Design-time Performability Optimization of Runtime Adaptation Strategies

Martina Rapp
Max Scheerer
FZI Research Center for Information Technology
Karlsruhe, Germany
surname@fzi.de

Ralf Reussner
Karlsruhe Institute of Technology
FZI Research Center for Information Technology
Karlsruhe, Germany
reussner@kit.edu

ABSTRACT

Self-Adaptive Systems (SASs) adapt themselves to environmental changes during runtime to maintain *Quality of Service* (QoS) goals. Designing and optimizing the adaptation strategy of an SAS regarding its impact on quality properties is a challenging problem. Usually the design space of adaptation strategies is too large to be explored manually and, hence, requires automated support to find optimal strategies. Most approaches address this problem with optimization at runtime requiring the system is already implemented. However, one expects design-time optimized adaptation strategies to more effectively maintain QoS goals than purely runtime optimized strategies. Also formal guarantees benefit from designed and analysed strategies. We claim that design-time analysis and optimization of adaptation strategies improve in particular quality properties such as performability. To address the research gap between runtime optimization and the ability to make statements on the achieved quality, we envision an approach that builds upon the concept of *Model-Based Quality Analysis* (MBQA). Many approaches in MBQA address single aspects such as formal languages for adaptation strategies, architectural description languages or QoS prediction. However, they lack integration, which leads, for example to prediction approaches assuming rather static systems. In this paper, we envision an unified approach by considering several sub-approaches as building blocks for performability-based optimization of adaptation strategies at design-time.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; **Model-driven software engineering**; **Software performance**.

KEYWORDS

performability , model-based quality analysis , adaptation strategy optimization , design-time analysis , self-adaptive systems , runtime adaptation strategies

ACM Reference Format:
Martina Rapp, Max Scheerer, and Ralf Reussner. 2022. Design-time Performability Optimization of Runtime Adaptation Strategies. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3491204.3527471

1 INTRODUCTION

In recent years a great deal of attention has been devoted to software systems with automated self-adaptation capabilities at runtime. Self-Adaptive Systems (SASs) rely on adaptation strategies to realize runtime adaptations due to changed environmental conditions [16]. The addressed optimization problem with regard to runtime adaptation strategies is to find the best adaptation option that guarantees the compliance of the overall adaptation goals without deteriorating the system quality attributes (QA) despite the uncertainty of the context of the system, that triggers the adaptation (referred to as "taming" uncertainty [35]). Due to uncertainty of the context behaviour at design-time, the SAS decision making process uses formal verification techniques at runtime by applying machine learning (ML) [18] to (i) analyze adaptation options (quantitative verification [6, 25], statistical model checking [22, 44]) or to (ii) take proactive adaptation options [29, 40] . Performing verification at runtime suffers from its inherent state-space explosion problem caused by the increasing number of state variables in the system. This results in a serious effort to perform verification both in terms of resources and time. Because adaptations are only applied in response to changes in the short term, it may result in inefficiencies since the system might execute sequences of sub-optimal adaptations. Proactive approaches that make adaptation decisions with a look-ahead horizon try to improve this drawback.

Unlike static software systems, SASs introduce an additional level of complexity induced by the permanent change of system configurations over time [33], referred to as uncertainty "Parameter over time" [13]. This additional complexity makes it inherently hard to analyse and optimize adaptation strategies at design-time. Hence, optimization of adaptation strategies is commonly performed at runtime where uncertainties are resolved by collecting runtime data. However, it is desirable to support early architectural decision making for adaptation strategies before system implementation. Even runtime optimization approaches require an initially designed strategy as a starting point. We argue that design-time analysis and optimization not only help to explore and identify possible candidates of adaptation strategies, but arguably increase the convergence behaviour of runtime optimization by identifying promising adaptation strategy candidates in advance.

Performability has brought together the disciplines for modeling and evaluating performance, reliability and availability attributes of software systems. Failure types (e.g. hardware, software and network failures) are known causes in performance engineering that lead to undesirable conditions or system degradation. Well-known works in this context extensively studied the influence of failures on system performance and developed modeling and analysis techniques to evaluate the ability of static software systems to meet quality requirements in uncertain situations [20, 28]. We see the challenge, that designing adaptation strategies which ensure continuous operation in uncertain scenarios even in the presence of faults is a key aspect of SASs.

Model-based Quality Analysis (MBQA) has quite a success story in modelling and simulating software architectures to predict quality attributes such as performance or reliability (e.g. Palladio approach [31] for static systems, Simulizar [4] for SASs). Moreover, there have been some efforts in developing *Domain-Specific Languages* (DSLs) to specify adaptation strategies [8, 36]. We claim that combining MBQA with state-of-the-art DSLs are the two main building blocks for design-time optimization of runtime adaptation strategies.

In this paper, we envision an approach that builds on the MBQA concept. We discuss key features of a DSL required for optimization of runtime adaptation strategies. We outline how such a DSL can be integrated into the MBQA approach of [34] as an unified approach for performability-based optimization of adaptation strategies at design-time. Finally, we propose our vision on optimizing rule-based adaptation strategies at design-time that may complement several lines of research with tools to improve the adaptation logic during design-time for higher runtime effectiveness.

2 PROBLEM STATEMENT

In this section, we formulate the optimization problem faced by software engineers at design-time. In previous works [33, 34], we consider an SAS as *Markov Decision Process* (MDP). An MDP is formally described by a tuple (S, A, t, r) that comprises two sets, namely the set of states S and the set of actions A. The transition function $t : S \times A \times S \longrightarrow [0, 1]$ evaluates the probability to transition to state s_{t+1} at time $t + 1$ given the current state s_t and action a_t at time t. The deterministic function $r : S \times A \times S \longrightarrow \mathbb{R}$ is known as reward function that returns a value R that is called reward and corresponds to a real number $R \in \mathbb{R}$. We consider the engineering challenge to develop a policy or adaptation strategy $\pi : S \longrightarrow A$ that determines the action to be taken in each state so that the sum of rewards generated is maximized over time. We encode quality objectives as rewards such that decisions of a policy π are evaluated w.r.t. quality requirements.

