et al. (2010), the authors proposed similarity-based test case selection technique that uses genetic algorithms to minimize similarities between test cases. However, a test suite minimization framework is proposed in Farooq and Lam (2009), the authors formalized test case reduction as a combinatorial optimization problem. A test case prioritizing approach based on GA is proposed in Sharma et al. (2014). Covering all work is beyond the scope of this dissertation. Thus, we redirect to the survey by Wu et al. (2012).

2.3.4 Search-Based Software Engineering and Model Driven Engineering

SBSE has been used to tackle major MDE challenges for a while, as the associated search spaces have the potential to be very large, SBSE techniques are gaining popularity in both academia and industry since they are very beneficial in terms of finding good solutions in a reasonable time Boussaïd et al. (2017).

The idea of formalizing model transformations as a combinatorial optimization problem was first proposed in Kessentini et al. (2008), several work followed this initiative to use search-based optimization techniques with MT for different intents. The pioneer contributions applied the search-based techniques to the model transformation by example (MTBE) Varró (2006); Wimmer et al. (2007); Kappel et al. (2012) either to generate transformation rules Kessentini et al. (2010); Faunes et al. (2013b); Baki et al. (2014), recover transformation traces Saada et al. (2013), or to generate target models Kessentini et al. (2008, 2012). While MTBE approaches do not include the search for modularization when searching for model transformations, we discussed in chapter III an orthogonal problem, namely finding the best modules structure for a given transformation. In recent work, searching for good solutions in terms of transformation rule applications for a particular transformation in combination with a transformation context is investigated which is used for in chapter III as a prerequisite by reusing the MOMoT framework. There are two related approaches to MOMoT. First, Denil et al. (2014) proposes a strategy for integrating multiple single-solution search techniques directly into a model transformation approach. In particular, they apply exhaustive search, randomized search, Hill Climbing and Simulated Annealing. Second, Abdeen et al. (2014) also addresses the problem of finding optimal sequences of rule applications, but they deal with population-based search techniques. Thereby, this work is considered as a multi-objective exploration of graph transformation systems, where they apply NSGA-II Deb et al. (2002) to drive rule-based design space exploration. The MOMoT approach follows the same spirit as the previous mentioned two approaches, however, we aim to provide a loosely coupled framework which is not targeted

to a single optimization algorithm but allows to use the most appropriate one for a given transformation problem. In addition, the addressed problem in this work is different from finding the optimal sequence of transformation rules.

There is a number of studies that used an SBSE approach to detect or recommend model-refactoring opportunities; The authors in Jensen and Cheng (2010) proposed the REMODEL approach which uses both genetic programming and software metrics (based on QMOOD Bansiya and Davis (2002)) to generate design refactorings. The main two objectives of REMODEL are: (i) using QMOOD metrics to improve the design quality, and (ii) improving the maintainability of the software by introducing design patterns. A multi-level refactoring approach was presented in Moghadam and Cinnéide (2012); Moghadam (2011), where both the source code and the design are taken into consideration during the refactoring process. The developer initially tailors the desired target design that is better in terms of quality metrics or developer's perspective (or both) and the source code will then be refactored accordingly.

Model transformation testing is considered as one of the main challenges in MDE as detailed in Straeten et al. (2009); Bryant et al. (2011). The authors in Jilani et al. (2014); Shelburg et al. (2013); Wang et al. (2013); Gomez et al. (2012); Sahin et al. (2015) focused on test data generation. Others worked on minimizing the test suite Rose and Poulding (2013), the definition of oracle function Kessentini et al. (2011), and the automatic derivation of well-formedness rules Faunes et al. (2013a).

Besides testing and refactoring, the SBSE approach is extended to cover various MDE challenges such as model versioning or model merging Kessentini et al. (2013); Mansoor et al. (2015); Debreceni et al. (2016) and transformation rules orchestration Denil et al. (2014); Fleck et al. (2015); Gyapay et al. (2004); Mkaouer et al. (2013); Abdeen et al. (2014).

CHAPTER III

Modularization of Model Transformations

3.1 Introduction

Model-Driven Engineering (MDE) is a methodology that advocates the use of models throughout the software engineering life cycle to simplifying the design process and increase productivity. Model transformations are the cornerstone of MDE Czarnecki and Helsen (2006); Lúcio et al. (2014) as they provide the essential mechanisms for manipulating and transforming models. Most of these model transformations are expressed by means of rule-based languages. In MDE, models and model transformations are considered development artifacts which must be maintained and tested similar to source code in classical software engineering.

In object-oriented systems, composition and modularization are used to tackle the issues of maintainability and testability. Similar to any software systems, model transformation programs are iteratively refined, restructured, and evolved due to many reasons such as fixing bugs and adapting existing transformation rules to new metamodels version. Thus, it is critical to maintain a good quality and modularity of implemented model transformation programs to easily evolve them by quickly locating and fixing bugs, flexibility to update existing transformation rules, improving the execution performance, etc. Although language support for modularization in model transformation is emerging Kusel et al. (2015), it has not been studied in that much detail and has not been widely adopted. For instance, this is

also reflected by the current application of modularization within the ATL Transformation Zoo, which does not contain any modularized transformation Kusel et al. (2013). Thus, most of the existing ATL transformations are difficult to evolve, test and maintain.

We therefore propose an automatic approach to modularize large model transformations by splitting them into smaller model transformations that are reassembled when the transformation needs to be executed. Smaller transformations are more manageable in a sense that they can be understood more easily and therefore reduces the complexity of testability and maintainability. In particular, we focus on the modularization of ATL rules Kurtev et al. (2007) and helper functions. To the best of our knowledge, the problem of the automated modularization of model transformations beyond the rule concept has not been tackled so far.

The modularization of model transformation programs is a very subjective process and developers has to deal with different conflicting quality metrics to improve the modularity of the transformation rules. The critical question to answer is what is the best way to regroup together the rules that are semantically close by reducing the number of intra-calls between rules in different modules (coupling) and increasing the number of inter-calls between rules within the same module (cohesion). In such scenario, it is clear that both of these quality metrics are conflicting. To this end, we leverage the usage of search-based algorithms Harman (2007) to deal with the potentially large search space of modularization solutions. We measure the improvement of both testability and maintainability through common metrics such as coupling and cohesion, which have been adapted for model transformations and which are also used to guide the search process. Our many objective formulation, based on NSGA-III Deb and Jain (2014), finds a set of modularization solutions providing a good trade-off between four main conflicting objectives of cohesion, coupling, number of generated modules and the deviations between the size of these modules.

In our evaluation, we demonstrate the necessity for such approach by outperforming random search in all selected case studies (sanity check). Furthermore, we investigate

the quality of our generated solutions by determining their recall and precision based on comparison with other algorithms and manual solutions, ensuring quality of the produced results. We consider six different-sized transformations, of which five are available in the ATL Zoo and one has been created within our research group. Specifically, we show the configuration necessary to apply our modularization approach and how the different metrics of the selected transformations can be improved automatically. We found that, on average, the majority of recommended modules for all the ATL programs are considered correct with more than 84% of precision and 86% of recall when compared to manual solutions provided by active developers. The statistical analysis of our experiments over several runs shows that NSGA-III performed significantly better than multi-objective algorithms and random search. We were not able to compare with existing ATL modularization approaches since our study is the first to address this problem. The software developers considered in our experiments confirm the relevance of the recommended modularization solutions for several maintenance activities based on different scenarios and interviews. Therefore, the contributions of this section can be summarized as follows:

1. **Problem Formulation.** We define the problem of modularizing model transformations as a many-objective optimization problem.

2. **Problem Instantiation.** We instantiate our proposed problem formulation for the use case of ATL, which supports modularization through superimposition, and apply our approach on six different-sized ATL case studies and investigate their results.

3. **Solution Quality.** We demonstrate the quality of our approach by comparing the quality of the automatically generated solutions of NSGA-III with other multi-objective algorithms, one mono-objective algorithm and manually created solutions.

4. **Approach Usability.** The qualitative evaluation of the performed user study confirms the usefulness of the generated modularized solutions based on ATL.

3.2 Approach

This section presents our generic approach for tackling the model transformation modularization problem using SBSE techniques as well as how it is instantiated for ATL transformations.

3.2.1 Many-Objective Transformation Modularization

We formulate the model transformation modularization problem as a many-objective problem using Pareto optimality. For this, we need to specify three aspects. First, we need to formalize the model transformation domain in which transformations, both unmodularized and modularized, can be defined in a concise way. This formalization should be independent of any specific transformation language to make the approach more widely applicable and generic. Second, we need to provide modularization operations which can be used to convert an unmodularized transformation into a modularized one. Each modularization operation serves as decision variables in our solution. Finally, we need to specify a fitness function composed of a set of objective functions to evaluate the quality of our solutions and compare solutions among each other. We resort on well-established objectives from the software modularization domain and adapt them for the model transformation domain. An overview of our approach is depicted in Figure 3.1

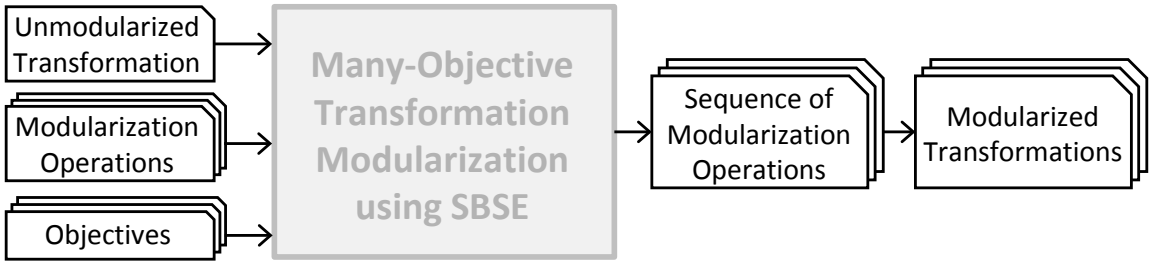


Figure 3.1: Overview of our modularization approach

3.2.1.1 Transformation Representation

We formalize the problem domain of transformation modularization in terms of a dedicated Modularization domain-specific language (DSL), whose abstract syntax is depicted in Figure 3.2

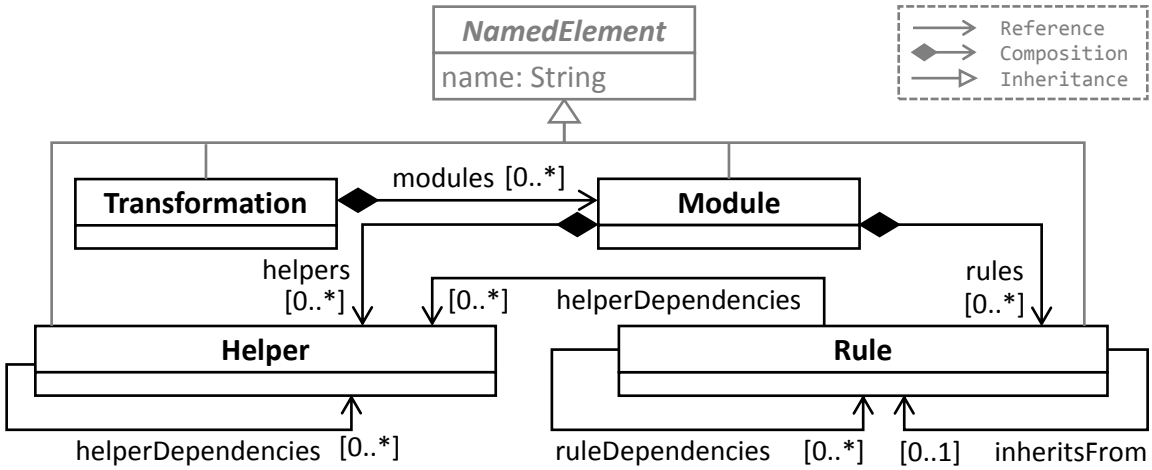


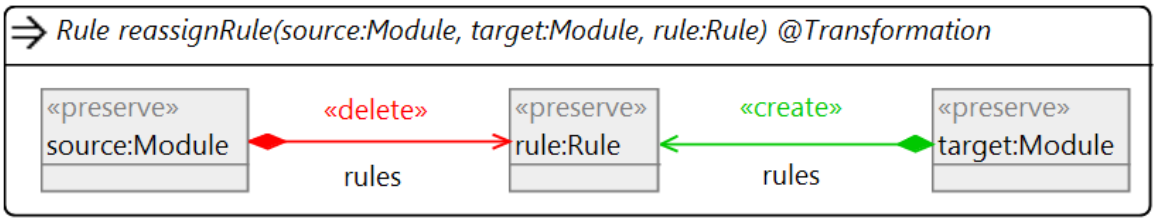
Figure 3.2: Modularization Metamodel

In this DSL, a transformation is composed of transformation rules and auxiliary functions which are named helpers. A transformation rule can inherit the functionality of another rule and may realize its own functionality by implicitly or explicitly invoking other transformation rules and helpers. A helper, in turn, provides a piece of executable code which can be called explicitly by any rule or helper. In our DSL, dependencies between rules and helpers are made explicit. The identification of the transformation elements, i.e., modules, helpers, and rules, is done through a unique name (cf. class NamedElement)

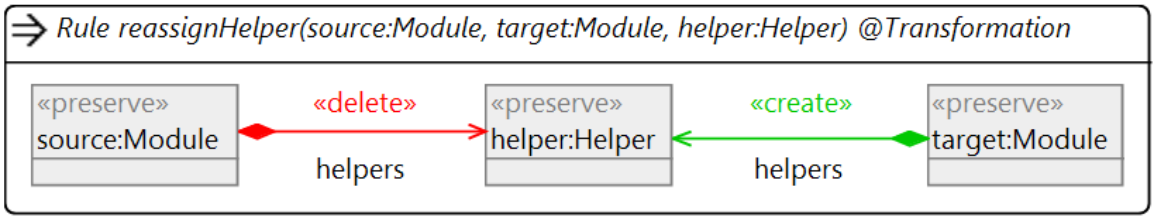
3.2.1.2 Solution Representation

A solution must be able to convert an unmodularized transformation into a transformation with modules, where the modules names are assigned random strings. To represent the process of this conversion, we consider a solution to be a vector of decision variables, where each decision variable in this vector corresponds to one application of a modularization operation. Initially, all rules and helpers of a transformation are contained in one

module. The modularization operations assign a rule or helper from one existing module to another existing or newly created module. Thus, the two rules depicted in Figure 3.3 are sufficient.



(a) *ReassignRule* Operation to move a rule to another module



(b) *ReassignHelper* Operation to move a helper to another module

Figure 3.3: Rules realizing the modularization operation

The parameters of these operations are the rule or helper that is shifted and the respective source and target module. We use an injective matching strategy, i.e., no two left-hand side nodes are assigned to the same model element, e.g., the source and target module parameter in the rules can not be assigned to the same module element. The bounds for helper and rule parameters are given by the set of rules and helpers in the unmodularized transformation. The bound for the module parameter is a set of modules, where there can be no more than n modules, where n is the total number of rules and helpers, i.e., the case in which all rules and helpers are in their own module. By having such a precise upper bound for the parameters, we can define the length of the solution vector as n, i.e., a solution where each helper and rule is assigned exactly once.

3.2.1.3 Solution Fitness

To evaluate the quality of the solutions, we consider four objective functions based on the resulting modularized transformation. An overview of these functions is depicted

in Table 3.1. Specifically, we aim to minimize the number of modules (NMT), minimize the difference between the lowest and the highest number of transformation artifact, i.e., rules and helpers, in a module (DIF), minimize the coupling ratio (COP) and maximize the cohesion ratio (COH). Since the multi-objective problem formulation only deals with minimization, in practice, we take the negated value of the cohesion ratio.

Table 3.1: Objective functions for a modularization solution

ID	Description	Type
NMT	Number of modules in the transformation	Min
DIF	Min/Max difference in transformation artifacts	Min
COH	Cohesion ratio (intra-module dependencies ratio)	Max
COP	Coupling ratio (inter-module dependencies ratio)	Min

The formulas for each objective function are given in (3.1) to (3.4) (adapted from Masoud and Jalili (2014)). In these formulas, M is the set of all modules and n is the number of all transformations elements. $U(m)$, $R(m)$ and $H(m)$ refer to all transformation elements, rules, and helpers of a given module m, respectively. Furthermore, $D_{RR}(m_i, m_j)$ in Brambilla et al. (2017), $D_{RH}(m_i, m_j)$ in Sendall and Kozaczynski (2003), and $D_{HH}(m_i, m_j)$ in Mens and Van Gorp (2006) specify the number of rule-to-rule, rule-to-helper and helper-to-helper dependencies between the given modules m_i and m_j, respectively; while $R_R(m_i, m_j)$ Brambilla et al. (2017), $R_H(m_i, m_j)$ Sendall and Kozaczynski (2003), and $H_H(m_i, m_j)$ Mens and Van Gorp (2006) which represent the ratio of rule-to-rule, rule-to-helper and helper-to-helper dependencies between the given modules mi and mj, respectively. It means that the total number of rules and helpers within such modules is taken into account for the calculation of the ratios (see denominator). Finally, $D(m_i, m_j)$ in Deb and Jain (2014) represents the total ratio of dependencies between the given modules mi and mj.

Please note that in the formulae for calculating coupling and cohesion ratios, a zero can be obtained in the denominators. In such cases, the result assigned to the division is zero. The reason for this is to favor those solutions that do not have modules with only one rule or only one helper. Specifically, it is not taken into account, i.e., not considered

for the calculation of the ratios, the dependencies that the only rule of a module has with itself, and the same thing for modules with only one helper (cf. equations 3.7 and 3.9 when $i = j$). This is used to optimize cohesion (which measures the dependencies within modules). It is not taken into account, either, the dependencies from rules to the only rule of a module, and those from helpers to the only helper of a module. On the contrary, those dependencies from the only rule in a module to other rules in modules with more than one rule, or from the only helper in a module to other helpers with more than one helper, are taken into account (cf. equations 3.7 and 3.9 when $i \neq j$). With this strategy, modules with only one rule or only one helper are partially taken into account for the calculation of coupling (which measures the dependencies among modules). Finally, when we have a module with a rule and a helper, the module has more than one artifact, so it is considered for the calculation of cohesion and coupling. This is the reason why equation 3.8 cannot have 0 in its denominator. Several different ways of defining coupling and cohesion in different contexts have been proposed, where we have followed the approach defined by some of them for solving the class responsibility assignment problem Masoud and Jalili (2014); Bowman et al. (2007, 2010) due to its similarity to our problem.

The underlying assumption to minimize the NMT objective is that a small number of modules is easier to comprehend and to maintain. Additionally, distributing the number of rules and helpers equally among the modules mitigates against small isolated clusters and tends to avoid larger modules, as also discussed by Praditwong et al. (2011). Furthermore, we optimize the coupling and cohesion ratios which are well-known metrics in clustering problems. Both coupling and cohesion ratios set the coupling, i.e., the number of inter-module dependencies, and the cohesion, i.e., the number of intra-module dependencies, in relation to all possible dependencies between the associated modules. Typically, a low coupling ratio is preferred as this indicates that each module covers separate functionality aspects. On the contrary, the cohesion within one module should be maximized to ensure that it does not contain rules or helpers which are not needed to fulfill its functionality.

$$\text{NMT} = |M| \tag{3.1}$$

$$\text{DIF} = max(|U(m)|) - min(|U(m)|), m \in M \tag{3.2}$$

$$\text{COH} = \sum_{m_i \in M} D(m_i, m_i) \tag{3.3}$$

$$\text{COP} = \sum_{\substack{m_i, m_j \in M \\ m_i \neq m_j}} D(m_i, m_j) \tag{3.4}$$

$$D(m_i, m_j) = R_R(m_i, m_j) + R_H(m_i, m_j) \tag{3.5}$$

$$+H_H(m_i, m_j) \tag{3.6}$$

$$R_R(m_i, m_j) = \frac{D_{RR}(m_i, m_j)}{|R(m_i)| \times |R(m_j) - 1|} \tag{3.7}$$

$$R_H(m_i, m_j) = \frac{D_{RH}(m_i, m_j)}{|R(m_i)| \times |H(m_j)|} \tag{3.8}$$

$$H_H(m_i, m_j) = \frac{D_{HH}(m_i, m_j)}{|H(m_i)| \times |H(m_j) - 1|} \tag{3.9}$$

Finally, to define the validity of our solutions, we enforce through constraints that all transformation artifacts need to be assigned to a module and that each module must contain at least one artifact. Solutions which do not fulfil these constraints are not part of the feasible search space, as defined in Section 2.2.5.

3.2.1.4 Change Operators

In each search algorithm, the variation operators play the key role of moving within the search space. Subsequently, we describe the two main used change operators of crossover and mutation

Crossover. When two parent individuals are selected, a random cut point is determined to split them into two sub-vectors. The crossover operator selects a random cut-point in the

interval [0, *minLength*] where *minLength* is the minimum length between the two parent chromosomes. Then, crossover swaps the sub-vectors from one parent to the other. Thus, each child combines information from both parents. This operator must enforce the maximum length limit constraint by eliminating randomly some modularization operations. For each crossover, two individuals are selected by applying the *SUS* selection. Even though individuals are selected, the crossover happens only with a certain probability. The crossover operator allows creating two offspring P1' and P2' from the two selected parents P1 and P2. It is defined as follows. A random position k is selected. The first *k* operations of P1 become the first *k* elements of P1'. Similarly, the first *k* operations of P2 become the first *k* operations of P2'.

Mutation. The mutation operator consists of randomly changing one or more dimensions (modularization operator) in the solution (vector). Given a selected individual, the mutation operator first randomly selects some positions in the vector representation of the individual. Then, the selected dimensions are replaced by other operation. When applying the mutation and crossover, we used also a repair operator to delete duplicated operations after applying the crossover and mutation operators.

3.2.2 Problem Instantiation: Many-Objective Modularization for ATL Transformations

We now instantiate our approach for ATL by performing three steps (cf. also Figure 3.4). First, we translate the given ATL transformation into our aforementioned modularization DSL. By doing this translation, we explicate the dependencies within the transformation. Second, we perform the modularization using the modularization operations and fitness function as described above. To modularize the transformation we apply our search-based framework MOMoT with the NSGA-III algorithm. Third, we translate the optimized modularization model with 1 to *n* modules to ATL files, i.e., transformation modules and libraries. In the following, these steps are explained in detail.

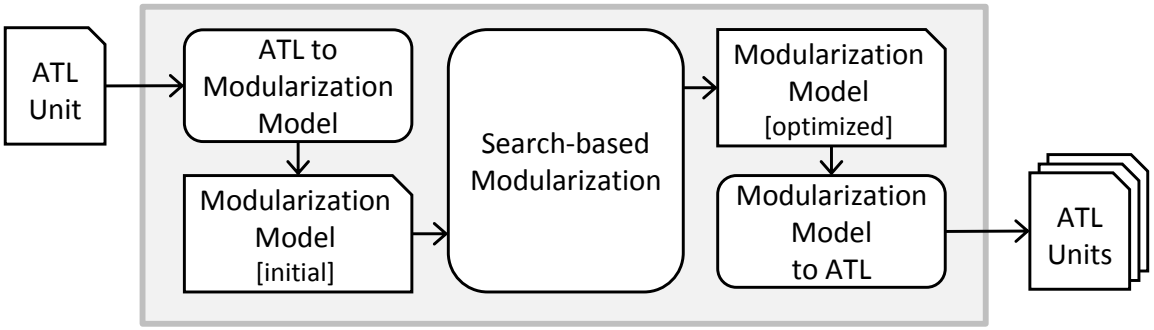


Figure 3.4: Overview of the ATL modularization approach

3.2.2.1 Converting ATL Transformations to Modularization Models

ATL provides explicit concepts for modules, rules, and helpers, thus they can be mapped directly to the modularization DSL. However, the extraction of the dependencies between transformation elements is more challenging. In fact, we can distinguish between implicit dependencies based on automatic resolution of matched rules and explicit dependencies based on explicit invocations of *lazy rules*, *called rules*, and *helpers* Troya et al. (2016). While explicit invocations are directly manifested in the syntax of ATL transformations, additional reasoning is needed to statically identify the dependencies among matched rules.

We have automated the way of producing the dependency model with a high-order transformation (HOT) Tisi et al. (2009) that takes the transformation injected into a model-based representation as well as the metamodels of the transformation as input and statically infers information about types in the transformation. As mentioned, the most challenging task is to extract the dependencies among matched rules. This is done by the HOT in two steps. First, the types of the rules are statically extracted, i.e., the classes of the metamodels that participate in the rules. This means that it needs to extract the types of the elements that are reached by OCL navigations Burgueño et al. (2015). In the second step, after the types of the different parts of the rules are extracted, we can trivially calculate the dependencies. Thus, we consider that a rule, R1, depends on another rule, R2, if the intersection of the types of the bindings of R1 with the ones of the *InPatternElements* of R2 is not empty. For instance, in Listing II.1, rule Class2Table depends on *ClassAttribute2Column* since

some of the objects retrieved in the second binding of the former rule, c.attr → select(e | not e.multivalued), have the same type as the one specified in the *InPatternElement* of the latter rule, i.e., *Class!Attribute* where the multivalued attribute is set to false. For more information on the dependency types that can take place in ATL transformations and how we statically obtain the types of the elements appearing in the rules, we kindly refer the interested reader to Troya et al. (2016). The model produced by the HOT conforms to our modularization DSL and is composed of one module and the same number of rules and helpers as the ATL transformation contains. However, all dependencies between rules and helpers are explicitly declared in this model.

3.2.2.2 Search-based Modularization

Having the modularization model at hand, we apply our search-based framework MOMoT Fleck et al. (2015, 2016a), to find the Pareto-optimal module structure. MOMoT[1] is a task- and algorithm-agnostic approach that combines SBSE and MDE. It has been developed in previous work Fleck et al. (2015) and builds upon Henshin[2] Arendt et al. (2010) to define model transformations and the MOEA framework[3] to provide optimization techniques. In MOMoT, DSLs (i.e., metamodels) are used to model the problem domain and create problem instances (i.e., models), while model transformations are used to manipulate those instances. The orchestration of those model transformations, i.e., the order in which the transformation rules are applied and how those rules need to be configured, is derived by using different heuristic search algorithms which are guided by the effect the transformations have on the given objectives. In order to apply MOMoT for the given problem, we need to specify the necessary input. The instance model is the modularization model obtained in the previous step, while the rules are the modularization operations defined as Henshin rules shown in Figure 3.3 deciding which elements go into which module, we have

[1]MOMoT: `http://martin-fleck.github.io/momot/`
[2]Henshin: `http://www.eclipse.org/henshin`
[3]MOEA Framework: `http://www.moeaframework.org`

to create modules. Thereby, we produce input models with different number of modules in the range of [1, n], where n is the number of rules and helpers combined. This covers both the case that all rules and helpers are in one single module and the case in which each helper and rule is in its own module. The objectives and constraints described in Section 3.2.1.3 are implemented as Java methods to provide the fitness function for MOMoT. Finally, we need to select an algorithm to perform the search and optimization process. For this task, we choose the NSGA-III, as it is know to be able to manage many-objective problems

3.2.2.3 Converting Modularization Models to ATL Transformations

After retrieving the solutions produced by MOMoT, each module is translated to an ATL unit, resulting in n ATL files. Modules solely containing helpers are translated to libraries and modules which have at least one rule are translated into normal ATL modules. As mentioned in Section 3.2.1.2, the names given to the different ATL files are random strings. Of course, users of our tool may decide to change these names and add more meaningful names after the modularization process finishes. The whole transformation is again implemented as a HOT

3.3 Evaluation

In order to evaluate our approach by instantiating it for ATL, we answer four research questions regarding the need for such an approach, the correctness of the solutions and the usability of the modularization results. In the next sub-sections, we describe our research questions and the seven case studies and metrics we use to answer these questions. Finally, we discuss the answer to each research question and overall threats to validity of our approach.

