Integrating the Designer in-the-loop for Metamodel/Model Co-Evolution via Interactive Computational Search

Wael Kessentini
University of Montreal
Montreal, Quebec, Canada
kessentw@iro.umontreal.ca

Manuel Wimmer
CDL-MINT, TU Wien
Vienna, Austria
wimmer@big.tuwien.ac.at

Houari Sahraoui
University of Montreal
Montreal, Quebec, Canada
sahraouh@iro.umontreal.ca

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. In this paper, we propose an interactive multi-objective approach that dynamically adapts and interactively suggests edit operations to developers and takes their feedback into consideration. Our approach uses NSGA-II to find a set of good edit operation sequences that minimizes the number of conformance errors, maximizes the similarity with the initial model (reduce the loss of information) and minimizes the number of proposed edit operations. The designer can approve, modify, or reject each of the recommended edit operations, and this feedback is then used to update the proposed rankings of recommended edit operations. We evaluated our approach on a set of metamodel/model co-evolution case studies and compared it to fully automated co-evolution techniques.

KEYWORDS

Coupled Evolution, Interactive Optimization, Metamodel/Model Co-Evolution, Search Based Software Engineering

ACM Reference Format:
Wael Kessentini, Manuel Wimmer, and Houari Sahraoui. 2018. Integrating the Designer in-the-loop for Metamodel/Model Co-Evolution via Interactive Computational Search. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239375

1 INTRODUCTION

The evolution of metamodels, models, and transformation rules is inevitable in model-driven engineering (MDE) [5]. The different evolution scenarios can range from requirements change to fixing errors and improving several quality aspects. With the current growing adoption of MDE in practice (e.g. automotive industry, health-care, etc.) the support of evolution activities is becoming critical. A large number of instance models has to be updated even after small changes introduced to metamodels.

Several co-evolution studies proposed to repair models after metamodels evolution [12]. We can classify them into two categories: manual and fully-automated approaches. In manual co-evolution [7, 10, 37], the designer has to identify the parts of the models that require attention and trigger the model updates by hand. The main support provided by these manual approaches is the automated execution of the edit operations selected manually by the designer such as retype model elements, extract elements, etc. However, the manual co-evolution is error-prone, time-consuming, not scalable and not useful for radical changes that require an extensive application of edit operations to correct model instances after metamodels evolution [34]. In fully-automated co-evolution [4], the goal is to find an entire edit operations sequence that revise models in accordance with the new metamodel version. Several techniques proposed to translate metamodel changes into model level edit operations using a set of generic transformation rules [14, 17, 19, 40]. Recently, few automated approaches [4, 22, 24] used search-based software engineering to generate a revised model that minimizes as much as possible the number of conformance errors. These automated co-evolution tools are appealing since they require little designer effort, but they suffer from several limitations as well. Firstly, they lack flexibility since the designer has to either accept or reject the entire co-evolution solution. Secondly, they fail to consider the designer perspective, as the designer has no opportunity to provide feedback on the model edit operations solution as it is being created. Thirdly, the designers have to accept the entire co-evolution solution even though they prefer, in general, step-wise approaches where the process is interactive and they have control of the model changes being applied. Finally, it is almost impossible to co-evolve metamodel changes that impact the semantics without the input of the designer.

To address the above challenges, we propose an approach to metamodel/model co-evolution that supports a co-evolution solution centric interaction. We postulate that enabling the designer to interact with the co-evolution solution is essential both to create a better revised model as well as to create model-level edit operations that support any metamodel change not limited to the ones that preserve the semantics of languages. In this paper, we designed and implemented an interactive multi-objective approach that dynamically adapts and interactively suggests edit operations to developers and takes their feedback into consideration. Our approach uses NSGA-II [8] to find a set of good edit operation sequences that minimizes the number of conformance errors, maximizes the

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239375

similarity with the initial model (reduce the loss of information) and minimizes the number of proposed edit operations. The designer can approve, modify, or reject each of the recommended edit operations, and this feedback is then used to update the proposed rankings of recommended edit operations. We evaluated our approach 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 [6, 41]. Furthermore, we compared our approach to fully automated co-evolution techniques [22, 24, 41]. The manual evaluation of the revised models to meet new metamodel changes confirms the effectiveness of our interactive approach.

The remainder of this paper is structured as follows. Section 2 presents the background and a motivating example. Section 3 describes our approach to interactive metamodel/model co-evolution while the results obtained from our experiments are presented and discussed in Section 4. Threats to validity are discussed in Section 5. Section 6 provides a discussion of related work. Finally, in Section 7, we summarize our conclusions and present some ideas for future work.

2 BACKGROUND AND MOTIVATING EXAMPLE

This section introduces the necessary background for this paper, namely the metamodel/model co-evolution problem as well as a motivating example to demonstrate the challenges related to this co-evolution problem.

In MDE, metamodels are the means to specify the abstract syntax of modeling languages [5]. Theoretically speaking, metamodels give the intentional description of all possible models of a given language. In practice, metamodels are instantiated to produce models which are, in essence, object graphs, i.e., consisting of objects (instances of classes) representing the modeling elements, object slots for storing values (instances of attributes), and links between objects (instances of references). 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 [20, 36].

