Aspect Weaving 유효성 검증을 해결하기 위한 Trace Mechanism에 관한 연구

김진향*, 송영재**
*경희대학교 컴퓨터공학과
**경희대학교 컴퓨터공학과
e-mail:hesperjh@khu.ac.kr

A Trace Mechanism to Demonstrate the Verify of Aspect Weaving

Jin-Hyang Kim*, Yong-Jae Song*
*Dept of Computer Engineering, Kyunghee University
**Dept of Computer Engineering, Kyunghee University

요 약

AOP(Aspect-Oriented Programming)는 어플리케이션을 다양한 관점으로 분해하여 객체지향에서 추구하는 모듈화를 더욱 잘 지원하도록 하는 프로그래밍 기법이다. AOP의 단점은 거의 모두가 실행 전에 메소드를 차단하도록 구성되어 있다. 그리고 클래스와 Aspect의 위빙시 join point의 유효성 결함이 많이 발생한다. 따라서 본 논문에서는 trace mechanism을 이용하여 유효성 결함을 감소시키며, 클래스와 Aspcet간의 메소드 호출 관계를 명백히하기 위해 참조테이블을 생성하였다. Weaver에 의해 위빙된 후 생성된 XML코드와 저장소에 저장된 참조테이블 정보는 개발자가 원하는 요구사항에 맞게 적당한 컴포넌트에 배치되도록 Validation Agent를 사용하였다.

1. 서론

AOP(Aspect-Oriented Programming)는 어플리케이션을 다양한 관점(핵심 관심사에 대한 관점과 횡단 관심사에 대한 관점)으로 분해하여 객체지향에서 추구하는 모듈화를 더욱 잘 지원하도록 하는 프로그래밍 기법이다.[1] 즉, AOP를 적용하게 되면 핵심 관심사에서 직접 호출하는 것이 아니라 Aspect를 활용하여 컴파일 또는 런타임 시에 weaving 과정을 거쳐 하나의 시스템으로 조립하는 것을 가능하게 해준다. AOP의 단점은 거의 모두가 실행 전에 메소드를 차단하고 Trace.WriteLine("Method entered.")을 실행하는 aspect를 적용하도록 구성되어 있다.

클래스와 어스펙트의 위빙 시점에서 join-point의 유효성 결함이 많이 발생하며, 이러한 결함을 해결하기 위해서 적절한 메소드가 클래스에 호출되었는지에 대한 검증이 필요하다. 이를 보완하기 위해 효과적인 trace 메커니즘이 필요하다. Trace 메커니즘으로는 Walker and Viggers, Douence, Property checking 방법론이 있다. Walker and Viggers 방법론은 시스템 실행에서 이벤트의 순서를 설명하여 Tracingcut과 original program의 관계를 완벽하게 해주지만, tracecut 과정 동안 base program의 행위를 복사해야 하는 과정이 있어서 더욱 복잡하다. Douence 방법론은 join point가 pointcut 메소드와 매칭되는지 확인하여 advice를 실행하며, 모든 연관된 코드에 symbol이 생성되어 트래킹하는 것이 복잡하고 어려워진다.

본 논문은 유효성 검사를 위해 효과적인 trace 매커니즘을 이용한 위빙 아키텍처를 제안하여 클래스와 aspect의 보다 완전한 결합을 지원함으로써 aspect의 유효성을 향상시킨다.

본 논문의 구성은 다음과 같다. 제 2장에서는 기존의 관련 연구들을 기술하고, 제 3장에서는 제안하는 Aspect 유효성을 모니터링하기 위한 Agent 설계에 대해 상세히 기술한다. 제 4장에서는 결론 및 향후 개선해야 할 과제에 대하여 기술한다.

2 관련연구

2.1 AOP(Aspect-Oriented Programming)

AOP 개념은 원래 XeroxPARC에서 GregorKiczales와 그의 팀이 도입한 방법으로 소프트웨어 설계 및 구현에서 관심사 분리 방법을 보다 향상시킨 기법이다[1]. AOP는 여러 모듈에 산재되어 있는 횡단 관심사를 추출하여 별도의 모듈로 구현한 후 그 안에 모듈 합성 규칙을 지정한다. 이 후 직조(Weaving) 단계에서 핵심 모듈과 결합하는 방법으로 최종 시스템을 구축하는 것이다. AOP를 실제로 수행하려면 관심사 분리, 관심사 구현, 직조의 세 단계 명세 방법이 필요하다.