There are a variety of policies π for system adaptation that must be taken into account by a software engineer to meet quality requirements. Let Π be the set of all possible policies that we denote as *Policy Space*. As stated before, each policy $\pi \in \Pi$ determines (w.r.t. transition function t) how an SAS moves through the state space and generates a reward over time: $R_0, R_1, ..., R_T$. Let $v_\pi(T) := \sum_{i=0}^{T} R_i$ be a function that accumulates all generated rewards of a policy π until a fixed time instance T (based on the value function introduced in [37]). The function $v_\pi(T)$ induces a partial order on set Π: $\pi \geq \pi' \iff v_\pi(T) \geq v_{\pi'}(T)$. We consider the optimization problem as search in policy space Π for an optimal policy π^* that maximizes the accumulated reward over time such that:

$$\forall \pi \in \Pi : v_{\pi^*}(T) \geq v_\pi(T) \tag{1}$$

In this paper, we restrict the optimization problem to performability-related quality objectives. However, it is worth noting that the optimization problem generalizes other quality dimensions (e.g. strategy-related qualities such as stability properties) which can be encoded as reward or directly reflected in the reward function r.

3 STATE OF THE ART

3.1 Languages for specifying adaptation strategies

This section summarizes existing work on self-adaptation languages to formalize adaptation strategies both at runtime and design-time.

3.1.1 Languages and tools. There are various formal languages (general-purpose and modeling languages) and tools to specify adaptation strategies, mainly applied at runtime. CoBRA [23] realizes adaptation strategies as dynamic Java AOP aspects; [12] utilizes the Prism-MW architectural middleware platform to specify adaptation strategies as Java classes; Starfish [5] provides a policy specification language based on event-condition-actions paradigm; PBAAM [17] and StarMX [3] specify policies as expert system rules. IRIDIUM [2] represents adaptation strategies by adaptation rules as a tree-based composition of expressions. Expressions include operators that are mapped to actions at a configurable evaluation rate. Model-transformation languages use graph-grammars to specify adaptations on the model level: UML-based graph transformation approach for domain-specific model transformation implementations [1], Story diagrams [15], *GRAF* adaptation management [11]. [41] presents a model-driven approach specifying adaptation strategies in terms of model transformation rules using Triple Graph Grammars. [42] proposes a high-level graph-based architectural (re)configuration language by modeling architectures with categorical diagrams and reconfiguration by algebraic graph rewriting. Formal modeling approaches (e.g. Darwin [26], ArchWare [30]) use π-calculus semantics to specify reconfigurations of distributed systems.

3.1.2 DSL. There are various works on DSLs for adaptation strategy specification both at runtime and design-time The runtime-modeling DSL *Stitch* [8] for architecture-based self-adaptation comprises a set of key features to codify self-adaption at runtime, e.g a DSL for self-adaptation, architectural abstractions as first-class entities, QoS-based choice, constraints and event-condition-actions. The runtime-modeling DSL *S/T/A (strategies/tactics/actions)* [21] describes runtime adaptations in component-based system architectures based on architecture-level system QoS models by separating knowledge about possible adaptation steps from the actual adaptation plans. [36] presents a design-time modeling DSL to express self-adaptation actions and the impact on system performance by specifying actions together with their transient effects expressed as resource demands.

3.2 Performability analysis and modeling of (dynamically) reconfigurable Component-Based Software Systems (CBSS)

This section summarizes existing work on performability modeling and analysis of (dynamically) reconfigurable CBSS. Meyer [28] introduced the *performability* concept by a general modeling framework. The framework models the performance of a system S over a utilization period T as a random variable Y that takes values in a set A of accomplishment levels. As a result each element of A represents a possible performance outcome that can be attained by S. Thus the performability model is defined as a stochastic process X referred to as base model X. Changes in structure, internal state and environment can all have an influence on system performance [28]. Haverkort et al. [20] proposes a performability modeling and analysis approach based on a (Semi-)Markov reward process. The approach uses behavioural decomposition by breaking down the overall base model X into (i) a *structural model* (i.e. for describing changes in the system structure) and (ii) a family of *performance models* (i.e. analytical models such as queuing networks to evaluate changes in the structural model). Rewards are used to quantify performance gains and losses of system state changes based on certain events. Roughly speaking, rewards are determined each time a new event occurs or when adaptations are made by evaluating the changed structural model with the analytical models. The models can be applied to calculate probabilistic reward measures like the reward at some time instant t, or the total reward accumulated over some time interval $[0, t]$ [19]. Introducing dynamic reconfiguration features to CBSS adds even more complexity to QoS assessment in terms of design and verification of (non-)functional requirements [19]. Grassi et al. [19] propose a model-driven approach to support the analysis of CBSS focusing on non-functional quality attribute assessment (i.e. performance and reliability) by using *performability* models as analytical models. The *performability* concept can be easily extended to other kinds of reconfigurable systems by refining the structural model to a *reconfiguration model* describing changes in the system configuration by manual intervention of system administrators [19].

3.3 Optimization of adaptation strategies

This section summarizes existing work on optimization approaches of adaptation strategies classified into classical optimization approaches and machine learning (ML).

