Journal of Digital Contents Society Vol. 11 No. 1 Mar. 2010(pp. 25-38)

Spring 프레임워크 AOP의 UML/XML 확장 표현 및 변환 기법

이후재*, 류성열**, 김종배***

요 약

오픈 소스 프레임워크 기반의 Spring AOP에는 명세화를 위한 지침이나 표준이 없어 개발과 유지보수에 혼란을 야기 시키고 있으며, 특히 기존의 MVC 모델, Struts 프레임워크에서 사용하는 AOP 모델 연구와 UML 다이어그램과의 불일치 문제는 Aspect의 재사용을 더욱 어렵게 하고 있다.

본 연구는 기존의 MVC 모델, Struts 프레임워크에서 사용하는 AOP 모델의 혼용을 Spring AOP에서 수용 처리하기 위하여, Aspect, Pointcut과 Advice를 UML과 XML로 어느 것으로 표현하여도 가능할 수 있도록 기존의 표현법을 확장하여 제시하고, 제시된 표현 기법이 상호변환 가능할 수 있도록 관계성을 정의한 후, 상호변환의 사례를 보여 개발과 유지보수를 용이하게 함을 입증 하였다. 또 Aspect의 재사용을 보다 효율적으로 활용하기 위한 패키지화 방법을 제안하고 그 재사용 가능성을 검증하였다.

A Method of the Widening Expression and Conversion of the Spring Framework AOP into UML/XML

Hoo-Jae Lee*, Sung Yul Rhew**, Jong-Bae Kim***

Abstract

There is no guideline or standard for the specification of the open-source-framework-based Spring AOP, and it causes confusion in development and maintenance. Moreover, the inconsistency between the existing MVC model, the AOP model that is used for Struts framework and the UML diagram makes the aspect reuse more difficult.

In this study, a widened existing method was proposed so that Aspect, Pointcut and Advice could be expressed by either UML or XML to ensure that the Spring AOP can accept the combined use of the existing MVC model and the AOP model, which is used for Struts framework. Relationship was defined so that the mutual conversion could be possible with the proposed expression method, and the realization of simple development and maintenance was verified via the examples of mutual conversion. In addition, a packaging method to efficiently reuse aspect was proposed, and the possibility of reuse was verified.

Keywords : Spring Framework, Aspect, Package, Reuse

1. 서론

최근 기업들은 기술, 환경, 비즈니스의 변화가 가속화 되면서 저비용, 고효율, 빠른 소프트웨어 개발을 하기 위한 방안으로 오픈 소스 소프트웨어(Open Source Software)의 사용을 늘리고 있다. Spring 프레임워크는 AOP(Aspect Oriented Programming)와 DI(Dependency Injection)같은 새로운 프로그램 기법으로 코딩의 복잡함을 줄이고 다양한 구현이 가능하여 많은 사람들의 관심을 받으며 널리 사용되고 있으며[1], 오픈 소스 소프트웨어에서 주로 사용되는 프레임워크이다.

하지만, Spring 프레임워크 기반의 개발에서

※ 제일저자(First Author) : 이후재
접수일:2009년 12월 16일, 완료일:2010년 03월 30일
* 숭실대학교 컴퓨터학과
hooklee@ssu.ac.kr
** 숭실대학교 컴퓨터학과 교수
*** (주)이엔터프라이즈 대표이사
▣ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2010-(C1090-0903-0004).

AOP의 주요 요소인 Aspect, Pointcut과 Advice 정의를 위한 표준화된 지침 및 명세가 없어[2], AOP가 요구되어 접근하는 개발자들에게 텍스트 기반의 AOP 작성과 표기 방법의 혼용은 AOP에 대한 이해와 사용법이 쉽지 않아 어려움이 있다. 이에 Spring 프레임워크 기반의 AOP 연구가 필요하다.

아래는 Spring AOP에 대한 문제점을 문헌을 통해 정리한 것이다.

첫째, Spring AOP 명세화 방법은 자동화 지원이 되지 않고 개발자가 텍스트 기반의 XML로 작성 하는데, XML작성은 Spring AOP에서 제공하는 방법에 의한 작성, AOP 네임스페이스를 사용한 작성과 어노테이션(Annotation)을 사용한 작성 등 다양하게 존재한다. 이처럼, 명세화 방법의 혼용은 개발자가 프로그램 흐름이 어떻게 이루어지고 어떤 클래스들과 연관되어 있는지 XML로 이해하는데 부담을 주어, 시스템 개발 및 유지보수에 어려움이 있다[3][4][5][6].

둘째, 기존 AOP 모델 연구 중 Naoyasuh의 연구[7]는 표준 자바 클래스로 작성하는 Advice[6]를 AOP 모델에 클래스로 표현을 하지 않고 Aspect 클래스 내부 요소로 표현을 하여, 복잡 다양한 Spring AOP 구성 요소[8] 표현의 제한과 Aspect 재사용에 어려움[9]이 있다.

본 연구에서는 Spring AOP의 주요 요소인 Pointcut과 Advice는 각각 클래스로 구조화하고, Aspect는 패키지로 표현하는 Spring AOP UML 표현 및 XML 변환 기법을 제안 한다. UML 표현은 개발자들에게 이미 친숙한 객체 지향 모델링과 유사하기 때문에 새로운 표현 방식을 익히지 않더라도 복잡 다양한 Spring AOP XML을 시각적, 직관적으로 제공할 수 있어서 XML 코드보다 이해하기 쉽다. 이에, 제안 기법은 Spring 프레임워크의 활용에 있어서 AOP의 주요 요소인 Aspect, Pointcut과 Advice를 UML로 표현하고, 이에 상응하는 Spring AOP XML 변환 기법을 제시하여 UML에 의해서 프로그램의 흐름을 이해하고 추적할 수 있다. 또한, 제안 기법은 Spring AOP 개발 시 UML 표현을 통해 Spring AOP XML로 코드화 할 수 있고 유지보수 시에는 Spring AOP UML 모델을 재사용하여 Spring AOP XML로 변환할 수 있는 기법이다.

본 연구를 통해 제시된 Spring AOP UML 표현 및 XML 변환 기법으로 다음과 같은 기여가 있을 것으로 기대된다. Spring AOP를 UML 표기법에 적합하도록 표현할 수 있으며, 문서화 및 시각화를 통해서 가독성, 이해성 이 증대되고, UML 설계문서를 활용한 유지보수가 가능함으로 모델의 재사용이 가능하다. 또한, 본 연구는 Spring AOP XML의 일관된 변환 기법을 제시하여, 기법에 따라 Spring AOP XML을 변환하면 잘못된 AOP 설정으로 인한 시스템의 다운 타임과 그에 따른 비용을 감소할 수 있다.

2. 관련연구

본 장에서는 AOP를 UML로 도식하는 관련 연구를 한다.

