EJB 컴포넌트의 맞춤 테스트 기법
(A Testing Technigue for Customized EJB Component)

윤 회 진 † 최 병 주 ††
(Hoijin Yoon)(Byoungju Choi)

요 약 '컴포넌트 기반 소프트웨어 개발'에서 컴포넌트 사용자는 컴포넌트 맞춤, 즉 컴포넌트를 특정 도메인 요구사항에 맞추는 작업, 을 반복적으로 수행함으로써, 소프트웨어를 개발한다. 따라서 컴포넌트 기반 소프트웨어 테스트의 주요 대상은 맞춤으로 변형된 컴포넌트에서 발생하는 오류이다. 본 논문에서는 엔터프라이즈 자바빈즈(Enterprise JavaBeans : EJB)를 기반으로 구체화된 EJB 컴포넌트 맞춤 테스트 기법을 개발한다. 본 기법은 EJB 컴포넌트의 인터페이스 가운데 맞춤 오류가 일어나는 곳에만 오류를 삽입하여, 맞추어진 컴포넌트와 오류가 삽입된 컴포넌트를 차별하는 테스트 케이스를 선정한다. 따라서 본 기법은 EJB 컴포넌트 맞춤에 의한 오류를 발견할 가능성이 높은 테스트 케이스를 선정할 수 있다. 또한 본 논문에서는 EJB 컴포넌트 맞춤 테스트 기법을 EJB 컴포넌트 맞춤에 적용한 사례를 보이고 이를 분석하여, 본 기법이 EJB 컴포넌트에 적용 가능한 것임을 보인다.

Abstract In Component-Based Software Development, the component user develops a component-based software through the customization, which modifies component to fit into the domain-specific requirement, for each component repeatedly. So it is the error caused by component customization that component-based software testing mainly focuses. In this paper, we propose a component customization testing technique applicable to EJB component architecture. Our technique selects the test cases for the customization error by injecting a fault to the specific part of component's interface. So our test cases have high effectiveness for detecting customization errors. This paper also implements a case study for appling our technique to EJB component in order to show that our technique is applicable to EJB compoent.

1. 서 론

CBSD(Component-Based Software Development)는 컴포넌트를 생성하여 제공하는 컴포넌트 제공자와 컴포넌트를 사용하는 컴포넌트 사용자라는 두개의 시각으로 볼 수 있다. 이 두 시각의 차이점 중의 하나가 소소 코드의 사용 가능성이다. 컴포넌트 제공자는 소스코드에 접근할 수 있고, 컴포넌트 사용자는 소스코드에 접근할 수 없다[1]. 컴포넌트 사용자는, 컴포넌트의 소스코드를 사용할 수 없기 때문에 컴포넌트 사용자 입장에서 수행하는 테스팅에 제약이 있다. 따라서 CBSD의 시각 가운데 특히 컴포넌트 사용자의 입장, 즉 컴포넌트 제공자에 의해 제공받은 컴포넌트를 사용하여 컴포넌트 기반 소프트웨어(CBS : Component-Based Software)를 구축할 때, 적용할 수 있는 새로운 컴포넌트 맞춤 테스트 기법 연구가 필요하다.

컴포넌트 기반 소프트웨어, 즉 CBS는 이미 만들어진 컴포넌트를 사용하여 구축되는 소프트웨어이다. 컴포넌트 사용자가 CBS를 개발할 때, 컴포넌트를 새로운 소프트웨어 도메인의 요구사항에 맞추기 위하여 맞춤(customization)하는 활동이 이루어진다. Paul Allen은 컴포넌트 맞춤을 CBSD의 주요작업으로 정의하였으며[2], Souza는 컴포넌트를 소프트웨어 구현 패키지로 정의하고, 컴포넌트가 지니고 있는 속성을 사용자의 요구에 따라 맞춤할 수 있어야 한다는 것을 컴포넌트가 가져야 하는 특성으로 제시하였다[3]. 결국, 컴포넌트 맞춤은, 1) 도메인 기반 요구사항에 따라 컴포넌트의 속성을 간

· 본 연구는 한국과학재단 목적기초연구(과제번호:2000-0-303-02-3)지원으로 수행되었음.
† 학생회원 : 이화여자대학교 컴퓨터학과
hoijin@mm.ewha.ac.kr
†† 종신회원 : 이화여자대학교 컴퓨터학과 교수
bjchoi@mm.ewha.ac.kr
논문접수 : 2000년 7월 27일
심사완료 : 2000년 12월 23일

