Automated Co-Evolution of Metamodels and Transformation Rules: A Search-Based Approach

Wael Kessentini[1], Houari Sahraoui[1], and Manuel Wimmer[2]

[1] University of Montreal,
kessentw,sahraouh@iro.umontreal.ca,
[2] CDL-MINT, TU Wien,
wimmer@big.tuwien.ac.at

Abstract. Metamodels frequently change over time by adding new concepts or changing existing ones to keep track with the evolving problem domain they aim to capture. This evolution process impacts several depending artifacts such as model instances, constraints, as well as transformation rules. As a consequence, these artifacts have to be co-evolved to ensure their conformance with new metamodel versions. While several studies addressed the problem of metamodel/-model co-evolution[3], the co-evolution of metamodels and transformation rules has been less studied. Currently, programmers have to manually change model transformations to make them consistent with the new metamodel versions which require the detection of which transformations to modify and how to properly change them. In this paper, we propose a novel search-based approach to recommend transformation rule changes to make transformations coherent with the new metamodel versions by finding a trade-off between maximizing the coverage of metamodel changes and minimizing the number of static errors in the transformation and the number of applied changes to the transformation. We implemented our approach for the ATLAS Transformation Language (ATL) and validated the proposed approach on four co-evolution case studies. We demonstrate the outperformance of our approach by comparing the quality of the automatically generated co-evolution solutions by NSGA-II with manually revised transformations, one mono-objective algorithm, and random search.

Keywords: Model Transformation Evolution, Search-based Software Engineering, ATL

1 Introduction

Model-driven engineering (MDE) [2] relies on metamodels as first-class entities [21] which evolve to accommodate new features, improve structural and semantical concerns and fix errors [27]. While this evolution process is vital, it impacts several depending artifacts such as model transformations since transformation rules need to be adapted to new metamodels versions as they use the metamodel elements as part of their type system [15]. Thus, a systematic process is needed to guide the co-evolution

[3] Please note the potential name clash for the term co-evolution. In this paper, we refer to the problem of having to co-evolve different dependent artifacts in case one of them changes. We are not referring to the application or adaptation of co-evolutionary search algorithms.

of the transformations when the involved metamodels evolve [27]. However, currently, this co-evolution process is mostly done manually which leads to significantly increased fault-proneness and cost of maintenance [14, 15].

Several studies have been proposed for automated co-evolution within the MDE literature, cf. [14] for a survey. The co-evolution of metamodels and their models have been addressed using various techniques to make model instances consistent with new metamodel versions by translating metamodel changes into model changes using a set of manually defined rules [8] or automatically adapting models towards reducing the number of conformance errors with the metamodels [18]. In addition, the co-evolution of metamodels and constraints written in the Object Constraint Language (OCL) has also been studied to reduce OCL errors when evolving metamodels by localizing the set of constraints to repair, and then, fixing them either manually [19] or automatically [1]. However, the co-evolution of metamodels and transformation rules—although it is considered as a significant problem [15]—is still less studied with only a few studies that identify metamodel changes, then manually define templates to map the metamodel changes into co-changes applied to the transformations, e.g., cf. [9,10,12,20,23,24,26]. None of the existing studies addressed the central question of how to automate the metamodel/transformation co-evolution without the need to manually define higher-order transformations to map metamodel changes into transformation changes. These higher-order transformations are language specific and require the correct identification of metamodel changes which is a challenge on its own. As a result, the co-evolution of metamodels and transformations is still far from being automated.

This paper remedies the gap by proposing, as one of the first studies in the MDE literature, an automated approach to revise transformation rules when metamodels evolve. In particular, we focus on the automated co-evolution of transformations expressed in the ATLAS Transformation Language (ATL) [16]. We leverage the use of search-based software engineering algorithms [13] to deal with the large search space of possible co-evolution solutions to repair the rules based on three main criteria: maximizing the coverage of metamodel changes and minimizing the number of static errors in the transformation and the number of applied changes to the transformation. Since these objectives are intuitively conflicting, we used a multi-objective algorithm, based on NSGA-II [7], to find a trade-off between them when exploring the search space of possible transformation co-evolutions. We considered differently-sized transformations available in the ATL Zoo[4], a public repository of model transformations, to validate our approach by comparing the newly generated ATL rules by our approach and the expected rules that are manually co-evolved. Since it is the first formulation of the metamodels and transformation rules co-evolution as a search problem, we also compared our results to a mono-objective algorithm, combining the different objectives, and random search. Furthermore, we evaluated the performance of our automated co-evolution approach comparing to the manual correction of co-evolution issues by a total of 6 participants on one of the case studies. On average, for all of our four studied ATL projects, 89% of the proposed edit operations were correct while the random search, mono-objective and manual techniques have a correctness of respectively 41%, 66% and 76%.