2.1. Naoyasuh의 AOP 모델 연구

Naoyasuh[7]는 Model Driven Architecture (MDA)[10]의 Platform Independent Model(PIM)로 Struts 프레임워크 기반의 AOP UML 표현을 제안했다. Naoyasuh의 기법은 1단계에서 PIM 모델에서 연관 클래스들을 통합하고, 2단계에서는 PIM 모델에서 통합된 연관 클래스를 Struts 프레임워크 기반의 Platform Specific Model(PSM) 모델로 작성을 한다. 3단계에서는 웹 브라우저의 요청을 실행하는 연관 클래스에 대한 Action 클래스 생성과 execute() 오퍼레이션을 정의한다. Naoyasuh의 연구는 AOP 모델을 설계하는 혁신적인 연구이다. 하지만, Naoyasuh의 연구는 표준 자바 클래스로 작성하는 Advice[6]를 AOP 모델에 클래스로 표현을 하지 않고 Aspect 클래스 내부 요소로 표현을 하여, 복잡 다양한 Spring AOP 구성 요소[8] 표현의 제한과 Aspect 재사용에 어려움[9]이 있다. Naoyasuh의 연구는 AOP 모델과 Aspect 설정 파일간의 상호 변환 검증에 대한 연구를 하지 않아 정상적으로 변환됨을 보장하기 어렵고, Struts 프레임워크 기반에 국한 된 연구여서 Spring 프레임워크에 반영하기 어렵다.

2.2. Groher의 AOP 모델 연구

Groher[11]는 Aspect 구조 설계 모델링을 위한 Aspect Modeling Language(AML)을 제시하

였다. 초기 추출된 횡단 관심사를 AML로 표기하여 이를 UML로 매핑하는 방법이다. AML은 UML로 표현하기 힘든 횡단 관심사를 클래스 다이어그램과 같은 형태로 표현하였고 이를 기반으로 코드 정보를 추출하도록 지원한다. Groher의 연구는 이해하기 어려운 클래스 레벨 AOP 모델의 문제점을 해결하기 위해서 관심사를 Base 패키지, Aspect 패키지, 그리고 이를 연결하는 Connector 패키지로 표현하였다. 또한, AspectJ에 AML을 적용한 AspectJ Connection Model을 제시하였다. 하지만, 명세 정보를 개발자가 이해하여 코드로 변환하기에는 복잡한 구조를 가지고 있으며, 커넥터 패키지를 명세하기 위한 명확한 지침이 없어 초기 개발자가 설계하기에는 어려움이 있다.

2.3. Jacobson의 AOSD 모델 연구

Jacobson의 연구[12]는 Aspect 지향 설계는 분석 단계에서 객체 지향 설계와 동일하게 핵심 객체를 식별 하지만 추가적으로 Aspect를 식별하고 개체와 Aspect간의 관계를 식별을 한다. 설계 단계에서는 식별된 객체와 Aspect를 독립적인 클래스로 작성 후 클래스 다이어그램을 완성한다. 구현 단계에서는 객체 지향 설계의 구현사항에 추가적으로 AOP 설정 파일을 포함하여 구현을 한다. Jacobson의 연구는 유스 케이스(Use Case)를 사용한 AOSD(Aspect Oriented Software Development) 모델링에 관련된 연구이며, 이 연구에서는 시스템 관심사들을 유스 케이스를 기반으로 분석하고 설계하는 관점 지향 4단계 모델링 방법을 제시하였으나, Aspect에 대한 상세 모델과 Spring AOP XML로의 변환에 대한 연구는 하지 않아 개발자의 잘못된 AOP 설정으로 인한 시스템의 다운 및 AOP 명세화 방법이 혼용될 수 있어서 가독성, 이해성의 문제가 있다.

3. AOP의 UML 확장 표현 기법

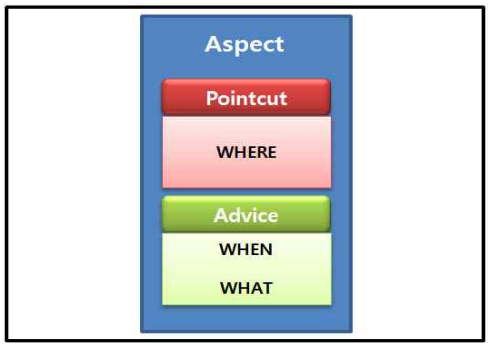
본 장에서는 기존 AOP 모델 연구의 Aspect 클래스화의 문제점을 문헌과 자료를 토대로 도출한다. 이를 기반으로 Spring AOP UML 표현 기법을 제안한다.

3.1. Aspect 클래스화의 문제점

본 절에서는 Aspect 구성 요소의 관계와 Aspect가 수행 되는 과정, Spring AOP XML을 분석하여 Aspect 클래스화의 문제점을 도출한다.

첫 번째, Aspect 구성 요소 분석을 통해 Aspect가 클래스로 표현되는 것의 문제점을 도출한다. Aspect는 Advice와 Pointcut 모두를 포함한다. 두 가지 정보가 합쳐지면 Aspect가 무엇을 언제 어디서 할지가 완성이 된다. Spring AOP에서 Aspect는 Advisor 인터페이스를 구현한 클래스의 인스턴스이다. Advice는 Aspect가 무엇(WHAT)을 언제(WHEN) 할지를 정의한다. 즉 Aspect가 해야 하는 작업(무엇)과 언제 그 작업을 수행해야 하는지를 정의 한 것이 Advice이다. Advice가 정의하는 언제는 작업이 Method 호출 이전, 이후, 이전 / 이후, 예외 발생 시에 적용 되는 것을 의미한다. Aspect가 무엇을 언제 할 것이냐를 정의하는 것이 Advice라면, Pointcut은 어디서(WHERE)를 정의한다.

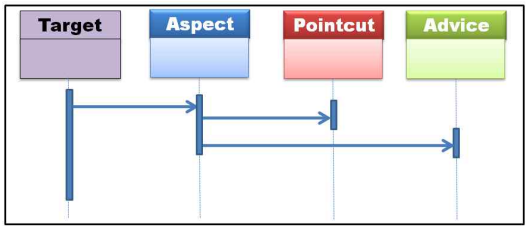
(그림 1)은 Aspect내에 포함되는 Pointcut과 Advice의 관계를 도식한 것이다. Spring 기반의 모든 Advice는 표준 자바 클래스로 작성을 한다[6]. 그러므로, Advice가 클래스의 속성을 가지고 있기 때문에 Aspect 클래스 내에 Advice를 포함할 수 없다.



