Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

RINGA: Design and verification of finite state machine for self-adaptive software at runtime

Euijong Lee[a,*], Young-Gab Kim[b,**], Young-Duk Seo[a,*], Kwangsoo Seol[a,*], Doo-Kwon Baik[a,*]

[a] Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea
[b] Department of Computer and Information Security, Sejong University, Seoul, Republic of Korea

A R T I C L E I N F O

Keywords:
Self-adaptive software
Model checking
Finite state machine
Runtime

A B S T R A C T

Context: In recent years, software environments such as the cloud and Internet of Things (IoT) have become increasingly sophisticated, and as a result, development of adaptable software has become very important. Self-adaptive software is appropriate for today's needs because it changes its behavior or structure in response to a changing environment at runtime. To adapt to changing environments, runtime verification is an important requirement, and research that integrates traditional verification with self-adaptive software is in high demand.
Objective: Model checking is an effective static verification method for software, but existing problems at runtime remain unresolved. In this paper, we propose a self-adaptive software framework that applies model checking to software to enable verification at runtime.
Method: The proposed framework consists of two parts: the design of self-adaptive software using a finite state machine and the adaptation of the software during runtime. For the first part, we propose two finite state machines for self-adaptive software called the self-adaptive finite state machine (SA-FSM) and abstracted finite state machine (A-FSM). For the runtime verification part, a self-adaptation process based on a MAPE (monitoring, analyzing, planning, and executing) loop is implemented.
Results: We performed an empirical evaluation with several model-checking tools (i.e., NuSMV and CadenceSMV), and the results show that the proposed method is more efficient at runtime. We also investigated a simple example application in six scenarios related to the IoT environment. We implemented Android and Arduino applications, and the results show the practical usability of the proposed self-adaptive framework at runtime.
Conclusions: We proposed a framework for integrating model checking with a self-adaptive software lifecycle. The results of our experiments showed that the proposed framework can achieve verify self-adaptation software at runtime.

1. Introduction

Nowadays, various software platforms (e.g., smartphone operating systems, cloud environments, Arduino, Raspberry Pi, and web-based applications) are available. As a result, various software applications depending on platforms such as the cloud and Internet of Things (IoT)[1] have become widespread. Furthermore, rapid advancements in mobile and IoT devices have led to a demand in software systems that can operate in various environments. Hence, self-adaptive software is software that changes its behavior or structure in a changing environment at runtime [1]. Self-adaptive software satisfies the current need for software that can operate in various environments. Verification is one of the most important tasks for self-adaptive software, and it needs to be performed at runtime [1]. To viably support runtime verification, the integration of traditional verification with self-adaptive software is preferable [2,3]. Model checking is an effective static verification method for software and is governed by the state-based model [4]. Despite excellent verification performance, model checking incurs a problem at runtime: state explosion [5]. Therefore, model checking needs to be integrated in the self-adaptation lifecycle during runtime verification.

There are several studies [6–13] on state machines and model checking in self-adaptive software. On one hand, some studies [6–8] apply state machines and model checking to self-adaptive software

* Corresponding author at: Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea.
** Corresponding author at: Department of Computer and Information Security, Sejong University, Seoul, Republic of Korea.
E-mail addresses: kongjjagae@korea.ac.kr (E. Lee), alwaysgabi@sejong.ac.kr (Y.-G. Kim), seoyoungd@korea.ac.kr (Y.-D. Seo), seolks@korea.ac.kr (K. Seol), baikdk@korea.ac.kr (D.-K. Baik).
[1] Full definition of acronyms is described in Annex.

http://dx.doi.org/10.1016/j.infsof.2017.09.008
Received 13 September 2016; Received in revised form 8 September 2017; Accepted 16 September 2017
0950-5849/ © 2017 Elsevier B.V. All rights reserved.

design and evaluation. Such methods are useful during design-time verification and post-processing for evaluating self-adaptive software, but they have limitations when applied at runtime for verification. On the other hand, some studies [10–12] apply probabilistic model checking to verify self-adaptive software at runtime and use an interactive state machine (ISM) to verify self-adaptive software that suffers from uncertainty. An ISM is a state machine that can interactively update its model and requirements in response to environment changes. We assume that the runtime environment of self-adaptive software is not predictable, and therefore, an ISM is more suitable for verifying self-adaptive software at runtime. However, previous studies on ISMs have only been optimized to solve uncertainty problems. Therefore, in this study, we propose a finite state model to design and verify self-adaptive software.

