Model-based Simulation at Runtime for Self-adaptive Systems

Danny Weyns,
Katholieke Universiteit Leuven, Belgium
Linnaeus University, Växjö, Sweden
Email: danny.weyns@kuleuven.be

M. Usman Iftikhar,
Linnaeus University
Växjö, Sweden
Email: usman.iftikhar@lnu.se

***Abstract*—Modern software systems are subject to uncertainties, such as dynamics in the availability of resources or changes of system goals. Self-adaptation enables a system to reason about runtime models to adapt itself and realises its goals under uncertainties. Our focus is on providing guarantees for adaption goals. A prominent approach to provide such guarantees is automated verification of a stochastic model that encodes up-to-date knowledge of the system and relevant qualities. The verification results allow selecting an adaption option that satisfies the goals. There are two issues with this state of the art approach: i) changing goals at runtime (a challenging type of uncertainty) is difficult, and ii) exhaustive verification suffers from the state space explosion problem. In this paper, we propose a novel modular approach for decision making in self-adaptive systems that combines distinct models for each relevant quality with runtime simulation of the models. Distinct models support on the fly changes of goals. Simulation enables efficient decision making to select an adaptation option that satisfies the system goals. The tradeoff is that simulation results can only provide guarantees with a certain level of accuracy. We demonstrate the benefits and tradeoffs of the approach for a service-based telecare system.**

***Keywords*-Self-adaptation; models and simulation at runtime; TAS exemplar;**

I. Introduction

Over the past years, various self-adaptation approaches have been proposed to deal with the dynamics and uncertainties of software systems (e.g., the availability of resources or changes of system goals may be difficult to predict at design time). Central to these approaches are feedback loops equipped with models that are updated at runtime, when new knowledge becomes available. The system uses these models to reflect upon itself and achieve its goals by adapting itself in response to changing conditions [7], [11]. With the increasing demand for self-adaptation in applications with critical goals, providing guarantees for the system goals has become an important subject of research [25], [8], [27].

One prominent approach to provide such guarantees is runtime automated verification that allows checking whether certain properties hold for the system model during operation. There is a particular interest in using stochastic models that encode system behaviour and knowledge of relevant qualities. The probabilities of the transitions can be based on value estimates provided by domain experts, but as these values may change over time they need to be updated online [12]. Verification of a stochastic model through exhaustive analysis of the state-transition graph of the system model enables to calculate expected quality properties (e.g., likelihood of failures, expected response times) for different adaptation options, allowing the system to select an option that satisfies the system goals and adapt accordingly [5], [4].

There are two issues with this state of the art approach. Encoding the system behavior and knowledge of different quality properties in a single model lacks flexibility to change goals at runtime, which is an important, but challenging type of uncertainty [11]. Furthermore, exhaustive verification suffers from the state-space explosion problem, which puts constraints on the time and resources required to perform verification, and the size of the models that can be verified. This problem becomes particular relevant for verification at runtime, when time and resources are often constrained. Optimisation techniques have been proposed, for example caching and lookahead [16], but new approaches will be required to provide guarantees for self-adaptation at runtime in an efficient way.

In this paper, we propose a novel modular approach for decision making in self-adaptive systems that is based on distinct models for each relevant quality combined with runtime simulation of the models. Distinct models support on the fly changes of goals. Simulation enables efficient decision making to select an adaptation option that satisfies the system goals. By using statistical techniques, simulation allows to provide results with a required level of accuracy. Simulation is less time and resource consuming compared exhaustive verification approaches. However, the tradeoff is that the guarantees are bounded to a certain level of accuracy.

The remainder of this paper is structured as follows. Section II discusses selected related work. In Section III, we introduce a telecare system that we use for illustration and evaluation. Section IV introduces the novel modular approach for decision making in self-adaptive systems. In Section V, we evaluate the approach using the telecare system. Section VI wraps up and outlines directions for future work.

II. Related Work

We have divided related work in three parts: i) runtime automated verification, ii) simulation in self-adaptive systems, and iii) adaptation goals. We limit the discussion to a selection of representative approaches from the huge body of work that has been developed over the past years.

There is an increasing trend in the use of formal methods at runtime in self-adaptive systems [29]. [12] represents the possible execution flows of a system at runtime with a discrete time Markov chain. The probabilities that represent uncertainties are dynamically updated with a Bayesian estimator. [13] proposes a two-step verification approach: a pre-computation step computes a set of symbolic expressions, which represent satisfaction of the requirements, a verification step then evaluates the formula by replacing the variables with values gathered at runtime. In [5], formally specified requirements are automatically analyzed using runtime model checking techniques to identify and enforce optimal configurations and resource allocations of service systems that are modeled using a Markov model. [17] uses a Markov decision model of the system that enables an interpreter to drive the execution of the system and guarantee the highest utility for a set of quality properties. [14] propose a effective lightweight filtering approach that learns and continuously updates the transition probabilities of discrete time Markov models of the system. In summary, state of the art proposes to equip the feedback loop with a stochastic model that maintains up-to-date knowledge about the relevant qualities and uncertainties of the system. This model is kept alive and used by automated verification to identify system configurations that comply with the required goals and adapt the system as required.

Simulation allows to explore many different states of the system without being prevented by an infinite (or at least very large) state space. Simulation runs can provide guarantees at different levels of fidelity, based on the level of abstraction of the model used and number of simulation runs applied. Simulation is a commonly used for evaluating novel approaches for self-adaptation [29]. Evaluation can also be used during the engineering process, e.g., to iteratively improve the design of self-adaptive systems, as in [6]. Simulation is rarely used to support decision making at runtime in self-adaptive systems. One example is [23], that presents an approach that automatically builds a dynamic model of a business process to realise service level agreements, while optimizing system resources. The prediction is based on a simulation model whose parameters are tuned at runtime. As recently pointed out [27], simulation offers interesting opportunities to provide guarantees for self-adaptive systems at runtime. However, the approach has not been well studied yet.

