Interactive Metamodel/Model Co-Evolution Using Unsupervised Learning and Multi-Objective Search

Wael Kessentini
wkessent@depaul.edu
College of Computing and Digital Media
DePaul University
Chicago, IL, USA

Vahid Alizadeh
alizadeh@depaul.edu
College of Computing and Digital Media
DePaul University
Chicago, IL, USA

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. In this paper, we propose an interactive multi-objective approach that dynamically adapts and interactively 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. The proposed approach proposes to the user few regions of interest by clustering the set of recommended co-evolution solutions of the multi-objective search. Thus, users 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.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**.

KEYWORDS

Metamodel/Model Co-Evolution, Interactive Multi-objective Search, Search Based Software Engineering

ACM Reference Format:
Wael Kessentini and Vahid Alizadeh. 2020. Interactive Metamodel/Model Co-Evolution Using Unsupervised Learning and Multi-Objective Search . In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3365438.3410966

MODELS '20, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-7019-6/20/10…$15.00
https://doi.org/10.1145/3365438.3410966

1 INTRODUCTION

There is an urgent need to find better ways to evolve software systems and, consequently, improve developers' productivity. Like source code, the 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) [5]. 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 make them conform with 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 [21, 40].

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 [7, 9, 12, 30, 31]. 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 [23, 24, 40]. Several techniques proposed to translate metamodel changes into model level edit operations using a set of generic transformation rules [16, 18, 20, 44]. However, several transformations require interactions with the user especially when new elements are added to the new meta-model.

Recently, an approach has been proposed to interactively evaluate the co-evolved models using search-based software engineering [25]. 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.

In this paper, we propose an interactive approach that combines multi-objective search (NSGA-II [10]), interactive optimization, and unsupervised learning to reduce the designer's interaction effort when co-evolving models. 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, to guide the designers in selecting their region of interests and reduce the effort to explore the Pareto-front. 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 preferences 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 selected 16 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 [8, 45]. Furthermore, we compared our approach to existing fully automated co-evolution techniques [23, 24, 45] and to an interactive technique [25]. The manual evaluation of the revised models to meet new metamodel changes confirms the effectiveness of our clustering-based interactive approach.

2 BACKGROUND AND MOTIVATING EXAMPLE

In MDE, metamodels are the means to specify the abstract syntax of modeling languages [5]. 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 [21, 40].

Figure 1 shows an example of a simplified metamodel evolution, based on simple staff modeling language taken from [39] 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 constraints violations, the constraint shown in Listing 1 is violated when considering the initial model shown in Figure 1c and its conformance to the new metamodel version in Figure 1b.

