Automatic generation of evolution rules for model-driven optimisation

Alexandru Burdusel and Steffen Zschaler

Department of Informatics, King's College London, London, UK, WC2R 2LS
`alexandru.burdusel@kcl.ac.uk, szschaler@acm.org`

Abstract. Over recent years, optimisation and evolutionary search have seen substantial interest in the MDE research community. Many of these techniques require the specification of an optimisation problem to include a set of model transformations for deriving new solution candidates from existing ones. For some problems—for example, planning problems, where the domain only allows specific actions to be taken—this is an appropriate form of problem specification. However, for many optimisation problems there is no such domain constraint. In these cases providing the transformation rules over-specifies the problem. The choice of rules has a substantial impact on the efficiency of the search, and may even cause the search to get stuck in local optima.

In this paper, we propose a new approach to specifying optimisation problems in an MDE context *without the need to explicitly specify evolution rules*. Instead, we demonstrate how these rules can be automatically generated from a problem description that consists of a meta-model for problems and candidate solutions, a list of meta-classes, instances of which describe potential solutions, a set of additional multiplicity constraints to be satisfied by candidate solutions, and a number of objective functions. We show that rules generated in this way lead to optimisation runs that are at least as efficient as those using hand-written rules.

1 Introduction

There has been a good deal of interest in optimisation of models in recent years [1,2,3,4,5,6,7,8]. These approaches aim to provide support for search-based software engineering [9] in an MDE context. Many of these approaches focus on using evolutionary techniques for finding models that optimise some objective function(s)—for example an OCL query or a simulation-based evaluation. To guide the exploration of the search space, a user has to provide a set of model transformations, which can create new candidate solution models from existing ones. Overall, the optimisation problem is thus specified by providing (1) a meta-model, instances of which are candidate solutions, (2) a set of objective functions, (3) additional constraints to be satisfied by valid solutions, and (4) a set of transformations to evolve models.

In some cases, these transformations are an inherent part of the optimisation problem. For example, when using evolutionary search to find an optimal refactoring of model transformations [10], or when finding optimal reconfigurations of a cloud data centre [11], it is important to ensure that any solutions have been derived from the starting point only through the application of rules from a pre-defined set. In the latter case, we may even have an objective function based on the number of transformation steps that have been applied. However, in many other scenarios specifying the transformation rules as part of the optimisation problem is less natural and leads to over-specification. For example, the well-known class–responsibility assignment (CRA) problem, which was also a problem case at the 2016 Transformation Tool Contest (TTC) [12], simply looks for an optimal allocation of features to classes. How the search algorithm arrives at this allocation is not a natural part of the problem. In fact, when solving this problem in an evolutionary manner, there are different sets of evolution rules that might potentially be applied, and different sets of rules will lead to results of different optimality. Requiring users to specify the rules with the problem, then, forces them to over-specify and risks missing the best solutions.

In this paper, we show how optimisation problems over models can be specified *without the need to specify a set of evolution transformations as well.* We show how a set of rules can be automatically generated from a meta-model, objective functions, and a set of additional constraints. The rule generation algorithm presented in this paper uses an extended variant of the SERGe (SiDiff Edit Rule Generator) algorithm presented in [13]. We demonstrate, using the CRA case study, that we are able to generate rules that enable efficient optimisation runs leading to good results.

The remainder of this paper is structured as follows: Section 2 gives a brief description of related work in model optimisation. Then, in Sect. 3, we describe the case study used throughout the paper, followed by Sect. 4, where we present our solution. In Sect. 5, we present an evaluation, including a comparison to the VIATRA-DSE solution to the TTC '16 CRA case [14]. Finally, in Sect. 6, we discuss lessons learned and highlight future research.

2 Related Work

We have introduced MDEOptimiser (MDEO) previously in [2,15]. MDEO performs model-based optimisation by running evolutionary optimisation with candidate solutions represented by model instances of a given meta-model. Evolution steps are obtained by applying endogenous model transformations using Henshin transformation rules [16]. Since our submission to TTC 2016, the tool has been improved, the evaluation in Sect. 5 is be based on the most recent version of the tool. A description of the improvements is included in Sect. 4.

In [5] the authors introduce the MOMoT (Marrying Optimisation and Model Transformations) tool. The tool is built in the context of Eclipse Modelling Framework (EMF)[1] and it uses Henshin transformation rules to generate optimisation solutions. The tool uses the MOEA framework[2] for the implementation of the search algorithms. Alongside the MOEA framework algorithms, MOMoT

[1] `https://eclipse.org/modeling/emf/`
[2] `http://moeaframework.org/`

also supports single-objective and local search optimisation algorithms. It defines a custom DSL for problem descriptions, consisting of a meta-model, a set of Henshin transformation rules, a set of objectives and constraints specified either as Java or OCL implementations and the search algorithm to be used. The output produced consists of a set of analysis artifacts, the resulting models, found objective values and a chain of rule applications used to obtain the solution models. The MOMoT framework is very similar to MDEO, the main difference being that MDEO runs the optimisation directly on models rather than on sequences of rule applications from which models can be generated.

