System Architecture Synthesis for Performability by Logic Solvers

Máté Földiák
Kristóf Marussy
foldiak.mate@edu.bme.hu
marussy@mit.bme.hu
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary

Dániel Varró
daniel.varro@mcgill.ca
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada

István Majzik
majzik@mit.bme.hu
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary

ABSTRACT

In model-based systems engineering, system architectures often have to make compromises to meet hard constraints of functional and extra-functional requirements while optimizing for a target objective. Design space exploration (DSE) techniques have been developed to automatically propose candidate architectures over an extremely large design and configuration space. (1) Meta-heuristic exploration algorithms are often used to provide practical, best-effort solutions for DSE, but they lack any guarantees of completeness or optimality. (2) Logic synthesis based approaches may offer strong theoretical guarantees, but frequently face scalability issues. In the paper, we propose two logic solver-based approaches to evaluate complex design spaces by using partial models in order to find an optimal solution with respect to performability objectives. One approach uses performability analysis as a post-filtering of valid system architecture candidates, while the other approach uses performability analysis for guiding the actual search over partial models. We evaluate both approaches on an interferometry mission architecture case study using view transformations for performability analysis and compare our approach with a well-known DSE framework based on meta-heuristic search.

CCS CONCEPTS

• **Software and its engineering** → Architecture description languages; *Search-based software engineering*; • **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **Theory of computation** → **Discrete optimization**.

KEYWORDS

performability, design-space exploration, logic solver, model generator, partial models

ACM Reference Format:
Máté Földiák, Kristóf Marussy, Dániel Varró, and István Majzik. 2022. System Architecture Synthesis for Performability by Logic Solvers. In *MODELS '22: ACM / IEEE 25th International Conference on Model Driven Engineering Languages and Systems, October 23–28, 2022, Montreal, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

MODELS '22, October 23–28, 2022, Montreal, Canada

ACM ISBN 978-1-4503-XXXX-X/22/10…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Context. Model-based systems engineering has been widely used to manage complexity when designing critical cyber-physical systems (CPS). Architecture modeling languages, such as SysML, Palladio [55], Æmilia [6], and domain specific languages are used to design system architectures, configurations, and deployments on heterogeneous computing and communication infrastructures [79].

Domain-specific standards, such as AUTOSAR [7] in the automotive industry (with more than 500 well-formedness constraints) and ARINC 653 [3] in the avionics domain prescribe strict design rules in addition to the functional requirements (e.g. critical and non-critical components should not be mixed). Fortunately, checking well-formedness constraints over complex system models can be carried out in a scalable way [78].

Furthermore, engineers often need to make major compromise to satisfy possibly contradicting *extra-functional requirements* and functional constraints. In particular, *performability* [77] goals are affected by both performance (which necessitate an efficient allocation of resources) and dependability (which necessitate redundancy) characteristics of the system, while the actual resource allocation can also be limited by design rules or functional constraints.

However, the mathematically precise analysis of system-level extra-functional properties [31] necessitates to automatically derive *analysis models* in various modeling formalisms (such as process algebra [32], Markov chains and Stochastic Petri nets [58, 61]) by complex model transformations [18, 29, 30, 35, 76] from system models. Moreover, sophisticated backend solvers (such as PEPA [32], PRISM [49]) need to be used to actually carry the analysis of such extra-functional properties and then back-annotate the results.

Problem statement. The large number of possible system configurations each with highly varying extra-functional characteristics poses a major challenge to find the most suitable system architecture in an early design phase [20]. Engineers can manually inspect only a handful subset of candidate architectures. Therefore, various automated *design space exploration* (DSE) tools have been developed *for system architecture synthesis* to assist in finding viable candidate architectures that satisfy all functional and extra-functional constraints while optimizing for a target objective. DSE tools tackling this challenge are broadly classified into two categories:

(A) *(Meta-)heuristic* techniques, such as genetic algorithms or multi-objective optimization [1, 6, 13, 27, 28, 55], can support a wide variety of analyses directly inside the DSE process to derive near-optimal design candidates. However, they do not guarantee a complete (exhaustive) enumeration of the design space and the optimality of the generated candidates [43]. Moreover, encoding hard constraints either requires custom soft constraints, objective functions and mutation operators or it could significantly degrade the performance or scalability of the exploration [72].

(B) *Logic solver* based DSE techniques (e.g., [38, 41]) have guaranteed *soundness* and *completeness*. They usually allow encoding complex logical hard constraints and logical formulas or graph patterns [15, 48, 71, 78] and may provide an explanation when the synthesis task is unsatisfiable. However, purely logical constraints cannot capture most extra-functional requirements that rely on an external numerical solver to analyze. Thus, solvers have to be specifically extended for optimization tasks, such as in [11, 51] to handle both logical and numerical constraints. Unfortunately, for complex extra-functional analysis tasks, such optimizing solvers are often unavailable. In such cases, a *post-filtering based* DSE approach can be used [25] where (i) logic solvers derive well-formed candidate architectures, which are then (ii) mapped to analysis models and finally (iii) investigated by dedicated analysis tools one by one to select good design candidates (without optimality guarantees).

Recently logic solvers based on partial models have offered a scalable solution for graph constraints [71], attribute constraints [68] or scope constraints [56] by abstract graph reasoning [64, 65] along refinement. Abstract interpretation has been used for performance analysis in [33], but without incorporating functional constraints. However, *lifting the analysis of both functional and extra-functional properties from concrete models to abstract models* during design space exploration has remained an *open challenge*.

Objectives and contributions. This paper provides two logic-solver based architecture synthesis techniques to automatically derive system models that (a) satisfy both functional and extra-functional constraints and (b) optimize the system architecture with respect to *performability objectives*. The first approach applies *post-filtering on valid design candidates derived by a graph solver* to carry out performability analysis using an external stochastic analyzer. The second approach carries out *abstract performability analysis on partial design candidates by introducing conservative approximations* to fill uncertain or incomplete information when calling the external analysis tool. The specific contributions of the paper include:

- As a conceptual basis, we extend the *partial model* [26, 71] formalism to incorporate performability metrics (Section 3.2).
- We introduce a *view transformation over partial models of system architectures* to derive *Continuous-Time Markov Chain* (CTMC) reward models that conservatively approximate extra-functional properties for future design decisions. In particular, we extend a graph solver based on partial models with *extra-functional propagation* (Section 3.3) operators to incorporate CTMC analysis results into the partial models.
- We provide a *prototype tool implementation* on top of open-source software tools, namely, the VIATRA Generator [69] graph solver and the PRISM [49] stochastic model checker. The tool is available in the accompanying artifact at https://doi.org/10.5281/zenodo.6974248.
- We *evaluate the performance of the two approaches* (Section 4) and compare it to a meta-heuristic synthesis approach in the context of a case study from the NASA JPL [37].

2 SYSTEM ARCHITECTURE SYNTHESIS

2.1 Motivating case study

In our paper, we study the *architecture synthesis challenge* in the context of a complex case study introduced in [37] by NASA JPL engineers who used the MOMoT DSE tool [27] which exploits meta-heuristic search to create candidates for satellite constellations that can collect scientific data for early mission planning.

A satellite mission architecture consist of a ground station equipped to receive data from satellites. Each satellite has a type (i.e., 3U, and 6U CubeSat or small satellites), they can be equipped with various communication subsystems operating at different bandwidth (Ka, X or UHF band), and optionally, with an interferometry payload to make measurements.

Satellites need to forward measurement data to the ground station either directly or indirectly via another satellite, thus an appropriate communication topology needs to be designed. Each satellite must use one communication subsystem to downlink data and it can optionally be equipped with another one for relaying purposes. The ground station must be reachable from every satellite, and the topology of the communication must be acyclic.

The main objective is to maximize the *scientific coverage*

$$c_t(n) = \left(1 - \frac{2}{n}\right) + 0.05\frac{t}{3}, \tag{1}$$

i.e., the amount of data gathered by the n installed payloads and successfully relayed to the ground station, of the mission given a fixed observation time t and a component budget.

To study the effects of failure processes on the architecture synthesis, we assume that exponential failure rates are attached to the components as reliability information. The *performability metric* computes the expected coverage given the occurrence of component failures within the observation time window, i.e.,

$$\mathbb{E}[C_t(n)] = \sum_{i=2}^{n} \mathbb{P}\left\{\begin{matrix} i \text{ payloads have working} \\ \text{downlink at time } t \end{matrix}\right\} \cdot c_t(i) \tag{2}$$

2.2 High-level overview

We propose two model synthesis approaches to generate system architecture candidates subject to hard logical (functional and extra-functional) constraints while optimizing for performability metrics. Conceptually, our approach combines partial model refinement [56, 68] with view transformations to derive a performability model investigated by performability analysis. Figure 1 illustrates the architecture of the proposed approaches.