If metamodels are changing, the conformsTo relationship to already existing models may no longer be given. In this context, it is important to categorize the consequences of metamodel changes on the models [7][1].

- **Non-breaking operations**: A change at the metamodel level requires no migrations of the instances, e.g., adding a new optional attribute to a metaclass.
- **Breaking and resolvable operations**: A change at the metamodel level is reflected on the instance level by an automatic migration, e.g., adding a mandatory attribute with a default value to a metaclass.
- **Breaking and unresolvable operations**: Some changes at the metamodel level need complex migrations which possibly require user input to introduce additional information to the models, e.g., adding a mandatory attribute for which no default value is available to a metaclass.

[1]Please note that this change categorization is solely based on syntactical considerations and that the semantic interpretation of changes as well as of their resolution may lead to different categorizations [12].

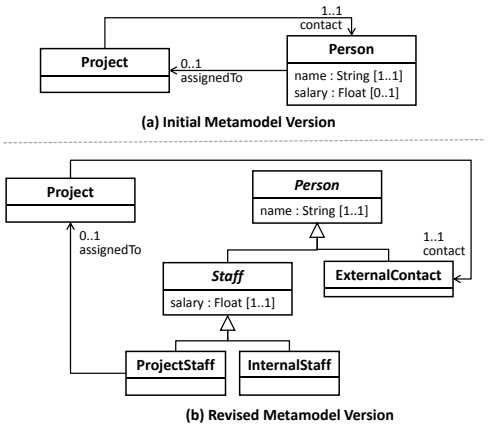

Figure 1: Metamodel evolution example.

Depending on the complexity of the metamodel evolution, the migration transformation needed to adapt the models to the new metamodel version may be derived from a difference model with the list of changes done to the metamodel during its evolution. Typically, this can be achieved for breaking and resolvable operations. As an example, consider that a metaclass is deleted, a default migration strategy is to just delete its instance in all models too. This leads to models conforming again to the metamodel from a syntactical point of view, but whether this default migration is valid from a semantical point of view has still to be decided by the user. For instance, for some cases the instances may have to be casted to other metaclasses instead of just being deleted.

While a lot of research has been invested for breaking and resolvable operations, only limited support is currently provided for breaking and unresolvable changes as is also noted in [15]. To make the challenge of supporting unresolvable changes more concrete, we are introducing a motivating example.

Figure 1 shows an example of a simplified metamodel evolution, based on simple staff modeling language taken from [35]. 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 and the contact links are now only allowed to point to *ExternalContact* instances. This evolution results in the fact that decisions have to be done during the migration of existing instances. In particular, we have to consider how to deal with a stricter metamodel version with respect to the distribution of the previous properties of the *Person* class over several partially disjoint classes.

To re-establish conformance of existing models for the given example, assume for now that only one operation type on models is used in this context. Non-conforming objects may be *retyped* (reclassified as instances of the concrete classes). Thus, several possibilities exist now how to downcast *Person* instances and this may have different consequences such as which information may be lost of *Person* instances. Of course, some situations can be automated such as persons who are assigned to a project and not having an incoming contact link are retyped to instances of *ProjectStaff*.

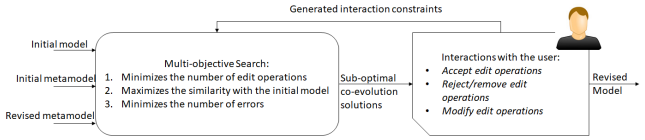

Figure 2: High-level overview of the proposed interactive co-evolution approach.

However, we may also have persons who have a salary, are the contact for a particular project, and may be assigned to a project as staff. As we do not assume an overlapping inheritance semantics, but a disjoint one, we have to properly decide how such persons are treated in the new metamodel structure.

In previous work [24], we have shown how to use meta-heuristic search to find interesting solutions to such metamodel/model co-evolution problems. However, in particular cases, as we see also in this motivating example, there may be several solutions which have the same fitness (same number of values are lost, same number of constraints of the conformsTo relationship are violated), and thus, several equally good solutions exists where the user has to decide which solution to take, and potentially, which additional changes to (un)apply. To this end, we consider the metamodel/model co-evolution problem worth to be treated with interactive search which should support the user to better understand how solutions are computed, allow to take decisions during the search process, and to react to feedback from the user to improve the search process to better consider the user preferences. In the next section, we show how such an approach can be realized based on well-established search algorithms.

3 INTERACTIVE METAMODEL/MODEL CO-EVOLUTION

We describe, in this section, the overview of our interactive approach, and subsequently, we explain the different adaptation steps of the proposed search algorithm to the metamodel/model co-evolution problem including the solutions generation, the evaluation functions, and the interaction phase.

3.1 Overview

The main objective of this work is to enable designers to interactively update the generated edit operations (co-evolution solution) on the initial model and consider that feedback to generate a target model conforming to the new metamodel. The general structure of our approach is sketched in Fig. 2.

Our approach includes two main components. The first component is the multi-objective algorithm, NSGA-II, executed for a number of iterations to find the non-dominated co-evolution solutions, defined as a set of edit operations applied to the initial model, balancing the three objectives of minimizing the number of suggested edit operations, maximizing the similarity with the initial model, and minimizing the number conformance errors with the revised metamodel.

