ISSN 1884-0760

GRACE TECHNICAL REPORTS

Qualitative Analysis for Multiple Adaptation Loops

Kenji Tei Ryuichi Takahashi Nicolas D'Ippolito
Hiroyuki Nakagawa Shinichi Honiden

GRACE-TR 2015–07 June 2015

CENTER FOR GLOBAL RESEARCH IN
ADVANCED SOFTWARE SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF INFORMATICS
2-1-2 HITOTSUBASHI, CHIYODA-KU, TOKYO, JAPAN

WWW page: http://grace-center.jp/

Qualitative Analysis for Multiple Adaptation Loops

Kenji Tei†, Ryuichi Takahashi‡, Nicolas D'Ippolito⋆
Hiroyuki Nakagawa⋄, Shinichi Honiden†

† National Institute of Informatics, Japan
‡ Waseda University, Japan
⋆ Universidad de Buenos Aires, Argentina
⋄ Osaka University, Japan

June 26, 2015

Abstract

Using multiple adaptation loops enlarges the adaptation space of self-adaptive systems, but poses an adaptation chain problem. Adaptation loops are triggered not only by changes in the environment but also by changes caused by other adaptation loops. The chain of adaptations should be analyzed to validate specifications of these loops early in the development cycle of self-adaptive systems. In this paper, we focus on qualitative analysis of multiple adaptation loops to validate their specifications. Our analysis explores all possible adaptation chains and checks whether their loops are possible to lead the qualities of the managed system to the desired levels. To this end, we devise a process for constructing behavior models of adaptation loops and the managed system, and a tool for automatically transforming the behavior models into a formal model used for model checking with a reachability property. We present case studies based on extended Znn.com to validate the results of our qualitative analysis. Furthermore, we also evaluate the scalability of the analysis and show that it can be completed within a reasonable amount of time.

1 Introduction

Adaptive systems are required to automatically adapt in response to changes in their environment and goals. Such changes can occur at any point and adaptive systems are expected to cope with them by modifying their configuration and behavior at runtime without human intervention. In other words, their correct operation must be ensured by the adaptation mechanisms. In addition to correct behavior, quality attributes are also key to satisfy system's expectations. Hence, adaptation mechanisms should also

aim for guaranteeing desired levels for expected quality attributes (e.g. response time, availability, user experience). Design of adaptive systems based on adaptation loops has shown to be a promising approach to equip systems with self-adaptive capabilities. Adaptation loops typically follow a monitor-analize-plan-execute (MAPE [12]) strategy that requires the managed system to provide an interface to observe its state and act over it by changing behavior or configuration. Thus, the managed system must provide an adaptation interface.

In order to achieve its goals, complex software systems require the coordination of multiple adaptation loops that monitor and modify the managed system as it is required. To support these changes, modern software systems support a wide range of configuration options and provide one or more variation points and adaptation interfaces corresponding to these points. Recent studies[11, 5, 16, 24] have investigated the use of multiple adaptation loops for controlling variation points, as shown in Figure 1, in order to explore a wide adaptation space and to control different quality attributes. For example, Kephart et al. [11] introduced multiple adaptation loops in web application to control performance and energy consumption, and Ledoux et al. [5] also introduced them in cloud systems to manage the infrastructure of the IaaS layer and applications on the SaaS layer.

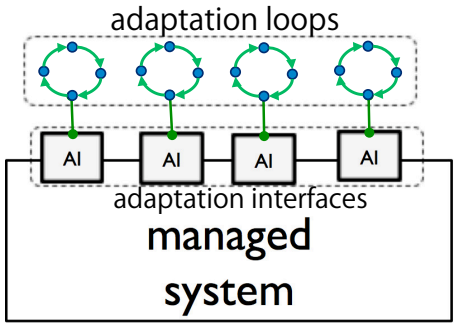


Figure 1: Self-adaptive system with multiple adaptation loops

Despite their advantage, it is not easy to construct a correct specification for multiple adaptation loops. Unlike a single loop, multiple loops pose an *adaptation chain* problem. Here, an adaptation loop is triggered not only by changes in the environment but also by changes caused by other adaptation loops. A configuration change made by one adaptation loop may affect the qualities of the system, which may in turn trigger other loops. The adaptation chain should thus be modeled and analyzed to verify the correctness of the specification. Vogel and Giese [24] proposed sequencing techniques to merge adaptation loops into one to avoid the adaptation chain, but this does

not allow concurrent execution of adaptation loops. Oliverira et al. [5] proposed a coordination and synchronization protocol for adaptation loops in cloud systems in order to improve the synergy between the loops in different layers, but this could not handle the adaptation chain problem.

This issue is not new and it has been studied in the control theory domain. However, the analysis is restricted to specific combination or hierarchies of control loops, failing to provide general approaches to combine control loops while guaranteeing their stability and convergence. We, on the other hand, propose an approach to model, analyze and verify the correctness of the specification.

We believe that model checking-based qualitative analysis is a promising and effective approach for reasoning about correctness of adaptation loops at runtime. It has been applied to self-adaptive systems, but most of the studies aimed to guarantee the behavior of the managed system [26, 27]. A few studies have attempted to apply qualitative analysis to adaptation loops [10, 15]. Iftikhar and Weyns proposed ActivFORMS[10]. The ActivFORMS enables qualitative analysis of single adaptation loops by modeling the behavior of the loop and the managed system as timed automata, and assures the deadlock freedom. However, it does not consider adaptation chains. Luckey and Engels [15] pointed out the adaptation chain problem and proposed a qualitative analysis for multiple adaptation loops. They model multiple adaptation loops and the managed system by using UML-based notation, and transform them into a labeled transition system (LTS) that reflects the state space of the adaptation chain. The qualitative analysis uses an LTS with a stability property to assure that the adaptations do not occur infinitely. However, this analysis can not determine whether or not the adaptation chains will lead the system to a desirable or undesirable state.

In this paper, we present an approach for development-time analysis of multiple adaptation loops based on qualitative analysis. Our analysis checks the reachability property to see if the given multiple adaptation loops are unable to maintain the qualities of the system at the desired levels, in a qualitative way. Intuitively, the analysis clarifies undesirable situations of the adaptation loops. Model checking exhaustively explores all possible adaptation chains leading from the current state of the managed system and checks whether at least one chain can reach states where the quality attributes are at the desired levels. If the model checking fails, it is guaranteed that the adaptation loops will never reach the desired quality levels from the current situation.

In our approach, we model specification of the adaptation loops and description of their effects on the managed system as state machines whose interactions represent the adaptation chain. To improve scalability of the model checking, we abstract away the details of the managed system, unlike Luckey's approach of modeling the structure of the managed system. In-

stead, we construct the state machines from the view point of the qualities.

The contributions of this paper are twofold. First, we propose a procedure for constructing these models. Our procedure guides developers in making the two kinds of state machines by using *manipulation variables*, which represent configuration options of the managed system, and *controlled variables*, which represent the quality attributes of it. Second, we offer a tool that supports the transformation of the model into a formal model. We use NuSMV [3], which is a model checking tool for computation tree logic (CTL), for the qualitative analysis. The tool automates transformation of the state machines into a NuSMV model, to be used by NuSMV model checker that reports the results of its analysis. It allows developers to perform the qualitative analysis without having to know the details of the model checking.

The rest of the paper is organized as follows. Section 2 explains a motivating example, and Section 3 explains our procedure and tool. Section 4 presents the case studies and experimental results obtained from them, and Section 5 discusses the applicability and limitations of our analysis. Section 6 describes related work. Finally, Section 7 concludes the paper and outlines future work.

2 Motivating Example

The motivating example is a web application providing a news service. The example is based on *Znn.com* [2], which is a well-known exemplar in research on self-adaptive software [16, 23]. The Znn.com imitates a news Web service, like *cnn.com*, consisting of a set of client processes that make stateless content requests to one of the servers, a server pool that manages a set of replicated application servers, a load balancer that balances requests across the server pool, a database server, and a server monitor that periodically checks the current status of the application servers (Figure 2). The important quality attributes of the system are the *response time*, *server load*, and *quality of content* (QoC). The web application may face unexpected performance degradations caused by hardware failures or sudden increases in requests from users, but it should maintain these attributes at the desired levels.