As inputs, our system architecture synthesis relies on

- an **(A) initial partial model** to be extended, which is either the most general (maximally uncertain) partial model, or contains the architecture elements that are required to be present in all generated solutions;

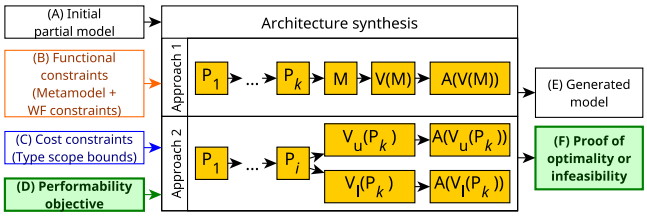

Figure 1: Overview of architecture synthesis

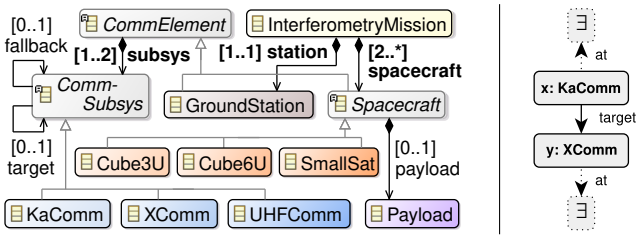

Figure 2: Metamodel and sample error pattern for case study

- **(B) functional constraints** to restrict valid system models to be compliant with a metamodel and a set of well-formedness (WF) constraints;
- **(C) cost constraints** expressed as type scope bounds or linear inequalities to provide an upper bound on the size of the model; and
- a **(D) performability objective** captured either as an *optimization goal* or a *threshold thr* to be satisfied.

Our framework derives either an *optimal* system architecture that has the highest obtainable performability value, or a *near-optimal* candidate architecture with performability metric over some user-defined *threshold thr*. In either case, each derived candidate satisfies all WF constraints and stays within the cost bounds.

- If the problem is *feasible*, a **(E) generated design candidate** will be derived as output.
- If no near-optimal solution exists above the specified threshold, the synthesis task is *infeasible* and a **(F) proof of infeasibility** will be provided. For optimization problems, a **(F) proof of optimality** can be provided.

Our approach is *parametric* in the model transformation $\mathcal{V}$ that derives the performability analysis models from the design candidates amenable to performability analysis. While we provide such a transformation in the paper, our overall synthesis can be combined with other existing performability analysis approaches. We provide two approaches by combining a graph solver with partial model refinement with transformation-driven performability analysis:

- **Approach 1:** Whenever the generator obtains a valid concrete model, performability analysis is executed as *post-filtering* to further restrict the generated candidates.
- **Approach 2:** Partial view model transformations $\tilde{\mathcal{V}}_\ell(P)$ and $\tilde{\mathcal{V}}_u(P)$ are used to conservatively estimate lower and upper bounds for performability metrics by initiating performability analysis on partial design candidates.

The remainder of this section introduces further details for **Approach 1**, while section 3 elaborates on **Approach 2**.

2.3 Hard constraints for system models

2.3.1 Metamodel and well-formedness constraints. We will rely on a general-purpose or domain-specific *systems modeling language* to represent system model serving as candidate architectures. We assume that such languages are defined by (1) a metamodel to capture the vocabulary of the system modeling language with classes and references, while (2) consistent (valid) systems models also have to satisfy *well-formedness* (WF) constraints. On a technical level, we rely on metamodels of Eclipse Modeling Framework (EMF) [74], and WF constraints (and constraint validations) implemented using the VIATRA Query [78] language and technology.

Similarly to [68], we rely on first-order logic (FOL) to uniformly formalize (a) the constraints associated with the systems modeling language, (b) additional design rules, and (c) extra functional requirements. This is the case for many constraint languages widely used in model-driven engineering, such as OCL constraints [70] and graph patters [78]. For example, if the system modeling language can describe the physical and functional system architecture and their allocations, FOL constraints can capture whether all functions are allocated to physical components in a valid manner.

Definition 2.1. A *metamodel* is a FOL signature $\langle \Sigma, \alpha \rangle$, where the set of *symbols* Σ includes unary class C_i and existence ε and binary *reference* R_j and equivalence $\sim$ symbols, and $\alpha \colon \Sigma \to \mathbb{N}$ is the *arity* function $\alpha(C_i) = \alpha(\varepsilon) = 1$, $\alpha(R_j) = \alpha(\sim) = 2$.

Each class symbol $C_1, \ldots, C_c \in \Sigma$ corresponds to an *EClass* and each reference symbol $R_1, \ldots, R_r \in \Sigma$ corresponds to an *EReference*. We use the existence symbol ε to reason about whether a given component is present in the architecture model and the equivalence symbol $\sim$ to reason about whether two components may be *merged* into one or a component may be *split* into different instances.

A *theory* $\mathcal{T}$ is a set of FOL predicates $\mathcal{T} = \{\varphi_1, \ldots, \varphi_k\}$ that capture the hard constraints for design candidate as *error patterns*, i.e., valid candidate architectures should satisfy none of φ_i.

Example 2.2. Figure 2 shows the metamodel for the case study adapted from [56]. We extended the metamodel with a fallback reference to model redundant communication links between satellites.

On the right side of the figure, we depict the error pattern

$$\varphi = \exists x \colon \mathrm{KaComm}(x) \wedge \exists y \colon \mathrm{XComm}(y) \wedge \mathrm{target}(x, y),$$

which forbids connecting incompatible communication subsystems.

2.3.2 Cost constraints. The compromises between performance and dependability characteristics are interpreted within the context of some *cost bounds* (budget) which restrict the number and the type of components used in candidate architectures. First, we determine a *model scope* (model size) within which near-optimal or optimal design candidates are sought.

Moreover, we use *linear inequality* bounds [56] to encode cost constraints in the form of $\sum_{C_i \in \Sigma} cost(C_i) \cdot \#C_i \leq U$, where $cost(C_i)$ is the cost of a component of type C_i, $\#C_i$ is the number of objects of class C_i, and U is the available budget. These inequalities generalize type scope bounds [38] of the form $L \leq \#C_i \leq U$, which refer to the number of instances of a given class.

In the following, we will assume that all WF and cost constraints are hard constraints, i.e., they are part of the theory $\mathcal{T}$.

2.4 Partial models and refinement

2.4.1 Partial models. We recap the concept of *partial models* to explicitly encode unknown or uncertain parts of a model [71] (i.e., design decisions yet to be made). We use 3-valued logic, where the usual 1 (true) and 0 (false) values are extended with a third 1/2 (unknown) value that encodes uncertainty.

Definition 2.3 ([71]). A *partial model* is a FOL structure $P = \langle O_P, \mathcal{I}_P \rangle$ over signature $\langle \Sigma, \alpha \rangle$, where O_P is a finite set of *objects*, and $\mathcal{I}_P$ is an *interpretation function* with $\mathcal{I}_P(\varsigma) : O_P^{\alpha(\varsigma)} \to \{1, 0, 1/2\}$ for all symbols $\varsigma \in \Sigma$.

Partial model objects o with $\mathcal{I}_P(\varepsilon)(o) = \mathcal{I}_P(\sim)(o, o) = 1/2$ (potentially existing and possibly equal with themselves) are *multi-objects* [56], which may represent multiple concrete model elements in regular instance models.

FOL predicates φ can be evaluated on a partial model P yielding $[\![\varphi]\!]_P \in \{1, 0, 1/2\}$ according to the rules of 3-valued logic [71]. A partial model P is *logically consistent* with the *theory* $\mathcal{T}$, written as $P \models \mathcal{T}$, if $[\![\varphi_i]\!]_P \neq 1$ for all error predicates $\varphi_i \in \mathcal{T}$.

A partial model $M = \langle O_M, \mathcal{I}_M \rangle$ is *concrete* if $\mathcal{I}_M$ contains only 1 and 0 values. Concrete models correspond to regular *instance models* that represent candidate architectures.

Example 2.4. Figure 3 shows three partial models P_1, P_2, P_3. References with 1 and 1/2 logic values are depicted as solid and dashed lines, respectively, while 0 references are omitted. For example, in P_1, $\mathcal{I}_{P_1}(\sim)(new_X, new_X) = \mathcal{I}_{P_1}(\text{subsys})(S_2, new_X) = 1/2$ and $\mathcal{I}_{P_1}(\text{subsys})(S_2, C_2) = 1$. Objects with dashed outlines have uncertain existence, i.e., $\mathcal{I}_{P_1}(\varepsilon)(new_X) = 1/2$, which makes new_X into a multi-object representing all XComm instances to be added.

2.4.2 Partial model refinements. Partial model refinements gradually add information to partials models to reduce uncertainty, as a designer would incorporate design decisions into the model. Our approach derives candidate architectures along such refinements.

The *information ordering* $\succcurlyeq$ of logic values [71] specifies that 1/2 may be refined into either 1 or 0, while other logic values must stay unchanged, i.e., $(X \succcurlyeq Y) \Leftrightarrow (X = 1/2) \vee (X = Y)$.

Definition 2.5 ([56]). The function $abs : O_Q \to O_P$ is a *refinement* from partial model $\langle O_P, \mathcal{I}_P \rangle$ to the partial model $\langle O_Q, \mathcal{I}_Q \rangle$, written as $P \succcurlyeq_{abs} Q$, if (i) for all $p \in O_P$ with $\mathcal{I}_P(\varepsilon)(p) = 1$, we have some $q \in O_Q$ with $abs(q) = p$; and (ii) for all $\varsigma \in \Sigma$ and $q_1, \ldots, q_{\alpha(\varsigma)}$, $\mathcal{I}_P(\varsigma)(abs(q_1), \ldots, abs(q_{\alpha(\varsigma)})) \succcurlyeq \mathcal{I}_P(q_1, \ldots, q_{\alpha(\varsigma)})$.

We may use $P \succcurlyeq Q$ to simplify notation.

Graph solvers derive consistent concrete models M via a sequence of refinements $P_0 \succcurlyeq P_1 \succcurlyeq \cdots \succcurlyeq M$. It can be shown [71] that inconsistencies can be detected early during refinement, i.e., $[\![\varphi_i]\!]_{P_j} = 1$ implies $[\![\varphi_i]\!]_M = 1$ for any error pattern $\varphi_i \in \mathcal{T}$. Such inconsistent partial models can be discarded by *backtracking*, reducing the number of partial models explored during model synthesis.