The output of this first step is a set of Pareto-equivalent co-evolution solutions that optimizes the above three objectives. NSGA-II is able to generate not only one good co-evolution solution, but a diverse set of non-dominated solutions. The second component of our approach is to manage the interaction with the designer. It dynamically updates the ranking of recommended co-evolution solutions based on the feedback of the designer by generating a set of constraints to consider in the next iterations of NSGA-II. This feedback can be to approve/apply or modify or reject the suggested edit operations one by one as a sequence of transformations. Thus, the goal is to guide, *implicitly*, the exploration of the Pareto front to find good co-evolution recommendations. Since the ranking is updated dynamically, our interactive algorithm allows the implicit move between non-dominated solutions of the Pareto front.

After a number of interactions, designers may have modified or rejected a high number of suggested edit operations. Whenever the developers stop the interaction session by closing the suggestions window, the first component of our approach is executed again on the background to update the last set of non-dominated co-evolution solutions by continuing the execution of NSGA-II based on the three objectives defined in the first component and also the new constraints summarizing the feedback of the developer. In fact, we consider the rejected edit operations by the developer as constraints to avoid generating solutions containing already several rejected edit operations. The opposite, for accepted edited operations. This may lead to reducing the search space and thus a fast convergence to better solutions. Of course, the continuation of the execution of NSGA-II takes as input the updated version of the system after the interactions with designers. The whole process continues until the designer decides that there is no necessity to update the target model any further.

3.2 Adaptation

We describe in the following subsections the details of the two components of our framework.

3.2.1 Multi-objective formulation. As explained in Algorithm 1, the process starts with a complete execution of a regular NSGA-II algorithm based on the three objectives that will be described later. Then, an interaction phase with the designer is executed to collect feedback. The output of the interaction phase will be used in the next iterations of NSGA-II to update the Pareto front of co-evolution solutions. This loop will be executed until the designer decides to select a revised model after applying the set of edit operations composing the co-evolution solution.

The first iterations of the algorithm identify the Pareto front of the non-dominated co-evolution solutions based on the fitness functions that will be discussed later. Then, during the interactive phase, the designer may accept, reject or modify the edit operation recommendations (Section 3.2.2). Finally, the last dynamic component uses the interaction data with the user to reduce the search space of possible co-evolution solutions and improve the future suggestions by repairing the Pareto front as detailed later the next sub-sections.

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,

Algorithm 1 Interactive NSGA-II at generation t

```
Input
IM, IMM, TMM, Op: intial metamodel, target metamodel, intial model, list of edit operation types, Pt: parent population
Output
P_{t+1}
Begin
/* Test if any user interaction occurred in the previous iteration */
if UserFeedback = TRUE then
    /* Rejected edit operations as constraints */
    Ct ← GetConstraints();
    /* Updated intial model after applying changes */
    Sys ← GetRefactored − System();
    UserFeedback ← FALSE;
end if
S_t ← ∅, i ← 1;
Q_t ← Variation(P_t);
R_t ← P_t ∪ Q_t;
P_t ← evaluate(P_t, C_t, IM, IMM, TMM);
(F_1, F_2, ...) ← NonDominatedSort(R_t);
repeat
    S_t ← S_t ∪ F_i;
    i ← i + 1
until (|S_t| ≥ N)
F_l ← F_i;                                  ▷ //Last front to be included
if |S_t| = N then
    P_{t+1} ← S_t;
else
    P_{t+1} ← ∪_{j=1}^{l−1} F_j;
    /*Number of points to be chosen from F_l */
    K ← N − |P_{t+1}|;
    /*Crowding distance of points in Fl */
    Crowding − Distance − Assignment(F_l);
    Quick − Sort(F_l);
    /*Choose K solutions with largest distance*/
    P_{t+1} ← P_{t+1} ∪ Select(F_l, k);
end if
if t + 1 = Threshold then
    UserFeedback ← TRUE;
    /* Select and rank the best front */
    Rank − Solution(F_1);
    Threshold ← Threshold + t + 1;
end if
End
```

vectors representing different solutions may have different sizes, i.e., number of operations.

Table 1 shows the possible edit operations that can be applied to model elements. These operations are inspired by the catalog of operators for metamodel/model co-evolution presented in [18]. 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. The model elements that can be modified are described in Table 1.

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 instances of classes are called objects, instances of features are called slots, and instances of references are called links.

The initial population is generated by randomly assigning a sequence of edit operations to a randomly chosen set of initial model elements. After interactions with the designer, several constraints can be automatically generated to take into consideration the feedback from the designers as explained in the next section. The user can specify as part of the parameters setting: the minimum and maximum size of the edit operations, and the types of edit operations that will be considered if the designer thinks that not all of them are relevant. The reader can look at the tool demo in the following link[2] for more details.

Variation operators. In each search algorithm, the variation operators play the key role of moving within the search space with the aim of driving the search towards optimal solutions.

For the crossover, we use the one-point crossover operator. It starts by selecting and splitting at random two parent solutions. Then, this operator creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and vice versa for the second child. This operator must ensure the respect of the length limits by eliminating randomly some edit operations. It is important to note that in multi-objective optimization, it is better to create children that are close to their parents in order to have a more efficient search process.

