Self-healing Mechanism for Reliable Computing*

Jeongmin Park
School of Information and Communication Engineering, Sungkyunkwan University, Suwon 400-746, South Korea
jmpark@ece.skku.ac.kr

Jinsoo Jung
School of Information and Communication Engineering, Sungkyunkwan University, Suwon 400-746, South Korea
seba702@ece.skku.ac.kr

Shunshan Piao
School of Information and Communication Engineering, Sungkyunkwan University, Suwon 400-746, South Korea
sspiao@ece.skku.ac.kr

Eunseok Lee
School of Information and Communication Engineering, Sungkyunkwan University, Suwon 400-746, South Korea
eslee@ece.skku.ac.kr

Abstract

One of the four major processes of an autonomic computing system is to free people from discovering, recovering, and failures, this process is called self-healing. Systems designed to be self-healing are able to heal themselves at runtime in response to changing environmental or operational circumstances. Thus, the goal is to avoid catastrophic failure through prompt execution of remedial actions. This paper proposes a self-healing mechanism that monitors, diagnoses and heals its own internal problems using self-awareness as contextual information. The self-management system that encapsulates the self-healing mechanism related to reliability improvement addresses: (1) Monitoring layer, (2) Diagnosis & Decision Layer, and (3) Adaptation Layer, in order to perform self-diagnosis and self-healing. To confirm the effectiveness of self-healing mechanism, practical experiments are conducted with a prototype system.

1. Introduction

Reliability is a measure of trustworthiness of a computing system, which can be defined as the probability for component, communication, service or system to successfully achieve their tasks and objectives. One of the four major processes of an autonomic computing system is to free people from discovering, recovering, and preventing system failures, this process is called self-healing. The Self-healing technologies improve system reliability by eliminating

* This work was supported in parts by ITRC IITA-2007-(C1090-0701-0046), Grant No. R01-2006-000-10954-0, Basic Research Program of the Korea Science &Engineering Foundation, and the Post-BK21 Project. Corresponding author: Eunseok Lee

or dramatically reducing the requirement for human operation, as human configuration and maintenance of complicated systems is often error prone [1].

Approximately 40% of all computer problems are attributable to errors made by system administrators [4]. Thus, the current system management method, which depends mainly on professional managers, is required to be improved and are increasingly expected to dynamically self-manage to accommodate resource variability, detect fault appearances, and recover from system failure.

The traditional approaches for improving reliability have been focused on log-based level [8,9], model-based level [6,7], component-based level[16,19]. Most approaches support part of the process of the self-healing mechanism [16], but not whole process including monitoring, filtering, translation, analysis, diagnosis, decision and healing.

Thus, this paper concentrates on the whole process of the self-healing mechanism for reliable system and describes the architecture of self-healing. The architecture that encapsulates the self-healing mechanism related to reliability improvement is designed with three layers: the monitoring layer, the diagnosis & decision layer, and the adaptation layer. This paper has two central goals: 1) to describe the proposed layered architecture that encapsulates the self-healing mechanism; and 2) to present each phase of the self-healing mechanism in detail. To confirm the self-healing mechanism, practical experiments are conducted with a prototype system.

This paper begins by describing related work in Section 2. Section 3 describes layered software architecture for self-healing. Section 4 describes the process of the self-healing mechanism including monitoring, filtering, translation, analysis, diagnosis, decision and healing. In Section 5, implementation and evaluation are discussed. Finally, in Section 6, conclusions are presented.

2. Related work

Self-healing system has the capability to modify its actions in response to changes such as system faults, resource variability and so on. These conventional self-healing behaviors are the essence of the self-healing system or, in other words, the self-healing system must be composed of these self-adaptive behaviors for reliability improvement. Oreizy et. al. [6] proposed the following processes for self-healing software: Monitoring the system, planning changes, Deploying change descriptions and enacting the changes [7]. According to our research, the conventional reliable self-healing mechanisms could be divided into three categories. In this section, we analyze advantages and disadvantages of the conventional reliable self-healing mechanisms: 1) Internal adaptation mechanism, 2) Model-based mechanism and 3) Log-based mechanism.

