DOI: 10.3745/KIPSTD.2008.15-D.2.271

자가치유 기법을 기반한 시스템 문제결정 자동화 방법론

박 정 민[†] · 정 진 수[††] · 이 은 석[†††]

요 약

자가치유란 시스템에 정의된 제약사항들을 평가하고 위배 시에 적절한 전략을 적용하는 방법론이다. 오늘날 복잡해져가는 컴퓨팅 환경에서 자가치유를 위해 시스템에 발생한 문제를 스스로 인식하는 능력을 부여하는 연구가 중요한 이슈가 되고 있다. 그러나 대부분의 기존연구들은 목표시스템을 자가치유하기 위해 자가치유 개발자들이 제약조건을 모델링하고 분석해야 하는 노력이 크다. 따라서 본 논문에서는 자가 치유 기법을 기반으로 시스템의 내외부 문제 결정을 자동화하는 방법론을 제안한다. 본 방법론은 1) 목표 시스템의 설계단계에서 생성된 설계모델들로 시스템을 명세화하고, 2) 명세화 된 내용을 기반으로 시스템의 내외부 대한 공통 제약 사항을 자동 생성 한다. 3) 자동 생성된 내부 상태 규칙을 통해 컴포넌트간의 의존관계를 해석하여 4) 생성된 공통 제약사항과 분석된 연관성 모델을 코드로 변환하고 문제결정 수준을 결정한다. 5) 문제결정 수준을 기반으로 시스템의 내외부 상태를 모니터링을 하고, 비정상 상태 발생 시 전략을 적용한다. 이러한 자동화된 제안 방법론의 특징을 통해 자가 치유 개발자의 분석의 부하를 줄이며, 나아가서는 시스템의 외적 환경뿐 아니라 내부 상태 문제에 관한 비정상적인 동작을 신속하게 정상적인 상태로 회복하고, 시스템 다운과 같은 고장 횟수를 줄이는 것이 가능해 진다. 본 논문에서는 평가를 위해 제안 방법론을 비디오 회의 시스템에 적용하고 기존 방법론과의 자가치유를 위한 활동을 비교하여 그 유효성을 확인한다.

키워드 : 자가치유, 문제결정, 외적자원 환경, 소프트웨어 내부 상태

An Automated Approach to Determining System's Problem based on Self-healing

Jeongmin Park[†] · Jinsoo Jung[††] · Eunseok Lee[†††]

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 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, this paper proposes an automated approach to determine problem arising in external and internal system environment. The approach proposes: 1) Specifying the target system through the models created in design phase of target system. 2) Automatically creating constraints for external and internal system environment, by using the specified contents. 3) Deriving a dependency model of a component based on the created internal state rule. 4) Translating the constraints and dependency model into code evaluating behaviors of the target system, and determinating problem level. 5) Monitoring an internal and external status of system based on the level of problem determination, and applying self-healing strategy when detecting abnormal state caused in the target system. Through these, we can reduce the efforts of self-healing developers to analyze target system, and heal rapidly not only abnormal behavior of target system regarding external and internal problem, but also failure such as system break down into normal state. To evaluate the proposed approach, through video conference system, we verify an effectiveness of our approach by comparing proposed approach's self-healing activities with those of the existing approach.

Key Words : Self-Healing, Problem Determination, External Resource Environment, Software Internal State

※ 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 지식경제부의 유비쿼터스 컴퓨팅 및 네트워크 원천 기반 기술개발 사업(08B3-B1-10M), 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업 IITA-2008-(C1090-0801-0046), 교육인적자원부의 2단계 BK21사업의 연구결과로 수행되었음.
† 준 회 원 : 성균관대학교 대학원 컴퓨터공학과 박사과정
†† 준 회 원 : 성균관대학교 대학원 컴퓨터공학과 석사과정
††† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2007년 9월 3일, 심사완료 : 2007년 11월 21일

1. 서 론

자가치유란 시스템의 부적절한 동작을 시스템 스스로 감지하고, 문제에 대한 올바른 행동을 적용하기 위한 방법론을 의미한다[1]. 오늘날 복잡해져가는 컴퓨터 환경에서 인간이 시스템을 유지하고 관리하는 것은 매우 중요하고 어려운 작업이다. 특히, 시스템에서 발생한 문제를 인식하고, 감지된 문제를 해결 하는데는 많은 시간과 노력이 요구된다. 이러한 문제를 해결하기 위한 방법론으로서, 자가치유(self-healing)는 현재 주목받고 있는 연구 분야이다.

자가 치유 시스템에 대한 기존 연구들은 컴포넌트 기반[2,3], 모델기반[4,5], 로그기반 방법론[6,7]으로 구분될 수 있다. 이 연구들의 공통적인 문제점은 자가치유 개발자가 내부를 알 수 없는 목표 시스템을 직접 분석해야 한다는 점이다. 예를 들어, 목표시스템의 외적 자원 환경 파악을 위해서는 제약조건[4,5]을 모델링해야 하고, 내부 상태 환경 파악을 위해서는 컴포넌트의 인터렉션에 대한 상태 모델[2,3]을 분석해야 한다. 이러한 분석의 노력들은 자가치유 개발자가 목표시스템을 이해하는 정도에 따라 목표시스템에 부여되는 자가치유 능력의 정도가 다를 수 있다. 따라서 목표시스템의 설계 단계에서 생성된 산출물들을 이용하여 자가치유 능력을 부여하는 것이 가능하다면 자가치유 개발자의 분석의 부하를 줄이는 것이 가능하다.

본 논문에서는 시스템 설계자와 자가치유 개발자와의 협업을 가정하여 설계 단계에서 생성된 UML(Unified Modeling Language) 모델을 이용한다. UML 모델을 통하여 시스템의 외부 자원 환경과 내부 상태 문제를 파악할 수 있는 소프트웨어의 구조와 그것의 내부에 포함된 자가치유 기법 기반의 문제결정 자동화 방법론을 제안한다. 제안 방법론은 다음과 같은 역할을 수행한다.

- 시스템의 설계단계에서 생성된 설계모델들을 통해 시스템을 명세화 한다.
- 명세화된 내용을 기반으로 시스템의 내외부에 대한 제약사항들을 자동생성 한다.
- 자동 생성된 내부 제약 사항들을 통해 컴포넌트간의 의존관계를 해석한다.
- 생성된 시스템의 내외부 제약사항들과 분석된 의존관계를 통해 자가치유를 위한 모니터링 코드 변환과 문제결정 수준을 생성한다.
- 생성된 모니터링 코드와 문제결정 수준을 기반으로 시스템의 내외부 상태를 모니터링을 하고, 만약 문제 발생한다면, 정의된 재구성 전략(reconfiguration strategy)[9]을 적용한다.

위의 제안된 사항들을 통해, 자가 치유 개발자가 가진 목표시스템에 대한 분석의 노력을 최소화 하여 시스템의 비정상적인 상태가 정상상태로 회복하는 것이 가능하며 전체적으로 안정적인 운영이 가능하다.

본 논문에서는 평가를 위해 제안된 자동화 방법론을 적용한 비디오 회의 시스템을 구현하였다. 평가 방법은 비디오 회의 시스템을 구성하는 서버 또는 클라이언트에서 비정상 상태가 발생할 때 정상 상태로 회복시키는 과정을 실험한다. 실험을 통해서, 제안 방법론과 기존 방법론의 자가치유 활동에 대한 분석을 통해 자가 치유 개발자의 분석의 부하를 최소화 할 수 있다는 것을 확인 한다.

본 논문의 구성은 다음과 같다. 2장에서는 대표적인 관련 연구들을 상세 분석하여 공통 취약성을 기술하며, 3장에서는 취약성들을 해결하기 위한 문제결정 소프트웨어 구조를 제안한다. 4장에서는 문제결정 소프트웨어 구조에 포함된 문제결정 프로세스를 다섯 단계로 소개하고, 5장에서는 제안 방법론을 적용한 비디오 회의 시스템과 평가 결과를 소개하며, 마지막 6장에서는 결론과 향후 과제를 기술하였다.

2. 관련연구

최근 유비쿼터스 컴퓨팅 패러다임의 등장과 함께, 상황인식(context-awareness)이나 조용한 컴퓨팅(calm computing)[10], 오토노믹 컴퓨팅(autonomic computing)[11]의 개념들을 기반으로 애플리케이션에 자가치유 방법론이 적용되고 있다. 적용된 자가치유 방법론의 기본 전략들은 대체 전략이나 재시작과 같은 기본적인 재구성 전략을 가지고 있다. 특히, 외적 자원환경(cpu, ram, bandwidth 등)인식[4,5,12]에 관한 연구들을 통해 비정상적인 상태에서 정상 상태로 회복하게 하는 전략들이 대부분이다. 소수의 연구로서는 목표시스템 내부 상태의 신뢰성 강화를 위해 소프트웨어 설계단계에서 자가치유의 능력을 생성하기 위한 추가 모듈을 설계하는 컴포넌트 기반 자가치유 프레임워크[2,3]가 있다. 본 장에서는 자가치유에 대한 기존 방법론들에 대한 특징과 취약점을 분석한다.

