Multi-Objective Optimization in Rule-Based Design Space Exploration

Hani Abdeen
DIRO Université de Montréal
Montréal, Canada
abdeenha@iro.umontreal.ca

Dániel Varró
Dept. of Measurement and Information Systems, BME
DIRO Université de Montréal
varro@mit.bme.hu

Houari Sahraoui
DIRO Université de Montréal
Montréal, Canada
sahraouh@iro.umontreal.ca

András Szabolcs Nagy
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
nasz013@gmail.com

Ábel Hegedüs
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
abel.hegedus@mit.bme.hu

Ákos Horváth
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
ahorvath@mit.bme.hu

ABSTRACT

Design space exploration (DSE) aims to find optimal design candidates of a domain with respect to different objectives where design candidates are constrained by complex structural and numerical restrictions. Rule-based DSE [10, 14, 18] aims to find such candidates that are reachable from an initial model by applying a sequence of exploration rules. Solving a rule-based DSE problem is a difficult challenge due to the inherently dynamic nature of the problem.

In the current paper, we propose to integrate multi-objective optimization techniques by using Non-dominated Sorting Genetic Algorithms (NSGA) to drive rule-based design space exploration. For this purpose, finite populations of the most promising design candidates are maintained wrt. different optimization criteria. In our context, individuals of a generation are defined as a sequence of rule applications leading from an initial model to a candidate model. Populations evolve by mutation and crossover operations which manipulate (change, extend or combine) rule execution sequences to yield new individuals.

Our multi-objective optimization approach for rule-based DSE is domain independent and it is automated by tooling built on the Eclipse framework. The main added value is to seamlessly lift multi-objective optimization techniques to the exploration process preserving both domain independence and a high-level of abstraction. Design candidates will still be represented as models and the evolution of these models as rule execution sequences. Constraints are captured by model queries while objectives can be derived both from models or rule applications.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods and Search—*heuristic methods, graph search strategies*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*evolutionary prototyping, computer-aided software engineering*

Keywords

rule-based design space exploration; multi-objective optimization; model-driven engineering

1. 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 objectives. It frequently supports activities like configuration design of avionics and automotive systems. Many of such traditional static DSE problems can be solved by using advanced search and optimization algorithms or constraint satisfaction programming techniques [5, 10, 15, 18, 29].

In model-driven engineering (MDE), *rule-based DSE* [10, 14, 18] aims to find instance models of a domain that are (i) reachable from an initial model by applying a sequence of exploration rules, while (ii) constraints simultaneously include complex structural and numerical restrictions. Model driven techniques offer expressive modeling languages and advanced tools to capture the DSE problem of different domains independently on a high level of abstraction close to the domain itself. However, solving a rule-based DSE problem is a difficult challenge due to the inherently dynamic nature of the problem. Such dynamic DSE problems may arise in complex reconfiguration challenges of supervising cyber-physical systems (CPS) or IT infrastructure [18] or quick fix generation in domain-specific modeling environments [17].

As a practical observation, the solution space of a rule-based DSE problem is dense in principle, but one cannot put an *a priori upper bound* on the number of model elements used in a design candidate (i.e. model elements may be created and deleted during exploration). Unfortunately, this makes the exhaustive exploration of the design space intractable. Furthermore, many practical problems necessitate to continue the exploration of the design space incrementally from a previous solution (instead of starting the search from scratch each time). Such incremental solving is rarely handled by state-of-the-art constraint solvers (as demonstrated in [20]).

Rule-based model-driven DSE problems have additional challenges also from an optimization perspective. First, some objectives are not values of simple cost attributes but complex model metrics calculated by model queries. Furthermore, certain cost calculations may depend on the sequence of exploration rules applied on the design model. Finally, we may not find a single combined objective function but multiple objectives may need to be incorporated to identify the best design candidates.

Existing rule-based DSE solutions exploit (1) model checking with powerful graph-based symmetry reduction [14], (2) dependency analysis and hints by formal abstractions [18] or (3) different search strategies (e.g. hill climbing, simulated annealing) [10]. As a commonality in these approaches, the core exploration procedure follows a *local-search based approach*, i.e. it gradually extends the search towards promising candidates by priorities defined by local heuristics.

Global search techniques (like genetic algorithms or multi-objective optimization) have already proved to be successful in various MDE scenarios for finding constraints [16], model transformations [25] or solving static DSE problems [35] where an exhaustive search algorithm becomes infeasible. Moreover, they provide graceful degradation for problems where no solutions exist which meet all the objectives and constraints by relaxing hard constraints to soft constraints.

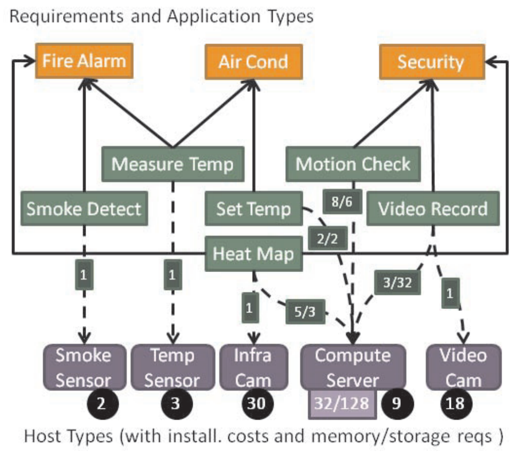
In the current paper, we propose to integrate multi-objective optimization techniques by using the Non-dominated Sorting Genetic Algorithm (NSGA-II) [9] to drive rule-based design space exploration. For this purpose, finite populations of the most promising design candidates are maintained wrt. different optimization criteria. In our context, individuals of a generation are defined as a sequence of rule applications leading from an initial model to a candidate model. Populations evolve by mutation and crossover operations which manipulate (change, extend or combine) rule execution sequences to yield new individuals. However, a key technical challenge that we face in genetic rule-based DSE is to preserve the feasibility of candidate solutions which are generated using genetic operators. Indeed, in rule-based DSE, crossing two feasible solutions, and/or randomly mutating a feasible solution, may yield infeasible candidates if the corresponding rule execution sequence is infeasible. In our approach, candidate solutions generated using genetic operators are automatically corrected to preserve their feasibility.

The main added value of our multi-objective optimization approach for rule-based DSE is to seamlessly lift multi-objective optimization techniques to a domain-independent model-level exploration process while preserving a high-level of abstraction. Design candidates will still be represented as models and the evolution of these models as rule execution sequences. Constraints are captured by model queries while objectives can be derived both from models or rule applications. On the theoretical level, models are formalized as graphs, model queries as graph patterns and exploration rules as graph transformation rules, thus our work can be considered as a multi-objective exploration of graph transformation system. Our approach is supported fully automated tooling built on advanced components of the Eclipse framework.

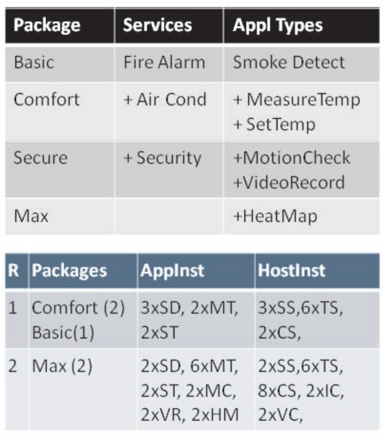
2. BACKGROUND

2.1 Motivating example

As a motivating example of the paper, we consider modeling the configuration of a smart building which offers offices to rent with highly configurable services such as fire alarm, air conditioning and security monitoring (see Fig. 1). A smart building is frequently considered as a cyber-physical system (CPS) where services need to be deployed on both embedded (sensors, controllers, etc.) and virtual (e.g. servers, cloud) computational units, which significantly complicates the design and maintenance of service configurations over a changing infrastructure.



