韓國컴퓨터情報學會 論文誌
第16卷 第9號, 2011. 9.

2011-16-9-1-7

UML모델을 이용한 모니터링 모듈 생성 방법

박 정 민*, 정 옥 란**

An Approach to Generation Monitoring Module using UML Model

Jeongmin Park*, Ok-Ran Jeong**

요 약

자가 치유란 시스템에 정의된 제약사항들을 평가하고 위배 시에 적절한 전략을 적용하는 방법론이다. 오늘날 복잡해져가는 컴퓨팅 환경에서 자가 치유를 위해 시스템에 발생한 문제를 스스로 인식하는 능력을 부여하는 연구가 중요한 이슈가 되고 있다. 그러나 대부분의 기존연구들은 목표시스템의 자가 치유를 위해 자가 치유 개발자들이 제약조건을 모델링하고 분석해야 하는 노력이 크다. 따라서 본 논문에서는 이러한 문제를 해결하기 위해서 UML 모델을 이용한 '모니터링 모듈 생성 방법'을 제안한다. 제안 방법론은 1) UML 모델로부터 자가 치유를 위해 요구되는 시스템 지식(system knowledge)을 정의, 2) 모니터를 생성하는 프로세스 그리고 생성된 모니터링 모듈을 이용한 문제 모니터링 프로세스를 나타낸다. 제안 사항들을 통해, 자가 치유 개발자가 가진 목표시스템에 대한 분석의 노력을 최소화하고, 시스템 내부 정보를 기반을 둔 모니터링의 범위 확보, 모니터링 환경의 자동 구축을 위한 시스템 구축 부하를 최소화하는 것이 가능하다. 본 논문에서는 평가를 위해서 ATM 프로토타입 시스템에 제안 방법론을 적용한 정성적 평가와, 자가 치유의 수행 능력을 평가하기 위해 이전 연구에서 수행한 비디오 회의 시스템을 통해 정량적 평가를 수행한다.

▸ Keyword : 자율 컴퓨팅, UML모델, 모니터링 모듈

Abstract

Self-healing is an approach to evaluating constraints defined in target system and to applying an appropriate strategy when violating the constrains. Today, the computing environment is very complex, so researches that endow a system with the self-healing's ability that recognizes problem

• 제1저자 : 박정민 • 교신저자 : 정옥란
• 투고일 : 2011. 08. 30, 심사일 : 2011. 09. 08, 게재확정일 : 2011. 09. 19.
* 동양미래대학교 소프트웨어정보과(Dept. of Software Engineering, Dongyang Mirae University)
* 경원대학교 전자컴퓨터공학부(School of Electronic and Computer Science, Kyungwon University)
※ 이 논문은 2011년도 경원대학교 교내연구비 지원에 의한 결과이고, 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업지원을 받아 수행된 것임(2011-0014747).

www.kci.go.kr

arising in a target system are being an important issues. However, most of the existing researches are that self-healing developers need much effort and time to analyze and model constraints. Thus, in order to improve these problems, this paper proposes the method that automatically generates monitoring module by using UML models for self-healing. The approach proposes: 1) defining system knowledge required for self-healing from UML model, 2) process for generating monitor, by using monitor generated, and process for monitoring the problems. Through these, we can reduce the efforts of self-healing developers to analyze target system, and secure monitoring scope based on information of system knowledge. Also we can minimize the efforts to develop the monitoring environment automatically. to evaluate the proposed approach, we apply proposed approach to ATM prototype system for qualitative result, and perform quantitative evaluation through video conference system in our existing research.

▸ Keyword : Autonomic computing, UML Model, Monitor Module

I. 서 론

오늘날 복잡해가는 컴퓨터 환경에서 인간이 시스템을 유지하고 관리하는 것은 매우 중요하고 어려운 작업이다. 특히, 시스템에서 발생한 문제를 인식하고, 감지된 문제를 해결 하는 데는 많은 시간과 노력이 요구된다[1,2]. 이러한 문제를 해결하기 위한 방법론으로 자가치유(self-healing)는 현재 주목 받고 있는 연구 분야이며, 시스템 관리의 한계를 극복하기 위해 시스템 스스로 문제를 인식하고 이를 치유하는 자가 치유 방법이 현재 활발하게 연구 되고 있다[1][3].

자가 치유 시스템을 구현하기 위해서는 아래의 이슈들이 해결되어야 한다[1]:

1) 모니터링(Monitoring): 실행 중인 소프트웨어의 상태를 모니터링 한다. 모니터링 대상은 소프트웨어의 실행 환경도 포함될 수 있다.
2) 평가(Evaluation): 소프트웨어에게 발생된 문제를 분석하고 자가 치유의 필요성을 판단한다. 문제의 심각성 정도에 따라 자가 치유의 필요 여부가 결정될 수 있다.
3) 전략 계획 (Planning): 발생된 문제를 해결하기 위한 자가 치유 전략을 찾는다.
4) 자가 치유 실행(Execution): 자가 치유 전략에 따라서 실행 중인 시스템의 구조와 행위를 동적으로 변경한다.

위의 이슈들 중에서 모니터링은 가장 중요하고 어려운 문제이다[2]. 이 연구 분야는 초기 단계이기 때문에 관련 연구 및 적용사례가 많지 않으나 크게 자가 치유 시스템에 대한 기존 연구들은 외부 환경을 위한 모델 기반의 자가 치유[1,3,4,5], 컴포넌트 기반의 자가 치유[6,7,8], 로그 기반의 자가 치유[9]로 구분될 수 있다.

기존 연구들의 공통적인 문제점은 자가치유 개발자가 내부를 알 수 없는 목표 시스템을 직접 분석해야 한다는 점이다. 예를 들어, 목표시스템의 외적 자원 환경 파악을 위해서는 제약조건[6,7,8]을 모델링, 내부 상태 분석을 위해 컴포넌트의 상태 모델[1,3,4,5,10,11]을 분석해야 한다. 이러한 분석의 노력들은 자가치유 개발자가 목표시스템을 이해하는 정도에 따라 목표시스템에 부여되는 자가 치유를 위한 모니터링 능력의 정도가 다르다. 따라서 본 논문에서는 자가치유를 위해 요구되는 추가적 부하들을 감소하고 모니터링 성능을 향상시키기 위한 모니터링 모듈의 프레임워크와 그것을 포함하고 있는 모니터링 모듈 생성 방법론을 제안한다. 제안 방법론은 UML 모델로부터 자가치유를 위해 요구되는 시스템 지식(system knowledge)을 정의, 모니터링 모듈 생성시키는 프로세스, 그리고 생성된 모니터링 모듈을 이용한 문제 모니터링 프로세스를 나타낸다.

제안 사항들을 통해, 자가 치유 개발자가 가진 목표시스템에 대한 분석의 노력을 최소화하고, 시스템 내부 정보를 기반을 둔 모니터링의 범위 확보, 모니터링 환경의 자동 구축을 통한 시스템 구축 부하를 최소화하는 것이 가능하다.