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)
```

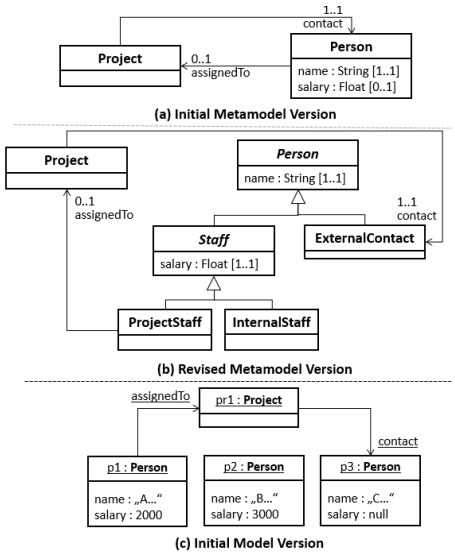

Figure 1: Example of metamodel evolution.

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

Several co-evolution studies proposed to revise models after metamodels evolution from manual to fully automated approaches [14]. Recently, few automated/interactive tools [23–25] used search-based software engineering to generate revised models. The proposed tools refines 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 designer has to inspect a large list of potential good recommended solutions, which is time consuming, and designer may miss to select the most preferred once. 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 considerable time to understand the

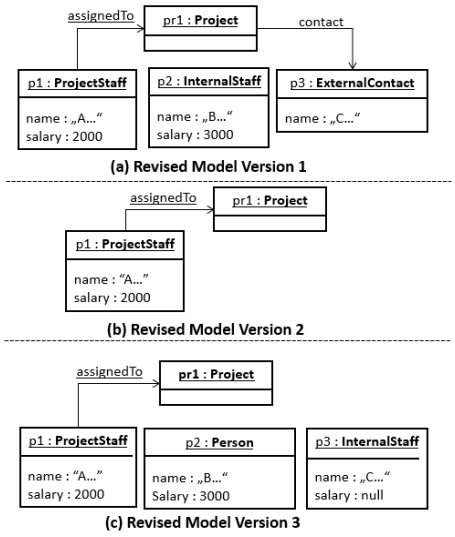

Figure 2: Example model co-evolution.

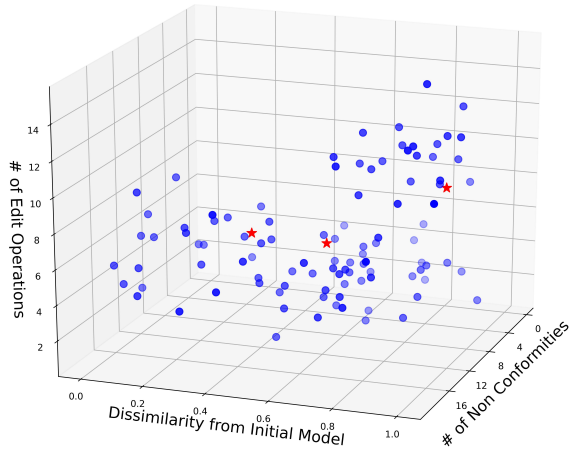

Figure 3: Pareto front of co-evolution solutions generated by just using the multi-objective search.

differences between the solutions and their impacts on the different co-evolution objectives.

Figure 3 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 solutions that are represented in a shape of stars in Figure 3. This figure shows that there may be several possible solutions where the user has to decide which one to select in the search space based on the preferences. Thus, there is to reduce the search space in the next iterations and reduce the interactions effort using the feedback from the user.

3 CLUSTERING-BASED INTERACTIVE MULTI-OBJECTIVE MODEL CO-EVOLUTION

The general structure of our approach is sketched in Fig. 4. Our approach includes three 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 [10], 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 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. 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 the non-dominated sorting genetic algorithm (NSGA-II) to interactively find a trade-off between three objectives that will be described later. A multi-objective optimization problem can be formulated in the following form:

$$\begin{aligned} &\textit{Minimize} && F(x) = (f_1(x), f_2(x), ..., f_M(x)), \\ &\textit{Subject to} && x \in S, \\ & && S = \{x \in R^m : h(x) = 0, g(x) \geq 0\}; \end{aligned}$$

where S is the set of inequality and equality constraints and the functions f_i are *objective* or *fitness* functions. In multi-objective optimization, the quality of a solution is recognized by dominance. The set of feasible solutions that are not dominated by any other solution is called *Pareto-optimal* or *Non-dominated* solution set.

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,

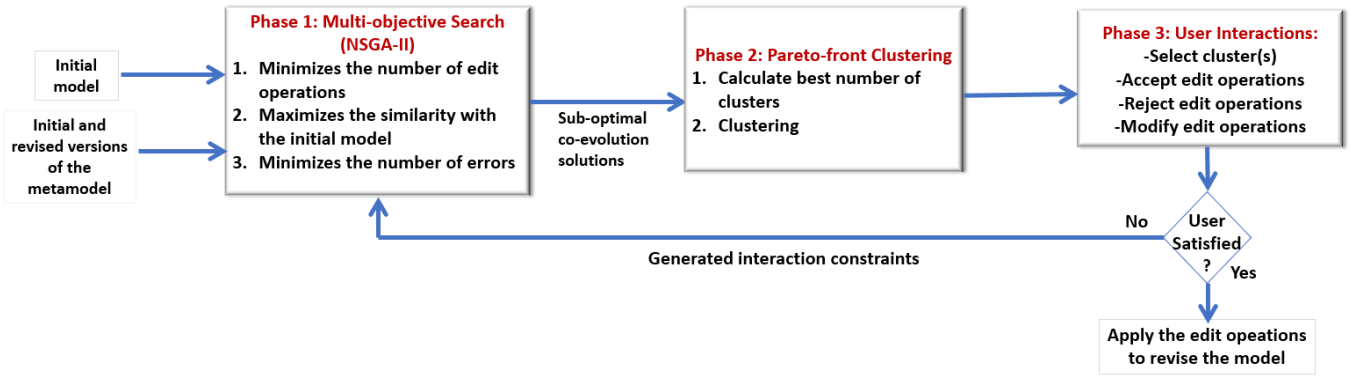

Figure 4: High-level overview of the proposed interactive clustering co-evolution approach.

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.

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 [19]. 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 5 represents a solution that can be applied to the initial model of our motivating example described in Section 2.

