Model Transformation Modularization as a Many-Objective Optimization Problem

Article *in* IEEE Transactions on Software Engineering · January 2017
DOI: 10.1109/TSE.2017.2654255

5 authors, including:

Javier Troya
Universidad de Sevilla
52 PUBLICATIONS **318** CITATIONS

SEE PROFILE

Model Transformation Modularization as a Many-Objective Optimization Problem

Martin Fleck, Javier Troya, Marouane Kessentini, Manuel Wimmer and Bader Alkhazi

Abstract—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, modular design is a desirable property for model transformations as it can significantly improve their evolution, comprehensibility, maintainability, reusability, and thus, their overall quality. Although language support for modularization of model transformations is emerging, model transformations are created as monolithic artifacts containing a huge number of rules. To the best of our knowledge, the problem of automatically modularizing model transformation programs was not addressed before in the current literature. These programs written in transformation languages, such as ATL, are implemented as one main module including a huge number of rules. To tackle this problem and improve the quality and maintainability of model transformation programs, we propose an automated search-based approach to modularize model transformations based on higher-order transformations. Their application and execution is guided by our search framework which combines an in-place transformation engine and a search-based algorithm framework. We demonstrate the feasibility of our approach by using ATL as concrete transformation language and NSGA-III as search algorithm to find a trade-off between different well-known conflicting design metrics for the fitness functions to evaluate the generated modularized solutions. To validate our approach, we apply it to a comprehensive dataset of model transformations. As the study shows, ATL transformations can be modularized automatically, efficiently, and effectively by our approach. We found that, on average, the majority of recommended modules, for all the ATL programs, by NSGA-III 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 model transformations 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.

Index Terms—model transformation, modularization, ATL, NSGA-III, MDE, SBSE

1 Introduction

Model-Driven Engineering (MDE) is a methodology that advocates the use of models throughout the software engineering life cycle to simplify the design process and increase productivity. Model transformations are the cornerstone of MDE [1], [2] 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 [3], 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 [4], which does not contain any modularized transformation [5]. Thus, most of the existing ATL transformations are difficult to evolve, test and maintain.

In this paper, we therefore propose, for the first time in the MDE literature, 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. We see this need for an automatic approach as there are several model transformation languages now offering modularization concepts [3]. 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 transformations [6]. 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 have 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 cluster the rules that

- *M. Fleck and M. Wimmer are with the TU Wien. E-mail: {lastname}@big.tuwien.ac.at*
- *J. Troya is with the Universidad de Sevilla. E-mail: jtroya@us.es*
- *M. Kessentini and B. Alkhazi are with the University of Michigan. E-mail: {firstname}@umich.edu*

are semantically close by reducing the number of inter-calls between rules in different modules (coupling) and increasing the number of intra-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 [7] 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 [8], 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 an 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. In this paper, we consider seven different-sized transformations, of which six 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 article 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 differently-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.

The remainder of this paper is structured as follows. In Section 2 we introduce the concepts of model-driven engineering, search-based software engineering and the modularization capabilities of model transformations. Section 3 introduces our approach and describes how we define the modularization problem in our framework. Section 4 describes the evaluation of our solutions retrieved for the modularization problem, before we give an overview on related work in Section 5. Finally, Section 6 concludes the paper with an outlook on future work.

2 Background

In this section we present the main pillars on which our approach is built, namely MDE and SBSE.

2.1 Model-Driven Engineering

Model-Driven Engineering (MDE) [9], [10] is a methodology that advocates the use of models as first-class entities throughout the software engineering life cycle. It is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design and promoting communication between individuals and teams working on the system. Next we describe the key concepts of MDE.

2.1.1 Metamodels

A metamodel is a model that specifies the concepts of a language, their relationships, and the structural rules to build valid models. As an example, the metamodel 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 metamodel, so there has to hold a conformance relation between a model and its metamodel. A metamodel is itself a model, and consequently, it is written in the language defined by its meta-metamodel. Metamodels allow to specify general-purpose languages as well as domain-specific languages (DSLs). For realizing model transformations, there exist dedicated DSLs which are explained next.

2.1.2 Model Transformations

They are a key technique to automate software engineering tasks in MDE [10], [11], by providing the essential mechanisms for manipulating models. In fact, they allow to transform models into other models or into code, and are essential for synthesizing systems in MDE. In [1], [12], an overview of transformation language concepts as well as a classification of different transformation types are presented. In this paper, 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 [1] in Fig. 1. One important aspect is that model transformations are developed on the metamodel level and are thus reusable for all valid models.

Many different model transformation languages emerged in the last decade such as Henshin [13], AGG [14], AToM^3 [15], e-Motions [16], VIATRA [17], QVT [18], Kermeta [19], JTL [20], and ATL [21]. In this paper, we focus on ATL since it has come to prominence in the MDE community. This success is due to ATL's flexibility,

support of the main metamodeling standards, usability that relies on tool integration with Eclipse, and a supportive development community [22].

2.1.3 ATLAS Transformation Language (ATL)

ATL is a hybrid model transformation language containing a mixture of declarative and imperative constructs. Listing 1 shows an excerpt of an ATL transformation taken from the ATL Zoo [4] that generates a relational schema from a class diagram. The input and output metamodels of this transformation are depicted in Fig. 2. 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.

