Search-Based Model Transformations with MOMoT

Martin Fleck[1], Javier Troya[2], and Manuel Wimmer[1]

[1] Business Informatics Group, TU Wien, Vienna, Austria
{fleck,wimmer}@big.tuwien.ac.at
[2] ISA Research Group, ETS de Ingeniería Informática, Universidad de Sevilla, Seville, Spain
jtroya@us.es

Abstract. Many scenarios require flexible model transformations as their execution should of course produce models with the best possible quality. At the same time, transformation problems often span a very large search space with respect to possible transformation results. Thus, guidance for transformation executions to find good solutions without enumerating the complete search space is a must.

This paper presents MOMoT, a tool combining the power of model transformation engines and meta-heuristics search algorithms. This allows to develop model transformation rules as known from existing approaches, but for guiding their execution, the transformation engineers only have to specify transformation goals, and then the search algorithms take care of orchestrating the set of transformation rules to find models best fulfilling the stated, potentially conflicting transformation goals. For this, MOMoT allows to use a variety of different search algorithms. MOMoT is available as an open-source Eclipse plug-in providing a non-intrusive integration of the Henshin graph transformation framework and the MOEA search algorithm framework.

Keywords: Search-Based Software Engineering · Model transformation · Henshin · MOEA

1 Introduction

Model transformations are the key technology to manipulate models in Model-Driven Engineering (MDE) [4]. As the applicability of MDE is expanding in software engineering and beyond, model transformations have to cope with many challenges. One of these challenges is how to deal with the large search spaces of many transformation problems. Of course, one approach is to develop problem-specific heuristics which allow to deal with the associated search space without having to enumerate all possible solutions, which is mostly not possible due to practical space and time restrictions. However, finding such problem-specific heuristics is challenging. Therefore, an alternative approach is the usage of meta-heuristics that are problem-independent. This line is investigated by Search-Based Software Engineering (SBSE) [11], which is a lively research field applying

-

search-based optimization techniques to software engineering problems. Search-based optimization techniques deal with large or even infinite search spaces in an efficient manner. Concrete algorithms include local search methods such as Tabu Search [10] and Simulated Annealing [14], or genetic algorithms [12] such as NSGA-II [6] and NSGA-III [5]. Especially in recent years, SBSE has been applied successfully in the area of MDE [13]. Very recently, several approaches have been proposed to provide more efficient search capabilities for model transformations [1,8,9].

MOMoT is one of these emerging approaches and was first presented in [9]. It is based on Henshin [2] as base model transformation framework and MOEA[1] as base meta-heuristic search framework. Thus, MOMoT combines different search techniques with model transformations to produce output models that optimize one or more potentially conflicting quality criteria. Reusing the existing functionality of these base frameworks as much as possible is the central principle of our framework. The MOEA framework is an open-source Java library that provides a set of multi-objective evolutionary algorithms with additional analytical performance measures and that can be easily extended with new algorithms as we have already done for introducing local searchers such as Hill Climbing [9]. While in the rest of the paper we discuss our framework in the light of Henshin and MOEA, the conceptual approach itself is generic so that it may be used for other framework cobminations.

MOMoT is the subject for the proposed tool demonstration. Therefore, in this paper we highlight the integration of Henshin and MOEA from an architectural viewpoint and show the concrete tool support for specifying search-based model transformations by using the Search Configuration Modeling Language (SCML).

The remainder of this paper is structured as follows. First, we introduce MOMoT based on its architecture and provided features in Sect. 2. Then, we present the running example for this paper and the accompanying tool demonstration in Sect. 3. Section 4 illustrates how to configure the search at design time, while Sect. 5 shows the runtime results obtained by MOMoT and how the results are analyzed. Finally, Sect. 6 concludes this paper with an outlook on future work.

2 Features and Architecture of MOMoT

MOMoT offers the following features for developing search-based model transformations: (*i*) a generic way to describe the problem domain and the concrete problem instance, (*ii*) an encoding for the solution of the concrete problem instance based on model transformation solutions, (*iii*) a random solution generator that is used for the generation of an initial, random individual or random population, and (*iv*) a set of search-based algorithms to execute the search. To further support the use of multi-objective evolutionary algorithms, we additionally provide (*v*) generic objectives and constraints for our solution encoding, (*vi*) generic mutation operators that can modify the respective solutions, and

[1] http://www.moeaframework.org.

