Automated Metamodel/Model Co-Evolution: A Search-Based Approach

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

[1]*DIRO, Université de Montréal, Canada*

[2]*CDL-MINT, TU Wien, Austria*

Abstract

Context: Metamodels evolve over time to accommodate new features, improve existing designs, and fix errors identified in previous releases. One of the obstacles that may limit the adaptation of new metamodels by developers is the extensive manual changes that have be applied to migrate existing models. Recent studies addressed the problem of automating the metamodel/model co-evolution based on manually defined migration rules. The definition of these rules requires the list of changes at the metamodel level which are difficult to fully identify. Furthermore, different possible alternatives may be available to translate a metamodel change to a model change. Thus, it is hard to generalize these co-evolution rules.

Objective: We propose an alternative automated approach for the metamodel/-model co-evolution. The proposed approach refines an initial model instantiated from the previous metamodel version to make it as conformant as possible to the new metamodel version by finding the best compromise between three objectives, namely minimizing (i) the non-conformities with new metamodel version, (ii) the changes to existing models, and (iii) the textual and structural dissimilarities between the initial and revised models.

Method: We formulated the metamodel/model co-evolution as a multi-objective optimization problem to handle the different conflicting objectives using the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and the Multi-Objective Particle Swarm Optimization (MOPSO).

Results: We evaluated our approach on several evolution scenarios extracted from different widely used metamodels. The results confirm the effectiveness of our approach with average manual correctness, precision and recall respectively higher than 91%, 88% and 89% on the different co-evolution scenarios.

Conclusion: A comparison with our previous works, confirms the out-performance of our multi-objective formulation.

Keywords: Metamodel/model co-evolution, Model migration, Coupled evolution, Search Based Software Engineering

1. Introduction

In Model-Driven Engineering (MDE) [1], several techniques were proposed for modeling language engineering, e.g., metamodeling, as well as the management and the manipulation of models, *e.g.*, transformation, synchronization, and merging [2]. These techniques increased the popularity and applicability of MDE in practice. Thus, the evolution rate of metamodels and models increased dramatically [3] to improve the performance, fix errors, integrate new features, etc. However, little support was provided to reduce the evolution effort for modelers, which may limit the benefits, adaptability and popularity of new metamodel releases.

Many research contributions targeted the definition and implementation of sets of operators for the manual evolution of metamodels and models [4, 5, 6, 7]. However, when the changes are performed on a metamodel, one should ensure that the related models are updated for preserving their conformance with the new metamodel version. Thus, recent approaches emerged with the aim of specifically tackling the metamodel/model co-evolution [8]. 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 conforming to the new metamodel [9].

Despite the important advances brought by these approaches, several challenges are still to be addressed. Indeed, 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 regarding both the previous and new versions of the metamodel. In addition, the detection of precise changes at the metamodel level is complex [10]. For example, when one attribute is removed and another is added, this can be interpreted as a rename, or a deletion and addition of independent attributes. The combinatory of the possibilities increases dramatically when multiple changes are performed at the same time. Finally, existing approaches produce exactly one solution for a metamodel/model co-evolution scenario, while other solutions might be possible and may be more suitable in a particular context. Due to these challenges, modelers are, sometimes, reluctant to migrate their models to a new metamodel version, considering the high effort required to adapt these models.

To address these challenges, we proposed, in a previous work, an approach that tackles the co-evolution of models without the need of computing differences between metamodel versions [11, 12]. In particular, we view the metamodel/model co-evolution as a 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 structural and semantics deviation (dissimilarities) 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 and to prevent from deviating a lot from the initial design. Finally, the third objective allows us to limit the loss of information when migrating a model.

In this paper, we extended our previous work [11, 12] from different perspectives. First, we revised our multi-objective formulation by integrating textual similarity measures [13] when comparing the initial and revised models. Indeed, in our previous work, we were limited to only calculating structural similarities between the elements. In fact, some revised model structures could be different in the revised model but still reflect the same features/context. The textual similarity allows to identify such situations, which motivated us to integrate it into the third fitness function together with the structural similarity. This new formulation of the objectives is compared to the initial one in the empirical evaluation. We additionally implemented our heuristic search using a second multi-objective algorithm, namely MOPSO [14]. This extension allowed us to evaluate the impact (result correctness and performance) of using one (NSGA-II [15]) or the other (MOPSO [14]) algorithm with the same setting. Another important extension consisted in adding new and more complex metamodels and scenarios in our empirical validation. Finally, we updated and extended the related work as discussed in the initial publication of our approach.

To evaluate our metamodel/model co-evolution approach, we conducted a set of experiments based on four different case-studies of evolved metamodels. We chose to analyze the extensive evolution of three Ecore metamodels from the Graphical Modeling Framework (GMF)[1], an open-source project for generating graphical modeling editors, and a well-known evolution case of UML Metamodel for State Machines [16]. The obtained results provide compelling evidence that our proposal is, in average, efficient with more than 90% of valid edit operations, precision and recall achieved for the studied metamodel evolutions. To calculate these evaluation metrics, we used a set of manually defined reference sequences of edit operations (expected results) and compared them to the recommended ones by our approach. However, the reference sequences (manual ones) are not required as an input of our approach and only used to calculate the precision and recall. The results also supports the claim that our NSGA-II formulation provides a good trade-off between the three objectives, and outperforms on average our previous work [11] and a non-search based approach [17].

The remainder of this paper is structured as follows. Section 2 provides the background of metamodel/model co-evolution and presents a motivating example which is subsequently used as a running 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 with an outlook on future work 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. Further-

[1] `https://www.eclipse.org/modeling/gmp`

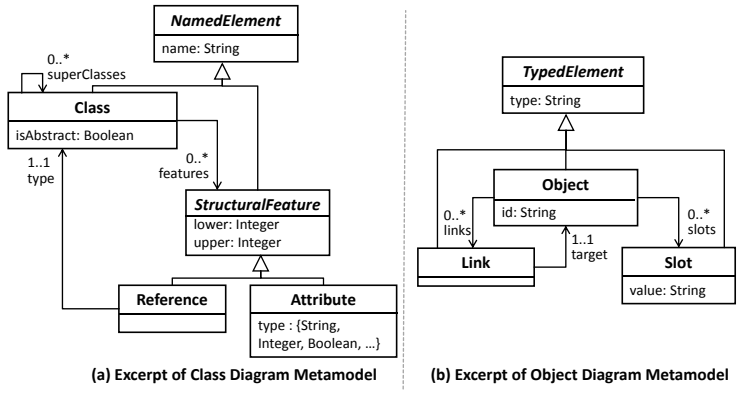

Figure 1: Metamodel excerpts of UML class diagrams and UML object diagrams.

more, we provide 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 [1]. For defining metamodels, there are metamodeling 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. They have to conform to the respective UML class diagrams. 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 [3, 18].

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. Figure 1 illustrates the used elements of both languages. 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, for instance, using the Object Constraint Language (OCL) as shown in Listing 1.

Listing 1: Type/Object Relationship formalized as OCL Constraint

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

The make the introduced concepts more concrete, we present a motivating example in the following. In particular, we use a simple language that allows to define different staff members based on their project involvement. Example model versions and corresponding metamodel versions are shown in Figure 3 and Figure 2, respectively.

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 [7].

As most of the current metamodeling frameworks are strict in the sense that only fully conformant models can be used, metamodel evolution requires to co-evolve, i.e., migrate, already existing models, which may no longer conform to the new metamodel version. In such cases, model migration scripts [4] have to be developed in current tools 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 simple staff modeling language taken from [19]. The metamodel evolution comprises three steps: extract sub-classes for *Person* class resulting in *ProjectStaff*, *InternalStaff*, and *ExternalContact*, make class *Person* abstract, refine the types of the *assignedTo* and *contact* references, as well as restrict the existence of the *salary* attribute only for *Staff* instances. This evolution results in the fact that, besides other constraints violations,

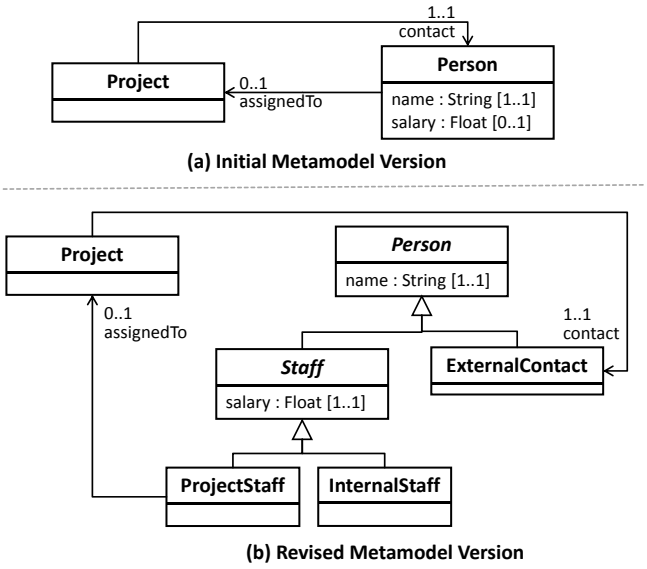

Figure 2: Example Metamodel Evolution

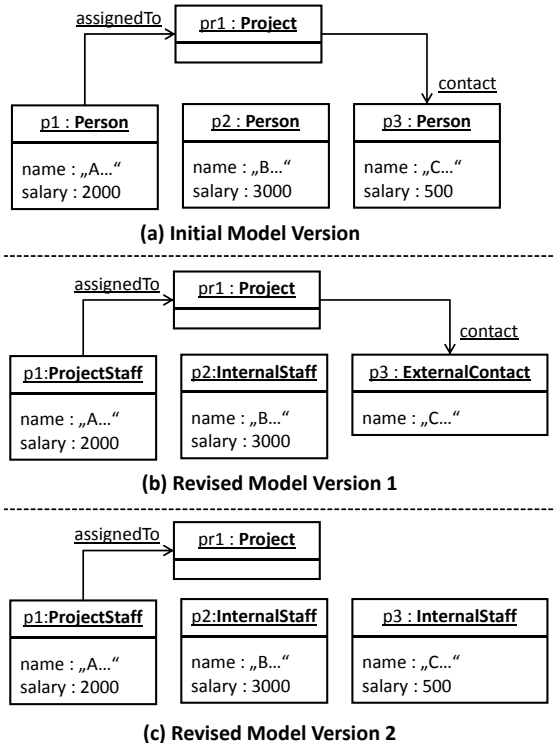

Figure 3: Example Model Evolution

the constraint shown in Listing 1 is violated when considering the initial model shown in Figure 3a 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 edit operation types 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 two, probably two of the preferred ones, are shown in Figure 3b-c.

These two co-evolutions seem of interest due to the following reasons: (i) number of changes introduced (three retype operations are used), (ii) amount of information loss (mostly no information loss as the same bag of labels are provided by both model versions having the same object structure), and (iii) number of violated conformance constraints (no constraint is violated). 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.