(a) Architecture



Package	Services	Appl Types
Basic	Fire Alarm	Smoke Detect
Comfort	+ Air Cond	+ MeasureTemp + SetTemp
Secure	+ Security	+MotionCheck +VideoRecord
Max		+HeatMap

R	Packages	AppInst	HostInst
1	Comfort (2) Basic(1)	3xSD, 2xMT, 2xST	3xSS,6xTS, 2xCS,
2	Max (2)	2xSD, 6xMT, 2xST, 2xMC, 2xVR, 2xHM	2xSS,6xTS, 8xCS, 2xIC, 2xVC,

(b) Services and Requests

Figure 1: Configuration model of a smart building

In our smart building example, companies are offered to rent offices with multiple rooms each of which can be configured according to four service packages (see Fig. 1b):

- **Basic**: This package runs the compulsory fire alarm service which requires one smoke sensor for each room.
- **Comfort**: This package also offers air conditioning by measuring temperature by three sensors (per room) and setting the required temperature. Measuring the temperature also offers a backup solution for fire alarm should the sensors fail.
- **Secure**: This package extends the *Comfort* package to offer security surveillance by a video camera continuously recording events in the room and a motion check application which highlight critical events automatically.
- **Max**: This package enhances the *Secure* package by providing a heat map of the room which can be used for fire alarm as well as for surveillance purposes.

Two sample company requests are also listed in Fig. 1b which summarizes the selected packages for a certain number of rooms together with the application instances to be deployed and hardware devices to be installed. For instance, the first request consists of 2 rooms with comfort package and 1 room with basic package, and it necessitates to run smoke detector (SD) service (for 3 rooms, 1 device per room), the measure temperature (MT) service (for 2 rooms, 3 devices per room) and the set temperature application (for 2 rooms, jointly installed on a compute server).

Domain metamodel. The main concepts of configuration design for this smart building are generalized in the metamodel of Fig. 2. Renting companies will issue Requests for implying a set of Requirements each of which identifies a required service (called application type, shortly ApplType) and the number of redundant instantiations for running this application (count attribute). Sensors and computation units are uniformly called HostTypes. Each application type may claim several host types and sufficient amount of resources (e.g. memory, storage) for its execution as defined by a resource requirement (shortly, ResReq). For instance, calculating a heat map requires 5 GB memory and 3 GB permanent storage as defined by the labels of dashed arrows in Fig. 1a.

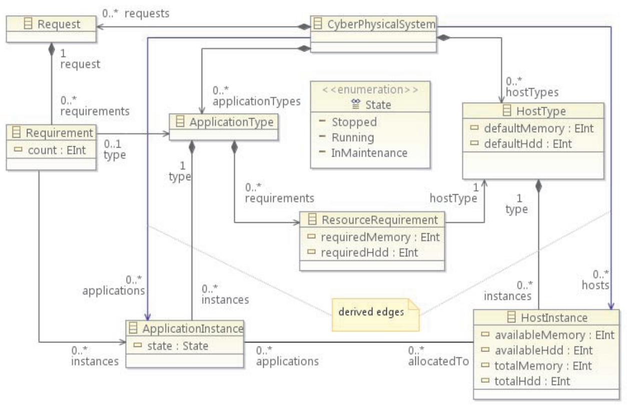


Figure 2: Metamodel for smart building configuration

In the running system, multiple application instances, ApplInst shortly, of an application type may exist each of which is deployed on a host instance (shortly, HostInst) corresponding to a specific host type. Such deployment consumes a certain amount of resources in the host instance (up to its total memory and storage) as specified by the corresponding resource requirement. Application instances need to be started after they are allocated to a host instance, and stopped when they are no longer needed, which fact is represented by the state attribute.

2.2 A rule-based DSE problem

Configuring the smart building (i.e. installing devices, allocating and running applications) can be considered as a design space exploration problem. However, this configuration is not a static problem as (1) requests may change over time (new requests arrive, existing ones are canceled) and (2) certain faulty devices may no longer function. This way, we are interested in an incremental approach for calculating a new configuration which starts from the last configuration and incorporates the changes in the context and requirements of the system. Furthermore, there are multiple constraints and objectives which needs to be incorporated for a good candidate configuration.

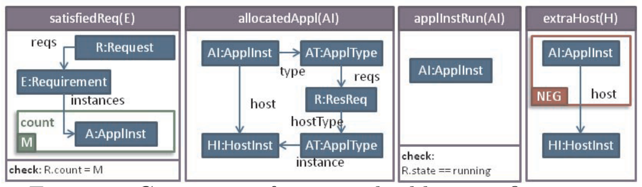


Figure 3: Constraints for smart building configuration

Constraints.

Constraints capture valid or invalid configurations. They are frequently formalized by graph patterns as well or ill-formedness criteria which capture the violations of the corresponding constraint. In our example (see Fig. 3),

- **satisfiedReq(E)** identifies a requirement E of a request R which is instantiated into a sufficient number of application instances (i.e. the number of instances is equal to the required redundancy);
- **allocatedAppl(AI)** identifies an application instance AI which is allocated to a host instance HI to fulfill the resource requirement between the corresponding application type AT and host type HT;
- **appInstRun(AI)** identifies an application instance of a configuration which is running;
- **extraHost(H)** identifies a host instance H which does not host any application instances.

The first three constraints capture desired situations, while the fourth constraint captures an undesired case. All these constraints are captured by means of graph patterns [6] which denote structural conditions that the underlying graph model needs to respect. Given a constraint c and an instance model M, $m : c \mapsto M$ denotes a graph morphism identifying a violating fragment of M.

Objectives.

In order to obtain a good configuration, we need to address various objectives:

- Ideally, all configuration constraints need to be satisfied, thus this is a top-level objective.

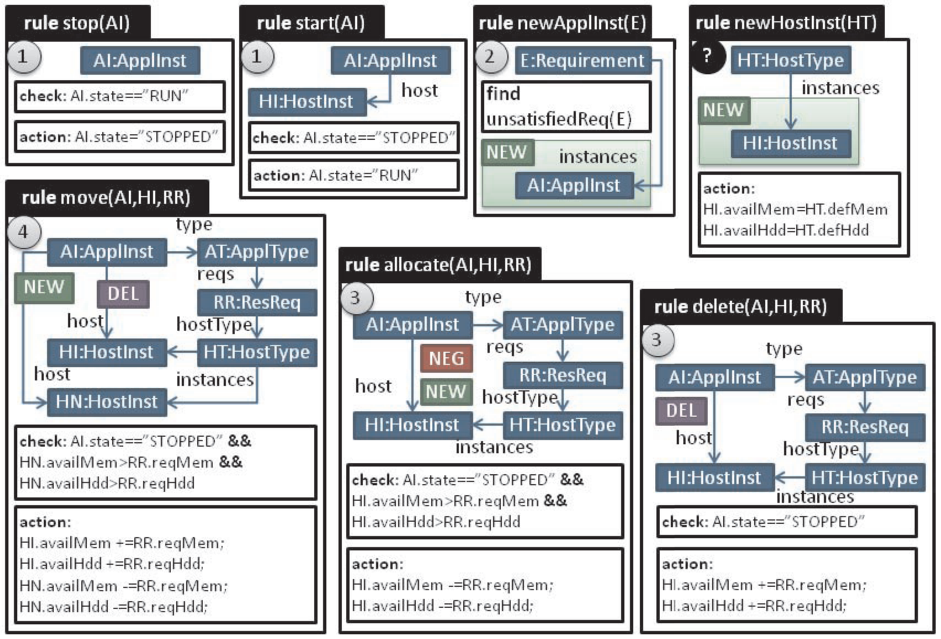


Figure 4: Exploration rules of the smart building example

- The smart building operators aim at maximizing the utilization of compute servers. Here, the best utilization of memory or storage is incorporated for each server, and the average of utilization is taken as a resource objective.
- Installing new host instances and other configuration operations imply certain cost. Minimizing such costs is another objective of configuration design.