Listing 1. Excerpt of the *Class2Relational* Transformation.

```
1  module Class2Relation;
2  create OUT : Relational from IN : Class;
3
4  helper def : objectIdType : Relational!Type =
5     Class!DataType.allInstances() -> select(e | e.name =
          'Integer') -> first();
6
7  rule ClassAttribute2Column {
8    from
9      a : Class!Attribute (a.type.oclIsKindOf(Class!Class)
            and not a.multiValued)
10   to
11     foreignKey : Relational!Column (
12       name <- a.name + 'Id',
13       type <- thisModule.objectIdType) }
14
15 rule Class2Table {
16   from
17     c : Class!Class
18   to
19     out : Relational!Table (
20       name <- c.name,
21       col  <- Sequence {key}->union(c.attr->select(e |
              not e.multiValued)),
22       key  <- Set {key}),
23     key : Relational!Column (
24       name <- 'objectId',
25       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.

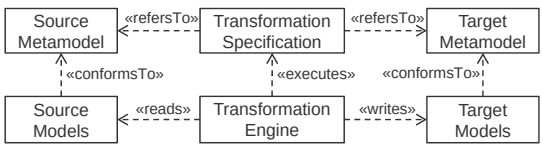
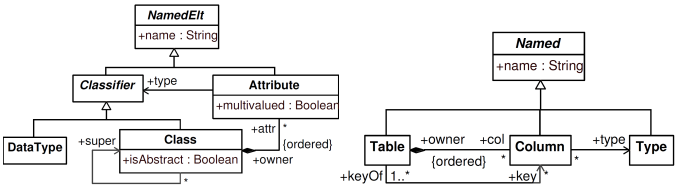


Fig. 1. Model transformation pattern (from [1]).



(a) Source metamodel. (b) Target metamodel.

Fig. 2. Metamodels of the Class2Relational transformation.

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 Fig. 3. 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 metamodel shown in Fig. 2a. The right-hand side shows the model produced by the *Class2Relation* transformation and that conforms to the metamodel in Fig. 2b. 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*.

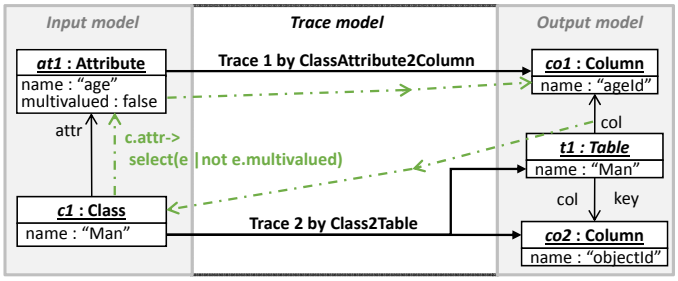


Fig. 3. Representation of a sample transformation execution.

Thus, 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 Section 3. 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 Fig. 4.

2.1.4 Modularization in ATL

ATL supports three kinds of units: modules, queries, and libraries. An ATL module corresponds to a M2M transformation. This kind of ATL unit enables ATL developers to specify the way to produce a set of target models from a set of source models. Modules are composed of a set of

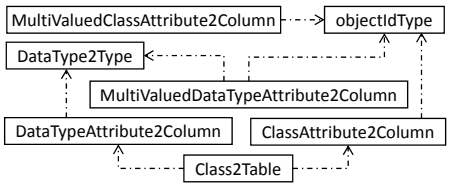


Fig. 4. *Class2Relational* transformation elements dependencies.

transformation rules, and may also contain helper functions (*helpers*). ATL modules are the only kind of unit that can return output models. ATL queries are operations that compute a primitive value from a set of source models. ATL queries are not considered in this paper due to their rare use [5]. Finally, ATL libraries enable to define a set of ATL helpers that can be called from different ATL units. Compared to both modules and queries, an ATL library cannot be executed independently.

All ATL units support composition. In particular, ATL libraries can import other libraries and ATL modules can import libraries and other modules. Wagelaar et al. [23] distinguish between external composition, where the output of one transformation serves as input for the next one, and internal composition, where one transformation is split into a number of transformation definitions which are combined when the transformation needs to be executed. The *superimposition* feature of ATL is the internal composition method.

Module superimposition [23] is used to split transformation modules into modules of manageable size and scope, which are then superimposed on top of each other. This results in (the equivalent of) a transformation module that contains the union of all transformation rules. In addition it is also possible to override rules with superimposition. However, in this paper, we are not interested in this aspect, since our purpose is to split a transformation composed of a large module into modules of smaller size, but we do not modify existing rules.

One of the reasons that motivate our work is that the ATL Zoo [4] does not provide a single modularized transformation currently. An explanation for this may be that the modularization mechanism was introduced retrospectively for ATL, while ATL has been used for some time in practice. Furthermore, we believe there is a huge potential for an automatic modularization approach as there are many legacy transformations available that have an extensive size. The introduction of module concepts in other transformation languages [3], such as QVT-O, TGGs, QVT-R, ETL and RubyTL, reinforces the importance of modules for transformations reaching a critical size, where the rule concept is not sufficient anymore. However, we could not find any populated transformation repository for these languages, as it is provided for ATL.

2.2 Search-based Software Engineering

Search-based software engineering [7] 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 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 [24] or Simulated Annealing [25]. On the other hand, evolutionary algorithms [26] manage a set of solutions, referred to as population, at once. Some widely used evolutionary algorithms include NSGA-II [27] and NSGA-III [8].

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 Multiobjective Optimization (EMO) is considered one of the most active research areas in evolutionary computation [28]. 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 [29], [30], [31], the optimization of regression tests for model transformations [32], the detection of high-level model changes [33] or the enhancement of the readability of source code for given metrics [34], [35].

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 1). The nth 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.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \qquad B_0 = 1 \tag{1}$$

Considering the first Bell numbers (cf. sequence A000110[1] 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.

In the following subsections, we introduce some details about the used algorithm to handle the different conflicting objectives considered in our modularization formulation.

2.2.1 Multi/Many-objective Problem

A multi-objective problem can be stated as follows [36]:

$$\begin{cases} Min\, f(x) = [f_1(x), f_2(x), \ldots, f_M(x)]^T & \\ g_j(x) \geq 0 & j = 1, \ldots, P; \\ h_k(x) = 0 & k = 1, \ldots, Q; \\ x_i^L \leq x_i \leq x_i^U & i = 1, \ldots, n; \end{cases}$$

1. http://oeis.org/A000110

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

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 and many-objective problems. A many-objective problem, as opposed to a multi-objective one, is a problem with at least four objectives, i.e., $M > 3$.

2.2.2 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) and in (3) for strict inequality [7], 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.

$$F(\overline{x_1}) \geq F(\overline{x_2}) \Leftrightarrow \forall i \; f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \tag{2}$$

$$F(\overline{x_1}) > F(\overline{x_2}) \Leftrightarrow \forall i \; f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \wedge \exists i \; f_i(\overline{x_1}) > f_i(\overline{x_2}) \tag{3}$$