Going back to the concrete example, we can see that the *Person* objects have to be retyped because the *Person* class is now no longer a concrete class, thus no direct instances are allowed. It is clear that we do not want to delete objects as long as they can be casted into subtypes of *Person*. The combination of casts which is resulting in the shown solutions is optimal due to the fact that no shorter sequence of changes is possible as we have to deal with the existing *Person* objects in some way. Furthermore, these combinations allow to keep most the information except the salary for *Person* p3 in the revised model version 1 and the contact link in the revised model version 2. Please note that also other solutions are equally good, e.g., a solution which would type *Person* p3 as *ProjectStaff*. Thus, this example highlights that several solutions may be possible and users may select the most preferred one by trading different conflicting objectives.

To this end, we consider the metamodel/model co-evolution problem as a multi-objective one which corresponds to finding the set of best sequences of edit operations. This 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. Metamodel/Model Co-evolution: a Multi-Objective Problem

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

3.1. Overview

The proposed approach takes as input the initial and revised metamodel versions, a set of initial models to migrate and a list of possible types of model edit operations. It has to be noted that our approach, in contrast to existing approaches, does not require as input the list of changes between the two metamodel versions. The goal of our approach is to generate, for each of the considered models, a minimally adapted new version that conforms to the new version of the metamodel as much as possible. Thus, for each input model, the search space is the set of possible models that can be generated by applying on the input model combinations of the edit operations. The exploration of this search space is guided by three objectives, which aims at minimizing (i) the number of non-conformities with the new version of the metamodel, (ii) the number of changes applied to the initial model, and (iii) the consistency between the initial model and the revised one using structural and textual measures. Based on these three objectives, the revised model has to be similar, as much as possible, to the initial model while conforming to the new metamodel version. A summary of our approach is graphically depicted in Figure 4.

It is possible to reduce the search space, by considering the meta-model changes, which may help NSGAII to provide good results within a low execution time. Our current adaptation could be easily adjusted to consider as input the metamodel changes (if available) as a set of constraints to satisfy when generating the population of solutions (sequence of edit operations) automatically. We are considering, in this paper, mainly the cases where the metamodel changes are not available or time-consuming

and difficult to translate into generic co-evolution rules. However, it is possible to give these co-evolution rules, if available and even incomplete, as an input to our approach to further reduce the search space.

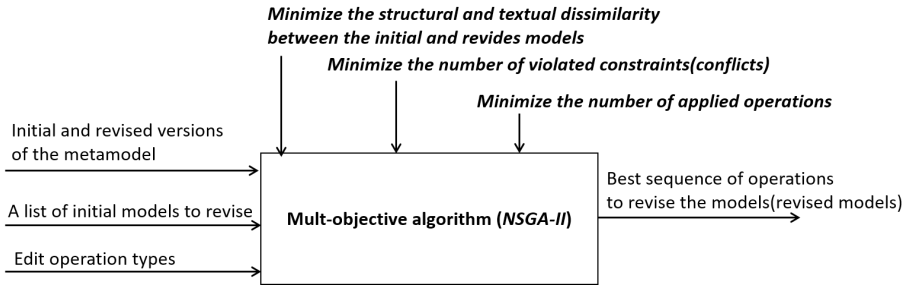

Figure 4: Metamodel/Model Co-Evolution Framework

To handle these three objectives together, we addressed the metamodel/model co-evolution problem using a multi-objective optimization algorithm that derives the edit operations sequence with the best trade-off. The space of all possible sequences of operations can be too 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 meta-heuristic search. More specifically, we adapt two multi-objective algorithms to our problem, namely NSGA-II [15] and MOPSO [14]. It is impossible to predict the performance of a metaheuristic algorithm. We compared NSGA-II with MOPSO because they are both global search algorithms (unlike tabu search and simulated annealing) to ensure a fair comparison based on the same number of iterations and so on. It is expected that a global search (population-based) can generate better solutions than a local search (limited mainly to the mutation operator) when the same number of iterations are used. The next section will give an overview about these algorithms.

3.2. Multi-objective Algorithms

We now characterize multi-objective optimization before we explain the two optimization algorithms used in this paper.

3.2.1. Multi-objective Optimization

To better understand our contribution, we present some background 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

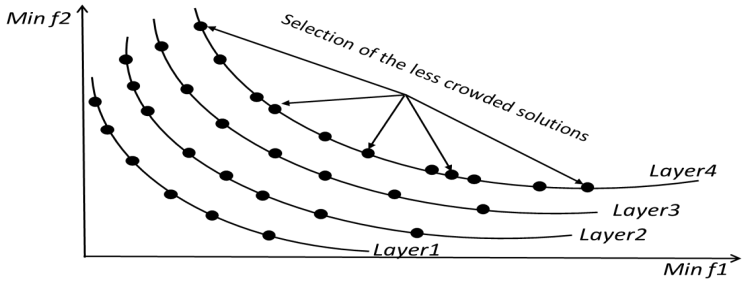

Figure 5: NSGA-II Selection Mechanism for a two-Objective Problem.

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 $PF^* = \{f(x), x \in P^*\}$.

3.2.2. NSGA-II

NSGA-II is among the widely-used algorithms to address real-world problems involving conflicting objectives [15]. For example, a recent survey on search-based software engineering shows that NSGA-II was successfully applied to problems from different domains such as the next release problem [20], software product lines [21], refactoring [22], etc. It has also been applied to solve other co-evolution problems in recent work [23].

NSGA-II begins with a generation of an offspring population from an initial set of parent individuals using two change operators of crossover and mutation. Both populations of the offspring and parent have the same size (lines 1-2 of Algorithm 1). Then, the merged population (parents and children) is ranked into several non-dominance layers, called fronts.

As described in Figure 5, the non-dominated solutions receive the rank of 1 that represents the first layer, called the Pareto 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 (lines 5-6 of Algorithm 1). After assigning solutions to fronts, each solution is assigned a diversity score, called crowding distance [6], which ranks the solutions inside each front (lines 7-12 of Algorithm 1). This distance 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 (lines 13-16

Algorithm 1 Pseudocode for NSGA-II

```
Create an initial population P_0
Create an offspring population Q_0
t = 0
while stopping criteria not reached do
    R_t = P_t ∪ Q_t
    F = fast-non-dominated-sort(R_t)
    P_{t+1} = ∅ and i = 1
    while | P_{t+1} | + | F_i | ⩽ N do
        Apply crowding-distance-assignment(F_i)
        P_{t+1} = P_{t+1} ∪ F_i
        i = i + 1
    end while
    Sort(F_i, ≺ n)
    P_{t+1} = P_{t+1} ∪ F_i[N− | P_{t+1} |]
    Q_{t+1} = create-new-pop(P_{t+1})
    t = t+1
end while
```

of Algorithm 1). 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. In the example of Figure 5, the solutions of the three first layers are included but not all those of the 4th one. Some solutions of the 4th layer are selected based on their crowding distance values. The remaining solutions and those of the subsequent layers (not displayed in the figure) are not considered for the next iteration. In this way, most crowded solutions are the least likely to be selected; thereby emphasizing population diversification. The Pareto ranking encourages convergence towards the near-optimal solution while the crowding ranking emphasizes diversity.

3.2.3. MOPSO

MOPSO is also a well-known multi-objective optimization algorithm, which generalizes the mono-objective particle swarm optimization algorithm. MOPSO generates, first, a population of random solutions, called particles. Each particle in MOPSO flies in the search space with a specific velocity value. The velocity is dynamically updated by the particle own flying experience and those of the other solutions. Each individual is treated as a point in the n-dimensional search space. A particle t has then a position $X_t(X_{t1}, .,X_{tn})$. Its velocity is denoted by $V_t(V_{t1}, .,V_{tn})$. In the single objective PSO, the goal is to find a global best solution. However, when tackling multi-objective problems, the goal is to find a set of solutions that constitute the Pareto Front. So a repository of non-dominated solutions is kept, where all non-dominated solutions found at each iteration are stored. The latter is used by the particles to identify a leader that will guide the search.

At each iteration, the position and velocity of each particle are updated according to two best values, *pbest* and *gbest*. *pbest* is the individual best solution that the particle has achieved so far. As we are dealing with many objectives, *pbest* is randomly selected

from the set of particle best non-dominated solution (local Pareto set). *gbest* is one of the global best solutions chosen randomly from the repository of non-dominated solutions obtained in the swarm until the current iteration (global Pareto set). The size of this repository is restricted to a predefined number. If the number of non-dominated solutions is higher that this number, then, the crowding distance is used to eliminate the exceeding solutions. The position and velocity updating equations are as follows.
Velocity update:

$$v_i(t+1) = wv_i(t) + r_1c_1(P_l^i(t) - x_i) + r_2c_2(P_g(t) - x_i) \quad (1)$$

Position update:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

Where w is the inertia weight, x_i the position of a particle i, v_i the velocity of i, c_1 and c_2 are the cognitive and social parameters of the MOPSO, r_1 and r_2 are random numbers between 0 and 1, P_l^i is the local best position that the particle i has had, which is selected from the local Pareto set and P_g is the global leader of particle i that is taken from the global Pareto set, i.e., the repository for our problem.

Algorithm 2 summarizes the different steps of the MOPSO algorithm. We start by generating a population of particles, where v_i=0 for each particle (lines 1-4) and by initializing the memory of each particle (lines 5-7). After that, we evaluate each of the particles in the population (line 10). Then, we store in the memory of each particle PBESTS the best locale position achieved so far and we store the positions of the particles that represent nondominated vectors in the repository of global solutions (lines 11-12). after that we compute the velocity of each particle (Equation 1) and we update the position (Equation 2) (lines 13-14). Once we have the new positions, we evaluate each of the particles in the population and we update the repository. This update consists of inserting all the currently nondominated locations into the repository. Any dominated locations from the repository are eliminated in the process. Since the size of the repository is limited, whenever it gets full, we apply a secondary criterion for retention: those particles located in less populated areas of objective space are given priority over those lying in highly populated regions (line 16). When the current position of the particle is better than the position contained in its memory, the particle's position is updated. The criterion to decide what position from memory should be retained is simply to apply Pareto dominance (i.e., if the current position is dominated by the position in memory, then the position in memory is kept; otherwise, the current position replaces the one in memory; if neither of them is dominated by the other, then we select one of them randomly). The algorithms stops when reaching a stopping criteria.

To allow the comparison between the two algorithms when addressing the meta-model/model co-evolution problem, we use, for both NSGA-II and MOPSO, the problem formulation, including the objectives. We used these two algorithms because they were widely adapted to similar problems in software engineering such as the next release problem [20], refactoring [24], test cases generation [25], etc. We describe in the following section the problem formulation.