For mutation, we use the bit-string mutation operator that picks one or more edit operations from the co-evolution solutions and replace or modify or delete them. While the crossover operator does not introduce or modify an edit operation of a solution but just the sequence (a swap between edit operations of different solutions), the mutation operator definitely can add or modify or delete an edit operation when applied to any solution of the population. When a mutation operator is applied, the goal is to slightly change the solution for the purpose to probably improve its fitness functions.

When applying both change operators, we are using a repair operator to detect and fix possible redundancies between the model

[2] https://youtu.be/T-jg29t0TWY

elements. When a redundancy is detected, we remove one of redundant model elements from the solution (vector). Similar to the solution representation configuration, the user can specify the probability thresholds for mutation and crossover as highlighted in the tool demo.

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

The first fitness function $nvc(s)$ counts the number of violated constraints w.r.t. the evolved metamodel after applying a sequence s of edit operations. We apply, first, the sequence of edit operations (solution) on the initial model then we load the evolved model on the target metamodel to measure the number of conformance errors based on the number of violated constraints. We consider three types of constraints, as described in [31]: 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. [36] and Richters et al. [31] 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 [23]

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 [28]. 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

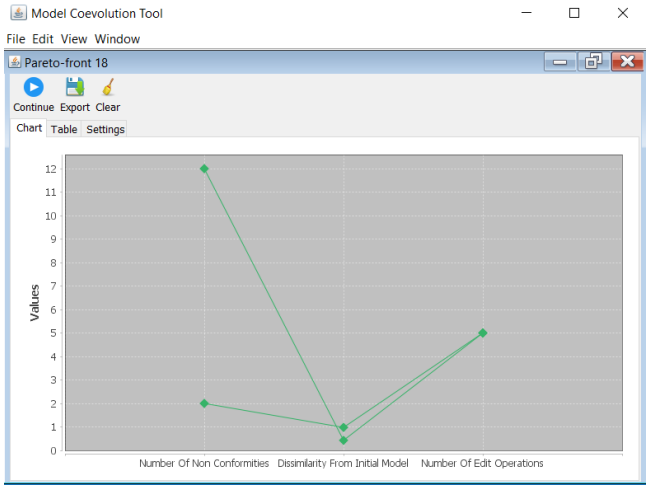

Figure 3: A simplified example of a Pareto front of co-evolution solutions generated by our tool.

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.

Figure 3 shows a simplified example of a Pareto front for two non-dominated co-evolution solutions (every solution is represented by a line). The designer can select a solution based on his preferences and introduce some changes as explained the next section. After the interaction phase, NSGA-II is executed again to repair the solutions based on the designer feedback.

3.2.2 Interaction phase. Our tool proposes to the designer a set of interaction features after selecting one of co-evolution solutions, including 6 edit operations, generated by the first step based on NSGA-II. As described in Figure 4, the designer can either add, modify, reject (remove) or evaluate the list of edit operations one by one. 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. For

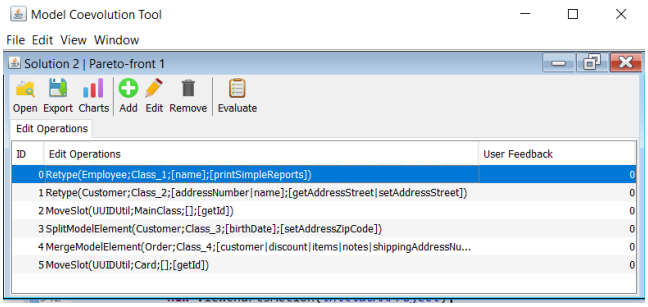

Figure 4: An illustration of the interaction features (add, modify, remove and evaluate) with the user for a generated co-evolution solution.

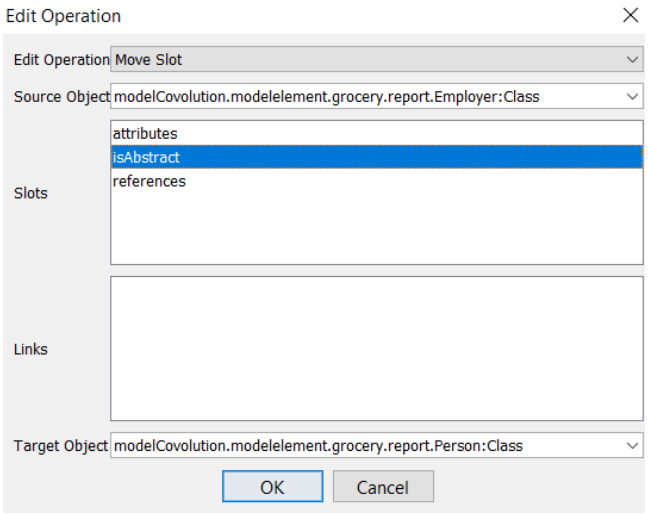

Figure 5: An example of an interaction feature to modify a Move Slot operation.

instance, Figure 5 shows an example of a MoveSlot edit operation that is modified by the designer.

After every action selected by the developer, the algorithm automatically generates three types of constraint for the next iterations to reduce the search space when generating new solutions: 1. For the rejected/removed edit operations, they will be injected with low probability in the solutions generated in the next iterations. 2. For the accepted edit operations (evaluated as good by the designer), they will be injected with high probability in the solutions generated in the next iterations. 3. For the modified edit operations, they will be replacing the initial ones generated by NSGA-II.

