MDEOptimiser: A Search Based Model Engineering Tool

Alexandru Burdusel
Department of Informatics, King's College London
London, United Kingdom
alexandru.burdusel@kcl.ac.uk

Steffen Zschaler
Department of Informatics, King's College London
London, United Kingdom
szschaler@acm.org

Daniel Strüber
Universität Koblenz-Landau
Koblenz, Germany
strueber@uni-koblenz.de

ABSTRACT

Model Driven Engineering (MDE) is a methodology that aims to simplify the process of designing complex systems, by using models as an abstract representation of the underlying system.

This methodology allows domain experts to more easily focus on system design, where their knowledge is more useful, without having to work with the system implementation complexities. Search Based Model Engineering applies MDE concepts to optimisation problems. The goal is to simplify the process of solving optimisation problems for domain experts, by abstracting the complexity of solving optimisation problems and allowing them to focus on the domain level issues..

In this tool demostration we present MDEOptimiser (MDEO), a tool for specifying and solving optimisation problems using MDE. With MDEO the user can specify optimisation problems using a simple DSL. The tool can run evolutionary optimisation algorithms that use models as an encoding for population members and model transformations as search operators. We showcase the functionality of the tool using a number of case studies. We aim to show that with MDEO, specifying optimisation problems becomes a less complex task compared to custom implementations.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

model driven engineering, search based optimisation, search based model engineering, search based software engineering

ACM Reference Format:
Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. 2018. MDEOptimiser: A Search Based Model Engineering Tool. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3270112.3270130

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

ACM ISBN 978-1-4503-5965-8/18/10…$15.00
https://doi.org/10.1145/3270112.3270130

1 INTRODUCTION

In this demonstration we are going to show MDEOptimiser [1], a search based model engineering (SBME) tool that aims to simplify the process of specifying optimisation problems. The tool uses MDE to help users specify optimisation problems through an easy-to-use domain-specific language (DSL). MDEO requires the user to provide a problem specification that is easy to define using the implemented DSL. This is then used to run evolutionary optimisation algorithms to search for valid solutions to the specified problem. The tool uses models to represent individual candidate search solutions and the metamodel to define the problem search space.

The motivation for our approach is to use models as individual solution encodings. This encoding reduces the need to convert the genotype to a phenotype to evaluate the quality of a model. In our approach, search operators can be implemented as model transformations. The idea of searching directly on models was first described in [4]. Running search directly on models, is often needed when we are interested in finding optimal models [10]. Other approaches, seek to optimise transformation chains, where the goal is to optimise how to reach the optimal solutions [1, 7].

The remainder of this demonstration is organised as follows. First in Sect. 2 we give a a background of the concepts needed to understand for this demonstration. Then, in Sect. 3 we introduce two case studies which will be used in this demonstration. In Sect. 4 we describe the architecture of our tool, followed by a description of the MDEO DSL in Sect. 5. In Sect. 6 we describe a video demonstration of the tool. In Sect. 7 we present related work and in Sect. 8 we conclude with a discussion about future tool improvements.

2 BACKGROUND

In this section we introduce the concepts that are at the core of MDEO. We will start by giving a description of MDE, followed by a description of search-based software engineering.

MDE is a methodology that allows engineers to reason about complex systems using an abstract representation, allowing them to avoid the complexity of working directly with the implementation details. The main abstraction layers introduced by MDE are system, model, metamodel and meta-metamodel [5]. A core concept introduced by MDE are model transformations, defined as a set of rules that can be used to convert a source model to a target model [11]. A transformation that runs on a model in the same language is classified as endogenous [11].

Henshin is an in-place model transformation language, built to run on EMF models [2]. The transformation engine is based on algebraic graph transformation concepts. The tool visual representation of this concept allowing users to specify complex transformation

[1] https://mde-optimiser.github.io

rules with ease. A Henshin transformation rule consists of a left hand side (LHS) and a right hand side (RHS) graph. The tool also allows the specification of application conditions to identify the conditions under which a transformation rule can be applied.

Search-based model optimisation is a term that describes the use of MDE to solve search based optimisation problems [10]. This approach aims to simplify the specification of optimisation problems by taking advantage of the abstraction benefits offered by MDE.

