Class Responsiblity Assignment Case: a VIATRA-DSE Solution*

András Szabolcs Nagy
nagya@mit.bme.hu

Gábor Szárnyas
szarnyas@mit.bme.hu

Budapest University of Technology and Economics
Department of Measurement and Information Systems
MTA-BME Lendület Research Group on Cyber-Physical Systems

Abstract

This paper presents a solution for the Class Responsibility Assignment Case of the 2016 Transformation Tool Contest. The task is to assign features (methods and attributes with dependencies to each other) to classes and optimize a software metric called CRA-Index. The solution utilizes the VIATRA-DSE framework's Non-dominated Sorting Genetic Algorithm (NSGA-II) and it extends the framework with a domain-specific state encoder to identify similar solutions to reach better performance. Furthermore, it also uses a domain-specific mutation operator and a slightly modified version of the provided transformation rule.

1 Introduction

Automated model transformations are a key factor in modern model-driven system engineering. Model transformations allow the users to query, derive and manipulate large industrial models, including models based on existing systems, e.g. source code models created with reverse engineering techniques. Since such transformations are frequently integrated to modeling environments, they need to feature both high performance and a concise programming interface to support software engineers.

*This work was partially supported by the MTA-BME Lendület Research Group on Cyber-Physical Systems.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

Design space exploration (DSE) aims to explore different design candidates with respect to well-formedness constraints and objectives to aid system engineers in finding the best design or to dynamically reconfigure a system at runtime. While DSE has a long history (20–30 years) [8], especially for embedded systems, it has been adapted to model-driven system engineering only in the recent years (discussed as related work in [1]).

VIATRA aims to provide the tooling support needed for these challenges by 1) an expressive model query language, 2) a carefully designed API for transformations and 3) a design space exploration tool easily integrated to the model-driven design process.

This paper presents a solution using VIATRA [2, 5] for the TTC 2016 Class Responsibility Assignment Case [7], which can be formalized as a DSE problem. The source code of the solution is available as an open-source project.[1] Additionally, there is a SHARE image available with the source code and scripts to run the solution on the provided input models.[2]

2 Case Description[3]

The problem to solve is a simplified version of the class responsibility assignment (CRA) problem. As an input model, a set of attributes and methods are given with dependencies between them, in particular, methods can use certain attributes and other methods. The task is to assign all these features to classes with the goal of optimizing a software metric called *CRA-Index*. The CRA-Index is an objective function to maximize and it combines the cohesion ratio (inner dependencies of a class divided by the cardinality of the features)

[1] `https://github.com/FTSRG/ttc16-cra-viatra-dse`

[2] `http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=ArchLinux64_TTC-Arch-CRA-VIATRA.vdi`

[3] The full description can be found here [7].

and coupling ratio (dependencies between two classes divided by the cardinality of the features) of the class diagram.

Contestants are given five models with increasing complexity to solve and they are to produce the corresponding high-quality models by using model transformation tools. The resulting models also have to satisfy the following constraints: 1) all features have to be assigned, 2) classes must have a unique name and 3) empty classes are not allowed.

3 Background of the Solution

VIATRA-DSE is a rule-based DSE framework [1, 9], which can explore different design candidates satisfying multiple criteria with respect to multiple objectives using graph transformation rules.

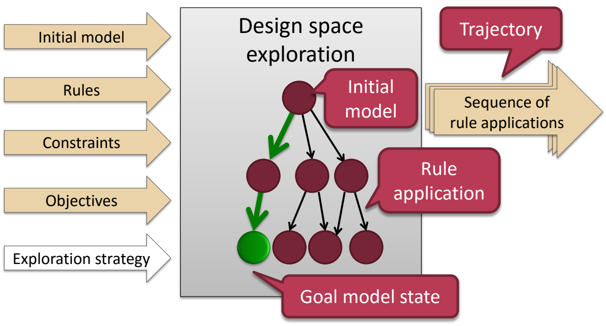

Figure 1: Overview of the rule-based DSE approach.

3.1 Approach of rule-based DSE

Figure 1 overviews the most important concepts of this approach from a perspective. A rule-based DSE problem consists of the following parameters:

- an *initial model* M_0,
- a set of *transformation rules* R that defines how the initial model can be manipulated,
- a set of *well-formedness constraints* C and
- a set of *objectives* O to optimize (minimize or maximize).

A solution for such a problem is a sequence of transformation rule applications (also called *trajectory*), which transforms the initial model M_0 to a model M_s, which satisfies all the well-formedness constraints in C. This solution is expected to be optimal (high-quality) with respect to objectives O.