3.3.1 Classic Optimization Approaches. Ewing and Menasce [14] employ various heuristic algorithms such as hill-climbing, beam search, neighbourhood filtering, simulated annealing and evolutionary programming to optimize architecture search and service selection in SOA software systems. Peropteryx [27] uses a multi-objective evolutionary algorithm to identify a set of pareto-optimal architecture candidates to support architectural decision making by detailed exploration. De Gyvés Avila and Djemame [10] propose a *Fuzzy-logic* based optimization approach based on historical and real QoS analysis data by estimating the benefit of adaptation through service selection. Sutton [37] applies *Dynamic programming* based optimization approaches in the context of reinforcement learning.

3.3.2 ML. Saputri and Lee [32] present a systematic literature research (SLR) that investigates the application of various ML approaches in SASs. They identified eight types of ML techniques to address adaptation: reinforcement learning (RL), fuzzy learning, regression, Bayesian theory, clustering, neural networks, decision trees and genetic algorithms whereas the two top most approaches are RL and fuzzy learning. Gheibi et al. [18] present an SLR on applying ML in SAS by systematically listing the problems tackled by machine learning in SAS.

In summary no approach like our envisioned approach for design-time optimization of runtime adaptation strategies yet exists.

4 RESEARCH GAP

Current SAS research mainly focus on system runtime models by observing their evolution over time.

Most DSLs to specify adaptation strategies are solely applied at runtime on already implemented software systems (see section 3.1.2). Optimization takes places on the initially defined software architecture based on collected runtime data during system operation. Thus, adaptation decisions are based on a well-founded data basis. In contrast, design-time approaches facilitate a higher degree of abstraction by omitting extraneous information while still delivering reliable analysis results. The proposed approach of Stier and Koziolek [36] is the only existing DSL approach for modeling and analysing adaptations at design-time known to us. The approach details a reconfiguration with its enclosed actions by mapping their impact and effect on system performance but lacks a mechanism to describe a suitable set of rules required for optimization purposes. From this, it is necessary to identify the lacking but required elements of an adaptation DSL (e.g. parameters, if-then-rules, set of rules) to describe the currently missing parameters and variation points required for design-time optimization.

In the past performability evaluation was traditionally applied for design-time analyses of static systems. The approach does not cover structural changes of the system configuration over time resulting from triggered actions of selected dynamic reconfigurations due to changed environmental conditions at runtime. With reference to Grassi et al. [19] we suggest to extend this concept to SAS as they are a more recent representative of dynamic reconfigurable systems. In particular, we see performability as an starting point to evaluate and optimize adaptation strategies with respect to designing resilient software systems. Hence, we claim that applying performability evaluation at design-time on SAS can help to build more resilient software systems by preventing system degradation through the design of effective adaptation strategies. Therefore, we propose to refine the traditionally used Markov Reward models by expanding them to MDPs where adaptations form the semantically equivalent concept of actions. This extension allows us to explore the impact and effects of adaptation actions on system QoS attributes and lay the foundation for subsequent optimization approaches.

To the best of our knowledge, there is no approach for the optimization of adaptation strategies at design-time. This is arguably due to several uncertainties associated with the development process of an SAS at design-time [13]. However, advances in MBQA enable the modeling and simulation of software-architectures to predict quality attributes such as performance (Palladio approach [31], Simulizar [4]). Simulizar supports the simulation of adaptation strategies but neglects the evaluation of their long-term effects by comparison of different strategies. In addition, the work of [27] builds upon MBQA to optimize architectures of static software systems w.r.t. several quality attributes.

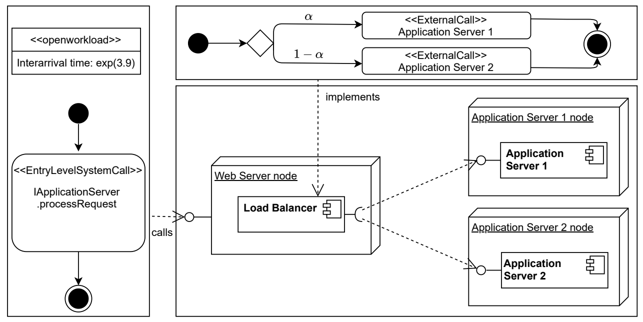
<<openworkload>>
Interarrival time: exp(3.9)
<<EntryLevelSystemCall>>
IApplicationServer .processRequest
calls
α
1 − α
<<ExternalCall>> Application Server 1
<<ExternalCall>> Application Server 2
implements
Web Server node
Load Balancer
Application Server 1 node
Application Server 1
Application Server 2 node
Application Server 2

Figure 1: Znn.com-based system taken from [34]

Thus, the research gap that this work focuses on is whether MBQA is also suitable for optimizing (w.r.t. the optimization problem introduced in section 2) adaptation strategies of SASs at design-time. Finally, the question of which optimization approach to use (see section 3.3) must be thoroughly evaluated as each approach has its own requirements (e.g. training data, simulated environments).

5 VISION

An abstract system representation is the basic prerequisite for SAS design-time analyses and when investigating adaptation strategies. By creating abstract system representations in terms of models, *MBQA* provides means to explore various architectural design alternatives together with their impact on the SAS's quality attributes. These techniques are applicable along the whole development phase.*MBQA* supports software architects in elaborating adaptation strategies by considering the effects that adaptation strategies can have on the SAS's quality attributes resulting in well-founded design decisions. There are already various model-driven approaches available in literature to specify each of the aspects we highlight hereafter individually. No approach exists yet that combines all aforementioned aspects into a single unified meta model to perform design-time optimization. In the following, we envision such an unified approach.