To solve a problem using a search-based algorithm the following components are required:

- A representation for solution candidates;
- A method to specify and apply search operators to generate new solution candidates from existing ones;
- A method to evaluate the quality of a solution candidate.

In SBME we use models as a solution representation. The metamodel defines the available search space. Search operators, such as mutation or breeding are implemented using endogenous transformations. To evalute the fitness of a solution we use model querying languages such as OCL or Java.

3 RUNNING EXAMPLE

For this demonstration we are going to use two case studies to showcase the features of our tool. The first case study, described in Sect. 3.1 is a single objective optimisation problem. We will use this case study to describe the functionality of MDEO in the remainder of this paper. The second case study, described in Sect. 3.2 is a multi-objective optimisation problem. We are including it to show how our tool can be used to solve optimisation problems that have more than one objective.

3.1 Class-Responsibility Assignment

The Class-Responsibility Assignment case study, has been presented at the Transformation Tool Contest 2016 [8]. The problem is from the field of software engineering. The input of the CRA case study is a responsibilities dependency graph (RDG), formed of a set of methods, attributes and dependencies between them. The goal of the problem is to create a high-quality class diagram model, starting from the RDG model. To solve this single objective problem, the user is required to create classes and assign to them methods and attributes with the following objectives and constraints:

- **Objective 1** maximise the CRA index;
- **Constraint 1** all RDG features must be assigned to a class;
- **Constraint 2** no solution must have empty classes;

The latter constraint is enforced by the problem metamodel multiplicities. A complete description of the case study and how the CRA index is calculated can be found in the case study paper [8].

3.2 Scrum Planning

The Scrum Planning case study is an optimisation problem that aims to solve the challenging task of maximising stakeholder value when implementing a software application using the Scrum agile framework [12]. In this adaptation of the Scrum methodology, we seek to find the best distribution of work items across a number of Sprints, such that no Stakeholder has to wait too long before any of the work he is interested in is being planned.

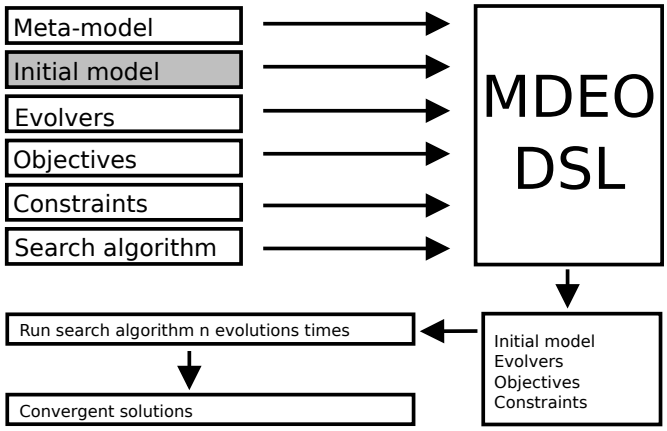

Figure 1: MDEO Tool Architecture

Scrum is a development methodology that introduces sprints as fixed time product development iterations. Each Sprint is employed by a set of roles: the product owner, who represents the product stakeholders; the development team, who build the product; and the Scrum master, who is the Scrum process facilitator. Two main artifacts of Scrum are the product backlog, a list of product requirements, and the Scrum backlog, a list of WorkItems that the development team have to address in the next Sprint. At the beginning of each Sprint, the development team, agree on the development tasks list which can be undertaken in the next Sprint. Each work item has the development effort required to implement it measured in story points. The average total of story points from a Sprint shows the team velocity.

The inputs of the Scrum Planning problem are: a model containing a product backlog, a list of stakeholders and a stakeholder importance metric for each backlog item; the team velocity metric.

The goal of the problem is to create a plan, by creating several Sprints, and to assign WorkItems to each Sprint with the following objectives and constraints:

- **Objective 1** minimise the Sprint effort deviation (the team velocity in any Sprint is not less than the total number of planned story points);
- **Objective 2** minimise the Stakeholder Satisfaction Index (ensure stakeholder task importance is equally distributed between sprints);
- **Constraint 1** all WorkItems must be assigned to a sprint;
- **Constraint 2** no sprint must have more story points than the team velocity.