3.3.1 Research Questions

Our study addresses the following four research questions. With these questions, we aim to justify the use of our metaheuristic approach, compare the use of NSGA-III with other algorithms (Random Search, ε-MOEA and SPEA2), argue about the correctness of the modularization results retrieved from our approach and validate the usability of our approach for software engineers in a real-world setting.

RQ1: Search Validation: Do we need an intelligent search for the transformation modularization problem?

To validate the problem formulation of our modularization approach, we compared our many-objective formulation with Random Search (RS). If Random Search outperforms a guided search method, we can conclude that our problem formulation is not adequate Harman et al. (2012); Arcuri and Briand (2014); Harman et al. (2012). Since outperforming a random search is not sufficient, the question is related to the performance of NSGA-III, and a comparison with other multi-objective algorithms.

RQ2 Search Quality: How does the proposed many-objective approach based on NSGA-III perform compared to other multi-objective algorithms?

Our proposal is the first work that tackles the modularization of model transformation programs. Thus, our comparison with the state of the art is limited to other multi-objective algorithms using the same formulation. We elect two algorithms, ε-MOEA and SPEA2, to do this comparison. We have also compared the different algorithms when answering the next questions.

RQ3.1 Solution Correctness: How close are the solutions generated by our approach to solutions a software engineer would develop?

To see whether our approach produces sufficiently good results, we compare our generated set of solutions with a set of manually created solutions by developers based on precision and recall.

RQ3.2 Solution Correctness: How good are the solutions of our approach based

on manual inspection?

While comparison with manually created solutions yields some insight into the correctness of our solutions, good solutions which have an unsuspected structure would be ignored. In fact, there is no unique best modularization solution, thus a deviation with the expected manually created solutions could be just another good possibility to modularize the ATL program. Therefore, we perform a user study in order to evaluate the coherence of our generated solutions by manually inspecting them.

The goal of the following two questions is to evaluate the usefulness and the effectiveness of our modularization tool in practice. We conducted a non-subjective evaluation with potential developers who can use our tool related to the relevance of our approach for software engineers:

RQ4.1 Approach Usability: How useful are modularizations when identifying or fixing bugs in a transformation?

Identifying and fixing bugs in a transformation is a common task in MDE, where transformations are seen as development artifacts. As such, they might be developed incrementally and by different people, leading to potential bugs in the transformation. We investigate whether the performance of this task can be improved through modularization.

RQ4.2 Approach Usability: How useful are modularizations when adapting transformation rules due to metamodel changes?

During the lifecycle of an application, the input and/or output metamodel of a model transformation might change, e.g., due to new releases of the input or output language. When the input or output metamodel changes, the model transformation has to be adapted accordingly not to alter the system semantics. We evaluate whether the adaptation of the transformation rules can be improved through modularization.

In order to answer these research questions we perform experiments to extract several metrics using seven case studies and two user studies. The complete experimental setup is summarized in the next subsection

3.3.2 Experimental Setup

3.3.2.1 Case Studies

Our research questions are evaluated using the following seven case studies. Each case study consists of one model transformation and all the necessary artifacts to execute the transformation, i.e., the input and output metamodels and a sample input model. Most of the case studies have been taken from the ATL Zoo, a repository where developers can publish and describe their ATL transformations.

CS1 **Ecore2Maude:** This transformation takes an Ecore metamodel as input and generates a Maude specification. Maude Clavel et al. (2007) is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications.

CS2 **OCL2R2ML:** This transformation takes OCL models as input and produces R2ML (REWERSE I1 Markup Language) models as output. Details about this transformation are described in Milanović et al. (2007).

CS3 **R2ML2RDM:** This transformation is part of the sequence of transformations to convert OCL models into SWRL (Semantic Web Rule Language) rules Milanović (2007). In this process, the selected transformation takes a R2ML model and obtains an RDM model that represents the abstract syntax for the SWRL language.

CS4 **XHTML2XML:** This transformation receives XHTML models conforming to the XHTML language specification version 1.1 as input and converts them into XML models consisting of elements and attributes.

CS5 **XML2Ant:** This transformation is the first step to convert Ant to Maven. It acts as an injector to obtain an xmi file corresponding to the Ant metamodel from an XML file.

CS6 XML2KML: This transformation is the main part of the KML (Keyhole Markup Language) injector, i.e., the transformation from a KML file to a KML model. Before running the transformation, the KML file is renamed to XML and the KML tag is deleted. KML is an XML notation for expressing geographic annotation and visualization within Internet-based, two-dimensional maps and three-dimensional Earth browsers.

CS7 XML2MySQL: This transformation is the first step of the MySQL to KM3 transformation scenario, which translates XML representations used to encode the structure of domain models into actual MySQL representations.

We have selected these case studies due to their difference in size, structure and number of dependencies among their transformation artifacts, i.e., rules and helpers. Table 3.2 summarizes the number of rules (R), the number of helpers (H), the number of dependencies between rules (D_{RR}), the number of dependencies between rules and helpers (D_{RH}) and the number of dependencies between helpers (D_{HH}) for each case study.

ID	Name	R	H	D_{RR}	D_{RH}	D_{HH}
CS1	Ecore2Maude	40	40	27	202	23
CS2	OCL2R2ML	37	11	54	25	8
CS3	R2ML2RDM	58	31	137	68	17
CS4	XHTML2XML	31	0	59	0	0
CS5	XML2Ant	29	7	28	33	5
CS6	XML2KML	84	5	0	85	2
CS7	XML2MySQL	6	10	5	16	5

Table 3.2: Size and structure of all case studies.

3.3.2.2 Evaluation Metrics

To answer our research questions, we use several metrics depending on the nature of the research question.

Search Performance Metrics: In order to evaluate research questions RQ1 and RQ2, we compare the results of NSGA-III with Random Search, ε-MOEA and SPEA2 based on Hypervolume and Inverted Generational Distance for all case studies.

- Hypervolume (IHV) corresponds to the proportion of the objective space that is dominated by the Pareto front approximation returned by the algorithm and delimited by a reference point. The larger the proportion, the better the algorithm performs. It is interesting to note that this indicator is Pareto dominance compliant and can capture both the convergence and the diversity of the solutions. Therefore, IHV is a common indicator used when comparing different search-based algorithms.

- Inverse generational distance (IGD) is a convergence measure that corresponds to the average Euclidean distance between the Pareto front approximation produced by the algorithm and the reference front. We can calculate the distance between these two fronts in an M-objective space as the average M-dimensional Euclidean distance between each solution in the approximation and its nearest neighbor in the reference front. Better convergence is indicated by lower values.

Solution Correctness Metrics: In order to evaluate research questions RQ3.1 and RQ3.2, we inspect our solutions with respect to manual solutions and as standalone solutions. Ideally, we would compare our solutions with ATL modularized solutions. However, as mentioned in Section 3.1, there is no single modularized solution in the ATL Zoo, what made us follow this approach. Specifically, for RQ3.1, we automatically calculate the precision (PR) and recall (RE) of our generated solutions given a set of manual solutions. Since there are many different possible manual solutions, only the best precision and recall value are taken into account, as it is sufficient to conform to at least one manual solution. For answering RQ3.2 with the manual validation, we asked groups of potential users to evaluate, manually, whether the suggested solutions are feasible and make sense given the transformation. We therefore define the manual precision (MP) metric

- To automatically compute precision (PR) and recall (RE), we extract pair-wise the true-positive values (TP), false-positive values (FP) and false-negative values (FN) of each module. TPs are transformation artifacts which are in the same module and should be, FPs are artifacts which are in the same module but should not be and FNs are artifacts which should be together in a module but are not.

$$\text{PR} = \frac{TP}{TP+FP} \in [0,1]$$
$$\text{RE} = \frac{TP}{TP+FN} \in [0,1]$$

  Higher precision and recall rates correspond to results that are closer to the expected solutions and are therefore desired.

- Manual precision (MP) corresponds to the number of transformation artifacts, i.e., rules and helpers, which are modularized meaningfully, over the total number of transformation artifacts. MP is given by the following equation

$$\text{MP} = \frac{|\text{coherent artifacts}|}{|\text{all artifacts}|} \in [0,1]$$

  A higher manual precision indicates more coherent solutions and therefore solutions that are closer to what a user might expect.

For each case study and algorithm, we select one solution using a knee point strategy Bechikh et al. (2011). The knee point corresponds to the solution with the maximal trade-off between all fitness functions, i.e., a vector of the best objective values for all solutions. In order to find the maximal tradeoff, we use the trade-off worthiness metric proposed by Rachmawati and Srinivasan Rachmawati and Srinivasan (2009) to evaluate the worthiness of each solution in terms of objective value compromise. The solution nearest to the knee point is then selected and manually inspected by the subjects to find the differences with an expected solution. Then, we evaluated the similarity between that knee point solution and

the expected ones based on Precision and Recall. When two expected solutions have the same average of Precision and Recall, we presented in the results the average of Precision and the average of Recall. However, this scenario never happens in our experiments since in that case the two expected solutions are very different (which is very rare to happen in practice).

Modularization Usability Metrics: In order to evaluate research questions RQ4.1 and RQ4.2, we consider two dimensions of usability: the estimated difficulty and the time that is needed to perform each task. These tasks are related to bug fixing in the transformations (T1) and adapting the transformations due to metamodel changes (T2).

- Subjects in the usability study (cf. Section 3.3.2.4) are able to rate the difficulty to perform a certain task (DF) using a five-point scale. The values of this scale are *very difficult*, *difficult*, *neutral*, *easy* and *very easy*. The more easy or less difficult in the rating, the better the result.

- In order to get a better estimate about the efficiency a modularized transformation can provide, we ask our study subjects (cf. the Section 3.3.2.4) to record the time that is needed to perform each of the tasks. The time unit we use is *minutes* and the less time is needed, the better the result.

As a helpful remainder for the rest of this evaluation, Table 3.3 summarizes, for each research question, the evaluation metrics that are used and the type of study it is – the meaning of ST is explained in Section 3.3.3.5.

3.3.2.3 Statistical Tests

Since metaheuristic algorithms are stochastic optimizers, they can provide different results for the same problem instance from one run to another. For this reason, our experimental study is performed based on 30 independent simulation runs for each case study and the obtained results are statistically analyzed by using the Mann-Whitney U test Arcuri and

Briand (2011) with a 99% significance level ($\alpha = 0.01$). The Mann-Whitney U test Mann and Whitney (1947), equivalent to the Wilcoxon rank-sum test, is a nonparametric test that allows two solution sets to be compared without making the assumption that values are normally distributed. Specifically, we test the null hypothesis (H_0) that two populations have the same median against the alternative hypothesis (H_1) that they have different medians. The p-value of the Mann-Whitney U test corresponds to the probability of rejecting the H_0 while it is true (type I error). A p-value that is less than or equal to α means that we accept H_1 and we reject H_0. However, a p-value that is strictly greater than α means the opposite. Since we are conducting multiple comparisons on overlapping data to test multiple null hypotheses, p-values are corrected using the Holm's correction Holm (1979). This correction procedure sorts the p-values obtained from n tests in an ascending order, multiplying the smallest value by n, the next one by n 1, etc

For each case study, we apply the Mann-Whitney U test for the results retrieved by the NSGA-III algorithm and the results retrieved by the other algorithms (Random Search, ε-MOEA and SPEA2). This way, we determine whether the performance between NSGA-III and the other algorithms is statistically significant or simply a random result.

3.3.2.4 User Studies

In order to answer research questions RQ3.1 to RQ4.2, we perform two studies, a correctness study for RQ3.1 and RQ3.2 and a usability study for RQ4.1 and RQ4.2. The

RQ	Evaluation Metric	Type of Study
RQ1	IHV, IGD	Performance Study
RQ2	IHV, IGD	Performance Study
RQ3.1	PR, RE, MP	Correctness Study
RQ3.2	PR, RE, MP	Correctness Study
RQ4.1	DF, T, ST	Usability Study
RQ4.2	DF, T, ST	Usability Study

Table 3.3: Evaluation metric and type of study for each Research Question (RQ).

correctness study retrieves the precision, recall and manual precision of our generated solutions in order to evaluate how *good* these solutions are. The usability study consists of two tasks that aim to answer the question of usefulness of modularized transformations.

Solution Correctness Study: For RQ3.1, we produce manual solutions to calculate the precision and recall of our automatically generated solutions (cf. Section 3.3.2.2). These manual solutions are developed by members of our research groups which have knowledge of ATL but are not affiliated with this work. Our study involved 23 subjects from the University of Michigan. Subjects include 14 undergraduate/master students in Software Engineering, 8 PhD students in Software Engineering, 2 post-docs in Software Engineering. Nine of them are females and 17 are males. All the subjects are volunteers and familiar with MDE and ATL. The experience of these subjects on MDE and modeling ranged from 2 to 16 years. All the subjects have a minimum of 2 years experience in industry (Software companies).

For RQ3.2 we need transformation engineers to evaluate our generated solutions, independent from any solution they would provide. More precisely, we asked the 23 subjects from the University of Michigan to inspect our solutions manually. The manual precision is computed not with respect to the best manual solutions (that is used for the precision and recall). The manual precision is computed by asking the developers to give their opinion about the correctness of the knee point solution by validating the modularization operations one by one. In fact, a deviation with expected solutions may not mean that the recommended operations are not correct, but it could be another possible way to re-modularize the program. We computed the average k-agreement between the developers for all the votes on all the evaluated operations and the average Cohen's kappa coefficient is 0.917. Thus, there is a consensus among the developers when manually evaluating the correctness of the modularization operations. The subjects were asked to justify their evaluation of the solutions and these justifications are reviewed by the organizers of the study. Subjects were aware that they are going to evaluate the quality of our solutions, but were not told from

which algorithms the produced solutions originate. Based on the results retrieved through this study, we calculate the manual precision metric as explained in Section 3.3.2.2.

Modularization Usability Study: In order to answer RQ4.1 and RQ4.2, we perform a user study using two of the seven case studies: Ecore2Maude (CS1) and XHTML2XML (CS4). These two case studies have been selected because they represent a good diversity of case studies as they differ in their size and structure. The Ecore2Maude transformation has a balanced and high number of rules and helpers and quite a high number of dependencies of all kinds. The XHTML2XML transformation, on the other hand, only consists of rules and has a comparatively low number of rule dependencies. In this study, subjects are asked to perform two tasks (T1 and T2) for each case study and version, once for the original, unmodularized transformation and once for the modularized transformation:

T1 Fixing a Transformation: The first task (T1) is related to fixing a model transformation due to bugs that have been introduced throughout the developing cycle. Such bugs usually alter the behavior of a transformation without breaking it, i.e., the transformation still executes but produces a wrong output. To simulate such a scenario, we introduced two bugs into the XHTML2XML transformation and four bugs into the Ecore2Maude transformation since it is larger and, therefore, it is more likely to contain bugs. The bugs have been created according to some mutation operators Troya et al. (2015); Mottu et al. (2006); Alhwikem et al. (2016), and the same bugs have been introduced both in the original and modularized versions. They are all of equal size and simulate bugs that are likely to be caused by developers. In this sense, a study of the specific faults a programmer may do in a model transformation is presented in Mottu et al. (2006). Out of the several different faults, we have applied changes in the navigation (according to the relation to another class change mutation operator Mottu et al. (2006)) and in the output model creation. Specifically, in the XHTML2XML transformation, one bug has to do with the incorrect initialization of a string attribute, while the other bug has to do with the incorrect assignment of a

reference (the reference should point to a different element). In the Ecore2Maude transformation, three bugs have to do with the incorrect initialization of a string attribute and the fourth one with the incorrect assignment of a reference. In order to avoid distorting the results for the comparison, all the bugs have been introduced in bindings, so the difficulty in finding them should be similar. To gain more insight in our evaluation, we split this task into two subtasks: the task of locating the bugs (T1a) and the task of actually fixing the bugs (T1b).

T2 **Adapting a Transformation**: The second task (T2) we ask our subjects to perform is to adapt a model transformation due to changes introduced in the input or output metamodels. These changes can occur during the lifecycle of a transformation when the metamodels are updated, especially when the metamodels are not maintained by the transformation engineer. In many cases, these changes break the transformation, i.e., make it not compilable and therefore not executable. Furthermore, either only one or both the input and output metamodels may evolve in real settings. To simulate reality, we have modified the input metamodel of the XHTML2XML transformation and the output metamodel of the Ecore2Maude transformation. Thus, we have changed the name of three elements in the XHTML metamodel and of two elements in the Maude metamodel (since this metamodel is a bit smaller). Therefore, the changes are again equal in nature.

The usability study was performed with software engineers from the Ford Motor Company and students from the University of Michigan. The software engineers were interested to participate in our experiments since they are planning to adapt our modularization prototype for transformation programs implemented for car controllers. Based on our agreement with the Ford Motor Company, only the results for the ATL case studies described previously can be shared in this chapter. However, the evaluation results of the software engineers from Ford on these ATL programs are discussed in this section. In total, we had 32 subjects that performed the tasks described above including 9 software engineers from

the IT department and innovation center at Ford and 23 participants from the University of Michigan (described previously). All the subjects are volunteers and each subject was asked to fill out a questionnaire which contained questions related to background, i.e., their persona, their level of expertise in software engineering, MDE and search-based software engineering. We have collected the data about the participants when completing the questionnaire about their background including the years/months of professional experience. The experience of these subjects on MDE and modeling ranged from 2 to 16 years. All the subjects have a minimum of 2 years experience in industry (Software companies). To rate their expertise in different fields, subjects could select from *none* (0-2 years), *very low* (2-3 years), *low* (2-4 years), *normal* (4-5 years), *high* (5-10 years) and *very high* (more than 10 years). After each task, in order to evaluate the usability of the modularized transformations against the original, unmodularized transformations, subjects also had to fill out the experienced difficulty to perform the task and the time they spent to finish the task (cf. metric description in Section 3.3.2.2).

For our evaluation, we divided the 32 subjects into four equal-sized groups, each group containing eight people. The first group (G1) consists of most software engineers from Ford, the second and third groups (G2 and G3) are composed of students from the University of Michigan and the fourth group (G4) contains one software engineer from Ford, 2 post-docs and 5 PhD students from the University of Michigan. All subjects have high to very high expertise in software development, model engineering and software modularization and on average a little bit less experience in model transformations and specifically ATL. To avoid the influence of the learning effect, no group was allowed to perform the same task on the same case study for the modularized and unmodularized versions. The actual assignment of groups to tasks and case studies is summarized in Table 3.4.

Please note that since the bugs introduced in the transformations are semantic bugs, neither the syntax nor the runtime analyzers of ATL will throw any error. This means that the participants will have to spot the errors by inspecting the ATL transformations,

CS	Task	Original	Modularized
CS1	Task 1	Group 1	Group 3
	Task 2	Group 2	Group 4
CS4	Task 1	Group 3	Group 1
	Task 2	Group 4	Group 2

Table 3.4: Assignment of groups to tasks and case studies. No group is allowed to perform a task on the same case study twice.

for which we expect a modularized ATL transformation to be useful with respect to a non-modularized one. Regarding available search tools for ATL, users can rely on the out-of-the-box tools offered by Eclipse. Eclipse allows to search for text in the current opened file as well as to search for text in a group of files. For better navigability and comprehensibility, ATL offers the possibility of realize code navigation and shows the text using syntax coloring. Syntax errors should also appear highlighted in the IDE.

3.3.2.5 Parameter Settings

In order to retrieve the results for each case study and algorithm, we need to configure the execution process and the algorithms accordingly. To be precise, all our results are retrieved from 30 independent algorithm executions to mitigate the influence of randomness. In each execution run, a population consists of 100 solutions and the execution finishes after 100 iterations, resulting in a total number of $10,000$ fitness evaluations.

To configure all algorithms except Random Search, which creates a new, random population in each iteration, we need to specify the evolutionary operators the algorithms are using. As a selection operator, we use deterministic tournament selection with $n = 2$. Deterministic tournament selection takes n random candidate solutions from the population and selects the best one. The selected solutions are then considered for recombination. As recombination operator, we use the one-point crossover for all algorithms. The one-point crossover operator splits two parent solutions, i.e., orchestrated rule sequences, at a random position and merges them crosswise, resulting in two, new offspring solutions. The

underlying assumption here is that traits which make the selected solutions fitter than other solutions will be inherited by the newly created solutions. Finally, we use a mutation operator to introduce slight, random changes into the solution candidates to guide the search into areas of the search space that would not be reachable through recombination alone. Specifically, we use our own mutation operator that exchanges one rule application at a random position with another with a mutation rate of five percent. With these settings, the NSGA-III algorithm is completely configured. However, the ε-MOEA takes an additional parameter called *epsilon* that compares solutions based on ε-dominance Laumanns et al. (2002) to provide a wider range of diversity among the solutions in the Pareto front approximiaton. We set this parameter to 0.2. Furthermore, in SPEA2 we can control how many offspring solutions are generated in each iteration. For our evaluation, we produce 100 solutions in each iteration, i.e., the number of solutions in the population.

As fitness function we use the four objectives described in Section 3.2.1.3. As a reminder, these objectives are the number of modules in the transformation (NMT), the difference between the number of transformation artifacts, i.e., rules and helpers, in the module with the lowest number of artifacts and the module with the highest number of artifacts (DIF), the cohesion ratio (COH) and the coupling ratio (COP). The initial objective values for each case study are listed in Table 3.5.

ID	Name	NMT ↓	DIF ↓	COH ↑	COP ↓
CS1	Ecore2Maude	1	0	0.15830	0.0
CS2	OCL2R2ML	1	0	0.17469	0.0
CS3	R2ML2RDM	1	0	0.79269	0.0
CS4	XHTML2XML	1	0	0.06344	0.0
CS5	XML2Ant	1	0	0.31609	0.0
CS6	XML2KML	1	0	0.30238	0.0
CS7	XML2MySQL	1	0	0.48888	0.0

Table 3.5: Initial objective values for all seven case studies. The arrow next to the objective name indicates the direction of better values.

3.3.3 Result Analysis

3.3.3.1 Results for RQ1

In order to answer RQ1 and therefore evaluate whether a sophisticated approach is needed to tackle the model transformation problem, we compare the search performance of our approach based on NSGA-III with the performance of Random Search (RS). If RS outperforms our approach, we can conclude that there is no need to use a sophisticated algorithm like NSGA-III. Comparing an approach with RS is a common practice when introducing new search-based problem formulations in order to validate the search effort Harman et al. (2012). Specifically, in our evaluation we investigate the Hypervolume indicator (IHV) and the Inverted Generational Distance indicator (IGD), cf. Section 3.3.2.2, on 30 independent algorithm runs for all case studies.

The results of our evaluation are depicted in Figure 3.5. The details of p-value and effect for each case study for the IHV and IGD metrics are given in Table 3.7 and in Table 3.8, respectively. In figure 3.5, each box plot shows the minimum value of the indicator (shown by the lower whisker), the maximum value of the indicator (shown by the upper whisker), the second quantile (lower box), the third quantile (upper box), the median value (horizontal line separating the boxes) and the mean value of the indicator (marked by an 'x'). We can clearly see that for the IHV indicator, RS has lower and therefore worse values than NSGA-III for all case studies. To investigate these results, we have deployed the Mann-Whitney U test with a significance level of 99%. As a result, we find a statistical difference between NSGA-III and RS for all case studies, except XHTML2XML. One reason for this result might be that the XHTML2XLM case study has a rather simple structure compared to most of the other case studies. To further investigate the differences between RS and NSGA-III we calculate the *effect size* for both indicators using Cohen's d statistic Cohen (1988). Cohen's d is defined as the difference between the two mean values $\overline{x_1} - \overline{x_2}$ divided by the mean squared standard deviation calculates by $\sqrt{(s_1^2 + s_2^2)/2}$. The effect size is considered:

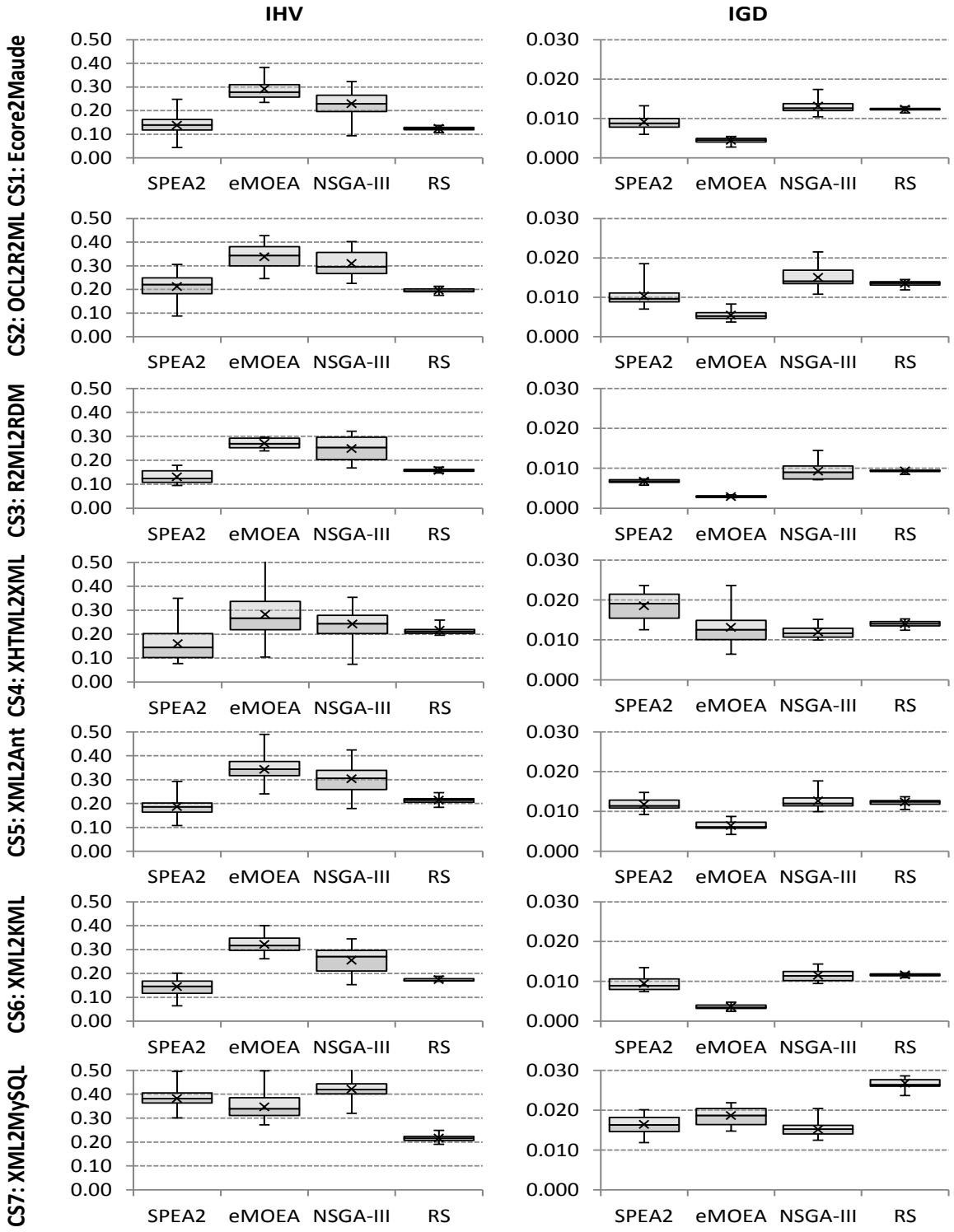


Figure 3.5: Hypervolume (IHV) and Inverted Generational Distance (IGD) indicator for all case studies and algorithms. The 'x'marks the mean value retrieved from a specific algorithm for a specific case study. All results are retrieved from 30 independent algorithm runs.

