자가 치유를 위한 외부 자원 모니터 자동 생성 기법

이희원°, 이준훈, 정진수, 박정민, 이은석
성균관대학교 컴퓨터공학과
e-mail : {lhw1105°, trsprs, seba702, jmpark, eslee}@ece.skku.ac.kr

An Automated Approach to Monitoring External Resource for Self-Healing

Heewon Lee°, Joonhoon Lee, Jinsoo Jung, Jeongmin Park, Eunseok Lee
Dept. of Computer Engineering, Sungkyunkwan University

요 약

최근의 소프트웨어들이 다양한 기능을 갖추어가면서 점차 복잡도가 증가하고 있으며, 이에 따라 오류로부터의 복구도 어려워져 가고 있다. 이러한 변화는 소프트웨어의 자가 치유 연구에 중요한 이슈가 되고 있다. 하지만 자가 치유 방법론에서 중요한 요소 중에 하나인 모니터는 아직까지 개발자가 일일이 작성해야 하는 한계가 있다. 따라서 본 논문은 외부 자원으로 인한 오류를 탐지하는 모니터 모듈의 생성을 자동화하는 방법론을 제시하고, 이것을 적용한 소프트웨어 아키텍처를 제안한다. 본 방법론은 1) UML의 배치 다이어그램으로부터 소프트웨어와 하드웨어간의 연결을 분석하고, 2) 기술된 제약사항을 이용하여 모니터링 모듈을 자동으로 생성한다. 3) 이후 생성된 모듈을 소프트웨어 사양에 맞게 수정한 후 컴포넌트에 추가한다. 이러한 제안 방법론을 통해 기존에 수동으로 만들어야 했던 외부 자원 모니터를 자동화하는 것이 가능해 진다. 본 논문에서는 평가를 위해 제안 방법론을 비디오 회의 시스템의 클라이언트에 적용하여, 외부 자원의 오류를 올바르게 탐지해내는지 확인한다.

1. 서론

사용자들이 더욱 다양한 기능들을 요구함에 따라 이를 지원하는 소프트웨어의 구조도 점점 복잡해져 가고 있다. 소프트웨어 구조가 복잡해져 갈수록 소프트웨어 내외부에서 발생한 오류에 대처하기가 어렵고 수정을 위해 많은 비용을 요구하게 된다[1]. 이에 따라 자가 치유가 가능한 소프트웨어 컴포넌트에 대한 연구가 진행 중에 있다. 자가 치유란 생물학 시스템에서 상처를 치유하는 개념과 같이 스스로 오류의 탐지 및 복구를 수행하며, 정상적인 성능 수준을 되찾으려는 행위를 말한다[1]. 오류에는 소프트웨어 내부에서 발생하는 오류도 있지만, 외부 자원으로 인하여 발생하는 문제도 있다. 하지만 기존의 연구에서는 외부 자원으로 인하여 발생하는 문제를 탐지하기 위한 모니터를 전부 수동으로 작성해야만 했다[2].

본 논문에서는 이러한 한계를 해결하기 위해 UML의 배치 다이어그램 (Deployment diagram)[5]을 분석하며, 이를 이용해 외부 자원의 이상을 탐지하기 위한 모니터의 자동 생성을 보조해주는 전략을 제안한다.

- UML의 배치 다이어그램으로부터 소프트웨어와 하드웨어간의 연결을 분석한다.
- 각종 제약사항을 이용하여 모니터링 모듈을 자동으로 생성한다.
- 생성된 모니터링 모듈을 소프트웨어 사양에 맞게 수정한 후 컴포넌트에 추가한다.

위 제안 방법론을 통해 기존에 수동으로 작성해야 했던 외부 자원 모니터를 자동으로 생성 할 수 있다. 또한 개발자는 생성된 모니터를 수정하여 해당 소프트웨어에 적합한 모니터로 확장할 수 있으며, 치유 전략을 별도로 추가할 수도 있다. 본 논문에서는 평가를 위해 제안 방법론을 비디오 회의 시스템의 클라이언트에 적용하였다. 평가는 제안 방법론이 적용된 비디오 회의 클라이언트에서 외부 자원 오류가 발생하였을 때 이를 적절히 탐지할 수 있는지 검증하는 방법으로 한다.

본 논문은 다음과 같은 순서로 기술한다. 2장에서는 관련 연구들을 분석하여 설명하였으며, 3장에서는 모니터의 자동 생성을 위한 방법론을 제안한다. 4장에서는 해당 방법론을 적용한 비디오 회의 시스템의 평가 결과를 소개하고, 마지막으로 5장에서는 결론 및 향후 과제를 기술한다.

