

Monitoring Methodology using Aspect Oriented Programming in Functional based System

Giljong Yoo, Eunseok Lee

Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 440-746, Republic of Korea
gjyoo@ece.skku.ac.kr, eslee@ece.skku.ac.kr

Abstract— The problem of system complexity has recently attracted much attention. The self-healing system that is considered as a novel solution is one of the autonomic computing methodologies. It automatically recognizes faulty states of the system and provides the ability of self recovery without human intervention. There have been plenty of studies on monitoring methodologies from both inside or outside of a system. However, most of those approaches have focused on sub-partitioning the system; therefore, in this paper, we need to invent another type of methodology that is capable of considering the system as a whole. To solve above mentioned problem, we adapt the AOP (Aspect Oriented Programming) technique to the proposed system that can configure scope/constraints of each function and discover even a tiny abnormal state. In addition to that, we proposed an architecture for sensing the abnormal flow of the entire system with utilizing a state diagram based on each function and a module-function relation list. Finally, we prove the proposed system by adapting it to the case study.

Keywords— AOP (Aspect Oriented Programming), Component Based System, Self-Healing, Monitoring Methodology

I. INTRODUCTION

An autonomic system is self-managing, meaning that it is self-protecting, self-configuring, self-healing and self-optimizing [1]. These four capabilities overcome the rapidly growing complexity of computing systems management and address user needs, faults in a dynamic environment. Especially, the core of autonomic computing is self-healing that analyses data and predicts potential problems in real-time.

The main functions of self-healing are as follows [2].

- A. Maintenance of health
- B. System recovery
- B. Detection of system failure

The self-healing system makes a healing plan after it detects a failure of the target system. For this reason, detecting system failure is the most important function among those functions. But, most existing studies focus on partial functional aspects such as the internal component of the system. Therefore, they overlook the overall aspect of the system.

So, we offer a wider scope monitoring methodology. The purpose of this paper is to offer a monitoring technique using AOP in functional based systems. In exploring the questions of monitoring of functional based systems, this paper will be limited to the consideration of self-healing.

This paper is organized as follows. Section 2 presents the research background and related work. Section 3 describes the proposed architecture. Section 4 evaluates the proposed approach via case study and experiment. Section 5 concludes the paper and identifies our future work.

II. RELATED WORK

System monitoring methodology is divided into two parts.

- A. The internal monitoring technique
- B. The external monitoring technique

The internal monitoring technique monitors a target system using a programming language such as the white box test. On the contrary, the external monitoring technique [3, 4] is a methodology where a monitoring module is located in an external system such as the black box.

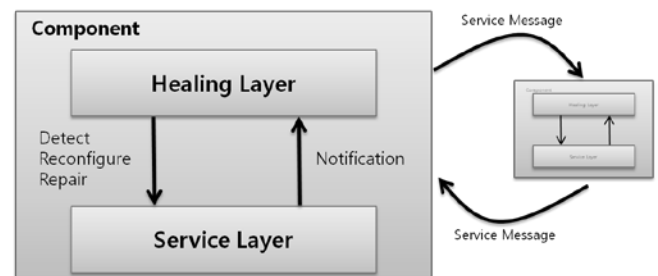


Figure 1. The component based self-healing architecture

Michael E. Shin proposed a component based self-healing approach [5]. This approach is one of the internal monitoring techniques. Each component consists of a service layer and a healing layer, as is shown in Fig. 1.

However, this method cannot adapt with monitoring the entire system flow because it focuses only on the internal component. Even if a component provides a normal service, the goal of the system may fail. Therefore, Michael E. Shin's system needs a wider aspect than the component level.

The aspect-oriented programming (AOP) is a programming paradigm that increases modularity by allowing the separation of cross-cutting concerns, forming a basis for aspect-oriented software development [6]. M. Richters presents the aspect-oriented monitoring of UML and OCL Constraints [7]. This method monitors a constraint of system modelling using AOP with UML and OCL (Object Constraint Language) [8]. However, he focuses on the point-cut of AOP for maintaining a constraint of system. So, even if M. Richters' point-cut has no problem, the operation flow of the entire system may be faulty.