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.

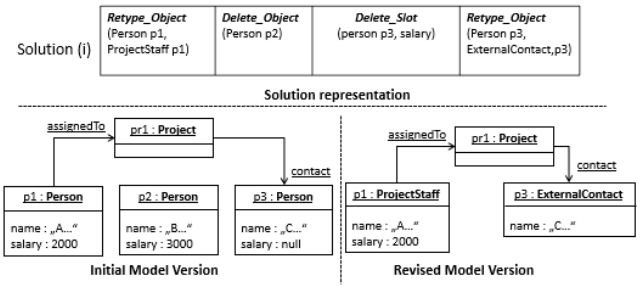

Figure 5: Solution representation.

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.

Table 1: Model edit operations.

The first fitness function $nvc(s)$ counts the number of violated constraints w.r.t. the evolved metamodel after applying a sequence s of edit operations. We apply, first, the sequence of edit operations (solution) on the initial model then we load the evolved model on the target metamodel to measure the number of conformance errors based on the number of violated constraints. We consider three types of constraints, as described in [36]: 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. [40] and Richters et al. [36] 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. The full list constraints can be found in this link [1]

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

For the second fitness function, which aims at minimizing the changes to the initial models, we simply count the number of edit operations $nbOp(s)$ of a solution s (size of s). The third fitness function dis(s) measures the difference between the model elements in the initial and revised model. As the type of a model element may change because of a change in the metamodel, we cannot rely on elements' types. Alternatively, we use the identifiers to assess whether information was added or deleted when editing a model. In this case, the renamed or extracted model elements will be considered different than the initial model element. Thus, we considered the assumption that two model elements could be syntactically similar if they use a similar vocabulary. Thus, we calculated for the textual similarity based on the Cosine similarity [32]. In the first step, we tokenize the names of initial and revised model elements. The textual and/or context similarity between elements grouped together to create a new class is an important factor to evaluate the cohesion of the revised model. The initial and revised models are represented as vectors of terms in n-dimensional space where n is the number of terms in all considered models. For each model, a weight is assigned to each dimension (representing a specific term) of the representative vector that corresponds to the term frequency score (TF) in the model. The similarity among initial and revised model elements is measured by the cosine of the angle between its representative vectors as a normalized projection of one vector over the other. The cosine measure between a pair of model elements, A and B, is defined as follows:

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

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

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

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

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

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

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 10 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;
```