평가를 위해 ATM 프로토타입 시스템에 제안 방법론을 적용한 정성적 평가와, 자가치유의 수행 능력을 평가하기 위해 이전 연구[2]에서 수행한 비디오 회의 시스템을 통해 정량적 평가를 수행한다. 본 논문의 구성은 다음과 같다. 2장에서는 대표적인 관련 연구들을 상세 분석하여 공통 취약성을 기술하며, 3장에서는 제안 모니터링 방법론의 개념과 아키텍처 및 프로세스에 대해서 기술한다. 4장에서는 사례연구와 평가를 수

www.kci.go.kr

행하고, 마지막 5장에서는 결론과 향후 과제를 나타낸다.

II. 관련 연구

최근 자율 컴퓨팅(autonomic computing)의 개념들을 기반으로 애플리케이션에 자가 치유 방법론이 적용되고 있다[12]. 본 장에서는 자가치유에 대한 기존 방법론들에 대한 특징과 취약점을 분석 한다.

1. 외부 환경을 위한 모델 기반의 자가치유

모델 기반의 자가치유 방법론[1,3,5,7]들은 시스템을 모니터링, 해석, 분석, 재구성하기 위해 사용될 수 있는 계층화된 외적환경 관점의 구조적 분석 모델을 표현하였다. 이 방법론들은ADL(Architecture Description Language)을 기반으로 목표 시스템의 외부화 관점에서 최적화된 자원 환경이나 프로세스를 모델링 한 것이다. 이를 기반으로 목표시스템의 수정을 일으키지 않고 , 목표시스템의 내부정보 없이 외부 자원 환경 관점의 자가치유 애플리케이션의 개발을 용이하게 한다. 그러나 자가 치유 개발자가 목표시스템을 모델링하기 위해서 내부 정보 없이 외부 자원 환경에 대한 분석의 부하가 매우 높다.

2. 컴포넌트 기반의 자가치유

컴포넌트 기반의 자가치유 방법론[6,7,8]들은 소프트웨어 아키텍쳐의 강건성을 위해 컴포넌트 내부에 서비스 계층과 치유 계층을 각각 개별적으로 설계한다. 이를 통해 컴포넌트의 강건성을 보장한다. 이 연구들은 서비스 계층 내부에 있는 태스크들의 행동을 구체화한 상태모델을 통하여 컴포넌트 기반의 동적 재구성을 용이하게 한다, 재구성 후 상태 모델을 기반으로 컴포넌트가 정상 동작이 가능한지 아닌지의 여부를 판단할 수 있는 기초를 제공한다. 그러나 이들은 내부 상태에 대한 이벤트에 관해 초점을 두고 있기 때문에 컴포넌트 구동환경인 자원 부족에 관한 문제 결정에 어려움이 있어 이로 인해 발생하는 고장(failure)을 치유할 수 없다.

3. 로그 기반의 자가치유

로그기반의 자가치유 방법론[9]은 다양한 타입의 시스템들(웹서버, 데이터베이스 관리 시스템 등)에서 생성된 이벤트 로그들을 자동 수집하여 자가치유 가능하게 하는 시스템을 제안하였다. 시스템 내부에 생성된 로그 이벤트를 어댑터(adapter)가 공통로그포맷 (common base event) 으로 변환하여 시스템의 내부 상태에 관한 에러를 해석한다. 생성된 공통로그포맷은 자율관리자(autonomic manager)에 의해서 분석되어 증상 서비스(symptom service)와 정책적용엔진(policy engine)을 통해 고장에 대한 자가치유 전략을 수행하는 특징이 있다. 이 방법론은 목표시스템의 내부 상태 관점에 관한 에러 로그 발생 시에 관련 문제를 찾아 사후 관리를 위한 개선점을 제공하는 것이 용이하다. 그러나 로그가 발생하지 않는 경우, 시스템의 내부 관점을 모니터링하고 분석하고 치유 하기 위한 제약조건을 자가치유 개발자가 직접 모델링 해야 한다. 또한 로그의 내용만으로는 하나의 컴포넌트로 인해 발생하는 문제가 다른 컴포넌트에게도 영향을 미치는지에 관한 연관성 분석이 쉽지 않다.

4. 모니터의 자동 생성

모니터링 연구 분야에서는 'formal'하게 기술된 시스템의 정보를 기반으로 모니터를 생성하는 연구[4,13]가 활발히 진행되어 왔왔다. 시스템의 정보는 룰, 제약 사항, 목표 그래프, 결함 트리 등으로 다양하게 사용되고 있다. 하지만 이러한 모니터링 연구들에서 사용되는 시스템의 정보 또한 기존의 자가치유 방법론들과 유사하게 미리 수동 분석되어야 한다, 시스템의 정보 모델링 또한 'formal method'를 사용하기 때문에 지식의 명세화가 어렵다.

5. 공통적인 문제점

위에 언급된 기존 연구들의 공통적인 문제점은 자가치유를 위한 시스템 모델링의 부하와 추가적으로 요구되는 작업량, 치유를 위한 시스템 내부 정보의 정확성, 그리고 모니터링 환경의 구축을 위한 비용이 많이 요구 된다. 결과적으로 본 논문에서는 기존연구들의 공통 취약성들을 부분적으로 해결하기 위해서 자가 치유 시스템을 위한 모니터를 UML로부터 시스템 지식을 추출하여 자동으로 생성하는 것을 목표로 한다. 다음 장에서 이에 대한 상세한 설명을 기술한다.

III. 모니터링 모듈 프레임워크

본 장에서는 UML로부터 '시스템 지식'을 추출하여 자가치유 시스템을 위한 모니터(monitor)를 자동 생성하는 방법을 설명한다.

1. 시스템 지식

시스템의 모니터링을 위해서는 모니터링 대상의 식별이 요구된다. 컴포넌트 기반 시스템에서 컴포넌트들은 여러 개의

www.kci.go.kr

클래스로 구성되며 각 컴포넌트마다 상태 정보를 가지고 있다. 컴포넌트의 상태는 보통 설계 시에 컴포넌트의 동적 환경에 대한 정보를 통해 구현 및 설계에 대한 이해를 돕기 위해서 기술된다. 따라서 상태 정보는 시스템 모니터링 시의 컴포넌트 내부 동작을 분석할 때 설계자의 의도를 이해할 수 있도록 한다.

본 방법론에서는 함수 수준의 모니터링을 제공하기 때문에 클래스에 대한 정보(속성 및 함수)가 필요하다.