We extended the Znn.com so that it would support multiple adaptation loops. We added four variation points and their adaptation interfaces (white rectangles in Figure 2) to control the quality attributes: a server pool management point to add or remove application servers in the pool, a compression rate management point to change the compression ratio of content to be delivered, a transfer speed management point to adjust the transfer speed of the content to the clients, and a content provisioning management point to switch the type of content (multimedia or text). We also added

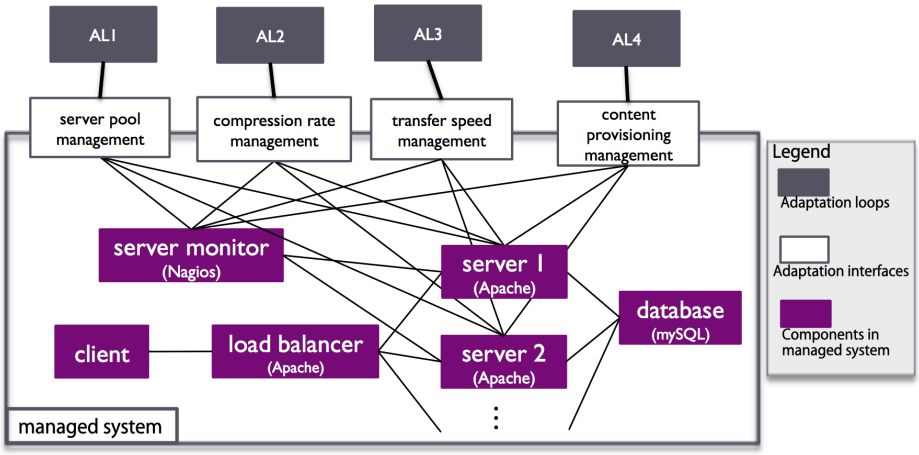


Figure 2: System architecture of a web application with multiple adaptation loops

adaptation loops for some of the adaptation interfaces (gray rectangles in Figure 2) to maintain the quality attributes at the desired levels.

The adaptation loops may cause adaptation chains. For example, the adaptation loop for compression rate management will switch the rate from low to high when the response time gets worse. The change aims to control the response time, but it requires additional computations for the compression and consequently, it has a negative effect on the server load. Therefore, the change may trigger the adaptation loop for server pool management, which maintains the server load, and a change by the loop for server load management, in turn, may trigger the other loops.

3 Qualitative analysis for multiple adaptation loops

Generally speaking, software specifications S should satisfy requirements R within the context of the given environment E[25].

$$S, E \models R$$

Here we think about qualitative analysis of multiple adaptation loops. S corresponds to the specifications of the adaptation loops, E corresponds to the qualities of the managed system, and R corresponds to the reachability property about states of the managed system. We provide a procedure for constructing models of S and E as state machines and a model transformation tool to generate a formal model for model checking using the state machines.

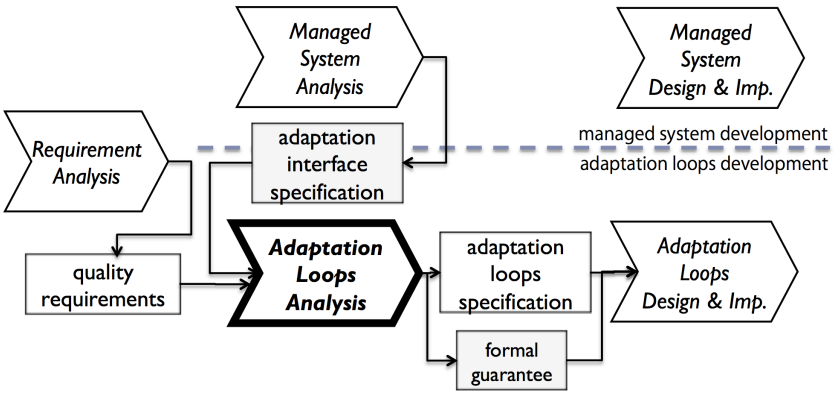


Figure 3: Scope of our work

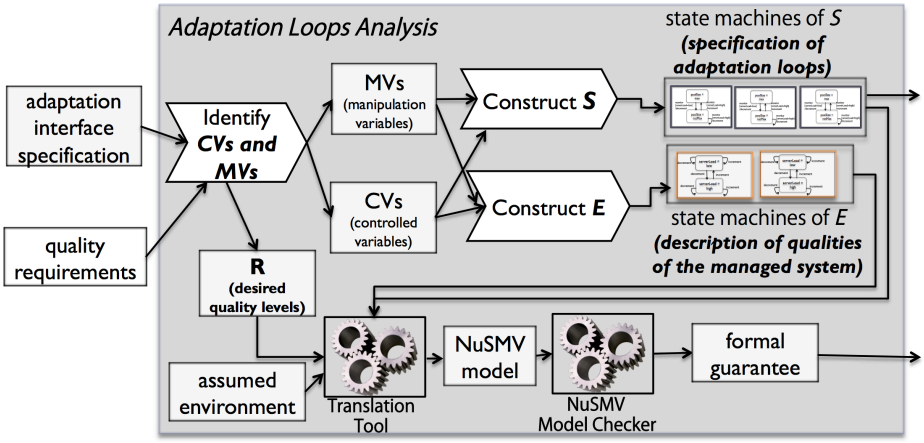


Figure 4: Process of adaptation loop analysis

3.1 Overview

Figure 3 outlines the scope of our work. Our focus is *adaptation loops analysis* phase to create specifications of adaptation loops based on qualitative analysis. We assume that the quality requirements to be achieved by self-adaptation has been determined in the requirement analysis phase. Besides, we also assume that available and valid configuration options of the managed system have been determined and specifications of adaptation interfaces have been constructed in the managed system analysis phase. Our process outputs the specifications of adaptation loops together with formal guarantees.

Figure 4 shows the adaptation loop analysis. First, developers construct

S and E as state machines by following the procedure described in Section 3.2. Then, they use the transformation tool described in Section 3.3 to generate a NuSMV model from the state machines and desired quality levels. Finally, they use the NuSMV model checker to perform qualitative analysis to obtain formal guarantee, as described in Section 3.4. Note that they do not need to know the details of the NuSMV model. Instead, they only need to know whether the model checking fails or not for each possible current state of the managed system.

3.2 Constructing S and E

Let us show how developers can construct S and E as state machines whose interactions represent adaptation chains. We use *controlled variables* and *manipulation variables* [20] to model S and E.

3.2.1 Identifying controlled variables and manipulation variables

The process starts by identifying the controlled variables and manipulation variables by analyzing the specifications of the adaptation interfaces. The controlled variables represent values that the adaptation loops are intended to control. These correspond to quality attributes of the managed system and can be identified from monitoring operations in the adaptation interfaces. In addition, the controlled variables are used for specifying the desired levels that satisfy the quality requirements.

The manipulation variables represent values that can be changed by the adaptation loops. These correspond to configuration options of the managed system, which can be parameter-based configurations or architectural configurations. Developers model configuration options of the managed system, which have been determined in the managed system analysis phase, as manipulation variables. They also identify *actions* to change the manipulation variables by analyzing the effector operations.

We define the controlled variables and the manipulation variables as discrete variables. When a quality attribute or a parameter for a configuration takes real values, we discretize them by setting a threshold determined by analyzing the desired level of the quality attribute or range of the parameter. By using these discrete variables, we model the system and adaptation loops as state machines.

Example There are four adaptation interfaces in the news service illustrated in Figure 2. Here, three controlled variables ($serverLoad$, $responseTime$, QoC) were identified as shown in Table 1 Then, the desired quality levels are specified as ($serverLoad = low \wedge responseTime = fast \wedge QoC = high$) by using them. In addition, four manipulation variables, and eight actions were

Table 1: Controlled variables (CVs), manipulation variables (MVs) and actions in the news service

CVs	$serverLoad$	$\{low, notLow\}$
	$responseTime$	$\{fast, notFast\}$
	QoC	$\{high, notHigh\}$
MVs	$poolSize$	$\{max, notMax\}$
	$cRate$	$\{fast, slow\}$
	$tSpeed$	$\{high, low\}$
	$contentType$	$\{MM, text\}$
actions	for $poolSize$	$\{increment, decrement\}$
	for $cRate$	$\{switchToHigh, switchToLow\}$
	for $tSpeed$	$\{switchToFast, switchToSlow\}$
	for $contentType$	$\{switchToMM, switchToText\}$

identified. Note that $poolSize$ and $transferSpeed$ represent parameter-based configurations that can be mapped to parameters in the managed system, and $cRate$ and $contentType$ do that of architectural configuration that can be mapped to predefined configurations of components and connectors in the managed system.