A variety of goal models have been proposed to deal with adaptation. We highlight a few representative examples. The RELAX language [31] allows temporal relaxation of requirements to capture uncertainty. FLAGS [1] proposes "crisp goals" specified in linear temporal logic and "fuzzy goals" specified in fuzzy temporal language. [24] distinguishes "awareness requirements" that refer to situations that require adaptation and "evolution requirements" that prescribe what to do in these situations. While several approaches support the operationalisation of adaptation based on goal models, further research is required to support solutions that allow changing adaptation goals on the fly [11], [8].

III. Tele Assistance System

The Tele Assistance System (TAS) provides health support to users in their home [5], [28]. Users wear a device that uses third-party remote services from health care, pharmacy, and emergency service providers. Fig. 1 shows the TAS workflow that comprises different services. The workflow can be triggered periodically to measure the user's vital parameters and invoke a medical analysis service. Depending upon the analysis result a pharmacy service can be invoked to deliver new medication to the user or change his/her dose of medication, or the alarm service can be invoked, dispatching an ambulance to the user. The user can also invoke the alarm service directly via a panic button.

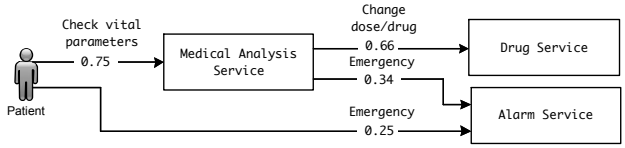

Figure 1: TAS workflow

Multiple service providers provide concrete services for Alarm service, Medical analysis service, and Drug service, abbr. by AS, MAS, and DS respectively. Concrete services have a failure rate F_rate and an invocation $Cost$. Table I shows the initial values declared by the service providers.

Table I: Third party service profiles for TAS

S.No	AS		MAS		DS	
	F_rate	Cost	F_rate	Cost	F_rate	Cost
1	0.11	4.0	0.12	4.0	0.01	5.0
2	0.04	12.0	0.07	14.0	0.03	3.0
3	0.18	2.0	0.18	2.0	0.05	2.0
4	0.08	3.0	0.10	6.0	0.07	1.0
5	0.14	5.0	0.15	3.0	0.02	4.0

As a default behavior we assume that TAS selects a particular configuration of services, e.g. $\{AS_3,\ MAS_4,\ DS_1\}$. We consider two types of uncertainties in TAS. The first one is related to the actions performed to the system. As shown in Fig. 1, we assume that on average 75% of the requests are (automatically triggered) checks of vital parameters and

25% are emergency calls invoked by the user. After checking vital parameters, depending upon the result 66% of the requests invoke the drug service, and 34% of the requests invoke the alarm service. However, these probabilities can change over time. The second uncertainty is related to the concrete services of the system. These uncertainties include the availability of services and quality parameters of running services. Depending upon load on the system, the network and other conditions the initial values of the failure rates and response times of the services are subject to change.

We apply self-adaptation to TAS to deal with uncertainty related to failures, cost, and service time. An offline analysis may find a configuration which supports the set of requirements. But as there are many uncertainties associated with TAS, there is a need for adapting the current configuration at runtime based on the actual values of these uncertainties.

IV. Modular Decision Making Approach for Self-Adaptation

We introduce the novel modular decision making approach for self-adaptation in two steps. We start with a high level overview of the model for self-adaptation we use in this research. Then we zoom in on change management, the central part of decision making for self-adaptation.

A. Model for Self-adaptation

In this research, we study architecture-based self-adaptation, where a self-adaptive system consists of a managed system that provides the domain functionality and a managing system that monitors and adapt the managed system [22], [15], [21], [26]. Furthermore, we look at managing systems that are realised with a MAPE-K based feedback loop that is divided in four components: Monitor, Analyze, Plan, and Execute [19], [30], that share common Knowledge (hence, MAPE-K). Knowledge comprises models that provide a causally connected self-representation of the managed system referring to the structure, behavior, goals, and other relevant aspects of the system [3].

Fig. 2 shows the high-level overview of the model for self-adaptation that we use in our research. The model conforms to the three-layer model of Kramer and Magee [21].

The Managed System is the software that is subject of adaptation. At a given time the managed system has a particular configuration that is determined by the arrangement and settings of the running components that make up the Managed System. The set of possible configurations can change over time. We refer to the different choices for adaptation from a given configuration as the *adaptation options*, or alternatively the *possible configurations*. Adapting the managed system means selecting an adaptation option and changing the current configuration accordingly. We assume that the Managed System is equipped with probes and effectors to support monitoring the system and apply adaptations. How these probes and effectors are realised is out

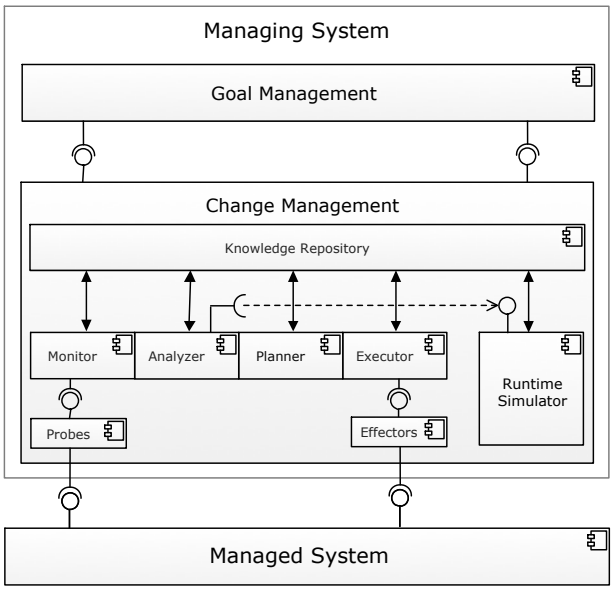

Figure 2: High-level model for self-adaptation

of scope of this paper. The Managed System is deployed in an environment that can be the physical world or computing elements that are not under control of the Managed System. The Managed System and the environment may expose stochastic behavior.