The key strengths of this approach are that 1) models are attributed typed graphs and rules are graph transformation rules, which allows tight integration with model-driven system design, 2) the solution (i.e. the trajectory) also describes how to reach the found model from the initial state, which is important in certain problems, e.g. runtime reconfiguration of a system needs to answer how to reach the candidate configuration and 3) objectives can be derived from the model directly using even black box tools and from the trajectory as well.

To solve such a problem, a solver has to traverse a state space (also called design space) with an *exploration strategy*. This state space has an initial state representing the initial model and further states can be reached by applying the transformation rules. The state space can be infinitely large (e.g. a rule can make elements without an upper bound) and it can contain cycles (e.g. a rule can delete elements what an other rule just created). Figure 2 shows a partial design space of a small CRA problem, where there are three methods and a single attribute.

3.2 VIATRA

VIATRA is an open-source Eclipse project written in Java and Xtend [6] and it builds upon the Eclipse Modeling Framework [4]. The VIATRA project provides the following main features:

- A declarative language for writing queries over models, which are evaluated incrementally upon model changes (formerly known as EMF-INCQUERY).
- An internal DSL over the Xtend [6] language to specify both batch and event-driven, reactive transformations.
- A complex event-processing engine over EMF models to specify reactions upon detecting complex sequences of events.
- A rule-based design space exploration framework to explore design candidates as models satisfying multiple criteria. Presented in Section 3.3.
- A model obfuscator to remove sensitive information from a confidential model, e.g. for creating bug reports.

3.3 VIATRA-DSE

VIATRA-DSE provides an easy way to specify a rule-based DSE problem and to extend it with domain specific needs. The condition (left hand side) of the transformation rules can be specified by the VIATRA Query language and the operation (right hand side) by simple Java code. Both constraints and objectives can be specified either by the VIATRA Query language or by any custom Java code. Furthermore, it supports the calculation of objectives both on the actual model and on the trajectory as well (e.g. executing certain rules has a cost). VIATRA-DSE has several built-in strategies such as depth-first search, breadth-first search for systematic full exploration of the design space, fixed-priority search which uses pri-

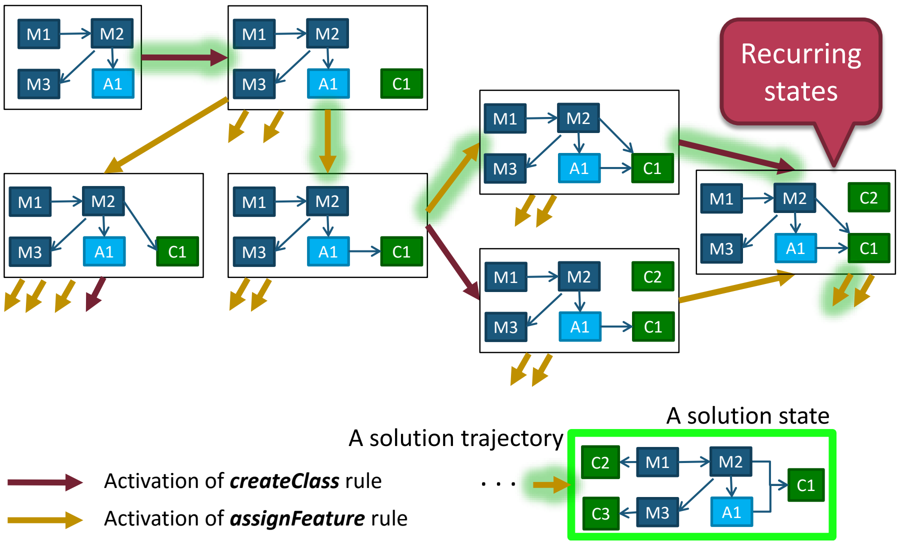

Figure 2: A part of the state space of a DSE problem and solution.

orities assigned to rules, and has metaheuristic strategies such as hill climbing and evolutionary algorithms, including Non-dominated Sorting Genetic Algorithm (NSGA-II) [3] and Pareto envelope-based Selection Algorithm (PESA) [10]. Custom, domain specific strategies can be integrated as well.

To recognize similar model states it uses a state encoding technique, which encodes model states into a textual representation. While these state codes can be easily compared to each other, the encoding process has a great impact on the exploration time. Additionally, rule applications are also encoded into a textual representation called activation codes. This allows to store a trajectory by its activation codes and to re-execute it later on an arbitrary model. VIATRA-DSE has a built-in generic state coder, which works perfectly for most of the time, but it can be exchanged with a custom, domain-specific state coder, which can improve the performance of the exploration.