2. 관련 연구

본 장에서는 Michael E. Shin[2,3]이 제안한 자가 치유 컴포넌트 아키텍처와, 자율 고장-탐지 알고리즘(Autonomic Failure-Detection algorithm)[4]을 분석하여 이에 대한 장단점을 기술한다.

2.1 자가 치유를 위한 컴포넌트 구조

Michael E. Shin[2,3]이 제안한 자가 치유 컴포넌트는 각 컴포넌트마다 치유 계층(Healing layer)과 서비스 계층(Service layer)으로 나뉘어져 있다. 다른 컴포넌트나 시스템으로부터 요청받은 작업을 수행하는 서비스 계층은 활동 객체, 활동 객체 간 연결자, 그리고 활동 객체에

의해 접근되는 수동 객체로 이루어져 있다. 활동 객체는 제어권을 가지고 있어 다른 활동 객체 혹은 수동 객체를 실행 시킬 수 있다. 이에 반해, 수동 객체는 활동 객체에 의해 호출되며 제어권을 가지고 있지 않아 외부에서 호출되지 않는 한 스스로 수행될 수 없다. 활동 객체 간 연결자는 활동 객체를 대신하여 다른 객체에게 메시지를 전달하며, 이를 동기화하는 역할을 수행한다. 이러한 서비스 계층에 이상이 생겼을 경우 치유를 담당하는 치유 계층은 여섯 개의 모듈로 구성되어 있다. 서비스 계층에서 보내는 메시지를 이용하여 각 객체들의 행위를 감시하는 *Component Monitor* 와 서비스 계층에 속한 객체의 구성 정보를 담고 있어 오류 발생 시 재구성 전략을 생성하여 주는 *Component Reconfiguration Plan Generator* 가 있으며, 서비스 계층에 속한 객체의 복구 전략들을 담고 있으며 오류가 발생한 객체의 치유 전략을 세우는 *Component Repair Plan Generator* 가 있다. 또한 이러한 전략을 실제로 수행하기 위한 수행 모듈 *Component Reconfiguration Executor*, *Component Repair Executor* 가 있으며, 이러한 다섯 개의 모듈들을 조율하고 제어하기 위한 *Component Self-Healing Controller* 가 있다. 위 아키텍처는 다음과 같은 장단점을 가지고 있다.

- 컴포넌트 내의 객체 중 오류를 발생시킨 객체를 정확히 알아낼 수 있다.
- 해당 오류에 대응하는 전략이 내장되어 있어 올바른 치유 전략을 세울 수 있다.
- 그러나, 각 객체별 치유 전략을 미리 구성하여야 하고, 메시지 전달 과정에서 미세한 오류도 허용하지 않기 때문에 개발자가 아키텍처를 설계할 때 많은 노력이 소요된다.
- 외부 오류의 탐지도 가능하지만 컴포넌트 내부에서 발생한 오류만 탐지 가능하도록 설계하였다.

2.2 자율 고장 탐지 알고리즘

K. Mills et al[4]이 제안한 자율 고장 탐지 알고리즘은 모니터링 대상이 되는 객체나 장치가 마치 심장 박동과 같은 일정한 주기로 모니터에게 신호를 보내며, 모니터가 이에 응답하여 신호를 보내는 원리이다. 모니터링 대상은 여러 개가 될 수 있으며 응답하는 모니터는 하나이다. 만약 지정된 주기 내에 모니터링 대상으로부터 신호가 오지 않으면 해당 객체나 장치가 이상 상태에 놓였다고 판단한다. 한 번의 신호 주기를 H_P(Heartbeat period) 라고 한다면 이 시간 내에 오류를 탐지할 수 있으므로 오류 탐지에 걸리는 최대 시간은 H_P가 된다. 하지만 오류는 해당 신호 주기 내의 어느 시간에라도 발생할 수 있으므로 오류 탐지에 걸리는 평균 시간은 $H_P / 2$ 가 된다. 위 알고리즘은 짧은 시간 내에 모니터링 대상의 오류 여부를 파악할 수 있다는 장점이 있지만, 잦은 신호 교환으로 인한 오버헤드가 크다는 것이 단점이다.

3. 제안 시스템

본 논문에서는 소프트웨어 내부에서 발생한 오류의 경우 기존의 자가 치유 아키텍처[2, 3]를 적용하여 치유할 수 있다고 가정하며, 외부 자원 환경으로 인해 발생한 오류에 대응할 수 있도록 지원하는 모니터 자동 생성 기법을 제안한다. 이를 위하여 "heartbeat" 알고리즘[4]을 활용하며, 기존의 알고리즘과는 반대로 모니터가 외부 자원에 신호를 보내어 값을 측정하거나 이상 유무를 탐지한다.

