Automated Metamodel/Model Co-Evolution using a Multi-Objective Optimization Approach

Wael Kessentini
GEODES-DIRO
University of Montreal, Canada
kessentw@iro.umontreal.ca

***Abstract*—Metamodels undergo many changes during the evolution of several software modeling languages and projects. As a consequence, models have to be updated for preserving their conformance with the new metamodel versions. A common practice is to manually define rules for each metamodel evolution to co-evolve the corresponding models. In this paper, we propose a generic automated approach for the metamodel/model co-evolution. In our approach, we view the co-evolution as a multi-objective optimization problem, and we solve it using the NSGA-II algorithm. Our algorithm search for solutions that minimize (1) the non-conformities with the new metamodel version, (2) the changes to the existing models, and (3) the loss of information. We successfully evaluated our approach on the evolution of the well-known UML state machine metamodel.**

***Index Terms*—Metamodel/model co-evolution, Model migration, Coupled evolution, NSGA-II**

I. Introduction

Model-Driven Engineering (MDE) considers models as first-class artifacts during the software lifecycle [1]. Available techniques, approaches, and tools for MDE are growing and they support a huge variety of activities, such as model creation, model transformation, and code generation. As in MDE the modeling languages are explicitly modeled and therefore, as any other model changeable, the evolution of models often depends on the evolution of their metamodels. Metamodels are subject to many changes during the evolution of software modeling languages and language maintenance projects, especially when it comes to domain-specific modeling languages [2]. Thus, models have to be updated for preserving their conformance with the new metamodel version.

The remainder of this paper is structured as follows. Section II provides the background of model co-evolution and demonstrates the challenges addressed in this paper based on a motivating example. In Section III, we give an overview of our proposal and explain how we adapted the NSGA-II algorithm to find optimal new models. Section IV discusses the design and results of the empirical evaluation of our approach. After surveying related work in Section V, we conclude with some pointers to future work in Section VI.

II. MotivatingExample

This section introduces the background, namely the basic notions of metamodels and models, the *conformsTo* relationship, as well as an example to demonstrate the challenges one is facing when dealing with metamodel/model co-evolution.

A. Background: Metamodels and Models

Metamodels are the means in MDE to specify the abstract syntax of modeling languages [3]. For defining metamodels, there are meta-modeling standards (such as MOF, Ecore) available which are mostly based on a core subset of the UML class diagrams, i.e., there are classes, attributes, and references used to specify the modeling languages, i.e., the intentional description of all possible models of a given language. These 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 the objects (instances of references), which have to conform to the UML class diagram describing the metamodel. Therefore, the abstract syntax of models is often represented in terms of UML object diagrams. In order for a model to conform to its metamodel, there have to be several constraints fulfilled. These constraints are normally referred to as *conformsTo* relationship [4], [5].

To make the *conformsTo* relationship more concrete, we give an excerpt of the constraints concerning objects in models and their relationship to classes in metamodels. Objects are instantiated from classes. Thus for each referred type of an object in a given model, a corresponding class must exist in the metamodel (name equivalence) and the corresponding class must not be abstract. Such constraints may be formulated in in the following way:

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

An example metamodel and corresponding model is shown in Figure 2a and Figure 1a, respectively. This simple language allows to define simple state machines consisting of states having a name and predecessors as well as successor states.

B. Metamodel/Model Co-Evolution: A Motivating Example

While some metamodels, such as UML, are standardized and changed rarely, metamodels for Domain-Specific Modeling Languages (DSMLs) [6], representing concepts within a certain domain, are frequently subject to change [2].

As most of the current metamodeling frameworks are currently strict in the sense that only fully conformed models can be used, metamodel evolution requires to co-evolve the already