2.1 외부 환경을 위한 모델 기반의 자가치유

모델 기반의 자가치유 방법론[4][5]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 자가치유를 가능하게 하는 기초를 제공하고, 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.

2.2 컴포넌트 기반의 자가치유

컴포넌트 기반의 자가치유 방법론[2,3]들은 소프트웨어 아키텍쳐의 강건성을 위해 컴포넌트 내부에 서비스 계층(service layer)과 치유 계층(healing layer)을 각각 개별적으

로 설계하여 컴포넌트의 강건성(robustness)을 보장한다. 이 연구들은 서비스 계층 내부에 있는 태스크(task)들의 행동(behavior)을 구체화한 상태모델(state model)을 통하여 컴포넌트 기반의 동적 재구성을 용이하게 하고, 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. 따라서 컴포넌트의 구동환경인 외적 자원 환경에 관한 문제결정 지원 방법론이 필요하다.

2.3 로그 기반의 자가치유

로그기반의 자가치유 방법론[6,7]들은 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 해석하고, 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 방법론들은 목표시스템의 내부 상태 관점에 관한 에러 로그 발생 시에 관련 문제를 찾아 사후 관리를 위한 개선점을 제공하는 것이 용이하다. 또한 로그의 내용이 실시간으로 기록되기 때문에 데이터베이스로 구조화하기 쉽고, 상향식 분석 (bottom-up analysis)이 용이하다.

그러나 로그가 발생하지 않는 경우, 시스템의 내부 관점을 모니터링하고 분석하고 치유 하기 위한 제약조건을 자가치유 개발자가 직접 모델링 해야 하고, 로그의 내용만으로는 하나의 컴포넌트로 인해 발생하는 문제가 다른 컴포넌트에게도 영향을 미치는지에 관한 연관성 분석이 쉽지 않다. 이러한 문제점들에 대한 대안은 로그가 발생하지 않아도 시스템의 자가치유를 지원할 수 있는 제약조건 생성과 연관성 분석을 제공하는 방법론이 필요하다.

위에 언급된 기존 연구들의 *공통적인 문제*점은 시스템을 분석하는 자가치유 개발자의 부하가 매우 크다. 첫째로, 목표시스템의 내외부 상태에 대한 문제결정 수준을 개발자가 직접 결정하기 때문에 이에 대한 목표시스템에 대한 분석의 부하가 높고, 목표시스템의 외부 환경을 파악하기 위한 제약조건 모델링의 부하가 높다. 둘째로, 목표시스템의 내부 상태를 파악하기 위한 상태 모델링의 부하가 높고, 특정 컴포넌트의 문제가 발생했을 때 문제가 발생한 컴포넌트와 연관된 컴포넌트들이 무엇인지의 연관성 분석이 쉽지 않다.

결과적으로 본 논문에서는 기존연구들의 공통 취약성들을 부분적으로 해결하기 위해서 목표시스템의 외적 환경과 내적 환경을 만족하는 자가치유 기법을 기반으로 자동화된 문제결정 프로세스와 이를 캡슐화한 소프트웨어 구조를 제안한다. 다음 장에서 이에 대한 상세한 설명을 기술한다.

3. 문제결정을 위한 자가치유 소프트웨어 구조

본 장에서는 시스템 설계 시에 생성된 UML[13] 설계 모델들을 기반으로 시스템의 외부 자원 환경, 시스템 내부 상태 변화의 문제를 해결하는 요구사항을 제시하고, 요구사항을 포함한 자가치유 소프트웨어 구조를 제안한다.

3.1 문제결정을 위한 요구사항

3.1.1 외적 자원 환경관점 요구사항

분산 환경에서 컴포넌트들은 외적 자원 환경 관점에 의해 발생한 문제가 많이 있을 수 있다. 따라서 외적환경 관점의 요구사항은 목표 시스템의 구동환경인 컴퓨팅 자원 부족의 문제나 상호 작용하는 컴퓨팅 개체에 의한 문제를 식별해야한다.

3.1.2 시스템 내부 상태관점 요구사항

시스템의 내부 상태 관점의 요구사항은 시스템을 구성하는 컴포넌트 또는 클래스 내부에서 일어나는 문제를 식별하는 것이다. 내부 상태관점 요구사항을 만족하기 위해서 다음과 같은 요소들을 만족해야 한다. 첫째로, 목표시스템의 설계 시 명세화한 설계모델들을 문제결정을 위해 적용되어야 한다. 둘째로, 명세화된 설계 모델들을 기반으로 컴포넌트 또는 클래스 내부의 행동들을 감시 가능하게 하는 내부 상태 관점의 규칙 모델을 자동 생성해야 한다. 셋째로, 특정 컴포넌트의 비정상적 동작이 다른 컴포넌트에게 영향을 줄 수 있는지를 파악하기 위한 연관성 분석이 이루어져야 한다. 넷째로, 생성된 내부 규칙모델과 연관성 분석을 통해 컴포넌트 또는 클래스의 문제결정 수준을 자동으로 결정해야 한다. 마지막으로, 문제 결정 수준을 기반으로 목표시스템의 내부 상태를 모니터링하고, 내부규칙에 위배되는 상황이 발생하면 컴포넌트 또는 클래스의 문제를 치유해야 한다. 3.2절에서는 시스템의 외부 자원 환경과 내부 상태 관점의 요구사항을 포함한 소프트웨어 구조를 설명한다.

3.2 문제결정을 위한 자가치유 소프트웨어 구조

본 논문에서는 설계단계에서 생성된 UML설계모델들을 이용한다. UML 설계모델들은 크게 외부모델(external model)과 내부모델(internal model)로 나눌 수 있다. 외부 모델은 외적 자원 환경 파악을 위한 배치 모델을 의미하고 본 논문에서는 배치모델에 제약조건을 추가적으로 기술하였다. 내부 모델은 시스템의 내부 상태를 파악하기 위해 클래스 모델, 시퀀스 모델, 상태 모델, 활동 모델이 사용된다.

제안 소프트웨어 구조는 기존 연구에서 자가치유 개발자가 목표시스템을 분석하는데 소모되었던 부하를 부분적으로 감소시키기 위해 설계되었으며, 외부 자원 환경과 시스템 내부 상태에 관한 제약조건, 연관성조건, 그리고 문제결정 수준을 자동화하는 것을 목표로 한다.

3.2.1 제안 구조의 구성

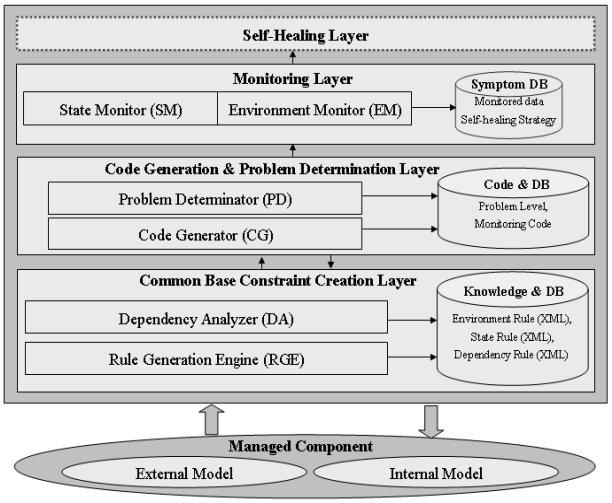
자가치유 소프트웨어 구조는 목표시스템의 기초정보인 외적환경모델(external environment model), 내적환경모델(internal state model)정보를 기반으로 공통제약생성 계층(Common Base Constraint Creation Layer)과, 코드 생성 & 문제결정 계층(Code Generation & Problem Determination Layer), 모니터링 계층(Monitoring Layer), 그리고 자가치유 계층(Self-Healing Layer)의 네 부분으로 구성되며, 전체적인 구조는 (그림 1)과 같다.

• **Rule Generation Engine (RGE):** 공통제약생성계층내에 있는 RGE는 설계모델들의 출력정보인 XMI(XML Meta-Interchange, UML 설계모델의 출력물)[14]정보를 입력받는다. 배치모델을 이용하여 자원제약조건을 파싱하여 환경규칙(Environment Rule)을 생성하고, 클래스 모델, 시퀀스 모델, 상태 모델, 활동 모델을 이용하여 컴포넌트의 내부 상태규칙(State Rule)을 자동 생성한다. 생성된 규칙들은 Knowledge & DB에 저장된다.

• **Dependency Analyzer (DA):** 특정 컴포넌트의 고장이 다른 컴포넌트에게 영향을 미칠수 있기 때문에 DA는 시퀀스 모델을 이용하여 각 컴포넌트들 간의 연관성 분석을 수행한다. DA는 연관성 분석을 위해 서비스 요청과 서비스 응답을 기반으로 분석한다. 연관성 분석을 통해, *Dependency Matrix*를 유도하고, 유도된 *Dependency Matrix*를 통해 컴포넌트간의 연관성을 나타낸 연관규칙(dependency rule)을 자동 생성하여 Knowledge & DB에 저장된다.