3.1 외부 자원 모니터 자동 생성 기법

본 장에서는 외부 자원 모니터를 자동으로 생성하기 위한 기법을 설명하며, 구조적인 측면과 프로세스적인 측면으로 나누어 기술한다.

3.1.1 외부 자원 모니터 자동 생성 기법의 구조

외부 자원 모니터를 자동으로 생성하기 위한 설계 구조는 크게 분석 과정과 생성 과정으로 나뉘어져 있다. 그림 1은 이러한 구조의 흐름을 보여주고 있다.

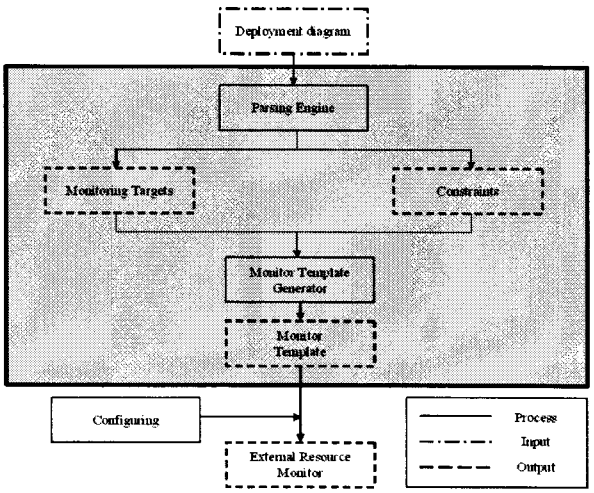

그림 1 외부 자원 모니터의 자동 생성 기법의 흐름

- Deployment Diagram : XMI(XML Meta-Interchange, UML 설계모델의 출력물)[6] 형식으로 변환된 UML의 배치 다어그램으로 본 아키텍처의 입력물이다.
- XMI Parser : 배치 다이어그램을 분석하여 외부 장치와의 연결 상황이나 자원 제약 사항들을 분석한다. 분석 과정에서의 결과물은 모니터링 하려는 대상(Monitoring Targets)과 제약 사항(Constraints)이며, XML 형식으로 파싱된다.
- Monitor Template Generator : XMI 파서의 출력 결과를 전달받은 모니터 템플릿 생성기는 해당 입력 값을 이용하여 모니터링 하고자 하는 장치나 자원의 오류 상황을 탐지해주는 모니터 템플릿을 자동으로 생성한다. 이 템플릿은 주로 특정 언어에 대한 소스코드로 구현된다.
- Configuring : 자동 생성된 모니터 템플릿은 소프트웨어의 구성 상태에 맞게 변형되어야 하기 때문에 개발자에 의한 모니터 모듈의 수정 과정이다.
- External Resource Monitor : 위 아키텍처의 최종 결과물로서 소프트웨어에 바로 적용이 가능한 외부 자

원 모니터 모듈이다.

3.1.2 외부 자원 모니터 자동 생성 기법의 프로세스

본 절에서는 그림 2와 같이 모니터 자동 생성을 위한 프로세스를 네 단계로 나누어 설명한다.

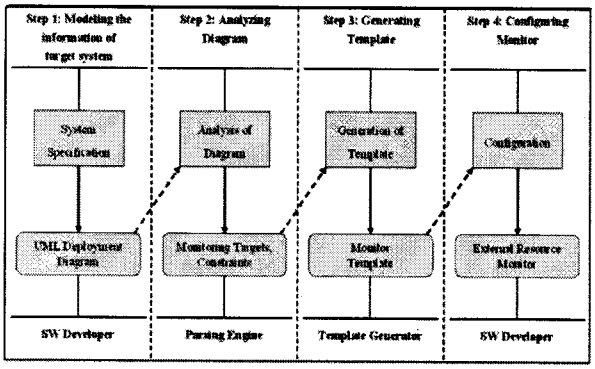

그림 2 모니터 자동 생성을 위한 프로세스 (4-steps)