The Managing System comprises two sublayers: Change Management and Goal Management. Change Management adapts the Managed System at runtime, using the MAPE components of the feedback loop that interact with the Knowledge Repository. The MAPE components can trigger one another, for example, the Analyser may trigger the Planner once analysis is completed. The Analyser is supported by a Runtime Simulator that can run simulations on the models of the Knowledge Repository during operation. In our current work on modular decision making for self-adaptation, we consider single MAPE loops. Extensions to interacting MAPE loops is subject of our future work. Goal Management enables to adapt Change Management itself. Goal Management offers an interface to the user to change the adaptation logic, for example, to change the models of the knowledge repository, or change the MAPE functions. Changing the adaptation software should be done safely, e.g., in quiescent states [20]. We do not elaborate in these technical aspects as the primary focus of this paper is on Change Management.

Example – TAS is an example of a Managed System and a configuration of this system is an orchestration of a set of concrete services. The adaptation options for TAS are the different combinations of available concrete services. These possible configurations can change over time, e.g., when concrete services are no longer available or new

services appear. The current configuration can be adapted by replacing one or more concrete services that provide better quality of service. TAS exposes stochastic behavior both with respect to the actions invoked to the system as changes in the quality and other parameters of the concrete services. To deal with the uncertainties, a Managing System is added to TAS that aims to guarantee the system goals regardless of the uncertainties.

We have developed a concrete realisation of the modular approach for decision making in self-adaptive systems that we used for evaluation (see Section V). For additional information, we refer to the project website.[1]

B. Change Management

Fig. 3 shows the key elements of Change Management. We start with explaining the elements of the Knowledge Repository. Then we explain the MAPE components.

1) Knowledge Repository: The Managed System Model and Environment Model capture the essential elements of the managed system and its environment that are needed to make adaptation decisions. The Quality Models capture the characteristics of the different qualities that are subject of adaptation. All the models are parameterised, where the parameters represent variability and/or uncertainty of model elements. A central aspect of the modular approach for decision making in self-adaptive systems is the use of distinct quality models. The approach does not assume any particular types of models; any type of model that supports simulation at runtime (if needed) can be used. In our current realisation, we use stochastic timed automate (STA) as modeling language. STA are a stochastic extension of timed automata [2], [9]. A timed automaton is a finite state machine extended with a set of real-valued clocks. STA allow to represent uncertainties by probabilities associated with transitions in the models. Furthermore, STA models can be parameterized to capture variations or changes in the system or the environment.

For each quality model, a set of Simulation Queries is provided. A simulation query enables determining an estimate for the value of a quality property for a possible configuration by running one or more simulations on the configuration model with the corresponding quality model. A simulation query is formulated as $simulate\ N[<= bound]\{E1, .., Ek\}$, where N is the number of simulation runs to be performed, $bound$ is the time bound on the simulation runs, and $E1, ..., Ek$ are state-based expressions that need to be monitored during the simulation. The time is the simulation time, where each tick represents a period of wall clock time. So, a query simulates the system N times over a given period of time to provide insight to the user on the behavior of the system for the expressions $E1, ..., Ek$.

[1] homepage.lnu.se/staff/daweaa/ActivFORMS/Model-based-simulation.htm

The Adaptation Goals define the objectives that need to be realised by the MAPE components. The modular approach for decision making in self-adaptive systems supports any type of representation of adaptation goals. In our current realisation, we represent adaptation goals as a set of rules defined over the quality properties that are subject of adaptation.

Finally, the Current Adaptation Options list the possible configurations of the managed system that may be ranked based on the adaptation goals. The Current Plan comprises the set of actions that are required to adapt the current configuration to the selected adaptation option.

Example – The Managed System and Environment Model of TAS capture essential aspects of the telecare system and its users. E.g., the environment model represents the behavior of the user, where the preferences for user actions can be expressed as probabilities. The model of the managed system captures the essential elements of the TAS workflow. The concrete services that are used by the system are parameters in the model. In TAS, we use distinct models for failure behavior of services, service times of service invocations, and cost for using TAS. Each quality model is provided with a simulation query that enables determining an estimate for the value of a quality property for a possible configuration. For example, a query to estimate the expected average failure rate of a possible configuration based on 35 simulations, each for a period of 100 time units could be:

$$simulate\ 35[<= 100]\{AssistanceService.failureRate\}$$

Examples of adaptation goals in TAS are:

$R1: averageCost <= 8\ (\times 10^{-3})$

$R2: averageResponseTime <= 2.5\,s\ (\times 10^{-3})$

The first rule states that the average cost per invocation should not exceed 8 units per 1000 invocations. The second rule states that the average response time per service invocation should be below 2.5 seconds per 1000 invocations.

The current adaptation options is a list of the possible configurations of the managed system with estimates of the respective qualities. Here is an example entry in this list:

$\{MAS_1, DS_4, AS_5\}$, fRate= 0.12, cost = 8.5, sTime = 17

The current plan in TAS contains a set of required service replacements to transfer the current configuration to the new configuration selected by the decision making mechanism.