VIATRA-DSE [1] is another tool performing optimisation on models. It uses the VIATRA2 [17] model transformation framework which is built on the EMF. In order to run model optimisation, the tool requires as input an initial model, a set of transformation rules, a set of constraints and a set of objectives. For search space exploration the tool supports several algorithms such as Hill Climbing and Non-dominated Sorting Genetic Algorithm (NSGA-II)[18]. This tool, similarly to MOMoT requires the user to specify the transformation rules as part of the optimisation problem specification.

In [4], the authors propose another model optimisation tool. Crepe Complète is an extension of Crepe [19], which allows multi-objective optimisation of models. It has been developed as an improvement of the Crepe tool which only supported single objective optimisation. Crepe Complète is built on top of the Epsilon Object Language (EOL) and can run multi-objective optimisation on models. Crepe Complète can run optimisation on any problem that can be encoded in a meta-model. The tool supports generic search operators and a generic encoding of models using integer vectors. However, optimisation performance can quickly become sub-optimal as the encoding is non-locality-preserving [20].

There remains a clear gap for approaches that run evolutionary search over models but do not require manual definition of evolution rules as part of the problem specification. In this paper, we propose a first such approach.

3 Running Example

Throughout this paper, we will use a running example to help explain and evaluate our approach. For this, we are reusing the well-studied Class–Responsibility Assignment (CRA) problem, in the form introduced as a challenge case at the 2016 Transformation Tool Contest [12].[3] The goal in this problem is to find an optimal set of classes and class–feature allocations that minimise coupling and maximise coherence.

More specifically, a CRA problem is an instance of the CRA meta-model in Fig. 1, without any instances of `Class`. A valid solution is an instance of the same meta-model with a number of `Class` instances and *all* `Feature` instances allocated to a class via association `isEncapsulatedBy`. Note that this means

[3] This problem case also required all classes to have unique names. Given that this can be achieved by a simple post-processing step [15], we ignore the requirement for this paper.

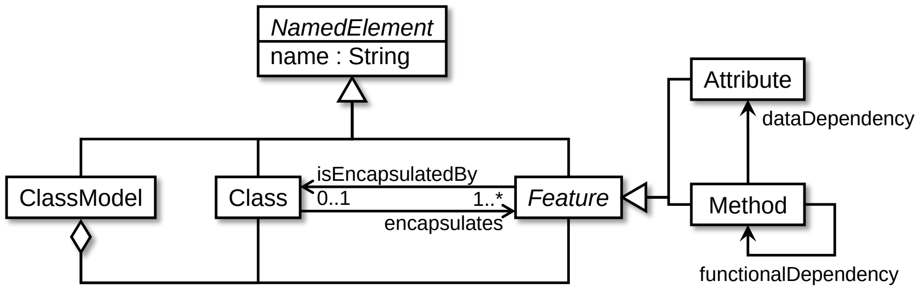


Fig. 1: Problem meta-model for the CRA problem

that for solutions we have a stricter multiplicity constraint than for problem descriptions; in particular, the lower bound multiplicity of `isEncapsulatedBy` is 1 rather than 0. `Feature`s can be either `Attribute`s or `Method`s. Different kinds of dependencies can be represented between features of different kinds. The goal is to allocate `Feature`s to (newly created) `Class`es to minimise dependencies between classes (coupling) and maximise dependencies within classes (coherence). This is specified as a single objective combining the coupling and coherence objectives into a single so-called CRA measure. Details of the definition of this measure can be found in the TTC case [12]. The case study description includes five input models which can be used to evaluate the proposed case solutions [12]. A summary of these input models, which vary in size and complexity, has been included in Table 1. In Sect. 5 we are discussing the results of our approach using each of these models as input.

Table 1: Summary of input models

	A	**B**	**C**	**D**	**E**
Attributes	5	10	20	40	80
Methods	4	8	15	40	80
Data Dep.	8	15	50	150	300
Functional Dep.	6	15	50	150	300

This is a common way in which search problems are phrased: an initial model (instance of the problem meta-model) is given to describe a specific search problem. Typically, the elements provided in this model are not meant to be changed as a result of the search. Some elements of the meta-model are not (or only partly) instantiated: these will be used to represent potential solutions. Because the problem meta-model, thus, needs to be a valid meta-model for both problem specifications and candidate solutions, some of its multiplicity constraints (namely those at the boundary between the elements responsible for problem descriptions and those responsible for solution descriptions) may need to be strengthened for valid solutions. This is similar to more traditional ways of specifying optimisation problems, where constraints are a standard part of a problem specification. In model-based optimisation, we can say that the *problem meta-model* is refined by the *solution meta-model* by providing stronger multiplicity bounds in some places. Thus, every candidate solution will also be a valid instance of the problem meta-model, while a problem specification will typically not yet be a valid instance of the solution meta-model. The task of the search algorithm, then, is to continuously modify the given problem specification until it satisfies all the additional multiplicity constraints of the solution meta-model and, then, to find an optimal solution model.