2.1.1관심사 분리(SeparationofConcern)

관심사는 전체적인 소프트웨어 시스템의 목적을 처리

해야 하는 구체적인 요구사항이나 고려사항을 의미한다. 관심사는 핵심 관심사(core concern)와 횡단 관심사(cross-cuttingconcern)로 분류한다. 핵심 관심사는 각 모듈에서 수행하는 기본적이고 대표적인 업무처리 기능을 취급하고 횡단 관심사는 여러 개의 모듈에 걸치는 시스템 전체적인 부가적인 요구사항을 다룬다.

2.1.2 관심사 구현

일정한 범주로 나눈 관심사를 독립적으로 구현한다. 핵심 관심사는 일반적인 프로그래밍 방법으로 구현하고 횡단 관심사는 AOP 기법을 이용하여 구현한다.

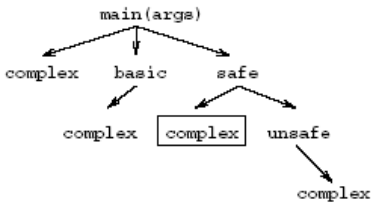
2.1.3 직조(weaving)

Aspect 모듈에 이미 정의된 합성 규칙을 지정하여 직조 또는 통합하는 과정을 거쳐 최종 시스템으로 조립하는 것이다.

2.2 Trace 매커니즘

2.2.1 Walker and Viggers

프로토콜을 실행하기 위한 수단으로 declarative event patterns (DEPs)을 사용하며, 시스템의 실행에서 이벤트의 순서를 설명한다. 개발자에게 High-level에서 프로토콜을 설명하는 것을 허락하고, Aspect언어의 확장인 proof-of-concept으로 실행되어진다[3].



[그림1] 프로토콜을 포함하는 실행 트리

Walker and Viggers 메커니즘은 Tracecut과 original program의 관계를 완벽하게 해주지만, tracecut은 matching 과정 동안 base program의 행위를 복사해야 하는 과정이 있어서 더욱 복잡하다.

2.2.2 Property checking

```
1: state decl any_pointer v;
2:
3: start: { kfree(v) } ==> v.freed;
4:
5: v.freed: { *v } ==> v.stop,
6:    { err("using %s after free!", mc_identifier(v)); }
7: | { kfree(v) } ==> v.stop,
8:    { err("double free of %s!", mc_identifier(v)); }
9: ;
```

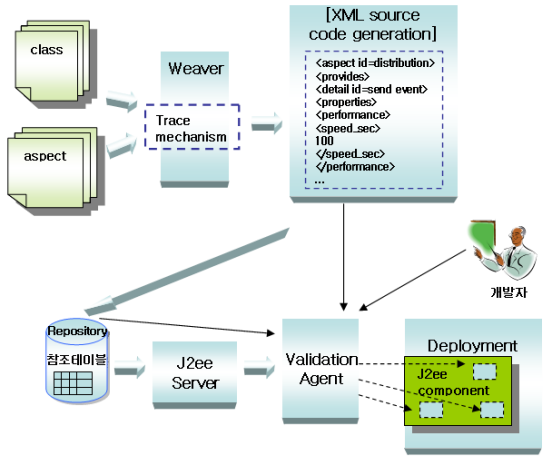
[그림 2] Free checker

Property checking 메커니즘은 시스템의 Freed pointer가 언제 Dereference되는지를 보여주며, 그림2에서는 디버깅하는 동안 12, 17 line에서 두 가지 에러를 찾아주고 있다. 이 방법론의 단점은 시스템의 보다 광범위한 속성들을 다룰 수 있게 확장되어진다는 것이다[2].

3. Aspect 유효성을 모니터링하기 위한 시스템 아키텍처 설계

본 논문에서 제안하는 Aspect 유효성을 모니터링하기 위한 Agent설계 구조는 그림[3]과 같다.

일반적인 클래스 파일과 Aspect 파일은 Weaver에 의해 XML 소스 코드로 생성된다.. Weaving시 적절한 메소드가 클래스에 호출되었는지 검증하기 위해 트레이스 메커니즘을 이용하여 위빙된다. Weaver에 의해 생성된 XML 소스코드는 Repository에 저장되며, 클래스간의 메소드 호출관계를 모니터링하기 쉽도록 하기 위해 참조테이블이 생성된다. XML코드와 Repository 정보를 참조하고 있는 Validation Agent[4]는 개발자의 요구사항에 맞게 컴포넌트를 배치시킨다.