단하게 수정하는 것을 포함하여, 2) 새로운 기능을 추가하거나 3) 기존의 기능을 변경하기 위해 또 다른 컴포넌트와 조립하는 작업들로서, 새로운 소프트웨어 개발에 컴포넌트를 재사용하기 위해 컴포넌트를 변형하는 일련의 활동을 지칭한다. 엔터프라이즈 자바빈즈 (Enterprise JavaBeans : EJB) 모두의 컴포넌트 아키텍처의 경우, 이 세 가지 맞춤의 유형을 모두 포함하고 있다[4].

논문 [5,6]에서는 공개하지 않는 컴포넌트의 핵심기능부분과 공개하는 컴포넌트의 인터페이스 부분을 일반적으로 표현하기 위하여 각각 블랙박스클래스와 화이트박스클래스로 정의하고, 컴포넌트 맞춤을 통해 변형된 화이트박스클래스에 오류를 삽입하여 맞춤으로 발생할 수 있는 오류를 테스트하는 테스트 케이스를 선정하는 기법을 제안하였다. 이것은 컴포넌트 맞춤에 효과적인 테스트 케이스를 선정한다는 장점을 갖는 기법이다. 그러나 이 기법은 컴포넌트의 일반적 특성에 기반하여 제안된 기법이므로, 실제 컴포넌트 기반 소프트웨어 개발에 적용하기에는 추상적이다. 본 논문에서는 추상적이었던 [5,6]를 토대로 하여, 현재 각광을 받는 컴포넌트 아키텍처인 EJB가 명세하는 컴포넌트에 적용할 수 있도록, 'EJB 컴포넌트의 맞춤 테스트 기법'을 제안한다. 또한 제안한 테스트 기법을 EJB 컴포넌트에 실제 적용한 적용사례를 보임으로써, 본 논문이 제안한 기법에 대한 이해를 돕고, 본 기법을 직접 EJB에 적용할 수 있음을 보인다.

본 논문은 2장에서 관련 연구를 기술하고, 3장에서는 본 논문에서 제안하는 EJB 컴포넌트 맞춤 테스트 기법을 개발한다. 4장에서는 그를 실제 컴포넌트 아키텍처에 적용하는 사례를 보이며, 마지막 5장에서는 본 논문의 결론 및 향후 연구 과제를 기술하겠다.

2. 관련 연구

소프트웨어 오류 삽입 기법은 프로그램 코드에 오류를 임으로 심어서 그 결과 시스템이 어떤 행동을 하는지를 관찰하여 테스팅을 수행하는 기법이다. 이는 특히 프로그램 수행 경로가 매우 복잡하고 오류가 쉽게 표현되지 않는 시스템에 적합한 방법이다. 컴포넌트도 공개되지 않은 부분을 갖음으로써, 그 수행경로를 정확하게 알 수 없기 때문에, 오류 삽입 기법이 적합하다. 또한 오류 삽입 기법에서 테스트 케이스의 적정성 측정의 예가 뮤테이션 테스트이다. 뮤테이션 테스트는 문법적 오류가 없는 뮤턴트와 원래 프로그램을 차별화하는 테스트 케이스를 추출한다.

따라서 컴포넌트 맞춤 테스트[5,6,7]는 오류 삽입 기법과 뮤테이션 테스트를 사용한다. 그러나 컴포넌트의 공개된 인터페이스의 모든 부분에 뮤테이션 기법을 적용하지 않고, 인터페이스 가운데 특정 부분을 오류 삽입 대상으로 선정하여 그곳에만 오류를 삽입한다. 여기에서 중요한 점은 오류 삽입 대상을 정확하게 파악하는데 있다. 컴포넌트 맞춤 테스트는 컴포넌트의 맞춤으로 변형된 인터페이스가 공개되지 않은 핵심기능 부분에 영향을 미치면서 컴포넌트 행위에 일으키는 오류를 테스트하는 컴포넌트의 핵심기능 부분과 인터페이스의 통합 테스트이다. 따라서 인터페이스를 구성하는 구성요소들 가운데, 컴포넌트의 핵심 기능 부분과 밀접한 관계를 맺는 부분을 오류 삽입 대상으로 선정하여 오류 삽입 기법과 뮤테이션 테스트 기법을 적용한다. 본 논문은 이 컴포넌트 맞춤 테스트를 EJB 기반으로 보완하고 구체화한다.

