외장형 자가 적응 시스템의 성능 개선을 위한 제어 모듈의 자동 생성

서동영, 박정민, 이은석
성균관대학교 컴퓨터공학과
e-mail : {shadok, jmpark ,eslee}@ece.skku.ac.kr

Autonomic Generation of Control Module for Performance of Externalized Self-Adaptation System

Dongyoung Seo, Jeongmin Park, Eunseok Lee
Dept. of Computer Engineering, SungKyunkwan University

요 약

기존에 연구된 자가 적응 시스템은 하위 레벨에서 서로가 연관되어 있는 경우가 많기 때문에 분석, 변경, 재사용이 어렵다는 단점을 갖고 있었다. 이러한 문제점을 해결하기 위해 제안된 시스템이 외장형 자가 적응 시스템이다. 하지만 외장형 자가 적응 시스템은 probe, gauge 등 다수의 컴포넌트가 동시에 동작하기 때문에 시스템의 복잡도를 증가시키는 한계가 있다. 따라서 본 논문은 불필요한 컴포넌트의 사용으로 인한 리소스 낭비를 방지하기 위한 제어 모듈을 자동으로 생성하는 방법론은 제시하고, 이것을 적용한 소프트웨어 아키텍처를 제안한다. 이러한 제안 방법론을 통해 기존에 필요 여부와 관계없이 동시에 동작하던 컴포넌트의 실행을 효율적으로 관리해 시스템의 복잡도를 감소시킬 수 있게 된다. 본 논문에서는 평가를 위해 제안 방법론을 웹 서버에 적용하여, 일반적인 경우와 제어모듈이 생성되어 작동할 때의 성능을 비교하였다.

1. 서론

기존의 소프트웨어 공학에서 행해진 연구는 시스템이 오프라인 상에서 만들어지고, 수정된다는 가정 하에 이루어졌다. 그러나 최근의 시스템에는 변화하는 사용자 요구와 외부 자원 상황 하에서도 지속적으로 작동하는 것이 요구된다. 이를 해결하기 위해 시스템 스스로 오류를 탐지하고, 실행 중에 이를 복구할 수 있는 능력을 가진 자가 적응 시스템에 관한 연구가 활발히 진행되고 있다[1]. 하지만 자가 적응 기능을 대상 시스템에 내부적으로 구현할 경우 적응 시스템과 대상 시스템이 서로 엉켜져 적응 시스템의 구조 변경이 어렵게 되고, 비용도 많이 들게 된다. 또한 비슷한 이유로 인해 한 시스템에서 사용하던 적응 시스템을 다른 시스템에서 사용하기 힘들어진다. 이에 대한 대안으로 외장형 구조[2,3]를 사용하여 자가 적응 시스템을 구현하는 방법이 있는데, 이 구조는 적응 시스템이 대상 시스템과 분리되어 있기 때문에 대상 시스템의 수정을 초래하지 않고, 변경과 확장이 쉬워진다는 장점을 갖고 있다.

외장형 적응 시스템에는 실행중인 대상 시스템을 감시하고, 이 데이터를 분석, 해석한 다음 대상 시스템이 재구성이 필요한 상태인지 결정해 이를 실행하기 위해 다음과 같은 컴포넌트가 존재한다. Probe는 대상 시스템을 감시해 이벤트를 발생시킨다. Gauge는 이러한 이벤트를 시스템 모델에 맞게 분석하고 해석한다. Controller는 분석된 자료를 토대로 대상 시스템에 재구성이 필요한지를 판단한다. Effecor는 이 결정을 실제로 수행한다. 그러나 이러한 컴포넌트들이 필요 여부에 관계없이 동시에 동작하기 때문에 시스템의 복잡도를 증가시키게 된다[4].

이에 대한 대안으로 각 기능 별로 실행되는 컴포넌트를 파악해 컴포넌트의 동작을 ON/OFF할 수 있는 기능을 가진 스위치를 사용해 시스템의 복잡도를 감소할 수 있는 방안이 제안되었다[4]. 그러나 이러한 방법은 관리자가 시스템의 기능을 직접 모델링 해 수동으로 작성해야하기 때문에 많은 시간과 비용의 발생을 초래하며, 컴포넌트의 수가 증가할 수록 이 비용은 더욱 더 증가한다. 또한 모델링이 잘못 되었을 경우 시스템이 정상적으로 동작하지 않을 위험이 존재한다.