3.2.2 Constructing E

The state machines for qualities (E) represent how the quality attributes are affected by the configuration changes.

Model of E E is modeled as a set of state machines referred as to SM_{qa}. An SM_{qa} is constructed for each controlled variable. Specifically, SM_{qa} is a tuple of $< S_{cv}, T_{qa} >$, where S_{cv} is the set of states of controlled variables and T_{qa} represents the set of state transitions representing effects on the controlled variables. T_{ai} is a tuple of $< s_s, s_t, e >$. $s_s \in S_{cv}$ and $s_t \in S_{cv}$ respectively represent the source and target states of the transitions. $e \in Action_{ai}$ represents an event triggering a transition, where $Action_{ai}$ is a set of actions available at adaptation interfaces which might affect the controlled variable.

Steps for constructing E E is constructed by repeating the following steps (Figure 5) for each controlled variable.

Step 1: Describe state space of the quality attribute
Developers describe the state space of the quality attribute as a set of states (S_{mv}) of SM_{qa}. It is defined with a relevant controlled variable defined in Section 3.2.1.

Step 2: Describe effects of configuration changes

Developers describe all possible effects of the configuration changes on the quality attributes. The effects are defined as state transitions of the controlled variable, which are caused by actions that adjust the manipulation variables. A quality attribute might be affected by actions of different adaptation interfaces. Developers select actions relevant to the controlled variable from all actions and define $Action_{ai}$.

Developers also analyze the possible effects for each action in $Action_{ai}$ and describe transitions as T_{qa}. Note that the effects are often uncertain at development time. An action may or may not trigger the state transitions of a quality attribute. In this case, developers model the both possibilities. The uncertain effects are modeled as nondeterministic transitions. Developers should model all possible effects of actions.

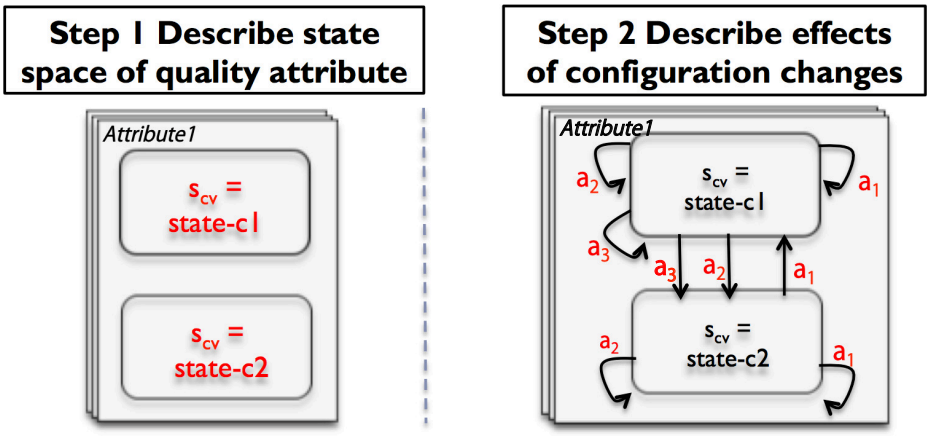


Figure 5: Steps in constructing SM_{qa}

Example In the case of the news service, three SM_{qa}s are constructed, which correspond to $serverLoad$, $responseTime$, and QoC. The left part of Figure 6 outlines SM_{qa} for $serverLoad$. It has two states ($serverLoad = low$ and $serverLoad = notLow$). $serverLoad$ is relevant to the actions configuring the server pool ($increment$ and $decrement$) and the actions configuring the compression rate ($switchToHigh$ and $switchToLow$). Therefore, the possible effects of these actions on $serverLoad$ are specified as state transitions. For example, the state cannot be changed by $increment$ when $serverLoad$ is low, because adding servers to the pool will not increase the server load. However, the state of the server load may or may not be

changed by *increment* when *serverLoad* is *notLow*. It is uncertain at development time. Therefore, the both are specified as non-deterministic state transitions.

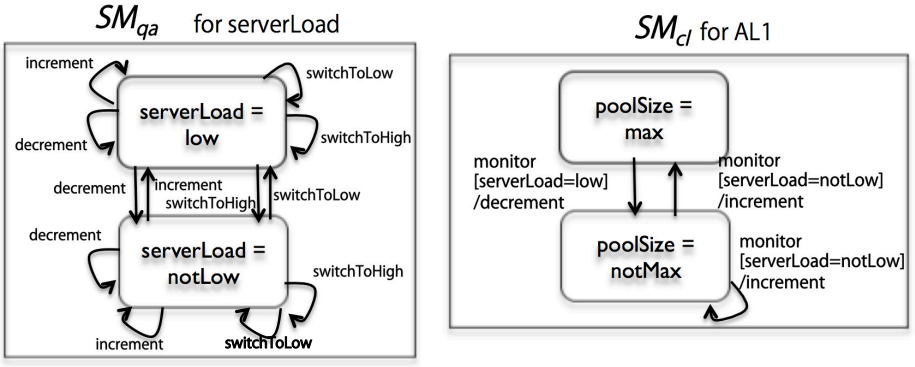


Figure 6: Part of SM_{al} and SM_{qa} for the news service

3.2.3 Constructing S

The state machines represent how the adaptation loops (S) change the configurations of the managed system.

Model of S S is the set of state machines SM_{al}. An SM_{al} is constructed for each adaptation loop. Specifically, SM_{al} is a finite state modeled as a tuple of $< S_{mv}, T_{al} >$, where S_{mv} is the set of states of manipulation variables and T_{al} is the set of state transitions. T_{al} is a tuple of $< s_s, s_t, e, g, a >$. $s_s \in S_{mv}$ and $s_t \in S_{mv}$ correspond to the source and target states of the transition. $e \in \{monitoring\}$ represents an event triggering the transitions. An adaptation loop is only triggered by periodically monitoring the managed system, and is represented as *monitoring*. $g \in Guards_{al}$ and $a \in Actions_{ai}$ represent a guard and an action of the transition. $Guards_{al}$ is the set of guards of the transitions, which are specified with controlled variables.

Steps for constructing S S is constructed by repeating the following steps (Figure 7) for each adaptation loops.

Step 1: Describe state space of the configuration
Developers describe the state space of the configuration that adaptation loop changes. The set of states (S_{mv}) of SM_{al} is defined with a relevant manipulation variable described in Section 3.2.1.

Step 2: Specify behavior of the adaptation loop

Developers specify when and how the adaptation loop should change the configuration, as T_{al}. They specify triggering conditions of the adaptation loop with the controlled variables, and actions to be performed by it to change configurations. State transitions in SM_{al} occur as a result of the actions. They also specify possible changes of manipulation variables by the actions, as T_{al}. Note that the triggering conditions are key to model adaptation chains. An SM_{al} can affect SM_{qa}s by taking an action, and the changes in the SM_{qa}s in turn may be matched to triggering conditions of other SM_{al}s.

