Effects of Graph Transformation Rules to Design Space Exploration Problems

András Szabolcs Nagy, Dániel Varró
Budapest University of Technology and Economics, Department of Measurement and Information Systems, Hungary
MTA-BME Lendület Research Group on Cyber-Physical Systems, Hungary
Email: {nagya, varro}@mit.bme.hu

***Abstract*—Design space exploration (DSE) aims to explore different design candidates that satisfies multiple criteria and is optimal with respect to different quality properties. The strength of rule-based DSE is that the exploration rules can be tailored to a specific problem, shaping the design space into a more concise form than traditional approaches. However, experts may have several choices to define the exploration rules and choosing the good one may increase exploration performance significantly. In this paper, we present several ways to define the exploration rules of a rule-based DSE problem and investigate the impact of these rules.**

I. Introduction

As a challenging branch of search based software engineering (SBSE), design space exploration (DSE) aims at searching through different design candidates to fulfill a set of constraints and then proposing optimal designs with respect to certain (multiple) objectives. It frequently supports activities like configuration design of avionics and automotive systems or dynamic reconfiguration of systems with high availability at runtime. Many of such traditional DSE problems can be solved by using advanced search and optimization algorithms or constraint satisfaction programming techniques [1], [2].

In model-driven engineering (MDE), rule-based DSE [3], [4], [5] aims to find instance models of a domain that are 1) reachable from an initial model by applying a sequence of exploration rules, while 2) constraints simultaneously include complex structural and numerical restrictions. A solution of such a problem is a sequence of rule applications which transforms the initial model to a desired model. Multi-objective rule-based DSE (MODSE) may also include multiple optimization objectives [4] which help to distinguish between solutions.

One of the major characteristics of rule-based DSE against traditional techniques is that the domain expert can define the atomic steps the exploration process can use to modify a candidate solution. These exploration steps have a high impact on the actual design space that an algorithm has to explore and thus it will affect the overall performance of the exploration. However, the exploration steps can be defined in multiple ways even for a relatively small problem and affecting the design space and thus the performance differently.

This paper is partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

The objective of this paper is to give an insight to newcomer DSE users through an example how the exploration rules can impact the performance of the exploration and to help decide on defining the exploration rules.

The paper is structured as follows: Sec. II briefly presents a motivating example and introduces the most important concepts of rule-based DSE, Sec. III provides an insight of the effects of transformation rules and Sec. V concludes the paper.

II. Rule-Based Design Space Exploration

Case study: The motivating example of this paper is the class responsibility assignment problem from the 9th Transformation Tool Contest (TTC16) [6].

The exploration task is taken from a reengineering challenge of object-oriented programs: create classes for a set of initially given methods and attributes (features) where methods can depend on other methods and attributes, in such a way that the resulting class model is optimal with respect to a software metric called CRA-Index (a metric derived from cohesion and coupling).

Besides the CRA-Index, there are two important constraints that the resulting class model has to satisfy: 1) there should be no empty classes in the resulting model and 2) all the features have to be assigned.

Domain model and instance model: Model-driven system design (MDSD) aims to lift the abstraction level of a problem allowing a better overview of it. For this purpose a **domain model** is created which describes the possible elements of the problem, their properties and their relations. For example the domain model of the CRA problem defines classes, features (methods and attributes) and relations between them. Domain models are also called **metamodels** as a metamodel describes the possible components of a semantic model, also called an **instance model**. In a metamodel *types* (or *classes*) describe objects from the domain which can have *attributes*, while *references* specify the relations between types.

Graph patterns and matches: A common task is to obtain data from *instance models* using queries. For this, **graph patterns** provide a good formalism which can be seen as a small, deficient instance model that we search for as part of the actual instance model. A graph pattern can capture elements, relations, negative or positive conditions on attributes and

multiplicity. A graph pattern can have multiple **match**es on an instance model similarly as a database query can return multiple rows.