[4] https://git.eclipse.org/c/gerrit/www.eclipse.org/atl.git/tree/atlTransformations

2 Background

2.1 Metamodels and Model Transformation

Model transformations are considered as the heart and soul of MDE [31]. Model transformations are not only used for deriving implementations out of models, but also to analyze, compare, merge, and improve models [25]. In this context, metamodels contribute important information for model transformations. In particular, they introduce the type systems which can be used in model transformation programs [6]. The elements contained in a metamodel are accessible through model transformation languages and represent essential information needed to formulate transformations. Figure 1(a) shows the model transformation pattern which illustrates that on the metamodel level the transformation is defined and executed on the model level. Of course, when metamodels change, this has a direct impact on the existing transformations as the referred types and features have to exist in the metamodels. Figures 1(b-c) show the cases of source metamodel evolution and target metamodel evolution and the required transformation co-evolutions, respectively. Please note that both cases may occur simultaneously. The quest is to find the corresponding delta (i.e., changes) to patch the transformation for a given metamodel delta.

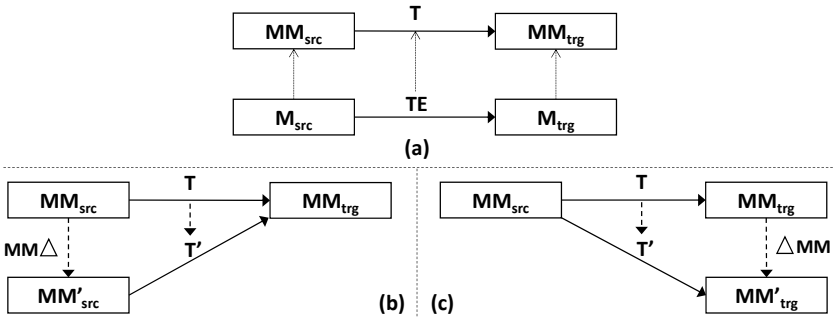

Fig. 1. Metamodel Evolution/Transformation Co-Evolution Context: (a) Model Transformation Pattern, (b) Source Metamodel Evolution/Transformation Co-Evolution, and (c) Target Metamodel Evolution/Transformation Co-Evolution; (b) and (c) may occur in combination.

ATL [16] is a model transformation language which follows the mentioned model transformation pattern. In particular, ATL transformations are rule-based programs (cf. *rule* keyword in Listing 1.1) which are executed on fixed input models to produce output models. For this process, matches in the input model are computed based on the input patterns (cf. *from* keyword in Listing 1.1) of the transformation rules which trigger the creation of output elements based on the output patterns (cf. *to* keyword in Listing 1.1) of the transformation rules. Please note that ATL transformations are typed by the source and target metamodels, i.e., the input and output pattern elements have to refer to existing elements in the involved metamodels. In addition, OCL expressions may be employed for filter definitions to restrict the matches in the input model as well as for computing values with so-called bindings for setting features of the produced output elements.

2.2 Metamodel/Transformation Co-Evolution: A Motivating Example

To further introduce ATL as well as to motivate the need of automatically repairing ATL transformations when metamodels evolve, an excerpt of an example ATL transformation is shown in Listing 1.1. Furthermore, we show in Figure 2 an evolution scenario for the input metamodel of the given ATL transformation.

The transformation example we are using is a simple transformation for generating documentation from class diagrams. In particular, we focus on transforming the features into list items. The content of the items is derived from the feature names and types–cf. the binding at line 8 of Listing 1.1.

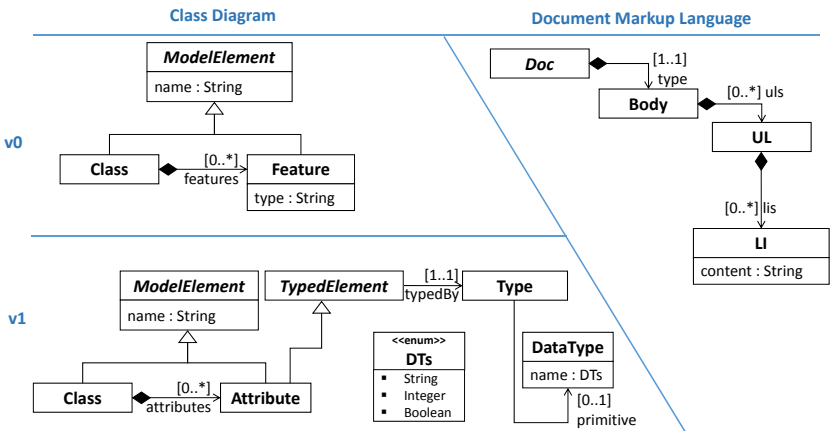

Fig. 2. Motivating Example: Metamodel Evolution

Listing 1.1. Excerpt of the initial and migrated Class2Doc Transformation.

```
1  -- deletions are shown in red, additions in green
2  module Class2Doc;
3  create OUT : Doc from IN : Class;
4  ...
5  rule Attribute2ListItem {
6    from f : Class! ~~Attribute~~Feature
7    to li : Doc!LI (
8      content <- f.name + '␣:␣' + f.~~type~~typedBy.primitive.name
9    )
10 }
11 ...
```

Let us assume that the class diagram language evolves by some rename refactorings as well as by explicating the types of features. This process results in a new metamodel version which now speaks about attributes instead of features. Attributes are typed elements whereas a typed element refers to an explicit type object which may describe the used types in more detail. Given the discussed changes in the source metamodel, the excerpt of the transformation example shown in Listing 1.1 has to be adapted. In particular, the type of the input pattern has to be changed as well as the binding for the *content* feature. The type of an attribute has now to be retrieved by following a navigation path before the required value can be accessed.

