Automated Metamodel/Model Co-Evolution using a Multi-Objective Optimization Approach

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

[1] DIRO, University of Montreal, Canada
[2] Business Informatics Group, Vienna University of Technology, Austria

Abstract. We propose a generic automated approach for the metamodel/model co-evolution. The proposed technique refines an initial model to make it as conform as possible to the new metamodel version by finding the best compromise between three objectives, namely minimizing 1) the non-conformities with new metamodel version, (2) the changes to existing models, and (3) the loss of information. Consequently, we view the co-evolution as a multiobjective optimization problem, and solve it using the NSGA-II algorithm. We successfully validated our approach on the evolution of the well-known UML state machine metamodel. The results confirm the effectiveness of our approach with average precision and recall respectively higher than 87% and 89%.

Keywords: Metamodel/model co-evolution, Model migration, Coupled evolution, NSGA-II

1 Introduction

Models are considered as first-class artifacts in the Model-Driven Engineering (MDE) paradigm. Available techniques, approaches, and tools for MDE support a huge variety of activities, such as model creation, model transformation, and code generation. However, there is little support available for model evolution. Like other software artifacts, metamodels are subject to many changes during the evolution of software modeling languages and language maintenance projects, especially for domain-specific languages [13]. Thus, models have to be updated for preserving their conformance with the new metamodel version.

Recently, several approaches emerged with the aim of tackling the metamodel/-model co-evolution (e.g., [10, 13, 14, 24]). Most of the automated co-evolution approaches focus on the detection of differences between the metamodel versions. Then, they find a set of generic rules to transform the initial models into revised ones, which conform to the new metamodel [12]. Nevertheless, the following challenges remain. The migration rules have to be defined manually for the change types which are detectable at the metamodel level, and they are difficult to generalize for all potential changes of metamodels. The definition of these rules may require a high level of expertise/knowledge from the designer regarding both the previous and new versions of the metamodel. In addition, the detection of changes at the metamodel level is complex due to the graph isomorphism problem where different transformation possibilities are

equivalent. Finally, existing approaches produce exactly one solution for a metamodel/model co-evolution scenario, while other solutions might be possible and could be more suitable in a particular context. Due to these challenges, developers are, sometimes, reluctant to migrate to a new metamodel version, considering the high manual effort required to adapt existing models.

To address these challenges, we propose to tackle the co-evolution of models without the need of computing differences between the metamodel versions. In particular, we view the metamodel/model co-evolution as an automated multi-objective optimization process that searches for a good combination of edit operations, at the model level, by minimizing (i) the number of constraints the revised model violates with respect to the new metamodel version, (ii) the number of changes applied on the initial model to produce the revised model, and (iii) the deviation (dissimilarity) between the initial and revised models. These three objectives are the heuristics that allow us to approximate the evolution of models without an explicit knowledge on the differences between the two metamodel versions and the rules to apply to migrate the models. The first objective ensures that the modified model conforms to the new metamodel. As changes in the metamodel are generally limited to a small subset of its elements, the second objective is used to reflect this property at the model level. Finally, the third objective allows us to limit the loss of information when migrating a model.

To implement our multi-objective approach, we adapt the NSGA-II [5] algorithm to search for solutions that offer the best tradeoff between the three aforementioned objectives. We evaluate our approach on the evolution of the state machine metamodel. In addition to study the proposed transformations to the considered state-machine models, we compare our approach to a random search algorithm, a mono-objective algorithm, and an existing approach in which the migration rules are manually defined after finding the changes between the initial and revised metamodels. The obtained results provide evidence that our proposal is, in average, efficient with more than 92% of manual precision achieved for the studied metamodel evolution.

The remainder of this paper is structured as follows. Section 2 provides the background of model co-evolution and presents a motivating example. In Section 3, we detail our approach. Section 4 discusses an empirical evaluation of our approach. After surveying the related work in Section 5, a conclusion is provided in Section 6.

2 Background and Motivating Example

This section introduces the necessary background for this paper, namely the basic notions of metamodels and models, including their *conformsTo* relationship, as well as a motivating example to demonstrate the challenges related to the metamodel/model co-evolution problem.

2.1 Metamodels and Models

In MDE, metamodels are the means to specify the abstract syntax of modeling languages. For defining metamodels, there are meta-modeling standards (such as MOF, Ecore) available which are mostly based on a core subset of the UML class diagrams,

i.e., classes, attributes, and references. Theoretically speaking, metamodels give the intentional description of all possible models of a given language. In practice, metamodels are instantiated to produce models which are, in essence, object graphs, i.e., consisting of objects (instances of classes) representing the modeling elements, object slots for storing values (instances of attributes), and links between objects (instances of references). The object graphs are often represented as UML object diagrams and have to conform to the UML class diagram describing the metamodel. This means, for a model to conform to its metamodel, a set of constraints have to be fulfilled. This set of constraints is normally referred to as *conformsTo* relationship [11, 21].