2.1. Internal adaptation Mechanism

In the past, system adaptation has largely been handled internally to the application. For example, applications typically use generic mechanisms such as exception handling, or heartbeat mechanisms to trigger application-specific response to an observed fault. For other aspects of adaptation, such as resource-based adaptation, systems typically wire in application-specific policies and mechanisms. For example, a video teleconferencing application may decide how to reduce its fidelity when transmission

bandwidth is low using some combination of compression and reduced framesize and resolution [20]. Code based mechanism that allows a system to detect and recover from errors are typically wired into applications at the level of code where they are hard to change, reuse, or analyze. However, the problems in these internal mechanisms can be summarized as follows:

- They make it difficult to change adaptation policies, because they are so intertwined with the normal code of the system.
- Most of the modern computing systems are usually composed of solutions from different vendors. Because of this heterogeneous nature, there would be some difficulty to achieve the reliable ubiquitous computing.

2.2. Model-based Mechanism for Self-healing

[3,7,13] used architectural models as the basis for monitoring, problem detection, and repair for self-healing systems. The architectural models can be specialized to the particular style of the system such as reliability, performance or security. However, they have the following disadvantages:

- In system level, they are geared towards specific system topology, software applications [2].
- It is very difficult to model the target system with respect to the normal state.

2.3. Log-based Mechanism for Self-healing

[8,9] proposed the log-based system that consists of a 5-step process, including *Monitoring, Translation, Analysis, Diagnosis and Feedback*. These 5 processes are applied in the form of self-adaptive behaviors. The functions are as follows:

- Firstly, the *Adapters* [9] monitor the logs from the various components (Monitoring).
- Secondly, the *Adapter* [9] translates the log generated by the component into the Common Based Event (CBE) format (*Translation*).
- Thirdly, the *Autonomic Manager* [9] analyzes the CBE log. This step identifies the relationship between the components through dependency (*Analysis*)
- Fourthly, the *Autonomic Manager* [9] retrieves the appropriate healing method by means of the *Symptom Rule* [9] and *Policy Engine*[9] and then applies the healing method to the applicable component. The feedback from the *Resource Manager* [9] enables the system to heal itself (*Diagnosis and Feedback*).
- Finally, in the event that the component has a critical problem, and cannot be solved easily, the Autonomic Manager transmits a *Call Home Format1* [9] message to the Support Service provider (SSP) / Vendor, requesting assistance.

[1] Call Home Format: This is the message transmission code between healing system and SSP/Vendor that IBM&CISCO are accepted for standardization.

However, the problems in these existing systems can be summarized as follows:

- The size of the log in the CBE format is larger than that of the untranslated log (this will reduce system performance).
- The disk, CPU and memory usage drastically increase in the process of conversion, due to the complex calculations involved.
- They require a reasonably long healing time, however, the immediate action time corresponding to emergency situations is generally short.
- Furthermore, in the event that the component does not generate a log, it is impossible for the system to heal itself.

In this taxonomy, approaches for the reliable system were characterized with internal adaptation mechanism, model-based mechanism and log-based mechanism. According to analyzed the merits and the demerits, we propose a *self-healing mechanism* that models, monitors, filters, translates, analyzes, diagnoses, decides and heals internal problems.