4 ARCHITECTURE

In this section we give a high level overview of the MDEO architecture.

In Fig. 1 we include a diagram showing the architecture of MDEO. The tool is built as an XText DSL [2] and works with models built in the context of the Eclipse Modelling Framework [3].

[2] https://www.eclipse.org/Xtext/
[3] https://www.eclipse.org/modeling/emf/

```
basepath <src/main/resources/models/cra/>
metamodel <architectureCRA.ecore>
model <TTC_InputRDG_A.xmi>
objective CRA maximise java { "models.cra.fitness.MaximiseCRA" }
constraint MinimiseClasslessFeatures java
    { "models.cra.fitness.MinimiseClasslessFeatures" }
mutate using <craEvolvers.henshin> unit "createClass"
mutate using <craEvolvers.henshin> unit "assignFeature"
mutate using <craEvolvers.henshin> unit "moveFeature"
mutate using <craEvolvers.henshin> unit "deleteEmptyClass"
optimisation provider moea algorithm NSGAII variation mutation
parameters {
        population: 40
    }
termination {
        time: 60
    }
batches 30
```

Figure 2: MDEO DSL Specification of the CRA Problem

The optimisation algorithms supported by MDEO are implemented using the MOEA Framework [4]. MDEO implements a wrapper around the MOEA Framework, which translates the algorithm parameters specified in the DSL to the MOEA Framework algorithm factory.

When running the tool, the DSL parser validates the specification given by the user, and if the validation is successful the tool initialises the optimisation algorithm using the specified parameters. Following a successful run, the tool outputs a `csv` file containing overall experiment information containing information about the batch id, duration, objective and constraint values. When running multiple batches of the same experiment, the tool outputs the best solutions found in the results folder for each of the batches. The results for all the batches are grouped in a `csv` file saved in the results folder created for the experiment.

5 DSL DESCRIPTION

In this section we include an example of our DSL showing a specification of the CRA case study. We then explain the language syntax.

In Fig. 2 we include an example specification of the CRA case study for MDEO. For specifying optimisation problems using MDEO the user is required to use the following keywords to load the problem artifacts and to configure the optimisation algorithm:

(1) The `basepath` keyword configures the root location were the metamodel, model and transformations are located. This path must be relative to the project path in Eclipse or to the directory from where MDEO is executed when running in headless mode.
(2) The `metamodel` keyword specifies a metamodel describing the problem search space. This metamodel must describe the complete set of possible models that can be valid solutions for the problem being specified.
(3) `model` loads an initial model serialised in `xmi` format. This model must conform to the loaded `metamodel` and can be randomly generated valid metamodel instance or an existing model which needs to be improved using search-based optimisation.
(4) The `objective` keyword is used to load one or more objective functions implemented either as simple OCL queries over models or as more complex Java implementations. When loading an objective implemented using Java, the user must specify the full namespace of the function, so that it can be loaded using reflection. Objectives implemented using java must implement the `IGuidanceFunction` interface.
(5) The `constraint` keyword is used to load constraint functions. The functionality is similar to the `objective` keyword. The only difference is that functions loaded using this keyword will be treated as constraints by the DSL.
(6) Using the `mutate` or `breed` keywords the user can load a number of Henshin model transformations that can be used to derive new candidate solutions from existing ones. The `mutate` keyword loads mutation operators that are randomly applied during the search. The `breed` keyword loads a breeding operator that combines to models to create a solution. In Fig. 2 we only include the `mutate` keyword. An example of a breeding operator encoded as a Henshin transformation has been included in Fig. 3. In Fig. 4 we include two example mutation operators for the CRA case study, one to create a `class` and assign an existing `feature` to it, that has not already been assigned to a `class` and one to delete an empty `class`. The most recent version of MDEO can automatically generate atomic consistency preserving search operators (aCPSO). aCPSOs are atomic search operators that can create or delete a node or an edge, and also include the necessary repair operation to ensure these operations can be performed without generating invalid models.
(7) The `optimisation` keyword is used to specify an optimisation algorithm to use when solving the search problem. The tool supports a number of custom parameters for the termination condition and for the chosen algorithm. The user can specify a termination condition using the `termination` keyword. The supported parameters for algorithm termination are `time` which must be given in `seconds` or `evolutions`, as an integer.