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 [37]. 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 article, we always refer to the Pareto front approximation. The following sub-section gives an overview about the considered many-objective algorithm to address our modularization problem.

2.2.3 NSGA-III

NSGA-III is a very recent many-objective algorithm proposed by Deb et al. [8]. The basic framework remains similar to the original NSGA-II algorithm with significant changes in its selection mechanism. The algorithm displayed in Fig. 6 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 Fig. 5. 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

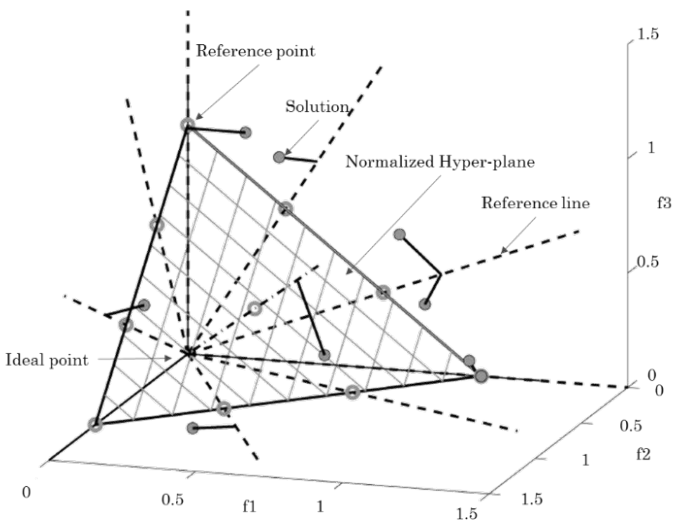


Fig. 5. Normalized reference plane for a three-objective case.

Input: H structured reference points Z_s, population P_t
Output: population P_{t+1}
1: $S_t \leftarrow \emptyset, i \leftarrow 1$
2: $Q_t \leftarrow$ VARIATION(P_t)
3: $R_t \leftarrow P_t \cup Q_t$
4: $(F_1, F_2, \ldots) \leftarrow$ NONDOMINATED_SORT(R_t)
5: **repeat**
6: $S_t \leftarrow S_t \cup F_i$
7: $i \leftarrow i + 1$
8: **until** $|S_t| \geq N$
9: $F_l \leftarrow F_i$ *// last front to be included*
10: **if** $|S_t| = N$ **then**
11: $P_{t+1} \leftarrow S_t$
12: **else**
13: $P_{t+1} \leftarrow \bigcup_{j=1}^{l-1} F_j$
14: $K \leftarrow N - |P_{t+1}|$ *// number of points chosen from F_l*
// normalize objectives and create reference set Z^r
15: 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 $\pi(s)$
16: $[\pi(s), d(s)] \leftarrow$ ASSOCIATE(S_t, Z^r)
// Compute niche count of a reference point $j \in Z^r$
17: $p_j \leftarrow \sum_{s \in S_t / F_l} ((\pi(s) = j)\ ?\ 1 : 0)$
// Choose K members one by one from F_l to construct P_{t+1}
18: NICHING$(K, p_j, \pi(s), d(s), Z^r, F_l, P_{t+1})$
19: **end if**

Fig. 6. NSGA-III procedure at generation t.

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 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. We describe in the following section our adaptation of NSGA-III for our modularization problem.

3 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.

3.1 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. And 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 Fig. 7.

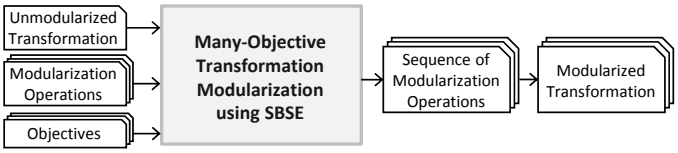


Fig. 7. Overview of our modularization approach.

3.1.1 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 Fig. 8.

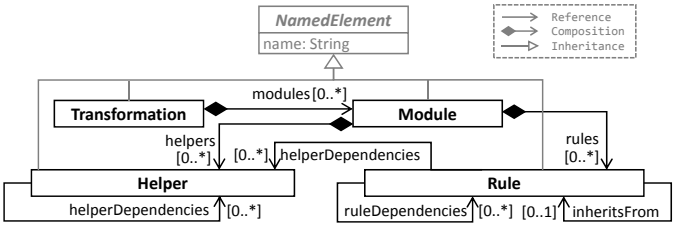


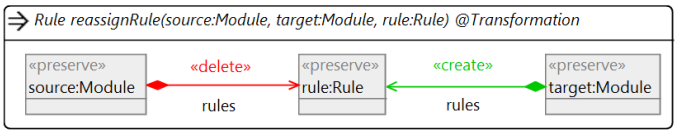
Fig. 8. Modularization Metamodel.

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*).

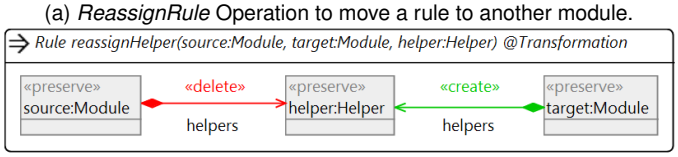
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

3.1.2 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 Fig. 9 are sufficient.



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



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

Fig. 9. 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.

3.1.3 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 1. Specifically, we aim to minimize the number of modules (NMT), minimize the difference between the lowest and the highest number of transformation elements, 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.

The formulas for each objective function are given in Equations (4)-(7) (adapted from [38]). 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 (10), $D_{RH}(m_i, m_j)$ in (11), and $D_{HH}(m_i, m_j)$ in (12) specify the

TABLE 1
Objective functions for a modularization solution.

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

total number of rule-to-rule, rule-to-helper and helper-to-helper dependencies between the given modules m_i and m_j, respectively. These numbers are calculated to compute $R_R(m_i, m_j)$ (10), $R_H(m_i, m_j)$ (11), and $H_H(m_i, m_j)$ (12), which represent the ratio of rule-to-rule, rule-to-helper and helper-to-helper dependencies between the given modules m_i and m_j, 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 (8) represents the total ratio of dependencies between the given modules m_i and m_j.

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 10 and 12 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 10 and 12 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 11 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 [38], [39], [40] 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 rules and helpers equally among the modules mitigates against small isolated clusters and tends to avoid larger modules, as also discussed by [41]. Furthermore, we optimize the coupling and cohesion ratio 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 depen-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

dencies, 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 fulfil its functionality.

$$\text{NMT} = |M| \tag{4}$$

$$\text{DIF} = max(|U(m)|) - min(|U(m)|), m \in M \tag{5}$$

$$\text{COH} = \sum_{m_i \in M} D(m_i, m_i) \tag{6}$$

$$\text{COP} = \sum_{\substack{m_i, m_j \in M \\ m_i \neq m_j}} D(m_i, m_j) \tag{7}$$

$$D(m_i, m_j) = R_R(m_i, m_j) + R_H(m_i, m_j) \tag{8}$$

$$+H_H(m_i, m_j) \tag{9}$$

$$R_R(m_i, m_j) = \frac{D_{RR}(m_i, m_j)}{|R(m_i)| \times |R(m_j) - 1|} \tag{10}$$

$$R_H(m_i, m_j) = \frac{D_{RH}(m_i, m_j)}{|R(m_i)| \times |H(m_j)|} \tag{11}$$

$$H_H(m_i, m_j) = \frac{D_{HH}(m_i, m_j)}{|H(m_i)| \times |H(m_j) - 1|} \tag{12}$$

Finally, to define the validity of our solutions, we enforce through constraints that all transformation elements are assigned to a module and that each module must contain at least one element. Solutions which do not fulfil these constraints are not part of the feasible search space, as defined in Section 2.2.

3.1.4 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 $P2'$. Then, the remaining elements of $P1$ become the remaining elements of $P2'$ and the remaining elements of $P2$ become the remaining elements of $P1'$.

Mutation. The mutation operator consists of randomly changing one or more dimensions in the solution. 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 operations.

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 Problem Instantiation: Many-Objective Modularization for ATL Transformations

We now instantiate our approach for ATL by performing three steps (cf. also Fig. 10).[2] 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.

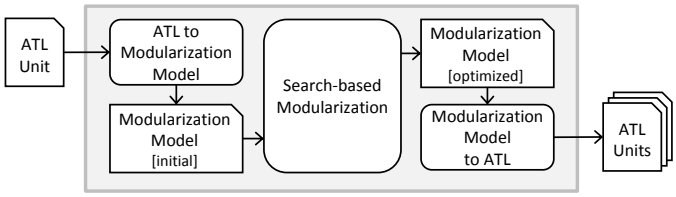


Fig. 10. Overview of the ATL modularization approach.

3.2.1 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 [42]. 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) [43] 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

2. The complete transformation chain with several case studies is available at our project website: https://github.com/martin-fleck/momot/tree/master/examples/tse

