UML 을 이용한 Heart-beat 기반 모니터링 코드 자동 생성 기법[1]

이준훈, 유길종, 이은석
성균관대학교 전자전기컴퓨터공학과
e-mail : {trsprs, gjyoo, eslee}@ece.skku.ac.kr

Automatic code Generation Technique for Heart-beat Monitoring using UML

Joonhoon Lee, Giljong Yoo, Eunseok Lee
Dept. of Electrical and Computer Engineering, Sungkyunkwan University

요 약

모니터링은 시스템의 문제 상황을 판단하기 위해서 아주 중요한 요소이다. 응답시간과 같은 외부적인 요인들은 쉽게 모니터링이 가능하지만, 시스템의 동작 상태와 같은 내부의 상태를 모니터링 하도록 개발하는 것은 많은 노력이 든다. 본 논문에서는 UML 을 이용하여 HB(Heart-Beat) 모니터링 할 수 있는 모니터링 코드를 생성하는 방법을 제안한다. 본 방법론에서는 AOP 를 이용한 모니터링 코드를 생성하여 기존 시스템의 코드를 수정하지 않고도 모니터링 기능을 추가할 수 있으며, UML 을 기반으로 큰 노력 없이 자동으로 모니터링 코드 생성이 가능하다.

1. 서론

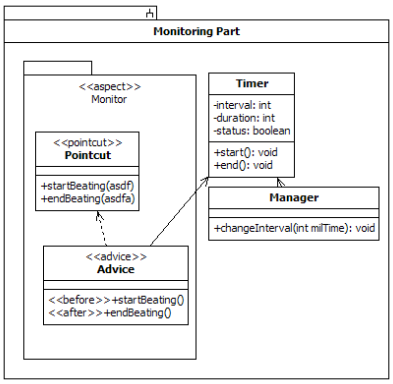
오늘날 시스템 스스로 시스템의 상태를 관리하는 자율컴퓨팅 (Autonomic Computing)이 대두되고 있다. 자율 컴퓨팅은 모니터링, 진단, 컨트롤, 적용의 네 가지 단계로 구성된다. 여기서 모니터링과 적용 단계는 대상 시스템과 직접적인 연결을 필요로 한다. 따라서 개발자나 관리자가 이러한 부분을 직접 설정해야 한다. 이러한 추가적인 개발 부하를 해결하기 위한 여러 가지 연구가 있다 [1, 2, 3]. 하지만 정형적인 모델을 사용하기 때문에 일반 개발자가 접근하기는 쉽지 않다. 본 논문에서는 자율 컴퓨팅 환경을 구성하기 위한 요소 중의 하나인 모니터링에 초점을 맞추어 널리 사용 중인 UML 을 기반으로 모니터링을 위한 소스 코드를 자동으로 생성하는 기법을 제안한다. 이러한 방법을 통해 개발자가 추가로 개발해야 하는 모니터링 기능에 대한 노력을 줄일 수 있다.

2. 관련연구

Steling 의 HB(Heart-Beat) 모니터링은 모니터링 메시지 교환을 줄이기 때문에 모니터링 부하 줄일 수 있는 방법이다 [1]. 자율 컴퓨팅은 이 방법을 이용하여 모니터링 하는 것이 일반적이다. 하지만 이러한 모니터링 코드를 생성하는데 드는 개발자의 추가적인 노력이 필요하다.

JavaMoP [4]는 정형명세를 이용하여 소프트웨어의 내부 상태를 모니터링 하는 방법을 제안한다. 정형명세의 정확한 동작에 따라 코드를 자동으로 생성한다. 하지만 FTLTL(Future Time Linear Time Logic)이나 PTLTL(Past Time Linear Time Logic)과 같은 정형명세는 일반 개발자가 다루기 어려운 문제가 있다.

본 논문에서는 이러한 문제를 해결하기 위하여 AOP(Aspect-Oriented Programming)을 이용하여 일반적으로 사용하는 UML 로부터 모니터링 코드를 생성하여 기존 시스템에 큰 영향을 주지 않고도 모니터링 기능을 추가할 수 있는 방법을 제안한다.



(그림 1) 모니터링 관점 정의

3. 모니터링 방법

본 장에서는 UML 을 이용하여 컴포넌트나 객체의 동

[1] 이 논문은 2009 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. 2009-0077453)

작 여부를 판단하는 모니터링 관점(Aspect) 코드를 생성하는 방법을 제안한다. 여기서의 관점 코드는 AspectJ [5]를 이용한다. 클래스 다이어그램의 클래스에서 모니터링이 필요한 클래스에 스테레오 타입 <<target>>를 부여한다. 스테레오 타입의 부여는 시스템 설계자나 관리자의 판단에 따른다. 스테레오 타입이 부여된 클래스는 모니터링 대상이 되며, 본 방법론을 통해 자동 생성되는 관점 코드와 결합하여 모니터링이 된다.