Example 2.6. Figure 3 shows two refinements $P_1 \succcurlyeq_{abs_1} P_2$ and $P_2 \succcurlyeq_{abs_2} P_3$. In P_2, $abs_1(L_1) = abs_1(new_L) = new_L$, i.e., the multi-object new_L representing all new Payload instances was *split* to create a new Payload L_1. As a result, $\#new_L$, the number of remaining Payload objects allowed by the cost constraints in $\mathcal{T}$, also decreased from 2 to 1. In P_3, abs_2 is the identity function, i.e., no new objects were created. However, the new reference $\mathcal{I}_{P_3}(\text{fallback})(C_2, C_1) = 1$ violates a WF constraint φ_{loop} that forbids cycles in the communication topology ($[\![\varphi_{loop}]\!]_{P_3} = 1$), which makes P_3 inconsistent.

2.5 Transformations for performability analysis

Performability requirements necessitate a compromise between the performance and dependability characteristics of the system [77]. They can evaluated by stochastic modelling, such as Continuous-Time Markov Chain (CTMC) reward models, where the CTMC describes the dependability attributes of the system, while the achieved performance is captures by the reward structure. In *architecture based* performability analysis [10, 45], a CTMC *analysis model* A is derived from the architecture model M by a (unidirectional) *view transformation* $A = \mathcal{V}(M)$. The transformation $\mathcal{V}$ constructs A by composing performability model fragments that describe the extra-functional characteristics of each of the component instances in M. Afterwards, a *stochastic analysis tool* $\mathcal{A}$ analyzes A to obtain a *performability metric* value $m = \mathcal{A}(A)$ by numerical solution or simulation. As a result, the performability of architecture M is evaluated as $\mathcal{A}(\mathcal{V}(M))$.

In this paper, we encode CTMC analysis models in the PRISM language [49] and solve them with the PRISM model checker tool.

Example 2.7. Figure 4 shows a concrete model M for a candidate architecture, a visual representation of the analysis model $\mathcal{V}(M)$, and the corresponding PRISM code for a CTMC reward model.

We introduce exponentially distributed failure times for satellites (Spacecraft) and communication subsystems (CommSubsys). Each SmallSat has a mean time to failure (MTTF) of 70 hours (failure rate of $\lambda_{\text{SmallSat}} = \frac{1}{70}\frac{1}{\text{hour}}$), while each XComm connected to a satellite has an MTTF of 13 hours ($\lambda_{\text{XComm}} = \frac{1}{13}\frac{1}{\text{hour}}$). Based on these reliability attributes and c_t from (1), the performability analysis $\mathcal{A}(\mathcal{V}(M))$ determines the value of the performability metric (2).

The analysis model $\mathcal{V}(M)$ is depicted as a Static Fault Tree (SFT). The *basic events* S1, S2, S3, C1, C2, C3, Cg correspond to the failures of satellites and communication subsystem. The Spacecraft S*i* can successfully transmit, represented by the *gate* Tr*i*, if the target OR fallback of its transmitting CommSubsys can receive the transmitted data. It can operate and gather data (On*i*) if itself (S*i*) and its transmitting CommSubsys is fault-free and it can transmit the acquired data (Tr*i*). Lastly, a CommSubsys (C*j*) may receive data (Rc*i*) if both it and its spacecraft are operational. The performance metric can be computed from the number of operational satellites online that are equipped with a Payload (On1, On2, On3). Note that the top event corresponds to the *correct* operation of the system to facilitate computing the performability metric, as opposed to the more usual *faulty* top event.

In the PRISM model, there is a module that contains a bool variable for each basic event. The stochastic commands, starting with the [] symbol, set the variables to false whenever a failure occurs according to the failure rate λ. We omitted the stochastic command for Cg, because we model CommSubsys on the ground as always fault-free. The gates of the fault tree were mapped to formula expressions. The rewards structure "utility" measures the system performance according to C_t in (2).

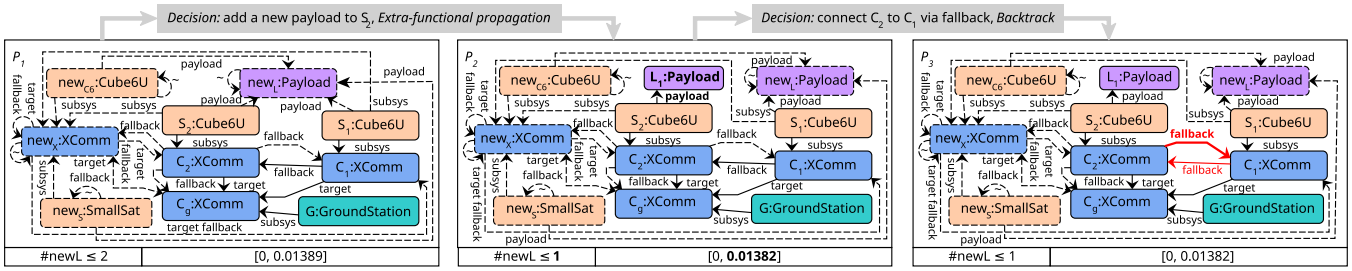

Figure 3: Example refinements of partial models: $P_1 \succcurlyeq P_2$ and $P_2 \succcurlyeq P_3$

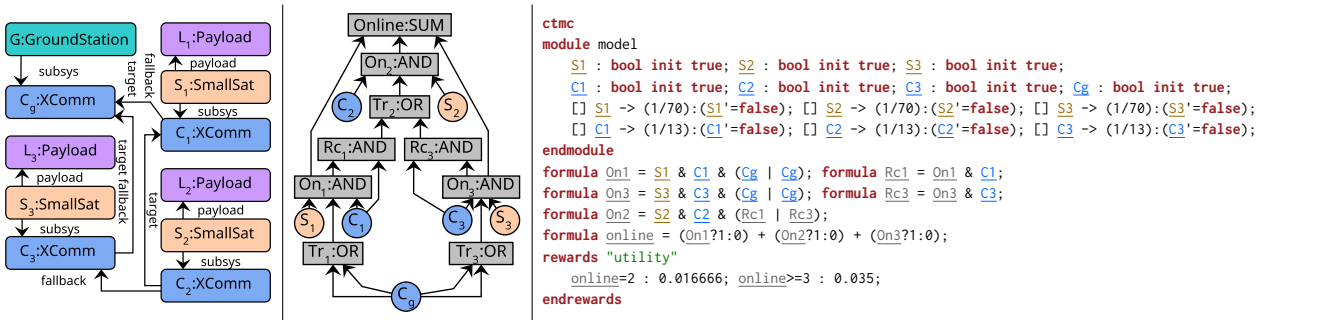

Figure 4: Architecture (instance) model, fault tree and corresponding PRISM analysis model

To obtain the expected value of C_t as the performability metric for a *mission time* of $t = 1$ hour, we run the *immediate reward* query R{"utility"}=?[I = 1] in PRISM.

3 ABSTRACT PERFORMABILITY ANALYSIS

3.1 Exploration overview

Figure 5 shows an overview of architecture synthesis for **Approach 2**, where abstract performability analysis is fully integrated into state space exploration. Our framework explores a *state space* of partial models along refinements to derive either a feasible concrete solution or a proof of infeasibility consisting of partial models that were pruned during the exploration. Novel components proposed in this paper were highlighted in **bold** in the figure. In particular, the generator has the following components:

(1) Decision and unit propagation rules introduce new information into the partial model. A *decision* step explores a possible design decision (i.e., the addition of a new component instance or a new connection between existing components). *Unit propagation* rules incorporate the logical consequences of WF constraints after a decision according to the semantics of 3-valued logic following the set of rules in [71], which were proven to be sound and complete. We avoid executing decisions that violate cost constraints by *object scope analysis* [56], which compares the size of the model to the type scope bounds.

(2) Constraint evaluation uses an *incremental graph query engine* [78] to *over-* and *under-approximate* the WF constraints on the partial model [71]. Partial models that surely violate some WF constraint are *inconsistent* and are discarded by backtracking.

(3) State coding based on *graph shapes* [64] avoids repeatedly exploring isomorphic partial design candidates.

(4) Extra-functional propagation is a new component to ensure the synthesis of near-optimal or optimal candidates: The **(5) PM to analysis model transformation** constructs analysis models from partial models to *under-* and *over-approximate* the achievable values of the performability metric by calling PM view transformations either (**Approach 1**) when a concrete model is reached or (**Approach 2**) after each decision step. We evaluate the analysis method by the PRISM Model Checker [49] as an **(6) External analysis tool**. By **(7) Interpreting the analysis results**, refined bounds for the achievable values of the performability metric are obtained for the PM. If this range falls outside the prescribed threshold, the PM is *infeasible* and is pruned by backtracking.

These components are coordinated by a **(8) State space exploration** strategy that explores the potential refinements of the initial PM according to a *best-first* heuristic search. The heuristic is computed from the number of remaining potential WF constraint violations in the PM and the lower bound of the achievable performability objective to prioritize design candidates with high performability values. To ensure completeness, refinements are explored exhaustively until either each PM is pruned by backtracking or a feasible solution is found. The pruned PMs serve as a *formal proof* of infeasibiltiy or optimality.

For space considerations, we provide further details exclusively for the novel components of our framework in the sequel.