4 Searching Optimal Models with Generated Rules

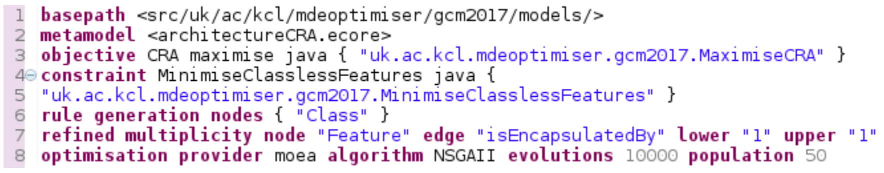
We have implemented our approach using our MDEO tool [2,15]. The tool is an Eclipse plugin allowing the user to specify model optimisation in the EMF context through a DSL. It uses Henshin-encoded endogenous model transformation rules as search mutation operators, to explore the search space. The optimisation algorithms are implemented using the MOEA framework. In earlier versions of the tool, the user was required to manually create the Henshin transformation rules and then specify them in the DSL configuration. In this paper we are using the SERGe [13] meta-tool to automatically generate the initial consistency preserving edit rules (CPERs). For each of the generated rules we then make a copy to which we apply a set of refinements to better guide the evolutionary process by ensuring that edit operations encoded in the rules can be applied to models conforming to both the problem and the solution meta-models.

The rest of this section is structured as follows: In Section 4.1 we describe how the optimisation problem for rule generation is specified in our DSL, then in Section 4.2 we describe our rule generation algorithm. Section 4.3 describes how we configured our tool to run the experiments for the CRA case.

4.1 Specifying the Optimisation Problem

The problem description required by our DSL consists of the following elements:

1. *A problem meta-model.* Specific search problems are given as instances of this meta-model. In the CRA case this is the meta-model shown in Fig. 1;
2. *Objective functions.* These can be provided as Java implementation or as OCL queries and return a numerical value for a given model. In the context of the CRA case, we have only one objective, namely the CRA value which we are seeking to maximise during the search;
3. *A meta-model subgraph.* Only a subset of the elements from the problem meta-model represent solution information. Instances of these elements can be modified during the search, everything else should be kept constant as it represents problem context only. The *multiplicity refinements* provided next can only apply to elements in this sub-graph of the problem meta-model;
4. *Additional multiplicity constraints.* These constraints, which we also call *multiplicity refinements* are constraints that form the *solution meta-model*, a subset of the problem meta-model. These refinements must be satisfied by



```
basepath <src/uk/ac/kcl/mdeoptimiser/gcm2017/models/>
metamodel <architectureCRA.ecore>
objective CRA maximise java { "uk.ac.kcl.mdeoptimiser.gcm2017.MaximiseCRA" }
constraint MinimiseClasslessFeatures java {
"uk.ac.kcl.mdeoptimiser.gcm2017.MinimiseClasslessFeatures" }
rule generation nodes { "Class" }
refined multiplicity node "Feature" edge "isEncapsulatedBy" lower "1" upper "1"
optimisation provider moea algorithm NSGAII evolutions 10000 population 50
```

Fig. 2: MDEO CRA problem specification with automatic rule generation

all valid solution candidates. In the CRA case, as described in Sect. 3, the requirement is that there are no features which are not encapsulated in a class, so a refinement is to restrict the multiplicity of the isEncapsulatedBy edge from [0..1] to [1..1]. These multiplicity constraints must refine those in the original problem meta-model;

5. *Constraint functions.* These represent the additional multiplicity constraints in a form that can be used by a search algorithm (*i.e.,* a function that must be zero for valid candidate solutions). In principle, these could be generated from the additional multiplicity constraints, but our prototype currently does not support this;
6. *An optimisation algorithm.* This specifies the algorithm provider[4], along with the search algorithm to use and the necessary evolutions and population configuration. In this paper we are only using the NSGA-II algorithm with multiple configurations for the evolutions and populations variables. It is beyond the scope of this paper to present a comprehensive comparison between multiple algorithms.

An example of the CRA problem specification using the MDEO DSL can be seen in Fig. 2. The configuration keywords in the DSL are intuitive. The **basepath** element is required, can only be used once and it defines the Eclipse resource set working path, then the **meta-model** element is also required, can be only one and it's used for specifying this optimisation problem meta-model. The next element is the **objective**, which is required and can be used multiple times. This is used to specify the optimisation objectives which can be loaded from Java files or specified as OCL queries. The next keyword is **constraint**, it's optional and it defines the constraints to be used in the optimisation process. Then the **rule generation node** keywords allow the user to specify the nodes for which Henshin transformation rules are going to be generated. Finally the **optimisation** keyword is used to configure the search algorithm and its parameters.

4.2 Generating the Rules

In this section, we discuss how we generate the evolution transformation rules from the information provided above.

[4] As registered in the underlying instance of the MOEA Framework

Previous work on automatic generation of transformation rules from meta-model information has been reported in [13]. SERGe is a meta-tool which generates a complete set of complete and consistency-preserving edit operations (CPEOs) for a given meta-model. The tool has been developed in the EMF context and the generated transformation rules are encoded in Henshin. The algorithm implemented in SERGe is designed to ensure that for any rules it generates, any change between two models is always consistency preserving with regards to the meta-model. The SERGe tool, by default, provides an extensive set of configuration options. A complete description of the rule generation process supported is described in [21]. However for the purpose of proving our approach with the CRA case we have restricted the generation of rules to only a subset of all the possible operations. A complete list of the SERGe rules generated for our case can be seen in Table 2. To generate the rules, SERGe performs the following steps:

1. *Create node.* For each non-root type `A` with mandatory neigbours and no children, SERGe will generate a create node rule. The rule also connects the node `A` to its mandatory children, if any are present in the metamodel, by creating a containment edge. The node is connected to its mandatory neighbours `B` by creating edges of type `a`. If edge `a` has an opposite edge of type `b` then this will also be created. If there is an upper bound multiplicity in the meta-model for edge `b`, then a Negative Application Condition (NAC) will be generated to ensure that the multiplicity is respected when connecting nodes of type `B` to nodes of type `A`;
2. *Delete node.* The node delete rules are created as inverses of the Create rules, by swapping the left and right-hand side of the Henshin rule graph. The generated rule will delete the node `A` and all its mandatory children. The node is also disconnected from its mandatory neighbours. If multiplicity constraints are present, then a Positive Application Condition (PAC) will be generated to ensure that node `A` can be deleted safely without invalidating the meta-model lower bound multiplicities between the deleted node and its neighbours;
3. *Add node edge.* The add edge rule is generated for each reference `a` or opposing reference `b` of a node `A`. If the reference is not a containment and if the lower bound multiplicity is not equal to the upper bound multiplicity then, a rule is generated to add an edge a between type `A` and the type at the opposing end `B`. If edge a has an opposite edge of type `b` then this will also be created. If there is an upper bound multiplicity in the meta-model for edge `b`, then a NAC will also be generated to ensure that the multiplicity is respected when connecting nodes of type `B` to nodes of type `A`;
4. *Remove node edge.* Similar to the delete node rule the remove node edge rule is generated by swapping the left-hand side with the right-hand side of the add edge rule. In the case of remove edge rules, NAC applications are not required, but if multiplicity constraints are present, then a PAC will be generated to ensure that the edge can be deleted safely without invalidating the model;

5. *Change node edge.* This rule type is a simplified version of the combined application chain of a *Remove node edge* and *Create node edge* rules, performing the individual steps of each of these rules in a single application. This rule is generated by SERGe when ran with a meta-model that has a fixed multiplicity between two nodes. The generated rule is the same as the *Refined Remove Edge Rule* in Table 2.

When using the transformation rules generated by SERGe for the problem meta-model, the optimisation process has a tendency to get stuck in local optima. This is because the SERGe rules are generated so that the produced models are consistent w.r.t to the problem meta-model only. The solution meta-model is a subset of the problem meta-model, as a result of the refined multiplicity constraints applied to the problem meta-model. It is still possible for solution meta-model instances to be discovered using the rules generated by SERGe for the problem meta-model, however when using these rules to transform instances of the solution meta-model, the validity of the resulting model instances cannot be ensured. The validity depends on which part of the problem meta-model the transformation output model conforms to, the valid solution meta-model subset which satisfies the problem constraints or the rest of the problem meta-model which includes all possible solutions, both valid and invalid w.r.t. the problem constraints. During the optimisation process, if a valid solution becomes invalid because of a constraint invalidation, it automatically becomes infeasible and it is dominated by other valid solutions [18,22]. This happens even if a subsequent transformation on the same solution would make it dominant. A solution is dominant if it is feasible with regards to its constraints and it is at least as good for all objective values as the other solutions and better for at least one objective value [22].

Performing a CRA case optimisation run using these transformation rules generated for the problem meta-model and using a constraint that invalidates solutions with unassigned `Features`, is not sufficient to obtain the best possible results. The evolution gets stuck in local optima after all the `Features` are assigned to a `Class` and the model becomes consistent w.r.t the solution meta-model. Then, the only way to move a `Feature` from a `Class` to another `Class`, is to remove the `isEncapsulatedBy` edge from a `Class` using rule *REMOVE_Class_(encapsulates)_TGT_Feature* and then add it again for another `Class` using rule *ADD_Class_(encapsulates)_TGT_Feature*. While this is done, the specified constraint is invalidated, creating a solution which has one unassigned `Feature`. This constraint violation causes the solution to become infeasible and it becomes dominated by the other solutions which do not have a good CRA value to that point, but are feasible because they don't invalidate the constraint to have no unassigned `Features`. As a result, the new candidate is removed from the population and never explored further. We could try to fix this by encoding constraints as objectives instead. However, while this would allow the search to escape local optima, by not having any solutions considered invalid, it would not guarantee all resulting search solutions to be valid.