시스템 지식(system knowledge)은 시스템을 구성하고 있는 컴포넌트들에 대한 정보를 정의 한다. 이것은 소스 코드 수준의 모니터링을 가능하게 하기 위해 요구되며 컴포넌트를 구성하고 있는 클래스들과 상태 정보를 정의 한다. 또한 정확한 의도의 컴포넌트의 상태뿐만 아니라 실제 구현된 클래스에 대한 정보도 제공한다. 이를 기반으로 시스템 정보를 추출하는 'probe'가 삽입될 지점을 식별하고, 시스템 설계 시 목적과 부합되는 모니터링 정보를 생성한다. 시스템 지식, SK는 컴포넌트들의 집합 Comset 과 컴포넌트들 간의 관계 Rset 을 정의한다.

$$SK = \{Com_{set}, R_{set}\}$$

Comset 과 Rset의 정보를 통해 SK는 시스템을 구성하고 있는 컴포넌트들의 구성 및 이름에 대한 정보뿐만 아니라, 컴포넌트 간의 관계에 대한 정보를 포함한다. 이를 통해 특정 컴포넌트에 문제 발생 시 전체 시스템에 미치는 영향에 대한 정보를 제공할 수 있다. Comset 을 구성하는 Rset컴포넌트 는 컴포넌트를 구성하는 클래스들의 정보 ClassInfo와 컴포넌트의 상태 정보 StateInfo을 포함한다. 클래스 정보 ClassInfo는 클래스 집합 Clsset과 클래스간의 관계 집합 ClsRelset 로 구성된다. 클래스 Cls는 클래스 이름 classname, 속성집합 Atr과 행동집합 Oper로 구성된다. 여기서 클래스의 이름, 속성, 함수의 이름은 구현된 코드를 분석하여 모니터링 대상을 식별할 때 요구되며 클래스의 생성 및 속성의 변경, 함수의 수행과 같은 세부적이니 수준의 모니터링을 가능하게 한다.

Comi= {ClassInfo, StateInfo}

ClassInfo = {Clsset, ClaRelset}

Cls = {classname, Atrset, Operset}

상태 정보 StateInfo는 상태집합 Sset 와 상태 전이 Tset 로 구성되며 다음과 같이 표현된다. 상태 집합 Sset 을 구성하는 상태 S는 상태 이름 statename, 상태로의 진입 전이Tin , 상태에서의 탈출 전이 Tout 로 구성된다. 상태 전이 집합 Tset을 구성하는 상태 전이T는 전이 이름 transname , 전이의 시작 상태 statesour, 전이의 목적 상태 statedest로 구성된다. 이러한 상태 정보는 시스템 이상이 발생했을 때 특정 컴포넌트의 상태에 대한 정보를 제공하고, M.E. Shin의 연구[7,8]에서와 같이 모니터링 룰 추출에도 사용된다. 여기서 상태 전이 T는 시스템 내부의 함수라고 가정한다.

StateInfo = {Sset, Tset}

S ← {statename, Tin, Tout}

T ← {transname, statesour, state dest}

결과적으로 모니터는 이러한 정보들을 포함한 시스템 지식을 기반으로 하기 때문에 소스 코드 수준의 시스템 내부 모니터링이 가능하다.

2. 모니터링 모듈의 프레임워크

프레임워크의 전체적인 구조는 그림 1과 같다.

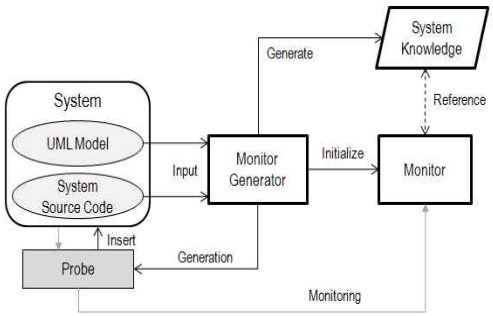

그림 1. 모니터링 모듈의 프레임워크
Fig. 1. Framework of Monitoring Module

(1) 모니터 생성기

'모니터 생성기(Monitor Generator)'는 UML 모델로부터 생성된 '시스템의 지식'을 저장하고, 시스템에 종속적인 'probe'를 삽입한다. 삽입된 'probe'들은 모니터에 등록하여 템플릿 모니터로 초기화한다. 이러한 구조는 모니터가 특정 컴포넌트에 종속되지 않기 때문에 각 컴포넌트마다 모니터가 추가적으로 설계될 필요를 없게 한다. 세부적인 모니터 생성기의 구조는 그림 2와 같다.

- Input Analyzer: UML모델을 입력받아 모니터링 지식의 추출을 위해 각각의 다이어그램을 분석한다.
- System Knowledge Generator: 'Input Analyzer'에서 분석한 정보를 기반으로 '시스템 지식'을 추출한다. 컴포넌트 다이어그램, 클래스 다이어그램, 상태 다이어그램의

www.kci.go.kr

정보를 분석하여 시스템을 구성하고 있는 각 컴포넌트의 정보를 '시스템 지식'으로 형식화한다.

- Template Monitor Initializer: 생성된 모니터링 지식을 기반으로 템플릿 모니터를 생성시킨다.
- Probe Generator: 생성된 모니터링 지식을 기반으로 모니터링 해야 하는 컴포넌트에 'probe'를 삽입한다.

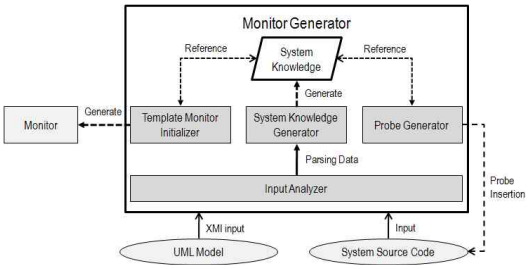

그림 2. 모니터 생성기의 구조
Fig.2. Architecture of Monitor Generator

'Monitor Generator'를 위한 모니터 생성 프로세스는 그림 3과 같다.

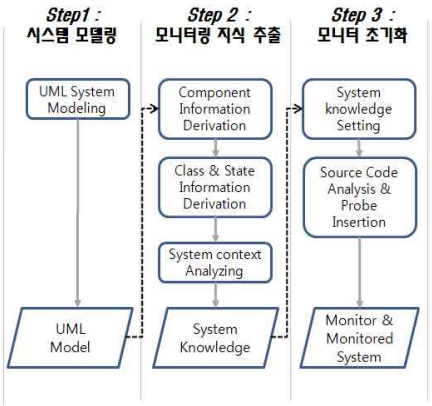

그림 3. 모니터 생성 프로세스
Fig.3. Monitor Generation Process

- Step1: 시스템 모델링 단계

자가 치유의 대상이 되는 목표 시스템에 대한 시스템 모델링을 통해, 목표 시스템의 산출물들, UML[14] 설계모델의 컴포넌트 다이어그램, 클래스 다이어그램, 상태 다이어그램으로 구성된다. 이들 다이어그램들은 그림 4처럼 XMI(XML Meta-Interchange, UML 설계모델의 출력물)정보를 자동 생성한다[15]. XMI 정보는 UML 도구에서 자동으로 생성되는 정보이다.