3.2 Partial models for performability analysis

First, we extend partial models to track the required or achievable value of the performability metrics in architecture synthesis.

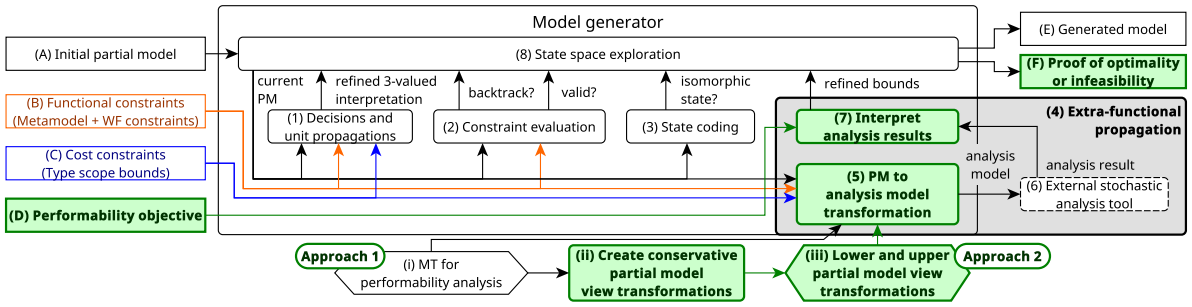

Figure 5: Overview of architecture synthesis with abstract performability analysis

Definition 3.1. A *partial model with performability objective* is a triple $P = \langle O_P, \mathcal{I}_P, \mu_P \rangle$, where $\langle O_P, \mathcal{I}_P \rangle$ is a partial model (see Definition 2.3), and $\mu_P \subseteq \mathbb{R}$ is a real interval.

Definition 3.2. Function *abs* is a *refinement* from $P = \langle O_P, \mathcal{I}_P, \mu_P \rangle$ to $Q = \langle O_Q, \mathcal{I}_Q, \mu_Q \rangle$ if (i) it is a refinement between the partial models $\langle O_P, \mathcal{I}_P \rangle$ and $\langle O_Q, \mathcal{I}_Q \rangle$ (Definition 2.5), and (ii) $\mu_Q \subseteq \mu_P$, i.e., refinement cannot relax the performability objective.

The interval μ_P contains the desired values of the performability objective. A concrete model $M = \langle O_M \mathcal{I}_M, \mu_M \rangle$ is *performability consistent* with the view transformation $\mathcal{V}$ if $\mu_M = \{\mathcal{A}(\mathcal{V}(M))\}$ is the singleton interval containing the performability metric of M determined by analysis. Therefore, if $P \succcurlyeq M$ and M is performability consistent, then we must have $\mathcal{A}(\mathcal{V}(M)) \in \mu_P$.

M is *consistent* if it is both logically consistent with the theory $\mathcal{T}$ (Section 2.4.1) and performability consistent with $\mathcal{V}$. During model generation, we aim to obtain concrete models M that are both consistent and refinements of the initial partial model P_{init}.

The initial partial model $P_{init} = \langle O_{P_{init}}, \mathcal{I}_{P_{init}}, \mu_{P_{init}} \rangle$ may be set to contain model elements that must be present in all design candidates. Alternatively, we may set P_{init} to the *most general partial model* for the signature $\langle \Sigma, \alpha \rangle$ according to [56]:

- We let $O_{P_{init}}$ contain a new multi-object new_{C_i} for each *concrete* class symbol $\mathrm{C}_i \in \Sigma$ with $\mathcal{I}_{P_{init}}(\mathrm{C}_i)(new_{\mathrm{C}_i}) = 1$ and $\mathcal{I}_{P_{init}}(\varepsilon)(new_{\mathrm{C}_i}) = \mathcal{I}_{P_{init}}(\sim)(new_{\mathrm{C}_i}, new_{\mathrm{C}_i}) = 1/2$; and
- for any other symbol $\varsigma \in \Sigma$ and objects $o_1, \ldots, o_{\alpha(\varsigma)} \in O_{P_{init}}$, we set $\mathcal{I}_{P_{init}}(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)}) = 1/2$.

In synthesis problems with a threshold *thr* for the performability objective, we set $\mu_{P_{init}} = [thr, +\infty)$ to only obtain refinements as solutions that are over the threshold. By setting this threshold *thr* to 0, one can initiate state space exploration for a regular optimization problem aiming to find the optimal design candidate.

3.3 Extra-functional propagation

In the extra-functional propagation step, we obtain new bounds of the achievable values of the performability metric for some partial model $P = \langle O_P, \mathcal{I}_P, \mu_P \rangle$ being considered. We incorporate the new information into the partial model by a refinement $P \succcurlyeq P'$.

Extra-functional propagation uses *view transformations (VT) over partial models* to derive (concrete) performability analysis models from partial model P. *Upper* $\tilde{\mathcal{V}}_u$ and *lower* $\tilde{\mathcal{V}}_\ell$ partial model VTs are conservative approximations of view transformation $\mathcal{V}$.

Definition 3.3. Upper $\tilde{\mathcal{V}}_u$ and lower $\tilde{\mathcal{V}}_\ell$ partial model VTs *conservatively approximate* the view transformation $\mathcal{V}$ over the signature $\langle \Sigma, \alpha \rangle$ and the theory $\mathcal{T}$ if for all partial models P over $\langle \Sigma, \alpha \rangle$ and concrete models M with $P \succcurlyeq M$ and $M \vDash \mathcal{T}$, we have $\mathcal{A}(\tilde{\mathcal{V}}_\ell(P)) \leq \mathcal{A}(\mathcal{V}(M)) \leq \mathcal{A}(\tilde{\mathcal{V}}_u(P))$.

Estimates from partial model VTs shrink along refinement: if $P \succcurlyeq Q$, then $\mathcal{A}(\tilde{\mathcal{V}}_\ell(P)) \leq \mathcal{A}(\tilde{\mathcal{V}}_\ell(Q)) \leq \mathcal{A}(\tilde{\mathcal{V}}_u(Q)) \leq \mathcal{A}(\tilde{\mathcal{V}}_u(P))$. In particular, for a concrete and consistent model M, we always have $\mathcal{A}(\tilde{\mathcal{V}}_\ell(P)) = \mathcal{A}(\mathcal{V}(M)) = \mathcal{A}(\tilde{\mathcal{V}}_u(M))$.

Given a pair of conservative partial model VTs $\tilde{\mathcal{V}}_\ell, \tilde{\mathcal{V}}_u$, then *extra-functional propagation* $P = \langle O_P, \mathcal{I}_P, \mu_P \rangle \succcurlyeq P' = \langle O_P, \mathcal{I}_P, \mu'_P \rangle$ derives a new partial model with $\mu'_P = \mu_P \cap [\mathcal{A}(\tilde{\mathcal{V}}_\ell(P)), \mathcal{A}(\tilde{\mathcal{V}}_u(P))]$.

Thanks to conservativeness, the propagation does not remove any potential valid design candidates: if $P \succcurlyeq M$ for some consistent concrete model M, then we also have $P' \succcurlyeq M$.

The role of the refined interval μ'_P in the design candidate synthesis process is twofold: Firstly, the lower bound of the interval is *incorporated as a heuristic* into the best-first state space exploration to prioritize partial models where a high value of the performability metric is likely to be achievable. Secondly, partial design candidates with $\mu'_P = \emptyset$ can be *pruned* outright, since they are unable to achieve a performability metric in the desired interval μ_P.

Example 3.4. In Figure 3, extra-functional propagation is applied after the decision step which insert a new Payload instance in refinement $P_1 \succcurlyeq P_2$, which tightens the upper bound of the interval μ_{P_2} from 0.01389 to 0.01382.

3.4 Conservative concretization with relaxation

In **Approach 2**, the partial model VTs $(\tilde{\mathcal{V}}_\ell, \tilde{\mathcal{V}}_u)$ may be derived from the concrete view transformation $\mathcal{V}$ in various domain- or problem-specific ways. However, the choice of these VTs may have a significant impact on the performance of DSE. If the estimates $\mathcal{A}(\tilde{\mathcal{V}}_\ell(P))$ and $\mathcal{A}(\tilde{\mathcal{V}}_u(P))$ are not tight enough, the generator will have to explore a larger fraction of the state space, because it cannot detect the infeasibility of partial solutions early for pruning. Therefore, in this section, we provide guidelines for designing $\tilde{\mathcal{V}}_\ell, \tilde{\mathcal{V}}_u$.