(그림 1) Aspect, Pointcut, Advice 관계도

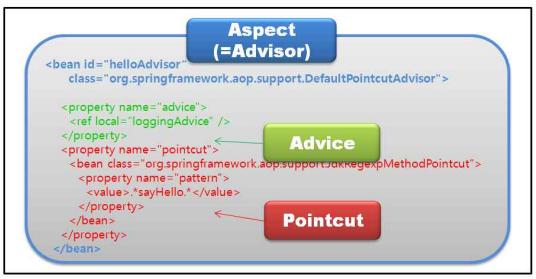
두 번째, Aspect가 수행 되는 과정을 통해 Aspect가 클래스로 표현되는 것의 문제점을 도출한다. (그림 2)는 Aspect 와 Pointcut, Advice의 관계를 도출하기 위해 UML에서 지원하는 Sequence Diagram으로 Aspect 수행 시의 순서를 표현한 것이다. Target은 비즈니스 오퍼레이션을 수

행하는 클래스이다. Target 클래스가 수행이 되면서 Aspect를 호출한다. Aspect는 Pointcut을 호출한 후 Advice를 호출한다. (그림 2)를 보면, Pointcut과 Advice는 직접적인 관계는 없다. 단지, Pointcut과 Advice는 Aspect에 의해 관계가 형성이 된다. 그러므로 Aspect 클래스에 Pointcut과 Advice를 함께 표현하는 것은 동일한 어트리뷰트(Attribute), 오퍼레이션(Operation), 관계(Relationship), 그리고 의미를 공유하는 객체들을 나타내는 클래스의 정의에 어긋난다[13].



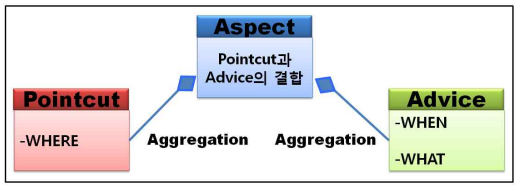
(그림 2) Aspect 수행 시 Sequence Diagram

세 번째, Aspect가 구현된 Spring AOP XML을 분석하여 Aspect가 클래스로 표현되는 것의 문제점을 도출한다. (그림 3)은 Spring AOP XML 구현 예이다. Advisor는 AOP의 Aspect와 같은 개념을 가진다. Aspect가 Pointcut과 Advice의 결합인 것처럼 Advisor를 생성하기 위해서는 Pointcut과 Advice가 필요하다[14].



(그림 3) Spring AOP XML 예제 코드

Spring AOP XML 코드를 보면, Pointcut과 Advice는 동일 수준에 목적이 다르다. Aspect가 Pointcut과 Advice를 결합하고 있다. 이러한 관계를 통해 Aspect와 Pointcut, Advice는 (그림 4)와 같은 클래스 다이어그램으로 도식 할 수 있다. Pointcut은 Aspect를 적용하려는 곳을 정의하고, Advice는 Pointcut이 적합할 때 어떤 Advice 기능을 실행할 것인지를 정의한다. Aspect는 Pointcut과 Advice에 정의된 것들을 결합하기 때문에 Aspect 클래스에 대해 Pointcut 클래스와 Advice 클래스는 Aggregation 관계로 표현된다.



(그림 4) Aspect, Pointcut, Advice의 Class Diagram

Aspect 구성 요소의 관계와 Aspect가 수행 되는 과정, Spring AOP XML을 분석한 결과 아래와 같은 문제가 있었고, 이를 해결하기 위한 Spring AOP UML 표현 기법이 필요하다.

첫째, UML 다이어그램과 구현 과정에서 클래스로 표현이 되는 Advice가 기존 AOP 모델 연구에서는 Aspect 클래스의 오퍼레이션 및 어트리뷰트로 표현을 하여 모델 불일치의 문제가 있다. 이는 모델을 통한 Aspect의 재사용에 어려움이 있다.

둘째, 기존 AOP 모델 연구는 Pointcut과 Advice를 오퍼레이션이나 어트리뷰트로 표현하여, 각각에 대한 복합적인 속성이나 값 등을 Aspect 클래스에 모두 표현하기에는 어려움이 있어 이해성, 가독성에 문제가 있다.

3.2. Spring AOP UML 표현

본 절에서는 3.1절에서 도출한 기존 AOP 모델 연구의 Aspect 클래스화의 문제점을 해결하기 위하여 Spring AOP UML 표현 기법을 제안한다.

클래스는 동일한 어트리뷰트, 오퍼레이션, 관계, 그리고 의미를 공유하는 객체들을 나타내는 것이고, 패키지는 거대한 소프트웨어를 기능적으로 분리하여 모델을 조직화하고 모델을 쉽게 이해하도록 도와준다[13]. UML 모델 기반의 AOP 모델을 하여 모델간의 불일치를 제거하고 Aspect의 재사용이 가능하도록 해야 한다.

또한, Pointcut과 Advice에 대한 복합적인 속성이나 값 등의 AOP 모델로 이해가 가능 하여야 한다.

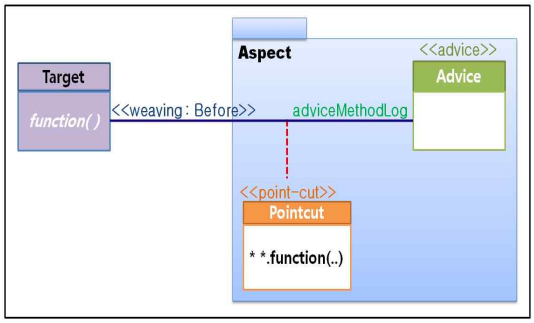
그래서, Pointcut과 Advice를 Aspect 클래스에 포함하여 표현하는 것보다는 Aspect를 패키지화하고 Pointcut과 Advice를 각각의 클래스로써 Aspect 패키지에 포함하여 표현하면 AOP 모델이 구현과 일치되어 Aspect의 재사용이 가능하다. 또한, Pointcut과 Advice를 클래스로 표현하면 복합적인 속성이나 값 등을 오퍼레이션과 어트리뷰트로 세부적인 표현을 할 수 있어서 Aspect, Pointcut과 Advice의 이해성 및 가독성이 향상할 것이다.

<표 1>은 Spring AOP UML 표현 기법을 정리한 것이다. Aspect는 Pointcut 클래스와 Advice 클래스가 포함된 패키지로 표현을 하고, Pointcut은 스테레오 타입을 <<point-cut>>로, Advice는 스테레오 타입을 <<advice>>로 표현한다. Pointcut에 의해서 결정된 Jointpoint를 Target 클래스에 Advice를 삽입하는 과정을 Weaving이라 한다. Weaving의 스테레오 타입 표현은 비즈니스 오퍼레이션 호출 이전을 <<weaving:Before>>, 오퍼레이션 호출 이후는 <<weaving:After-returning>>, 예외 처리는 <<weaving:Throws>>, Before와 After-returning 그리고 Throws Advice를 모두 하나로 처리할 때는 <weaving:Around>>, 코드 변경 없이 클래스에 오퍼레이션과 변수(Variable)를 추가하는 기능은 <<weaving:Introduction>>로 표현한다. Association Class는 Pointcut과 Advice 오퍼레이션과의 관계, Pointcut에 해당하는 Advice를 Target 클래스로 Weaving하는 시점에 대한 연관 관계를 표현한다.