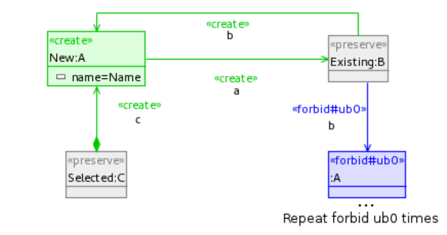
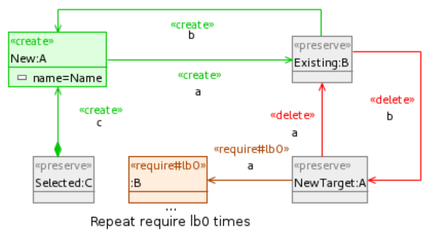
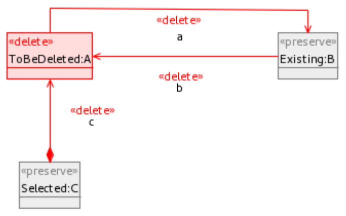
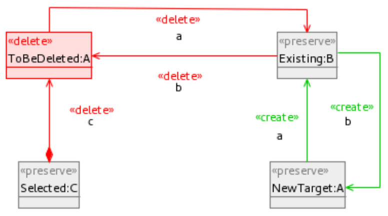
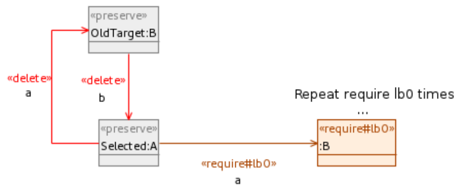
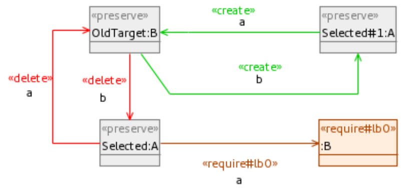
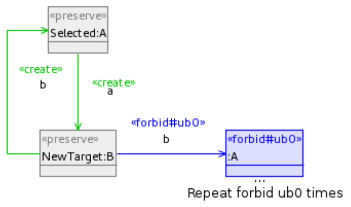
SERGe Create Rule	Refined Create Rule
 «create» New:A name=Name «preserve» Existing:B «preserve» Selected:C «forbid#ub0» :A «create» b «create» a «create» c «forbid#ub0» b Repeat forbid ub0 times	 «create» New:A name=Name «preserve» Existing:B «preserve» Selected:C «require#lb0» :B «preserve» NewTarget:A «create» b «create» a «create» c «delete» a «delete» b «require#lb0» a Repeat require lb0 times
SERGe Delete Rule	**Refined Delete Rule**
 «delete» ToBeDeleted:A «preserve» Existing:B «preserve» Selected:C «delete» a «delete» b «delete» c	 «delete» ToBeDeleted:A «preserve» Existing:B «preserve» Selected:C «preserve» NewTarget:A «delete» a «delete» b «delete» c «create» a «create» b
SERGe Remove Edge Rule	**Refined Remove Edge Rule**
 «preserve» OldTarget:B «preserve» Selected:A «require#lb0» :B «delete» a «delete» b «require#lb0» a Repeat require lb0 times	 «preserve» OldTarget:B «preserve» Selected#1:A «preserve» Selected:A «require#lb0» :B «create» a «create» b «delete» a «delete» b «require#lb0» a
SERGe Add Edge Rule	
 «preserve» Selected:A «preserve» NewTarget:B «forbid#ub0» :A «create» b «create» a «forbid#ub0» b Repeat forbid ub0 times	No refinements necessary for this rule.

Table 2: Generated SERGe and refined rules

Generally, the problem here is that we are running SERGe with the problem meta-model and that the solution meta-model introduces additional multiplicity constraints. These are not taken into account by the rules generated. Running

SERGe with the solution meta-model does offer a solution to the problem: for the case of a *[1..1]* multiplicity constraint (as between `Feature` and `Class` in the CRA case), with the right configuration settings, SERGe can generate a *change edge* rule. In addition to this rule, two other rules are generated: add an unassigned Feature to an existing `Class` and create a `Class` and assign an unassigned `Feature` to it. The problem with these rules however, is that the search space cannot be fully explored once all `Features` have been assigned to a `Class`. After this happens, the only possible operation is to apply the *change Feature* rule to the search models and move `Features` between classes, but the transformations cannot perform any create and delete `Class` solution model changes. This limitation of rule applications on valid solutions leads to an incomplete search space exploration.

What we need, is an algorithm that generates transformation rules that are applicable to an instance model of the problem meta-model, allowing all types of transformations for the nodes we are interested in, but that ensure that any model edit operations will not introduce additional invalidations of the solution meta-model constraints. The generated rules must be able to perform the same edit operations on models conforming with both the problem meta-model and the solution meta-model. We ensure this by post-processing the rules produced by SERGe for the problem meta-model.

We have adapted the SERGe meta-tool by applying a set of refinements to the generated rules to ensure that the search process does not get stuck in local optima due to constraint invalidation. Our refinements, are additions to copies of generated SERGe rules, to ensure that when an instance of the solution meta-model is found, the search process can still evolve it by applying CPEOs to it, without breaking the constraints defined in the solution meta-model.

In our approach, we have implemented refinements aimed at solving the CRA case, therefore the list presented in this paper is not exhaustive and we aim to implement the remaining refinements in future work.

