Searching Models, Modeling Search:

On the Synergies of SBSE and MDE

Marouane Kessentini
Computer Science Department
Missouri University of Science and Technology
Rolla, USA
marouanek@mst.edu

Philip Langer, Manuel Wimmer
Business Informatics Group
Vienna University of Technology
Vienna, Austria
{langer,wimmer}@big.tuwien.ac.at

***Abstract*—In the past years, several researchers applied search-based optimization algorithms successfully in the software engineering domain to obtain automatically near-optimal solutions to complex problems posing huge solution spaces. More recently, such algorithms have also been proven useful for solving problems in model engineering. However, applying search-based optimization algorithms to problems in model engineering efficiently and effectively is a challenging endeavor demanding for expertize in both, search-based algorithms as well as model engineering formalisms and techniques. In this paper, we report on our experiences in applying such search-based algorithms for model engineering problems and propose a model-driven approach to ease the adoption of search-based algorithms for the area of model engineering.**

***Index Terms*—Search-based Engineering, Model Engineering, Model-driven Software Engineering, Genetic Algorithms**

I. Introduction

Model-driven Engineering (MDE), as one of the most prominent protagonists of *model engineering* [8], is a discipline that relies on *models* as first class entities and that aims to develop, maintain, and evolve software systems by performing *model transformations* [1] on models abstracting from concrete implementations of the systems.

As many other domains of software engineering, the MDE community is currently concerned with finding *exact* solutions to these problems causing high efforts to actually achieve these solutions. An alternative approach is to search for solutions that fall within a specified *acceptance margin*. Search-based optimization techniques are well suited for this purpose. Although, Search-based Software Engineering (SBSE) has been successfully applied to a number of different MDE tasks, such as model transformation[2][12], model evolution [4][9], model analysis [4], and model transformation testing [10], applying SBSE to complex MDE problems necessitates expertise in both, search-based optimization algorithms and MDE formalisms and techniques. It is thus desirable to define a generic process supported by a dedicated framework that can be applied to solve various MDE problems by using search-based algorithms with low adaptation effort and expertise.

II. Related Work

The term SBSE was coined by Harman and Jones in 2001[11]. As its name implies, SBSE treats software engineering problems as search problems, and seeks to apply search techniques in order to solve them.

MDE represents one of the application domains of SBSE. We define Search-based Model-Driven Engineering (SBMDE) as the application of optimization techniques to solve MDE problems. The approach is motivated by the fact that, for many MDE problems, it is infeasible to apply a precise analytic algorithm that produces the “optimal” solution to the problem. Thus, as indicated by a recent SBSE survey [2], the application of search techniques in MDE is emerging, but still not as prominent as for traditional code-based software engineering.

To improve the design quality, many SBSE approaches have been proposed [14][15][16]. Most of these works search to find the best combination of refactorings that can improve quality metrics. Due to the large number of possible refactoring combinations, different metaheuristic search approaches are used and compared. The clustering problem is also studied using search-based techniques [19][20]. The majority of these studies are based on the use of cohesion and coupling to find the best clustering of model elements that increases cohesion and decreases cohesion. The generation of test cases to test metamodels/models and transformation mechanisms are also considered as an optimization problem [17][18] where the goal is to find the best set of test cases that covers metamodel elements and transformation possibilities.

Although a few reusable frameworks emerged to ease adopting of search-based approaches for software engineering problems, there is no framework *(i)* that specializes on adopting SBSE for MDE, *(ii)* that exploits the benefits of MDE itself, and *(iii)* allows developers to remain in the technical space of MDE when developing search-based algorithms for MDE.

III. Applying Search-based Engineering in MDE

In this section, we summarize our experiences of applying SBSE techniques, in particular genetic algorithm (GA) [13], for MDE problems. In the following we focus on the process of applying GA, as well as the interfaces that have to be realized for adopting GA for a specific problem. After introducing the process of GA in general, we show how we applied GA to solve optimization problems in the MDE domain and discuss some lessons learned.

