Transforming Interactive Multi-objective Metamodel/Model Co-evolution into Mono-objective Search via Designer's Preferences Extraction

Wael Kessentini(✉) and Vahid Alizadeh(✉)

College of Computing and Digital Media, DePaul University, 243 South Wabash Avenue Chicago, Chicago, IL 60604, USA
{wkessent,alizadeh}@depaul.edu

Abstract. The simultaneous evolution of metamodels and models is called the meta-models/models co-evolution problem. While some Interactive/automated metamodel/model co-evolution techniques have been proposed using multi-objective search, designers still need to explore a large number of possible revised models. In this paper, we propose an approach to convert multi-objective search into a mono-objective one after interacting with the designer to identify a set of model changes based on his/her preferences. The first step consists of using a multi-objective search to generate different possible model edit operations by finding a trade-off between three objectives. Then, the designer may give feedback on some proposed solutions. The extracted preferences are used to transform the multi-objective search into a mono-objective one by generating an evaluation function based on the weights for the existing fitness functions that are automatically computed from the feedback. Thus, the designer will just interact with only one solution generated by the mono-objective search. We evaluated our approach on a set of meta-model/model co-evolution case studies and compared it to existing fully automated and interactive meta-model/model co-evolution techniques. The results show that the mono-objective search after the interaction with the users significantly improved the co-evolution changes for several widely used metamodels.

Keywords: Model co-evolution · Interactive multi-objective search

1 Introduction

Similar to the source code of large systems, the modeling languages (i.e., meta-models) are subject to evolution due to changing requirements and technological constraints requiring existing models to be adapted [1,2]. Thus, a set of changes must be applied to the initial model versions to fix the inconsistencies with the new metamodel version. This process is called metamodel/model co-evolution [1,3].

Several co-evolution studies are proposed where most of them are providing either a manual or semi-automated support based on pre-defined templates of evolution scenarios [4–8]. In addition to being pre-defined, these templates are specific to the artifact/models to co-evolve and the metamodel. Few fully automated co-evolution studies tried to find an entire edit operations sequence that revises models in accordance with the new metamodel version [3,9,10]. However, several transformations require interactions with the user especially when new elements are added to the new metamodel they are hard to full-automate. Recently, an approach has been proposed to interactively evaluate the co-evolved models using search-based software engineering [11]. The designers can provide feedback about the co-evolved models and introduce manual changes to some of the edit operations that revise the model. However, this interactive process can be expensive, and tedious since designers must evaluate every recommended set of edit operations and adapt them to the targeted design, especially in large models where the number of possible co-evolution strategies can grow exponentially. Besides, it is challenging to define upfront weights to some edit operation since the designer needs to have a look at the generated solutions to express his preferences.

In this paper, we propose an approach that takes advantage of existing Metamodel/Model co-evolution works. Thus, we propose a way to convert multi-objective search into a mono-objective one after few interactions with the developer. The first step consists of using a multi-objective search, based on the evolutionary algorithm NSGA-II [12], to generate a diverse set of model migration strategies by finding a trade-off between three objectives of reducing the number of edit operations, the dissimilarity with the initial models to reduce the information loss and the number of inconsistencies between the models and new metamodel. Then, the designer may give some feedback on the generated solutions to express his/her preferences by selecting relevant ones. The extracted preferences from the designer, via an analysis of the location of these solutions in the objective space, are used to transform the multi-objective search into a mono-objective one by generating an evaluation function based on the weights that are automatically calculated from the selected solutions. Therefore, the designer will interact in the next iterations with only one co-evolution solution generated by the mono-objective search.

Our approach is taking the advantages of mono-objective search, multi-objective search, and interactive computational intelligence. Multi-objective algorithms are powerful in diversifying solutions and finding trade-offs between many objectives but generate many solutions as an output. The interactive algorithm is useful in terms of extracting designers' knowledge and preferences. Mono-objective algorithms are the best in terms of optimization power once the evaluation function is well-defined and generate only one solution as an output. We selected 16 active developers to manually evaluate the effectiveness of our tool on four well-known metamodel/model co-evolution case studies. The results show that the participants found their desired revised models faster and more accurate than the current state of the art. A supplementary appendix materials can be found in the following link [13].