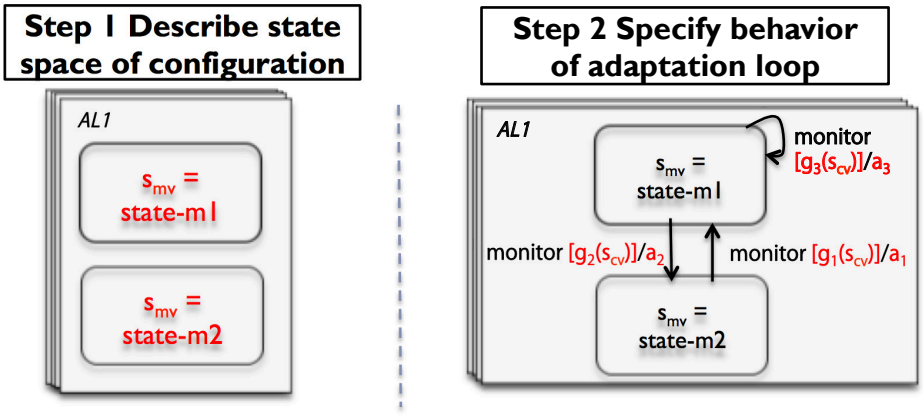


Figure 7: Steps in constructing SM_{al}

Example In the case of the news service, five SM_{al}s are constructed, and they correspond to adaptation loops (AL_1, AL_2, AL_3, and AL_4). The right part of Figure 6 outlines SM_{al} representing AL_1. A manipulation variable relevant to AL_1 is *poolSize*, whose states can be *max* or *notMax*. In response to the monitoring event, AL_1 takes the *decrement* action when the *poolSize* is *max* and *serverLoad* is *low*. Moreover, AL_1 takes the *increment* action when *poolSize* is *notMax* and *serverLoad* is *notLow*. The *decrement* action occasionally causes a state transition from *notMax* to *max*. It depends on the current value of the manipulation variable, but it has been abstracted away at development time.

3.3 Transforming S and E into a NuSMV model

Let us show how NuSMV models are generated from S and E. The NuSMV model consists of state variable definitions, state machine definitions, and

CTL specifications. The transformation tool generates a NuSMV model from the set of the SM_{qa}s and the set of the SM_{al}s. The transformation is performed as follows.

Step 1: State variable definitions
State variables in the NuSMV model, which are used to specify properties, correspond to controlled variables. The state definitions of the controlled variables are mapped to the state variable definitions. Note that the initial states of the controlled variables will be substituted later.

Step 2: CTL specification
Here, the EF property is used to check reachability, which is written as $EF(< desiredqualitylevels >)$ in the CTL specification. The desired quality level specified with the controlled variables is mapped to this. If there exists a path beginning at the initial state such that $< desiredqualitylevels >$ holds at the current state or at some future state, it becomes true.

Step 3: State machine definitions
SM_{al}s in S and SM_{qa}s in E are mapped to state machines definitions in the NuSMV model. Figure 8 shows the details of the state machine definitions in the template. SM_{qa} is transformed into a state machine in the NuSMV model, whereas the limitations of the NuSMV language result in one SM_{al} being transformed into two state machines: one for the state transitions of the manipulation variables, the other for actions corresponding to the transitions. Note that the initial states of the manipulation variables will be substituted later.

We implemented a prototype of the transformation tool in Eclipse [1]. Figure 9 shows a screenshot of the tool. We used the Papyrus[2] UML editor to construct the state machines. We implemented the model-to-model transformation rules in the ATL[3]. The ATL engine outputs an XMI-based NuSMV model by following the rules. We also implemented the model-to-text transformation rules in the Acceleo language[4]. The Acceleo engine outputs a text-based NuSMV model.

Example We constructed a NuSMV model for the news service of which AL_1, AL_3, and AL_4 are shown in Figure 2. AL_1 tries to control $serverLoad$ by adjusting $poolSize$, AL_3 tries to control $responseTime$ by adjusting

[1]Source code of the prototype is available at `https://github.com/kiyo07/UMLStateMachine2NuSMV`

[2]Papyrus : `http://www.eclipse.org/papyrus/`

[3]ATL : `https://eclipse.org/atl/`

[4]Acceleo : `https://eclipse.org/acceleo/`

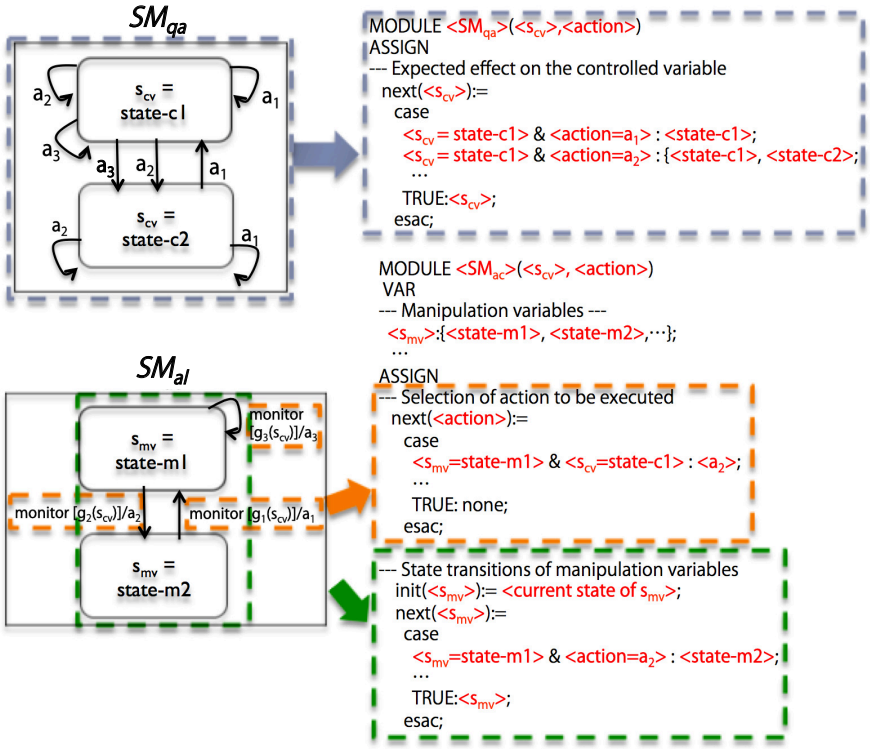


Figure 8: Details of the state machine definition part in the NuSMV template

tSpeed, and AL_4 tries to control *QoC* by adjusting *contentType*. We used the transformation engine to generate a NuSMV model from the models. Figure 10 overviews the NuSMV model highlighting parts corresponding to the state machines in Figure 6[5].

3.4 Performing qualitative analysis

Let us show how developer performs the qualitative analysis from the generated NuSMV model. The generated NuSMV model is used for model checking. The model checker explores the adaptation chains caused by the adaptation loops from the initial states, and checks the reachability property specified in CTL.

Note that results of the qualitative analysis depends on the initial states of E and S. The initial states of E correspond to current states of the

[5]Complete NuSMV models are available at `http://research.nii.ac.jp/~tei/cl-abstraction/`

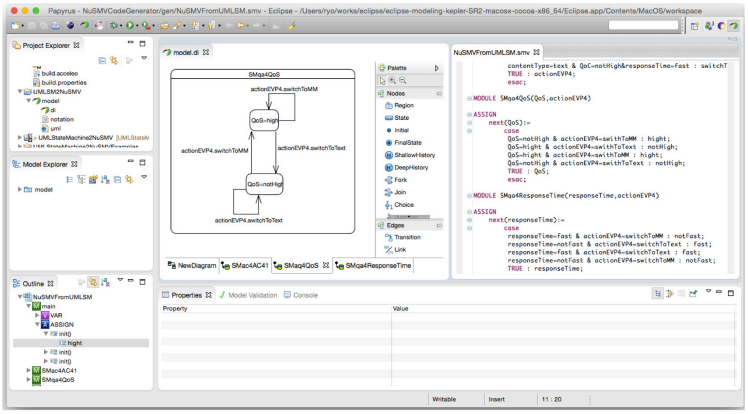


Figure 9: Prototype of the transformation tool in Eclipse

controlled variables, and those of S correspond to current states of the manipulation variables. Developers assume a situation and input states of controlled variables and manipulation variables for the situation in the NuSMV models and perform model checking.

If the model checking fails, it is guaranteed that the adaptation loops will never satisfy quality requirements from the assumed situation. They repeat the model checking for all possible situations. In so doing, they identify sets of states for which the model checking fails. These sets represent situations where the given adaptation loops never work.

Scope of this paper is qualitative analysis at development time, but developers have several strategies of how to cope with the situations. One of the strategies might be to prepare different sets of adaptation loops at development time, and to switch adaptation loops to another set where qualitative analysis succeeds, at runtime. The situations identified by qualitative analysis at development time can be used as triggering conditions to adapt adaptation loops at runtime.

