Software and Systems Modeling
https://doi.org/10.1007/s10270-022-00978-2

SPECIAL SECTION PAPER

Semi-automated metamodel/model co-evolution: a multi-level interactive approach

Wael Kessentini[1] · **Vahid Alizadeh**[1]

Received: 11 March 2021 / Revised: 25 December 2021 / Accepted: 18 January 2022

Abstract
Metamodels evolve even more frequently than programming languages. This evolution process may result in a large number of instance models that are no longer conforming to the revised metamodel. On the one hand, the manual adaptation of models after the metamodels' evolution can be tedious, error-prone, and time-consuming. On the other hand, the automated co-evolution of metamodels/models is challenging, especially when new semantics is introduced to the metamodels. While some interactive techniques have been proposed, designers still need to explore a large number of possible revised models, which makes the interaction time-consuming. Existing interactive tools are limited to interactions with the designers to evaluate the impact of the co-evolved models on different objectives of the number of inconsistencies, number of changes and the deviation from the initial models. However, designers are also interested to check the impact of introduced changes on the decision space which is composed by model elements. These interactions help designers to understand the differences of the co-evolved models solution that have similar objectives value to select the best one based on their preferences. In this paper, we propose an interactive approach that enables designers to select their preference simultaneously in the objective and decision spaces. Designers may be interested in looking at co-evolution operations that can improve a specific objective such as number of non-conformities with the revised metamodel (objective space), but such operations may be related to different model locations (decision space). A set of co-evolution solutions is generated at first using multi-objective search that suggests edit operations to designers based on three objectives: minimizing the deviation with the initial model, the number of non-conformities with the revised metamodel and the number of changes. Then, the approach proposes to the user few regions of interest by clustering the set of recommended co-evolution solutions of the multi-objective search. Also, another clustering algorithm is applied within each cluster of the objective space to identify solutions related to different model element locations. The objective and decision spaces can now be explored more efficiently by the designers, who can quickly select their preferred cluster and give feedback on a smaller number of solutions by eliminating similar ones. This feedback is then used to guide the search for the next iterations if the user is still not satisfied. We evaluated our approach on a set of metamodel/model co-evolution case studies and compared it to existing fully automated and interactive co-evolution techniques.

Keywords Metamodel/model co-evolution · Interactive multi-objective search · Search-based software engineering

1 Introduction

There is an urgent need to find better ways to evolve software systems and, consequently, improve developers' productivity. Like source code, design is subject to evolution due to changing requirements and technological constraints. The evolution of metamodels, models, and transformation rules is inevitable in model-driven engineering (MDE) [7]. The changes can be related to the design of software systems, from initial development to maintenance. When metamodels evolve, the instantiated models need to be updated to

Communicated by S. Abrahao, E. Syriani, H. Sahraoui, and J. de Lara.

✉ Wael Kessentini
wkessent@depaul.edu

Vahid Alizadeh
v.alizadeh@depaul.edu

[1] College of Computing and Digital Media, DePaul University, 243 South Wabash Ave., Chicago, IL, USA

Published online: 06 April 2022

make them conform to the new metamodel version. Thus, a set of change operations 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 [34,72].

Several co-evolution studies are proposed; most of them are providing either a manual or semi-automated support based on pre-defined templates of evolution scenarios [10,12, 19,52,53]. In addition to being pre-defined, these templates are specific to the artifact to co-evolve with the metamodel. Few fully automated co-evolution studies try to find an entire edit operations sequence that revises models in accordance with the new metamodel version [38,39,72]. Understanding metamodel changes and their impact on the models is challenging as one metamodel change can impact a large number of model elements. Thus, it is impossible to manually fix all the violations without the help of at least a semi-automated support.

Several techniques proposed to translate metamodel chan ges into model level edit operations using a set of generic transformation rules [28,30,32,76]. However, several transformations require interactions with the user, especially when a lot of elements are changed in the new meta-model.

In our recent work, we proposed an approach to interactively evaluate the co-evolved models using search-based software engineering [37]. The designers can provide feedback about the co-evolved models and may 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. The study showed that even the clustering of non-dominated co-evolution solutions based on objectives will still generate a considerable number of co-evolution solutions to explore. Indeed, designers in practice want to co-evolve the model based on objectives and also the model elements to target (decision space) when deciding which edit operations to apply. However, existing co-evolution tools do not consider the interactive exploration of both objectives and model elements to co-evolve during the co-evolution process.

In this paper, we extend our previous work by proposing an interactive approach that combines multi-objective search (NSGA-II [13]), interactive optimization, and unsupervised learning to reduce the designer's interaction effort in exploring both objective and decision spaces when co-evolving models. Indeed, we followed the feedback collected from the participants of our previous work [37] used in our validation to define the scope of this extension and also the current state of the art. They liked the mix between expressing some preferences via the fitness function and providing feedback when exploring the co-evolution solutions and their impact. The main feedback received was to provide better support to understand co-evolution operations' impact and preferences not only based on the fitness functions but also based on the distributions of these operations in the model/meta-models. For instance, designers express reluctance to make major changes to the models if there are opportunities to make them conform with the new metamodel with a minimum set of co-evolution operations. These types of preferences are hard to define upfront based on the participants' feedback and they need to be balanced with the potential gains.

We generate, first, using multi-objective search, different possible sets of edit operations by finding the edit operation sequences that minimize the number of conformance errors, the deviation with the initial model (reduce the loss of information) and the number of proposed edit operations. After a number of iterations, a near-optimal set of solutions (Pareto front) are generated to the user representing potential sets of edit operations that co-evolve a model to the evolved metamodels. However, the Pareto front of possible solutions can be large. Therefore, it is essential to provide designers with additional support for managing and understanding this set. Thus, an unsupervised learning algorithm clusters the solutions into different categories based on the objectives. Finally, another clustering algorithm is applied within each cluster of the objective space to help designers explore the impact of the recommended edit operations while choosing the model element to co-evolve. The input for the second clustering is generated from the first clustering step; hence, both algorithms are hierarchical. In other words, the designer can interact with our tool by exploring both the decision and objective spaces to identify relevant edit operations based on their preferences quickly. Thus, the developers can focus on their regions of interest in both the objective and decision spaces. The designers are in general concerned about improving specific objectives, and then they will look for the edit operations that target the model elements of their interests. Therefore, we followed this pattern in our approach by clustering first the objective space and then we showed the designers the distribution of the co-evolution solutions into different decision space clusters for their preferred objective space cluster.

The feedback from the designers, both at the cluster and solution levels, are used to automatically generate constraints to reduce the search space in the next iterations and focus on the region of designer's preferences/interest. For instance, the designer can select the most relevant cluster of solutions, called region of interests, based on his/her preferences and then the multi-objective search will reduce the space of possible solutions, in the next iterations, by generating constraints from the interaction data such as eliminating part of the model elements that are not relevant for the co-evolution. We note that non-breaking metamodel/model changes are outside of the scope of your approach.

We selected 20 participants to manually evaluate the effectiveness of our tool on a set of three Ecore metamodels from the Graphical Modeling Framework (GMF) and a well-known evolution case of the UML metamodel for Class Diagrams extracted from [11,77]. Furthermore, we compared our approach to existing fully automated co-evolution techniques [38,39,77], an interactive clustering approach without considering the decision space [37], and an interactive approach without clustering the solutions [40]. The manual evaluation of the revised models to meet new metamodel changes confirms the effectiveness of our clustering-based interactive approach. Since the execution of the two clustering algorithms is hierarchical, the final results are actually the combination of two clustering steps. We recorded in our tool all the interactions with the user and we found that all participants used both the objective and decision space clusters before selecting a final solution. In the post-study feedback, participants emphasized that both the decision and objective space interactions helped them to find a relevant co-evolution solution. The common pattern was to establish their goals from co-evolving the models to ensure conformance with the new metamodels and then they used the decision space to find a solution that matched their context (e.g., model elements to change).

The main contributions of this paper can be summarized as follows:

1. Our paper focuses on the interactive exploration of the objective and decision spaces while existing work focus only on either the objective space or the decision space and they often lack user interaction in the decision space. Our approach is not about a simple filtering of the edit operations based on the model/metamodel elements locations or the clustering of the Pareto front based on the locations. We enabled designers to interactively navigate between both objective and decision spaces to understand how the edit operations are distributed if they are interested to improve specific objectives. Then, our approach can generate even more relevant suggestions after extracting that knowledge from the exploration of the Pareto front.
2. Our contribution is beyond the adoption of an existing metaheuristic technique to co-evolve models. The proposed approach includes an algorithm to enable the exploration of both decision and objective spaces by combining two level of clustering algorithms with multi-objective search.
3. We implemented and validated our tool using a variety of metamodels, and we compared it to the current state of the art of co-evolution tools . The results support the hypothesis that the combination of both the objective and decision spaces significantly improved the edit operation recommendations to co-evolve models. The online appendix related to this paper can be found in [1]

The remainder of this paper is structured as follows. Sect. 2 provides the background of metamodel/model co-evolution and presents a motivating example. Sect. 3 describes our approach, while the results obtained from our experiments are presented and discussed in Sect. 4. Threats to validity are discussed in Sect. 5. After surveying the related work in Sect. 6, a conclusion with an outlook on future work is provided in Sect. 7.

2 Background and motivating example

2.1 Metamodel/model co-evolution by a motivating example

In MDE, metamodels are the means to specify the abstract syntax of modeling languages [7]. Metamodels are instantiated to produce models which are, in essence, object graphs, i.e., consisting of objects (instances of classes) representing the modeling elements, object slots for storing values (instances of attributes), and links between objects (instances of references). The object graphs are often represented as UML object diagrams and have to conform to the UML class diagram describing the metamodel. This means, for a model to conform to its metamodel, a set of constraints have to be fulfilled. This set of constraints is normally referred to as *conformsTo* relationship [34,72].

Figure 1 shows an example of a simplified metamodel evolution, based on simple staff modeling language taken from [69] and a model conform to the initial metamodel version. 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 constraint violations, the constraint shown in Listing 1 is violated when considering the initial model shown in Fig. 1c and its conformance to the new metamodel version in Fig. 1b.

To re-establish conformance for the given example, let us assume that only two edit operation types on models are used. 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.