Algorithm 2 Pseudocode for MOPSO

```
for i=0 to Max do                          ▷ Initialize the population pop. Max=number of particles
    Initialize pop[i]
    v[i]=0
end for
for i=0 to Max do                          ▷ Initialize the memory of each particle
    PBEST[i]=pop[i]
end for
while termination condition not reached do  ▷ Store the positions of particles that represent nondominated solutions in the repository REP.
    for each particle in the swarm do
        Evaluate the particle
        if the current position of the particle is better than the position contained in it's memory then
            Current position is set as the new PBEST
        end if
        update the velocity of the particle according to equation (1)
        update the position of the particle according to equation (2)
    end for
    Update the contents of repository
end while
```

3.3. Problem Formulation

In this subsection, we describe the adaption of the two generic multi-objective algorithms introduce before to our problem. This adaptation consists in defining (i) how to represent a co-evolution solution, (ii) how to derive new solutions from existing ones to explore the search space, and (iii) how to evaluate a solution.

Solution representation. To represent a candidate solution (i.e., an individual) in NSGA-II, 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]. The catalog included both metamodel and model changes. Thus, we selected from the catalog all the edit operations that can be applied to the model level since we are not changing, in this paper, the metamodels and only models are revised. The metamodel/-model elements that can be modified are described in Table 1. The instances of classes are called objects, instances of features are called slots, and instances of references are called links.

For MOPSO, the solution is also presented as a set of operations. But the operations are represented as a set of IDs where each type of operation has an Integer ID. We use this representation to be consistent with the notion of position in MOPSO, which define

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

Table 1: Model Edit Operations

the coordinates in the search space (cf. Figure 6). This representation in MOPSO helps the derivation of the solution using the velocity.

Figure 6 represent a solution that can be applied to our motivating example in Section 2

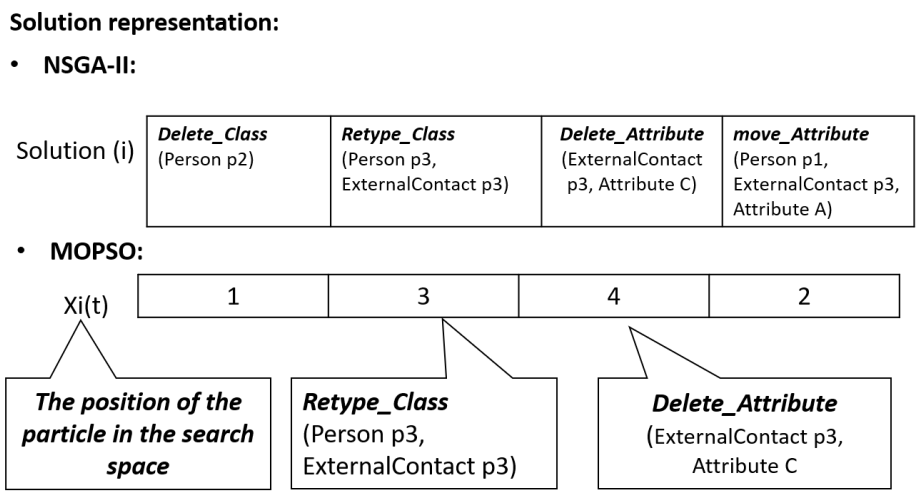

Figure 6: Solution representation

The two algorithms first generate a population of random operation sequences (solution candidates), which are used in the subsequent iterations to produce new solutions.

Solution evaluation. The investigated co-evolution problem involves searching for the best sequence of edit operations to apply among the set of possible ones. The two algorithms have the same solution evaluation. 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 apply, first, the sequence of edit operations (solution) on the initial model then we load the evolved model on the target metamodel to measure the number of conformance errors based on the number of violated constraints. We consider three types of constraints, as described in [26]: 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 inspired by Schoenboeck et al. [18] and Richters et al. [26] with slight adaptations. The constraints are hard-coded in the implementation of the algorithm and most of them are from the EMF conformance verification constraints that already exists in EMF. Thus, we use the following constraints:

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).

The sequence of edit operations to fix the non-conformities are dependent to each others thus it is not possible to treat the different issues in isolation. In fact, the edit operations used to fix one violation may impact other violations and create new ones. Thus, we have to treat all the violations together when generating the set of edit operations as a possible solution.

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. In this case, the renamed or extracted model elements will be considered different than the initial model element. Thus, we considered the assumption that two model elements could be syntactically similar if they use a similar vocabulary. Thus, we calculated for the textual similarity based on the Cosine similarity [13]. In the first step, we tokenize the names of initial and revised model elements. The textual and/or context similarity between elements grouped together to create a new class is an important factor to evaluate the cohesion of the revised model. The initial and revised models are represented as vectors of terms in n-dimensional space where n is the number of terms in all considered models. For each model, a weight is assigned to each dimension (representing a specific term) of the representative vector that corresponds to the term frequency score (TF) in the model. The similarity among initial and revised model elements is measured by the cosine of the angle between its representative vectors as a normalized projection of one vector over the other. The cosine measure between a pair of model elements, A and B, is defined as follows:

$$Sim(A, B) = cos(\theta) = \frac{A * B}{\vec{A} * \vec{B}}$$

Let Id_i and Id_r be the sets of identifiers present respectively in the initial (M_i) and revised (M_r) models. The inconsistency between the models is measured as the complement of the similarity measure $sim(s)$ which is the proportion of similar elements in the two models based on the cosine similarity. Formally the third fitness function is defined as:

$$Dis(s) = 1 - (CosineSimilarity(id_i, id_r)/Max(|M_i|, |M_r|))$$

where $CosineSimilarity(id_i, id_r)$ is defined as follows:

$CosineSimilarity(id_i, id_r)$ = $\sum_{j=1}^{|M_i|} MaxSimilarity(Id_j, (id_r)_{k=1}^{|M_r|})$

This function will compare between each of the initial model elements and all the elements of the revised model to find the best matching.

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.

For NSGA-II, 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 [15]. 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 two sibling solutions $\{S1_1, S2_2\}$ and $\{S2_1, S1_2\}$. Our crossover operator

could create a child sequence that contains conflicting 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: (i) swapping the two selected operations in the sequence or (ii) 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).

In MOPSO the deviation of a solution is applied using the velocity. We calculate the velocity of every dimension in the vector. Then we update the position of the particle using Equations 1 and 2 to obtain a derived solution. For example, in Figure 7 ,the operation move-Attribute became delete-Attribute after updating the position using the velocity equation.

Most of the edit operations are dependent to each others thus the order is important. However, some others are not dependent. We did not consider the detection of dependencies between the edit operations in our current work, when applying the variation operators, since it depends not only on the type of the operations but also where they are applied which make the process very complex.

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 $dis(s)$. These values are used later to establish the dominance relation between solutions.

4. Validation

To validate our metamodel/model co-evolution approach, we conducted a set of experiments based on four different case studies of evolved metamodels[2]. We chose to analyze the extensive evolution of three Ecore metamodels from the Graphical Modeling Framework (GMF), an open source project for generating graphical modeling editors, and a well-known evolution case of UML Metamodel for State Machine from v1.4 to v2.0 [11]. These evolution cases are selected because of the extensive revisions introduced to the GMF metamodels during a period of two years and to be able to compare with existing co-evolution studies using the same data sets. The UML State Machine evolution case may show the generality of our approach for different types of metamodels.

In our case study, we created two evolution scenarios per metamodel covering a period of two years. The obtained results are subsequently statistically analyzed, on 30 runs, with the aim to compare our proposal with another multi-objective algorithm, namely Multi-Objective Partical Swarm Optimazation (MOPSO), single population-based approach based on a genetic algorithm (GA) aggregating the three objectives and random search (RS). In addition, we compared our approach to an existing co-evolution technique not based on meta-heuristic search proposed by Wimmer et al. [17]. In this

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

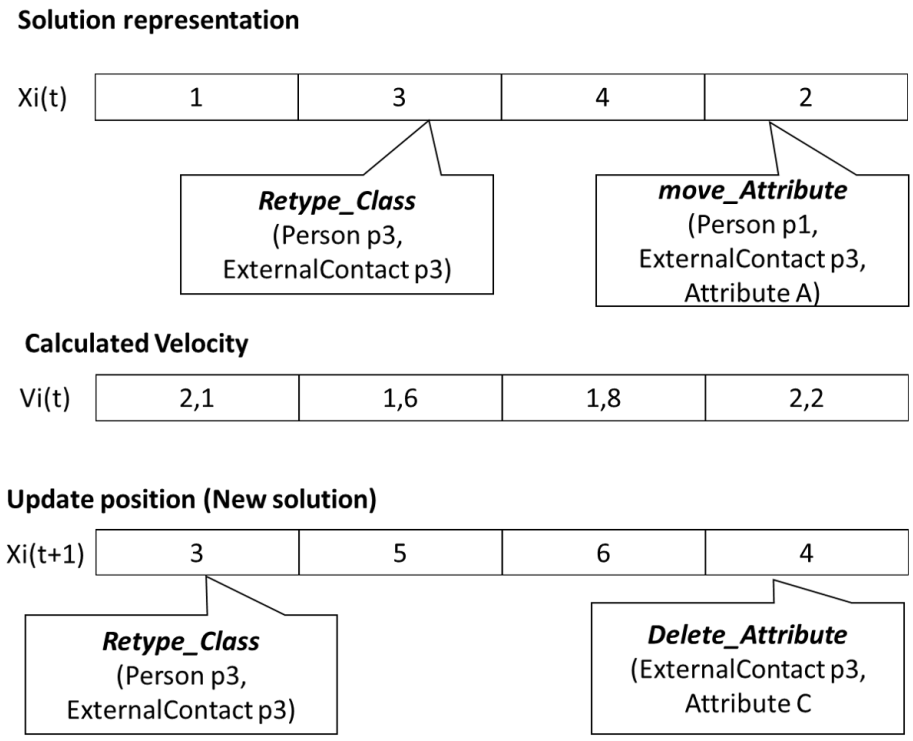

Figure 7: MOPSO adaptation

section, we start by presenting our research questions and the experimental setting. Then, we describe and discuss the obtained results. Finally, we describe some potential validity threats related to our experiments.

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 [17]. More specifically, we aimed at answering the following research questions:

- **RQ1:** Search validation–how does our approach perform compared to RS? To validate the multi-objective formulation of our co-evolution approach, we compared our formulation with RS (using the same fitness functions). If RS outperforms our guided multi-objective search strategy, then we can conclude that our formulation is not adequate for the addressed co-evolution problem. Since just outperforming a RS solution is not sufficient, the next questions are related to evaluation of the performance of NSGA-II, and a comparison with an existing deterministic metamodel/model co-evolution approaches.

- **RQ2:** 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, and the impact of the size of models on the results)? Of course, our approach will be considered useful by developers if it will automate the process of co-evolving models with an acceptable correctness rate.

