Using Multi-objective optimization to solve the Class Responsibility Assignment Case

Martin Fleck, Javier Troya, and Manuel Wimmer

Business Informatics Group, Vienna University of Technology, Austria
{fleck,troya,wimmer}@big.tuwien.ac.at

Abstract. This paper describes a solution to the Class Responsibility Assignment (CRA) case study of the eighth Transformation Tool Contest. The goal of the case study was to produce high-quality class diagram models from existing responsibility dependency graphs (RDG) containing methods and attributes as well as dependencies between them, but no classes. The quality of the generated class diagrams is evaluated using a combined measure of coupling and cohesion, the CRA-Index. The design space of all possible class diagrams grows exponentially with the input size, specifically the number of methods and attributes that need to be assigned to classes. We therefore propose to use search-based optimization techniques to tackle this case study. To be more precise, we use the MOMoT framework to deploy several evolutionary algorithms such as the NSGA-II to efficiently handle the large design space. By using evolutionary algorithms we separate the definition of the quality criteria from the definition of the transformation rules resulting in a highly flexible approach with respect to the quality criteria.

1 Introduction

As mentioned in the case description, the CRA is a problem with an exponentially 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., 18 features as provided by the second input model already yields 682076806159 possible ways to create classes. Even when dismissing special cases where we group all features into one class or every feature into a separate class, the number is still enormous. We therefore propose the usage of search-based optimization techniques which can deal with very large or even infinite design spaces.

Search-based software engineering (SBSE) [10] is a field that applies search-based optimization techniques to software engineering problems. Search-based optimization techniques can be categorized as metaheuristic approaches that deal with large or even infinite search spaces in an efficient manner. Concrete algorithms include local search methods such as Tabu Search [9] or Simulated Annealing [13] or genetic algorithms [11] such as the NSGA-II [3] or the NSGA-III [2]. Especially in recent years, SBSE has also been applied successfully in the area of model and program transformations. Examples include the generation of model transformations from examples [12, 15, 16], the optimization of regression tests for model transformations [14], the detection of high-level model changes [5] or the enhancement of the readability of source code for given metrics [6, 7].

For the CRA case study we use a combination of different algorithms provided by the *Marrying Optimization and Model Transformations* (MOMoT) approach [8] which is based on Henshin [1] and the MOEA framework [1] and has been developed in previous work. MOMoT combines different search techniques with model transformations to produce output models that optimize one or more quality criteria. MOMoT is summarized in the next section.

2 MOMoT at a Glance

This section summarizes the MOMoT approach and gives an overview on the most important aspects related to the case study. For a more detailed view on MOMoT, please consider previous work by the authors [8]. The actual orchestration of the approach for the case study as well as the produced class diagrams are described in Section 3.

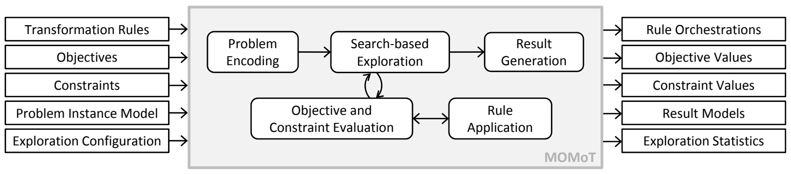

Fig. 1. Overview of the MOMoT approach

MOMoT[2] is a generic, task- and algorithm-agnostic approach that combines SBSE and model-driven engineering (MDE) to tackle complex problems. An overview of the necessary inputs and provided outputs is depicted in Figure 1. Specifically, domain-specific languages (DSLs) are used to model the problem domain and create problem instances, model transformations then manipulate those instances and SBSE techniques are used to efficiently handle the potentially infinite search spaces and guide the orchestration of rules according to given quality criteria Therefore, the purpose of MOMoT is to support model engineers in their task to orchestrate a given set of in-place transformation rules in order to produce an output model that optimizes certain quality criteria. In this sense, an orchestration refers to an ordered sequence of transformation rules, where the parameters of each transformation rule are assigned a specific value. Deriving such a rule orchestration is a non-trivial task whose complexity depends on the number of rules, the number of potential rule matches in the input model, and the value range of the rule parameters. Furthermore, the effect a certain transformation rule has on a specific quality criteria of the output model is often hidden implicitly in the behavior of the rule and