4. AOP UML 클래스에서의 XML 변환 기법

본 장에서는 3장에서 제안한 Spring AOP UML 표현 기법을 가지고 Spring AOP XML로의 변환 기법을 제안한다.

(그림 5)는 <표 1> Spring AOP UML 표현 기법을 바탕으로Aspect를 패키지화하여 모델링을 한 예이다.



(그림 5) Spring AOP UML 모델링의 예

본 예제는 Pointcut 클래스가 Aspect 패키지에서 수행할 Jointpoint를 지정하고 Advice 클래스는 Pointcut 조건에 맞을 때에 Advice 클래스

<표 1> Spring AOP UML 표현 기법

구분	AOP UML 표현	설 명
Aspect		- Aspect는 Pointcut과 Advice를 포함하는 패키지로 표현
Pointcut		- 어디(Where)에 해당하는 Pointcut은 <<point-cut>> 스테레오 타입을 사용하여 클래스로 표현
Advice		- 언제(When)/무엇(What)에 해당하는 Advice은 <advice>> 스테레오 타입을 사용하는 클래스로 표현
Weaving	<<weaving : [weaving 기능]>>	- Target 클래스에 Advice를 삽입하는 과정인 Weaving의 표현
Advice Method	[Advice Method Name]	- Advice가 수행될 때 호출하는 Advice 클래스의 오퍼레이션
Association Class		- Pointcut 클래스와의 연관관계를 표현

내의 오퍼레이션인 adviceMethodLog를 Target 클래스에 Weaving 하는 것이다.

(그림 6)은 (그림 5) Spring AOP UML 모델링의 예를 XML로 변환한 것이다. Target 클래스의 객체는 'instance id="obj_Target"'로 변환하고, Advice 클래스의 객체는 'instance id="obj_Advice" '로 변환한다. Aspect 패키지는 <aspect>와 </aspect>로 변환하고 그 안에 Pointcut 클래스와 Advice클래스를 각각 변환한다. Pointcut 클래스는 Spring AOP UML 모델링의 스테레오 타입으로부터 <point-cut>과 </point-cut>으로 변환하고 Jointpoint를 <value>" * *.function(..)"</value>로 변환한다. Advice 클래스는 Spring AOP UML 모델링의 스테레오 타입으로부터 <advice>와 </advice>로 변환하고 참조하는 Advice 객체를 '<advice reference local="obj_Advice">'로 변환한다. Weaving 시점을 표시하는 <<weaving:Before>>와 Advice 오퍼레이션인 adviceMethodLog, Pointcut과의 관계를 표현한 Association Class는 <advice> 엘레먼트 내에 Weaving 시점, 호출할 Advice 오퍼레이션, Pointcut과의 연관 관계를 '<before function= "adviceMethodLog" point-cut id=" alias_pointcut"/>'로 변환한다.

(그림 7)은 (그림 6) Spring AOP UML 모델의 XML을 Spring의 AOP 네임스페이스을 이용한 표현식으로 변환한 것이다. Spring의 AOP 지원 형태는 여러 가지가 있다. @AspectJ와 같은 어노테이션과 XML 설정을 혼합해서 사용할 경우 관리가 복잡해지기 때문에 본 연구에서는 Spring AOP 네임스페이스를 사용하여 AOP XML을 작성한다[6].

<표 2>는 (그림 6) Spring AOP UML 모델의 XML과 (그림 7) Spring AOP XML간의 변환관계를 도출한 것이다. <표 2>를 보면, Spring AOP UML 모델의 XML에서는 Advice가 reference local="obj_Advice"로 변환된 것이 Spring AOP XML에는 Aspect가 ref="obj_Advice"로 변

```
< instance id="obj_Target"  class="Target"  />
< instance id="obj_Advice"  class="Advice"  />
<aspect>
<!-- 1. Pointcut 정의  : WHERE  -->
<point-cut name=" alias_pointcut ">
        <value>" **.function(..)" </value>
</ point-cut >
<!-- 2. Advice 정의 : WHEN, WHAT  -->
<advice reference local="obj_Advice" >
        <before function= "adviceMethodLog" point-cut id=" alias_pointcut"  />
</ advice >
</aspect>
```

(그림 6) Spring AOP UML 모델의 XML

```
< bean id="obj_Target"  class="Target"  />
< bean id="obj_Advice"  class="Advice"  />
<aop:config>
<aop:aspect ref="obj_Advice">
        <aop:pointcut id="alias_pointcut"expression="* *.function(..)"/>
        <aop:before method=" adviceMethodLog" pointcut_ref="alias_pointcut" />
</aop:aspect>
<aop:config>
```

(그림 7) Spring AOP XML

<표 2> Spring AOP UML 모델의 XML과 Spring AOP XML과의 관계

구 분	Spring AOP UML 모델의 XML	Sping AOP XML
인스턴스 생성	<instance id="obj_Target" class="Target" /> <instance id="obj_Advice" class="Advice" />	< bean id="obj_Target" class="Target" /> < bean id="obj_Advice" class="Advice" />
Aspect	<aspect> … </aspect>	<aop:config> <aop:aspect ref="obj_Advice"> … </aop:aspect> <aop:config>
Pointcut	<point-cut name=" alias_pointcut " > <value>" **.function(..)" </value> </ point-cut >	<aop:pointcut id="alias_pointcut" expression="* *.function(..)"/>
Advice	<advice reference local="obj_Advice" > <before function="adviceMethodLog" point-cut id=" alias_pointcut"/> </ advice >	<aop:before method=" adviceMethodLog" pointcut_ref="alias_pointcut" />

<표 3> Spring AOP UML 모델의 XML과 AspectJ AOP 코드와 관계

구 분	Spring AOP UML 모델의 XML	AspectJ AOP 코드

Aspect	<aspect>
	
	…
	
	</aspect>
		package aspect.aspcet;
		
		public aspect Alias_aspect {
		…
		}

Pointcut	<point-cut name=" alias_pointcut " > <value>" **.function(..)" </value> </ point-cut >	pointcut alias_pointcut (): call(* *.function(..));

Advice	<advice reference local="obj_Advice" >
	<before function="adviceMethodLog"
	point-cut id=" alias_pointcut"/>
	</ advice >
		before() : alias_pointcut () {
		
		adviceMethodLog();
		
		}

환된다. <instance>는 <bean>으로, <aspect>는 <aop:config><aop:aspect>로, <point-cut>은 <aop:pointcut>로, <advice>는 <aop:before>로 각각 변환된다.

현재 Aspect 프로그래밍 언어들 중 가장 많이 사용하고 있는 AspectJ를 제안 기법에 적용하였다[9] [18]. AspectJ의 AOP 코드는 Java 코드처럼 보이나 일반적인 Java 컴파일러로는 사용이 불가능하고 전용 컴파일러를 사용해야 한다[15]. (그림 8)은 (그림 5) Spring AOP UML 모델링의 예에 상응하는 AspectJ AOP 코드로 변환한 것이다.