• **Code Generator (CG):** CG는 공통제약생성계층에서 생성된 환경 규칙, 상태 규칙, 연관 규칙을 *Rule Model*[15]과 같은 방법을 이용하여 코드로 변환한다. 변환된 코드는 Code & DB에 저장되고, 목표시스템의 외부환경 문제와 내부 상태 문제를 모니터링 할 수 있는 기반이 된다.

• **Problem Determinator (PD):** PD는 수집된 외부, 내부 모니터링 데이터의 문제 상태수준을 자동 결정한다. 첫째로, 외부 자원 환경 상태수준은 애플리케이션 도메인 내에 존재하는 자원 환경에 따라 문제 수준을 *normal, abnormal, panic*과 같은 수준으로 자동결정 한다. 둘째로, 내부 환경 상태수준은 각 컴포넌트들의 상태 리스트가 있고, 컴포넌트가 만족하는 상태 성공률에 따라 컴포넌트의 상태 문제 수준을 *normal, error, panic*과 같은 수준으로 자동 결정한다. 결정된 외적, 내적 문제결정 수준을 Symptom DB에 저장한다.



(그림 1) 문제결정을 위한 자가치유 소프트웨어 구조

• **Environment Monitor (EM):** 배치모델에 명시된 자원 제약사항을 모니터링하고, 모니터링된 정보와 변환된 외부 환경 규칙과 비교하여 제약조건에 위배되는지 아닌지를 판단하기 위해 사용된다.

• **State Monitor (SM):** 상태모델에 명시된 컴포넌트의 내부 상태를 모니터링 한다. 모니터링된 정보와 변환된 내부 상태 규칙과 비교하여 제약조건이 위배되는지 아닌지를 판단한다. 즉, 하나의 컴포넌트가 가지는 각각의 상태들을 만족하는지 아닌지를 파악하기 위해 사용된다.

• **Self-Healing Layer:** 이것은 확장을 위한 향후 연구로써 SymptomDB에 저장된 문제결정 수준에 맞추어 전략을 생성하고 전략의 우선순위를 결정하기 위한 것이다. 전략을 생성하는 부분에 있어서 관리자의 역할이 개입될 수도 있다. 본 논문에서는 시스템의 문제 해결을 위해 적용한 치유 전략은 미리 정의된 재구성 전략[9]을 적용하였다. 예를 들어, 서버의 자원 환경이 좋지 않아 접속된 클라이언트들을 다른 서버로 대체하고, 클라이언트의 내부 특정 컴포넌트가 에러가 발생할 때 그 컴포넌트를 재시작, 재인스톨과 같은 전략을 수행한다.

4. 자가치유를 위한 문제결정 프로세스

본 장에서는 3장에 언급된 소프트웨어 구조 내부에 캡슐화된 문제결정 프로세스를 설명한다. 문제결정 프로세스는 (그림 2)와 같이, 1) 목표시스템의 자가치유 개발자와 소프트웨어 개발자에 의해 UML 설계모델을 나타내는 단계, 2) RGE가 외적 환경 규칙과 내부 상태 규칙을 생성하는 단계, 3) DA가 *Dependency Matrix*와 연관규칙을 생성하는 프로세스를 수행하는 단계를 진행하고, 4) CG가 이전단계에서 생성된 규칙들을 코드화 하는 프로세스를 수행하며, PD가 문제결정 수준을 자동화 하는 프로세스를 수행한다. 5) 마지막으로, EM과 SM이 외적 자원 환경과 내적 상태환경을 모니터링 한다.

4.1 Step1: UML 설계모델을 통한 시스템 명세화 단계

자가치유를 지원하기 위해서는 목표시스템에 대한 특정 관리 관점을 모델링하는 것은 매우 중요하다[4]. 목표시스템의 외부 환경 상태와 내부 컴포넌트 상태를 파악하기 위해, 소프트웨어 설계 시 모델링한 UML설계 모델들을 이용한다. 설계단계에서 생성한 설계모델들을 자가치유를 위한 기초정보로 이용할 수 있다면, 소프트웨어의 문제 발생 시 시스템 스스로가 이를 해결할 수 있다.

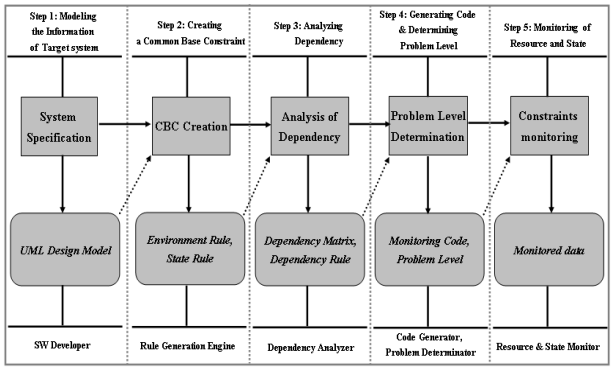
외부 환경 상태를 파악하기 위해서 소프트웨어 설계 시,

애플리케이션 도메인에서 목표시스템의 외부 환경을 명세화한 *배치 모델*을 이용한다. 목표시스템의 물리적 환경을 모델링한 배치모델은 애플리케이션 도메인과 연관된 자원 환경에 관한 제약사항이 기술 될 수 있다. 배치모델에는 특정 애플리케이션 도메인에서 사용되는 시스템들의 제약조건을 추가적으로 기술되어 이것을 기반으로 목표 시스템의 자원 사용률을 파악한다.

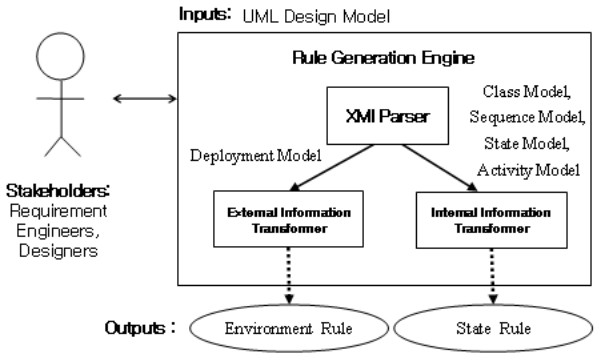
목표 시스템의 내부 상태 파악을 위하여 본 논문에서는 시스템 설계 시 명세화된 *클래스 모델, 시퀀스 모델, 상태 모델, 활동 모델*을 기반으로 한다. 클래스 모델은 관련 컴포넌트 또는 클래스들을 식별하고 컴포넌트들의 수행 오퍼레이션을 파악하는데 사용하고, 시퀀스 모델은 컴포넌트 또는 객체들의 상호작용을 식별할 수 있기 때문에 컴포넌트의 의존관계를 규명하기 위해서 사용된다. 상태모델은 각 컴포넌트의 내부 상태 변화를 식별하기 위해 사용되며, 활동 모델은 각 컴포넌트의 상태 전이 시 만족해야 하는 활동을 식별하기 위해 사용된다. 이러한 UML 설계 모델들은 소프트웨어의 기능적 요구사항을 다양한 뷰 관점으로 표현할 수 있기 때문에 목표시스템의 내부 상태 분석에 많은 도움을 줄 수 있다.

4.2 Step2: 시스템의 외부, 내부 제약사항 생성단계

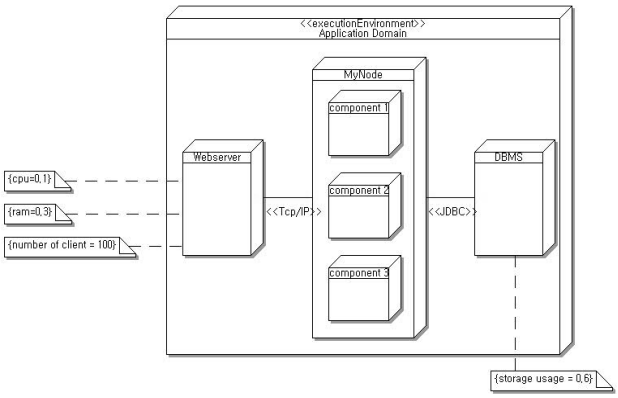
시스템의 제약사항 생성단계에서는 RGE가 외부 제약사항과 내부 제약사항을 생성한다. RGE는 규칙모델[15]과 같은 방법을 적용하여 (그림 3)과 같은 구조로 표현된다. RGE는 UML 설계 모델들을 기반으로 XMI parser가 설계모델들로부터 식별 정보를 파싱한다. 파싱된 정보는 External Information Transformer와 Internal Information Transformer를 통해서 XML 규칙으로 파싱하는 역할을 수행한다. RGE는 외부 제약규칙을 생성하기 위해서 배치모델을 이용하고, 내부 제약규칙을 생성하기 위해서 클래스, 시퀀스, 상태, 활동 모델을 사용한다.