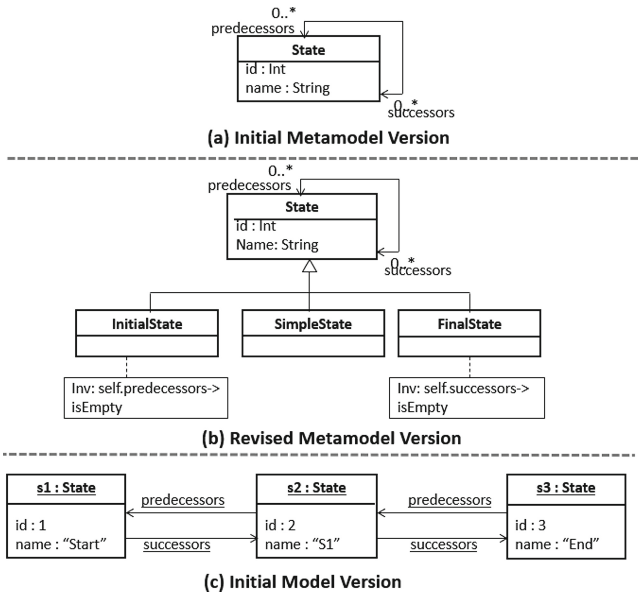


Fig. 1. A simplified metamodel evolution example

2 Background and Motivations

Figure 1 shows an example of a simplified metamodel evolution, based on the simple state machine language and an initial model conform to it. The meta-model evolution comprises three steps: extract sub-classes for *State* class resulting in *InitialState*, *SimpleState*, and *FinalState*, make class *State* abstract, and refine the cardinalities of the predecessor/successor references for the subclasses. This evolution results in the fact that, besides other constraints violations, the constraint which is shown in Listing 1.1 is violated when considering the initial model of Fig. 1c and its conformance to the new metamodel of Fig. 1b.

Listing 1.1. 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)
```

To re-establish conformance for the given example, assume for now that only two operations on models are used in this context. Non-conforming objects may either be retyped (reclassified as instances of the concrete classes) or deleted.

Thus, the potential solution space for retyping or deleting non-conforming elements contains $(c + 1)^O$ solutions (with c = number of candidate classes + 1 for deletion, o = number of non-conforming objects). This means, in our given example, we would end up with 64 possible co-evolutions.

Several co-evolution studies proposed to revise models after metamodels evolution from manual to fully automated approaches [2]. Recently, few automated/interactive tools [9–11] used search-based software engineering to generate revised models. The proposed tools refine an initial model instantiated from the previous metamodel version to make it as conform 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 dissimilarities between the initial and revised models. During the process, the designer may provide some feedback on the proposed solutions in order to improve them in the next iterations. The output is several equally good solutions (edit operations that revise the model) presented to designers to select the appropriate one based on his/her preferences.

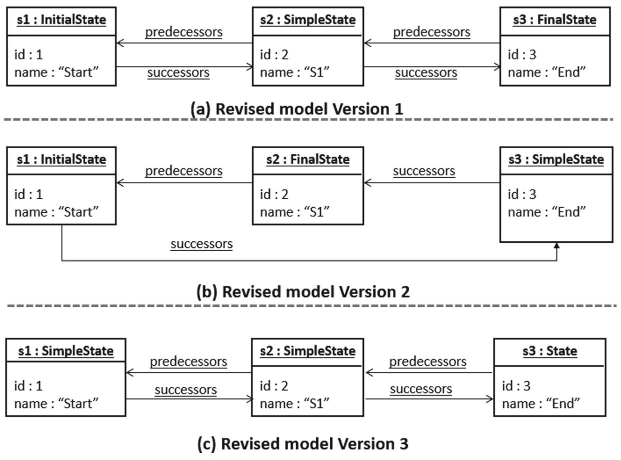


Fig. 2. Tentative revised models

Figure 2 shows modified models after applying a set of edit operations extracted from the output of an existing tool [11]. This figure shows that there may be several possible solutions where the user has to decide which one to select in the search space based on the preferences.

3 Approach Overview

Our approach includes three main phases. The first phase is the multi-objective algorithm, NSGA-II, executed for several iterations to generate a set of non-dominated co-evolution solutions called Pareto-optimal solutions [12], defined as a set of edit operations applied to the initial model, balancing the three objectives of minimizing the number of suggested edit operations, the deviation with the initial model, and the number conformance errors with the revised metamodel.