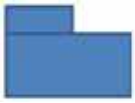
```
package aspect.aspcet

public aspect Alias_aspect {
    pointcut alias_pointcut (): call(* *.function(..));
    before() : alias_pointcut () { adviceMethodLog(); }
}
```

(그림 8) AspectJ AOP 코드

<표 3>은 (그림 6) Spring AOP UML 모델의 XML과 (그림 8) AspectJ AOP 코드와의 변환 관계를 도출한 것이다. Spring AOP UML 모델의 XML에서는 <aspect>로 표현되는 Aspect가 AspectJ AOP 코드에서는 패키지 선언문과 클래스 명으로 변환이 되고, <point-cut>은 pointcut

<표 4> Spring AOP UML 모델과 Spring AOP XML 코드 변환간의 관계

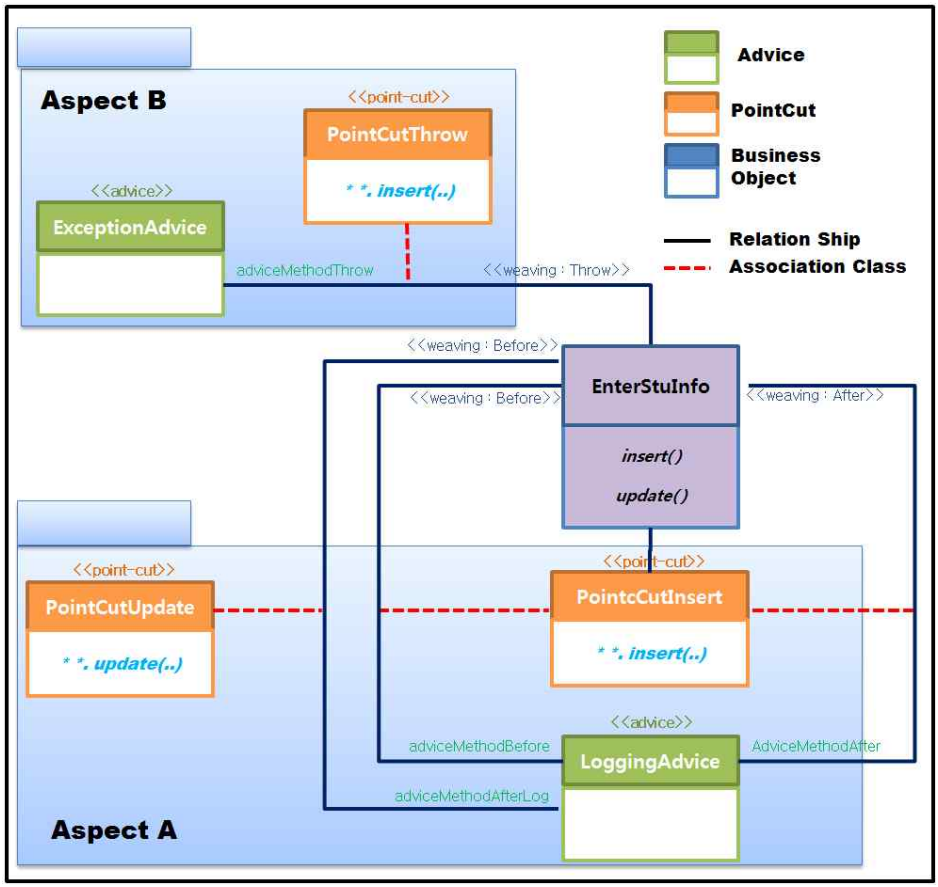
구분	AOP UML 모델	Spring AOP XML
Aspect		… <aop:config> <aop:aspect ref="obj_Advice"> … </aop:aspect> <aop:config>
Pointcut		… <aop:pointcut id="alias_pointcut" expression="* *.function(..)"/> …
Advice		… < bean id="obj_Advice" class="Advice" /> … <aop:aspect ref="obj_Advice"> …
Weaving	<<weaving : Before>>	… <aop:before method=" adviceMethodLog" …
Advice Method	adviceMethodLog	… <aop:before method=" adviceMethodLog" …
Association Class		… <aop:before method=" adviceMethodLog" pointcut_ref="alias_pointcut" /> …

으로, <advice>는 before()로 각각 변환된다.

<표 4>는 (그림 5) Spring AOP UML 모델링의 예와 (그림 7) Spring AOP XML간의 변환 관계를 도출한 것이다. Aspect는 Spring AOP UML 모델에서 패키지로 표현되고, Spring AOP XML에서는 <aop:config> 와 <aop:aspect>로 변환된다. Pointcut은 Spring AOP UML 모델에서 <<point-cut >> 스테레오 타입의 클래스로 표현되고, Spring AOP XML에서는 <aop:pointcut>으로 변환된다. Advice는 Spring AOP UML 모델에서 <<advice>> 스테레오 타입의 클래스로 표현되고, Spring AOP XML에서는< bean id="obj_Advice" class="Advice" />와 ref="obj_Advice"로 변환된다. Spring AOP UML 모델의 Weaving 시점의 표현인 <<weaving:Before>>와 Advice 오퍼레이션인 adviceMethodLog는 Spring AOP XML에서 각각 aop:before와 method ="adviceMethodLog"로 변환된다. Association Class는 Weaving 시점의 표현인 <<weaving:Before>>, Advice 클래스의 오퍼레이션인 adviceMethodLog와 Pointcut의 연관 관계로부터 <aop:before method="adviceMet hodLog" pointcut_ref="alias_pointcut"/>로 변환된다.

5. 적용가능성 및 평가

본 연구에서 제시한 Spring AOP UML 모델을 AOP의 관리가 요구되는 D대학의 입시 시스템에 적용해 보았다. D대학의 입시 시스템은 Unix를 운영체제로 Spring Framework 2.0.1 기반의 AOP로 개발되어 운영 중인 시스템으로 년도별, 모집 요강 별로 구분을 하여 입학 원서 접수, 성적(자료) 처리, 사정 관리, 합격자 관리, 통계자료 관리, 신입생 등록 업무를 하는 시스템이다. 입시 감사 자료를 제공하기 위하여 학년도별 입시 관련 자료는 5년간 유지해야 한다. 그래서, 학년도별 입시 시스템의 운영기간도 5년을 유지해야 한다. 이전 학년도의 입시 시스템을 복사하여 새로운 학년도의 입시 시스템을 만들고 학년도별 2회 이상 요구사항을 반영한다. 이를 위해, 자바 소스 코드와 Spring AOP XML 수정이 요구되었으나 Spring AOP XML 작성 표준 및 S



(그림 9) 입학원서 접수업무의 Spring AOP UML 모델링

pring AOP XML에 대한 문서화가 되어 있지 않아 AOP를 이해하고 추적하기가 어려운 문제가 있었다. 이러한 이유로 입시 시스템의 AOP를 관리할 수 있는 방안이 요구되어 제안 기법을 적용하였다.