We introduce the *upper* C_u and *lower* C_ℓ *conservative concretization* operators that derive concrete models $M_u = C_u(P)$ and $M_\ell = C_\ell(P)$ as refinements $P \succcurlyeq M_u$ and $P \succcurlyeq M_\ell$, respectively. We *relax* the functional WF constraints and cost constraints of the theory $\mathcal{T}$. In general, M_u and M_ℓ might not be consistent w.r.t. $\mathcal{T}$, but we may nevertheless execute the view transformation $\mathcal{V}$ on them to obtain

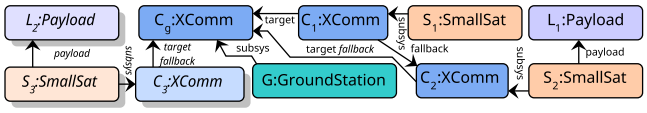

Figure 6: Example upper conservative concretization

performability analysis models. Then we set $\tilde{\mathcal{V}}_u(P) = \mathcal{V}(C_u(P))$ and $\tilde{\mathcal{V}}_\ell(P) = \mathcal{V}(C_\ell(P))$. Therefore, in order to make $\tilde{\mathcal{V}}_\ell$ and $\tilde{\mathcal{V}}_u$ conservative, we must ensure that the performability metrics associated with the concrete (but not necessarily consistent) models $M_u = C_u(P)$ and $M_\ell = C_\ell(P)$ over- and under-approximate, respectively, the performability metrics associated with any concerete and consistent refinement M of P, i.e.,

$$\mathcal{A}(\mathcal{V}(C_\ell(P))) \leq \mathcal{A}(\mathcal{V}(M)) \leq \mathcal{A}(\mathcal{V}(C_u(P))).$$

Informally, given a component dependency model, one may construct C_ℓ and C_u if $\mathcal{V}$ is *monotonic* in the following sense: the value of the performability metrics is not decreased by

(1) adding new components without new dependencies;
(2) replacing a component with a more reliable one; or
(3) replacing common causes of failure with independent components providing the same service

This is the case, e.g., for static fault trees (SFT) [40] as dependability models where the performability metric computed according to the number of operational components, because STFs cannot contain negations. Therefore, the assumptions hold for our interferometry mission architecture case study.

If the assumptions (1)–(3) hold, we may obtain $M_u = C_u(P)$ by

- adding new components to P up to cost bounds allowed by P (even by violating WF constraints) according to (1);
- if there are any unmet required or optional dependencies in the model, satisfing them with new independent components (even violating WF and cost constraints) with the highest possible reliability according to (2) and (3); and
- concretizing the model by replacing any remaining $1/2$ values in $\mathcal{I}_{M_u}$ with 0.

For the lower concretization $M_\ell = C_\ell(P)$, we simply set $1/2$ values in $\mathcal{I}_{M_\ell}$ to 0, leaving any component dependencies unmet.

Example 3.5. Figure 6 shows the upper conservative concretization $C_u(P_2)$ of P_2 from Figure 3. Objects and references added by concretization are shown in *italic*.

Since cost constraints enforce $\#new_{\mathsf{L}} \leq 1$ in P_2 (at most one new Payload can be added), C_u has added a new instance L_2. Instead of connecting L_2 to the existing S_1, it also added a new SmallSat S_3 (the most reliable available Spacecraft) and a new XComm C_3 (the most reliable available CommSubsys), potentially violating other cost constraints. The target and fallback references of C_3 are connected directly to C_g, the most reliable path to the GroundStation. Likewise, the fallback reference of C_2 was connected to C_g.

The concretization rules C_u above that create most reliable, but not necessarily consistent architecture possible from a partial model are to be provided as a parameter for the generator.

3.5 Soundness and completeness

Soundness. Our architecture synthesis methods are *sound*, i.e., if a near-optimal design candidate M is output, it satisfies the theory $\mathcal{T}$ of functional WF constraints and cost constraints ($M \vDash \mathcal{T}$), as well as the prescribed threshold ($\mathcal{A}(\mathcal{V}(M)) \geq thr$).

For optimization problems, we guarantee δ-optimality, i.e., for any concrete and consistent $P \trianglerighteq M'$, we have $\mathcal{A}(\mathcal{V}(M)) + \delta \geq \mathcal{A}(\mathcal{V}(M'))$. This avoids numerical instability issues, where the external stochastic analysis tool $\mathcal{A}$ would return noisy (within an error bound of δ) analysis results for solutions with equal performability metrics due to the underlying numerical computations.

Completeness. The *complete* enumeration of design candidates within the (finite) *model scope* spanned by the cost constraints relies on the completeness of the decision and propagation rules in [71]. The conservativeness of the partial model view transformations ensures that we do not prune any potential solutions even after extra-functional propagation, which maintains completeness.

If a near-optimal design candidate is sought, the synthesis stops at the first feasible solution. In optimization problems, we adapt a technique from optimizing SMT solvers [11, 51]. After a solution M^* is found, we update each remaining partial model P in the state space by $\mu'_P = \mu_P \cap [\mathcal{A}(\mathcal{V}(M^*)) + \delta, +\infty)$, i.e., any further solution must improve on the performability metric of M^* by at least δ.

Infeasible problems. If no feasible solution is found, our approach *degrades gracefully* by retrieving partial models that were pruned during extra-functional propagation. Such partial models represent design candidates that (a) may have consistent refinements, i.e., were not pruned during constraint evaluation, and (b) got closest to the desired performability objective value before pruning. By covering the design space of consistent models, these pruned partial candidates constitute a *formal proof of infeasiblity*.

4 EVALUATION

We carried out an experimental evaluation to answer the following research questions:

(RQ1) How does the performance of our approaches compare to meta-heuristic techniques to derive optimal architectures?
(RQ2) How do various exploration steps contribute to exploration time when generating near-optimal system architecture candidates of increasing size?

4.1 Measurement setup

4.1.1 Case studies. We aim to evaluate the performance of various approaches proposed in the paper in the context of the complex satellite case study defined in [37]. In the original setting (referred to as **Sat**), satellite constellations are constructed without redundancy, i.e. communication subsystems can only transmit data to a single target communication subsystem.

In addition, we extended the original challenge to include redundancy in the communication topology by adding an alternative outgoing link to each communication subsystems as a fallback. This redundant setup necessitates the extra well-formedness constraint requiring that the target and fallback links need to connect to different communication subsystems located on different satellites. This extended setup is referred to as **Sat+fb**.

Various components used in architecture synthesis in both cases have a predefined *price*. A SmallSat costs \$2,900K, a Cube6U costs \$650K while a Cube3U costs \$150K. In addition, a communication subsystem costs \$100K and each satellite is required to have at least one of such communication subsystem. Each payload allocated to satellite costs \$50K. While the original study [37] used a non-linear cost function, we turn it into a linear form by taking the maximum possible costs for each component.

As further constraints on total cost and model size, we used the following restrictions: (1) *Small models* with at least 8 (minimal functional size) and up to 10 components (excluding the root element) had a cost limitation of \$5,000K, (2) *Medium models* with at least 8 and up to 20 components had an upper cost limit of \$9,000K, and (3) *Large models* up to 30 components had a cost limitation of \$15,000K. These cost limits are aligned with the actual costs of satellite architectures reported in [37], and large models are of equivalent size with the largest models derived by MOMoT in the original study (with a runtime of 6 hours).

4.1.2 Compared approaches. As a baseline for our experiments, we have selected **MOMOT**, which is the DSE tool using meta-heuristic search used also in the original study [37]. As in [37], we replaced the default settings of the eMOEA algorithm with *tournament selection*, *one point crossover*, *transformation placeholder mutation* and *transformation variable mutation*. Moreover, we configured **MOMOT** with a transformation length of 180, and set population size to 500.

We followed [12] in encoding multiplicity constraints in the preconditions of transformation rules. The remaining 11 hard constraints that could not be captured in this way were passed to **MOMOT** as constraint functions.

The **MOMOT** implementation selects a candidate solution if all hard constraints are satisfied, then it executes the view transformation and uses the same PRISM solver (in identical configuration) to calculate performability metrics. Then the search is guided by the value of this extra-functional metric.

We compare the performance of both **Approach 1** (**APPR1**) and **Approach 2** (**APPR2**), where the actual implementations are integrated into the open source Viatra Generator [69] tool. **APPR1** searches a well-formed model (randomly) and then it executes the view transformation and calls the PRISM solver on this concrete candidate model. **APPR2** repeatedly calls PRISM during the search for approximating upper and lower bounds as guidance hints.

For all three approaches, we evaluate the value of the performability metric reached within a given time limit of 20 minutes (following the evaluations of other solver-based approaches, e.g. [8, 51, 67, 71, 73]). We repeated every measurement 30 times [82], and report the individual results of each run, as well as the medians of the 30 runs (highlighted with **bold** lines).

A full replication package is available at https://doi.org/10.5281/zenodo.6974248.

4.1.3 Execution setup. Each measurement was executed with 6 CPU cores and a 32 GiB memory limit. The graph generator and the PRISM 4.6 analysis tool ran on openjdk 11.0.11. For **MOMOT**, we had to downgrade openjdk to 1.8.0_312 due to compatibility issues. Each run was carried out in a separate JVM.

4.2 RQ1: Performance comparison

The performability results of the three approaches on small, medium and large models models (respectively) are depicted in the three left-most columns in Figure 7. The 1st row corresponds to the **SAT** case while the 2nd row represents the **SAT+FB** case.

MOMOT was *unable to find any consistent models* in any of the investigated scenarios withing the time bounds. We expect that this is due to the large number of complex (global) hard constraints in the domain [37, 56]. By relaxing some of these constraints, **MOMOT** would likely derive solutions, although, for a simplified problem.

For *small models*, both **APPR1** and **APPR2** derived solutions with near-identical performability values. Moreover, the best architecture was derived by **APPR2** earlier (in less amount of time) than by **APPR1** in the **SAT** case.

For *medium models*, in the **SAT** case, **APPR2** lags behind **APPR1**, but eventually reaches similar performability values. In the **SAT+FB** case, **APPR2** falls behind in the media performability value due to several runs failing to produce any valid design candiates, even though several runs do indeed find solutions quickly with identical performability values as **APPR1**.

For *large models*, only **APPR1** was able to consistently provide valid solutions within the time limit, while **APPR2** was only able to do so in 3 and 2 our of 30 runs in the **SAT** and **SAT+FB** cases studies, respectively. The models found be **APPR2** had lower performability values. We will carry out a more detailed root cause analysis of these results as part of the next research question.