To make the *conformsTo* relationship more concrete, we give an excerpt of the constraints concerning objects in models and their relationship to classes in metamodels. Objects are instantiated from classes. Thus, for each referred type of an object in a given model, a corresponding class must exist in the metamodel (name equivalence), and the corresponding class must not be abstract. Such constraints may be formulated in the following way using OCL:

```
context M!Object
  inv typeExists: MM!Class.allInstances() ->
    exists(c|c.name = self.type and not c.abstract)
```

An example metamodel and a corresponding model are shown in Figure 2a and Figure 1a, respectively. This simple language allows to define state machines consisting of states having a name as well as predecessor/successor states.

2.2 Metamodel/Model Co-Evolution: A Motivating Example

While some metamodels, such as UML, are standardized and changed rarely, metamodels for Domain-Specific Modeling Languages (DSMLs) , representing concepts within a certain domain, are frequently subject to change [13].

As most of the current metamodeling frameworks are strict in the sense that only fully conform models can be used, metamodel evolution requires to co-evolve already

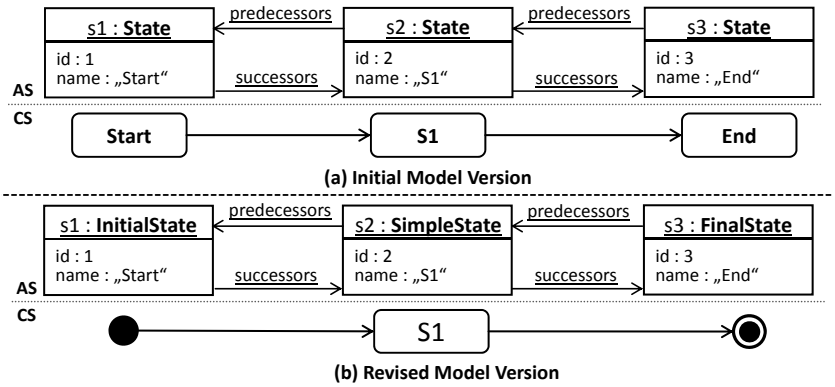

Fig. 1. Example Model Evolution

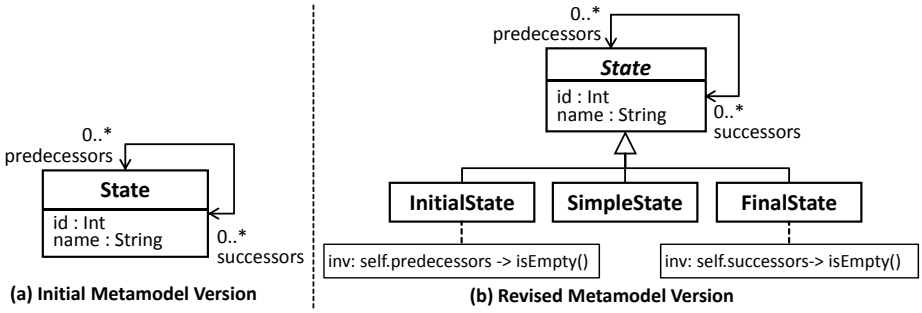

Fig. 2. A Simplified Metamodel Evolution Example

existing models, which may no longer conform to the new metamodel version. In such cases, model migration scripts have to be provided in current tools [10] to re-establish the conformance between models and their metamodels. However, finding the best migration scripts to co-evolve models is left to the user of such tools or default migration scripts are provided. The exploration of the actual co-evolution space is considered as an open challenge.

Figure 2 shows an example of a simplified metamodel evolution, based on the simple state machine language. The metamodel evolution comprises three steps: extract sub-classes for *State* class resulting in *InitialState*, *SimpleState*, and *FinalState*, make class *State* abstract, and refine the cardinalities of the predecessor/successor references for the subclasses. This results in the fact that, besides other constraints violations, the constraint shown previously is violated when considering the initial model shown in Figure 1a and its conformance to the new metamodel version in Figure 2b.

To re-establish conformance for the given example, assume for now that only two operations on models are used in this context. Non-conforming objects may either be retyped (reclassified as instances of the concrete classes) or deleted. Thus, the potential solution space for retyping or deleting non-conforming elements contains $(c+1)^O$ solutions (with c = number of candidate classes + 1 for deletion, o = number of non-conforming objects). This means, in our given example, we would end up with 64 possible co-evolutions while one (probably the preferred one) of these is shown in Figure 1b. This one seems the preferred one due to the following reasons: (i) number of changes introduced, (ii) amount of information loss, and (iii) number of violated conformance constraints. In fact, designers may prefer solutions that introduce the minimum number of changes to the initial model while maximizing the conformance with the target metamodel. When applying changes to the initial model, some model elements could be deleted leading to a better conformance with the new metamodel version but it will reduce the design consistency with the initial model. Thus, these three preferences of the designers are conflicting.

To this end, we consider the metamodel/model co-evolution problem as a multi-objective one which corresponds to finding the best sequence of edit operations which provides a balance between the consistency of the new model with the previous version of the model as well as with the new metamodel version.

3 Model Co-evolution: a Multi-Objective Problem

We describe in this section our proposal and, in particular, how we can formulate the model co-evolution as a multi-objective optimization problem.

3.1 Overview

The goal of our approach is to generate a new version of an existing model, which conforms to a new version of its metamodel. We view this derivation as a search in the space of all possible sequences of edit operations on the initial model. The search is guided by three objectives, which aims at minimizing (1) the number of non conformities with the new version of the metamodel, (2) the number of the changes to the initial model and (3) the consistency between the initial model and the revised one.