(1) small if $0.2 \leq d < 0.5$, (2) medium if $0.5 \leq d < 0.8$, or (3) large if $d \geq 0.8$. For IHV, all differences are considered large.

Interestingly, when we compare RS and NSGA-III for the IGD indicator the same way, the results are different. Please note that for IGD, lower values are considered better, as they indicate an overall better convergence of the algorithm. For IGD, there is no significant difference between the results of NSGA-III and RS, except for the simplest case study, XML2MySQL, where also the effect size yields a large difference. At the same time, in none of the cases the results of RS were significantly better due to the huge number of possible solutions to explore (high diversity of the possible remodularization solutions). Also interesting is the fact that RS produces solutions with a much lower variance of values. While IHV and IGD capture the efficiency of the search, we are also interested in the solutions found by each algorithm. To be more precise, we look at the median value of each objective value and its standard deviation. The results are depicted in Table 3.6, the bottom two lines of each case study. The arrow next to the objective name indicates the direction of better values. As we can see from the table, in the median case, the results of NSGA-III are better for NMT, COH and COP by a factor of around 2 in some cases. The only exception is DIF, where RS yields lower values in most case studies. This may be explained through the way NSGA-III tries to balance the optimization of all objective values and by doing so yields good results for all objectives, but may be outperformed when looking only at single objectives.

In conclusion, we determine that the transformation modularization problem is complex and warrants the use of a sophisticated search algorithm. Since in none of the cases RS significantly outperforms NSGA-III, while on the other hand there are many instances where NSGA-III dominates RS, we further infer that our many-objective formulation surpasses the performance of RS thus justifying the use of our approach and metaheuristic search.

CS	Approach	NMT ↓		DIF ↓		COH ↑		COP ↓	
CS1	SPEA2	28	10.09	40	15.08	1.89	1.22	31.14	27.45
	ε-MOEA	21	7.81	45	12.00	2.72	1.42	10.37	13.87
	NSGA-III	23	7.96	44	12.48	3.72	1.72	13.10	11.87
	RS	35	4.26	28	5.28	2.66	1.26	49.66	19.62
CS2	SPEA2	14	4.96	27	8.03	2.68	1.38	3.91	5.15
	ε-MOEA	13	4.51	28	7.45	3.44	1.17	0.84	3.75
	NSGA-III	13	2.94	23	3.77	5.23	1.03	3.09	2.31
	RS	19	3.56	21	4.72	2.56	1.15	5.80	4.58
CS3	SPEA2	30	9.76	49	14.76	1.12	0.87	12.17	12.88
	ε-MOEA	27	8.09	50	12.60	1.28	0.91	2.83	6.20
	NSGA-III	25	3.54	46	6.56	3.22	1.19	7.31	5.84
	RS	39	4.82	32	5.65	1.45	1.01	19.13	12.24
CS4	SPEA2	7	2.76	21	3.85	0.78	0.54	1.35	2.80
	ε-MOEA	7	2.74	20	3.66	0.57	0.36	0.36	2.01
	NSGA-III	7	3.00	18	4.38	1.06	0.66	0.43	2.64
	RS	6	2.31	22	2.97	0.52	0.34	0.31	1.87
CS5	SPEA2	10	3.99	19	6.58	1.55	0.86	4.95	4.30
	ε-MOEA	8	3.36	19	5.48	1.76	1.01	2.06	2.80
	NSGA-III	9	2.18	18	3.89	2.76	0.98	3.05	2.05
	RS	13	3.03	15	3.98	1.53	0.89	6.60	4.67
CS6	SPEA2	23	9.38	57	13.70	0.73	0.69	10.11	8.30
	ε-MOEA	18	7.00	59	10.32	1.50	0.85	7.30	5.88
	NSGA-III	19	3.25	55	6.82	2.08	0.96	11.13	4.63
	RS	30	5.30	47	6.92	1.00	0.82	19.17	6.73
CS7	SPEA2	5	1.78	6	2.84	2.93	1.30	1.50	2.23
	ε-MOEA	4	1.72	7	2.70	3.04	1.16	1.33	1.84
	NSGA-III	4	1.75	6	3.16	3.42	1.48	1.05	2.16
	RS	6	1.73	5	2.61	2.23	1.16	3.17	2.35

Table 3.6: Median objective values and standard deviations for all objectives in the fitness functions, all algorithms and all case studies. The arrow next to the objective name indicates the direction of better values. All results are retrieved from 30 independent algorithm runs.

3.3.3.2 Results for RQ2

To answer RQ2, we compared NSGA-III with two other algorithms, namely ε-MOEA and SPEA2, using the same quality indicators as in RQ1: Hypervolume (IHV) and the Inverted Generational Distance (IGD). All results are retrieved from 30 independent algo-

rithm runs and are statistically evaluated using the Mann-Whitney U test with a significance level of 99%.

A summary of the results is illustrated in Figure 3.5. The details of p-value and effect for each case study for the IHV and IGD metrics are given in Table 3.7 and in Table 3.8, respectively. As Figure 3.5 shows, NSGA-III and ε-MOEA produce better results than SPEA2 for the IHV indicator. In fact, the statistical analysis shows that NSGA-III produces significantly better results than SPEA2 and is on par with ε-MOEA for most case studies. While ε-MOEA has a more efficient search for CS1 and CS6, NSGA-III is the best algorithm for CS7. A slightly reversed picture is shown for the IGD indicator, where ε-MOEA always produces the best results and NSGA-III produces worse results than SPEA2. An exception to that is CS4 where ε-MOEA and NSGA-III are equally good and SPEA2 is worse and CS5 and CS7 where NSGA-III and SPEA2 produce statistically equivalent results. One possible explanation for this might be that these case studies are small comparing to the remaining ones. According to Cohen's *d* statistic, the magnitude of all differences is large.

Investigating the results further on basis of the retrieved objective values (cf. Table 3.6), we see that NSGA-III and ε-MOEA produce similar median values and standard deviations for most objectives and case studies, closely followed by SPEA2. For NMT, the difference between NSGA-III and ε-MOEA is very small while for DIF NSGA-III produces better median results for all case studies. The reverse is true for COH and COP where ε-MOEA produces the best results.

In conclusion, we can state that NSGA-III produces good results, but is occasionally outperformed by ε-MOEA. This is interesting as NSGA-III has already been applied successfully for the remodularization of software systems Mkaouer et al. (2015). However, in the case of software remodularization, the authors used up to seven different objectives in the fitness function which makes the difference of using many-objective algorithms compared to multi-objective algorithms more evident. Therefore, we think that NSGA-III is

still a good choice of algorithm for our model transformation problem as it allows to extend the number of objectives without the need to switch algorithms. Nevertheless, we also encourage the use of other algorithms. If necessary, only little work is needed to use our approach with a different algorithm Fleck et al. (2015).

3.3.3.3 Results for RQ3.1

In order to provide a quantitative evaluation of the correctness of our solutions for RQ3.1, we compare the produced modularizations of NSGA-III, ε-MOEA , SPEA2 and RS with a set of expected modularization solutions. Since no such set existed prior to this work, the expected solutions have been developed by the subjects of our experiments (cf. Section 3.3.2.4). We had a consensus between all the groups of our experiments when considering the best manual solution for every program. In fact, every participant proposed a possible modularization solution then after rounds of discussions we selected the best one for every ATL program based on the majority of the votes and we computed the average k-agreement between the developers for all the votes on all the proposed manual solutions. The average Cohen's kappa coefficient was 0.938, meaning there was a consensus among the developers when selecting the best manual solution. Then, to quantify the correctness of our solutions, we calculate the precision and recall of our generated solutions as described in Section 3.3.2.2.

Our findings for the average precision (PR) for each algorithm and for all case studies are summarized in Figure 3.6. From these results, we can see that independent of the case study NSGA-III has the solutions with the highest precision value, while RS produces solutions that are rather far away from what can be expected. More precisely, our approach based on NSGA-III produces solutions with an average of 89% precision and significantly outperforms the solutions found by the other algorithms. The solutions found by ε-MOEA have an average precision of 75% and the solutions found by SPEA2 have an average precision of 73%. The modularizations produced by RS have the least precision with an

average of 43% which can not be considered good. Based on the average and individual values for all case studies, a ranking of the algorithms would be NSGA-III on the first place, ε-MOEA on second place, SPEA2 on third place, and RS on the last place.

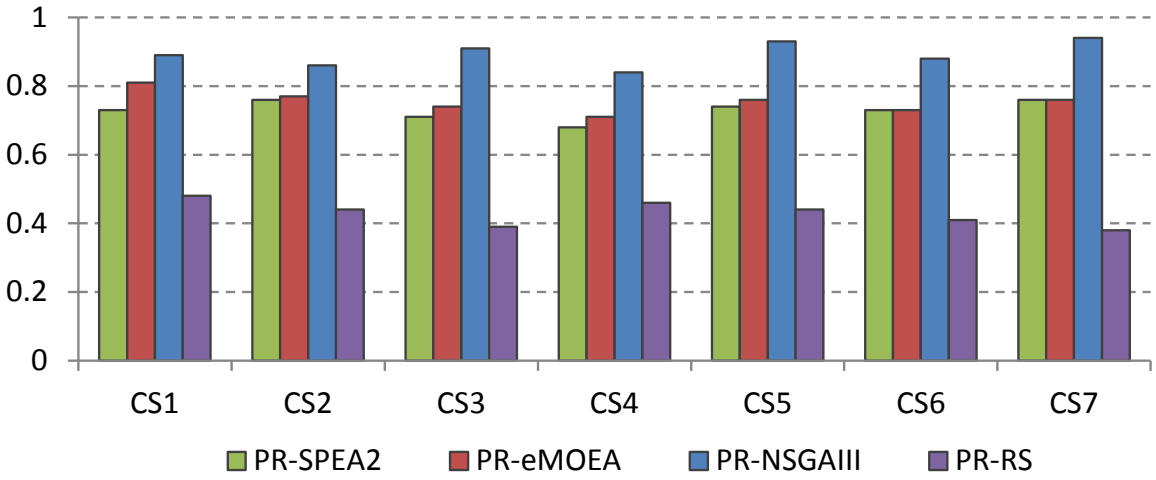


Figure 3.6: Qualitative correctness evaluation using precision (PR) for all case studies and algorithms. Higher values indicate better results.

A similar result can be seen for recall (RE) depicted in Figure 3.7, where NSGA-III produces solutions with the highest values, followed by ε-MOEA and SPEA2, and RS produces solutions with the lowest values. Particularly, the average recall of the solutions found across all case studies by NSGA-III is 90%, for ε-MOEA it is 82%, for SPEA2 it is 72% and for RS it is 48%. The performance of all algorithms is stable independent of the case study size, the highest standard deviations are RS and SPEA2 with 4%. As with precision, the values produced by the sophisticated algorithms can be considered good whereas RS solutions have a too small recall to be considered good. Based on the average and individual values for all case studies, a ranking between the algorithms would look the same as for the precision value.

Concluding, we state based on our findings that our approach produces good modularization solutions for all cases studies in terms of structural improvements compared to a set of manually developed solutions. In fact, NSGA-III produces the solutions with the highest precision and recall in all case studies compared to the other sophisticated algorithms

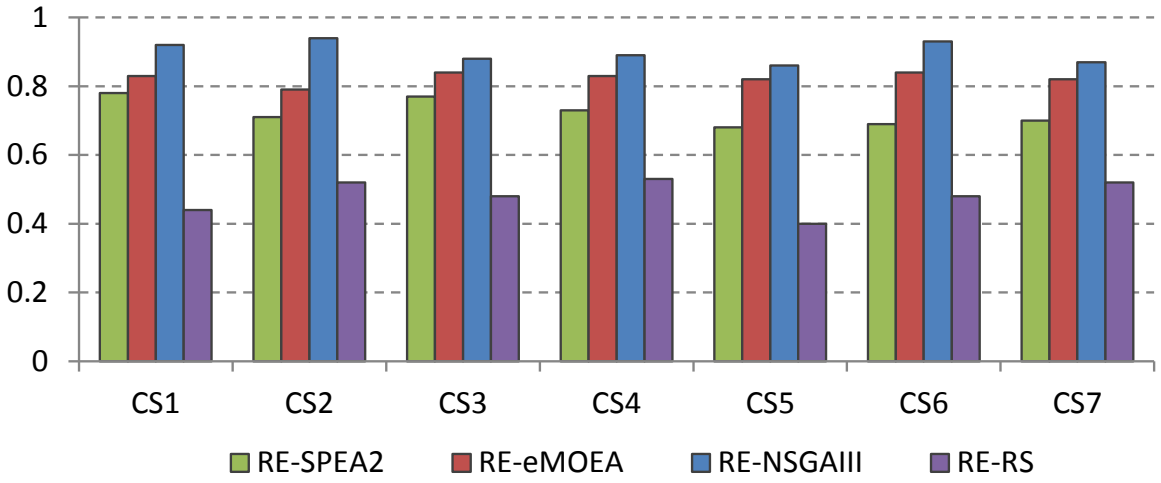


Figure 3.7: Qualitative correctness evaluation using recall (RE) for all case studies and algorithms. Higher values indicate better results.

ε-MOEA and SPEA2. Furthermore, all sophisticated algorithms significantly outperform RS. It is interesting to note, that the quality of the solutions and the ratio among the algorithms are quite stable across all case studies.

3.3.3.4 Results for RQ3.2

In RQ3.2, we focus more on the qualitative evaluation of the correctness of our solutions by gaining feedback from potential users in an empirical study (cf. Section 3.3.2.4) as opposed to the more quantitative evaluation in RQ3.1. To effectively collect this feedback, we use the manual precision metric which corresponds to the number of meaningfully modularized transformation artifacts as described in Section 3.3.2.2.

The summary of our findings based on the average MP for all considered algorithms and for all case studies is depicted in Figure 3.8. From these results, we can see that the majority of our suggested solutions can be considered meaningful and semantically coherent. In fact, for NSGA-III, the average manual precision for all case studies is around 96% and for the smaller case studies, i.e, XML2Ant (CS5) and XML2MySQL (CS7), even 100%. This result is significantly higher than that of the other algorithms. To be precise, ε-MOEA yields solutions with an average of 85% MP and SPEA2 has an average of 77%

MP over all case studies. On the other hand, the solutions found by RS only yield solutions with an average of 49% and the worst being 44% for the R2ML2RDM case study (CS3).

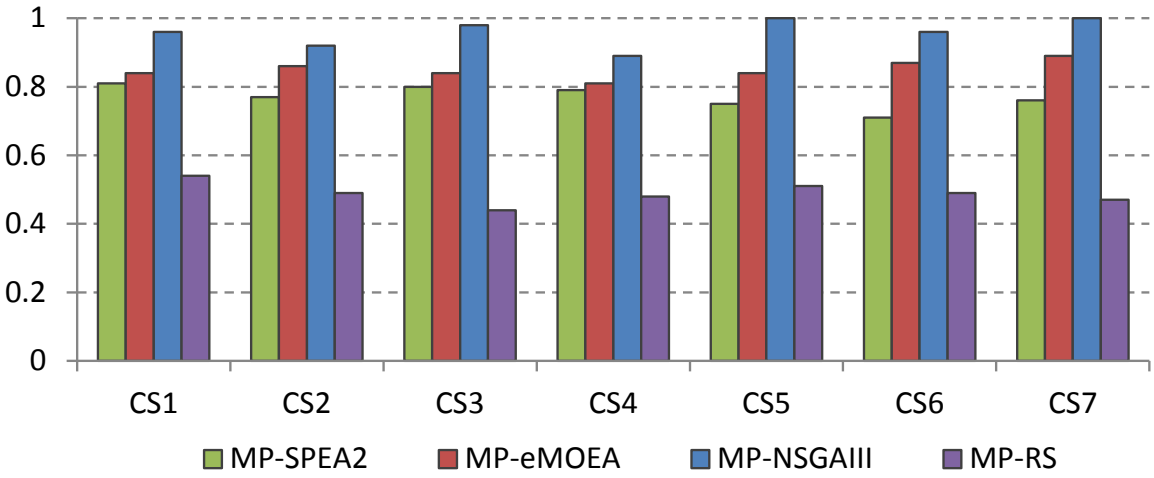


Figure 3.8: Qualitative correctness evaluation using manual precision (MP) for all case studies and algorithms. Higher values indicate better results.

In conclusion, we state that our many-objective approach produces meaningfully modularized transformation solutions with respect to the MP metric. While other sophisticated algorithms also yield satisfactory results that can be considered good, our approach based on NSGA-III clearly outperforms these algorithms.

3.3.3.5 Results for RQ4.1

In order to answer RQ4.1 to evaluate how useful modularizations are when faced with the task of fixing bugs in a transformation, we have performed a user study as described in Section 3.3.2.4. In this study, subjects first needed to locate several bugs in the transformation (T1a) and then fix those bugs by changing the transformation (T1b). Both subtasks were performed for the original and the modularized version of the Ecore2Maude (CS1) and XHTML2XML (CS4) case studies. For the evaluation, we focused on the experienced difficulty and the time that was spent to perform the task.

The results retrieved from the questionnaires for the experienced complexity to perform the task are depicted in Figure 3.9, *CS1-T1a Original* to *CS4-T1b Modularized*. The sta-

CS	IHV	SPEA2	eMOEA	RS
CS1	p-value	4.05E-17	3.09E-21	5.97E-37
	effect	0.813	0.834	0.881
CS2	p-value	3.3405E-19	2.37E-24	4.83E-40
	effect	0.824	0.806	0.837
CS3	p-value	5.12E-22	4.72E-24	5.87E-37
	effect	0.811	0.919	0.892
CS4	p-value	4.71E-28	2.19E-27	5.04E-37
	effect	0.861	0.937	0.849
CS5	p-value	3.09E-19	4.19E-21	3.97E-36
	effect	0.891	0.829	0.884
CS6	p-value	2.05E-19	3.69E-31	5.94E-37
	effect	0.810	0.836	0.894
CS7	p-value	3.39E-25	1.89E-26	4.46E-40
	effect	0.836	0.943	0.916

Table 3.7: Detailed values of adjusted p-value, using the Holm correction, and effect of the Hypervolume indicator (IHV) for each case study based on 30 independent runs for all case studies (NSGA-III vs. SPEA2, eMOEA, and RS, respectively).

tistical test concerning the p-value and effect is provided in Table 3.9. In Figure 3.9, we see how many of the eight people in each group have rated the experienced difficulty from *very easy* to *very difficult*. As can be seen, the modularized version only received ratings between *very easy* and *neutral* while the original, unmodularized version received only ratings from *neutral* to *very difficult*. This is true for both subtasks, i.e., locating a bug and actually fixing the bug.

The second dimension we investigate to answer RQ4.1 is the time that is spent to perform the task. To gain this data, subjects were asked to record their time in minutes. The results of this part of the study are depicted in Figure 3.11, *CS1-T1a Original* to *CS4-T1b Modularized*. In the figure, each subtask performed by a group for a specific case study and a specific version is shown as a boxplot indicating the minimum and maximum time recorded by each member of the group as well as the respective quartiles. The mean value is marked by an 'x'. As we can see, there is a significant difference between the time needed

CS	IHV	SPEA2	eMOEA	RS
CS1	p-value	2.95E-22	1.09E-21	3.96E-40
	effect	0.816	0.913	0.891
CS2	p-value	1.91E-29	2.32E-24	2.16E-40
	effect	0.819	0.812	0.881
CS3	p-value	3.42E-29	2.19E-27	2.91E-37
	effect	0.819	0.914	0.914
CS4	p-value	3.15E-31	2.01E-29	4.16E-37
	effect	0.917	0.811	0.926
CS5	p-value	1.85E-29	3.49E-30	1.98E-40
	effect	0.947	0.812	0.823
CS6	p-value	2.55E-29	2.19E-27	2.96E-40
	effect	0.843	0.911	0.914
CS7	p-value	3.14E-31	2.04E-30	3.76E-37
	effect	0.861	0.814	0.924

Table 3.8: Detailed values of adjusted p-value, using the Holm correction, and effect of the Inverted Generational Distance indicator (IGD) for each case study based on 30 independent runs for all case studies (NSGA-III vs. SPEA2, eMOEA, and RS, respectively).

CS	Approach	Original Program
CS1-T1a	p-value	2.19E-35
	effect	0.882
CS4-T1a	p-value	1.77E-31
	effect	0.803
CS1-T1b	p-value	2.24E-31
	effect	0.883
CS4-T1b	p-value	3.14E-33
	effect	0.922
CS1-T2	p-value	1.13E-35
	effect	0.891
CS4-T2	p-value	3.41E-31
	effect	0.884

Table 3.9: Detailed values of p-value, using the Holm correction, and effect for the time needed for tasks for Ecore2Maude (CS1) and XHTML2XML (CS4) based on all the subjects: original vs modularized transformation.

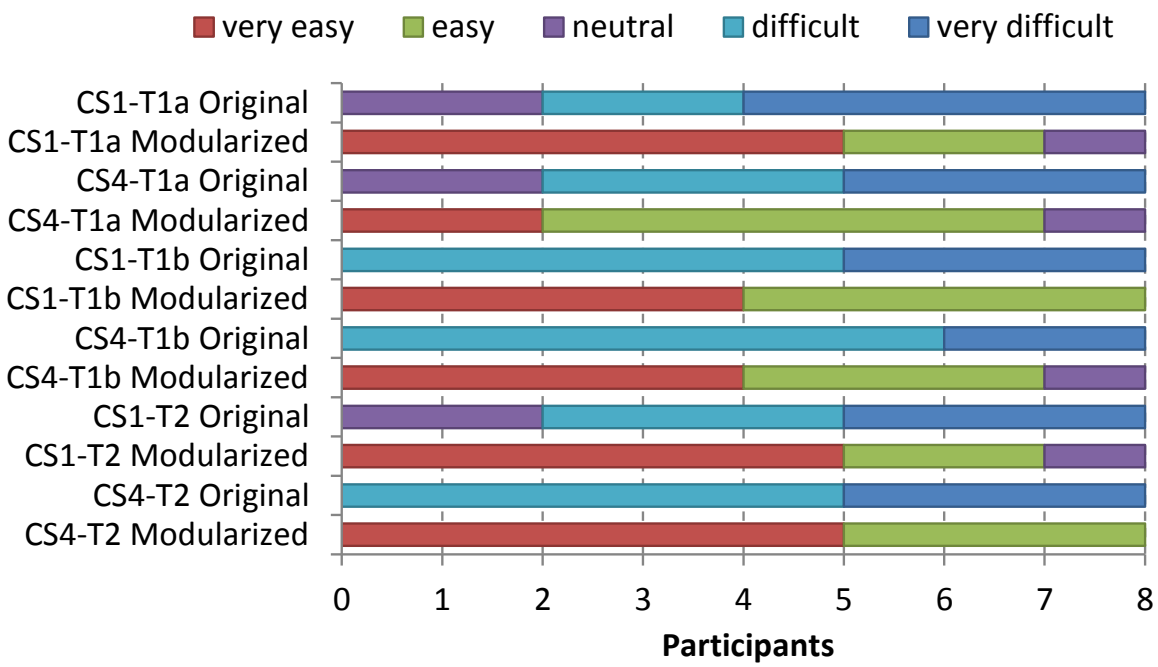


Figure 3.9: Evaluation of experienced difficulty to fulfill the user study tasks for Ecore2Maude (CS1) and XHTML2XML (CS4): Original vs Modularized Transformation.

to perform the tasks on an unmodularized transformation compared to a modularized transformation. In fact, the data shows that in all cases, the time needed for the modularized version is around 50% and less of the time needed in the unmodularized version. This seems to be true for both subtasks, even though the distribution within one group may vary.

Concluding, we state that the results clearly show that, independent of the group that performed the evaluation and independent of the respective case study, the task of bug fixing in a model transformation is much easier and faster with a modularized model transformation than with an unmodularized transformation. In this aspect, we think our approach can help model engineers to automate the otherwise complex task of transformation modularization and therefore increase the investigated aspects of the usability when working with model transformations.

Since evaluating the time to complete the tasks may not be sufficient, we have checked the completeness and correctness of the tasks by the developers as described in Figure 3.10. In 4 out of the 6 tasks for Ecore2Maude (CS1) and XHTML2XML (CS4), all the

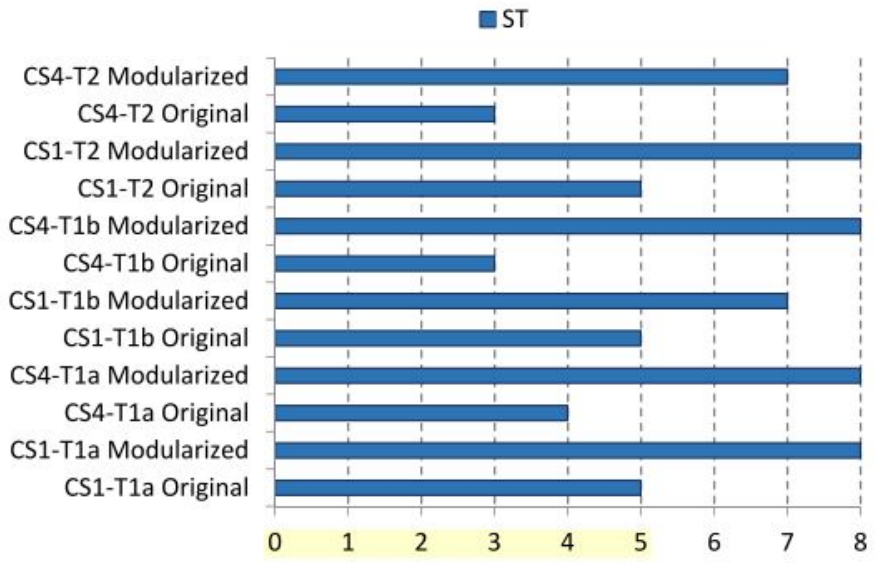


Figure 3.10: Evaluation of the number of participants who completed the tasks T1 and T2 successfully (ST) for Ecore2Maude (CS1) and XHTML2XML (CS4): original vs modularized transformation

participants completed the tasks successfully when working on the modularized programs. However, less than half of the 8 participants successfully completed the tasks on the same programs before modularization. These results confirm that it is less difficult to work on the modularized programs comparing to the original versions.

3.3.3.6 Results for RQ4.2

To answer RQ4.2 which is concerned with the adaptability of model transformations due to metamodel changes, we have performed a user study as described in Section 3.3.2.4. In this part of the study, subjects were asked to adapt a model transformation after the input or output metamodel has been changed. The necessary changes have been introduced by us, as described previously. As for RQ4.1, the task was performed for the original and the modularized versions of the Ecore2Maude (CS1) and XHTML2XML (CS4) case studies and we focused on the experienced difficulty and the necessary time.

