M Ű E G Y E T E M 1 7 8 2

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

System Architecture Evaluation and Synthesis for Performability with Graph Queries

Scientific Students' Association Report

Author:

Máté Földiák

Advisor:

Kristóf Marussy

2022

Contents

Kivonat

A kritikus rendszerek, például a vasúti infrastruktúra, autonóm járművek, vagy okos városok, hibás működése súlyos anyagi károkon túl akár emberi életet is veszélyeztethet. Így ezen rendszerek tervezése során matematika precizitással kell igazolni nem csak funkcionális követelmények, hanem olyan, extra-funkcionális követelményeket teljesítését is, mint a szolgáltatásbiztonság vagy teljesítmény. Ezeket a sok esetben egymásnak ellentmondó jellemzőket jelentős mértékben a megvalósított rendszer architektúra befolyásolja, így a megfelelő architektúra kiválasztása kulcsfontosságú.

A modellvezérelt rendszertervezésben a az extra-funkcionális jellemzők közötti megfelelő kompromisszumok megkeresésére elterjedtek a tervezésitér-bejáró (Design-Space Exploration, DSE) algoritmusok, melyekkel architektúra javaslatok igen nagy, akár végtelen méretű halmaza is automatikusan bejárható. Ehhez azonban szükséges az extra-funkcionális követelmények automatizált kiértékelése teljes vagy félkész architektúra-javaslatokon. A komplex, szolgáltatásbiztonsággal kapcsolatos extra-funkcionális követelményeket általában valamilyen alacsony szintű, sztochasztikus matematikai formalizmusra történő transzformációval kell megfogalmazni. Így a DSE használata a sztochasztikus modellezéssel kapcsolatos specializált szaktudást igényelhet, valamint kapott analízis modellek sztochasztikus megoldóval való kiértékelése sok esetben skálázhatósági problémákhoz vezet.

A dolgozat célja a magas szintű leíráson alapuló, skálázható architektúra szintézis támogatása. Ennek eléréséhez a dolgozat javasol egy módszert, mellyel a szolgáltatásbiztonsági követelmények közvetlenül a teljes vagy félkész architektúra modelleken, gráfmodell lekérdezésekkel írhatók le. Ezen követelmények kiértékeléséhez a dolgozat a gráfminta-illesztő algoritmusokat a döntési diagrammokon alapuló hibafa-analízis technikákkal kombinálja. Ezen felül a dolgozat kiterjeszt egy logikai megoldón alapuló DSE eszközt a lekérdezésekkel specifikált követelmények szerinti hatékony architektúra-szintézishez.

Az eredményekkel egyrészt lehetővé válik a mérnökök számára a kövelemények megfogalmazása egy magas szintű nyelven, másrészt a szintézis során kihasználhatóak a skálázható modell lekérdező eszközök olyan jellemzői, mint a lekérdezés-optimalizálás és az inkrementális kiértékelés. A javasolt módszert és prototípus implementációját a NASA JPL által bemutatott interferometriai konstelláció szintézis esettanulmányon keresztül vizsgálom és összehasonlítom egy modell transzformáción és sztochasztikus kiértékelőn alapuló módszerrel.

Abstract

Unsafe behavior of critical systems, such as railway infrastructure, autonomous vehicles, or smart cities can lead to severe economic damage or even loss of life. Standards and regulations mandate mathematically precise proof of the satisfaction of not only functional, but also extra-functional requirements, such as dependability and performance. The extra-functional properties can often be conflicting and are profoundly affected by the system architecture. Thus, selection of a suitable architecture during design is essential.

In model-driven systems engineering, Design-Space Exploration (DSE) techniques are available to explore the tradeoffs between various extra-functional requirements. They can automatically enumerate very large, even infinite spaces of design candidates. However, this necessitates the automatic evaluation of extra-functional requirements on complete or partial candidate architectures. Complex dependability requirements are often expressed as model transformations to low-level stochastic mathematical formalisms for automatic evaluation. Therefore, engineers need specialized stochastic modeling expertise to employ DSE, and the analysis of the obtained mathematical models with a stochastic solver can lead to scalability issues.

The aim of this work is to support scalable architecture synthesis with a high-level description of dependability and performability requirements. We present an approach to describe these requirements on complete or partial architecture models using graph queries. We integrate decision diagram based reasoning techniques from fault tree analysis with graph pattern matching to efficiently evaluate dependability requirements graph queries. In addition, we extend a logic solver base DSE tool to synthesize candidate architectures according to extra-functional objectives and constraints specified in this manner.

As a result, engineers are able to express a practically relevant class of extra-functional requirements using a high-level language. Moreover, architecture synthesis can take advantage of the features of efficient graph query engines, such as query optimization and incremental evaluation, to improve scalability. We evaluate the proposed approach and its prototype implementation on an interferometry mission synthesis case study from the NASA JPL, and compare it to architecture synthesis based on model transformations and an external stochastic solver.

Chapter 1

Introduction

1.1 System architecture design

Modern systems show increasing complexity which is managed through model-based system engineering approaches and modeling languages such as SysML, Palladio [51], Æmilia [9]. This increasing complexity makes handling the negative effects of functional and extra-functional requirements more complicated. In many areas standards are defined to ensure safety and correct operation. The AUTOSAR [11, 10] standard in the automotive industry define thousands of requirements and constraints. Similarly in aviation, ARINC 653 [5] prescribe strict design rules in addition to the functional requirements. These functional and extra-functional requirements usually have a negative effect on other. For example, a negative effect is when increasing reliability through redundancy also increase the costs and potentially reduce performance. As a result, during system design engineers have to make compromises and choose from the available alternatives. However, the number of satisfactory architectures is usually out of the manually manageable scope thus selecting a good one is a complicated task. These candidate architectures also have highly diverse extra-functional metrics and so selecting the most preferable is not a trivial task.

Analyzing the extra-functional requirements in systems usually necessitate stochastic methods to accurately describe processes, like component level failures or changes in the physical environment. There are many formalism to manage analysis models such as process algebra [36], Markov chains and Stochastic Petri nets [56, 54]. During evaluation, analysis models are derived from the architecture by complex model transformations [38, 67, 35, 34, 24] and an analysis is carried out by sophisticated external solvers like PEPA [36] or PRISM [1]. Based on this result the satisfaction of extra-functional requirements, like availability and expected performance can be verified. Furthermore, it may allow engineers to isolate the root causes of unsatisfied requirements thus increasing productivity.

To manage the large number of candidate architectures, functional and extra-functional requirements design space exploration (DSE) tools were introduced. These tools are broadly classified into two groups:

(Meta-)heuristic techniques, such as genetic algorithms or multi-objective optimization [51, 3, 33, 18, 9, 31], can support the analysis of extra-functional metrics directly inside the DSE process thus allowing architecture optimization. However, they do not guarantee that all possible models were considered (completeness) and the optimality of the generated model. Moreover, encoding hard constraints must be through either custom soft constraints (constraints that may be violated), objective functions (optimizing the model to not violate constraints) and mutation operators (disallow some mutations to enforce constraints) and it could significantly degrade the performance or scalability of the exploration process [64].

The other group is Logic solver based DSE techniques like [40, 42]. These techniques have guaranteed *soundness* (hard constraints are satisfied) and *completeness.* They usually allow encoding complex logical hard constraints with logical formulas or graph patterns [21, 46, 69, 61]. If the synthesis task is unsatisfiable to an extent these tools may even provide an explanation for its cause. However, purely logical constraints cannot capture most extra-functional due to stochastic nature of the requirement and the necessity of an external numerical solver to analyze them. Thus, solvers have to be specifically extended for optimization tasks, such as in [47, 14] to handle both functional (through logic constraint) and extra-functional requirements. Unfortunately, in many cases such optimizing solvers are unavailable. Alternatively in these cases, a post-filtering based DSE approach can be used [30] where the logic solver synthetize many sound architecture models. After synthesis the analysis models are created for each of them and evaluated separately. After the analysis is complete the models are evaluated and one of them is selected without optimality guarantees.

Logic solvers based on partial modeling offers a well-scaling solution for graph constraints [61], attribute constraints [63] and scope constraints [52] by abstract graph reasoning [59, 58] along refinement. These solvers can efficiently handle constraints of functional requirements but managing and reasoning over extra-functional requirements during synthesis is still an open challenge.

1.2 Aims of this work

The aim of this work is to support scalable architecture synthesis with a high-level description of dependability and performability requirements. To do so we propose a method which allows formalization of extra-functional metrics with graph patterns that requires stochastic models to evaluate. We present this approach to facilitate analysis of extra-functional requirements on complete and partial architecture models. We integrate decision diagram [17] based reasoning techniques from fault tree analysis with graph pattern matching to efficiently evaluate dependability requirements graph queries. With this integration we aim to keep the experienced efficiency of decision diagrams [57] and the scalability of graph pattern matching [69]. In addition, we extend a logic solver base DSE tool [62] to synthesize candidate architectures according to extra-functional objectives and constraints specified in this manner.

As a result, engineers are able to express a practically relevant class of extra-functional requirements using a high-level language. This high-level language mask the underlying formalism of stochastic models thus reducing the need for specialized knowledge and thus increasing productivity. Moreover, architecture synthesis can take advantage of the features of efficient graph query engines, such as query optimization and incremental evaluation, to improve scalability. We evaluate the proposed approach and its prototype implementation on an interferometry mission synthesis case study from the NASA JPL [39], and compare it to architecture synthesis based on model transformations and an external stochastic solver.

Chapter 2

Background

2.1 Requirements and meta-model

2.1.1 Functional requirements and well formedness constraints

Functional requirements describe the tasks that the system must be capable to perform. In case of communication networks, it can be that data from one part of the system bust be able to reach another part of the system. High level requirements often can be refined to multiple smaller requirements to be more manageable. These requirements are generally well defined and formalizable. In system development it is expected from the requirements to be consistent, which means that each and every one can be met simultaneously. If the requirements contain conflicting ones is an issue because automated tools may detect the unsatisfiability of the formalized requirements but cannot resolve them.

Definition 1 (Well-formedness constraints [63]). Hard well-formedness constraints can be formalized as error patterns. These patterns are first order logic predicates that if satisfied means that the well-formedness constraint is violated.

A *theory* $\mathcal{T}$ is a set of first order logic predicates $\mathcal{T} = \{\varphi_1, \ldots, \varphi_k\}$ where φ_i is an error pattern. A candidate considered valid in none of the patterns in $\mathcal{T}$ is satisfied. ▪

Example 1 (Well-formedness constraint). *In our domain an example for a well-formedness constraint is that a* CommSubsystem *should only communicate with* CommSubsystem*s of the same type. As a well formedness constraint it is satisfied if a* target *or* fallback *link exists between two* CommSubsystem *then those should have the same type. Formally the requirement is that*

$$\begin{aligned}\varphi_r = (\mathsf{target}(x,y) \lor \mathsf{fallback}(x,y)) \Rightarrow & (\mathsf{KaComm}(x) \land \mathsf{KaComm}(y)) \lor \\ & (\mathsf{UHFComm}(x) \land \mathsf{UHFComm}(y)) \lor \\ & (\mathsf{XComm}(x) \land \mathsf{XComm}(y))\end{aligned} \tag{2.1}$$

and so the error pattern is $\varphi_e = \neg\varphi_r$.

2.1.2 Extra-functional requirements

Extra-functional requirements are used to capture properties that are not strictly functions of the system. Such requirements are usually related to the system's operation parameters. Many of them are characteristically not satisfied by a single component but the whole system. Extra-functional requirements can be categorized into groups like reliability, performance, cost and some other[15].

Reliability related requirements define the expected operational parameters of the system in case of some error. Characteristic metrics in this group are *availability* (probability of being operational at a given time), mean downtime (average time from failure to operation), mean uptime (average time between failures) or mean time to first failure (expected time of first system failure).

Performance related requirements define what type of stress the system needs to be able to handle. Such metrics for example in a server environment are the long-term throughput (like requests served per hour) or maximal short-term load. This category may also include requirements like maximal power consumption.

We will often use the term *performability* [68] for the combination of reliability and performance. In our case a performability metric is the expected performance of a system with respect to the effects of faults and potential redundancies.

Cost requirements specifies what the system or components should cost to build and operate. Furthermore, this may also include non-monetary cost like personnel required to operate or maintain the system.

Example 2 (Extra-functional requirements for the cases study). *In our case study a* reliability *related requirement is that a* CommSubsystem *must have two outgoing links to different* Commsubsystems *unless the receiver is on the ground station. This introduces redundancy to the system in case one of the receiving* CommSubsystem *malfunctions.*

A performance *related extra functional requirement is that the system architecture should maximize the scientific coverage of the mission according to the original coverage metric of*

$$c_t(n) = \left(1 - \frac{2}{n}\right)^{1+\frac{9}{t}} + 0.05\frac{t}{3} \tag{2.2}$$

for t mission time and n equipped interferometry payload [39]. The performability metric is the expected scientific coverage where the ideal coverage for the given number of Payloads *is weighted with the probability of that many* Payload *being available.*

$$\mathbb{E}[C_t(n)] = \sum_{i=2}^{n} \mathbb{P}\left\{\begin{matrix} i \textit{ payloads have working} \\ \textit{downlink at time } t \end{matrix}\right\} \cdot c_t(i) \tag{2.3}$$

Cost related requirements may include that a constellation must not be larger than a specified number of satellites or must be under a specific total cost that is derived from different component costs. For example the cost of one CommSubsystem *is \$100K or the cost of a satellite is ranging from \$150K to \$2.9M [39] depending on the type.*

2.1.3 Meta-model

A potential way to formalize such requirements is to use meta-modeling techniques. In this technique a meta-model is defined to capture the underlying structure of a concrete system on an abstract level. A domain specific meta-model contains the vocabulary of the domain, such as components and relations that a system architecture may contain. This meta-model is the first thing that regulate a system model. Trivially system model cannot contain elements that are not defined in the meta-model and some relations can

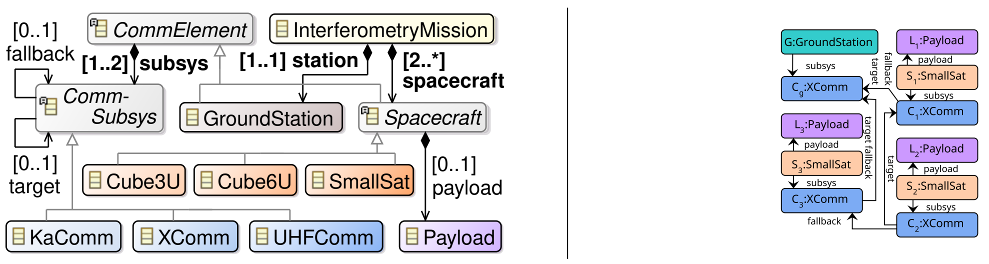


Figure 2.1: Meta-model and concrete model for the satellite constellation mission domain

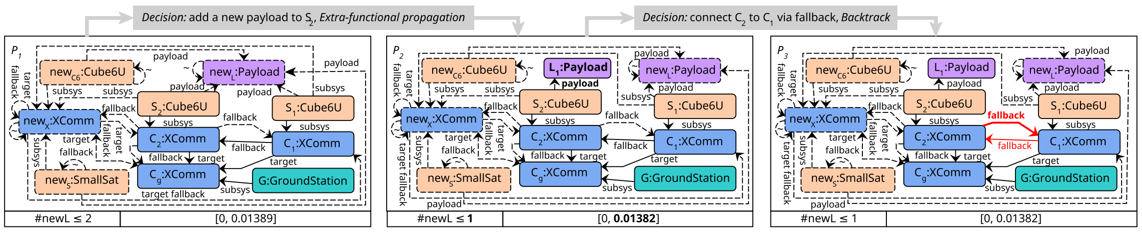


Figure 2.2: Example of refinement and error pattern matching

only be between specific types of components. The latter one is important because in itself it enforces a topology schema that cannot be violated.