5.1 Motivating example

We motivate our vision by considering the widely used SAS community example system *Znn.com* [9]. The version we use here originated from [34]. The system is managed by an SAS to distribute the incoming load in order to keep the system responsive (see Figure 1). The system contains a load balancer and two individual application server components each on a separate node. The system encounters overload scenarios with high workloads that deteriorates system performance. Since the context of this work is performability, we transfer the system into a scenario that focuses on performability. Thus both application server nodes are subject to individual node failures. Regarding performability, hardware failures (i.e. failures of server nodes) are considered as a second factor that potentially degrades the system performance. We can take the situation of high workloads and the unavailability of application server 1 as an example. Thus, the adaptation problem becomes more complex because the system must not only remain responsive in high load scenarios but also deal with situations where hardware resources are not available to distribute the incoming load as best as possible. With respect to performability, an optimized adaptation strategy might consider various aspects, namely variations in the applied threshold values, an optimally balanced load distribution factor, the provisioning time and number of required replicated resources to keep the system responsive in case of failures.

5.2 Envisioned meta model

As we consider the adaptation strategy covering the overall MAPE-K loop, this section first discusses the unified *MAPE-K meta model* representing the MAPE-K loop. The meta model serves as conceptual basis for our discussion of the prerequisites and requirements for a formal language to describe adaptations strategies at design-time in general and the set of rules applied during the planning phase in particular.

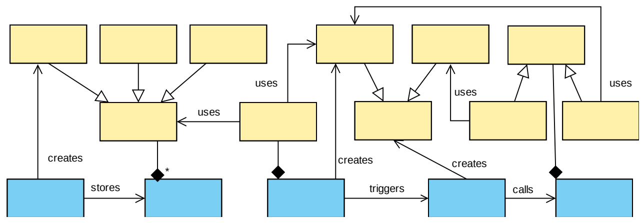

Figure 2: Envisioned meta model - DSL

5.2.1 MAPE-K meta model. SAS design commonly follows the MAPE-K concept [24]. Modeling the information of the various aspects contained herein at system design time calls for the existence of a proper meta model. We sketched a first DSL version for our envisioned meta model in figure 2. Therefore the unified *MAPE-K meta model* should consist of the following sub-meta models organized in different sub-packages (SP) by analogy of MAPE-K loop: The *monitor SP* defines the available monitors to observe system properties by collecting runtime data. The *analyze SP* contains the set of rules to evaluate the collected monitor data. The *planning SP* defines the set of rules and specified model-transformations. The *execute SP* is implicitly represented by the specific application of the determined adaptation. The *knowledge SP* contains the current system state and the knowledge about the variability points in form of a *variability meta model* containing the required parameters. In addition to the SPs that represent the classical MAPE-K phases, we introduce the *set of rules SP*. It contains the set of rules that refers and accesses the concepts of the MAP(E)-K SPs or to put it more precisely both the analyse and planning phases. Also, the *set of rules SP* specifies the if-then-rules that decide based on the available monitors which actions should be executed during the execute phase. The MAPE-K representation by means of a unified meta model predefines the available information that can be analyzed while deriving suitable system adaptations in case of unexpected situations. Thus it would be appropriate to design a formal language for describing adaptation strategies in a similar way such that its partial aspects can be assigned to individual stages of the MAPE-K loop. We see the need of a formal language reflected by a unified meta model that should include at least the following aspects:

In the *monitor phase* the system observes distinct system properties by collecting runtime data. Following the common practice as described by Garlan et al. [16] this is usually achieved by the definition of suitable monitors for system properties of interest which are then accessible via respective sensors. This allows the observation of different system properties at runtime. Thus the language needs a formalism to represent the system properties of interest by specifying monitors through suitable metrics that describe the desired system properties.

The *analyze phase* evaluates the previously collected data based on a set of rules to detect abnormal system behaviour. The collected monitor data supply the events the system must react on. Thus the language must provide ways to describe the different event types. Events can arise from the external system environment (e.g. node failures) and from internal (e.g. software failures) sources. Also the language needs a formalism to represent a set of rules to allow the specification of certain conditions and criteria that requires either system adaptation by new planning or else continue working with the current system configuration. Of particular interest is the specification of conditions and criteria because the same concept needs to be reapplied in another context of the follow-up planning phase. Thus, reusing the same formalism for set of rules specification is a desirable objective. We believe that the conditions and criteria should be at least expressed as *Service Level Objective* (SLO) specifications and environmental conditions.

The *planning phase* evaluates a predefined set of rules to determine the required adaptations for the detected abnormal system behaviour. Thus, the language needs a formalism to allow the definition of a set of rules that can be evaluated and applied during the planning phase. The set of rules must allow to represent Event-Condition-Action (E-C-A) rules. The formulation of the condition part can be reused from the analyze phase. Basically E-C-A rules are expressed in the form of *if (condition) then (action)* statements. Conditions specify the evaluation of monitor data for given events. They reflect the point in time and the present system state at which a modification is required. Actions describe the resulting modifications of the system's structure and behaviour. Conditions can be considered as the temporal part. Whereas actions can be seen as the operational part. The planning phase determines a set of actions, that when applied transitions the system back into a steady state.

The *execute phase* carries out the planned reconfiguration by executing the selected actions. These actions are bundled in a concrete adaptation with a fixed parameterization. The impact on the overall system behaviour may become critical, if inappropriate operations are conducted due to a poor planning result.

We argue that the requirements for a formal specification language for system adaptation strategies must consider different aspects of the MAPE-K loop. A dedicated DSL will support the expression of adaptation rules and adaptation actions. A further challenge is the incorporation of our meta model with the existing ones SLO and monitor [4] and variability [27].