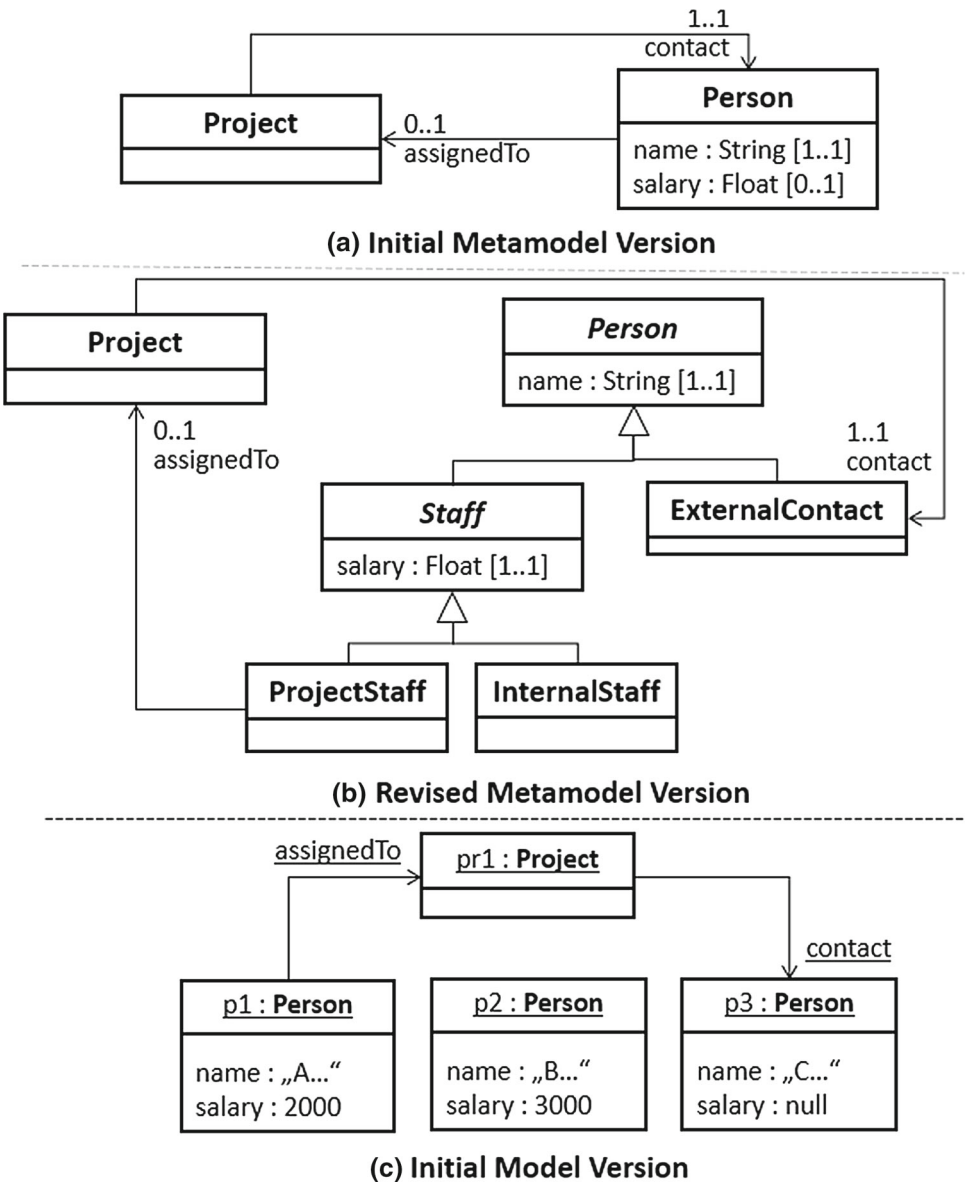

Fig. 1 Example of metamodel evolution

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)
```

Several co-evolution studies proposed to revise models after metamodels evolution from manual to fully automated approaches [26]. Recently, few automated/interactive tools [38–40] have 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 conformant as possible to the new meta-model 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. 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. 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. However, these tools suffer from several limitations.

First, they lack flexibility since the designer has to inspect a large list of potential solutions, which is time consuming, and s/he may miss to select the best ones. Second, designers always have a concern on expressing their preferences upfront as an input for a tool to guide the search for co-evolved models suggestions. They prefer to get insights from some generated co-evolution solutions then decide which ones want to improve. Third, the users may spend consider-

able time to understand the differences between the solutions and their impacts on the different co-evolution objectives.

Figure 3a shows an example of a large number of equally good solutions in terms of objectives (represented in points) where the designer has to decide which solution to select, and which additional changes to apply to the proposed solutions generated based only on the multi-objective search (similar to existing approaches). Figure 2 shows modified models after applying the set of edit operations of the proposed solutions. The recommended models are represented in a shape of stars in Fig. 3. This figure shows that there are several possible solutions that may contain inconsistencies with the new metamodel version or dissimilarities from the initial model, and the user has to decide which one to select based on his/her preferences. Thus, there is a need to reduce the search space in the next iterations and reduce the interactions effort using the feedback from the user. While designers were interested in giving feedback for some co-evolution solutions, they still find the interaction process time-consuming [37]. Figure 3b shows that even when the co-evolution solutions are clustered based on the objectives, the number of solutions to be checked by designers can be substantial. Thus, they want to know how different the solutions are within the same objective space. It may be possible to find more than one co-evolution solution that offers the same level of objective improvements but co-evolves different model elements. Thus, the objective and decision spaces clustering are necessary. Existing co-evolution techniques do not enable designer interaction based on both the decision space and objective space, that is the main challenge of this paper.

2.2 Multi-objective algorithms

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 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 x_1 is said to dominate another solution x_2, if x_1 is no worse than x_2 in all objectives and x_1 is strictly better than x_2 in at least one objective". Formally, if we consider a set of objectives f_i , $i \in 1..n$, to maximize, a solution x_1 dominates x_2.

$$\forall i,\ f_i(x_2) \leq f_i(x_1) \text{ and } \exists j \mid f_j(x_2) < f_j(x_1)$$

"

Definition 4 (Pareto optimal set). For a MOP $f(x)$, A solution x' is said to dominate another solution x (denoted by f(x') $\preceq$ f(x)) and the Pareto optimal set is $P^* = \left\{x \in \Omega | \neg \exists x' \in \Omega,\ f(x') \preceq f(x)\right\}$.

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

2.2.1 Non-Sorting Genetic Algorithm II (NSGA-II)

NSGA-II is among the widely-used algorithms to address real-world problems involving conflicting objectives [13].

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. The *crossover* operator is responsible for creating new solutions based on already existing ones, e.g., re-combining solutions. The *mutation* operator is used to introduce slight random changes into candidate co-evolution solutions. This operator guides the algorithm into areas of the search space that would not be reachable through recombination alone and avoids the convergence of the population towards a few elite solutions.

Both populations of the offspring and parent have the same size. Then, the merged population (parents and children) is ranked into several non-dominance layers, called fronts.

As described in Fig. 4, 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 until no non-dominated solutions remain. After assigning solutions to fronts, each solution is assigned a diversity score, called crowding distance, which ranks the solutions inside each front. This distance aims, later, to favor diverse solutions in terms of objective values. A solution is then characterized by its front and its crowding distance inside the front. To finish an iteration of the evolution, NSGA-II performs the environmental selection to form the parent population for the next generation by picking half of the solutions. The solutions are included iteratively from the Pareto front to the lowest layers. If half of the population is reached inside a front, then the crowding distance is used to discriminate between the solutions. In the example of Fig. 4, 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

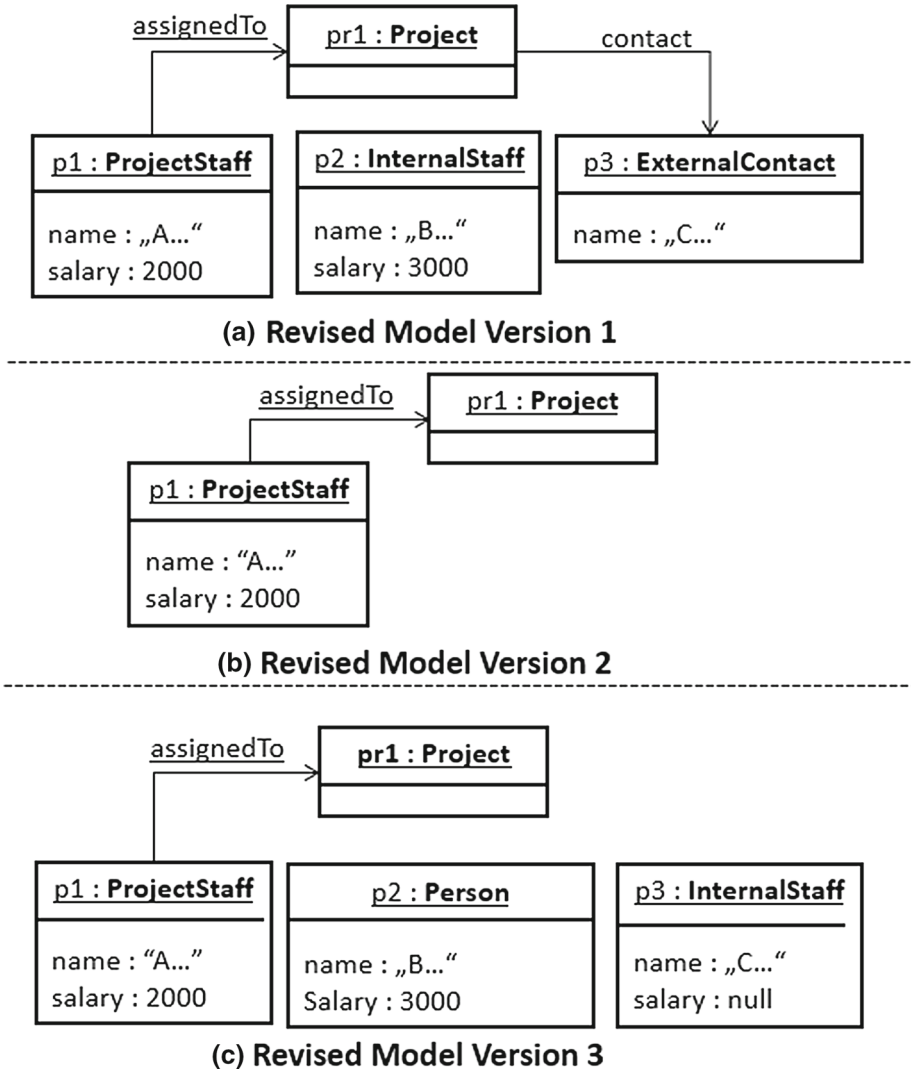

Fig. 2 Example model co-evolution

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 toward the near-optimal solution, while the crowding ranking emphasizes diversity.

3 Proposed approach

The general structure of our approach is sketched in Fig. 5. Our approach includes four main components. The first component is the multi-objective algorithm, NSGA-II, executed for a number of iterations to generate a diverse set of non-dominated co-evolution solutions called Pareto-optimal solutions [13], 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 of 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 designers with additional support for understanding and managing this set of solutions. The goal of the second phase is to cluster the solutions based on their objective functions and the similarity among them. Then, a representative solution is identified from each cluster to present it to the user. The third step takes, as input, the identified cluster(s) from the user's choice in the objective space and execute decision space clustering algorithm to cluster the co-evolution solutions based on their model elements locations.

The last phase is to manage the interaction with the user where s/he can visualize the clusters of solutions and the representative solution of each cluster in both the objective and decision spaces. For instance, designers may select a cluster from the objective space clustering that meets their objective preferences. Then the second clustering will show

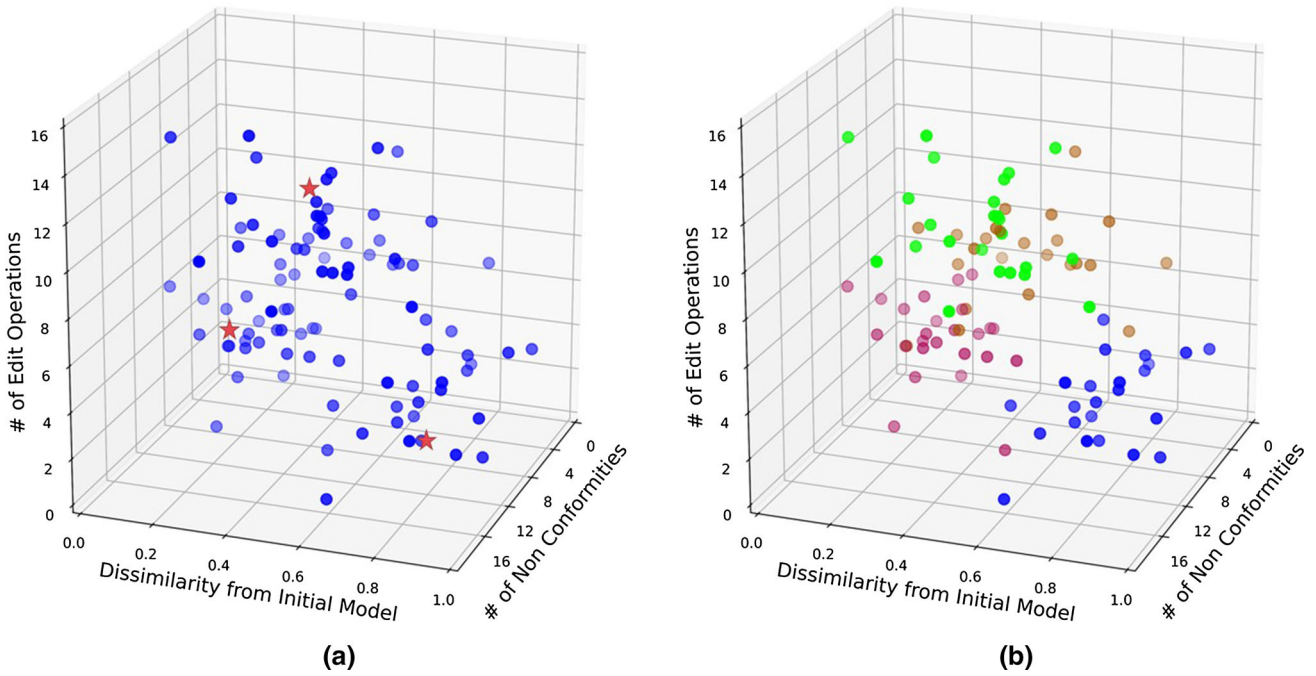

Fig. 3 Pareto front of co-evolution solutions generated using: (a) the multi-objective search only, (b) the clustering of the objective space

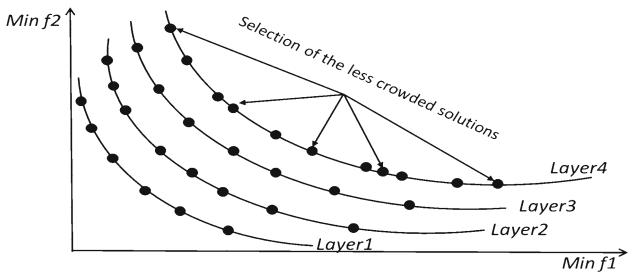

Fig. 4 NSGA-II selection mechanism for a two-objective problem