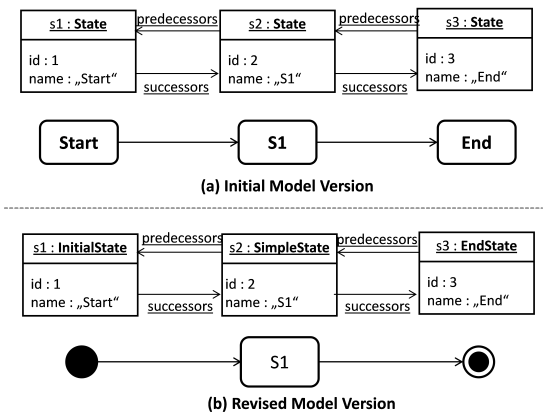

Fig. 1. Example Model Evolution

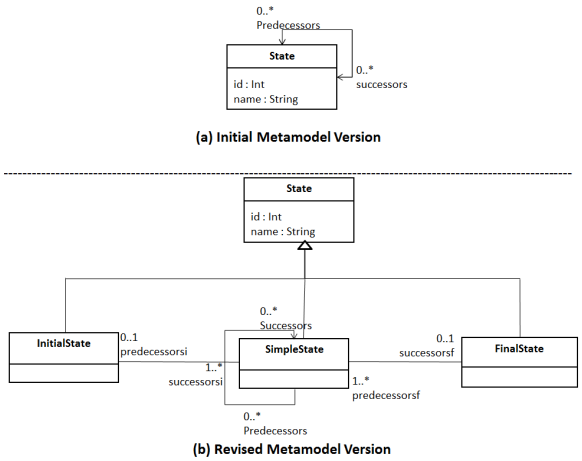

Fig. 2. A Simplified Metamodel Evolution Example

existing model instances, which may no longer conform to the new metamodel version. In such cases, model migration scripts have to be provided in current tools [7], to re-establish the conformance between models and their metamodels. However, finding the best migration scripts to co-evolve the models is left to the user of such tools or default migration scripts are provided. However, exploring the actual co-evolution space is still an open challenge.

Figure 2 shows an example of a simplified metamodel evolution, based on the simple State Machine language. The metamodel evolution comprises three steps: extract sub-classes for *State* class resulting in *InitialState*, *SimpleState*, and *FinalState*, make class *State* abstract, and push-down, create and refine the cardinalities of the predecessor/successor references for the subclasses. This results in the fact that besides other constraints violations, the constraint shown previously is violated when considering the initial model shown in Figure 1a and its conformance to the new metamodel version in Figure 2b.

To re-establish conformance for the given example, assume for now that only two operations on models are used in this context. Non-conforming objects may either be retyped (reclassified as instances of the concrete classes) or deleted. Thus, the potential solution space for retyping or deleting non-conforming elements contains $(c + 1)^O$ solutions (with c=number of candidate classes + 1 for deletion, o=number of nonconforming objects).

This means, in our given example, we would end up with 64 possible co-evolutions while one (probably the preferred one) of these is shown in Figure 1b. This one seems the preferred one due to several reasons such as the number of changes introduced (e.g., loss of information by deleting some model elements) and the number violated conformance constraints.

To this end, we propose in this paper to consider the model co-evolution problem as a multi-objective one to find a balance between the consistency with the previous version of the model and the conformance to the new metamodel.

III. Model Co-evolution: a Multi-Objective Problem

A. Overview

The goal of our approach is to derive a tentative new version, of an existing model, that conforms to a new version of its original metamodel. We view this derivation as a search in the space of all possible sequences of modifications of the original models. The search is guided by three objectives, which aims at minimizing (1) the number of non conformities with the new version of the metamodel, (2) the number of the changes to the initial model and (3) the loss of information after modifying the initial model. In other words, the generated new revised model has to be similar as much as possible to the initial model while conforming to the new metamodel version. Therefore, we implemented our idea in the form of a multi-objective optimization algorithm that computes an optimal sequence of edit operations representing the best trade-off between the three objectives. More concretely, our algorithm takes as inputs the initial and revised versions of the metamodel, a list of models to update and a list of possible edit operations to apply to the models (retype and delete). It generates as output a sequence of edit operations that should be applied to the initial version of the model to generate a new version compatible with the new metamodel version.

The space of all possible sequences of editing operations can be very large, especially when dealing with large models. An exhaustive search method could be inefficient in this cases. Alternatively, we use a heuristic search with a multi-objective evolutionary algorithm, NSGA-II [8]. In the next paragraphs, we describe the adaptation of the generic NSGA-II algorithm to the co-evolution problem.

B. Adapting NSGA-II for Model Co-Evolution

NSGA-II [8] is one of the most-used multi-objective evolutionary algorithms (EAs) in tackling real-world problems, including software engineering ones [9], [10] to find trade-offs between different objectives. It begins by generating an offspring population from a parent one by means of variation

operators (crossover and mutation) such that both populations have the same size. After that, it ranks the merged population (parents and children) into several non-dominance layers, called fronts.

Non-dominated solutions are assigned a rank of 1 and constitute the first layer (Pareto front). After removing solutions of the first layer, the non-dominated solutions form the second layer and so on and so forth until no non-dominated solutions remain. After assigning solutions to fronts, each solution is assigned a diversity score, called crowding distance, inside each front. This distance defines a partial ranking inside the front which aims, later, at favoring solutions that are far from the others in terms of objective values. A solution is then characterized by its layer and its crowding distance inside the layer.