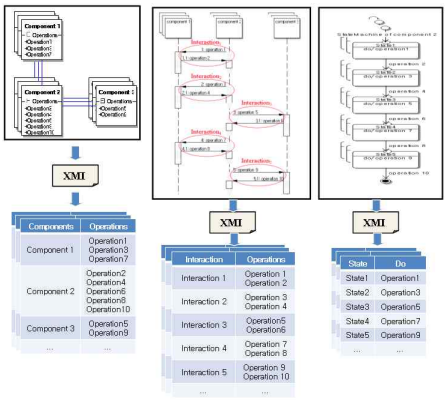

그림 4. UML 모델에서 XMI 추출
Fig.4. XMI Extraction from UML Model

- Step2: 모니터링을 위한 지식 추출

'Input Analyzer'는 XMI를 분석하여 각 다이어그램 별로 구분하고 사용하지 않는 정보를 제거하고 'System Knowledge Generator'에 전달한다. 이후 'System Knowledge Generator'는 컴포넌트 다이어그램에서의 컴포넌트 개체 정보를 추출하여 시스템의 컴포넌트들 Comset과 각 컴포넌트들간의 관계들 Rset 을 추출한다. 그리고 각 컴포넌트들과 연관된 클래스 다이어그램을 기반으로 각 컴포넌트 Com의 클래스 정보 ClassInfo와 상태 정보 StateInfo를 추출하여 최종적으로 시스템 지식 SK를 생성한다. 그림 5는 Step2의 처리결과를 보여준다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SystemKnowledge SYSTEM "G:₩SystemKnowledge.dtd">
<SystemKnowledge>
    <ComponentInfo>
        <Component id="component1" name="ATM" classinfo="classinfo1" stateinfo="stateinfo1">
    </ComponentInfo>
    <ClassInfo id="classinfo1">
        <Class id="class1" name="class1" isActive="true">
            <Attr type="int" id="attr1" name="attr1"/>
            <Method id="method1" returnType="int" name="method1">
                <Parameter type="int" id="param1" name="param1"/>
            </Method>
        </Class>
        .....
    </ClassInfo>
    <StateInfo id="stateinfo1">
        <State id="state1" type="init" name="state1" outgoing="event1"/>
        .....
        <Transition id="event1" name="test5" source="state1" target="state2"/>
        .....
    </StateInfo>
    <RelationInfo>
        <Relation id="relation1" type="association" source="component1" target="component2"/>
    </RelationInfo>
</SystemKnowledge>
```

그림 5. 시스템 지식
Fig.5. System Knowledge

www.kci.go.kr

- Step3: 모니터 초기화 (모니터 생성)

'Template Monitor Initializer'은 Step2에서 추출된 모니터링 지식을 지식 데이터베이스에 저장한다. 저장된 모니터링 지식은 모니터링 프로세스가 수행되는 동안 문제의 감지 및 시스템의 상황 분석을 위해서 이용된다. 다음으로 생성된 지식을 기반으로 목표 시스템으로부터 정보를 추출하기 위한 'probe'를 삽입한다. 'probe'는 각 컴포넌트에서의 전이를 기반으로 모니터링 하기 위해 각 컴포넌트의 클래스 내의 함수마다 삽입된다. 우리는 목표시스템에 'probe' 삽입을 하기 위해 T.Zenmyo의 연구[16]에서와 같이 AOP(Aspect Oriented Programming)개념을 도입하여 모니터링을 수행하기 위한 컴포넌트, 클래스, 메소드, 속성, 이벤트 등과 같은 'aspect'를 추출한다. 현재 구현에서는 컴포넌트 내부의 모든 클래스의 메소드들의 실행을 모니터링 한다. 따라서 메소드들은 모두 'pointcut'으로 정의되어, 모든 함수는 'around' 정책으로 묶이게 되고 실행 전과 실행 후에 'probe'를 통해서 계속 모니터링 된다. 생성된 'probe'들은 모니터에 등록된다. 이와 같이 '모니터 생성 프로세스'를 통해 목표 시스템의 수정 없이 목표 시스템 코드에 'probe' 삽입이 가능하고, 생성된 모니터는 목표 시스템에 종속적인 구조를 가지지 않기 때문에, 기존 연구의 문제점인 목표 시스템을 분석하기 위한 부하를 감소 시킬 수 있다.

(2) 모니터

모니터(Monitor)는 시스템의 모니터링을 담당하는 것으로 목표 시스템에 대한 지식 및 시스템의 이상 감지를 위한 룰 베이스, 정책 등을 포함하고 있다. 이것은 시스템 지식 데이터베이스, 목표 시스템의 정보를 모니터에게 전달하는 'probe'로 구성된다(그림 6 참조). 모니터는 다음과 같은 모듈로 구성된다.

- Event Capturer: 시스템에 삽입된 'probe'로부터 이벤트를 수집한다.
- Anomaly Detector: 모니터링된 정보를 기반으로 시스템의 문제 발생을 감지한다. 시스템의 문제 감지는 'system knowledge Database'에 저장된 정책 및 룰을 기반으로 한다.
- System Context Analyzing Engine: 이상 감지기에 의해서 이상이 감지되면 시스템의 상황 분석을 시작한다.
- Emergent Anomaly Handler: 'System Context Analyzing Engine' 에서 추출된 시스템 상황 정보를 기반으로 필요로 되는 긴급 행동을 수행한다.