3.1.2.1 Step1: UML 배치 다이어그램을 통한 시스템 명세화 단계

소프트웨어 개발자는 외부 자원 모니터의 자동화를 위해서 우선 시스템의 배치 다이어그램(그림 3)을 작성하여야 한다. 배치 다이어그램이란 UML 설계 모델 중 시스템의 정적인 모습을 나타내는 다이어그램을 의미하며, 컴포넌트 간의 연결 관계를 도형으로 나타낸 모델[5]이다. 본 제안 방법론이 파싱하는 제약 사항은 표 1과 같다. Method는 네트워크 연결이나 물리적 장치의 연결 방법을 의미하며, 비정상적 연결 종료를 탐지하고자 할 때 사용한다. 제안 방법론에서는 네트워크 연결 탐지에 대한 코드 자동 생성을 수행한다. Duration은 오류 판정을 내리기까지의 지속 시간으로, 일시적인 현상에 의한 상태를 오류로 잘못 판정하지 않기 위한 대기 시간을 말한다. 지정된 시간동안 지속적으로 제약 사항을 위반한다면 해당 자원이나 장치가 오류 상황에 있다고 최종 결정을 내린다. 제안 방법론에서 설정한 기본 값은 1초이다. 이러한 제약 사항은 본 연구에서 실험을 위해 설정한 것으로 향후 변경이 가능하다.

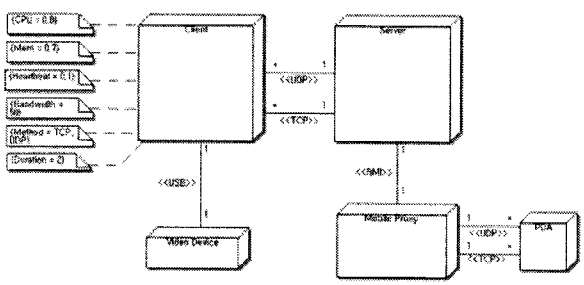

그림 3 외부 환경을 위한 배치 다이어그램 예제

3.1.2.2 Step2: 다이어그램 분석 단계

배치 다이어그램에 나타나있는 정보를 이용하여 두 가지 부류의 자원 유형을 구분한다. 우선, 시스템 내에 존재하는 외적 환경(예를 들어, Client, Server 등등)들을 식별한다. 다음으로 CPU, Memory, Bandwidth, Heartbeat rate 등과 같은 제약 사항들을 식별한다. Parsing Engine 은 배치 모델로부터 생성된 XMI 정보

표 1 제약 사항 목록

항 목	입 력 값	비 고
CPU	0.0~1.0 사이의 한계 점유율	퍼센트
Mem	0.0~1.0 사이의 한계 사용률	퍼센트
Heartbeat	0.1~1.0 사이의 오류 탐지 간격	초
Bandwidth	사용자 지정 최소 대역폭	KB/s
Method	사용자 지정 연결방식	
Duration	오류 판정까지의 지속 시간	초

(그림 4의 왼쪽 참조)를 파싱하여 두 분류의 유형에 대한 내용을 XML (그림 4의 오른쪽 참조)로 규칙화하여 생성한다.

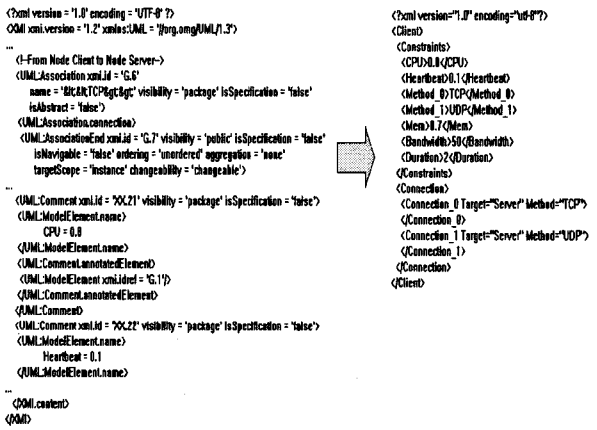

그림 4 배치 다이어그램으로부터 추출된 XMI[6] 정보와 XML 기반 제약 조건 분석모델

3.1.2.3 Step3: 모니터 템플릿 생성 단계

XMI 파서에 의해 분석된 정보들을 이용하여 실제 외부 자원 모니터의 템플릿을 생성한다. Template Generator (TG)는 Parsing Engine 이 생성한 XML 문서를 분석하여 모니터 모듈의 생성을 수행하며, 각 제약 조건별 이상 탐지 루틴과 오류 처리 루틴을 작성한다. 제약 조건별 이상 탐지 방법은 표 2와 같다.