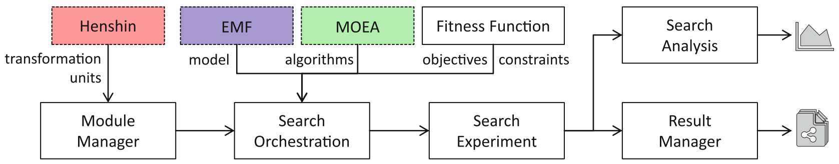


Fig. 1. Overview of MOMoT's workflow

(*vii*) a configuration language that also provides feedback about the specified search configuration. Since our approach combines MDE techniques with SBSE techniques, the key building blocks are an environment to enable the creation of metamodels and models, a model transformation engine and language to manipulate those models and a set of meta-heuristic algorithms that perform a search to find transformation orchestrations that optimize the given objectives and fulfil the specified constraints. Figure 1 shows the typical MOMoT workflow as well as the involved artifacts which are explained in the following sections.

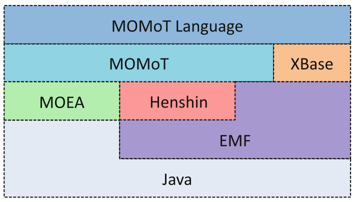


Fig. 2. MOMoTs architecture

To unify the MDE and SBSE worlds in a single framework, we bridge the Eclise Modeling Framework (EMF), the Henshin graph transformation framework, and the MOEA framework. For realizing the MOMoT's SCML, we build on the functionality of XBase for having a model-based representation of search configurations to provide dedicated support for transformation engineers to make use of search-based algorithms. The resulting technology stack is depicted in Fig. 2. The complete source code of MOMoT with further explanations as well as the case studies currently realized with MOMoT can be found on our project website[2].

3 Running Example

In this section, we introduce the running example for demonstrating MOMoT. We selected the example from the model quality assurance domain. It is well-known that the quality of an object-oriented design has a direct impact on the quality of the code produced. The Class Responsibility Assignment (CRA) problem [3] deals with the creation of such high-quality object-oriented models. When solving the CRA problem, one has to decide where responsibilities, under the form of class methods and attributes they manipulate, belong and how objects should interact [15].

Modeling the CRA Problem. For this paper, we propose a simplified version of the CRA problem. As given elements we have a set of methods and attributes as well as dependencies between them. Such structure is also referred to as responsibilities dependency graph (RDG). Based on the RDG, the goal is

[2] http://martin-fleck.github.io/momot/.

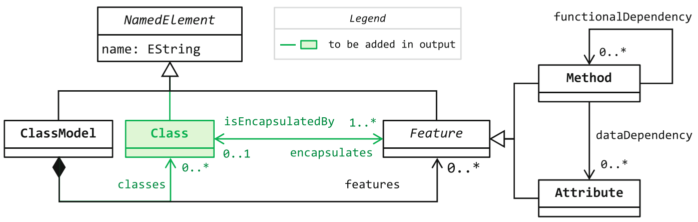


Fig. 3. RDG/CD metamodel. (Color figure online)

to generate a high-quality class diagram (CD). For this purpose, a RDG2CD model transformation is needed to evolve a RDG into a CD. Figure 3 depicts the metamodel that is used to represent both, the RDG and the CD. The RDG is the subgraph of the metamodel containing only the features and their dependencies (shown in black), while the additional class and relationships are needed to produce a CD (shown in green).

Transformation Goals. The goal is to produce high-quality CDs from RDGs. The CRA problem is a problem with a fast growing search space of potential class partitions given by the Bell number $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$. Already starting from a low number of features, the number of possible partitions is unsuitable for exhaustive search, e.g., 15 features yields 190899322 possible ways to create classes.

For determining the quality of the obtained CDs, we use two common metrics for considering the quality of grouping functionality into classes: coupling and cohesion [3]. *Coupling* refers to the number of external dependencies a specific group has, whereas *cohesion* refers to the dependencies within one group. Typically, low coupling is preferred as this indicates that a group covers separate functionality aspects of a system. On the contrary, the cohesion within one group should be maximized to ensure that it does not contain parts that are not part of its functionality. Mapping these definitions to our problem, we can calculate coupling and cohesion as the sum of external and internal dependencies, respectively. The formulae to calculate all necessary metrics and values are given below (taken from [15])[3]. Please note that $M(c)$ and $A(c)$ refer to all methods and attributes of class c, respectively, and $MMI(c_i, c_j)$ and $MAI(c_i, c_j)$ indicate the number of method-method and method-attribute interactions between classes c_i and c_j, respectively.

$$CohesionRatio = \sum_{c_i \in Classes} \frac{MAI(c_i, c_i)}{|M(c_i)| \times |A(c_i)|} + \frac{MMI(c_i, c_i)}{|M(c_i)| \times |M(c_i) - 1|}$$

$$CouplingRatio = \sum_{\substack{c_i, c_j \in Classes \\ c_i \neq c_j}} \frac{MAI(c_i, c_j)}{|M(c_i)| \times |A(c_j)|} + \frac{MMI(c_i, c_j)}{|M(c_i)| \times |M(c_j) - 1|}$$

[3] Zero is assigned to the result of a division whenever its denominator is zero.