3.2 Phase 2: Clustering the Pareto Front of Co-Evolution Solutions

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. is an internal clustering validation measure based on two criteria: compactness and separation [6]. CH evaluates the clustering results based on the average sum of squares between and within clusters and it defines as follows:

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

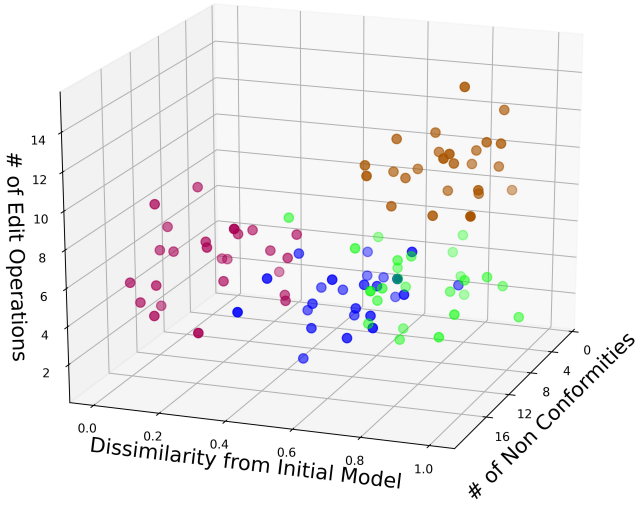

Figure 6: Interactive solution charts in our tool.

where N is the size of data, K is the number of clusters, $dist(a, b)$ is the Euclidean distance, $\overline{c_k}$ 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). 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 [35] 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:

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

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.

Figure 6 shows the result of the clustered solutions presented in our motivating example via interactive colored graphical charts. This figure shows that the clustering feature identified four main different clusters.

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 $\leftarrow$ cluster with Max score;
begin User Preference Extraction
  /* Representative solution as reference */
  RS $\leftarrow CR_{PC}$;
  foreach $[eo_i, elt_i] \in PC$ **do**
    $OWP_p \leftarrow$ AverageWeightedFreq(eo_p);
    $MWP_q \leftarrow$ AverageWeightedFreq(elt_q);
Return PC, Preference Parameters[];

3.3 Phase 3: Developers Interaction 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. 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].

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. After the interaction is done and the user decides to continue to the next round, the score of each solution and cluster 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 achieved 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, these parameters can be computed as follow:

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

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

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.

3.4 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.

4 EVALUATION

4.1 Research Questions

We defined two main research questions to measure the correctness, relevance and benefits of our interactive clustering-based multi-objective model co-evolution tool(IC-NSGA-II) comparing to : (1) an approach based on interactive multi-objective search (I-NSGA-II) [25] 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 automated multi-objective co-evolution approach (without the interaction component) [24] and (3) an existing automated co-evolution approach based on pre-defined rules without using search methods [45].

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 clustering relevance.** To what extent can our clustering-based approach **efficiently** reduce the interaction effort?

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 [8, 45]. These case studies are interesting scenarios since they represent real metamodel evolutions, used in an empirical study [17] and studied in other contributions [11, 15, 37]. 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 [8, 45] thus we were not involved in the selection of models and metamodel changes. In order to ensure a fair comparison with Wimmer et al. [45], 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 work of Kessentini et al. [25]

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

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

since it is the only approach offering interaction with the user, and it will help us understand the real impact of clustering the Pareto 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 interactive clustering-based approach, an existing interactive multi-objective approach [25] (without the clustering feature) and the two automated techniques of Kessentini et al. [23] and Wimmer et al. [45].

4.3 Study Participants and Parameters Setting

Our study involved 16 master students in Software Engineering. All the participants are volunteers and familiar with model-driven engineering and co-evolution/refactoring since they are part of a graduate course on Software Testing & Quality Assurance and most of them participated in similar experiments in the past, either as part of a research project or during graduate courses. Furthermore, 12 out the 16 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 4 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 developers' 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 [34].

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 [22].

4.4 Results

Results for RQ1: Co-evolution relevance. We report the results of the empirical qualitative evaluation (MC) in Figure 7. 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 92% of generated edit operations. The remaining approaches have an average of 89% and 87% respectively for the interactive multi-objective approach [25] and the fully automated multi-objective approach [24]. Both of the interactive tools outperformed fully-automated ones which shows the importance of integrating the human in the loop when co-evolving models. Furthermore, it is clear that adding the clustering feature to enable 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 7 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. [45] provide a set of co-evolution rules. Further adaptations are required to make this set of rules working on other metamodels.