3. Proposed system

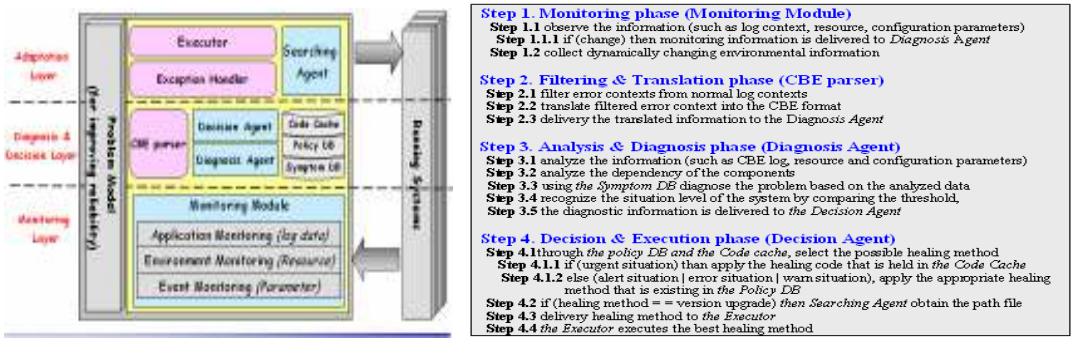

Figure 1. Layered architecture and overall behavior for self-healing

In this section, we describe proposed layered architecture that encapsulates the self-healing mechanism including modeling, monitoring, filtering, translating, analyzing, diagnosing, deciding, and healing internal problems. Self-healing architecture is designed as a layered architecture, structured with three layers (Fig.1) – the monitoring layer, Diagnosis & Decision Layer and Adaptation Layer. The monitoring layer consists of modules for monitoring the information such as log context, resource, configuration parameters. Once monitoring module in the monitoring layer detects an anomalous behavior and presumes that the behavior needs to be treated. In the abnormal phase, modules in the diagnosis & decision layer are triggered. The diagnosis & decision layer constitutes modules that filters, translates, analyzes, diagnoses the problems, and decides its strategy. Finally, the adaptation layer composes modules that execute the

http://www.cisco.com/application/pdf/en/us/guest/partners/partners/c644/ccmigration_09186a0080202dc7.pdf

strategy selected in diagnosis & decision layer. The behavior corresponding to the overall architecture is presented in Fig.1.

3.1. Self-healing Mechanism

Through the historical data, we suppose that solving the problem caused in known fault may be a reliable system. Fig. 2 depicts the self-healing process against anomalous behaviors. The self-healing process is divided into two steps – preprocess steps and adaptation steps. In the preprocess steps, in order for us to model the target system, we input the goal of the target system and model the known problems causing abnormal behaviors. The problem model can be classified as generic problem and specific problem. An example of configuration problem can be illustrated as a generic problem, and dependency problem arising in configuration problem can be taken as an example of a specific problem. Self-healing in this paper focuses on the generic model that models the configuration problem caused in known fault.

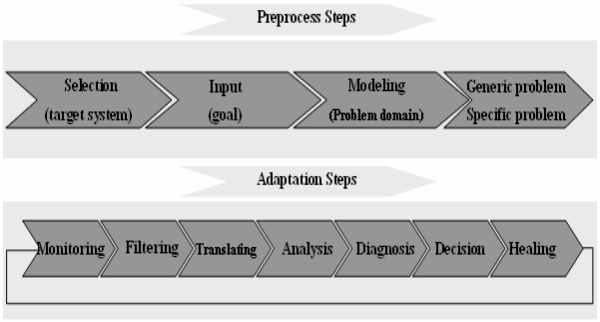

Figure 2. Self-healing process

In the adaptation steps, on the basis of the problem model, we use logs data to help in solving problem. Our self-healing mechanism carries out (1) monitoring to observe behavior properties of problem model, (2) filtering to extract error log context from normal log context, (3) translating the filtered error context into the CBE format, (4) analyzing the CBE log, resource information and the dependency of the components, (5) diagnosing problems from observed symptoms, (6) through the policy DB and the code cache, decision to the possible healing method and (7) applying the appropriate healing method.