The output of the first component can be a large number of possible solutions. Thus, it is essential to provide the designers with additional support for interacting with this set of solutions. In the second phase, the user can interact with the tool at the solution level, by accepting or rejecting or modifying suggested edit operation(s) and s/he can also give a score to a selected solution between −1 and 1 (the highest is the better). Finally, we extract the preferences automatically and use them to transform the multi-objective problem into a mono-objective one by generating weights for each of the three objectives based on the selected solutions' locations in the objective space. The output of the mono-objective search is a single solution fitting the user's expectations and preferences; then the designer can interact with that solution if needed and continue the execution of the mono-objective algorithm until selecting a final co-evolution solution. In the following, we will explain, in detail, the phases of our approach.

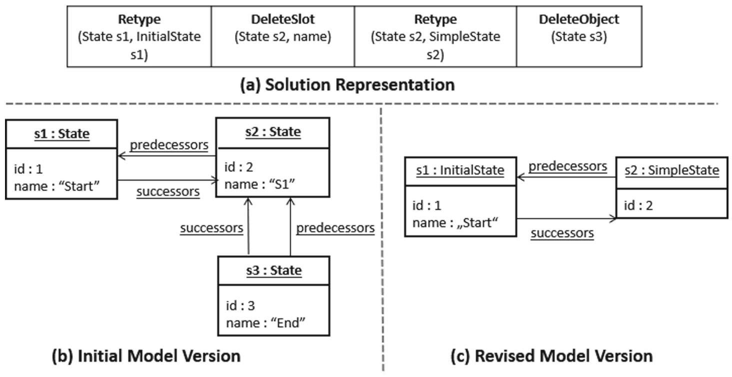


Fig. 3. Solution representation.

3.1 Phase 1: Multi-objective Metamodel/Model Co-evolution

Solution Representation. A co-evolution solution consists of a sequence of n edit operations to revise the initial model. The vector-based representation is used to define the edit operations sequence. Each vector's dimension has an operation, and its index in the vector indicates the order in which it will be applied. Consequently, vectors representing different solutions may have different sizes, i.e., the number of edit operations.

Table 1 shows the possible edit operations that can be applied to model elements. The instances of classes are called objects, instances of features are called slots, and instances of references are called links. These operations are inspired by the catalog of operators for metamodel/model co-evolution presented in [14]. The catalog includes both metamodel and model changes. Thus, we selected from it all the edit operations that can be applied to the model level since we are not changing the metamodels in this paper. Figure 3 represents a solution that can be applied to the initial model of our motivating example described in Sect. 2.

Table 1. Model edit operations.

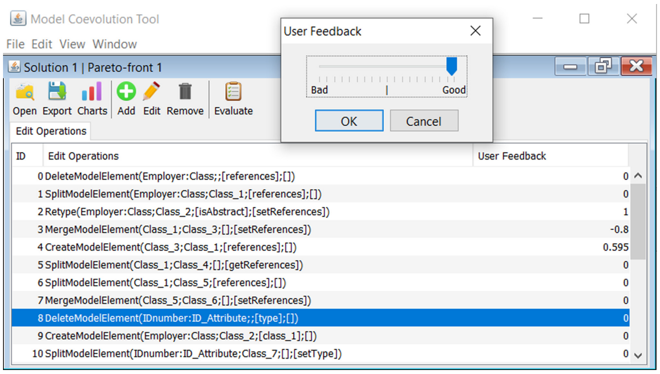
Operations	**Element**	**Description**
Create/delete	Object, link, slot	Add/remove an element in the initial model.
Retype	Object	Replace an element by another equivalent element having a different type.
Merge	Object, link, slot	Merge several model elements of the same type into a single element.
Split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

Fitness functions. The investigated co-evolution problem involves searching for the best sequence of edit operations to apply among the set of possible ones. A good solution s is a sequence of edit operations to apply to an initial model with the objectives of minimizing the number of non-conformities $f_1(s) = nvc(s)$ with the new metamodel version, the number of changes $f_2(s) = nbOp(s)$ applied to the initial model, and the inconsistency $f_3(s) = dis(s)$ between the initial and the evolved models such as the loss of information.

The first fitness function $nvc(s)$ counts the number of violated constraints w.r.t. the evolved metamodel after applying a sequence s of edit operations. We 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 [15]: related to model objects, i.e., model element (denoted by O.*), related to objects' values (V.*), and related to objects' links (L.*). We use the implementation of these constraints in our experiments inspired by Schoenboeck et al. [3] and Richters et al. [15] 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 exist in EMF. The full list constraints can be found in this link [13].