The results retrieved for the experienced complexity are depicted in Figure 3.9, *CS1-T2 Original* to *CS4-T2 Modularized*. The statistical test concerning the p-value and effect is

provided in Table 3.9. Similar to what we have seen for the task of fixing a transformation, there is a significant difference between performing this task for the original, unmodularized transformation and for the modularized transformation. The modularized version received ratings between *very easy* and *neutral* while the original, unmodularized version received ratings from *neutral* to *very difficult*. Compared to the bug fixing task, the results may suggest that the gain in modularizing transformations when adapting transformations is a bit higher. This difference, however, may be caused by the personal interpretation of a few subjects in one group and can not be said to be statistically significant.

The time the subjects spent on adapting the transformation for each case study and version is depicted in Figure 3.11, *CS1-T2 Original* to *CS4-T2 Modularized*. Here we can see the same trend as with the bug fixing task: a significant reduced time of around 50% and more for the modularized version of the transformation compared to the unmodularized version. Interestingly, we can see that while the time needed to adapt the larger of the two transformations (Ecore2Maude, CS1) is higher than for the smaller transformation as expected, the gain for the larger transformation is also higher, resulting in a reversed result for the two case studies.

In conclusion, we determine that modularizing a transformation has a significant impact on the complexity and time needed to perform model transformation adaptations. Therefore, we think our approach can be useful for model engineers to automate the otherwise complex task of transformation modularization and improve these two metrics with respect to the investigated task.

3.3.4 Discussion

Despite the module concept is still not a wide-spread used transformation language concept, we believe it is important for keeping evolving the MDE community. The fact that the modularization of model transformations is not known by many MDE practitioners may be related to the maturity of the MDE field itself. Indeed, the lack of any modularized version

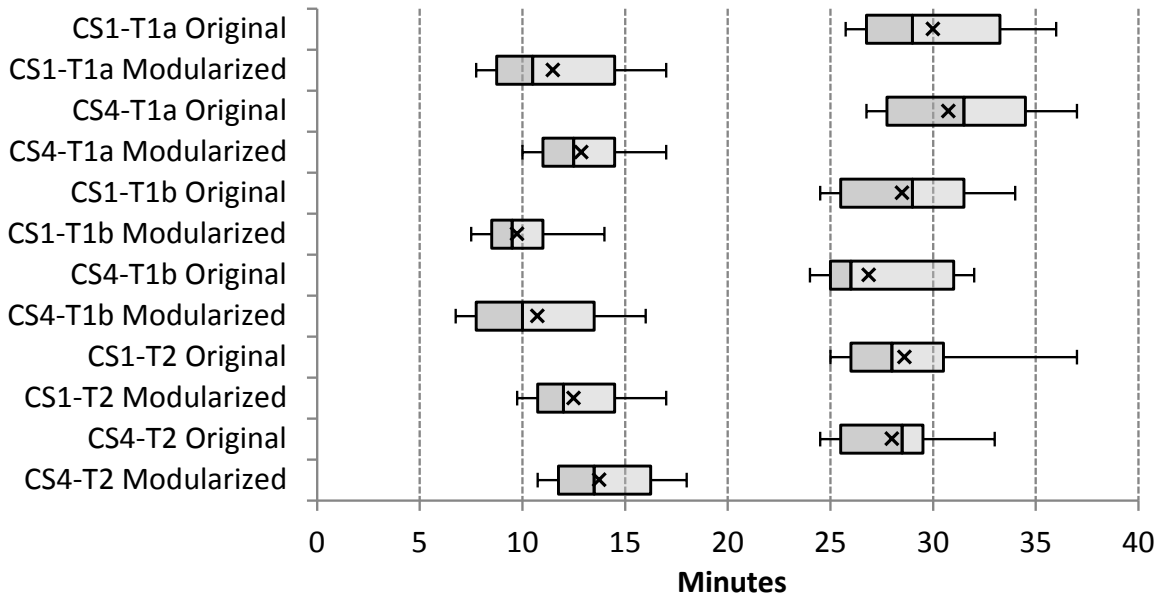


Figure 3.11: Time needed for tasks for Ecore2Maude (CS1) and XHTML2XML (CS4): original vs modularized transformation.

in the transformations of the ATL Zoo, despite the fact that ATL offers the superimposition mechanism, proves this.

However, while in the previous years the focus was on the functionality of model transformations and how to encode this functionality, there is currently a stronger trend to reason not only about the correctness Troya Castilla and Vallecillo Moreno (2011); Cuadrado et al. (2014b); Oakes et al. (2018), but also about the non-functional aspects of model transformations Lúcio et al. (2016); Nalchigar et al. (2013). We see the proper usage of modularization for transformations as a major cornerstone for reaching a transformation engineering discipline. This claim is supported by the results of our survey, which clearly show the need of automation support for modularization. Furthermore, as there are hidden dependencies in declarative transformation code Troya et al. (2016), having an automated way to reason about the quality of different modularization possibilities is considered important. Indeed, the alternative of doing this by a manual approach is not realistic as the benefit of the abstraction power of declarative languages is lost when designers have to reason about the operational semantics that is needed to fully uncover the dependencies.

3.3.5 Threats to Validity

According to Wohlin et al. (2012), there are four basic types of validity threats that can affect the validity of our study. We cover each of these in the following paragraphs.

3.3.5.1 Conclusion Validity

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. We use stochastic algorithms which by their nature produce slightly different results with every algorithm run. To mitigate this threat, we perform our experiment based on 30 independent runs for each case study and algorithm and analyze the obtained results statistically with the Mann-Whitney U test with a confidence level of 99% (α = 0.01) to test if significant differences existed between the measurements for different treatments. This test makes no assumption that the data is normally distributed and is suitable for ordinal data, so we can be confident that the statistical relationships we observed are significant.

3.3.5.2 Construct Validity

Construct validity is concerned with the relationship between theory and what is observed. Most of what we measure in our experiments are standard metrics such as precision and recall that are widely accepted as good proxies for quality of modularization solutions. A possible construct validity threat is related to the absence of similar work to modularize model transformations. For that reason we compared our proposal with random search and other search algorithms. Another construct threat can be related to the corpus of manually defined modularization solutions since developers may have different opinions. We will ask some new experts to extend the existing corpus and provide additional feedback regarding the manually defined solutions.

3.3.5.3 Internal Validity

There are several internal threats to validity that we would like to mention. For instance, even though the trial-and-error method we used to define the parameters of our search algorithms is one of the most used methods Eiben and Smit (2011), other parameter settings might yield different results. Therefore, we need to investigate this internal threat in more detail in our future work. In fact, parameter tuning of search algorithms is still considered an open research challenge. ANOVA-based techniques could be an interesting direction to study the parameter sensitivity. Also, the order in which we placed the objectives might influence the outcome of the search. We plan to further investigate this influence by evaluating different combinations of the objectives in future work. Furthermore, our objectives are limited to static metrics analysis to guide the search process. The use of additional metrics that also capture the runtime behavior of a transformation, e.g., execution time, might yield different results. While it is quite easy to introduce new objectives into our approach, we need to further investigate the use of other metrics in future work, e.g., capturing the performance of a transformation before and after modularization. Moreover, there are four threats to the validity of the results retrieved from the user studies: selection bias, learning effect, experimental fatigue, and diffusion. The selection bias is concerned with the diversity of the subjects in terms of background and experience. We mitigate this threat by giving the subjects clear instructions and written guidelines to assert they are on a similar level of understanding the tasks at hand. Additionally, we took special care to ensure the heterogeneity of our subjects and diversify the subjects in our groups in terms of expertise and gender. Finally, each group of subjects evaluated different parts of the evaluation, e.g., no group has worked on the same task or the same case study twice. To avoid the influence of the learning effect, no group was allowed to perform the same task on the same case study for the modularized and unmodularized versions. Different cases are solved by different participants in one task. There may be learning between the different tasks, however the types of bugs to identify and fix are different and related to different levels (rules,

model/metamodel elements, etc.). The same observation is valid for the features to implement into the ATL programs. The used ATL programs are also completely different in the context and structure. All these factors may minimize the risk of the learning mitigation The threat of experimental fatigue focuses on how the experiments are executed, e.g., how physical or mentally demanding the experiments are. Since the two case studies used in the experiments differ in size and number of bugs introduced, fatigue could have had an impact in the results. We have tried to prevent the fatigue threat with two strategies. First, we provided the subjects enough time to perform the tasks and fill out the questionnaires. All subjects received the instructions per e-mail, were allowed to ask questions, and had two weeks to finish their evaluation. Second, to try to balance the effort performed by all the four groups, each group realized two tasks, one with each of the case studies (cf. Table 3.4). Finally, there is the threat of diffusion which occurs when subjects share their experiences with each other during the course of the experiment and therefore aim to imitate each others results. In our study, this threat is limited because most of the subjects do not know each other and are located at different places, i.e., university versus company. For the subjects who do know each other or are in the same location, they were instructed not to share any information about their experience before a given date.

3.3.5.4 External Validity

The first threat in this category is the limited number of transformations we have evaluated, which externally threatens the generalizability of our results. Our results are based on the seven case studies we have studied and the user studies we have performed with our expert subjects. None of the subjects were part of the original team that developed the model transformations and to the best of our knowledge no modularized transformations exist for the evaluated case studies. Therefore, we can not validate the interpretation of the model transformation and what constitutes a good modular structure of our subjects against a “correct” solution by the transformation developers. We cannot assert that our results can

be generalized also to other transformations or other experts. In any case, additional experiments are necessary to confirm our results and increase the chance of generalizability. Second, we focus on the ATL transformation language and its superimposition feature, what allows to divide a model transformation into modules. However, ATL is not the only rule-based model transformation language. In order for our approach to be generalized also to other model transformation languages, we aim to apply it also to other popular model transformation languages which also provide the notion of modules, such as QVT-O, QVT-R, TGGs, ETL, and RubyTL.

3.4 Conclusion

In this chapter, we proposed an automated search-based approach to modularize model transformations based on higher-order transformations. Their application and execution are guided by our search framework which combines an in-place transformation engine and a search-based algorithm framework.

CHAPTER IV

Automatic Refactoring of ATL Model Transformations

4.1 Introduction

Model-driven engineering (MDE) is a methodology using models as executable development artifacts. MDE is becoming recently more popular in industry within diverse domains Völter et al. (2013); Czarnecki and Helsen (2006). This relatively new approach helps creating high-level abstractions in which they later can be executed or transformed using model transformations Schmidt (2006); Brambilla et al. (2017). Due to the evolution of languages and metamodels, model transformations -like any regular software- continuously adapt to changes. Therefore, model transformation programs slowly become more complex, less readable, less comprehensible, and less maintainable, leading to a possible increase in the maintenance activities both in time and cost Mohamed et al. (2009). In fact, most existing model transformation programs are still written in one module containing all the complex transformation rules despite their large number.

One of the most popular model transformation languages is the ATLAS Transformation Language (ATL) which is broadly used in both academia and industry Allilaire et al. (2006). ATL is a hybrid language, extensively used to write model transformation programs. Yet, few studies have been proposed refactoring techniques for ATL programs to improve the quality of model transformation. Most of these studies are mainly providing a manual support to apply few types of refactoring such as extract rule and rename elements to only

improve few metrics such as Fan-in and Fan-out van Amstel and van den Brand (2010, 2011); Porres (2003); Taentzer et al. (2012); Strüber et al. (2016); Wimmer et al. (2012); Cuadrado et al. (2017). However, manual refactoring is error-prone, time-consuming and not scalable which may explain the current low quality of existing model transformations programs.

Recently, there are some attempts to automated the refactoring of ATL programs Wimmer et al. (2012); Fleck et al. (2017) including our MODELS 2016 paper "Automated refactoring of ATL model transformations: a search-based approach" Alkhazi et al. (2016). We proposed an automated approach for refactoring ATL programs that find a trade-off between four different objectives related to fan-in, fan-out, reducing the number of rules and suggested refactorings. Thus, the search is guided based on those metrics. While the results are promising on refactoring ATL programs, our previous work was still limited to few basic metrics and refactoring types to mainly improve the modularity of ATL programs similar to Fleck et al. (2017).

In this chapter, we are extending our previous work Alkhazi et al. (2016) by (i) defining a new quality model for model transformation programs taking inspiration from the hierarchical quality model QMOOD Bansiya and Davis (2002) to consider important quality attributes beyond just the use of coupling and cohesion. We first select the most affected quality attributes by the design of an ATL program before adapting the formula associated with each attribute —following the same model detailed on the aforementioned paper. (ii) We adapted our multi-objective formulation to consider the new ATL-based quality metrics and refactoring types as detailed in section 4.3. To find the optimal trade-off between the various –and possibly conflicting— objectives and to deal with this large search space of possible refactoring solutions, we propose to use a multi-objective formulation based on NSGA-II Deb et al. (2002). (iii) We extended our validation with seven case studies from the ATL Zoo Project (2015) to evaluate the performance of our approach. We compared our approach with our previous multi-objective formulation not based on QMOOD Alk-

hazi et al. (2016), and also an existing semi-automated refactoring approach not based on heuristic search Wimmer et al. (2012).

Statistical analysis of our experiments showed that our proposal performed significantly better than random search, our previous multi-objective work not based on QMOOD Alkhazi et al. (2016), a mono-objective formulation and Wimmer et al. (2012) with an average precision and recall of 89% and 95% respectively when compared to manual solutions provided by a set of developers. The software developers, who participated in our experiments, confirmed also the relevance of the suggested refactorings as an outcome of a survey study.

4.2 Motivating Example and Challenges

In this section, we present a motivating example, and discuss the challenges of refactoring ATL transformations.

4.2.1 Motivating Example

To further introduce ATL as well as to motivate the need of automatically refactoring ATL transformations, an excerpt of an example ATL model transformation is shown in Listing IV.1. The transformation has been extracted from the ATL transformation zoo which is a public repository for collecting ATL transformations frequently used for research purposes. The transformation is, in essence, a simple copy transformation which converts MOF-based metamodels into KM3-based metamodels. The output metamodel excerpt for this transformation excerpt is shown in Figure 4.1. Please note that the input metamodel excerpt has the same class structure and inheritance hierarchy with slight name differences as can be observed in the ATL transformation. As can be further seen in the transformation, several duplicated bindings for the rules transforming attributes and references are used. The reason for this is simple. The two concepts share many common features which are defined by common superclasses.

Similar as using inheritance between classes in metamodels, ATL also allows to use rule

inheritance to introduce abstract rules for defining, for instance, the bindings for setting the features of the *TypedElement* class, namely for setting the *type*, *lower bound*, *upper bound*, and *ordered* features. Furthermore, it can be also observed that the *name* binding is occurring for all three rules in the transformation excerpt which could be also defined for the *ModelElement* class by introducing a top rule for the transformation definition from which all other rules directly or indirectly inherit. Rule inheritance is then used to build a hierarchy of transformation rules whereas the subrules inherit the *input pattern elements* including the *filter conditions* as well as the *output pattern elements* including the *bindings* of the superrules. Listing IV.2 gives an idea on how rule inheritance may be introduced for the *Attribute* and *Reference* transformation rules. In particular, the refactoring operations as shown in Table 4.1 are applied to produce the new transformation design. Please note that the refactoring operations are reused from previous work and the full refactoring catalogue for ATL can be found in Wimmer et al. (2012) and additional refactorings concerning the module concept of ATL are presented in Fleck et al. (2017). The refactorings presented in Wimmer et al. (2012) are classified into renaming, restructuring, inheritance-related, and OCL-related. In the motivating example, we focus on two inheritance-related refactoring operations – see Table 4.1.

By using rule inheritance, the binding duplicates can be removed. On the one hand, this has a positive impact on certain design metrics which have been discussed for ATL in Wimmer et al. (2012). The average number of bindings per rule is reduced as several feature bindings are pushed to the superrules. On the other hand, it has also a negative impact on other design metrics. For instance, the number of rules is increased which may lead to a higher complexity w.r.t. understanding how a particular rule may be executed by considering the exact rule inheritance semantics of ATL.

Listing IV.1: Excerpt of the initial Ecore 2 KM3 transformation

```
module Ecore2KM3;
create OUT : KM3 from IN : MOF;

rule Class {
    from i : MOF!EClass
    to o : KM3!Class (
      name <- i.name,
        structuralFeatures <- i.eStructuralFeatures,
        supertypes <- i.eSuperTypes,
      isAbstract <- i."abstract"
    )
}

rule Attribute {
    from i : MOF!EAttribute
    to o : KM3!Attribute (
        name <- i.name,
        type <- i.eType,
        lower <- i.lowerBound,
        upper <- i.upperBound,
        isOrdered <- i.ordered
    )
}

rule Reference {
    from i : MOF!EReference
    to o : KM3!Reference (
        name <- i.name,
        type <- i.eType,
        lower <- i.lowerBound,
        upper <- i.upperBound,
        isOrdered <- i.ordered,
        opposite <- i.eOpposite,
        isContainer <- i.containment
    )
}
```

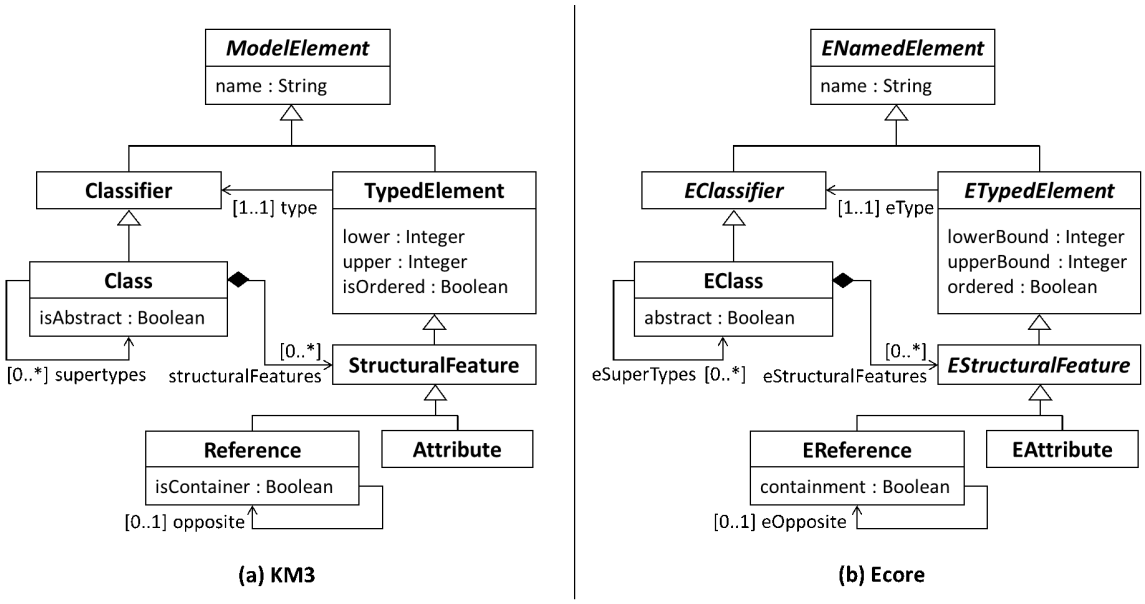


Figure 4.1: Metamodels of the transformation example; (a) excerpt of the KM3 metamodel and (b) excerpt of the Ecore metamodel.

Listing IV.2: Excerpt of the refactored Ecore 2 KM3 transformation

```
module Ecore2KM3;
create OUT : KM3 from IN : MOF;

abstract rule ModelElement {
    from i : MOF!ENamedElement
    to o : KM3!ModelElement (
      name <- i.name
    )
}

rule Class extends ModelElement {
    from i : MOF!EClass
    to o : KM3!Class (
      name <- i.name,
        structuralFeatures <- i.eStructuralFeatures,
        supertypes <- i.eSuperTypes,
      isAbstract <- i."abstract"
    )
}

abstract rule TypedElement extends ModelElement {
    from i : MOF!ETypedElement
    to o : KM3!TypedElement (
      type <- i.eType,
        lower <- i.lowerBound,
        upper <- i.upperBound,
        isOrdered <- i.ordered
    )
}

rule Attribute extends TypedElement {
    from i : MOF!EAttribute
    to o : KM3!Attribute ()
}

rule Reference extends TypedElement {
    from i : MOF!EReference
    to o : KM3!Reference (
       opposite <- i.eOpposite,
       isContainer <- i.containment
    )
}
```

Refactoring	Description
Extract Superrule	Rules may have several commonalities which should be extracted in one unique definition. The precondition for extracting a superrule is to have common supertypes for the input and output pattern elements of the selected rules. The postcondition is to have a new rule which is becoming the superule for the selected set of rules sharing the commonalities.
Pull Up Binding	A binding which is duplicated in all subrules of a superrule can be pulled up to the superrule in order to eliminate duplicates. The precondition is to have the feature which is computed as well as the features used in the value computation defined as features of the types used in the superrule. The postcondition is to have the binding presented in the superrule and the binding deleted in all subrules.

Table 4.1: List of considered refactorings for our motivating example based on Wimmer et al. (2012)

4.2.2 Challenges

This simple example already points out the main challenges of refactoring ATL transformations. Optimizing the different design metrics which have been proposed for ATL Wimmer et al. (2012); van Amstel and van den Brand (2010, 2011); Fleck et al. (2017) may lead to different potentially conflicting decisions how to refactor a particular ATL transformation. Even more challenging, there may not only exist one refactoring solution, but a huge set of possible refactored solutions which provide different design metrics configurations. As ATL transformations may become large containing over 80 rules and several helper definitions Kusel et al. (2013) as well as a large set of ATL refactoring operations has been proposed Wimmer et al. (2012); Fleck et al. (2017), the refactoring space of ATL transformations is enormous and enumerative approaches may fail to successfully explore this space efficiently. Therefore, we propose in Section 4.3 a search-based approach to refactoring of ATL transformations. Before introducing the search-based approach, a comprehensive quality model is required for ATL in order to extend existing work on design metrics for ATL in our search based framework.

4.3 Search-Based Model Transformations Refactoring

In this section, we start introducing an adaptation of the QMOOD model for model transformations, then we give an overview of our approach, followed by detailed description of how we formulated the refactoring recommendation process as a multi-objective optimization problem in addition to the multi-objective algorithm's (NSGA-II) adaptation.

4.3.1 QMOOD for Model Transformations

The quality of a software heavily relies on its design. In software, assessing quality means measuring several conflicting attributes. The quality value, however, depends on multiple factors and circumstances. For instance, what is considered very critical to one developer or designer might be less important for others since people have different preferences when they are designing or implementing a system. For instance, when the requirements of the transformations are not very clear (e.g., some rules need to be added or deleted), the flexibility attribute could be very important. When we are close to the release date, other attributes might be more critical to maintain. Thus, it is useful to somehow be able to quantify the quality of model transformations in order to make it easier for developers to compare and select between multiple refactoring paths. If we know where we stand —in terms of design quality— then we would be able to take better decisions as to where to move forward and what correction steps need to be performed to improve the model transformation programs.

In this regard, the authors of Bansiya and Davis (2002) linked object-oriented design properties to quality attributes in an effort to measure the quality of the software's design formally and validated the QMOOD model empirically on many projects. In this paper, we are adapting the QMOOD model to assist the computation of the quality of model transformations (i.e., ATL). It is important to note that model transformation languages are different than object-oriented programming languages, and thus, some design properties and metrics need to be mapped to their closest equivalent counterparts in the context of

ATL. In other words, we are using the hierarchical model, QMOOD, as a foundation for the quality attributes computation formulas (Table 4.2), ATL design metrics (Table 4.3) and the relationships between them (Table 4.4).

The equations of Table 4.2 provide explanations on how the different quality attributes are calculated. The details regarding how we are going to use this QMOOD model in practice to improve the quality of model transformation programs, in our automated ATL refactoring endeavours, will be described in the following sub-sections.

4.3.2 Approach Overview

The approach can be illustrated in the high-level overview shown in Figure 4.2. An ATL Analyser is applied to the ATL code in order to come up with the various design metrics listed in Table 4.3. These values are used later to measure the quality attributes shown in Table 4.2, which will eventually be used in the fitness function. The other input of the algorithm is the possible refactoring operations along with their pre- and post-conditions. The main target of the approach is to find the best sequence of refactorings that meets the following optimization objectives: (1) Maximize the quality attributes values (Table 4.2), (2) minimize the number of rules, and (3) minimize the number of changes.

The objectives mentioned above are not necessarily proportional. In fact, most of them are contrasting with each other. What makes the matters more complicated is the fact that there are multiple refactoring routes. In other words, the order in which we apply the refactoring operations makes a significant difference. Thus, with the substantial number of possible refactorings routes, and the conflicting objectives, we use a multi-objective genetic algorithm (NSGA-II) which is detailed in subsection 2.2.5.3.

4.3.3 Search-Based Formulation

Solution representations: A solution consists of a sequence of *n* refactoring operations involving one or multiple rules/modules of the ATL program to refactor. The vector-based

Quality Attribute	Index Computation Equation
Reusability	0.415 * Cohesion – 0.085 * Coupling + 0.67 * Design Size
Flexibility	0.583 * Composition – 0.166 * Coupling + 0.583 * Polymorphism
Understandability	0.385 * Cohesion – 0.275 * Abstraction – 0.275 * Coupling – 0.275 * Polymorphism – 0.275 * Complexity – 0.275 * Design Size
Functionality	0.175 * Cohesion + 0.275 * Polymorphism + 0.275 * Design Size + 0.275 * Hierarchies
Extendibility	0.5 * Abstraction – 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.25 * Abstraction + 0.25 * Composition + 0.25 * Inheritance + 0.25 * Polymorphism

Table 4.2: Computation Formulas for Quality Attributes.

ATL Metric	Name	Description
DSM	Design Size in Modules	The count of the total number of modules in the program
NOH	Number of Hierarchies	The count of the number of rule hierarchies
ANA	Average Number of Ancestors	The average number of rules from which a rule inherits information.
DMC	Direct Module Coupling	The count of the number of different modules that a module is directly related to.
CAR	Cohesion Among Rules	The metric computes the relatedness among rules of a module.
MOA	Measure of Aggregation	The metric counts the number helpers in ATL programs
MFA	Measure of Functional Abstraction	The metric is the ratio of the number of rules inherited by another rule to the total number of rules accessible by member rules of the module.
NOP	Number of Polymorphic Rules	This metric is a count of the rules that can exhibit polymorphic behavior.
NOR	Number of Rules	Total number of rules defined in a module

Table 4.3: Design Metrics Description.

Design Property	Derived Design Metric
Design Size	Design Size in Modules (DSM)
Hierarchies	Number of Hierarchies (NOH)
Abstraction	Average Number of Ancestors (ANA)
Coupling	Direct Module Coupling (DMC)
Cohesion	Cohesion Among Rules (CAR)
Composition	Measure of Aggregation (MOA)
Inheritance	Measure of Functional Abstraction (MFA)
Polymorphism	Number of Polymorphic Rules (NOP)
Complexity	Number of Rules (NOR)

Table 4.4: Relationship Between Design Properties and Design Metrics.