Definition 2 (Meta-model). Formally a *meta-model* is a first order logic 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 $\alpha(\mathsf{C}_i) = \alpha(\varepsilon) = 1$, $\alpha(\mathsf{R}_j) = \alpha(\sim) = 2$. ▪

Example 3 (Meta-model). *In our satellite constellation mission domain the meta-model is defined as a class diagram in UML syntax on Figure 2.1. In it the classes represent system component types and the containment edges and references are the potential relations between them. Such component types are (not exclusively) the Cube3U and Cube6U cube satellites or the CommSubsystem. All of these types are included in Σ as class symbols of the meta-model.*

The meta-model also defines relations such as subsystem or the target and fallback links. These relations are the reference symbols of the formal meta-model. In case of the subsystem relation it specifies specifies which Spacecraft is equipped with which CommSubsystem. The target or fallback relations symbolize which CommSubsystem forwards data to which CommSubsystem.

Using this definition the predicate symbols in equation 2.1 can be resolved to meta-model symbols from Σ.

2.2 Modeling and partial modeling

2.2.1 Concrete models

Concrete models are fully completed abstract representation of a system architecture. In such model, every physical or software component of the implemented system are bidirectionally mapped to a model element. This model lists all system components and defines which relations are or should be implemented between which components.

Definition 3 (Concrete model [63]). A concrete is a first order logic structure $M = \langle \mathcal{O}_M, \mathcal{I}_M \rangle$ over signature $\langle \Sigma, \alpha \rangle$, where $\mathcal{O}_M$ is a finite set of *objects*, and $\mathcal{I}_M$ is an *interpretation function* with $\mathcal{I}_M(\varsigma) \colon \mathcal{O}_M^{\alpha(\varsigma)} \to \{1, 0\}$ for all symbols $\varsigma \in \Sigma$. ▪

In the definition $\mathcal{O}_M$ is the set of the components the model contain. This set only treats these components as nodes in the graph representation of the model so by itself it does not distinguish CommElements from CommSubsystems. The $\mathcal{I}_M$ provides the meanings for the components. The interpretation of a class symbol on a component tells whether the component is an instance of that class or not. The same goes for the relation symbols in the metamodel, these unambiguously tell if the relation exists between two elements of $\mathcal{O}_M$ or not.

Example 4 (Concrete model). *On the right of figure 2.1 a concrete model is shown. It establishes that the constellation consists of three SmallSat each equipped with an X band CommSubsystem and a Payload. The constellation also includes a GroundStation and its XComm communication subsystem.*

The model also tells that for example C_2 sends data on target to C_1 and to C_3 on fallback.

2.2.2 Partial models

We can see that concrete models are useful to describe one architecture but one cannot describe multiple ones thus partial models were introduced in [32]. This is done through introducing uncertainty to the model. Formally the interpretation function of the concrete model is now allowed to return $1/2$ values which means that the interpretation of the symbol may hold for the arguments or it may not.

Definition 4 (Partial model [61]). A *partial model* is a first order structure $P = \langle \mathcal{O}_P, \mathcal{I}_P \rangle$ over signature $\langle \Sigma, \alpha \rangle$, where $\mathcal{O}_P$ is a finite set of *objects*, and $\mathcal{I}_P$ is an *interpretation function* with $\mathcal{I}_P(\varsigma) \colon \mathcal{O}_P^{\alpha(\varsigma)} \to \{1, 0, 1/2\}$ for all symbols $\varsigma \in \Sigma$. ▪

We also define regular partial models where components are omitted from $\mathcal{O}_P$ if $\mathcal{I}_p(\varepsilon)(x) = 0$ or $\mathcal{I}_P(\sim)(x, x) = 0$. The former means that the partial model does not contain elements which will not be present in the represented system. The latter means that the patrial model does not contain elements that cannot later be concretized to one component. The extended interpretation of the existence and equality symbols allow one component to represent multiple component in a partial model [52]. For regular partial models these are the following[52]:

- $\mathcal{I}_p(\varepsilon)(x) = 1$ and $\mathcal{I}_P(\sim)(x, x) = 1$: x represents exactly one component.
- $\mathcal{I}_p(\varepsilon)(x) = 1$ and $\mathcal{I}_P(\sim)(x, x) = 1/2$: x represents at least one component.
- $\mathcal{I}_p(\varepsilon)(x) = 1/2$ and $\mathcal{I}_P(\sim)(x, x) = 1$: x represents at most one component.
- $\mathcal{I}_p(\varepsilon)(x) = 1/2$ and $\mathcal{I}_P(\sim)(x, x) = 1/2$: x represents any number of components.

With this in some cases many model components can be represented as one node in the patrial model thus allowing a more compact form. From these four cases the first and last carries extra significance. Elements in $\mathcal{O}_P$ that is in the first category represents the surely known components of the architecture just as in concrete models. While elements in the last are called multiobjects. These multiobjects can be used to represent what components may be later added to the system which is important to keep the model size low.

Example 5 (Partial model). *On figure 2.2 three regular partial models are shown. Elements with continuous border indicated that its existence and self-equivalence is 1such are S_1 and S_2 satellites. Each of these nodes nodes also indicates its most concrete type, so in case if C_g it shows that it is a XComm but it also implies that it is a CommSubsystem too. Dashed border and an arrow with ~ mean that the node is a multiobject of a type. For example, new_L represents the potential payloads that can be added to the architecture.*

Continuous arrows mean relation with 1values. Such relations are the target from C_1 towards C_g. Dashed lines mean relations with ½values. For example the payload relation from S_1 to new_L means that S_1 may be equipped with a payload instance split from the Payload multiobject.

Finally, according to the definition of regular partial models, elements with 0values are omitted. In this case it implies that no Cube3U is in or can be added to the architecture. Furthermore, in case of an omitted relation it means that it does not stand like C_1 is not and never will be a subsystem of S_2.

2.2.3 Refinements of partial models

Partial models were introduced to handle uncertainty but it is also important define relations between partial models based on the contained certain and uncertain parts. It can be seen that some partial models may contain less uncertainty than another while being highly similar to it. To capture this a refinement relation is defined between partial models.

Definition 5 (Information ordering of logic values [63]). $X \succcurlyeq Y \Leftrightarrow (X = 1/2) \vee (X = Y)$ which means Y is a refinement of X if X is ½ meaning that it can be set to either 1 or 0, or X and Y is equal which means other logic values must not change. ▪

An important consequence of this definition is that if a value has been refined to 1 or 0, then it is locked and cannot be changed to anything by refinement. Using this definition between logic values a refinement function can also be defined on partial models.

Definition 6 (Refinement of partial models [52, 63]). Partial model Q is a refinement of partial model P denoted as $P \succcurlyeq Q$ (read P refined to Q) if three is a refinement function $ref{:}\mathcal{O}_P \rightarrow 2^{\mathcal{O}_Q}$ that meets the following conditions:

- $\mathcal{O}_Q = \bigcup_{x \in \mathcal{O}_P} ref(x)$ meaning that every element of $\mathcal{O}_Q$ is a refinement of an element in $\mathcal{O}_P$.
- $\dot{}\forall x \in \mathcal{O}_P{:}\mathcal{I}_P(\varepsilon)(x) = 1 \Rightarrow ref(x) \neq \emptyset$ meaning that if an element surely exists in$P$ then it also must exists in Q.
- All $s \in \Sigma$, $p_1, \ldots p_{\alpha(s)} \in \mathcal{O}_P$, and $q_1 \in ref(p_1), \ldots, q_{\alpha(s)} \in ref(p_{\alpha(s)})$ implies that $\mathcal{I}_P(s)(p_1, \ldots, p_{\alpha(s)}) \succcurlyeq \mathcal{I}_Q(s)(q_1, \ldots, q_{\alpha(s)})$. Meaning that every interpretation in Q must be a refinement of an interpretation in P with respect to the refinement function. ▪

Informally these conditions ensures that a refinement cannot remove a surely existing element or relation from a partial model, every new element introduced in Q must have a source from which it is refined in P. The last is that any relations that were present in P must also be present in Q with all its refinements. In practice means that if a new object is split from a multiobject then it also inherits all the multiobjects relation.

We can also see that each refinement step increases the precision of the partial model. It is also true that every unknown part can be refined and thus eliminated from a partial model resulting in a partial model that does not contains any $1/2$ values which is by definition a concrete model.

These refinement steps can be categorized into two groups. *Decision type* refinements where both 1 and 0 values are possible and *propagation type* refinements where only one value can be correct. This is important because some decisions may imply that other currently $1/2$ values must either 1 or 0.

Example 6 (Refinement of partial model). *Figure 2.2 also shows a series of refinements. On P_1 a decision can be made to add a new payload to S_2 and the resulting partial model is P_2. This means that $\mathcal{I}(\varepsilon)(L_1) = \mathcal{I}(\sim)(L_1, L_1) = \mathcal{I}(\mathsf{payload})(S_2, L_1) = 1$. From the metamodel it is known that a payload can only be assigned to one satellite thus the following propagation type refinements can be made $\mathcal{I}(\mathsf{payload})(S_1, L_1) = \mathcal{I}(\mathsf{payload})(new_S, L_1) = \mathcal{I}(\mathsf{payload})(new_{C6}, L_1) = 0$.*

2.3 Graph queries on concrete and partial models

Furthermore on partial models logic predicates can also be evaluated to $\{1, 0, 1/2\}$ on both concrete and partial models. To define the semantic first a variable binding must be introduced as $Z : \{v_1, \ldots, v_n\} \mapsto \mathcal{O}$. This binds the parameters of φ predicate to elements of the object set. With these the following semantics can be defined [63, 61, 52]:

- $[\![1]\!]^P_Z := 1$ which means that the logic true is true with any parameter binding.
- $[\![0]\!]^P_Z := 0$ which means that the logic false is false with any parameter binding.
- $[\![1/2]\!]^P_Z := 1/2$ which means that the logic unknown is unknown with any parameter binding.
- $[\![C(v)]\!]^P_Z := \mathcal{I}_P(C)(Z(v))$ checks if v is an instance of class C in P partial model with Z variable binding.
- $[\![R(v_1, v_2)]\!]^P_Z := \mathcal{I}_P(R)(Z(v_1), Z(v_2))$ checks if v_1 and v_2 is in relation of R in P partial model and Z variable binding.
- $[\![v_1 = v_2]\!]^P_Z := \mathcal{I}_P(\sim)(Z(v_1), Z(v_2))$ checks if v_1 and v_2 are equals in P partial model and with Z variable binding.
- $[\![\varepsilon(v)]\!]^P_Z := \mathcal{I}_P(\varepsilon)(Z(v))$ checks if v exists in P partial model and with Z variable binding.
- $[\![\neg\varphi]\!]^P_Z := 1 - [\![\varphi]\!]^P_Z$ is the semantic for negation.
- $[\![\varphi_1 \land \varphi_2]\!]^P_Z := min\{[\![\varphi_1]\!]^P_Z, [\![\varphi_2]\!]^P_Z\}$ is the semantic for and of predicates.
- $[\![\varphi_1 \lor \varphi_2]\!]^P_Z := max\{[\![\varphi_1]\!]^P_Z, [\![\varphi_2]\!]^P_Z\}$ is the semantic for or of predicates.
- $[\![\exists v : \varphi]\!]^P_Z := max\{[\![\varepsilon(v) \land \varphi]\!]^P_{Z, v \mapsto o} : o \in \mathcal{O}_P\}$ It is important to note that is not enough to satisfy only the predicate but the existence of v also required.
- $[\![\forall v : \varphi]\!]^P_Z := min\{[\![\neg\varepsilon(v) \lor \varphi]\!]^P_{Z, v \mapsto o} : o \in \mathcal{O}_P\}$ It is important to note that if variable v not exist the predicate does not need to be satisfied.

- $[[R^{+}(v_1, v_2)]]^P_Z \coloneqq max^{|\mathcal{O}_P|}_{k=0}\{[[R(v_1, u_1) \wedge R(u_1, u_2) \wedge \ldots \wedge R(u_k, v_2)]]^P_Z\}$ which means that transitive relations can be defined alongside a relation which length is at most the size of the model. (Meaning that all nodes are in the path.)

This means that any error predicate defined for concrete models can also be evaluated on partial models and the most significant difference is that it may be evaluated to $1/2$. It also implies that on partial model if it is evaluated to 0 then it cannot be violated regardless of the unknown parts. Similarly if it is evaluated to 1 then it is violated. If the error pattern is evaluated to $1/2$ that means the partial model not necessarily violates the constraint but there may be a more refined version where it is surely violated. Informally it means that if an error is introduced through a refinement, then it cannot be resolved by any more refinement.