To finish an iteration of the evolution, we perform the environmental selection to form the parent population for the next generation by picking half of the solutions. The solutions are included iteratively from the Pareto front to the lowest layers. If half of the population is reached inside a front than the crowding distance is used to complete the parent population.

1) Problem formulation: The model 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 nvc with the new metamodel version, the number of changes $nbOp$ to the initial model, and the loss of information $disIm$ between the initial and the evolved models. Formally:

$$\begin{cases} Minimize & f_1(s) = nvc(s) \\ Minimize & f_2(s) = nbOp(s) \\ Minimize & f_1(s) = disIm(s) \end{cases}$$

The first fitness function $nvc(s)$ counts the number of violation constraints w.r.t. the evolved metamodel after applying a sequence s of edit operations.

We used in our experiments the implementation constraints proposed by Schoenboeck et al. [4].

For the two remaining objectives, which aims at minimizing the changes to the initial models, we used simple approximations. $nbOp(s)$ is simply the number of editing operations in a sequence s. $disIm(s)$ measure the difference in size between the initial M_i and the revised M_r models. If the difference is negative or null then $disIm(s) = 0$ (no information loss). Otherwise, $disIm(s) = size(M_i) - size(M_r)$.

2) NSGA-II Application : To adapt NSGA-II to our problem, it is necessary to define (1) how to represent a co-evolution solution, (2) how to derive new solutions from existing ones, and (3) how to evaluate a solution.

Solution representation. We mentioned earlier that a solution is a sequence of edit operations to be applied to the model to evolve. This can be seen as an inplace transformation. However, as we are dealing with two versions of a metamodel, we decided to represent a solution as an outplace transformation. Rather than editing the initial model, we create a new model. Thus, we only have creation operations, but not retyping ones. Although we are not supposed to delete elements directly in the initial model, we use deletion operations to "repair" candidate solutions during the evolution, *i.e.,* deleting an element that was previously created. A candidate solution (individual) is then represented with a vector whose elements are the creation/deletion operations to create the new version of a model from the initial one. Each edit operation specifies the model element to which it is applied. Consequently, vectors encoding different solution candidates may have different sizes depending on the number of operations in the sequences.

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

Solution derivation. In a search algorithm, the variation operators play a key role of moving within the search space with the aim of driving the search towards better solutions. In each iteration, we select $population\_size/2$ individuals from the population pop_i to form population pop_{i+1}. These $(population\_size/2)$ selected individuals will produce other $(population\_size/2)$ new individuals using a crossover and mutation operators. To select parents for reproduction, we used the principle of the roulette wheel [11]. According to this principle, the probability to select an individual for crossover and mutation is directly proportional to its relative fitness in the population.

We use a one-point crossover operator. For our problem, this operator split each parent operation sequence $S1$ (resp. $S2$) into two subsequences $\{S1_1, S1_2\}$ (resp. $\{S2_1, S2_2\}$) according to a cut position k. Then, it combines the subsequences to create to sibling solutions $\{S1_1, S2_2\}$ and $\{S2_1, S1_2\}$. Our crossover operator could create a child sequence that contains conflict operations. In this case, it will be penalized by the component nvc of the fitness function.

The mutation operator consists in randomly selecting one or two operations in a solution vector and modifying them. Two modifications are used: (1) swapping the two selected operations in the sequence or (2) replacing an operation by a randomly created one.

Solution evaluation. As mentioned in the problem formulation, a solution is evaluated according to three objectives. Thus, for each solution s, we calculate $nvc(s)$, $nbOp(s)$, and $disIm(s)$. These values are used later to establish the dominance relation between solutions.

IV. Validation

A. Research Questions

The validation study was conducted to quantitatively and qualitatively assess the completeness and correctness of our co-evolution approach when applied to realistic settings and to compare its performance with an existing deterministic approach [12]. More specifically, we aimed at answering the following research questions:

- **RQ1:** To what extend the obtained results are attributable to our approach and not to the fact of exploring a large

number of solutions? If a random search, exploring a same number of solutions as our approach, gives same or better results, this means that there is no need to use a metaheuristic search.

- **RQ2:** To what extent can the proposed multi-objective approach co-evolve models to make them comply with a new metamodel version (in terms of correctness and completeness of proposed edit operations)?