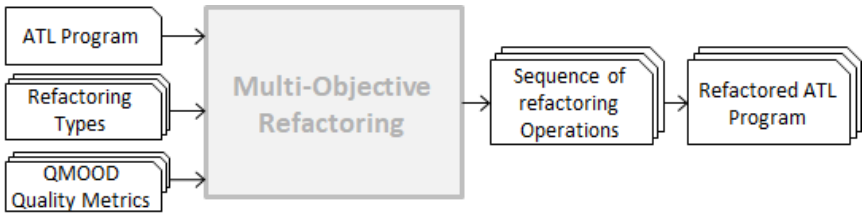


Figure 4.2: Overview of the multi-objective ATL refactoring approach.

representation is used to define the refactoring sequence. Each vector's dimension has a refactoring operation and its index in the vector indicates the order in which it will be applied. For every refactoring, pre- and post-conditions are specified to ensure the feasibility of the operation as detailed in Wimmer et al. (2012). The initial population is generated by randomly assigning a sequence of refactorings to a randomly chosen set of rules or modules. The different types of refactorings considered in our experiments are *Extract Helper/Rule*, *Inline Helper/Rule*, *Merge Rule*, *Split Rule*, *Extract Superrule*, *Eliminate Superrule*, *Pull Up Binding*, *Pull Up Filter*, *Push Down Binding*, *Push Down Filter* Wimmer et al. (2012), *Extract Module*, *Merge Modules*, and *Move Rule/Helper* Fleck et al. (2017).

The size of a solution, i.e., the vector's length is randomly chosen between upper and lower bound values. The determination of these two bounds is similar to the problem of bloat control in genetic programming where the goal is to identify the tree size limits. Since

ExtractSuperrule("TypedElement" , MOF!ETypedElement, KM3!TypedElement, [Attribute, Reference])	*PullUpBinding*(type, [Attribute, Reference], TypedElement)	*PullUpBinding* (lower, [Attribute, Reference], TypedElement)

Figure 4.3: Example of a simplified solution representation.

the number of required refactorings depends mainly on the size and quality of the ATL program, we performed, for each target project, several trial and error experiments using the HyperVolume (HP) performance indicator Deb et al. (2002) to determine the upper bound after which, the indicator remains invariant. For the lower bound, it is arbitrarily chosen. The experiments section will specify the upper and lower bounds used in this study.

Figure 4.3 shows a simplified example of a solution including three refactorings applied to the ATL program described in Listings IV.1 and IV.2. The solution includes two refactoring types with the following controlling parameters: *ExtractSuperrule*(name, inputType, outputType, subrules), *PullUpBinding*(binding, subrules, superrules).

Solution variation: In each search algorithm, the variation operators play the key role of moving within the search space with the aim of driving the search towards optimal solutions.

For the crossover, we use the one-point crossover operator. It starts by selecting and splitting at random two parent solutions. Then, this operator creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and vice versa for the second child. This operator must ensure the respect of the length limits by eliminating randomly some refactoring operations. It is important to note that in multi-objective optimization, it is better to create children that are close to their parents in order to have a more efficient search process. An example of this operation is illustrated in figure 4.4.

For mutation, we use the bit-string mutation operator that picks probabilistically one

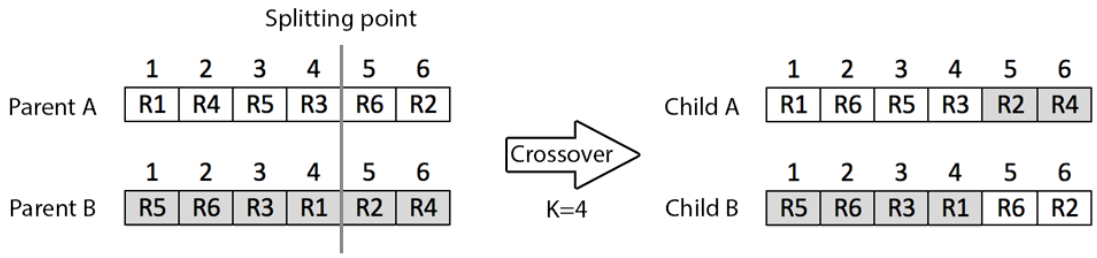


Figure 4.4: Example of crossover operation.

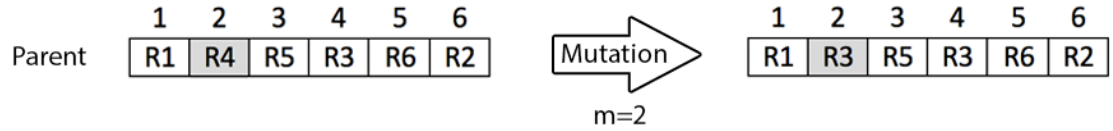


Figure 4.5: Example of the mutation operation.

or more refactoring operations from its or their associated sequence and replaces them by other ones from the initial list of possible refactorings as shown in figure 4.5. When applying the change operators, the different pre- and post-conditions are checked to ensure the applicability of the newly generated solutions. For example, to apply the refactoring operation *extract rule* a number of necessary pre-conditions should be satisfied such as the rule should exist. A post-condition example is to check that the rule exists and a new rule was created containing some of the metamodel elements of the original rule. More details about the adapted pre- and post-conditions for refactorings can be found in Wimmer et al. (2012). We also apply a repair operator that randomly selects new refactorings to randomly replace those creating conflicts.

Solution evaluation: The generated solutions are evaluated using three fitness functions as detailed in the following.

Maximize the quality attributes values: the formulas listed in Table 4.2 gives us the advantage of calculating the values of the various quality attributes easily. Worth mentioning that we are treating the quality attributes equally in this paper. Whereas in some practical situations, the developer might want to give more weight to one or more attributes depending on the circumstances and the objective of the refactoring operations.

FF1: Max(x) *where x is the sum of Reusability, Flexibility, Understandability, Functionality, Extendibility, and Effectiveness.*

Minimize the number of recommended refactorings: The application of a specific suggested refactoring sequence may require an effort that is comparable to that of re-implementing part of the system from scratch. Taking this observation into account, it is essential to minimize the number of suggested refactorings in the solution since the designer may have some preferences regarding the percentage of deviation with the initial ATL program design. In addition, most developers prefer solutions that minimize the number of changes applied to their design and rules modification. Thus, we formally defined the fitness function as the number of recommended refactorings.

FF2: Min (n) *where n is the number of recommended refactorings.*

Minimize the number of rules: the metric can be easily calculated on ATL programs. The reason to use this metric is to avoid that some refactorings such as split rule or extract rule will generate a high number of new rules when optimizing the remaining objectives.

FF3: Min(r) *where r is the number of rules.*

In fact, the use of multiple quality attributes to guide the search for relevant refactorings may increase dramatically the number of rules such as an intensive use of extract rules to improve the extendibility quality attributes.

4.4 Evaluation

In order to evaluate the ability of our automated refactoring approach to generate good refactoring recommendations for ATL programs, we conducted a set of experiments based on several transformation programs available in the ATL Zoo. Each experiment is repeated 30 times, and the obtained results are subsequently statistically analyzed. In this section, we first present our research questions and the pilot study, and then describe and discuss the obtained results.

4.4.1 Research Questions

We defined five research questions that address the applicability, performance, and the usefulness of our multi-objective formulation. The five research questions are as follows:

RQ1: Search validation (sanity check). To validate the problem formulation of our approach, we compared our multi-objective formulation with a random search algorithm (RS). If RS outperforms an intelligent search method, we can conclude that there is no need to use a metaheuristic search.

RQ2: To what extent can the proposed approach improve the quality of ATL programs using the combination of multi-objective search and QMOOD?

RQ3: How does our multi-objective ATL-based refactoring formulation perform compared to a mono-objective one and our previous multi-objective refactoring work Alkhazi et al. (2016)?

A multi-objective algorithm provides a trade-off between the four objectives where developers can select their desired refactoring solution from the Pareto-optimal front. A mono-objective approach uses a single fitness function that is formed as an aggregation of the four normalized objectives and generates as output only one refactoring solution. This comparison is required to ensure that the solutions provided by NSGA-II provide a better trade-off between the four objectives than a mono-objective approach. Otherwise, there is no benefit to our multi-objective adaptation. Furthermore, it is important to compare the performance of our new multi-objective QMOOD formulation to our previous multi-objective work of MODELS2016 Alkhazi et al. (2016) to evaluate the relevance of considering new quality attributes on the relevance of recommended refactoring recommendations.

RQ4: How does the proposed multi-objective ATL refactoring approach perform compared to existing semi-automated approach not based on heuristic search?

While it is interesting to show that maybe our proposal outperforms random search or a mono-objective refactoring approaches, developers will consider our approach useful, if it

can outperform other existing tools that are not based on optimization techniques. Thus, we compared our approach to the semi-automated refactoring approach proposed in Wimmer et al. (2012).

The last research question is related to the benefits of our approach for software engineers.

RQ5 (Insight): Can our ATL refactoring approach be useful for software developers in practice?

We conducted a post-study questionnaire with the subjects of our experiments that collects their opinions of our tool.

4.4.2 Case Studies

Our research questions are evaluated using the following seven case studies. Each case study consists of one model transformation and all the necessary artifacts to execute the transformation, i.e., the input and output metamodels and a sample input model. Most of the case studies have been taken from the ATL Zoo Project (2015), a repository where developers can upload and describe their ATL transformations. We briefly describe in the following the different ATL transformation programs used in our study.

Ecore2Maude: This transformation takes an Ecore metamodel as input and generates a Maude specification. Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications.

OCL2R2ML: This transformation takes OCL models as input and produces R2ML (REWERSE I1 Markup Language) models as output.

R2ML2RDM: This transformation is part of the sequence of transformations to convert OCL models into SWRL (Semantic Web Rule Language) rules. In this process, the selected transformation takes a R2ML model and obtains an RDM model that represents the abstract syntax for the SWRL language.

XHTML2XML: This transformation receives XHTML models conforming to the XHTML language specification version 1.1 as input and converts them into XML models consisting of elements and attributes.

XML2Ant: This transformation is the first step to convert Ant to Maven. It acts as an injector to obtain an XMI file corresponding to the Ant metamodel from an XML file.

XML2KML: This transformation is the main part of the KML (Keyhole Markup Language) injector, i.e., the transformation from a KML file to a KML model. Before running the transformation, the KML file is renamed to XML and the KML tag is deleted. KML is an XML notation for expressing geographic annotation and visualization within Internet-based, two-dimensional maps and three-dimensional Earth browsers.

XML2MySQL: This transformation is the first step of the MySQL to KM3 transformation scenario, which translates XML representations used to encode the structure of domain models into actual MySQL representations.

We have selected these case studies due to their difference in size, structure and number of dependencies among their transformation artifacts, i.e., rules and helpers. Table 4.5 summarizes the number of rules, the number of helpers and the number of dependencies between rules.

To answer RQ1, RQ2, RQ3 and RQ4, it is important to validate the proposed refactoring solutions from both quantitative and qualitative perspectives. For the quantitative validation, we evaluated the improvements of the different quality metrics used by our approach before and after refactorings. Since the metrics improvement evaluation is not sufficient, we asked a group of developers, as detailed later, to manually identify several refactoring opportunities and apply several refactorings to fix the detected possible quality improvements on the five transformation programs. Table 4.5 summarizes the number of expected refactorings for every ATL program. Then, we calculated precision and recall scores to compare between refactorings recommended by our approach and those suggested manu-

ally by the subjects:

$$RC_{Recall} = \frac{\text{suggested operations} \cap \text{expected operations}}{\text{expected operations}} \in [0,1] \tag{4.1}$$

$$PR_{Precision} = \frac{\text{suggested operations} \cap \text{expected operations}}{\text{suggested operations}} \in [0,1] \tag{4.2}$$

For the qualitative validation, we asked the group of potential users of our tool to evaluate, manually, whether the suggested refactorings are feasible and efficient at improving the ATL programs quality and achieving their maintainability objectives. We define the metric Manual Correctness (MC) to mean the number of meaningful/relevant refactorings divided by the total number of suggested refactorings. MC is given by the following equation:

$$MC = \frac{\#\text{coherent applied refactorings}}{\#\text{proposed refactorings}} \tag{4.3}$$

To avoid the computation of the MC metric being biased by the developer's feedback, we asked the developers to manually evaluate the correctness of the recommended refactorings on the ATL programs that they did not refactor using our tool.

To answer the first research question RQ1, a random multi-objective algorithm was implemented where at each iteration the population is randomly created without the use of change operators. The random search used the same fitness functions of our QMOOD formulation but without the use of the change operators. The obtained best refactoring solution was compared for statistically significant differences with NSGA-II using PR, RC, MC and the execution time (CT). To answer RQ2, we evaluate the results of our NSGA-II algorithm using all the above evaluation metrics. To answer RQ3, we compared our approach to a mono-objective Genetic Algorithm where all the four objectives were normalized in the range [0..1] and aggregated into one objective to minimize. To answer RQ4, we compare NSGA-II to an existing semi-automated ATL refactoring approach Wimmer et al. (2012) where the refactoring operations have to be explicitly triggered by the user

and only the execution of the manually identified refactorings is automated. We used all the above evaluation metrics to perform the comparisons in RQ3 and RQ4 as well.

Case Study	#Rules	#Helpers	#Dependencies	#Expected Refactorings
Ecore2Maude	40	40	27	19
OCL2R2ML	37	11	54	16
R2ML2RDM	58	31	137	22
XHTML2XML	31	0	59	18
XML2Ant	29	7	28	16
XML2KML	84	5	0	37
XML2MySQL	6	10	5	9

Table 4.5: Statistics of the Case Studies.

Our study involved 27 participants from the University of Michigan. Participants include 21 master students and 8 Ph.D. students in Software Engineering. All the participants are volunteers and familiar with ATL and model transformations. All the graduate students have strong background in refactoring and software quality since they all took a graduate course on Software Quality Assurance extensively covering these topics. The experience of these participants on programming and refactoring ranged from 2 to 16 years in industry. Eleven out the twenty-seven participants are active programmers in software companies.

To answer RQ5, we used a post-study questionnaire that collects the opinions of developers on our tool. Participants were first asked to fill out a pre-study questionnaire containing five questions. The questionnaire helped to collect background information such their programming experience, their familiarity with software refactoring and ATL. In addition, all the participants attended one lecture about ATL and software refactoring, and passed six tests to evaluate their performance to evaluate and suggest refactoring solutions for ATL programs.

Each participant in the study received a questionnaire, a manuscript guide to help them to fill the questionnaire, the tools and results to evaluate, and the ATL source code of the studied transformations. Since the application of refactoring solutions is a subjective

process, it is normal that not all the developers have the same opinion. In our case, we considered the majority of votes to determine if suggested solutions are correct or not. Each participant evaluates different refactoring solutions for the different techniques and systems.

We asked every participant to manually suggest and apply refactorings to improve the quality of the ATL programs. As an outcome of this first scenario, we calculated the differences between the recommended refactorings and the expected ones (manually suggested by the developers). In the second scenario, we asked the developers to manually evaluate the best recommended solution by our algorithm and the remaining techniques. We performed a cross-validation between the participants to avoid the computation of the MC metric being biased by their manual recommendations. In the third scenario, we asked the participants to use our tool during a period of two hours on the different programs and then we collected their opinions based on a post-study questionnaire that will be detailed later. The participants were asked to justify their evaluation of the solutions and these justifications are reviewed by the organizers of the study.

For each case study and algorithm, we select one solution using a knee point strategy Rachmawati and Srinivasan (2009). The knee point corresponds to the solution with the maximal trade-off between all fitness functions, i.e., a vector of the best objective values for all solutions. In order to find the maximal trade-off, we use the trade-off worthiness metric proposed by Rachmawati and Srinivasan (2009) to evaluate the worthiness of each solution in terms of objective value compromise. The solution nearest to the knee point is then selected and manually inspected by the subjects to find the differences with an expected solution. While the knee point selection may not be the perfect way, it is the only strategy to ensure a fair comparison with the mono-objective and deterministic approaches since they generate only one solution (sequence of refactorings) as output. Subjects were aware that they are going to evaluate the quality of our solutions, but were not told from which algorithms the produced solutions originate.

4.4.3 Experimental Setting

Parameter setting influences significantly the performance of a search algorithm on a particular problem. For this reason, for each algorithm and for each ATL program, we perform a set of experiments using several population sizes: 50, 100, 200, 300 and 500. The stopping criterion was set to 100,000 evaluations for all algorithms in order to ensure fairness of comparison. The other parameters' values were fixed by trial and error and are as follows: crossover probability = 0.7; mutation probability = 0.4 where the probability of gene modification is 0.2. Each algorithm is executed 30 times with each configuration and then the comparison between the configurations is done using the Wilcoxon test. The upper and lower bounds on the chromosome length used in this study are set to 10 and 50 respectively.

4.4.4 Statistical test methods

Since metaheuristic algorithms are stochastic optimizers, they can provide different results for the same problem instance from one run to another. For this reason, our experimental study is based on 30 independent simulation runs for each problem instance and the obtained results are statistically analyzed by using the Wilcoxon rank sum test with a 95% confidence level ($\alpha = 5\%$). The latter tests the null hypothesis, H_0, that the obtained results of two algorithms are samples from continuous distributions with equal medians, against the alternative that they are not, H_1. The p-value of the Wilcoxon test corresponds to the probability of rejecting the null hypothesis H_0 while it is true (type I error). A p-value that is less than or equal to α (≤ 0.05) means that we accept H_1 and we reject H_0. However, a p-value that is strictly greater than α (> 0.05) means the opposite. In fact, for each problem instance, we compute the p-value obtained by comparing the results of the different algorithms with our approach. In this way, we determine whether the performance difference between our technique and one of the other approaches is statistically significant or just a random result. The results presented were found to be statistically significant on 30

independent runs using the Wilcoxon rank sum test with a 95% confidence level ($\alpha < 5\%$). The Wilcoxon rank sum test verifies whether the results are statistically different or not; however, it does not give any idea about the difference in magnitude. Thus, we used the Vargha-Delaney A measure which is a non-parametric effect size measure. In our context, given the different performance metrics (such as PR, RC, MC, etc.), the A statistic measures the probability that running an algorithm B1 (NSGA-II based on QMODD) yields better performance than running another algorithm B2 (such as RS, Mono-objective GA, etc.). If the two algorithms are equivalent, then A = 0.5.

Overall, we have found the following results: a) on small scale ATL programs (XML2MySQL, XHTML2XML and XML2Ant) our approach is better than all the other algorithms based on all the performance metrics with an A effect size higher than 0.92; and b) on medium and large scale ATL programs (XML2KML, Ecore2Maude, OCL2R2ML and R2ML2RDM), our approach is better than all the other algorithms with an A effect size higher than 0.89.

4.4.5 Results and Discussions

Results for RQ1: The results for the first research questions are summarized in Figures 4.6, 4.7 and 4.8. It is clear that QMOOD-NSGA-II is better than random search based on the different metrics of PR, RC and MC on all the 7 ATL case studies. The average precision, recall and manual correctness values of random search on the different ATL programs are lower than 28%. This can be explained by the huge search space to explore to generate relevant refactorings. Figure 4.9 shows that the execution time (CT) of random search is lower than QMOOD-NSGA-II however the difference is just limited to an average of 15 minutes. Furthermore, ATL refactoring is not requiring a strict time constraints unlike real-time application which is not the case here. We do not dwell long in answering the first research question, RQ1, which involves comparing our approach based on QMOOD-NSGA-II with random search. The remaining research questions will reveal more about the performance, insight, and usefulness of our approach. We conclude that

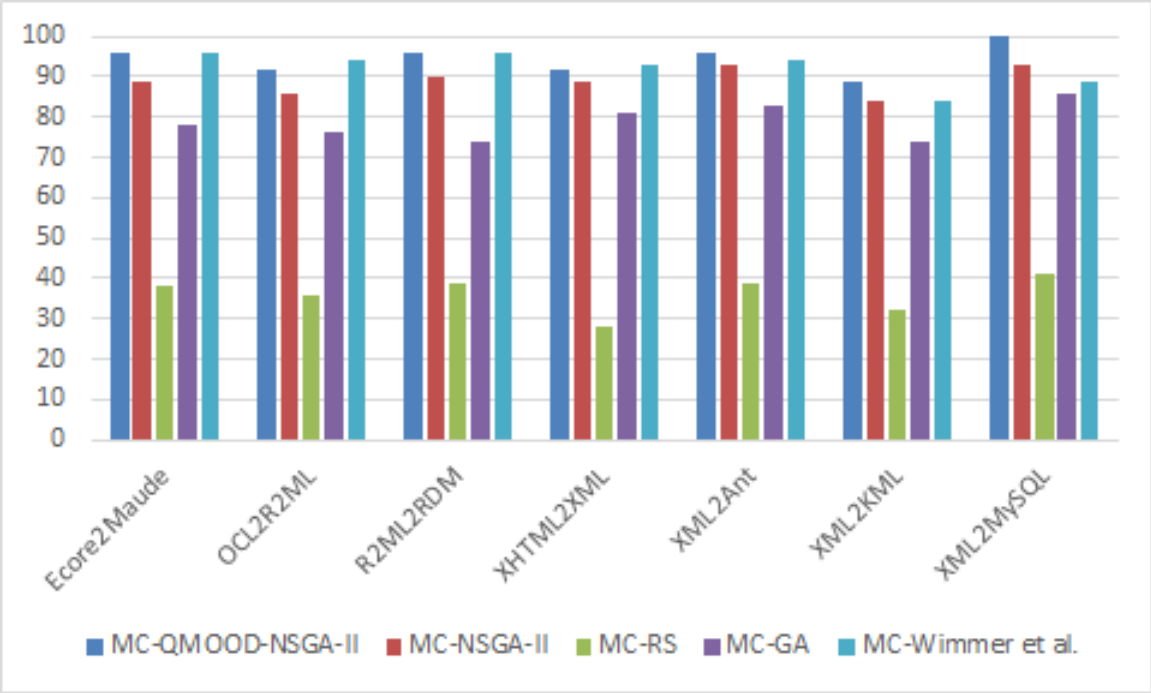


Figure 4.6: Median Manual Correctness (MC) over 30 runs on all the 7 ATL programs using the different ATL refactoring techniques with a 95% confidence level ($\alpha < 5\%$).

there is empirical evidence that our multi-objective formulation based on QMOOD surpasses the performance of random search thus our formulation is adequate (this answers RQ1).

Results for RQ2: As reported in Figure 4.6, the majority of the refactoring solutions recommended by our multi-objective approach were correct and approved by developers. On average, for all of our seven studied projects, 94% of the proposed ATL refactoring operations are considered as feasible, improve the quality and are found to be useful by the software developers of our experiments. The highest MC score is 100% for the XML2MySQL program and the lowest score is 89% for XML2KML program. Thus, it is clear that the results are independent of the size of the ATL programs and the number of recommended refactorings. Most of the refactorings that were not manually approved by the developers were found to be either violating some post-conditions or introducing design incoherence.

Since the MC metric just evaluates the correctness and not the relevance of the recom-

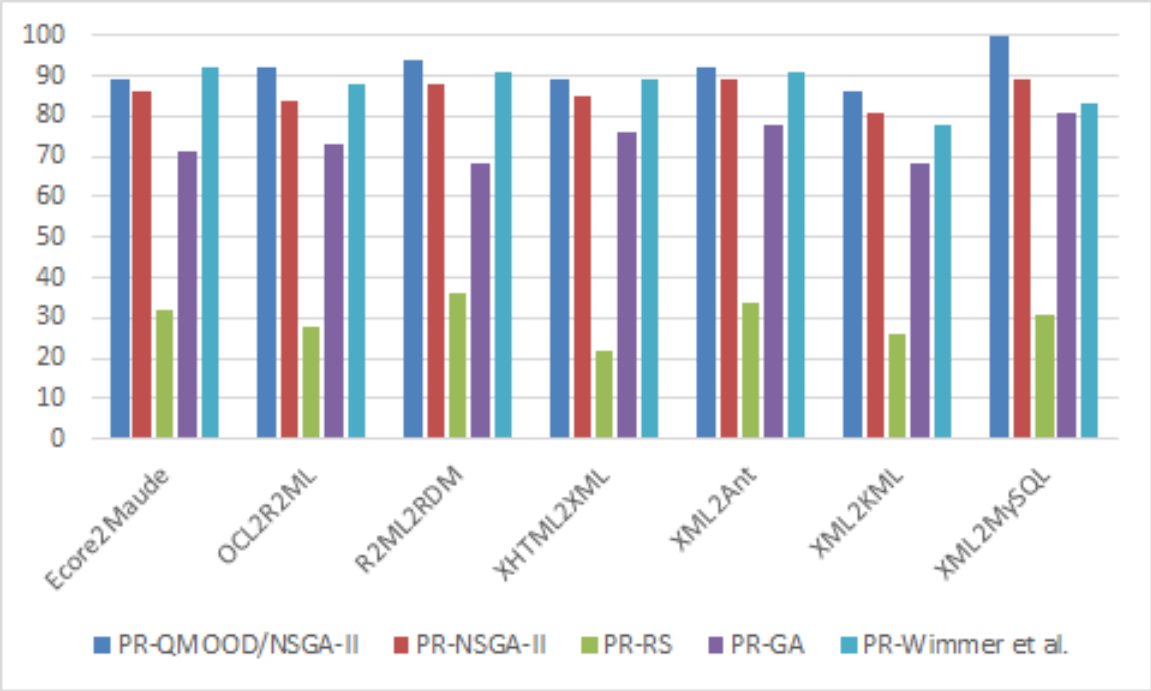


Figure 4.7: Median Precision (PR) over 30 runs on all the 7 ATL programs using the different ATL refactoring techniques with a 95% confidence level ($\alpha < 5\%$).

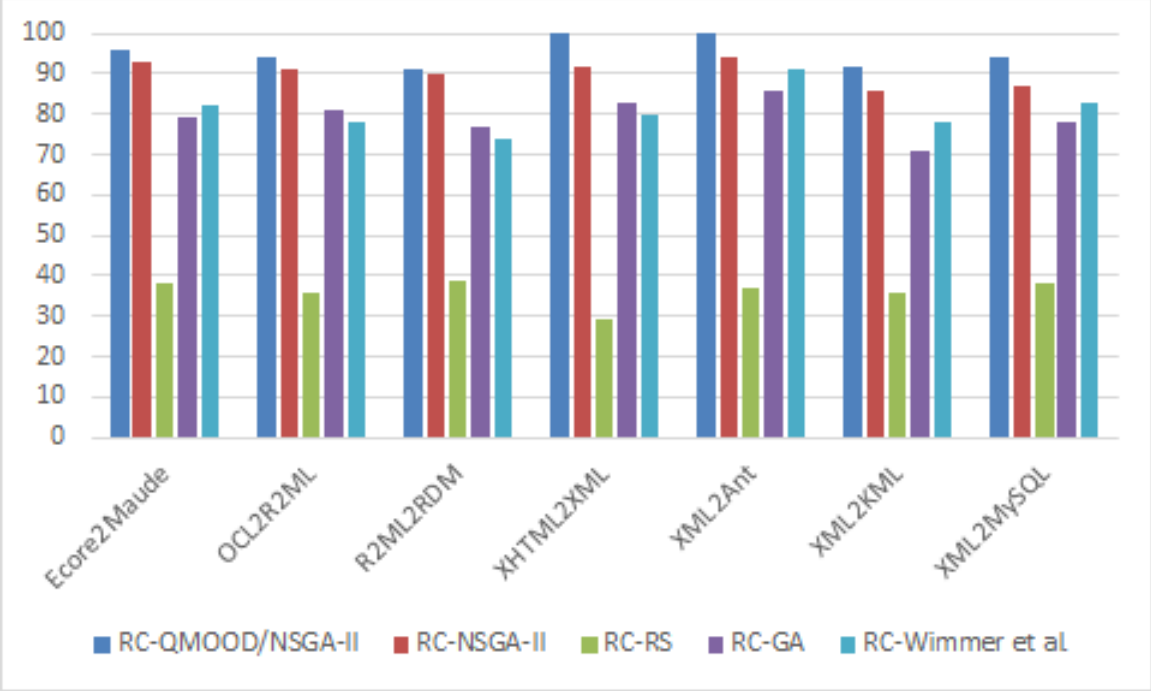


Figure 4.8: Median Recall (RC) over 30 runs on all the 7 ATL programs using the different ATL refactoring techniques with a 95% confidence level ($\alpha < 5\%$).