In Figure 1, the general process of GA, as well as the interfaces to be realized for applying GA to a specific optimization problem, is depicted. In general, the following interfaces have to be realized: encoding of the individuals, creation of an initial population of individuals, evaluation of individuals using a fitness function to determine a quantitative measure reflecting their ability to represent a solution for the problem under consideration, selection of the individuals to transmit from one generation to another, and the creation of new individuals using genetic operators (crossover and mutation) to explore the search space. In the following, we show how we have realized this process for two important MDE problems.

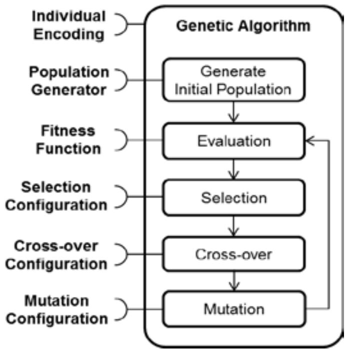


Figure 1: Interfaces of Genetic Algorithms.

A. Experience 1: Model Transformation as Optimization Problem

In [2][12], we introduced an optimization-based approach to automate model transformations. In particular, model transformations are treated as a combinatorial optimization problem where the transformation of a source model is obtained by finding, for each of its model elements, a similar transformation from an already existing example base, i.e., an example comprises a source model, a target model, and trace links between source and target model elements.

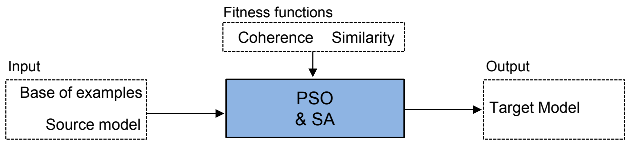


Figure 2: Searching for Target Models.

Due to the large number of possible combinations, a heuristic-search strategy has been used to build the transformation solution from a set of individual model element transformations (cf. Figure 2). To apply search-based techniques, we had to determine the following aspects: **Search algorithm**: For generating the solutions, we used two strategies based on two search-based algorithms, namely particle swarm optimization (PSO) [6] and simulated annealing (SA) [7]; **Individual encoding**: For representing models in the technical space of SBSE, we used a predicate representation that serializes models into a simple human-readable text format. Because of the complex structure of models as well as the type information, we could not reuse a generic binary encoding; **Fitness function**: The example base comprises models that are considered as good solutions for the current transformation problem. The fitness functions are evaluating a produced solution, i.e., the target model, concerning its deviation to the existing target models in the example base, as well as additional design metrics for determining the quality of the target model.

B. Experience 2: Model Change Detection as Optimization Problem

Recently, we proposed an approach for detecting model changes, such as refactoring applications, that have been applied between two successive versions of a model [9]. In particular, we addressed the problem that several different model evolutions may describe the transition from the same initial model version to the revised one. Due to the large number of possible change combinations, a heuristic-search strategy has been used to build the sequence of change operation applications from a given set of change operation types (cf. Figure 3). We implemented the search-based approach in the following manner: **Search algorithm**: For computing the solutions, we used and adapted genetic algorithm (GA) as global heuristic search method to compute the sequence of change operations; **Individual encoding**: For representing change models that have to be optimized, we used a predicate representation that represents model evolutions as sequences of change operation applications; **Fitness function**: In the first version, we used a mono-objective approach searching for change sequences that are able to produce as much as possible a similar model than the given revised model. Thus, we apply the detected changes on the initial model by executing the operations as given by the change sequence. The output of this execution is a computed revised model that is compared using model differencing approaches [5] with the actual revised model.

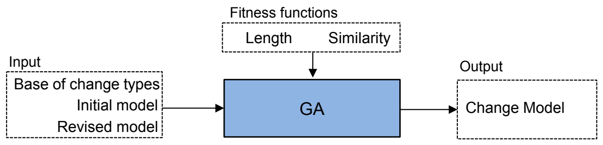
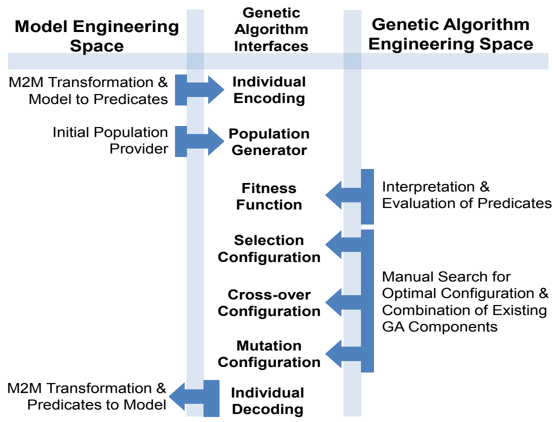


Figure 3: Searching for Change Sequence.