A qualitative analysis of the results show 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 not relevant to the designers' context or 2) emphasizing specific cluster that prioritizes some objectives 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

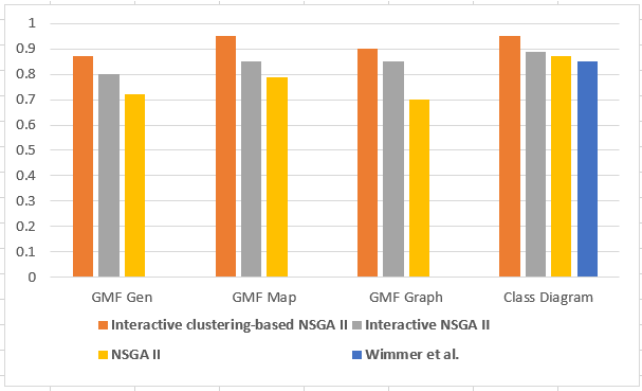

Figure 7: The median manual evaluation scores, MC, on the four metamodel evolution scenarios with 95% confidence level (α = 5%)

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

	Approaches			
Metamodels	IC-NSGA-II (T,NI)		I-NSGA-II. (T,NI)	
GMF Gen	52	20	71	32
GMF Map	34	16	55	25
GMF Graph	40	24	83	35
Class Diagram	20	8	39	5

higher than 0.82 for GMF GEN, GMF MAP, GMF Graph; and an A effect size higher than 0.88 for Class Diagram.

Results for RQ2: Interactive clustering relevance. Table 3 summarizes the time, in minutes, and the number of interaction with the participants to find the most relevant solutions using our tool (IC-NSGA-II), and the interactive approach (I-NSGA-II) of Kessentini et al. [25]. All the participants spent less time to find the most relevant model edit operations on the different metamodels comparing to I-NSGA-II. For instance, the average time is reduced from 71 minutes to just 52 minutes for the case of GMF Gen. The time includes the execution of IC-NSGA-II and the different phases of interaction until the designer is satisfied with a specific solution. It is clear as well that the time reduction is not correlated with the number of interaction. For instance, the deviation between IC-NSGA-II and I-NSGA-II for GMF Graph in terms of number of interaction is limited to 9 (24 vs 35) but the time reduction is 43 minutes. However, it is clear that our approach reduced as well the number of interaction comparing to I-NSGA-II. while increasing the manual correctness as described in RQ1.

Figure 8 shows a qualitative example extracted from our experiments using our tool with a population size of 100 based on three phases of interactions. After the generation of the Pareto front, the clustering feature identified four main different clusters . After the clustering phase, the user selected the solution that belongs to the purple cluster as the preferred one after exploring several solutions. Thus, the next iterations of IC-NSGA-II prioritized that "region of interest", so more solutions were generated based on his preference. Figure 8 shows the new reduced space of solutions with three new clusters generated around the previously selected cluster and more solutions are generated towards the preferred cluster.

14 out the 16 participants mention, during the post-study questionnaire, that our interactive clustering-based tool is faster and much easier to use than the one without the clustering component [25] to identify quickly relevant edit operations based on their interests. Similar observation is valid when comparing our tool to the fully-automated multi-objective tool [24]. 12 out of the 16 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 [45]. 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 [4] 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. Future replications of this study are necessary to confirm our findings.