While there are already existing approaches for dealing with transformation co-evolution, most of them are based on certain change patterns such as the renaming refactoring in metamodels which may have an associated co-refactoring for the transformation rules. For more complex change patterns, as it is the case for retrieving the type information in our example by following a longer navigation path with several hops is currently not supported by existing approaches. Thus, we motivate our approach by the fact that for more complex evolution scenarios, a sophisticated search process is needed to repair the transformations to get rid of static typing errors in the transformation, but still have as much as possible the same behaviour as for the initial transformation. Furthermore, detecting metamodel changes precisely is still a challenge, especially when it comes to the detection of refactorings as is the case in our example. Our approach does not rely on computing such metamodel changes. Finally, while there are approaches for detecting static type errors in ATL programs [6], there are no approaches which consider this kind of information explicitly in the co-evolution process. Thus, one may end up with co-evolved transformations which have static type errors.

3 Multi-Objective Metamodel/Transformation Co-Evolution

3.1 Approach Overview

A co-evolution solution to our problem consists of a sequence of rule-level change operations to revise the existing transformation rules to make them conformed to the evolved source or target metamodel. The search space is determined not only by the number of possible rule-level change operations combination but also by the number of existing transformation rules, and the order in which these changes are applied. A heuristic-based optimization method is used to generate co-evolution solutions. The best solution should optimize 3 objectives: (1) minimize the number of errors; (2) minimize the number of recommended change operations to the transformation rules; and (3) maximize the coverage of the evolved source or target metamodel. To handle these conflicting objectives, we formulate this co-evolution problem as a multi-objective one using the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [7].

Our approach takes as inputs a model transformation program and the evolved source and target metamodels. It generates as output a sequence of recommended changes to the initial transformations program. To calculate the fitness functions, we used AnATLyzer [6] to identify the number of static type errors in the transformation rules and the ATL footprint tool [3] to estimate the coverage of the evolved source or target metamodels by the newly revised transformation rules.

3.2 NSGA-II Adaptation for Metamodel/Transformation Co-Evolution

Solution representation. A candidate solution to the problem is a set of revised transformation rules, i.e., a set of change operators applied to the initial transformation rules. A valid solution assigns a set of different rule-level changes to the transformation rules. We used a set of 27 types of operations that are defined in a previous study [5]. A complete description of all change operators for ATL can be found in [17].

We adopt the *vector-based* encoding where a candidate solution is represented as a *vector* of n positions, where n is the number of change operations to be applied to

Fig. 3. Solution encoding.

a transformation program. Each position corresponds to specific change operation. For instance, Figure 3 shows an example of a solution composed of three change operations applied to the transformation rules discussed in the motivating example (Listing 1.1). The generated solution included two types of change operations that were instantiated: NavigationModification(variable, navigationExpression, replacement) and InPatternElementModification(objectToModify, oldFeatureValue, replacement). Thus, the generation of solutions consists of selecting the type of operations and their parameters (objects to modify, rules to revise, etc.). The initial population is completely random where a maximum number of rule-level change operations n is fixed; then the generated changes are randomly assigned to several rules of the transformation.

Fitness Functions. We defined three objective functions in our adaptation.

Minimize the number of recommended rule-level changes. The underlying assumption to minimize the number of changes to the transformation is to reduce the effort of understanding the new program after evolution. Thus, this fitness function is defined as:

$$Minf1(S) = |S| \tag{1}$$

Where *S* is the solution to evaluate. Thus, this fitness function calculates the size of S (number of changes) which corresponds to the number of dimensions in the vector.

Minimize the number of transformation errors. We used AnATLyzer [6] to identify the number of errors in the transformation after applying the solution on the initial transformation. Thus, the second fitness function is defined as follows:

$$Minf2(S) = r \tag{2}$$

Where *r* is the number of errors identified in the revised transformation program. The errors are mainly the non-conformance between the metamodels and rules that can be statically detected by static semantic constraints for ATL transformations [6].

Maximize the metamodel coverage. The footprint tool of Burgueno et al. [3] estimates the coverage of the evolved source or target metamodels by the newly revised transformation after applying the recommended rule-level changes. The third fitness function is as follows:

$$Maxf3(S) = |re \cap mme| \tag{3}$$

Where *re* is the set of covered metamodel elements by the revised transformation rules as identified by the footprint tool and *mme* is the set of the evolved metamodel elements.