Example Here, we think qualitative analysis of the adaptation loops (AL_1, AL_3, AL_4). We assume a situation where the *serverLoad*, the *responseTime*, and the QoC are set to fast, notFast, and high respectively, and *poolSize*, *tSpeed* and *contentType* are respectively $notMax$, $high$ and MM. In this situation, the property, $serverLoad = low \wedge responseTime = fast \wedge QoC = high$, does not hold at the time, but the model checking successes. This means that the given adaptation loops are possible to lead these qualities to the desired levels in a future by performing an adaptation or an adaptation chain. Actually, there is still a possibility that AL_3 may improve *responseTime* by adjusting the transfer speed.

Let us think of a situation where *tSpeed*, which is the manipulation variable of AL_3, has been changed to *fast*, but the *responseTime* is still notFast. The model checking for the situation fails. This is because AL_3 can no longer improve *responseTime*. Developers can thus know that the adaptation loops are not appropriate for the situation.

In the situation, developers can prepare different adaptation loops such as (AL_1, AL_2, AL_4). Here, model checking of (AL_1, AL_2, AL_4) successes for the situation. The loops can be used instead of (AL_1, AL_3, AL_4) for the situation.

Note that all model checking that we performed here were completed within 20 ms.

4 Evaluation

This section presents evaluations of the qualitative analysis for multiple adaptation loops. The evaluations answered the following research questions:

RQ1 Are the formal guarantees also valid at the implementation of the self-adaptive systems?

RQ2 How scalable is the model checking for complicated self-adaptive systems?

4.1 Validation

RQ1 is about a gap between specification and implementation. Our qualitative analysis provides formal guarantees at the specification level, but it is not clear how valid it is at the implementation level. To answer RQ1, we compared the results of the analysis with data obtained from experiments.

4.1.1 Experimental setup : news service

We used the news service scenario for the case studies. The managed system of the news service was developed with the Znn.com package that includes Apache httpd-2.4.2 as the HTTP server and a load balancer, MySQL 5.5.25 as the database server, and Nagios 3.4.1 as the server monitoring tool, as illustrated in Figure 2. The managed system has four adaptation interfaces to get current values of the controlled variables (*responseTime*, *serverLoad* and *QoC*)[6] and and to execute actions on the managed variables (*poolSize*,

[6]State spaces of the controlled variables were modeled as follows. The *responseTime* was recognized as *fast* when the average response time of the most recent 30 requests was shorter than 3 s, and it was otherwise *notFast*. The *serverLoad* was recognized as *low* when the load average monitored was less than 1, and it was otherwise *notLow*. The *QoC* was recognized as *high* when the multimedia news was provided, and it was recognized as *notHigh* when text news was provided.

Table 2: Results of model checking (development-time results) corresponding to the situations in the experiments. (S:Success, F:Fail)

	at 60 s	120 s	180 s	240 s	300 s	360 s
multiple	S	S	F	F	S	S
single($AL1$)	F	F	F	F	F	F
single($AL2$)	S	S	F	F	S	F
single($AL4$)	S	S	F	F	F	F

cRate, *tSpeed* and *contentType*). Developer can construct all or some of adaptation loops, (AL_1, AL_2, AL_3 and AL_4), which work on the interfaces.

At development time, we constructed specifications of the adaptation loops by following the process shown in Section 3.2. Then we performed qualitative analysis with the specifications for all possible initial states of manipulation variables and controlled variables, and identified a set of situations where qualitative analysis fails.

We also implemented adaptation loops on the top of the Znn.com package, in accordance with the specification. For example, Figure 11 overviews the implementation of AL_1 for the server pool management point, which is written in C#. It has if-then rules to select an action (incrementing or decrementing the number of servers in the pool) to control the server load, in accordance with the specification shown in Figure 6. All adaptation loops worked every 30 seconds.

We executed the implemented news service system and measured the server load, the response time, and the QoC. Then, we compared the experimental data with the results of qualitative analysis obtained at development time to validate the qualitative analysis.

4.1.2 Results : single and multiple adaptation loops

Here, we prepared four self-adaptive systems. Three of them are self-adaptive systems with a single adaptation loop (AL_1 managing pool size, AL_2 managing compression rate, or AL_4 managing type of content), and the other one is a self-adaptive system with multiple adaptation loops (AL_1, AL_2, and AL_4). We executed them in an environment where user traffic changed at runtime.

The (1-a), (1-b) and (1-c) in Figure 12 show the experimental results of the server load, the response time, and the QoC[7], respectively. We also measured states of the controlled variables and manipulation variables at 60, 120, 180, 240, and 300 s. In order to compare the experimental results with results of the qualitative analysis, we extracted results of model checking

[7]QoC for AL_1 and AL_2 were always *high*. Therefore we omitted them from the graphs.

corresponding to the situations from the results we have achieved at development time. Table 2 shows the corresponding results of model checking.

For the first 120 s, the user traffic was 5 requests per second. Table 2 shows that model checking of AL_1 failed. Experimental results also show that AL_1 (the blue lines) could not maintain the response time and the server load at the desired levels. Table 2 also shows that model checking of the others succeeded, and the experimental results show that they could actually lead the qualities at the desired level.

From 120 s to 240 s, the user traffic increased to 7. Then results of model checking changed due to the changes in the environment. All model checking failed. The experimental results show that the all adaptation loops actually never lead the qualities at the desired levels until 240 s.

After 240 s, the user traffic was back to 5. At 300 s and 360 s, the model checking of the multiple loops succeeded again. Experimental results of the multiple loops (the red lines) actually could maintain all the qualities. It might be curious that the model checking succeeded again although it had failed at 120 s. This change was caused not by adaptation loops but by changes in the environment (by reducing the user traffic). At 120 s, model checking could not consider the change of the user traffic, because we do not model changes caused by factors except for adaptation loops. The model checking of AL_2 also succeeded at 300 s. The server load of AL_2 (the green lines) exceeded the threshold at the 300 s, but it had possibilities to lead the server load to the desired level by changing the compression rate. It actually tried this by performing *switchToLow* action, but it could not speed up the response time enough. Then model checking for the situations after 300 s failed, and the experimental results also show that AL_2 actually could not do that.

In summary, the formal guarantees were actually valid in the experiments. If the system is in a situation where model checking fails, the adaptation loops could not maintain all the quality attributes at the desired levels after that unless the environment changes due to factors except for the adaptation loops.

4.1.3 Results : static and adaptive multiple loops

As we discussed in Section 3.4, if the model checking fails, developers can prepare other sets of control loops for the situations. In the experiments, we prepared two kinds of self-adaptive systems, referred as to *static adaptive* that uses a set of adaptation loops and *dynamic adaptive* that switches a set of adaptation loops at runtime in accordance with results of the qualitative analysis. We compared their experimental results with the results of the qualitative analysis with them.

Experimental setup was almost same with the previous one, but the number of requests per second changes at runtime, which was initially 2 and

Table 3: Results of model checking (development-time results) corresponding to the situations in the experiments. (S:Success, F:Fail)

situation	60 s	120 s	180 s	240 s	300 s	360 s
static adaptive	S	S	S	S	F	F
dynamic adaptive	S	S	S	S	S	S

gradually increased up to 5. In addition, threshold of the response time was 1 s. (2-a) and (2-b) in Figure 13 plot the response times and server loads for this scenario, respectively [8]. Table 3 shows results of model checking corresponding to situations at 60, 120, 180, 240, 300, and 360 s.

Model checking for the static adaptive system with (AL_1, AL_3, AL_4) succeeded until 240 s, The red lines show that it actually could maintain the qualities until 240 s. At 240 s, *loadAverage* had changed to *notLow*, but the model checking succeeded. The experimental results of *loadAverage* had actually changed to low after that as a result of an adaptation chain performed by the loops. At 300 s, *responseTime* had changed to *notFast*. The model checking failed this time. and the load average actually increased sharply. Until 360 s, the results of model checking were valid in the experimental results. It might be curious that the load average sharply dropped after 360 s. This was caused not by the adaptation loops, but by other factors in the environment. At the time, success rate of user requests dropped from 100 % to 50 % due to timeout, and this reduced the load average and the response time.