3.1.1. Common Base Event (CBE)

Common Base Event (CBE) specification defines a new mechanism for managing events in application and how to communicate self-healing events in the autonomic computing model. The following 3-tuple information is captured per each event: (1) the reporting component, (2) the affected component, and (3) the situation. The reporting component is the component being affected by the situation [5]. After analyzing logs from different products, CBE concluded that different events that probably occur in computing systems can be categorized into eleven predefined situations; and one user-defined situation [5]. The set of predefined situations includes: *Start Stop*, *Connect*, *Request*, *Configure*, *Available*, *Report*, *Create*, *Destroy*, *Feature*, and *Dependency* situations. For

example, *FeatureSituation* denotes that some feature has become either available or unavailable on some component. Each of those situations has some parameters, such as *reasoningScope* that denotes whether the impact of the situation is internal to the affected component or it propagates to other components.

3.1.2. Monitoring Layer (*Monitoring*)

As shown in Fig 3, the functions of the Monitoring Module in the monitoring layer are as follows: It monitors resource (such as RAM, CPU, etc) status and the size of the log file generated by the application. To deal with paradoxical situation where the self-healing system itself may need healing, it monitors error events arising in the predefined State Model. Through resource status, log files and error state arising in the self-healing system, if the Monitoring Module detects suspicious events of the component, it deliveries the monitored list to the CBE Parser in the Diagnosis & Decision layer. Also, if it monitors the anomaly behaviors arising in the self-healing system, it deliveries the error state to the Diagnosis Agent.

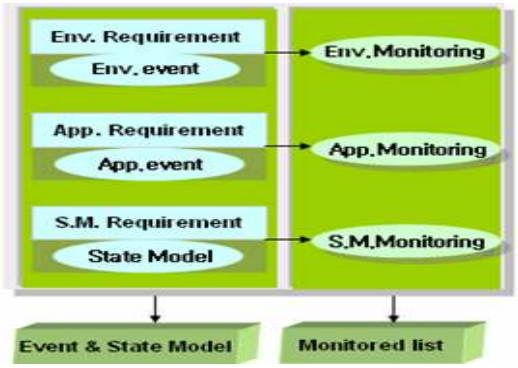

Figure 3. Monitoring Layer's behavior

3.1.3. CBE Parser (*Filtering and Translation*)

The functions of the CBE Parser are as follows: It gathers the monitored list provided by the Monitoring Module, filters error context from normal log context File. The error context is filtered by means of designated keywords, such as "*not", "error", "reject", "notify",* etc. It translates the filtered error context into the CBE format, and deliveries the translated information to the Diagnosis Agent.

3.1.4. Diagnosis Agent (Analysis and *Diagnosis*)

The first major attribute of a self-healing system is self-diagnosing [9]. The Diagnosis Agent analyzes the CBE log, resource information (received from the Monitoring Module), the dependency of the process and the state of self-healing system and then diagnoses the current problem (through *the Rule Model*). It suggests the recovery actions to automatically resolve problems from observed symptoms. The results of the diagnosis can be used to trigger automated reaction. If the self-healing

system itself needs healing, the diagnosis agent calls an administrator. As shown in Error Level of Fig. 4, it classifies the Error Event, and sets up the priorities. The results of the diagnosis recognize the situation level. In addition, the Diagnosis Agent generates the Error Report and modifies the CBE. The Error Report is an administrator document, and the CBE is a document for the system. Using the first-order logic, we can recognize the situation level of system and represent the policy for it. Fig. 4 illustrates context predicate and its example.