In our context, we distinguish between *model objectives* which are calculated from design candidates (e.g. by querying the underlying model) and *trajectory objectives* which are calculated along trajectories of applied exploration rules.

Exploration rules.

In rule-based DSE, design candidates are allowed to evolve by executing exploration rules. In the paper, such exploration rules are captured by graph transformation rules $r = (L, R)$ where L denotes the left-hand side graph pattern prescribing the precondition of rule application, while R declaratively describes the effects of the rule. Rules may have (i) input and output parameters to pass contextual objects to other rules, and (ii) costs incurring when a rule is applied. A rule r is applied on a model M by (1) finding a match m of graph pattern L in M (denoted as $m : L \mapsto M$ and also called an activation of rule r), (2) then removing elements from M which have an image in $L \setminus R$ and finally (3) creating new elements in M for elements in $R \setminus L$. As a result of rule application, we obtain a new model M_1 which step is denoted as $M \xrightarrow{r,m} M_1$. The exploration rules of our example are depicted (in a combined notation like in GROOVE [14]) in Fig. 4.

- Rule **newHostInst** installs a new host instance HI of a host type HT and sets the available resource parameters to that of the host type.
- Rule **newApplInst** creates a new application instance AI in accordance with the count attribute of requirement E (by reusing the condition defined by graph pattern unsatisfiedReq(E)).
- Rule **start** initializes a stopped application instance AI which is already allocated to a host device while rule **stop** stops a running instance.
- Rule **allocate** aims to allocate (an unallocated and stopped) application instance AI to a host instance HI in accordance with the resource requirement RR provided that sufficient memory and storage space is still available at HI.
- Rule **delete** removes an existing allocation of a stopped application instance AI from a host instance HI, and frees the related memory and storage resources of HI.
- Rule **move** combines the allocate and delete rule into one, and changes the allocation of an application instance AI from host instance HI to HN, and adjusts the resource usage accordingly.

A cost is associated to the application of each exploration rule. The cost of rule **newHostInst** is specific to the matched host type element HT as defined by the black circles in Fig. 1a, while the cost of all other rules are defined in gray circles in Fig. 4.

3. APPROACH

In this section, we describe our approach to integrate multi-objective optimization techniques by using the Non-dominated Sorting Genetic Algorithm (NSGA-II) [9] to drive rule-based design space exploration. In the following, we describe the basic principles of the used multi-objective optimization technique, then we present an overview of our approach and its implementation details.

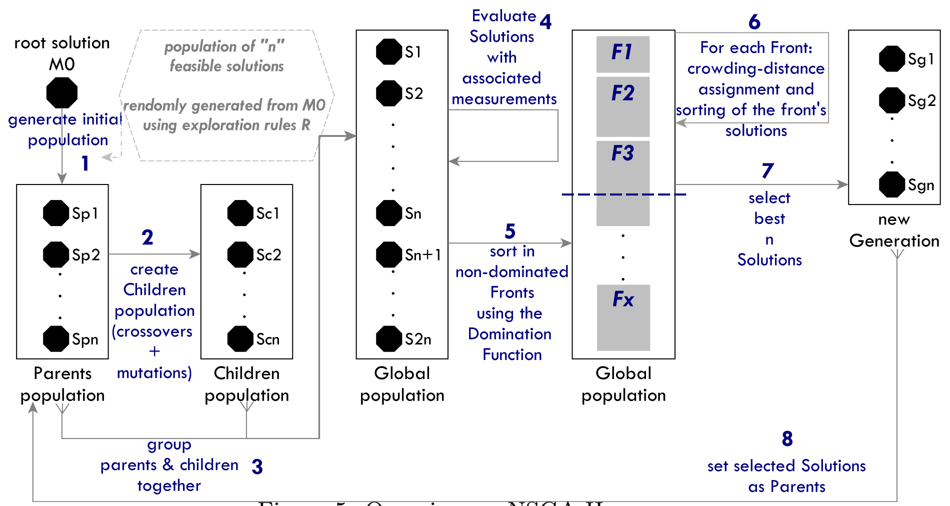

root solution M0
generate initial population
1
population of "n" feasible solutions
rendomly generated from M0 using exploration rules R
Sp1
Sp2
Spn
Parents population
2
create Children population (crossovers + mutations)
Sc1
Sc2
Scn
Children population
group parents & children together
3
S1
S2
Sn
Sn+1
S2n
Global population
Evaluate Solutions with associated measurements
4
5
sort in non-dominated Fronts using the Domination Function
F1
F2
F3
Fx
Global population
6
For each Front: crowding-distance assignment and sorting of the front's solutions
7
select best n Solutions
Sg1
Sg2
Sgn
new Generation
8
set selected Solutions as Parents

Figure 5: Overview on NSGA-II process

3.1 Optimization algorithm principles

Fig. 5 shows an overview on the NSGA-II. The aim of NSGA-II is to find a set of Pareto optimal solutions in a single run. As a genetic algorithm, NSGA-II performs a global exploration of the search space by making and evolving finite populations of candidate solutions using selection and genetic operators. The output of the algorithm is a set of the fittest solutions (*i.e.,* the Pareto front) produced along all generations. The decision maker can select one of the fittest solutions according to his/her preference (*e.g.,* the solution which satisfies all the requirements).

When searching for solutions to a problem using Pareto optimality (*i.e.,* multi-objective), the search yields a set of solutions that are not-dominated [37]. If all objective functions are for maximization, we say that a feasible solution s_1 *dominates* another feasible solution s_2 ($s_1 \succ_O s_2$), if and only if, s_1 is better than s_2 for at least one objective, while s_2 is not better than s_1 regarding all the objectives in O [9]:

$$\exists o_j \in O: \; o_j(s_1) > o_j(s_2) \;\wedge\; \nexists o_i \in O: \; o_i(s_2) > o_i(s_1) \qquad (1)$$

NSGA-II sorts solutions in ordered fronts, from best to worst, according to their non-domination level.

However, from a practical MDE viewpoint, constrained multi-objective optimization is important in the context of DSE. This is due to the fact that the primary objective of a DSE approach is to find valid solutions that satisfy *all the requirements* of the underlying problem. Other objectives, such as reducing the cost of obtained solutions, could not add an effective value to the optimization results unless the obtained solutions are valid. Hence, we adapt the constraint-handling strategy with NSGA-II that was proposed in [9, 22].