Evolutionary operators. Population-based search algorithms deploy crossover and mutation operators to improve the fitness functions of the solutions in the population in each iteration. Change operators such as crossover and mutation aim to drive the

search towards near-optimal co-evolution solutions. The *crossover* operator is responsible for creating new solutions based on already existing ones, e.g., re-combining solutions. In our adaptation, we use a single random cut-point crossover to construct offspring co-evolution solutions. It starts by selecting and splitting at random two-parent co-evolution solutions. Then crossover creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and vice versa for the second child.

The *mutation* operator is used to introduce slight random changes into candidate co-evolution solutions. This operator guides the algorithm into areas of the search space that would not be reachable through recombination alone and avoids the convergence of the population towards a few elite solutions. In our adaptation, we used a mutation operator that picks at random one or more positions (change operation) from their vector and replaces them by either another type of change operator or modifying the parameters of the operation type to apply it for another rule.

When applying crossover and mutation operators, we ensure the validity of the solution using a repair function. This function consists of removing edit operations from the solution when conflicts are detected using a set of constraints (redundancy, inapplicable edit operations after changes, etc.).

4 Validation

4.1 Research Questions and Evaluation Metrics

Our study addresses the following research questions:

- **RQ1 Solution Correctness:** To what extent do the co-evolution solutions generated by our approach compare to manually developed solutions?
- **RQ2 Benefits:** To what extent can our approach reduce the number of changes and manual effort to evolve the rules after a metamodel evolution?
- **RQ3.1 Search Validation:** Do we need a metaheuristic search for the metamodel/-transformation co-evolution problem?
- **RQ3.2 Search Quality:** How does the proposed multi-objective approach based on NSGA-II perform compared to a mono-objective one (aggregating the three objectives)?

Our research questions are evaluated using the following four ATL case studies. We selected to use ATL to validate our approach since it is one of the widely used model transformation languages [4, 30]. 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. For replication purposes, the different case studies used in our experiments along with a description of the used ATL change operations, the implementation of our approach, and the detailed 30 runs result of the different approaches can be found in [17].

We have selected these case studies due to their difference in size, structure and number of dependencies among their transformation artifacts, i.e., rules and helpers. Furthermore, the metamodel evolution scenarios used in our experiments were defined in a previous work based on the selected ATL case studies [11]. Table 1 summarizes,

Table 1. Selected ATL Case Studies.

ID	Name	Rules (R)	Coevolved Rules (CR)	Edit Operations (EOp)
Case Study 1 (CS1)	Ecore2Maude	40	12	21
Case Study 2 (CS2)	R2ML2RDM	58	16	34
Case Study 3 (CS3)	XHTML2XML	31	8	17
Case Study 4 (CS4)	XML2Ant	29	7	13

for each case study, the number of rules in the transformation (R), the number of rules to co-evolve/modify (CR) and the number of expected operations to fix the rules based on the manually created solutions in [11].

To see whether our approach produces sufficiently good results (RQ1), we compare our generated set of solutions with a set of manually created solutions based on a manual correctness measure (MC) defined as the intersection between the recommended changes operations and expected ones then divided by the number of expected operations. Since the number of correct recommendations may not be sufficient to evaluate the correctness, we evaluate the number of rules (FR) fixed by the recommended changes.

To evaluate the benefits of our approach (RQ2), we reported the execution time (T) of the different search algorithms to obtain good co-evolution solutions compared to manually fixing the transformation programs. Furthermore, we evaluate the ability of our approach to recommend the best co-evolution solutions with a minimum number of change operations (NOp).

To validate the problem formulation of our approach (RQ3.1), we compared our multi-objective approach with Random Search (RS), using MC, FR, and NOp, to justify the use of a metaheuristic search. If RS outperforms an intelligent search method, we can conclude that there is no need to use a metaheuristic search. To allow such a comparison, we used the knee-point [28] strategy to select a unique solution from each of the final Pareto sets of RS and NSGA-II. Thus, we identified the solution from the set of non-dominated ones providing the maximum trade-off using the following strategy when comparing between RS and NSGA-II. To find the maximal trade-off solution of the multi-objective algorithm, we use the trade-off worthiness metric proposed by Rachmawati and Srinivasan [28] to evaluate the worthiness of each non-dominated solution in terms of compromise between the objectives. This metric is expressed as follows:

$$\mu(x_i, S) = \underset{x_j \in S, x_i \not\prec x_j, x_j \not\prec x_i}{Min} T(x_i, x_j) \;\; \textit{where,}\; T(x_i, x_j) = \frac{\sum_{m=1}^{M} max\left[0, \frac{f_m(x_j) - f_m(x_i)}{f_m^{max} - f_m^{min}}\right]}{\sum_{m=1}^{M} max\left[0, \frac{f_m(x_i) - f_m(x_j)}{f_m^{max} - f_m^{min}}\right]}$$

We note that x_j denotes members of the set of non-dominated solutions S that are non-dominated with respect to x_i. The quantity $\mu(x_i, S)$ expresses the least amount of improvement per unit deterioration by substituting any alternative x_j from S with x_i. We note also that $f_m(x_i)$ corresponds to the m^{th} objective value of solution x_i and f_m^{max}/f_m^{min} corresponds to the maximal/minimal value of the m^{th} objective in the population individuals. In the above equations, normalization is performed to prevent some objectives being predominant over others. In the last equation, the numerator

expresses the aggregated improvement gained by substituting x_j with x_i. However, the denominator evaluates the deterioration generated by the substitution.