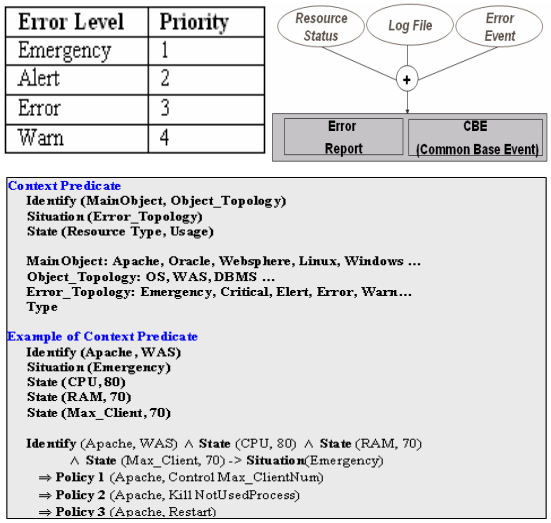

Figure 4. Recognition of System level

3.1.5. Decision Agent and Executor (*Decision and Execution*)

With the recovery actions, the Decision Agent uses a policy database that maintains high-level policies (for example, *"If (daytime) then do not restart service x"),* and then can take proactive and immediate action corresponding to Emergency situation (Priority '1') through the code cache. According to the policy database, a suitable policy is implemented. The Executor then executes the best healing method. The Decision Agent handles emergency situations in accordance with the Rule Model, and applies the best healing method.

Through the information delivered by the Diagnosis Agent, The Decision Agent determines the appropriate healing method with the help of the *Policy DB*. It also receives feedback information from the administrator in order to apply the more efficient healing method. The Information received from the Diagnosis Agent is used to determine the healing method. The Decision Agent determines the solutions that can be classified into root healing, temporary healing, first temporary healing and second root healing. Temporary healing is a way of resolving a problem temporarily, such as disconnecting a network connection, assigning temporary memory. The root healing is the fundamental solutions on the diagnosed result, including re-setting, restarting, and rebooting. The Decision Agent stores the methods in the DB as below Table 1, and decides how to select the appropriate healing method. The Table is the table to determine the optimal resolution method by analyzing given attributes. Looking at the **DECISION** column, when placed under the current diverse context, it helps to determine **R** (Root Solution), **T** (Temporary Solution), or **TR** (first Temporary Solution, second Root Solution). The **FEEDBACK** Column is showing feedbacks that were executed by the System Agent to heal the system.

Table 1. Decision Table

PROBLEM	TEMPRARY SOULTION	ROOT SOLUTION	CURRENT JOB	FUTURE JOB	AVAILABLE MEMORY	DECISION	FEEDBACK
CONNECT	CHANGE CONFIGURATION	REINSTALL	NULL	NULL	80%	T	POSITIVE
AVAILAVBLE	NULL	RESTART	NULL	NULL	80%	R	POSITIVE
OVERFLOW	ALLOCATE MEMORY	REBOOT	BACKUP	NULL	30%	TR	POSITIVE

The Decision Agent compares the fields with the information received by the System Agent, these fields are **CURRENT JOB**, **FUTURE JOB** and **AVAILABLE MEMORY**. If the value of the **FEEDBACK** Column is **POSITIVE**, the appropriate method is determined.

3.1.6. Searching Agent, Code Cache and Rule Model

Searching Agent is used to search the vendor's website for the knowledge required to solve the problem. This Agent uses search engine (such as *Google*). It sends the resulting search information to the administrator. The Code Cache is used to provide healing code to solve the error of the component arising in *emergency situations*. Separation of concerns [14] often provides some powerful guidance to complicated problems. Separation of concerns has led to the birth of many novel technologies such as aspect-oriented programming, subject-oriented programming. We used Rule Model approach [15] for self-healing: extract scattered rules from different procedures.