them how the co-evolution solutions in the preferred objective space cluster are different in the decision space. The user can easily avoid looking at many solutions that are similar in the decision space (modifying almost the same model elements). The user can interact with the tool at the solution level, by accepting or rejecting or modifying suggested edit operations, or the cluster level, by specifying a cluster as a region of interest. Thus, the goal is to guide, *implicitly*, the exploration of the Pareto front to find good co-evolution recommendations. We extract the user preferences from these activities to consider them in the next round of iterations to converge towards to user's region of interest. This loop will continue until the user is satisfied and a set of edit operation is chosen to apply to the model to revise.

In the following, we describe the different main components of our approach.

3.1 Phase 1: Multi-objective formulation

The process starts with exploring the search space to find non-dominated solutions.

To explore this search space, we propose an adaptation of the non-dominated sorting genetic algorithm (NSGA-II) to interactively find a trade-off between three objectives that will be described later.

The first iteration of the process begins with a complete execution of adapted NSGA-II to our model co-evolution recommendation problem based on the fitness functions that will be discussed later. At the beginning, a random population of encoded edit operation solutions, P_0, is generated as the initial parent population. Then, the children population, Q_0, is created from the initial population using crossover and mutation. Parent and children populations are combined together to form R_0. Finally, a subset of solutions is selected from R_0 based on the crowding distance and domination rules. This selection is based on elitism which means keeping the best solutions from the parent and child population. Elitism does not allow an already discovered non-dominated solution to be removed. This process is continued until the stopping criteria is satisfied.

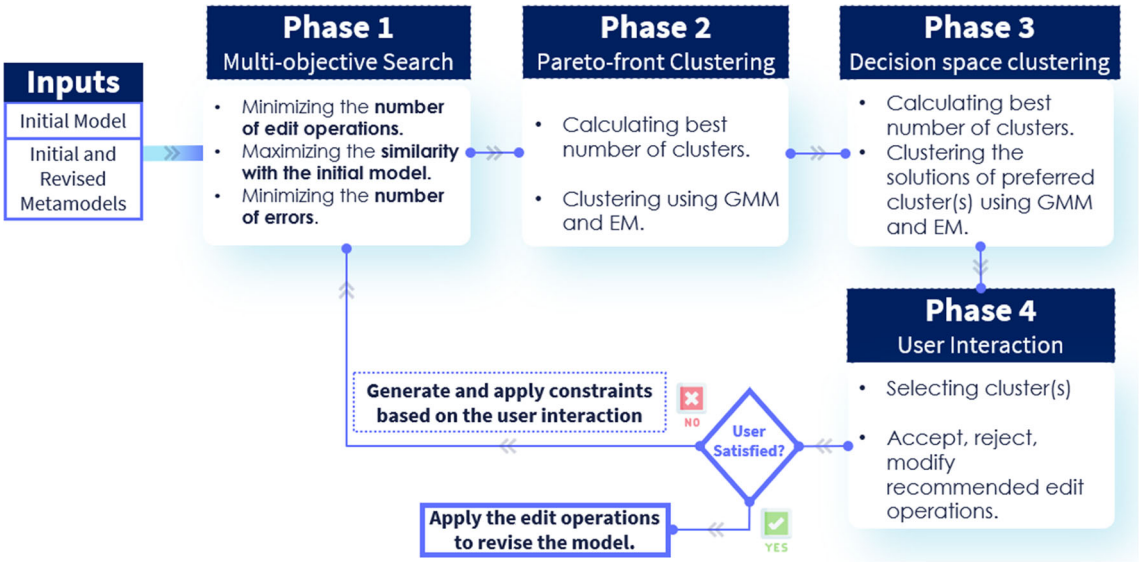

Fig. 5 High-level overview of the proposed decision and objective space interactive clustering-based multi-objective model co-evolution approach (DOIC-NSGA-II)

The results of the first execution of search algorithm are a set of non-dominated solutions that will be clustered and then updated by the users. After this interactions phase, the multi-objective search algorithm will continue to run using the new constraints generated at the cluster and solution levels.

3.1.1 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., 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 [31]. 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 6 represents a solution that can be applied to the initial model of our motivating example described in Sect. 2.

3.1.2 Variation Operators

Variation operators help to navigate through the search space and to maintain a good diversity in the population. There are two variation operators used in the optimization algorithm known as crossover, and mutation.

- **Crossover**: The process of combining parents in order to generate new off-springs is called parent crossover. We utilized "Single Point Crossover" operator for this mean. In this operator, a random crossover point is chosen and then the two sides of the parents are swapped to produce new children.
- **Mutation**: A small random modification in solution individual is named mutation. This process aid to keep diversity in the population. However, by assigning a low probability to this operator, we avoid a random search. We employed "Bit Flip Mutation" with which a random edit operation is selected and replaced with another randomly selected available edit operation. When a mutation operator is applied, the goal is to slightly change the solution for the purpose to probably improve its fitness functions.

3.1.3 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 apply-

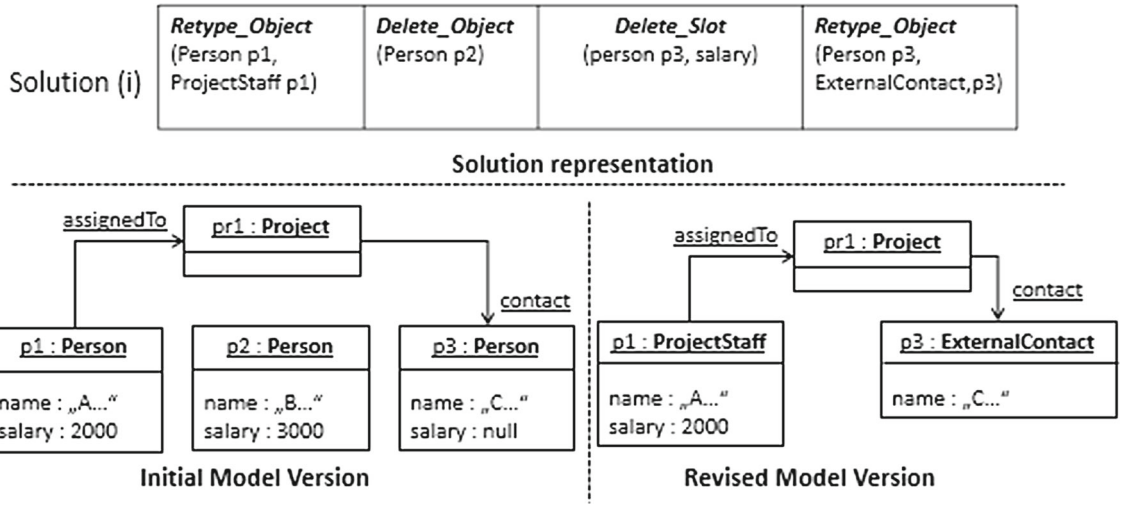

Fig. 6 Solution representation

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.

ing 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 [64]: 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. [72] and Richters et al. [64] 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.

The second fitness function $nbOp(s)$ aims at minimizing the changes to the initial models. The "complex" edit operations of Table 1 are combinations of atomic/primitives operations. Thus, we decomposed the complex edit operations into atomic ones (mostly add/remove operations) following the work in [31] to count the number of operations

$nbOp(s)$ of a solution s (size of s) to ensure a fair normalization. Since the approach generates the solution (the set of edit operations) randomly during part of the NSGA-II process, some of the edit operations can be irrelevant while not generating any errors. For example, a solution that creates and deletes the same element multiple times. Thus, the minimization model edits can help in removing redundancies and irrelevant operations. The goal is to find for the designer a solution with the minimum number of constraints violations, the minimum number of changes and as much as possible similar to the initial model.

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 the textual similarity using the Cosine similarity [56]. However, identical identifier names are matched automatically first before even making approximations based on the following similarity heuristics. 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:

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

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.

To summarize, we defined first the cosine measure that can estimate the similarity between two identifiers. Since an identifier in an initial model can be similar to more than one identifier, we calculate the cosine similarity between the identifier of the initial model with all the identifiers of the revised model then we match the identifier of the initial model to the most similar one in the revised model. Since multiple changes could happen at the same time to multiple identifiers, we have to calculate the maximum cosine similarity score between the names to identify the correct matching.

3.2 Phase 2: Objective space clustering

The goal of this phase is to reduce the effort to investigate the solutions in Pareto-optimal front. This phase tries to group the solutions based on their fitness function values without filtering or removing any of them. In this way, the solutions can be categorized based on the similarity among them in the objectives space. Then, a representative solution is identified from each partition to recommend to the decision maker (center of the cluster). For this purpose, we used clustering analysis technique. Clustering is one of the most important and popular unsupervised learning problems in Machine Learning. It helps to find a structure in a set of unlabelled data in a way that the data in each cluster are similar together, while they are dissimilar to the data in other clusters.

One of the challenges in cluster analysis is to define the optimal number of clusters. Therefore, we need cluster validity index as a measure of clustering performance. Different partitions are computed and the ones that fit the data better are selected. The procedure of Phase 2 is illustrated in Algorithm 1.

3.2.1 Calinski Harabasz (CH) Index

CH Index is an internal clustering validation measure based on two criteria: compactness and separation [9,62]. CH evaluates the clustering results based on the average sum of squares between and within clusters and it defines as follows:

$$CH = \frac{(N-K)}{(K-1)} \frac{\Sigma_{k=1}^{K} |c_k| \; dist(\overline{c_k}, \overline{S})}{\Sigma_{k=1}^{K} \Sigma_{s_i \in c_k} \; dist(s_i, \overline{c_k})} \quad (1)$$

Algorithm 1: Pareto-front Clustering