In the non-interactive co-evolution approaches, the set of edit operations, suggested by the best-chosen solution, needs to be fully executed on the initial model. Thus, it may not be feasible to repair the co-evolution solution at this late stage. In this context, the goal of this work is to cope with the above-mentioned limitation by granting to the designer's the possibility to customize the set of suggested edit operations either by accepting, modifying or rejecting them. The novelty of this work is the approach's ability to take into account the developer's interaction in terms of introduced customization to the existing solution. We believe that our approach may narrow the gap that exists between automated co-evolution techniques and human intensive development. Our interactive process differs from simply "filtering" operations based on a given preference as it "learns" from the designer's decision making and dynamically break the tie between Pareto-equivalent solutions by upgrading those with the highest number of successful recommendations (applied edit operations) while penalizing those who contain rejected edit operations.

4 EVALUATION

To evaluate our interactive metamodel/model co-evolution approach, we conducted a set of experiments based on four different case studies of evolved metamodels. We chose to analyze the extensive evolution of three Ecore metamodels from the Graphical Modeling Framework (GMF), an open source project for generating graphical modeling editors, and a well-known evolution case of UML Metamodel for Class Diagram extracted from [6, 41]. These evolution cases are selected because of the extensive revisions introduced to the GMF metamodels during a period of two years and to be able to compare with existing co-evolution studies using the same data-sets. The UML Class Diagram evolution case may show the generality of our approach for different types of metamodels and enable a comparison with an existing study not based on the use of heuristic search [41].

In our case study, we created two evolution scenarios per metamodel covering a period of two years. We compared our interactive approach with two of our former works: an existing automated multi-objective co-evolution approach (without the interaction component) [24] and an existing automated co-evolution approach based on pre-defined rules [41]. In addition, we compared our approach to the manual metamodel/model co-evolutions. In the following, we start by presenting our research questions and the experimental setting. Then, we describe and discuss the obtained results. The reader can find a tool demo of our interactive approach in the following link[3].

4.1 Research Questions

We aimed at answering the following research questions:

- **RQ1:** To what extent can the proposed interactive multi-objective approach co-evolve models to make them comply with a new metamodel version with a reasonable interaction effort (in terms of correctness and completeness of proposed edit operations, and the number of required interactions)?
- **RQ2:** How does our interactive approach perform compared to an existing automated multi-objective co-evolution tool [24] and a deterministic automated co-evolution approach [41]?
- **RQ3:** How does our interactive approach perform compared to manual co-evolution in terms of effort reduction?

4.2 Experimental Setting

4.2.1 Studied Metamodels and Models. To answer the different research questions, we considered the evolution of GMF covering a

[3] https://youtu.be/T-jg29t0TWY

period of two years and the UML Class Diagram metamodel evolution from [6, 41]. These case studies are very interesting scenarios since they represent real metamodel evolutions, used in an empirical study [16] and studied in other contributions [9, 13, 32]. 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 three different models to evolve for the GMF case-studies and five different models to evolve for the class diagram case. 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 5 generated models for the case of UML metamodel class diagram evolution from the deterministic work of [6, 41] thus we were not involved in the selection of models and metamodel changes. In order to ensure a fair comparison with Wimmer et al. [41], 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. The quality of our results was measured by two methods: automatic correctness (AC) and manual correctness (MC). Automatic correctness consists of comparing the proposed edit operations to the reference ones, operation by operation using precision (AC-PR) and recall (AC-RE). For an operation sequence corresponding to a given solution, precision indicates the proportion of correct edit operations (w.r.t. the baseline sequence) in a solution. Recall is the proportion of correctly identified edit operations among the set of all expected operations. Both values range from 0 to 1, with higher values indicating good solutions. AC method has the advantage of being automatic and objective. However, since different edit operation combinations exist that describe the same evolution (different edit operations but same target model), AC could reject a good solution because it yields different edit operations from reference ones. To account for those situations, we also use MC which manually evaluates the proposed edit operations, one by one. In addition to these metrics, we report the number of interactions (either accept, remove or modify an edit operation) with the users (NI) needed by our interactive approach and 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 all the three research questions including the comparison between our interactive approach, the two automated techniques of Kessentini et al. [24] and Wimmer et al. [41], and the fully manual co-evolution. We have also extracted one qualitative example from our experiments to explain why a perfect revised model cannot be obtained using the automated techniques and without interactions with the user.

4.3 Study Participants

Our study involved 16 master students in Software Engineering. All the participants are volunteers and familiar with model-driven engineering and co-evolution/refactoring since they are part of a graduate course on Software Quality Assurance. All the graduate students have already taken at least one position as software engineer in industry for at least three years as software developer and most of them (11 out of 16 students) 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 [30].

4.3.1 Parameter Settings. The parameters' values of both 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 [21].

4.4 Results

Results for RQ1. Figure 6 confirms that our interactive approach was successful in recommending relevant co-evolution model changes. Based on the different case studies, our approach was able to correctly recommend, at least, 92% of generated edit operations (precision) and a minimum of 97% of correct recommendations among expected ones (recall). For all the GMF Map and GMF Gen scenarios, the users successfully obtained the extact expected revised model thus both precision and recall were 100%. The same observation is valid for the manual correctness where all the revised models were considered correct by the users.