Graph transformation rules: Modifications to an instance model are often described as **graph transformation rules**. A rule consists of a precondition or a left hand side (LHS), which is captured by a graph pattern and a right hand side (RHS), which declaratively defines the effects of the operation. A rule is applied on a model by 1) finding a match of graph pattern LHS (also called an **activation** of the rule), then 2) removing elements from the model which have an image in $LHS \setminus RHS$, then 3) changing the value of attributes which are reassigned in RHS and finally 4) creating new elements $LHS \setminus RHS$. A rule can have multiple activations or non at all as expected.

Rule-based design space exploration problem: The aim of rule-based DSE is to evolve a system model along transformation rules and constraints to find an appropriate system design. The state space of design candidates is potentially infinite but usually it also has a dense solution space.

The input of a **rule-based DSE problem** consists of three elements $RDSE = (M_0, G, R)$: 1) an initial model M_0, 2) a set G of goals given by graph patterns, which should be satisfied by the solution model M_{Si} and 3) a set R of transformation rules $(r_1, r_2, \dots r_r)$ which define how the initial model M_0 can be manipulated to reach a solution model M_{Si}. As a result it produces several solutions $(M_{S0}, M_{S1} \dots M_{Sn})$ satisfying all of the goals and each of them is described by a sequence of rule applications (or **trajectories**) $(r_{\vec{S}0}, r_{\vec{S}1} \dots r_{\vec{S}n})$ on the initial model M_0.

Furthermore, there are two optional inputs: **global constraints** GC and **objectives** O. Global constraints have to be satisfied on the model along each valid execution path (*i.e.,* trajectory) and are usually defined by graph patterns. An objective defines a function over either the model or the trajectory to derive a fitness value (*e.g.,* cost, response time, reliability) which can be used to distinguish between solutions in quality.

Design space: To solve an RDSE problem a search of the design space has to be conducted in accordance with an exploration strategy. The design space is a directed graph where the nodes represent the different states of the model and edges represent the rule applications (activations). There is one initial node that represents the state of the initial model and usually there are multiple goal states that satisfy the goal constraints. Depending on the RDSE problem, a model state can be reached in multiple trajectories (*e.g.,* there are independent rule activations) and the design space can have cycles in it (*e.g.,* one of the rule applications creates an element and the other deletes it).

III. Effects of Exploration Rules

In this section, we show six approaches to define the graph transformation rules for the CRA problem and discuss the properties of these approaches. We find this research relevant as it is a recurring problem in rule-based DSE to create new elements (classes) and to connect them (assign features to classes) and it is not trivial if either approach is better or worse than the other. With this research, we help newcomers to identify different approaches and to decide between them with the help of this evaluation.

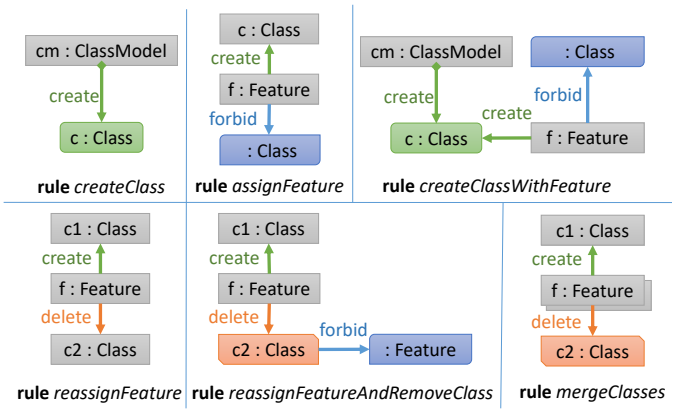


Fig. 1: Graph transformation rules used to solve the CRA problem

In Fig. 1, we present the different graph transformation rules. First, we describe them and then we will refer to them in the next paragraphs.

Rule 1 *createClass*: creates a class and inserts it to the model.

Rule 2 *assignFeature*: assigns an unassigned feature to an existing class.

Rule 3 *createClassWithFeature*: creates a class for a feature that is not assigned yet.

Rule 4 *reassignFeature*: reassigns a feature from a class to an other existing class.

Rule 5 *reassignFeatureAndRemoveClass*: reassign a feature and remove the class if it has no features.