In other words, the revised model has to be similar, as much as possible, to the initial model while conforming to the new metamodel version. Therefore, we implemented our idea in the form of a multi-objective optimization algorithm that derives an optimal sequence of edit operations finding the best trade-off between the three objectives. More concretely, our algorithm takes as inputs the initial and revised versions of the metamodel, a model to update and a list of possible edit operations that can be applied to this model. It generates as output a sequence of edit operations that should be applied to the initial model to migrate it to the new metamodel version. The space of all possible sequences of operations can be large, especially when dealing with large models. An exhaustive search method cannot be applied within a reasonable timeframe. To cope with the size of the search space, we use a heuristic search with a multi-objective evolutionary algorithm, NSGA-II [5].

3.2 Adapting NSGA-II for Model Co-Evolution

Multi-Objective Optimization and NSGA-II. To better understand our contribution, we present some definitions related to multi-objective optimization.

Definition 1 (MOP). A multi-objective optimization problem (MOP) consists in minimizing or maximizing an objective function vector $f(x) = [f_1(x), f_2(x), ..., f_M(x)]$ of M objectives under some constraints. The set of feasible solutions, *i.e.,* those that satisfy the problem constraints, defines the search space Ω. The resolution of a MOP consists in approximating the whole Pareto front.

Definition 2 (Pareto optimality). In the case of a minimization problem, a solution $x^* \in \Omega$ is Pareto optimal if $\forall x \in \Omega$ and $\forall m \in I = \{1, ..., M\}$ either $f_m(x) = f_m(x^*)$ or there is at least one $m \in I$ such that $f_m(x) > f_m(x^*)$. In other words, x^* is Pareto optimal if no feasible solution exists, which would improve some objective without causing a simultaneous worsening in at least another one.

Definition 3 (Pareto dominance). A solution u is said to dominate another solution v (denoted by $f(u) \preceq f(v)$) if and only if f(u) is partially less than f(v), *i.e.,* $\forall m \in \{1, ...M\}$ we have $f_m(u) \leq f_m(v)$ and $\exists m \in \{1, ..., M\}$ where $f_m(u) < f_m(v)$.

Definition 4 (Pareto optimal set). For a MOP f(x), the Pareto optimal set is $P^* = \{x \in \Omega | \neg \exists x' \in \Omega, f(x') \preceq f(x)\}$.

Definition 5 (Pareto optimal front). For a given MOP f(x) and its Pareto optimal set P* the Pareto front is $PF^* = \{f(x), x \in P^*\}$.

NSGA-II [5] is one of the most-used multi-objective evolutionary algorithms (EAs) in tackling real-world problems, It begins by generating an offspring population from a parent one by means of variation operators (crossover and mutation) such that both populations have the same size. After that, it ranks the merged population (parents and children) into several non-dominance layers, called fronts, as depicted by Figure 3. Non-dominated solutions are assigned a rank of 1 and constitute the first layer (Pareto

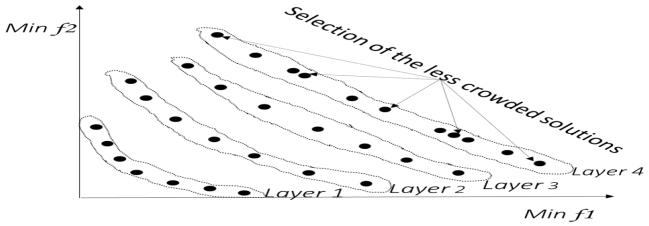

Fig. 3. NSGA-II Selection Mechanism for a two-Objective Problem.

front). After removing solutions of the first layer, the non-dominated solutions form the second layer and so on and so forth until no non-dominated solutions remain. After assigning solutions to fronts, each solution is assigned a diversity score, called crowding distance [5], inside each front. This distance defines a partial ranking inside the front which aims, later, at favoring diverse solutions in terms of objective values. A solution is then characterized by its front and its crowding distance inside the front.

To finish an iteration of the evolution, NSGA-II performs the environmental selection to form the parent population for the next generation by picking half of the solutions. The solutions are included iteratively from the Pareto front to the lowest layers. If half of the population is reached inside a front than the crowding distance is used to discriminate between the solutions. Figure 3 shows an example of the selection process for two objectives. The solutions of the three first layers are included but not all those of 4th one. Some solutions of the 4th layer are selected based on their crowding distance values. In this way, most crowded solutions are the least likely to be selected; thereby emphasizing population diversification. To sum up, the Pareto ranking encourages convergence towards the near-optimal solution while the crowding ranking emphasizes diversity.

Problem formulation. The model co-evolution problem involves searching for the best sequence of edit operations to apply among the set of possible ones. A good solution s is a sequence of edit operations to apply to an initial model with the objectives of minimizing the number of non-conformities $f_1(s) = nvc(s)$ with the new metamodel version, the number of changes $f_2(s) = nbOp(s)$ applied to the initial model, and the

inconsistency $f_3(s) = dis(s)$ between the initial and the evolved models such as the loss of information.