[1] `http://www.moeaframework.org`
[2] `https://code.google.com/p/momot/`

may depend on the context in which the rule is applied. Consequently, deriving such an orchestration manually is a complex and costly task requiring deep knowledge about the problem domain and the transformation rules. To allow for an automated search for such an orchestration, MOMoT uses dedicated, heuristic search algorithms to guide the search based based on a given set of objectives and constraints.

2.1 Transformation Rules

As mentioned previously, MOMoT is based on Henshin. Therefore all transformation rules that should be considered during the search for an optimal rule orchestration must be provided in the Henshin format. In Henshin all model transformation rules are based on a provided DSL (Ecore metamodel) and can be specified as graph transformation using the provided graphical notation.

2.2 Objectives & Constraints

Objectives and constraints are desired and undesired properties of the solution respectively. Together, they make up the *fitness* of the solution, i.e., the quality a solution possesses.

Objectives are properties that are desired in the resulting solution and can be broadly divided into domain-specific objectives and solution-specific objectives. Domain-specific objectives are related to the problem domain modelled in the DSL, e.g., increasing the CRA-Index of the resulting output model. Solution-specific objectives on the other hand are related to the solution representation, e.g., minimizing the number of necessary transformation rules. Both kinds of objectives can be specified in MOMoT using either OCL queries or Java methods. When using more than one objective, it has to be decided how solutions are compared to decide whether a solution is better than another solution. One way to do this, is to combine all objectives into one value. Another way is to use the notion of Pareto-optimality, i.e., one solution is better than another if it is better in at least one individual objective and no worse according to all others. In that case, the search finds a set of non-dominated solutions which is called the *Pareto front*.

In a similar fashion we can categorize constraints, i.e., properties that are not allowed in a solution in order for a solution to be considered valid. Domain-specific constraints relate to the problem domain, e.g., all features must be assigned to a class, whereas solution-specific constraints may put an upper bound on the number a certain transformation rule should be applied. During the search, all solutions that are violating any constraints are removed from the search space if possible. However, if the search is not executed long enough or the rules do not allow the production of valid solutions, invalid solutions may also be produced. Again, we can leverage OCL and Java to specify constraints. Additionally, it is also possible to provide constraints in the form of graph patterns, e.g., negative application conditions in the rule.

A generic fitness function which interprets the OCL, executes the Java and evaluates graph patterns based on Henshin, is used during the search to evaluate all provided objectives and constraints.

2.3 Exploration Configuration

As MOMoT is an algorithm-agnostic approach, the user has to specify a few exploration parameters. Most notably, the user has to decide which algorithm(s) to use. Depending on the algorithm type, different parameters can be set. As of now, MOMoT supports local search methods and evolutionary search methods. In any case, the generic fitness function is used to evaluate the quality of the solutions.

Local search methods Local search methods are algorithms that maintain one candidate solution, i.e., a sequence of parameterized rules, at once and try to improve that solution by looking at its neighbor solutions. A neighbor of a solution is a slight variation of a solution, e.g., a rule has a different parameter value or one out of many rules is different. Depending on the concrete algorithm, the search may only accept better solutions or also slightly worse solutions than the current one to move across the search space.

Evolutionary search methods Evolutionary search methods on the other hand are part of the population-based search methods. These search algorithms maintain a set of solutions (population) at once and typically employ three operators on that population. The initial population is usually random generated. First, a subset of the solutions are selected based on their fitness value and maybe some other criteria (*selection* operator). The selected solutions are then re-combined into new solutions (*recombination* operator), e.g., the one-point crossover operator splits the rule sequence of both solutions at a random point and combines the first part of the first solution with the second part of the second solution and vice-versa. Here, the assumption is that the combination of two good parent solutions may yield even better children. Finally, some of the solutions in the population are mutated (*mutation* operator), i.e., a slight change such as a parameter or a rule change is introduced. This avoids an early convergence of the search into one good area of the search space.