Example 7 (Query evaluation on partial model). *A well-formedness constraint in our case study it that no communication loops should exits. A simplified formalization of this requirement with an error constrain can be the following:* $link(a_1, a_2) = \mathsf{target}(a_1, a_2) \vee \mathsf{fallback}(a_1, a_2)$ *and* $\varphi(v_1, v_2) = link(v_1, v_2) \wedge link(v_2, v_1)$

We can see on figure 2.2 that $link(v_1, v_2)^{P_2}_{v_1 \mapsto C_1, v_2 \mapsto C_2} = 1$ *and* $link(v_1, v_2)^{P_2}_{v_1 \mapsto C_2, v_2 \mapsto C_1} = 1/2$ *meaning that there is surely a transmittion link from* C_1 *to* C_2 *but not necessarily in reverse thus as expected* $\varphi(v_1, v_2)^{P_2}_{v_1 \mapsto C_2, v_2 \mapsto C_1} = 1/2$.

If the connection from C_2 *to* C_1 *is refined to* 1 *we can also evaluate the error predicate and we can see that* $\varphi(v_1, v_2)^{P_3}_{v_1 \mapsto C_2, v_2 \mapsto C_1} = 1$ *which means that an error were introduced into the partial model.*

2.4 Reliability analysis and fault trees

Up until this point we did not talk about fault and failures in the system. According to Murphy's first law, "Anything that can go wrong will go wrong." In this context it is wise to analyze how this failure impacts the whole system. When analyzing reliability one of the first thing needed is the reliability characteristics of the included components. This can come in many forms such as probability of failure on demand, probability of failure over time or expected lifetime.

The other necessary input for reliability calculations is the system architecture. This introduces the concrete components that are present and can malfunction, furthermore it also describes the dependencies between components. From these different analysis models can be constructed such as Petri-nets [56], Markov models [23] or fault trees [71].

2.4.1 Fault trees

Fault trees are widely used to model the propagation of component failures and to check which failure combination result in a system level failure and with what probability [71]. Generally, a fault tree can contain basic events and logic gates (*and*, *or* and sometimes *not*) but there are many variations that extends this formalism for higher applicability in modeling dynamic systems.

The *basic event* is considered to be the bottom of the fault tree. These events represent independent components or actions that may malfunction and each of them has a probability of failure. Static fault trees are not concerned about the underlying reasons for the failure nor the order of their occurrence. The top of the fault tree is the *top-level event*.

This is the event that is considered to be a system level failure. The goal of a reliability analysis using fault trees is to quantify the probability of the top-level event or to identify certain system states in which it occurs.

Between the top-level event and the basic events logic gates can be defined. Regularly a fault tree is a tree graph where the basic events are leaves and the logic gates are the other nodes. This simple structure allows simple evaluation but it can also severely reduce the expressiveness of this method. To extend the expressiveness, in some cases this restriction can be waived at the cost of computational complexity. For example, the strict tree structure can be replaced with an acyclic graph. Static fault trees can contain the following gates [71]:

- AND gate: true if all the inputs are.
- OR gate: true it at least one of the inputs are true.
- Voting gate: true if at least k of the n inputs are true.

Although fault trees are generally used to model failures there are inverse fault trees that models operation with the same components. Other variations of fault trees also include *negation* and *exclusive or* gates along with a wide variety of other gates to take fault occurrence order into account [71].

Example 8 (Fault and performability modeling). *In our case study we assume that only* CommSubsystems *and* Spacecrafts *can fail and each of the different types have different probability of failure. In [32] we introduced failure rates for these components. Based on that we can calculate that the probability of failure or operation at a given time. For example the probability of operation at one hour in operation is 98.4% for a* Cube3U *satellite or 92% for a* UHFComm *communication subsystem.*

The next is to decide what to model. In our case the performance of the architecture is dependent on the number of payloads available in the system thus it is logical to assume that what we need is the probabilities of receiving the data from that payload.

Our reliability model says that CommSubsystems *on the* GroundStation *are always operational and ready to receive data. A* Spacecraft *is online if the* Spacecraft *and its transmitting* CommSubsystem *are operational, furthermore any of the receiving* CommSubsystem *is ready to receive data. A* CommSubsystem *on a* Spacecraft *is considered ready if the* Spacecraft *it is equipped on is online and the* CommSubsystem *component is operational.*

A fault tree modeling this would contain the hardware level operational status as basic events and the online *and* ready *properties would be intermediate events. Modeling the communication between* Spacecrafts *are formalized with connecting an input to the output of another intermediate event.*

From this we can calculate the probability of at least i payload equipped satellite being online with a voting gate and weight its probability with the matching scientific coverage according to equation 2.3.

2.5 Decision diagrams

We can see that modeling reliability with fault trees has similarities with Boolean functions. In fault trees the top-level event's occurrence is a Boolean function of the basic events. As such the satisfying combinations can be treated as solutions for its Boolean function.

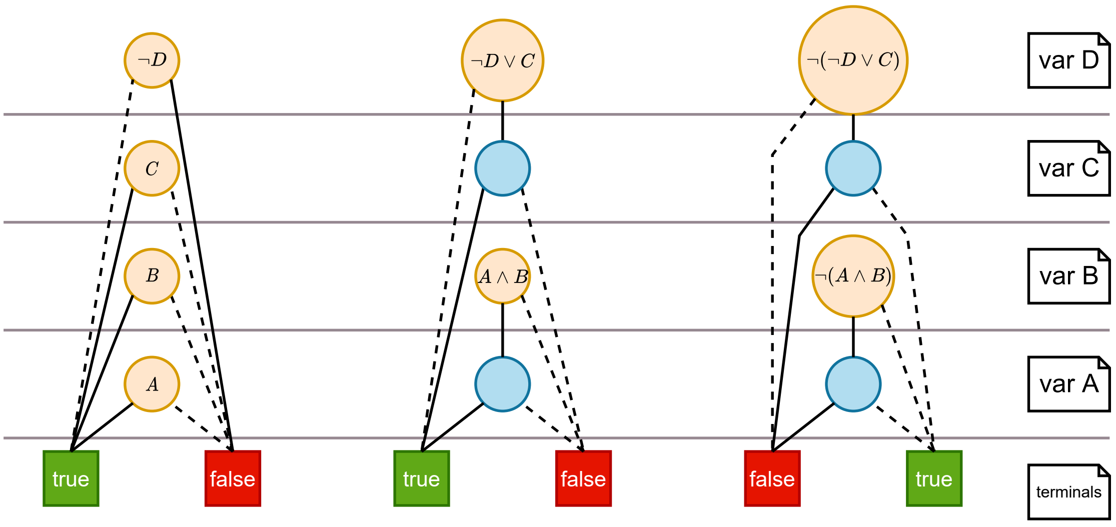


Figure 2.3: Example of a reduced binary decision diagram and logic operations

Unfortunately SAT problems are known to be NP-complete thus in worst case scenarios finding the solutions are exponentially hard. There have been various normal forms introduced to make this task efficient but if the function is not given in that normal form, then transformation is required which makes it exponentially hard again.

Fortunately, with a practical graph representation in many cases it can be relatively efficiently handled [17]. The core concept of this is to fix the variable order and build the satisfaction as a graph

For a Boolean problem there are two bottom level (terminal) nodes one of which represents that the function is satisfied the other is that it is not. Next, we take all the variables and assign each of them to a single level above the terminal nodes. To each layer decision nodes can be added that has two outgoing edges, one that represents that the current layer's variable is true and one if not. Every edge lead to either a lower level node or to a terminal node. When a terminal node is reached then the output of the function is the value of the terminal.

Example 9 (Decision diagram). *On figure 2.3 three decision diagram is shown with the same variable order. On this diagram continuous lines represent that the variable is true and dashed if it is false at the starting node.*

To illustrate such decision diagram we will use the ones on figure 2.3. We have four variables from A to D. On the left side there are four simple Boolean functions denoted with brown circles. Each of it refers to a single variable. For example, the functions represented by the bottom three nodes are true id their respective variables are true, thus continuous lines go to the terminal true and dashed to the terminal false. D is the opposite of it so continuous lines to the terminal false and dashed to the terminal true.

You can also perform operations on the root nodes of the formulas. This is done by manipulating the edges of the decision diagram. For example, in the very simple case of performing the logic or operation between $\neg D$ and C. These operations are done by applying the logic operator on the branches with same values on the same path. We must mention the in this diagram nodes that has the same node on both outgoing edges are hidden, such hidden node in on the D layer with both output to C. Here from the D level we can take the true output of the $\neg D$ and the hidden node and perform the logic or *operation. One edge lead to C and the other to terminal $false$ with means that the new edge goes to C. Similarly, with the $false$ edges, one of the outputs is the terminal true thus it is also the output. When both outputs are nodes then a new node is created and the output is determined by the lower layers.*

The negation that is shown on the right is the simplest one. Here only the terminal nodes have to be reversed to get the output.

It also should be mentioned that in case of Reduced Ordered Binary Decision Diagrams the graph forms of a Boolean function is always the same with the same variable order.

2.5.1 Efficient handling of Reduced Ordered Binary Decision Diagrams

[8] summarize how ROBDDs can be effectively crated and manipulated. This starts by defining the variable order as $x_1 < x_2 < x_3 < \cdots < x_n$ meaning that x_1 is the variable on the lowest level and x_n is the variable at the highest level. Nodes in the ROBDD are represented by numbers from $0, 1, 2, \ldots$. Using this a node in the ROBDD is identified by the assigned variable and the nodes on the true and false edges. The 0 and 1 node is reserved for the terminal nodes.

The ROBDD is stored using tables T and H where $H : \langle i, t, f\rangle \mapsto u$ and $T : u \mapsto \langle i, t, f\rangle$. Here i is the index of the variable, t is the umber of the node at the true edge and f is the node at the false edge. Trivially T is the inverse of H.

For us the important consequence of this is that whenever a seemingly new node $\langle i, t, f\rangle$ is required it can be looked up and reused as presented in [8]. The *member* checks if the arguments are present in the table, *add* and *insert* is to create a new node and to add an existing node respectively to a table. The *lookup* is the get the get the node with the matching arguments.

```
Make[T,H](i,t,f)
    if t = f then return t //edges lead to the same node
    else if member(H,i,t,f) then //the node is already in the graph
        return lookup(H,i,t,f)
    else u ↤ add(T,i,t,f) //create a new node and maintain the global model
        insert(H,i,t,f)
        return u

Build[T,H]($\varphi$)
    function Build'[T,H]($\varphi$,i) =
        if i > n then //all variables are locked
            if $\varphi$ is false then
                return 0
            else
                return 1
        else
            $v_0$ ↤ Build'[T,H]($\varphi$[0/$x_i$], i+1) //lock $x_i$ to false and expand the rest
            $v_1$ ↤ Build'[T,H]($\varphi$[1/$x_i$], i+1) //lock $x_i$ to true and expand the rest
            return Make(i,$v_1$,$v_0$) //create or look up the appropriate node
    end Build'
    return Build'[T,H]($\varphi$,1)
```

This means that whenever an already existing node is encountered the recursive call the call chain is cut short and the already existing node is returned. This practically caches the result of previous calls thus provide an efficient way to manage future operations. Similar caching algorithm also exists for logic operations on ROBDD nodes [8] thus it is not required to perform the operation of the Boolean function and build the graph from the result.

Chapter 3

Overview

Our aim in this work is to support scalable architecture synthesis with logic solvers. We acknowledge that incremental approaches and graph solves proven effective in such tasks [52, 61]. To complement these solutions with the capability of efficient extra-functional reasoning we introduce stochastic graph patterns. These graph patterns will be used to formalize performability related requirements and behavior such as expected system level performance or availability of complex subcomponents. Later on we also intend to use this to approximate the performability metric of partial models and through this support architecture optimization in system architecture synthesis.

3.1 Architecture of the stochastic graph query evaluation

To evaluate graph queries, we propose the following high level system architecture. The query evaluator tool can be utilized for both (A) concrete and partial models (B). Each of these contains a model on which formalize the current system state and query specifications based on the model.

- **(M) Meta-model** is the base of the model and the queries. It contains all the potential component types, relations and basic event types.
- In cases of a concrete model **(A)** the input consist of the **(A2) Concrete model** which is the abstract representation of the system architecture and the **(A1) Query specification for concrete models** which contains the graph queries required for evaluation.
- In case of a partial model **(B)** the input consists of **(B2) Partial model** which unlike a concrete model can contain uncertainty and also a **(B1) Query specification for partial models** which contains the required stochastic query definitions. A preemptive note here is that while a partial model can contain uncertainty, the queries shall concretize it in some way in order to estimate the reliability.

The evaluator tool contains an **(C1) Internal representation** of the input model which is the stochastic view of the model. It contains all the basic events that may occur in the system. This is to keep the representation enumerable, meaning that the queries can only create events that are explicitly derived from other events. This also necessitate that the basic events must be introduced and maintained separately from the queries. This is done in **(1) Representation synchronization** where unnecessary basic events are removed

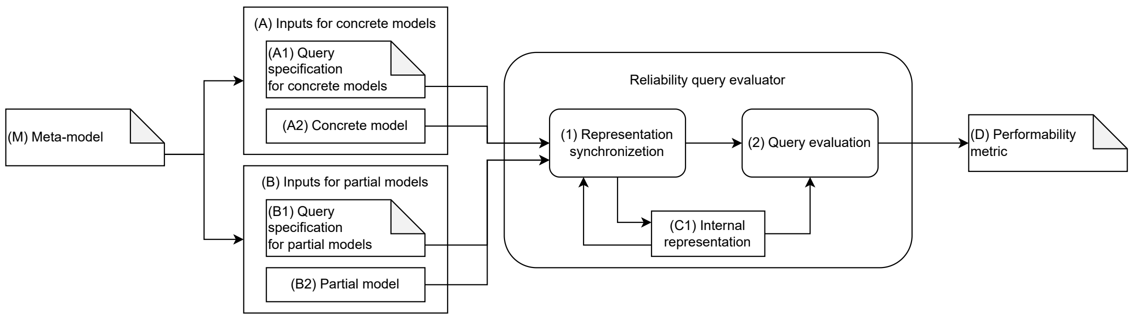

(A) Inputs for concrete models
(A1) Query specification for concrete models
(A2) Concrete model
(B) Inputs for partial models
(B1) Query specification for partial models
(B2) Partial model
(M) Meta-model
Reliability query evaluator
(1) Representation synchronizetion
(2) Query evaluation
(C1) Internal representation
(D) Performability metric

Figure 3.1: Overview of stochastic query evaluation

and the required ones are introduced. When the basic events are synchronized with the model then the graph queries are evaluated in **(2) Query evaluation** and producing the value of the **(D) Performability metric**.