The first fitness function $nvc(s)$ counts the number of violated constraints w.r.t. the evolved metamodel after applying a sequence s of edit operations. We consider three types of constraints, as described in [16]: related to model objects, i.e., model element (denoted by O.*), related to objects' values (V.*), and related to objects' links (L.*). We use in our experiments the implementation of these constraints proposed by Schoenboeck et al. [21]. These constraints are:

O.1 For an object type, a corresponding class must exist (name equivalence).
O.2 Corresponding class must not be abstract.
V.3 For all values of an object, a corresponding attribute in the corresponding class (or in its superclasses) must exist (name equivalence).
V.4 For all (inherited) attributes in a class, a corresponding object must fulfil minimal cardinality of values.
V.5 For all (inherited) attributes in a class, a corresponding object must fulfil maximum cardinality of values.
V.6 For all values of an object, the value's type must conform to the corresponding attribute's type (Integer, String, Boolean).
L.7 For all links of an object, a corresponding reference in its corresponding class (or in its superclasses) must exist (name equivalence).
L.8 For all (inherited) references in a class, a corresponding object must fulfil minimal cardinality of links.
L.9 For all (inherited) references in a class, a corresponding object must fulfil maximum cardinality of links.
L.10 For all links of an object, the target object's type must be the class defined by the reference (or its subclasses).

For the second fitness function, which aims at minimizing the changes to the initial models, we simply count the number of edit operations $nbOp(s)$ of a solution s (size of s). The third fitness function $dis(s)$ measures the difference between the model elements in the initial and revised model. As the type of a model element may change because of a change in the metamodel, we cannot rely on elements' types. Alternatively, we use the identifiers to assess whether information was added or deleted when editing a model. Let Id_i and Id_r be the sets of identifiers present respectively in the initial (i) and revised (r) models. The inconsistency between the models is measured as the complement of the similarity measure $sim(s)$ which is the proportion of common elements in the two models. Formally:

$$dis(s) = 1 - sim(s) \text{ and } sim(s) = \frac{|Id_i \cap\ Id_r|}{Max(|Id_i|, |Id_r|)}$$

NSGA-II Application. To adapt NSGA-II to our problem, we define (1) how to represent a co-evolution solution, (2) how to derive new solutions from existing ones, and (3) how to evaluate a solution.

Solution representation. To represent a candidate solution (individual), we use a vector containing all the edit operations to apply to the initial model. Each element in

the vector represents a single operation (with links to the model elements to which it is applied) and the order of operations in this vector represents the sequence in which the operations are applied. Consequently, vectors representing different solutions may have different sizes, i.e., number of operations. Table 1 shows the possible edit operations that can be applied to model elements. These operations are inspired by the catalog of operators for the metamodel/model co-evolution in [9].

Operations	Description
Create/delete	Add/remove an element in the initial model.
Retype	Replace an element by another equivalent element having a different type.
Merge	Merge several model elements of the same type into a single element.
Split	Split a model element into several elements of the same type.
Move	Move an attribute from a model element to another.

Table 1. Model Edit Operations

Figure 4 depicts an example of a solution that can be applied to revise the initial model of Figure 1. The solution consists of 10 different edit operations to apply: *Retype(State s1, InitialState s1), Retype(State s2, SimpleState s2) , DeleteElement(SimpleState s2.name),Retype(State s3, FinalState s3), CreateElement(SimpleState s2, SimpleState s2, successors), CreateElement(FinalState s1), MoveElement(FinalState s1, SimpleState s2.id), CreateElement(FinalState s3, FinalState s1, successors), CreateElement(InitialState s3), DeleteElement(InitialState s3).* The proposed algorithm first

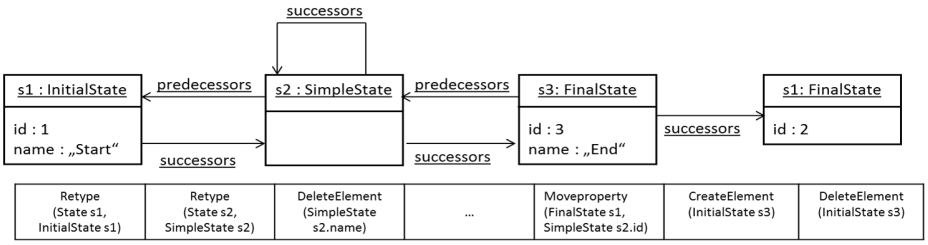

Fig. 4. Example of a solution representation

generates a population of random operation sequences (solution candidates), which are used in the subsequent iterations to produce new solutions.