The overall rule-refinement process is the following:

1. For all nodes with given multiplicity-constraint refinements, check the validity of the refinements. In this step we ensure that the given refinement constraints are valid w.r.t. the problem meta-model, by ensuring that they specify a solution meta-model which is a subset of the problem meta-model;
2. Generate a new meta-model including only the upper-bound refinements. This meta-model is then used in the following step to generate the SERGe rules. Note that SERGe already handles upper-bound refinements the way we need them. Lower-bound refinements require post-processing of rules;
3. Run the SERGe meta-tool with the new meta-model and generate rules for the nodes specified in the problem specification. In this step, we run the SERGe algorithm with the meta-model having the specified upper-bound refinements set. This generates the rules as seen in the SERGe rules column in Table 2;
4. Create a copy of each of the generated rules and apply the following refinements to them, each refinement resulting in a new rule. For each of the

refinements described in the following list, a before and after comparison can be found in Table 2:

(a) If the rule is creating a new node type `A` and there is a lower bound refinement of an edge `a` or `b` at either side, then find another existing node of type `A` and for each created edge between the new node and the existing mandatory neighbours, add a delete edge between the existing node type `A` and the existing mandatory neighbours `B`. These refinements allow the rule to create a new node when there are no mandatory neighbours available to be assigned, by taking one from an existing node of the same type;

(b) If the rule is deleting a node type `A` and there is a lower-bound refinement of an edge `a` or `b` at either side of them, then find another existing node `A` and for each deleted edge between the deleted node `A` and the existing mandatory neighbours, add a create edge between the existing node `A` and the existing mandatory neighbours. The rule changes added by these refinements allow a node to be deleted and not leave mandatory neighbours dangling, by moving them to other existing nodes of the same type as the deleted node;

(c) If the rule is deleting an edge `a` between node type `A` and node type `B` and there is a lower bound refinement at either side of `A` or `B` then find another node of type `A` and create the deleted edge between `B` and the found node type `A`. This refinement results in a Change edge rule, which SERGe can also generate if configured to do so and the edge has a fixed multiplicity;

5. Remove duplicate rules by using the SERGe duplicate checker;

4.3 Running the Optimisation

Once the rule refinements are generated, MDEO groups the rules generated for the meta-model with the upper bound refinements and the new refined rules and runs the optimisation process with the complete set of generated rules. To run the optimisation for the CRA case we have implemented our proof of concept as a new feature of the MDEO tool. For this experiment we have created a standalone launcher for the tool to allow us to run the optimisation without having to run the tool as an Eclipse plugin for each of the configurations.

5 Evaluation

Our aim is to automatically generate evolvers from a metamodel so that we can then run MDEO to perform evolutionary optimisation on models without having to design the rules manually. We describe the ideal solution meta-model and which sections should be transformed and then the tool automatically generates the necessary transformation rules so that models can be evolved to become valid solution candidates. We have evaluated our solution starting from the following

Table 3: Summary of MDEO TTC '16 input models results

MDEO M I	**A**	**B**	**C**	**D**	**E**
Best CRA	3.0	2.999	2.015	N/A	N/A
Mean CRA	1.978	1.954	1.232	N/A	N/A
Mean Time	0m 0s 505ms	0m 1s 083ms	0m 2s 705ms	0m 8s 946ms	0m 18s 906ms
MDEO M II					
Best CRA	3.0	3.104	2.910	5.531	3.098
Mean CRA	1.950	1.911	1.972	4.103	0.816
Mean Time	0m 2s 464ms	0m 5s 337ms	0m 12s 293ms	0m 54s 193ms	3m 7s 864ms
MDEO R I					
Best CRA	3.0	3.166	1.858	N/A	N/A
Mean CRA	2.627	2.114	0.327	N/A	N/A
Mean Time	0m 1s 188ms	0m 1s 892ms	0m 3s 816ms	0m 9s 760ms	0m 18s 234ms
MDEO R II					
Best CRA	3.0	4.083	3.177	5.794	2.618
Mean CRA	2.478	2.424	2.033	3.703	-0.035
Mean Time	0m 5s 650ms	0m 10s 290ms	0m 18s 358ms	1m 05s 752ms	3m 13s 674ms
MDEO S I					
Best CRA	1.75	0.791	-0.930	-2.646	N/A
Mean CRA	0.654	-0.629	-4.207	-8.293	N/A
Mean Time	0m 0s 566ms	0m 1s 117ms	0m 2s 390ms	0m 6s 767ms	0m 13s 723ms
MDEO S II					
Best CRA	2.333	0.983	-0.601	-3.785	-4.855
Mean CRA	0.783	-0.523	-4.732	-7.647	-11.555
Mean Time	0m 2s 745ms	0m 5s 491ms	0m 13s 331ms	0m 44s 963ms	2m 21s 917ms
MDEO C I					
Best CRA	3.0	2.833	2.017	N/A	N/A
Mean CRA	1.936	1.964	0.908	N/A	N/A
Mean Time	0m 0s 958ms	0m 1s 448ms	0m 3s 290ms	0m 9s 385ms	0m 18s 256ms
MDEO C II					
Best CRA	3.0	3.104	3.634	6.436	3.011
Mean CRA	2.072	2.050	2.454	4.770	0.401
Mean Time	0m 3s 560ms	0m 7s 480ms	0m 19s 614ms	1m 12s 672ms	3m 57s 076ms

research question: Can we generate evolvers that perform optimisation as well as or better than the ones defined manually?

In this section we compare the results obtained with the latest version of MDEO running with manual user defined evolvers, the SERGe generated evolvers for both the problem and the solution meta-models and the automatically generated evolvers with our refinements. By doing this comparison we show that the automatically generated rules are just as good as the user defined rules. We also compare our CRA case results with VIATRA-DSE results from TTC '16 [14].