- **RQ3:** How does our approach perform compared to another multi-objective algorithm (MOPSO) and to our previous work [11] (since we integrated textual similarity measures when comparing between initial and revised models)? The goal is to assess whether NSGA-II is the good choice for the approach since any multi-objective search algorithm may be used for the proposed approach. It is critical to compare with other algorithms since which algorithm to select is always considered as an important conclusion of any search-based software engineering technique. The comparison with our previous work is required to find out whether using textual similarities improve the results or not.

- **RQ4:** How does the multi-objective metamodel/model co-evolution approach perform compared to a mono-objective one? A multi-objective algorithm provides a trade-off between the three objectives where developers can select the desired co-evolution solution from the Pareto-optimal front. A mono-objective approach uses a single fitness function that is formed as an equal aggregation of three objectives and generates as output only one solution of the generated model (e.g., model changes to make it coherent to the new metamodel version). This comparison is required to ensure that the solutions provided by NSGA-II and MOPSO provide a better trade-off between the three objectives than a mono-objective approach. Otherwise, there is no benefit to our multi-objective adaptation since the objectives are not conflicting in that case.

  The mono-objective formulation aggregates the different objectives (with same weight) into one fitness function. Thus, the algorithm will generate only one co-evolution solution that corresponds to the best fitness function value. The multi-objective formulation is based on the non-dominance concept where multiple solutions can be generated (Pareto front) due the conflicting nature of the independent fitness functions. For example, Solution A could contains less number of edit operations than Solution B but the latter may generates less conformance errors with the new metamodel. Thus, both solutions will be considered as non-dominated ones and included in the Pareto front. For the case of our experiments, we selected the solution at the knee point of the Pareto front which is the widely used method in the multi-objective optimization literature [15]. This solution corresponds to the solution with the maximal trade-off between the different three objectives.

- **RQ5:** How does our approach perform compared to an existing co-evolution approach[17] not based on metaheuristic search? While it is very interesting to show that our proposal outperforms a mono-objective approach, developers will consider our approach useful, if it can outperform other existing co-evolution tool, proposed by Wimmer et al. [17], not based on optimization techniques.

The approach of Wimmer et al. finds the model that satisfy the constraints of a merged initial and revised metamodel and then eliminating elements from the initial model based on the differences between the initial and target metamodels.

4.2. Experimental Setting

4.2.1. Studied Metamodels and Models

To answer the different research questions, we considered the evolution of GMF covering a period of two years and the UML State Machine Metamodel from version 1.4 to 2.0. These case studies are very interesting scenarios since they represent real metamodel evolutions, used in an empirical study [27] as well as studied in other contributions [12, 28, 29, 30]. For GMF, we chose to analyze the extensive evolution of three Ecore metamodels, an open source project for generating graphical modeling editors. We considered the evolution from GMF's release 1.0 over 2.0 to release 2.1 covering a period of two years. For achieving a broad data basis, we analyzed the revisions of three metamodels, namely the Graphical Definition Metamodel (GMF Graph for short), the Generator Metamodel (GMF Gen for short), and the Mappings Metamodel (GMF Map for short). Therefore, the respective metamodel versions had to be extracted from GMF's version control system and, subsequently, manually analyzed.

From the different metamodel releases of GMF and UML, we created different scenarios based on the number of changes that were introduced at the metamodels level. We merged the releases that did not include extensive changes and we generated two evolution scenarios per metamodel type. Table 4.2.1 describes the minimum and maximum number of model elements over the different versions and the list of models used for the co-evolution scenarios. For GMF We created 3 models per scenario ranging between 34 and 566 model elements. The different models and metamodels can be classified as small-sized through medium-sized to large-sized. In our experiments, we have a total of 7 different co-evolution scenarios where each scenario included three different models to evolve. The percentage of changes between the different releases is estimated based on the number of modified metamodel elements divided by the size of the metamodel. The created models for our experiments are ensuring the metamodels coverage. Furthermore, we used an existing set of 10 models for the case of UML State Machine Metamodel evolution from the work of Wimmer et al. [17] thus we were not involved in the selection of models and metamodel changes.

In order to ensure a fair comparison with Wimmer et al. [17], we only compared both approaches on the existing UML data set. Furthermore, the implementation of the approach of Wimmer et al. is dependent to the UML State Machine metamodel thus it was not possible to consider the GMF metamodels for the comparison with [17].

The next section will give details about the used evaluation metrics on the different studied metamodels and models.

Metamodel				Models		
Release	#of elements	#of changes	%of changes	#of models	#of model elements (Model 1..n)	#of expected edit operations
GMF Gen 1.41 to 1.90	From 885 to 1120	347	31%	3	382 — 427 — 556	38 — 52 — 64
GMF Gen 1.90 to 1.248	From 1120 to 1216	362	27%	3	402 — 463 — 566	56 — 61 — 72
GMF Map 1.45 to 1.52	From 382 to 413	62	15%	3	182 — 214 — 304	36 — 42 — 53
GMF Map 1.52 to 1.58	From 413 to 428	10	1.8%	3	243 — 278 — 362	47 — 53 — 56
GMF Graph 1.25 to 1.29	From 278 to 279	14	5%	3	142 — 157 — 193	32 — 37 — 48
GMF Graph 1.25 to 1.33	From 279 to 281	42	14%	3	144 — 162 — 211	39 — 42 — 53
UML 1.4 to 2.0	From 56 to 65	9	12%	10	34—44—56—63—72 93—106 —126—156—196	8—11—13—15—10 9—24—12—33—44

Table 2: Statistics related to the collected data.

4.2.2. Evaluation Metrics

The quality of our results was measured by two methods: automatic correctness (AC) and manual correctness (MC). Automatic correctness consists of comparing the proposed edit operations to the reference ones, operation by operation using precision (AC-PR) and recall (AC-RE). 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. AC method has the advantage of being automatic and objective. However, since different edit operation combinations exist that describe the same evolution (different edit operations but same target model), AC could reject a good solution because it yields different edit operations from reference ones. To account for those situations, we also use MC which manually evaluates the proposed edit operations, one by one. In addition to these metrics, we report the number of operations (NO) per solution and the computation time (CT) for the different evolution scenarios.

All these metrics are used for the comparison between our approach and RS, MOPSO, GA, Kessentini et al. [11], and Wimmer et al. [17]. We note that the mono-objective GA algorithm and Wimmer et al. provide only one solution of the proposed edit operations, while NSGA-II and MOPSO generate a set of non-dominated solutions. To make meaningful comparisons, we select the best solution for NSGA-II, MOPSO, and Kessentini et al. [11] using a knee point strategy [15]. The knee point corresponds to the solution with the maximal trade-off between the different three objectives. Thus, we select the knee point using the trade-off "worth" metric [31] that corresponds to the solution having the median hyper volume value. We aim by this strategy to ensure fairness when making comparisons against the mono-objective approach. For the latter, we use the best solution corresponding to the median observation on 30 runs. However, there is no guarantee that the knee point solution is the best but it is widely used to determine the maximum trade-off between the objectives. In many cases, the trade-off between the objectives should be manually explored by the user by selecting the best solution based on given preferences. However, we opted to use the automated knee point strategy to avoid biasing the results with the manual intervention of the user.

4.2.3. Statistical Tests

Since the used metaheuristic algorithms 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 [32] with a 95% confidence level. Roughly speaking, this test verifies the null hypothesis H0 that the observed differences between the alternative results were obtained by chance or if they are statistically significant (alternative hypothesis H1). The p-value of the Wilcoxon test corresponds to the probability of rejecting the null hypothesis H0 while it is true (type I error). A p-value that is less than or equal to 0.05) means that we reject H0 and accept H1. A p-value that is less than or equal to $\alpha(\leq 0.05)$ means that we accept H1 and we reject H0. However, a p-value that is strictly greater than α (> 0.05) means the opposite. In fact, we compute the p-value of GA, MOPSO, mono-objective, and Kessentini et al. [11] search results

with NSGA-II ones. In this way, we could decide whether the superior performance of NSGA-II to one of each of the others (or the opposite) is statistically significant or just a random result.

4.2.4. Parameter Settings

Parameter setting has an influence on the performance of a search algorithm on our models co-evolution problem. Thus, for each algorithm and for each scenario, we executed a set of experiments using several population sizes: 30, 50, 100, 150 and 200. The stopping criterion was set to 1000 evaluations for all algorithms to ensure fairness of the comparative study. The MOPSO used in this paper is the Non-dominated Sorting PSO (NSPSO) proposed by Li et al. [33]. The list of parameter settings are the following for NSGA-II: (1) crossover probability = 0.7; mutation probability = 0.2; stopping criterion = 1000 evaluations. For MOPSO, the cognitive and social scaling parameters c1 and c2 were both set to 3.0 and the inertia weighting coefficient w decreased gradually from 2.0 to 0.6 [14]. The maximum length of the individuals is limited to the size of the evaluated model (number of model elements). We found, based on the used trials and error method, that the number of the proposed edit operations does not exceed, in general, the total number of model elements (equivalent to the number of changes to create a model from scratch). We considered these parameters setting based on successful applications of mutli-objective algorithms in model-driven engineering [34, 35].

4.3. Results

Results for RQ1. Figures 8, 9 and 10 confirm that both multi-objective algorithms NSGA-II and MOPSO are better than random search based on the three different metrics of precision, recall and manual correctness on all the different evolution case studies. The statistical test showed that in 30 runs, NSGA-II, MOPSO, and Kessentini et al. [11] results were significantly better than RS. Figure 8 shows the overview of the results of the precision on the different metamodel versions. NSGA-II and MOPSO outperforms RS in all cases and specially on large models involving a high number of expected edit operations such as GMF Gen. The high number of changes introduced to the metamodel releases makes the search process much more challenging and complicated due to the very large number of possible edit operations combination. We have also found that NSGA-II is better than RS algorithm based on all the remaining metrics of recall and manual correctness as described in Figures 9 and 10 all the case studies. NSGA-II has precision (AC-PR and MC) and recall (AC-RE) more than twice higher than the ones of random search as shown in Figure 8 (85% vs 25%) for GMF Gen. The number of suggested edit operations (NO) is much lower using our approach comparing to random search as showed in Figure 13. However, the execution time of RS was lower than NSGA-II as described in Figure 11. The slight difference in execution time in favor of random search (see Figure 11), due to the crossover and mutation operators, is largely compensated by quality of the obtained results.

RS is not efficient to generate good co-evolution solutions using all the above metrics in all the experiments. Thus, an intelligent algorithm is required to find good trade-offs to propose efficient solutions. We conclude that there is empirical evidence

that our multi-objective formulation surpasses the performance of RS search thus our formulation is adequate and the use of metaheursitic search is justified (this answers RQ1).

Results for RQ2. To answer RQ2, we evaluated the average of PR, RE, NO and MC scores for non-dominated co solutions proposed by NSGA-II. Figures 8, 9 and 10 confirm that our NSGA-II adaptation was successful in recommending relevant co-evolution model changes with a minimum number of edit operations. Based on the different case studies, our approach was able to correctly recommend, at least, 81% of generated edit operations (precision) and a minimum of 83% of correct recommendations among expected ones (recall).