that participate in the rules. This means that it needs to extract the types of the elements that are reached by OCL navigations [44]. 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 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 [42].

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 Search-based Modularization

Having the modularization model at hand, we apply our search-based framework MOMoT [45], [46], to find the Pareto-optimal module structure. MOMoT[3] is a task- and algorithm-agnostic approach that combines SBSE and MDE. It has been developed in previous work [45] and builds upon Henshin[4] [13] to define model transformations and the MOEA framework[5] 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 Fig. 9. Before deciding which elements go into which module, we have 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 3.1.3 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. MOMoT: http://martin-fleck.github.io/momot/
4. Henshin: http://www.eclipse.org/henshin
5. MOEA Framework: http://www.moeaframework.org

3.2.3 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 3.1.2, 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.

4 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 subsections, 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.

4.1 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 [47], [48], [49]. 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 select 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

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.

4.2 Experimental Setup

4.2.1 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 [4], 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 [50] 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 [51].

CS3 **R2ML2RDM:** This transformation is part of the sequence of transformations to convert OCL models into SWRL (Semantic Web Rule Language) rules [52]. 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 elements, i.e., rules and helpers. Table 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.

TABLE 2
Size and structure of all case studies.

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

4.2.2 Evaluation Metrics

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

Search Performance Metrics (RQ1, RQ2): 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.
- Inverted Generational Distance (IGD): A number of performance metrics for multi-objective optimization has been proposed and discussed in the literature, which aims to evaluate the closeness to the Pareto optimal front and the diversity of the obtained solution set, or both criterion. Most of the existing metrics require the obtained set to be compared against a specified set of Pareto optimal reference solutions. In this study, the IGD is used as the performance metric since it has been shown to reflect both the diversity and convergence of the obtained non-dominated solutions. The IGD corresponds to the average Euclidean distance separating each reference solution from its closest non-dominated one. Note that for each system, considered in our experiments, we use the set of Pareto optimal solutions generated by all algorithms over all runs as reference solutions. Better convergence is indicated by lower values.

Solution Correctness Metrics (RQ3.1, RQ3.2): 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 2.1.4, there is not a single modularized solution in the ATL Zoo [4], 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 elements which are in the same module and should be, FPs are elements which are in the same module but should not be and FNs are elements 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 elements, i.e., rules and helpers, which are modularized meaningfully, over the total number of transformation elements. MP is given by the following equation

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

  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 [53]. 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 [54] 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 (RQ4.1, RQ4.2): In order to evaluate research questions RQ4.1 and RQ4.2, we consider two dimensions of usability: the estimated difficulty and the time (T) 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 4.2.4) 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 4.2.4) 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

As a helpful remainder for the rest of this evaluation, Table 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 4.3.5.

TABLE 3
Evaluation metric and type of study for each Research Question (RQ).

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

4.2.3 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 [55] with a 99% significance level ($\alpha = 0.01$). The Mann-Whitney U test [56], 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 [57]. 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.

4.2.4 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 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 4.2.2). These manual solutions are developed by members of our research groups which have knowledge of ATL but are not affiliated with this paper. 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. 9 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 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 4.2.2.

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 [58], [59], [60], 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 [59]. Out of the several different faults, we have applied changes in the *navigation* (according to the *relation to another class change* mutation operator [59]) 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 paper. 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 4.2.2). We did not find a consistent difference between the students and professionals when we checked the results based on the same questions. One of the reasons could be related to the fact that most of the students considered in our experiments have also good experience in industry and very well familiar with ATL and model transformations.

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 4.

TABLE 4
Assignment of groups to tasks and case studies (no group is allowed to perform a task on the same case study twice).

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

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, 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.

4.2.5 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 [61] to provide a wider range of diversity among the solutions in the Pareto front approximation. 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 3.1.3. As a reminder, these objectives are the number of modules in the transformation (NMT), the difference between the number of transformation elements, i.e., rules and helpers, in the module with the lowest number of elements and the module with the highest number of elements (DIF), the cohesion ratio (COH) and the coupling ratio (COP). The initial objective values for each case study are listed in Table 5.