To evaluate the need for a multi-objective approach, we compared the results of our NSGA-II approach with the results retrieved from a mono-objective Genetic Algorithm (GA) aggregating the three fitness functions into one (with equal weights to all the objectives after normalizing them in the range [0,1]).

We limited the investigation of the relevance of our automated approach comparing to manually fixing the co-evolution issues to only CS1. CS1 represents the average case among the four case studies regarding the complexity (rules, expected edit operations and co-evolved rules) since the most complex case study is CS2 and the simplest one is CS4. Thus, the use of CS1 can be a good representative among all the case studies. Our study involved 6 master students in Software Engineering. All the participants are volunteers and familiar with MDE and co-evolution/refactoring since they are part of a graduate course on Software Quality Assurance (SQA). All the graduate students have already taken at least one position as software developer/engineer in industry for at least three years and most of them (5 out of 6 students) participated in similar experiments in the past, either as part of a research project or during the SQA graduate course. Furthermore, 3 out of the 6 students are currently working as full-time or part-time developers in the software industry. Participants were first asked to fill out a pre-study questionnaire containing four questions. The questionnaire helped to collect background information such as their modeling experience, and their familiarity with MDE and co-evolution/refactoring. Also, all the participants attended one lecture about model transformations and ATL, and passed four tests to evaluate their performance in evaluate and suggest co-evolution solutions.

4.2 Parameters Setting and Statistical Tests

The initial population/solution of NSGA-II, GA and RS are completely random. The stopping criterion for all the studied search algorithms is 100,000 evaluations. After several trial runs of the algorithms, the parameter values of the three techniques are fixed to 100 as population size and 20,000 iterations. For the change operators, we set crossover rate to 0.8 and mutation at 0.3 probability. We used a high mutation rate to ensure the diversity of the population and avoid premature convergence to occur. Indeed, there are no general rules to determine these parameters, and thus, we set the combination of parameter values by the trial-and-error method.

Our experimental study is based on 30 independent simulation runs for each problem instance, and the obtained results are statistically analyzed by using the Wilcoxon rank sum test with a 95% confidence level (α = 5%). In fact, for each problem instance, we compute the p-value obtained by comparing the results of the different algorithms with our approach. In this way, we determine whether the performance difference between our technique and one of the other approaches is statistically significant or just a random result. The Wilcoxon rank sum test verifies whether the results are statistically different or not; however, it does not give any idea about the difference in magnitude. Thus, we used the Vargha-Delaney A measure which is a non-parametric effect size measure.

Table 2. Mean manual correctness (MC) based on 30 runs for NSGA-II, RS, and GA.

Manual Correctness	Case Study 1	Case Study 2	Case Study 3	Case Study 4
NSGA-II	19/21 (90%)	29/34 (85%)	14/17 (82%)	13/13 (100%)
Genetic Algorithm	13/21 (61%)	19/34(55%)	13/17(76%)	10/13 (74%)
Random Search	9/21 (42%)	10/34 (29%)	8/17 (45%)	6/13 (48%)

Table 3. Mean number of fixed rules (FR) based on 30 runs for NSGA-II, RS, and GA.

Fixed Rules	Case Study 1	Case Study 2	Case Study 3	Case Study 4
NSGA-II	11	14	7	7
Genetic Algorithm	6	9	5	4
Random Search	5	6	3	3

4.3 Results

Results for RQ1. As reported in Table 2, the majority of the ATL changes recommended by our multi-objective approach were correct and similar to the ones manually applied by developers in [11], for the different evolution scenarios. On average, for all of our four studied projects, 89% of the proposed ATL change operations are correct. We decided to compare our recommendations with the ones manually proposed in another study (rather than manually checking the proposed solutions) to avoid biasing our experiments with our judgments. The highest MC score is 100% where all the changes applied to the ATL program were correct for the XML2Ant program, and the lowest score is 82% for the R2ML2RDM transformation program. Thus, it is clear that the results are independent of the size of the ATL programs and the number of recommended changes. The deviation between the expected and recommended rule-level change operations is limited up-to four which means that the number of recommended changes was similar to the expected ones.