5.2.2 Set of rules and control structure representation. This section takes a closer look on the requirements regarding the representation of the set of rules and control structures. The planning phase addresses the set of rules interpretation in order to select appropriate adaptations that are executed in the subsequent execute phase. However, one must distinguish between the definition of a set of rules and its actual execution. The former requires a formalism to provide an abstract representation of a set of rules. The latter requires a formalism to explicitly define the execution logic of a set of rules, i.e. the representation of suitable control structures like if/else statements, loops, switches, break , etc.. As model-transformations like QVTo already provide the required language elements to support control structure representation, we exclude the definition of control structures from our *set of rules meta model* and focus only to describe the set of rules on an abstract level. We will leave the definition of model-transformations based on the abstract representation of the set of rules as explicit task of an domain expert. The mere set of rules is represented by *if (condition) then (action)* rules. The planning phase evaluates the set of rules at runtime by selecting the most appropriate rule(s) which best matches the currently observed conditions. A formal language to describe a set of rules therefore requires language elements to specify *if (condition) then (action)* rules. The set of rules meta model references the unified MAPE-K meta model as input for condition and action specification. An *condition term* reflects system property values that must be fulfilled by the system. The unified MAPE-K meta model provides the observed system properties together with the monitored data as input for condition evaluation. The specification of condition terms should support the definition of simple Boolean expressions and the combination of more complex terms by linking multiple simple expressions through logical or arithmetic operators. The *action part* describes the system's structural and behavioural adaptations. It defines individual modification steps in detail as abstract parameterized operations. We consider actions as an abstract representation of adaptations which are realized through model-transformations. The specification of actions should support the construction from both atomic operations and complex operations build from multiple atomic operations. Again, the unified MAPE-K meta model provides input information about the current system state and knowledge about the variability points as part of its knowledge SP.

5.2.3 Action parameterization using variability. This section discusses possible optimization parameters. Abstract actions describe the structural and behavioural modifications that the system must carry out to comply with its quality objectives. Actions are defined in an abstract way in the *then part.* Each abstract action will be mapped to a respective model-transformation. We assume that the model-transformations are defined in advance by domain experts. Part of the planning phase is the fine planning of the model-transformation as action parameterization by referencing a variability model instance of the variability meta model. In general a large number of different variations exist resulting from the design space of all potential adaptations. From a design-time perspective, however, only a limited set of adaptations can contribute to the overall system goals in a meaningful way. Thus the main goal should be to support decision making in such a way that it helps to determine optimal variability parameters for adaptations. From our point of view, we distinguish between the structural description of an adaptation and its parameterization, e.g. the specific amount of structural elements like components or connectors. Therefore, it is sufficient to describe adaptations and their variability points on an abstract level and leave the formulation of adaptation specification as part of the action implementation.

The *variability* meta model encodes the system's variability points. It specifies the system structure parts that can be adapted or changed accordingly. Also, the variability meta model shall specify the system's behavioural part, e.g. the distribution factor of a load balancer component. The integration of behavioural specifications regarding control flow into the variability meta model is still an open question and requires further investigation. We consider the variability as part of the *knowledge* that will be applied during the planning phase to determine concrete adaptations. The variability model is again part of the set of rules. There, the variability model is parameterized with support of using e.g. optimization techniques. Each adaptation inherently contains a certain degree of variability for which one would like to find out the best possible option. It thus becomes possible to classify the optimization parameters into *when* and *how* parameters. The first category describes when a rule is triggered, the second category defines the variability how to approach the environmental event, i.e. the concrete adaptation.

5.2.4 Modeling uncertainties in SAS at design-time. We think a challenging question is the way of modeling uncertainties in SAS at design-time and in particular how these uncertainties shall be resolved for the optimization at design-time.

SASs are exposed to a range of potential sources of uncertainties. Weyns defines uncertainty in SAS as "any deviation of deterministic knowledge that may reduce the confidence of adaptation decisions made based on the knowledge" ([43, p. 139]). The major challenge in SAS lies on *taming uncertainty* by "providing guarantees for the compliance of a self-adaptive system to its adaptation goals despite the uncertainty the system is exposed to" ([43, p. 137]). This involves dealing with the contrariness of uncertainties on the one hand and guarantees for the system goals on the other hand. Weyns [43, 140] lists sources of uncertainties classified into four groups each with a set of characteristic sources of uncertainty that relate to the: (i) system itself, which can refer to both the managed system and the managing system, (ii) goals of a SAS, (iii) context of a SAS, (iv) humans involved in the functioning or operation of a SAS. The manifold sources of uncertainties can manifest themselves at design-time, runtime or both. We see the challenge to identify the potential candidates of uncertainty sources that have an impact at design-time and provide modeling means for their adequate representation. In particular, we argue that depending on the system concerns under study, only a subset of uncertainty sources must be considered for optimization at design-time. Optimizing runtime adaptation strategies with regards to performability, means to handle disruptive events like unexpected failures or changes in the demand. The meta model shall be capable of modelling the uncertainties relevant for the particular system concerns of interest.

SAS typically apply formal verification techniques at runtime to make adaptation decisions for taming uncertainty by following a two-step decision making process. In step 1 the analyzer evaluates the adaptation options by using formal verification techniques. In step 2 the planner selects the best adaptation option based on the previous verification results. The most widely used runtime instances of this approach in SAS literature are: (i) quantitative verification at runtime to analyze the adaptation options (QoSMOS [6], PRISM [25]), (ii) statistical model checking applied at runtime to analyze the adaptation options (ActivFORMS [22, 44]), (iii) probabilistic model checking applied at runtime to make proactive adaptation decisions ([29], mRUBiS [40]). We argue, that the selected decision making approach, i.e. one out of approaches (i)-(iii), has a significant impact on the design of the feedback loop. In particular, the knowledge representation heavily changes both in terms of the required data and parameters that need to be represented in terms of adaptation goal specification and parameterization of the used quality models. Also, the accessed information from the knowledge base by the analyzer and planner slightly differs in terms of data layout and access mode. Thus regarding modeling, we see the challenge of finding a common abstract representation of the knowledge base and the decision making process that is suitable for design-time optimization.