For the second fitness function, which aims to minimize 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 the 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 [16]. In the first step, we tokenize the names of initial and revised model elements. The textual and context similarity between elements are grouped together to create a new class, which is an essential factor in evaluating the revised model's cohesion. 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. This function will compare each of the initial model elements and all the elements of the revised model to find the best matching.

3.2 Phase 2: Interaction and Preference Extraction



ID	Edit Operations	User Feedback
0	DeleteModelElement(Employer:Class;;[references];[])	0
1	SplitModelElement(Employer:Class;Class_1;[references];[])	0
2	Retype(Employer:Class;Class_2;[isAbstract];[setReferences])	1
3	MergeModelElement(Class_1;Class_3;[];[setReferences])	-0.8
4	CreateModelElement(Class_3;Class_1;[references];[])	0.595
5	SplitModelElement(Class_1;Class_4;[];[getReferences])	0
6	SplitModelElement(Class_1;Class_5;[references];[])	0
7	MergeModelElement(Class_5;Class_6;[];[setReferences])	0
8	DeleteModelElement(IDnumber:ID_Attribute;;[type];[])	0
9	CreateModelElement(Employer:Class;Class_2;[class_1];[])	0
10	SplitModelElement(IDnumber:ID_Attribute;Class_7;[];[setType])	0

Fig. 4. User interactions with the solutions of the multi-objective search

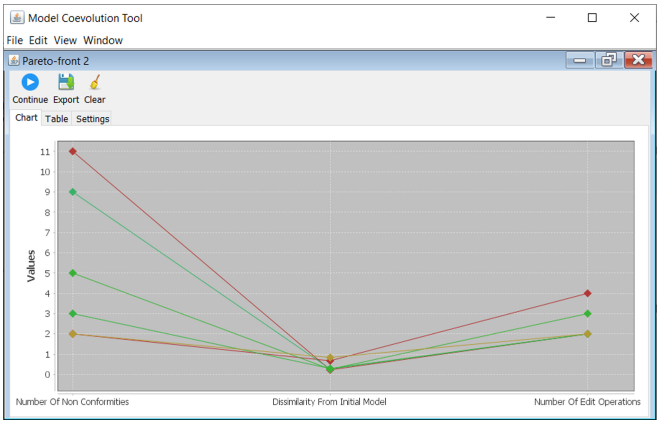


Fig. 5. The output of phase 2 (interactions with the user). (Color figure online)

The main goal of this step is to enable the designer's interactions with the solutions generated by the first phase of the multi-objective search. The interaction can be performed at the solution and edit operation levels depending on the user's desire. The feedback is quantified to a continuous score in the range of $[-1, 1]$. The user can evaluate a solution by modifying its edit operations (edit, add, delete, re-order) or just rate the whole solution, as shown in Fig. 4. After the designer's interaction, solution scores ($Score_{s_i}$) are computed as the average score of edit operations in a solution. The solutions with the highest score are considered as the region of interest. It indicates the preferred objectives and edit operations. The line chart of Pareto-front solutions after interactions is shown in Fig. 5. The color of each line indicates user preferences (green as preferred solutions versus red as non-preferred solutions.

3.3 Phase 3: Preference-Based Mono-objective Co-evolution

One of the main contributions of this paper is the ability to convert a multi-objective algorithm into a mono-objective one after interacting with the designer to extract his/her preferences. Mono-objective algorithms are known to be the best in terms of optimization but require that the fitness function should be well-defined based on the decision maker's preferences. The Multi-objective Evolutionary Algorithm used in Phase 1 might not provide high-quality solutions in the region of interest of the developer because of the high dimensionality nature of the problem and the need to find trade-offs. Therefore, it is important to consider the user preferences extracted in Phase 2.

The goal of this phase is to use the preferences extracted from the designer after the multi-objective optimization to transform the problem into a single objective optimization problem by aggregating objectives according to the user's preferences. This transformation gives the decision maker a single solution. Consequently, our proposed approach is a combination of all three categories of preference-based search where the preferences are expressed after the first evolutionary process, then they are incorporated to guide the single-objective optimization.