- Monitor Controller: 위에 언급된 컴포넌트들을 관리하며, 모니터의 정보를 관리한다.
- System Knowledge DB: 시스템 지식 데이터베이스는 모니터의 구조와 독립적으로 존재하기 때문에 여러 개의 시스템을 모니터링할 수 있도록 함으로써 유연성과 확장성을 제공한다. 'System Knowledge DB'는 시스템을 구성하고 있는 컴포넌트, 각 컴포넌트의 상태 및 내부 정보를 포함하고 있다.

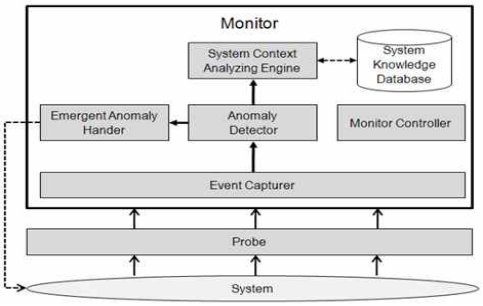

그림 6. 모니터의 구조
Fig.6. Architecture of Monitor

요구되는 목표 시스템의 정보 수집은 'probe'를 통해서 수행되고, 'probe'는 컴포넌트의 상태 및 행동, 속성 값과 같은 컴포넌트의 내부 정보를 'Monitor'에 전달한다. 'Monitor'가 수행하는 모니터링 프로세스는 그림 7과 같다.

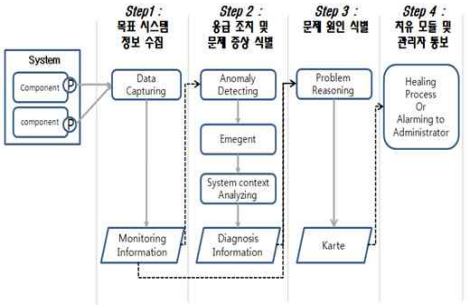

그림 7. 모니터링 프로세스
Fig.7. Monitoring Process

(1) Step1: 목표 시스템 정보 수집

목표 시스템에 삽입된 'probe'는 각 컴포넌트에서 정의된 행동이 수행될 때마다 해당 컴포넌트의 정보와 시스템의 상황 정보를 모니터에게 전달한다. 2.2절 step3에서 언급한 '포인트컷'에서 동작이 수행될 때마다 모니터에게 정보가 보내지며 전달 정보는 다음과 같다(표1, 표2 참조

표1. 'probe'로부터 전달되는 정보
Table 1. Information delivered from 'probe'

정보	모니터링 정보
probe 식별자	probe의 고유 식별자
컴포넌트 정보	컴포넌트이름:클래스:이벤트종류:이벤트정보

www.kci.go.kr

표2. 'probe' 식별 테이블
Table 2. 'probe' Identification Table

식별자	해당 컴포넌트
probe_id1	component1
probe_id2	component2

모니터는 'probe'식별자를 이용하여 'probe' 식별 테이블을 통해 각 'probe'가 담당한 컴포넌트를 식별한다. 'probe'로부터 온 모니터링 정보에는 컴포넌트의 속성, 행동, 상태 등에 대한 정보가 포함되어 있다.

(2) Step2: 응급 조치 및 문제 증상 식별

'probe'로부터 전달되는 컴포넌트의 정보는 'Anomaly Detector' 에 의해서 검사된다. 'Anomaly Detector'는 사전에 설계자에 의해서 정의된 정책과 룰을 이용하여 문제의 증상을 식별한다. 'Anomaly Detector'가 시스템의 이상을 인식하면 'System Context Analyzing Engine' 에게 관련 모니터링 이벤트의 정보를 전달하여 현재 시스템의 상황 분석을 요청한다. 'System Context Analyzing Engine' 은 모니터링 이벤트 정보와 모니터링 지식을 기반으로 문제가 발생한 컴포넌트, 컴포넌트의 행동 및 상태, 관련 컴포넌트 등의 정보를 분석하여 표3과 같은 진단 정보를 생성한다.

표3. 진단 정보
Table 4. Diagnosis Information

진단 정보	내용
컴포넌트 분석	문제가 발생한 컴포넌트 식별
클래스 분석	컴포넌트 내에 문제 발생한 클래스 식별
행동분석	클래스 내에 발생한 행동 식별
상태분석	문제 발생 당시의 상태 식별
연관 분석	문제 컴포넌트와 연관된 컴포넌트 분석

진단 정보가 생성되면 'Monitor Controller'는 문제의 원인을 분석하기 전에, 먼저 'Emergent Anomaly Handler'에게 시스템 상황 분석 정보를 기반으로 한 응급 조치를 명령한다. 예를 들어, 재시작, 대체전략과 같은 명령이 여기에 속한다. 문제 컴포넌트가 정상적으로 동작하지 않는데 사용자들이 계속해서 서비스를 사용할 경우 큰 경제적, 사회적 손실을 초래할 수 있다. 예를 들어, 금융 시스템에서 입출금과 같은 현금서비스를 담당하는 컴포넌트의 오류가 발생했다고 가정하자. 이러한 상태에서의 사용자들의 서비스 이용은 사용자 및 은행에게 큰 손실을 초래할 수 있다. 이러한 응급 조치는 정책에 따라서 다양하게 사용될 수 있다. 예를 들어, 문제 컴포넌트의 사용 불가를 사용자 및 관리자, 관련 컴포넌트들에게 알리거나, 특정 응급 조치 루틴을 삽입하여 문제 발생 시 모니터가 이를 호출할 수 있다.

(3) Step3: 문제의 원인 식별

'Emergent Anomaly Handler'에 의해서 응급조치가 완료되면 'System Context Analyzing Engine' 이 문제가 발생한 컴포넌트의 위치 및 상태 정보, 시스템의 환경 정보를 기반으로 현재 발생한 문제의 원인을 분석한다. 'probe'로부터 전달된 컴포넌트의 모니터링 정보와 System Context Analyzing Engine' 에서 생성한 시스템 상황 정보를 기반으로 시스템 지식 데이터베이스의 증상을 참조하여 문제의 원인들을 식별하고 의심되는 원인들로 구성된 카르테(karte)를 생성한다(표4 참조).

표4. 카르테의 구성
Table 4. Composition of Karte

구성	세부 항목
진단 정보	문제가 발생한 컴포넌트
	문제가 발생한 클래스 및 함수의 위치
	문제가 발생한 상태
	관련된 컴포넌트
분석 정보	히스토리 정보
	의심되는 원인 리스트
	해결 방법
	위험도

(4) Step4: 치유 모듈 및 관리자에게 통보

문제의 분석이 끝나면 분석된 정보를 자가치유 모듈이나 관리자에게 알린다. 치유 모듈 및 관리자는 문제가 발생한 상황 및 의심되는 원인들에 대한 정보를 받음으로써 문제의 분석과 해결을 수행한다.