RQ1: *In case of small models, both approaches provided valid system architectures with good performability values. For large models, **APPR1** was the only scalable solution. Both **APPR1** and **APPR2** outperformed the baseline **MOMOT** solution.*

4.3 RQ2: Runtime analysis

To further investigate the internal behavior of **APPR1** and **APPR2**, the right-most column of Figure 7 depicts the median of runtime used for each exploration step for models of different size. The 1st row shows the **SAT** case while the 2nd row shows the **SAT+FB** case.

In case of **APPR1**, decision and unit propagation steps dominate execution time for all model sizes, followed by exploration time, and a significant increase in state coding time for large models. Since the external PRISM checker is called only for valid system architectures that satisfy all the hard constraints, its overall runtime is still negligible for **APPR1** even in case of large models.

On the other hand, in case of **APPR2**, the execution time is increasingly dominated by the performability analysis time of the external PRISM checker. In fact, for large models (where **APPR2** mostly failed to find a solution within the given time limit), execution is almost exclusively spent by PRISM. For small models, performability analysis takes a few milliseconds, but for large models, each run takes several seconds, which becomes prohibitively expensive to be used for guiding the search.

RQ2: *Internal steps of the underlying graph solver dominate the runtime for **APPR1**, while the total runtime of performability analysis by PRISM is negligible. The runtime of external PRISM-based performability analysis dominates the runtime for **APPR2**, causing a major scalability limitation for large models.*

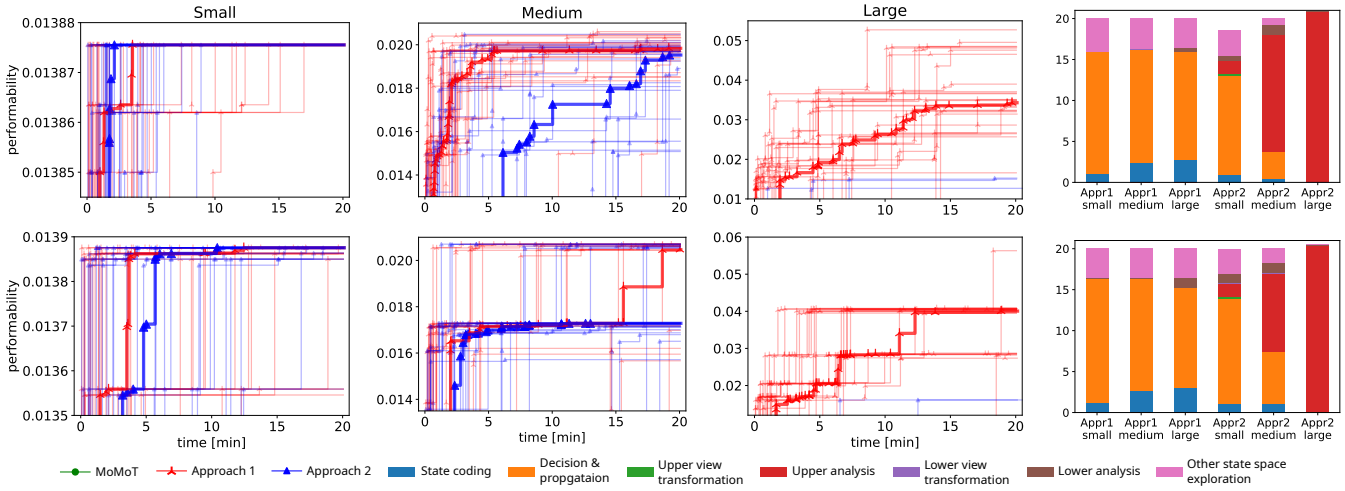

Figure 7: Left 3 columns: Comparison of performability metrics (on small, medium and large models) for MOMOT, APPR1 and APPR2; Right: runtime analysis of internal phases (for APPR1 and APPR2); 1st row: SAT case, 2nd row: SAT+FB case

Note that the results in the context of our performability case studies are substantially different compared to existing results [56, 68] where external solvers were effectively used with partial models for guiding the search during model generation (in the presence of only hard constraints). We believe that the essence of these findings can be summarized in the following guideline.

Guideline: Using external analyzer tools as heuristics to guide design space exploration for partial models along conservative approximations is an effective technique if (1) the runtime of the external analyzer is a small fraction of total execution time, and (2) it scales well with increasing model size. Otherwise, external analysis tools should be used by post-filtering consistent (concrete) models that already satisfy all hard constraints.

4.4 Threats to validity

Internal validity. The baseline of our measurements is based dominantly on the MOMoT configuration used in the original case study [37]. While we investigated several search algorithms prior to the experiments and choose the most promising one, a better performance could still be obtained by various settings of the MOMoT DSE tool. A non-linear cost constraint used in the original paper would be unlikely to increase the performance of MOMoT. We could reduce the variance in the measurement results by increasing the number of runs. Moreover, **Approach 2** could possibly scale better with tighter over- and under-approximations.

External validity. The generalizability of our results is limited by our choice of case studies, which are conceptually very different challenges, but they are from the same domain. Unfortunately, existing case studies used by other DSE tools exclude performability analysis, thus we opted for showcasing optimal architecture synthesis in the context of a complex engineering challenge originating from NASA researchers. However, our approach can in principle be executed on other domains by providing the appropriate metamodel, FOL constraints, and view transformations. While the proposed approaches can be parametrized by the actual view transformations that derive the analysis models, one would likely obtain different results in case of extra-functional properties other than performability. Furthermore, other meta-heuristic search algorithms may perform differently than the one used as our baseline.

Construct validity. While both of our approaches are theoretically complete, we did not manage to find an actual proof of optimality within the time limit. Thus, there is no guarantee that an optimal architecture has been synthesized.

5 RELATED WORK

5.1 Architecture-based stochastic analysis

Methods for the construction of stochastic analysis models are widespread in the evaluation of reliability, availablity, dependability, and performance metrics architecture models, especially for component-based design [10, 45]. Underlying analyis formalisms include fault trees [29, 30, 40, 80], Markov chains [46], queuing networks [47], and Generalized Stochastic Petri Nets (GSPN) [9, 18, 52, 53, 59]. It would be possible to adapt these transformations to serve as the view transformation in our approach.

Meedeniya et al. [57] proposed an evaluation approach for uncertain architectures based on random sampling. In contrast, our approach can generate the refinements of an uncertain architecture exhaustively along with performability metrics.

5.2 Meta-heuristic approaches

Meta-heuristic approaches rely on search algorithms like simulated annealing [21], tabu search [34], or evolutionary algorithms like NSGA-II [22] and eMOEA [23]. They support extensible analyses [62], such as reliability and performability analyses [81], as well as multi-objective optimization. But, they provide no completeness guarantees and may fail to provide globally optimal solutions [43, 44]. We can classify these approaches whether they map the architectures to some *genotype* vector before DSE to allow the easy formulation of search and mutation operators, or directly work with graph-based representations, such as MBSE artifacts.

Approaches with explicit points of variability. By introducing a *fixed number* of explicit points of variability into a system model, such as the number of redundant component instances and the possible allocations of functions, a genotype vector for systems models can be constructed [55]. ArcheOpteryx [5] and AQOSA [50] provide DSE for AADL in this manner, while PerOpteryx [14] targets the Palladio Component Model and offers a framework for attaching component-level redundancy and other system configuration parameters to genotypes [55]. GDSE [66] provides a general framework for translating genotypes back to system models.

EvoChecker [28] introduces variability points directly into a stochastic model described in the PRISM [49] language. Rodes [16] extends this approach to the synthesis of robust stochastic models.

While these approaches offer scalability due to the fixed-length, domain-specific genotype encoding, such encodings are not directly applicable for problems with a *variable number of objects and connections*, such as communication network topologies.

Graph-based approaches. Graph-based techniques use graph transformations [2] or refactorings to generate candidate designs as graph models. They either rely on *model-based* search, where a graph model is being mutated, or *rule-based* search, where solutions are encoded as a sequence of graph transformation operations [39].

MOMoT [27] and MDEOptimiser [13] rely on the Henshin model transformation language [75] for model-based exploration. WF constraints pose a challenge for such approaches: they are either handled by relaxing hard constraints into *soft* constraints or by encoding them in the transformation rules. Burdusel et al. [12] proposed the automated generation of transformation rules that preserve a limited class of WF constraints (multiplicity constraints). PLEDGE [73] combines evolutionary search with SMT solving to preserve WF constraints over object attributes.

Viatra-DSE [1, 36] is a rule-based DSE tool that relies on the Viatra [78] language. However, synthesizing long chains of model transformations might be challenging in the presence of logical WF constraints; e.g., the effectiveness of evolutionary *crossover* operators is diminished compared to *mutation* operators [1]. SHEPhERd [19] and EASIER [6] aim to derive sequences software architecture refactorings according to extra-functional criteria.

In contrast, our approach can directly satisfy logical WF constraints without having to encode them into transformation rules. This makes it suitable for use in design spaces with only a few solutions, such as those investigated in [81].

5.3 Solver based approaches

Solver based techniques for generating models include constraint programming, such as in UMLtoCSP [15] and DesertFD [24], SAT solving, such as in Alloy [38] and CoBaSA [54], and Satisfiability Module Theory (SMT), such as in FORMULA [41]. As a benefit, can provide a *complete* exploration of the design space.