본 논문에서는 이러한 한계를 해결하기 위해 컴포넌트의 실행을 제어할 수 있는 모듈을 자동으로 생성하는 방법을 제안한다. 또한 제안 방법론을 스레드 풀 기능을 가진 웹서버에 적용하여, 일반적인 경우와 제어 모듈이 생성되어 작동했을 때의 성능을 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구들을 분석하여 설명하였으며, 3장에서는 제어 모듈의 자동 생성을 위한 방법론을 제안한다. 4장에서는 해당 방법론을 적용한 웹서버의 평가 결과를 소개하고, 마지막으로 5장에서는 결론 및 향후 과제를 기술한다.

2. 관련 연구

본 장에서는 자가 적응 시스템을 위한 외장형 접근 방법인 David S. Wile[2], David Garlan[3]이 제시한 제안 시스템들을 살펴보고, 그에 대한 특징과 장단점을 기술한다.

첫 번째로, David S. Wile이 제안한 Idealized Safe Email Infrastructure Architecture[2]이다. 이 아키텍쳐는 Probe, Gauge, Controller, Effector 4개의 계층으로 구성되어있다. Probe는 대상 시스템을 감시해 얻은 정보를 Probe Bus에 전달한다. Gauge들은 이 정보를 아키텍쳐 모델에 맞게 해석하고, 상위 계층에서 이용할 수 있도록 변형해 Gauge bus에 전달한다. Controller들은 Gauge bus로 부터의 정보를 가지고 통제결정을 내리고 이것을 Effector들에게 전달하면 Effector들은 대상 시스템에 영향을 끼친다.

두 번째로 David Garlan이 제안한 Gauge Infrastructure[3]이다. 이 구조는 Probe, Gauge, Gauge Consumer 3개의 계층으로 이루어져 있다. Probe는 대상 시스템에 배치되어 시스템을 감시하고 발생한 정보를 Probe bus에 전달한다. Gauge는 Probe bus로부터 얻은 자료를 상위 단계의 모델 속성에 맞추어 번역하고 이 정보를 사용하거나, Gauge reporting bus를 통해 정보를 상위 계층에 전달한다. Gauge Consumer들로 이루어진 최상위 계층은 Gauge로부터 얻어진 정보를 사용한다.

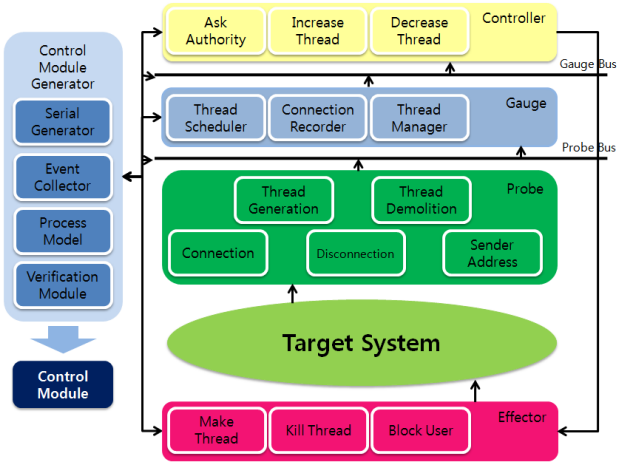
위의 두 시스템은 외장형 자가 적응 구조를 이용하여 적응 시스템과 대상 시스템을 분리한 결과, 내장형 구조의 문제점이었던 적응 시스템과 대상 시스템이 서로 엉켜져 적응 시스템 구조의 변경이 어려워지고, 다른 시스템으로의 이식이 어려웠던 문제를 해결하였다.

그러나 두 시스템 모두 대상 시스템 외에 다수의 컴포넌트들이 동시에 동작함으로써 시스템의 복잡도가 증가한다는 한계가 있다. 특히 외부 자원 환경에 제약이 많은 휴대용 장치에서 실행될 경우 위의 문제점은 더욱 심각해질 것이다.

위와 같은 문제점을 해결하기 위해, 본 논문에서는 외장형 적응 시스템 내에 존재하는 컴포넌트의 동작을 감시해 그에 따라 컴포넌트를 제어할 수 있는 모듈을 자동으로 생성하는 방법으로 위의 문제를 해결한다.