(그림 9)는 입시 시스템 중 입학 원서 접수 업무를 분석하여 Spring AOP UML 모델링을 한 것이다. (그림 9)의 EnterStuInfo 클래스는 입학 원서 정보를 관리하는 클래스이다. AspectA 패키지는 입학 원서의 정보의 일괄 작업과 수정과 관련된 Pointcut 클래스와 Advice 클래스로 구성을 하였고, AspectB 패키지는 일괄 작업 중 발생되는 오류 데이터 처리와 관련된 Pointcut 클래스와 Advice 클래스로 구성을 하였다. 관련 업무별 클래스와의 관계는 다음과 같다.

인터넷 업체로부터 받은 입학 원서 정보를 일괄 작업할 때의 관련 Advice는 LoggingAdvice 클래스이고, Pointcut은 PointCutInsert 클래스이다. LoggingAdvice 클래스는 EnterStuInfo 클래스의 insert () 오퍼레이션 호출 이전·후 로그 정보를 관리하는 오퍼레이션인 adviceMethod Before와 adviceMethodAfter를 호출한다. PointCutInsert 클래스는 Jointpoint를 insert인 모든 오퍼레이션으로 정의하여 입학 원서 일괄 작업을 할 때 적용되도록 하였다.

입시 담당 부서에서 입학 원서 정보를 수정할 때의 관련 Advice는 LoggingAdvice 클래스이고, Pointcut은 PointCutUpdate 클래스이다. Logging Advice 클래스는 EnterStuInfo 클래스의 update() 오퍼레이션 호출 이후 로그 정보를 관리하는 오퍼레이션인 adviceMethodAfterLog를 호출한다. PointCutUpdate Jointpoint를 update인 모든 오

퍼레이션으로 정의하여 입학 원서 정보를 수정할 때 적용되도록 하였다.

입학 원서 일괄작업 중 발생되는 오류 데이터의 처리시의 관련 Advice는 ExceptionAdvice 클래스이고 Pointcut은 PointCutThrow 클래스이다. ExceptionAdvice 클래스는 EnterStuInfo 클래스의 오퍼레이션 호출 이후 발생되는 예외 처리 오퍼레이션인 adviceMethodThrow를 호출한다. PointCutThrow 클래스는 Jointpoint를 insert인 모든 오퍼레이션으로 정의하여 입학 원서 일괄작업 중 발생되는 오류 데이터를 별도로 저장하도록 하였다.

(그림 10)은 (그림 9) 입학 원서 접수 업무의 Spring AOP UML 모델링을 기반으로 Spring AOP 네임스페이스에 의한 Spring AOP XML 코드로 변환한 것이다.

```
<bean id="enterstuinfo"        class=" EnterStuInfo" />
<bean id="loggingadvice "      class = "LoggingAdvice"  />
<bean id="exceptionadvice"     class = "ExceptionAdvice"/>

<!--  AspectA 패키지   -->
<aop:config>
      <aop:aspect ref="loggingadvice">
           <aop:before method=" adviceMethodBefore"  pointcut="execution(* *. insert (..))" />
      </aop:aspect>
</aop:config>
<aop:config>
      <aop:aspect ref="loggingadvice">
           <aop:after  method=" adviceMethodAfter"     pointcut="execution(* *. insert (..))" />
      </aop:aspect>
</aop:config>
<aop:config>
      <aop:aspect ref="loggingadvice">
           <aop:before method=" adviceMethodAfterLog" pointcut="execution(* *. update(..))" />
      </aop:aspect>
</aop:config>

<!--  AspectB 패키지   -->
<aop:config>
   <aop:aspect ref="exceptionadvice">
        <aop:after-throwing method="adviceMethodThrow" pointcut="execution(* *. insert(..))" />
   </aop:aspect>
</aop:config>
```

(그림 10) 입학 원서 접수 업무의 Spring AOP XML 코드 변환

본 연구에서는 위와 같은 기법 적용 사례를 바탕으로, 6년 이상의 입시 시스템 개발 경험을 가진 중급 기술자 10명에게 입학 원서 접수 업무, 성적(자료) 처리, 사정 관리, 합격자 관리, 통계자료 관리, 신입생 등록 업무 등에 제안 기법을 적용하도록 하였다.

<표 5>는 제안 기법 적용 전·후의 AOP XML LOC(Line of Code)를 비교한 것이다. 제안 기법 적용 전보다 제안 기법 적용 후에 AOP XML의 LOC가 감소되었다. 이유는 제안 기법 적용 전에는 Spring AOP XML 작성 표준 및 Spring AOP XML에 대한 문서화가 되어 있지 않아서 중복된 기능의 구현이 발생이 되었다. 제안 기법 적용 후 Spring AOP UML 모델을 통해 중복된 기능의 구현을 제거할 수 있었으며, Spring AOP UML 모델을 재사용하여 정제됨으로써 Spring

<표 6> 제안 기법 적용 전·후 개발 노력 비교

구 분	모집	개발자										평균	평균 합계	기법 적용 전 대비 M/M의 백분율
		1	2	3	4	5	6	7	8	9	10			
기법 적용 전	수시	2.5	2.0	2.5	3.0	3.0	3.0	2.5	2.5	3.0	2.5	2.65	5.45	100%
	정시	3.0	2.5	3.0	2.5	3.0	2.5	3.0	3.0	2.5	3.0	2.80		
기법 적용 후	수시	3.5	3.0	3.5	3.5	3.0	3.5	3.5	3.0	3.5	2.5	3.25	5.05	92%
	정시	2.0	1.5	2.0	2.0	2.0	1.5	2.0	2.0	1.5	1.5	1.80		

*단위: M/M(Man Month)

<표 7> 제안 기법 적용 전·후의 Spring AOP XML의 LOC 비교

구 분	모집	개발자										평균	평균 합계	기법 적용 전 대비 LOC의 백분율
		1	2	3	4	5	6	7	8	9	10			
기법 적용 전	수시	650	654	623	765	707	862	837	715	740	835	738	748	100%
	정시	737	685	648	754	722	851	838	726	764	869	550		
기법 적용 후	수시	550	464	523	565	507	662	637	615	640	635	579	525	70%
	정시	437	473	438	434	452	521	538	426	464	529	471		

*단위: LOC(Line of Code)

AOP XML의 LOC가 제안 기법 적용 전 대비 약 30% 감소되었다.

<표 6>는 제안 기법 적용 전·후로 구분하여 모집 시기별 입시시스템 개발 및 수정에 소요된 시간을 M/M 단위로 측정하여 개발 노력으로 비교한 것이다.

제안 기법 적용 전 새로운 입시 요강에 맞춰 개발 및 수정 보완하는데 소요되는 평균 개발 노력의 합은 5.45M/M이었으나 제안 기법을 적용 후 5.05M/M로써 0.40M/M의 개발노력이 감소하였다. 제안 기법을 최초 적용한 수시 모집에서 제안 기법 적용 전의 AOP를 UML로 작성하는 노력이 소요된 것을 고려하면 개발 및 수정에 소요된 노력의 감소는 0.40M/M 이상이다. 제안 기법 적용 전 대비 개발 노력의 감소를 백분율로 환산하면 약 8%이상 감소하였다.