표 2 각 제약 조건별 이상 탐지 방법

제약 조건	이상 탐지 방법
CPU	지정된 한계 점유율 초과시
Memory	지정된 한계 사용률 초과시
Bandwidth	지정된 최소 대역폭 미만시
Method	지정된 연결의 비정상 종료시

TG는 지정된 제약 조건에 위배되는 상황이 발생할 경우 이를 처리하여주는 루틴을 삽입한다. 하지만 본 연구에서는 오류 상황 발생 시의 처리 방법을 자동화 할 수 없었기 때문에, 다음 단계인 모니터 구성 단계에서 개발자가 직접 오류 처리 루틴을 작성하여야 한다. 또한 TG는 클라이언트/서버 개념과 같이 IP나 포트번호와 같은 상세한 정보를 알 수 없어 자동으로 탐지 모듈을 생성할 수 없는 경우에는 개발자가 직접 구현할 수 있도록 주석으로 가이드라인을 제공한다. TG에는 각 제약 조건 별로 코드를 생성하여주는 생성자들이 존재하며(표 3), 각 코드 생성자들이 작성한 결과물을 조합하여 최종 모니터 템플릿(그림 5)을 생성하게 된다.

표 3 각 제약 조건에 따른 코드 생성자

제약 조건	코드 생성자
CPU	WriteCode_CPU()
Memory	WriteCode_Memory()
Bandwidth	WriteCode_Bandwidth()
Method	WriteCode_Method()
Heartbeat	WriteCode_Heartbeat()

```
namespace // Please type namespace here
{
  class ResourceMonitor
  {
    // Monitor controller module
    private Timer heartbeatTimer = new Timer();

    /// <summary>
    /// Initialize heartbeat timer
    /// </summary>
    public ResourceMonitor()
    {

      heartbeatTimer.Tick += new EventHandler(HeartbeatProcess);
      heartbeatTimer.Interval = 100;
      heartbeatTimer.Start();
    }

    /*
     * CPU monitoring variables and routines
     */
    const float fCPUlimit = 80; // CPU Constraints
    public float fCPUusage = 0; // Monitored CPU usage
    private CounterSample CPUgauge_pre; // First CPU usage measurement
    private CounterSample CPUgauge_after; // Second CPU usage measurement

    ...

    /// <summary>
    /// Heartbeat timer process
    /// </summary>
    private void HeartbeatProcess(Object timerObject, EventArgs timerEventArgs)
    {

      /*
       * CPU usage check routine
       */
      PerformanceCounter objCPU = new PerformanceCounter("Processor", "% Processor Time", "_Total");
      CPUgauge_pre = objCPU.NextSample(); // Get CPU usage

      if (CPUgauge_after != null)
        fCPUusage = System.Diagnostics.CounterSample.Calculate(CPUgauge_after, CPUgauge_pre); // Calculate CPU usage

      CPUgauge_after = CPUgauge_pre; // Save previous CPU usage to calculate it next time

      if (fCPUusage > fCPUlimit) // If CPU usage exceeds the limit?
        ExceedCPUlimit(); // Call error handler routine
```

그림 5 Template Generator가 생성한 코드 예제 (C#)

3.1.2.4 Step4: 모니터 구성 단계

모니터 자동 생성 기법의 마지막 단계로서 개발자가 소프트웨어의 환경에 따라 외부 자원 모니터를 수정하는 단계이다. 모니터 템플릿 생성 단계에서 언급된 오류 처리 핸들러 혹은 주석처리 된 가이드라인을 실제로 구현하며, 필요하거나 수정되어야 하는 부분에 대한 커스터마이징(Customizing) 작업을 수행한다. 이후 최종적으로 치유 계층 혹은 컴포넌트에 생성된 외부 자원 모니터를 등록한다.

3.2 외부 장치 및 자원의 오류 탐지 기술

본 절에서는 관련 연구에서 언급된 자율 고장-탐지 알고리즘을 제안 방법론에 맞게 변경한 부분(그림 6)에 대하여 설명한다. 치유 계층의 외부 자원 모니터는 모니터링 장치에게 보낸 신호나 특정 자원의 값을 요청하는 신호에 대한 응답이 돌아오지 않는다면, 이는 해당 자원이 이상 상태에 놓여 있다고 판단한다. 만약 외부 자원의 값이 제약 사항을 위반한다면 해당 자원을 이용하기에 적절하지 않다고 판단한다. 이러한 상황은 치유 계층으로 하여금 재구성 전략을 수립하여 수행하게 만든다. 기존의 연구와 다른 점은, 모니터가 먼저 모니터링 되는 자원에 신호를 보내기 때문에 최대 이상 탐지 시간(L_{MAX})과 평균 이상 탐지 시간(L_{AVG})이 1.5배 길어졌다는 것이다. 만약 모니터링 되는 자원이 모니터에게 응답을 한 직후 오류가 발생하였다면 모니터는 외부 자원이 아직 정상 상태에 있다고 판단하여, 새로운 주기를 알리는 신호를 보낼 것이다. 하지만 외부 자원은 이미 오류 상태에 빠져있으므로 이를 탐지하기 위해서 한 번의 주기가 낭비되게 된다. 이러한 상황으로 인하여 기존 연구에 비해 이상 탐지 시간에 차이가 발생한다.