B. Experimental Setting

1) Studied Meta-models and Models: To answer the four research questions, we considered the evolution of UML State Machine Metamodel from version 1.4 to 2.0 [13]. Therefore, the two versions were manually analyzed to determine the actually applied changes. Additionally, we collected from previous work [4], [12] all the edit operation types that can be applied to models for this metamodel evolution. We also selected 10 models from version 1.4 and evolved them manually to version 2.0, according to the collected edit operation types. The manually defined sequences for the selected models are used as baseline sequences for the calculation of precision and recall scores.

2) Evaluation Metrics: To compare our approach with the other alternatives, we use precision and recall measures. For an operation sequence corresponding to a given solution, precision indicates the fraction of correctly edit operations (w.r.t. the baseline sequence) among the set of all operations in the sequence. Recall is the fraction of correctly identified edit operations among the set of all expected operations. Roughly speaking, the precision represents the probability that a detected operation is correct whereas the recall is seen as the probability that an expected operation is detected. Both values range from 0 to 1, with higher values indicating good solutions.

The baseline sequences do not represent unique evolution solutions for the used models. Indeed, more than one alternative can be possible to evolve a given model. Thus, in addition to automatic precision (AC-PR) and recall (AC-RE), we calculated a manual precision (MC). For manual precision, rather than comparing automatically the produced sequence with the expected one, we checked operation by operation if they are correct w.r.t. the evolved model.

3) Statistical Tests: Since the used metaheuristic algorithms (NSGA-II and GA) are stochastic by nature, different executions may produce different results for the same model with the same execution parameters. For this reason, our experimental study is performed based on 30 independent simulation runs and the obtained results by the alternative approaches are compared using the Wilcoxon rank sum test [14] with a 95% confidence level ($\alpha = 5\%$).

4) Parameter Settings: Parameter setting has a significant influence on the performance of a search algorithm. We tried different values to set the execution parameters. For the reported results, we used crossover probability = 0.8, mutation probability = 0.5, population size = 100, number of iteration = 1000. The same parameters were used for the multi- and mono-objective algorithms. For the random search, we generated 1000 x 100 solutions and took the one with the best fitness. This ensures that the tree alternatives (multi-objective, mono-objective, and random) explore the same number of solutions to have a fair comparison.

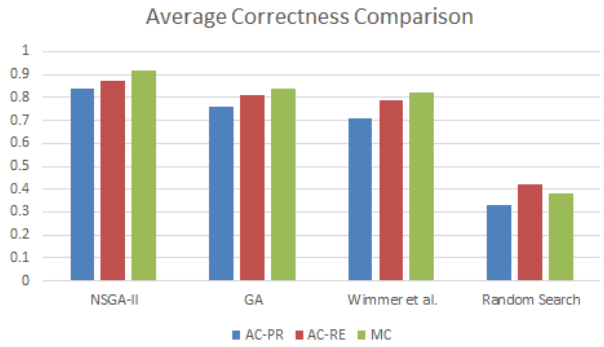

Fig. 3. Average correctness results of NSGA-II, GA, Wimmer et al., Random Search on the 10 models. The results were statistically significant on 30 independent runs using the Wilcoxon rank sum test with a 95% confidence level ($\alpha < 5\%$).

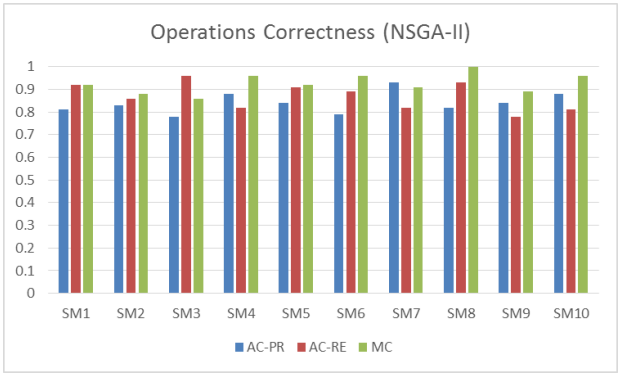

Fig. 4. Correctness results of NSGA-II on the 10 State Machine models.

C. Results

Results for RQ1. We do not dwell long in answering the first research question (RQ1) that involves comparing our approach based on NSGA-II with random search. Figure 3 confirm that using NSGA-II (as well as the GA and the deterministic algorithm) produce results by far better (and statistically significant) than just randomly exploring a comparable number of solutions. NSGA-II has precisions (AC-PR and MC) and recall (AC-RE) more than twice higher than the ones of random search as shown in Figure 3 ($\sim 85\%$ vs $\sim 40\%$).