Rule 6 *mergeClasses*: reassign all the features to a target class and remove the source class.

a) Atomic Modifications with Bounds: The idea of the first approach is to create classes and assign features separately (*createClass* rule and *assignFeature* rule). Using such atomic modifications, one can easily see that all the possible solutions are reachable (it is complete) and probably this is the first to think of when defining a DSE problem as these are the simplest rules. However, having a transformation rule that can create objects (*e.g.,* classes) without upper bound can make the exploration strategy to create too many of them and in different order.

To overcome this issue, we incorporate an upper bound for creating classes to the condition of the *createClass* rule. In our example, this bound is the number of features available in the model as creating more classes is unnecessary.

Unused objects (*e.g.,* empty classes) is an other problem of this approach and can be handled in two ways (without modifying the exploration rules). Besides the essential goal constraints (all features are assigned) additional goal constraints (no empty classes) can be added that forbids unused objects in solutions. However, the exploration may fail to

remove these unused objects and thus fail to return with a valid solution. Alternatively, these objects can be easily removed after the exploration has finished but in this case the found solutions may vary only in the number of unused objects.

b) Atomic Modifications with Maximum One Bound: This approach is the very same as the previous one, except that we incorporate a stronger bound to the *createClass* rule: maximum one unused class can be present in the model and a second one cannot be created.

c) The Generative Approach: A similar approach to the previous ones is to design the rules in such a way to never have an unused object (*e.g.,* classes). Using the *createClassWithFeature* rule instead of *createClass* rule, a newly created class will instantly have an assigned feature resulting in two advantages: 1) there is an upper bound for creating classes (number of features) and 2) the "no empty class" constraint is fulfilled automatically.

An interesting property of this approach is that the *createClassWithFeature* rule has more activations (number of unassigned features) as opposed to the first approach (maximum one activation) and it can affect the performance and the results of the exploration depending on the algorithm.

d) The Preprocess Approach: Another approach is to initially create the maximum number of required objects (*e.g.,* one class for each feature) in a preprocess phase and then use rules that connects the elements (*assignFeature* rule).

While in overall, there will be less transitions in the search space than the first approach as the classes are already created, there will be many activations available in the same state on average as any feature can assigned to any class. Also a post process is required to remove empty classes.

e) The Initial Solution Approach: The fifth approach is to create a valid initial model and use transformations that keeps the model valid throughout the exploration. In our example, this means to initially create and assign a class for each feature and use the *reassignFeatureAndRemoveClass* rule (*InitialSolution*).

f) Initial Solution with mergeClasses rule: Alternatively to the *reassignFeatureAndRemoveClass* rule, the *mergeClasses* rule can be used instead, which allows to merge two classes that have more than one features assigned (*InitialSolutionMergeClasses*).

IV. Evaluation

We carried out measurements for each approaches by 1) traversing the full search space with a depth-first search algorithm for initial models containing 3-7 features to understand the characteristics of the search space and by 2) searching for optimal solutions with the NSGA-II [4] algorithm for an initial model containing 18 features (initial model B introduced in the TTC case). The NSGA-II algorithm was configured with population size of 20 and with a mutation rate of 0.8.

The measurements were executed on a Intel® Core™ i5-2450M CPU @2.5 GHz desktop computer with 8 GB memory. The approaches were implemented in the open-source VIATRA-DSE framework [7].

Fig. 2 shows the first set of measurements with the number of states, transitions and the transitions-states ratio of the search space by model size (number of features). The measurements show that using atomic modifications without significant bound of creating objects results in a huge state space compared to other approaches. In the other hand, preprocessing the initial model and using carefully crafted rules (*e.g., mergeClasses* rule) may shrink the state space significantly.

The results of running an NSGA-II algorithm with different exploration rules shows interesting results. Fig. 3 depicts the median and the maximum found CRA-Index and the median time taken by different approaches where each point is an aggregation of ten separate runs. Points are missing where there were at least one run that couldn't return a valid solution, *i.e.,* there were unassigned features remaining. The horizontal axis shows the number of allowed fitness evaluations during exploration (250, 500, 750, 1000, 2000, ..., 7000).