In the presence of constraints (*i.e.,* requirements), a solution can be either valid (*i.e.,* it satisfies all the constraints) or invalid (*i.e.,* it does not satisfy all the constraints, totally or partially). Considering the top-level preference of constraints' satisfaction over other objectives that we described in Sec. 2.2, the domination function in Equation (1) is modified as follows:

Definition. A solution s_1 is said to **constrained-dominate** a solution s_2, if one of the following conditions is true:

1. s_1 is valid and s_2 is not.
2. Both solutions, s_1 and s_2, are invalid, but s_1 has *a smaller overall constraint violation.*
3. s_1 and s_2 are valid and s_1 dominates s_2 with the usual domination function (Equation (1)) .

The idea behind this constrained-domination strategy is that, on the one hand, any valid solution has a better non-domination rank than any invalid solution. On the other hand, valid solutions are ranked into their non-domination level based on their associated quality as measured by the values of objective functions. And, invalid solutions are ranked into their non-domination level in descending order according to their associated constraint violation.

In the following we present our adaptation of the NSGA-II to the problem of rule-based DSE by defining our implementation of the following elements:

- Representation of candidate solutions (individuals).
- Optimization objectives.
- Genetic operators used to explore the search space.

3.2 Approach overview

The optimization inputs and procedure. Fig. 6 shows an overview of our approach. Given an initial model M_0 belonging to a domain model DM, a set of requirements to be satisfied Req, and a set R of exploration rules, our multi-objective search-based approach for rule-based DSE explores the design space starting from M_0 by maintaining finite populations of the most promising design candidates wrt. the set of requirements Req and other optimization criteria O. In our approach, each individual of evolving populations is a map of a candidate model M_n obtained as a sequence of rule executions, denoted as $\vec{r_n}$, leading from the initial model M_0 to M_n: $\vec{r_n} = M_0 \xrightarrow{r_1,m_1} M_1 \ldots \xrightarrow{r_n,m_n} M_n$, where $r_i \in R$ and m_i is the match of r_i in M_i.

The generation of the initial population starting from M_0 (*step 1* of the optimization process in Fig. 6) is achieved by applying random (but executable) exploration rules from R on M_0. The second step of the optimization process in Fig. 6 is based on the NSGA-II as explained in Sec. 3.1 and

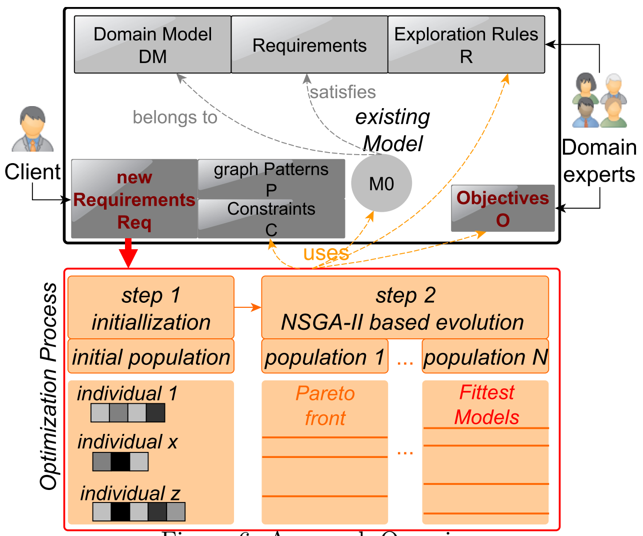


Figure 6: Approach Overview

Fig. 5, where populations evolve by applying mutation and crossover operations which change and/or extend existing rule execution sequences to yield new individuals. The requirements are defined as soft constraints C that are captured by model queries (see Fig. 3) while other optimization objectives can be derived both from models or rule execution sequences. Each constraint c_i in C can be associated to a specific weight w_i describing the relative importance of this constraint, thus each violation of c_i (calculated as a match of the corresponding graph pattern) will be weighted accordingly.

Candidate solution representation. The *multi-objective rule-based design space exploration problem* can be defined as: $DSE = (M_0, C, O, R)$ where M_0 denotes the initial model, C denotes the (structural and attribute) constraints characterizing valid design candidates, O is a set of numerical objectives which needs to be optimized, while R is a set of exploration rules describing valid evolutions of the design.

A *candidate solution* of a DSE problem is a pair $S_{cand} = (M_{cand}, \vec{r_{cand}})$ where (1) the candidate model M_{cand} fulfills all (or some) constraints in C and (2) it is reached from the initial model M_0 by a sequence of rule executions $\vec{r_{cand}}$. A *model objective* o_m is evaluated for a candidate solution S_{cand} on the candidate model M_{cand} while a *trajectory objective* o_t for S_{cand} is evaluated on the sequence of rule executions $\vec{r_{cand}}$ leading to S_{cand}. Considering our running example, reducing the number of constraint violations is a model objective or increasing utilization, while reducing the cost associated to the trajectory $\vec{r_{cand}}$ is a trajectory objective.

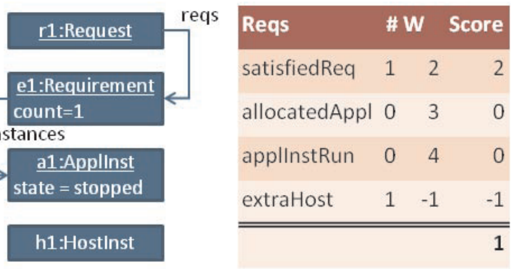
Definition of Feasible and Valid candidate solutions. By definition, a candidate solution is called *feasible* if its rule execution sequence $\vec{r}$ is executable. In a genetic approach for rule-based DSE, infeasible solutions can occur when applying genetic operators (*e.g.,* crossover) on existing feasible solutions. However, in our approach, such infeasible solutions are automatically corrected (or truncated) to guaranty their feasibility, or omitted if they cannot be corrected.

Consider $newHostInst(HI); allocate(AI, HI, RR)$ which is a rule execution sequence that aims to create a new host instance HI and then allocate an application instance AI to it is not executable if there is no application instance AI to be allocated. To correct this infeasible sequence, the creation of a new application instance by rule $newAppInst(AI)$, should precede the $allocate(AI, HI, RR)$ rule. Otherwise, the sequence must be truncated so that it includes only the rule execution $newHostInst(CS1)$.

A feasible solution $S = (M, \vec{r})$ is defined as a *valid* solution if its associated model M fulfills *all constraints* in C: *i.e.,* $\forall c \in C \nexists m : c \mapsto M$.

The objective of constraints fulfillment. As constraints are formalized by graph patterns, in order to evaluate the degree of well-formedness constraints are met or ill-formedness constraints are violated in a candidate model M_n, we use the weighted sum of the number of matches for the corresponding graph patterns P.

For instance, let us evaluate the constraints of Fig. 3 on a sample model M of Fig. 7. For this model, our optimization approach will returns that the graph pattern *satisfiedReq* has 1 match and the graph pattern *extraHost* has 1 match. In our example, the weight of *satisfiedReq* is set to $w_1 = 2$ and the weight of *extraHost* is set to $w_2 = -1$, therefore, the degree of constraint violations in M is: $ConstViol(M) = 1 \times w_1 + 1 \times w_2 = 1$.



Reqs	#	W	Score
satisfiedReq	1	2	2
allocatedAppl	0	3	0
applInstRun	0	4	0
extraHost	1	-1	-1
			1

Figure 7: Objective of constraint fulfillment

Formally, let $matches(p, M)$ return the number of matches of the graph pattern $p \in P$ in the model M, and let w_p denote the weight associated to the constraint described by p (where w_p is a positive value for well-formedness constraints and a negative value for ill-formedness constraints). Then our objective of constraints fulfillment is defined as follows:

$$ConstFulfillment(M) = \sum_{\forall p \in P} w_p \times matches(p, M) \tag{Obj.1}$$

The primary optimization objective of our approach, as explained in Sec. 3.1, is to maximize $ConstFulfillment(M)$, i.e. to maximize the fulfillment of positive constraints and minimize the degree of negative constraint violations.