C. Lessons Learned

Generalizing from the two case studies where we applied SBSE in the context of MDE, we may derive a general approach for realizing the interfaces of GA as depicted in Figure 4. In our cases, we switched from the MDE field to the SBSE field by encoding models into a dedicated format and by tailoring existing search-based algorithms to this representation using a general purpose programming language. The computed solutions are also transferable again to the MDE field for reasoning on their quality. Some fitness functions are basic functions, such as computing the number of elements, others are complex functions, often referred to as model management operators, such as model differencing. For bridging the gap between MDE and SBSE, we used model transformations (model-to-text) to encode the models in SBSE-specific formats.

Whereas we achieved promising results of applying SBSE for solving MDE problems, we also faced major challenges. First, selecting an appropriate optimization algorithm for a specific problem is the major challenge. While the decision whether a mono-objective or a multi-objective technique should be applied is determined by the understanding of the specific problem, the selection of the actual algorithm out of the huge set of existing ones requires for a subsequent evaluation. Second, concerning the formulation of the fitness functions, efficiency is the main goal. Often, it is even necessary to trade precision with efficiency. Third, because of the first two challenges, the availability of well-understood case studies to reason on the quality of the computed solutions is of paramount importance. To achieve an objective evaluation, several runs of the algorithms, several kind of inputs, automatic and manual evaluation methods have to be considered. Fourth, while MDE dictates the usage of models to raise the abstraction and ease analyzing the problem domain, current adoptions of SBSE for MDE are hard-coded for specific encodings and employ general purpose programming langauges. Here the question arises whether domain-specific languages may stimulate the adoption of SBSE for MDE by exploiting the MDE paradigm itself. Applying search-based algorithms for solving problems in MDE is possible but challenging. However, as we have experienced in our recent case studies, several of the components that realize certain parts of a search-based application bear the potential for reuse across different applications of search-based algorithms in MDE. Therefore, we propose a framework leveraging MDE techniques for applying SBSE in the context of MDE, not only to increase efficiency and reuse, but also to lower the barriers of applying SBSE for MDE. One major goal of this framework is to lift those artifacts that currently have to be developed in the SBSE space to the MDE space, such as the realization of the fitness function, as well as the configuration and composition of optimization operators. In the following, we sketch initial ideas of such a model-based framework focusing on GA in a first step.



Model Engineering Space
Genetic Algorithm Interfaces
Genetic Algorithm Engineering Space
M2M Transformation & Model to Predicates
Individual Encoding
Initial Population Provider
Population Generator
Fitness Function
Interpretation & Evaluation of Predicates
Selection Configuration
Cross-over Configuration
Mutation Configuration
Manual Search for Optimal Configuration & Combination of Existing GA Components
M2M Transformation & Predicates to Model
Individual Decoding

Figure 4: Realizing GA Interfaces in the context of MDE.

IV. MDE FOR SBSE FOR MDE

A. Individual Encoding

The first task is to realize the encoding of models into the technical space of GA. Therefore, one typically has to transform the input models into formats dictated by the used GA framework. When implementing the encoding for our recent case studies, we realized that an intermediate modeling language, formalized by the *encoding metamodel*, which follows certain GA-specific patterns and which is close to a common format suitable for GA encoding, would enable employing a generic *bidirectional model-to-text transformation*, as well as generic components realizing multiple of the remaining steps in the GA automatically, such as selection, cross-over, and mutation. Having designed such an encoding metamodel, the input models would just have to be transformed into the encoding metamodel using model-to-model transformations.