Krueger의 연구에서 소프트웨어 재사용은 대해 소프트웨어 개발 과정에서 소프트웨어 개발 기간을 단축시키고 소프트웨어의 생산성과 품질 향상, 유지보수 비용과 테스트 비용 등을 절감할 수 있다고 하였다[16]. 이에, 수시 모집 시 적용한 Spring AOP UML 모델을 정시 모집에 재사용하여 평균 개발 노력이 0.40M/M 이상의 기간과 비용을 감소할 수 있었다.

<표 7>는 제안 기법 적용 전과 후에 사용한 테스트 케이스 예를 나타낸 것이다.

<표 8>는 제안 기법 적용 전과 후 모든 입시 모집에 87개의 테스트 케이스를 가지고 테스트를 한 결과 발생된 결함 발생 건수와 결함 중요도를 비교한 것이다. 결함 중요도가 Major인 경우에는 발생된 결함으로 인하여 시스템 사용이 불가능하거나 정상적인 제어 상태로 복귀가 불가능한 경우이고, Minor인 경우에는 순간적으로 오동작이나 잘못된 반환 값이 발생 할 수 있으

<표 7> 테스트 케이스

테스트 케이스 ID	입력	예상 결과	실행 결과	특수 조건
TC 1	입학원서 파일을 선택한 후 [파일 일괄작업] 버튼을 클릭한다.	정상적으로 일괄작업 완료이며, 일괄작업으로 등록된 학생수를 표시한다.		입학원서 파일이 오류가 없는 상태
TC 2	입학원서정보 중 비고란에 '테스트'를 입력 후 [저장]버튼을 클릭한다.	정상적으로 저장이 완료되며, 정상 저장된 입학원서 정보를 표시한다.		
…	…	…	…	…
TC 87	성적순으로 조회를 위해 성적별 항목 선택후 [조회] 버튼을 클릭한다.	정상적으로 리스트가 조회가 되어 보고서가 표시된다.		고교성적이 입력되어 계산되어 있는 상태

<표 8> 결합 발생 건수와 결함 중요도 비교

구 분	모집	결함 발생 건수	결함 중요도별 건수		오류 발생 건수 합계	기법 적용 전 대비 결함 발생 건수의 백분율
			Major**	Minor***		
기법 적용 전	수시	38	28	86	86	100%
	정시	48	36	12		
기법 적용 후	수시	17	5	21	21	24%
	정시	4	0	4		

*단위: 건

** Major:결함으로 인한 시스템 사용 불가능한 경우, 정상적인 제어 상태로 복귀 불가능한 경우

*** Minor:순간적인 오동작인 경우, 잘못된 반환 값이 발생 할 수 있으나 곧 정상적인 제어상태로 복귀한 경우

나 곧 정상적인 제어 상태로 복귀한 경우이다. 제안 기법 적용 전에는 Spring AOP XML 작성 시 총 86건의 결함이 발생되었지만, Spring AOP UML 표현 기법과 Spring AOP XML 변환 기법을 사용할 때에는 총 21건의 결함이 발생되었다. 기법 적용 후 결함 발생 건수는 약 76% 감소하였다. 제안 기법 적용 후 총 21건의 결함은 제안 기법을 처음 도입하는 시기인 수시 모집에서 17건 정시모집에서 4건이 발생이 되었지만, 정시모집에서는 결함 중요도가 Major에 해당하는 결합이 발생이 되지 않아서 제안 기법 적응 과정상의 문제라고 간과 할 수 있다.

제안 기법 적용 후 <표 6>과 같이 개발 노력의 절감과 <표 8>와 같이 결함의 발생 건수가 감소되었다. 기법에 따라 Spring AOP XML 변환을 하면 잘못된 AOP 설정으로 인한 시스템의 다운 타임과 그에 따른 비용을 감소할 수 있었다.

제안 기법 적용 전, 개발자들은 AOP 명세화 방법이 혼용 작성된Spring AOP XML 코드를 통해서 프로그램의 이해와 추적을 하는데 어려움이 있었으나 제안 기법 적용 후에는 Spring AOP UML 모델링을 참조하여 프로그램을 이해하고 추적할 수 있었다. Peterson의 연구에기인하면 그래픽적 표현이 텍스트적 표현보다 가독성과 이해성이 좋다고 주장하였다[17]. 이에 따라 본 연구의 제안 기법은 AOP를 그래픽적 표현인 UML로 표현하여 텍스트 표현인 XML에 비해 가독성과 이해성이 좋다.

<표 9>은 관련연구와 문헌을 통해 AOP 평가 요소를 도출하였고, 기존 연구들과의 비교평가 한 것이다.

<표 9> AOP 모델 평가 요소와 기존 연구와의 비교 평가

구분	Naoyasuh	Groher	Jacobson	제안 기법
Aspect의 재사용	X	X	X	○
Aspect 가독성과 이해성	○	○	△	○
Aspect 패키지화 지원	X	○	X	○

* ○가능, △부분적 가능, X 불가능

Aspect의 재사용에 대한 평가 요소는 AOP UML 모델로부터 Aspect가 재사용되어 AOP 설정 파일의 명세화가 혼용되지 않고 작성됨을 평가할 수 있는 평가 요소가 필요하여 도출하였고, 이에 AOP UML 모델과 AOP 설정 파일간의 변환 기법 제공 여부로 평가하였다.

Aspect의 가독성과 이해성에 대한 평가 요소는 UML 모델만으로 AOP를 이해할 수 있는지를 평가할 수 있는 평가 요소가 필요하여 도출하였고, Aspect 상세 모델 지원 여부로 평가하였다.

Aspect의 패키지화 지원에 대한 평가 요소는 패키지는 소프트웨어를 기능적으로 분리하여 모델을 조직화하고 모델을 쉽게 이해하도록 도와준다는 Booch의 연구[13]로부터 도출하였고, 이에 Aspect에 대한 모델이 패키지로의 지원 여부로 평가하였다.

기존 연구는 AOP UML 모델이 제한적이거나 모델에 대한 명확한 지침이 없어서 AOP UML 모델로부터 Aspect를 재사용하여 AOP 설정 파일로 일관된 변환을 보장하기 어렵다.

Jacobson의 연구는 Aspect 식별을 통해 UML 모델로 제공하고 있지만, Aspect에 대한 상세 모델을 지원하지 않기 때문에 Aspect에 대한 가독성과 이해성 지원은 부분적으로만 가능하였다.

Groher의 연구는 Aspect에 대한 모델이 패키지로 지원되지만, 패키지 작성에 대한 명확한 지침이 없어 초기 개발자가 설계하기에는 어려움 있었다.