3. 제안시스템

본 장에서는 위에서 제시된 단점을 해결하기 위해 제어 모듈 생성의 자동화를 위한 기법을 설명하겠다. 그림 1은 본 논문에서 제안한 “스레드 풀링 웹서버”의 아키텍쳐로서, 이 시스템은 일정한 수의 스레드를 미리 생성해 사용자가 접속했을 경우 대기 중인 스레드 중 하나를 통해 서비스를 제공해 사용자가 접속할 때마다 스레드를 생성함으로 인해 응답이 지연되는 것을 방지한다. 또한 생성된 스레드에 비해 대기 중인 스레드가 일정한 비율에 미치지 못하거나, 일정한 비율을 초과할 경우 스레드를 생성, 종료시키고, 특정 주소로부터 과도한 트래픽이 발생할 경우 사용자에게 해당 주소로부터의 접근을 차단할지를 결정해 목표 시스템에 영향을 준다.



(그림 1) 스레드 풀링 웹서버의 아키텍쳐

3.1. 동작 시나리오

이 시스템의 행동을 시나리오를 통해 살펴보겠다. 사용자가 접속했을 경우 사용자에게 서비스를 제공하는 스레드를 알아내고(Connection), 이 스레드가 실행중이라는 것을 알린 뒤(Thread Manager) 다음 사용자에게 서비스할 스레드를 결정한다(Thread Scheduler). 그리고 대기 중인

<표1> 컴포넌트의 이름과 역할

타 입	이 름	역 할
Probe	Connection	접속한 사용자에게 서비스하는 스레드를 알아낸다.
	Disconnection	접속을 종료한 사용자게에 서비스했던 스레드를 알아낸다.
	Thread Generation	스레드가 생성되었음을 알아낸다.
	Thread Demolition	스레드가 종료되었음을 알아낸다.
	Sender Address	접속한 사용자의 주소를 알아낸다.
Gauge	Thread Manager	현재 생성되어 있는 스레드 중 실행, 대기 중인 스레드를 파악한다.
	Thread Scheduler	다음 사용자에게 서비스할 스레드를 결정한다.
	Connection Recorder	최근 5초 동안의 접속자에 대한 기록을 관리한다.
Controller	Ask Authority	특정 주소로부터의 접근을 차단할 것인지 결정한다.
	Increase Thread	스레드를 추가로 생성할지 결정한다.
	Decrease Thread	생성되어 있는 스레드를 종료할지 결정한다.
Effector	Make Thread	스레드를 생성한다.
	Kill Thread	스레드를 종료한다.
	Block User	특정 주소로부터의 접근을 일정 시간 동안 차단한다.

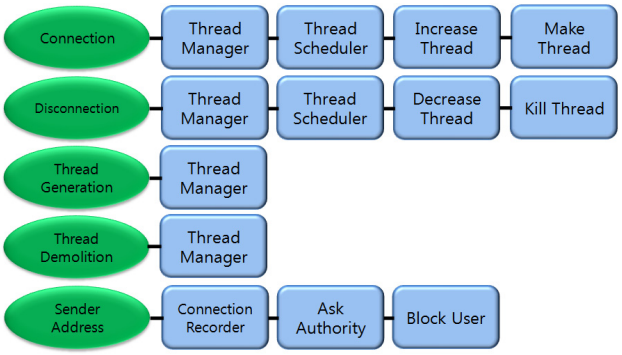
스레드의 비율을 고려해 스레드를 더 생성해야 하는지를 결정하고(Increase Thread), 결정에 대한 행동을 취한다

(Make Thread). 또한 접속한 사용자의 주소를 파악해(Sender Address) 특정 주소로부터 과도한 트래픽이 발생하는지 분석한 뒤(Connection Record) 만약 그렇다면 그 주소로부터의 접근을 차단할 것인지 결정해(Ask Authority) 결정에 대한 행동을 취한다(Block User). 사용자가 접속을 종료했을 경우 사용자에게 서비스를 제공한 스레드를 알아내고(Disconnection), 이 스레드가 대기중이라는 것을 알린 뒤(Thread Manager) 대기 중인 스레드의 비율을 고려해 스레드의 수를 감소시켜야하는지를 결정하고(Decrease Thread), 결정에 대한 행동을 취한다(Kill Thread). 스레드가 생성, 종료되었을 경우 이를 탐지해(Thread Generation, Thread Demolition) 해당 스레드가 생성 혹은 종료되었음을 알린다(Thread Manager).