In this paper, to resolve the above design and verification issues, we propose two types of finite state machines for the design and verification of self-adaptive software. Furthermore, we propose a self-adaptive framework called RINGA (Runtime verIfication with fiNite state machine desiGn for self-Adaptation software) using the two types of finite state machines. RINGA consists of two parts: the design-time part is responsible for the design of finite and abstract-state machines for performance at runtime, and the runtime part consists of a MAPE (Monitoring-Analyzing-Planning-Executing) loop that analyzes the environment with the abstract state machine extracted from the design-time part. We performed an empirical evaluation using symbolic model checking tools, and our results show that the proposed method can be used to perform verification at runtime. Furthermore, we implemented an Android and Arduino application with an IoT-based example application in six scenarios to measure the adaptability of the proposed framework.

The remainder of the paper is organized as follows. Section 2 provides background on self-adaptive software and work related to runtime model checking for self-adaptive software. Section 3 introduces the proposed framework. Section 4 presents the empirical evaluation. Section 5 presents the results of experiments with a simple IoT-based example application. Section 6 discusses the limitations of the proposed approach and extensions that could overcome these limitations. Section 7 provides the concluding remarks and discusses future work.

2. Related work

In this section, we introduce various self-adaptive frameworks and platforms, as well as previous studies on the verification requirements in self-adaptive software research. We summarize previous research and compare it with RINGA in Table 1. Details regarding the various studies are described in subsections on the frameworks, platforms, and verification of self-adaptive software.

2.1. *Self-adaptive software frameworks*

As mentioned earlier, to adapt to changing environments at runtime, self-adaptive software dynamically changes its behavior or structure [1]. Therefore, self-adaptive software detects the state of an environment and changes its behavior or structure if possible when its aim has been violated [1]. That is, self-adaptive software monitors its environment and analyzes the situation and environment to adapt to any changes. To accomplish this, the MAPE loop mechanism was proposed [1,14–16] and implemented in several self-adaptive software and autonomic computers. A MAPE loop consists of four parts:

- Monitoring: responsible for collecting and correlating data from the environment and internal software changes.
- Analyzing: responsible for analyzing the symptoms related to situation changes using the monitored data.
- Planning: determines the adaptive strategies, i.e., is responsible for determining what is to be changed and how.
- Executing: responsible for activating the adaptive strategies.

Table 1
Comparison of previous research and RINGA.

Work by	Goal	Lifecycle	Model for system design	Technique for verification	Experimental domain
Tesei et al. [6]	Self-adaptive system design and verification of constraint violations	N/A	State machine	Model checking	Ecology
Abeywickrama and Zambonelli [7]	Self-adaptive software design using a goal-based model	N/A	SOTA (a goal-based model)	Model checking	e-mobility
Johnson et al. [9]	Reverification of component-based software system	N/A	Probabilistic model	Model checking	Cloud services
Filieri et al. [10–12]	Statically generated verification conditions at runtime	Control loop	Discrete-time Markov chain	Model checking	Server–client web and lower wireless buses
Yang et al. [13]	Verification of self-adaptive software involving uncertainty in environmental interactions	Reaction loop	ISM	Model checking	Robot cars
Tallabaci and Souza [17]	Adaptive system design using a goal-based model	MAPE loop	Goal-based model	Checking fulfillment of goal-based model	Automated teller machines
Barna et al. [18]	Self-adaptive cloud based exemplar	MAPE loop	N/A	N/A	Cloud environments
Garlan et al. [19]	Support reusable infrastructure	Control loop	Architecture	Software architecture and reusable infrastructure	Web-based client–server systems
Knauss et al. [21]	Adaptation of contextual requirements	MAPE loop	Contextual requirement	Machine learning, data mining	Activity-scheduling systems
Wuttke et al. [23]	Self-adaptive traffic routing based exemplar	N/A	N/A	N/A	Traffic routing
Weyns and Calinescu [24]	Self-adaptive service-based system exemplar	MAPE loop	N/A	N/A	Tele-assistance systems
RINGA	**Self-adaptive software design and verification at runtime**	**MAPE loop**	**SA-FSM, A-FSM**	**Model checking**	**Light control applications**