2.4 Generated output

After executing the search with all necessary inputs, i.e., the input model, the considered transformation rules, the objectives and constraints as well as the search configuration, MOMoT produces several artefacts. Besides the rule orchestration, i.e., the (Pareto-)optimal sequence of parameterized transformation rules, MOMoT also provides the output model in XMI format and the evaluation values for the defined objectives and constraints. Additionally, MOMoT can be configured to collect statistical data during the search in order to reason about the search process itself or to compare different algorithm runs. This functionality which is already provided by the MOEA framework, provides several indicators such as Hypervolume, Generational Distance or the Maximum Pareto Front Error. Most collected and calculated data can also be used to plot graphs giving a better overview about the algorithm executions.

3 Evaluation

This section describes the specification of the Class Responsibility Assignment case study for our approach. Furthermore, it shows how the approach can be executed for the provided specification and gives an overview of the produced class models for the provided input models.

3.1 Case Study Specification

In this section we present the case study-specific artefacts that are needed to execute MOMoT. All model artefacts such as the transformation rules are based on the meta-model provided by the case study.

Transformation Rules. As our approach also builds upon the Henshin graph transformation engine, we can re-use the rules that have been specified in the case description. Furthermore, since in our approach we separate the objectives from the rules, no further adaptations to those rules are necessary. For convenience, the rules have been taken from the case description and are depicted in Figure 2. Concerning the rule parameters, we only have to take of the `className` parameter of the `createClass` rule as the parameters of the `assignFeature` rule can be matched automatically by Henshin based on the class model generated so far. To make sure that we only produce class names that are unique, we use a simple naming scheme that starts with `Class_A` as a name and increments the last section of the following sequence of names: `Class_A`, ..., `Class_Z`, `Class_Za`, ..., `Class_Zz`, `Class_Zaa`, ...

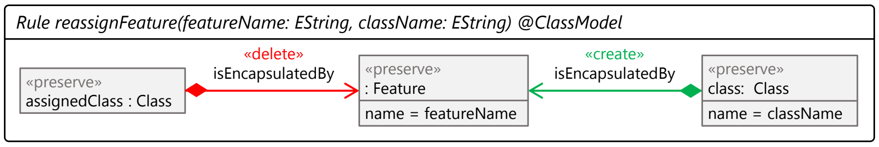

Fig. 2. Implementation of the reassign feature rule in Henshin.

Objectives. The calculation of the objective values given in the case description as mathematical formulae have been implemented as a Java method doing the same calculation. Additionally we add an objective that calculates the number of classes in the class model. This allows us to produce the Pareto-optimal set of class models with one algorithm run. Having a look at the quality characteristics of the Pareto set, we can further reason about the optimal number of classes when only considering the *CRA-Index* as an objective.

Constraints. From the original three constraints in the case description, only one needs to be implemented as a constraint in the fitness function to make sure that all features are actually assigned. To be more precise, the implementation returns the degree of constraint violation, i.e., the number of unassigned features times a penalty value. In most cases, models that have any constraint violation, are removed from the search space during the algorithm execution. In case this is not possible, i.e., there are no rule executions that produce a completely valid model, the algorithm favours the models with the least violation. Summarized, the constraints are handled as described in Table 1.

Table 1. Handling of the case constraints

Constraint	Solution
No Empty Classes	Taken care of by the provided metamodel.
Unique Class Names	Only unique class names are produced when applying the *create-Class*-rule, i.e., class names are generated incrementally.
All Features Assigned	The fitness function assigns penalty values in case features are not assigned. During the search, models with penalty values are removed if possible.