It is clear that the evolution history of the metamodels is different but our results were consistent in terms of precision and recall. The Generator Metamodel (GMF Gen for short) was extensively modified within the two studied releases leading to large number of model changes, but most of them were recommended using our technique with a high recall of 87%. the Graphical Definition Metamodel (GMF Graph for short) was subjected to also more than 19% of changes. Thus, the evolution of these metamodels is a very representative mixture of different scenarios for the co-evolution of different sizes of models leading to a precision of more than 92% and a recall of more than 94% on these two metamodels. The evolution of the third metamodel under consideration, The Mapping Metamodel (GMF Map for short), contained as well several revisions but less than the two other metamodels,the obtained results of precision and recal are more than 89%.

Figures 8 and 9 show that the suggested solutions using NSGA-II are similar to the reference solution with more 89% in average for all the metamodels. However, a deviation with the set of reference solutions does not necessarily mean that there is an error with our multi-objective solutions, but it could be another possible good model evolution solution different from the reference ones. To this end, we evaluated the suggested edit operations by NSGA-II manually and we calculated a manual correctness score. With regards to manual correctness (MC), the precision and recall scores for all the models were improved since we found interesting co-evolution alternatives that deviates from the reference/expected ones. For instance, the precision for the GMF Map model was improved from 88% to 90%. In addition, we found that a sequence of applying the edit operations is sometimes different between generated operations and reference ones as described in Figure 13. We found that sometimes a different sequence can reduce the number of edit operations used while generating another possible model alternative conforming to the new metamodel.

Figure 11 summarizes the average execution time of NSGA-II on the different metamodels. The execution time of NSGA-II is higher than that of MOPSO and Kessentini et al. [11] with the same number of iterations. The execution time required by mono-objective algorithms (25 minutes in average) is lower than both NSGA-II, MOPSO, and Kessentini et al. [11] (33 minutes in average for NSGA-II, 30 minutes for MOPSO, and 27 minutes for Kessentini et al. [11]). It is well known that a mono-objective algorithm requires less execution time for convergence since only one objective is handled. However, the execution times of NSGA-II and MOPSO are not so far from those of mono-objective algorithms. For this reason, we can say that the difference in running time is compensated by the benefits provided by NSGA-II in terms

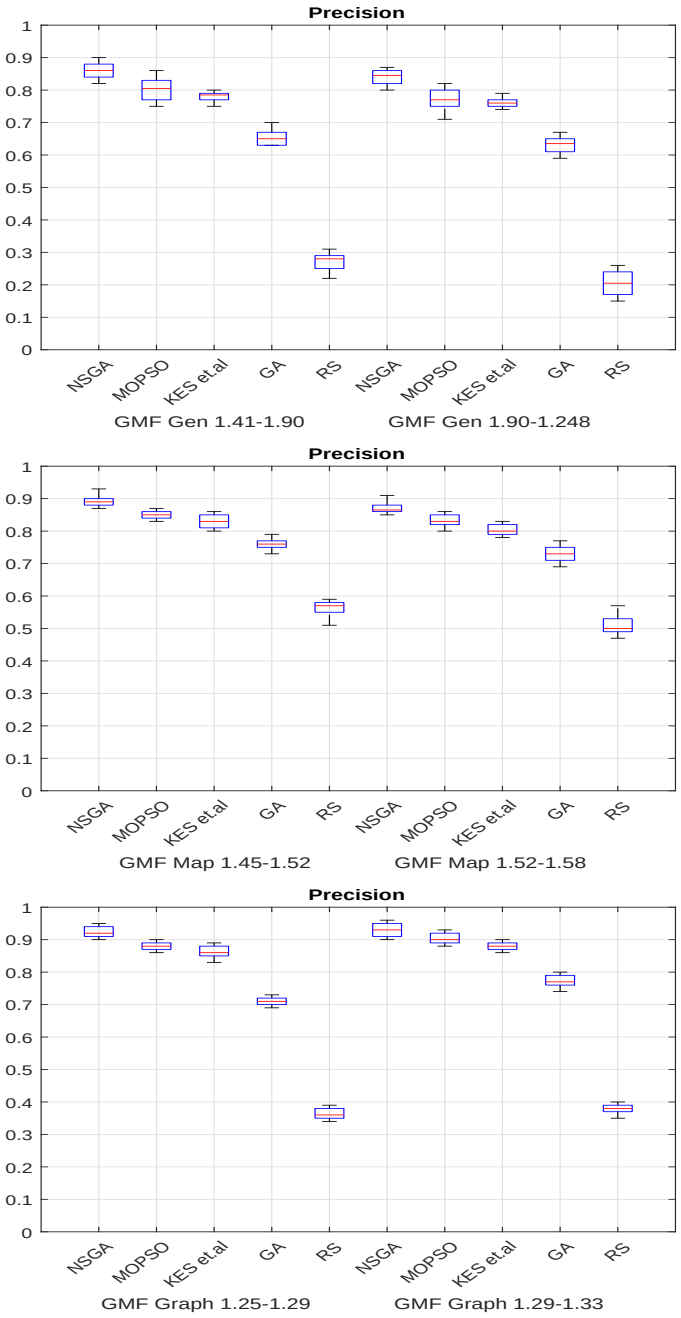

Figure 8: Average precision results of NSGA-II, MOPSO, Kessentini et al. [11], GA, and Random Search on 30 runs.

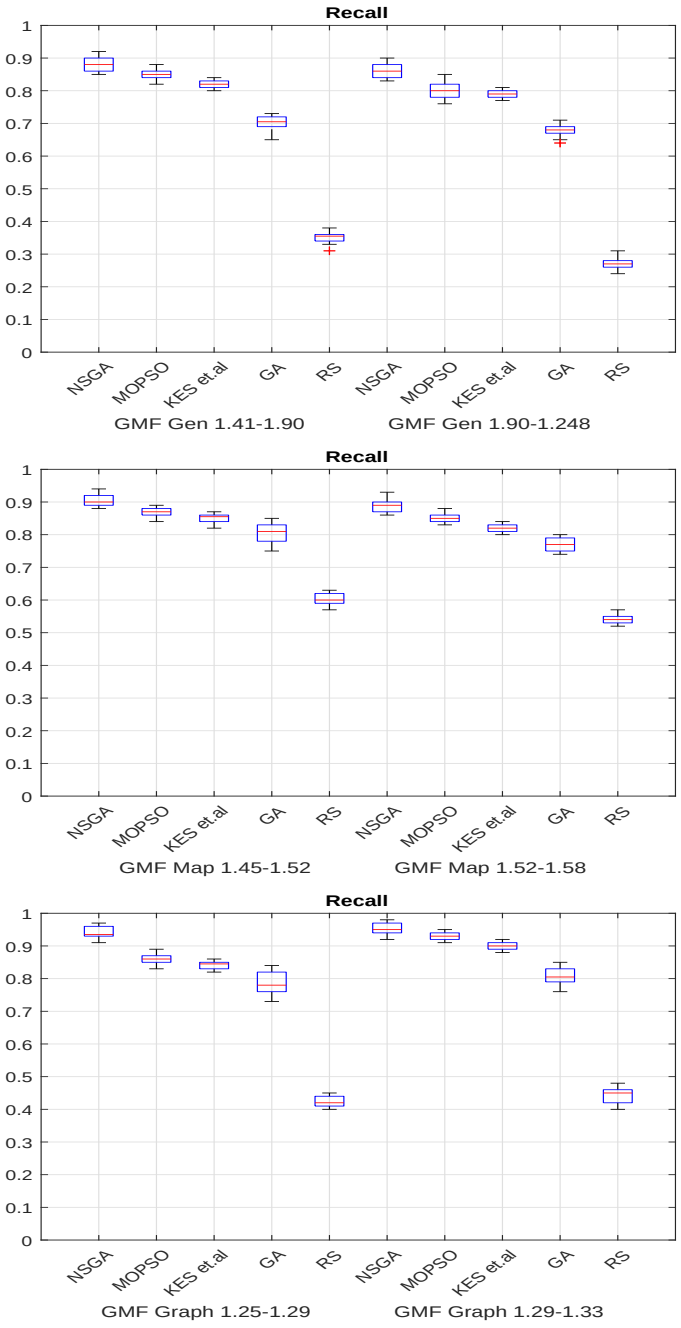

Figure 9: Average recall results of NSGA-II, MOPSO, Kessentini et al. [11], GA, and Random Search on 30 runs.

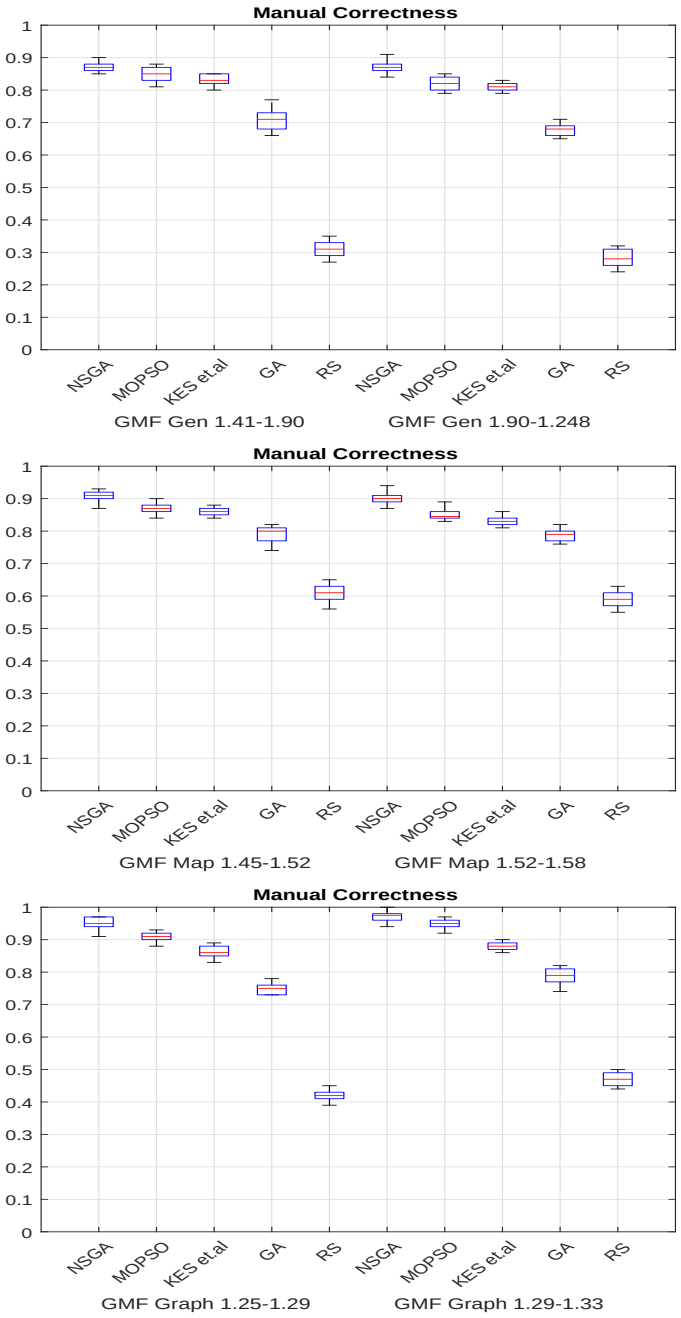

Figure 10: Average MC results of NSGA-II, MOPSO, Kessentini et al. [11], GA, and Random Search on 30 runs.