2) MAPE Components: The Monitor component tracks the behavior of the managed system and the environment through probes updating the runtime models. The monitor comprises distinct Updating components for each runtime model. The approach does not assume any particular type of updating mechanism. Examples are basic updating mechanisms that update the parameter values of models based on changes in the underlying system or the environment, or more advanced mechanisms such as Bayesian and reinforcement learners.

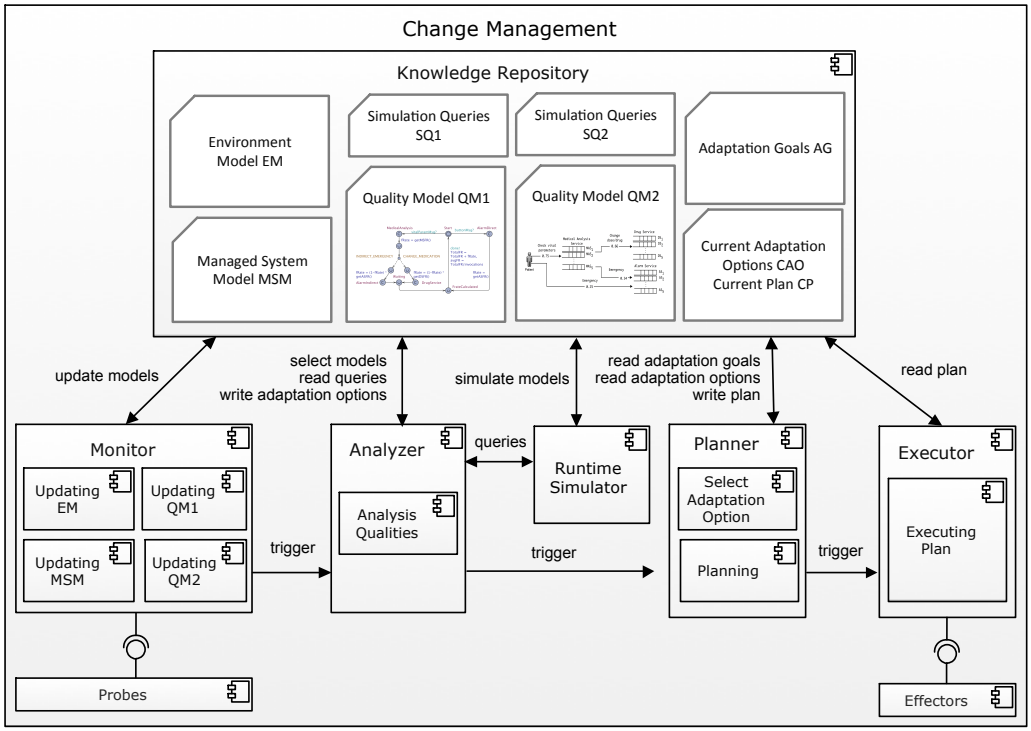

Figure 3: Change Management of the modular approach for decision making in self-adaptive systems

The Analyzer component analyses the up to date knowledge of the models to determine whether an adaption is required. To that end, the Analyzer uses the Runtime Simulator to estimate the qualities of each possible configuration. Concretely, the Analyzer starts with selecting models (managed system model for a concrete configuration, environment model, and a particular quality model. Then the Analyzer invokes the simulation query for the given quality model. The parameters of the simulation query (N and bound) are configured based on the required accuracy. The simulator uses the selected models to compute an estimate for the quality using the simulation query. This estimate is returned to the Analyzer. In our current research, we use the *standard error of the mean* (SEM) as a measure to determine the accuracy of the simulation queries. The SEM quantifies how precisely a simulation result represents the true mean of the population (and is thus expressed in units of the data). SEM takes into account the value of the standard deviation and the sample size. Concretely, we use the relative SEM (RSEM), which is the SEM divided by the sample mean and expressed as a percentage. For example, a RSEM of 5 % represents an accuracy with a SEM of plus/minus 0.5 for a mean value of 10. Evidently, more accurate results (better estimates) require smaller RSEM values and thus more simulation runs. Currently, we empirically determine the number of simulation runs required for a particular accuracy based on offline experiments. Once the Analyzer has performed an analysis of all the qualities for all the possible configurations, it writes the adaption options to the Knowledge Repository.

The Planner component ranks the adaptation options based on the adaptation goals and creates a plan for the highest ranked option. This plan is then used by the Executor component to adapt the managed system.

Example – The TAS Monitor comprises Updating components for the different runtime models. The preferences of user actions of the environment model are periodically updated based on information directly retrieved from the service providers. The Updating mechanism of the managed system model tracks concrete services that disappear or new concrete services that become available and updates the knowledge accordingly. For the properties of the different quality models (failure rates, response time, queue lengths) we use simple learning algorithms that track the averages of the respective properties over a period of time. The TAS Analyzer uses the Uppaal-SMC engine [9] to perform the simulations of the runtime models. To determine the parameters of the simulation queries we performed a series of offline experiments for different TAS configurations with different qualities and parameter settings. Based on these results, we have set the number of required simulations to 50 for a RSEM of 10% and to 125 for a RSEM of 5%. For details of this experiments, we refer to the project website. The selection of the adaption option for TAS is based on sequentially applying the rules that define the

adaptation goals. As an example, for an adaptation scenario that considers failure rates, cost, and service time, first the possible configurations with a failure rate below a required value are selected. From this set the possible configurations with a cost below a certain value are selected. Finally, the configuration with the lowest service time is selected and a plan is generated and executed to adapt the system.