3.2. 제어 모듈의 생성

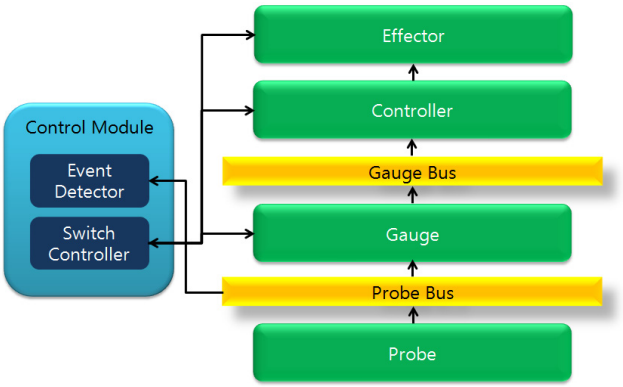
Probe와 Gauge로부터 발생된 이벤트는 Probe Bus와 Gauge Bus에 전달되는데, Control Module Generator(이하 제어 모듈 생성기)에서는 이 이벤트에 일련번호를 부여하고(Serial Generator), 이후 이 이벤트가 처리되는 과정을 추적해(Event Collector) 각 Probe가 발생시킨 이벤트에 대한 처리 모델을 만들고 이를 동적으로 유지하며(Process Model), 이벤트가 발생할 때마다 이 모델이 올바른지 검증한다(Verification Module). 일정 수의 이벤트에 대해 이벤트 처리 모델이 올바르다는 것이 검증되면 제어 모듈 생성기에서 Control Module(이하 제어 모듈)을 생성해 이를 통해 컴포넌트들의 동작을 제어한다.

- Serial Generator : Probe Bus와 Gauge Bus에 전달되는 이벤트를 중간에 가로채 일련번호를 부여한 후 이를 다시 Probe Bus 혹은 Gauge Bus에 전달한다.
- Event Collector : 이벤트를 발생시킨 컴포넌트와, 해당 이벤트를 처리하는 컴포넌트를 파악해 Process Model과 Verification Module에 이 정보를 전달한다.
- Process Model : Event Collector로부터 받은 정보를 토대로 각 Probe로부터 발생된 이벤트가 어떠한 컴포넌트를 통해 처리되는지 리스트를 통해 동적으로 관리한다.



(그림 2) Process Model의 예제

- Verification Module : Event Collector로부터 받은 정보를 토대로 현재의 Process Model이 올바른지 검증하고, 지정된 개수의 이벤트가 Process Model에 맞게 처리되면 제어 모듈 생성기에 제어 모듈의 생성을 요청한다.
- Control Module Generator : Verification Module로부터 제어 모듈의 생성이 요청되면, Process Model과 Event Collector를 통합, 변경해 제어 모듈을 생성한다.



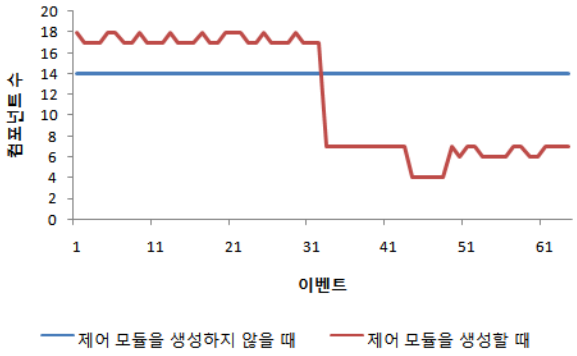
(그림 3) 제어 모듈 생성 이후의 적응 시스템 아키텍처

제어 모듈이 생성되면 제어 모듈 생성기는 실행을 종료한다. 만약, 제어 모듈이 생성될 때까지 한 번도 실행되지 않은 컴포넌트가 존재한다면 그 컴포넌트의 동작은 제어하지 않는다. 그림 3은 제어 모듈이 생성되었을 때의 시스템 구성을 나타낸 것이다. 제어 모듈은 크게 두 부분으로 구성된다.

- Event Detector : Probe Bus에 전달되는 이벤트를 통해 어느 Probe로부터 이벤트가 발생했는지 파악한다.
- Switch Controller : Process Model을 바탕으로 이벤트를 발생시킨 Probe에 따라 컴포넌트의 동작을 정지하거나 재개한다.