Solution derivation. In a search algorithm, the variation operators play a key role of moving within the search space with the aim of driving the search towards better solutions. In each iteration, we select $population\_size/2$ individuals from the population pop_i to form population pop_{i+1}. These $(population\_size/2)$ selected individuals will produce other $(population\_size/2)$ new individuals using a crossover and

mutation operators. To select parents for reproduction, we used the principle of the roulette wheel [5]. According to this principle, the probability to select an individual for crossover and mutation is directly proportional to its relative fitness in the population. We use a one-point crossover operator. For our problem, this operator split each parent operation sequence $S1$ (resp. $S2$) into two subsequences $\{S1_1, S1_2\}$ (resp. $\{S2_1, S2_2\}$) according to a cut position k. Then, it combines the subsequences to create to sibling solutions $\{S1_1, S2_2\}$ and $\{S2_1, S1_2\}$. Our crossover operator could create a child sequence that contains conflict operations. In this case, it will be penalized by the component nvc of the fitness function. The mutation operator consists in randomly selecting one or two operations in a solution vector and modifying them. Two modifications are used: (1) swapping the two selected operations in the sequence or (2) replacing an operation by a randomly created one. When applying both change operators, we are using a repair operator to detect and fix possible redundancies between the model elements. When a redundancy is detected, we remove one of redundant model elements from the solution (vector).

Solution evaluation. As mentioned in the problem formulation, a solution is evaluated according to three objectives. Thus, for each solution s, we calculate $nvc(s)$, $nbOp(s)$, and $disIm(s)$. These values are used later to establish the dominance relation between solutions. Based on the example of the solution of Figure 4, the value of the fitness functions are the following: $f_1(S) = 1$ (a final state should not have a successor), $f_2(S) = 10$ (10 types of operations) and $f_3(S) = 1 - (9/Max(15, 13))$ (Since the intersection between the ids of the initial model of Figure 1 :(State.s1 , s1.id , s1.name, s1.successors, State.s2 , s2.id , s2.name, s2.successors, s2.predecessors, State.s3 , s3.id , s3.name, s3.predecessors) and the ids of the revised one of Figure 4 : (InitialState.s1 , **s1.id** , **s1.name**, **s1.successors**, InitialState.s2 , **s2.successors**, **s2.predecessors**, FinalState.s3 , **s3.id** , **s3.name**, **s3.predecessors**, s3.successors, FinalState s1, **s2.id**) is 9 (bold elements in the revised model), the size of the initial model of Figure 1 is 13 and the size of the revised one of Figure 4 is 15).

4 Validation

In order to validate our meta-model/model co-evolution approach, we conducted a set of experiments based on a well-known evolution case of UML Metamodel for State Machine from v1.4 to v2.0 [1][3].

4.1 Research Questions

The validation study was conducted to quantitatively and qualitatively assess the completeness and correctness of our co-evolution approach when applied to realistic settings and to compare its performance with an existing deterministic approach [25]. More specifically, we aimed at answering the following research questions:

- **RQ1: Sanity check:** *To what extend the obtained results are attributable to our search strategy and to the large number of solutions that we explore?*

[3] The data of the experiments can be found in https://sites.google.com/site/datapaperswaelkessentini/data

Models	SM1	SM2	SM3	SM4	SM5	SM6	SM7	SM8	SM9	SM10
Number of expected edit operations	8	11	12	11	12	9	20	14	10	21
Number of model elements	34	38	72	64	46	37	82	56	47	94

Table 2. The Models Studied

- **RQ2: Correctness:** *To what extent can the proposed multi-objective approach co-evolve models to make them comply with a new metamodel version (in terms of correctness and completeness of proposed edit operations)?*
- **RQ3:** *How does the multi-objective metamodel/model co-evolution approach perform compared to a mono-objective one?* We use a genetic algorithm with a fitness function that represents the average of the normalized scores of the three objectives.
- **RQ4:** *How does our approach perform compared to an existing co-evolution approach [25] not based on metaheuristic search?*

4.2 Experimental Setting

Studied Meta-models and Models To answer the four research questions, we considered the evolution of UML State Machine Metamodel from version 1.4 to 2.0 [1]. This is a very interesting case since it represents a real metamodel evolution and was studied in other contributions [25]. Therefore, the two versions were manually analyzed to determine the actually applied changes. We also used 10 models from version 1.4 and evolved them manually to version 2.0 to create new models from the initial ones using the edit operations.

As described in Table 2. The manually defined sequences for the selected models are used as baseline sequences for the calculation of precision and recall scores. Table 2 shows for each model, the number of expected edit operations and the size in terms of elements.

Evaluation Metrics To compare our approach with the other alternatives, we use precision and recall measures. For an operation sequence corresponding to a given solution, precision indicates the proportion of correct edit operations (w.r.t. the baseline sequence) in a solution. Recall is the proportion of correctly identified edit operations among the set of all expected operations. Both values range from 0 to 1, with higher values indicating good solutions. The baseline sequences do not represent unique evolution solutions for the used models. Indeed, more than one alternative can be possible to evolve a given model. Thus, in addition to automatic precision (AC-PR) and recall (AC-RE), we calculated a manual correctness (MC). To this end, we manually checked the solutions to determine the accuracy of their operations.

Based on the example of the solution of Figure 4 the value of the results are the following: AC-PR=0.3(the set of correctly edit operations which is 3, divided by the set of all actually applied operations which is 10), AC-RE=1(the set of correctly edit operations which is 3, divided by the set of all expected operations which is 3), MC=0.37(when

we check the solution operation by operation we found that the sequence of the 2 operations: CreateElement(InitialState s3), DeleteElement(InitialState s3)can be considered correct since the InitialState s3 was created and then deleted). In addition to the three above-mentioned metrics, we report the number of operations (NO) per solution and the computation time (CT). Finally, it is worth noting that the mono-objective GA and the deterministic [25] approaches produce a unique co-evolution solution, while NSGA-II generates a set of non-dominated solutions (Pareto front). In order to have meaningful comparisons, we select the best solution for NSGA-II using a knee point strategy [5].