Model-specific objectives. Our approach allows to define domain-specific objectives captured by graph patterns over the underlying model. Thanks to the incremental query evaluation approach (see Sec. 3.3), the re-evaluation of such model objectives is instantaneous upon model changes.

In the context of our motivation example in this paper, we define the model-specific objective of maximizing the utilization of compute servers ($CSUtil$), so that the best utilization of memory or storage is incorporated for each server.

Let $Util(CS_i)$ return the (normalized) resource utilization for the computer server CS_i while the system-level utilization of computer servers $CSUtil$ for a solution $S = (M, \vec{r})$ is defined as the mean of the utilization of each individual computer server element in M:

Obj.2

$$CSUtil(S) = \frac{1}{n} * \sum_{j=1}^{j=n} Util(CS_j)$$

In the above equation, n is the number of computer server instances in the underlying model M.

Rule sequence objectives. Two valid solutions can be achieved via two different feasible sequence of rule executions. Therefore, we may define objectives specific to rule execution sequences to evaluate the cost incorporated in achieving a valid solution along a specific path.

For this purpose, we define the cost of a feasible solution $S = (M, \vec{r})$ as the sum of costs of all rule executions in its sequence of rule executions $\vec{r}$:

Obj.3

$$Cost(S) = \sum Cost(M_{i-1} \xrightarrow{r_i, m_i} M_i) \quad \forall M_{i-1} \xrightarrow{r_i, m_i} M_i \in \vec{r}$$

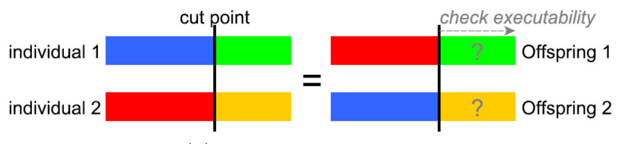
$Cost(S)$ takes its values in the interval $[0..\infty[$. Minimizing $Cost(S)$ is an objective of our optimization approach. Computing the cost $Cost(M_{i-1} \xrightarrow{r_i, m_i} M_i)$ of a rule execution in $\vec{r}$ depends on three parameters to be defined by the domain experts:

- *Fixed cost:* the fixed cost C_b of the applied rule r_i; formally, $Cost(... \xrightarrow{r_i, m_i} ...)^F = C_b(r_i)$. As defined in Fig. 4, creating an application instance will always have the same cost, which is $C_b(newApplInst) = 2$.
- *Match cost:* the match cost C_m which is associated to the match m_i in M_i; formally $Cost(... \xrightarrow{r_j, m_j} ...)^M = C_b(r_j) + C_{m_j}(r_j)$, where $C_{m_j}(r_j)$ returns the cost of r_j according to its match m_j. For instance, the cost of rule $newHostInst$ is specific to the matched host type element HT (as defined by the black circles in Fig. 1a).
- *Sequence cost:* the sequence cost C_s may depend on the position of the rule execution $... \xrightarrow{r_i, m_i} ...$ in $\vec{r}$. For this purpose, we define the cost of such a rule execution as relative to its position in $\vec{r}$ on the same match m: $Cost(... \xrightarrow{r_k, m_k} ...)^S = position(r_k, m_k) \times Cost(... \xrightarrow{r_k, m_k} ...)^M$, where $position(r_k, m_k)$ returns the position of r_k application on the same match m_k in $\vec{r}$.

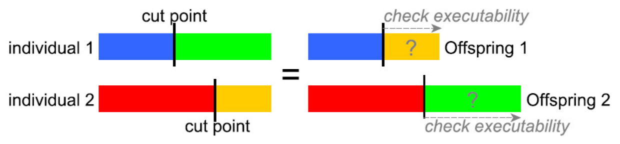
Genetic operators (mutation and crossover). In this paper, we define and use different types of mutation and crossover operators for exploring the design space. In our context, mutating a solution $S = (M, \vec{r})$ means to modify the sequence of rule executions $\vec{r}$, which is conceptually different from most genetic approaches used for DSE purposes. This can be achieved in different ways:

- *Add new rule execution:* a new sequence of rule executions $\vec{r}'$ is generated by selecting an appropriate exploration rule r' from R, that can be applied on M, and execute it: $\vec{r}' = \vec{r} + \{M \xrightarrow{r', m} M'\}$
- *Delete a random rule execution:* a new sequence of rule executions $\vec{r}'$ is generated by deleting a random rule execution re_i in $\vec{r}$: $\vec{r}' = \{re_0, \ldots, re_{i-1}, re_{i+1}?, \ldots ?\}$. The question marks in the aforementioned sequence denote the execution rules which will be checked for executability after delete is performed. Indeed, after deleting re_i, the executability of the new sequence $\vec{r}'$ is checked starting from the rule execution re_{i+1}, so that if re_{i+1} is not executable anymore it is then ignored (removed from the sequence), and so on for each re_{i+k}, $k > 1$.
- *Swap between two rule executions:* a new sequence of rule executions $\vec{r}'$ is generated by selecting a random rule execution re_i in $\vec{r}$, then selecting another rule execution re_j $(j > i)$ in $\vec{r}$, that can replace re_i and still executable, then swap between re_i and re_j so that: $\vec{r}' = \{re_0, \ldots, re_j, re_{i+1}?, \ldots ?, re_i?, re_{j+1}?, \ldots ?\}$. Similarly to the case of delete a random rule execution discussed above, the executability of the new sequence $\vec{r}'$ is checked starting from the rule execution re_{i+1}.

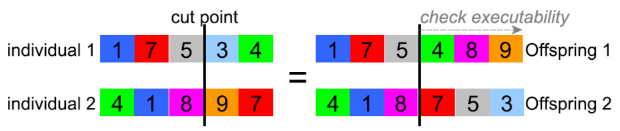
The crossover operators apply on two individuals represented by the sequences of rule executions of two parent solutions $S^1 = (M^1, \vec{r}^1)$ and $S^2 = (M^2, \vec{r}^2)$, and generate two new offspring individuals (children). Fig. 8 describes the three crossover operators that our optimization process uses:



(a) One-point crossover



(b) Cut-and-splice crossover



(c) One-point permutation crossover

Figure 8: Crossover operators

- *One-point crossover (Fig. 8a):* a single crossover point on both sequences of rule executions of parents is selected. All rule executions beyond that point in either sequences of rule executions are swapped between the two-parent sequences. The resulting sequences of rule executions are the children.
- *Cut-and-splice crossover (Fig. 8b):* cut-and-splice crossover is a variation of the one-point crossover where the difference is each parent's sequences of rule executions has a separate choice of crossover point. As a result, the children sequences of rule executions will have different length than that of their parents.

- *One-point permutation crossover (Fig. 8c):* in this crossover operator, every rule execution re_i in either sequences of rule executions (parents) will have an id. This id is based on the applied rule r_i and the match element m_i of re_i. Hence, two rule executions, r_i and r_j, can have the same id iff they are applications of the same rule r on the same match m. Representing the rule executions by their *id*s, a sequence of rule executions will have a permutation representation, as in Fig. 8c. With the one-point permutation crossover, one crossover point is selected on both sequences of rule executions of parents, from the first (second) parent the permutation is copied up to this point, then the second (first) parent is scanned and if the *id* of the rule execution is not yet in the offspring, the rule execution is added.