W. N. Robinson suggests the requirements monitoring framework for monitoring the user's needs. This method uses the KAOS (Knowledge Acquisition in Automated Specification) to design the requirement model for the ReqMon [9, 10]. However, this method is hard to use.

Therefore, we propose a functional based monitoring approach to mitigate the aforementioned problems. We focus specially on the 'Detection of system failure' for achieving self-healing. Our monitoring scope determines whether or not the function, the system and the user requirements have been violated.

III. PAPER SUBMISSION

System architecture consists of function, a module as shown in Fig. 2. The function means the minimum executable unit in the application or system. The module is an independent agent that has a goal in the system. The module executes various functions, and the execution result is reflected in the system

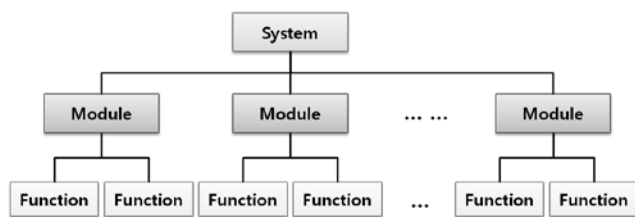


Figure 2. Basic system organization

The function creates a result value of its execution with many input and variables according to the situation change. In other words, it monitors an abnormality status through checking 1) input values, 2) variables & their results and 3) output values (changed variables) of the function.

The monitoring agent determines what happened over in the executing function and perceives the effect of execution. Then, it is possible to fully monitor the entire system or to partially monitor the system. Figure 3 is the architecture of the proposed approach.

The Controller controls the Function-Module Relation List, the Function based State Diagram and the Scope/Constraint DB.

The Function-Module Relation List is used for perceiving a violation of simple requirements. This list includes the effected requirements when a module uses a specific

function. Therefore, if an error occurs, then the Controller shows the related requirements of the error to the user.

The Function based State Diagram describes the state of the entire flow based on the function. We can perceive the current state of a system and monitor a transition of the state of the entire flow.

The scope/constraint of the function is the basis for monitoring an abnormality of input/output related variables via the Function Monitor.

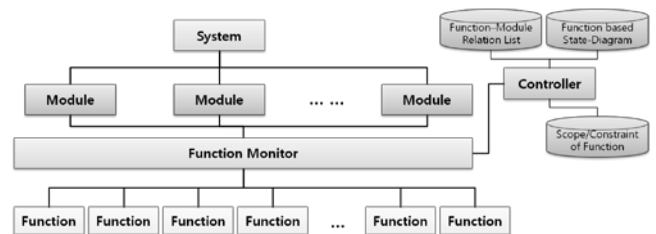


Figure 3. Proposed architecture for the functional based monitoring

The monitoring process of the Function Monitor is as follows:

First, the process of a monitoring before function execution:

- 1) Reporting the module and function set to the Controller.
- 2) Receiving the scope/constraint of the function from the Controller.
- 3) Checking and reporting any abnormalities of the input value.

Second, the process of a monitoring after function execution:

- 1) Checking the constraint of the output and the related value.
- 2) Reporting execution complement and abnormality, or the lack of, to the Controller.

The monitoring process of the Function Monitor is as follows:

First, the process of monitoring before function execution:

- 1) Sending the scope/constraint of function to the Function Monitor.
- 2) Checking the state diagram and confirming an abnormal state.

Second, the process of monitoring after function execution:

- 1) Checking an abnormal state in the state diagram.

Third, if an error has occurred then...

- 1) Checking any requirement violation between the function and the module.

IV. CASE STUDY

We design a messenger system for evaluating our approach. Our experiment environment has four modules and seven functions as shown in Fig. 4.

The basic function of the messenger system is as follows:

- 1) Message sending/receiving
- 2) History of message sending/receiving

The basic requirement of the messenger system is as follows: We assume that the network state is good.