A complete specification must be written in a file with the `mopt` extension. The mopt files can be executed from inside Eclipse or in standalone mode. In Fig. 5 we include the instructions printed by the tool when running it in standalone mode. This feature is useful when running experiments on headless servers.

Using the inputs provided by the user, the tool initialises the selected evolutionary algorithm and starts the search. The steps performed by the tool to run an evolutionary algorithm are the following:

(1) An initial population is generated using the given input model, by making a copy for each population member and running a single random mutation operator;
(2) The selected optimisation algorithm is initialised;
(3) Until the termination condition is reached, an algorithm step is executed. For evolutionary algorithms, this step consists of applying a single mutation to each of the population individuals and then comparing the resulting offspring against the algorithm archive, using the algorithm dominance comparator.
(4) As soon as the termination condition is reached, the tool will save the serialized solution models together with detailed experiment information details.

[4] http://moeaframework.org/

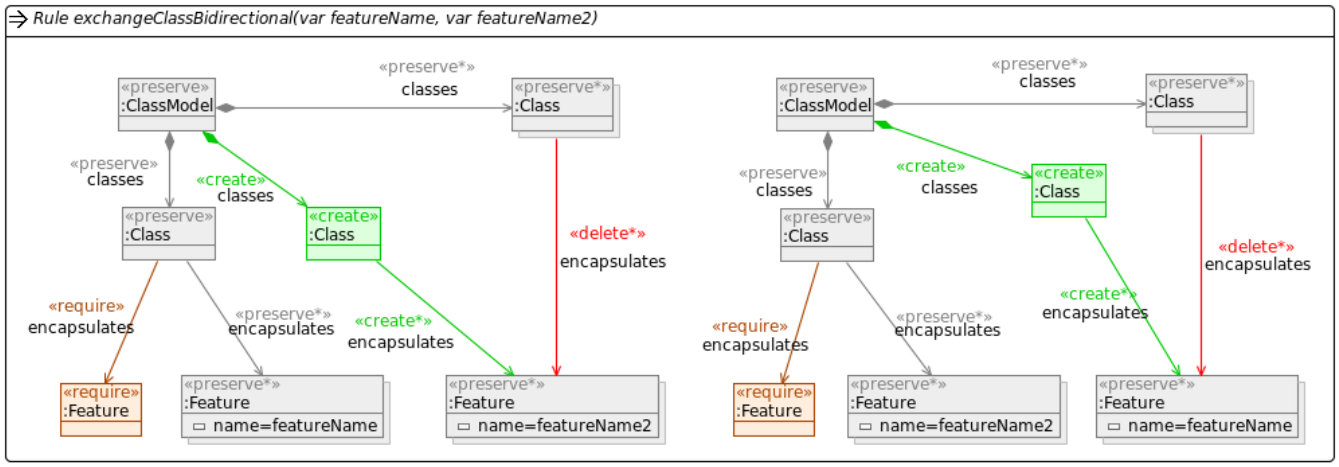

Figure 3: Breeding Search Operator for the CRA case encoded as a Henshin transformation rule