One way to convert a multi-objective optimization problem to a mono-objective problem and achieve a single solution is called the Weighted Sum Method (WSM). In this method, the single preference fitness function is computed as a linear weighted sum of multiple objectives. The main drawback of the WSM method is that it needs the weights parameters to be given. Fortunately, in our case, those parameters are computed automatically from the decision-maker preferences of the interactive optimization process (preferred solutions based on the interaction scores) in the objectives space. Thus, the weight of one or more objectives can get the value 0 (or almost) if the selected solution(s) by the developer penalized them while favoring other objectives. Also, the WSM is not computationally expensive, unlike the other scalarization methods.

In order to solve the converted mono-objective problem, we adopted a standard Genetic Algorithm (GA). To adapt the GA algorithm to our co-evolution problem, we use the same solution representation and fitness functions as reported in phase 1. The importance (weights) of the objectives are based on the preferred solutions by the user with an interaction score higher than 0.5. The obtained single fitness function is employed to evaluate the solutions in the execution of adapted GA. Thus, the weight of each objective is calculated as the average of the objective values of the preferred solution(s) by the user. We note that all the objectives of the multi-objective search are normalized using the min-max function.

Instead of generating the initial population randomly, we acquire the user preferred solutions as the elite set of solutions from which the search process is initiated. Thus, we do not generate solutions randomly for the mono-objective GA, but we take the preferred solutions as the initial population, so we do not lose the knowledge extracted from the developer. If the number of preferred solutions is low, then we apply the mutation operator to generate more solutions. The solutions are evaluated via the preference function aggregated from multiple objectives. When the stopping condition is satisfied, the single optimal solution is recommended to the user. Similar to Phase 1, the user can interact with this solution via editing/adding/removing the edit operations.

4 Evaluation

4.1 Research Questions and Experimental Setup

- **RQ1: Search validation.** How does our approach perform compared to random search (RS)?
- **RQ2: Benefits.** To what extent can our approach make relevant recommendations for designers compared to existing metamodel/model co-evolution techniques including multi-objective search and an existing deterministic method?

– **RQ3: The relevance of designers' preferences extraction.** To what extent can our approach reduce the interaction effort comparing to existing metamodel/model co-evolution techniques?

Studied Metamodels and Models. To answer the research questions, we considered the evolution of GMF covering a period of two years and the UML Class Diagram metamodel evolution from [17,18]. These case studies are interesting scenarios since they represent real metamodel evolutions, used in an empirical study [19] and studied in other contributions [20–22]. For GMF, we chose to analyze the extensive evolution of three Ecore metamodels. 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. We created different scenarios based on the number of changes introduced at the metamodel level from the different metamodel releases of GMF and UML. We merged the releases that did not include extensive changes, and we generated two evolution scenarios per metamodel type.

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 eight different models to evolve for the GMF case-studies. 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 metamodels coverage. Furthermore, we used an existing set of 10 generated models for the case of UML metamodel class diagram evolution from the deterministic work of [17,18]; thus, we were not involved in the selection of models and metamodel changes. To ensure a fair comparison with Wimmer et al. [17], we only compared both approaches to the existing UML dataset. Table 2 describes the statistics related to the collected data.

Table 2. Statistics related to the collected data of the investigated cases.

Metamodels				**Models**		
Release	#of elements	#of changes	%of changes	#of models	#of model elements (Min,Max)	#of expected edit operations (Min, Max)
GMF Gen 1.41 to 1.90	From 885 to 1120	347	31%	8	389, 744	39, 70
GMF Gen 1.90 to 1.248	From 1120 to 1216	362	27%	8	433, 686	66, 83
GMF Map 1.45 to 1.52	From 382 to 413	62	15%	8	203, 394	46, 69
GMF Map 1.52 to 1.58	From 413 to 428	10	1.8%	8	347, 402	57, 81
GMF Graph 1.25 to 1.29	From 278 to 279	14	5%	8	142, 283	34, 55
GMF Graph 1.25 to 1.33	From 279 to 281	42	14%	8	149, 301	29, 43
UML CD [17]	From 23 to 29	8	8%	10	28, 49	11, 23

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. The 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 used MC measured by the designers. It consists of the number of relevant edit operations identified by the designer over the total number of edit operations in the selected solutions. In addition, we report the number of interactions (NI) required on the Pareto front comparing to the one required once the mono-objective search is executed. This evaluation will help to understand if we efficiently extracted the developer preferences after the Pareto-front interactions. We decided to limit the comparison to only the interactive multi-objective work of Kessentini et al. [11] since it is the only approach offering interaction with the user, and it will help us understand the real impact of the knowledge extraction and mono-objective features (not supported by existing studies) on the recommendation and interaction effort. We also report the computation time (T) for the different evolution scenarios to estimate the effort required to obtain the best co-evolution solutions.