Table 3 shows that the recommended co-evolution solutions fixed most of the ATL transformation rules to make them consistent with the source or target metamodel evolution. The maximum number of rules that were not fixed are two (the case of R2ML2RDM) and for the remaining cases, up-to only one rule remains to be fixed by the designer manually. Some of these rules are hard to fix automatically due to a significant number of non-trivial metamodel changes that renamed several elements.

Results for RQ2. Table 4 shows that our approach requires a reasonable execution time to converge towards good co-evolution solutions within less than 20 minutes. The highest execution time was reported on the largest case study of Ecore2Maude (19.5 minutes) and the lowest one on XML2Ant (9 minutes). The execution time is significantly lower than the average of two hours spent by developers to fix the ATL programs manually as reported in [11]. Furthermore, the number of errors detected after applying the recommended changes to the ATL rules was limited up-to two rules which may require low effort from the developers to fix them rather than writing all the co-evolution changes manually.

Table 5 describes the number of changes to be applied on the ATL programs to make them consistent with the new metamodels. It is clear that the number of changes

Table 4. Mean execution time (T) based on 30 runs for NSGA-II, GA, and RS.

Execution Time (Minutes)	Case Study 1	Case Study 2	Case Study 3	Case Study 4
NSGA-II	19.5	17.4	12	9
Genetic Algorithm	15.4	14.2	8.2	7.1
Random Search	8.3	7.2	2.5	3

Table 5. Number of edit operations (Nop) mean values of NSGA-II, GA, and RS over 30 independent runs.

Recommendation	Case Study 1	Case Study 2	Case Study 3	Case Study 4
NSGA-II	18	30	15	13
Genetic Algorithm	24	37	22	15
Random Search	29	42	26	20

is correlated with the number of rules to evolve and the metamodel changes. However, our multi-objective approach generated the minimum number of changes compared to the two other approaches as detailed later. The highest number of changes is 29 to evolve a total of 16 rules, which is reasonable since our tool enables the automated execution and testing of these changes.

Results for RQ3.1. and RQ3.2 The results summarized in Tables 2-5 confirm that NSGA-II is better than random search based on the different evaluation metrics of MC, FR, and Nop on all four ATL case studies. The average manual correctness values of random search on the different ATL programs are lower than 41%. RS also proposed the highest number of errors and number of recommendations among all the algorithms with the lowest number of fixed rules. This can be explained by the huge search space to explore to generate relevant rule-level changes.

Tables 2-5 confirm the average superior performance of our multi-objective approach compared to a mono-objective GA. Table 2 shows that our approach provides significantly higher manual correctness results (MC) than a mono-objective formulation having MC scores between 55% and 76% on the different ATL programs. The same observation is valid for FR and NOp as described in Tables 5 and 3. Thus, it is clear that all the three different objectives considered in our formulation are conflicting justifying the outperformance of NSGA-II.

Table 6. Statistical tests summary. A + symbol at the i^{th} position means that the evaluation metric value of algorithm A is statistically different from algorithm B on CSi. A - symbol at the i^{th} position means the opposite.

	NSGA-II (CS1,CS2,CS3,CS4)				GA (CS1,CS2,CS3,CS4)				RS (CS1,CS2,CS3,CS4)			
	MC	T	Nop	FR	MC	T	Nop	FR	MC	T	Nop	FR
NSGA-II					(++++)	(++++)	(++++)	(++++)	(++++)	(++++)	(++++)	(++++)
GA	(++++)	(++++)	(++++)	(++++)					(-+++)	(++++)	(+++-)	(-++-)
RS	(++++)	(++++)	(++++)	(++++)	(-+++)	(++++)	(+++-)	(-++-)				

Since our proposal is based on multi-objective optimization, it is important to evaluate the execution time (T). It is evident that NSGA-II requires a higher execution time than RS and GA since NSGA-II is considering more objectives and evolutionary operators. All the search-based algorithms under comparison were executed on machines with Intel i7 processors 4Ghz and 8 GB RAM. Overall, RS and GA algorithms were faster than NSGA-II. In fact, the average execution times for NSGA-II, GA and RS were respectively 14.5, 11 and 6 minutes. However, the execution for NSGA-II is still reasonable because the algorithm is not executed daily by the developers, and the co-evolution of ATL programs is not a real-time problem.

An average of 16 edit operations (mean value among all participants) were correctly identified manually by the subjects, which corresponds to 76% as average manual correctness. Our automated multi-objective approach successfully recommended an average of 19 edit operations out of the expected 21 operations (91% of manual correctness). The minimum number of manually identified correct edit operations is 14 (one participant), and the maximum is 17 (two participants) while three participants correctly identified 16 operations. Our automated approach successfully fixed, on average, a total of 11 out of 12 rules which outperforms the average number of rules fixed manually, which corresponds to 9 rules. A maximum of 10 rules was fixed manually by two participants while one participant was able only to fix 8 rules. The controlled experiment was limited to two hours thus all the results are obtained in two hours, which is significantly higher than the execution time of our approach limited to an average of 19 minutes. Thus, our automated approach can significantly improve the productivity of developers during the evolution process.