```
Input  : Pareto-front solutions (S)
Output: Labeled solutions (LS),
        Clusters Representative Solution (CR)
1  begin Calculate best number of clusters-K
2      for i ← 2 to Numberof Clusters do
3          LS = GMMClustering (i, S);
4          Score_i =CalinskiHarabaszIndex(LS);
5      K ← MaxScoreIdx();
6  begin GMMClustering (K,S)
7      μ_k, Σ_k, π_k ← Initialize-K-Gaussian();
       /* Expectation-Maximization          */
8      while ¬ converge do
9          γ(s_nk) ← Expectation();
10         μ_k, Σ_k, π_k ← Maximization();
11         EvaluateLikelihood();
12     foreach s_n ∈ S do
           /* assigning cluster labels      */
13         L_n ← MaxResponsibilityIdx(s_n);
       /* Find Clusters Representative      */
14     foreach Cluster C_k do
15         CR_k ← MaxDensity(s_nk ∈ C_k)
16 Return LS, CR;
```

where N is the size of data, K is the number of clusters, $dist(a, b)$ is the Euclidean distance, and $\overline{c_k}$ and $\overline{S}$ are the cluster and global centroids, respectively. The first step in Pareto-front clustering is to execute the clustering process with different number of components and to compute CH score for each. The best number of clusters (K) is defined as the one that achieves the highest CH score.

3.2.2 Gaussian mixture model (GMM)

GMM is a probabilistic model-based clustering algorithm with which a mixture of k Gaussian distributions is fitted on the data. GMM is soft-clustering approach in which each data point is assigned a degree that it belongs to each of the clusters. The parameters that need to fit are Mean (μ_k), Co-variance (Σ_k), and Mixing coefficient (π_k).
GMM clustering begins by random initiation of parameters for K components. Then, Expectation-Maximization (EM) algorithm [63] is employed for parameter estimation. EM is an iterative process to train the parameters and has two steps. In the expectation step, an assignment score to each Gaussian distribution, called "responsibility" or "membership weight", is determined for each solution point as follow:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(s_n|\mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(s_n|\mu_i, \Sigma_i)} \tag{2}$$

The responsibility coefficient will be used later for preference extraction step. In the maximization step, the parameters of each Gaussian are updated using the computed responsibility coefficients.

3.3 Phase 3: Decision space clustering

Our tool gives designers the ability to select their preferences in a different space than the optimization space related to the location of model elements to co-evolve. In the exploration of the decision space, user preferences are defined for the set of controlling parameters (mainly model elements to be co-evolved) that each edit operation has (see Table 1). After selecting a preferred objective space cluster, the designer may want to see "the distribution of the solutions within that region of interest". In other words, the clustering in the decision space will show designers the co-evolution solutions that improve specific objective(s) at the same level (within the same objective space cluster) but targeting different parts of model (model elements). To do this, we group the solutions by their similarity in the decision space and present them to the developer.

To get an optimal grouping of co-evolution solutions in the decision space of where the edit operations are applied, we use a procedure similar to the one used in the objective space with additional pre-processing steps to project the co-evolution solutions on the decision space. We define a projection operator based on the frequency of changes to the model elements by the edit operations and their locations (co-evolved model elements). Since the edit operations affect model elements differently, where some make changes only at the object while others have effect at the slots and links of the model, we only count the co-evolved model elements in our work to have a consistent representation for all vectors and to create a new representation for the co-evolution solution vector in the decision space. In this new domain space, the co-evolution solutions are represented as vectors of integers where the co-evolved model elements are the dimensions of the space, and the values are the number of edit operations for that element. The projection operator is used for the entire Pareto-front and enables having two different representations of the same solution set. Note that the number of co-evolved model elements depends on the size of the co-evolution solutions. Since we considered the same minimum and maximum size thresholds of co-evolution solutions for all executions of the algorithm, the time to generate the clusters is similar even for larger metamodels/models since the size is not based on all model elements of the project but just those in the co-evolution solutions. A larger set of modified model elements may generate more clusters to explore, which can make the interaction more time-consuming. Additionally, the decision space clustering heavily depends on how many model elements are co-evolved within each solution. If the majority of the solutions in the Pareto front are co-evolving almost the same model elements (for instance, one object), then mainly

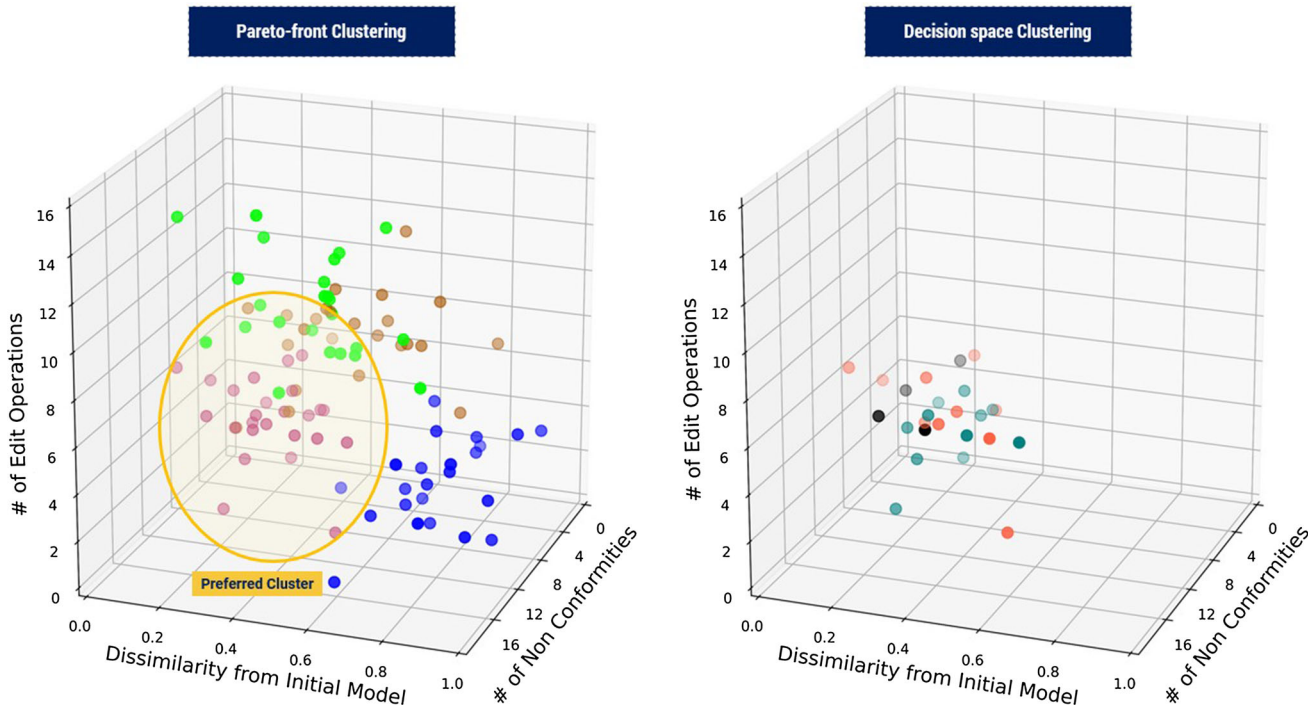

Fig. 7 Illustration of the clustered solutions in the objective space and the decision space

one big cluster will be generated in the decision space. It is true that a large co-evolution solution may have a higher probability to modify larger model elements than a smaller one, but it is more accurate to estimate the number of possible clusters in the decision space based on the model elements that are edited by the co-evolution solutions in the Pareto front.

Figure 7 shows an example using our tool with a population size of 100. After the generation of the Pareto front and the execution of the objective space clustering, the clustering feature identified four main different clusters. The designer then selected the purple cluster in the objective space as the preferred one for further exploration. After that, the solutions composing it are clustered (in the decision space) based on the model element locations and their frequency. The designer can observe that within this cluster, there are three different clusters in the decision space, where each cluster represents solutions that co-evolve specific model elements. A user can click on the preferred co-evolution solution to see the revised model elements by the edit operations.

The main contribution of our work is enabling the exploration of a diverse set of co-evolution solutions within the same objectives space. This amounts to having multiple solutions that are neighbors in the objective space but completely different in the decision space. To do this, we go through all the clusters determined in the previous step and then use the GMM clustering algorithm with the same steps described above to group similar solutions in the decision space. Thus, designer can co-evolve the models toward their preferred objectives while only co-evolving the elements of the model that interest them.

3.4 Phase 4: developers interaction tool and preferences extraction

In this phase, the user has the ability to explore the recommended solutions and clusters efficiently and discover the shared underlying characteristics of the solutions in a cluster at a glance. He may investigate the center solution of each cluster, or search further and examine the solutions inside a cluster of interest. It is also possible to compare the initial and revised models (charts feature) to evaluate the correctness and relevance of the recommended operations and take appropriate interaction actions. Every edit operation can be evaluated by the user. As described in Algorithm 2, we translate each evaluation feedback to a continuous score in the range of [-1,1].
Figure 8 shows a solution with four edit operations that are evaluated by the designer.

The user can interact with the tool at the solution level by accepting/rejecting/modifying specific edit operation or the cluster level by specifying a specific cluster as the region of interest. Figure 9 shows an example of a SplitModelElement edit operation that is modified by the designer.

After the interaction is done and the user decides to continue to the next round, the score of each solution and cluster

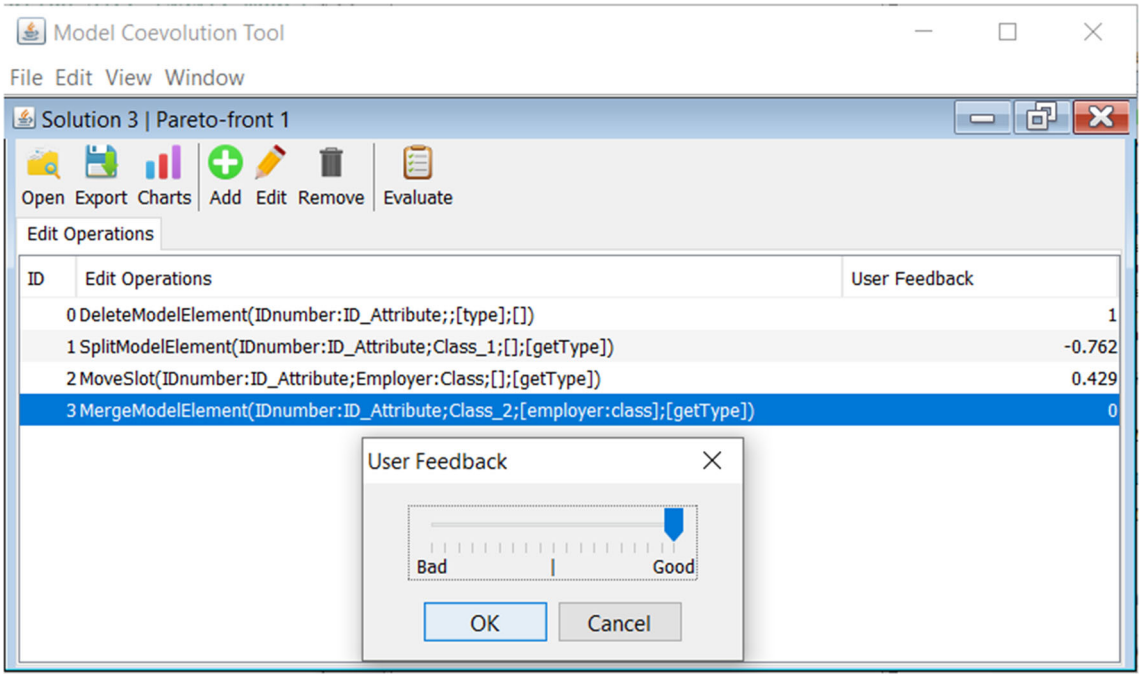

Fig. 8 An illustration of the interaction feature (evaluate edit operation) with the designer for a generated co-evolution solution

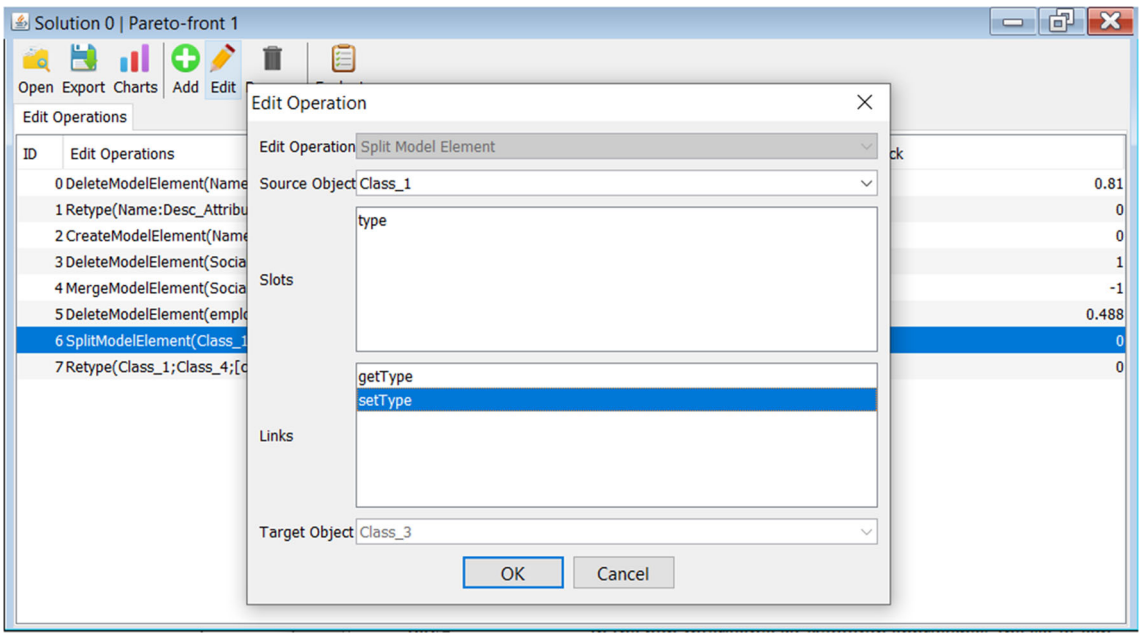

Fig. 9 An illustration of the interaction feature (modifying an edit operation) with the designer for a generated co-evolution solution

are computed. Solution score ($Score_{s_i}$) is defined as the average of all edit operations score exists in the solution vector. Similarly, Cluster score ($Score_{c_k}$) is calculated as the average of all solutions score assigned to the cluster. Then, the cluster that achieve the highest score among all clusters is considered as the user preferred partition in Pareto-front space from which the preference parameters will be extracted.

The next step of phase 3 of our proposed approach is to extract user preference parameters from the interaction step. We consider the representative solution of the preferred cluster as the reference point. Then, we compute the weighted probability of edit operations (OWP) and target model elements (MWP). Assuming the selected cluster's index is j,

Algorithm 2: Interaction and User Preferences

Input : Labeled solutions (LS)
Output: Preferred Cluster (PC),
Preference Parameters=[
MWP(Model elements Weighted Probability,
OWP(Edit Operation Weighted Probability),
RS(Reference Solution)]

begin User Interaction and Feedback
  while ¬ *interaction is done* **do**
    $Feedback_i \leftarrow$ UserEvaluation(Eo_i);
    $V_i \leftarrow$ Score($Feedback_i$);
  /* SOLUTIONS AND CLUSTERS SCORE */
  $Score_{s_i} \leftarrow$ Average($V_i \in s_i$);
  $Score_{c_k} \leftarrow$ Average($Score_{s_i} \in c_k$);
  PC ← cluster with Max score;
begin User Preference Extraction
  /* REPRESENTATIVE SOLUTION AS REFERENCE */
  RS ← CR_{PC};
  foreach $[eo_i, elt_i] \in PC$ **do**
    $OWP_p \leftarrow$ AverageWeightedFreq(eo_p);
    $MWP_q \leftarrow$ AverageWeightedFreq(elt_q);
Return PC, Preference Parameters[];

these parameters can be computed as follows:

$$OWP_p = \frac{\sum_{s_i \in c_j} \gamma_{ij} \times (|o_p \in s_i|)}{\sum_{o_m \in Eo} \sum_{s_i \in c_j} \gamma_{ij} \times (|o_m \in s_i|)} \quad (3)$$

$$MWP_q = \frac{\sum_{s_i \in c_j} \gamma_{ij} \times (|elt_q \in s_i|)}{\sum_{elt_m \in Elts} \sum_{s_i \in c_j} \gamma_{ij} \times (|elt \in s_i|)} \quad (4)$$

where s_i is the solution vector, γ_{ij} is the membership coefficient of solution i to the cluster j, o is the edit operation action, Eo is the set of all edit operations, and $Elts$ is the set of all model elements.

3.5 Applying preference parameters

If the user decides to continue the search process, then the preference parameters will be applied during the execution of different components of multi-objective optimization as described in the following:

- *Preference-based initial population*: The solutions from preferred clusters will make up the initial population of next iteration as a means of customized search starting point. In this way, we initiate the search from the region of interest rather than randomly. New solutions need to be generated to fill and achieve the pre-defined population size. Instead of random creation of the edit operations based on a unify probability distribution, we utilize OWP and MWP as a probability distribution.
- *Preference-based mutation*: For this operator, similarly, if a solution is selected to mutate, we give a higher chance to edit operations of interest to replace the chosen one based on the probability distribution OWP.
- *Preference-based selection*: the selection operator tends to filter the population and assign higher chance to the more valuable ones based on their fitness values. In order to consider the user preferences in this process, we adjusted this operator to include closeness to the reference solution as an added measure of being a valuable individual of the population. That means the chance of selection is related to both fitness values and distance to the region of interest as:

$$Chance(s_i) \propto \frac{1}{dist(s_i, CR_j)}, Fitness(s_i) \quad (5)$$

where $dist()$ indicates Euclidean distance and CR_j is the representative solution of cluster j.

The above-mentioned customized operators aid to keep the stochastic nature of the optimization process and at the same time take the user preferred edit operations into account.

The generation and selection of the solutions in the next iteration of the multi-objective search, after the interactions, is based on 1) the probability formulas for both co-evolution operations and their locations extracted from the preferred decision and objective clusters to select and change solutions in the next iterations, and 2) the initial population of the next iteration of the search algorithm after interactions are seeded from the solutions of the preferred cluster. These are the key factors to integrate preferences, rather than defining constraints or using the fitness functions as it is done in the related work. We note that users first start to interact with clusters in the objective space based on their goals. Once they select their preferred objective (PO) space cluster, a decision space clustering is performed on the solutions of PO to show the user the diversity of the solutions in the decision space. Then, the user can select the preferred decision space cluster, called PS. The solutions in PS are used to initiate the population of the next iteration of the multi-objective search and to calculate the probabilities for the selection and change operators.

Our proposed approach will help the designer to understand the diversity of the co-evolution solutions when visualizing the clusters thus it will help the user to locate her/his region of interest in both the objective and decision spaces. The goal of the interactions and clustering is to gradually reduce the number of co-evolution solutions to be explored by the users based on their preferences.

The clustering phase will reduce the search space; however, the interaction with the user is the critical factor affecting the size of the clusters since the approach continues to filter the solutions based on those interactions. Thus, the

number of clusters and solutions to explore in both decision and objective spaces depend on the preferences of the user.

4 Validation

4.1 Research questions

We defined two main research questions to measure the correctness, relevance and benefits of our decision and objective space interactive clustering-based multi-objective model co-evolution tool(DOIC-NSGA-II) comparing to : (1) an approach based on interactive multi-objective search (I-NSGA-II) [40], but the interactions were limited to accept/reject edit operations and there is no clustering of the Pareto front or learning mechanisms from the interaction data, (2) an approach that is based on interactive clustering-based multi-objective model co-evolution (OIC-NSGA-II) only in the objectives space [37], (3) an automated multi-objective co-evolution approach (without the interaction component) [39] and (4) an existing automated co-evolution approach based on pre-defined rules without using search methods [77].

The research questions are as follows:

- **RQ1: Co-evolution relevance.** To what extent can our approach make meaningful recommendations compared to existing metamodel/model co-evolution techniques?
- **RQ2: Interactive objective and decision spaces clustering relevance.** To what extent can our clustering-based approach **efficiently** reduce the interaction effort?

The first research question aims to manually evaluate the correctness of the co-evolution operations and generated models while the second research question is more dedicated to the comparison with the SBSE state of the art.

4.2 Experimental setting

4.2.1 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 [11,77]. These case studies are interesting scenarios since they represent real metamodel evolutions, used in an empirical study [29] and studied in other contributions [17,27,66]. 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. 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.

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 the 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 [11,77] and thus we were not involved in the selection of models and metamodel changes. In order to ensure a fair comparison with Wimmer et al. [77], we only compared both approaches on the existing UML dataset. Table 2 describes the statistics related to the collected data.

4.2.2 Evaluation metrics

To evaluate the relevance of our tool, we used the manual correctness (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 for the interactive model co-evolution approaches. This evaluation will help to understand if we efficiently reduced the interaction effort. We decided to limit the comparison to only the interactive multi-objective approaches [37,40] since they are the only approaches offering interaction with the user, and it will help us understand the real impact of the decision space exploration (not supported by existing studies) on the recommendation and interaction effort. Furthermore, we report the computation time (T) for the different evolution scenarios to estimate the effort required to obtain the best co-evolution solutions.

All these metrics are used for the research questions including the comparison between our decision and objective space interactive clustering-based multi-objective model co-evolution approach (DOIC-NSGA-II), an approach that is based on interactive clustering-based multi-objective model co-evolution (OIC-NSGA-II) only in the objectives space [37], an existing interactive multi-objective approach [40] (without the clustering feature) (I-NSGA-II) and the two automated techniques of Kessentini et al. [38] and Wimmer et al. [77].

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 [77]	From 23 to 29	8	8	10	28, 49	11, 23

4.3 Study participants and parameters setting

Our study involved 20 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, 16 out the 20 students are working as full-time or part-time developers in 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 5 groups, each composed by 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 designers' experience. The participants were asked to manually co-evolve the different models and evaluate the results of the different approaches based on a counter-balanced design [60].

The parameters' values of the different search algorithms were fixed by trial and error and are as follows: crossover probability = 0.3; mutation probability = 0.5 where the probability of gene modification is 0.3; stopping criterion = 100,000 evaluations. Trial and error is a fundamental method of problem solving. It is characterized by repeated and varied attempts of algorithm configurations [35].

4.4 Results

Results for RQ1: Co-evolution relevance. We report the results of the empirical qualitative evaluation (MC) in Fig. 10. The majority of the co-evolution solutions recommended by our approach were correct and validated by the participants on the different case studies. On average, for all of our four studied metamodels/models, our approach was able to correctly recommend 95% of generated edit operations. The remaining approaches have an average of 91%, 86% and 77% respectively for interactive with objective space clustering [37], the interactive multi-objective approach [40] and the fully automated multi-objective approach [39].

Compared to the interactive approaches, we found that some of the co-evolution solutions of DOIC-NSGA-II are not proposed by OIC-NSGA-II [37] and I-NSGA-II [40]. In fact, one of the main challenges of multi-objective search is the noise introduced by sacrificing some objectives and trying to diversify the solutions. Thus, the decision space exploration can help the designers know the most diverse co-evolution solutions among one preferred cluster in the objective space. Thus, designers did not waste time on evaluating co-evolution solutions that are similar.

The interactive tools outperformed fully automated one which shows the importance of integrating the human in the loop when co-evolving models. Furthermore, it is clear that adding the clustering feature enables the designers to select a region of interests based on which objectives they want to prioritize and what solutions they partially liked.

The deterministic approach defines generic rules for a set of possible metamodel changes that are applied to the co-evolved models. Figure 10 shows that our interactive approach clearly outperform, in average, the deterministic technique. The comparison is limited to the only case of UML Class Diagram evolution since for this case Wimmer et al. [77] provide a set of co-evolution rules. Further adaptations are required to make this set of rules working on other metamodels.

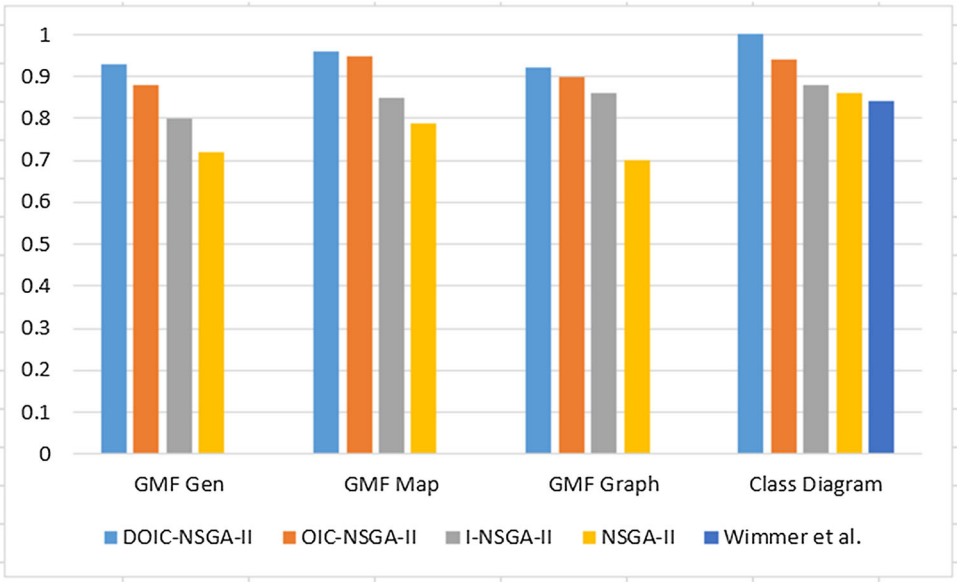

Fig. 10 The median manual evaluation scores, MC, on the four metamodel evolution scenarios with 95% confidence level ($\alpha = 5\%$)

A qualitative analysis of the results shows that several interactions with the designers helped to reduce the search space by avoiding the edit operations that were rejected by them. We found that the best final co-evolution solutions identified by the designers after several interactions with our tool cannot be recommended by the remaining approaches. In fact, all these solutions are obtained either after 1) eliminating/modifying edit operations applied to models that are not relevant to the designers' preferences or 2) emphasizing specific cluster that prioritizes some objectives/model elements and penalizes others.

All the results based on the MC metric on the different case studies were statistically significant with 95% of confidence level using the Kruskal-Wallis test. Regarding the effect size, we found that our approach is better than the others with an A effect size higher than 0.82 for GMF GEN, GMF MAP, GMF Graph; and an A effect size higher than 0.88 for Class Diagram. While the results of the manual correctness are consistent independently from the type of the edit operations, we note that in the case of a metamodel element addition that change will not break the initial existing models (non-breaking changes). Thus, it is expected that the addition of new metamodel elements will create the least number of inconsistencies at the model level. However, the deletion of metamodel elements will generate significant conflicts at the model level and thus the optimization process will try to reduce these issues by removing the instances of the deleted elements in the models while adjusting the relationships accordingly.