3.2 Architecture synthesis for performability

We introduced a model transformation-based architecture synthesis method in [32]. This is based on a branch-and-bound approach where the *decision type refinements* from a partial model create new branches that lead to the newly created partial models. In there, whenever a new partial model is created then an under-approximation and an over-approximation is calculated for the partial model. The under-approximation can be used as a guideline to compare partial models to each other and select which partial model is to be refined next. The over-approximation is the more important one. This is used to cut branches where the performability metric will never satisfy any of the refinable models from that partial model.

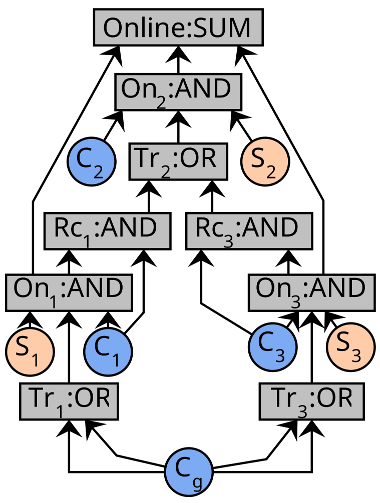
3.2.1 Soundness and completeness

When talking about logic solvers there are two key guaranties that these type of solvers can provide and it is important to uphold these. The first is *soundness* which is that the candidate architecture satisfies all the well-formedness and cost constraints. Formally as defined earlier none of the φ_i error predicates in $\mathcal{T} = \{\varphi_1, \dots, \varphi_k\}$ holds. Furthermore, the extra-functional characteristics of that model also reaches the required thresholds.

The other key aspect of a logic solver is the *completeness*. It means that all concrete models that are valid with the given well-formedness constraints in a finite model scope can be enumerated. This is so that every potential model can either be synthesized with respect to the constraints or the incorrectness can formally be proven.

3.2.2 Architecture synthesis options for logic solvers

There are two potential approaches to synthesize architectures for performability with logic solvers. The first approach is the *post-filtering* approach where the solver randomly generates a large number of candidate architectures and the extra-functional requirements evaluated later. Such approach benefit from the soundness of the generator tool by only having to evaluate functionally correct architectures. Another benefit is that in this approach the analysis model for the extra-functional requirements can be generated using



```
ctmc
module model
    S1 : bool init true; S2 : bool init true; S3 : bool
     init true;
    C1 : bool init true; C2 : bool init true; C3 : bool
     init true; Cg : bool init true;
    [] S1 -> (1/70):(S1'=false); [] S2 -> (1/70):(S2'=
     false); [] S3 -> (1/70):(S3'=false);
    [] C1 -> (1/13):(C1'=false); [] C2 -> (1/13):(C2'=
     false); [] C3 -> (1/13):(C3'=false);
endmodule
formula On1 = S1 & C1 & (Cg | Cg); formula Rc1 = On1 & C1;
formula On3 = S3 & C3 & (Cg | Cg); formula Rc3 = On3 & C3;
formula On2 = S2 & C2 & (Rc1 | Rc3);
formula online = (On1?1:0) + (On2?1:0) + (On3?1:0);
rewards "utility"
    online=2 : 0.016666; online>=3 :  0.035;
endrewards
```

Figure 3.2: Fault tree and corresponding PRISM analysis model

model transformations. The downside of this is that in this case the completeness is lost unless all consistent models are synthesized and evaluated which can be practically impossible.

The second approach is to include a reasoning over extra-functional requirements inside the logic solver. This approach can keep both the soundness and completeness of the generator tool while also eliminating the need for post evaluation. However, this approach raises a handful new complications. One of them that it such approach the extra-functional analysis has to be performed on the internal representation of the logic solver. This representation can be very different from for example a domain specific SysML or UML model. The second concern is that in such tool intermediate results may be evaluated at many points however a slow analysis method can diminish the performance of the synthesis tool.

Example 10 (Reliability model). *To demonstrate performability analysis we use the concrete model from figure 2.1. In this constellation S_1 and S_3 Spacecrafts are directly connected to ground while S_2 can only relay data to ground via either S_1 or S_3.*

From this model we van construct an inverse fault tree (i.e., the occurrence of the top-level event is desired) like model. In this model we can see that the Tr_i intermediate events occur when any of the outgoing links can be used. The On_i event occur when for a given satellite both the necessary hardware components are operational and the links are up. The Rc_i event represent if a Commsubsystem can receive and forward data thus it requires the associated satellite to be available and also the CommSubsystem.

At the top level event we count how many of the Spacecrafts are available and calculate the probability of it.

Example 11 (Model transformation). *A model transformation example is when the concrete model from figure 2.1 is mapped to a textual PRISM model equivalent to the fault tree on figure 3.2.*

In this model the blue and brown basic events are mapped to variables representing the components status and transitions that represent component failures. Using these formulas are defined according to the fault tree and a reward is introduced the weight the possible system states according to the extra-functional metric.

Chapter 4

Performability analysis with graph queries

4.1 Probabilistic predicates

To evaluate complex performability related properties we introduce a formalism that is defined over a model and stochastic properties. To do this we extend the definition of the meta-model and concrete model so that stochastic properties can be represented inside of it. Furthermore, we define probabilistic predicates and provide a semantic for it so that the stochastic behavior can be expressed through these predicates.

Definition 7 (Probabilistic meta-model). The probabilistic meta-model is an extension of the regular meta-model $M = \langle \Sigma, \alpha \rangle$ where $\Sigma = \{C_1, \dots, C_n, R_1, \dots, R_m, B_1, \dots, B_l\}$ is the set of symbols. C_i and R_i symbols are from the meta-model denoting the classes and relations. B_i is the newly introduced probabilistic symbols that denotes event classes. And at last α is the arity function $\alpha : \Sigma \to \mathbb{N}$ where $\alpha(C_i) = 1$, $\alpha(R_i) = 2$ and $\alpha(B_i) \in \mathbb{N}$. ∎

Example 12 (Probabilistic symbol). *In our case study a probabilistic symbol is the* component *that represent whether the model element (*Spacecraft *or* CommSubsystem*) is considered operational.*

Using this we can extend the definition of model to also capture the stochastic properties. To do this we add σ-algebra to the model and an event function that assigns measurable events to probabilistic symbols and model elements.

Definition 8 (Probabilistic model). Formally, we define probabilistic model as an extension of concrete models over a probabilistic meta-model. $M = \langle \mathcal{O}_M, \mathcal{I}_M, \mathcal{E}_M, \Omega, \mathcal{F}, \mathbb{P} \rangle$ over the probabilistic $\langle \Sigma, \alpha \rangle$. Where the $\mathcal{O}_M$ and $\mathcal{I}_M$ is the object set and interpretation function as in concrete models. Ω is the sample space, $\mathcal{F}$ is a σ-algebra over Ω and $\mathbb{P} : \mathcal{F} \to [0, 1]$ is the probability measure. The newly introduced $\mathcal{E}_M(B_i^n) : \mathcal{O}_P^{\alpha(B_i)} \times \{1, \dots, n\} \to \mathcal{F}$ is the event function that assigns an event to the arguments of a probabilistic symbol and a multiplicity marker. (Shorter form in case of $n = 1$ is $\mathcal{E}_M(B_i) := \mathcal{E}_M(B_i^1)$) ∎

In our case a $\Omega = \{0, 1\}^n$ where n is the number of basic events and $\mathcal{F}$ is the power set of Ω thus these are indeed a σ-algebra.

Example 13 (Probabilistic model for satellite domain). *The samples in Ω is the states that the system can be in. In this domain only* Spacecrafts *and* CommSubsystems *can malfunction on component level and each of them can be either in a state of operational (1) or non-operational (0). Thus Ω is the Cartesian product of the states of the components. If we have a model with one* Spacecraft *and one* CommSubsystem *then $\Omega = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$ where the tuples are the $\langle$operation of S_1, operation of $C_1\rangle$. The $\mathbb{P}$ is the probability measure which is $\mathbb{P}(\omega) = \prod_{\omega_i \in \omega} p_i^{\omega_i}(1-p_i)^{1-\omega_i}$ where p_i is the probability of finding that component operational for the basic events.*

$\mathcal{F}$ is the set of measurable events, such as $\{\langle 0,1\rangle, \langle 1,1\rangle\} \in \mathcal{F}$ which is for finding C_i in an operational state. By definition $\mathcal{F}$ contains all measurable event but to being the power set of Ω.

The event function is to get the event that satisfy the probabilistic symbol over the arguments like $\mathcal{E}_M(\text{component})(C_1) = \{\langle 0,1\rangle, \langle 1,1\rangle\}$

With this we can define probabilistic predicates that can be used to formalize stochastic properties.

Definition 9 (Probabilistic predicate). Formally the probabilistic predicate $\psi_Z^M : \mathcal{O}_M^{\alpha(\psi)} \to \mathcal{F}$. ▪

Informally, a probabilistic predicate evaluates to an event that satisfies it. Before we continue it is important to emphasize that φ denotes deterministic predicates that is evaluated to 1 or 0 and ψ denotes probabilistic queries that is evaluated to an event.

To complete all the requirements to evaluate probabilistic predicates we provide the following semantics:

1. $(\!|\Omega|\!)_Z^M := \Omega$ which means that the sure event is a sure event regardless of the model.
2. $(\!|\varnothing|\!)_Z^M := \varnothing$ which means that the impossible event is the impossible event regardless of the model.
3. $(\!|\varphi|\!)_Z^M = \begin{cases} \Omega, & \text{if } [\![\varphi]\!]_Z^M = 1 \\ \varnothing, & \text{otherwise} \end{cases}$ which is to cast deterministic predicates to probabilistic ones so that deterministic predicates can be used in probabilistic ones.
4. $(\!|B(v_1,\ldots,v_{\alpha(B)},n)|\!)_Z^M = \mathcal{E}_M(B^n)(Z(v_1),\ldots,Z(v_{\alpha(B)}))$ is to introduce events to a predicate that may or may not be satisfied and so no longer only sure and impossible events can occur in a predicate. $(\!|B(v_1,\ldots,v_{\alpha(B)})|\!)_Z^M := (\!|B(v_1,\ldots,v_{\alpha(B)},1)|\!)_Z^M$ is a shorter form in case of $n = 1$.
5. $(\!|\neg\psi|\!)_Z^M := \Omega \setminus (\!|\psi|\!)_Z^M$ meaning that a negated probabilistic predicate is equivalent to the complement set of the predicate over all samples.
6. $(\!|\psi_1 \wedge \psi_2|\!)_Z^M := (\!|\psi_1|\!)_Z^M \cap (\!|\psi_2|\!)_Z^M$ meaning that the logic and relation of probabilistic predicates are satisfied if both predicates are.
7. $(\!|\psi_1 \vee \psi_2|\!)_Z^M := (\!|\psi_1|\!)_Z^M \cup (\!|\psi_2|\!)_Z^M$ meaning that a logic or of probabilistic predicates are satisfied if any of the predicates are.
8. $(\!|\exists v\colon (\varphi \wedge \psi)|\!)_Z^M := \bigcup_{o \in \mathcal{O}_M} \{(\!|\psi|\!)_{Z,v\mapsto o}^M \mid [\![\varphi]\!]_{Z,v\mapsto o}^M = 1\}$ is a guarded existential quantification and its purpose to general existential quantification is to suppress irrelevant probabilistic predicates.

9. $(\!|\forall v\colon (\varphi \Rightarrow \psi)|\!)^M_Z := \bigcap_{o \in \mathcal{O}_M} \{(\!|\psi|\!)^M_{Z,v\mapsto o} | [\![\varphi]\!]^M_{Z,v\mapsto o} = 1\}$ is a guarded universal quantification. Similarly, to the previous this is to exclude predicates that are irrelevant.

10. $(\!|\psi^+(v_1,v_2)|\!)^M_Z := \bigcup_{k=0}^{|\mathcal{O}_M|} \{(\!|\psi(v_1,v_2) \lor (\exists m\colon \psi(v_1,m) \land \psi^+(m,v_2))|\!)^M_Z\}$

Naturally, this can be used to create more complex predicates like *k-gates*. Terminal cases are when nothing is needed to be satisfied and then it is surely satisfied or it is impossible to satisfy because there are not enough predicates. It can also be satisfied if we split the first element and there is k out of the rest or ψ_1 is satisfied and $k-1$ is satisfied from the rest.

$$(\!|k \text{ of } \{\psi_1,\psi_2\ldots,\psi_n\}|\!) := \begin{cases} \Omega, & \text{if } k = 0 \\ \varnothing, & \text{if } k > |\{\psi_1,\ldots,\psi_n\}| \\ (\!|(k \text{ of } \{\psi_2,\ldots,\psi_n\}) \lor & \\ (\psi_1 \land (k-1) \text{ of } \{\psi_2,\ldots,\psi_n\})|\!), & \text{otherwise} \end{cases} \tag{4.1}$$

$$groundcomm(v_1) := \exists gsn : \mathsf{GroundStation}(gsn) \land \mathsf{subsystem}(gsn, v_1) \tag{4.2}$$

$$\begin{aligned} ready(v_1) :=& groundcomm(v_1) \lor \exists sat : (\neg groundcomm(v_1) \land \\ & \mathsf{subsystem}(sat, v_1) \land online(sat) \land \mathsf{component}(v_1)) \end{aligned} \tag{4.3}$$

Example 14. *An example for a probabilistic predicate in our domain is whether a CommSubsystem can receive and forward data to ground. In this domain we assume that a CommSubsystem on the ground is always operational.*

To do this we have a deterministic predicate (equation 4.2) that can evaluate if the CommSubsystem is on the ground. The ready (equation 4.3) is the predicate for receiving. The left argument of the $\lor$ is a deterministic predicate and if satisfied, casted to a probabilistic one and evaluates to Ω which is what we expect from a ground communication subsystem. The right argument is for non-ground ones. There the first two is a deterministic query that indirectly acts as the guard of the existential quantifier while the rest is the probabilistic predicate.

The online is a user defined probabilistic predicate to show an inclusion of a probabilistic predicate. The component is a probabilistic symbol from the meta-model and evaluated accordingly to the $\mathcal{E}_M$. As a result, the ready predicate evaluates to a sample set where the satellite is online (i.e. can forward data to ground) and the CommSubsystem is operational so the receiving and forwarding functionality is available.