Model checking for the dynamic adaptive system, which initially used (AL_1, AL_3, AL_4) until 300 s and used (AL_1, AL_2, AL_4) after that, succeeded all the time by switching adaptation loops by following the results of the qualitative analysis. The blue lines show that experimental results were almost same with the static one until 300 s, and it could maintain all the qualities after that by switching adaptation loops. Note that the success rate was always 100 % in the dynamic adaptive system.

4.2 Scalability

RQ2 is about scalability. Multiple adaptation loops enlarge adaptation space, but will require more time for the qualitative analysis. Although our approach enables state space of each state machine to be small by abstraction based on the controlled variables and manipulation variables, the qualitative analysis should analyze parallel composition of all state machines. We evaluated scalability by measuring execution time for the qualitative

[8] Horizontal green solid lines in these figures indicate thresholds of the response times and server loads, respectively

analysis.

First we evaluated scalability for the number of controlled variables. The left graph in Figure 14 shows execution times of three cases (10, 20, and 30 adaptation loops where each loop affects one controlled variable) by changing the number of controlled variables from 5 to 30. Model checking for the 10 and 20 loops were completed within 0.2 and 150 seconds, respectively. These are reasonable time for qualitative analysis at development time. Model checking for the 20 loops exponentially increased up to 20 controlled variables, but after that it slowly increased. This is because the 20 adaptation loops can control with up to 20 controlled variables. As same with the 20 loops, the results for the 10 loops slowly increased after 10 controlled variables. Model checking for the 30 loops also increased exponentially, and was completed in reasonable time up to 22 controlled variables (457 seconds). However, it took 3 hours for 23 controlled variables and more than 24 hours for 24.

Next we evaluated scalability for the number of adaptation loops. The right graph in Figure 14 shows execution times of three cases (10, 20, and 30 controlled variables) by changing the number of adaptation loops from 5 to 30. The results were almost same with the previous one. Model checking for the 10 and 20 controlled variables can be completed within reasonable time. That for the 30 controlled variables are reasonable up to 22 loops.

5 Discussion

Validation of qualitative analysis Section 4.1 showed that our qualitative analysis was valid in the situations in the experiments until changes caused by other factors occur, such as sudden increases in requests from users. This is because that we model all possible effects of configuration changes on the qualities as non-deterministic transitions, but do not model the changes that spontaneously occurred, which are usually unknown at the analysis phase. Developers can identify situations where given adaptation loops will not satisfy quality requirements by the qualitative analysis, and can prepare other sets of loops that are valid in the situations to keep maintaining quality attributes, as shown in Section 4.1.3.

However, developers should carefully model all possible changes in E as non-deterministic transitions, but they might miss to model some necessary changes or include unnecessary changes that are not occurred at runtime. The inconsistency will result in incorrect results of qualitative analysis. Sykes et al. [21] proposed a learning technique to revise E to keep the consistency by using feedback from the running system. Such a technique can be used to handle the issue.

Scalability Scalability tests in Section 4.2 showed our qualitative analysis can be completed in reasonable time up to totally 50 state machines consisting of adaptation loops and controlled variables. Although real complex systems have several hundreds configuration options [4] internally, only a small part of the configuration options is controlled by adaptation loops in actual. Similarly, although many quality attributes exist in complicated system, only a small part of them are relevant to quality requirements of the system and are used as controlled variables. Actually, self-adaptive systems used in existing studies [11, 5, 16, 24, 1] equips just less than 10 adaptation loops, and controls less than 10 quality attributes. Our qualitative analysis can be practically used for these applications.

Applicability Our qualitative analysis does not depend on the specific architecture of the managed systems as long as they provide adaptation interfaces. Therefore, our analysis can be applied to managed systems which provide adaptation interfaces on different layers, such as cloud systems that provide adaptation interfaces for the IaaS layer and SaaS layer, and mobile systems that provide interfaces for hardware layer and application layer.

Our analysis models the adaptation loops by using manipulation variables that can represent parameterized configurations (e.g. the number of pools) and architectural configurations (e.g. type of content). Specifications of the adaptation loops can also be naturally implemented as a program consisting of a set of if-then rules by using thresholds as shown in Figure 11. The way of implementation based on if-then rules is actually simpler than other ways such as machine learning-based implementation, but it is popularly supported by practical adaptable systems. For example, *autoscaling* used in Amazon EC2 provides the way of implementation with the if-then rules to configure the number of virtual machines.

6 Related work

There have been many existing studies on making formal verifications on managed systems. Zhang et al. [26, 27] used Petri nets to represent the behavior of a managed system, and used model checking to analyze them for their safety and liveness properties. Sykes et al. used model checking with a reachability property to find plans to change architectural configuration of the managed system [22]. Filieri et al. [8] devised a runtime-efficient probabilistic model checking by dividing the model checking in two steps, one performed at development time, the other at runtime, to improve the runtime performance of the analysis. However, they did not intend to check correctness of adaptation loops.

A few recent studies have applied model checking to adaptation loops. Iftikhar et al. proposed ActivFORMS [10] that specifies a managed sys-

tem and an adaptation loop by using timed automata. ActiveFORMS used Uppaal model checker to verify reachability or liveness properties specified in timed computational tree logic (TCTL). It provides a virtual machines of adaptation loops that assure that the properties verified at development time are guaranteed at runtime. However, ActiveFORMS does not treat multiple adaptation loops causing the adaptation chains. In addition, it has not been tested on medium or large systems, and it might not scale well for complicated systems exposing many adaptation interfaces because it models the detailed behaviors of the managed system. Luckey et al. [15] targeted multiple adaptation loops and proposed a UML-based language called ACML for modeling adaptation loops. Models specified by ACML are transformed into labelled transition systems that are used to check stability of the adaptation loops and if they have any dead locks. However, their study focused on termination and deadlock, not on reachability. In this paper, we focused on qualitative analysis of multiple adaptation loops with a reachability property.

As another approach to assuring the qualities of the system is to use quantitative analysis with techniques used in control science [19, 13, 6, 7, 14]. This approach uses closed formulas to model the managed system and adaptation loops and control theory to guarantee the system and adaptation loops mathematically. However, it is not easy to model software systems, which are usually non-linear, in terms of closed formulas. There are some studies on mitigating the difficulties in modeling a system as closed formula. Filieri et al. [9] proposed a generalized method that can automatically construct an approximate model of a system and synthesize suitable adaptation loops for managing its non-functional requirements. They modeled a software system as a linear model whose coefficients can be automatically determined by ARPE [17], which is based on linear regression, with a dataset obtained by testing a set of sampled values of a manipulation variable and measuring the effect on the quality attributes of the system. An appropriate adaptation loop for the system can be synthesized from the model by using the Z-transformation. Klein et al. [14] proposed a similar methodology. They modeled a cloud application as a linear model with disturbances. However, these methodologies can only handle one manipulation variable and one controlled variable whereas our method can handle multiple manipulation variables and controlled variables.

Finding proper configurations is an important issue for modern adaptable software systems, and there have been many studies on it. Zhang and Ernst [28, 29] proposed techniques to diagnose configuration errors caused by invalid combinations of configurations. They used dynamic profiling, execution trace comparison, and static analysis to check the validity of a given configuration for the variation points. Nadi et al. [18] proposed techniques to automatically extract configuration constraints from the source code of the system by using variability model analysis and code analysis. These

studies dealt with changes that occur offline, such as evolution of the managed systems. In addition, whereas they dealt with functional errors, we deal with deterioration in the qualities of the system.

7 Conclusion

We described qualitative analysis for multiple adaptation loops. We model the managed system and adaptation loops as state machines interacting each others, by using the controlled variables and the manipulation variables. The case study showed that the results of our analysis are valid at the implementation unless spontaneous changes in the environment occur and that the analysis can complete within reasonable time up to 50 state machines consisting of adaptation loops and controlled variables. We also developed a process and tools to help developers to construct models.

We plan to automate the adaptation of adaptation loops in the future. Currently, we rely on developers to adapt the loops in accordance with the results of the qualitative analysis. In the future, we will study runtime adaptation of adaptation loops with the model checker.