The results of our experiments, on all the case studies, algorithms and the evaluation metrics, were found to be statistically significant on 30 independent runs using the Wilcoxon rank sum test with a 95% confidence level when comparing our multi-objective approach to the remaining techniques (RS, GA and manual) as described in table 6. In our experiments, we have found the following results as well: (a) on small-scale programs (XHTML2XML and XML2Ant) our approach is better than all the other algorithms based on all the performance metrics with an A effect size higher than 0.92; and (b) on medium and large-scale programs (Ecore2Maude, and R2ML2RDM), our approach is better than all the other algorithms with an A effect size higher than 0.88 using all the evaluation metrics.

4.4 Threats To Validity

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. The parameters tuning of the different optimization algorithms used in our experiments creates an internal threat that we need to evaluate in our future work. The parameters' values used in our experiments are found by trial-and-error. However, it would be an interesting perspective to design an adaptive parameter tuning strategy for our approach so that parameters are updated during the execution to provide the best possible performance.

Internal validity is concerned with the causal relationship between the treatment and the outcome. We dealt with internal threats to validity by performing 30 independent simulation runs for each problem instance. This makes it highly unlikely that the ob-

served results were caused by anything other than the applied multi-objective approach. However, the comparison between multi-objective and mono-objective approaches is challenging since multiple solutions are generated by NSGA-II while the GA algorithm can generate only one co-evolution solution. We selected, in our experiments, the solution that represents the maximum trade-off between the three objectives (knee-point [7]) to compare with the GA's solution. However, we treated the different objectives with equal weights in our GA adaptation, which can be considered as an internal threat.

External validity refers to the generalizability of our findings. In this study, we performed our experiments on four different ATL programs belonging to different domains and having different sizes. However, we cannot assert that our results can be generalized to other programs. In addition, our study was limited to the use of specific change types related to ATL rules. Furthermore, the manual evaluation was limited to only one case study and a total of 6 participants. Thus, the main threats are the difficulty in generalizing the obtained manual results and the impact of participants expertise on them. To deal with these threats, we selected CS1 that represents the average case among the four case studies regarding the complexity (#rules, #expected edit operations, and #co-evolved rules). Furthermore, the participants are selected based on their experience in MDE, thus they can be representative of the average expertise of developers in practice.

5 Related Work

Co-evolution in the area of MDE has been heavily studied in the last decade [14]. The starting point was the metamodel/model co-evolution challenge [8] which attracted much research interest in dealing with large migration spaces [29]. In this context, search-based approaches have been proposed [18]. However, other co-evolution scenarios are understudied. We now outline work which has been done for co-evolving OCL expressions and model transformations.

Concerning the co-evolution of OCL expressions, dedicated approaches have been presented very recently. Approaches which are based on coupling changes for metamodels with co-changes for OCL expressions are presented in [19,22]. The main goal of these approaches is to repair OCL expressions for a set of provided metamodel change types. A search-based formulation of this problem has been also proposed [1].

Concerning transformation co-evolution, several approaches followed the idea of building on a set of metamodel changes for which co-changes for transformations can be derived [9, 10, 12, 20, 23, 24, 26]. For instance, Levandovsy et al. [23] proposed a higher-order transformation to adapt existing transformations. They classify metamodel changes, with respect to the effect on transformations into three categories [23]: (i) fully automated, i.e., changes affecting existing transformations that can be automatically migrated without user intervention, (ii) partially automated, i.e., changes or modifications that affect existing transformations which can be adapted automatically, even though some manual fine-tuning is required to complete the adaptation, and (iii) fully semantic, i.e., changes that effect transformations that cannot be automatically migrated, and the user has to completely define the adaptation.

All the mentioned approaches require the full correctness of the detected metamodel changes, which is still a challenge, especially when it comes to the intention behind the changes. Furthermore, the co-evolution is only possible for a set of predefined change

types. In our work, we do not require the metamodel changes and use a larger set of transformation co-evolution rules and a sophisticated search algorithm which allows migrating a transformation in any promising direction.

6 Conclusion

We propose, in this paper, an automated approach for metamodel/transformation co-evolution that finds a trade-off between different three objectives. Our approach allows developers to benefit from search-based rule-level change recommendations without defining a generic template to map metamodel changes into rule-level changes. To evaluate the effectiveness of our tool, we conducted a study based on four evolution scenarios of the source or target metamodels of ATL programs and compared it with random search, mono-objective formulation and manual technique. Our evaluation results provide evidence that our tool improves the applicability and automation of existing co-evolution techniques between metamodels and transformation rules.

Future work involves validating our technique with additional types of rule-level changes, more multi-objective algorithms and other transformation languages to conclude about the general applicability of our methodology. We focused, in this paper, on checking the correctness of the co-evolution solutions. We will use the quality indicators, such as the Hypervolume, when we compare between intelligent search algorithms such as MOPSO vs. NSGA-II.