4.3 Result Analysis

4.3.1 Results for RQ1

To answer RQ1 and therefore evaluate whether a sophisticated approach is needed to tackle the model transformation modularization 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 [47]. Specifically, in our evaluation we investigate the Hypervolume indicator (IHV) and the Inverted Generational Distance indicator (IGD), cf. Section 4.2.2, on 30 independent algorithm runs for all case studies.

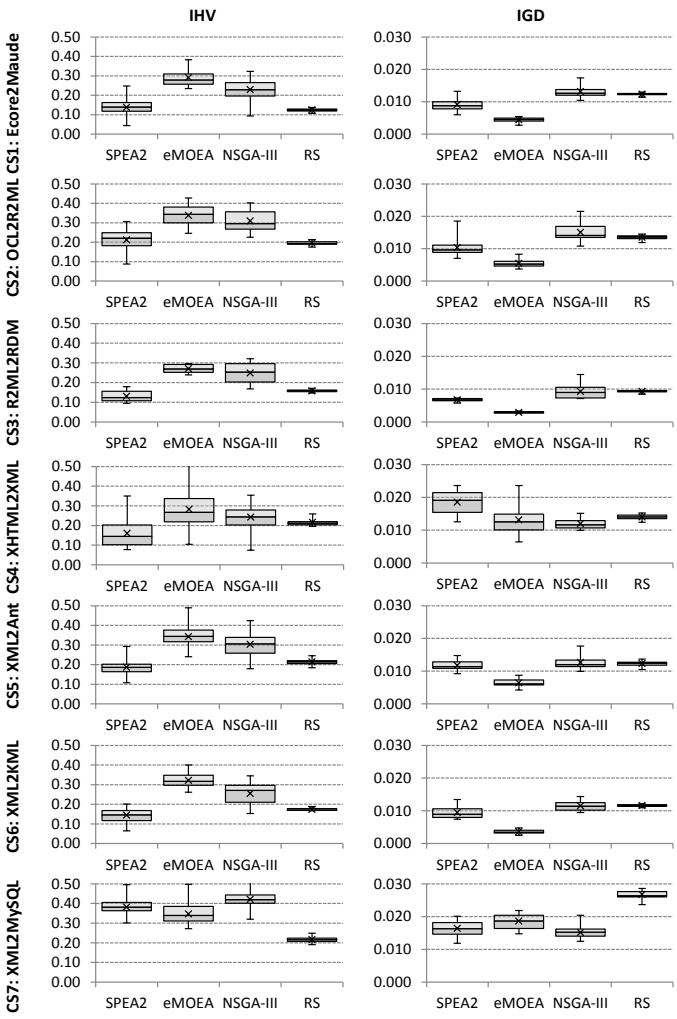


Fig. 11. Hypervolume (IHV) and Inverted Generational Distance (IGD) indicator for all case studies and algorithms. The 'x'marks the mean value retrieved from a specific algorithm for a specific case study. All results are retrieved from 30 independent algorithm runs.

TABLE 5
Initial objective values for all seven case studies (the arrow next to the objective name indicates the direction of better values).

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

The results of our evaluation are depicted in Fig. 11. The details of p-value and effect for each case study for the IHV and IGD metrics are given in Table 7 and in Table 8, respectively. In Fig. 11, 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 [62]. 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: (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 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.

TABLE 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.

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

4.3.2 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: IHV and IGD. All results are retrieved from 30 independent algorithm 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 Fig. 11. The details of p-value and effect for each case study for the IHV and IGD metrics are given in Table 7 and in Table 8, respectively. As Fig. 11 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 compared 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 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 [36]. 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 for our 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 as MOMoT is configurable in this respect [45].

4.3.3 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 4.2.4). 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 4.2.2.

Our findings for the average precision (PR) for each algorithm and for all case studies are summarized in Fig. 12. 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 useful. 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.

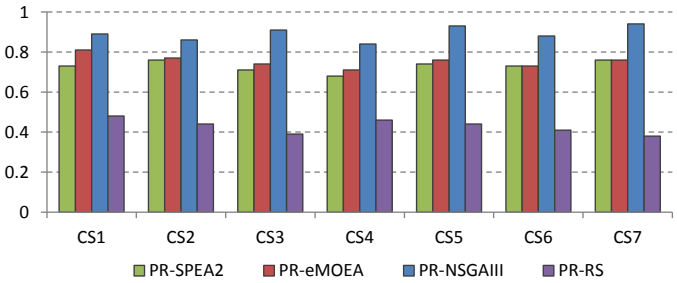


Fig. 12. Quantitative correctness evaluation using precision (PR) for all case studies and algorithms (higher values indicate better results).

A similar result can be seen for recall (RE) depicted in Fig. 13, 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

TABLE 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).

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 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	IGD	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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

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.

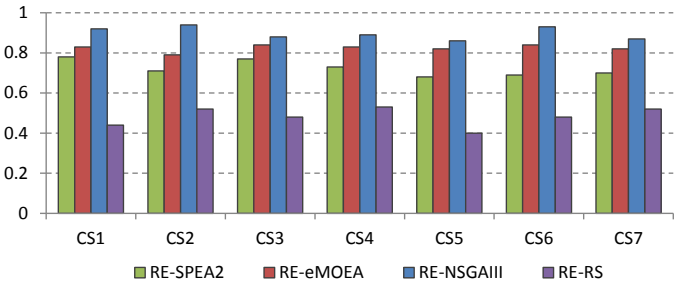


Fig. 13. Quantitative correctness evaluation using recall (RE) for all case studies and algorithms (higher values indicate better results)

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, 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.

4.3.4 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 4.2.4) 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 elements as described in Section 4.2.2.

The summary of our findings based on the average MP for all considered algorithms and for all case studies is depicted in Fig. 14. 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).

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.

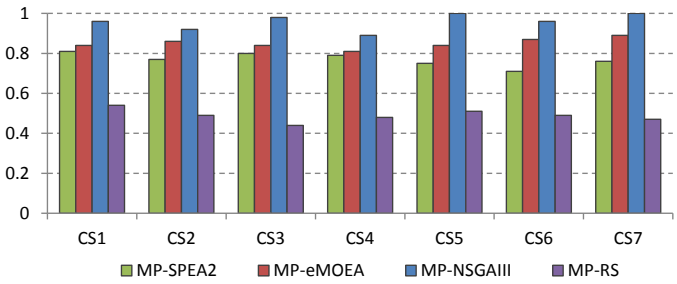


Fig. 14. Qualitative correctness evaluation using manual precision (MP) for all case studies and algorithms (higher values indicate better results).

4.3.5 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 4.2.4. 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 Fig. 15, *CS1-T1a Original* to *CS4-T1b Modularized*. The statistical test concerning the p-value and effect is provided in Table 9. In Fig. 15 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 Fig. 17, *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 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

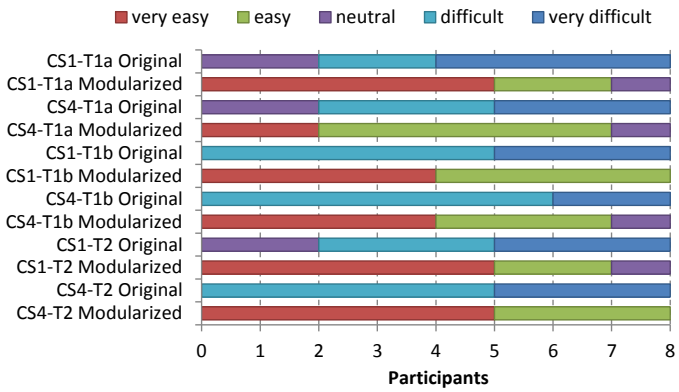


Fig. 15. Evaluation of experienced difficulty to fulfill the user study tasks T1 and T2 for Ecore2Maude (CS1) and XHTML2XML (CS4): original vs modularized transformation.

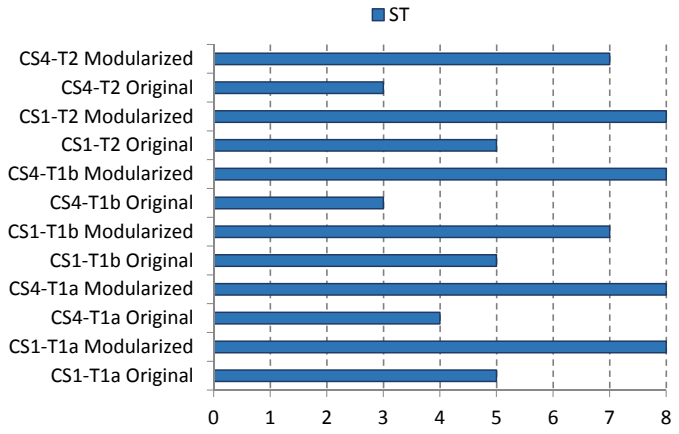


Fig. 16. Evaluation of the number of participants who completed the tasks T1 and T2 successfully (ST) for Ecore2Maude (CS1) and XHTML2XML (CS4): original vs modularized transformation.

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 Fig. 16. In 4 out of the 6 tasks for Ecore2Maude (CS1) and XHTML2XML (CS4), all the 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.

4.3.6 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 4.2.4. 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

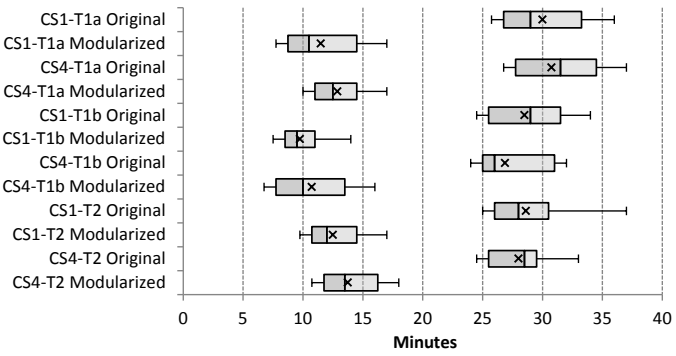


Fig. 17. Time needed for tasks T1 and T2 for Ecore2Maude (CS1) and XHTML2XML (CS4): original vs modularized transformation.

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 Fig. 15, *CS1-T2 Original* to *CS4-T2 Modularized*. The statistical test concerning the p-value and effect is provided in Table 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 Fig. 17, *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.

4.4 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 in the transformations of the ATL Zoo (cf. Section 2.1.4), 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 [63], [64], [65], but also about the non-functional aspects of model transformations [2], [66]. 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 [42], 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.

4.5 Threats to Validity

According to Wohlin et al. [67], 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.

4.5.1 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% ($\alpha = 0.01$) 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.

TABLE 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.

Task	Time	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

4.5.2 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.

4.5.3 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 [68], 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2017.2654255, IEEE Transactions on Software Engineering

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 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.