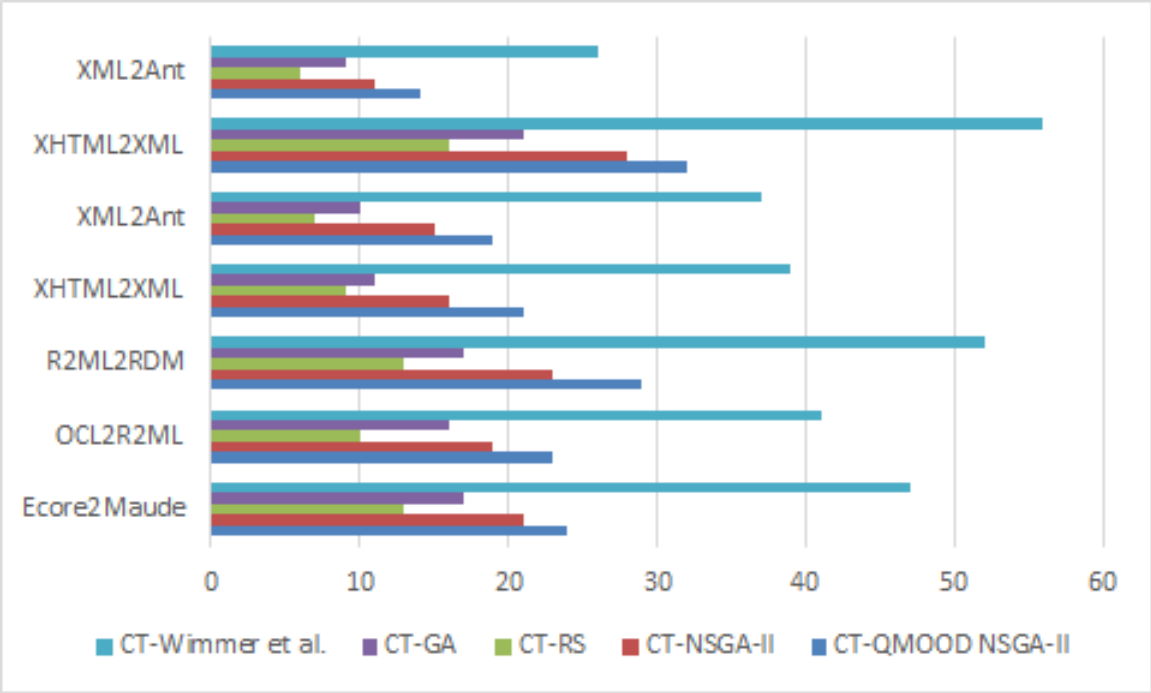


Figure 4.9: Median execution time (CT) over 30 runs on all the 7 ATL programs using the different ATL refactoring techniques with a 95% confidence level ($\alpha < 5\%$).

mended refactorings, we also compared the proposed operations with some expected ones defined manually by the different participants for several ATL code fragments extracted from the seven programs. Figure 4.7 and Figure 4.8 summarize our findings. We found that a considerable number of proposed refactorings, with an average of more than 91% in terms of precision and 96% of recall, were already applied by the software development team and suggested manually (expected refactorings). The recall scores are higher than precision ones since we found that the refactorings suggested manually by developers are incomplete compared to the solutions provided by our automated approach and this is was confirmed by the qualitative evaluation (MC). In addition, we found that the slight deviation with the expected refactorings is not related to incorrect operations but to the fact that different refactoring strategies are equivalent in terms of quality even if the applied refactoring types are different. Furthermore, the use of the fitness function to minimize the number of refactorings may helped to reduce the noise in the recommended solutions and focus mainly on the refactorings which improved the quality metrics.

We decided to evaluate the performance of our approach using evaluation metrics different than the fitness functions (quality metrics) to ensure a fair comparison with existing techniques as detailed in the next research questions. To summarize and answer RQ2, the experimentation results confirm that our QMOOD based multi-objective approach helps the participants to refactor their ATL programs efficiently by finding the relevant refactorings and improve the quality of all the five programs.

Results for RQ3: Figures 4.6, 4.7 and 4.8 confirm the average superior performance of our QMOOD multi-objective approach compared to our previous work based on NSGA-II (and limited to fan-in and fan-out) Alkhazi et al. (2016) and a mono-objective GA aggregating all the objectives in an equal way. Figure 4.6 shows that our approach provides significantly higher manual correctness results (MC) than NSGA-II (as used in Alkhazi et al. (2016)), a mono-objective formulation having MC scores between 89% and 79% on the different ATL programs. The same observation is valid for the precision and recall as described in Figures 4.7 and 4.8. Thus, it is clear that all the four different objectives considered in our formulation are conflicting justifying the outperformance of NSGA-II whether based on QMOOD or not. Furthermore, the results confirm that the QMOOD metrics formulation are more aligned with the preferences of ATL developers than the limited use of fan-in and fan-out.

Since our proposal is based on multi-objective optimization, it is important to evaluate the execution time (CT). It is evident that both NSGA-II adaptations require higher execution time than RS and GA since NSGA-II is considering higher number of objectives and change operators. In addition, the use of QMOOD metrics made the execution time slower comparing to our previous multi-objective work. All the search-based algorithms under comparison were executed on machines with Intel Xeon 3 GHz processors and 8 GB RAM. Overall, RS, GA and NSGA-II algorithms were faster than QMOOD-NSGA-II. In fact, the average execution time for QMOOD-NSGA-II NSGA-II, GA and RS were respectively 23, 19, 15 and 10 minutes. However, the execution for QMOOD-NSGA-II is

reasonable because the algorithm is not executed daily by the developers and the refactoring of ATL programs is not a real-time problem.

To conclude, our QMOOD multi-objective approach provides better results, on average, than our previous multi-objective work, a mono-objective refactoring algorithm aggregating the different objectives (answer to RQ3).

Results for RQ4: Since it is not sufficient to compare our proposal with only search-based work, we compared the performance of QMOOD-NSGA-II with the semi-automated refactoring approach proposed in Wimmer et al. (2012). Figures 4.6, 4.7 and 4.8 summarizes the results of the precision, recall and manual correctness obtained on the 7 ATL programs. The precision of the semi-automated refactoring approach is slightly lower than NSGA-II in all the programs in average of 90%; however the precision scores are lower than our proposal on all the programs. The manual precision of both approaches is comparable and almost the same.

In fact, the good precision achieved by the semi-automated approach can be easily explained by the fact that the refactoring are manually detected by the programmers but just automatically executed. In addition, the recall is lower than QMOOD-NSGA-II because it is time consuming for the programmers to identify a large set of relevant refactorings which is automatically generated using QMOOD-NSGA-II. It is also clear that the semi-automated refactoring approach Wimmer et al. (2012) is time consuming with an average of more than 45 minutes however our QMOOD-NSGA-II algorithms can recommend and apply relevant refactorings in a time frame lower than 25 minutes as described in Figure 4.9. To conclude, our QMOOD-NSGA-III adaption also outperforms, on average, an existing approach not based on meta-heuristic search (RQ4).

Results for RQ5: We have asked the participants to take a post-study questionnaire after completing the refactoring tasks using our multi-objective refactoring tool and all the techniques considered in our experiments. The post-study questionnaires collected the opinions of the participants about their experience in using our approach compared also to

the semi-automated refactoring tool Wimmer et al. (2012) and our previous multi-objective work not based on QMOOD Alkhazi et al. (2016). The post-study questionnaire asked participants to rate their agreement on a Likert scale from 1 (complete disagreement) to 5 (complete agreement) with the following statements:

- The automated refactoring recommendations are a desirable feature in ATL.
- The multi-objective automated manner of recommending refactorings by our approach is a useful and flexible way to refactor ATL model transformation programs compared to semi-automated or manual refactorings.
- The use of QMOOD quality attributes is relevant to improve the quality of ATL programs.

The agreement of the participants was 4.8, 4.4 and 4.8 for the three statements, respectively. This confirms the usefulness of our approach for the software developers considered in our experiments. The remaining questions of the post-study questionnaire were about the benefits and also limitations (possible improvements) of our multi-objective approach. In addition, the questionnaire confirms that the developers found the use of QMOOD attributes is relevant to identify refactoring opportunities for model transformation programs. We summarize in the following the feedback of the developers. Most of the participants mention that our automated approach is faster than semi-automated or manual refactoring since they spent a long time with these techniques to find the locations where refactorings should be applied and which ones to select. For example, developers spend time when they decide to extract a rule to find the elements to move to the newly created rule. Thus, the developers liked the functionality of our tool that helps them to automatically recommend the refactorings and finding quickly the right controlling parameters based on the recommendations. Furthermore, refactorings may affect several locations in the ATL source code, which is a time-consuming task to perform manually, but they can perform it instantly using our tool.

Another important feature that the participants mentioned is that our approach allows them to take the advantages of using multi-objective optimization for ATL refactoring without the need to learn anything about optimization and exploring explicitly the Pareto front to select one "ideal" solution. The implicit exploration of the Pareto front using the Knee point strategy represents an important advantage of our tool.

The participants also suggested some possible improvements to our multi-objective ATL refactoring approach. Some participants believe that it will be very helpful to extend the tool by adding a new feature to apply automatically some regression testing techniques on ATL programs to generate test cases to test applied refactorings. Another possibly suggested improvement is to use some visualization techniques to evaluate the impact of applying a refactoring sequence. In addition, they did not appreciate sometimes the long list of refactoring suggested by our tool since they want to take control of modifying and rejecting some refactorings. In addition, the validation of this long list of refactorings is time-consuming. Finally, the developers also highlighted that it will be interesting to conisder the quality attributes of QMOOD with different weights since they may not be equally important.

4.4.6 Threats to Validity

Following the methodology proposed by Wohlin et al. (2012), there are four types of threats that can affect the validity of our experiments. We consider each of these in the following paragraphs.

4.4.6.1 Conclusion Validity

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. We addressed conclusion threats to validity by performing 30 independent simulation runs for each problem instance and statistically analyzing the obtained results using the Wilcoxon rank sum test with a 95% confidence level (α = 5%). However,

the parameter tuning of the different optimization algorithms used in our experiments creates another internal threat that we need to evaluate in our future work. The parameters' values used in our experiments are found by trial-and-error. However, it would be an interesting perspective to design an adaptive parameter tuning strategy for our approach so that parameters are updated during the execution in order to provide the best possible performance. In addition, our multi-objective formulation treats the different types of refactoring with the same weight in terms of complexity when calculating one of the fitness functions. However, some refactoring types can be more complex than others to apply by developers.

4.4.6.2 Internal Validity

Internal validity is concerned with the causal relationship between the treatment and the outcome. We dealt with internal threats to validity by performing 30 independent simulation runs for each problem instance. This makes it highly unlikely that the observed results were caused by anything other than the applied multi-objective approach.

4.4.6.3 Construct Validity

Construct validity is concerned with the relationship between theory and what is observed. To evaluate the results of our approach, we selected solutions at the knee point when we compared our approach with the mono-objective GA and random search, but the developers may select a different solution based on their preferences to give different weights to the objectives when selecting the best refactoring solution. The different developers involved in our experiments may have divergent opinions about the recommended refactorings in terms of correctness and readability. We considered in our experiments the majority of votes from the developers. For the selection threat, the participant diversity in terms of experience could affect the results of our study. We addressed the selection threat by giving a lecture and examples of ATL refactorings already evaluated with arguments and justification. For the fatigue threat, we did not limit the time to fill the questionnaire

and we also sent the questionnaires to the participants by email and gave them the required time to complete each of the required tasks.

4.4.6.4 External Validity

External validity refers to the generalizability of our findings. In this study, we performed our experiments on seven different ATL programs belonging to different domains and having different sizes. However, we cannot assert that our results can be generalized to other programs, and to other practitioners. Future replications of this study are necessary to confirm our findings. In addition, our study was limited to the use of specific refactoring types and ATL metrics. Future replications of this study are necessary to confirm our findings, e.g., if the general approach is also applicable for OCL-related refactorings Correa et al. (2007); Correa and Werner (2004, 2007); Reimann et al. (2012).

4.5 Conclusion

In this chapter, we propose a novel set of quality attributes to evaluate refactored ATL programs based on the hierarchical quality model QMOOD. We used these conflicting quality attributes to guide the selection of the best refactorings to refactor ATL programs using multi-objective search. We validate our approach on a comprehensive dataset of model transformations. The statistical analysis of our experiments shows that our automated approach recommended useful refactorings based on a benchmark of ATL transformations and compared to random search, mono-objective search formulation, a previous work based on a different formulation of multi-objective search with few quality metrics, and a semi-automated refactoring approach not based on heuristic search.

CHAPTER V

Test Case Selection for ATL Model Transformations

5.1 Introduction

Model-driven engineering (MDE) Bézivin and Gerbé (2001); Brambilla et al. (2017) raised the portability, and maintainability of software systems by using models as first-class entities Hutchinson et al. (2011). The used models can be executed, manipulated, or migrated using recent model transformations advances Schmidt (2006). Nowadays, model transformations are used in a wide spectrum of critical industrial projects Mohagheghi and Dehlen (2008), making their correctness and robustness as a top priority.

To check the correctness of model transformations, several testing techniques have been proposed Lin et al. (2005); Cabot et al. (2010); Guerra et al. (2013); Wimmer and Burgueño (2013); Sahin et al. (2015). Besides the conventional software testing difficulties Bertolino (2007), model transformations have their own additional testing challenges Lin et al. (2005); Baudry et al. (2010) making it harder to automatically generate test cases and execute them efficiently. Several research contributions have discussed the test cases generation issue for model transformations Wang et al. (2013); Fleurey et al. (2009); González and Cabot (2012). The main challenge is the large number of test cases required to ensure the coverage of the source and target meta-model elements as well as of the model transformation rules. The overlap between test cases may results in days or even weeks to complete executing their test suite Elbaum et al. (2000). In practice, developers and testers usually

have limited time to complete certain tasks; the increased pressure to minimize the product's time to market may pose risks of overlooking major expensive defects. Therefore, the quality of test cases is not the only factor to be considered, execution cost is equally important. Furthermore, the overlap between the test cases covering the same rules and elements increases the execution time without improving the efficiency to identify errors. Currently, the current state of the art did not address the problem of test cases selection for model transformations unlike other paradigms such as Object Oriented programming languages.

One possible way to reduce the cost of testing is test cases selection that provided promising results at the code level Bates and Horwitz (1993); Binkley (1995); Yau and Kishimoto (1987); Seawright and Gerring (2008); Goodenough and Gerhart (1975); Yoo and Harman (2007). The primary objective of these techniques is to select a subset of the test cases that maximizes the coverage criteria and minimizes the number of selected test cases. However, test cases selection and prioritization received not enough attention in the MDE community.

In this chapter, we propose a test case selection technique for model transformation programs. We formulate the problem of test cases selection as a multi-objective problem, using NSGA-II, that finds the best combinations of test cases that satisfies two conflicting objectives: (i) maximizing rule coverage and (ii) minimizing test suite's execution time. We evaluated our approach based on a set of model transformation programs extracted from the ATL zoo and previous studies. The results confirm that our test cases selection approach significantly reduce the time to test ATL programs while keeping a high level of coverage.

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

1. This work introduces one of the first studies for selecting test cases for model transformations. To handle the conflicting objectives of coverage and cost, we adapted a multi-objective algorithm to select the test cases maximizing the coverage and mini-

mizing the execution time.

2. We report the results of an empirical study on an implementation of our approach. The obtained results provide evidence to support the claim that our proposal is more efficient, on average, than existing test cases generation approaches in terms of reducing the execution time with high coverage.

5.2 Motivating Example

The ATLAS Transformation Language (ATL) has been chosen as transformation language demonstrator for this work, because it is one of the most widely used transformation languages, both in academia and industry, and there is mature tool support[1] available. ATL is a rule-based language which builds heavily on the Object Constraint Language (OCL), but provides dedicated language features for model transformations which are missing in OCL, like the creation of model elements.

An ATL transformation is mainly composed by a set of *rules*. A rule describes how a subset of the target model should be generated from a subset of the source model. Consequently, a rule consists of an *input* pattern—having an optional *filter* condition—which is matched on the source model and an *output* pattern which produces certain elements in the target model for each match of the input pattern. OCL expressions are used to calculate the values of target elements' features, in the so-called *bindings*.

To further illustrate ATL, we use the BibTeXML to DocBook transformation example, a prominent ATL program taken from ATL Zoo Project (2015). As the name suggests, BibTeXML to DocBook generates a DocBook document from a BibTeXML model. BibTeXML is a schema that describes the model contents of BibTeX using XML syntax to allow users to extend the bibliography data with custom ones. The BibTeXML to DocBook transformation's objective is to create a DocBook document that consists of four

[1] `http://www.eclipse.org/atl`

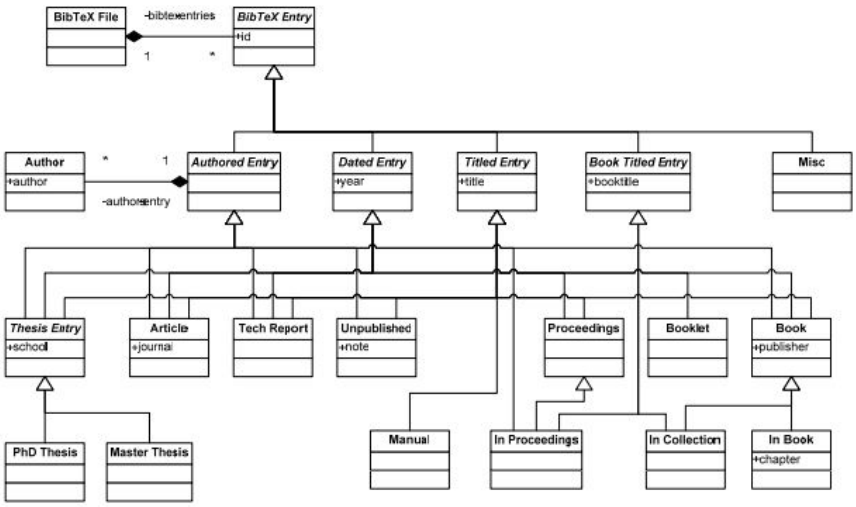


Figure 5.1: The BibTeXML metamodel (taken from INRIA (2005))

sections: (i) reference list, (ii) author list, (iii) title list, and (iv) journal list. An excerpt of the transformation is shown in Listing V.1 and the metamodels of the source and target models are shown in Figures 5.1 and 5.2, respectively. The full details can be found on the documentation section at Eclipse's ATL Transformations Zoo INRIA (2005).

Having this transformation specified, testing is required to find out if the transformation is working as expected for all possible inputs or if there are bugs in the transformation leading to unintended output models for certain input models Baudry et al. (2010). Testing ATL transformations has been discussed in several papers in the past González and Cabot (2012); Guerra (2012); Gogolla et al. (2015); Gogolla and Vallecillo (2011) to mention just a few. However, due to the complex input and output parameters (i.e., the input and output models) as well as sophisticated language semantics of ATL, testing ATL transformations is still a challenge. In particular, different coverage metrics have been proposed such as metamodel element coverage as well as transformation element coverage McQuillan and Power (2009); Guerra (2012). Moreover, many different approaches for test case generation have been proposed in the past showing different advantages and disadvantages (cf. Selim et al. (2012) for a survey). As a result, different approaches may be used to generate test cases, and still, often manually developed test cases for testing particular situations are

Listing V.1: Excerpt of the BibTeXML to DocBook transformation

```
...
rule Main {
  from
      bib : BibTeX!BibTeXFile
  to
    doc : DocBook!DocBook (
      books <- Sequence{boo}
    ),
    boo : DocBook!Book (
      articles <- Sequence{art}
    ),
    art : DocBook!Article (
      title <- 'BibTeXML to DocBook',
      sections_1 <- Sequence{se1, se2, se3, se4}
    ),
    se1 : DocBook!Sect1 (
      title <- 'References List',
      paras <- BibTeX!BibTeXEntry.allInstances()->sortedBy(e | e.id)
    ),
    se2 : DocBook!Sect1 (
      title <- 'Authors list',
      paras <- thisModule.authorSet
    ),
    se3 : DocBook!Sect1 (
      title <- 'Titles List',
      paras <- thisModule.titledEntrySet->collect(e | thisModule.resolveTemp(e, 'title_para'))
    ),
    se4 : DocBook!Sect1 (
      title <- 'Journals List',
      paras <- thisModule.articleSet->collect(e | thisModule.resolveTemp(e, 'journal_para'))
    )
}

rule Author {
  from
    a : BibTeX!Author (
       thisModule.authorSet->includes(a)
      )
  to
    p1 : DocBook!Para (
      content <- a.author
    )
}

rule Article_Title_Journal {
  from
    e : BibTeX!Article (
      thisModule.titledEntrySet->includes(e) and
      thisModule.articleSet->includes(e)
    )
  to
    entry_para : DocBook!Para (
      content <- e.buildEntryPara()
    ),
    title_para : DocBook!Para (
      content <- e.title
    ),
    journal_para : DocBook!Para (
      content <- e.journal
    )
}

...
```

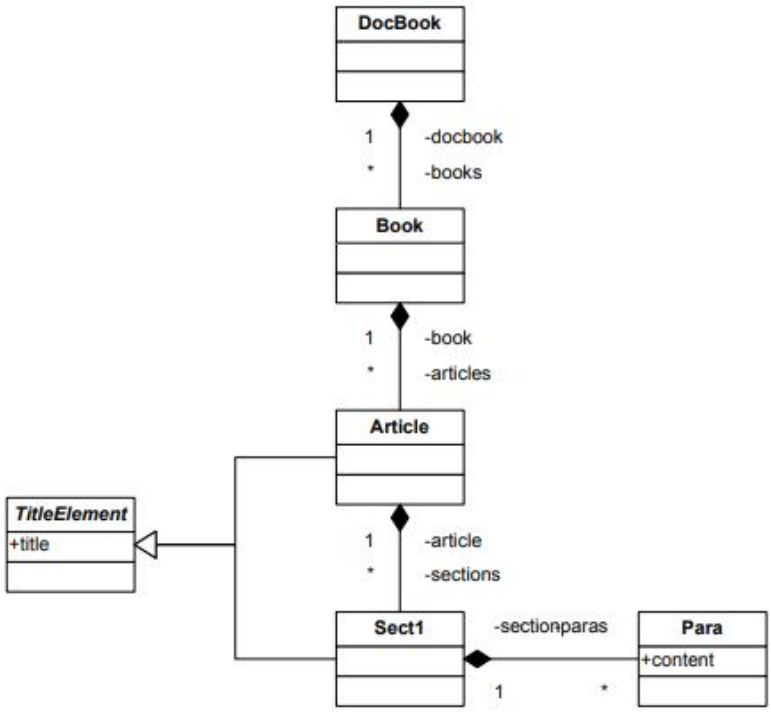


Figure 5.2: The DocBook metamodel (taken from INRIA (2005))

created.

For instance, for the ATL program shown in Listing V.1, we have collected a total of 111 test models where most of them are reused from a previous study on fault localization for ATL Troya et al. (2018a) and some additional models are created to improve the transformation rule coverage. Each model covers specific parts of the transformation program and of the metamodels. An example model is shown in Listing V.2 which should activate the rules dealing with InProceedings entries as well as Article entries.

The number of rules in the example transformation is 9 and the total number of input and output metamodel classes is 29. With the given test suite, we can have a good coverage of the rules and metamodel elements. However, the next question arises: are all given models actually needed for testing the given transformation or is a subset equally effective? Therefore, we propose in the next section an approach which help transformation tester to build and maintain an effective test suite for their ATL transformations.