- 1) The message must be sent and received within 3 seconds.
- 2) Only the authenticated user can send and receive a message.
- 3) Each user has an address book.
- 4) The user can confirm the history of the message sending/receiving, address book.
- 5) It must always record the user who sends and receives the message.
- 6) It must always record the time of sending and receiving the message.
- 7) It must always record the message that is sent and received.

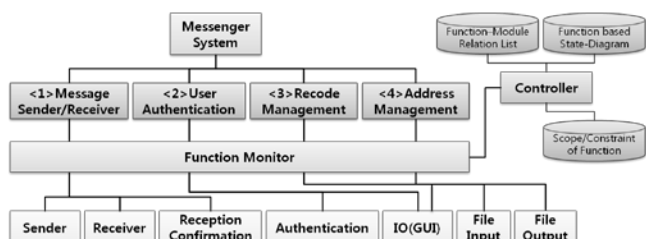


Figure 4. The Messenger system of the case study

A. Function-Module Relation List

The function-module relation list has the requirements of a function and a module. Table 1 is an example of the function-module relation list of the case study.

Table 1. An example of the Function-Module Relation List

Function	Module	# of Requirement
File Input	Record management	5, 6, 7
Message reception confirmation	Message sending/receiving	1

B. Function based State Diagram

The Controller has a function state diagram such as shown in Fig. 5. The Controller confirms the state of the function and performs the modification of the state. The <number> of Fig. 5 means message sender/receiver, user authentication, record management and address management.

Therefore, the monitor system can determine the changes of each state using the state diagram through monitoring the function inside the module. Also, the monitor system monitors whether or not the module is suited to one's purpose. The module must use the suitable and right function.

C. Scope and Constraint of Function

The scope and constraint of each function are described in a particular file. Figure 6 shows the scope and constraint of the message reception function. This function is registered in the Controller and is performed by the function monitor.

Module	Func	Ptime	Type	Dtype	Size1	Size2	Req
M/S/R	Send	Before	input1	string	>0	<20	ti=System.currentTimeMillis();
M/S/R	Send	Before	input2	string	>=0	<512	
M/S/R	Send	Before	t1	long	!=0		
M/S/R	Send	After	output	bool			cu=(System.currentTimeMillis()-ti)/1000;
M/S/R	Send	After	cu	long			if(cu>3) r1_error();

Figure 6. The scope and constraint of the message reception function

The column 'Module' and 'Func' presents the modules and functions of the target system. The column 'Ptime' describes the execution of the function before and after. The column 'Dtype' presents a data type. Column 'Size1' and 'Size2' determine the constraint of the variable. The column 'Req' describes an additive constraint of the function.

Table 2. A comparison between the existence and nonexistence of monitoring functionality

Monitoring Functionality	○	×
Execution time (average)	47ms	45ms
Requirement violation check	○	×
Function call check	○	×
Abnormal region check	○	×

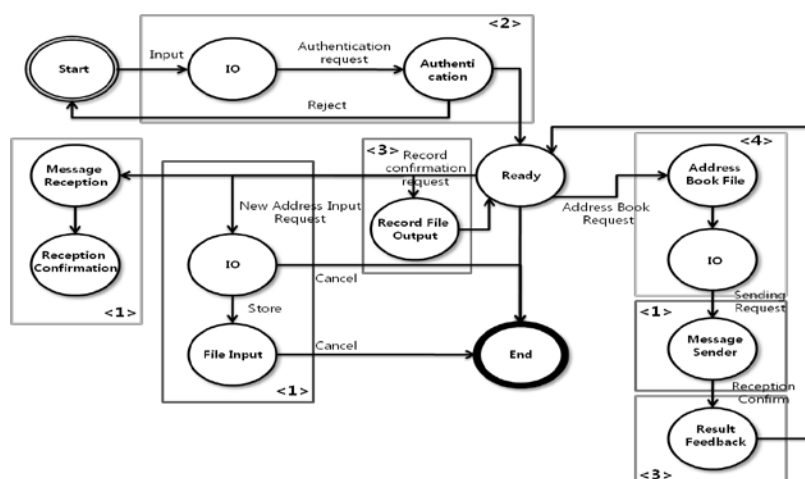


Figure 5. An example of the function based state diagram

```

Starting Message Manager
ID: gjyoo
PW: *****
$Send
1. Allice
2. David
3. Micheal
Enter the receiver's number : 0
Variable input1 is not valid
Fail to make a transition from State 3
Related Module and Function is module Sender and function Send
Related Requirements :
Requirement 1 : Couldn't send the message in 3 seconds
Requirement 4 : Send message / managing history failed
System failure occurred