Summing up, the challenge of this case study is to find a way to properly orchestrate transformation rules to optimize the quality of the produced CDs.

4 Developing Transformations with MOMoT

This section describes how transformations are developed with MOMoT based on the CRA case study.

Transformation Rules. First, MOMoT reuses Henshin to develop the necessary transformation rules. Furthermore, since in our approach we separate the objectives from the rules, no further adaptations to those rules are necessary. The rule required for the CRA case study is depicted in Fig. 4[4]. As we start with a random CRA solution which is improved by running the transformation, we simply need one rule which is re-assigning the features between different classes.

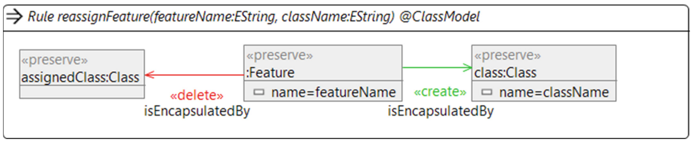


Fig. 4. Implementation of the reassign rule in Henshin.

Objectives. In addition to the rules, the objectives for the transformation have to be defined (cf. Listing 1.1). The calculation of the objective values given before as mathematical formulae have been implemented in Java for computing the coupling ratio and the cohesion ratio. An alternative provided by MOMoT as well would be to use OCL directly in the objective definitions. We also provide default objectives such as done for the solution length, i.e., the length of rule application sequences of the computed solutions. Moreover, constraints may be defined for determining the fitness of a solution. However, due to space restriction we do not further show this aspect for this case study and refer the interested reader to [9].

Listing 1.1. Specifying the Search Objectives

```
fitness = {
  objectives = {
    CouplingRatio : minimize { FitnessCalculator.calculateCoupling(root) }
    CohesionRatio : maximize { FitnessCalculator.calculateCohesion(root) }
    SolutionLength : minimize new TransformationLengthDimension } }
```

[4] Please note that MOMoT supports different Henshin transformation units and more complex transformations. However, for the purpose of the tool demonstration, we simply use one transformation rule and put the emphasis on the MOMoT specific features.

Search Configuration. After defining the objectives, the concrete search configuration used to find solutions best fulfilling the objectives is needed. For tackling this case study, we use three algorithms which are executed sequentially (cf. Listing 1.2). Specifically, we use NSGA-III and ε-MOEA [7] for multi-objective search which is needed as we have three partially conflicting objectives (cf. Listing 1.1). In addition, we use random search as a baseline comparison to demonstrate the need for a meta-heuristic search. As we are using population-based algorithms, we have to configure the population size for each generation as well as the stopping criteria as maximum evaluations per run. As meta-heuristic search includes some randomness, one may also define that the algorithms are executed several times to allow to draw statistical conclusions about the performance of the different algorithms.

Listing 1.2. Configuring the Search Algorithms and Parameters

```
algorithms = {
 Random:moea.createRandomSearch()
 NSGAIII:moea.createNSGAIII()
 eMOEA:moea.createEpsilonMOEA()}
experiment = {
 populationSize = 100
 maxEvaluations = 10000
 nrRuns = 30 }
```

5 Running and Analysing Transformations with MOMoT

In this section, we show how the developed MOMoT transformation is executed and discuss the transformation's output.

Transformation Input. The execution of MOMoT transformations are started with dedicated run configurations that execute the compiled MOMoT search configurations, as shown in Listing 1.3. Please note that input models are modeled in EMF and encoded in XMI. In order to allow for an efficient search, a preprocessing is possible to prepare an initial structure beneficial to perform the search (as done for the CRA case study by adding some new classes with random feature assignment) or to slice the model to reduce the memory consumption during the search process.

Listing 1.3. Defining the Transformation Input and Preprocessing

```
model = {
 file = "problem/Cart_Item.xmi"
 adapt = { var cm = root as ClassModel
  for(i:0 ..< cm.features.size - cm.classes.size) ... // add classes
  for(feature : cm.features) ... // distribute features randomly
  return cm } }
```

Transformation Results. MOMoT provides as transformation results: (i) the set of orchestrated transformation sequences leading to (ii) the set of Pareto-optimal output models with (iii) their respective objective values. The objective values may give an overview of how well the objectives are optimized. Listing 1.4 provides an excerpt of this configuration, and in addition, shows how results may be postprocessed and how specific solutions are selected.