5.3 Vision on optimizing rule-based adaptation strategies

In the previous sections, we discussed formal modeling languages for representing and optimizing adaptation strategies. Based on these two building blocks, we now envision how such strategies can be optimized at design-time.

Optimizing strategies at design-time is challenging due to the inability of observing extrinsic events in the environment (e.g. varying workloads), intrinsic events of the system (e.g. resource failures) or quality attributes (e.g. response time) that are crucial for optimization. In addition, optimization requires interaction with the environment to assess the quality of decisions. Hence, a framework is required that (i) simulates the operating environment of an SAS and its interaction, (ii) simulates the adaptation process triggered by the adaptation strategy to represent system changes and (iii) predicts quality attributes (e.g. response time). Such a framework allows not only to evaluate optimization approaches, but most importantly to answer the question whether it is generally possible to optimize adaptation strategies at design-time.

Advances in MBQA have proven to be successful to model, simulate and predict quality attributes (see section 4). Based on MBQA, we developed a model-based framework [34] for SASs which satisfies the aforementioned requirements $(i) - (iii)$. The framework enables validation of adaptation strategies in terms of their effectiveness by probabilistically simulating the environment, the adaptation process and predicting quality attributes to assess the impact of decisions made by the strategy. The framework serves as foundation and is supposed to be extended for optimizing adaptation strategies.

Based on the framework, the eligibility of several optimization approaches can be evaluated at design-time. Considering the state-of-the-art optimization approaches, we focus on two well-researched approaches, namely reinforcement learning (e.g. [45]) and dynamic programming (e.g. [37]). In addition, we consider a promising ML technique termed *Learning Classifier Systems* [38] that is focused on optimizing rule-based systems. Furthermore, the chosen optimization approach also raises questions about the requirements of training data (kind and origin).

Quality attributes predicted at design-time deviate from the real values measured at runtime. Therefore, a strategy optimized at design-time deviates from a strategy optimized at runtime. The deviation, however, decreases with the accuracy of the prediction tools and provides a near-optimal candidate that is more suitable than an initially designed strategy. There are good reasons to move the optimization to runtime, for instance, all possible environmental conditions cannot be enumerated or even foreseen at design time, or uncertainty can only be resolved at runtime. However, a challenging question is to which extend is it still possible to already identify and drop unsuited candidates and thus significantly reduce the adaptation space later explored at runtime by identifying a meaningful pre-selected set of adaptation strategies. Van et al. [39] apply ML to reduce large adaptation spaces of SAS. We argue, that already the exploration of the adaptation space at design-time can help to identify inappropriate adaptation strategies in advance that might violate system quality attributes at runtime. As stated in our problem statement (see section 2) our goal regarding design-time optimization is twofold; for one thing selecting one strategy out of a given set of possible adaptation strategy families and secondly, optimizing the selected strategy in terms of its optimal parameterization. We argue, that by identifying candidates of appropriate adaptation strategy families at design-time, significantly reduces the adaptation space that needs to be explored later at runtime. A good candidate of an adaptation strategy family distinguishes itself by the characteristic that it meets the SAS's specific uncertainties in terms of proper quality attribute fulfillment. Thus our goal is to select the best candidate of an adaptation strategy family with its optimal parameterization. Optimization approaches aim to select the best candidate out of a given set of objects in relation to the properties to be optimized. In our case we try to find an adaptation strategy from a set of adaptation strategy families. Runtime optimization approaches start their selection process without any previous knowledge. Thus finding such a strategy takes considerable time. We argue, that a preceding design-time optimization step facilitates to restrict the candidates of strategies to a reduced set with a positive impact on the system QA. While starting the runtime optimization from a restricted set of candidates, chances are high that the approach will find an optimal strategy quicker and thus converges arguably faster. We argue that design-time analysis and optimization not only helps to explore and identify possible candidates of adaptation strategies, but arguably increase the convergence behaviour of runtime optimization.

SAS offline/online approaches (e.g. MOSAICO [7]) have the disadvantage that both the managing and the managed system are already implemented to some extend. Thus the interplay between the managing system in terms of the applied adaptation strategy and its managed system regarding different characteristics is fixed. Consequently required changes are much more expensive and time consuming compared to an abstract representation of the system at design-time that allows the exploration of different design alternatives. Thus another advantage of design-time optimization compared to offline/online approaches is the possibility to additionally investigate the impact an adaptation strategy might have on its managed system. In particular, the execution of actions during the adaptation phase can have a significant impact on the managed system and its QA. A design-time approach could also support in reasoning about the actual cause of the problem and its effect on the system architecture as it is directly observable in the model.

As the optimization of SAS adaptation strategies is typically performed at runtime but rather neglected at design time this also calls for a trade-off between how much optimization should or actually can be done effectively at design time and how much at runtime. We think that we need to deal with the challenge to what degree the integration of design-time and runtime optimization are supportive for the design of SAS adaptation strategies.

Especially when less domain knowledge is available, design-time optimization frees software engineers from the burden of initially engineering strategies without knowing how they behave at runtime. Finally, runtime optimization can use the near-optimal strategy as a starting point to increase the convergence behaviour.

6 CONCLUSION AND FUTURE WORK

In this paper, we envisioned a model-based approach for design-time performability optimization of runtime adaptation strategies. At first, we reviewed a representative selection of the state-of-the-art. Then we outlined a formal language for describing adaptation strategies and a model-based framework based on previous work to integrate and optimize modeled adaptation strategies at design-time. We plan to implement our envisioned approach within the MOSAIC project which aims to develop tools for designing self-adaptive IoT systems in the cloud. In addition, we plan to implement a self-adaptive IoT system that we use as evaluation case study to enable the comparison of our design-time predictions with runtime results of the case study.