(그림 2) 자가치유를 위한 문제결정 프로세스 (5-steps)



(그림 3) Rule Generation Engine의 구조

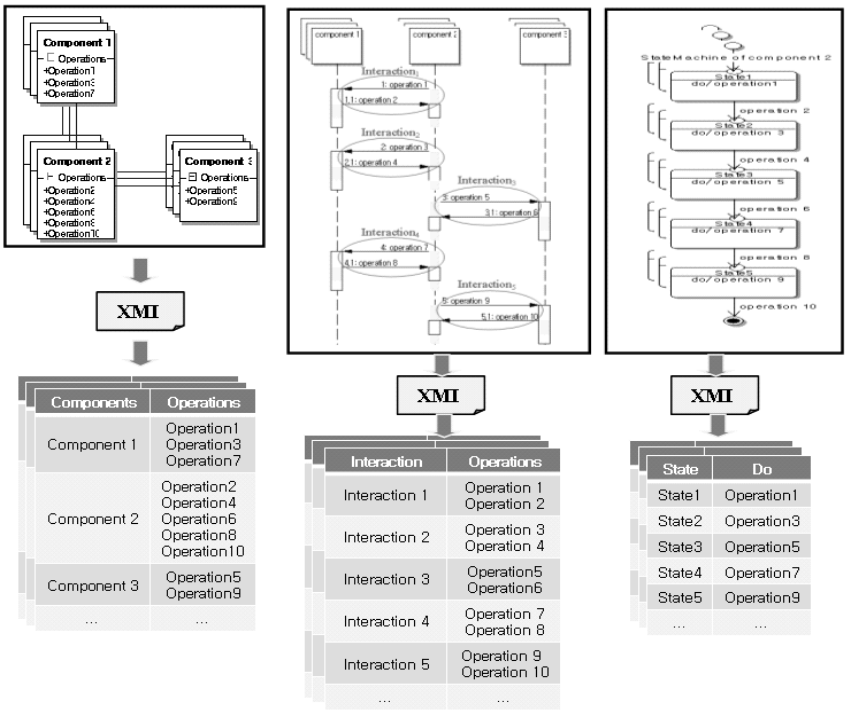


(그림 4) 환경 규칙 생성을 위한 배치모델

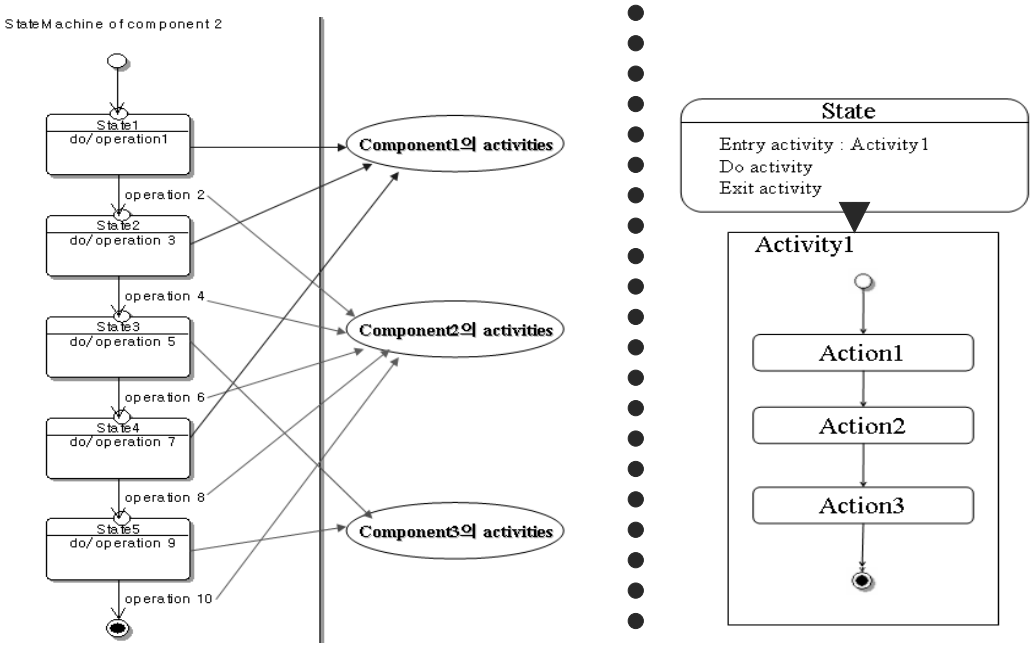
```
<?xml version="1.0" encoding="UTF-8"?>
...
<uml:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" ...>
 <ownedRule xmi: name="Constraint1">
  <specification>
    <body>cpu=0.1</body>
  </specification>
 <ownedRule xmi: name="Constraint2">
  <specification>
    <body>ram=0.3</body>
  </specification>
 <ownedRule xmi: name="Constraint3">
  <specification>
    <body>number of client=100</body>
  </specification>
 <ownedRule xmi: name="Constraint4">
  <specification>
    <body>storage usage = 0.6</body>
  </specification>
</uml:Model>
```

```
<?xml version=1.0'?>
...
<rule>
 <condition>
  <events>
   <component>
    <name>webserver</name>
    <clientnumber>
     <upperlimit>100</upperlimit>
    </clientnumber>
   </component>
  </events>
  <states>
   <relation>and</relation>
   <cpu>
   <available>0.1</available>
   </cpu>
   <ram>
   <available>0.3</available>
   </ram>
  </states>
 </condition>
</rule>
...
```

(그림 5) 외부환경을 위해 추출된 XMI[14] 정보와 XML 기반의 제약 조건 모델



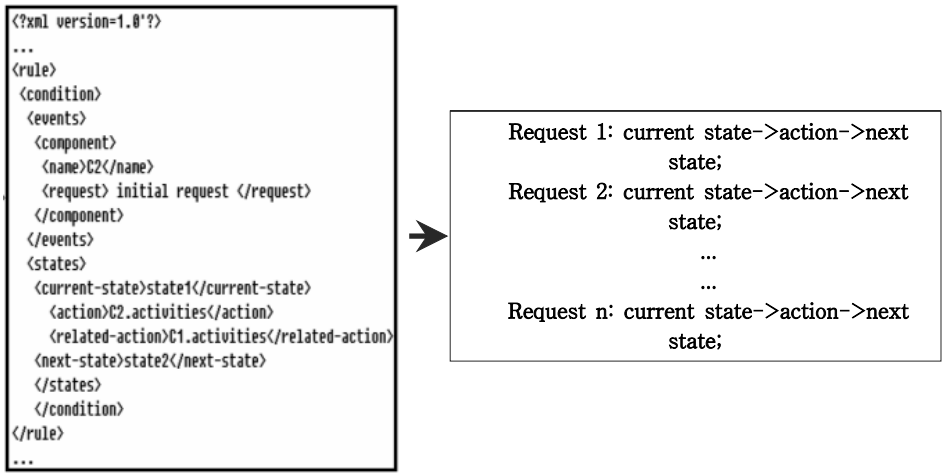
(그림 6) 상태 규칙 생성을 위해 식별된 정보



(그림 7) 특정 상태를 만족하기 위한 활동모델

외부환경을 위한 환경 규칙 생성을 위해서 두 가지의 자원 유형을 구분 한다 (그림 4 참조). 첫째로, 애플리케이션 도메인에 존재하는 외적환경을 식별한다. (예를 들어, DBMS, Webserver 등), 둘째로, ram, cpu, bandwidth, number of client, storage usage 등과 같은 사항들을 식별한다. RGE는 배치모델에서 생성된 XMI 정보 (그림 5의 왼쪽 참조)를 파싱하여 두 가지 유형에 대한 자원 사용률의 임계치를 XML (그림 5의 오른쪽 참조)로 규칙화하여 환경 규칙을 자동 생성한다.

소프트웨어 내부 상태에 관한 상태 규칙 생성을 위해 클래스 모델, 시퀀스 모델, 상태모델, 활동모델을 통해서 생성된 XMI 정보를 이용하여 필요 정보를 식별한다(그림 6 참조). 클래스 모델은 참여 클래스들이 식별되고, 관련 오퍼레이션들이 식별된다. 시퀀스 모델은 각 클래스들의 오퍼레이



(그림 8) 내부 상태 경로 규칙

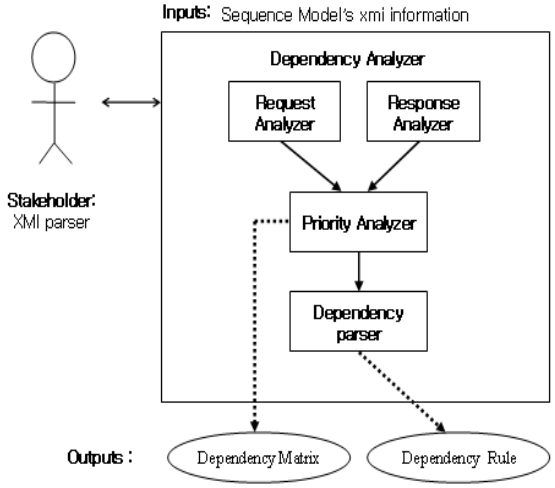
션 요청과 응답에 관한 상호작용들이 식별되고, 상태 모델은 각 클래스가 가지는 상태들이 식별된다. 실제로 각 클래스는 여러 개의 상태를 가질 수 있기 때문에, StarUML[16]과 같은 설계 도구를 통해 상태 모델링 시에 컴포넌트의 특정 상태를 만족하는 활동(activity)을 식별할 수 있다. 본 논문에서는 활동 모델은 여러 개의 "action"들로 이루어지고 이는 실제 함수 이름으로 설계 되어 상태 모델과 연동될 수 있다고 가정한다(그림 7 참조). 식별된 정보를 얻기 위해 설계 모델들에서 생성된 출력 파일인 XMI 정보를 추출하고, RGE는 XMI 정보를 이용하여 내부 상태 경로 규칙을 생성한다 (그림 8 참조).