[그림 3] 시스템 아키텍처

3.1 AOP 클래스 참조 테이블

참조테이블은 컴포넌트와 Aspect간의 함수 호출관계를 보다 쉽게 이해할 수 있도록 도와준다. 아래 두 개의 클래스 코드와 두 개의 Aspect 코드간의 간단한 참조테이블을 보여준다.

[Human.java]

```
package firstSpringAOP;

public interface Human {
        public void sayName();
}
```

[kjh.java]

```
package firstSpringAOP;

public class kjh implements Human {
        public void sayName() {
                System.out.println("Hello Aspect");
        }
}
```

[MannarAOP.java]

```
package firstSpringAOP;

import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
@Aspect
public class MannerAOP {
        @Pointcut("execution(public * Human.sayName(..))")
        public void greeting() {
        }
        @Before("greeting()")
        public void hi() {
                System.out.println("Hello.");
        }
    }
}
```

[TestFirstAOP.java]

```
package firstSpringAOP;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestFirstAOP {
        public static void main(String[] args) {

        BeanFactory bf = new ClassPathXmlApplicationContext(

"firstSpringAOP/aopAppContext.xml");
                Human human = (Human) bf.getBean("kjh");
                human.sayName();
        }
}
```

스테레오 타입	name	join_type (call or set)	join 함수명	advice (before or after)	연관 클래스
aspect	MannerAOP	call	greeting	before	sayName
aspect	TestFirstAOP	call	sayName		Human
class	Human				
class	kjh				Human

[표 1] AOP 클래스 참조 테이블

3.2 Trace Mechanism

기존의 Douence Trace Mechanism[4]은 메소드가 호출될 때마다 심볼이 생성되기 때문에 시스템이 더욱 복잡하고 이해하기 힘들다. 이를 위해 아래와 같은 Trace Mechanism을 제안한다.

$$
\begin{aligned}
A ::= & \ \mu a.A \\
& \mid C \triangleright I; a \quad recursion \\
& \mid C \triangleright I; A \quad case\ of\ recursion \\
& \mid C \triangleright I; S \quad skip \\
& \mid A_1 \square A_2 \quad choice
\end{aligned}
$$

만약 join-point가 컨트롤 C와 매칭된다면, advice I가 실행되며, 컨트롤은 a또는 A에 전달되는데, 만약 기존에 호출되었던 메소드는 컨트롤에 전달되지 않고, skip하기위한 S인수에 전달된다. 이로 인해 불필요한 메소드의 심볼 생성을 제거할 수 있다.

4. 결론 및 향후 연구 방향

본 논문에서 제안한 Weaving trace mechanism을 이용한 Agent 개발 시스템은 AOP와 클래스의 결합시 발생되는 join point의 유효성에 결함이 생기는 단점을 해결해 준다. 위빙 후 생성된 소스코드는 참조테이블 템플릿 형태로 저장소에 저장되기 때문에 클래스간에 참조된 메소드 정보를 쉽게 확인할 수 있다는 장점이 있다. 생성된 XML 소스코드와 저장소의 참조테이블 정보를 기반으로 Validation Agent는 개발자의 요구사항에 맞게 적절한 컴포넌트에 합성된 클래스가 배치되어지므로, 재사용성을 용이하게 해준다.

향후에는 본 논문에서 제시한 Weaving trace mechanism을 이용한 agent개발 시스템을 기반으로 Multi-agent에서 실행될 수 있도록 확장하는 작업이 필요하다.

참고문헌

[1] G. Kiczales, G. et al. "Aspect Oriented Programming," Lecture Notes in Computer Science. Vol. 1241, 1997. pp. 220-242

[2] Alessandro F. Garcia, Viviane Torres da Silva, Christina Chavez, Carlos José Pereira de Lucena: Engineering multi-agent systems with aspects and patterns. J. Braz. Comp. Soc. 8(1): 57-72 (2002)

[3] Robert J. Walker, Kevin Viggers: Implementing protocols via declarative event patterns. SIGSOFT FSE 2004: 159-169

[4] John C. Grundy, Guoliang Ding: Automatic Validation of Deployed J2EE Components Using Aspects. ASE 2002: 47-