V. Evaluation

We now evaluate the modular approach for decision making in self-adaptive systems using a prototype realisation of TAS. We start with presenting the different runtime models that we used in the experiments. Then, we present the results of a first series of experiments in which we consider failure rates and service invocation costs. Next, we extend the first case taking into account the service time of TAS, demonstrating the flexibility of the modular approach. We conclude with experiments that show the scalability of the approach by comparing the adaptation time with an exhaustive approach based on runtime quantitative verification.

A. Runtime Models

The runtime models of TAS are modeled using stochastic timed automata (STA); an extension of timed automata with stochastic behavior. STA communicate through broadcast signals and shared variables creating networks of STA.

Fig. 4 shows the environment model that represents the actions invoked to TAS. We use a scenario where each time tick either a sample of the vital parameters is taken from the user (with a probability value of *p_ANALYSIS*) or the alarm button is pushed by the user (with a probability *p_EMERGENCY*, which is equal to *1 - p_ANALYSIS*). A sample is sent for analysis via the signal *medicalAnalysis!*, while pushing the alarm button triggers an emergency call via the *emergency!* signal. The probabilities are updated at runtime. After invoking an action, TAS processes the request. Once the service completes, the user is notified via the *serviced?* signal.

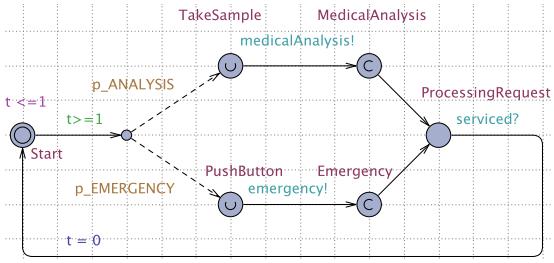

Figure 4: Environment model

Fig. 5 shows the model of the managed system. The system starts with assigning concrete services to the workflow using the function *assignServices(AD, MAS, DS)* and then waits for incoming requests. The parameters *AD, MAS, DS* can be assigned any concrete instance that is available of the alarm services, medical analysis services, and drug services respectively. Upon receiving a request for medical analysis or an emergency call, respectively the signals *vitalParamMsg!* or the *buttonMsg!* are sent to the workflow model (i.e., a selected quality model as we will discuss below). The managed system model keeps track of the number of invocations, which is required by the quality models to calculate averages. After invoking the workflow the request is processed in the *Processing* state until the *done?* signal is received, which triggers a notification to the Environment model via the *serviced!* signal.

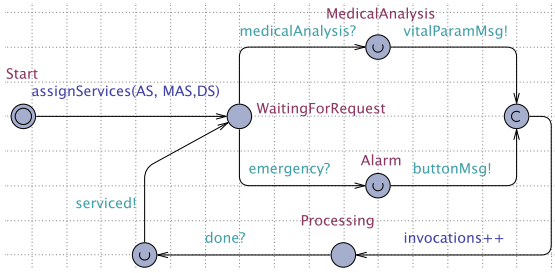

Figure 5: Managed System model

Fig. 6 shows the quality model of failure behavior of the assistance service of TAS. The model allows estimating failure rates of assistance service invocations. An assistance service invocation fails if any of the services that is needed fails. If the alarm service is directly invoked calculating the failure rate is straightforward and equal to the actual failure rate of the concrete alarm service that is used (*getASFR()*). If the medical analysis service is invoked the failure rate is calculated by summing the failure rate of the concrete analysis service plus a fraction of the failure rate of the concrete alarm service or drug service, depending on service that is required service (that is, the path that is taken based on the probabilities *p_INDIRECT_EMERGENCY* and *p_CHANGE_MEDICATION*). Finally, the average failure rate is calculated using the total number of invocations. By increasing the number of simulations of the model, the estimated average failure rate will get closer to the real average. The simulation queries to estimate failure rates with an accuracy of RSEM 10% and 5% respectively are:

$$simulate 1[<= 50] AssistanceService.avgFRate$$
$$simulate 1[<= 125] AssistanceService.avgFRate$$

Fig. 7 shows the quality model to calculate estimated costs of using the assistance service. The total cost of an invocation is equal to the sum of the costs of the concrete services used. Similarly to estimating the failure rate, the total cost per invocation depends on the path that is taken in the workflow, based on the probabilities of the actions taken and types of services invoked. The simulation queries to estimate average cost with RSEM 10% and 5% respectively

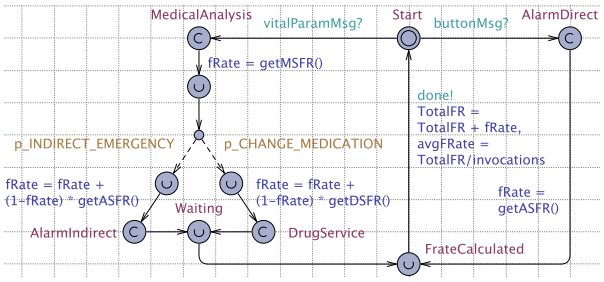

Figure 6: Quality model: Failure rate

are:

$$simulate1[<= 50]AssistanceService.avgCost$$
$$simulate1[<= 125]AssistanceService.avgCost$$

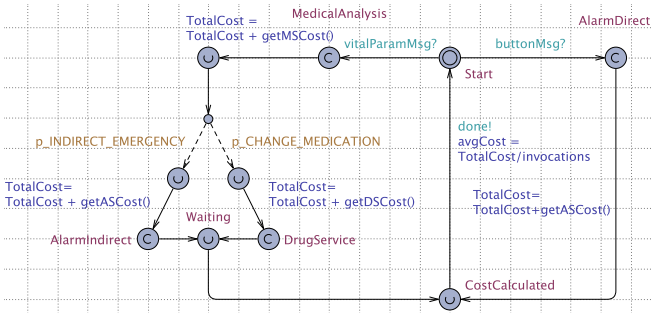

Figure 7: Quality model: Cost

B. Experiments with Two Qualities

In the first experiment, we focus on two qualities: failure rate and cost. Concretely, adaptation should guarantee the following quality requirements:

R1. $failureRate <= 1.5\ (\times 10^{-3})$
R2. $averageCost <= 8\ (\times 10^{-3})$
R3. *Subject to $R1$ and $R2$ being satisfied, the $failureRate$ should be minimized.*

We use the TAS setting with five concrete instances per service type as described in Section III and Table I in particular. We added uncertainty to probabilities of the service failure rates and invoked requests based on a normal distribution with a standard deviation of 0.05 and 0.10 respectively. The experiments are performed on a Macbook with 2.5 GHz Core i7 processor, and 16 GB 1600MHz DD3 RAM.

Fig. 8 shows the simulation results of all possible configurations of TAS and selected configuration for adaptation at a given point in time. Each configuration is represented by a dot that shows the estimated values for failure rate and average cost of that configuration. The rectangle area demarcated by the dotted lines contains all the possible configurations that comply with requirements $R1$ and $R2$. Based on requirement $R3$ the configuration with the lowest

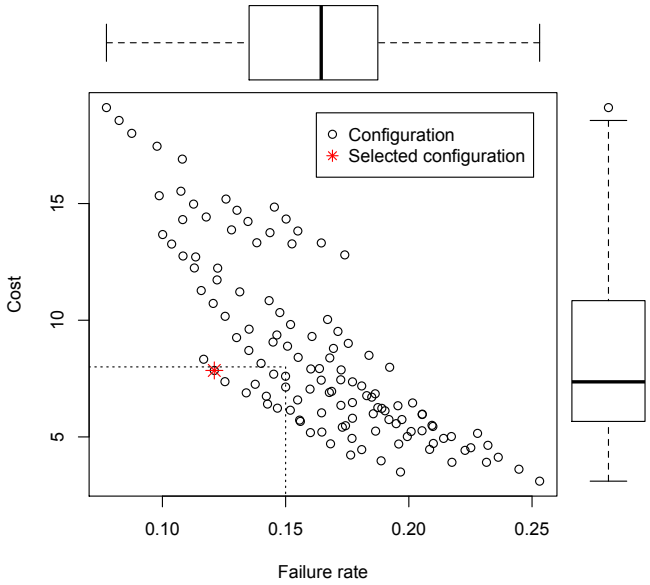

Figure 8: Adaptation options with the selected configuration

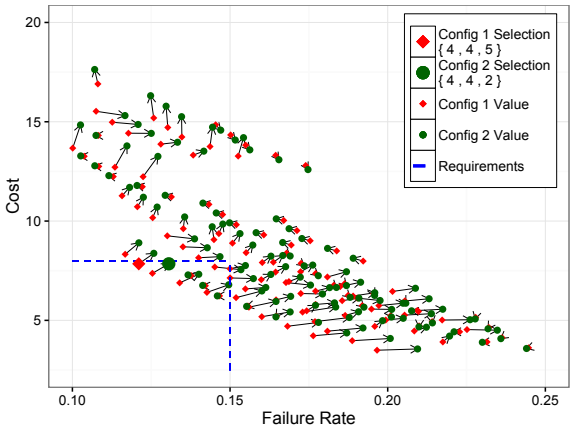

Figure 9: Differences between two configurations

failure rate is selected for adaption. Note that the dot for each configuration is an estimate with an accuracy that is based on the simulation query used, in this particular case a query with accuracy of SEM 5%.

Fig. 9 shows how the change of estimated quality properties over time due to uncertainties, incl. changes in the user behavior and the quality properties. We can see that the cost requirement ($R2$) of the initially selected configuration in Fig. 8 (Config 1 in Fig. 9) is violated. Hence, another configuration is selected now (Config 2). This figure underpins the importance of adaptation at runtime.

We now discuss the results of adaptation. Fig. 10 shows the result of a series of 10000 invocations of the assistance service with simulation queries of RSEM 5% and 10%. The managed system checks for adaptation every 500 invocations. Aligned with requirements $R1$ and $R2$, we have

calculated the failure rates and average costs over a moving window of 1000 invocations. The boxplots for failure rate show that the required value of 0.15 is satified. The boxplots for the average cost show that the cost remains below the required 8 units at all times with some exceptions for RSEM of 10%. The boxplots for adaptation times show the tradeoff of getting stronger guarantees based on different accuracy levels, i.e., RSEM of 5% takes almost double the adaptation time as RSEM of 10%; the major part of this time is used for runtime simulation to estimate the quality properties of the adaptation options.

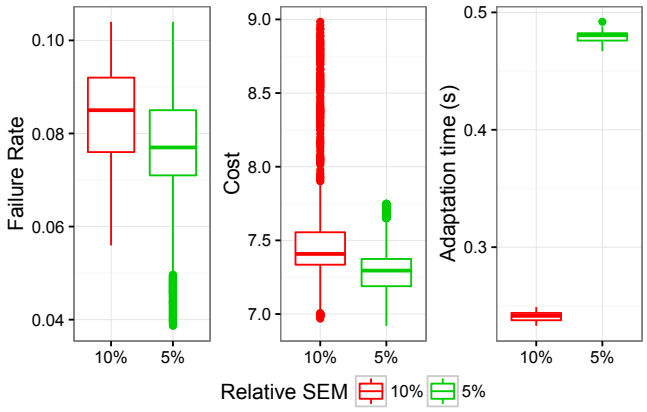

Figure 10: Results over 10000 runs for the first experiment

C. Experiments with Three Qualities

We now demonstrate the flexibility of our approach by adding additional goals and rules. Concretely, we add a new quality property that is subject of adaptation: service time. Service time comprises two components: the response time of invocations of concrete services and the waiting time due to queues with pending invocations. Service time is an important concern in TAS as users may need to get treatments quickly.