4.3 Step3: 연관성 분석 단계

컨테이너 기반 프레임워크[1]는 컴포넌트 간의 의존관계를 XML 기술을 통하여 표현하여, 컨테이너가 컴포넌트간의 연관 관계를 추적하는 기반 연구를 수행하였다. 본 연구에서는 이와 유사하게 연관성 분석의 필요성을 지원하기 위해서 (그림 9)와 같이 DA의 구조를 나타낸다.

시퀀스 모델의 XMI 정보를 기반으로 Request Analyzer는 서비스 요청을 분석하고, Response Analyzer는 서비스 응답을 분석한다. 예를 들어, C2가 C1에게 서비스 요청을 하고, C1은 C2에게 서비스 응답을 한다면, C1, C2는 상호작용을 한 것이다(그림 10의 테이블 참조). 이처럼 분석된 정보들은 Priority Analyzer에게 전달되어, 컴포넌트의 서비스 요청과 응답에 관한 단순한 표현식을 통해 인터렉션 I가컴포넌트 C에 의존적이다 라는 사실을 0 또는 1로써 표현하고, I=1인 개수에 비례하여 우선순위를 파악하고, Dependency Matrix 테이블을 생성한다.

(그림 10)의 왼쪽 테이블은 C2가 가장 우선순위가 높다는 것을 알 수 있다. Dependency Parser는 Dependency Matrix 테이블을 기반으로, 컴포넌트간의 의존관계를 나타낸 Dependency rule을 자동생성 한다. <system-element- dependency>는 특정 컴포넌트의 우선순위와 의존관계를 나타내고, <input-dependency>는 특정 컴포넌트에게 서비스를 요청한



(그림 9) Dependency Analyzer의 단순화된 구조

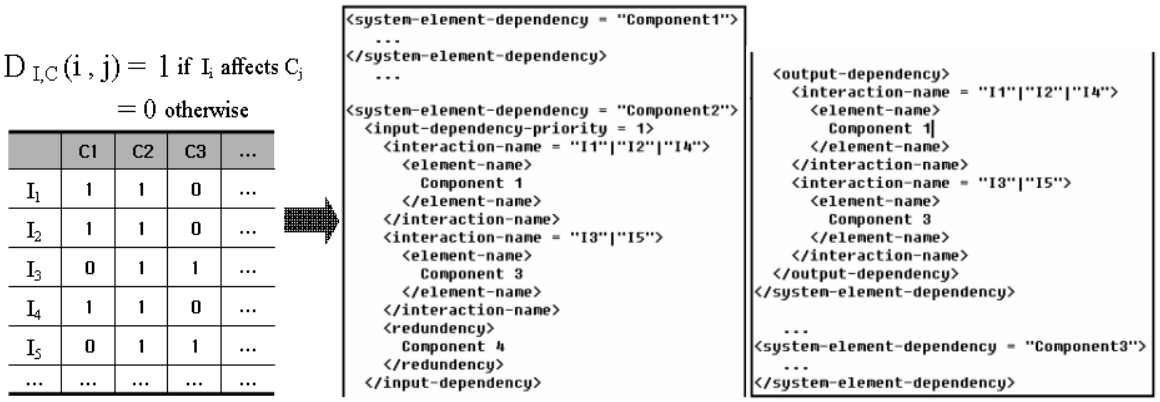
의존적인 입력 컴포넌트들을 나타낸다. <output- dependency>는 요청에 대한 서비스를 응답하는 출력 컴포넌트들을 나타낸다. <interaction-name>은 인터렉션의 이름을 나타내고, <element-name>은 인터렉션이 발생할 때, 해당하는 입력 컴포넌트, 출력 컴포넌트를 식별할 수 있다. 마지막으로, <redundency>는 우선순위가 높은 컴포넌트의 고장 발생 시 대체 전략으로 사용할 수 있는 컴포넌트를 명시한다. 이것은 Self-Healing Layer의 역할을 필요로 하며, 향후 연구에 반영될 요소 이다. 본 연구에서는 <redundency>에 관한 자가치유 전략 사항은 언급하지 않는다.

4.4 Step4: 코드 생성과 문제 수준 결정단계

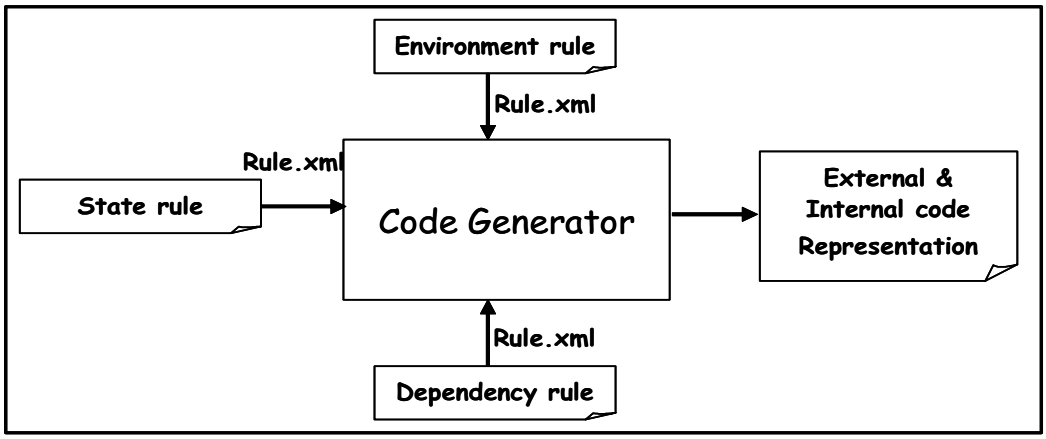
4.4.1 코드 생성 단계

(그림 11)은 이전 단계에서 생성된 제약조건들을 모니터링 가능한 코드로 변환하는 절차를 다음과 같이 나타내고 있다.

- Environment rule (외부 환경 관점 규칙) -> 자원 환경을 위한 코드 변환



(그림 10) Dependency Matrix Table과 Dependency Rule



(그림 11) 규칙모델을 이용한 코드 생성

- State rule (내부 상태 관점 규칙) -> 컴포넌트의 내부 상태를 파악하기 위한 코드 변환
- Dependency rule (연관 관점 규칙) -> 컴포넌트 간의 연관분석을 위한 코드 변환

첫째로, 외부 환경 관점의 코드생성에서, CG는 rule.xml로 부터 환경 규칙을 입력받아 엘리먼트들 사이에 있는 논리적인 관계를 나타낸다. 예를 들어, 외부 환경에 대하여서 (그림 5)의 외부 관점의 rule.xml 내부에 있는 <event>와 <state>엘리먼트들이 순서대로 리스트화 된다. 다음 단계에서, rule.xml 규칙 내부의 이벤트를 통해 외부환경 모니터링 인터페이스에 있는 *operation*과의 관계를 만든다. 예를 들어, (그림 5)에서는 CG가 <event>엘리먼트에 대하여, 컴포넌트의 getClientNumber() 함수와의 관계를 매핑한다. 이것은 <upperlimit> 엘리먼트로부터 getClientNumber()의 값이 100보다 클 때, 이벤트가 일어남을 나타낸다. (그림 5)의 <cpu>, <ram>엘리먼트에 대하여 모니터링 인터페이스 안에 있는 관련 operation들과 매핑한다. 이러한 방법으로 XML로 표현된 규칙들은 아래와 같이 외부 환경에 관해 (그림 12)와 같은 코드로 변환된다.