With this we can manage predicates and such predicate evaluates to a sample set. However, at some point a probability is needed for the performability analysis thus we need to quantify it. This is formally done with $\mathbb{P}$ to probability measure as $\mathbb{P}^M_Z(\psi) \in [0,1]$.

4.2 Efficient representation of probabilistic predicates

We can see that the formal definitions while work require huge amount of storage and computation to manage all predicates thus implementing it as is is impractical. However Reduced Ordered Binary Decision Diagrams [57, 8] (ROBDD) can do just that.

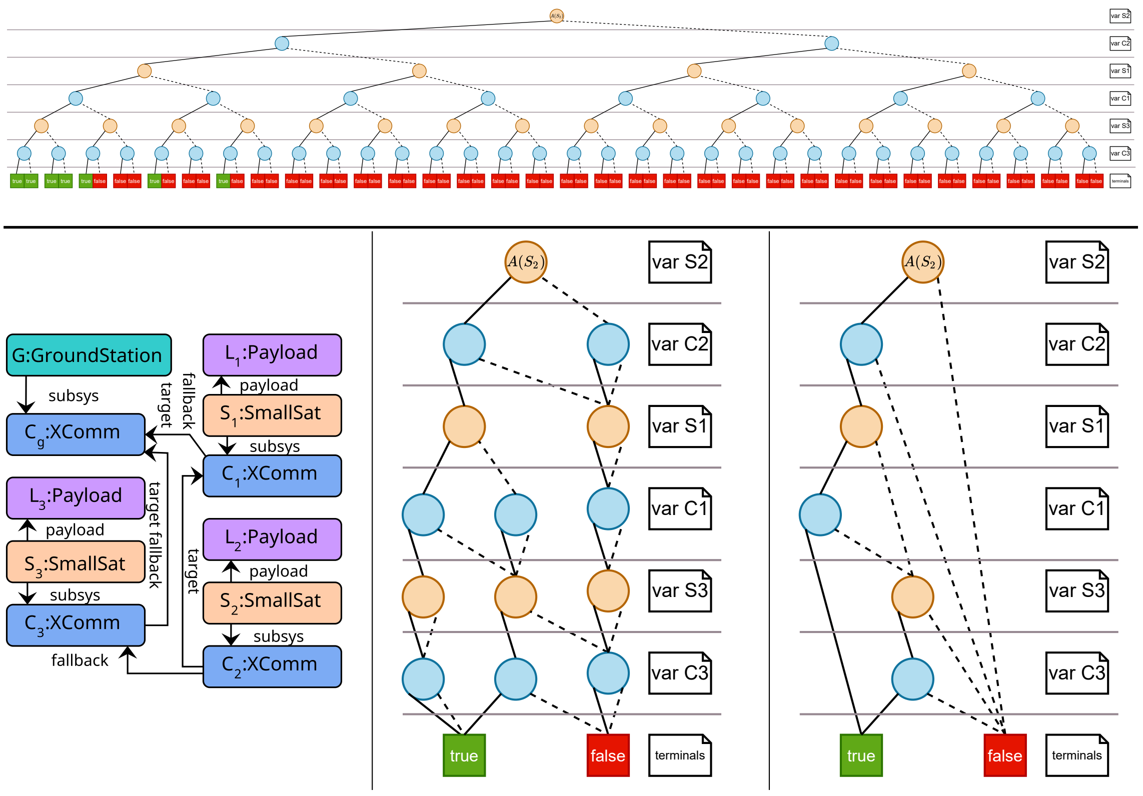


Figure 4.1: Availability of S_2 Spacecraft from the architecture (bottom left) with Decision Tree (top), Binary Decision Diagram (bottom center) and Reduced Ordered Binary Decision Diagram (bottom right)

4.2.1 Binary decision trees

To illustrate why reduced ordered binary decision diagrams are so practical we take a few step back to decision diagrams.

Decision diagrams consists of nodes and terminals that are connected with edges annotated with true or false in a tree structure. The nodes are in layers where every node in the layer is assigned to the same variable. The inputs of every node come from the layer on top and the outputs are going to the layer on bottom. Every node has two children, one for each value of the decision and a single parent. The bottom level nodes are connected to terminals which represents the output of the predicate. Traveling from the root node to a terminal expresses the values of each variable and at the end the value of the predicate. The downside of this representation is that the size of the diagram is exponential in the number of variables with a factor of two.

Example 15 (Binary decision tree). *On figure 4.1 there is the decision three of the availability of S_2 from the concrete model. There the continuous lines represent decisions where the variable is true and dashed where false. We can see that for example the series of decisions true, true, false, false, true and true lead to the true terminal meaning that the satellite is available.*

The other thing that we can notice that it is highly redundant, like 75% of the right side is false regardless of the bottom four variable.

4.2.2 Decision diagrams and reduced ordered decision diagrams

Decision diagrams are a more refined version of decision trees where identical subtrees are only included once. This is through allowing that the children of a node can be any node or terminal from the lower layer and a node can have multiple parents from the higher layer. Such diagrams eliminate the redundant parts of the decision tree while each variable on any path toward a terminal are explicitly visited.

Reduced Ordered Decision Diagrams takes it two steps further with (A) requiring a variable order and (B) eliminating nodes where both outgoing edges goes to the same node or terminal. This form of the decision diagram has the advantage of always being isomorphic for an equivalent Boolean function given a fix variable order. Furthermore, logic operations that can be performed on Boolean functions can also be performed on the ROBDDs of the functions [8].

Example 16 (Reduced ordered binary decision diagrams). *On figure 4.1 we can see that this method proved a much more compact representation for the Boolean function while retaining the same information of when it is satisfied.*

For this example, the decision tree contains 63 nodes, 64 terminals and 126 edges. While an equivalent decision diagram contains only 14 nodes, 2 terminals and 28 edges which is a significant decrease from the decision tree. The ROBDD representation is even more compact with just 6 nodes and 12 edges.

It must be noted that there is no guarantee for getting such compact representation for any Boolean function and in a worst-case scenario such graph can still scale exponentially. But in practice this usually provides a well-scaling solution for Boolean function representation.

4.2.3 Probabilistic predicate evaluation with ROBDD

Here we would like to show that the semantic operations defined earlier can be performed using Reduced Ordered Binary Decision Trees as a more efficient representation of events.

Earlier we used that the sample space Ω is effectively the Cartesian product of the sample space of the components.

- Ω is a ROBDD with the true terminal as the root node. This means that the function is satisfied regardless of the variables just as expected from the sure event.
- $\emptyset$ is a ROBDD with the false terminal as the root node. This means that it is never satisfied just as expected from the impossible event.
- The $\cup$ of events is informally when any of them occur and so it is equivalent to the operation *or* on the ROBDD representation of the event sets [17].
- Similarly, the $\cap$ of event sets is equivalent to the operation *and* on the ROBDD representations of the inputs [17].
- The $\neg$ or complement creation can also be done on a ROBDD by swapping the terminal nodes [53].
- To get the ROBDD representation of the events in $\mathcal{F}$ can be done through ROBDDs as well. Any $\omega = \langle \omega_1, \ldots, \omega_n \rangle \in \Omega$ can be considered a sequence of variables where ω_i is the value of variable i. This implies that every sample can be represented as a

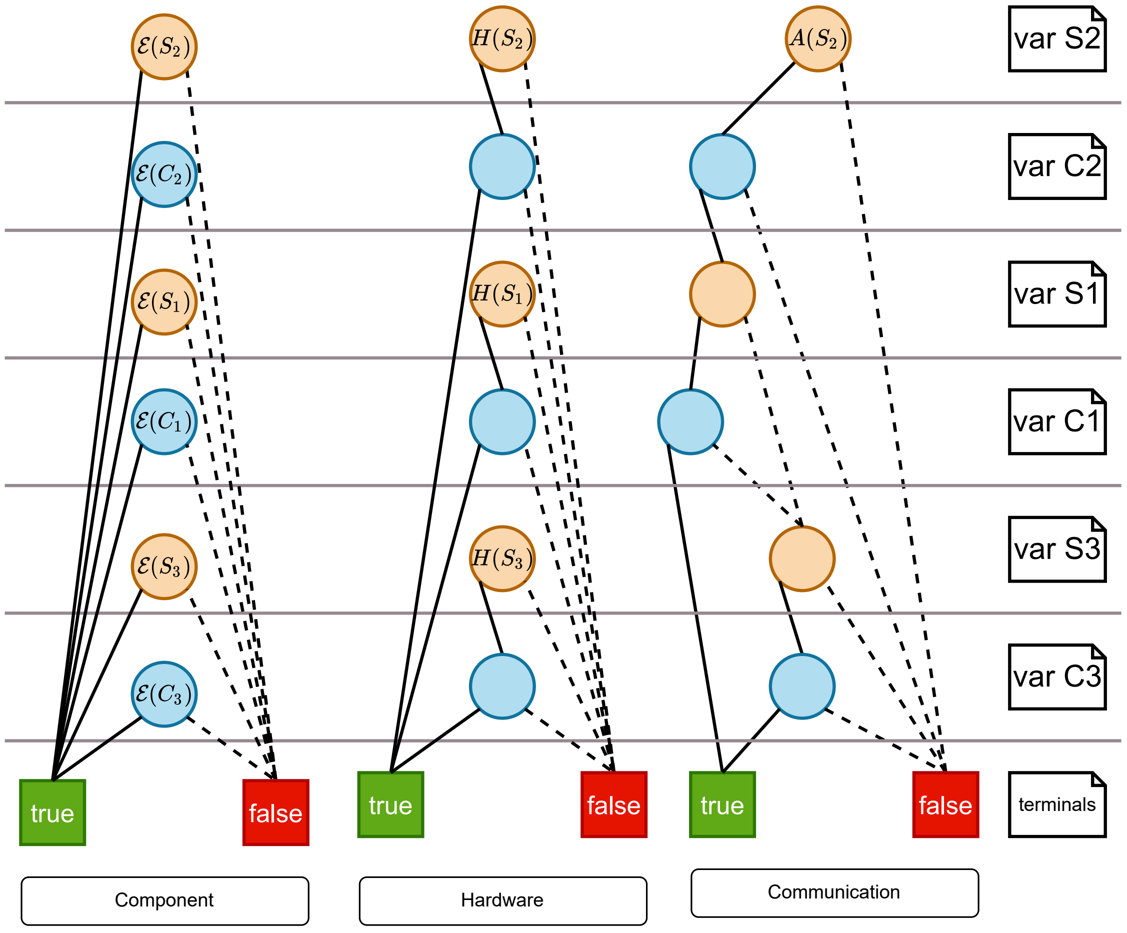


Figure 4.2: ROBDD operation example on satellite availability

predicate where $\omega \iff \varphi_\omega(v_1,\dots,v_n) = \bigwedge_{k=1}^{n} \omega_k = v_k$ and so as a ROBDD. As a result every $e \in \mathcal{F}$ can be expressed as a ROBDD using that $e \iff \varphi_e = \bigvee_{\omega \in e} \varphi_\omega$.

Example 17 (Availability through ROBDD). *To illustrate how the introduced formalism interact with the ROBDD in practice we use the following example. The graphical representation of the ROBDDs are on figure 4.2.*

We use the already established three satellite constellation as an example. In the reliability model we have the probabilistic symbol component *for the event of a component's operation. The ROBDD representation of each is an ROBDD node on the appropriate level with the true edge to the terminal true and the false edge to the terminal false. This is shown on the left side of figure 4.2 where $\mathcal{E}_M(\text{component})(x)$ is shortened to $\mathcal{E}(x)$.*

The next we need the complex hardware level operation which is

$$H(sat) = \exists css : \mathsf{Spacecraft}(sat) \land \mathsf{CommSubsystem}(css) \land \mathsf{subsystem}(sat, css) \land \\ \mathsf{component}(css) \land \mathsf{component}(sat). \tag{4.4}$$

Here the deterministic predicates ensures that no inappropriate probabilistic predicates are created and the probabilistic ones formalize the required operation of both component is needed for correct operation. The result is shown on the center of figure 4.2.

Next, we need to formalize the communication topology. In this example the $ready(C_3) = H(S_3)$ *and* $ready(C_1) = H(S_1)$ *due to being directly connected to ground which is defined in the example of a probabilistic predicate. For a spacecraft availability is*

$$\begin{aligned} A(sat) = &(\exists css, trg : \mathsf{Spacecraft}(sat) \land \mathsf{CommSubsystem}(css) \land \mathsf{subsystem}(sat, css) \land \\ &\mathsf{CommSubsystem}(trg) \land \mathsf{target}(css, trg) \land H(sat) \land ready(trg)) \lor \\ &(\exists css, flb : \mathsf{Spacecraft}(sat) \land \mathsf{CommSubsystem}(css) \land \mathsf{subsystem}(sat, css) \land \\ &\mathsf{CommSubsystem}(flb) \land \mathsf{fallback}(css, flb) \land H(sat) \land ready(flb)). \end{aligned} \tag{4.5}$$

Which is the formalization of the availability for S_2 *defining that* $H(S_2)$ *and either* $H(S_1)$ *or* $H(S_3)$ *needs to be operational in order to* S_2 *be available. The result is shown on the right side of figure 4.2.*

4.3 Traversing ROBDD for event probability

At this point we established the semantics for a probabilistic query and combined it with an effective representation. With this we can create highly expressive probabilistic predicates however we cannot quantify the results. To do this we need to traverse the ROBDD and calculate the probability of the root node. This can be done easily with the following pseudo-code.

```
P[T,H,V,C](n)
    if n = 0 then return 0.0 // bottom of the ROBDD is reached with not satisfying value
    if n = 1 then return 1.0 // bottom of the ROBDD is reached with satisfying value
    if member(C,n) then return lookup(C,n) // check is the probability is cached
    i,t,f ↤ lookup(H,n) // get node definition
    p ↤ lookup(V,i) // get the probability of variable $i$
    v ↤ p*P[T,H,V,C](t) + (1-p)*P[T,H,V,C](f) // calculate probability of $n$
    insert(C,n,v) // update cache
    return v
```

Here we augmented the global model with $V : i \mapsto [0,1]$ which contains the probability of variable i being true. Similarly $C : u \mapsto [0,1]$ (cache) which is the probability of node u being satisfied. We also note that in the calculation step we use that the variables are independent and the rest of the ROBDD does not refer to the current variable.