4. 구현 및 평가

본 논문의 평가를 위해 MS Windows XP에서 Java2 SDK1.4.2를 기반으로 프로토타입을 구현하였다. 대상 시스템은 사용자가 URL을 요청할 경우 해당하는 페이지를 전송하는 웹서버로 구현하였으며, 제안 시스템의 각 컴포넌트는 자바의 스레드 방식을 이용해 구현하였다. 평가는 제어 모듈을 생성하지 않는 경우와 제어 모듈을 생성해 실행될 때의 동작 컴포넌트 수를 비교하는 방식으로 수행



(그림 4) 일반 시스템과 제안 시스템의 성능 비교

하였다. 그림 4는 제어 모듈을 생성하지 않는 경우와 제어 모듈을 생성할 경우 동작하는 컴포넌트의 수를 나타낸 것이다. 제어 모듈이 생성되기 이전에는 제어 모듈 생성기가 작동하기 때문에 일반적인 경우보다 3~4개의 컴포넌트가 추가로 동작해야 하지만 제어 모듈이 생성된 이후에는 동작하는 컴포넌트의 수가 50% 이상 감소됨을 확인할 수 있었다.

또한 제어 모듈을 수동으로 작성하는 방법과 비교해 기능 별로 동작해야 하는 컴포넌트 외에 제어 모듈의 두 개의 컴포넌트가 추가적으로 동작하기 때문에 약간의 오버헤드가 따르지만 시스템이 복잡해질수록 개발자의 부하가 기하급수적으로 증가한다는 것을 감안할 때 제어 모듈을 자동으로 생성하는 방법이 보다 효과적이라는 것을 알 수 있었다.

5. 결론 및 향후 연구

본 연구에서는 이벤트의 처리 과정을 추적해 기능에 따라 컴포넌트의 동작을 감시하고, 그에 따라 컴포넌트를 제어할 수 있는 모듈을 자동으로 생성하는 방법을 제안하였다. 이를 통해 얻을 수 있는 이점은 다음과 같다.

- 동시에 동작하는 컴포넌트의 수를 줄여 리소스의 낭비를 줄인다.
- 기존에 수동으로 작성해야 했던 제어 모듈을 자동으로 생성해 관리자/개발자의 부하를 줄인다.
- 적응 시스템과 제어 모듈 생성기가 분리되어 있기 때문에 약간의 수정만 거치면 기존에 존재하는 외장형 자가 적응 시스템에 쉽게 이식할 수 있다.

현재까지 외장형 자가 적응 시스템은 대상 시스템 외에 별도로 실행되는 컴포넌트가 다수 존재하기 때문에 시스템의 복잡도가 증가하고, 이를 해결하기 위해서는 제어 모듈을 수동으로 작성해야 하는 문제점이 있었다. 하지만 본 논문에서 우리는 외장형 자가 적응 시스템에서 동시에 동작하는 컴포넌트의 수를 제어할 수 있는 모듈을 자동으로 생성할 수 있음을 확인하였다. 또한, 평가를 위해 프로토타입 시스템을 통하여 제어 모듈이 올바르게 작동함을 확인하였다.

그러나 시스템의 실행 초기에 적응 시스템의 컴포넌트 외에 추가로 작동해야 하는 컴포넌트가 존재하기 때문에 일반적인 경우보다 시스템의 오버헤드가 더 크다는 한계가 존재한다. 또한 만약 제어 모듈이 생성될 때까지 한 번도 실행되지 않은 기능은 감지해내지 못한다는 것과 이후 적응 시스템의 구조가 변경되었을 경우에는 제어 모듈을 다시 생성해야 한다는 단점이 있다. 이를 개선하기 위해서는 대상 시스템으로부터 발생되는 이벤트만 추적하는 것이 아니라 제어 모듈 생성기 스스로 가능한 모든 타입의 이벤트를 발생시켜 이의 처리 과정을 추적하도록 하는 연구가 필요하다. 제어 모듈 생성을 위한 컴포넌트의 수를 최소화하고 이벤트를 직접 만들어내는 연구는 향후 연구를 통해 확장할 것이다.

참고문헌

[1] David Garlan and Bradley Schmerl "Model-based Adaptation for Self-Healing Systems", Proceedings of the Workshop on Self-Healing Systems (WOSS '02), Nov. 2002

[2] David S. Wile and Alexander Egyed "An Externalized Infrastructure for Self-Healing Systems", IEEE/IFIP Conference on Software Architecture (WICSA '04), Apr. 2004

[3] David Garlan, Bradley Schmerl, and Jichuan Chang "Using Gauges for Architecture-Based Monitoring and Adaptation", Proceedings of the Working Conference on Complex and Dynamic Systems Architecture, Dec. 2001

[4] 최윤규, "외장형 자가적응 인프라를 위한 제어 모듈의 성능개선", 제 26회 한국 정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호, Nov. 2006