4.5.4 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.

5 Related Work

Concerning the contribution of this paper, we discuss three threads of related work. First, we summarize works considering modularization in the general field of software engineering. Second, we discuss modularization support in different transformation languages. Third, we summarize approaches combining search-based techniques and model transformations.

5.1 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 [69] 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 [70] 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. Magbool and Babri [71] 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 Azerpos [72] 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. Mancoridis et al. [73] 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 [74], [75] 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. [76] 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, [77] 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. [78] 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. [36] 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 paper, 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. [41] 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. While there are several modularization approaches, we are not aware of any approach dealing with peculiarities of modularizing rule-based transformations.

5.2 Modularization in Transformation Languages

The introduction of an explicit module concept going beyond rules as modularization concept [6] 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 [79], the declarative transformation languages TGGs [80] and QVT-R [79], and the hybrid transformation languages ETL [81] and RubyTL [82], [83]. 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 [84] 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.

5.3 SBSE and Model Transformations

Search-based approaches for model transformation have been first applied to learn model transformations from existing transformation examples, i.e., input/output model pairs. This approach is called model transformation by example (MTBE) [29], [30], [31]. Because of the huge search space one has to search for finding the best model transformations for a given set of input/output model pairs, search-based techniques have been applied to automate this complex task [85], [86], [87], [88], [89], [90]. While MTBE approaches do not include the search for modularization when searching for model transformations, we discussed in this paper 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 this paper as a prerequisite by reusing the MOMoT framework. There are two related approaches to MOMoT. First, Denil et al. [91] propose 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. [92] also address 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 [27] 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.

To conclude, while there are several approaches enabling search-based orchestration of transformation rules, we are not aware of any instantiation of these approaches for the transformation modularization problem tackled in this paper.

6 Conclusion and Future Work

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 this work, 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 number 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.

The promising results of our approach give 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 [93], and even megamodels [94].

Finally, 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 [95], [96], 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.

ACKNOWLEDGMENTS

This work has been partially supported by the Christian Doppler Forschungsgesellschaft and the BMWFW, Austria, by the European Commission (FEDER), by Spanish Government under CICYT project BELI (TIN2015-70560-R), the Andalusian Government project COPAS (P12-TIC-1867), and by the Ford Motor Company (Ford Alliance Program).

REFERENCES

[1] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–646, 2006.

[2] L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. M. K. Selim, E. Syriani, and M. Wimmer, "Model transformation intents and their properties," *SoSyM*, vol. 15, no. 3, pp. 647–684, 2016.

[3] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger, "Reuse in model-to-model transformation languages: are we there yet?" *SoSyM*, vol. 14, no. 2, pp. 537–572, 2015.

[4] Eclipse Modeling Project, "ATL Zoo," http://www.eclipse.org/atl/atlTransformations, 2015.

[5] A. Kusel, J. Schoenboeck, M. Wimmer, W. Retschitzegger, W. Schwinger, and G. Kappel, "Reality Check for Model Transformation Reuse: The ATL Transformation Zoo Case Study," in *Proc. of AMT*, 2013.

[6] I. Kurtev, K. van den Berg, and F. Jouault, "Rule-based modularization in model transformation languages illustrated with ATL," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 138–154, 2007.

[7] M. Harman, "The Current State and Future of Search Based Software Engineering," in *Proc. of FOSE @ ICSE*, 2007, pp. 342–357.

[8] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Trans. on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 8 2014.

[9] A. R. da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, pp. 139 – 155, 2015.

[10] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[11] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

[12] T. Mens and P. V. Gorp, "A Taxonomy of Model Transformation," *Electr. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, 2006.

[13] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations," in *Proc. of MODELS*, 2010, pp. 121–135.

[14] G. Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software," in *Proc. of AGTIVE*, 2003, pp. 446–453.

[15] J. de Lara and H. Vangheluwe, "AToM3: A Tool for Multi-formalism and Meta-modelling," in *Proc. of FASE*, 2002, pp. 174–188.

[16] J. E. Rivera, F. Duran, and A. Vallecillo, "A Graphical Approach for Modeling Time-dependent Behavior of DSLs," in *Proc. of VL/HCC*, 2009, pp. 51–55.

[17] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varró, "VIATRA - visual automated transformations for formal verification and validation of UML models," in *Proc. of ASE*, 2002, pp. 267–270.

[18] J. Greenyer and E. Kindler, "Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars," *SoSyM*, vol. 9, no. 1, pp. 21–46, 2010.

[19] J.-M. Jézéquel, O. Barais, and F. Fleurey, "Model Driven Language Engineering with Kermeta," in *Generative and Transformational Techniques in Software Engineering III*, 2011, pp. 201–221.

[20] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio, "JTL: A Bidirectional and Change Propagating Transformation Language," in *Proc. of SLE*. Springer, 2011, pp. 183–202.

[21] F. Jouault and I. Kurtev, "Transforming Models with ATL," in *Satellite Events at the MoDELS 2005 Conference*, 2006, pp. 128–138.