인터페이스를 변형 대상으로 하는 기존의 유사 연구 [8]가 있다. 본 논문과의 차이점을 기술한다. 연구[8]와 [5,6]은 다음 두 가지의 차이가 있다. 첫째, [8]의 인터페이스는 본 논문에서 대상으로 하는 컴포넌트의 인터페이스와는 다른 정의를 갖는다. [8]의 인터페이스는 모듈사이의 관계로서, '연결'의 의미를 갖으며, 호출 그래프에서 관계를 갖는 두 모듈사이의 호출 문장이 [8]의 인터페이스가 된다. 그러나 컴포넌트의 인터페이스는 공개되지 않는 컴포넌트의 핵심 기능 부분에 접근하지 않고, 컴포넌트의 행위와 모습을 변경시키는 기능을 하는 부분으로서, 매우 다양한 특성을 지닌 요소들로 이루어져 있으며, 각 컴포넌트 아키텍처마다 인터페이스 구현 기술을 달리하고 있다. 따라서 [8]을 컴포넌트 기반 소프트웨어에 직접 적용하기에는 무리가 있다. 둘째, [8]의 인터페이스 뮤테이션은 인터페이스에 해당하는 호출 문장을 이루는 요소들, 즉 함수 호출, 반환값, 광역변수를 오류가 존재하는 장소로 보고 그곳에 대한 뮤테이션을 수행한다. 그러나 본 논문에서는 인터페이스의 모든 요소에 오류를 삽입하지 않고, CBSD에서의 컴포넌트의 핵심 기능 부분과 인터페이스의 상호작용을 분석을 통해, 오류 삽입 장소를 선정하여 효율적인 컴포넌트 기반 소프트웨어 테스트 케이스를 선정하는데 목적을 두고 있다. 이렇게 함으로써, 오류를 발견할 가능성이 높은 테스트 케이스를 선정할 수 있으며, 뮤테이션 테스트의 문제점인 'computational bottleneck'[9]을 감소시킬 수 있다.

컴포넌트 특성을 반영하여 개발한 테스트 기법으로서 논문 [5,6,7]이 제시한 기법은 큰 의미를 갖고 있다. 그러나 이는 특정 컴포넌트 아키텍처의 명세를 기반으로

하지 않고, 컴포넌트에 대한 일반적인 특성을 기반으로 개발된 기법이므로 따라서 특정 컴포넌트 아키텍처를 이용하여 개발하는 컴포넌트 기반 소프트웨어 개발 현장에 직접 적용되기에는 추상적인 단점을 갖는다. 그에 반해 본 논문은 현재 각광을 받고 있는 EJB를 기반으로 구체적인 컴포넌트 맞춤 테스트 기법을 제안한다. [5,6,7]의 컴포넌트 맞춤 테스트 기법은 컴포넌트 맞춤 패턴을 객체지향 소프트웨어 개발을 위한 패턴과 컴포넌트 맞춤에 대한 정의를 참조하여 정의한 반면, 본 논문의 기법은 EJB 스펙을 분석하여 EJB에서 가능한 EJB 컴포넌트 맞춤 패턴을 정의하였다. 또한 [5,6,7]의 컴포넌트 맞춤 테스트 기법이 인터페이스를 일반적인 객체지향프로그램 코드로 보고 오류 삽입 대상과 연산자를 선정한 것에 비해, 본 논문의 기법은 EJB의 인터페이스를 구현하는 요소들과 EJB 컴포넌트 내부와의 상호작용을 분석하여 EJB의 오류 삽입 대상과 연산자를 선정하였다. 결국 본 논문의 기법은 [5,6,7]와 달리 EJB 컴포넌트 아키텍처에서 이루어지는 컴포넌트 맞춤을 테스트할 수 있도록 구체적인 기법이다.