Acknowledgements. This work has been partially funded by the Austrian Federal Ministry of Science, Research and Economy, National Foundation for Research, Technology and Development, by the Austrian Science Fund (FWF) P 28519-N31, and by the Canada NSERC grant RGPIN/06702-2014.

References

1. E. Batot, W. Kessentini, H. A. Sahraoui, and M. Famelis. Heuristic-based recommendation for metamodel - OCL coevolution. In MoDELS, pages 210–220, 2017.
2. M. Brambilla, J. Cabot, and M. Wimmer. Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers, 2nd edition, 2017.
3. L. Burgueo, J. Troya, M. Wimmer, and A. Vallecillo. Static fault localization in model transformations. IEEE Transactions on Software Engineering, 41(5):490–506, 2015.
4. Z. Cheng, R. Monahan, and J. F. Power. A sound execution semantics for atl via translation validation. In ICMT, pages 133–148, 2015.
5. J. S. Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL transformations with speculative analysis. Software & Systems Modeling, 2016.
6. J. S. Cuadrado, E. Guerra, and J. de Lara. Static analysis of model transformations. IEEE Trans. Software Eng., 43(9):868–897, 2017.
7. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.
8. D. Di Ruscio, L. Iovino, and A. Pierantonio. What is Needed for Managing Co-evolution in MDE? In Workshop on Model Comparison in Practice, pages 30–38, 2011.
9. H. Ehrig, K. Ehrig, and C. Ermel. Refactoring of model transformations. ECEASST, 2009.
10. J. Etzlstorfer, E. Kapsammer, and W. Schwinger. On the evolution of modeling ecosystems: An evaluation of co-evolution approaches. In MODELSWARD, pages 90–99, 2017.

11. M. Fleck, J. Troya, M. Kessentini, M. Wimmer, and B. Alkhazi. Model transformation modularization as a many-objective optimization problem. IEEE Trans. Software Eng., 43(11), 2017.
12. J. García, O. Díaz, and M. Azanza. Model transformation co-evolution: A semi-automatic approach. In SLE, pages 144–163, 2012.
13. M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv., 45(1):11:1–11:61, 2012.
14. R. Hebig, D. E. Khelladi, and R. Bendraou. Approaches to co-evolution of metamodels and models: A survey. IEEE Trans. Software Eng., 43(5):396–414, 2017.
15. L. Iovino, A. Pierantonio, and I. Malavolta. On the impact significance of metamodel evolution in MDE. Journal of Object Technology, 11(3):1–33, 2012.
16. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. Sci. Comput. Program., 72(1-2):31–39, 2008.
17. W. Kessentini. https://sites.google.com/site/coevolutionkessentini/data.
18. W. Kessentini, H. A. Sahraoui, and M. Wimmer. Automated metamodel/model co-evolution using a multi-objective optimization approach. In ECMFA, pages 138–155, 2016.
19. D. E. Khelladi, R. Bendraou, R. Hebig, and M. Gervais. A semi-automatic maintenance and co-evolution of OCL constraints with (meta)model evolution. Journal of Systems and Software, 134:242–260, 2017.
20. S. Kruse. On the use of operators for the co-evolution of metamodels and transformations. In Models and Evolution Workshop, 2011.
21. T. Kühne. Matters of (Meta-)Modeling. SoSyM, 5(4), 2006.
22. A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Retschitzegger, J. Schoenboeck, W. Schwinger, and M. Wimmer. Systematic co-evolution of OCL expressions. In APCCM, pages 33–42, 2015.
23. T. Levendovszky, D. Balasubramanian, A. Narayanan, and G. Karsai. A novel approach to semi-automated evolution of DSML model transformation. In SLE, pages 23–41, 2010.
24. W. Lohmann and G. Riedewald. Towards automatical migration of transformation rules after grammar extension. In CSMR, pages 30–39, 2003.
25. 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. Software and System Modeling, 15(3):647–684, 2016.
26. D. Mendez, A. Etien, A. Muller, and R. Casallas. Towards Transformation Migration After Metamodel Evolution. In Models and Evolution Workshop, 2010.
27. B. Meyers and H. Vangheluwe. A framework for evolution of modelling languages. Sci. Comput. Program., 76(12):1223–1246, 2011.
28. L. Rachmawati and D. Srinivasan. Multiobjective evolutionary algorithm with controllable focus on the knees of the pareto front. IEEE Transactions on Evolutionary Computation, 13(4):810–824, 2009.
29. D. D. Ruscio, J. Etzlstorfer, L. Iovino, A. Pierantonio, and W. Schwinger. Supporting variability exploration and resolution during model migration. In ECMFA, pages 231–246, 2016.
30. G. M. K. Selim, J. R. Cordy, and J. Dingel. How is ATL really used? language feature use in the ATL zoo. In MODELS, pages 34–44, 2017.
31. S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. IEEE Software, 20(5):42–45, 2003.