둘째로, 내부 상태 관점의 코드생성에서, CG는 내부 관점의 rule.xml 로부터 엘리먼트들 사이의 논리적 관계를 나타낸다. 예를 들어, 내부 상태에 대하여 (그림 7, 8)의 rule.xml 내부에 있는 <request-name>의 <current-state>, <action>, <next-state> 엘리먼트들은 순서대로 리스트화한다. 다음 단계에서, rule.xml 규칙 내부의 이벤트를은 내부 환경 모니터링 인터페이스에 있는 "operation"과 "attribute"의 관계를 매핑된다. 예를 들어, <request-name>엘리먼트는 컴포넌트의 getRequest()함수와의 관계로 매핑되고. <current-state>는 currentState 속성과의 관계로 매핑된다. <action>은

```
/* External code representation, C=Component*/
invariant (C.getClientNumber() < 100) && (CPU.available()>=0.1) && (RAM.available() >=0.3))
        ! -> C.setInstanceNumber(120)
 ...
```

(그림 12) 외부 환경을 위한 코드 변환

```
/* Internal code representation, C=Component*/
// State code representation
while(request)
{
 ...
 if(C.getRequest() == "initial request")
 {
  C.currentState = STATE1_START;
  C.activity = C.getCurrentAction();
  C.relationActivity = C.getRelationActivity();

   if((C.activity == false) || (C.relationActivity == false))
   {
     C.currentState == STATE1_FAILURE;
     return C.currentState;
   }

   if((C.activity == true) && (C.relationActivity == true))
   {
     C.currentState == STATE1_OK;
     C.nextState == STATE2
     return C.currentState;
   }
 }
 ...
}
```

(그림 13) 내부 상태를 위한 코드 변환

```
/* Internal code representation, C=Component*/
// Dependency-code representation
if (C2.interactionName == "I1" || "I2" || "I4")
 {
   currentName = C2
   inputName = C1;
   outputName = C1;

   inputAssociation(currentName, inputName);
   outputAssociaton(currentName, outputName);
 }

if (C.interactionName == "I3" || "I5")
 {
   currentName = C2
   inputName = C3;
   outputName = C3;

   inputAssociation(currentName, inputName);
   outputAssociaton(currentName, outputName);
 }
```

(그림 14) 연관성 분석을 위한 코드 변환

getCurrentAction()함수와의 관계로 매핑된다. <next-state>는 nextState 속성과의 관계로 매핑 한다. 마지막으로, 만약 서비스의 요청에 대한 응답을 위해 다른 컴포넌트에게 서비스 요청을 필요로 한다면, <related-activity>는 getRelation Activity() 함수와의 관계로 매핑한다. getCurrentAction() 함수와 getRelationActivity() 함수는 boolean 타입으로 컴포넌트의 상태를 만족하기 위해서 필요로 하는 함수들이 수행되었는지를 통지하기 위해서, 참 또는 거짓의 반환값을 나타낸다. 이러한 방법으로 XML 표현된 규칙들은 아래와 같이 내부 상태 파악을 위해 (그림 13)과 같이 코드로 변환된다.

셋째로, 연관성 분석 관점의 코드 생성을 위하여, CG는 연관 관점의 rule.xml로부터 연관 규칙을 입력받아 엘리먼트 사이의 논리적인 관계를 나타낸다. 예를 들어, 의존관계에 대하여 (그림 10)의 rule.xml 내부에 있는 <input-dependency>와 <output-dependency> 엘리먼트를 순서대로 리스트화 한다. <input-dependency>는 서비스를 요청하는 다른 컴포넌트를 의미하고, <output-dependency>는 서비스에 응답하는 다른 연관된 컴포넌트를 의미한다. 이들 엘리먼트들은 이전 단계에서 시퀀스 모델을 통해 파악이 가능하기 때문에 리스트화 될 수 있다. 이들을 통해서 특정 컴포넌트에 대하여 상호작용할 수 있는 연관 컴포넌트를 식별할 수 있다. <input-dependency>엘리먼트는 inputAssociation() 함수와 currentName 속성으로 매핑되고, <output-dependency> 엘리먼트는 outputAssociation() 함수와 currentName속성으로 매핑된다. <interaction-name>은 interactionName 속성과 매핑되고, <element-name>은 inputName 속성과 output Name 속성으로 매핑된다. 이러한 방법으로 XML 표현된 규칙들은 아래와 같이 연관성 파악을 위해 (그림 14)와 같이 코드로 변환된다. 본 논문의 코드 생성 단계의 내부 상태를 위해 생성된 코드는 Michael E. Shin[2]의 연구와 같이 치유계층에 삽입되어 모니터링 될 수 있다.

4.4.2 문제 수준 결정 단계

문제 수준(problem level)결정 단계에서는 시스템의 외부 환경과 컴포넌트의 내부 상태를 결정 하기 위한 문제 상태 수준을 결정한다. 외적 문제 결정은 <표 1>과 같이 “Normal”, “Abnormal”, “Panic”의 세 가지 상태로 나타낸다. 예를 들어, 웹서버에 연결된 클라이언트의 수가 100보다 크면 안된다는 제약조건이 있고, 이 제약조건의 위배 시에 취해지는 적응 행동이 웹서버의 인스턴스 수를 증가 시키는 것이라면, 이전 단계에서 수행된 외부환경 규칙을 기반으로 자원사용량에 관한 임계치를 초과한 경우에 “Abnomal”상태로 인식하고, 임계치를 초과한 경우에서도 자원사용량이 매우 높다면 “Panic”상태로 인식하여 문제결정 수준을 자동화 한다. Problem Level 2 인 경우에는 메모리 확보를 위해 사용하지 않는 프로세스를 종료하고, Problem Level 3인 경우에는 시스템을 재시작 한다.

내부 상태 문제 결정은 상태 규칙, 연관 규칙을 기반으로 컴포넌트의 우선순위를 결정 한다(상호작용이 많은 컴포넌트에 높은 우선순위를 부여). 예를 들어, 하나의 컴포넌트가 만족 해야 하는 다섯 개의 상태가 있다고 가정 할 때, 컴포

〈표 1〉 외적 자원 환경에 관한 문제 결정 수준

Problem level	Name	Description
1	Normal	정상
2	Abnormal	전략 적용을 위한 지원환경 비정상
3	Panic	자원환경 사용 불가

〈표 2〉 내부 상태에 관한 문제 결정 수준

Problem level	Name	Description
1	Normal	정상
2	Error	컴포넌트 상태 에러
3	Panic	컴포넌트 서비스 제공 불가

넌트가 만족하는 상태 성공률에 따라 컴포넌트의 상태 문제 수준을 <표 2>와 같이 Normal, Error, Panic 으로 나타낸다. 만약 특정 컴포넌트에 문제가 발생하였다면, 우선순위가 높은 컴포넌트의 상태를 조사한다.

4.5 Step5: 외부 자원 환경 모니터링 과 내부 상태 모니터링 단계

모니터링 단계에서는 이전 단계들에서 생성된 규칙과, 생성된 코드, 문제결정 수준과 같은 산출물들을 기반으로 컴포넌트의 비정상적인 환경과 상태 문제 발생 감지를 위해 외부 환경과 내부 상태에 대한 모니터링을 지원하는 단계 이다.

외적 환경 모니터링 (external environment monitoring) 단계에서는 JMX(Java Management Extension)가 자원 환경 관점을 모니터링 할 수 있는 표준 인터페이스를 제공하기 때문에 외부 환경 모니터링을 가능하게 한다. 배치모델에서 생성한 자원 조건 상황을 모니터링하여 자원 환경에 대한 문제 수준이 “normal”, “abnormal”, “panic”인지를 판단한다. 생성된 규칙을 기반으로 만약 문제의 수준이 “panic” 상태라면 시스템을 재부팅하고 관리자에게 통보한다. 만약 문제의 수준이 “abnormal” 상태라면 사용되지 않는 프로세스를 “kill” 하거나, 다른 서버로 리다이렉트 할 수 있다.

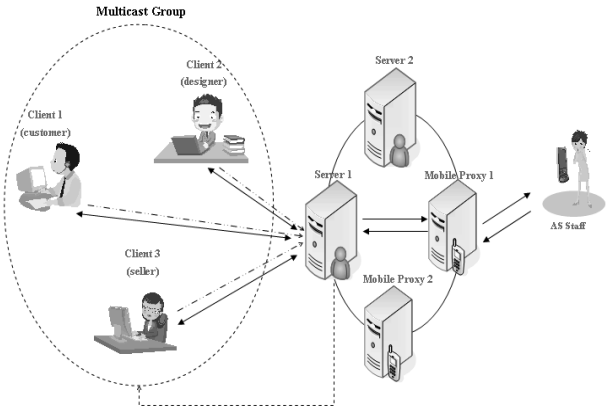
내부 상태 모니터링 (internal state monitoring)단계에서는 컴포넌트의 내부 상태를 모니터링 하는 목적을 가지며, 하나의 컴포넌트가 가지는 여러 상태를 모니터링 한다. 클래스 모델, 상태 모델, 활동 모델에서 분석된 정보를 기반으로 우선순위가 높은 컴포넌트를 먼저 모니터링 하고, 모니터링을 위해 변환된 코드를 컴포넌트 내부에 삽입하여 컴포넌트의 상태를 만족하는 활동들을 체크한다. 만약 내부 상태 모니터링 결과가 *error* 상태라면 컴포넌트를 블락킹 하여 초기화하거나, 대체 컴포넌트를 적용할 수 있고, *panic*상태라면 컴포넌트의 서비스를 제공할 수 없기 때문에 관리자에게 통보 후, 컴포넌트를 재 시작하게 할 수 있다. 본 연구에서는 모니터링 결과에 대한 전략 적용과 전략의 우선순위 문제는 향후 연구로 남긴다.

5. 구현 및 평가

비디오 회의 시스템의 기본 설계는 UML로 표현하고, 이것의 목적은 성공적인 미팅의 수행이다. 미팅의 수행기간 동안 사용자들은 파일을 송수신 하며, 네트워크 문제 등으로 방해받지 않아야 한다.

5.1 비디오 회의 시스템의 구성 환경 및 동작

(그림 15)는 비디오 시스템을 통해서 온라인 회사의 제품을 매매하는 *customer, seller,* 그리고 제품 *designer*와 *AS staff*가 참여하여 함께 회의를 진행한다. 비디오회의 시스템에 사용된 구성 환경으로 다음과 같다.