ACKNOWLEDGMENT

This paper was partly funded by the Federal Ministry of Education and Research under grant number 01IS18069A.

REFERENCES

[1] Aditya Agrawal, Gabor Karsai, and Feng Shi. 2003. A UML-based graph transformation approach for implementing domain-specific model transformations. In *International Journal on Software and Systems Modeling.*

[2] Sandro Santos Andrade and Raimundo Jose de Araujo Macedo. 2009. A Non-Intrusive Component-Based Approach for Deploying Unanticipated Self-Management Behaviour. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '09).* IEEE Computer Society, USA, 152–161. https://doi.org/10.1109/SEAMS.2009.5069084

[3] Reza Asadollahi, Mazeiar Salehie, and Ladan Tahvildari. 2009. StarMX: A Framework for Developing Self-Managing Java-Based Systems. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '09).* IEEE Computer Society, USA, 58–67. https://doi.org/10.1109/SEAMS.2009.5069074

[4] Matthias Becker. 2017. *Engineering Self-Adaptive Systems with Simulation-Based Performance Prediction.* Ph. D. Dissertation. Universität Paderborn, Heinz Nixdorf Institut, Softwaretechnik.

[5] Themistoklis Bourdenas and Morris Sloman. 2010. Starfish: Policy Driven Self-Management in Wireless Sensor Networks. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (Cape Town, South Africa) *(SEAMS '10).* Association for Computing Machinery, New York, NY, USA, 75–83. https://doi.org/10.1145/1808984.1808993

[6] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS Management and Optimization in Service-Based Systems. 37, 3 (2011), 387–409. https://doi.org/10.1109/tse.2010.92

[7] Javier Cámara, Bradley Schmerl, Gabriel A. Moreno, and David Garlan. 2018. MOSAICO: offline synthesis of adaptation strategy repertoires with flexible trade-offs. 25, 3 (2018), 595–626. https://doi.org/10.1007/s10515-018-0234-9

[8] Shang-Wen Cheng and David Garlan. 2012. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software* 85, 12 (dec 2012), 2860–2875. https://doi.org/10.1016/j.jss.2012.02.060

[9] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. 2009. Evaluating the Effectiveness of the Rainbow Self-Adaptive System. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*

(SEAMS '09). IEEE Computer Society, USA, 132–141. https://doi.org/10.1109/SEAMS.2009.5069082

[10] S. de Gyves Avila and K. Djemame. 2013. Fuzzy Logic Based QoS Optimization Mechanism for Service Composition. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering* (2013-03). IEEE, San Francisco, CA, USA, 182–191. https://doi.org/10.1109/sose.2013.28

[11] Mahdi Derakhshanmanesh, Mehdi Amoui, Greg O'Grady, Jürgen Ebert, and Ladan Tahvildari. 2011. GRAF: Graph-Based Runtime Adaptation Framework. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Waikiki, Honolulu, HI, USA) *(SEAMS '11)*. Association for Computing Machinery, New York, NY, USA, 128–137. https://doi.org/10.1145/1988008.1988026

[12] George Edwards, Joshua Garcia, Hossein Tajalli, Daniel Popescu, Nenad Medvidovic, Gaurav Sukhatme, and Brad Petrus. 2009. Architecture-Driven Self-Adaptation and Self-Management in Robotics Systems. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '09)*. IEEE Computer Society, USA, 142–151. https://doi.org/10.1109/SEAMS.2009.5069083

[13] Naeem Esfahani and Sam Malek. 2013. *Uncertainty in Self-Adaptive Software Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 214–238. https://doi.org/10.1007/978-3-642-35813-5_9

[14] John M. Ewing and Daniel A. Menascé. 2014. A Meta-Controller Method for Improving Run-Time Self-Architecting in SOA Systems. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering* (Dublin, Ireland) *(ICPE '14)*. Association for Computing Machinery, New York, NY, USA, 173–184. https://doi.org/10.1145/2568088.2568098

[15] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. 2000. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In *Theory and Application of Graph Transformations*. Springer Berlin Heidelberg, 296–309. https://doi.org/10.1007/978-3-540-46464-8_21

[16] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (oct 2004), 46–54. https://doi.org/10.1109/mc.2004.175

[17] John C. Georgas and Richard N. Taylor. 2008. Policy-Based Self-Adaptive Architectures: A Feasibility Study in the Robotics Domain. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems* (Leipzig, Germany) *(SEAMS '08)*. Association for Computing Machinery, New York, NY, USA, 105–112. https://doi.org/10.1145/1370018.1370038

[18] Omid Gheibi, Danny Weyns, and Federico Quin. 2021. Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review. *ACM Trans. Auton. Adapt. Syst.* 15, 3, Article 9 (aug 2021), 37 pages. https://doi.org/10.1145/3469440

[19] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. 2007. A Model-Driven Approach to Performability Analysis of Dynamically Reconfigurable Component-Based Systems. In *Proceedings of the 6th International Workshop on Software and Performance* (Buenes Aires, Argentina) *(WOSP '07)*. Association for Computing Machinery, New York, NY, USA, 103–114. https://doi.org/10.1145/1216993.1217011

[20] Boudewijn R. Haverkort. 2001. *Performability modelling : techniques and tools*. Wiley, Chichester [u.a.].