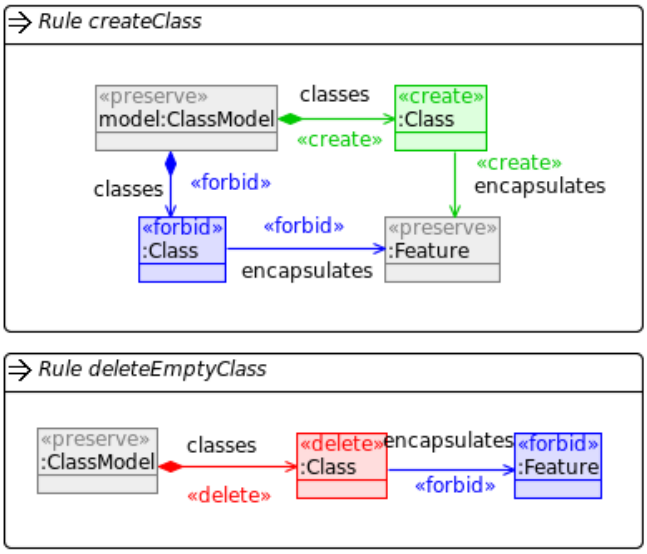

Figure 4: Mutation Search Operators for the CRA class encoded as Henshin transformation rules. The createClass rule creates one class and assigns a feature that has not been previously assigned to a class. The deleteEmptyClass rule deletes a class that has no features assigned.

```
usage: mdeo [-b <arg>] -m <arg> -p <arg> [-s]
Run MOPT specs from the command line.

 -b,--batch <arg>          run a batch with this id
 -m,--moptPath <arg>       use given MOPT configuration file
 -p,--projectPath <arg>    use given path as the root of the tool
 -s,--classic-matching     use classic matching strategy in Henshin
```

Figure 5: MDEO standalone mode. This figure includes the printed help menu displayed when running the tool outside of Eclipse without any parameters.

6 EXAMPLE RUN

In this section we describe an example run of MDEO for the two case studies described in Sect. 3. We have selected these two case studies for this demonstration to show how our tool runs for case studies with both a single objective and multiple objectives. A video showing the steps described in this section can be found on vimeo.com [5]. Because the video is time limited we focused on showing the tool functionality, and not on finding good results for the two case studies used in this demonstration. For this reason we limited both example runs to 30 seconds, using a time based termination condition.

We start the video by showing the `mopt` specification file for the CRA case study. Then, we show the Ecore metamodel file, mutation operators and the CRA fitness function implemented as an XTend class. For this case study we randomly picked one of the five input models that were proposed in the case study description. The search is started by using a `Run Configuration` in Eclipse and the search outcome is stored in the `mdeo-results` folder.

In the second part of the video we show the `mopt` specification file for the Scrum Planning case study. We also show the Ecore metamodel file, mutation operators and one of the objective functions, implemented also as an XTend function. For this case study we picked a model that represents a scenario with 5 stakeholders and 119 WorkItems that need to be organised into Sprints. The search is also started by using a `Run Configuration`. After the termination condition is reached, the search outcome is stored in the `mdeo-results` folder.

7 RELATED WORK

Over the years, there have been a number of tools proposed to solve optimisation problems using MDE [3]. To make a comparison with MDEO, a classification of these tools can be made based on the type solution encoding used. In the remainder of this section, we will describe the two main categories of tools using this classification.

[5] https://vimeo.com/281965931

7.1 Model Based Optimisation

Williams [14] proposed Crepe and MBMS (model-based metaheuristic search framework), a tool that uses a generic encoding for models to run meta-heuristic optimisation on models. The encoding used is a sequence of integers. The tool is further extended by Efstathiou et al. [6] to support multi-objective optimisation. Crepe uses generic search operators, applied on the integer sequence encoding, and provides a transformation system between the encoding and the corresponding models.

Unlike Crepe, the model based encoding used by MDEO, does not require the extra genotype to phenotype transformation each time the fitness function is evaluated. In Crepe, the application of mutation operators on the integer vector can also result in the generation of invalid solutions. For these the tool runs an additional repair step. Mandow et al. proposed a solution to this problem [10].

Strüber proposed FitnessStudio, a model optimisation tool that uses models as encoding for solutions [13]. The tool uses a meta-learning algorithm to find a suitable set of transformations that can be used as search operators. Then, the generated search operators are used to optimise models that conform to the same metamodel as the training model.

The main drawback of FitnessStudio, is that the user is required to first run a training step to find suitable search operators. The quality of these operators is also dependent on the metamodel coverage of the model used for training. For the CRA case study, Fitness Studio, found better overall solutions than MDEO. We are currently investigating these results to see what contributed to the quality of the solutions found by Ftiness Studio, compared to MDEO, for the same case study.

7.2 Rule Based Optimisation

Tools in this category use a sequence of model transformations as the encoding of solution candidates. The tools apply search operators such as mutation and crossover to a transformation chain. The resulting sequence of transformations is applied to the initial model. The quality of the output model is evaluated using fitness functions. The application of search operators to a transformation chain can lead to the generation of invalid solutions. To solve this issue, the tools use an additional step to repair the transformation chain to ensure that a feasible solution is generated.