While the *InitialSolutionMergeClasses* approach has the smallest state space, after 2000 evaluations it could not increase the CRA-Index significantly and was slower than most of the approaches. On the contrary, the *AtomicModificationsMaxOne* could improve consistently by passage of time and was the fastest among the others.

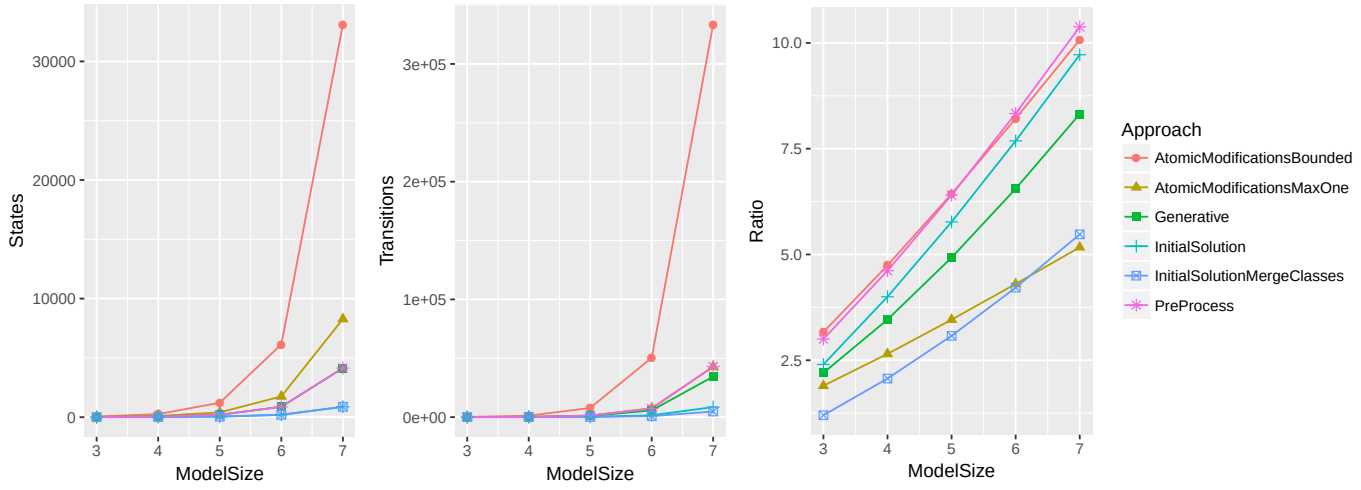
It is clear that *AtomicModificationsBounded* was the least effective approach of all and *Preprocess* was the best on average, however the best solutions are found by the slowest approach: *InitialSolution*. The *Generative* approach has the best trade-off: it found good solutions in reasonable time.

The *InitialSolution* approach was probably slow because the nature of the activations: there are relatively lot of them in each state and a good portion of them changes (disappears or appears). The used VIATRA-DSE framework evaluates a solution by transforming the initial solution and then backtracking to reuse the model. While the framework stores the activations (and activation codes) incrementally, it still has a lot of work maintaining them.