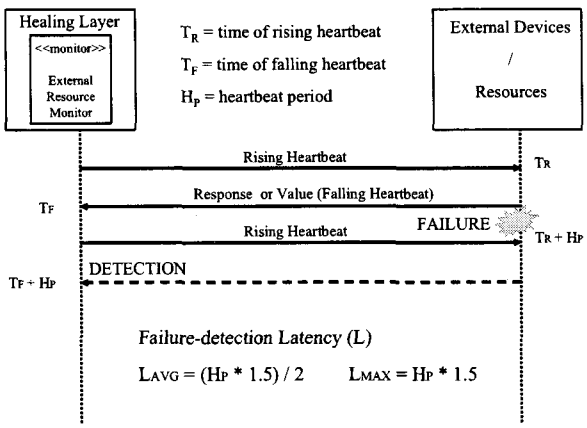

그림 6 외부 장치 및 자원의 오류 상황 탐지 기술

3.3 모니터링 기능이 포함된 자가 치유 컴포넌트

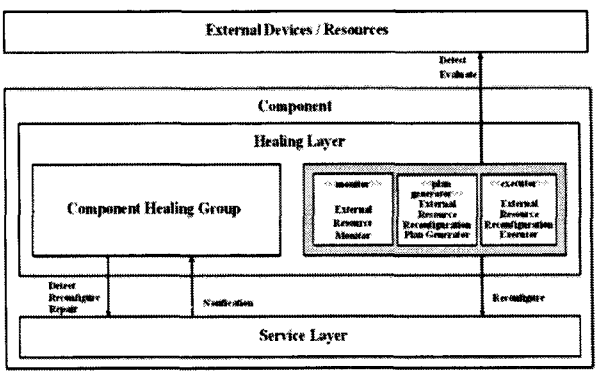

그림 7 모니터링 가능한 자가 치유 컴포넌트 구조

그림 7은 관련연구[2,3]을 확장한 모니터링 가능한 자가 치유 컴포넌트 구조이다. 제안 구조에는 관련 연구에서 언급된 컴포넌트 치유를 위한 여섯 개의 모듈 이외에 외부 자원 탐지 및 재구성을 위한 세 가지 모듈이 추가되었다. 추가된 모듈은 외부 장치 및 자원 상황을 모니터링하는 1) 외부 자원 모니터(External Resource Monitor) 와 외부 상황에 맞추어 서비스 계층의 재구성 전략을 수립하는 2) 외부 자원 재구성 전략 생성기(External Resource Reconfiguration Plan Generator), 그리고 해당 재구성 전략을 실행하는 3) 외부 자원 재구성 실행기(External Resource Reconfiguration Executor)이다.

이중에서 외부 자원 재구성 전략 생성기는 [2]에 언급되어있는 컴포넌트 재구성 전략 방식과 유사하게, 외부 자원으로부터 영향받는 객체를 치유하기 위하여 이를 고립시킴으로써 다른 정상적인 객체들이 외부 자원으로 인해 영향을 받지 않도록 하는 전략을 수립하는 것이 목적이다. 치유 계층 내의 객체들을 통제하는 Self-healing Controller 는 외부 자원 재구성 실행기가 미리 수립된 재구성 전략을 이용하여 서비스 계층의 재구성을 수행하도록 제어한다. 외부 오류에 대하여 [2]에 제안된 방식과 동일한 재구성을 수행하며, 외부 자원으로 인한 서비스 계층의 이상을 최소화 할 수 있도록 한다.

4. 구현 및 평가

본 논문의 평가를 위해 간단한 비디오 회의 시스템을 구현하였고, 이것의 기본 설계는 UML로 표현하였다. 본 시스템의 목적은 성공적인 비디오 회의의 수행이다. 회의 수행 기간 동안 클라이언트 사용자들은 소프트웨어 외적인 문제(CPU, Memory, Network 등)로 인한 방해를 받지 않아야 한다. 본 논문에서는 제안 방법론을 통하여 외부 자원 모니터를 자동 생성한 후 비디오 회의 시스템의 클라이언트에 적용하여, 소프트웨어 외적인 문제로 인해 발생한 오류를 탐지할 수 있는지 확인하는 것을 평가 목표로 한다.