Figure 7 shows that our interactive approach did not require, in average, a huge effort in terms of number of interactions with the

Metamodel				Models		
Release	#of elements	#of changes	%of changes	#of models	#of model elements	#of expected edit operations
GMF Gen 1.41 to 1.90	From 885 to 1120	347	31%	3	382 | 427 | 556	38 | 52 | 64
GMF Gen 1.90 to 1.248	From 1120 to 1216	362	27%	3	402 | 463 | 566	56 | 61 | 72
GMF Map 1.45 to 1.52	From 382 to 413	62	15%	3	182 | 214 | 304	36 | 42 | 53
GMF Map 1.52 to 1.58	From 413 to 428	10	1.8%	3	243 | 278 | 362	47 | 53 | 56
GMF Graph 1.25 to 1.29	From 278 to 279	14	5%	3	142 | 157 | 193	32 | 37 | 48
GMF Graph 1.25 to 1.33	From 279 to 281	42	14%	3	144 | 162 | 211	39 | 42 | 53
UML CD [41]	From 23 to 29	8	8%	5	23|26|29|34|38	11|14|9|13|16

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

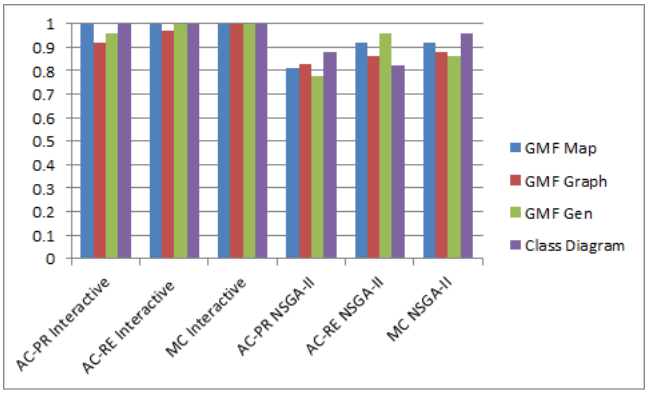

Figure 6: Median correctness results of the interactive and the automated multi-objective [24] approaches on the different subjects.

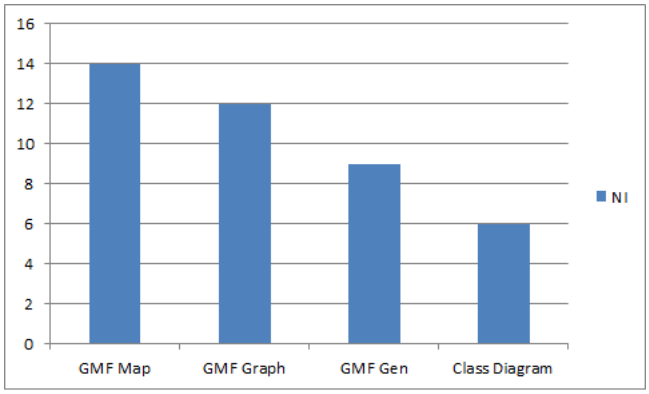

Figure 7: Median number of interactions by the participants to find the best co-evolution solution on the different models.

participants. The highest number of interactions was observed on GMF Map models with a total of 14 and the lowest one is limited to only 6 interactions. This confirms that the multi-objective search using our fitness functions identified good co-evolution solutions requiring few fixes to generate valid revised models. Furthermore, it may give an indication that the generated constraints from the feedback helped to identify better solutions with minimum "repetitive" feedback.

Results for RQ2. To make meaningful comparison with an automated multi-objective search technique, we select the best solution for the automated multi-objective approach of Kessentini et al. [24] using a knee point strategy [8]. Figure 6 shows that the solutions provided by our interactive approach have the highest manual and automatic correctness values on most of the scenarios. The same observation is valid for MC. It is clear from the results that the interaction component significantly improved the results comparing to the automated multi-objective search.

To further investigate the performance of our co-evolution technique, we compared its performance to a deterministic approach proposed by Wimmer et al. [41] as described in Figure 8. The deterministic approach defines generic rules for a set of possible metamodel changes that are applied to the co-evolved models. Figure 8 shows that our interactive approach clearly outperform, in average, the deterministic techniques based on all measures: precision, recall and manual correctness. The comparison is limited to the only case of UML Class Diagram evolution since for this case Wimmer et al. [41] provide a set of co-evolution rules. Further adaptations are required to make this set of rules working on other metamodels. The outperformance of our approach may be explained by the fact that it is hard to generalize all possible metamodel/model co-evolution changes since several possible model-level edit operations could be recommended for the same metamodel change.

Figures 9 and 10 show a qualitative example extracted from our experiments related to the UML class diagram scenario. The UML class diagram evolution included several elements deleted and added at the metamodel which make the co-evolution process very challenging to automate. We found that both automated approaches were not able to recommend the right set of elements to delete and add since it was impossible to automatically detect which attributes should be identifiers and which one are just descriptions, thus it was impossible to generate the revised model without the interaction phase.