As a weakness, the supported metrics and objectives is limited by the input language and the supported *background theories* [42]. For supported theories, optimizing SMT solvers provide capabilities for finding globally optimal solutions [11, 51]. Alternatively, the Guided Improvement Algorithm [63] can explore optimal solutions as long as the metric of interest can be formalized with SMT. The AUTOFocus3 tool [25] combines SMT-based DSE with the visualization of solutions to select design candidates according to complex objectives after they have been generated by the solver.

Cortellessa et al. [18] investigated the improvement of extra-functional properties of software architectures with JTL bidirectional model transformations [17]. A refactoring is applied to the analysis model that is known to improve the performance metric, which allows synthesizing improved architectures even though the used solver lacks direct support for performance evaluation.

However, scalability of may be limited in the case of graph-like synthesis problems [71]. Tableau-based reasoning for graphs has been proposed in [4, 60, 67], but these approaches lack the support for extra-functional properties. Recently, partial modeling [64, 65] based graph generation was proposed [71] as an implementation of a DPLL [42] decision procedure for graph models. The refinement unit [68] paradigm allowed extending this approach to linear inequality constraints [56], attribute constraints [68], and geometrical constraints [8]. In this context, our approach provides a refinement unit for performability constraints.

6 CONCLUSIONS

In this paper, we addressed to problem of design space exploration by architecture synthesis in the presence of hard constraints (functional and extra-functional), and performability as an optimization target. We proposed two alternative approaches that innovatively combine graph model generation techniques along partial model refinement with performability analysis carried out by the PRISM stochastic model checker. One approach first synthesizes a valid concrete model as candidate architecture which satisfies all hard constraints, and then it uses a view transformation on this candidate model to derive a performability model amenable to analysis. The other approach proposed conservative under- and over-approximations of achievable performability values by repeatedly calling the back-end stochastic analyzer. We have highlighted that key theoretical properties (such as soundness, δ-optimality or completeness) hold for our architecture synthesis approach.

We carried out experimental evaluation in the context of a complex case study where DSE techniques have been used by NASA engineers for early mission planning for satellite constellations [37]. Our results show that the first approach scales better when deriving larger models, while the second approach may find the best design in a shorter amount of time. Nevertheless, both approaches showed significantly better performance than the meta-heuristic search based baseline used in [37].

As an important finding (and limitation related to recent results [56, 68]), partial model refinement can be effectively combined with external analysis tools for design space exploration when the analysis runtime is a small fraction of total exploration time. Otherwise, post-filtering by analyzing only valid concrete models that satisfy all hard constraints provides better scalability.

Acknowledgements. Project no. 2019-1.3.1-KK-2019-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme. This paper is partially supported by the NSERC RGPIN-2022-04357 project.

REFERENCES

[1] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. 2014. Multi-objective optimization in rule-based design space exploration. In *ASE*. ACM, 289–300.

[2] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. 2002. Generative Programming via Graph Transformations in the Model-Driven Architecture. In *Workshop on Generative Techniques in the Context of Model Driven Architecture, OOPSLA*.

[3] Airlines electronic engineering committee (AEEC). 2006. Avionics application software standard interface - ARINC specification 653 - part 1 (supplement 2 - required services).

[4] Ahmad Salim Al-Sibahi, Aleksandar S. Dimovski, and Andrzej Wasowski. 2016. Symbolic execution of high-level transformations. In *SLE*. Springer, 207–220.

[5] Aldeida Aleti, Stefan Björnander, Lars Grunske, and Indika Meedeniya. 2009. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *MOMPES*. IEEE, 61–71.

[6] Davide Arcelli, Vittorio Cortellessa, Mattia D'Emidio, and Daniele Di Pompeo. 2018. EASIER: An Evolutionary Approach for Multi-objective Software ArchItecturE Refactoring. In *ISCA*. IEEE, 105–114.

[7] AUTOSAR Consortium. 2013. The AUTOSAR standard. https://www.autosar.org/

[8] Aren A. Babikian, Oszkár Semeráth, Chuning Li, Kristóf Marussy, and Dániel Varró. 2021. Automated generation of consistent, diverse and structurally realistic graph models. *Softw. Syst. Model.* (2021).

[9] Simona Bernardi, Susanna Donatelli, and Giovanna Dondossola. 2004. Towards a Methodological Approach to Specification and Analysis of Dependable Automation Systems. Springer, 36–51.

[10] Simona Bernardi, José Merseguer, and Dorina C. Petriu. 2012. Dependability modeling and analysis of software systems specified with UML. *ACM Comput. Surv.* 45, 1 (2012).

[11] Nikolaj S. Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νZ - An Optimizing SMT Solver. In *TACAS (LNCS, Vol. 9035)*. Springer, 194–199.

[12] Alexandru Burdusel, Steffen Zschaler, and Stefan John. 2021. Automatic generation of atomic multiplicity-preserving search operators for search-based model engineering. *Softw. Syst. Model.* 20, 6 (2021), 1857–1887.

[13] Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. 2018. MDEoptimiser: A Search Based Model Engineering Tool. In *MODELS*. ACM, 12–16.

[14] Axel Busch, Dominik Fuchss, and Anne Koziolek. 2019. PerOpteryx: Automated Improvement of Software Architectures. In *ICSA*. IEEE, 162–165. https://doi.org/10.1109/ICSA-C.2019.00036

[15] Jordi Cabot, Robert Clarisó, and Daniel Riera. 2007. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In *ASE*. ACM, 547–548.

[16] Radu Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. 2017. RODES: A Robust-Design Synthesis Tool for Probabilistic Systems. In *QEST (LNCS, Vol. 10503)*. Springer, 304–308. https://doi.org/10.1007/978-3-319-66335-7_20

[17] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. 2010. JTL: A Bidirectional and Change Propagating Transformation Language. In *SLE (LNCS, Vol. 6563)*. Springer, 183–202.

[18] Vittorio Cortellessa, Romina Eramo, and Michele Tucci. 2020. From software architecture to analysis models and back: Model-driven refactoring aimed at availability improvement. *Inf. Softw. Technol.* 127 (2020), 106362.

[19] Vittorio Cortellessa, Raffaela Mirandola, and Pasqualina Potena. 2015. Managing the evolution of a software architecture at minimal cost under performance and reliability constraints. *Sci. Comput. Program.* 98 (2015), 439–463. https://doi.org/10.1016/j.scico.2014.06.001

[20] Cyber-Physical Systems Public Working Group. 2017. Framework for Cyber-Physical Systems: Volume 1, Overview. https://doi.org/10.6028/NIST.SP.1500-201

[21] Robert I. Davis and Alan Burns. 2008. Response Time Upper Bounds for Fixed Priority Real-Time Systems. In *RTSS*. IEEE, 407–418.

[22] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197.

[23] Kaylanmoy Deb, Manikanth Mohan, and Shikhar Mishra. 2003. *A fast multi-objective evolutionary algorithm for finding well-spread Pareto-optimal solutions.* Technical Report 20032002. IIT Kanpur. https://www.egr.msu.edu/~kdeb/papers/k2003002.pdf

[24] Brandon K. Eames, Sandeep Neema, and Rohit Saraswat. 2010. DesertFD: a finite-domain constraint based tool for design space exploration. *Des. Autom. Embed. Syst.* 14, 2 (2010), 43–74.

[25] Johannes Eder and Sebastian Voss. 2016. Usable Design Space Exploration in AutoFOCUS3. In *OSS4MDE@MODELS (CEUR Workshop Proceedings, Vol. 1835)*. CEUR-WS.org, 51–58. http://ceur-ws.org/Vol-1835/paper08.pdf

[26] Michais Famelis, Rick Salay, and Marsha Chechik. 2012. Partial models: Towards modeling and reasoning with uncertainty. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 573–583.

[27] Martin Fleck, Javier Troya, and Manuel Wimmer. 2016. Search-Based Model Transformations with MOMoT. In *ICMT@STAF (LNCS, Vol. 9765)*. Springer, 79–87.

[28] Simos Gerasimou, Giordano Tamburrelli, and Radu Calinescu. 2015. Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering. In *ASE*. IEEE.

[29] Sinem Getir, Lars Grunske, André van Hoorn, Timo Kehrer, Yannic Noller, and Matthias Tichy. 2018. Supporting semi-automatic co-evolution of architecture and fault tree models. *J. Syst. Softw.* 142 (2018), 115–135.

[30] Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. 2017. Model-Based Safety Analysis for Vehicle Guidance Systems. In *SAFECOMP*. Springer, 3–19.

[31] Stephen Gilmore, László Gönczy, Nora Koch, Philip Mayer, Mirco Tribastone, and Dániel Varró. 2011. Non-functional properties in the model-driven development of service-oriented systems. *Softw. Syst. Model.* 10, 3 (2011), 287–311.

[32] Stephen Gilmore and Jane Hillston. 1994. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Computer Performance Evaluation, Modeling Techniques and Tools, 7th Int. Conf., Vienna, Austria, May 3-6, 1994, Proceedings (LNCS, Vol. 794)*, Günter Haring and Gabriele Kotsis (Eds.). Springer, 353–368.

[33] Stephen Gilmore, Jane Hillston, and Natalia Zon. 2016. Abstract Interpretation of PEPA Models. In *Semantics, Logics, and Calculi - Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays (LNCS, Vol. 9560)*, Christian W. Probst, Chris Hankin, and René Rydhof Hansen (Eds.). Springer, 140–158.

[34] Fred W. Glover, Manuel Laguna, and Rafael Martí. 2018. Principles and Strategies of Tabu Search. In *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*. Chapman and Hall/CRC, 361–377.