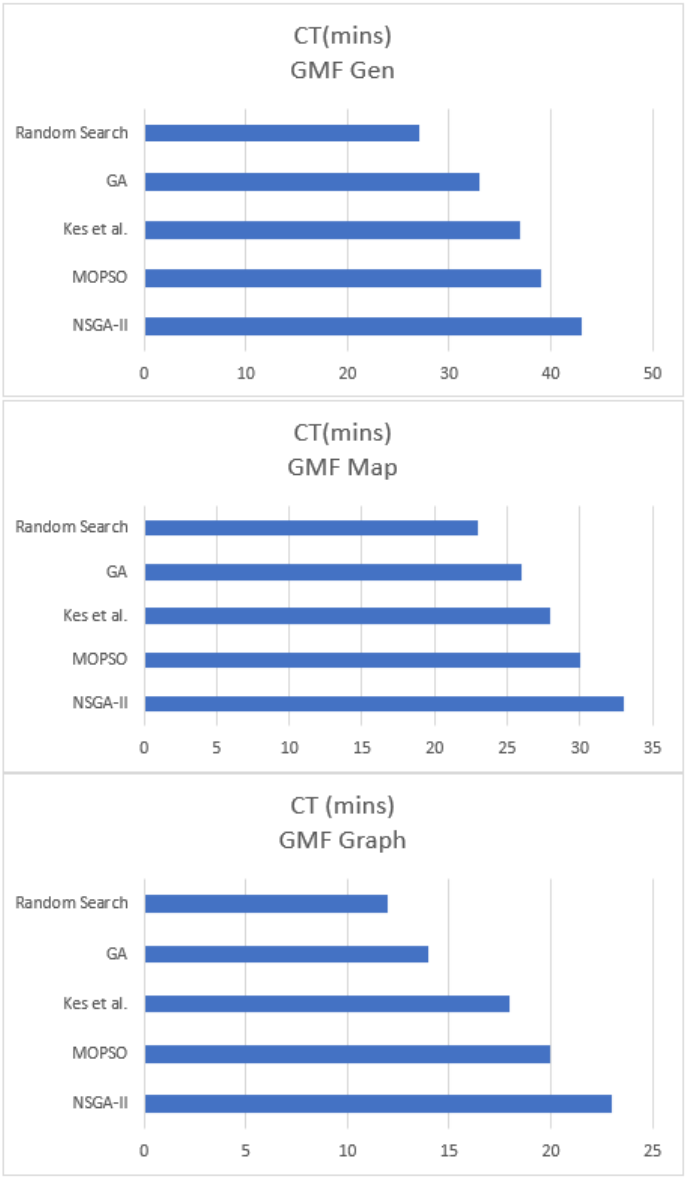

Figure 11: Average execution time of NSGA-II, Kessentini et al. [11], GA, and Random Search.

of accurate recommendation of edit operations. Furthermore, it is not required to provide real-time solutions for the problem of metamodel/model co-evolution. Since it is not sufficient to only evaluate the execution time, we have also evaluated the impact of the size of models on the performance of NSGA-II. It is clear from Figure 12 that the performance of NSGA-II in terms of both manual and automatic correctness

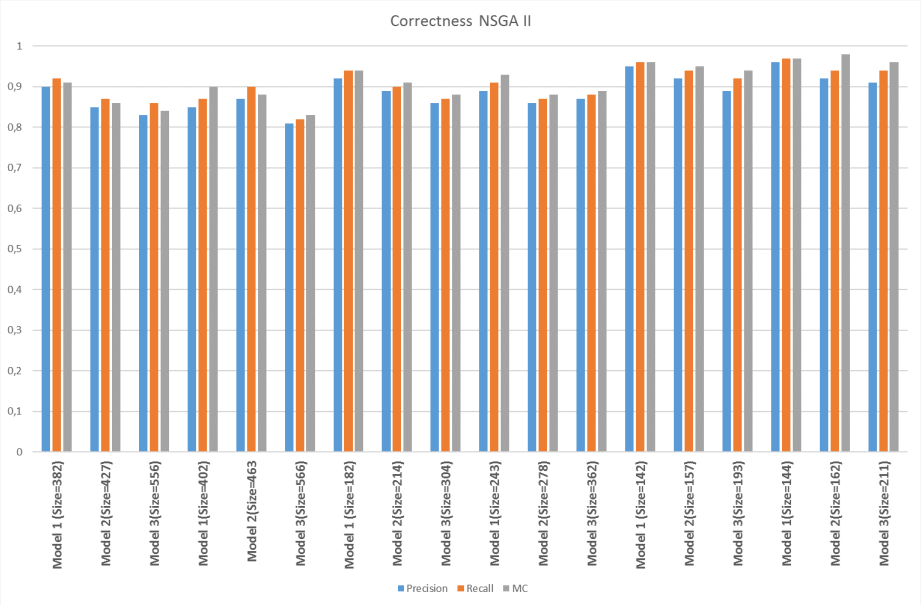

Figure 12: Impact of the size of models on the performance of NSGA-II.

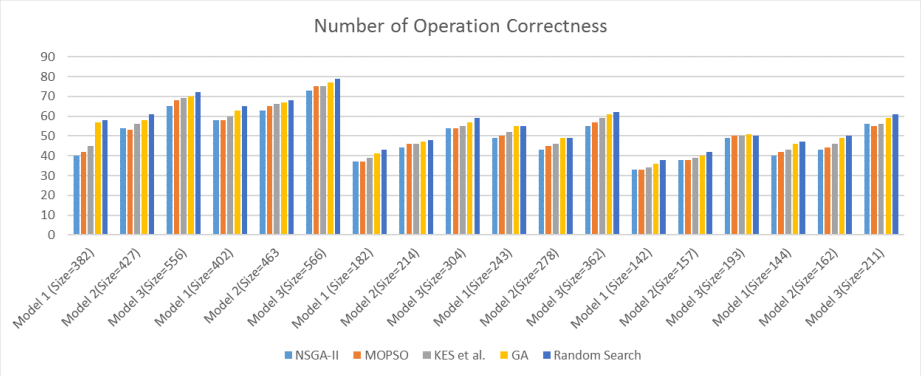

Figure 13: Average number of edit operation results of NSGA-II, MOPSO, Kessentini et al.[11], GA, and Random Search on 30 runs

is independent from the size of models or the number of expected edit operations. For example, the third model of GMF Map has a higher number of elements than the fifth model but the correctness results of model latter was the highest ones. To conclude and 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, RQ4 and RQ5. In this section, we compare our NSGA-II adap-

tation to several existing metamodel/model co-evolution approaches. To answer RQ3, we compared NSGA-II to another widely used multi-objective algorithm, MOPSO and to our previous work, using the same adapted fitness functions. We use all the metrics NO, PR, RE, MC and execution time to compare different between the following techniques: our NSGA-II adaptation, our previous work, MOPSO, a mono-objective genetic algorithm that has a single fitness function aggregating the three objectives and the deterministic approach of Wimmer et al. [17]. To make meaningful comparisons, we select the best solution for NSGA-II, Kessentini et al. [11] and MOPSO using a knee point strategy [15]. Thus, for NSGA-II and MOPSO, we select the knee point from the Pareto approximation having the median hyper volume value. The results of the comparison from 30 runs are depicted in Figures 8, 9 and 10 and 11. NSGA-II, MOPSO and Kessentini et al. [11] provide a better trade-off between our three objectives than a mono-objective algorithm in all the case studies. As described in Figure 13, for NO, the number of edit operations using NSGA-II and MOPSO, is lower than Kessentini et al. [11] in all the scenarios.This shows the benefits of using the detection of similarities between initial and revised model elements. The mono-objective GA approach that aggregates the three objectives in one, provides the highest number of edit operations than NSGA-II and MOPSO in all the cases. This confirms that it is difficult to find trade-offs between conflicting objectives aggregated in only one fitness function.

For PR, RC and MC, Figures 8, 9 and 10 show that the solutions provided by NSGA-II have the highest manual and automatic correctness values on most of the scenarios. In fact, the average PR and RC value for NSGA-II is higher than 89% and it is lower than 85% for all the remaining algorithms on all the evolution scenarios. The same observation is valid for MC, NSGA-II has the highest MC average value with 91% while the remaining algorithms their MC average is lower than 88%. Figures 10 and 13 reveal also an interesting observation that there is no correlation between the number of edit operations to recommend and the correctness values. More precisely, we sort PR, RE and MC based on the number of edit operations for each metamodel/-model evolution scenario. From this data, we conclude that PR, RE and MC are not necessarily affected negatively by a larger number of edit operations to recommend. For instance, we found that MC even increases from 86% to 88% for respectively one model from GMF Gen 1.41 and another model from GMF Gen 1.90 when the number of edit operations increases from 52 (for the model from GMF Gen 1.41) to 61 (for the model from GMF Gen 1.90). All these results were statistically significant on 30 independent runs using the statistical test with a 95% confidence level ($\alpha = 5\%$).

It is difficult to predict in general that a specific search algorithm is the best for a specific problem since the only valid proof is the experimentation results. Thus, we compared the results of NSGA-II with another multi-objective search algorithm, namely multi-objective particle swarm optimization (MOPSO). The results for MOPSO were also acceptable and very close to NSGA-II performance. In fact, this can be explained by two main reasons. First, the problem adaptation is the same for both algorithms (solution representation and fitness function). Second, the two algorithms have common parts and the main difference between them is the crowding distance and change operators. However, we noticed that when the number of expected changes become high the performance of NSGA-II is better than MOPSO.

To further investigate the performance of our co-evolution technique, we compared

its performance to a deterministic approach proposed by Wimmer et al. [17]. The deterministic approach defines generic rules for a set of possible metamodel changes that are applied to the co-evolved models. Figure 14 shows that our multi-objective approach clearly outperform, in average, the deterministic techniques based on all measures: precision, recall and manual correctness. The comparison is limited to the only case of UML State Machine evolution since we were only able to execute Wimmer's approach only that scenario, further adaptations are required to make it working on other metamodels. The outperformance of our approach could be explained by the fact that it is hard to generalize all possible metamodel/model co-evolution changes since several possible model-level edit operations could be recommended for the same meta-model change.