To formally answer **RQ1**, we state that *there is an empirical evidence that the quality of the co-evolution results obtained are due to our multi-objective approach and not to the number of explored solutions*.

Results for RQ2. We evaluated the averages AC-PR, AC-RE and MC scores for non-dominated co-evolution solutions proposed by NSGA-II.

For the precision and recall, Figure 4 shows that the produced solutions using NSGA-II are similar to the baseline ones with more than 80% of precision and recall (AC-PR and AC-RE) in general. For four models (SM1, SM2, SM5, and SM8), we obtained more than 90% for the recall. From another perspective, we did not observe a correlation between the size or the number of operations of the models and the precision and recall. For example, we obtained higher precision and recall for SM7 (40 elements and 82 operations) than for SM2 (17 elements and 38 operations). This means that the correctness of the results is not degraded as the size of the models or the size of necessary modifications increase. For the manual precision, the results are even better. Except for SM2 and SM9 all the MCs are higher than 90% with a perfect score for SM8. Here again, the scalability in terms of correctness is valid for MC. Indeed MC increases from 86% (SM2) to 94% (SM10) while the size of the model (resp. operation sequence) goes from 17 to 44 (resp. 38 to 94).

To formally answer **RQ2**, we state that *the multi-objective co-evolution approach allows to migrate models with higher precision and recall and with a limited number of edit operations. This achieved without any explicit knowledge on the specific changes that occurred on the metamodel.*

D. Threats to Validity

There are several validity threats to the design of this study. The first threat is related to the parameter tuning of our multi-objective algorithm. Further experiments are required to evaluate the impact of the parameters setting on the quality of the solutions. A second threat is related to our choice of taking the average of the three objective function in the mono-objective algorithms. Other forms of combination, e.g., weighted average, may give different results. Further experiments are required to compare our approach with different settings of the mono-objective algorithm. In this study, we performed our experiments on a single evolution scenario (state machine metamodel from v1.4 to v2.0). Future replications of this study are necessary to confirm our findings, in particular, with industrial settings. In addition, the comparison of the performance of NSGA-II to other existing approaches is limited to the approach of Wimmer et al. [12]. We plan to conduct other comparisons by requesting the tools from the authors or by re-implementing the state-of-the-art approaches.

V. RELATED WORK

Several approaches emerged which aim to tackle metamodel/model co-evolution from different angles using different techniques (cf. e.g., [2], [15], [16] for an overview). Metamodel/model coevolution approaches can be classified in three categories [16]:

- Manual-specification approaches in which the migration strategy is encoded manually by the modeler using general purpose programming language (e.g. Java), or transformation languages (e.g ATL, QVT) [17] [18].
- Metamodel-matching techniques used to infer a migration strategy from the difference between the original metamodel and the evolved metamodel. [19] [2] [20] [21].
- Operator-based approaches that records the metamodel changes as a sequence of co-evolutionary operations used later to infer a complete migration strategy [22] [7] [23] [24].

None of the existing approaches allows the exploration of different possible co-evolution strategies. On the contrary, only one specific strategy is either automatically derived or manually developed from the calculated set of metamodel changes. The only work we are aware of discussing metamodel/model co-evolution using some search-based techniques is [25]. In this paper, they authors discuss the idea of using search-based algorithms to reason about possible model changes, but in contrast to our approach, they rely again on metamodel differences which have to be computed (probably using a search-based approach) before the co-evolution of models can be performed. To the best of our knowledge, this approach is unique compared to previous approaches and outperforms logic-based approaches for repairing models [12]. Furthermore, we are not dependent on the quality of metamodel change detection algorithms.

VI. CONCLUSION

This paper proposes a multi-objective approach for the co-evolution of models by finding the best operation sequence of changes applied to the initial model that generate the target model conforming as much as possible to the evolved metamodel. Therefore, a generated revised model is necessary that minimizes the number of inconsistencies (with the new metamodel), the number of operations and the dissimilarity with the initial model. As the search space in terms of all possible sequences of operations is potentially huge and we have three objectives to optimize, we considered in this paper the co-evolution process as a multi-objective optimization problem.

We evaluated our proposal with a metamodel evolution scenario concerning a state machine modeling language. The experiment results indicate clearly that the best generated models have a precision and recall of more than 80% and a manual precision of more than 90%. Furthermore, the results provide strong evidence to support the claim that our proposal outperforms both a mono-objective and deterministic approaches for model co-evolution.