Results for RQ2: interactive clustering relevance. Figures 11 and 12 summarizes the time, in minutes, and the number of interaction with the participants to find the most relevant solutions using DOIC-NSGA-II, OIC-NSGA-II, and the interactive approach I-NSGA-II (without clustering). The time includes the execution of the multi-objective search, both clusterings, and the different phases of interaction until the developer is satisfied with a specific solution. The execution time of OIC-NSGA-II includes all the steps of the multi-objective search, the objective space clustering and the interactions, while I-NSGA-II includes the multi-objective search and the user interactions. Thus, the main differences in the execution time can be observed in the interaction effort.

All the participants spent less time to find the most relevant model edit operations on the different metamodels comparing to OIC-NSGA-II and I-NSGA-II. For instance, the average time of our approach is reduced by over 44 minutes (60%) compared to I-NSGA-II for the case of GMF gen. The reduction of the execution time is mainly explained by the rapid exploration of fewer solutions after looking mainly to the different solutions in the decision space of the preferred cluster in the objective space. Our approach has more components (clustering at both objective and decision spaces) than OIC-NSGA-II and I-NSGA-II but the clustering at both spaces significantly reduced the user interactions and helped the user to quickly check only the solution based on his/her preference and his/her region of interest to co-evolve the model. The execution time is mainly affected by the designer's interaction effort which is also affected by the co-evolution solutions that they need to explore and check manually. The decision space clustering of the preferred cluster from the objective space dramatically reduces the number of solutions to check which resulted in fewer interactions. For instance, a designer can easily avoid checking many solutions within the same decision space cluster (modifying similar model elements) that have similar impacts on the objectives

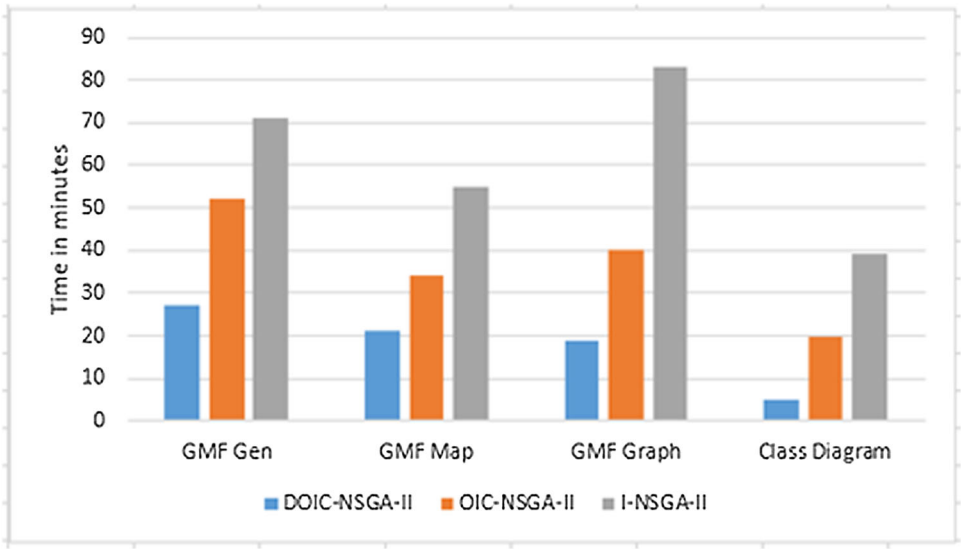

Fig. 11 Median time, in minutes, proposed by the different interactive approaches on the different metamodel/model co-evolution scenarios

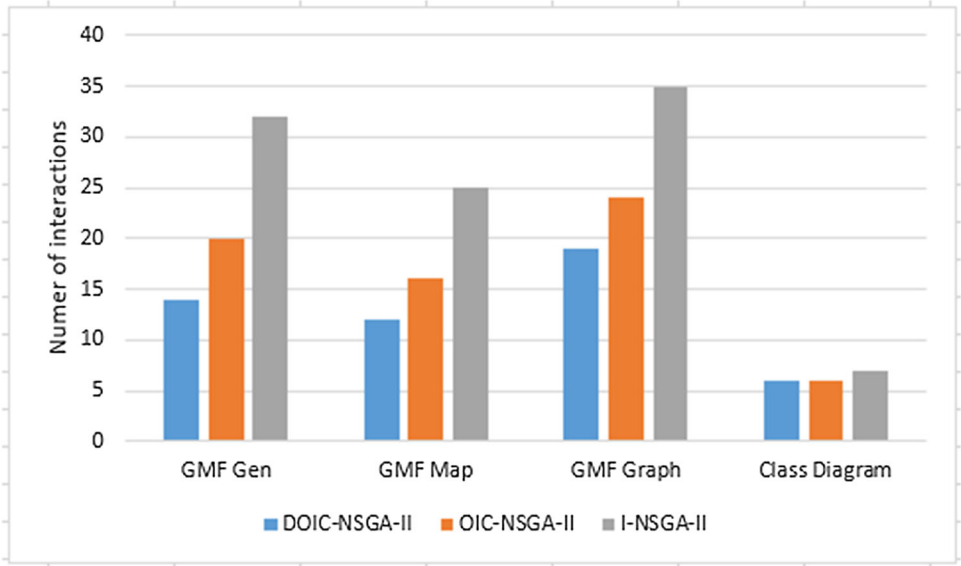

Fig. 12 Median number of interactions proposed by the different interactive approaches on the different metamodel/model co-evolution scenarios

In the post-study feedback, all the participants mentioned that both the decision and objective space interactions helped them to identify a relevant co-evolution solution. They found that the tool is faster and much easier to use than the one without the clustering component. Also, they mentioned that adding the decision space helped them to check diverse solutions based on the revised model elements. They used the objective space to achieve their goals (e.g. minimize the number of conformance errors in the co-evolved models) then they used the decision space to find a solution that target the model elements they want to co-evolve.

Similar observation is valid when comparing our tool to the fully automated multi-objective tool [39]. 17 out of the 20 participants highlighted the difficulty to select one relevant solution from a large set of non-dominated solutions and without offering any flexibility to update them.

All the users mentioned the high usability of the tool and the different options that are offered comparing to deterministic tool [77]. They did not appreciate the pre-defined transformations based on metamodel change types since the latter 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. Thus, the users appreciate that our tool automatically suggests solutions and update the list based on their feedback.

5 Threats to validity

Conclusion validity. The parameter tuning of the different optimization algorithms used in our experiments creates an

internal threat that we need to evaluate in our future work. The parameters' values used in our experiments are found by trial-and-error. However, it would be an interesting perspective to design an adaptive parameter tuning strategy [5] for our approach so that parameters are updated during the execution in order to provide the best possible performance.

Internal validity. An internal threat is related to the variation of correctness and speed between the different groups when using our interactive approach and other tools. In fact, our approach may not be the only reason for the superior performance because the participants have different skills and familiarity with MDE tools. To counteract this, we assigned the participants to different groups according to their experience so as to reduce the gap between the different groups and we also adapted a counter-balanced design. Regarding the selected participants, we have taken precautions to ensure that our participants represent a diverse set of participants with experience in model-driven engineering and also that the groups formed had, in some sense, a similar average skill set in the model maintenance area. To address the fatigue threat, we did not limit the time to fill the questionnaire and we also sent the questionnaires to the participants by email and gave them the required time to complete each of the required tasks.

External validity. In this study, we performed our experiments on different widely studied models and metamodels belonging to different domains and having different sizes. However, we cannot assert that our results can be generalized to other artifacts, and to other practitioners. Another threat is the limited number of participants and evaluated models/metamodels. In addition, our study was limited to the use of specific edit operation types. The considered edit operations may not cover every co-evolution scenario and we will extend the list of supported edits operations such as retyping model elements as part of our future work. However, the approach itself will remain the same and just the inputs should be changed. Furthermore, our assumption that designers are looking for both objectives and operation types (and their locations) when deciding which co-evolution strategy to adopt may require a separate empirical study to be generalized and validated. Future replications of this study are necessary to confirm our findings.

6 Related work

The evolution of metamodels and related artifacts (e.g., models, model transformations, OCL constraints, etc.) are interdependent. A change in one artifact (metamodel) must be reflected in all other related artifacts. We discuss in this section approaches explicitly developed to target the metamodel co-evolution problem such as the co-evolution of metamodel/Model, metamodel/Transformation Rules and metamodel/OCL

6.1 Model co-evolution

Co-evolution has been subject for research since several decades in the database community [65], and especially, the introduction of object-oriented database systems [6] 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., [26,33,52,59,66,68] for an overview). They can be classified in three categories [68]:

- 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) [57,67,75].
- Metamodel matching techniques used to infer a migration strategy from the difference between the original metamodel and the evolved metamodel [10,12,19,22,52,53,77].
- Operator-based approaches that record the metamodel changes as a sequence of co-evolutionary operations used later to infer a complete migration strategy [4,28,32,33,76].

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

6.1.1 Manual specification approaches

In one of the early work [73], Sprinkle and Karsai present a domain-specific visual language. The co-evolution of models is tackled by defining patterns which describe the migration steps based on metamodel change types. Rose et al. [67] proposed a textual migration language, using transformation rules. Flock copies model elements of the old version and that are still conform to the new version of the metamodel automatically. Then the user defines manually the migration specification to co-evolve the remaining non-conforming model elements. Narayanan et al. [57] present a Model Change Language (MCL) which represents the differences between the source metamodel and the target metamodel in terms of rules, that helps the user to specify model migrations Another manual specification approach is presented in [75] where a specific transformation language is derived to

describe the evolution on the metamodel level and derive a transformation for the model level.

6.1.2 Metamodel matching approaches

Matching approaches are based on the matching between the initial and evolved metamodel versions. In [22], 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 [67] and in the framework described in [57].

In [12,19,53], the authors compute differences between two metamodel versions 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 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., [32,47,48,53,74,77]), 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.

Additionally, in [3], weaving models are employed to connect 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.

Most of the above approaches focus on identifying conceptually high-level changes to the metamodel in order to co-evolve models. They detect such changes either by manually comparing the two metamodel versions or by recording, matching or calculating their differences. Thus, these approaches apply various change-specific strategies aimed at mirroring the high-level conceptual changes. We tackled co-evolution of artifacts without the need of computing differences on the metamodel level. Instead, we search for solutions which fulfill multiple goals expressed as our fitness functions. 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 appropriate ones and interactively update them.

6.1.3 Operator-based approaches

Other contributions are based on using coupled operations [28,30,32,76]. In [76], Wachsmuth provides a library of co-evolutionary operators for MOF metamodels, each of these operators provides a model migration strategy. In [28], Herrmannsdoerfer provides a tool support for the evolution of Ecore-based metamodels, which 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 [32] 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 [4], 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.

Schoenboeck et al. [72] propose an approach that does not rely on metamodel changes. Authors apply a constrained-based search to detect the conformance violations and provide repair operations to re-establish this conformance relationship.

The next section discusses the approaches proposed to tackle the transformations co-evolution problem.

6.2 Transformation co-evolution