V. Conclusion

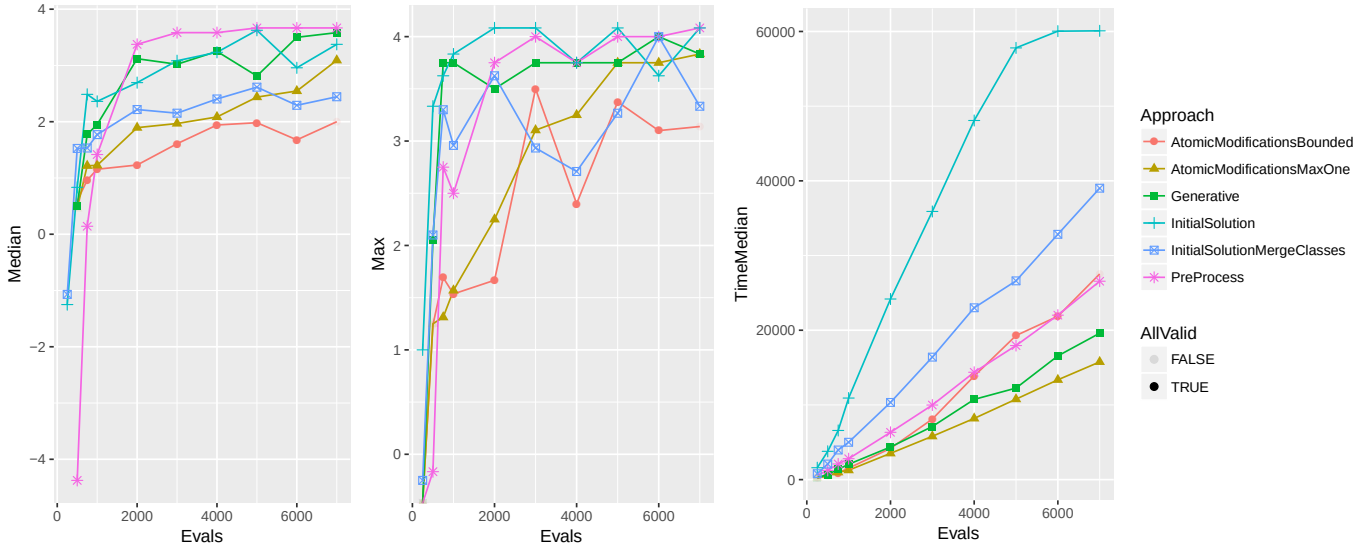
In this paper, we presented several ways to define the exploration rules of a rule-based DSE problem and investigated the impact of these rules. Based on the class responsibility assignment case, our observation is that using atomic modifications (creating single elements and connecting them separately) as exploration rules without any adjustments can be the worst choice. Instead, preprocessing the initial model and defining exploration rules that maintains the correctness of the model (and with that reducing the state space) may have significant positive impact on the exploration process. When using a genetic algorithm the best approach was either to create objects one at a time, maybe connecting them immediately to other elements, or creating the required amount of objects ahead of the exploration.

However, we investigated this observation only with a simple example and other problems may behave differently when modifying the exploration rules. Still, experimenting



(a) Explored number of states (b) Explored number of transitions (c) Ratio of explored number of transitions and states

Fig. 2: Comparison of approaches by model size using exhaustive search



(a) Median of reached CRA-Index (b) Maximum reached CRA-Index (c) Time taken to finish the exploration

Fig. 3: Comparison of approaches using NSGA-II by allowed number of evaluations. Each point is an aggregation of ten runs.

with different approaches is highly recommended for rule-based DSE problems to find the best system designs.

As for future work, more rule-based DSE problems could be investigated to derive a set of best practices for defining exploration rules when faced with a certain type of problem. Furthermore, based on the goal constraints and preconditions on the initial model, different set of exploration rules (approaches) could be generated automatically along with corresponding measurement skeletons.

References

[1] T. Basten, E. van Benthum *et al.*, "Model-driven design-space exploration for embedded systems: The Octopus toolset," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. LNCS, 2010, vol. 6415, pp. 90–105.

[2] J. Denil, M. Jukšs, C. Verbrugge, and H. Vangheluwe, "Search-based model optimization using model transformations," McGill University, Canada, Tech. Rep., 2014.

[3] Á. Hegedüs, Á. Horváth, and D. Varró, "A model-driven framework for guided design space exploration," *Automated Software Engineering*, pp. 1–38, 08/2014 2014.

[4] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, Á. Hegedüs, Á. Horváth, and C. Debreceni, "Multi-objective optimization in rule-based design space exploration," in *29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*.

[5] M. Fleck, J. Troya, and M. Wimmer, "Search-based model transformations with momot," in *Proceedings of ICMT 2016*. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-42064-6_6

[6] ——, "The class responsibility assignment case," in *9th Transformation Tool Contest (TTC 2016)*, 2016.

[7] "The VIATRA-DSE framework," https://wiki.eclipse.org/VIATRA/DSE, Accessed: 2017-01-23.