In Figure 5, we sketch a *generic encoding metamodel* that can be extended (in terms of subclassing) for a particular problem domain to obtain a problem-specific *encoding metamodel*. As a result, each concrete encoding metamodel consists of concrete metaclasses that describes the structure of an *individual*, which again consists of an ordered set of *individual atoms*, which in turn can be parameterized with *parameters* and *parameter values*. In the context of the case study related to model changes detection, an individual would be an ordered set of model changes (i.e., individual atoms), which can be parameterized in terms of the model elements the specific model change affected. Besides representing individuals, often *auxiliary models* are necessary for computing the fitness of individuals. For instance, in the case study outlined for changes detection, the initial and the revised model would act as such auxiliary models. The encoding metamodel shall act as the central hub for the remaining artifacts that need to be specified for realizing the complete GA process. Thus, it should be possible to specify and apply all components on model-level (according to the encoding metamodel) and not as it is done now, on the level of the raw predicate encoding.

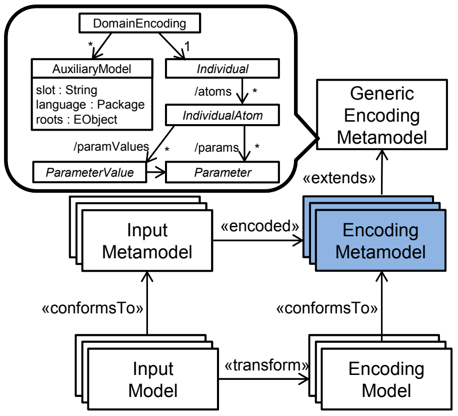


Figure 5: Model-based Framework for Encoding Individuals.

B. Population Generator and Fitness Function

In the first step of the GA, we have to provide a generator for an initial population (cf. Figure 6). Therefore, arbitrary model engineering languages, such as EOL[1], or dedicated frameworks, such as Ecore mutator[2], may be used to generate instances of the respective subclass of the metaclass *individual* in the encoding metamodel. With the selection of the most appropriate framework or language, the user has the flexibility

[1] http://www.eclipse.org/epsilon/doc/eol
[2] http://code.google.com/a/eclipselabs.org/p/ecore-mutator

to decide whether to generate random instances or to follow other generation strategies (e.g., pre-computed solutions by enumeration-based approaches that have to be optimized). Besides providing an initial population, a problem-specific fitness function has to be developed based on the encoding metamodel (cf. Figure 6). Therefore, we propose to use an extensible component called *fitness evaluator* that consists of an *individual interpreter* and the actual *fitness function*. The individual interpreter is necessary, because in many cases the solution has to be processed (e.g., compared, transformed, or simulated, etc.) before its fitness can be evaluated. The result of this processing can be saved in terms of an auxiliary model (e.g., a difference model, transformation trace, etc.). This auxiliary model can now be used by the actual fitness function, which can easily be realized by a model query (e.g., number of differences, or other quality metrics).

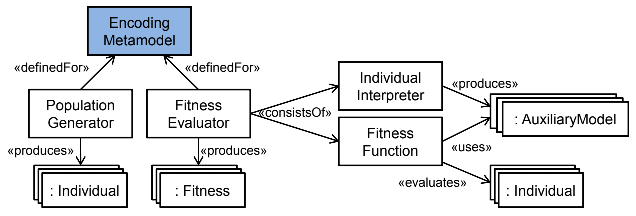


Figure 6: Model-based Framework for Generating an Initial Population and Evaluation of Fitness.

C. Selection, Cross-over, and Mutation

The selection is typically a query obtaining a subset of individuals that satisfy a certain fitness threshold. As the generic encoding metamodel dictates that every individual has to be an instance of *individual* (indirectly) and the framework knows how to compute the fitness (using the *fitness evaluator*), it is easy to provide a configurable component for that. The same is true for the cross-over component. Basically, an individual consists of parameterized *individual atoms*. Thus, new child individuals can be produced by intermingling the atoms of two parent individuals.