Statistical Tests Since the used metaheuristic algorithms (NSGA-II and GA) are stochastic by nature, different executions may produce different results for the same model with the same execution parameters. For this reason, our experimental study is performed based on 30 independent simulation runs and the obtained results by the alternative approaches are compared using the Wilcoxon rank sum test [2] with a 95% confidence level ($\alpha = 5\%$).

4.3 Results

Results for RQ1. We do not dwell long in answering the first research question (RQ1) that involves comparing our approach based on NSGA-II with random search. Figures 5 and 6 confirm that using NSGA-II (as well as the GA and the deterministic algorithm) produce results by far better (and statistically significant) than just randomly exploring a comparable number of solutions. NSGA-II has precisions (AC-PR and MC) and recall (AC-RE) more than twice higher than the ones of random search as shown in Figure 5 ($\sim 89\%$ vs $\sim 43\%$). Moreover, these results were obtained with smaller operation sequences in the 10 models (Figure 6) comparing to random search. The slight difference in execution time in favor of random search (Figure 7), due to the crossover and mutation operators, is largely compensated by quality of the obtained results.

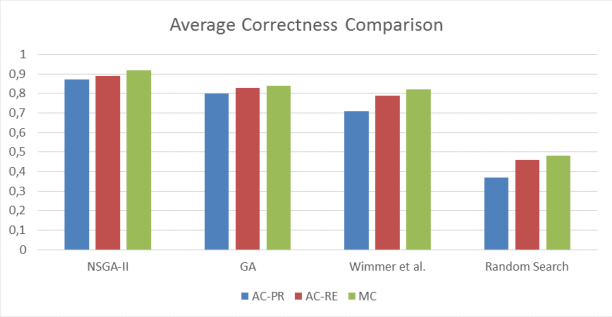

Fig. 5. Average correctness results of NSGA-II, GA, Wimmer et al., Random Search on the 10 models.

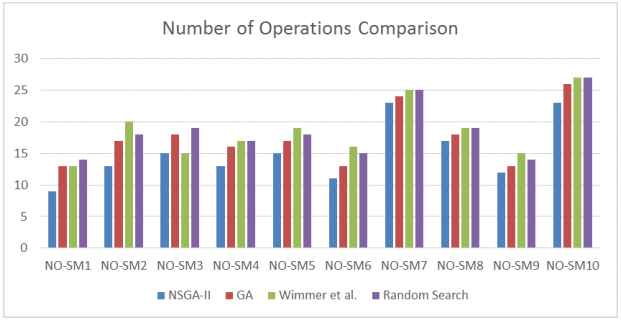

Fig. 6. Average number of suggested operations of NSGA-II, GA, Wimmer et al., Random Search on the 10 models.

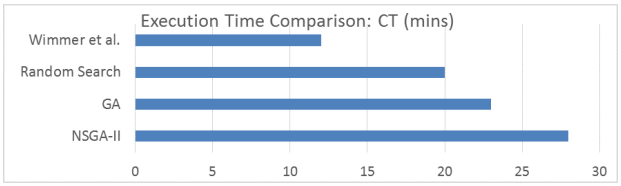

Fig. 7. Average execution time of NSGA-II, GA, Wimmer et al., Random Search on the 10 models.

To formally answer **RQ1**, we state that *there is an empirical evidence that the quality of the co-evolution results obtained are due to our multi-objective approach and not to the number of explored solutions*.

Results for RQ2. We evaluated the averages of NO, AC-PR, AC-RE and MC scores for non-dominated co-evolution solutions proposed by NSGA-II. Figure 6 shows that our NSGA-II adaptation was successful in generating good co-evolution solutions that minimize the number of operations. The number of suggested operations seems very reasonable if we consider the high number of changes applied at the metamodel of State Machine during the evolution from v1.4 to v2.0. In fact, 23 edit operations were recommended for the largest model (SM10) with an average of 16 operations for the remaining models.

For the precision and recall, Figure 8 shows that the produced solutions using NSGA-II are similar to the baseline ones with more than 85% of precision and recall (AC-PR and AC-RE) in general. For four models (SM1, SM3, SM5, and SM8), we obtained more than 90% for the recall. From another perspective, we did not observe a correlation between the size or the number of operations of the models and the precision and recall. For example, we obtained higher precision and recall for SM7 (82 elements and 20 operations) than for SM2 (38 elements and 11 operations). This means that the correctness of the results is not degraded as the size of the models or the size of necessary modifications increase. For the manual correctness, the results are even better. Except for SM2, SM3 and SM9 all the MCs are higher than 90% with a perfect score for SM8. Here again, the scalability in terms of correctness is valid for MC. Indeed MC

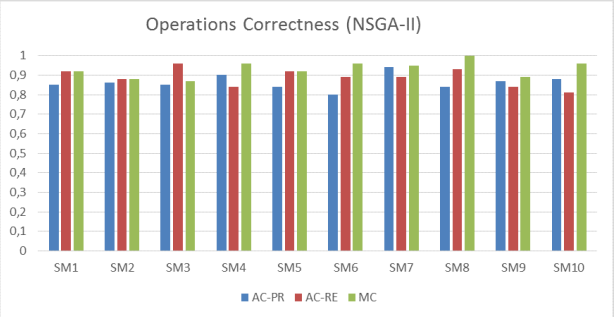

Fig. 8. Correctness results of NSGA-II on the 10 State Machine models.

increases from 88% (SM2) to 95% (SM10) while the size of the model (resp. operation sequence) goes from 38 to 94 (resp. 11 to 21).