Study Participants. Our study involved 16 master students in Software Engineering. All the participants are volunteers and familiar with model-driven engineering and co-evolution/refactoring since they are part of a graduate course on Software Testing & Quality Assurance and most of them participated in similar experiments in the past, either as part of a research project or during graduate courses. Furthermore, 12 out of the 16 students are working as full-time or part-time developers in the software industry. Participants were first asked to fill out a pre-study questionnaire containing five questions. The questionnaire helped to collect background information such as their role within the company, their modeling experience, and their familiarity with model-driven engineering and co-evolution/refactoring. In addition, all the participants attended two lectures about model transformations and evolution, and passed six tests to evaluate their performance in evaluate and suggest model evolution solutions. We formed 4 groups, each composed of 4 participants. The groups were formed based on the pre-study questionnaire and the test results to ensure that all the groups have almost the same average skill level. We divided the participants into groups according to the studied metamodels, the techniques to be tested, and the developers' experience. The participants were asked to co-evolve the different models manually and evaluate the results of the different approaches based on a counter-balanced design [23].

Statistical Tests. 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 [24] 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 $\alpha(> 0.05)$ means the opposite. 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.

Parameter Settings. The stopping criterion was set to 100,000 evaluations for all search algorithms to ensure fairness of comparison (without counting the number of interactions since it is part of the users' decision to reach the best solution based on his/her preferences). The mono-objective search was limited to 10,000 evaluations after the interactions with the user. The other parameters' values are as follows for both the multi-objective and mono-objective algorithms: crossover probability = 0.4; mutation probability = 0.7, where the probability of gene modification is 0.5. Each parameter has been uniformly discrete in some intervals. Values from each interval have been tested for our application. Finally, we pick the best values for all parameters. Hence, a reasonable set of parameters values have been experimented.

4.2 Results

Results for RQ1: Search Validation. Figure 6 confirms that using NSGA-II produces results by far better (and statistically significant) than just randomly exploring a comparable number of solutions based on the three different metrics of precision, recall, and manual correctness on all the different evolution 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 Fig. 6 ($\sim$96% vs. $\sim$42%). The difference in execution time in favor of random search (Table 3), due to the crossover and mutation operators, is largely compensated by the quality of the obtained results.

RS is not efficient in generating 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 metaheuristic search is justified (this answers RQ1).

Results for RQ2: Benefits. We report the results of the empirical evaluation in Fig. 6. The majority of the co-evolution solutions recommended by our approach were correct and validated by the participants on the different case

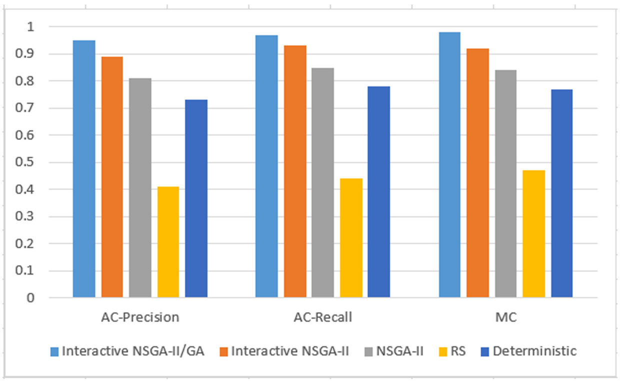


Fig. 6. The median evaluation scores on the four metamodel evolution scenarios with 95% confidence level ($\alpha = 5\%$)

Table 3. Median execution time, in minutes, on the different metamodel/model co-evolution scenarios and the number of interaction proposed by both interactive approaches

	Approaches						
	IMMO		I-NSGA-II		NSGA-II	RS	Deterministic
Metamodels	T	NI	T	NI	T	T	T
GMF Gen	44	18	71	32	38	30	31
GMF Map	30	16	55	25	28	22	17
GMF Graph	25	24	83	35	21	18	14
Class Diagram	20	8	39	5	16	14	12

studies. On average, for all of our four studied metamodels/models, our approach was able to recommend 96% of generated edit operations correctly. The remaining approaches have an average of 89% and 81%, respectively, for the interactive multi-objective approach [11] and the fully automated multi-objective approach [10]. Both of the interactive tools outperformed fully-automated ones, which shows the importance of integrating the human in the loop when co-evolving models. The deterministic approach defines generic rules for a set of possible metamodel changes that are applied to the co-evolved models. Figure 6 shows that our approach clearly outperforms, on average, the deterministic technique based on all measures: precision, recall, and manual correctness.