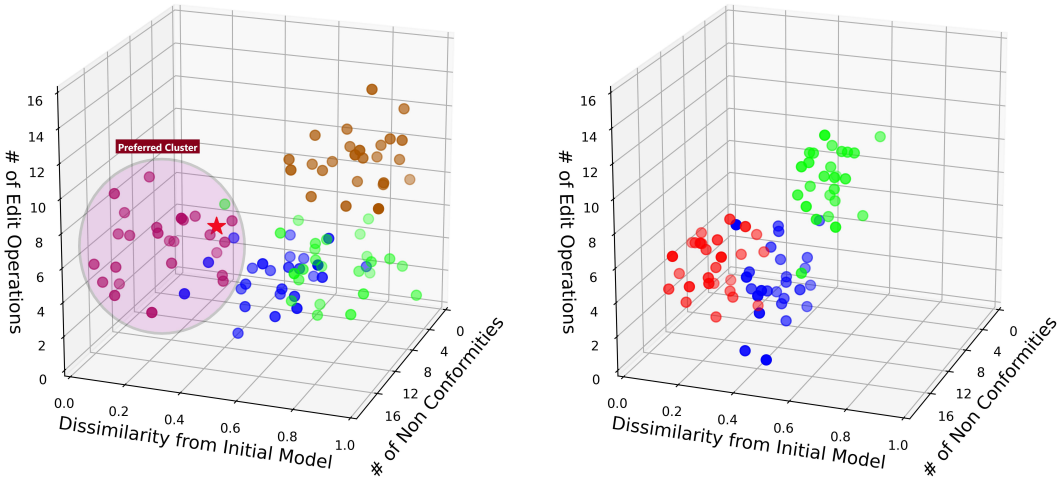

Figure 8: Illustration of the co-evolution solutions moving towards a region of interest selected by the user as extracted from the experiments.

6 RELATED WORK

6.1 Manual specification approaches

In one of the early works [41], the co-evolution of models is tackled by designing co-evolution transformations based on metamodel change types. In [9, 12], the authors compute differences between two metamodel versions which are then input to adapt models automatically. This is achieved by transforming the differences into a migration transformation with a so-called higher-order transformation, i.e., a transformation which takes/produces another transformation as input/output. In [13], the authors proposed an approach compromises multiple steps for model co-evolution: change detection either by comparing between metamodels or by tracing and recording the changes applied to the old version of the metamodel. The second step is a classification of the changes in metamodel and their impact in its instances. Finally, an appropriate migration algorithm for model migration is determined. For initial model elements for which no transformation rule is found, a default copy transformation rule is applied. This algorithm has been realized in the model migration framework Epsilon Flock [38] and in the framework described in [33]. Another manual specification approach is presented in [43] where a specific transformation language is derived to describe the evolution on the metamodel level and derive a transformation for the model level.

6.2 Metamodel matching approaches

Most of the mentioned approaches which are based on out-place model transformation intend to shield the user from creating copy rules. 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., [20, 26, 27, 31, 42]), 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.

6.3 Operator-based approaches

Other contributions are based on using coupled operations [16, 18, 20, 28, 29, 44]. In [44] the authors provide a library of co-evolutionary operators for MOF metamodels, each of these operators provides a model migration strategy. In [16] the author provides a tool support for the evolution of Ecore-based metamodels, that records the metamodel changes as a sequence of co-evolutionary operations that are structured in a library and used later to generate a complete migration strategy. But, when no appropriate operator is available, model developer does the migration manually, so those approaches depend on the library of reusable coupled operators they provide. To this end, the authors in [20] 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 [3], an approach is presented that uses in a first phase metamodel matching to derive the changes between two metamodel versions and in a second phase, operations are applied based on the detected changes to migrate the corresponding models. Additionally, in [2], weaving models are employed to connected the changes of the metamodels with the model elements to provide a basis for reasoning how to perform the migration of the models to the new metamodel versions.

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.

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 solutions are used to reduce the search space and converge to better solutions. 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.