그림 1 은 AOP 를 이용한 모니터링을 위한 관점 정의이다. 모니터링 대상은 대상 시스템의 부분에 속하며, 포인트컷(Pointcut)에 의해 어드바이스(Advice)가 호출된다. 어드바이스는 Timer 클래스를 객체화시킨 후, 모니터링 관리자인 Manager 객체에 등록을 한다. 이러한 과정을 통해 모니터링 대상의 타이머가 모니터 관리자에게 등록되며, 모니터링 대상은 이 타이머를 이용하여 모니터 관리자에게 신호를 보낸다. 그림 1 의 <<aspect>>가 부여된 패키지 이외의 부분은 이러한 관계를 보여 준다. 모니터 대상이 모니터 관리자에게 신호를 보낸 후, 타이머를 초기화 시킨다. 그 후, 다음 신호를 보낼 때까지 기다리면서 자신의 작업을 수행한다. 이런 과정은 정해진 간격마다 반복된다.

4. 모니터 코드 생성

클래스를 C 로 정의하고, $C_{st}(string)$ 을 string 을 스트레오 타입으로 가지는 클래스라 정의하면, 모니터링 대상이 되는 클래스 C^* 는 수식 (1)과 같다.

$$C^* = \{c_n | c_n = C_{st}(\text{'target'}), n = 1, \ldots, \infty\} \quad (1)$$

<표 1> 포인트컷 생성 알고리즘

```
alg. genMonitorPointcutCode
  input:  C* (C* ⊃ Cn)
  output: pCode {
  caller = '* *.*(..)';
  for each  Cn  {
     write("pointcut " + Nn + "monitor():";
     write("execution");
     write(" (caller, within(' +  Nn  +');";
  }
end.
```

<표 2> 어드바이스 생성 알고리즘

```
alg. genMonitorBodyCode
  input:  C* (C* ⊃ Cn)
  output: bCode {
  for each  Cn  {
     write("Object around():"+ Nn + "monitor() {");
     write("SignalMonitor sm = new SignalMonitor();");
     write("sm.start();");
     write("Object result = proceed();");
     write("sm.terminated();");
     write("return result;}");
  }
end.
```

표 1 은 수식 1 에서 얻은 정보를 이용하여 포인트컷을 생성하는 알고리즘이다. 대상이 되는 클래스를 파악하여 어드바이스를 호출할 수 있는 포인트컷을 생성한다. 여기서 N_n 은 C_n 의 클래스 이름이다.

표 2 는 표 1 의 알고리즘을 이용하여 만든 포인트컷이 적용되는 어드바이스를 생성하는 알고리즘이다.

표 3 은 코드 생성 알고리즘에 의해 생성된 코드를 이용하여 모니터링 부하를 비교한 것이다. NA 는 모니터링을 하지 않았을 때이고, A 는 생성한 코드로 모니터링을 할 때이다. 조건의 R 은 총 요청의 개수이며, A 는 관점의 개수이다. 해당 조건에서 응답시간을 관점의 개수에 따라 비교해 보면, 관점의 개수에 따라 다르지만 약 11%의 모니터링 부하가 있다. 모니터링 기능을 추가함으로써 모니터링 부하가 있지만, 기존의 코드를 수정하지 않고 동일한 모니터링 기능을 추가할 수 있는 장점이 있다.

<표 3> 모니터링 부하 비교

Condition	NA (sec)	A(sec)	Diff (%)
R=100, A=1	2.9953	3..0248	0.9753
R=500, A=5	3.4142	3.8765	11.9257
R=1000, A=10	4.3564	4.9266	11.5739

5. 결론

본 논문에서는 UML 의 클래스 다이어그램과 스테레오 타입을 이용하여 기존 코드를 수정하지 않고 HB 모니터링을 위한 코드를 자동 생성하여 시스템에 추가하는 방법을 제안했다. 본 방법을 통해 시스템 내부의 컴포넌트나 객체의 동작 여부를 모니터링 할 수 있으며, 모니터링의 결과로 이상이 있는 컴포넌트나 객체가 무엇인지를 파악할 수 있다.

향후 연구로 본 논문의 코드 생성 방법을 활용하여 소프트웨어 내부의 상태 이상을 감지할 수 있는 방법으로 확장할 것이다.

참고문헌

[1] Nelly Delgado, Ann Quiroz Gates, and Steve Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," IEEE Trans. On Software Engineering, Vol. 30, No. 12, pp. 859-872, 2004.

[2] Philip Koopman, "Elements of the Self-Healing System Problem Space," In Proceedings of International Conference on Software Engineering (ICSE2003), pp. 1-6, 2003.

[3] Jingjun Zhang, Yuejuan Chen, and Guangyuan Liu, "Modeling Aspect-Oriented Programming with UML Profile," In Proceedings of First International Workshop on Education Technology and Computer Science, pp. 242-245, 2009.

[4] Feng Chen, Dongyun Jin, Patrick Meredith, and Grigore Rosu, "Monitoring Oriented Programming – A Project Overview," In Proceedings of International Conference on Computer Information System, to appear, 2009.

[5] AspectJ, http://www.eclipse.org/aspectj.