The evolution of models and transformations can be treated separately, but when their metamodels are updated or adapted, those artifacts might be impacted. However, the co-evolution of metamodels and transformations is different from the co-evolution of metamodels and models. The former depends only on the type of change performed during metamodel evolution, and the latter depends not only on the type of change but also on how the updated elements may affect the transformation rules [42]. Surprisingly, transformation co-evolution problem has received less attention

in MDE compared to model co-evolution. Thus, this area is largely unexplored and may require further investigation.

Like the term *conformTo* that specifies the relation between metamodel and model, Mendez et al. [51] introduce the term *DomainConformTo* as the relation between transformations and their metamodels. For instance, the source elements of every transformation rule must correspond to a metaclass in the source and/or the target metamodel elements. Even though the authors have studied how a metamodel evolution may affect that relationship and have given first pointers to an operator based approach for transformation co-evolution [51].

Similarly, Kusel [42] proposed an initial set of atomic operators and some complex ones (that are made up of simple operators) that can be applied on a given metamodel to perform changes and allow automatic or semi-automatic co-evolution of transformation and analyze the influence of the changes in the original ATL rules.

Di Ruscio et al. [16] proposed an approach to the coupled evolution of metamodels and ATL transformation rules. This approach contains various activities that start with defining the dependencies between the transformation language and the metamodeling language (e.g., establishing the correspondence between ATL and ECore metamodels) which are used later to derive them between an evolving metamodel and the existing transformations. The second step is analyzing the impact of changes by detecting all the elements of the transformation that are affected after the evolution of the metamodel. Then, the designer evaluates the cost of adapting the affected transformation; if the adaptation is too expensive, the designer can decide to refine the metamodel changes to reduce the cost. After that, the affected transformation can be adapted.

Levendovszky et al. [45] proposed a Higher Order Transformation to adapt existing transformations developed in the GME/GReAT toolset. They classify metamodels, with respect to the affected transformation, to three categories:

- Fully automated : changes that affect existing transformations which can be automatically migrated without user intervention.
- Partially automated: changes or modifications that affect existing transformations which can be adapted automatically, even though some manual fine-tuning is required to complete the adaptation.
- Fully semantic: changes are those modifications which affect transformations and cannot be automatically migrated, and the user has to completely define the adaptation.

The proposed algorithm automatically alerts the user about missing information when the automation is not possible. However, the approach is limited to graph-based languages, considering simple changes and considering subtracting changes only as coarse-grained removals (i.e., rule level deletions). And this, due to the use of the Model Change Language (MCL), which represents the differences between the source metamodel and the target metamodel in terms of rules that have been designed for model co-evolution in order to migrate transformation rules. Thus the solution is only limited to a few evolution cases.

Garcia et al. [21] have proposed an approach compromise two steps for transformation co-evolution: detection and co-evolution. The detection step compromises the detection of simple changes (e.g., class renaming) and the detection of complex changes that are considered as predicates over simple changes. The former are detected by the difference between the original and the evolved metamodel using EMF Compare tool. The latter is realized as a transformation that takes a *Difference* model as input and obtains a *DiffExtended* model that includes both simple and complex changes. At the co-evolution step, authors adapt the classification of changes of model co-evolution [12] to the transformation co-evolution problem. They define higher order transformations as a set of ATL rules, that takes transformations as input and returns a modified transformations by taking as parameters the changes obtained during the detection step to define a correspondence that map the original transformation into an evolved one.

Similarly, in [43] and [20], authors have proposed a matching strategy that computes the equivalences and differences between a pair of metamodels and persists the results in a mapping model. Then a manual user intervention is needed to refine the generated mapping model. Finally, a HOT generates the adaptation transformation from the (refined) mapping model.

6.3 Artifacts co-evolution

In this section, we discuss approaches for the problem of co-evolving metamodels and other artifacts such as OCL constraints. Existing approaches can be classified as online or offline approaches. Online approaches perform instant co-evolution for each change during the metamodel evolution, whereas offline approaches wait after the metamodel has been evolved to perform co-evolution of the OCL constraints.

For online approaches, Demuth et al. [14,15] provide templates that define a fixed structure for OCL constraints that are then instantiated to update the constraints. However, they are limited to 11 templates that cannot cover all changes at metamodel level. Hassam et al. [24,25] propose a semi-automatic approach that highlights the constraints that should disappear after evolution and by formalizing the adaptation to be applied on impacted constraint after each operation on a metamodel using the QVT transformation language [58]. Similarly, Markovic et al. [49,50] proposed an approach

using QVT, in which they formalize the most important refactoring rules for class diagrams and classify them with respect to their impact on annotated OCL constraints. The advantage of online approaches is that the order of changes is preserved and no hidden changes are missed. However, the cancelling actions during evolution are apart of the detected changes.

For offline approaches, Kusel et al. [44] analyze the impact of metamodel evolution on OCL, then propose resolution actions in model transformation by means of ATL helpers. Cabot and Conesa [8] focused on the metamodel changes that entail deleting elements. In particular, they aimed to remove the parts of OCL constraints that use the deleted elements. Khelladi et al. [41] propose a semi-automatic approach that records in chronological order the changes to the metamodel. Then, they detect high-level changes and apply resolution strategies to adapt OCL constraints based on the structure of the impacted OCL constraint and the impacted location.

The above-mentioned approaches focus on identifying conceptually high-level changes to the metamodel in order to co-evolve OCL constraints. They detect such changes either by manually comparing the two metamodel versions or by recording, matching or calculating their differences. Subsequently, they apply various change-specific strategies aimed to mirror the high-level conceptual changes.

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 an artifact.

6.4 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 [23].

SBSE has been applied to related problems in Model-Driven Engineering such as model transformation [18], transformation testing [70], model refactoring [46], requirements modeling [36], product line testing [71], etc. However, these problems are different in nature than the co-evolution problem investigated in this paper.

A comprehensive survey of interactive SBSE approaches can be found in [61]. 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 [2,54,55]. Han et al. proposed in [2] an approach to enable the interactions with the user, and then a Delta Table can select the next refactoring quickly to improve a specific objective without calculating a fitness function. Morales et al. [54] proposed an algorithm to remove redundant refactoring solutions that may have the same refactorings with a different order in the sequence, but the final design is the same.

None of the above studies focused on the extraction of user preferences at the decision space.

7 Conclusion

In this paper, we proposed an interactive clustering-based multi-objective approach for metamodel/model co-evolution that reduces the interaction effort to find relevant co-evolution solutions. The feedback received from the designers and the clustering of the solution space are used to reduce the search space and converge to better solutions. The clustering of the decision space helped the designers identify the most diverse co-evolution solutions among the ones located within the same cluster in the objective space. We evaluate the effectiveness of our tool on several evolution scenarios extracted from different widely used metamodels, and we compared it to fully automated and interactive co-evolution tools. Our evaluation results provide clear evidence that our tool helped designers to quickly express their preferences and converge toward relevant revised models that met the their expectations.

We plan to extend this work by evolving interactively model transformation rules and OCL constraints when the source or target models are revised. Another possible research direction is to evaluate the feasibility of updating the test cases based on the evolution of models and metamodels. The initial population of co-evolution solutions is randomly generated. Another strategy is to learn constraints after the execution of the proposed algorithms to guide the selection of the edit operations. For instance, some constraints can be learned to avoid conflicting edit operations. We will explore this direction as part of our future work.

References

1. Supporting materials: online appendix / https://sites.google.com/view/sosym2021https://doi.org/10.5281/zenodo.5804649
2. An efficient approach to identify multiple and independent move method refactoring candidates. in IST **59**, 53 – 66 (2015)
3. Anguel, F., Amirat, A., Bounour, N.: Using weaving models in metamodel and model co-evolution approach. In: Proceedings of CSIT (2014)
4. Anguel, F., Amirat, A., Bounour, N.: Hybrid approach for metamodel and model co-evolution. In: Proceedings of CIIA (2015)
5. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of ICSE (2011)
6. Banerjee, J., Kim, W., Kim, H.J., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: Proceedings of SIGMOD (1987)

7. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers (2017)
8. Cabot, J., Conesa, J.: Automatic integrity constraint evolution due to model subtract operations. In: International Conference on Conceptual Modeling, pp. 350–362 (2004)
9. Calinski, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods **3**(1), 1–27 (1974)
10. Cicchetti, A., Ciccozzi, F., Leveque, T., Pierantonio, A.: On the concurrent versioning of metamodels and models: challenges and possible solutions. In: Proceedings IWMCP (2011)