To formally answer **RQ2**, we state that *the multi-objective co-evolution approach allows to migrate models with higher precision and recall and with a limited number of edit operations. This achieved without any explicit knowledge on the specific changes that occurred on the metamodel.*

Results for RQ3 and RQ4. For the recall and the automatic and manual precisions, Figure 5 show that the solutions of NSGA-II have the highest scores compared to the other algorithms. The same observation holds for the manual precision, for which NSGA-II is the only algorithm that has an average score higher than 91%. These differences in favor of NSGA-II are all statistically significant according to the Wilcoxon tests. Additionally, Figure 6 shows that, for all the models, NSGA-II produces higher scores with smaller operation sequences compared to those of the two contender algorithms.

Regarding the execution time (Figure 7), considering that the model evolution happens rarely (not a daily activity), the time magnitude (minutes) for all the algorithms is reasonable. Our NSGA-II approach takes more time than [25] (28 vs 12 minutes). We believe that this difference is acceptable knowing that for [25], we do not count the time of metamodel comparison and rules writing. The slight difference with the mono-objective algorithm is also acceptable if we consider the improvement in accuracy.

To formally answer **RQ3** and **RQ4**, we state that *there is an empirical evidence that the multi-objective algorithm outperforms the mono-objective and deterministic algorithms for AC-PR, AC-RE and MC. This must be mitigated by the fact that these good results come at the cost of more execution time*.

4.4 Threats to Validity

We used the Wilcoxon rank sum to test if significant differences exist between the scores of the co-evolution algorithms. This test makes no assumption on the data distribution, especially for small samples. We are then confident that our observations are valid. Another possible threat is related to the parameter tuning of our multi-objective algorithm.

Further experiments are required to evaluate the impact of the parameters setting on the quality of the solutions.

To ensure that the results are attributable to our multi-objective algorithm and not to chance, we performed 30 independent simulation runs for each model. This makes it unlikely that the observed differences are due to the probabilistic decisions of the algorithms. Another threat is related to our choice of taking the average of the three objective functions in the mono-objective algorithms. Other forms of combination, e.g., weighted average, may give different results.

With respect to generalizability of our findings, we performed our experiments on a single evolution scenario (state machine metamodel from v1.4 to v2.0). Future replications of this study are necessary to confirm these findings, in particular, with industrial settings. In addition, the comparison of the performance of NSGA-II to the state of the art is limited to the approach in [25]. The decision was made as the concerned tool was easily accessible to us.

5 Related Work

Co-evolution has been subject for research since several decades in the database community [17], and especially, the introduction of object-oriented database systems [3] gave rise to the investigation of this topic. However, metamodel/model co-evolution introduces several additional challenges mostly based on the rich modeling constructs for defining metamodels, and consequently, it has to be dealt with the specific *conformsTo* relationship between models and metamodels. Thus, in the last decade, several approaches emerged which aim to tackle metamodel/model co-evolution from different angles using different techniques (cf. e.g., [13, 19, 20] for an overview).

In one of the early works [22], the co-evolution of models is tackled by designing co-evolution transformations based on metamodel change types. In [4, 7], the authors compute differences between two metamodel versions which are then input to adapt models automatically. This is achieved by transforming the differences into a migration transformation with a so-called higher-order transformation (HOT). Going one step further concerning the nature of change types is presented in [23], where ideas from object-oriented refactoring and grammar adaptation are presented to provide the basis for metamodel/model co-evolution. In [8], a conservative copying algorithm is presented: for each initial model element for which no transformation rule is found, a default copy transformation rule is applied. This algorithm has been developed in a model migration framework Epsilon Flock [18] and in the framework described in [15]. In order to avoid copy rules at all, co-evolution approaches which base their solution on in-place transformations (i.e., transformations which are updating an input model to produce the output model) have been proposed [10, 12–14, 25]. In such approaches, the co-evolution rules are specified as in-place transformation rules by using a kind of unified metamodel representing both metamodel versions.

Although the main goal of all discussed approaches is similar to ours, there are several major differences. We tackle co-evolution of models without the need of computing differences on the metamodel level. Instead, we reason on the consistency of the models by following a similar research line as presented in the visionary paper by Demuth

et al. [6]. In particular, we search for transformation executions on the model level, which fulfil multiple goals expressed as our fitness functions. By this, the critical and challenging task of finding proper co-evolution rules of the model level is automated which allows exploring a much larger space of possible solutions which is possible when manually developing co-evolution transformations. To sum up, none of the existing approaches allows the exploration of different possible co-evolution strategies. On the contrary, only one specific strategy is either automatically derived or manually developed from the calculated set of metamodel changes.