Chapter 5

Syntesis

In this chapter we want to introduce the necessary formalism and methodology for optimizing architecture synthesis. We expect that such optimization to be theoretically capable to enforce extra-functional constraints while keeping the soundness and completeness of the logic solver. In addition to these requirements, we prefer if the synthesis process maintains the scalability and performance of the solver. To do this we reintroduce the necessary terminology in this section from [32].

5.1 Partial models for performability analysis

The first step is the extension of partial models to encapsulate extra-functional metrics.

Definition 10 (Partial model with performability objective [32]). A partial model with performability metric is the $P = \langle \mathcal{O}_p, \mathcal{I}_p, \mu_P \rangle$ triple where $\mathcal{O}_P$ and $\mathcal{I}_P$ is the object set and interpretation function of the regular partial model (Definition 4). $\mu_P \subseteq \mathbb{R}$ is an interval that contains the extra-functional value of all consistent models refined from P. ▪

Definition 11 (Refinement of partial models with performability objective [32]). Given $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mu_P \rangle$ and $Q = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mu_Q \rangle$ partial models, $P \succcurlyeq Q$ with respect to the interval iff $\langle \mathcal{O}_P, \mathcal{I}_p \rangle \succcurlyeq \langle \mathcal{O}_Q, \mathcal{I}_Q \rangle$ and $\mu_Q \subseteq \mu_P$. ▪

This interval represents the desired and potentially available value of the extra-functional metric. The Such extended partial model P is consistent if the regular partial model is consistent and $\mu_P \neq \emptyset$.

To calculate performability we introduce a *view transformation* denoted as $\mathcal{V}$ that creates an *analysis modes* A from a concrete model. Using this analysis model as an input, an analysis tool $\mathcal{A}$ can calculate the associated performability metric. At the end $performability = \mathcal{A}(\mathcal{V}(M))$ for M concrete model [32].

5.1.1 Approximations over partial models

The uncertainty in partial models implies that there may be many architectures with different extra-functional properties thus the reasoning must adapt to this. Due to the finite number of possible consistent concrete models refinable from the given partial model (if any) there must be a worst and a best one. We capture this by *under*- and *over-approximations* combined with the extra-functional interval from definition 10.

Definition 12 (Conservative approximations [32]). $\mathcal{V}$ is the view transformation for over-approximation and $\mathcal{V}$ is for the under-approximation. Such view transformation is conservative for P partial model if for all consistent concrete model M that is refined from P the following holds:

$$\mathcal{A}(\mathcal{V}_u(P)) \leq \mathcal{A}(\mathcal{V}(M)) \leq \mathcal{A}(\mathcal{V}_o(P)) \tag{5.1}$$

▪

Also we expect that the approximations get more precise with refinements meaning that if $P \succcurlyeq Q$ then

$$\mathcal{A}(\mathcal{V}_u(P)) \leq \mathcal{A}(\mathcal{V}_u(Q)) \leq \mathcal{A}(\mathcal{V}(M)) \leq \mathcal{A}(\mathcal{V}_o(Q)) \leq \mathcal{A}(\mathcal{V}_o(P). \tag{5.2}$$

We also expect form an approximation that for a concrete model, which is just a special case for a partial model, the approximation should equal to the real value as

$$\mathcal{A}(\mathcal{V}_u(M)) = \mathcal{A}(\mathcal{V}(M)) = \mathcal{A}(\mathcal{V}_o(M)). \tag{5.3}$$

Definition 13 (Extra-functional propagation[32]). In extra-functional propagation the extra-functional interval μ_P is modified according to the partial model.

$$\mu'_P = \mu_p \cap [\mathcal{A}(\mathcal{V}_u(P)), \mathcal{A}(\mathcal{V}_o(P))] \tag{5.4}$$

▪

This can be used to incorporate extra-functional consistency. By setting the interval to $[thr, +\infty)$ we can define a threshold that all consistent model must reach. As a result, all concrete models with insufficient extra-functional metric are excluded from the interval thus also from the consistent models. If during propagation $\mu_P = \varnothing$ then the partial model is inconsistent due to none of the refinement concrete models can satisfy the threshold. As a result, such partial model can be pruned without compromising the completeness of the logic solver.

In theory global optimization can also be done by setting an exclusive threshold to the extra-functional value of the last consistent concrete model with the previous threshold. When the state space is completely explored and no consistent model is found then the last consistent model is a globally optimal one. Note that multiple globally optimal architecture can exist but they must have the same extra-functional value.

Example 18 (Extra-functional propagation). *On figure 2.2 we can see the effect of extra-functional propagation between P_1 and P_2 where a new payload is introduced to the system. As a result, the value of the over-approximation lowers the interval.*

5.2 Approximating concretization of partial models

The defined probabilistic predicates are based on concrete models. This manifest in having the deterministic queries evaluated only to 1 or 0. However, a pseudo-concretization can be applied. This pseudo-concretization means that whenever a deterministic query is referred in a probabilistic query it should be forced into 1 or 0. This is obviously context dependent to which value is should be forced to but this is doable. These approximations will relax the synthesis problem by ignoring some well-formedness constraints. An approximation algorithm that can do it without relaxation is likely in possession of the optimal solution in which case the whole synthesis problem does not make much sense.

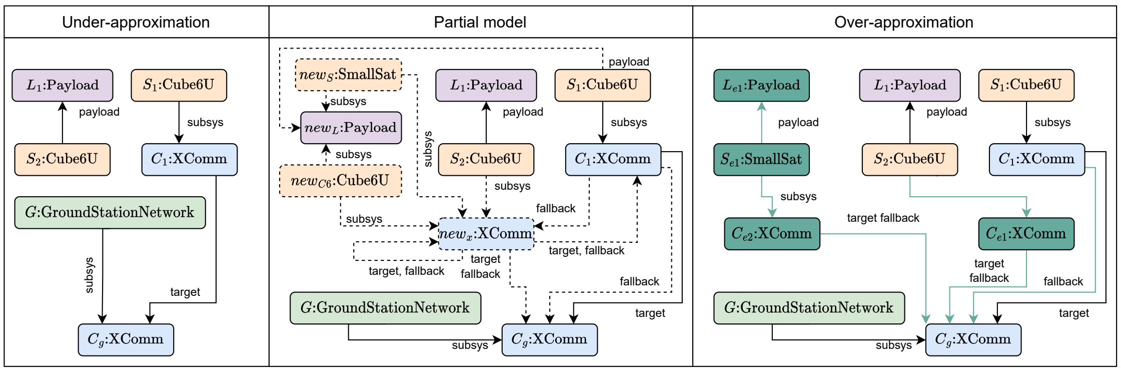


Figure 5.1: Approximating concretization of partial models

Viewing the architecture model as a dependency model some transformations are monotonic in the value of the performability metric. Such over-approximating transformations [32] are

- (1) adding a new component without dependencies;
- (2) replacing a component with a more reliable one;
- (3) replacing common causes of failure with independent failure causes.

Under-approximation is done on a simpler approach, if it is not certainly there then it is not there at all. This is equivalent to assuming that it is not operational.

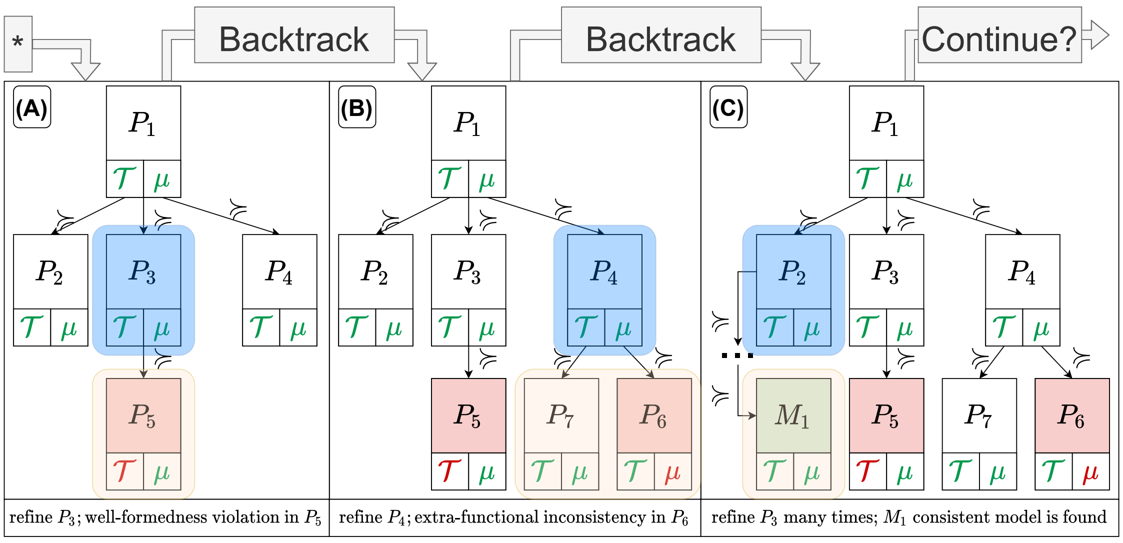
Example 19 (Conservative under-approximation). *As stated before, the estimation algorithm for under-approximation is simply forgetting everything that is not certainly part of the model. On figure 5.1 there is a partial model at the center and its under-approximation view on the left.*

Here we can see that even though S_2 must have a CommSubsystem *in a valid model, in the relaxed under-approximation it does not have thus its availability is severely under-approximated. Similarly, with C_1 the* fallback *edge is treated as non-existent thus C_1 can only rely on the* target*. This does not change the reliability of C_1 because C_g is always operational due to being on G.*

Example 20 (Conservative over-approximation). *Similarly the over-approximation is shown on figure 5.1. Here the uncertain* fallback *link from C_1 is estimated to go to the ground which is indifferent in this case. Here the relaxation is that connecting a link to ground is not always consistent with the well-formedness constraints.*

The second approximation is that if a Spacecraft *does not have a* CommSubsystem *with outgoing link then a new* CommSubsystem *is introduced with the highest possible reliability and direct links to the ground regardless of the current number of* CommSubsystems *assigned to the* Spacecraft*. Such estimation is done with S_2 and C_{e1}.*

The third type is to increase the number of payloads in the system. The number of potential new elements of a type is calculated by the generator based on the current partial model and scope constraints. In this example we assume that it is one but the approximation pattern is the same regardless. Here for each potential Payload *we add a new* Spacecraft *and* CommSubsystem *of the best available type with direct link to ground. Such architecture components are L_{e1}, S_{e1} and C_{e2}.*



*
Backtrack
Backtrack
Continue?
(A)
(B)
(C)
P_1
P_2
P_3
P_4
P_5
P_6
P_7
M_1
$\mathcal{T}$
μ
refine P_3; well-formedness violation in P_5
refine P_4; extra-functional inconsistency in P_6
refine P_3 many times; M_1 consistent model is found

Figure 5.2: Simplified illustration of state-space exploration

5.3 State-space exploration by refinements

In this approach the architecture synthesis is carried out by a series of refinements [52, 63, 61, 32]. First, a consistent partial model is selected and a refinement step is applied, meaning that a new node is added or a relation is refined to 1 or 0. After that, the propagation step is applied where other relations are defined if only one potentially consistent refinement is available. If during refinement or propagation the well-formedness constrains are surely violated then the model is eliminated and a backtrack is performed. Then the extra-functional propagation step is applied to narrow the interval. Similarly, if the interval is inconsistent then a backtrack is performed. This new partial model is also analyzed by a state coder to check if an equivalent partial model is already explored and if so, then eliminates this new partial model due to redundancy.

During backtrack the generator randomly selects an already existing partial model that has unexplored refinements and the refinements continue from there. If such partial model does not exist then the state-space is exhaustively explored and the generation process terminates. Termination is also possible when a consistent model is found, and neither global optimality is not mandated nor more models are required, then the synthesis is successful.

Example 21 (Architecture synthesis). *On figure 5.2 we illustrate the synthesis process. Here from a previous state the generator gets to the (A) state where there is four partial models P_1, P_2, P_3 and P_4. From these P_3 is selected for refinement and P_5 is the newly refined partial model. However, during refinement or propagation the well-formedness constraints are violated thus the partial model is inconsistent and a backtrack is performed from P_5.*

This leads to the (B) state where the new partial model to refine is P_4. Here we say that two partial models are created. It turns out that P_6 cannot satisfy the extra-functional constraint and thus eliminated from further exploration. P_7 can be further refined but let's say that at some point a backtrack is performed and the new selected partial model is P_2.

From P_2 the consistent concrete model M_1 is refined. Here the generator can terminate if sufficient number of consistent models are found or it can continue. If the generator con-

tinues and the goal is to find the globally optimal model then the extra-functional intervals are updated to exclude models not better than the new one and a backtrack is performed.

5.4 Optimizing synthesis by logic solvers

There are two potential approaches when synthesizing architecture models with logic solvers. Such approaches are post-filtering and optimizing searches [32]. In a post-filtering approach, the extra-functional metric is evaluated when a consistent concrete architecture model is synthesized. Based on this the model is either kept or discarded based on whether is satisfied the extra-functional requirements. The main advantages include that

- (A) only a small amount of extra-functional requirement analysis is needed thus complex, time consuming methods are acceptable;
- (B) the internal formalism of the generator is irrelevant and
- (C) only consistent models are analyzed.

The downside is that proving the global optimality of a candidate architecture is either done manually afterward which can be time consuming or all architectures must be synthesized which is even more time consuming.

The other approach, the optimizing searches, uses intermediate states to approximate the value of extra-functional properties and reason over them. The potential advantages are

- (A) state-space reduction by cutting extra-functionally surely unsatisfiable branches;
- (B) providing formal proof for optimality when the synthesis is complete and
- (C) providing formal proof for unsatisfiability.

The downside is that likely many intermediate approximations are needed thus adding a potentially huge extra time requirements and extra-functional formalization is needed in the internal language.

Chapter 6

Evaluation

To evaluate the applicability of the presented graph query-based performability formalization approach we carried out a series of measurements. The focus of our measurements is the performance of the presented approach in various potential contexts. This also serves as the next iteration of the earlier measurements in [32] where the logic solvers (post-filtering and model transformation-based) were compared to genetic algorithms.