Fleck et al. proposed Marrying Optimisation and Model Transformations (MOMoT), a tool that optimises model transformation chains [7]. MOMoT is implemented as an Eclipse plugin and allows users to specify optimisation problems using a flexible DSL. The model transformations used by MOMoT are encoded using Henshin. The tool uses mutation and crossover search operators, applied to the transformation chain. An additional repair step is required during the search to ensure that after the application of search operators, the transformation chains return a feasible solution. The optimisation algorithms used by MOMoT are implemented using MOEA Framework.

Abdeen et al. [1] proposed a multi-objective rule based design space exploration (DSE) framework built using the ViatraDSE framework [9]. The implementation uses the NSGA-II algorithm to find efficient model derivation chains. The framework uses mutation and crossover operators, and a repair mechanism is implemented to ensure that the resulting transformation chains produce feasible solutions. Infeasible transformation chains are truncated or discarded.

The main diference between MDEO and the MOMoT and ViatraDSE tools is the encoding used for the individual candidate search solutions. MDEO uses a model for this encoding, while MOMoT and ViatraDSE use a chain of transformations.

8 CONCLUSION

In this demonstration we have showed MDEOptimiser, a tool that runs optimisation on models. MDEO offers a DSL to simplify the process of specifying optimisation problems. The tool can be used as an Eclipse plugin or in standalone mode, as a `jar` binary. The tool uses evolutionary algorithms implemented using the MOEA Framework and the mutation operators are encoded as Henshin transformation rules.

In future work we plan to extend MDEOptimiser to support automatic algorithm selection, automatic initial model generation and to add support for automatic generation of more complex mutation operators. We are also planning to create a tool comparison, using multiple case studies, between MDEOptimiser and similar tools.

REFERENCES

[1] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. [n. d.]. Multi-objective Optimization in Rule-based Design Space Exploration. 289–300. https://doi.org/10.1145/2642937.2643005

[2] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. 2010. Henshin: advanced concepts and tools for in-place EMF model transformations. *Model Driven Engineering Languages and Systems* (2010), 121–135.

[3] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer. 2017. A survey on search-based model-driven engineering. *Automated Software Engineering* 24, 2 (2017), 233–294. https://doi.org/10.1007/s10515-017-0215-4

[4] Frank R Burton and Simon Poulding. 2013. Complementing metaheuristic search with higher abstraction techniques. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*. IEEE Press, 45–48.

[5] Alberto Rodrigues da Silva. 2015. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures* 43 (2015), 139–155.

[6] Dionysios Efstathiou, James R. Williams, and Steffen Zschaler. 2014. Crepe Complete: Multi-objective optimisation for your models. In *Proc. 1st Int'l Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA'14)*.

[7] Martin Fleck, Javier Troya, and Manuel Wimmer. 2015. Marrying search-based optimization and model transformation technology. *Proc. of NasBASE* (2015).

[8] Martin Fleck, Javier Troya, and Manuel Wimmer. 2016. The Class Responsibility Assignment Case. In *Proceedings of the 9th Transformation Tool Contest @STAF (CEUR Workshop Proceedings)*, Vol. 1758. 1–8. http://ceur-ws.org/Vol-1758/paper1.pdf

[9] Ábel Hegedüs, Ákos Horváth, István Ráth, and Dániel Varró. [n. d.]. A Model-driven Framework for Guided Design Space Exploration. 173–182. https://doi.org/10.1109/ASE.2011.6100051

[10] Lorenzo Mandow, Jose Antonio Montenegro, and Steffen Zschaler. 2016. Mejora de una representación genética genérica para modelos. In *Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2016)*. in press.

[11] Tom Mens and Pieter Van Gorp. 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 152 (2006), 125–142.

[12] Kenneth S Rubin. 2012. *Essential Scrum*. Addison-Wesley.

[13] Daniel Strüber. 2017. *Generating Efficient Mutation Operators for Search-Based Model-Driven Engineering*. Springer International Publishing, Cham, 121–137. https://doi.org/10.1007/978-3-319-61473-1_9

[14] James R. Williams. 2013. *A novel representation for search-based model-driven engineering*. Ph.D. Dissertation. University of York, UK. http://etheses.whiterose.ac.uk/5155/