Results for RQ3. While both interactive and manual techniques proposed perfect edit operations, Figure 11 shows that the interactive approach requires much less effort than manually evolving the models. For instance, the average time to manually find the perfect edit operations is over 150 minutes per metamodel/model co-evolution, in average, while it requires around 54 minutes for

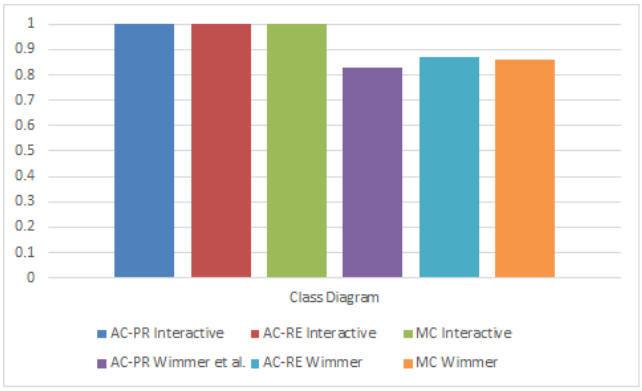

Figure 8: Median correctness results of the interactive and the automated deterministic [41] approaches on the different UML class diagrams.

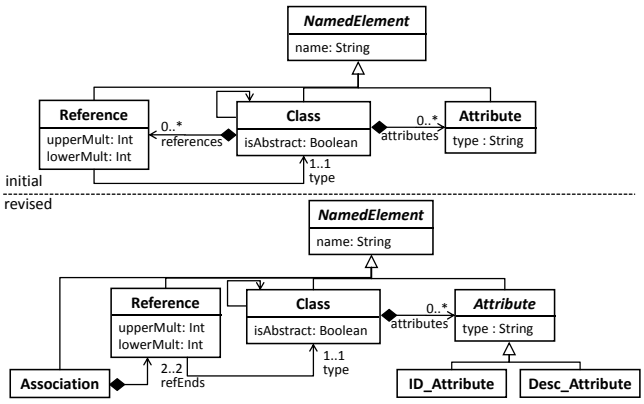

Figure 9: Qualitative example: Initial metamodel (above) and revised metamodel (below).

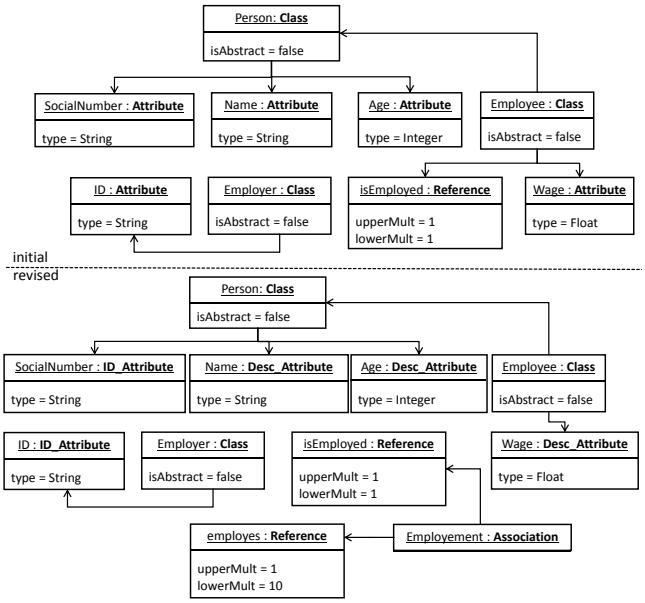

Figure 10: Qualitative example: Initial model (above) and revised model (below).

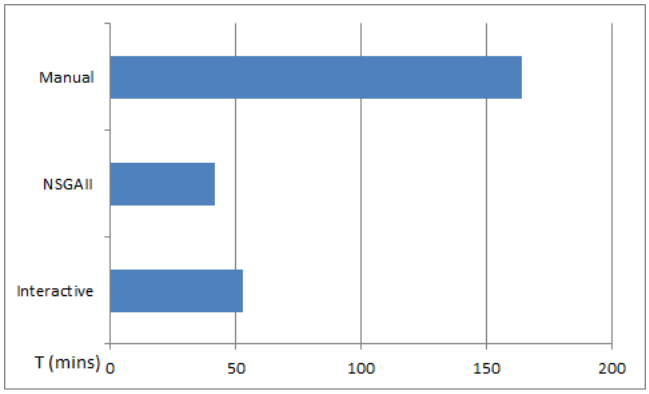

Figure 11: Median time in minutes spent by the participants to find the best co-evolution solution on the GMF subjects.

the case of the interactive approach. The lowest execution time is provided by the automated NSGA-II approach but it was just limited to only less than 10 minutes reduction comparing to the interactive technique with lower correctness than our interactive proposal.

5 THREATS TO VALIDITY

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. 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 [3] for our approach so that parameters are updated during the execution in order to provide the best possible performance.

Internal validity is concerned with the causal relationship between the treatment and the outcome. 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 developers to different groups according to their programming 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 refers to the generalization of our findings. 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.

6 RELATED WORK

6.1 Manual specification approaches

In one of the early works [37], the co-evolution of models is tackled by designing co-evolution transformations based on metamodel change types. In [7, 10], 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 [11], 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 [33]and in the framework described in [29]. Another manual specification approach is presented in [39] 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., [19, 25–27, 38]), 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 [14, 17, 19, 40]. In [40] the author provides a library of co-evolutionary operators for MOF metamodels, each of these operators provides a model migration strategy. In [14] 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 [19] 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 [2], 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 [1], 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.