We are working now on larger metamodels and models with larger lists of operations to apply. This is necessary to investigate more deeply the applicability of the approach in practice, but also to study the performance of our approach when dealing with very large models.

REFERENCES

[1] J. Bézivin, "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[2] B. Meyers and H. Vangheluwe, "A framework for evolution of modelling languages," *Sci. Comput. Program.*, vol. 76, no. 12, pp. 1223–1246, 2011.

[3] T. Kühne, "Matters of (Meta-) Modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[4] J. Schoenboeck, A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Schwinger, M. Wimmer, and M. Wischenbart, "CARE: A Constraint-based Approach for Re-establishing Conformance-relationships," in *Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling*, ser. APCCM '14, 2014, pp. 19–28.

[5] L. Iovino, A. Pierantonio, and I. Malavolta, "On the Impact Significance of Metamodel Evolution in MDE," *Journal of Object Technology*, vol. 11, no. 3, pp. 3:1–33, 2012.

[6] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle, *Domain-Specific Modeling*, 2007, ch. 7, pp. 1–20.

[7] M. Herrmannsdörfer, "COPE A Workbench for the Coupled Evolution of Metamodels and Models," in *Software Language Engineering*, ser. LNCS, 2011, vol. 6563, pp. 286–295.

[8] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," ser. Lecture Notes in Computer Science, 2000, vol. 1917, pp. 849–858.

[9] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, 2012.

[10] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: a multi-objective approach," *Journal of Automated Software Engineering*, vol. 20, no. 1, pp. 47–79, 2013.

[11] K. Deb and S. Gupta, "Understanding knee points in bicriteria problems and their implications as preferred solution principles," *Engineering Optimization*, vol. 43, no. 11, pp. 1175–1204, 2011.

[12] M. Wimmer, A. Kusel, J. Schoenboeck, W. Retschitzegger, and W. Schwinger, "On using inplace transformations for model co-evolution," in *MtATL Workshop*, 2010.

[13] http://www.omg.org, Object Management Group, Inc., OMG Unified Modeling Language Specification v1.4 and v2.0.

[14] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *33rd International Conference on Software Engineering*, 2011, pp. 1–10.

[15] L. Rose, M. Herrmannsdoerfer, S. Mazanek, P. Van Gorp, S. Buchwald, T. Horn, E. Kalnina, A. Koch, K. Lano, B. Schtz, and M. Wimmer, "Graph and model transformation tools for model migration," *Software and Systems Modeling*, vol. 13, no. 1, pp. 323–359, 2014.

[16] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack, "An Analysis of Approaches to Model Migration," in *Proc. Models and Evolution (MoDSE-MCCM) Workshop, 12th ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems*, 2009.

[17] A. Narayanan, T. Levendovszky, D. Balasubramanian, and G. Karsai, "Automatic domain model migration to manage metamodel evolution," in *Model Driven Engineering Languages and Systems*, ser. LNCS, 2009, vol. 5795, pp. 706–711.

[18] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Model migration with Epsilon Flock," in *International Conference on Model Transformation, volume 6142 of LNCS*, 2010, pp. 184–198.

[19] B. Meyers, M. Wimmer, A. Cicchetti, and J. Sprinkle, "A generic in-place transformation-based approach to structured model co-evolution," in *4th Int. Workshop on Multi-Paradigm Modeling @ MoDELS'10*, 2010.

[20] K. Garcés, F. Jouault, P. Cointe, J. Bézivin, and A. Emninria, "Managing model adaptation by precise detection of metamodel changes," in *ECMDA-FA09: European Conference on Model Driven Architecture-Foundations and Applications*, 2009, pp. 34–49.

[21] A. Cicchetti, D. D. Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in *12th International Enterprise Distributed Object Computing Conference (EDOC), IEEE Computer Society*, 2008.

[22] M. Herrmannsdoerfer, S. Benz, and E. Juergens, "Cope - automating coupled evolution of metamodels and models," in *ECOOP 2009 Object-Oriented Programming*, 2009, vol. 5653, pp. 52–76.

[23] G. Wachsmuth, "Metamodel adaptation and model co-adaptation," in *ECOOP07. Volume 4609 of LNCS., Springer*, 2007.

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

[25] J. R. Williams, R. F. Paige, and F. A. C. Polack, "Searching for model migration strategies," in *Proceedings of the 6th International Workshop on Models and Evolution*, 2012, pp. 39–44.