For every crossover operation, our approach performs an automatic executability check of rule executions occurring after the cut point(s) in children sequences of rule executions. The correction mechanism is identical to that used in the delete and swap mutations that we described above.

3.3 Implementation Overview

The proposed framework has been fully automated by implementing it on top of the ViatraDSE framework [18]. Input models and design candidates are represented as Eclipse Modeling Framework (EMF) models. The open source EMF-IncQuery [36] framework is used for evaluating constraints and calculating the values of objectives over instance models incrementally upon model changes. Finally, operations are captured by graph transformation rules. Additional search and configuration parameters can be set prior to starting a multi-objective optimization run.

For each individual in a population, we store the sequence of rule applications that leads to them from an initial model, thus individuals get recalculated several times. However, due to the incremental transformation support by EMF-IncQuery, this recalculation is very fast. Furthermore, each individual can be processed in parallel by different threads to speed up the exploration process.

Implementation challenges included to (1) identify if two individuals are identical in a population and (2) to efficiently encode the individuals themselves. Instead of performing costly graph isomorphism checks to address (1), we rely upon the vector of numeric objectives: if two individuals have the same values for all objectives, then one of them is removed from the population. For (2), we use domain-specific state encodings to identify and store matches of rules in order to replay them when deriving a new individual.

4. EXPERIMENTAL EVALUATION

As there are no widely established benchmarks available for evaluating rule-based DSE approaches, we carried out experimental evaluation in the context of our case study. For this purpose, we compare our multi-objective optimization (NSGA) approach with (1) random simulation (Random) and (2) a fixed priority local search (FPLS) strategy used as a basis of comparison in [18]. As a consequence, the DSE problem is identical in both cases, furthermore, the evaluation of graph patterns and execution of graph transformation rules is carried out by the same implementation. This way, any difference between the measurement results is expected to be affiliated to the substantially different search strategies. Our measurements aim to address which DSE approach finds better candidates wrt. different objectives. For this purpose, we test the following hypothesis using two-tailed Wilcoxon tests:

$\mathbf{H}_0$ There is no significant evidence that NSGA outperforms FPLS and/or Random.

$\mathbf{H}_1$ There is a statistical evidence that NSGA outperforms other DSE approaches wrt. different objectives.

4.1 Experimental scenario

In our experimental scenario, we generate requests for an increasing number of rooms (4, 6, 8, 12) with an equal use of all packages (and respective model sizes of 130, 200, 230 and 330 graph elements). The initial model only contains the requests with requirements and the application and host types but no host instances are available. Therefore, it is the role of the DSE process to (1) create a sufficient number of application and host instances, (2) allocate application instances to host instances, and then (3) start and stop the application instances by applying the appropriate exploration rules. In the most complex case, the different exploration techniques had to synthesize a rule sequence consisting of over 200 steps.

In a preparatory phase, we experimented with different configurations of our multi-objective DSE approach, and we decided on the most promising configuration parameters, such as population size of 15 individuals, iterations between 400 and 1200 steps, crossover by permutation, and high rate of mutations. Our measurements were run on 8-core desktop computers with 32 GB of RAM running on Linux operating system. The used Java version was 1.7.0_55 and the heap size was 24GB.

Then for the experiments, we set up a timeout of 2 minutes for test cases (except for the largest example where it was 5 minutes) and run 30 experiments on the different problem sizes. One experiment was constructed as follows:

- **NSGA**: We selected population size *pop* and iteration number *it* for a problem size, and run our algorithms. At the end of each run, we selected only one solution from the Pareto front produced by NSGA. We selected the solution which has the best constraints' fulfillment value. If several solutions have the best constraints' fulfillment value, then we select the solution which is characterized by the minimal cost and/or the maximum usage of computer servers.
- **Random**: We executed $pop \times it$ random simulation runs and recorded the best result.
- **Fixed priority LS (FPLS)**: We set up priorities in a way to guarantee that all application instances will eventually be allocated to host instances. Our priorities guaranteed that the first three soft constraints will definitely be guaranteed, but we may generate more host instances than necessary.

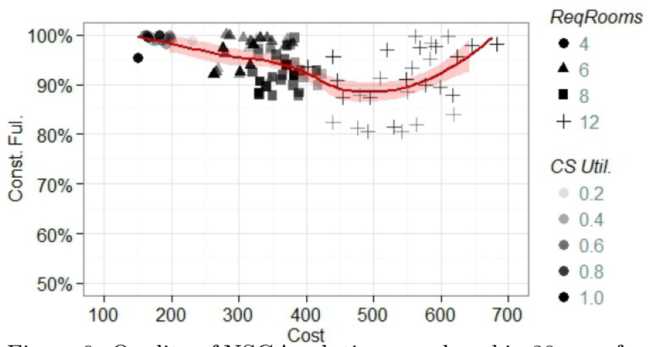


Figure 9: Quality of NSGA solutions produced in 30 runs for different problem sizes, as measured by the measurements normalized constraints' fulfillment, cost and computer server utilization

4.2 Results analysis

Before comparing the results of our approach NSGA with those of FPLS and Random, we analysis the quality of NSGA solutions. Figure 9 shows the distribution of NSGA produced solutions in 30 runs for considered problem sizes (4, 6, 8 and 12 requested rooms). The figure shows that in all considered scenarios, NSGA produced solutions have overall good quality. Considering all the produced solutions ($30 runs \times 4 scenarios = 120 solutions$), the minimum fulfillment of constraints is above 80%, and for the major body of produced solutions, the fulfillment of constraints is above 90%. Analyzing the evolution of the mean value of constraints' fulfillment through different problem sizes, we find that it takes its minimum value, around 90%, in the problem size 12 where the incorporated cost in the sequences of rule executions is relatively small, as compared to the cost of other solutions in the problem size 12. Indeed, this relative low fulfillment of constraints is mainly due to the following fact: the optimization process was stopped before reaching sequences of rule executions that have enough depth to satisfy all the requirements associated to this problem size. Increasing the number of iterations of the optimization process in this problem size, NSGA was able to find better solutions in terms of constraints' fulfillment, as demonstrated by solutions which have high cost. As for smaller problem sizes, such as in the scenarios of 4 and 6 requested rooms, in almost all runs NSGA was able to find fully valid solutions that satisfy all the constraints.

Table 1: Comparing between the results of our approach NSGA and FPLS, and between the results of NSGA and Random, with different problem sizes with regard to the number of requested rooms: two-tailed Wilcoxon tests with $\alpha = 0.05$ and adjusted $p.value$ using the Benjamani and Hochberg (BH) correction for multiple tests.

	Δ(NSGA - FPLS)			**Δ(NSGA - Random)**		
Pb size	Const. Ful.	Solution Cost	CS Util.	Const. Ful.	Solution Cost	CS Util.
4	**+20****	**−369****	**+0.32****	**+44****	**−145****	**+0.19****
6	**+27****	**−559****	**+0.37****	**+91****	**−118****	+0.06
8	**+20****	**−746****	**+0.46****	**+92****	**−239****	**+0.22****
12	+6	**−1058****	**+0.51****	**+55****	−8	**+0.18***

Values in **bold-face** denote that NSGA results outperformed the results of FPLS and/or Random, with statistical significance. ** and * denotes results which are statistically significant at $\alpha = 0.01$ and $\alpha = 0.05$, respectively.