Experiment Setup. We ran our experiments for the CRA case using three Henshin transformation rules (evolvers) configurations:

MDEO Manual (MDEO M) Using the user defined evolvers, specified by us and previously used in the TTC 2016 Submission [15];

MDEO Refined (MDEO R) Using the evolvers automatically generated by the SERGe generated rules improvements described in this paper;

MDEO SERGe (MDEO S) Using the evolvers generated by SERGe without any of our refinements; and

MDEO SERGe Solution Metamodel (MDEO C) Using the evolvers generated by SERGe from the solution meta-model without any of our refinements.

For each evolver configuration we ran 30 experiments using the NSGA-II algorithm and the following parameters: **I** 100 evolutions and population size of 40; and **II** 500 evolutions and population size of 40. We have chosen the values for configuration **I** to have a comparison configuration with the solution proposed by VIATRA-DSE [14] for TTC 2016. The authors presented the results of 30 experiments running with a population of 40 and 100 evolutions.

The source code of the experiment together with the discovered solutions for all experiments can be found on GitHub[5]. All the experiments have been executed in headless mode on an AWS EC2 c4.large spot instances running Amazon Linux 4.4.2331.54.amzn1. x86_64 and Java 1.8.0_121 openjdk.

Results. In all configurations, computation time is partly given by the number of evolvers that have to be applied to a model in order to find mutation matches. Fewer evolvers require less computations to identify potential matches when evolving solutions. This execution time difference can be observed between the MDEO R configurations which has seven evolvers and the other configurations which have three(MDEO C) and four evolvers(MDEO M, MDEO S), respectively. The NSGA-II algorithm used for our experiments requires more computation time when there are less convergent dominant solutions than the expected population size and it has to spend more time on sorting though crowded solutions.

In Table 3 we can see that all configurations have been able to find the the same maximum CRA value for input model A, except for MDEO S. By inspecting the generated solutions and the average CRA value we can observe that MDEO R has found the highest overall values for model A, followed by MDEO C. The execution time is smaller for MDEO C than for MDEO R in both configurations. For input model B we can see that the best results are also obtained by MDEO R in both configurations. MDEO R also found the most good solutions overall, having the highest mean CRA value.

MDEO C found the best CRA value for input model C. Because the generated solutions are crowded and not diverse, the MDEO C **II** configuration takes more time than MDEO R and MDEO M to find the results, despite having only three evolvers compared to MDEO R which has seven.

[5] `https://github.com/mde-optimiser/gcm-2017-experiments`

Table 4: Summary of VIATRA-DSE TTC '16 input models results

	A	**B**	**C**	**D**	**E**
Best CRA	3	4	3.002	5.08	8.0811
Mean CRA	3	3.75	19992	2.8531	5.0188
Mean Time	0m 4s 729ms	0m 13s 891ms	0m 17s 707ms	1m 19s 136ms	9m 14s 769ms

For models D and E we note that configuration **I** does not actually find a solution. This is likely because not all features can be allocated to classes during the first 100 evolutions. For configuration **II** we can find valid solutions. For model D, MDEO R finds a better CRA than MDEO M but worse than MDEO C. For model E, we suspect that MDEO R needs even more evolutions to produce good CRA values due to the large number of evolvers. However, by comparing the average CRA we can observe that MDEO M found the best overall solutions and is closely followed by MDEO C and MDEO R.

For all the input models evaluated, the MDEO S configuration is getting stuck in local optima, because there are no rules to allow it to move a feature without invalidating a solution consistent with the solution meta-model. The best solutions it can find are given by the ones where all the features are assigned to classes the first time, when the solution becomes valid, during the evolution process. After this step, the solutions cannot generate better candidates through the mutations allowed by the generated evolvers for this configuration.

Comparing the MDEO R results with the VIATRA-DSE results for the CRA case included in Table 4, we can see that for configuration **I**, MDEO R found an equal CRA value for model A, but a worse mean CRA. For configuration **I**, MDEO R found worse CRA values for all other input models. For configuration **II**, MDEO R found again an equal CRA for model A and a better CRA value for all other models except E. The mean CRA values are worse for models A, B, and E and better for C and D. However it is worth noting that for configuration **II**, MDEO R ran for 500 evolutions compared to VIATRA-DSE which only ran for 100 evolutions. Also, the conditions under which the experiments for both solutions have been performed are very different, therefore a performance comparison of the two solutions is not possible.

Because we seek to apply optimisation directly on models through endogenous transformations, 100 evolutions is not enough to fully explore solutions which have close to or more than 100 features that have to be assigned. This can be observed in the results obtained by the MDEO R, which has seven evolvers compared to MDEO C which has only three or MDEO M which has four. However, given enough evolutions, the MDEO C and MDEO M are at a disadvantage when compared to MDEO R on the quality and diversity of explored solutions, because once all features are assigned to a class, no new classes can be created. This can lead to a limitation in search space exploration. This behaviour can be observed by analysing the mean CRA values of the smaller models (A-B) for configurations **I** and **II**.

In summary, we can say that the rule generation approach proposed in this paper produces rules that are comparable to manually written evolution rules. We can see that the obtained results for MDEO R are close to MDEO M or in some cases, better. The main drawback for the refined rules configuration is that the number of evolvers is larger than the ones manually defined, this ending up as requiring a longer time and more evolutions to find good solutions.