Listing V.2: Sample Input Test Data

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:BibTeX="BibTeX">
  <BibTeX:InProceedings id="a" year="2016" title ="Automated refactoring of ATL model transformations" booktitle ="MODELS16">
    <authors author="Alkhazi, B."/>
    <authors author="Ruas, T."/>
    <authors author="Kessentini , M."/>
    <authors author="Wimmer, M."/>
    <authors author="Grosky, W."/>
  </BibTeX:InProceedings>

  <BibTeX:Article id="b" year="2017" title ="Model Transformation Modularization as a Many-Objective Optimization Problem" journal="IEEE Transactions on
       Software Engineering">
    <authors author="Fleck, M."/>
    <authors author="Troya, T."/>
    <authors author="Kessentini , M."/>
    <authors author="Wimmer, M."/>
    <authors author="Alkhazi, B."/>
  </BibTeX:Article>
</xmi:XMI>
```

5.3 Test-Cases Selection for Model Transformation

In this section, we first present an overview of our approach including the multi-objective formulation and the solution approach. We also describe briefly our adaptation of NSGA-II to apply on the test case selection problem for ATL transformations.

5.3.1 Approach Overview

The primary objective of our approach is to analyze a test suite and optimize it in order to satisfy certain criteria as illustrated in Figure 5.3. As an input, we take an ATL program and a number of test cases. Then, we pre-process each test case to collect some data about their coverage and execution time, which later will be used as the main constraints for the algorithm. Since these goals are inherently conflicting and we are potentially dealing with a huge search space, consequently, a multi-objective algorithm(NSGA-II) is used to find the Pareto-optimal solutions for this problem. This algorithm and its adaptation to the selection problem is described in the next subsection.

5.3.2 Solution Approach

To illustrate the approach, in particular how we perform the adaptation of NSGA-II to the problem of test case selection, with an example, we will use the example introduced in

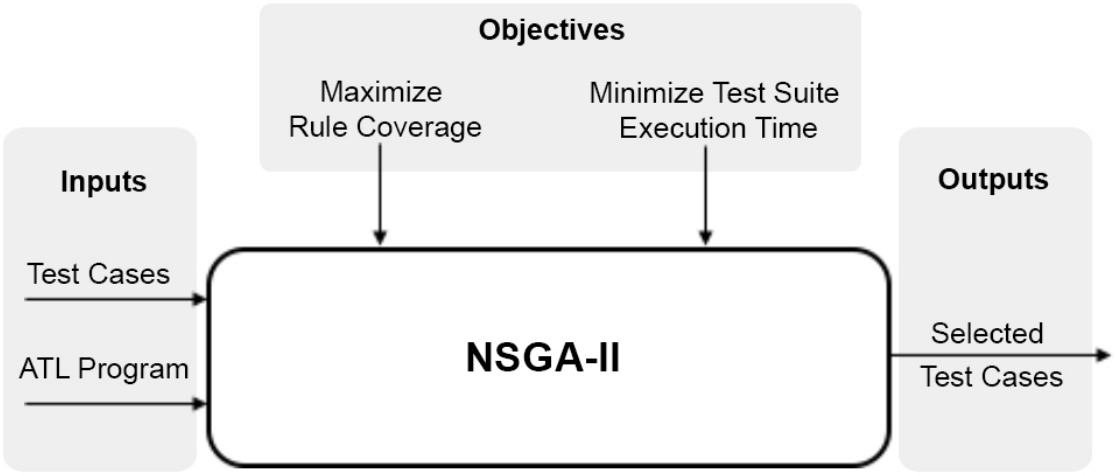


Figure 5.3: Test cases selection overview.

Section 5.2.

Solution Representation: A solution is a sequence of n test cases that are represented in a vector-based fashion, where each dimension represents a test case. A sample test model that are used as an input to the test case is shown in Listing V.2. An example of a solution vector is depicted in Table 5.1. Each vector's dimension represents an execution of a test case to analyze its impact in terms of execution time and rule coverage (Case ID, Execution Time, Covered Rules). For instance, executing the case shown in Listing V.2 (cf. Section 5.2) will cover three rules out of nine (33.33%) and the execution time is 219.7421 ms.

The initial population is randomly selected. The size of the vector V is bound by a maximum number VMAX that is proportional to the program size and the number of test cases.

1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(30, 202.11, Rules[R1,R4])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])

Table 5.1: Example of solution representation

Solution Evaluation: Solutions need to be evaluated to keep the fittest ones and eliminate/replace the lowest. We have two objectives; thus we are using two fitness functions:

$$f_1 = Max(RuleCoverage)$$

Objective 1: Maximize rules coverage, we trace the triggered rules during the execution of every test case to determine the rules covered by the entire test suite. We then measure the percentage of rule coverage after eliminating duplicates.

$$f_2 = Min(ExecutionTime)$$

Objective 2: Minimize test suite execution time. As the test cases are executed, we keep track of the time needed to complete the testing activities; solutions that require less time are preferred.

Solution Variation: Exploring the search space to look for better potential candidate solutions requires using variation operations such as the crossover and mutation. A one-point crossover operation is used as follows: two parent solutions are selected and each one is split at a random point before crossing the split parts between the two parents to create two new children. We use the bit-string mutation operator to pick at least one of the vector's dimensions and replace it randomly with a test case. An illustration to the mutation operator is depicted in Table 5.2.

1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(30, 202.11, Rules[R1,R4])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])

↓

1	TestCase(8, 342.08, Rules[R7,R1,R5,R7,R7])
2	TestCase(6, 170.05, Rules[R8,R6])
3	TestCase(2, 542.43, Rules[R10,R9,R3,R7,R8,R2])

Table 5.2: Example of applying mutation operator to the vector previously shown in Table 5.1

5.4 Evaluation

In order to evaluate our approach for test case selection, we conducted a set of experiments based on six ATL transformation programs, their size and structures are detailed in Subsection 5.4.2. The following subsections describe the research questions, followed by

the experimental setup and the obtained results. Finally, a discussion on threats to validity of our experiments is given.

5.4.1 Research Questions

We defined three research questions that address the performance, suitability, and scalability ISO (2011). The three research questions are as follows:

- **RQ1: How does our proposed multi-objective approach perform compared to a mono-objective selection algorithm?** To ensure that the objectives are conflicting, we use a single fitness function by aggregating the two normalized objectives. If the results are the same or the mono-objective formulation performed better than their multi-objective counterparts, we conclude that the latter is not needed.

- **RQ2: What is the cost-effectiveness of using our multi-objective test case selection approach?** Reducing the test suite is clearly beneficial when it comes to execution time, however, we need to keep an eye on the ability of the new test suite to reveal faults as it contain less number of test cases. Moreover, the selection process should not take more than the time gained by reducing the test suite Leung and White (1989).

- **RQ3: How does our proposed approach perform compared to a retest-all approach?** Since our hypothesis is to reduce the time and number of test cases for testing model transformation, we compared our approach with a traditional testing technique for ATL model transformations consisting of running all the pre-defined test cases after every change made to the transformations program.

5.4.2 Case Studies

To evaluate our research questions, six case studies have been used. Four cases taken from the ATL Zoo repository Project (2015), the remaining programs were taken from an

existing work for spectrum-based fault localization Troya et al. (2018b). The transformations used are diverse in terms of size, application domain, number of dependencies among their transformation artifacts, and structure. We briefly describe in the following the different transformation programs used.

- **UML2ER**: This transformation generates Entity Relationship (ER) diagrams from UML Class diagrams.
- **XSLT2XQuery**: The XSLT to XQuery transformation produces models based on the XQuery meta-model from XSLT code.
- **BibTeX2DocBook**: This transformation generates a DocBook composed document from a BibTeXML model. We have already introduced this transformation in Section 5.2.
- **XML2MySQL**: XML to MySQL transformation translates XML representations of the structure of domain model into actual MySQL representations.
- **CPL2SPL**: This program is a relatively complex transformation as it handles several aspects of two telephony DSLs: SPL and CPL Project (2015).
- **Ecore2Maude**: In this transformation, Ecore metamodels are used to generate Maude Clavel et al. (2007) specifications.

Table 5.3 summarizes the number of rules, number of helpers, number of lines of code (LoC), and number of classes in both input and output metamodels for every case study.

5.4.3 Experimental settings

The efficiency of search algorithms can be significantly influenced by parameter settings Arcuri and Fraser (2013). Selecting the right population size, stopping criterion, crossover and mutation rate is essential to avoid premature convergence. We used MOEA

Framework v2.12 Hadka (2012) for our experiments, and performed several experiments with various population sizes; 50, 100, 250, 500. The stopping criterion was set to 100k evaluations for all algorithms. For crossover and mutation, we used 0.7 and 0.3 probabilities, respectively.

MOEA Framework's default parameter setting values were used for all other parameters. Metaheristic algorithms are stochastic optimizers and may provide different results for the same problem. Thus, for each configuration, we performed 30 independent runs for every problem instance. Later, we statistically analyzed the obtained results using Wilcoxon test Arcuri and Fraser (2013) with $\alpha = 5\%$ (i.e. 95% confidence level). All experiments have been executed on Macbook Pro machine with 2.5 GHz Intel Core i7 processor, 16 GB 1600 MHz DDR3 RAM, and 500 GB SSD. The Eclipse Modeling Tools version Neon.3 (Release 4.6.3) was used in addition to ATL plugin (version 3.7.0) and ATL/EMFTVM (version 4.0.0).

5.4.4 Results and Discussions

Results for RQ1. In this subsection, we evaluate the performance of our NSGA-II adaptation to a mono-objective genetic programming formulation, where the normalized values of the time and coverage metrics are aggregated into one fitness function. Tables 5.6 and 5.7 show an overview of the average results of the 30 runs for each algorithm.

The mono-objective algorithm reduced test suite size 92.31% to 98.82% of the original

ID	Name	#Rules	#Helpers	#LoC	#MM Classes Input - Output
CS1	UML2ER	8	0	55	4 - 8
CS2	XSLT2Query	7	0	170	16 - 18
CS3	BibTex2DocBook	9	0	263	21 - 8
CS4	XML2MySQL	6	10	294	5 - 8
CS5	CPL2SPL	19	6	518	33 - 77
CS6	Ecore2Maude	39	41	1372	13 - 45

Table 5.3: Case studies and their sizes and structures.

test suite. Also, the computational cost is reduced by percentages ranges between 26.02% for CS4 to up to 97.99% for CS6. In all case studies, mono-objective's computational time was better, which is intuitive as more objectives usually requires more computation time to evaluate the different Pareto front options.

An other factor that influenced the computational time is the number of selected cases; higher number of test cases in a test suite leads to a higher computational cost. In all case studies, NSGA-II selected more cases compared to mono-objective GA formulation. An interesting observation here is that when the size of the case study increases, the difference in time reduction vanishes between the two algorithms as shown in larger cases such as CS5 and CS6. Worth mentioning that the time for both algorithms is calculated by adding the test suite execution time to the algorithms analysis time. If we have a closer look at CS2 and CS4, we see that the multi-objective time reduction was 12.97% and 15.61%, respectively. Both case studies have small number of test cases already (Table 5.4), thus the algorithm's computational time nearly exceeded the time gained by reducing the test suite.

In regard to rules coverage (Table 5.6), however, values are significantly better in our adaptation's favor, regardless of the test case's size or structure. The average coverage for the six case studies are 82.4% compared to 57.8% on average for the mono-objective formulation.

From these results, we conclude that the two considered objectives are conflicting and

Case Study	**#Test Cases**	**Max Possible Coverage (%)**	**Execution Time for all**
CS1	105	100	697.99
CS2	13	100	304.46
CS3	111	100	4358.36
CS4	17	100	617.02
CS5	108	94.73	9120.79
CS6	171	100	34994.96

Table 5.4: Test cases data for each case study.

therefore a multi-objective formulation is necessary to balance between the cost and coverage, which answers **RQ1**.

Results for RQ2. To answer this question, we created multiple mutations for each case study by manually introducing bugs at different locations in the transformations using the approach and operators proposed in Mottu et al. (2006); Troya et al. (2015). Table 5.5 summarizes the mutation operations used in our experiments, further details about the operators and their possible impact on the transformation is discussed in Troya et al. (2015). In addition to the manually created mutants, we also reused some of the existing mutations proposed in Troya et al. (2018b).

In total we had 104 mutants, where each mutant consist of one or more changes compared to the original transformation. Note that these are semantic mutations, thus, there will be no compilation or run-time error and we will wait until the execution of the transformation is complete to evaluate the results.

Table 5.8 summarizes the results for both approaches. Mono-objective algorithm was able to reveal faults 53.5% on average for all case studies. Mono-objective formulation already covers less rules as shown in table 5.6 and that automatically led to hindering its ability to detect bugs 46.5% of the time. In contrast, the multi-objective adaptation, reveals faults in 85.49% of the time. From these results, we see that reducing test suite cost by 54.26% on average, provides a good fault revealing percentage. This summarizes the answer to **RQ2**.

Results for RQ3. Running all test cases is the safest route, assuming no changes in

Concept	Mutation Operators
Matched Rule	Addition, Deletion, Name Change
In Pattern Element	Addition, Deletion, Class Change, Name Change
Filter	Addition, Deletion, Condition Change
Out Pattern Element	Addition, Deletion, Class Change, Name Change
Binding	Addition, Deletion, Value Change, Feature Change

Table 5.5: Mutations for ATL Transformations (From Troya et al. (2015)).

the specs have been made. However, this demands the most computational time and often companies do not have this option. Table 5.7 shows the significant size reduction in all case studies, this suggests that there are some unnecessary test cases in the original test suite. Which might be due to overlapping cases or because of changes in the transformation Leung and White (1990), without updating the test suite by updating the correlated test cases leading to obsolete ones. Also, we can see that the computational time was substantially reduced (>70%) for some case studies (CS3, CS5, and CS6), and reduced by double digits for the rest. We see that the larger search space (application size and test suite), the better results we are getting for our multi-objective adaptation.

As discussed for the results of RQ1 and RQ2, the coverage results (Table 5.6) and fault revealing capability (Table 5.8) shows strong evidence that we are getting high coverage and fault detection rates despite the big reduction in size and computational time. Note that for CS5, the maximum possible coverage when we run all available test cases is 94.73% (Table 5.3). Thus, the coverage and fault revealing results have room for improvement with more test cases to select from. Furthermore, in our formulation, we gave the same importance to both metrics (time and coverage). However, in certain practical applications, more weight may be given to the coverage, which will help in revealing more bugs.

ID	**Retest-All**		**Mono-objective**		**Multi-objective**	
	Coverage	Time	Coverage	Time	Coverage	Time
CS1	100	697.99	60.0	262.41	86.0	433.24
CS2	100	304.46	57.14	168.26	79.4	264.99
CS3	100	4358.36	55.55	327.45	81.4	559.57
CS4	100	617.02	50	456.47	84.7	520.70
CS5	94.73	7339.36	63.15	417.21	77.4	736.70
CS6	100	23522.79	61.53	473.65	85.6	903.99

Table 5.6: Average coverage and execution time for the three approaches.

5.4.5 Threats to validity

Internal Validity. This threat is concerned with the factors that might influence the results of our evaluation. The stochastic nature of our approach and the parameter tuning might be considered an internal validity threat. To address this problem, we performed 30 independent simulation for each problem instance, making it unlikely that the observations are not caused by the multi-objective formulation. Another internal threat to consider is concerned with using search-algorithms for test suite optimization. No particular meta-heuristic approach is recommended for test case selection problems, however, evolutionary algorithms proved to be successful for various multi- and many-objectives search problems in previous studies Kalyanmoy et al. (2001).

Construct Validity. The relationship between what we observe and theory is within the domain of this threat. We used well known performance measures such as computational cost and code coverage in our objective functions. To compare the different approaches, we additionally used test suite size and fault coverage to compare the performances. We plan to further investigate different metrics and performance measures in our future work. The absence of similar work in the area of model transformation to select test cases is another construct threat, thus we compared our work with mono-objective algorithm and retest-all approach to tackle this issue.

Conclusion Validity. Our ability to draw conclusions for the observed data is governed

ID	Mono-objective		Multi-objective	
	Time (%)	Size (%)	Time (%)	Size (%)
CS1	62.41	96.88	37.93	93.75
CS2	44.73	92.31	12.97	84.62
CS3	84.17	97.62	72.95	95.24
CS4	26.02	94.12	15.61	87.65
CS5	94.32	98.82	89.96	97.65
CS6	97.99	98.18	96.16	96.36

Table 5.7: Average percentage of time and test suite size reduction.

by conclusion validity. To address this threat, we analyzed the obtained results statistically with Welch's t-test with 95% confidence level (α = 5%). We used a popular trial-and-error method in the literature Eiben and Smit (2011), however, choosing different parameters may affect the results. However, we may use in the future an adaptive parameter tuning strategy where the values are updated during the execution to find the best possible combinations for an ultimate performance.

External Validity. This threat is concerned with our ability to generalize the findings. We used six case studies, four of them are taken from ATL Zoo repository which is widely used in research. The remaining two test cases are also used previously by a number of researchers in the field of MDE. The test cases are different in size, structure and application domain. Yet, we can not assert that our results are generalizable for all cases. Future empirical studies are required to confirm our findings.

5.5 Conclusion

In this chapter, we propose a test cases selection approach for model transformations based on multi-objective search. We used the non-dominated sorting genetic algorithm (NSGA-II) to find the best trade-offs between two conflicting objectives: (1) maximize the coverage of rules and (2) minimize the execution time of the selected test cases. We validated our approach on several evolution cases of medium and large ATL programs. The results showed a significant reduction on the execution time while maintaining a good

ID	# Mutations	Mono-objective	Multi-objective
CS1	13	61.1	86.6
CS2	14	45.0	78.5
CS3	28	66.6	89.16
CS4	10	56.25	85.62
CS5	22	62.5	83.52
CS6	17	40.9	89.54

Table 5.8: Average percentage of fault revealing capabilities of the different approaches.

testing performance.

CHAPTER VI

Conclusion

6.1 Summary

The main contribution of this dissertation is to propose a framework to enable the automatic modularization, refactoring and testing of model transformation programs. The main three components are summarized in the next subsections.

6.1.1 Modularization of Model Transformations

Modularizing large transformations can improve readability, maintainability and testability of transformations. However, most publicly available transformations do not use modularization even though most transformation languages support such a concept. One reason for this lack of adoption may be the complexity this task entails. In chapter III, we introduced a new automated search-based software engineering approach based on NSGA-III to tackle the challenge of model transformation modularization. Specifically, we formulate the problem as a many-objective optimization problem and use search-based algorithms to calculate a set of Pareto-optimal solutions based on four quality objectives: the number of modules in the transformation, the difference between the lowest and highest numbers of responsibilities in a module, the cohesion ratio and the coupling ratio. We have applied and evaluated our approach for ATL, a rule-based model transformation language. The evaluation consists of seven case studies and two user studies with participants from

academia and engineers from Ford. Our results show that modularizing model transformations require a sophisticated approach and that our approach produces good results. Furthermore, the use of modularized transformations versus non-modularized ones can reduce the complexity to perform common tasks in model-driven engineering and can improve productiveness in terms of time needed to perform these tasks.

6.1.2 Automatic Refactoring of ATL Model Transformations

In chapter IV, we proposed an automated approach for refactoring ATL programs to find a trade-off between different conflicting objectives. We have also adapted an existing quality model, QMOOD, for the case of model transformations to guide the search for relevant refactorings. Our automated approach allows developers to benefit from search-based refactoring tools without manually identifying refactoring opportunities. To evaluate the effectiveness of our tool, we conducted a user study on a set of software developers who evaluated the tool and compared it with random search, a multi-objective adaption based on two quality metrics, an existing mono-objective formulation, and an approach not based on heursitic search. Statistical analysis of our experiments showed that our proposal performed significantly better than random search, our previous multi-objective work not based on QMOOD Alkhazi et al. (2016), a mono-objective formulation and Wimmer et al. (2012) with an average precision and recall of 89% and 95% respectively when compared to manual solutions provided by a set of developers. The software developers, who participated in our experiments, confirmed also the relevance of the suggested refactorings as an outcome of a survey study.

6.1.3 Test case selection for ATL Model transformations

In chapter V, we proposed a test case selection approach for model transformations by considering transformation rule coverage as well as execution time spent for executing the test cases. The evaluation based on several cases shows a drastic speed-up of the testing

process while still showing a good testing performance.

6.2 Future Work

The promising results of our approach in chapter III raise to several future research lines. First of all, we will further investigate the possibilities for refactoring ATL transformations based on quality metrics. In particular, we plan to optimize ATL transformations through refactoring using performance and memory consumption. Furthermore, we are interested in how our proposed modularization metamodel can be generalized as a template in which developers can integrate other transformation languages, making our approach more broadly accessible. In particular, what is needed for adding support for additional transformation languages is the conversion transformations from language X to the modularization metamodel and vice versa. Of course, the main challenge is to detect dependencies which are not explicitly represented in the transformation programs. Estimating the complexity of the dependency discovery in other transformation languages such as QVT and ETL is considered as an interesting line of future work. Moreover, the modularization metamodel may be further abstracted to form a general framework for modularization problems which may be instantiated for particular structures. Such an approach would allow not only to modularize transformations but other artefacts used in MDE such as models, metamodels Fleck et al. (2016b), and even megamodels Bézivin et al. (2004).

A different approach could be followed to give names to the modules created by our approach. In this version, such names are random String values, what can be changed by users in a post-processing step. We will further study if assigning other names is more useful for the modularization usability Feldthaus and Møller (2013); Thies and Roth (2010), such as assigning names composed of rules names within the module or names of the classes from the input and output pattern elements of the rules. In any case, our evaluation has demonstrated that it is easier and faster for developers to localize the relevant rules using modularized ATL programs because they regroup together semantically similar rules and

helpers. Thus, the name of the created modules is not as important as the way how the rules and helpers are grouped together. For instance, we found that most of the changes to fix specific bugs were localized in rules that are part of the same module.

Secondly, future work inspired from the work in chapter IV involves the extension of our approach to support additional refactoring types. Also, an integration of an automated regression testing mechanism is useful since it is important to test the refactorings introduced to the ATL programs. Furtheromore, we will address the problem of identifying antipatterns in ATL programs rather than just relying on quality attributes.

Finally, in regard to test case selection, we see several dimensions to explore. First, the combination of test generation and test selection techniques is of interest. This would allow to automatically reduce the test suits when they are generated which would allow to concentrate the generation on cases which are not already covered. Second, adding further objectives such as trace diversity in the search process may be helpful for other approaches such as fault localization approaches Troya et al. (2018b). Moreover, further studies considering other model transformation languages may be of interest to see how generalizable our approach is.

BIBLIOGRAPHY

Abdeen, H., Ducasse, S., Sahraoui, H., and Alloui, I. (2009). Automatic package coupling and cycle minimization. In *2009 16th Working Conference on Reverse Engineering*, pages 103–112. IEEE.

Abdeen, H., Varró, D., Sahraoui, H. A., Nagy, A. S., Debreceni, C., Ábel Hegedüs, and Ákos Horváth (2014). Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 289–300.

Agrawal, H., Horgan, J. R., Krauser, E. W., and London, S. A. (1993). Incremental regression testing. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 348–357. IEEE.

Alhwikem, F. H. M., Paige, R. F., Rose, L. M., and Alexander, R. D. (2016). A systematic approach for designing mutation operators for mde languages.

Alkhazi, B., Ruas, T., Kessentini, M., Wimmer, M., and Grosky, W. I. (2016). Automated refactoring of atl model transformations: a search-based approach. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 295–304. ACM.

Allilaire, F., Bézivin, J., Jouault, F., and Kurtev, I. (2006). ATL-eclipse support for model transformation. In *Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006*.

Almendros-Jiménez, J. M. and Becerra-Terón, A. (2016). Automatic generation of Ecore models for testing ATL transformations. In *Proceedings of the International Conference on Model and Data Engineering (MEDI)*, pages 16–30. Springer.

Anquetil, N. and Lethbridge, T. C. (1999). Experiments with clustering as a software remodularization method. In *Sixth Working Conference on Reverse Engineering (Cat. No. PR00303)*, pages 235–255. IEEE.

Arcuri, A. and Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 1–10. IEEE.

Arcuri, A. and Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250.

Arcuri, A. and Fraser, G. (2013). Parameter tuning or default values? an empirical investigation in search-based software engineering. *Empirical Software Engineering*, 18(3):594–623.

Arendt, T., Biermann, E., Jurack, S., Krause, C., and Taentzer, G. (2010). Henshin: advanced concepts and tools for in-place emf model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 121–135. Springer.

Baki, I., Sahraoui, H. A., Cobbaert, Q., Masson, P., and Faunes, M. (2014). Learning implicit and explicit control in model transformations by example. In *International Conference on Model Driven Engineering Languages and Systems*, pages 636–652.

Ball, T. (1998). On the limit of control flow analysis for regression test selection. *ACM SIGSOFT Software Engineering Notes*, 23(2):134–142.

Bansiya, J. and Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17.

Bates, S. and Horwitz, S. (1993). Incremental program testing using program dependence graphs. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 384–396. ACM.

Baudry, B., Dinh-Trong, T., Mottu, J.-M., Simmonds, D., France, R., Ghosh, S., Fleurey, F., and Le Traon, Y. (2006). Model transformation testing challenges. In *Proceedings of the ECMDA Workshop on Integration of Model Driven Development and Model Driven Testing*.

Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y., and Mottu, J.-M. (2010). Barriers to systematic model transformation testing. *Communications of the ACM*, 53(6):139–143.

Bechikh, S., Said, L. B., and Ghédira, K. (2011). Searching for knee regions of the pareto front using mobile reference points. *Soft Computing*, 15(9):1807–1823.

ben Fadhel, A., Kessentini, M., Langer, P., and Wimmer, M. (2012). Search-based detection of high-level model changes. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 212–221. IEEE.

Benedusi, P., Cmitile, A., and De Carlini, U. (1988). Post-maintenance testing based on path change analysis. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 352–361. IEEE.

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, pages 85–103. IEEE Computer Society.

Bézivin, J. and Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE)*, pages 273–280. IEEE.

Bézivin, J., Jouault, F., and Valduriez, P. (2004). On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

Binkley, D. (1995). Reducing the cost of regression testing by semantics guided test case selection. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 251–260. IEEE.

Bisbal, J., Lawless, D., Wu, B., and Grimson, J. (1999). Legacy information systems: Issues and directions. *IEEE software*, 16(5):103–111.

Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., and O'Sullivan, D. (1997). A survey of research into legacy system migration. *Technique report*.

Biswas, S., Mall, R., Satpathy, M., and Sukumaran, S. (2009). A model-based regression test selection approach for embedded applications. *ACM SIGSOFT Software Engineering Notes*, 34(4):1–9.

Biswas, S., Mall, R., Satpathy, M., and Sukumaran, S. (2011). Regression test selection techniques: A survey. *Informatica*, 35(3).

Bonet, N., Garcés, K., Casallas, R., Correal, M. E., and Wei, R. (2018). Influence of programming style in transformation bad smells: mining of etl repositories. *Computer Science Education*, 28(1):87–108.

Bordbar, B., Draheim, D., Horn, M., Schulz, I., and Weber, G. (2005). Integrated model-based software development, data access, and data migration. In *International Conference on Model Driven Engineering Languages and Systems*, pages 382–396. Springer.

Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., and Chikha, S. B. (2013). Competitive coevolutionary code-smells detection. In *International Symposium on Search Based Software Engineering*, pages 50–65. Springer.

Boussaïd, I., Siarry, P., and Ahmed-Nacer, M. (2017). A survey on search-based model-driven engineering. *automated software engineering*, 24(2):233–294.

Bowman, M., Briand, L. C., and Labiche, Y. (2007). Multi-objective genetic algorithm to support class responsibility assignment. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 124–133. IEEE.

Bowman, M., Briand, L. C., and Labiche, Y. (2010). Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 3(1):1–207.

Brodie, M. L. and Stonebraker, M. (1995). *Legacy Information Systems Migration: Gateways, Interfaces, and the Incremental Approach*. Morgan Kaufmann Publishers Inc.

Brooks, F. and Kugler, H. (1987). *No silver bullet*. April.

Brottier, E., Fleurey, F., Steel, J., Baudry, B., and Le Traon, Y. (2006). Metamodel-based test generation for model transformations: an algorithm and a tool. In *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE)*, pages 85–94. IEEE.

Brown, W. H., Malveau, R. C., McCormick, H. W., and Mowbray, T. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc.

Bryant, B. R., Gray, J. G., Mernik, M., Clarke, P., and Karsai, G. (2011). Challenges and directions in formalizing the semantics of modeling languages. *Computer Science and Information Systems*, 8(2):225–253.

Burgueño, L., Troya, J., Wimmer, M., and Vallecillo, A. (2015). Static fault localization in model transformations. *IEEE Transactions on Software Engineering*, 41(5):490–506.

Cabot, J., Clarisó, R., Guerra, E., and De Lara, J. (2010). Verification and validation of declarative model-to-model transformations through invariants. *Journal of Systems and Software*, 83(2):283–302.