The framework is also capable of parallel exploration.

4 Implementation

In this section, we provide a detailed description about how we instantiated the task as a rule-based DSE problem.

4.1 Transformation rules

Our solution uses the two transformation rules provided by the case, namely the *createClass* and *assignFeature* rules, however we enhance the *createClass* rule with the following two modifications:

1. A class can be created only if there are no empty class in the current model. This allows to prune the search space without losing any solutions.
2. Newly created classes given a name *CX*, where *X* is a number depending on how many classes were created on the actual trajectory. This ensures that the classes have a unique name.

4.2 Well-formedness constraints

While the modified *createClass* rule ensures the unique name of the classes, we used VIATRA Query to capture the other two constraints.

4.3 Objectives

We use two objective functions: one that calculates the CRA-Index of a class diagram and one that measures the violations of well-formedness constraints. The CRA-Index is calculated in the provided way, expect we use VIATRA Query to calculate *MAI* and *MMI* incrementally upon model change. The other fitness function measures the number of unassigned features and it shall be minimized. This helps the exploration to reach a solution more easily.

These two objectives creates a multi-objective optimization problem, which makes comparing solutions nontrivial. In this solution, we use the domination function [3] to compare solution candidates: a solution candidate s_1 dominates an other solution candidate s_2 if there is an objective function o_i that $o_i(s_1) > o_i(s_2)$ and for any other objective $o_j \neq o_i : o_j(s_1) \geq o_j(s_2)$. This approach will find a well-formed solution and an ill-formed solution candidate equal if the ill-formed solution candidate has a higher CRA-Index.

4.4 Exploration with NSGA-II

For the exploration strategy, we used the NSGA-II genetic algorithm [3] crafted for multi-objective optimization problems. This algorithm maintains a population, i.e. a set of solution candidates (trajectories), and modifies them with genetic operators (mutations and crossovers) to derive new solution candidates. In an iteration, it combines the previous population and a newly created population and selects the best solution candidates to produce the next generation. The adaption of this algorithm as a rule-based DSE strategy can be found in [1].

We configured the NSGA-II strategy in the following way:

- Population size: 40.
- The first population is generated by a breadth-first search algorithm selecting a trajectory into the population with a given probability and with a minimal length of 2.
- The following genetic operators are used:
  1. A mutation that adds a random rule application to the end of a trajectory.
  2. A mutation that modifies a random rule application in the trajectory.
  3. Cut and splice crossover, which exchanges the tails of two trajectories creating two child trajectories.
  4. Swap rule application crossover.
  5. A *custom domain-specific mutation operator* that removes all *createClass* rule application, where the created class remained unused. This helps the algorithm to find a well-formed solution.
- Mutations are used more frequently than crossovers: mutation rate is 0.8.
- The used stop condition consists of two sub-conditions that has to be fulfilled at the same time: 1) in the current population there is at least one solution that survived 100 iterations (i.e. the exploration cannot create a better solution) and 2) there is a well-formed solution in the current population.

4.5 State Encoding

The solution uses a custom domain-specific state coder as the built-in state coder failed to recognize similar solutions. For example, if there are two methods $M1$ and $M2$, which are assigned to two different classes $C1$ and $C2$, then the built-in state coder creates a state code $C1(M1), C2(M2)$ or $C1(M2), C2(M1)$ depending on the trajectory, which are eventually representing the same solution (the actual state code is much longer and redundant). Using this state coder, NSGA-II can store duplications in the population, which decreases efficiency. Thus, we created a domain-specific state coder that encodes model sates leaving out the identifiers of the classes: $(M1), (M2)$. This state coder also has better performance reducing the exploration time.

4.6 An Alternative Solution

We also experimented with an other approach, where first we created a class for each feature in the initial model using the VIATRA Model Transformation API. Then we ran the exploration with a single rule that merges two classes. While, this approach could generate good solutions with positive CRA-Index with approximately in the same time, the presented solution produces better results.

5 Evaluation

In this section, we evaluate the results of our approach.

5.1 Setup

As NSGA-II is a metaheuristic algorithm, it cannot provide a consistent solution for each run and runtime may vary because of the adaptive stop condition. Thus, we run the exploration 30 times for each input model and consider the median of found fitness values and the median of runtime as result. This allows to easily compare our results to other contestants' work.