Results for RQ3: The relevance of Designers' Preferences Extraction. Table 3 summarizes the time, in minutes, and the number of interaction (for the interactive approaches) with the participants to find the most relevant solutions using our tool (IMMO), the interactive approach (I-NSGA-II) [11], the automated approach [10], Random search and the deterministic approach [17]. All the participants spent less time finding the most relevant model edit operations on the different metamodels than I-NSGA-II. For instance, the average time is reduced from 71 min to just 44 min for the case of GMF Gen. The time includes the execution of IMMO and the different phases of interaction until the designer is satisfied with a specific solution. It is clear as well that the time reduction is not correlated with the number of interaction. For instance, the deviation between IMMO and I-NSGA-II for GMF Graph in terms of the number of interactions is limited to 9 (24 vs. 35), but the time reduction is 58 min. The time includes the execution of the multi-objective and mono-objective search (if any) and the different phases of interaction until the designer is satisfied with a specific solution. The drop of the execution time is mainly explained by the fast execution of the mono-objective search and the reduced search space after the interactions with the designer.

It is clear that our approach reduced as well as the number of interaction comparing to I-NSGA-II. IMMO required much fewer designer interactions. For instance, only 16 interactions to modify, reject, and select solutions were observed on GMF Map using our approach, while 25 interactions were needed for I-NSGA-II. The reductions of the number of interactions are mainly due to the move from multi-objective to mono-objective search after one round of interactions since the designers will not deal anymore with a set of solutions in the front but only one.

5 Related Work

In one of the early works [25], the co-evolution of models is tackled by designing co-evolution transformations based on metamodel change types. In [7,8], 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, i.e., a transformation that takes/produces another transformation as input/output. In [26], the authors proposed an approach that compromises multiple steps for model co-evolution: change detection 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. 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 [27] and in the framework described in [28].

A comprehensive survey of interactive SBSE approaches can be found in [29]. The problems of contextualization to developer's regions of interest during the recommendation process have been treated in recent SBSE papers for the code refactoring problem [30–34]. Han et al. proposed in [32] an approach to enable the interactions with the user, then a Delta Table can select the next refactoring quickly to improve a specific objective without calculating a fitness function.

6 Conclusion

In this paper, we proposed a novel approach to extract designers' preferences to find good recommendations to co-evolve models. We combined the use of multi-objective search, mono-objective search, and user interaction in our approach. To evaluate the effectiveness of our tool, we conducted an evaluation with 16 participants who evaluated the tool and compared it with the state-of-the-art techniques. As part of our future work, we are planning to evaluate our approach on further metamodel evolution cases and a more extensive set of participants. We will also adapt our approach to address other problems requiring designers' interactions, such as model transformation rules.