To confirm our claim that NSGA produces good results, we compare NSGA's solutions to those produced by FPLS and Random. Table 1 shows clearly that NSGA outperforms both FPLS and Random, with statistical evidence, in almost all cases. Indeed, only in the problem size 12, there is no significant statistical evidence that NSGA outperforms FPLS with regard to constraints' fulfillment. However, in this case, NSGA significantly outperforms FPLS in reducing the cost of solutions and increasing the usage of computer server resources. Hence, NSGA overall significantly outperforms FPLS, and the same finding apply for the Random approach. As a consequence, we reject the null hypothesis H_0 and accept H_1.

5. RELATED WORK

Rule based design space exploration frameworks. Model checking approaches to analyze GT systems are similar to our approach as they also perform state space exploration. One can categorize them as *compiled approaches* such as [2, 3, 11, 12, 34], which translate graphs and GT rules into off-the-shelf model checkers to carry out verification, and *interpreted approaches* like [1, 26, 30], which store system states as graphs and directly apply transformation rules to explore the state space, similarly to our approach.

In [14] the state space explored by the GROOVE framework is stored as a structured graph model that can be queried using logical expressions. This approach allows the evaluation of trajectories using cost functions defined after the exploration and even the combined assessment of multiple solutions.

Common in these approaches that they place emphasis on exhaustive traversal (e.g. by optimizing the storage of individual states), while we aim at finding solutions quickly using genetic algorithms.

In [10] the T-Core framework is used for implementing typical meta-heuristic exploration strategies, such as hill climbing and simulated annealing using the transformation primitives of the framework while the operations are specified as graph transformation rules. As a distinguishing feature, our approach supports trajectory based objective definition and mutation operations.

Compared to previous work of the authors, we extend our model-driven design space exploration framework [18] by providing support for multi-objective optimization algorithms that complement our guided local-search based approaches [19]. The application of multi-objective approaches in a model-driven DSE context provides a unique combination of expressiveness and thus allows its application to novel problem domains.

Other design space exploration. The *DESERT* tool suite [29] provides model synthesis and constraint-based DSE for DSMLs with structural semantics, using ordered binary decision diagrams for encoding and pruning the design space. Saxena and Karsai [32] present a generic DSE framework extending upon DESERT by supporting arbitrary analysis tools and includes model transformations for mapping design problems to intermediate and low-level formats.

The *OCTOPUS Toolset* [5] uses an intermediate representation for design problem specification and performs DSE using integrated analysis tools. It has been successfully applied to design software-intensive embedded systems [4].

The Gaspard *Framework* [15] is specifically focused on the design of massively parallel embedded systems and uses multilevel modeling where high-level UML models are automatically refined to allow design space exploration to evaluate performance characteristics through simulations.

An efficient design space exploration approach was also presented built on the *FORMULA* framework in [23]. The design problem is described using domain-specific languages and exploration is done with symbolic execution and automatic theorem proving by an SMT solver.

These are all compiled approaches, where the design problems are specified as models and model transformations are applied to derive inputs for third party analysis tools (e.g., SMT or SAT solvers). These analysis tools then perform the exploration and propagate the results back to the original model. However, as the analysis tools are usually used as black boxes when exploring the design space, they cannot be easily extended to support conceptually novel exploration algorithms (e.g., NSGA-II).

[28] presents a framework for the automatic deployment of software components to hardware architecture that uses design space exploration to find deployment alternatives that offer near-optimal reliability characteristics. The design problem consists of architecture models annotated with reliability-relevant properties, while the exploration uses an evolutionary algorithm to find possible alternatives. Unlike our approach, in this work (and also a follow-up paper [27]) global constraints are set as hard selection criteria to prevent the exploration (optimization) of invalid solutions.

Schätz *et al.* [33] developed an interactive, incremental process using declarative transformation rules for driving the exploration. The rules are modified interactively (user guided) to improve the performance of the exploration, while our approach uses genetic algorithms to guide the mutation and crossover operations to find solution models.

Multi-objective optimization in model driven engineering. Multi-objective optimization techniques are widely used in Model Driven Engineering (MDE) field [13, 16, 31, 35]. Recently, Kessentini *et al.* [24] proposed an MDE-based framework for easing the adoption of search-based techniques (such as genetic algorithms) to MDE problems. In this work, the authors describe the logic layer of their MDE-based framework based on previous experiences in using SBSE in hand-crafted applications. However, the realization of the framework is only planned as future work. Moreover, it is not clear how the proposed framework can be adopted for rule-based DSE. Etimaadi and Chaudron [13] proposed the AQOSA tool which uses a model-based approach to evaluate component-based software architecture quality. Similarly to our work, AQOSA uses multi-objective evolutionary algorithms to automatically optimize software architecture design with regard to multiple quality objectives, such as response time, processor utilization, safety, etc.

Despite the popularity of applying search-based techniques for MDE problems, to the best of the authors' knowledge, there is not existing work in the literature dealing with rule-based DSE using multi-objective optimization techniques. Indeed, existing work on Automatic Design Space Exploration (ADSE) using multi-objective optimization techniques are not rule-based DSE, and they are proposed for specific domain problems. For example, Calborean *et al.* [8] proposed recently the FADSE (Framework for ADSE) for DSE of computer systems using different multi-objective search-based algorithms. In this paper, the authors compare the results produced by different genetic algorithms for optimizing the parameters of the Grid ALU Processor (GAP) microarchitecture and the post-link code optimizer GAPtimize. In their framework, application-specific rules that describe existing knowledge can be defined and used as constraints to constrain the DSE process [21]. A similar work is performed by Bolchini *et al.* [7]. Bolchini *et al.* propose a framework based on the multi-objective genetic algorithm NSGA-II for DSE of reliable Field Programmable Gate Array devices.

The contribution of our approach over existing multi-objective optimization for DSE is that our approach is generic so that it can be extended and applied to other domains. Moreover, our approach uses exploration rules to guide the DSE process, while constraints are used to describe the requirements that must be fulfilled. Furthermore, unlike existing approaches which also used the NSGA-II (*e.g.,* [7, 8, 21, 31]), our approaches adapt the constraint-dominate strategy of NSGA-II so that constraints' fulfillment is described as a top-level soft optimization objective, rather than a reward/penalty parameter or as a hard selection criterion. Hence, our approach can even be applied to optimize an existing invalid solution with regard to the top-level objective, which is minimizing constraint violation.

6. CONCLUSIONS

In the paper, we proposed to integrate constrained multi-objective optimization as a search strategy for rule based design space exploration frameworks. In contrast to existing genetic approaches for design space exploration, in our approach, a genetic population consists of rule execution sequences from an initial model to design candidate, constraints are captured by model queries, objectives are calculated from models or rule sequences, while crossover and mutation operations are manipulating rule sequences.

A first key challenge in this setup is that traditional encoding of populations as fixed width bit vectors is unable to represent rule sequences of increasing depth, while it is very difficult or impossible to give a priori upper bounds for feasibility checks. Moreover, unlike in most application scenarios of genetic algorithms, crossover and mutation operations may derive non-executable application sequences as individuals which must not be added to a population. As a consequence, we had to integrate multi-objective optimization techniques to a model-driven rule-based DSE framework as a mapping from a rule based DSE to a genetic algorithm proved to be infeasible.

Our initial experiments demonstrated that multi-objective optimization is an effective strategy for solving rule-based design exploration problems. However, using a randomly synthesized initial population appears to be suboptimal choice in our context. Our future work will aim at investigating other strategies for this purpose.