Cicchetti, A., Di Ruscio, D., Eramo, R., and Pierantonio, A. (2010). Jtl: a bidirectional and change propagating transformation language. In *International Conference on Software Language Engineering*, pages 183–202. Springer.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2007). *All about maude-a high-performance logical framework: how to specify, program and verify systems in rewriting logic*. Springer-Verlag.

Cohen, J. (1988). Statistical power analysis for the behavioral sciences 2nd edn.

Comella-Dorda, S., Wallnau, K., Seacord, R. C., and Robert, J. (2000). A survey of legacy system modernization approaches. Technical report, Carnegie-Mellon univ pittsburgh pa Software engineering inst.

Correa, A. and Werner, C. (2004). Applying refactoring techniques to uml/ocl models. In *International Conference on the Unified Modeling Language*, pages 173–187. Springer.

Correa, A. and Werner, C. (2007). Refactoring object constraint language specifications. *Software & Systems Modeling*, 6(2):113–138.

Correa, A., Werner, C., and Barros, M. (2007). An empirical study of the impact of ocl smells and refactorings on the understandability of ocl specifications. In *International Conference on Model Driven Engineering Languages and Systems*, pages 76–90. Springer.

Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., and Varró, D. (2002). Viatra-visual automated transformations for formal verification and validation of uml models. In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pages 267–270. IEEE.

Cuadrado, J. and García Molina, J. (2008). Approaches for Model Transformation Reuse: Factorization and Composition. In *Proc. of ICMT*, volume 5063, pages 168–182.

Cuadrado, J. S., Guerra, E., and de Lara, J. (2014a). A component model for model transformations. *IEEE Trans. Software Eng.*, 40(11):1042–1060.

Cuadrado, J. S., Guerra, E., and de Lara, J. (2014b). Uncovering errors in atl model transformations using static analysis and constraint solving. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 34–44. IEEE.

Cuadrado, J. S., Guerra, E., and de Lara, J. (2017). Static analysis of model transformations. *IEEE Transactions on Software Engineering*, 43(9):868–897.

Cuadrado, J. S. and Molina, J. G. (2009). Modularization of model transformations through a phasing mechanism. *SoSyM*, 8(3):325–345.

Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

De Lara, J. and Vangheluwe, H. (2002). Atom 3: A tool for multi-formalism and metamodelling. In *International Conference on Fundamental Approaches to Software Engineering*, pages 174–188. Springer.

de Souza, L. S., Prudêncio, R. B., and Barros, F. d. A. (2014). A hybrid binary multi-objective particle swarm optimization with local search for test case selection. In *Proceedings of the Brazilian Conference on Intelligent Systems*, pages 414–419. IEEE.

Deb, K. and Jain, H. (2012). Handling many-objective problems using an improved nsga-ii procedure. In *Evolutionary computation (CEC), 2012 IEEE congress on*, pages 1–8. IEEE.

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

Debreceni, C., Ráth, I., Varró, D., Carlos, X. D., Mendialdua, X., and Trujillo, S. (2016). Automated model merge by design space exploration. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633*, pages 104–121.

Denil, J., Jukss, M., Verbrugge, C., and Vangheluwe, H. (2014). Search-based model optimization using model transformations. In *International Conference on System Analysis and Modeling*, pages 80–95.

Ehrig, K., Küster, J. M., and Taentzer, G. (2009). Generating instance models from meta models. *Software & Systems Modeling*, 8(4):479–500.

Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

Elbaum, S., Malishevsky, A. G., and Rothermel, G. (2000). *Prioritizing test cases for regression testing*, volume 25. ACM.

Farooq, U. and Lam, C. P. (2009). Evolving the quality of a model based test suite. In *Proceedings of the International Conference on Software Testing, Verification, and Validation Workshops*, pages 141–149. IEEE.

Faunes, M., Cadavid, J. J., Baudry, B., Sahraoui, H. A., and Combemale, B. (2013a). Automatically searching for metamodel well-formedness rules in examples and counter-examples. In *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems - Volume 8107*, pages 187–202.

Faunes, M., Sahraoui, H. A., and Boukadoum, M. (2013b). Genetic-programming approach to learn model transformation rules from examples. In *International Conference on Theory and Practice of Model Transformations*, pages 17–32.

Feldthaus, A. and Møller, A. (2013). Semi-automatic rename refactoring for javascript. In *ACM SIGPLAN Notices*, volume 48, pages 323–338. ACM.

Finot, O., Mottu, J.-M., Sunyé, G., and Attiogbé, C. (2013). Partial test oracle in model transformation testing. In *International Conference on Theory and Practice of Model Transformations*, pages 189–204. Springer.

Fischer, K., Raji, F., and Chruscicki, A. (1981). A methodology for retesting modified software. In *Proceedings of the National Telecommunications Conference*, pages 1–6.

Fischer, K. F. (1977). A test case selection method for the validation of software maintenance modifications. In *Proceedings of 1st International Computer Software and Applications Conference (COMPSAC)*, pages 421–426.

Fleck, M., Troya, J., Kessentini, M., Wimmer, M., and Alkhazi, B. (2017). Model transformation modularization as a many-objective optimization problem. *IEEE Transactions on Software Engineering*, 43(11):1009–1032.

Fleck, M., Troya, J., and Wimmer, M. (2015). Marrying search-based optimization and model transformation technology. *Proc. of NasBASE*, pages 1–16.

Fleck, M., Troya, J., and Wimmer, M. (2016a). Search-based model transformations with momot. In *International Conference on Theory and Practice of Model Transformations*, pages 79–87. Springer.

Fleck, M., Troya, J., and Wimmer, M. (2016b). Towards generic modularization transformations. In *Companion Proceedings of the 15th International Conference on Modularity*, pages 190–195. ACM.

Fleurey, F., Baudry, B., Muller, P.-A., and Le Traon, Y. (2009). Qualifying input test data for model transformations. *Software & Systems Modeling*, 8(2):185–203.

Fleurey, F., Breton, E., Baudry, B., Nicolas, A., and Jézéquel, J.-M. (2007). Model-driven engineering for software migration in a large industrial context. In *International Conference on Model Driven Engineering Languages and Systems*, pages 482–497. Springer.

Fleurey, F., Steel, J., and Baudry, B. (2004). Validation in model-driven engineering: testing model transformations. In *Proceedings of the First International Workshop on Model, Design and Validation*, pages 29–40. IEEE.

France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society.

Ghannem, A., Boussaidi, G. E., and Kessentini, M. (2013). Model refactoring using interactive genetic algorithm. In *SSBSE 2013 Proceedings of the 5th International Symposium on Search Based Software Engineering - Volume 8084*, pages 96–110.

Ghannem, A., Boussaidi, G. E., and Kessentini, M. (2014). Model refactoring using examples: a search-based approach. *Journal of Software: Evolution and Process*, 26(7):692–713.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

Gniesser, P. (2012). *Refactoring support for ATL-based model transformations*. na.

Gogolla, M. and Vallecillo, A. (2011). *Tract*able model transformation testing. In *Modelling Foundations and Applications - 7th European Conference, ECMFA 2011, Birmingham, UK, June 6 - 9, 2011 Proceedings*, pages 221–235.

Gogolla, M., Vallecillo, A., Burgueño, L., and Hilken, F. (2015). Employing classifying terms for testing model transformations. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, pages 312–321.

Gomez, J. J. C., Baudry, B., and Sahraoui, H. (2012). Searching the boundaries of a modeling space to test metamodels. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 131–140.

González, C. A. and Cabot, J. (2012). Atltest: a white-box test generation approach for atl transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 449–464. Springer.

González, C. A. and Cabot, J. (2012). Atltest: A white-box test generation approach for ATL transformations. In *Proceedings of the 15th International Conference Model Driven Engineering Languages and Systems (MODELS)*, pages 449–464.

Goodenough, J. B. and Gerhart, S. L. (1975). Toward a theory of test data selection. *IEEE Transactions on software Engineering*, (2):156–173.

Greenyer, J. and Kindler, E. (2010). Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. *Software & Systems Modeling*, 9(1):21.

Guerra, E. (2012). Specification-driven test generation for model transformations. In *Theory and Practice of Model Transformations - 5th International Conference, ICMT 2012, Prague, Czech Republic, May 28-29, 2012. Proceedings*, pages 40–55.

Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., and Schwinger, W. (2013). Automated verification of model transformations based on visual contracts. *Automated Software Engineering*, 20(1):5–46.

Gupta, R., Harrold, M. J., and Soffa, M. L. (1992). An approach to regression testing using slicing. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 299–308. IEEE.

Gyapay, S., Ákos Schmidt, and Varró, D. (2004). Joint optimization and reachability analysis in graph transformation systems with time. *Electronic Notes in Theoretical Computer Science*, 109:137–147.

Hadka, D. (2012). Moea framework: a free and open source java framework for multiobjective optimization. http://www.moeaframework.org.

Hainaut, J.-L., Cleve, A., Henrard, J., and Hick, J.-M. (2008). Migration of legacy information systems. In *Software Evolution*, pages 105–138. Springer.

Harman, M. (2007). The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357. IEEE Computer Society.

Harman, M., Hierons, R. M., and Proctor, M. (2002). A new representation and crossover operator for search-based optimization of software modularization. In *GECCO*, volume 2, pages 1351–1358.

Harman, M., McMinn, P., De Souza, J. T., and Yoo, S. (2012). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification*, pages 1–59. Springer.

Harman, M. and Tratt, L. (2007). Pareto optimal search based refactoring at the design level. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1106–1113. ACM.

Harrold, M. J. and Souffa, M. (1988). An incremental approach to unit testing during maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 362–367. IEEE.

Hartmann, J. and Robson, D. (1989). Revalidation during the software maintenance phase. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 70–80. IEEE.

Hartmann, J. and Robson, D. J. (1990). Retest-development of a selective revalidation prototype environment for use in software maintenance. In *Twenty-Third Annual Hawaii International Conference on System Sciences*, pages 92–101. IEEE.

Hemmati, H., Briand, L., Arcuri, A., and Ali, S. (2010). An enhanced test case selection approach for model-based testing: an industrial case study. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 267–276. ACM.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70.

Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 471–480. IEEE.

INRIA (2005). Atl transformation example: Bibtexml to docbook. https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf , Last accessed on 15-July-2014.

ISO (2011). Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. Technical report, International Organization for Standardization.

Jensen, A. C. and Cheng, B. H. (2010). On the use of genetic programming for automated refactoring and the introduction of design patterns. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1341–1348.

Jézéquel, J.-M., Barais, O., and Fleurey, F. (2009). Model driven language engineering with kermeta. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, pages 201–221. Springer.

Jilani, A. A., Iqbal, M. Z., and Khan, M. U. (2014). A search based test data generation approach for model transformations. In *International Conference on Theory and Practice of Model Transformations*, pages 17–24.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1–2):31 – 39.

Jouault, F. and Kurtev, I. (2005). Transforming models with atl. In *International Conference on Model Driven Engineering Languages and Systems*, pages 128–138. Springer.

Kalyanmoy, D. et al. (2001). *Multi objective optimization using evolutionary algorithms*. John Wiley and Sons.

Kapová, L., Goldschmidt, T., Becker, S., and Henss, J. (2010). Evaluating maintainability with code metrics for model-to-model transformations. In *QoSA'10 Proceedings of the 6th international conference on Quality of Software Architectures: research into Practice - Reality and Gaps*, pages 151–166.

Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., and Wimmer, M. (2012). Model Transformation By-Example: A Survey of the First Wave. In *Conceptual Modelling and Its Theoretical Foundations*, volume 7260, pages 197–215.

Kazmi, R., Jawawi, D. N., Mohamad, R., and Ghani, I. (2017). Effective regression test case selection: A systematic literature review. *ACM Computing Surveys (CSUR)*, 50(2):29.

Kessentini, M., Bouchoucha, A., Sahraoui, H. A., and Boukadoum, M. (2010). Example-based sequence diagrams to colored petri nets transformation using heuristic search. In *ECMFA'10 Proceedings of the 6th European conference on Modelling Foundations and Applications*, pages 156–172.

Kessentini, M., Langer, P., and Wimmer, M. (2013). Searching models, modeling search: On the synergies of sbse and mde. In *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 51–54. IEEE.

Kessentini, M., Sahraoui, H., Boukadoum, M., and Omar, O. B. (2012). Search-based model transformation by example. *Software and Systems Modeling*, 11(2):209–226.

Kessentini, M., Sahraoui, H., Boukadoum, M., and Wimmer, M. (2011). Search-based design defects detection by example. In *International Conference on Fundamental Approaches to Software Engineering*, pages 401–415. Springer.

Kessentini, M., Sahraoui, H. A., and Boukadoum, M. (2008). Model transformation as an optimization problem. In *MoDELS '08 Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, pages 159–173.

Kessentini, M., Sahraoui, H. A., and Boukadoum, M. (2011). Example-based model-transformation testing. *automated software engineering*, 18(2):199–224.

Kessentini, M., Werda, W., Langer, P., and Wimmer, M. (2013). Search-based model merging. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1453–1460.

Kessentini, M., Wimmer, M., Sahraoui, H., and Boukadoum, M. (2010). Generating transformation rules from examples for behavioral models. In *Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications*, BM-FA '10, pages 2:1–2:7, New York, NY, USA. ACM.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Klar, F., Königs, A., and Schürr, A. (2007). Model transformation in the large. In *Proc. of ESEC-FSE*, pages 285–294.

Kolovos, D. S., Paige, R. F., and Polack, F. (2008). The Epsilon Transformation Language. In *Proc. of ICMT*, volume 5063, pages 46–60.

Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.

Kumar, M., Sharma, A., and Kumar, R. (2012). Multi faceted measurement framework for test case classification and fitness evaluation using fuzzy logic based approach. *Chiang Mai Journal of Science*, 39(3).

Kurtev, I., van den Berg, K., and Jouault, F. (2007). Rule-based modularization in model transformation languages illustrated with ATL. *Sci. Comput. Program.*, 68(3):138–154.

Kusel, A., Schoenboeck, J., Wimmer, M., Retschitzegger, W., Schwinger, W., and Kappel, G. (2013). Reality check for model transformation reuse: The atl transformation zoo case study. In *Amt@ models*, pages 1–11.

Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., and Schwinger, W. (2015). Reuse in model-to-model transformation languages: are we there yet? *Software & Systems Modeling*, 14(2):537–572.

Lamari, M. (2007). Towards an automated test generation for the verification of model transformations. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 998–1005. ACM.

Lano, K. C. and Alfraihi, H. A. A. (2018). Technical debt in model transformation specifications. *Lecture Notes in Computer Science*.

Laski, J. and Szermer, W. (1992). Identification of program modifications and its applications in software maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 282–290. IEEE.

Laumanns, M., Thiele, L., Deb, K., and Zitzler, E. (2002). Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282.

Lee, J. A. and He, X. (1990). A methodology for test selection. *Journal of Systems and Software*, 13(3):177–185.

Leung, H. K. and White, L. (1989). Insights into regression testing (software testing). In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 60–69. IEEE.

Leung, H. K. and White, L. (1990). A study of integration testing and software regression at the integration level. In *Proceedings. Conference on Software Maintenance 1990*, pages 290–301. IEEE.

Lin, Y., Zhang, J., and Gray, J. (2005). A testing framework for model transformations. In *Model-driven software development*, pages 219–236. Springer.

Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G., Syriani, E., and Wimmer, M. (2014). Model Transformation Intents and Their Properties. *SoSyM*, pages 1–35.

Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G. M., Syriani, E., and Wimmer, M. (2016). Model transformation intents and their properties. *Software & systems modeling*, 15(3):647–684.

Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y., and Gansner, E. R. (1998). Using automatic clustering to produce high-level system organizations of source code. In *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No. 98TB100242)*, pages 45–52. IEEE.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

Mansoor, U., Kessentini, M., Langer, P., Wimmer, M., Bechikh, S., and Deb, K. (2015). Momm: Multi-objective model merging. *Journal of Systems and Software*, 103:423–439.

Mansoor, U., Kessentini, M., Wimmer, M., and Deb, K. (2017). Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. *Software Quality Journal*, 25(2):473–501.

Maqbool, O. and Babri, H. (2007). Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780.

Masoud, H. and Jalili, S. (2014). A clustering-based model for class responsibility assignment problem in object-oriented analysis. *Journal of Systems and Software*, 93:110–131.

McQuillan, J. A. and Power, J. F. (2009). White-box coverage criteria for model transformations. In *Proceedings of the 1st International Workshop on Model Transformation with ATL*, pages 63–77.

Mens, T. and Van Gorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Milanović, M. (2007). *Modeling rules on the Semantic Web*. PhD thesis, Master thesis, University of Belgrade.

Milanović, M., Gašević, D., Giurca, A., Wagner, G., and Devedžić, V. (2007). Towards sharing rules between owl/swrl and uml/ocl. *Electronic Communications of the EASST*, 5.

Mirarab, S., Akhlaghi, S., and Tahvildari, L. (2012). Size-constrained regression test case selection using multicriteria optimization. *IEEE Transactions on Software Engineering*, 38(4):936–956.

Misbhauddin, M. and Alshayeb, M. (2015). Uml model refactoring: a systematic literature review. *Empirical Software Engineering*, 20(1):206–251.

Mitchell, B. S. and Mancoridis, S. (2006). On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208.

Mitchell, B. S. and Mancoridis, S. (2008). On the evaluation of the bunch search-based software modularization algorithm. *Soft Computing*, 12(1):77–93.

Mkaouer, M. W., Kessentini, M., Bechikh, S., Cinnéide, M. Ó., and Deb, K. (2016). On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. *Empirical Software Engineering*, 21(6):2503–2545.

Mkaouer, M. W., Kessentini, M., Bechikh, S., and Tauritz, D. R. (2013). Preference-based multi-objective software modelling. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, pages 61–66.

Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., and Ouni, A. (2015). Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):17.

Moghadam, I. H. (2011). Multi-level automated refactoring using design exploration. In *SSBSE'11 Proceedings of the Third international conference on Search based software engineering*, pages 70–75.

Moghadam, I. H. and Cinnéide, M. (2012). Automated refactoring using design differencing. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 43–52.

Mohagheghi, P. and Dehlen, V. (2008). Developing a quality framework for model-driven engineering. In *Models in Software Engineering*, pages 275–286.

Mohagheghi, P. and Dehlen, V. (2008). Where is the proof?-a review of experiences from applying mde in industry. In *Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications*, pages 432–443. Springer.

Mohamed, M., Romdhani, M., and Ghedira, K. (2009). Classification of model refactoring approaches. *Journal of Object Technology*, 8(6):121–126.

Mooij, A. J., Eggen, G., Hooman, J., and van Wezep, H. (2015). Cost-effective industrial software rejuvenation using domain-specific models. In *International Conference on Theory and Practice of Model Transformations*, pages 66–81. Springer.

Mottu, J.-M., Baudry, B., and Le Traon, Y. (2006). Mutation analysis testing for model transformations. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 376–390. Springer.

Nalchigar, S., Salay, R., and Chechik, M. (2013). Towards a catalog of non-functional requirements for model transformations. In *Proceedings of the Second Workshop on the Analysis of Model Transformations (AMT 2013)*.

Oakes, B. J., Troya, J., Lúcio, L., and Wimmer, M. (2018). Full contract verification for atl using symbolic execution. *Software & Systems Modeling*, 17(3):815–849.

OMG (2005). *MOF QVT Final Adopted Specification*. Object Management Group.

Ouni, A., Gaikovina Kula, R., Kessentini, M., and Inoue, K. (2015). Web service antipatterns detection using genetic programming. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1351–1358. ACM.

Ouni, A., Kessentini, M., Inoue, K., and Cinnéide, M. O. (2017). Search-based web service antipatterns detection. *IEEE Transactions on Services Computing*, 10(4):603–617.

Panichella, A., Oliveto, R., Di Penta, M., and De Lucia, A. (2015). Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering*, 41(4):358–383.

Porres, I. (2003). Model refactorings as rule-based update transformations. In *International Conference on the Unified Modeling Language*, pages 159–174. Springer.

Praditwong, K., Harman, M., and Yao, X. (2011). Software Module Clustering as a Multi-Objective Search Problem. *IEEE Trans. Software Eng.*, 37(2):264–282.

Project, E. M. (2015). Atl transformations zoo.

Rachmawati, L. and Srinivasan, D. (2009). Multiobjective evolutionary algorithm with controllable focus on the knees of the pareto front. *IEEE Transactions on Evolutionary Computation*, 13(4):810–824.

Rahimi, S. K. and Lano, K. (2011). Integrating goal-oriented measurement for evaluation of model transformation. In *2011 CSI International Symposium on Computer Science and Software Engineering (CSSE)*, pages 129–134.

Reimann, J., Wilke, C., Demuth, B., Muck, M., and Aßmann, U. (2012). Tool supported ocl refactoring catalogue. In *Proceedings of the 12th Workshop on OCL and Textual Modelling*, pages 7–12. ACM.

Reus, T., Geers, H., and Van Deursen, A. (2006). Harvesting software systems for mda-based reengineering. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 213–225. Springer.

Rivera, J. E., Duran, F., and Vallecillo, A. (2009). A graphical approach for modeling time-dependent behavior of dsls. In *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*, pages 51–55. IEEE.

Rose, L. M. and Poulding, S. M. (2013). Efficient probabilistic testing of model transformations using search. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, pages 16–21.

Rosero, R. H., Gómez, O. S., and Rodríguez, G. (2016). 15 years of software regression testing techniques-a survey. *International Journal of Software Engineering and Knowledge Engineering*, 26(05):675–689.

Rothermel, G. and Harrold, M. J. (1993). A safe, efficient algorithm for regression test selection. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 358–367. IEEE.

Rothermel, G. and Harrold, M. J. (1994). Selecting tests and identifying test coverage requirements for modified software. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 169–184.

Rothermel, G. and Harrold, M. J. (1997). Experience with regression test selection. *Empirical Software Engineering*, 2(2):178–188.

Saada, H., Huchard, M., Nebut, C., and Sahraoui, H. A. (2013). Recovering model transformation traces using multi-objective optimization. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 688–693.

Saeki, M. and Kaiya, H. (2007). Measuring model transformation in model driven development. In *CAiSE Forum*.

Sahin, D., Kessentini, M., Wimmer, M., and Deb, K. (2015). Model transformation testing: a bi-level search-based software engineering approach. *Journal of Software: Evolution and Process*, 27(11):821–837.

Schmidt, D. C. (2006). Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25.

Seawright, J. and Gerring, J. (2008). Case selection techniques in case study research: A menu of qualitative and quantitative options. *Political Research Quarterly*, 61(2):294–308.

Selim, G. M., Wang, S., Cordy, J. R., and Dingel, J. (2012). Model transformations for migrating legacy models: an industrial case study. In *European Conference on Modelling Foundations and Applications*, pages 90–101. Springer.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5):42–45.

Seng, O., Bauer, M., Biehl, M., and Pache, G. (2005). Search-based improvement of subsystem decompositions. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1045–1051. ACM.

Sharma, C., Sabharwal, S., and Sibal, R. (2014). Applying genetic algorithm for prioritization of test case scenarios derived from uml diagrams. *arXiv preprint arXiv:1410.4838*.

Shelburg, J., Kessentini, M., and Tauritz, D. R. (2013). Regression testing for model transformations: A multi-objective approach. *symposium on search based software engineering*, pages 209–223.

Shtern, M. and Tzerpos, V. (2009). Methods for selecting and improving software clustering algorithms. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 248–252. IEEE.

Straeten, R. V. D., Mens, T., and Baelen, S. V. (2009). Challenges in model-driven software engineering. *model driven engineering languages and systems*, 5421:35–47.

Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., and Plöger, J. (2016). Rulemerger: automatic construction of variability-based model transformation rules. In *International Conference on Fundamental Approaches to Software Engineering*, pages 122–140. Springer.

Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., and Plöger, J. (2018). Variability-based model transformation: formal foundation and application. *Formal Aspects of Computing*, 30(1):133–162.

Syriani, E. and Gray, J. (2012). Challenges for addressing quality factors in model transformation. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 929–937.

Taentzer, G. (2003). Agg: A graph transformation environment for modeling and validation of software. In *International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 446–453. Springer.

Taentzer, G., Arendt, T., Ermel, C., and Heckel, R. (2012). Towards refactoring of rule-based, in-place model transformation systems. In *Proceedings of the First Workshop on the Analysis of Model Transformations*, pages 41–46. ACM.

Taha, A.-B., Thebaut, S. M., and Liu, S.-S. (1989). An approach to software fault localization and revalidation based on incremental data flow analysis. In *Proceedings of the Thirteenth Annual International Computer Software & Applications Conference*, pages 527–534. IEEE.

Thies, A. and Roth, C. (2010). Recommending rename refactorings. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 1–5. ACM.

Tichy, M., Krause, C., and Liebel, G. (2013). Detecting performance bad smells for henshin model transformations. *Proc. of the 2nd Workshop on the Analysis of Model Transformations (AMT), September 29, Miami, USA, 2013*.

Tisi, M., Jouault, F., Fraternali, P., Ceri, S., and Bézivin, J. (2009). On the use of higher-order model transformations. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 18–33. Springer.

Tolosa, J. B., Sanjuán-Martínez, O., García-Díaz, V., G-Bustelo, B. C. P., and Lovelle, J. M. C. (2011). Towards the systematic measurement of atl transformation models. *Software - Practice and Experience*, 41(7):789–815.

Troya, J., Bergmayr, A., Burgueño, L., and Wimmer, M. (2015). Towards systematic mutations for and with atl model transformations. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pages 1–10. IEEE.

Troya, J., Fleck, M., Kessentini, M., Wimmer, M., and Alkhaze, B. (2016). Rules and helpers dependencies in atl–technical report. *Universidad de Sevilla*.

Troya, J., Segura, S., Parejo, J. A., and Cortés, A. R. (2018a). Spectrum-based fault localization in model transformations. *ACM Trans. Softw. Eng. Methodol.*, 27(3):13:1–13:50.

Troya, J., Segura, S., Parejo, J. A., and Ruiz-Cortés, A. (2018b). Spectrum-based fault localization in model transformations. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(3):13.

Troya Castilla, J. and Vallecillo Moreno, A. (2011). A rewriting logic semantics for atl. *Journal of Object Technology*, 10:5–1.

Vallecillo, A., Gogolla, M., Burgueno, L., Wimmer, M., and Hamann, L. (2012). Formal specification and testing of model transformations. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 399–437. Springer.

van Amstel, M., Bosems, S., Kurtev, I., and Pires, L. F. (2011). Performance in model transformations: experiments with atl and qvt. In *ICMT'11 Proceedings of the 4th international conference on Theory and practice of model transformations*, pages 198–212.

van Amstel, M. F. and van den Brand, M. (2010). Quality assessment of atl model transformations using metrics. In *Proceedings of the 2nd International Workshop on Model Transformation with ATL (MtATL 2010), Malaga, Spain (June 2010)*, page 115.

van Amstel, M. F. and van den Brand, M. (2011). Using metrics for assessing the quality of atl model transformations. In *Proceedings of the Third International Workshop on Model Transformation with ATL (MtATL 2011)*, volume 742, pages 20–34.

van Amstel, M. M., Lange, C. C., and van den Brand, M. M. (2009). Using metrics for assessing the quality of asf+sdf model transformations. In *ICMT '09 Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, pages 239–248.

Van Der Straeten, R., Mens, T., and Van Baelen, S. (2008). Challenges in model-driven software engineering. In *International Conference on Model Driven Engineering Languages and Systems*, pages 35–47. Springer.

van Mf Marcel Amstel (2012). Assessing and improving the quality of model transformations.

Varró, D. (2006). Model Transformation by Example. In *Proc. of MoDELS*, pages 410–424.

Vieira, A. and Ramalho, F. (2014). Metrics to measure the change impact in atl model transformations. In *International Conference on Product-Focused Software Process Improvement*, pages 254–268.

Vignaga, A. (2009). Metrics for measuring atl model transformations. Technical report.

Völter, M., Stahl, T., Bettin, J., Haase, A., and Helsen, S. (2013). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

Wang, W., Kessentini, M., and Jiang, W. (2013). Test cases generation for model transformations from structural information. *MDEBE@MoDELS*, pages 42–51.

Weiderman, N., Northrop, L., Smith, D., Tilley, S., and Wallnau, K. (1997). Implications of distributed object technology for reengineering. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Wiggerts, T. A. (1997). Using clustering algorithms in legacy systems remodularization. In *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 33–43. IEEE.

Wimmer, M. and Burgueño, L. (2013). Testing M2T/T2M transformations. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 203–219. Springer.

Wimmer, M., Perez, S. M., Jouault, F., and Cabot, J. (2012). A catalogue of refactorings for model-to-model transformations. *Journal of Object Technology*, 11(2):2–1.

Wimmer, M., Strommer, M., Kargl, H., and Kramler, G. (2007). Towards Model Transformation Generation By-Example. In *Proc. of HICSS*.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Wu, H., Monahan, R., and Power, J. F. (2012). Metamodel instance generation: A systematic literature review. *arXiv preprint arXiv:1211.6322*.

Yau, S. S. and Kishimoto, Z. (1987). Method for revalidating modified programs in the maintenance phase. In *Proceedings of the IEEE Computer Society's International Computer Software & Applications Conference*. IEEE.

Yeo, K. T. (2002). Critical failure factors in information system projects. *International journal of project management*, 20(3):241–246.

Yoo, S. and Harman, M. (2007). Pareto efficient multi-objective test case selection. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 140–150. ACM.

Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120.

Zhang, J., Lin, Y., and Gray, J. (2005). Generic and domain-specific model refactoring using a model transformation engine. pages 199–217.