IV. 사례 연구 및 평가

본 장에서는 제안 방법 및 제안 프로세스의 유효성을 확인하기 위하여 구현한 간단한 ATM 시스템에 대해서 설명한다. 구현 환경은 윈도우즈 XP 환경에서 JDK 1.6, Eclispe 3.4플랫폼에서 AspectJ[17] 플러그인을 이용하여 개발하였다. 우리가 구현한 프로토타입은 간단하게 서버와 ATM클라이언트로 구성되고, 서로 다른 호스트에 존재하며 각각 하나의 컴포넌트이다. 비록 규모는 간단하지만 이를 통해 앞장에서 논했던 각 내용을 확인할 수 있다.

www.kci.go.kr

1. 모니터 초기화(생성) 프로세스

(1) Step1: 시스템 모델링 단계

설계자는 전체 시스템의 컴포넌트 다이어그램을 통해서 그림 8과 같이 ATM과 서버 컴포넌트를 설계한다.

그림 8. 목표 시스템의 컴포넌트
Fig.8. Components of Target System

ATM과 서버 간에는 연관관계가 성립하므로 관련 컴포넌트로서 인식되고, 그림 9처럼 각각의 ATM 클래스 다이어그램과 서버 클래스 다이어그램을 통해 각 컴포넌트를 구성하는 클래스 내부의 속성과 행동들에 대한 정보를 그림 10처럼 설계한다.

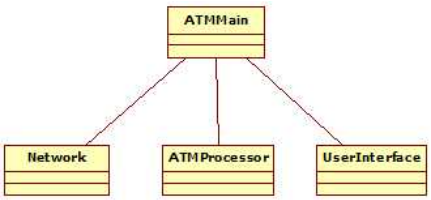

그림 9. ATM 컴포넌트의 클래스 다이어그램
Fig.9. Class Diagram of ATM Component

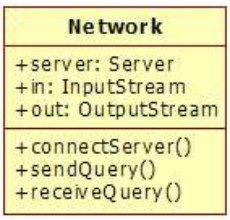

그림 10. ATM 컴포넌트의 Network 클래스
Fig.10. Network Class of ATM Component

클래스다이어그램 설계가 완료되면 각각에 대한 상태 다이어그램을 그림 11처럼 설계한다. 상태 다이어그램에 포함된 전이는 클래스 다이어그램에 기술된 함수 중에 하나이거나 자동 전이이다. UML 시스템 모델이 기술되면 XMI로 변환하여 'Monitor Generator'에 입력된다.

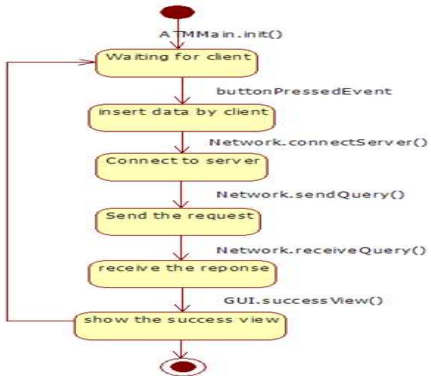

그림 11. ATMMain의 Network 관련 상태 다이어그램
Fig.11. State Diagram of ATMMain Network

(2) Step2: 모니터링을 위한 지식 추출

XMI 파일이 입력되면 'Input Analyer'가 이를 분석하고 'System Knowledge Generator'는 시스템을 구성하고 있는 컴포넌트들인 ATM과 Server 컴포넌트를 인식하여 Comset을 생성한다.

$$Com_{set} = \{ATM, Server\}$$

그리고 각각의 컴포넌트를 모델링한 클래스 다이어그램을 통해 각각의 클래스 정보를 추출한다. 이를 통해서 본 예제에서 추출된 정보는 다음과 같다.

$$ATMClassInfo = \{Cls_{set}, ClsRel_{set}\}$$
$$Cls_{set} = \begin{Bmatrix} ATMMain, Network, ATMProcessor, \\ UserInterface \end{Bmatrix}$$
$$ClsRel_{set} = \{Rel_1, Rel_2, Rel_3\}$$

Rel은 클래스 간의 관계를 나타내며, Rel1은 Network와 ATMMain의 관계로서 다음과 같다.

$$Rel_1 \leftarrow (Association, Network, ATMMain)$$

ATM컴포넌트의 클래스 ATMMain에 대한 정보는 다음과 같이 추출된다.

$$ATMMain = \{Atr_{set}, Oper_{set}\}$$
$$Atr_{set} = \{server, in, out\}$$
$$Oper_{set} = \{connectServer, sendQuery, receiveQuery\}$$

마지막으로 각 컴포넌트의 상태 정보를 추출한다. 위의 그림11에서 보여지는 상태 정보는 설명의 편의를 위해 네트워크 클래스와 연관된 부분만을 간략히 표현한 것이다. 각각의 함수들은 실제 클래스 내부의 함수의 이름과 같아야 한다. 각각

www.kci.go.kr

의 상태와 전이에 대한 정보는 임의의 식별자를 가진다. 이 상태 다이어그램을 통해서 추출되는 상태 정보는 다음과 같다.

$$StateInfo = \{State_{set}, Trans_{set}\}$$
$$State_{set} = \{Int, S_1, S_2, S_3, S_4, S_5, S_6, End\}$$
$$Trans_{set} = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$$

여기서는 상태 S4 와 전이 T4만 살펴본다. 상태 S4와 전이 T4 는 다음과 같이 표현된다.

$$S_4 \leftarrow (Send\,the\,request, T_4, T_5)$$

$$T_4 \leftarrow (Network.sendQuery, S_3, S_4)$$

이와 같은 프로세스를 통해 목표 시스템 내부의 컴포넌트들에 대한 모니터링 지식이 생성된다.

⑶ Step3: 모니터 초기화 (모니터 생성)

모니터 초기화 모듈은 생성된 시스템 지식을 시스템 지식 데이터베이스에 저장한다. 그리고 'probe'를 삽입하기 위해 모니터링 지식을 기반으로 'aspect'를 생성한다. 'aspect'의 'pointcut'은 각 클래스의 모든 함수이다. 모든 함수가 호출되기 전과 후에 우리가 구성한 프루브의 모니터링 이벤트 함수를 호출함으로써 모니터에게 시스템의 정보를 제공한다. Network 클래스를 위해서 생성된 'aspect'는 그림 12와 같다.