References

[1] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos. Dealing with multiple failures in zanshin: a control-theoretic approach. In *SEAMS 2014: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM Request Permissions, June 2014.

[2] S.-W. Cheng, D. Garlan, and B. Schmerl. Evaluating the effectiveness of the Rainbow self-adaptive system. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 132–141, 2009.

[3] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[4] M. B. Cohen, M. B. Dwyer, and J. Shi. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. *IEEE Transactions on Software Engineering,*, 34(5):633–650, 2008.

[5] F. A. de Oliveira, T. Ledoux, and R. Sharrock. A Framework for the Coordination of Multiple Autonomic Managers in Cloud Environments. In *IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO2013)*, pages 179–188. IEEE, 2013.

[6] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck. From data center resource allocation to control theory and back. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 410–417, July 2010.

[7] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 283–292. IEEE, 2011.

[8] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *ICSE '11: Proceeding of the 33rd International Conference on Software Engineering*. ACM Request Permissions, May 2011.

[9] A. Filieri, H. Hoffmann, and M. Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, pages 299–310. ACM Request Permissions, May 2014.

[10] 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.

[11] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 24–24. IEEE, 2007.

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

[13] M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark. Control-theoretic analysis of admission control mechanisms for web server systems. *World Wide Web*, 11(1):93–116, 2008.

[14] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez. Brownout: building more robust cloud applications. In *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, pages 700–711. ACM Request Permissions, May 2014.

[15] M. Luckey and G. Engels. High-quality specification of self-adaptive software systems. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 143–152, 2013.

[16] M. Luckey, B. Nagel, C. Gerth, and G. Engels. Adapt cases: extending use cases for adaptive systems. In *SEAMS '11: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM Request Permissions, May 2011.

[17] M. Maggio and H. Hoffmann. Arpe: A tool to build equation models of computing systems. In *Presented as part of the 8th International Workshop on Feedback Computing*, Berkeley, CA, 2013. USENIX.

[18] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki. Mining configuration constraints: static analyses and empirical results. In *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, pages 140–151. ACM Request Permissions, May 2014.

[19] T. Patikirikorala, A. Colman, J. Han, and L. Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '12, pages 33–42, 2012.

[20] M. Shaw. Beyond objects: A software design paradigm based on process control. *SIGSOFT Softw. Eng. Notes*, 20(1):27–38, Jan. 1995.

[21] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue. Learning revised models for planning in adaptive systems. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 63–71. IEEE, 2013.

[22] D. Sykes, W. Heaven, J. Magee, and J. Kramer. From goals to components: A combined approach to self-management. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, SEAMS '08, pages 1–8, New York, NY, USA, 2008. ACM.

[23] G. Tamura, N. M. Villegas, H. A. Müller, L. Duchien, and L. Seinturier. Improving context-awareness in self-adaptation using the dynamico reference model. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 153–162. IEEE Press, 2013.

[24] T. Vogel and H. Giese. Model-driven engineering of self-adaptive software with eurema. *ACM Trans. Auton. Adapt. Syst.*, 8(4):18:1–18:33, Jan. 2014.

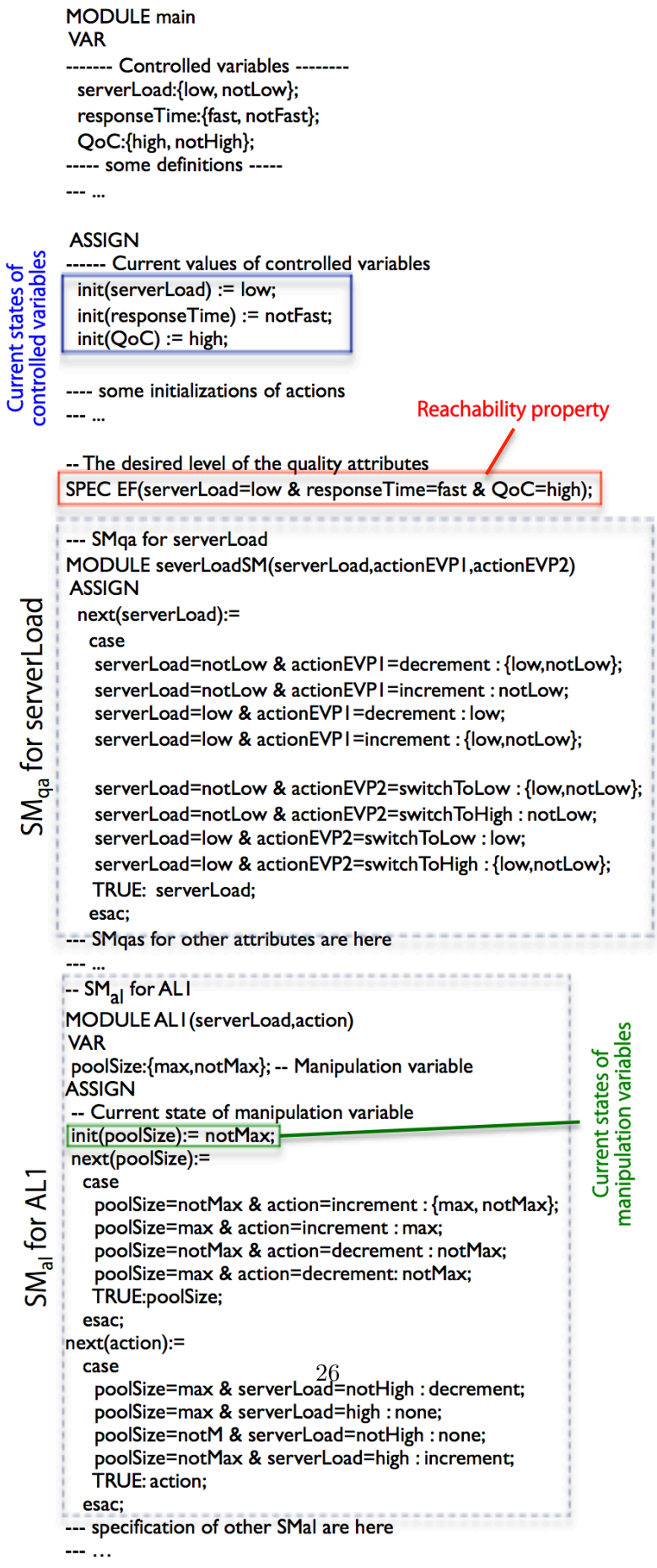
[25] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, Jan. 1997.

[26] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 371–380, New York, New York, USA, May 2006. ACM Request Permissions.

[27] J. Zhang, H. J. Goldsby, and B. H. C. Cheng. Modular verification of dynamically adaptive systems. In *AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 161–172, New York, New York, USA, Mar. 2009. ACM Request Permissions.

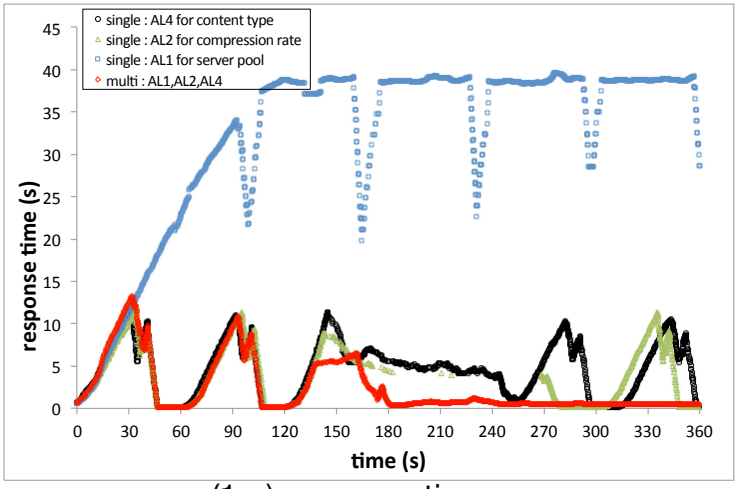
[28] S. Zhang and M. D. Ernst. Automated diagnosis of software configuration errors. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 312–321. IEEE Press, 2013.