4.1 구성 환경

본 논문에서 제안한 외부 장치 모니터 자동 생성 기법을 평가하기 위해, MS Windows XP에서 C# 언어를 이용하여 .NET Framework 2.0 기반으로 비디오 회의 시스템의 클라이언트를 구현하였으며, 시스템의 UML 모델링은 Borland 사의 Together를 이용하였다. 해당 시스템의 서버는 Java2 SDK 1.4를 이용하여 구현하였다. 클라이언트는 비디오 장치를 다루기 위해 DirectShow.NET 라이브러리를 추가로 이용한다. 배치 다이어그램 분석기와 외부 장치 모니터 템플릿 생성기 또한 C#을 이용하여 구현하였다. 평가를 위하여 사용한 배치 다이어그램은 그림 3과 같으며, 이를 입력한 Parsing Engine 프로토타입(그림 8)의 결과와 Template Generator 프로토타입(그림 9)의 결과는 다음과 같다.

4.2 정상적인 경우

모니터를 통하여 측정된 외부 자원의 수치가 정상 범위 내에 있거나 이상 없이 작동하는 경우, 외부 자원 모니터는 특별한 조치를 취하지 않으며 계속적으로 모니터링을 수행한다.

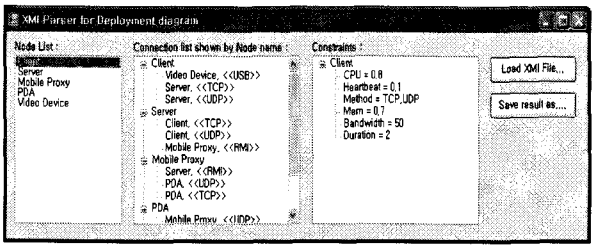

그림 8 Parsing Engine의 실행화면 프로토타입

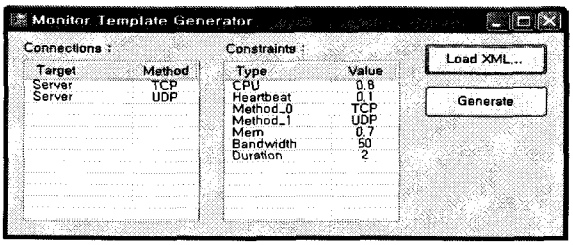

그림 9 Template Generator의 실행화면

4.3 비정상적인 경우

만약 측정된 값이 정상 범위를 벗어났거나 외부 장치와의 연결이 갑작스레 종료된 경우라면 모니터는 이를 탐지한다. 그림 10은 CPU 사용률이 허용 한계치 80%를 초과한 상황을 탐지한 경우이다. 하지만 본 논문의 제안 방법론은 비정상적인 경우를 탐지하는 모니터를 자동으로 생성하는 역할이 전부이며 별도로 자가 치유 전략을 수립하지는 않는다. 따라서 위에서 발생한 오류 상황을 치유하기 위한 전략은 개발자가 직접 수립해야하며, 본 논문에서 제안한 아키텍처(그림 6)는 이러한 자가 치유 전략을 수립하고 실행 할 수 있다.

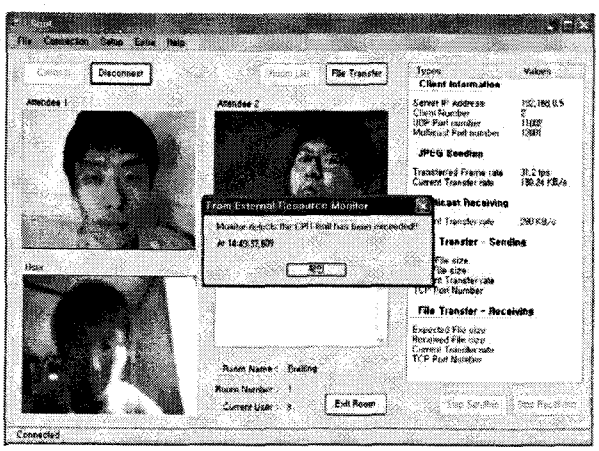

그림 10 모니터가 CPU의 이상 상황을 탐지한 경우

4.4 평가 목적 및 평가 결과

평가의 목적은 제안 방법론이 적용된 목표 시스템과 적용되지 않은 목표 시스템의 비교를 통하여, 외부 자원에서 이상이 발생하였을 경우 제안 방법론이 적용된 목표 시스템에서 지정된 시간 내에 이상 상황을 인지하는지 확인하는 것이다. 시스템의 비정상적인 상황을 시뮬레이션하기 위하여 벤치마킹 프로그램(Prime 2004, 3DMark 등)과 서버 강제 종료 등의 방법을 사용하였다. 또한 이상 탐지 시간(Failure-Detection Latency) 평가의 정확성을 위하여 클라이언트에 이상 상태 발생 즉시 시간을 보고하는 루틴과 외부 자원 모니터에 이상 탐지 시간 출력 루틴을 추가하였다. 제약 사항들의 이상 상태 탐지 여부를 확인한 결과는 표 4와 같으며 CPU를 대상으로 10회 반복 측정한 이상 탐지 시간의 평가 결과는 그림 11과 같다.