```
public aspect NetworkAspect {
    private Probe probe = new Probe();

    pointcut networkExecs() : within(Network) && execution(* *(..));

    Object around(): networkExecs() {
        probe.printMsg(thisJoinPoint);
        Object result = proceed();
        probe.printResult(result);
        return result;
    }
}
```

그림 12. Network 클래스를 위해 생성된 'aspect'
Fig.12. Generated 'aspect' for Network Class

2. 모니터링 프로세스

⑴ Step1: 목표 시스템 정보 수집

컴포넌트 내부의 함수가 수행될 때마다 'probe'는 표 6과 같이 모니터링 데이터를 모니터에게 전송한다. 즉, ATM 컴포넌트 내부의 함수가 수행될 때마다 모니터로 전송되는 메시지들이 표5와 같이 구성된다.

표5. 모니터로 전송되는 메시지 구성
Table 5. Message Component transmitted to Monitor

정보	모니터링 범위
probe식별 아이디	프루브 고유의 식별 아이디
수행된 함수 정보	수행 함수의 이름, 인자, 리턴 값
클래스 정보	클래스 이름
문제 발생 코드 위치	문제가 발생한 소스 코드 라인

⑵ Step2: 응급 조치 및 문제 증상 식별

ATM으로부터 수집되는 메시지는 'Anomaly Detector'의해서 계속해서 검사된다. 우리는 다음과 같은 상황을 설정하였다. 네트워크 클래스가 'send the request' 상태인 상황에서 갑자기 GUI.successView()가 호출된다(그림11참조). 'Anomaly Detector'는 이전 상태로 설정하고 다음의 Network.receiveQuery()함수의 호출을 기다리지만, GUI.successView()가 호출되었음을 알리는 정보를 받는다. 이는 상태 다이어그램에 정의되어 있는 상태의 시퀀스를 만족하지 않기 때문에 'Anomaly Detector'가 시스템의 이상으로 인식한다. 'Anomaly Detector'가 시스템의 이상을 인식하면 'Emergent Anomaly Handler' 를 통해서 사용자에게 현재 서비스 이용의 불가를 알리는 창을 띄운다. 'System Context Analyzing Engine'은 모니터링 이벤트 정보와 시스템 지식을 기반으로 표6과 같은 정보를 파악한다.

표6. 시스템 상황 정보
Table 6. Table Situation Context

정보	모니터링 범위
문제 발생 컴포넌트	ATM
관련 컴포넌트	Server
문제 발생 클래스	ATM의 Network, ATM의 GUI
문제 발생 시 수행된 함수	이전:Network.sendQuery() 이후: GUI.successView()
문제 발생 상황	S4(Send the request)에서 S6(show the success view)로 전이 중 발생

⑶ Step3: 문제의 원인 식별

응급조치가 완료되고 시스템 상황 정보가 생성되면, 문제 원인 분석 엔진이 생성된 시스템 상황 정보를 바탕으로 시스템 지식 데이터베이스에 있는 증상들과 비교하여 예상되는 원인들을 탐색한다. 현재 구현에서는 문제 발생 클래스를 기준으로 관련된 함수 및 발생 상황을 기준으로 증상이 기술되어 있다. 이를 기반으로 문제 발생 컴포넌트, 문제 발생 가능 클래스, 문제 발생 시의 수행된 함수, 문제 발생 상황과 관련된 모든 증상에 대한 정보가 원인으로써 추출된다.

www.kci.go.kr

⑷ Step4: 치유 모듈 및 관리자에게 통보

추출된 원인 및 시스템의 상황 정보들은 치유 모듈 혹은 관리자에게 제공된다. 본 논문에서는 모니터링 정보에 초점을 맞추고 있기 때문에 모니터링 이외의 자가치유 프로세스인 분석, 결정, 적용에 대해서는 자세히 다루지 않는다.

3. 제안 방법론의 평가

본 장에서는 제안 방법론의 평가를 정성적, 정량적으로 분석한다.

3.1 분석 단계에서의 자가 치유를 위한 활동 비교

제안 방법론은 UML 기반의 설계 모델로부터 목표 시스템에 대한 제약조건, 연관성 분석, 모니터링 코드 생성 그리고 문제결정 수준을 추출하는 것을 보다 쉽게 할 수 있도록 지원한다. 연관성 분석은 목표 시스템 내부의 컴포넌트들 간의 연관 관계를 나타내며, 코드생성과 문제결정은 제약조건을 위배하는지에 관한 코드를 통해 목표 시스템이 정상인지 비정상인지를 결정하는 것이다. 표7은 제안 방법론과 기존 연구들의 비교를 나타낸다.

표7. 기존 자가 치유 방법론들과의 비교
Table 7. Comparison of ours and existing approaches

활동	제안 방법론	기존 방법론
자가치유의 범위	UML 모델 기반의 시스템 상태 문제	시스템의 자원 환경 문제 또는 내부 상태 문제
제약조건 설정	UML 설계 모델들의 XMI 정보를 이용한 규칙 자동화	자가치유 개발 전문가에게 의뢰
연관성 분석	시퀀스 모델의 XMI 정보 분석을 통해 시스템 지식을 생성하여 컴포넌트 간의 연관 관계 도출	자가치유 개발 전문가에게 의뢰
모니터링 코드 생성	AOP기법의 'pointcut'을 이용한 모니터링 모듈 생성	자가치유 개발 전문가에게 의뢰

기존 방법론으로 자가치유 활동을 하기 위해서는 자가 치유 전문가의 역할이 소프트웨어 개발자의 역할 보다 매우 높다. 따라서 제약조건 설정, 연관성 분석, 코드 생성, 문제결정 수준 설정을 위해 목표 시스템의 분석과 자가치유 시스템 설계에 많은 시간과 노력을 필요로 한다. 하지만, 제안 방법론을 적용하면 자가치유 전문가, 요구사항 엔지니어, 소프트웨어 개발자 간의 최소한의 협업을 통해서 자가치유를 위한 기반 활동들인 제약조건 설정, 연관성 분석, 코드 생성, 문제결정 수준 설정 등을 자동화할 수 있다. 이를 통해 목표 시스템의 분석과 자가치유 시스템의 설계를 위한 노력을 줄일 수 있다. 또한 제안 방법론을 확장하면 기존 방법론과는 다르게 외부 자원 환경 문제 뿐만 아니라 내부 상태 문제에 대한 자가치유 전략을 적용할 수 있는 기반이 될 수 있다. 예를 들어, 시스템 내의 객체가 정해진 대로 행동하지 않을 때 즉시 이상 있는 객체를 탐지 할 수 있다.

3.2 비디오 회의 시스템을 통한 작업 처리 시간 평가

기존 방법론과 마찬가지로 자가 치유를 위한 모니터링 방법론의 성능은 정량적 평가가 어렵다. 따라서 본 논문에서는 자가 치유 능력 검증을 위해 이전 연구[2]에서 적용되었던 비디오 회의 시스템에 제안 방법론을 적용하였다. 제안 방법론이 자가치유의 능력을 가지는지를 보이기 위해서 동적 분산 적응 프레임워크에서 사용되었던 process_generator를 이용하였다. Process_generator는 dummy process를 생성하는 것으로 서버에 접속한 사용자 수와 리소스 사용량을 기록한 로그를 이용하여, 접속사용자의 증가를 시뮬레이션 하기 위해 개발되었다. 시뮬레이션 결과 비정상적인 경우 표8과 같이 작업처리 시간이 전체적으로 매우 높아졌다는 것을 알 수 있었다.

표8. 자가치유를 적용한 작업 처리 시간 평가
Table 8. Processing Time Evaluation

Operation	정상 상황	비정상 상황	제안 방법론 적용	비교
	프로세싱 타임 (ms)	프로세싱 타임 (ms)	프로세싱 타임 (ms)	차이 (ms)
Connect()	400	1800	510	+110
chatService() addClient()	300	1500	410	+110
sendMessage()	100	320	160	+60
messageDelivery()	200	410	240	+40
videoReceive() wait() videoTransmit() processCreator() transCreater() start()	1800	5330	1910	+110
openSession() getStream() getDatReceive()	1400	4610	1515	+115
Total	4,200 ms	13,970 ms	4,745 ms	+545 ms

Process_generator를 이용하여 서버에 많은 프로세스가 생성되었고, 서버의 접속 사용자에 관한 임계치를 넘었기 때문에 비정상적인 상황에서 작업처리 시간이 많이 지연되었다. 이러한 환경에서 제안방법론은 서버에 많은 프로세스가 생성

www.kci.go.kr

되어 외부 자원 환경이 비정상상태 임을 알게 되어, 비정상 상태를 회복시키기 위해서 대응전략인 서버의 구성환경을 변경하여 정상 상태로의 회복을 가능하게 한다.