The only work we are aware of discussing metamodel/model co-evolution using some search-based techniques is [24]. In this paper, they authors discuss the idea of using search-based algorithms to reason about possible model changes, but in contrast to our approach, they rely again on metamodel differences which have to be computed (probably using a search-based approach) before the co-evolution of models can be performed. We use a different approach by using an explicit definition and formalization of the *conformsTo* relationship which can be used as basis to formulate fitness functions for reasoning about the quality of a certain model co-evolution strategy. To the best of our knowledge, this approach is unique compared to previous approaches and outperforms logic-based approaches for repairing models [21]. Furthermore, we are not dependent on the quality of metamodel change detection algorithms. By this, we allow the automatic exploration of the model co-evolution space given a certain metamodel evolution and we developed formal quality characteristics to assess the quality of model co-evolutions.

6 Conclusion

This paper proposes a multi-objective approach for the co-evolution of models by finding the best operation sequence that generates, from the initial model, a new model version conforming as much as possible to the evolved metamodel. Therefore, a generated revised model should minimize the number of inconsistencies (with the new metamodel), the number of changes made to the initial model and the dissimilarity with the initial model. The experiment results indicate clearly that the best generated models have a precision and recall of more than 86% and a manual precision of more than 92%.

Although our approach has been evaluated with a real-world scenario with a reasonable number of applied operations and models, we are working now on larger metamodels and models with larger lists of operations to apply. This is necessary to investigate more deeply the applicability of the approach in practice, but also to study the performance of our approach when dealing with very large models. Moreover, we will further evaluate the performance of NSGA-II with several other multi-objective algorithms. More generally, we plan to extend this work by evolving model transformation rules when the metamodels were revised.

References

1. `http://www.omg.org`, Object Management Group, Unified Modeling Language Specification v1.4 and v2.0

2. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proc. of ICSE (2011)
3. Banerjee, J., Kim, W., Kim, H.J., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: Proc. of SIGMOD (1987)
4. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Proc. of EDOC (2008)
5. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In: Proc. of PPSN (2000)
6. Demuth, A., Lopez-Herrejon, R.E., Egyed, A.: Co-evolution of Metamodels and Models through Consistent Change Propagation. In: Proc. of ME Workshop (2013)
7. Garcés, K., Jouault, F., Cointe, P., Bézivin, J., Emninria, A.: Managing model adaptation by precise detection of metamodel changes. In: Proc. of ECMFA (2009)
8. Gruschko, B.: Towards synchronizing models with evolving metamodels. In: Proc. of MoDSE Workshop (2007)
9. Herrmannsdoerfer, M., Vermolen, S.D., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Proc. of SLE (2011)
10. Herrmannsdörfer, M.: COPE A Workbench for the Coupled Evolution of Metamodels and Models. In: Proc. of SLE (2011)
11. Iovino, L., Pierantonio, A., Malavolta, I.: On the Impact Significance of Metamodel Evolution in MDE. Journal of Object Technology 11(3) (2012)
12. Mantz, F., Lamo, Y., Taentzer, G.: Co-Transformation of Type and Instance Graphs Supporting Merging of Types with Retyping. ECEASST 61 (2013)
13. Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Sci. Comput. Program. 76(12) (2011)
14. Meyers, B., Wimmer, M., Cicchetti, A., Sprinkle, J.: A generic in-place transformation-based approach to structured model co-evolution. In: Proc. of MPM Workshop (2010)
15. Narayanan, A., Levendovszky, T., Balasubramanian, D., Karsai, G.: Automatic domain model migration to manage metamodel evolution. In: Proc. of MODELS (2009)
16. Richters, M.: A precise approach to validating UML models and OCL constraints. Tech. rep. (2001)
17. Roddick, J.F.: Schema evolution in database systems: An annotated bibliography. SIGMOD Rec. 21(4) (1992)
18. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Model migration with Epsilon Flock. In: Proc. of ICMT (2010)
19. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C.: An Analysis of Approaches to Model Migration. In: Proc. of MoDSE-MCCM Workshop (2009)
20. Rose, L., Herrmannsdoerfer, M., Mazanek, S., Van Gorp, P., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schätz, B., Wimmer, M.: Graph and model transformation tools for model migration. SoSyM 13(1) (2014)
21. Schoenboeck, J., Kusel, A., Etzlstorfer, J., Kapsammer, E., Schwinger, W., Wimmer, M., Wischenbart, M.: CARE: A Constraint-based Approach for Re-establishing Conformance-relationships. In: Proc. of APCCM (2014)
22. Sprinkle, J., Karsai, G.: A domain-specific visual language for domain model evolution. J. Vis. Lang. Comput. 15(3-4) (2004)
23. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: Proc. of ECOOP (2007)
24. Williams, J.R., Paige, R.F., Polack, F.A.C.: Searching for Model Migration Strategies. In: Proc. of ME Workshop (2012)
25. Wimmer, M., Kusel, A., Schoenboeck, J., Retschitzegger, W., Schwinger, W.: On using inplace transformations for model co-evolution. In: Proc. of MtATL Workshop (2010)