Also for the mutation, generic components can be provided, as the order of the atoms as well as the concept of *parameters* and *parameter values* are explicitly represented in the generic encoding metamodel. Thus, individuals can be mutated by re-ordering their atoms or assigning new parameter values randomly. To limit the search space, additional validation constraints can be attached to the encoding metamodel. Of course, generic components might not always satisfy all needs of users for applying GA to a MDE problem. Thus, users should be empowered to plug-in alternative implementations of these components. The main benefit, also if these implementations have to be provided by the users themselves, is that these components can still be realized using MDE techniques in contrast to implementing those on the level of the raw encoding. Probably we might need predefined mutation operations. Alternatively this can be done by changing the order of the individual atoms and their parameter values.

V. Next Steps

In this paper, we proposed an MDE-based framework for easing the adoption of SBSE to other MDE problems. The next step is to realize the sketched framework based on EMF. As soon as we complete a first prototype, we will evaluate it by re-producing existing hand-crafted applications of SBSE to MDE problems using the proposed framework instead.

References

[1] T. Mens, P. Van Gorp: A Taxonomy of Model Transformation. *Electr. Notes Theor. Comput. Sci*. 152:125-142 (2006).

[2] M. Harman, S.A. Mansouri, Y. Zhang: Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys* 45(1) (2012).

[3] M. Kessentini, H. Sahraoui and M. Boukadoum: Model Transformation as an Optimization Problem. In *Proc. of MODELS*, (2008).

[4] M. Shousha, L. Briand, Y. Labiche: A UML/SPT Model Analysis Methodology for Concurrent Systems Based on Genetic Algorithms. In *Proc. of MODELS*, (2008).

[5] D.S. Kolovos, R.F. Paige, F.A.C. Polack. Model Comparison: A Foundation for Model Composition and Model Transformation Testing. In *Proc. 1st International Workshop on Global Integrated Model* Management (GaMMa), (2006).

[6] J. Kennedy and R.C Eberhart: Particle swarm optimization. In *Proc. of Int. Conf. on Neural Networks*, (1995).

[7] D.S. Kirkpatrick, Jr. Gelatt and M.P. Vecchi: Optimization by simulated annealing. *Science*, 220(4598):671-680, (1983).

[8] J. Bézivin: On the unification power of models. *Software and System Modeling* 4(2):171-188 (2005).

[9] A. ben Fadhel, M. Kessentini, P. Langer, M. Wimmer: Search-based detection of high-level model changes. In *Proc. of ICSM*, (2012).

[10] M. Kessentini, H. A. Sahraoui, M. Boukadoum: Example-based model-transformation testing. *Autom. Softw. Eng.* 18(2): 199-224, (2011).

[11] M. Harman, B.F. Jones: Search-based software engineering. *Information & Software Technology* 43(14): 833-839, (2001).

[12] M. Kessentini, H.A. Sahraoui, M. Boukadoum, O. Ben Omar: Search-based model transformation by example. *Software and System Modeling* 11(2): 209-226, (2012).

[13] J.R. Koza.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. *MIT Press, Cambridge,* (1992).

[14] A. Ouni, M. Kessentini, H. Sahraoui and M. Boukadoum, Maintainability Defects Detection and Correction: A Multi-Objective Approach. *Autom. Softw. Eng.*, (2012).

[15] M. Harman, and L. Tratt, Pareto optimal search based refactoring at the design level, In *Proc of GECCO*, (2007).

[16] M. O'Keeffe, and M. O. Cinnéide, Search-based Refactoring for Software Maintenance. *Journal of Systems and Software*, 81(4), 502–516, (2008).

[17] B. Baudry, F. Fleurey, J.-M. Jezequel, and Y. L. Traon. Automatic test cases optimization using a bacteriological adaptation model. In *Proc. of ASE*, (2002).

[18] M. Kessentini, H.A. Sahraoui, M. Boukadoum: Example-based model-transformation testing. *Autom. Softw. Eng.* 18(2): 199-224 (2011).

[19] K. Praditwong, M. Harman, X. Yao: Software Module Clustering as a Multi-Objective Search Problem. *IEEE Trans. Software Eng.* 37(2): 264-282, (2011).

[20] K. Mahdavi, M. Harman, R.M. Hierons: Finding Building Blocks for Software Clustering. In *Proc. of GECCO*, (2003)