본 논문에서는 이러한 시나리오를 통해서 제안 방법론을 적용한 후의 작업처리 시간을 비정상적인 상황과 비교하여 위의 표8과 같이 나타내었다. 비교 결과 정상적인 상황에서의 작업처리시간 보다 전체적으로 +545ms의 작업처리 시간이 더 소요되었다.

하지만 비디오 회의 시스템의 정상 상황 4200ms와 비교하여 볼 때 +545ms(전체 4745ms)는 회의를 진행하는데 영향을 주지 않았다. 따라서 +545ms 시간은 목표 시스템에 따라 차이는 있겠지만 전체적인 성능에는 영향을 미치지 않은 것으로 판단하여 제안 방법론의 효율성을 입증하였다. 중대한 시스템(critical system)은 신뢰성, 보안성, 강건성, 유지보수성 등과 같은 'dependability'를 보장할 때, 'trade-off'가 발생할 수 있다[18]. 즉, 목표시스템에 자가 치유 능력을 추가할 때 성능이 떨어질 수 있다. 그러나 이는 본 논문의 고려 사항은 아니다.

현재의 평가 방법은 제안 방법론 자체를 평가하는 좋은 방법은 아니지만, RAINBOW 프레임워크[1]과 유사하게 제안 방법론을 수행하는 내부 모듈들이 비정상적인 상황에서 정상적인 상황으로 회복을 할 수 있는지를 시뮬레이션 하기 위한 좋은 사례가 된다.

V. 결 론

본 논문에서는 목표 시스템의 UML 설계 모델을 이용하여 목표 시스템 내부의 컴포넌트 모니터를 자동으로 생성함으로서 기존에 요구되었던 시스템 분석 부하와 모니터 생성 부하를 감소하기 위한 방법론을 제안했다. 간단한 ATM 시스템에 대한 사례연구를 통해 제안 방법론의 프로세스를 확인해보고 평가를 통해 시스템 분석 부하와 모니터 생성 부하의 감소를 확인할 수 있었다. 본 연구는 자가 치유 시스템의 자동화를 위한 시작 연구이며 시스템의 신뢰성이 한 차원 높아질 수 있는 자가 치유 기능을 자동으로 생성하는 것을 최종 목표로 한다. 지금까지는 개발 부하로 인해 실제 현장에서 많이 사용되지 못했던 자가 치유 시스템의 연구 방향을 확대하는 것을 기대한다. 향후 연구로는 UML 모델의 OCL을 이용한 제약 사항 기술을 통해 좀 더 다양한 모니터링 룰 및 정책을 자동으로 생성할 수 있도록 함으로써 계속해서 모니터링할 수 있는 범위와 분야를 개선해가는 것이다. 또한 현재의 프레임워크를 실제 현장에 적용할 수 있도록 다양한 각도에서의 평가를 진행할 것이다.

참고문헌

[1] D. Garlan, S.W. Cheng, A.C. Huang, B. Schmer and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure", IEEE Computer, Vol.37, No.10, pp.46-54, October 2004.

[2] J.M. Park, H.S. Youn and E.S. Lee, "An Autonomic Code Generation for Self-Healing", Journal of Information Science and Engineering, Vol.25, No 6, pp.1753-1781, November 2009.

[3] R. sterritt, D.F. Bantz, "Personal Autonomic Computing Reflex Reactions and Self-healing", IEEE Transactions on Systems Man and Cybernetics - PARTC: applications and reviews, Vol.36, No.3, pp, 219-228 May 2007.

[4] D.S. Wile and A. Egyed,"An Externalized Infrastructure for Self-Healing Systems", Procs. of the 4th Working IEEE/IFIP Conference on Software Architecture, pp.285-288, September 2004.

[5] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems", Procs of the 1st Workshop on Self-Healing Systems, pp. 27-32, November 2002.

[6] I.G. Chun, J.M. Kim, H.Y. Lee, W.T. Kim, S.M.Park and E.S.Lee "Faults and Adaptation Policy Modeling Method for Self-adaptive Robots", Procs. of the UCMA 2011, pp.156-164, May 2011.

[7] M.E. Shin, "Self-healing components in robust software architecture for concurrent and distributed systems", Science of Computer Programming, Vol.57, No.1, pp.27-44, July 2005.

[8] M.E. Shin and J.H. An "Self-Reconfiguration in Sel f-Healing Systems," Procs of the 3th IEEE International Workshop on Engineering of Autonomic & Autonomic Systems, p.89-98, March 2006.

[9] Jeffrey, O. Kephart and D.M. Chess, IBM Thomas J. Watson Research Center, "The Vision of Autonomic Computing", IEEE Computer, Vol.36, No.1, pp.41-50. January 2003.

[10] S.Y. Lee, "BPEL Based Service Oriented Business Process Modeling", Journal of the Korean Society of Computer and

www.kci.go.kr

Information, Vol. 15, No.12, pp. 143-150, December 2010.
[11] H.J. Joo, B.H. Hong and B.C. Joeng, "A Study on Web Minng System for Real-Time Monitoring of Opinion Information Based on Web 2.0", ournal of the Korean Society of Computer and Information, Vol. 15, No.1, pp. 149-157, January 2010.
[12] S.W. Cheng, D. Garlan, B.R. Schmer, "Evaluating the effectiveness of the Rainbow self-adaptive system", Procs. of the SEAMS 2009, pp. 132-141, March 2009.
[13] W.N. Robinson, "Monitoring Web service Requirements", Procs. of the 12th IEEE International Conference on Requirements Engineering, pp.65-74, June 2005.
[14] UML Online Document.http://www.omg.org/xml
[15] XMI Online Document.http://www.omg.org/xml.
[16] T. Zenmyo, H. Yoshida, T. Kimura, "A Self-healing technique using reusable component-level operation knowledge", Cluster Computing, Vol.10, Issue.4, pp 117-226, December 2007.
[17] AspectJ, www.eclipse.org/aspectj/
[18] Ian Sommerville, "Software Engineering 8th Edition" Addison Wesley, pp.43-61, 2007.

저 자 소 개

박 정 민

2003 : 한국산업기술대학교 컴퓨터공학과 공학사.
2005 : 성균관대학교 컴퓨터공학과 공학석사.
2009 : 성균관대학교 컴퓨터공학과 공학박사
현 재 : 동양미래대학교 소프트웨어정보과 전임강사
관심분야 : 자율컴퓨팅, 자가치유 소프트웨어
Email : ya23ma@dongyang.ac.kr

정 옥 란

2005 : 이화여자대학교 컴퓨터공학과 공학박사.
2006 : 서울대학교 컴퓨터공학부 박사후연구원
2007 : Univ. of Illinois of Urbana Champaign방문연구원
2009 : 성균관대학교 정보통신공학부연구교수
현 재 : 경원대학교 소프트웨어설계경영학과 조교수
관심분야: 웹마이닝, 정보검색, 소셜컴퓨팅
Email : orjeong@kyungwon.ac.kr

www.kci.go.kr