To sum up, none of the existing approaches allows the exploration of different possible co-evolution strategies. Only one specific strategy is either automatically derived from the calculated set of metamodel changes. So the resolution result is not guaranteed to be the one desired to co-evolve a model. Our approach, gives the user a better control over the result, since we propose a set of alternative resolution strategies (the best solutions from the Pareto front) to the user to select appropriate ones and interactively update them.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed an interactive multi-objective approach that dynamically adapts and interactively suggests edit operations to designers and takes their feedback into consideration. The search process is guided by three fitness functions: the number of proposed operations, structural and semantic similarity with the initial metamodel and the conformity with the constraints of the new metamodel. The feedback received from the designers is used to reduce the search space and converge to better solutions.

We evaluated our approach on several evolution scenarios extracted from different widely used metamodels and compared it to fully automated co-evolution techniques. We plan to extend this work, by evolving interactively model transformation rules and OCL constraints when the source or target metamodels are revised.

Acknowledgements. This work has been partially funded by the Austrian Federal Ministry of Science, Research and Economy, National Foundation for Research, Technology and Development, and by the Austrian Science Fund (FWF) P 28519-N31.

REFERENCES

[1] F. Anguel, A. Amirat, and N. Bounour. 2014. Using weaving models in metamodel and model co-evolution approach. In *Proceedings of CSIT*.
[2] Fouzia Anguel, Abdelkrim Amirat, and Nora Bounour. 2015. Hybrid Approach for Metamodel and Model Co-evolution. In *Proceedings of CIIA*.
[3] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of ICSE*.
[4] Edouard Batot, Wael Kessentini, Houari Sahraoui, and Michalis Famelis. 2017. Heuristic-based Recommendation for Metamodel OCL Coevolution. In *Proceedings of MODELS*.
[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice.* Morgan & Claypool Publishers.
[6] 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*.

[7] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. 2008. Automating co-evolution in model-driven engineering. In *Proceedings of EDOC*.

[8] 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*.

[9] Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. 2011. Automated Co-evolution of GMF Editor Models. In *Proceedings of SLE*.

[10] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. 2009. Managing model adaptation by precise detection of metamodel changes. In *Proceedings of ECMFA*.

[11] Boris Gruschko. 2007. Towards synchronizing models with evolving metamodels. In *Proceedings of MoDSE Workshop*.

[12] 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.

[13] Markus Herrmannsdoerfer. 2011. GMF: A Model Migration Case for the Transformation Tool Contest. In *Proceedings of TTC*.

[14] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. 2009. COPE - Automating Coupled Evolution of Metamodels and Models. In *Proceedings of ECOOP*.

[15] Markus Herrmannsdoerfer and Daniel Ratiu. 2010. Limitations of Automating Model Migration in Response to Metamodel Adaptation. In *Models in Software Engineering*. 205–219.

[16] Markus Herrmannsdoerfer, Daniel Ratiu, and Guido Wachsmuth. 2010. Language Evolution in Practice: The History of GMF. In *Proceedings of SLE*.

[17] 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*.

[18] 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*.

[19] Markus Herrmannsdörfer. 2011. COPE - A Workbench for the Coupled Evolution of Metamodels and Models. In *Proceedings of SLE*.

[20] 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.

[21] 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.

[22] Wael Kessentini. 2015. Automated Metamodel/Model Co-Evolution using a Multi-Objective Optimization Approach. In *Proceedings of the ACM Student Research Competition at MODELS*.

[23] Wael Kessentini. 2018. Constraints for model co-evolution: https://docs.google.com/document/d/1O1GjOcvvBgPuVacB12noygNaJ75y0nwySdGmiqGXouQ.

[24] Wael Kessentini, Houari A. Sahraoui, and Manuel Wimmer. 2016. Automated Metamodel/Model Co-evolution Using a Multi-objective Optimization Approach. In *Proceedings of ECMFA*.

[25] 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.

[26] 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.

[27] 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*.

[28] Lailil Muflikhah and Baharum Baharudin. 2009. Document clustering using concept space and cosine similarity measurement. In *Proceedings of ICCTD*.

[29] Anantha Narayanan, Tihamer Levendovszky, Daniel Balasubramanian, and Gabor Karsai. 2009. Automatic Domain Model Migration to Manage Metamodel Evolution. In *Proceedings of MODELS*.

[30] 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.

[31] Mark Richters. 2001. *A precise approach to validating UML models and OCL constraints*. Technical Report.

[32] 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.

[33] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. 2010. Model migration with Epsilon Flock. In *Proceedings of ICMT*.

[34] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona A. C. Polack. 2009. An Analysis of Approaches to Model Migration. In *Proceedings of MoDSE-MCCM Workshop*.

[35] 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*.

[36] 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*.

[37] 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.

[38] 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*.

[39] Sander Vermolen and Eelco Visser. 2008. Heterogeneous Coupled Evolution of Software Languages. In *Proceedings of MODELS*.

[40] Guido Wachsmuth. 2007. Metamodel adaptation and model co-adaptation. In *Proceedings of ECOOP*.

[41] 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*.