Algorithm Configuration. For tackling this case study, we use a combination of two algorithms which are run sequentially and yields the best result found by all algorithms. Specifically, we use the NSGA-II and ε-MOEA [4]. Each algorithm is executed between one and five times. The concrete algorithm configuration is summarized in Table 2.

Table 2. Algorithm configuration

Parameter	Value
Population Size	100
No. of Iterations	100
Selection	TournamentSelection with $k = 2$
Crossover	OnePointCrossover with $p = 1.0$
Mutation	PlaceholderMutation with $p = 0.2$
	ParameterMutation with $p = 0.1$

3.2 Execution

The MOMoT approach as well as the specification for the case study is provided as Eclipse plugins. All case study-specific classes can be found in the following project:

```
at.ac.tuwien.big.momot.examples.modularization.cra
```

The actual search process can be started by executing the class `ArchitectureCRA` with a specific input model.

3.3 Generated Class Diagrams

All produced class diagrams are saved in the necessary XMI format and attached in the zip file accompanying this solution. Actually, we produce multiple class diagrams for each input model: one model for each Pareto-optimal combination of CRA-Index and number of classes. However only the best models, according to the CRA-Index, have been submitted, independently of the number of classes. According to our own evaluation, wo fulfil the *completeness* and *correctness* criteria for all our models. The best CRA-Index value for each input model as well as the necessary execution times to achieve this model are summarized in Table 3.

Table 3. Summary of input models

	Input A	**Input B**	**Input C**	**Input D**	**Input E**
Attributes	6	10	15	18	20
Methods	6	15	17	26	40
Functional Dep.	10	15	30	45	50
Data Dep.	10	15	30	45	50
Correctness	✓	✓	✓	✓	✓
Completeness	✓	✓	✓	✓	✓
CRA-Index	3.0	3.083	-3.79	-23.63381	-66.65545
Performance	04:03.436	05:05.156	12:02.776	26:51.582	38:08.067

References

1. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In: Proc. of MODELS. LNCS, vol. 6394, pp. 121–135. Springer (2010)
2. 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. on Evolutionary Computation 18(4), 577–601 (8 2014)
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
4. Deb, K., Mohan, M., Mishra, S.: A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions. Tech. Rep. 2003002, Indian Institute of Technology Kanpur (2003)
5. ben Fadhel, A., Kessentini, M., Langer, P., Wimmer, M.: Search-based detection of high-level model changes. In: Proc. of ICSM. pp. 212–221. IEEE (2012)
6. Fatiregun, D., Harman, M., Hierons, R.M.: Search Based Transformations. In: Proc. of GECCO. LNCS, vol. 2724, pp. 2511–2512. Springer (2003)
7. Fatiregun, D., Harman, M., Hierons, R.M.: Evolving transformation sequences using genetic algorithms. In: Proc. of SCAM. pp. 66–75. IEEE (2004)
8. Fleck, M., Troya, J., Wimmer, M.: Marrying Search-based Optimization and Model Transformation Technology. In: Proceedings of the First North American Search Based Software Engineering Symposium. pp. 1–16. Elsevier (2015), To be published.
9. Glover, F.: Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research 13(5), 533–549 (1986)
10. Harman, M.: The Current State and Future of Search Based Software Engineering. In: Proc. of FOSE @ ICSE. pp. 342–357 (2007)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press (1992)
12. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model Transformation By-Example: A Survey of the First Wave. In: Conceptual Modelling and Its Theoretical Foundations. LNCS, vol. 7260, pp. 197–215. Springer (2012)
13. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)
14. Shelburg, J., Kessentini, M., Tauritz, D.: Regression Testing for Model Transformations: A Multi-objective Approach. In: Proc. of SSBSE, LNCS, vol. 8084, pp. 209–223 (2013)
15. Varró, D.: Model Transformation by Example. In: Proc. of MoDELS. pp. 410–424 (2006)

16. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards Model Transformation Generation By-Example. In: Proc. of HICSS. IEEE (2007)