References

1. Iovino, L., Pierantonio, A., Malavolta, I.: On the impact significance of metamodel evolution in MDE. J. Object Technol. (2012)
2. Hebig, R., Khelladi, D.E., Bendraou, R.: Approaches to co-evolution of metamodels and models: a survey. IEEE Trans. Softw. Eng. **43**(5), 396–414 (2017)
3. Schoenboeck, J., et al.: CARE: a constraint-based approach for re-establishing conformance-relationships. In: Proceedings of APCCM (2014)
4. Meyers, B., Wimmer, M., Cicchetti, A., Sprinkle, J.: A generic in-place transformation-based approach to structured model co-evolution. In: Proceedings of MPM Workshop (2010)
5. Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Sci. Comput. Program. **76**(12), 1223–1246 (2011)
6. Cicchetti, A., Ciccozzi, F., Leveque, T., Pierantonio, A.: On the concurrent versioning of metamodels and models: challenges and possible solutions. In: Proceedings IWMCP (2011)
7. Garcés, K., Jouault, F., Cointe, P., Bézivin, J.: Managing model adaptation by precise detection of metamodel changes. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 34–49. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02674-4_4
8. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Proceedings of EDOC (2008)
9. Kessentini, W., Sahraoui, H., Wimmer, M.: Automated metamodel/model co-evolution using a multi-objective optimization approach. In: Wąsowski, A., Lönn, H. (eds.) ECMFA 2016. LNCS, vol. 9764, pp. 138–155. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42061-5_9
10. Kessentini, W., Sahraoui, H.A., Wimmer, M.: Automated metamodel/model co-evolution: a search-based approach. Inf. Softw. Technol. **106**, 49–67 (2019)
11. Kessentini, W., Wimmer, M., Sahraoui, H.A.: Integrating the designer in-the-loop for metamodel/model co-evolution via interactive computational search. In: Wasowski, A., Paige, R.F., Haugen, Ø. (eds.) Proceedings of MODELS, pp. 101–111 (2018)
12. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer, M., et al. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 849–858. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_83

13. Immo. https://sites.google.com/view/ssbse2020/
14. Herrmannsdoerfer, M., Vermolen, S.D., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 163–182. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19440-5_10
15. Richters, M.: A precise approach to validating UML models and OCL constraints. Technical report (2001)
16. Muflikhah, L., Baharudin, B.: Document clustering using concept space and cosine similarity measurement. In: Proceedings of ICCTD (2009)
17. Wimmer, M., Kusel, A., Schoenboeck, J., Retschitzegger, W., Schwinger, W.: On using inplace transformations for model co-evolution. In: Proceedings of MtATL Workshop (2010)
18. Cicchetti, A., Ciccozzi, F., Leveque, T., Pierantonio, A.: On the concurrent versioning of metamodels and models: challenges and possible solutions. In: Proceedings of IWMCP (2011)
19. Herrmannsdoerfer, M., Ratiu, D., Wachsmuth, G.: Language evolution in practice: the history of GMF. In: van den Brand, M., Gašević, D., Gray, J. (eds.) SLE 2009. LNCS, vol. 5969, pp. 3–22. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12107-4_3
20. Herrmannsdoerfer, M.: GMF: a model migration case for the transformation tool contest. In: Proceedings of TTC (2011)
21. Rose, L.M., et al.: Graph and model transformation tools for model migration - empirical results from the transformation tool contest. SoSym **13**(1), 323–359 (2014)
22. Di Ruscio, D., Lämmel, R., Pierantonio, A.: Automated co-evolution of GMF editor models. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 143–162. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19440-5_9
23. Pollatsek, A., Well, A.D.: On the use of counterbalanced designs in cognitive research: a suggestion for a better and more powerful analysis. J. Exp. Psychol. Learn. Mem. Cogn. **21**(3), 785 (1995)
24. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of ICSE (2011)
25. Sprinkle, J., Karsai, G.: A domain-specific visual language for domain model evolution. J. Vis. Lang. Comput. **15**(3–4), 291–307 (2004)
26. Gruschko, B.: Towards synchronizing models with evolving metamodels. In: Proceedings of MoDSE Workshop (2007)
27. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Model migration with epsilon flock. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 184–198. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13688-7_13
28. Narayanan, A., Levendovszky, T., Balasubramanian, D., Karsai, G.: Automatic domain model migration to manage metamodel evolution. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 706–711. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04425-0_57
29. Ramirez, A., Romero, J.R., Simons, C.L.: A systematic review of interaction in search-based software engineering. IEEE Trans. Softw. Eng. **45**(8), 760–781 (2018)
30. Morales, R., Chicano, F., Khomh, F., Antoniol, G.: Efficient refactoring scheduling based on partial order reduction. J. Syst. Softw. **145**, 25–51 (2018)

31. Morales, R., Soh, Z., Khomh, F., Antoniol, G., Chicano, F.: On the use of developers' context for automatic refactoring of software anti-patterns. J. Syst. Softw. **128**, 236–251 (2017)
32. Han, A.R., Bae, D.H., Cha, S.: An efficient approach to identify multiple and independent move method refactoring candidates. IST **59**, 53–66 (2015)
33. Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., Ouni, A.: A cooperative parallel search-based software engineering approach for code-smells detection. TSE **40**(9), 841–861 (2014)
34. Alizadeh, V., Fehri, H., Kessentini, M.: Less is more: from multi-objective to mono-objective refactoring via developer's knowledge extraction. In: SCAM, pp. 181–192. IEEE (2019)