Listing 1.4. Defining the Transformation Output and Postprocessing

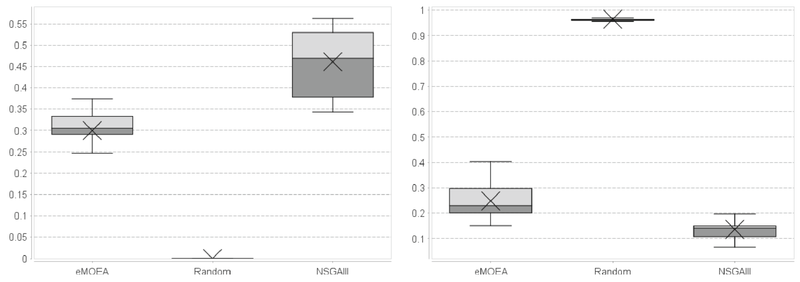
```
results = {
  adaptModels = { //remove empty classes
   root.classes.removeAll(cm.classes.filter[c | c.encapsulates.size == 0])}
  objectives = { outputFile = "output/objectives/objective_values.txt"}
  solutions  = { outputDirectory = "output/solutions/" }
  models = { outputDirectory = "output/models/" }
  models = { //select kneepoint models for further inspection
   neighborhoodSize = maxNeighborhoodSize
   outputDirectory  = "output/models/kneepoints/"}}
```

Results Analysis. MOMoT produces additional analysis to give more insights into the computed solutions and the relative algorithm performance (cf. Listing 1.5). For instance, we can statistically analyze dedicated performance indicators, such as Hypervolume, to compare the performance of different algorithms. This data can also be used to plot graphs to give a better overview about the analysis.

Listing 1.5. Defining the Statistical Analysis Methods

```
analysis = {
  indicators = [ hypervolume invertedGenerationalDistance ]
  significance = 0.01
  show = [ aggregateValues statisticalSignificance individualValues ] ...}
```

We use three algorithms in our case study, ε-MOEA, NSGA-III and Random Search (RS), and execute each algorithm 30 times. The results of the analysis are depicted in Fig. 5. We can clearly see that for the Hypervolume indicator, RS has the lowest and therefore worst value, while ε-MOEA has the highest value. A similar result is produced for the inverted generational distance, where lower values are considered better. The fact that a meta-heuristic search outperforms RS is a good indicator that the problem is suitable for SBSE techniques. In order to investigate the results further, MOMoT provides several other features to test and compare different algorithms [9].



(a) Hypervolume Indicator (b) Inverted Generational Distance

Fig. 5. Statistical analysis for the CRA case study results.

6 Conclusion and Future Work

In this paper we have shown the search capabilities of MOMoT for the CRA problem. We also contribute a tool for the scientific community to perform experimental research focusing on the usage of different meta-heuristic search algorithms for MDE problems.

While we already provide a wide spectrum of different search algorithms for orchestrating transformation rules, there is still room for future work. First, as we currently provide different algorithms but not their combination, we plan to incorporate Memetic Algorithms which allow for combined usage of global and local searchers. Second, we would like to explore the combination of search-based and approximate model transformations [16], i.e., how much precision may be traded for performance.

References

1. Abdeen, H., Varró, D., Sahraoui, H.A., Nagy, A.S., Debreceni, C., Hegedüs, Á., Horváth, Á.: Multi-objective optimization in rule-based design space exploration. In: Proceedings of ASE (2014)
2. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: Rouquette, N., Haugen, Ø., Petriu, D.C. (eds.) MODELS 2010, Part I. LNCS, vol. 6394, pp. 121–135. Springer, Heidelberg (2010)
3. Bowman, M., Briand, L., Labiche, Y.: Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE TSE **36**(6), 817–837 (2010)
4. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool, San Rafael (2012)
5. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
7. Deb, K., Mohan, M., Mishra, S.: A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions. Technical report, Indian Inst. of Technology Kanpur (2003)
8. Denil, J., Jukss, M., Verbrugge, C., Vangheluwe, H.: Search-based model optimization using model transformations. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (eds.) SAM 2014. LNCS, vol. 8769, pp. 80–95. Springer, Heidelberg (2014)
9. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proceedings of NasBASE (2015)
10. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)
11. Harman, M.: The current state and future of search based software engineering. In: Proceedings of FOSE @ ICSE (2007)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)

13. Kessentini, M., Langer, P., Wimmer, M.: Searching models, modeling search: on the synergies of SBSE and MDE. In: Proceedings of CMSBSE @ ICSE (2013)
14. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
15. Masoud, H., Jalili, S.: A clustering-based model for class responsibility assignment problem in object-oriented analysis. JSS **93**, 110–131 (2014)
16. Troya, J., Wimmer, M., Burgueño, L., Vallecillo, A.: Towards approximate model transformations. In: Proceedings of AMT @ MODELS (2014)