[21] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. 2013. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. 8, 1 (2013), 73–89. https://doi.org/10.1007/s11761-013-0144-4

[22] M. Usman Iftikhar and Danny Weyns. 2014. ActivFORMS: Active Formal Models for Self-Adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Hyderabad, India) *(SEAMS 2014)*. Association for Computing Machinery, New York, NY, USA, 125–134. https://doi.org/10.1145/2593929.2593944

[23] Florian Irmert, Thomas Fischer, and Klaus Meyer-Wegener. 2008. Runtime Adaptation in a Service-Oriented Component Model. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems* (Leipzig, Germany) *(SEAMS '08)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/1370018.1370036

[24] J.O. Kephart and D.M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (jan 2003), 41–50. https://doi.org/10.1109/mc.2003.1160055

[25] Marta Kwiatkowska, Gethin Norman, and David Parker. 2002. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer Berlin Heidelberg, 200–204. https://doi.org/10.1007/3-540-46029-2_13

[26] Jeff Magee and Jeff Kramer. 1996. Dynamic Structure in Software Architectures. *SIGSOFT Softw. Eng. Notes* 21, 6 (oct 1996), 3–14. https://doi.org/10.1145/250707.239104

[27] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. 2010. Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering* (San Jose, California, USA) *(WOSP/SIPEW '10)*. Association for Computing Machinery, New York, NY, USA, 105–116. https://doi.org/10.1145/1712605.1712624

[28] Meyer. 1980. On Evaluating the Performability of Degradable Computing Systems. *IEEE Trans. Comput.* C-29, 8 (aug 1980), 720–731. https://doi.org/10.1109/tc.1980.1675654

[29] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive Self-Adaptation under Uncertainty: A Probabilistic Model Checking Approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2786805.2786853

[30] Ron Morrison, Dharini Balasubramaniam, Flavio Oquendo, Brian Warboys, and R. Mark Greenwood. 2007. An Active Architecture Approach to Dynamic Systems Co-evolution. In *Software Architecture*. Springer Berlin Heidelberg, 2–10. https://doi.org/10.1007/978-3-540-75132-8_2

[31] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann. 2016. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, Cambridge, MA.

[32] Theresia Ratih Dewi Saputri and Seok-Won Lee. 2020. The Application of Machine Learning in Self-Adaptive Systems: A Systematic Literature Review. *IEEE Access* 8 (2020), 205948–205967. https://doi.org/10.1109/ACCESS.2020.3036037

[33] Max Scheerer, Jonas Klamroth, Ralf Reussner, and Bernhard Beckert. 2020. Towards Classes of Architectural Dependability Assurance for Machine-Learning-Based Systems. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Association for Computing Machinery, New York, NY, USA, 31–37. https://doi.org/10.1145/3387939.3388613

[34] Max Scheerer, Martina Rapp, and Ralf Reussner. 2020. Design-Time Validation of Runtime Reconfiguration Strategies: An Environmental-Driven Approach. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, Washington, DC, USA, 75–81. https://doi.org/10.1109/ACSOS49614.2020.00028

[35] Gabriela Félix Solano, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. 2019. Taming Uncertainty in the Assurance Process of Self-Adaptive Systems: A Goal-Oriented Approach. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Montreal, Quebec, Canada) *(SEAMS '19)*. IEEE Press, Montreal, Quebec, Canada, 89–99. https://doi.org/10.1109/SEAMS.2019.00020

[36] Christian Stier and Anne Koziolek. 2016. Considering Transient Effects of Self-Adaptations in Model-Driven Performance Analyses. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)* (2016-04). IEEE, Venice, Italy, 80–89. https://doi.org/10.1109/QoSA.2016.14

[37] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

[38] Ryan J. Urbanowicz and Will N. Browne. [n. d.]. *Introduction to Learning Classifier Systems*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-55007-6

[39] Jeroen Van Der Donckt, Danny Weyns, Federico Quin, Jonas Van Der Donckt, and Sam Michiels. 2020. Applying Deep Learning to Reduce Large Adaptation Spaces of Self-Adaptive Systems with Multiple Types of Goals. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Association for Computing Machinery, New York, NY, USA, 20–30. https://doi.org/10.1145/3387939.3391605

[40] Thomas Vogel. 2018. MRUBiS: An Exemplar for Model-Based Architectural Self-Healing and Self-Optimization. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (Gothenburg, Sweden) *(SEAMS '18)*. Association for Computing Machinery, New York, NY, USA, 101–107. https://doi.org/10.1145/3194133.3194161

[41] Thomas Vogel and Holger Giese. 2010. Adaptation and Abstract Runtime Models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (Cape Town, South Africa) *(SEAMS '10)*. Association for Computing Machinery, New York, NY, USA, 39–48. https://doi.org/10.1145/1808984.1808989

[42] Michel Wermelinger, Antónia Lopes, and José Luiz Fiadeiro. 2001. A Graph Based Architectural (Re)Configuration Language. In *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Vienna, Austria) *(ESEC/FSE-9)*. Association for Computing Machinery, New York, NY, USA, 21–32. https://doi.org/10.1145/503209.503213

[43] Danny Weyns. 2020. *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. WILEY IEEE COMPUTER SOC PR. https://www.ebook.de/de/product/39016291/danny_weyns_an_introduction_to_self_adaptive_systems_a_contemporary_software_engineering_perspective.html

[44] Danny Weyns and M. Usman Iftikhar. 2019. ActivFORMS: A Formally-Founded Model-Based Approach to Engineer Self-Adaptive Systems. (2019). arXiv:1908.11179 [cs.SE]

[45] Tianqi Zhao, Wei Zhang, Haiyan Zhao, and Zhi Jin. 2017. A reinforcement learning-based framework for the generation and evolution of adaptation rules. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, IEEE, Columbus, OH, USA, 103–112.