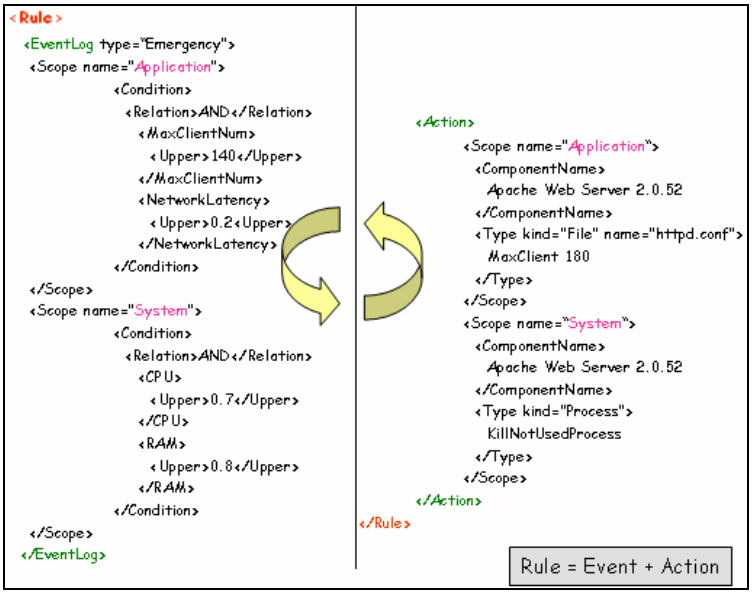

Figure 5. Rule Model of the proposed system

We consider only events that are involved in adaptive rules: Errors or failures that occur when procedures are executed. Most modern languages (such as Java) provide exception capture mechanisms. These types of events need self-healing to make software robust. Using the *Rule Model* that reconfigures services of the proposed system, agents can apply the appropriate

adaptation policy. Fig 7 shows that suitable actions are selected via a Rule Model. The Rule Model document identifies the different events that can be applied, namely the "*Emergency", "Alert", "Error"* and *"Warn" situation*. Theses situations have actions, linked to their respective situation, and then services of the proposed system are reconfigured by this strategy. We can understand their behavior from the above document. If the agent classifies the current situation as an emergency situation, it acts transformed code to heal the component. The above rule represented in XML is transformed to:

```
if(Scope=="Application"){
    if(ComponentName=="Apache Web Server 2.0.52"){
        if(TypeKind=="File"){
            if(Apache.MaxClientNum()>=140)&&(Net.Latency()>=2)
                Apache.MaxClientNum()=180;
        }
    }
}
else if(Scope=="System"){
    if ((ComponentName=="Apache Web Server 2.0.52"){
        if(TypeKind=="Process"){
            if(CPU.Utilization()>0.7&&RAM.Utilization()>=0.8)
                MySystem.KillNotUsedProcess();
        }
    }
}...
```

4. Implementation and Evaluation

As show in Fig 1, according to the layered architecture for self-healing, we represented the behavior of the agents. The implementation environment is as follows: we employed .NET Framework, JAVA SDK1.4, and used Oracle9i as the DBMS. Also we used JADE1.3 for the Agent development. The sample log used for the self-healing process was the contexts of log that are generated with *APACHE*. We implemented *the Agents* of the proposed system (in the form of a JADE Agent Platform [12]). Each of the agents is registered with each of the containers, and the ACL (Agent Communication Language) is used to communicate among the agents. We performed the simulation using six agents. The Fig 6 shows extracted log data and resource monitoring for self-management. The proposed system was evaluated and compared qualitatively and quantitatively in terms of the Log Monitoring Module, the Filtering & Translating Efficiency, and the healing Time.

Figure 6. The result of the Monitoring Module

(a) Log Monitoring Test. In the existing system, if the number of components is Ω, the system has to have Ω processes to monitor the log. In the proposed system, however, only one process is needed to monitor the log, as shown in Fig. 7. In this figure, the proposed system demonstrates its ability to stay at a certain level of memory usage, even when the number of components is increased.

(b) Filtering & Translation Efficiency Test. In the proposed system, the Component Agent searches for a designated keyword (such as "not", "reject", "fail", "error", etc.) in the log generated by the components. By using this approach, we were able to increase the efficiency of the system, in terms of the size of the log and the number of logs. We analyzed up to 500 logs, filtered out those logs not requiring any action to be taken, and evaluated the number and size of the logs in the case of both the existing and proposed systems. As a result of the filtering process, only about 20% of the logs were required for the healing process, as shown in Fig. 7. Therefore, the proposed system reduces the number and size of the logs, which require conversion to the CBE format.