11. 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)
12. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Proceedings of EDOC (2008)
13. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In: Proceedings of PPSN (2000)
14. Demuth, A., Lopez-Herrejon, R.E., Egyed, A.: Automatically generating and adapting model constraints to support co-evolution of design models. In: Automated Software Engineering (ASE), pp. 302–305 (2012)
15. Demuth, A., Lopez-Herrejon, R.E., Egyed, A.: Supporting the co-evolution of metamodels and constraints through incremental constraint management. In: International Conference on Model Driven Engineering Languages and Systems, pp. 287–303 (2013)
16. Di Ruscio, D., Iovino, L., Pierantonio, A.: What is Needed for Managing Co-evolution in MDE? In: Proc. of the 2nd International Workshop on Model Comparison in Practice, IWMCP'11 (2011)
17. Di Ruscio, D., Lammel, R., Pierantonio, A.: Automated co-evolution of gmf editor models. In: Proceedings of SLE (2011)
18. Fleck, M., Troya, J., Wimmer, M.: Search-based model transformations. J. Softw: Evol. Proc. **28**(12), 1081–1117 (2016)
19. Garces, K., Jouault, F., Cointe, P., Bezivin, J.: Managing model adaptation by precise detection of metamodel changes. In: Proceedings of ECMFA (2009)
20. Garces, K., Vara, J.M., Jouault, F., Marcos, E.: Adapting transformations to metamodel changes via external transformation composition. Softw. Syst. Model. **13**(2), 789–806 (2014)
21. Garcia, J., Diaz, O., Azanza, M.: Model transformation co-evolution: A semi-automatic approach. In: SLE, pp. 144–163 (2012)
22. Gruschko, B.: Towards synchronizing models with evolving metamodels. In: Proceedings of MoDSE Workshop (2007)
23. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: trends, techniques and applications. ACM Comp. Surv. (CSUR) **45**(1), 1–61 (2012)
24. Hassam, K., Sadou, S., Fleurquin, R.: Adapting ocl constraints after a refactoring of their model using an mde process. In: 9th edition of the BElgian-NEtherlands software eVOLution seminar (BENEVOL 2010), pp. 16–27 (2010)
25. Hassam, K., Sadou, S., Le Gloahec, V., Fleurquin, R.: Assistance system for ocl constraints adaptation during metamodel evolution. In: Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pp. 151–160. IEEE (2011)
26. 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)
27. Herrmannsdoerfer, M.: GMF: A model migration case for the transformation tool contest. In: Proceedings of TTC (2011)
28. Herrmannsdoerfer, M., Benz, S., Juergens, E.: Cope - automating coupled evolution of metamodels and models. In: Proceedings of ECOOP (2009)
29. Herrmannsdoerfer, M., Ratiu, D., Wachsmuth, G.: Language evolution in practice: The history of gmf. In: Proceedings of SLE (2010)
30. Herrmannsdoerfer, M., Vermolen, S., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Proceedings of SLE (2011)
31. Herrmannsdoerfer, M., Vermolen, S.D., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Proceedings of SLE (2011)
32. Herrmannsdorfer, M.: COPE - A Workbench for the Coupled Evolution of Metamodels and Models. In: Proceedings of SLE (2011)
33. Herrmannsdorfer, M., Wachsmuth, G.: Coupled evolution of software metamodels and models. In: Evolving Software Systems, pp. 33–63 (2014)
34. Iovino, L., Pierantonio, A., Malavolta, I.: On the impact significance of metamodel evolution in MDE. J. Object Technol. **11**(3), 31–33 (2012)
35. Jackson, R., Carter, C., Tarsitano, M.: Trial-and-error solving of a confinement problem by a jumping spider, portia fimbriata. Behaviour **138**(10), 1215–1234 (2001)
36. Kessentini, M., Mansoor, U., Wimmer, M., Ouni, A., Deb, K.: Search-based detection of model level changes. Empir. Softw. Eng. **22**(2), 670–715 (2017)
37. Kessentini, W., Alizadeh, V.: Interactive metamodel/model co-evolution using unsupervised learning and multi-objective search. In: MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020 (2020)
38. Kessentini, W., Sahraoui, H.A., Wimmer, M.: Automated metamodel/model co-evolution using a multi-objective optimization approach. In: Proceedings of ECMFA (2016)
39. Kessentini, W., Sahraoui, H.A., Wimmer, M.: Automated metamodel/model co-evolution: a search-based approach. Inf. Softw. Technol. **106**, 49–67 (2019)
40. Kessentini, W., Wimmer, M., Sahraoui, H.A.: Integrating the designer in-the-loop for metamodel/model co-evolution via interactive computational search. In: A. Wasowski, R.F. Paige, O. Haugen (eds.) In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018, pp. 101–111. ACM (2018)
41. Khelladi, D.E., Hebig, R., Bendraou, R., Robin, J., Gervais, M.P.: Metamodel and constraints co-evolution: A semi automatic maintenance of ocl constraints. In: International Conference on Software Reuse, pp. 333–349 (2016)
42. Kruse, S.: On the use of operators for the co-evolution of metamodels and transformations. In: Models and Evolution Workshop (2011)
43. Kusel, A., Etzlstorfe, J., Kapsammer, E., Schonbock, J.: Consistent co-evolution of models and transformations. MoDELS (2015)
44. Kusel, A., Etzlstorfer, J., Kapsammer, E., Retschitzegger, W., Schoenboeck, J., Schwinger, W., Wimmer, M.: Systematic co-evolution of ocl expressions. 11th APCCM (2015)
45. Levendovszky, T., Balasubramanian, D., Narayanan, A., Karsai, G.: A novel approach to semi-automated evolution of DSML model transformation. In: SLE, pp. 23–41 (2010)
46. Mansoor, U., Kessentini, M., Wimmer, M., Deb, K.: Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. Softw. Qual. J. **25**(2), 473–501 (2017)
47. Mantz, F., Lamo, Y., Taentzer, G.: Co-transformation of type and instance graphs supporting merging of types with retyping. ECEASST **61**, 24 (2013)
48. Mantz, F., Taentzer, G., Lamo, Y., Wolter, U.: Co-evolving metamodels and their instance models: a formal approach based on graph transformation. Sci. Comput. Program. **104**, 2–43 (2015)

49. Markovic, S., Baar, T.: Refactoring ocl annotated uml class diagrams. In: International Conference On Model Driven Engineering Languages And Systems, pp. 280–294 (2005)
50. Markovic, S., Baar, T.: Refactoring ocl annotated uml class diagrams. Software and Systems Modeling pp. 25–47 (2008)
51. Mendez, D., Etien, A., Muller, A., Casallas, R.: Towards Transformation Migration After Metamodel Evolution. In: Models and Evolution Workshop (2010)
52. Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Sci. Comput. Program. **76**(12), 1223–1246 (2011)
53. Meyers, B., Wimmer, M., Cicchetti, A., Sprinkle, J.: A generic in-place transformation-based approach to structured model coevolution. In: Proceedings of MPM Workshop (2010)
54. Morales, R., Chicano, F., Khomh, F., Antoniol, G.: Efficient refactoring scheduling based on partial order reduction. J. Syst. Softw. **145**, 25–51 (2018)
55. Morales, R., Soh, Z., Khomh, F., Antoniol, G., Chicano, F.: On the use of developer's context for automatic refactoring of software anti-patterns. J. Syst. Softw. **128**, 236–251 (2017)
56. Muflikhah, L., Baharudin, B.: Document clustering using concept space and cosine similarity measurement. In: Proceedings of ICCTD (2009)
57. Narayanan, A., Levendovszky, T., Balasubramanian, D., Karsai, G.: Automatic domain model migration to manage metamodel evolution. In: Proceedings of MODELS (2009)
58. Omg, Q.: Meta object facility (mof) 2.0 query/view/transformation specification. Final Adopted Specification (November 2005) (2008)
59. Paige, R.F., Matragkas, N.D., Rose, L.M.: Evolving models in model-driven engineering: state-of-the-art and future challenges. J. Syst. Softw. **111**, 272–280 (2016)
60. 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)
61. 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)
62. Rebai, S., Alizadeh, V., Kessentini, M., Fehri, H., Kazman, R.: Enabling decision and objective space exploration for interactive multi-objective refactoring. IEEE Transactions on Software Engineering (2020)
63. Redner, R.A., Walker, H.F.: Mixture densities, maximum likelihood and the EM algorithm. SIAM Rev. **26**(2), 195–239 (1984)
64. Richters, M.: A precise approach to validating UML models and OCL constraints. Tech. rep. (2001)
65. Roddick, J.F.: Schema evolution in database systems: an annotated bibliography. SIGMOD Rec. **21**(4), 35–40 (1992)
66. Rose, L.M., Herrmannsdoerfer, M., Mazanek, S., Gorp, P.V., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schatz, B., Wimmer, M.: Graph and model transformation tools for model migration - empirical results from the transformation tool contest. Softw. Syst. Model. **13**(1), 323–359 (2014)
67. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Model migration with Epsilon Flock. In: Proceedings of ICMT (2010)
68. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C.: An Analysis of Approaches to Model Migration. In: Proceedings of MoDSE-MCCM Workshop (2009)
69. Ruscio, D.D., Etzlstorfer, J., Iovino, L., Pierantonio, A., Schwinger, W.: Supporting variability exploration and resolution during model migration. In: Proceedings of ECMFA (2016)
70. Sahin, D., Kessentini, M., Wimmer, M., Deb, K.: Model transformation testing: a bi-level search-based software engineering approach. Journal of Software: Evolution and Process **27**(11), 821–837 (2015)
71. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: A case study in software product lines. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13 (2013)
72. Schoenboeck, J., Kusel, A., Etzlstorfer, J., Kapsammer, E., Schwinger, W., Wimmer, M., Wischenbart, M.: CARE: A Constraint-based Approach for Re-establishing Conformance-relationships. In: Proceedings of APCCM (2014)
73. Sprinkle, J., Karsai, G.: A domain-specific visual language for domain model evolution. J. Vis. Lang. Comp. **15**(3–4), 291–307 (2004)
74. Taentzer, G., Mantz, F., Arendt, T., Lamo, Y.: Customizable model migration schemes for meta-model evolutions with multiplicity changes. In: Proceedings of MODELS (2013)
75. Vermolen, S., Visser, E.: Heterogeneous coupled evolution of software languages. In: Proceedings of MODELS (2008)
76. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: Proceedings of ECOOP (2007)
77. Wimmer, M., Kusel, A., Schoenboeck, J., Retschitzegger, W., Schwinger, W.: On using inplace transformations for model coevolution. In: Proceedings of MtATL Workshop (2010)

Wael Kessentini is an assistant professor in the College of Computing and Digital Media (CDM) at DePaul University (Chicago, USA). He obtained his Ph.D. degree in Computer Science in 2018 from the Université de Montréal (Canada). Also he holds M.Sc and B.Sc. degrees in Computer Science from University of Tunis (Tunisia). His research interests focus on Software Quality, Software Evolution, Model-Driven Engineering, Search-based Software Engineering.

Vahid Alizadeh is an assistant professor in the College of Computing and Digital Media (CDM) at DePaul University (Chicago, USA). He obtained his Ph.D. degree in Computer Science in 2020 from University Of Michigan (USA). Also, he holds M.Sc. and B.Sc. degrees in Electrical Engineering from University of Tehran (Iran) and Iran University of Science and Technology (Iran), respectively. His research interests focus on Empirical Software Engineering, Software Quality, Software Refactoring / Maintenance, Intelligent / Search-based Software Engineering (AI4SE), and Software Engineering for AI-Enabled Systems (SE4AI).