[35] László Gönczy, Zsolt Déri, and Dániel Varró. 2008. Model Transformations for Performability Analysis of Service Configurations. In *Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, September 28 - October 3, 2008. Reports and Revised Selected Papers (LNCS, Vol. 5421)*, Michel R. V. Chaudron (Ed.). Springer, 153–166.

[36] Ábel Hegedüs, Ákos Horváth, and Dániel Varró. 2015. A model-driven framework for guided design space exploration. *Autom. Softw. Eng.* 22, 3 (2015), 399–436.

[37] Sebastian I. J. Herzig, Sanda Mandutianu, Hongman Kim, Sonia Hernandez, and Travis Imken. 2017. Model-transformation-based computational design synthesis for mission architecture optimization. In *IEEE Aerospace Conf.* IEEE.

[38] Daniel Jackson. 2002. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (2002), 256–290.

[39] Stefan John, Alexandru Burdusel, Robert Bill, Daniel Strüber, Gabriele Taentzer, Steffen Zschaler, and Manuel Wimmer. 2019. Searching for Optimal Models: Comparing Two Encoding Approaches. *J. Object Technol.* 18, 3 (2019), 6:1–22.

[40] Anjali Joshi, Steve Vestal, and Pam Binns. 2017. Automatic generation of static fault trees from AADL models. In *DSN Workshops*. Springer.

[41] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. 2010. An Approach for Effective Design Space Exploration. In *Monterey Workshop*. Springer, 33–54.

[42] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. 2016. Lazy proofs for DPLL(T)-based SMT solvers. In *FMCAD*. IEEE, 93–100.

[43] Aleksandr A Kerzhner, Michel D Ingham, Mohammed O Khan, Jaime Ramirez, Javier De Luis, Jeremy Hollman, Steven Arestie, and David Sternberg. 2013. Architecting cellularized space systems using model-based design exploration. In *AIAA SPACE 2013 Conference and Exposition*. 5371.

[44] Joshua D. Knowles and David Corne. 2000. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evol. Comput.* 8, 2 (2000), 149–172.

[45] Heiko Koziolek. 2010. Performance evaluation of component-based software systems: A survey. *Perform. Eval.* 67, 8 (2010), 634–658.

[46] Heiko Koziolek and Franz Brosch. 2009. Parameter Dependencies for Component Reliability Specifications. *Elec. Note. Theor. Comput. Sci.* 253 (2009), 23–38. Issue 1.

[47] Heiko Koziolek and Ralf Reussner. 2008. A Model Transformation from the Palladio Component Model to Layered Queueing Networks. (2008), 58–57.

[48] Mirco Kuhlmann, Lars Hamann, and Martin Gogolla. 2011. Extensive Validation of OCL Models by Integrating SAT Solving into USE. In *TOOLS (LNCS, Vol. 6705)*. 290–306.

[49] Maria Kwiatkowska, Gethin Normath, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*. Springer, 585–591.

[50] Rui Li, Ramin Etemaadi, Michael T. M. Emmerich, and Michel R. V. Chaudron. 2011. An evolutionary multiobjective optimization approach to component-based software architecture design. In *IEEE CEC*. IEEE, 432–439. https://doi.org/10.1109/CEC.2011.5949650

[51] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. 2014. Symbolic optimization with SMT solvers. In *POPL*. ACM, 607–618.

[52] Juan Pablo López-Grao, José Merseguer, and Javier Campos. 2004. From UML activity diagrams to Stochastic Petri nets: application to software performance

engineering. In *WOSP*. ACM, 25–36.
[53] István Majzik, András Pataricza, and Andrea Bondavalli. 2002. Stochastic Dependability Analysis of System Architecture Based on UML Models. In *Architecting Dependable Systems*. Springer, 219–244.
[54] Panagiotis Manolios, Daron Vroon, and Gayatri Subramanian. 2007. Automating component-based system assembly. In *ISSTA*. ACM, 61–72.
[55] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. 2010. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. 1st Joint WOSP/SIPEW Int. Conf. Perf. Eng.* ACM, 105–116.
[56] Kristóf Marussy, Oszkár Semeráth, and Dániel Varró. 2022. Automated Generation of Consistent Graph Models with Multiplicity Reasoning. *IEEE Trans. Softw. Eng.* 48 (2022), 1610–1629. Issue 5.
[57] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. 2011. Architecture-based reliability evaluation under uncertainty. In *ISARCS*. ACM, 85–94. https://doi.org/10.1145/2000259.2000275
[58] Michael K. Molloy. 1982. Performance analysis using stochastic Petri nets. *IEEE Transactions on computers* 31, 09 (1982), 913–917.
[59] Moulaye Ndiaye, Jean-François Pétin, Jean-Philippe Georges, and Jacques Camerini. 2016. Practical Use of Coloured Petri Nets for the Design and Performance Assessment of Distributed Automation Architectures. In *PNSE*. CEUR-WS, 113–131. http://ceur-ws.org/Vol-1591/paper10.pdf
[60] Karl-Heinz Pennemann. 2008. Resolution-like theorem proving for high-level conditions. In *ICGT (LNCS, Vol. 5214)*. Springer, 289–304.
[61] James L Peterson. 1977. Petri nets. *ACM Computing Surveys (CSUR)* 9, 3 (1977), 223–252.
[62] Sam Procter and Lutz Wrage. 2021. Guided architecture trade space exploration: fusing model-based engineering and design by shopping. *Softw. Syst. Model.* 20, 6 (2021), 2023–2045.
[63] Derek Rayside, H.-Cristian Estler, and Daniel Jackson. 2009. *The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization.* Technical Report MIT-CSAIL-TR-2009-033. Massachsetts Institue of Technology.
[64] Arend Rensink. 2006. Isomorphism Checking in GROOVE. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 1 (2006).
[65] Thomas W Reps, Mooly Sagiv, and Reinhard Wilhelm. 2004. Static program analysis via 3-valued logic. In *CAV*. 15–30.
[66] Tripti Saxena and Gabor Karsai. 2010. MDE-Based Approach for Generalizing Design Space Exploration. In *MODELS (LNCS, Vol. 6394)*. Springer, 46–60.
[67] Sven Schneider, Leen Lambers, and Fernando Orejas. 2017. Symbolic model generation for graph properties. In *FASE (LNCS, Vol. 10202)*. Springer, 226–243.
[68] Oszkár Semeráth, Aren A. Babikian, Anqi Li, Kristóf Marussy, and Dániel Varró. 2020. Automated generation of consistent models with structural and attribute constraints. In *MODELS*. ACM, 18–199.
[69] Oszkár Semeráth, Aren A Babikian, Sebastian Pilarski, and Dániel Varró. 2019. VIATRA Solver: a framework for the automated generation of consistent domain-specific models. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 43–46.
[70] Oszkár Semeráth, Ágnes Barta, Ákos Horváth, Zoltán Szatmári, and Dániel Varró. 2017. Formal validation of domain-specific languages with derived features and well-formedness constraints. *Softw. Syst. Model.* 16, 2 (2017), 357–392. https://doi.org/10.1007/s10270-015-0485-x
[71] Oszkár Semeráth, András Szabolcs Nagy, and Dániel Varró. 2018. A Graph Solver for the Automated Generation of Consistent Domain-Specific Models. In *ICSE*. ACM.
[72] Jaroslaw Skaruz, Artur Niewiadomski, and Wojciech Penczek. 2013. Evolutionary Algorithms for Abstract Planning. In *PPAM (LNTCS, Vol. 8384)*. Springer, 392–401.
[73] Ghanem Soltana, Mehrdad Sabetzadeh, and Lionel C. Briand. 2020. Practical Constraint Solving for Generating System Test Data. *ACM Trans. Softw. Eng. Methodol.* 29, 2 (2020), 11:1–11:48.
[74] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2009. *EMF: Eclipse Modeling Framework* (2 ed.). Addison-Wesley Prof.
[75] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. 2017. Henshin: A Usability-Focused Framework for EMF Model Transformation Development. In *ICGT@STAF (LNCS, Vol. 10373)*. Springer, 196–208.
[76] Mirco Tribastone and Stephen Gilmore. 2008. Automatic Translation of UML Sequence Diagrams into PEPA Models. In *Fifth Int. Conf. on the Quantitative Evaluaiton of Systems (QEST 2008), 14-17 September 2008, Saint-Malo, France*. IEEE Computer Society, 205–214.
[77] Kishor S. Trivedi, Gianfranco Ciardo, Manish Malhotra, and Robin A. Sahner. 1993. Dependability and Performability Analysis. In *SIGMETRICS (LNCS, Vol. 729)*. Springer, 587–612.
[78] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. 2015. EMF-IncQuery: An integrated development environment for live model queries. *Sci. Comput. Program.* 98, 1 (2015), 80–99.
[79] Birgit Vogel-Heuser, Alexander Fay, Ina Schaefer, and Matthias Tichy. 2015. Evolution of software in automated production systems: Challenges and research directions. *J. Syst. Softw.* 110 (2015), 54–84.
[80] Jianwen Xiang, Kazuo Yanoo, Yoshiharu Maeno, and Kumiko Tadano. 2011. Automatic Synthesis of Static Fault Trees from System Models. In *SSIRI*. IEEE.
[81] Marc Zeller and Christian Prehofer. 2013. Modeling and efficient solving of extra-functional properties for adaptation in networked embedded real-time systems. *J. Syst. Archit.* 59, 10-C (2013), 1067–1082.
[82] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evol. Comput.* 8, 2 (2000), 173–195. https://doi.org/10.1162/106365600568202