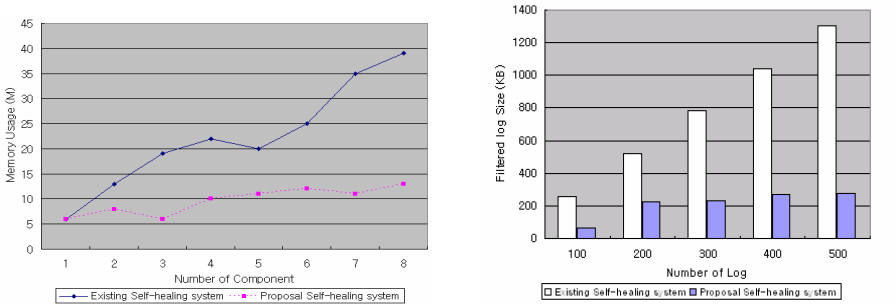

Figure 7. Memory usage and comparison of size and number of logs

(c) Average Healing Time Measurement. We measured the Average Adaptation Time arising in the existing self-healing system and the proposed self-healing system. For each adaptation time, we verified that the proposed system's parsing time and healing time are fastest than the existing system's, and rapidly responded problems arising in the urgent situation. However, because the number of monitoring factors is a little more, although the proposed system's monitoring time was relatively a little load through providing the meaningful much more information we verified that high quality of healing have been increased by monitoring information. Although In the event that the error component does not generate log, we couldn't measure the healing time arising in the existing self-healing system because the existing system was log-based healing system.

5. Conclusion

This paper has described self-healing mechanisms for reliable system. The monitoring layer consists of modules for monitoring the information such as log context, resource, event state. Once monitoring module in the monitoring layer detects an anomalous behavior and presumed that the behavior needs to be treated. In the abnormal phase, modules in the diagnosis & decision layer were triggered. The diagnosis & decision layer constituted modules that filters, translates, analyzes, diagnoses the problems, and decides its strategy. Finally, the adaptation layer composed modules that execute the strategy selected in diagnosis & decision layer. The advantages of this system are as follows. First, when prompt is required, the system can make an immediate decision and respond right away. Second, the Monitoring module monitors the generation of the log on the fly. Third, before converting the log into the CBE (Common Base Event) format, filtering is performed in order to minimize the memory and disk space used in the conversion of the log. Fourth, using the *Rule Model*, the appropriate adaptation policy is selected. However, further decision mechanism is likely to need to select the appropriate adaptation policy. Moreover this approach may be extended for the intelligent sub-modules in the Diagnosis & Decision layer.

References

[1] Yuan-Shun Dai, “Autonomic Computing and Reliability Improvement”, the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, May. 2005, pp.204-246.

[2] Rajesh Kumar Ravi, Vinaya Sathyanarayana, “Container based framework for Self-healing Software system”, the 10th IEEE International workshop on Future Trends of Distributed Computing system, May. 2004, pp.306-310.

[3] David S. Wile and Alexander Egyed, “An Externalized Infrastructure for Self-Healing Systems”, the 4th Working IEEE/IFIP Conference on Software Architecture, Jun. 2004, pp. 285-288.

[4] http://www.ibm.com/autonomic

[5] B. Topol, D. Ogle, D. Pierson, J. Thoensen, J. Sweitzer, M. Chow, M. A. Hoffmann, P. Durham, R. Telford, S. Sheth, T. Studwell, “Automating problem determination: A first step toward self-healing computing systems”, IBM white paper, Oct. 2003.

[6] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, Alexander L. Wolf, “An Architecture-Based Approach to Self-Adaptive Software”, IEEE Intelligent Systems, May. 1999, vol. 14, no. 3, pp. 54-62.

[7] David Garlan, Bradley Schmerl, "Model-based adaptation for self-healing systems", the 1st Workshop on Self-Healing Systems, Nov.2002, pp. 27-32.

[8] J. Baekelmans, P. Brittenham, T.Deckers, C.DeLaet, E.Merenda, BA. Miller, D.Ogle, B.Rajaraman, K.Sinclair, J. Sweitzer “Adaptive Services Framework”, CISCO white paper, Oct. 2003.