RQ1. How does the query-based evaluation compares to a model-transformation based evaluation for concrete models?

RQ2. How does the query-based approach compares to a model-transformation based approach using incremental evaluation on concrete models?

RQ3. How does the query-based approach compares to a post-filtering and model-transformation based approach?

RQ4. How do various exploration steps contribute to exploration time when generating near-optimal system architecture candidates of increasing size?

6.1 Measurements setup

6.1.1 Case studies

The domain we use for the case studies were introduced in [39]. For evaluation we utilize two case versions of the introduced case study. In the original case study (we refer to it as **SAT**) the mission architecture does not include redundancy. Which manifests in that a CommSubsystem can only have one outgoing link, namely the target. In the second version (**SATFB**) we introduced redundancy with the fallback link thus allowing a more complex system architecture. Using additional well-formedness constraints we enforce link separation by forcing to go to different CommSubsystems on different Spacecrafts.

We also include the component costs to use a cost constraint in addition to the size limitation. The cost of a SmallSat is \$2,900K, a Cube6U costs \$650K and a Cube3U costs \$150K. Furthermore, every CommSubsystem costs \$100K and every Spacecraft must be equipped with at least one. At last, a Payload included in the constellation costs an additional \$50K. We must not that the original case study in [39] uses a non-linear cost constraint, meaning that each additional component of the same type costs less than the previous, is linearized by using the maximal cost of a component.

We classify models into three groups by the number of components and mission architecture cost. All category must contain at least 8 components as this is the minimal size of a consistent solution. *Small models* are considered to contain less than 10 components with the total cost up to \$5M. *Medium models* contains up to 20 components with a total cost of maximum \$9M. *Large models* contains up to 30 components and costs at most \$15M. These cost limits are based on the architecture costs presented in [39].

6.1.2 Compared approaches for concrete models

In **RQ1** and **RQ2** we want to investigate the applicability of query-based evaluation as in a manual system design process. As a baseline we utilize our earlier measurements using model transformation, which is a well-established methodology [41], and an academic analysis tool [1].

For this we run the VIATRA generator [70] 40 times for each scope and case study to generate random inputs for the evaluation resulting in at least 28 models for each category. During evaluation we run both methods and assert the extra-functional results to ensure that equivalent verification algorithms were implemented.

In this setup, for **RQ1** we compare the performance of the query-based approach with the model-transformation based approach when evaluated on a concrete model. This measurement emphasizes the scalability of the underlying verification method such as Binary Decision Diagrams and Continuous Time Markov Chains.

For **RQ2**, we take an incremental approach were from an empty model the input model is created by adding one component or relation in each step. The intention with this is to simulate model building done by an architecture designer. After each step, the resulting model is evaluated by both approaches. This measurement is to analyze the usability of the methods in an incremental environment.

6.1.3 Measurements for applicability in a logic solver

As a baseline we use a **post-filtering** and a model transformation based approach using **PRISM-Nailgun** from [32]. In the earlier measurements the post-filtering approach was the most effective in all category. The PRISM based approach utilizes approximations thus it serves as a valuable baseline for assisted architecture synthesis.

In the *post-filtering* approach whenever the generator finds a consistent model it evaluates its extra-functional characteristics and logs. Then it continues the generation as if nothing has happened. In the **model transformation** based approach the partial models are approximated with PRISM after each step. In the **query based** approach the performability model is maintained through the query engine and the approximations are extracted from the queries.

For all measurements using the logic solver we limited the available time to 20 minutes similarly to [47, 60, 12, 61, 65] and repeated the measurements 30 times [72].

6.1.4 Execution environment

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. Between each run, in the same measurement category, 30 seconds were allocated for the java garbage collector.

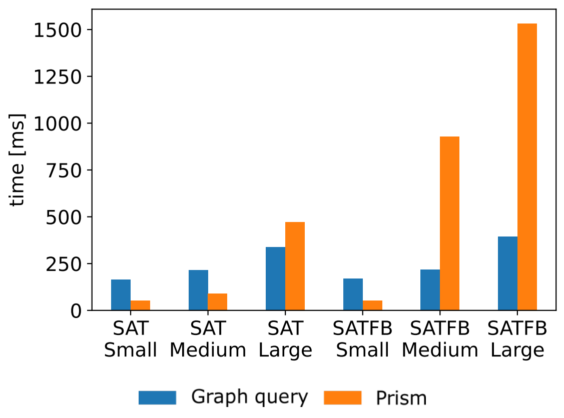


Figure 6.1: Evaluation of concrete model is one step

6.2 RQ1: Performance of concrete model evaluation

The results of the associated measurements are shown on figure 6.1. The blue bars represent the required time to analyze the input model using graph queries and the orange bars are the time required by the model-transformation based approach using PRISM.

For *small models* the Binary Decision Diagrams were significantly worse than the external solver. A potential explanation for that is at this scale the models are very simple thus the required analysis model is also simple and so the PRISM model is easy to solve. On the other hand, the query-based implementation likely suffers from an extra overhead for the first evaluation. We will address this phenomenon in RQ2.

For *medium models* the graph query based method show little increase in the required time for both case studies which is indicative of good scaling both for size and complexity. However the external solver explodes with the **SATFB** case study while in case of the **SAT** version it increases moderately compared to the small cases. A potential reason is that the analysis models are significantly more complex with the redundancy and thus the analysis problem is much harder.

For *large models* the external solver require significantly more time than for small or medium models. The difference between the complexity of the case studies is still highly affect the runtime but the proportional difference decreases between the medium and large sizes. The query-based evaluation also requires more time but the total increase in runtime is much smaller.

RQ1: *For small and simple models PRISM performs better but with increasing size and complexity graph queries with Binary Decision Diagrams perform significantly better.*

6.3 RQ2: Performance of incremental evaluation

The measurement results are depicted on figure 6.2. Each column shows a size category from small (left) to large (right). The first row is the **SAT** and the bottom is the **SATFB** case study. On this diagram the teal lines depict the individual analysis time for each step for the PRISM analysis tool and the blue line it the total runtime of PRISM. Similarly, the orange line if the time required by the query-based evaluation method and the red line is the total time.

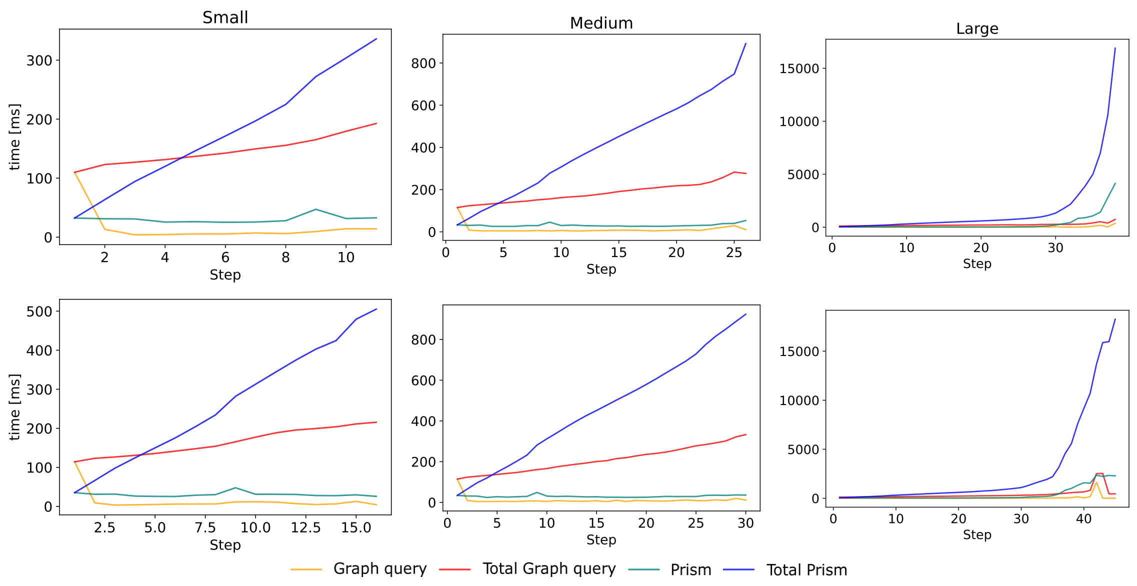


Figure 6.2: Runtime of graph query (ROBDD) and model transformation (PRISM) based evaluation methods

There are two notable phenomena on the diagrams. The first is that the query-based algorithm with Binary Decision Diagrams takes less time for every evaluation except the first one. Such behavior is indicative of an initialization step either by the query engine or the BDD library. However, this extra time is already taken back by the 10th step.

The second thing is that the PRISM analysis tool explodes around 25 to 30 steps in an exponential manner. This can be interpreted as a severe warning to reevaluate whether such tool is really necessary.

RQ2: *The graph-query based approach significantly outperform the model-transformation based one in an incremental use-case.*

6.4 RQ3: Performance of logic solver

The performability of the models encountered during exploration are shown on figure 6.3. The thin lines represent the best performability value encountered at the given time for a single run. The markers highlight changes in the best performability. The thick lines and markers are the median of the best encountered performability at the given time. The red color is associated with the post-filtering approach, blue with the model transformation based one and green is for the graph query-based approach.

For *small models* the new graph query based approach significantly under perform in case of the **SAT** case study and moderately in case of the **SATFB** case.

For *medium models* the graph query based approach performs similarly to the model transformation based approach but both is by some measure dominated by the post-filtering approach.

For *large models* both assisted synthesis method mostly fail to produce a result while the post-filtering approach performs well. However the query based approach reaches many times more consistent models compared to the very occasional findings of the model transformation based one.

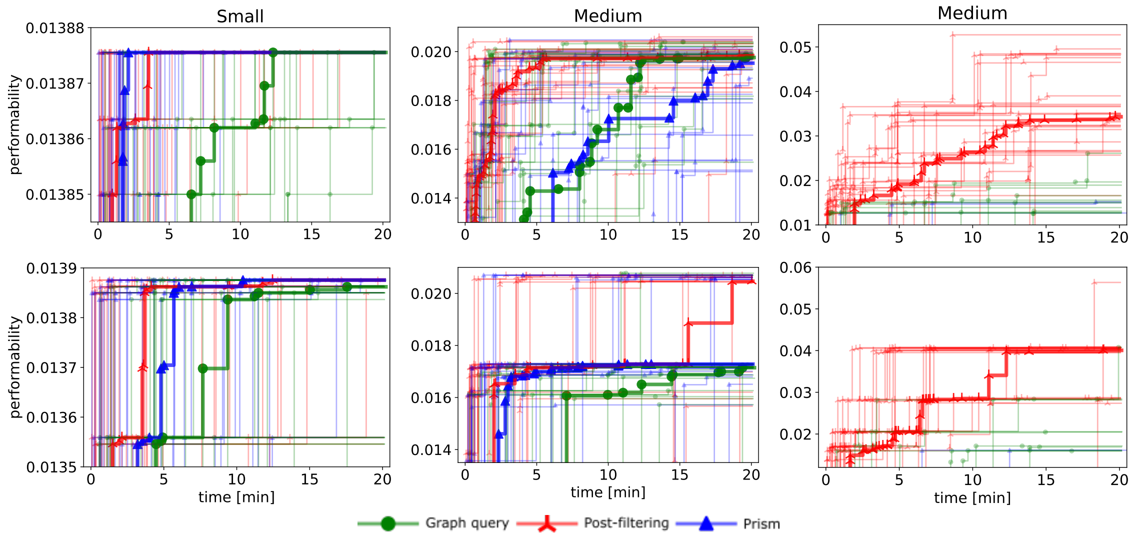


Figure 6.3: Comparison of performability metrics (on small, medium and large models) for Post-filtering, Prism and model transformation and Graph queries with BDD

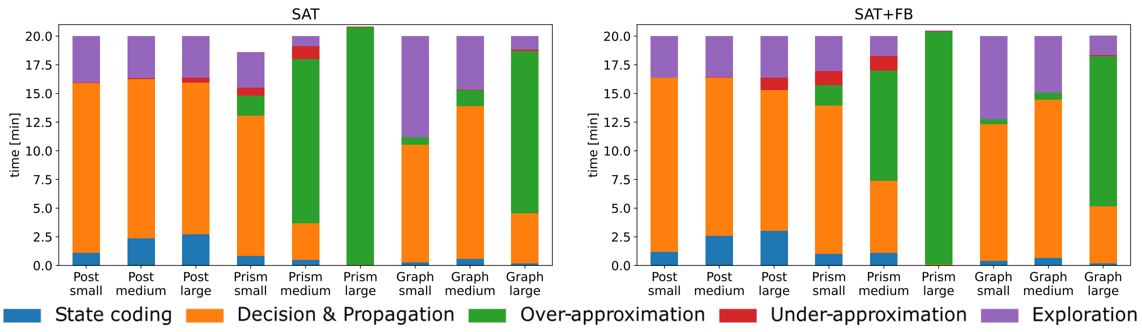


Figure 6.4: Runtime analysis of internal phases (for **Post-filtering**, **Prism based model transformation** and **Graph queries**)

RQ3: *With the current implementation the graph query-based approach is unable to beat the post filtering but performs significantly better than the model transformation-based approach.*

6.5 RQ4: Runtime analysis

Figure 6.4 shows the mean runtime of various exploration steps. The left diagram contains the data of the **SAT** case study and the right contains the data from the **SATFB**. In each diagram the left 3 column shown the runtimes for the post-filtering approach, the center 3 column for the model transformation-based approach and the right 3 columns for the graph query-based approach.

For *small models* the graph query based approach performs better than the PRISM solver. Furthermore, it has little overhead compared to the post-filtering approach.

For *medium models* the query based approach requires more time but the increment is insignificant compared to the external solver.

For *large models* the query based approach starts to struggle with complexity but does not exclusively dominate whole design space exploration.

RQ4: *The query-based approach shows significant improvements compared to the earlier solver extension but for complex models still foreshadow scalability issues.*

6.6 Threats to validity

Internal threats are that the query engine may reevaluate matching queries even when it is not required due to the implementation. This can have a negative effect on the overall performance of the synthesis tool. Furthermore we opted to not use caching when calculating the event probability due to initial measurements of the cache algorithm which likely improved the performance of the synthesis for *small* and *medium models* but may have negative effects for *large models*.

External threats is that the approaches were evaluated on the same domain and use only one baseline solution. This is mitigated by (1) using an external domain and case study for our measurements; (2) using two variations of the original case study with high conceptual difference and (3) an evolutionary approach already failed to provide results with identical measurement configuration.