[29] S. Zhang and M. D. Ernst. Which configuration option should I change? In *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, pages 152–163. ACM Request Permissions, May 2014.

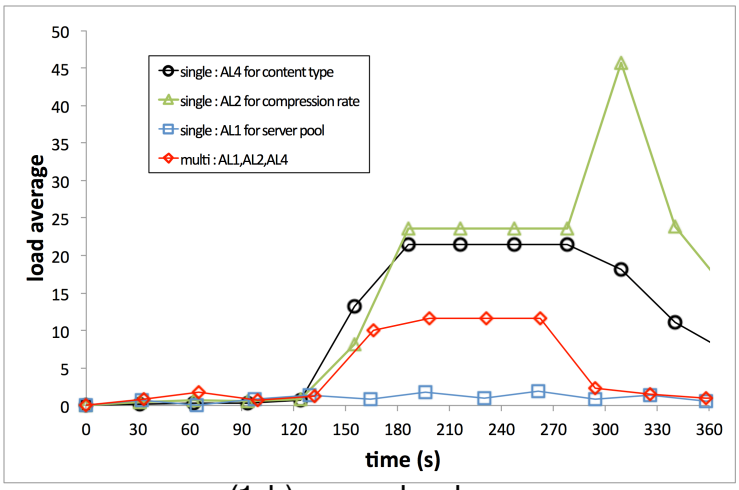

MODULE main
VAR
------- Controlled variables --------
serverLoad:{low, notLow};
responseTime:{fast, notFast};
QoC:{high, notHigh};
----- some definitions -----
--- ...
ASSIGN
------ Current values of controlled variables
init(serverLoad) := low;
init(responseTime) := notFast;
init(QoC) := high;
Current states of controlled variables
---- some initializations of actions
--- ...
Reachability property
-- The desired level of the quality attributes
SPEC EF(serverLoad=low & responseTime=fast & QoC=high);
--- SMqa for serverLoad
MODULE severLoadSM(serverLoad,actionEVP1,actionEVP2)
ASSIGN
next(serverLoad):=
case
serverLoad=notLow & actionEVP1=decrement : {low,notLow};
serverLoad=notLow & actionEVP1=increment : notLow;
serverLoad=low & actionEVP1=decrement : low;
serverLoad=low & actionEVP1=increment : {low,notLow};
serverLoad=notLow & actionEVP2=switchToLow : {low,notLow};
serverLoad=notLow & actionEVP2=switchToHigh : notLow;
serverLoad=low & actionEVP2=switchToLow : low;
serverLoad=low & actionEVP2=switchToHigh : {low,notLow};
TRUE: serverLoad;
esac;
SMqa for serverLoad
--- SMqas for other attributes are here
--- ...
-- SMal for AL1
MODULE AL1(serverLoad,action)
VAR
poolSize:{max,notMax}; -- Manipulation variable
ASSIGN
-- Current state of manipulation variable
init(poolSize):= notMax;
next(poolSize):=
case
poolSize=notMax & action=increment : {max, notMax};
poolSize=max & action=increment : max;
poolSize=notMax & action=decrement : notMax;
poolSize=max & action=decrement: notMax;
TRUE:poolSize;
esac;
next(action):=
case
poolSize=max & serverLoad=notHigh : decrement;
poolSize=max & serverLoad=high : none;
poolSize=notM & serverLoad=notHigh : none;
poolSize=notMax & serverLoad=high : increment;
TRUE: action;
esac;
SMal for AL1
Current states of manipulation variables
--- specification of other SMal are here
--- …

```
Namespace  NewsService {
  public class ServerPoolManager {
    // some declarations here
    // ···
    private static int maxPool = 10;
    private static double threshold = 1.0;

    // some methods here
    // ···
    private void selectAction() {
       // get average of CPU loads
       double load = getAverageOfCPULoads();

       // rules for this adaptation component
       if(load > threshold){
          if(getServers().Length < maxPool)
            incrementServer();
       } else {
          if(getServers().Length > 0)
            decrementServer();
       } } } }
```

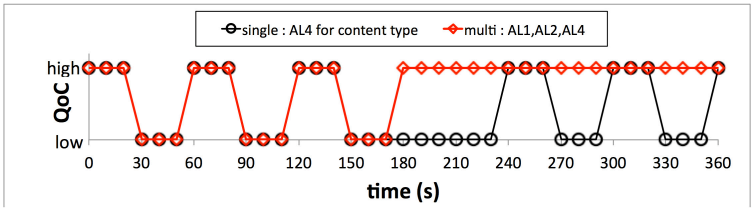
Figure 11: Implementation of AL_1 for the server load management



(1-a) response time



(1-b) server load



(1-c) QoC

Figure 12: Experimental results for single and multiple adaptation loops (User requests per second were 5 for 0 - 120 s, 7 for 120 - 240 s, and 5 after 240 s)

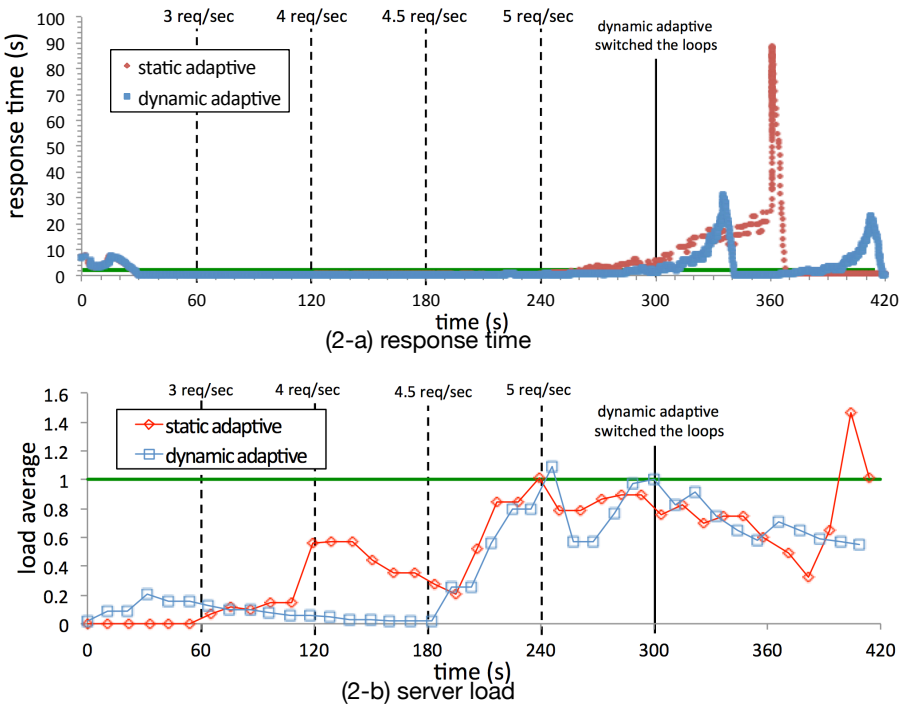


Figure 13: Response time and server load for the static and dynamic adaptive system

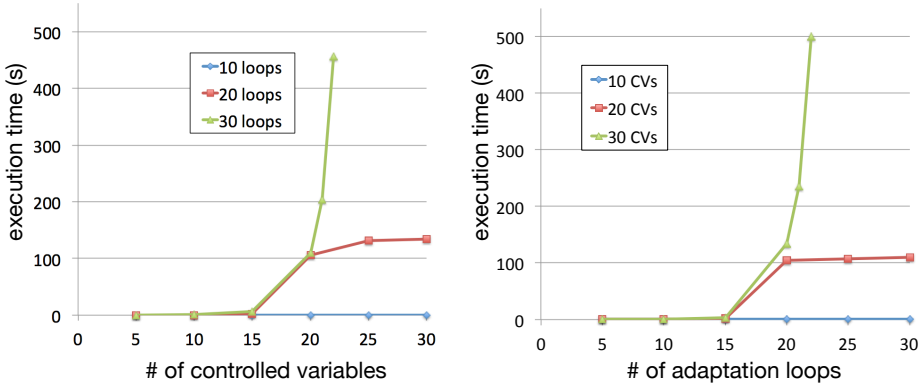


Figure 14: Execution time of the qualitative analysis