[22] Eclipse Modeling Project, "ATLAS Transformation Language – ATL," http://eclipse.org/atl, 2015.

[23] D. Wagelaar, R. V. D. Straeten, and D. Deridder, "Module superimposition: a composition technique for rule-based model transformation languages," *SoSyM*, vol. 9, no. 3, pp. 285–309, 2010.

[24] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] K. Deb and H. Jain, "Handling many-objective problems using an improved NSGA-II procedure," in *Proc. of CEC*, 2012, pp. 1–8.

[29] D. Varró, "Model Transformation by Example," in *Proc. of MoDELS*, 2006, pp. 410–424.

[30] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler, "Towards Model Transformation Generation By-Example," in *Proc. of HICSS*, 2007.

[31] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Model Transformation By-Example: A Survey of the First Wave," in *Conceptual Modelling and Its Theoretical Foundations*, 2012, pp. 197–215.

[32] J. Shelburg, M. Kessentini, and D. R. Tauritz, "Regression Testing for Model Transformations: A Multi-objective Approach," in *Proc. of SSBSE*, 2013, pp. 209–223.

[33] A. ben Fadhel, M. Kessentini, P. Langer, and M. Wimmer, "Search-based detection of high-level model changes," in *Proc. of ICSM*, 2012, pp. 212–221.

[34] D. Fatiregun, M. Harman, and R. M. Hierons, "Search Based Transformations," in *Proc. of GECCO*, 2003, pp. 2511–2512.

[35] ——, "Evolving transformation sequences using genetic algorithms," in *Proc. of SCAM*, 2004, pp. 66–75.

[36] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-Objective Software Remodularization Using NSGA-III," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 3, pp. 17:1–17:45, 2015.

[37] M. Harman and L. Tratt, "Pareto optimal search based refactoring at the design level," in *Proc. of GECCO*, 2007, pp. 1106–1113.

[38] H. Masoud and S. Jalili, "A clustering-based model for class responsibility assignment problem in object-oriented analysis," *JSS*, vol. 93, no. 0, pp. 110–131, 2014.

[39] M. Bowman, L. C. Briand, and Y. Labiche, "Multi-Objective Genetic Algorithm to Support Class Responsibility Assignment," in *Proc. of ICSM*, 2007, pp. 124–133.

[40] ——, "Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms," *IEEE Trans. on Software Eng.*, vol. 36, no. 6, pp. 817–837, 2010.

[41] K. Praditwong, M. Harman, and X. Yao, "Software Module Clustering as a Multi-Objective Search Problem," *IEEE Trans. Software Eng.*, vol. 37, no. 2, pp. 264–282, 2011.

[42] J. Troya, M. Fleck, M. Kessentini, M. Wimmer, and B. Alkhaze, "Rules and Helpers Dependencies in ATL–Technical Report," Universidad de Sevilla, Tech. Rep., 2016. [Online]. Available: http://www.lsi.us.es/~jtroya/documents/ATLDependenciesTechReport16.pdf

[43] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the Use of Higher-Order Model Transformations," in *Proc. of ECMDA-FA*, 2009, pp. 18–33.

[44] L. Burgueño, J. Troya, M. Wimmer, and A. Vallecillo, "Static Fault Localization in Model Transformations," *IEEE Trans. Software Eng.*, vol. 41, no. 5, pp. 490–506, 2015.

[45] M. Fleck, J. Troya, and M. Wimmer, "Marrying Search-based Optimization and Model Transformation Technology," in *Proc. of NasBASE*, 2015, pp. 1–16.

[46] ——, "Search-Based Model Transformations with MOMoT," in *Proc. of ICMT*, 2016, pp. 79–87.

[47] M. Harman, P. McMinn, J. Teixeira de Souza, and S. Yoo, "Search Based Software Engineering: Techniques, Taxonomy, Tutorial," in *Revised Tutorial Lectures of the International Summer Schools on Empirical Software Engineering and Verification (LASER)*, 2010, pp. 1–59.

[48] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, 2014.

[49] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, p. 11, 2012.

[50] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All About Maude – A High-Performance Logical Framework*. Springer, 2007.

[51] M. Milanovic, D. Gasevic, A. Giurca, G. Wagner, and V. Devedzic, "Towards sharing rules between OWL/SWRL and UML/OCL," *ECEASST*, vol. 5, 2006.

[52] M. Milanovic, "Modeling rules on the Semantic Web," Master's thesis, GOOD OLD AI Lab, Faculty of organizational sciences, University of Belgrade, 2007.

[53] S. Bechikh, L. Ben Said, and K. Ghédira, "Searching for Knee Regions of the Pareto Front Using Mobile Reference Points," *Soft Computing*, vol. 15, no. 9, pp. 1807–1823, 2011.

[54] L. Rachmawati and D. Srinivasan, "Multiobjective Evolutionary Algorithm with Controllable Focus on the Knees of the Pareto Front," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 810–824, 2009.

[55] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. of ICSE*, 2011, pp. 1–10.

[56] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[57] M. Holm.

[58] J. Troya, A. Bergmayr, L. Burgueno, and M. Wimmer, "Towards systematic mutations for and with ATL model transformations," in *Workshop Proc. of ICST*, 2015, pp. 1–10.

[59] J.-M. Mottu, B. Baudry, and Y. Le Traon, "Mutation analysis testing for model transformations," in *Proc. of ECMDA-FA*, 2006, pp. 376–390.

[60] F. Alhwikem, R. F. Paige, L. Rose, and R. Alexander, "A Systematic Approach for Designing Mutation Operators for MDE Languages," in *Proc. of MoDEVVa*, 2016, pp. 54–59.

[61] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining Convergence and Diversity in Evolutionary Multiobjective Optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, 2002.

[62] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Lawrence Erlbaum Associates Inc, 1988.

[63] J. Troya and A. Vallecillo, "A Rewriting Logic Semantics for ATL," *Journal of Object Technology*, vol. 10, pp. 5:1–5:29, 2011.

[64] J. Sánchez Cuadrado, E. Guerra, and J. de Lara, "Uncovering errors in ATL model transformations using static analysis and constraint solving," in *Proc. of ISSRE*, 2014, pp. 34–44.

[65] B. J. Oakes, J. Troya, L. Lúcio, and M. Wimmer, "Full contract verification for ATL using symbolic execution," *SoSyM*, pp. 1–35, 2016.

[66] S. Nalchigar, R. Salay, and M. Chechik, "Towards a Catalog of Non-Functional Requirements in Model Transformation Languages," in *Proc. of AMT*, 2013.

[67] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.

[68] A. E. Eiben and S. K. Smit, "Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[69] T. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization," in *WCRE*, 1997, pp. 33–43.

[70] N. Anquetil and T. Lethbridge, "Experiments with Clustering as a Software Remodularization Method," in *Proc. of WCRE*, 1999, pp. 235–255.

[71] O. Maqbool and H. A. Babri, "Hierarchical Clustering for Software Architecture Recovery," *IEEE Trans. Software Eng.*, vol. 33, no. 11, pp. 759–780, 2007.

[72] M. Shtern and V. Tzerpos, "Methods for selecting and improving software clustering algorithms," in *Proc. of ICPC*, 2009, pp. 248–252.

[73] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using Automatic Clustering to Produce High-Level System Organizations of Source Code," in *Proc. of IWPC*, 1998, pp. 45–52.