Table II shows initial estimated response times (in sec) and queue lengths (pending invocations) for the concrete services.

Table II: Average queue lengths and response times

S.No	AS		MAS		DS	
	Rtime	Qlen.	Rtime	Qlen.	Rtime	Qlen.
1	5.7	3	11.0	1	8.0	1
2	7.3	2	9.4	4	7.7	3
3	3.8	5	20.0	2	11.0	5
4	9.5	1	8.0	6	10.0	2
5	18.6	4	9.0	3	15.0	4

For the adaptation goals, we replace $R3$ as follows:

R3'. *Subject to R1 and R2 being satisfied, the serviceTime should be minimized.*

To realise this new requirement, we need to add a new quality model to the knowledge repository, update the adaptation goals, add an Updating component for the new quality model in the Monitor component and extend the decision making logic to handle the new quality. Our current realisation supports such updates on the fly, based on runtime executable formal models [18], [10] (see also the project website).

Fig. 11 shows the quality model to estimate service times. The service time per invocation is accumulated by the time the request has to wait in the queues plus the actual execution time, depending on the path that is taken.

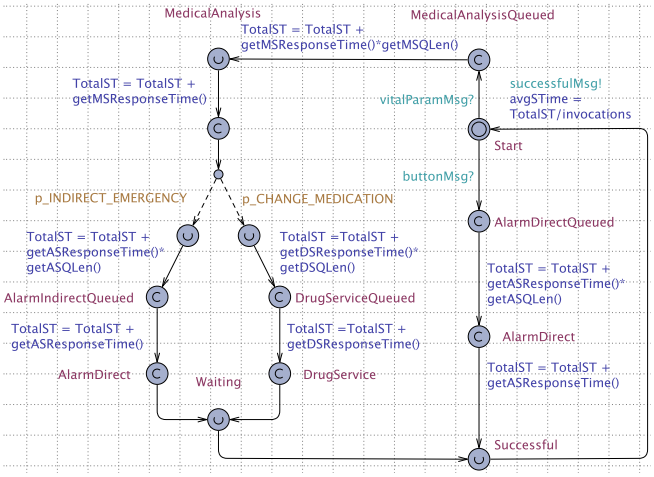

Figure 11: Quality model: Service time

For simulation, we used the following query:

$simulate1[<= 50]AssistanceService.avgSTime$
$simulate1[<= 125]AssistanceService.avgSTime$

Fig. 12 shows the simulation results of all possible configurations at a given point in time and the configuration that is selected for adaptation. Among all the configurations that comply to $R1$ and $R2$ (valid configurations), the one with the lowest service time is selected for adaptation.

Fig. 13 shows the result of 10000 invocations of the assistance service for RSEM of 10% and 5%. As for the first experiment, we show the quality properties for a sliding window of 1000 invocations. The boxplots for failure rate show similar results for RSEM 5% and 10%; both realise $R1$. The boxplots for cost show that RSEM of 5% gives slightly better results, but RSEM of 10% violates $R2$ some times. The boxplots for service times are similar. Similar to the first experiment, the boxplots for adaptation times show that RSEM of 5% takes almost double the time as RSEM of 10%.

D. Scalability

To conclude, we compare the scalability of our approach with Runtime Quantitative Verification (RQV), an exhaus-

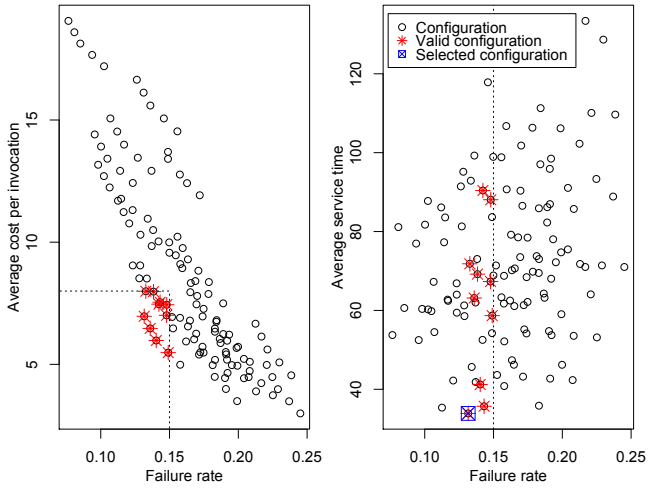

Figure 12: Adaptation options with selected configuration

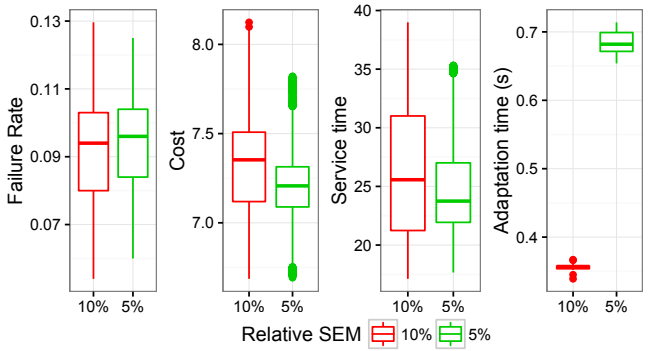

Figure 13: Results over 10000 runs for the first experiment

tively verification technique. For RQV, we used a minimal discrete time Markov model of TAS and used PRISM for verification of quality properties. We used the same setup as in first experiment and systematically increased the number of concrete services per service type. The probabilities of actions and quality properties are assigned randomly. Fig. 14 compares the time required for adaptation. The graph shows that the modular approach is significantly faster than RQV. On the other hand, RQV guarantees the required qualities, while the accuracy of simulation is bound to the selected RSEM. Additional test are required to further compare the tradeoffs of both approaches.