7. ACKNOWLEDGEMENTS

The authors would like to thank Stéphane Vaucher for his comments on this work. This work was partially supported by the MONDO (EU ICT-611125) project, and the TAMOP-4.2.2.C-11/1/KONV-2012-0001 project which has been supported by the European Union, co-financed by the European Social Fund.

References

[1] Paolo Baldan and Barbara König. Approximating the behaviour of graph transformation systems. In *Proc. ICGT 2002*, volume 2505 of *LNCS*, pages 14–29. Springer, 2002.

[2] Luciano Baresi, Vahid Rafe, Adel T. Rahmani, and Paola Spoletini. An efficient solution for model checking graph transformation systems. *ENTCS*, 213, 2008.

[3] Luciano Baresi and Paola Spoletini. On the Use of Alloy to Analyze Graph Transformation Systems. In *Graph Transformations*, volume 4178 of *LNCS*, pages 306–320. 2006.

[4] Twan Basten, Martijn Hendriks, Nikola Trčka, Lou Somers, Marc Geilen, Yang Yang, Georgeta Igna, Sebastian de Smet, Marc Voorhoeve, Wil van der Aalst, et al. Model-driven design-space exploration for software-intensive embedded systems. In *Model-Based Design of Adaptive Embedded Systems*, pages 189–244. Springer, 2013.

[5] Twan Basten, Emiel van Benthum, et al. Model-driven design-space exploration for embedded systems: The Octopus toolset. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *LNCS*, pages 90–105. Springer, 2010.

[6] Gábor Bergmann, Zoltán Ujhelyi, István Ráth, and Dániel Varró. A graph query language for emf models. In *Theory and Practice of Model Transformations, Fourth International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*, LNCS 6707, pages 167–182. Springer, 2011.

[7] Cristiana Bolchini, Pier Luca Lanzi, and Antonio Miele. A multi-objective genetic algorithm framework for design space exploration of reliable fpga-based systems. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[8] Horia Calborean, Ralf Jahr, Theo Ungerer, and Lucian Vintan. A comparison of multi-objective algorithms for the automatic design space exploration of a superscalar system. In Loan Dumitrache, editor, *Advances in Intelligent Control Systems and Computer Science*, volume 187 of *Advances in Intelligent Systems and Computing*, pages 489–502. Springer Berlin Heidelberg, 2013.

[9] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions Evolutionary Computation*, 6(2):182–197, 2002.

[10] Joachim Denil, Maris Jukšs, Clark Verbrugge, and Hans Vangheluwe. Search-based model optimization using model transformations. Technical report, McGill University, Canada, 2014.

[11] Osmar Marchi dos Santos, Fernando Luís Dotti, and Leila Ribeiro. Verifying object-based graph grammars. *ENTCS*, 109:125–136, 2004.

[12] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch-Lafuente. Heuristic search for the analysis of graph transition systems. In *Proceedings of the Third international conference on Graph Transformations*, volume 4178 of *LNCS*, pages 414–429. Springer-Verlag, 2006.

[13] Ramin Etemaadi and Michel RV Chaudron. A model-based tool for automated quality-driven design of system architectures. In *Proceedings of the 8th European Conference on Modelling Foundations and Applications (ECMFA'12)*, pages 2–5, 2012.

[14] I. Galvao Lourenco da Silva, E. Zambon, A. Rensink, L. Wevers, and M. Akşit. Knowledge-based graph exploration analysis. In *Fourth International Symposium on Applications of Graph Transformation with Industrial Relevance, AGTIVE 2011, Budapest, Hungary*, LNCS 7233, pages 105–120. Springer, October 2011.

[15] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):39, 2011.

[16] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, December 2012.

[17] Ábel Hegedüs, Ákos Horváth, István Ráth, Moisés Castelo Branco, and Dániel Varró. Quick fix generation for DSMLs. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2011*. IEEE Computer Society, 09/2011 2011.

[18] Ábel Hegedüs, Ákos Horváth, István Ráth, and Dániel Varró. A model-driven framework for guided design space exploration. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, Lawrence, Kansas, USA, 11/2011 2011. IEEE Computer Society, IEEE Computer Society.

[19] Ábel Hegedüs, Ákos Horváth, and Dániel Varró. Towards guided trajectory exploration of graph transformation systems. *Electronic Communications of the EASST, Petri Nets and Graph Transformations 2010*, 40, 08/2011 2011.

[20] Ákos Horváth and Dániel Varró. Dynamic constraint satisfaction problems over models. *Software and Systems Modeling*, 11:385–408, 2012 2012.

[21] Ralf Jahr, Horia Calborean, Lucian Vintan, and Theo Ungerer. Boosting design space explorations with existing or automatically learned knowledge. In JensB. Schmitt, editor, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, volume 7201 of *Lecture Notes in Computer Science*, pages 221–235. Springer Berlin Heidelberg, 2012.

[22] H. Jain and K. Deb. An evolutionary many-objective optimization algorithm using reference-point based

non-dominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. *Evolutionary Computation, IEEE Transactions on*, 2013.

[23] Eunsuk Kang, Ethan K. Jackson, and Wolfram Schulte. An approach for effective design space exploration. In *Monterey Workshop*, pages 33–54, 2010.

[24] Marouane Kessentini, Philip Langer, and Manuel Wimmer. Searching models, modeling search: On the synergies of sbse and mde. In *CMSBSE@ICSE*, pages 51–54, 2013.

[25] Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and OmarBen Omar. Search-based model transformation by example. *Software and Systems Modeling*, 11(2):209–226, 2012.

[26] Barbara König and Vitali Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *TACAS*, pages 197–211, 2006.

[27] Indika Meedeniya, Aldeida Aleti, and Lars Grunske. Architecture-driven reliability optimization with uncertain model parameters. *Journal of Systems and Software*, 85(10):2340 – 2355, 2012.

[28] Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011.

[29] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. Constraint-based design-space exploration and model synthesis. In Rajeev Alur and Insup Lee, editors, *Embedded Software*, volume 2855 of *LNCS*, pages 290–305. Springer, 2003.

[30] Arend Rensink. The GROOVE simulator: A tool for state space generation. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *LNCS*, pages 479–485. Springer, 2004. 10.1007/978-3-540-25959-6_40.

[31] Hajer Saada, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Recovering model transformation traces using multi-objective optimization. In *ASE*, pages 688–693, 2013.

[32] Tripti Saxena and Gabor Karsai. MDE-based approach for generalizing design space exploration. In Dorina Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 46–60. Springer, 2010.

[33] B. Schatz, F. Holzl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *Engineering of Computer Based Systems (ECBS)*, pages 173 –182, March 2010.

[34] Ákos Schmidt and Dániel Varró. CheckVML: A tool for model checking visual modeling languages. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *Proc. UML 2003: 6th International Conference on the Unified Modeling Language*, volume 2863 of *LNCS*, pages 92–95, San Francisco, CA, USA, October 20-24 2003. Springer.

[35] Marwa Shousha, Lionel C. Briand, and Yvan Labiche. A uml/spt model analysis methodology for concurrent systems based on genetic algorithms. In *MoDELS*, pages 475–489, 2008.

[36] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltan Szatmári, and Dániel Varró. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, (0):–, 2014.

[37] Yuanyuan Zhang. *Multi-Objective Search-based Requirements Selection and Optimisation.* Phd. thesis, King's College London, UK, 2010.