We have only experimented with the CRA case so far. We are aware that the presented approach may not be valid for other cases in its current form, but we are encouraged by the results obtained and we are planning to extend it to support other cases in future work.

6 Conclusions and Outlook

In this paper we have shown an approach to specifying optimisation problems in an MDE context without the need to explicitly specify evolution rules. We have shown an algorithm to generate the evolution rules from a problem specification consisting of a meta-model, a set of additional multiplicity constraints, a set of objectives and a list of meta-classes.

We have been encouraged by the results and we are planning to extend our tool so it can be used for additional types of constraints beyond multiplicity constraints. We also plan to test the approach with other case studies. One other improvement we are planning to implement is to support the generation of problem specifications for other tools requiring manual evolution rules such as MOMoT. Another improvement we are interested in adding to MDEOptimiser is support for a hyperheuristic algorithm to determine the best set of rule applications during an optimisation (e.g., using different rule sets during start up and during later stages of the search), so that we improve design space exploration in our optimisation process [23].

References

1. Hegedüs, Á., Horváth, Á., Ráth, I., Varró, D.: A model-driven framework for guided design space exploration. In: Proc 26th IEEE/ACM Int'l Conf. Automated Software Engineering (ASE'11). (November 2011) 173–182
2. Zschaler, S., Mandow, L.: Towards model-based optimisation: Using domain knowledge explicitly. In: Proc. Workshop on Model-Driven Engineering, Logic and Optimization (MELO'16). (2016)
3. Mészáros, T., Mezei, G., Levendovszky, T., Asztalos, M.: Manual and automated performance optimization of model transformation systems. International Journal on Software Tools for Technology Transfer **12**(3) (2010) 231–243
4. Efstathiou, D., Williams, J.R., Zschaler, S.: 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). (2014)
5. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proc. 1st North American Search Based Software Engineering Symposium (NasBASE'15). (2015) Preprint available at `http://martin-fleck.github.io/momot/downloads/NasBASE_MOMoT.pdf`.

6. Drago, M.L., Ghezzi, C., Mirandola, R.: A quality driven extension to the qvt-relations transformation language. Computer Science – Research and Development **30**(1) (2015) 1–20 First online: 24 November 2011.
7. Burton, F.R., Paige, R.F., Rose, L.M., Kolovos, D.S., Poulding, S., Smith, S.: Solving acquisition problems using model-driven engineering. In Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D., eds.: Proc. 8th European Conf. on Modelling Foundations and Applications (ECMFA'12). Volume 7349 of LNCS., Springer (2012) 428–443
8. Abdeen, H., Varró, D., Sahraoui, H., Nagy, A.S., Debreceni, C., Hegedüs, Á., Horváth, Á.: Multi-objective optimization in rule-based design space exploration. In Crnkovic, I., Chechik, M., Grünbacher, P., eds.: Proc. 29th ACM/IEEE Int'l Conf. Automated Software Engineering (ASE'14), ACM (2014) 289–300
9. Harman, M., Jones, B.F.: Search-based software engineering. Information and Software Technology **43**(14) (2001) 833–839
10. Fleck, M., Troya, J., Kessentini, M., Wimmer, M., Alkhazi, B.: Model transformation modularization as a many-objective optimization problem. IEEE Transactions on Software Engineering (99) (2017)
11. Chatziprimou, K., Lano, K., Zschaler, S.: Surrogate-assisted online optimisation of cloud IaaS configurations. In: IEEE 6th Int'l Conf. Cloud Computing Technology and Science (CloudCom). (2014) 138–145
12. Fleck, M., Troya, J., Wimmer, M.: The class responsibility assignment case. [24] 1–8
13. Kehrer, T., Taentzer, G., Rindt, M., Kelter, U.: Automatically deriving the specification of model editing operations from meta-models. In: International Conference on Theory and Practice of Model Transformations, Springer (2016) 173–188
14. Nagy, A.S., Szárnyas, G.: Class responsibility assignment case: a viatra-dse solution. [24] 39–344
15. Burdusel, A., Zschaler, S.: Model optimisation for feature class allocation using MDEOptimiser: A TTC 2016 submission. [24] 33–38
16. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place emf model transformations. Model Driven Engineering Languages and Systems (2010) 121–135
17. Eclipse.org: Viatra project Available at `http://eclipse.org/viatra/`.
18. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197
19. Williams, J.R.: A novel representation for search-based model-driven engineering. PhD thesis, University of York, UK (2013)
20. Mandow, L., Montenegro, J.A., Zschaler, S.: Mejora de una representación genética genérica para modelos. In: Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2016). (2016) in press.
21. Kehrer, T.: Calculation and propagation of model changes based on user-level edit operations. PhD thesis, University of Siegen (2015)
22. Deb, K.: Multi-objective genetic algorithms: Problem difficulties and construction of test problems. Evolutionary computation **7**(3) (1999) 205–230
23. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: International Conference on the Practice and Theory of Automated Timetabling, Springer (2000) 176–190
24. Garcia-Dominguez, A., Krikava, F., Rose, L.M., eds. In Garcia-Dominguez, A., Krikava, F., Rose, L.M., eds.: Proceedings of the 9th Transformation Tool Contest. Volume 1758., CEUR (2016)