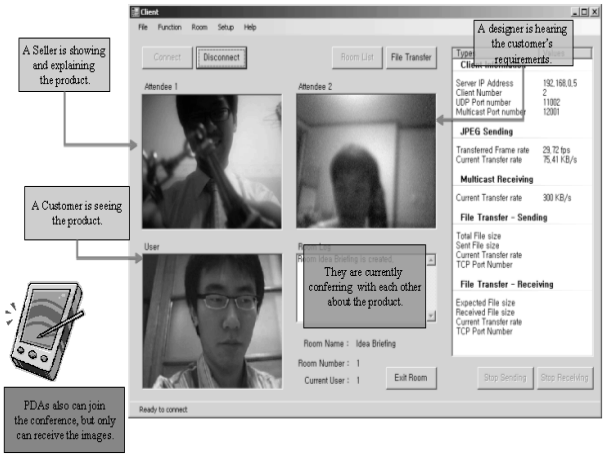
(그림 15) 비디오 회의 시스템의 구조

- Client 1과 Client2는 데스크탑 사용자로써 CPU 1.0GHz, 512MB의 사양을 기반으로 .NET 2.0, Direct X 8.0이상이 설치되어 있고, Client 3은 PDA 사용자로 CPU 400MHz, 64MB의 사양을 가지고 있다.
- Mobile Proxy1는 CPU 2.0GHz, 512MB의 사양을 기반으로 JDK 1.4 버전이 설치되어 있다. 이것은 PDA 사용자를 대신해서 화상회의 세션에 연결하는 역할을 한다. Mobile Proxy2는 Mobile Proxy1과 같은 사양을 가지며, Mobile Proxy1의 자원 환경이 좋지 않을 때, 재구성 전략[12]으로 사용하기 위해 이용하였다.
- Server1는 CPU 2.0GHz, 1GB의 사양을 가지며, JDK 1.4 이상 버전이 설치되어 클라이언트들의 세션 정보를 관리하고 멀티캐스팅을 하는데 이용하였으며, Server2는 Server1과 같은 사양을 가지며 Server1의 자원 환경이 좋지 않을 때, 재구성 전략[12]으로 사용하기 위해 이용하였다.

비디오 회의 시스템은 멀티캐스트를 기반으로 동작한다. 연결되어 있는 단말 중에서 PC는 서버에게 UDP로 영상을 전송하며, 서버는 부여된 포트 번호로 멀티캐스트를 이용하여 전송을 한다. PC는 서버에게 자신의 IP를 등록하고, 서버는 PC에게 사용할 UDP 포트 번호를 알려 주고, 서버는 등록된 IP를 통해서 세션을 관리하게 된다. 단말 중에 PDA는 Mobile Proxy를 통하여 PC 단말들과 화상회의를 진행할 수 있다. 각 단말들은 화상채팅을 위한 카메라와 영상 및 메시지를 주고 받으며 이를 사용자에게 보여줄 수 있는 스크린을 보유하고 있다.

비디오 회의를 진행하는 동안, (그림 16)과 같이 한 접속

(그림 16) 비디오회의 시스템의 비정상적인 경우



(그림 17) 자가치유 적용 후의 비디오 회의 시스템의 정상적인 상태

자의 클라이언트가 이상 상태에 놓여졌다. 이상 상태는 해당 클라이언트의 컴퓨터 및 네트워크에 많은 부하가 있는 상태 이며, 해당 클라이언트는 정상적으로 영상을 송신하지 못하고 전체적으로 멀티캐스트 전송 속도가 많이 저하된 상태이다. 자가 치유를 위해서 비디오 회의 시스템에 내장된 제안 방법론은 다음과 같은 동작들을 수행한다.

- 비디오 회의 시스템의 설계 모델에서 자동 생성된 규칙들을 (4절 참조) 바탕으로 문제 원인을 파악한다. 영상을 전송하는 것은 state 1 (카메라로부터 영상 전송) - state 2 (전송된 데이터를 처리) - state 3(처리된 데이터를 전송)의 상태를 거치게 된다는 것을 인식한다. 인식된 상태들을 바탕으로 각 상태의 특정 활동들을 인식한다.
- 카메라로부터 영상이 전송 되었는지를(state1) 파악하기 위해 상태 규칙을 위반하는 데이터가 있는지 없는지 확인하고 없으면 정상임을 인식한다. 전송된 데이터를 알맞게 처리하는 부분(state2)의 모니터링 데이터를 분석하던 중 카메라로부터 들어온 데이터의 처리에 오류가 발생하여 상대방에게 이해할 수 없는 데이터를 전송(state3)하고 있다는 것을 인식 하였다. 클라이언트의 특정 상태가 “error”로 파악되었기 때문에 이에 적당한 해결책으로서 기존에 미리 정의된 치유 정책에 따라 컴포넌트 초기화, 컴포넌트 재인스톨, 컴포넌트 재시작과 같은 재구성 전략을 수행한다.

비디오 회의 시스템의 비 정상상태는 치유 후, (그림 17)과 같이 다시 정상 상태로 돌아와 성공적으로 회의를 진행할 수 있게 되었다. 기존에는 문제가 발생하면 관리자가 처리하여 수동으로 소프트웨어 모듈들을 재구성하였지만, 자가치유 소프트웨어 구조를 내장한 경우 스스로 문제를 파악, 치유하여 성공적인 미팅을 수행하는 것이 가능해졌다.

구현을 통하여 4장에서 언급한 제안 방법론들이 비디오 회의 시스템에 적용 가능함을 확인하였다. 비 정상상태 발생 시 자가 치유 전략은 기존연구에서 사용된 재구성 전략을 주로 사용하였지만, 이것은 기존 연구들에서 가장 많이 사용하는 전략이다. 자가 치유 전략은 관리자의 정책에 맞게 외부 환경과 내부 상태에 따라 다양하게 미리 전략화 할 수 있다.

5.5 제안 방법론의 평가

제안 방법론은 UML 기반의 설계 모델로부터 목표 시스템의 외부 자원 환경과 내부 상태 문제에 대한 제약조건, 연관성 분석, 모니터링 코드 생성 그리고 문제결정 수준을 추출하는 것을 보다 쉽게 할 수 있도록 지원 한다. 제약조건은 목표시스템의 동작에 관한 임계 조건이고, 연관성 분석은 목표 시스템 내부의 컴포넌트들 간의 연관 관계를 나타내며, 코드생성과 문제결정은 제약조건을 위배하는지에 관한 코드를 통해 목표 시스템이 정상인지 비정상인지를 결정하는 것이다. 이러한 자가치유를 위한 활동들을 제안 방법론과 기존 연구들의 방법론으로 수행할 경우의 비교는 아래의 <표 3>과 같다.

기존 방법론으로 자가치유 활동을 하기 위해서는 자가 치

〈표 3〉 분석 단계에서의 자가치유를 위한 활동 비교

활동	제안 방법론	기존 방법론
자가치유의 범위	외부 자원 환경 문제 와 내부 상태 문제	외부 자원 환경 문제 또는 내부 상태 문제
제약조건 설정	UML 설계 모델들의 XMI 정보를 이용한 외부 자원 환경 규칙과 내부 상태 규칙 설정 자동화 가능	자가치유 시스템 개발 전문가에게 의뢰
연관성 분석	시퀀스 모델의 XMI 정보 분석을 통해 Dependency Matrix″ 를 자동 생성하여 컴포넌트 간의 연관 관계 도출	자가치유 시스템 개발 전문가에게 의뢰
모니터링 코드 생성	제약조건을 기술한 XML을 기반으로 태그 정보와 모니터링 인터페이스 매핑을 통한 코드 생성 가능	자가치유 시스템 개발 전문가에게 의뢰
문제결정 수준 설정	외부 환경 규칙과 내부 컴포넌트의 상태 성공률을 기반으로 한 문제결정 수준 자동화 가능	자가치유 시스템 개발 전문가에게 의뢰

Operation	정상적인 상황	비정상적인 상황	제안방법론 적용 후	비교
	Processing Time (ms)	Processing Time (ms)	Processing Time (ms)	Difference(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(), processorCreator(), transmitterCreater(), start()	1800	5330	1910	+110
openSession(), getReceiveStream(), getDataReceive()	1400	4610	1515	+115
Total	**4,200 ms**	**13,970 ms**	**4,745 ms**	**+545 ms**

〈표 4〉 제안방법론이 적용된 작업처리 시간 평가

유 전문가의 역할이 소프트웨어 개발자의 역할 보다 매우 높기 때문에 제약조건 설정, 연관성 분석, 코드 생성, 문제결정 수준 설정을 위해 목표 시스템의 분석과 자가치유 시스템 설계에 많은 시간과 노력을 필요로 한다. 또한, 기존 방법론은 주로 외부 자원 환경에 대한 자가 치유 전략을 적용하고 있기 때문에 내부 상태에 대한 문제를 결정할 수 없다. 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 자동화 하여 목표 시스템의 분석과 자가치유 시스템의 설계를 위한 노력을 줄일 수 있다. 또한 제안 방법론은 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 탐지할 수 있다.