REFERENCES

[1] [n.d.]. Constraints for model co-evolution: https://docs.google.com/document/d/1O1GjOcvvBgPuVacB12noygNaJ75y0nwySdGmiqGXouQ.
[2] F. Anguel, A. Amirat, and N. Bounour. 2014. Using weaving models in metamodel and model co-evolution approach. In *Proceedings of CSIT*.
[3] Fouzia Anguel, Abdelkrim Amirat, and Nora Bounour. 2015. Hybrid Approach for Metamodel and Model Co-evolution. In *Proceedings of CIIA*.
[4] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of ICSE*.
[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers.
[6] Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods* 3, 1 (1974), 1–27.
[7] Antonio Cicchetti, Federico Ciccozzi, Thomas Leveque, and Alfonso Pierantonio. 2011. On the concurrent versioning of metamodels and models: challenges and possible solutions. In *Proceedings IWMCP*.
[8] Antonio Cicchetti, Federico Ciccozzi, Thomas Leveque, and Alfonso Pierantonio. 2011. On the concurrent versioning of metamodels and models: Challenges and possible solutions. In *Proceedings of IWMCP*.
[9] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. 2008. Automating co-evolution in model-driven engineering. In *Proceedings of EDOC*.
[10] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. 2000. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Proceedings of PPSN*.
[11] Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. 2011. Automated Co-evolution of GMF Editor Models. In *Proceedings of SLE*.
[12] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. 2009. Managing model adaptation by precise detection of metamodel changes. In *Proceedings of ECMFA*.
[13] Boris Gruschko. 2007. Towards synchronizing models with evolving metamodels. In *Proceedings of MoDSE Workshop*.
[14] Regina Hebig, Djamel Eddine Khelladi, and Reda Bendraou. 2017. Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering* 43, 5 (2017), 396–414.
[15] Markus Herrmannsdoerfer. 2011. GMF: A Model Migration Case for the Transformation Tool Contest. In *Proceedings of TTC*.
[16] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. 2009. COPE - Automating Coupled Evolution of Metamodels and Models. In *Proceedings of ECOOP*.
[17] Markus Herrmannsdoerfer, Daniel Ratiu, and Guido Wachsmuth. 2010. Language Evolution in Practice: The History of GMF. In *Proceedings of SLE*.
[18] Markus Herrmannsdoerfer, SanderD. Vermolen, and Guido Wachsmuth. 2011. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In *Proceedings of SLE*.
[19] Markus Herrmannsdoerfer, Sander D. Vermolen, and Guido Wachsmuth. 2011. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In *Proceedings of SLE*.
[20] Markus Herrmannsdörfer. 2011. COPE - A Workbench for the Coupled Evolution of Metamodels and Models. In *Proceedings of SLE*.
[21] Ludovico Iovino, Alfonso Pierantonio, and Ivano Malavolta. 2012. On the Impact Significance of Metamodel Evolution in MDE. *Journal of Object Technology* 11, 3 (2012), 3:1–33.
[22] Robert Jackson, Chris Carter, and Michael Tarsitano. 2001. Trial-and-Error Solving of a Confinement Problem by a Jumping Spider, Portia fimbriata. *Behaviour* 138, 10 (2001), 1215–1234.
[23] Wael Kessentini, Houari A. Sahraoui, and Manuel Wimmer. 2016. Automated Metamodel/Model Co-evolution Using a Multi-objective Optimization Approach. In *Proceedings of ECMFA*.
[24] Wael Kessentini, Houari A. Sahraoui, and Manuel Wimmer. 2019. Automated metamodel/model co-evolution: A search-based approach. *Inf. Softw. Technol.* 106 (2019), 49–67.
[25] Wael Kessentini, Manuel Wimmer, and Houari A. Sahraoui. 2018. Integrating the Designer in-the-loop for Metamodel/Model Co-Evolution via Interactive Computational Search. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018*, Andrzej Wasowski, Richard F. Paige, and Øystein Haugen (Eds.). ACM, 101–111.