[74] B. S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool," *IEEE Trans. Software Eng.*, vol. 32, no. 3, pp. 193–208, 2006.

[75] ——, "On the evaluation of the Bunch search-based software modularization algorithm," *Soft Comput.*, vol. 12, no. 1, pp. 77–93, 2008.

[76] M. Harman, R. M. Hierons, and M. Proctor, "A New Representation And Crossover Operator For Search-based Optimization Of Software Modularization," in *Proc. of GECCO*, 2002, pp. 1351–1358.

[77] O. Seng, M. Bauer, M. Biehl, and G. Pache, "Search-based improvement of subsystem decompositions," in *Proc. of GECCO*, 2005, pp. 1045–1051.

[78] H. Abdeen, S. Ducasse, H. A. Sahraoui, and I. Alloui, "Automatic Package Coupling and Cycle Minimization," in *Proc. of WCRE*, 2009, pp. 103–112.

[79] OMG, *MOF QVT Final Adopted Specification*, Object Management Group, 2005.

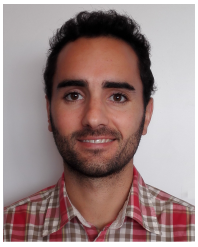
[80] F. Klar, A. Königs, and A. Schürr, "Model transformation in the large," in *Proc. of ESEC-FSE*, 2007, pp. 285–294.

[81] D. S. Kolovos, R. F. Paige, and F. Polack, "The Epsilon Transformation Language," in *Proc. of ICMT*, 2008, pp. 46–60.

[82] J. Cuadrado and J. García Molina, "Approaches for Model Transformation Reuse: Factorization and Composition," in *Proc. of ICMT*, 2008, pp. 168–182.

[83] J. S. Cuadrado and J. G. Molina, "Modularization of model transformations through a phasing mechanism," *SoSyM*, vol. 8, no. 3, pp. 325–345, 2009.
[84] J. S. Cuadrado, E. Guerra, and J. de Lara, "A component model for model transformations," *IEEE Trans. Software Eng.*, vol. 40, no. 11, pp. 1042–1060, 2014.
[85] M. Kessentini, H. A. Sahraoui, and M. Boukadoum, "Model Transformation as an Optimization Problem," in *Proc. of MODELS*, 2008, pp. 159–173.
[86] M. Kessentini, A. Bouchoucha, H. A. Sahraoui, and M. Boukadoum, "Example-Based Sequence Diagrams to Colored Petri Nets Transformation Using Heuristic Search," in *Proc. of ECMFA*, 2010, pp. 156–172.
[87] M. Kessentini, H. A. Sahraoui, M. Boukadoum, and O. Benomar, "Search-based model transformation by example," *SoSyM*, vol. 11, no. 2, pp. 209–226, 2012.
[88] I. Baki, H. A. Sahraoui, Q. Cobbaert, P. Masson, and M. Faunes, "Learning Implicit and Explicit Control in Model Transformations by Example," in *Proc. of MODELS*, 2014, pp. 636–652.
[89] M. Faunes, H. A. Sahraoui, and M. Boukadoum, "Genetic-Programming Approach to Learn Model Transformation Rules from Examples," in *Proc. of ICMT*, 2013, pp. 17–32.
[90] H. Saada, M. Huchard, C. Nebut, and H. A. Sahraoui, "Recovering model transformation traces using multi-objective optimization," in *Proc. of ASE*, 2013, pp. 688–693.
[91] J. Denil, M. Jukss, C. Verbrugge, and H. Vangheluwe, "Search-Based Model Optimization Using Model Transformations," in *Proc. of SAM*, 2014, pp. 80–95.
[92] H. Abdeen, D. Varró, H. A. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth, "Multi-objective optimization in rule-based design space exploration," in *Proc. of ASE*, 2014, pp. 289–300.
[93] M. Fleck, J. Troya, and M. Wimmer, "Towards generic modularization transformations," in *Companion Proc. of Modularity*, 2016, pp. 190–195.
[94] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *Proc. of the Workshop on Best Practices for Model-Driven Software Development*, 2004.
[95] A. Feldthaus and A. Møller, "Semi-Automatic Rename Refactoring for JavaScript," in *Proc. of OOPSLA*, 2013, pp. 323–338.
[96] A. Thies and C. Roth, "Recommending rename refactorings," in *Proc. of RSSE*, 2010, pp. 1–5.

Martin Fleck received his PhD degree in computer science from the TU Wien in May 2016 for his thesis on combining search-based optimization methods and model transformations. Previously, he has been working in the Business Informatics Group at the TU Wien as a project assistant for the EU project ARTIST. Since May 2016, he has been working at EclipseSource Vienna where he applies different modeling technologies in industrial settings. His current research interests are search-based software engineering, model-driven engineering, and model versioning.



Javier Troya is a post-doctoral researcher at the University of Seville since January 2016. Before that, he was for two years in the Business Informatics Group at the TU Wien. In 2013, he obtained his International PhD with honors from the University of Malaga, supervised by Prof. Antonio Vallecillo. During his PhD, he spent five months working with Prof. Iman Poernomo's group at King's College London, and four months at University of Waterloo (Ontario, Canada) with Prof. Krzysztof Czarnecki's group. His PhD thesis dealt with the model-driven performance and reliability analysis of dynamic systems. His current research interests comprise analysis and testing of model transformations, non-functional properties monitoring and search-based software engineering.

Marouane Kessentini is an assistant professor in software engineering at the University of Michigan, Dearborn campus. He received his PhD end of 2011 from University of Montreal. He is the founding director of the Search Based Software Engineering Lab in Michigan and he has several collaborations with different industrial companies on studying software engineering problems by computational search and artificial intelligence techniques. Dr. Kessentini has three best paper awards and he holds also best dissertation award in 2012 from University of Montreal. He is very active in the area of software refactoring with around 60 papers published in several software engineering journals and conferences. Dr. Kessentini served as a program committee member and chair of several software engineering conferences. He is also the founder of the North American SBSE conference that was held in Michigan on February 2015 (funded by NSF).

Manuel Wimmer is an assistant professor in the Business Informatics Group (BIG) at the TU Wien. He received his PhD degree in business informatics from the TU Wien in 2008. In 2014, he received his Habilitation (venia docendi) in Informatics from the TU Wien. He was guest professor in the Software Engineering Research Group at the Philipps-University Marburg and in the Institute of Automation and Information Systems at the TU München. Furthermore, he was a research associate at the University of Málaga. He is a co-author of the book Model-driven Software Engineering in Practice (Morgan & Claypool, 2012).

Bader Alkhazi is currently a PhD student in the Search Based Software Engineering Lab at the University of Michigan under the supervision of Prof. Marouane Kessentini. He received his master's degree in December 2014 from the University of Michigan. His PhD project is concerned with the application of SBSE techniques in different areas such as web services, MDE, refactoring and automotive industry. His current research interests are search-based software engineering, model-driven engineering and software quality.