```

Figure 7. An example of the occurrence of an error

The experimental result is shown in figure. 7 that shows a problem when the 'send' function generated an error due to the wrong parameter. If some problem is detected, then the target system triggers an automatic system shutdown. Therefore, the monitoring system can detect an abnormal module, function and the related requirement.

A system comparison of existence or non-existence of the monitoring functionality is shown in Table 2. We test ten-around operations to measure the average running time.

V. CONCLUSION

Four capabilities of autonomic computing overcome the rapidly growing complexity of computing systems management and address user needs, faults in a dynamic environment. Especially, the core of autonomic computing is self-healing that analyses data and predicts potential problems in real-time. The most important technique for self-healing is the monitoring methodology. In this paper, we propose an architecture that is capable of detecting the failure states of each function and of the entire system and confirms the requirement lists that cause such failure states. In addition, we address the role of each component of the proposed system through experiment. The experiment result of these features is as follows:

- 1) Each function is controlled depending on the situation through the scope and constraints of this system, and it achieves a detailed observation of the specified requirement.
- 2) We monitor the flow of the system through the function based state diagram and check the inadequate use of function/module.
- 3) The monitoring system perceives the related abnormal requirement if an error has occurred in the function-module relation list
- 4) We monitor the error location via the function based state diagram and the function monitor.
- 5) Our approach easily provides system modification and management.

We monitor the abnormal state that is generated from the function flow. We also monitor the related requirement of an abnormal function. Therefore, our approach provides advanced information of diagnosis and recovery using the monitoring methodology of self-healing.

There is still much to be learned about an efficient function based system in a more general sense to answer

questions such as (1) describing a state diagram and module-function relation list in the design phase (2) the need for autonomous monitoring and healing to resolve many failures of requirements.

ACKNOWLEDGEMENTS

This work was supported by the Korean Science and Engineering Foundation (KOSEF) grant funded by the Korean government (MEST) (No. 2009-0077453), Corresponding author: Eunseok Lee.

REFERENCES

- [1] Ganek, A. G., and Corbi, T. A., 2003. The dawning of the autonomic computing era. *IBM Syst. J.* 42 (1), 5–18.
- [2] Debanjan Ghosh, Raj Sharman, Raghav H. Rao, and Shambhu Upadhyaya. 2007. Self-healing systems - survey and synthesis. *Decis. Support Syst.*, 42(4):2164–2185.
- [3] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct 2004.
- [4] D. S. Wile, and A. Egyed, 2004. An Externalized Infrastructure for Self-Healing System. *Computer Society. IEEE*, 1-4.
- [5] M. E. Shin, 2005. Self-healing components in robust software architecture for concurrent and distributed systems. *Science of Computer Programming* 57, 27-44.
- [6] Aspect-Oriented Programming. http://en.wikipedia.org/wiki/Aspect-oriented_programming.
- [7] M. Richters and M. Gogolla. 2003. Aspect-oriented monitoring of UML and OCL constraints. In *Proc. of The 4th AOSD Modeling With UML Workshop*.
- [8] Unified Modeling Language. <http://www.uml.org>
- [9] W. N. Robinson. 2006. A Requirements Monitoring Framework for Enterprise Systems," *Requirements Engineering*, Springer 11(1), 17-41.
- [10] R. Westdijk, L. Rothkrantz, and A.V. van Leijen, 2007. Applying Requirements Monitoring for Autonomic Computing in a Combat Management System. In *Proc. of Autotestcon 2007*. 349-358.