VI. Conclusion and Future Work

This research contributes a novel modular approach for decision making in self-adaptive systems. The approach is based on distinct runtime models for different qualities supporting on the fly changes of quality models and adaptation goals. As the approach uses simulation to estimate required quality properties, it is inherently more efficient compared to exhaustive approaches. This is confirmed by initial test

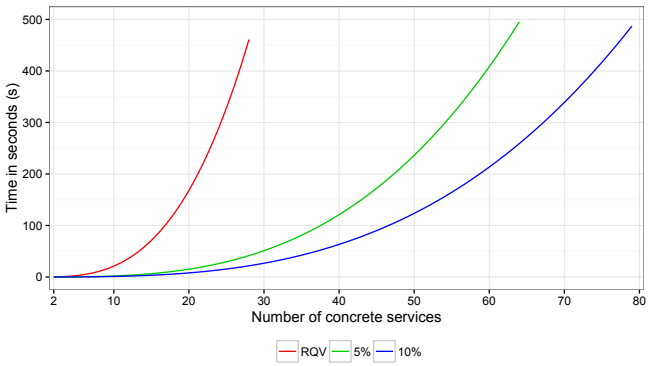

Figure 14: Comparison of scalability with RQV

results. However, the consequence of using simulation is a reduction of accuracy, which may lead to temporal violations of requirements. On the other hand, the approach allows to tradeoff the accuracy provided by the time that is required to adapt. In the future, we are planning an depth comparison between the proposed approach and exhaustive approaches. We also plan to extend the type of queries by studying how we can use statistical model checking at runtime to support efficient decision making in self-adaptive systems.

References

[1] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 125–134, Washington, DC, USA, 2010. IEEE Computer Society.

[2] J. Bengtsson and W. Yi. *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. 2004.

[3] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. *Computer*, 42(10):22–27, Oct 2009.

[4] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, Sept. 2012.

[5] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, May 2011.

[6] J. Cámara, G. A. Moreno, D. Garlan, and B. Schmerl. Analyzing latency-aware self-adaptation using stochastic games and simulations. *ACM Trans. Auton. Adapt. Syst.*, 10(4):23:1–23:28, Jan. 2016.

[7] B. H. Cheng et al. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[8] B. H. C. Cheng et al. *Models@run.time: Foundations, Applications, and Roadmaps*, chapter Using Models at Runtime to Address Assurance for Self-Adaptive Systems, pages 101–136. Springer International Publishing, Cham, 2014.

[9] A. David, K. G. Larsen, A. Legay, M. Mikuăionis, and D. B. Poulsen. Uppaal smc tutorial. *Int. J. Softw. Tools Technol. Transf.*, 17(4):397–415, Aug. 2015.

[10] D. G. de la Iglesia and D. Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 10(3):15:1–15:31, Sept. 2015.

[11] R. de Lemos et al. Software engineering for self-adaptive systems ii: International seminar, dagstuhl castle, germany, october 24-29, 2010 revised selected and invited papers. chapter Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[12] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 111–121, Washington, DC, USA, 2009. IEEE Computer Society.

[13] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 341–350, New York, NY, USA, 2011. ACM.

[14] A. Filieri, L. Grunske, and A. Leva. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 200–211, Piscataway, NJ, USA, 2015. IEEE Press.

[15] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct. 2004.

[16] S. Gerasimou, R. Calinescu, and A. Banks. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 115–124, New York, NY, USA, 2014. ACM.

[17] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 33–42, Piscataway, NJ, USA, 2013. IEEE Press.

[18] M. U. Iftikhar and D. Weyns. Activforms: Active formal models for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 125–134, New York, NY, USA, 2014. ACM.

[19] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[20] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. Softw. Eng.*, 16(11), Nov. 1990.

[21] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. In *2007 Future of Software Engineering*, FOSE '07, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.

[22] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, ICSE '98, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society.

[23] A. Solomon, M. Litoiu, J. Benayon, and A. Lau. Business process adaptation on a tracked simulation model. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 184–198, Riverton, NJ, USA, 2010. IBM Corp.

[24] V. E. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven software evolution. *Comput. Sci.*, 28(4):311–329, Nov. 2013.

[25] G. Tamura et al. Software engineering for self-adaptive systems ii: International seminar, dagstuhl castle, germany, october 24-29, 2010 revised selected and invited papers. chapter Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems, pages 108–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[26] D. Weyns and T. Ahmad. Claims and evidence for architecture-based self-adaptation: A systematic literature review. In *Proceedings of the 7th European Conference on Software Architecture*, ECSA'13, pages 249–265, Berlin, Heidelberg, 2013. Springer-Verlag.

[27] D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli. Perpetual assurances in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III*. Lecture Notes in Computer Science, Springer, 2016.

[28] D. Weyns and R. Calinescu. Tele assistance: A self-adaptive service-based system exemplar. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 88–92, May 2015.

[29] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 67–79, New York, NY, USA, 2012. ACM.

[30] D. Weyns, M. U. Iftikhar, and J. Söderlund. Do external feedback loops improve the design of self-adaptive systems? a controlled experiment. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 3–12, Piscataway, NJ, USA, 2013. IEEE Press.

[31] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2), 2010.