본 논문에서는 비디오 회의 시스템을 통해서 제안 방법론이 자가치유의 능력을 가지는지를 보이기 위해서, 우리의 이전 연구인 동적 분산 적응 프레임워크[19][20]에서 사용되었던 process_generator를 이용하였다. Process_generator는 dummy process를 생성하는 것으로 서버에 접속한 사용자 수와 리소스 사용량을 기록한 로그를 이용하여, 접속사용자의 증가를 시뮬레이팅 하기 위해 개발되었다. 시뮬레이션 결과 비정상적인 경우 <표 4>와 같이 작업처리 시간이 전체적으로 매우 높아졌다는 것을 알 수 있었다.

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 생성되어 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 <표 4>와 같이 나타내었다. 비교 결과 정상적인 상황에서의 작업처리시간 보다 전체적으로 +545ms의 작업처리 시간이 더 소요되었지만 전체적인 성능에는 영향을 미치지 않은 것으로 판단하여 제안 방법론의 효율성을 입증하였다. 이 평가 방법은 제안 방법론 자체를 평가하는 좋은 방법은 아니지만, RAINBOW 프레임워크[9]와 유사하게 제안 방법론을 수행하는 내부 모듈들이 비정상적인 상황에서 정상적인 상황으로 회복을 할 수 있는지를 시뮬레이션 하기 위한 좋은 사례가 된다.

6. 결 론

본 논문에서는 자가 치유 개발자의 분석 노력을 줄이기 위해서 설계 단계에서 생성된 산출물들을 이용하여 자동화된 문제결정 방법론을 지원하는 자가치유 소프트웨어 구조를 제안하였다. 이를 통해 얻을 수 있는 이점은 다음과 같다.

- **설계 모델을 이용한 외부, 내부 상황의 제약조건 자동 생성:** 설계 모델을 이용하여 목표시스템에 대한 제약조건을 자동 생성할 수 있다. 소프트웨어 환경이 점차 복잡해져 가고 있기 때문에 자가 치유 개발자가 제약조건을 명시적으로 생성하는 것은 많은 분석의 노력이 필요하다. 설계 단계에서 생성된 산출물들을 제약조건 자동 생성 메카니즘과 연결함으로써 제약조건으로 변환하는 것이 가능해 졌다. 따라서, 이를 통해서 자가 치유 개발자의 부하를 줄일 수 있다.
- **연관성 분석 지원:** 연관성 분석의 문제는 시퀀스 설계 모델을 기반으로 필요정보를 추출할 수 있다. 즉, 특정 컴포넌트와 상호작용하는 여러 컴포넌트들을 파악할 수 있는 기준을 제공하기 때문에 보다 세부적인 컴포넌트의 상호 연관관계를 파악할 수 있다.

- **문제결정수준 자동화:** 목표시스템의 외부환경, 내부상태에 관한 제약조건들을 기반으로 특정 상황에 대한 문제결정을 지원한다. 이것은 목표 시스템에 대한 문제 진단을 용이하게 하고, 기존의 자가치유 개발자의 부하이던 문제결정 수준에 대한 분석의 노력을 줄이는 것이 가능하다.
- **자가치유를 위한 설계 지원의 확장성:** 소프트웨어의 설계 단계에서 문제결정을 지원할 수 있는 산출물들을 확장하게 되면, 제안방법론을 통해 문제 상황을 정확하게 또는 유사하게 나타낼 수 있는 기반 정보의 추출이 용이하다. 이것은 제안 소프트웨어 자가 치유의 의미적 구조를 포함하고 있기 때문에, 자가 치유를 지원하기 위한 확장 설계에서도 지원이 가능하다.

본 연구가 실제로 애플리케이션에 적용되기 위해서는 여러 가지 문제들이 해결되어야 한다. 가장 큰 어려운 점은 소프트웨어 설계자가 자가치유를 지원하기 위한 지식을 추가해야하는 설계의 노력이 필요로 한다. 이것은 소프트웨어 설계자의 많은 경험뿐만 아니라 지속적인 설계의 갱신이 이루어져야 하는 부분으로 지속적인 노력이 있어야 효과적인 평가를 이룰 수 있다. 또한 문제에 대한 자가치유 전략을 얼마나 효과적으로 계획하느냐 하는가에 따라 목표시스템의 신뢰성이 차이가 있을 수 있다.

구현상의 문제에 있어서는 설계 모델들에서 생성된 XMI 정보의 방대함으로 인해 적절한 필요정보를 추출하는데 많은 시간이 소요 될 수 있고, 외적환경 규칙과 내적 상태 규칙간의 충돌이 발생할 수도 있다. 또한 재구성 전략만을 사용한 본 연구에서는 여러 전략들을 세분화해서 전략들의 우선순위를 정하여 세분화된 문제결정 수준이 분류되도록 향후 연구에서 다룰 예정이다.

참 고 문 헌

[1] Rajesh Kumar Ravi, Vinaya Sathyanarayana "Container based framework for Self-healing Software System," Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, pp.306-310, May, 2004.

[2] Michael 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, Jul., 2005.

[3] Michael E. Shin, and Jung Hoon An "Self-Reconfiguration in Self-Healing Systems," Proceedings of the 3th IEEE International Workshop on Engineering of Autonomic & Autonomic Systems, pp.89-98, Mar., 2006.

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

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

[6] IBM, Autonomic Computing, "IBM's Perspetive on the State of Information Technology," http://www.ibm.com/industries/government/doc/content/resource/thought/278606109.html.

[7] Jeongmin Park, Giljong Yoo, Chulho Jeong, and Eunsoek Lee, "Self-Management System based on Self-healing Mechanism," LNCS 4238, pp.372-382, Sep., 2006.

[8] Giljong Yoo, Jeongmin Park, and Eunseok Lee "Hybrid Inference Architecture and Model for Self-healing System," LNCS 4238, pp.566-569, Sep., 2006.

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

[10] MarkWeiser, "The Computer of 21st Century," Scientific American, 265(3), pp.94-104, Sep., 1991.

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

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

[13] UML Online Document. http://www.omg.org/xml.

[14] XMI Online Document. http://www.omg.org/xml.

[15] Qianxiang Wang, "Towards a Rule Model for Software-adaptive Software," ACM SIGSOFT Software Engineering Notes vol.30, No.1, pp.1-5, Jan., 2005.

[16] StarUML, http://staruml.sourceforge.net/en/.

[17] Kendra Cooper, Lirong Dai, Yi Deng, "Performance Modeling and analysis of software architectures: An aspect-oriented UML based approach," Science of Computer Programming Vol.57, No.1, pp.89-108, Jul., 2005.

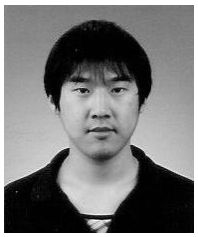
[18] R. Monroe, "Capturing software architecture design expertise with Armani," Carnegie Mellon University School of Computer Science, Technical Report No. CMU-CS, pp.98-163, C Oct., 1998.

[19] 이승화, 조재우, 이은석 "모바일 환경에서 웹 서비스 품질보장을 위한 동적 분산적응 프레임워크," 정보처리학회논문지D, 제13-D권 제6호, pp.839-846, 2006.

[20] Seunghwa Lee, Jee-Hyoung Lee, and Eunseok Lee, "An Inference Engine for Personalized Content Adaptation in Heterogeneous Mobile Environment," LNCS 4239, pp.158-170, Oct., 2006.

박 정 민

e-mail : jmpark@ece.skku.ac.kr
2003년 2월 한국산업기술대학교 컴퓨터공학과(공학사)
2005년 2월 성균관대학교 대학원 컴퓨터공학과(공학석사)
2005년 3월～현 재 성균관대학교 대학원 컴퓨터공학과 박사과정
2008년 3월～현 재 동양공업전문대학 전임강사
관심분야 : 소프트웨어공학, 오토노믹 컴퓨팅, 자가치유 소프트웨어, 컴포넌트 기반 개발 방법론



정 진 수

e-mail : seba702@ece.skku.ac.kr
2007년 2월 성균관대학교 정보통신공학과 (공학사)
2007년 3월～현 재 성균관대학교 대학원 컴퓨터공학과 석사과정
관심분야 : 소프트웨어공학, 오토노믹 컴퓨팅, 자가치유 소프트웨어

이 은 석

e-mail : eslee@ece.skku.ac.kr
1985년 2월 성균관대학교 전자공학과 (공학사)
1988년 3월 일본 동북(Tohoku)대학교 대학원 정보공학과(공학석사)
1992년 3월 일본 동북(Tohoku)대학교 대학원 정보공학과(공학박사)
1992년～1994년 일본 미쯔비시 정보전자연구소 특별 연구원
1994년～1995년 일본 동북(Tohoku)대학교 Assistant Prof
1995년 3월～현 재 성균관대학교 정보통신공학부 교수
관심분야 : 소프트웨어공학, 오토노믹/유비쿼터스 컴퓨팅, 에이전트 지향 지능형 시스템 등