표 4 제약 사항들의 이상 상태 탐지 여부

항 목	제약 사항	탐지 여부
CPU 점유율	최대 80%	탐 지
Memory 사용률	최대 70%	탐 지
Bandwidth	최소 50KB/s	탐 지
네트워크 연결 유지	비정상적인 네트워크 연결종료	탐 지

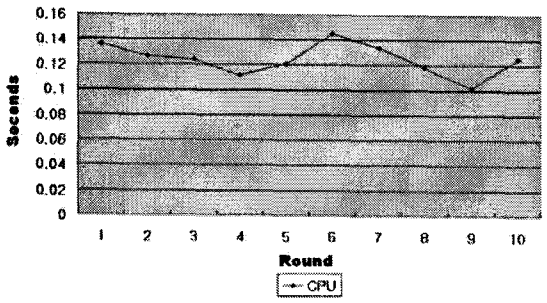

그림 11 외부 자원 모니터의 오류 탐지 시간

평가 결과 외부 자원 모니터는 제약 사항으로 설정된 4개 항목에 대하여 이상 발생 시 정확히 탐지 하였으며, 평균 이상 탐지 시간(L_{AVG} = (0.1 * 1.5) / 2 = 0.075(s))과 다소 차이가 있었지만 최대 이상 탐지 시간(L_{MAX} = 0.15(s)) 이내에 탐지함을 확인할 수 있었다.

5. 결론 및 향후 연구

본 연구에서는 UML의 배치 다이어그램을 이용해 외부 자원 모니터의 생성을 자동화하여 자가 치유 개발자의 노력을 줄이기 위한 방법론과 외부 자원의 탐지가 가능한 소프트웨어 아키텍처를 제안하였다. 이를 통해 얻을 수 있는 이점은 다음과 같다.

- 외부 자원 모니터 생성의 자동화를 통해 자가 치유 개발자의 부하를 줄인다.
- 외부 자원 오류에 대응하기 위한 전략을 수립하는 데 용이하다.
- 자가 치유를 요구하는 목표시스템의 신뢰성을 높이는 데 기여할 수 있다.

현재까지 자가 치유 기술을 구현하기 위해서는 개발자가 모든 자가 치유 모듈 및 전략을 생성해야만 하는 한계가 있었다. 하지만 본 논문에서 우리는 자가 치유 컴포넌트에 내장하기 위한 외부 자원 모니터는 배치 다이어그램으로부터 자동으로 생성할 수 있음을 확인하였다. 또한, 평가를 위해 프로토타입 시스템을 통하여 비정상적인 상황에서 모니터의 감지 활동이 올바르게 동작함을 확인하였다.

그러나 관련 연구의 단점으로 지적되었던 오류 탐지를 위한 잦은 신호 교환 오버헤드는 여전히 극복하지 못하였다. 이를 개선하기 위해서 모니터의 신호 교환 주기를 자율적으로 결정할 수 있도록 하는 연구가 필요하다. 또한 자가 치유 개발자의 최소의 노력을 통해 이상 상황으로부터 회복하는 자가 치유 전략과 그것의 우선순위 결정을 자동화하는 연구는 향후 과제를 통해 확장할 것이다.

참 고 문 헌

[1] D. Ghosh, R. Sharman, H. R. Rao, S. Upadhyaya, "Self-healing - survey and synthesis," Decision Support Systems in Emerging Economies, Vol. 42, Issue 4, pp.2164-2185, Jan. 2007.

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

[3] Michael E. Shin and Jung Hoon An, "Self-Reconfiguration in Self-Healing Systems," Proceedings of the Third IEEE International Workshop on EASE'06, pp.89-98, 2006.

[4] K. Mills, S. Rose, S. Quirolgico, M. Britton, C. Tan, "An autonomic failure-detection algorithm," ACM SIGSOFT Software Engineering Notes, Vol. 29, Issue 1, pp.79-83, 2004.

[5] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide," Addison Wesley, Reading MA, 1999.

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