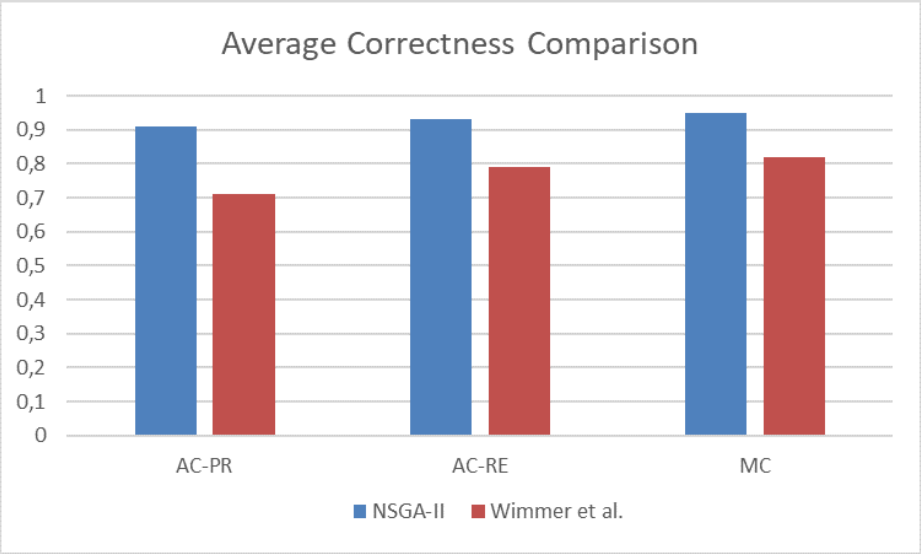

Figure 14: Average precision, recall and manual correctness of NSGA-II and Wimmer et al. [17] on UML State Machine metamodel evolution.

In conclusion, the results support the claim that our NSGA-II formulation provides a good trade-off between the three objectives, and outperforms an existing metamodel/model co-evolution approach [17].

4.4. Threats to Validity

There are four types of threats that can affect the validity of our experiments. We consider each of these in the following paragraphs.

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. We used the Wilcoxon rank sum test with a 95% confidence level 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 when comparing our NSGA-II formulation to the remaining

co-evolution techniques. Another threat related to our approach is that some solutions are satisfying well the fitness functions but still not generating the expected model due to the semantics change that were introduced to the metamodel level. Our approach just focuses on the use of information available at the metamodel and model level, however some complex scenarios of changes may also require to consider the developer in the loop to improve the precision and recall of our results. Another threat is that our multi-objective formulation treats the different types of edit operations with the same weight in terms of importance. However, some types of edit operations can be more important to recommend than others by developers.

Construct validity is concerned with the relationship between theory and what is observed. To evaluate the results of our approach, we selected solutions at the knee point as a proof-of-concept of the validity of our approach, but the programmers may select a different solution based on their preferences to give different weights to the objectives when selecting the best co-evolution solution. 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.

Internal validity is concerned with the causal relationship between the treatment and the outcome. We dealt with internal threats to validity by performing 30 independent simulation runs for each problem instance. This makes it highly unlikely that the observed results were caused by anything other than the applied multi-objective approach. Another threat is related to our choice of averaging rather than intelligently weighting the single objective function for the mono-objective genetic algorithm. Further experiments are required to evaluate the performance of the mono-objective approach with different weights to each component of the fitness function.

External validity refers to the generalizability of our findings. In this study, we performed our experiments on different widely used metamodels and models belonging to different domains and having different sizes. However, we cannot assert that our results can be generalized to other languages. Future replications of this study are necessary to confirm our findings. In addition, the comparison of the performance of NSGA-II to other multi-objective algorithms is limited to MOPSO and the approach presented in [12]. The decision was made as the concerned non-search based tool was easily accessible to us. Further empirical studies are required to deeply evaluate the performance of NSGA-II using the same problem formulation.

5. Related Work

In this section we discuss the different existing techniques related to our approach. As classified in the following, our work is related to two main research areas: (i) metamodel/model co-evolution (see [6, 7, 8, 29, 36, 37] for complementary literature studies) and (ii) search based software engineering [38].

5.1. Metamodel/Model Co-Evolution

Co-evolution has been subject for research since several decades in the database community [39], and especially, the introduction of object-oriented database systems [40] 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., [6, 7, 8, 29, 36, 37] for an overview). They can be classified in three categories [36]:

- Manual specification approaches in which the migration strategy is encoded manually by the modeler using general purpose programming languages (e.g., Java), or model transformation languages (e.g., ATL, QVT) [41, 42].
- Metamodel matching techniques used to infer a migration strategy from the difference between the original metamodel and the evolved metamodel [7, 43, 44, 45, 46].
- Operator based approaches that record the metamodel changes as a sequence of co-evolutionary operations used later to infer a complete migration strategy [4, 5, 6, 47].

We give an overview in the following about existing work in these three categories.

5.1.1. Manual specification approaches

In one of the early works [48], the co-evolution of models is tackled by designing co-evolution transformations based on metamodel change types. In [45, 46], 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), i.e., a transformation which takes/produces another transformation as input/output. In [49], the authors proposed an approach compromises five steps for model co-evolution: change detection, changes is detected either by comparing between metamodels or by tracing and recording the changes applied to the old version of the metamodel. The second step is a classification of the changes in metamodel and their impact in its instances in three categories: non-breaking changes: changes that do not break the models, breaking and resolvable changes that break models, but can be resolved automatically and breaking and non-resolvable: changes that break model instances but cannot be resolved automatically. Finally, an appropriate migration algorithm for model migration is determined. For initial model elements for which no transformation rule is found, a default copy transformation rule is applied. This algorithm has been realized in the model migration framework Epsilon Flock [42]and in the framework described in [41]. Another manual specification approach is presented in [50] where a specific transformation language is derived to describe the evolution on the metamodel level and derive a transformation for the model level.

5.1.2. Metamodel matching approaches

Most of the mentioned approaches which are based on out-place model transformation intend to shield the user from creating copy rules; a task which seems to accidental complexity of pure out-place model transformations applied for model co-evolution 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. In such approaches (cf. e.g., [4, 43, 51, 52, 53]), the co-evolution rules are specified as in-place transformation rules by using a kind of unified metamodel representing both metamodel versions, and then, to eliminate model elements that are not the part of evolved meta-model anymore, a check out transformation is performed. Thus, the models can be migrated to the new metamodel version without generating completely new models instead the models are simply rewritten as long as needed.

5.1.3. Operator based approaches

Other contributions are based on using coupled operations [4, 5, 9, 47]. In [5] the author provides a library of co-evolutionary operators for MOF metamodels, each of these operators provides a model migration strategy. In [47] the author provides a tool support for the evolution of Ecore-based metamodels, that records the metamodel changes as a sequence of co-evolutionary operations that are structured in a library and used later to generate a complete migration strategy. But, when no appropriate operator is available, model developer does the migration manually, so those approaches depend on the library of reusable coupled operators they provide. To this end, the authors in [4] extended the tool by providing two kinds of coupled operators: reusable and custom coupled operators. The reusable operators allow the reuse of migration strategy independently of the specific metamodel. The custom coupled operators allow to attach a custom migration to a recorded metamodel change. In [54], an approach is presented that uses in a first phase metamodel matching to derive the changes between two metamodel versions and in a second phase, operations are applied based on the detected changes to migrate the corresponding models. Additionally, in [55], weaving models are employed to connected the changes of the metamodels with the model elements to provide a basis for reasoning how to perform the migration of the models to the new metamodel versions.

5.1.4. Synopsis

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. [56]. In particular, we search for transformation executions on the model level, which fulfill 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 from the calculated set of metamodel changes. So the resolution result is not guaranteed to be the one desired to co-evolve a model. Our approach, gives the user a better control over the result, since we propose a set of alternative resolution strategies(the best solutions from the pareto front) to the user to select from.

5.2. Search Based Software Engineering

Most software engineering problems can be formulated as search problems, where the goal is to find optimal or near-optimal solutions [38]. This search is often complex with many competing constraints and objectives. The situation can be worse since nowadays successful software is more complex, more critical, and more dynamic leading to an increasing need to automate or semi-automate the search process of acceptable solutions for software engineers. As a result, an emerging software engineering area, called Search-Based Software Engineering (SBSE) [57], is rapidly growing. SBSE is a software development practice that focuses on couching software engineering problems as optimization problems and utilizing meta-heuristic techniques to discover and automate the search for near optimal solutions to those problems.

The aim of SBSE research is to move software engineering problems from human-based search to machine-based search, using a variety of techniques from the fields of meta-heuristic search, operations research and evolutionary computation paradigms. SBSE has proved to be a widely applicable and successful approach, with many applications right across the full spectrum of activities in software engineering, from initial requirements, project planning, and cost estimation to regression testing and onward evolution [38].

This increasing maturity of the field has led to a number of tools for SBSE applications. However, the only work we are aware of discussing metamodel/model co-evolution using some search-based techniques is [58]. In this paper, the 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 co-evolution approaches and outperforms logic-based approaches for repairing models [18]. 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.

Finally, we would like to mention that SBSE has been applied to related problems in Model-Driven Engineering such as model transformation [59], transformation testing [60], model refactoring [24, 61], requirements modeling [62, 63], product line testing [64] etc. However, these problems are different in nature than the co-evolution problem investigated in this paper.

6. Conclusion

In this paper, we proposed 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 with a minimum of adaptations. For this, the search process is guided by three fitness functions:

the number of recommended operations, structural and semantics similarity with the initial metamodel and the conformity with the constraints of the new metamodel. We evaluated our approach on several evolution scenarios extracted from different widely used metamodels. The results confirm the effectiveness of our approach with average manual correctness, precision and recall respectively higher than 91%, 88% and 89% on the different co-evolution scenarios. A comparison with our previous works, confirms the out-performance of our multi-objective formulation.

Although our approach has been evaluated with real-world scenarios with a reasonable number of applied operations and models, we plan to work on even larger data sets including different versions of metamodels and models as well as huge 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. Furthermore, we plan to consider other types of co-evolution problems such as metamodels/transformation co-evolution. Finally, we will consider the use of other meta-heuristics, such as the indicator-based algorithm [64], to our co-evolution problem and compare its performance with 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, and by the Austrian Science Fund (FWF) P 28519-N31.

References

[1] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool Publishers, 2017.

[2] D. C. Schmidt, Model-driven engineering, IEEE Computer 39 (2) (2006) 25–31.

[3] L. Iovino, A. Pierantonio, I. Malavolta, On the impact significance of metamodel evolution in MDE, Journal of Object Technology 11 (3) (2012) 1–33.

[4] M. Herrmannsdörfer, COPE - A Workbench for the Coupled Evolution of Metamodels and Models, in: Proceedings of the International Conference on Software Language Engineering (SLE), 2010, pp. 286–295.

[5] G. Wachsmuth, Metamodel adaptation and model co-adaptation, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2007, pp. 600–624.

[6] M. Herrmannsdörfer, G. Wachsmuth, Coupled evolution of software metamodels and models, in: Evolving Software Systems, Springer, 2014, pp. 33–63.