전체적으로 기존 연구와 제안 기법을 비교해 볼 때, 제안 기법은 AOP UML 모델로부터 Aspect를 재사용하여 AOP 설정 파일로 일관된 변환을 할 수 있는 변환 기법을 제공하고 있으며, Aspect에 대한 상세 모델의 지원과 UML 모델만으로 AOP를 이해할 수 있어서 Aspect에 대한 가독성과 이해성 향상이 가능 하였다.

하지만, 기존 연구는 AOP UML 모델과 AOP 설정 파일간의 변환 기법의 부재로 AOP UML 모델에서 AOP 설정 파일의 일관된 변환의 보장과 Aspect 모델의 활용측면에서 제안 기법이 기존 연구 보다 우수함을 알았다.

6. 결론

본 연구에서는 오픈 소스 프레임워크 기반의 Spring AOP 명세화 방법이 혼용되고 있어서 AOP의 유지보수를 위한 이해와 Aspect의 재사용이 어려운 문제점을 해결하기 위해서 AOP를 표현하는 주요 요소인 Aspect, Pointcut과 Advice를 UML과 XML로 표현할 수 있는 기법을 제안하였다.

기존 AOP 모델 연구의 Aspect 클래스화 문제점 분석을 통하여 UML 모델 기반의 AOP 모델로 모델간의 불일치를 제거하여 Aspect 모델의 재사용이 가능하고, Pointcut과 Advice에 대한 복합적인 속성이나 값 등의 AOP 모델로 이해가 가능한 Spring AOP UML 표현 기법을 제안하였다. 또한, Spring AOP UML 모델과 Spring AOP XML 코드 변환간의 관계를 정의하였다.

사례 연구를 통하여, 제안한 Aspect 패키지화한 Spring AOP UML 모델이 AOP 유지보수를 위한 이해와 Aspect 재사용에 용이함을 적용가능성 및 평가를 통해 증명을 하였으며, Aspect의 재사용 측면과 Aspect에 대한 가독성과 이해성을 지원측면, Aspect 패키지화 지원측면에서 기존 연구보다 우수함을 알았다.

그러나, 본 연구의 제안 기법은 다양한 시스템에 특수한 AOP 설정 요소들을 적용하는 연구가 미흡하였다.

향후 연구로는 본 연구에서 제안한 Spring AOP UML 표현 및 XML 변환 기법을 다양한 시스템과 타 AOP 지원프로그램에 적용을 통하여, AOP 지원 프로그램에 맞는 AOP 설정 파일을 자동 생성하는 자동화 도구에 대한 연구가 필요하다.

참 고 문 헌

[1] CRAIG WALLS, 'Spring in Action Second Edition', Manning, 2008.

[2] 박옥자, 유철중, 장옥배, "프로그램 개발 및 유지보수를 지원하는 횡단관심사 명세 기법", 정보과학회논문지 : 소프트웨어 및 응용 제 34 권 제 9 호, 2007.

[3] Timo Aaltomen, Joni Helin, Mika Katara, Pertti Kellomaki., "Coordinating Aspects and Objects", Electronic Notes in Theoretical Computer Science 68, No.3, 2003.

[4] Georgia, S., Sergio, S., Paulo, B., and Jaelson, C., "Separation of Crosscutting Concerns from Requirements to Design: Adapting and Use Case Driven Approach", Aspect-Oriented Requirements Engineering and Architecture Design Workshop, pp. 93-102, 2004.

[5] Mik Kersten, Gail C. Murphy, "Atalas: A Case Study in Building a Web-Based Learning Environment using Aspect-Oriented Programming", Technical Report Number TR-99-04, 1999.

[6] Jan Machacek, Jessica Ditt, Aleksa Vukotic, and Anirvan Chakraborty, 'Pro Spring 2.5', Apress, 2008.

[7] Naoyasu Ubayash, Tetsuo Tamai, Shinji Sano, Yusaku Maeno, Satoshi Murakami, "Model Evolution with Aspect-Oriented Mechanisms", Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE'05),IEEE, 2005.

[8] Spring AOP Schema, http://www.springframework.org/schema/aop/spring-aop-2.0.xsd.

[9] 김태웅, 김태공, "AOSD기반에서 Aspect의 동적결합을 위한 Connector", 한국정보처리학회, 2006.

[10] MDA Guide Version 1.0.1, http://www.omg.org/cgi

-bin/doc?omg/03-06-01.pdf, June, 2003.
[11] Iris Groher, Thomas Baumgarth, "Aspect-Orientation from Design to Code", in Proceedings of the Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design; AOSD, March, 2004.
[12] IvarPan-Wei'Aspect-Oriented Software Development with Use Cases', Addison Wesley, 2004.
[13] Grady Booch, James Rumbaugh, and Ivar Jacobson, 'The Unified Modeling Language User Guide', Addison-wesley, 1999.
[14] 박재성, 'Spring 프레임워크 워크북, 한빛미디어', 2006
[15] AspectJ Website, http://www.eclipse.org/aspectj/
[16] C. W. Krueger, "Software Reuse", ACM Computing Surveys, Vol.24, No.2, pp.131-184, Jun 1992.
[17] Barry T. Peterson, Steven J. Clancy, Kay Champion and Jerry W. McLarty, "Improving Readability of Consent Forms: What the Computers May Not Tell You", IRB: Ethics and Human Research, Vol. 14, No. 6, pp. 6-8, 1992.
[18] Ramnivas Laddad, 'AspectJ In Action: PRACTICAL ASPECT-ORIENTED PROGRAMMING', Manning, 2003.

이 후 재

2004년 : 숭실대학교 정보과학대학원 (공학석사)
2007년 : 숭실대학교 대학원 (박사과정수료)
2004년~현 재 : 대림대학 정보전략운영팀
관심분야 : 컴포넌트 기반 개발 (CBD), 소프트웨어 재사용, 소프트웨어 아키텍처, 소프트웨어 프로세스, 오픈 소스 소프트웨어 등

류 성 열

1998년~2000년 : 숭실대학교 정보과학대학원 원장
2004년~현재 : 한국품질재단 운영위원회 위원장
2006년~현재 : 공정거래위원회 성과관리위원회위원
2008년~현재 : 정보통신연구진흥원 비상임이사
1981년~현재 : 숭실대학교 컴퓨터학과 교수
관심분야 : 소프트웨어 유지보수, 소프트웨어 재사용, 오픈 소스 소프트웨어, 소프트웨어 재공학/역공학 등

김 종 배

2002년 : 숭실대학교 정보과학대학원 (공학석사)
2006년 : 숭실대학교 대학원 (공학박사)
2001년~현 재 : (주)이엔터프라이즈 대표이사
2004년~2006년 : 남서울대학교 컴퓨터학과 겸임교수
2006년~현 재 : 서울여자대학교 컴퓨터학부 겸임교수
2009년~현 재 : (사)해킹보안협회 학술연구위원장
관심분야 : 오픈 소스 소프트웨어, 정보보호(Personal Information), 유비쿼터스 컴퓨팅 등