[9] B. Topol, D. Ogle, D. Pierson, J. Thoensen, J. Sweitzer, M. Chow, M. A. Hoffmann, P. Durham, R. Telford, S. Sheth, T. Studwell “Automating problem determination: A first step toward self-healing computing system”, IBM white paper, Oct. 2003.

[10] MarkWeiser, “The Computer of 21st Century”, Scientific American, 265(3), Sep. 1991, pp94-104.

[11] Anand Ranganathan, Roy H. Campbell, “An infrastructure for context-awareness based on first order logic”, Personal and Ubiquitous Computing, Dec. 2003, Vol. 7, Issue 6, pp. 353-364.

[12] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco (TILAB, formerly CSELT) Giovanni Rimassa (University of Parma): JADE PROGRAMMER’S GUIDE.

[13] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmer, Peter Steenkiste, “ Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure,” IEEE Computer, Oct. 2004, Vol.37, No.10, pp. 46-54.

[14] David. Parnas, “Designing Software for Extension and Contraction”, the 3rd International Conference on Software Engineering, Jul. 1978, pp. 264-277.

[15] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, “Aspect-Oriented Programming”, the European Conference on Object-Oriented Programming, Springer-Verlag LNCS 1241. June 1997.

[16] Qianxiang Wang, “Towards a Rule Model for Self-adaptive Software” ACM SIGSOFT Software Engineering Notes Page 1 Jan. 2005, Vol.30, No.1, pp. 1-5.

[17] Michael E. Shin, “Self-healing components in robust software architecture for concurrent and distributed systems”, Science of Computer Programming, Jul. 2005, Vol. 57, Issue 1, pp. 27-44.

[18] E. Grishikashvili, R. Pereira, A. Taleb-Bendiab “Performance Evaluation for Self-Healing Distributed Services”, the 11th International Conference on Parallel and Distributed Systems, Feb. 2005, pp.135-139.

[19] V. Issarny and J. P. Banatre., “Architecture-based Exception Handling”, the 34th International Conference on System Sciences, Jan. 2001, pp.1-10.

[20] Garlan, D., Schmerl, BR, and Chang, J. “Using Gauges for Architecture-Based Monitoring and Adaptation”, The Working Conference on Complex and Dynamic System Architecture, Dec. 2001.

Authors

Jeongmin Park. received the B.S. degree in Computer Engineering from Korea Polytechnic University, Korea, in 2003, M.S. degree in Computer Engineering from Sungkyunkwan University, Korea, in 2005. He is an adjunct professor of the Department of Computer Engineering of Korea Polytechnic University. He is currently a doctoral candidate at Sungkyunkwan University and a member of the Software Engineering Lab. His research interests include the area of autonomic detection and diagnosis in distributed component-based system.

Jinsoo Jung received B.S. degree in Computer Engineering from Sungkyunkwan University, Korea, 2007. He is currently a M.S. candidate at Sungkyunkwan University and a member of the Software Engineering Lab. His research interests include the area of a monitoring system for self-healing, UML modeling and autonomic computing.

Shunshan Piao received the B.S. degree in Computer Science and Technology from Dalian Nationalities University, China, in 2004. She is currently working on a master's degree in Electrical and Computer Engineering at Sungkyunkwan University in Korea and working as a member of the Software Engineering Lab. Her primary research interests include the area of autonomic computing, self-healing, probabilistic dependency analysis, and machine learning.

Eunseok Lee received his Ph.D. and M.S. degrees in Department of Information Engineering from Tohoku University, Japan, in 1992 and 1988, respectively, and his B.S. degree in Electronic Engineering from Sungkyunkwan University, Korea, in 1985. He is a Professor of the Department of Computer Engineering of Sungkyunkwan University, Korea. From 1994 to 1995, he was an assistance professor of the Department of Information Engineering of Tohoku University, Japan. He was a research scientist in Information and Electronics Laboratory of Mitsubishi Electric Corporation, Japan, from 1992 to 1994. His areas of research include Software Engineering, Autonomic/Ubiquitous Computing and Agent-oriented Intelligence System.