Chapter 7

Related works

7.1 Model Transformation

Model transformations methods are used to generate a new model from an existing model in a different representation format. The main categories are model-to-model and model-to-text transformation [25]. Model-to-model transformations can be used to transform an input model to one or more output model [41]. These transformations can be either *declarative* where the transformation is defined by relations between source and target model elements but the exact way of transformation is not specified. *Imperative* model transformation is when the focus is on when and how the input model should be transformed and the element relations are not the source of the transformation rule. *Graph based* transformations uses graph transformation rules that describe when a transformation can be applied and what should the output should be. In this case both the input and the output model use some form of a graph representation. *Hybrid* approaches attempt to mix the positive properties of the earlier types and mitigate some of the limitations. A rule based model-to-mode transformation tool is the Henshin [66].

Model-to-text transformations are used for code generation or model serialization. Such code generators are (not exclusively) the Xtend [2] and Acceleo [16]. This can be used to generate analysis models from existing model like in [7].

7.2 Stochastic analysis methods

The use of stochastic methods in verification of availability, reliability, performance and other metrics are widespread.

In *Markov chains* [45] states and transitions are used to model the behavior of the system. Such Markov model consists of a finite or countably infinite set of states and transitions between these states. There are Discrete and Continuous time variations where the transitions occur according to a probability or transition rate respectively.

Another well-established formalism is the *Generalized Stochastic Petri Nets* [50, 13, 49, 55, 24]. This method uses a bipartite graph as the underlying formalism where nodes in one partition represents system states and the other partition contains the transitions. States in the model contain tokens and transitions can add or remove tokens from states. In the stochastic version such transitions fire with a specified probability.

Although our model transformation-based measurements use the Continuous Time Markov Chains as the underlying formalism it is possible to use other formalism too.

7.3 Design Space Exploration

We have mentioned but barely explained the *meta-heuristic* design space exploration approaches. These approaches rely on approaches like simulated annealing [26], tabu search [37], or evolutionary algorithms like NSGA-II [27] and eMOEA [28]. These algorithms usually support multi-objective optimizations meaning that many extra-functional objectives can be considered but they lack completeness guarantees or proof for global optimality [44, 43].

Some approaches use some form of a genotype vector to represent the architecture and perform the mutations on this vector. This method introduces explicit points of variability. Such point of variability is for example number of redundant components or function allocations. These approaches however are limited by using a fixed length genotype thus cannot handle varying number of components. A short list of example tools for such approaches are ArcheOpteryx [6], PerOpteryx [20], EvoChecker [33] and Rodes [22].

Other approaches use graph representation and transformations [4]. Some approaches like MOMoT [31] or MDEOptimiser [18] rely on the Henshin model transformation language [66] to mutate the graph representation and thus explore the design space. A limitation for these approaches is that the well-formedness constraints are hard to enforce. Such constraints can be either relaxed to optimization parameters thus a solution may violate them or encoded in the transformation rules to a limited degree [19].

7.4 Application of Decision Diagrams

Decision diagrams are widely used in architecture verification, logic synthesis and fault simulation [7]. In such cases the main benefit of using this method generally provide a manageable way to evaluate a variety of Boolean function related questions and operations like equivalence test of two Boolean functions, satisfiability of a Boolean function, synthesis from two existing function, universal and existential quantification [29].

In reliability evaluation one of the potential questions is under which conditions the system will fail or what is the smallest number of concurrent failures that results in a system failure. This is practically the enumeration of cut sets of a fault tree. In [48] Reduced Ordered Binary Decision Diagrams were applied to benchmark networks calculated the reliability within reasonable time.

Chapter 8

Conclusion and future works

In this work we addressed a severe scalability issue of logic solvers when optimizing architectures by utilizing a more efficient method with support for incremental evaluation. In the problem we included hard functional and extra-functional constraints alongside an optimization objective. We presented a theory on how to formalize extra-functional requirements combining stochastic methods with logic predicates. Then we introduced a prototype implementation and systematically analyzed the applicability of our approach from simple model evaluation to inclusion in an automated architecture synthesis tool. We compared its performance to alternatives approaches for the same problem.

Our conclusion is that this approach shows promising applicability for inclusion in performability reasoning. We conclude that stochastic predicates with Binary Decision Diagram as internal representation is applicable for formalizing extra-functional requirements on complex domains. Furthermore, such approach is practical in incremental environments.

Future works may include automated generation of the approximations from the problem specification. Additional optimization is also possible by introducing caching for performability values and addressing limitations in the implementation to improve the overall performance.

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.

Bibliography

[1] *Prism Model Checker*. URL `https://www.prismmodelchecker.org/`.

[2] *Xtend - Modernized Java*. URL `https://www.eclipse.org/xtend/`.

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

[4] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. Generative programming via graph transformations in the model-driven architecture. In *Workshop on Generative Techniques in the Context of Model Driven Architecture, OOPSLA*, 2002.

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

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

[7] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. Uml2alloy: A challenging model transformation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 436–450. Springer, 2007.

[8] Henrik Reif Andersen. An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, page 5, 1997.

[9] Davide Arcelli, Vittorio Cortellessa, Mattia D'Emidio, and Daniele Di Pompeo. EASIER: An evolutionary approach for multi-objective software architecture refactoring. In *ISCA*, pages 105–114. IEEE, 2018.

[10] AUTOSAR Consortium. Autosar model constraints. URL `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-2/AUTOSAR_TR_AutosarModelConstraints.pdf`.

[11] AUTOSAR Consortium. The AUTOSAR standard, 2013. URL `https://www.autosar.org/`.

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

[13] Simona Bernardi, Susanna Donatelli, and Giovanna Dondossola. Towards a methodological approach to specification and analysis of dependable automation systems. pages 36–51. Springer, 2004.

[14] Nikolaj S. Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νZ - an optimizing SMT solver. In *TACAS*, volume 9035 of *LNCS*, pages 194–199. Springer, 2015.

[15] Marco Bozzano and Adolfo Villafiorita. *Design and safety assessment of critical systems*. CRC press, 2010.

[16] Cédric Brun and Alfonso Pierantonio. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9 (2):29–34, 2008.

[17] RE Bryant. ªgraph-based algorithms for boolean function manipulation. º *IEEE Trans. Computers*, 35(8):677–691, 1986.

[18] Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. MDEoptimiser: A search based model engineering tool. In *MODELS*, pages 12–16. ACM, 2018.

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

[20] Axel Busch, Dominik Fuchss, and Anne Koziolek. PerOpteryx: Automated improvement of software architectures. In *ICSA*, pages 162–165. IEEE, 2019. DOI: `10.1109/ICSA-C.2019.00036`.

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

[22] Radu Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. RODES: A robust-design synthesis tool for probabilistic systems. In *QEST*, volume 10503 of *LNCS*, pages 304–308. Springer, 2017. DOI: `10.1007/978-3-319-66335-7\_20`.

[23] Milan Ceska, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of markov chains. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 172–190. Springer, 2019. DOI: `10.1007/978-3-030-17465-1\_10`. URL `https://doi.org/10.1007/978-3-030-17465-1_10`.

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

[25] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[26] Robert I. Davis and Alan Burns. Response time upper bounds for fixed priority real-time systems. In *RTSS*, pages 407–418. IEEE, 2008.

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

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

[29] Rolf Drechsler and Detlef Sieling. Binary decision diagrams in theory and practice. *International Journal on Software Tools for Technology Transfer*, 3(2):112–136, 2001.

[30] Johannes Eder and Sebastian Voss. Usable design space exploration in autofocus3. In *OSS4MDE@MODELS*, volume 1835 of *CEUR Workshop Proceedings*, pages 51–58. CEUR-WS.org, 2016. URL `http://ceur-ws.org/Vol-1835/paper08.pdf`.

[31] Martin Fleck, Javier Troya, and Manuel Wimmer. Search-based model transformations with MOMoT. In *ICMT@STAF*, volume 9765 of *LNCS*, pages 79–87. Springer, 2016.

[32] Máté Földiák, Kristóf Marussy, Dániel Varró, and István Majzik. System architecture synthesis for performability by logic solvers. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, page 43–54, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394666. DOI: `10.1145/3550355.3552448`. URL `https://doi.org/10.1145/3550355.3552448`.

[33] Simos Gerasimou, Giordano Tamburrelli, and Radu Calinescu. Search-based synthesis of probabilistic models for quality-of-service software engineering. In *ASE*. IEEE, 2015.

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

[35] Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. Model-based safety analysis for vehicle guidance systems. In *SAFECOMP*, pages 3–19. Springer, 2017.

[36] Stephen Gilmore and Jane Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In Günter Haring and Gabriele Kotsis, editors, *Computer Performance Evaluation, Modeling Techniques and Tools, 7th Int. Conf., Vienna, Austria, May 3-6, 1994, Proceedings*, volume 794 of *LNCS*, pages 353–368. Springer, 1994.

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

[38] László Gönczy, Zsolt Déri, and Dániel Varró. Model transformations for performability analysis of service configurations. In Michel R. V. Chaudron, editor, *Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, September 28 - October 3, 2008. Reports and Revised Selected Papers*, volume 5421 of *LNCS*, pages 153–166. Springer, 2008.

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

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

[41] Nafiseh Kahani, Mojtaba Bagherzadeh, James R Cordy, Juergen Dingel, and Daniel Varró. Survey and classification of model transformation tools. *Software & Systems Modeling*, 18(4):2361–2397, 2019.

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

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

[44] Joshua D. Knowles and David Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evol. Comput.*, 8(2):149–172, 2000.

[45] Heiko Koziolek and Franz Brosch. Parameter dependencies for component reliability specifications. *Elec. Note. Theor. Comput. Sci.*, 253:23–38, 2009.

[46] Mirco Kuhlmann, Lars Hamann, and Martin Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In *TOOLS*, volume 6705 of *LNCS*, pages 290–306, 2011.

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

[48] Hung-Yau Lin, Sy-Yen Kuo, and Fu-Min Yeh. Minimal cutset enumeration and network reliability evaluation by recursive merge and bdd. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, pages 1341–1346. IEEE, 2003.

[49] Juan Pablo López-Grao, José Merseguer, and Javier Campos. From UML activity diagrams to stochastic Petri nets: application to software performance engineering. In *WOSP*, pages 25–36. ACM, 2004.

[50] István Majzik, András Pataricza, and Andrea Bondavalli. Stochastic dependability analysis of system architecture based on UML models. In *Architecting Dependable Systems*, pages 219–244. Springer, 2002.

[51] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. 1st Joint WOSP/SIPEW Int. Conf. Perf. Eng.*, pages 105–116. ACM, 2010.

[52] Kristóf Marussy, Oszkár Semeráth, and Dániel Varró. Automated generation of consistent graph models with multiplicity reasoning. *IEEE Trans. Softw. Eng.*, 48:1610–1629, 2022.

[53] D.M. Miller and R. Drechsler. Negation and duality in reduced ordered binary decision diagrams. In *1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997*, volume 2, pages 692–696 vol.2, 1997. DOI: `10.1109/PACRIM.1997.620354`.

[54] Michael K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on computers*, 31(09):913–917, 1982.

[55] Moulaye Ndiaye, Jean-François Pétin, Jean-Philippe Georges, and Jacques Camerini. Practical use of coloured Petri nets for the design and performance assessment of distributed automation architectures. In *PNSE*, pages 113–131. CEUR-WS, 2016. URL `http://ceur-ws.org/Vol-1591/paper10.pdf`.

[56] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[57] Antoine Rauzy. Mathematical foundations of minimal cutsets. *IEEE Transactions on Reliability*, 50(4):389–396, 2001.

[58] Arend Rensink. Isomorphism checking in GROOVE. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 1, 2006.

[59] Thomas W Reps, Mooly Sagiv, and Reinhard Wilhelm. Static program analysis via 3-valued logic. In *CAV*, pages 15–30, 2004.

[60] Sven Schneider, Leen Lambers, and Fernando Orejas. Symbolic model generation for graph properties. In *FASE*, volume 10202 of *LNCS*, pages 226–243. Springer, 2017.

[61] Oszkár Semeráth, András Szabolcs Nagy, and Dániel Varró. A graph solver for the automated generation of consistent domain-specific models. Gothenburg, Sweden, 2018 2018. ACM, ACM.

[62] Oszkár Semeráth, Aren A Babikian, Sebastian Pilarski, and Dániel Varró. 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)*, pages 43–46. IEEE, 2019.

[63] Oszkár Semeráth, Aren A. Babikian, Anqi Li, Kristóf Marussy, and Dániel Varró. Automated generation of consistent models with structural and attribute constraints. In Eugene Syriani, Houari A. Sahraoui, Juan de Lara, and Silvia Abrahão, editors, *MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020*, pages 187–199. ACM, 2020. DOI: `10.1145/3365438.3410962`. URL `https://doi.org/10.1145/3365438.3410962`.

[64] Jaroslaw Skaruz, Artur Niewiadomski, and Wojciech Penczek. Evolutionary algorithms for abstract planning. In *PPAM*, volume 8384 of *LNTCS*, pages 392–401. Springer, 2013.

[65] Ghanem Soltana, Mehrdad Sabetzadeh, and Lionel C. Briand. Practical constraint solving for generating system test data. *ACM Trans. Softw. Eng. Methodol.*, 29(2): 11:1–11:48, 2020.

[66] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. Henshin: A usability-focused framework for EMF model transformation development. In *ICGT@STAF*, volume 10373 of *LNCS*, pages 196–208. Springer, 2017.

[67] Mirco Tribastone and Stephen Gilmore. 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*, pages 205–214. IEEE Computer Society, 2008.

[68] Kishor S. Trivedi, Gianfranco Ciardo, Manish Malhotra, and Robin A. Sahner. Dependability and performability analysis. In *SIGMETRICS*, volume 729 of *LNCS*, pages 587–612. Springer, 1993.

[69] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. EMF-IncQuery: An integrated development environment for live model queries. *Sci. Comput. Program.*, 98(1):80–99, 2015.

[70] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: three generations of the viatra framework. *Software & Systems Modeling*, 15(3):609–629, 2016.

[71] Liudong Xing and Suprasad V Amari. Fault tree analysis. *Handbook of performability engineering*, pages 595–620, 2008.

[72] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195, 2000. DOI: 10.1162/106365600568202.