[26] Florian Mantz, Yngve Lamo, and Gabriele Taentzer. 2013. Co-Transformation of Type and Instance Graphs Supporting Merging of Types with Retyping. *ECEASST* 61 (2013), 24.
[27] Florian Mantz, Gabriele Taentzer, Yngve Lamo, and Uwe Wolter. 2015. Co-evolving meta-models and their instance models: A formal approach based on graph transformation. *Sci. Comput. Program.* 104 (2015), 2–43.
[28] Josh Mengerink, Ramon R. H. Schiffelers, Alexander Serebrenik, and Mark van den Brand. 2016. DSL/Model Co-Evolution in Industrial EMF-Based MDSE Ecosystems. In *Proceedings of the 10th Workshop on Models and Evolution co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, October 2, 2016.* 2–7.
[29] J. G. M. Mengerink, Alexander Serebrenik, Ramon R. H. Schiffelers, and M. G. J. van den Brand. 2016. A Complete Operator Library for DSL Evolution Specification. In *ICSME 2016*. 144–154.
[30] Bart Meyers and Hans Vangheluwe. 2011. A framework for evolution of modelling languages. *Sci. Comput. Program.* 76, 12 (2011), 1223–1246.
[31] Bart Meyers, Manuel Wimmer, Antonio Cicchetti, and Jonathan Sprinkle. 2010. A generic in-place transformation-based approach to structured model co-evolution. In *Proceedings of MPM Workshop*.
[32] Lailil Muflikhah and Baharum Baharudin. 2009. Document clustering using concept space and cosine similarity measurement. In *Proceedings of ICCTD*.
[33] Anantha Narayanan, Tihamer Levendovszky, Daniel Balasubramanian, and Gabor Karsai. 2009. Automatic Domain Model Migration to Manage Metamodel Evolution. In *Proceedings of MODELS*.
[34] Alexander Pollatsek and Arnold D Well. 1995. On the use of counterbalanced designs in cognitive research: A suggestion for a better and more powerful analysis. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 21, 3 (1995), 785.
[35] Richard A Redner and Homer F Walker. 1984. Mixture densities, maximum likelihood and the EM algorithm. *SIAM review* 26, 2 (1984), 195–239.
[36] Mark Richters. 2001. *A precise approach to validating UML models and OCL constraints*. Technical Report.
[37] Louis M. Rose, Markus Herrmannsdoerfer, Steffen Mazanek, Pieter Van Gorp, Sebastian Buchwald, Tassilo Horn, Elina Kalnina, Andreas Koch, Kevin Lano, Bernhard Schätz, and Manuel Wimmer. 2014. Graph and model transformation tools for model migration - Empirical results from the transformation tool contest. *Software and System Modeling* 13, 1 (2014), 323–359.
[38] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. 2010. Model migration with Epsilon Flock. In *Proceedings of ICMT*.
[39] Davide Di Ruscio, Juergen Etzlstorfer, Ludovico Iovino, Alfonso Pierantonio, and Wieland Schwinger. 2016. Supporting Variability Exploration and Resolution During Model Migration. In *Proceedings of ECMFA*.
[40] Johannes Schoenboeck, Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Wieland Schwinger, Manuel Wimmer, and Martin Wischenbart. 2014. CARE: A Constraint-based Approach for Re-establishing Conformance-relationships. In *Proceedings of APCCM*.
[41] Jonathan Sprinkle and Gabor Karsai. 2004. A Domain-Specific Visual Language for Domain Model Evolution. *J. Vis. Lang. Comput.* 15, 3-4 (2004), 291–307.
[42] Gabriele Taentzer, Florian Mantz, Thorsten Arendt, and Yngve Lamo. 2013. Customizable Model Migration Schemes for Meta-model Evolutions with Multiplicity Changes. In *Proceedings of MODELS*.
[43] Sander Vermolen and Eelco Visser. 2008. Heterogeneous Coupled Evolution of Software Languages. In *Proceedings of MODELS*.
[44] Guido Wachsmuth. 2007. Metamodel adaptation and model co-adaptation. In *Proceedings of ECOOP*.
[45] M. Wimmer, A. Kusel, J. Schoenboeck, W. Retschitzegger, and W. Schwinger. 2010. On using inplace transformations for model co-evolution. In *Proceedings of MtATL Workshop*.