The benchmarks were conducted on a 64-bit Arch Linux virtual machine running in SHARE. The machine utilized a single core of a 2.00 GHz Xeon E5-2650 CPU and 1 GB of RAM. We used OpenJDK 8 to run the VIATRA-DSE framework.

5.2 Results

Optimality: Figure 3 shows a box plot for the CRA-Index of the generated solution models for each input model. As it can be seen, the smallest input model is solved pretty consistently, with a CRA-Index of 3. While for input model B, most of the runs returns a solution with a CRA-Index around 3.75, for the more complex input models the result varies greatly. However, the found solution models always has a positive CRA-Index.

Performance: Figure 4 shows exploration times of the different runs in seconds on a logarithmic scale. The median values are marked with red. The runtime of the exploration greatly varies, especially on the largest input model E. It could find a solution in 20 minutes, while in the worst case it needed 74 minutes. An interesting property of the input model C that while it has twice as many features and thrice as many dependencies than input model B, the exploration time is just slightly longer.

	A	B	C	D	E
CRA-Index (best)	3	4	2.9090	4.3008	7.0337
CRA-Index (median)	3	3.7917	2.0736	3.3816	4.7654
Time (median)	00:21.091	00:56.191	01:07.211	05:22.987	36:42.535

Table 1: Results

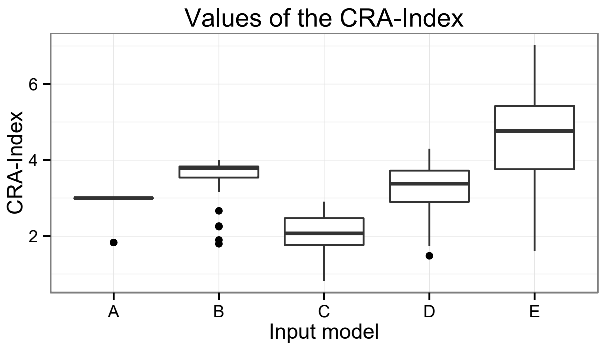

Figure 3: Values of the CRA-Index.

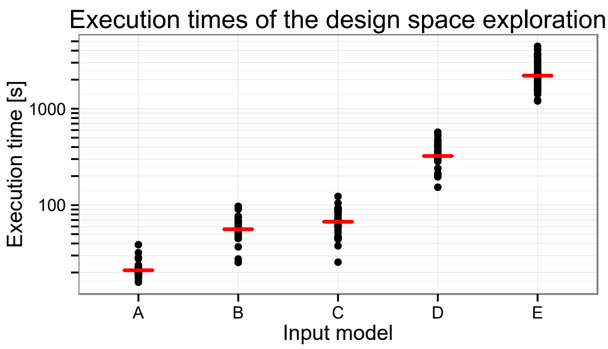

Figure 4: Execution times. The median values are marked with red.

Table 1 presents our aggregated results for each input model as stated in Section 5.1. We also included the metric of the best solutions our approach could find, to show that if execution time does not matter it can produce even better solutions.

6 Summary

This paper presented a complete solution for the Class Responsibility Assignment case of the 2016 Transformation Tool Contest. The approach of rule-based DSE and the VIATRA-DSE framework proved to be efficient for modeling the problem and sufficient for solving the case. The solution could be improved by gaining a deeper understanding of the CRA-Index metric and by adding a supplementary heuristic to the exploration.

References

[1] H. Abdeen, D. Varró, H. A. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth. Multi-objective optimization in rule-based design space exploration. In *ACM/IEEE Inter. Conf. on Autom. Soft. Eng.*, pages 289–300, 2014.

[2] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. Viatra 3 : A reactive model transformation platform. In *8th International Conference on Model Transformations*. Springer, 2015.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2):182–197, 2002.

[4] Eclipse.org. Eclipse Modelling Framework (EMF). `https://www.eclipse.org/emf/`.

[5] Eclipse.org. VIATRA Project. `https://www.eclipse.org/viatra/`.

[6] Eclipse.org. Xtend – Modernized Java. `https://www.eclipse.org/xtend/`.

[7] M. Fleck, J. Troya, and M. Wimmer. The class responsibility assignment case. In *9th Transformation Tool Contest (TTC 2016)*, 2016.

[8] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration*, 38(2):131–183, 2004.

[9] Á. Hegedüs, Á. Horváth, and D. Varró. A model-driven framework for guided design space exploration. *Autom. Softw. Eng.*, 22(3):399–436, 2015.

[10] J. D. Knowles, R. A. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization, First International Conference*, pages 269–283, 2001.