[7] B. Meyers, H. Vangheluwe, A framework for evolution of modelling languages, Sci. Comput. Program. 76 (12) (2011) 1223–1246.

[8] R. Hebig, D. E. Khelladi, R. Bendraou, Approaches to co-evolution of metamodels and models: A survey, IEEE Transactions on Software Engineering 43 (5) (2017) 396–414.

[9] M. Herrmannsdoerfer, S. Vermolen, G. Wachsmuth, An extensive catalog of operators for the coupled evolution of metamodels and models, in: Proceedings of the International Conference on Software Language Engineering (SLE), 2011, pp. 163–182.

[10] D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, M. Gervais, Detecting complex changes and refactorings during (meta)model evolution, Information Systems 62 (2016) 220–241.

[11] W. Kessentini, H. A. Sahraoui, M. Wimmer, Automated metamodel/model co-evolution using a multi-objective optimization approach, in: Proceedings of the European Conference on Modelling Foundations and Applications (ECMFA), 2016, pp. 138–155.

[12] W. Kessentini, Automated metamodel/model co-evolution using a multi-objective optimization approach, in: Proceedings of the ACM Student Research Competition at MODELS, 2015, pp. 13–18.

[13] L. Muflikhah, B. Baharudin, Document clustering using concept space and cosine similarity measurement, in: Proceedings of the International Conference on Computer Technology and Development (ICCTD), 2009, pp. 58–62.

[14] S. Mostaghim, J. Teich, Strategies for finding good local guides in multi-objective particle swarm optimization (mo), in: Proceedings of the Swarm Intelligence Symposium (SIS), 2003, pp. 26–33.

[15] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II, in: Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN), 2000, pp. 849–858.

[16] E. Cariou, C. Ballagny, A. Feugas, F. Barbier, Contracts for model execution verification, in: Proceedings of the European Conference on Modelling Foundations and Applications (ECMFA), 2011, pp. 3–18.

[17] M. Wimmer, A. Kusel, J. Schoenboeck, W. Retschitzegger, W. Schwinger, On using inplace transformations for model co-evolution, in: Proceedings of International Workshop on Model Transformation with ATL (MtATL), 2010, pp. 65–78.

[18] J. Schoenboeck, A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Schwinger, M. Wimmer, M. Wischenbart, CARE: A Constraint-Based Approach for Re-Establishing Conformance-Relationships, in: Proceedings of the Asia-Pacific Conference on Conceptual Modelling (APCCM), 2014, pp. 19–28.

[19] D. D. Ruscio, J. Etzlstorfer, L. Iovino, A. Pierantonio, W. Schwinger, Supporting variability exploration and resolution during model migration, in: Proceedings of the European Conference on Modelling Foundations and Applications (ECMFA), 2016, pp. 231–246.

[20] Y. Zhang, M. Harman, S. A. Mansouri, The multi-objective next release problem, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2007, pp. 1129–1137.

[21] C. Henard, M. Papadakis, G. Perrouin, J. Klein, Y. L. Traon, Multi-objective test generation for software product lines, in: Proceedings of the International Software Product Line Conference (SPLC), 2013, pp. 62–71.

[22] A. Ouni, M. Kessentini, H. A. Sahraoui, M. Boukadoum, Maintainability defects detection and correction: a multi-objective approach, Autom. Softw. Eng. 20 (1) (2013) 47–79.

[23] E. Batot, W. Kessentini, H. Sahraoui, M. Famelis, Heuristic-based recommendation for metamodel - OCL coevolution, in: Proceedings of International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017, pp. 210–220.

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

[25] X. Hu, R. Eberhart, Multiobjective optimization using dynamic neighborhood particle swarm optimization, in: Proceedings of the Congress on Evolutionary Computation (CEC), 2002, pp. 1677–1681.

[26] M. Richters, A precise approach to validating UML models and OCL constraints, Tech. rep. (2001).

[27] M. Herrmannsdoerfer, D. Ratiu, G. Wachsmuth, Language Evolution in Practice: The History of GMF, in: Proceedings of the International Conference on Software Language Engineering (SLE), 2009, pp. 3–22.

[28] M. Herrmannsdoerfer, GMF: A model migration case for the transformation tool contest, in: Proceedings of the Transformation Tool Contest (TTC), 2011, pp. 1–5.

[29] L. M. Rose, M. Herrmannsdoerfer, S. Mazanek, P. V. Gorp, S. Buchwald, T. Horn, E. Kalnina, A. Koch, K. Lano, B. Schätz, M. Wimmer, Graph and model transformation tools for model migration - empirical results from the transformation tool contest, Software and System Modeling 13 (1) (2014) 323–359.

[30] D. Di Ruscio, R. Lämmel, A. Pierantonio, Automated Co-evolution of GMF Editor Models, in: Proceedings of the International Conference on Software Language Engineering (SLE), 2011, pp. 143–162.

[31] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, V. G. da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, Trans. Evol. Comp 7 (2) (2003) 117–132.

[32] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the International Conference on Software Engineering (ICSE), 2011, pp. 1–10.

[33] X. Li, A non-dominated sorting particle swarm optimizer for multiobjective optimization, in: Proceedings of the International Conference on Genetic and Evolutionary Computation (GECCO), 2003, pp. 37–48.

[34] M. Kessentini, U. Mansoor, M. Wimmer, A. Ouni, K. Deb, Search-based detection of model level changes, Empirical Software Engineering 22 (2) (2017) 670–715.

[35] M. W. Mkaouer, M. Kessentini, S. Bechikh, M. Ó Cinnéide, A robust multi-objective approach for software refactoring under uncertainty, in: Proceedings of the International Symposium on Search-Based Software Engineering (SSBSE), 2014, pp. 168–183.

[36] L. M. Rose, R. F. Paige, D. S. Kolovos, F. A. C. Polack, An Analysis of Approaches to Model Migration, in: Proceedings of MoDSE-MCCM Workshop, 2009, pp. 6–15.

[37] R. F. Paige, N. D. Matragkas, L. M. Rose, Evolving models in model-driven engineering: State-of-the-art and future challenges, Journal of Systems and Software 111 (2016) 272–280.

[38] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Computing Surveys (CSUR) 45 (1) (2012) 11:1–11:61.

[39] J. F. Roddick, Schema evolution in database systems: An annotated bibliography, SIGMOD Rec. 21 (4) (1992) 35–40.

[40] J. Banerjee, W. Kim, H.-J. Kim, H. F. Korth, Semantics and Implementation of Schema Evolution in Object-Oriented Databases, in: Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data (SIGMOD), 1987, pp. 311–322.

[41] A. Narayanan, T. Levendovszky, D. Balasubramanian, G. Karsai, Automatic domain model migration to manage metamodel evolution, in: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), 2009, pp. 706–711.

[42] L. M. Rose, D. S. Kolovos, R. F. Paige, F. A. C. Polack, Model migration with Epsilon Flock, in: Proceedings of the International Conference on Theory and Practice of Model Transformations (ICMT), 2010, pp. 184–198.

[43] B. Meyers, M. Wimmer, A. Cicchetti, J. Sprinkle, A generic in-place transformation-based approach to structured model co-evolution, in: Proceedings of Multi-Paradigm Modeling (MPM) Workshop, 2010, pp. 1–13.

[44] A. Cicchetti, F. Ciccozzi, T. Leveque, A. Pierantonio, On the concurrent versioning of metamodels and models: challenges and possible solutions, in: Proceedings of the International Workshop on Model Comparison in Practice (IWMCP), 2011, pp. 16–25.

[45] K. Garcés, F. Jouault, P. Cointe, J. Bézivin, Managing model adaptation by precise detection of metamodel changes, in: Proceedings of the European Conference on Modelling Foundations and Applications (ECMFA), 2009, pp. 34–49.

[46] A. Cicchetti, D. D. Ruscio, R. Eramo, A. Pierantonio, Automating co-evolution in model-driven engineering, in: Proceedings of the International Enterprise Distributed Object Computing Conference (EDOC), 2008, pp. 222–231.

[47] M. Herrmannsdoerfer, S. Benz, E. Juergens, Cope - automating coupled evolution of metamodels and models, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2009, pp. 52–76.

[48] J. Sprinkle, G. Karsai, A domain-specific visual language for domain model evolution, J. Vis. Lang. Comput. 15 (3-4) (2004) 291–307.

[49] B. Gruschko, D. Kolovos, R. Paige, Towards synchronizing models with evolving metamodels, in: Proceedings of the Workshop on Model-Driven Software Evolution (MoDSE), 2007, pp. 1–9.

[50] S. Vermolen, E. Visser, Heterogeneous coupled evolution of software languages, in: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), 2008, pp. 630–644.

[51] F. Mantz, Y. Lamo, G. Taentzer, Co-Transformation of Type and Instance Graphs Supporting Merging of Types with Retyping, ECEASST 61 (2013) 1–24.

[52] G. Taentzer, F. Mantz, T. Arendt, Y. Lamo, Customizable model migration schemes for meta-model evolutions with multiplicity changes, in: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS), 2013, pp. 254–270.

[53] F. Mantz, G. Taentzer, Y. Lamo, U. Wolter, Co-evolving meta-models and their instance models: A formal approach based on graph transformation, Sci. Comput. Program. 104 (2015) 2–43.

[54] F. Anguel, A. Amirat, N. Bounour, Hybrid approach for metamodel and model co-evolution, in: Proceedings of Conference on Computer Science and Its Applications (CIIA), 2015, pp. 563–573.

[55] F. Anguel, A. Amirat, N. Bounour, Using weaving models in metamodel and model co-evolution approach, in: Proceedings of the International Conference on Computer Science and Information Technology (CSIT), 2014, pp. 142–147.

[56] A. Demuth, M. Riedl-Ehrenleitner, R. E. Lopez-Herrejon, A. Egyed, Co-evolution of metamodels and models through consistent change propagation, Journal of Systems and Software 111 (2016) 281–297.

[57] M. Harman, B. F. Jones, Search-based software engineering, Information and software Technology 43 (14) (2001) 833–839.

[58] J. R. Williams, R. F. Paige, F. A. C. Polack, Searching for Model Migration Strategies, in: Proceedings of Models and Evolution (ME) Workshop, 2012, pp. 39–44.

[59] M. Fleck, J. Troya, M. Wimmer, Search-based model transformations, Journal of Software: Evolution and Process 28 (12) (2016) 1081–1117.

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

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

[62] M. Kessentini, U. Mansoor, M. Wimmer, A. Ouni, K. Deb, Search-based detection of model level changes, Empirical Software Engineering 22 (2) (2017) 670–715.

[63] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh, K. Deb, MOMM: multi-objective model merging, Journal of Systems and Software 103 (2015) 423–439.

[64] A. S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: A case study in software product lines, in: Proceedings of the International Conference on Software Engineering (ICSE), 2013, pp. 492–501.