3. EJB 컴포넌트 맞춤 테스트 기법

컴포넌트 제공자는 컴포넌트 사용자에게 컴포넌트의 일부는 공개하고 일부는 공개하지 않는다. 즉, 컴포넌트는 공개되는 부분과 공개되지 않는 부분으로 구현된다. EJB에서도 자바소스파일이 아닌 자바클래스 파일 형태로 컴포넌트 사용자에게 공개하지 않는 부분과, XML 소스 파일 형태로 컴포넌트 사용자에게 공개하는 부분으로 EJB컴포넌트를 구성한다. 따라서 EJB 컴포넌트에, [5,6,7]에서 정의한 블랙박스클래스와 화이트박스클래스를 그대로 적용한다. 또한 EJB에서 컴포넌트 사용자는 어셈블러가 하는 일을 수행한다.

정의 1. 블랙박스클래스 (B)

블랙박스클래스란 컴포넌트 사용자에게 공개되지 않는 부분으로서, 컴포넌트 사용자가 수정할 수 없는 부분이다. 이를 B로 표현한다. 일반적으로 컴포넌트 제공자는 컴포넌트의 핵심 기능 부분을 B로 만든다.

예 1.

그림1의 EJB 컴포넌트에서 리모트 인터페이스(remote interface), 홈 인터페이스(home interface), 빈 클래스(bean class)에 해당하는 파일들이 Trading 컴포넌트의 B이다. 이들은 자바 소스 파일이 아닌 자바 클래스 파일의 형태로 존재하여, 그 소스 코드를 컴포넌트 사용자에게 제공하지 않는다. ■

정의 2. 화이트박스클래스 (W)

화이트박스클래스란 컴포넌트 사용자에게 공개되는 부분으로서, 컴포넌트 사용자가 이를 수정하여 컴포넌트를 맞춤할 수 있도록 하는 부분이다. 화이트박스클래스를 W로 표현한다. 컴포넌트를 재사용하여 개발할 새로운 소프트웨어의 요구사항에 맞춤된 W를 cW라고 하고, 오류 삽입을 통해 오류가 삽입된 cW를 fW라고 한다. 일반적으로 컴포넌트 제공자는 컴포넌트의 인터페이스를 W로 만든다.

예 2.

그림1의 EJB 컴포넌트에서 디플로이먼트 디스크립터에 해당하는 파일이 Trading 컴포넌트의 W이다. Trading 컴포넌트의 W는 XML 코드를 갖는 파일로서 소스가 공개되어져 있다. 컴포넌트 사용자가 이 파일을 변형함으로써 Trading 컴포넌트를 맞춤할 수 있다. ■

정의 3. 컴포넌트 (BW)

컴포넌트는 B와 W가 결합된 단위로서 BW로 표현한다. 맞춤으로 W가 변형된 BW를 cBW라고 하고, 오류가 삽입된 cBW를 fBW로 표현한다.

예 3.

그림1의 EJB 컴포넌트는 예 1.의 B와 예 2.의 W로 구성되어져 있는 BW이다. ■

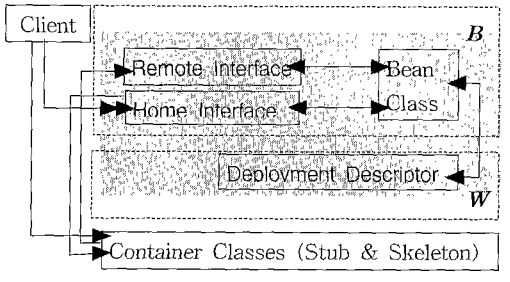

그림 1 EJB 컴포넌트의 B, W, BW

컴포넌트 기반 소프트웨어 테스트의 대상은 컴포넌트 맞춤으로 변형된 컴포넌트, 즉 cBW이다. cBW는 B와 cW로 이루어져 있고, B는 블랙박스클래스로서, 이미 오류가 없다고 전제되어져 있으며, cW는 화이트박스클래스로서 공개되어져 있으므로, 기존의 단위 테스트 기법으로 테스트할 수 있다. 따라서 B와 cW 각각은 오류가 없다고 가정해도 무리가 없다. 이들 B와 cW는 cBW로 통합되어, 컴포넌트 기반 소프트웨어를 구성한다. 비록 B와 cW 각각은 오류가 없는 단위일지라도, 이들이 통합되었을 때 예기치 못한 치명적인 오류가 발생할 수 있다. 이 오류는 B와 cW사이의 상호작용에서 발생하는 것으로서, B와 cW의 통합 테스트를 통해 해결할 수 있

다. 따라서 CBS 테스팅은 B와 cW의 통합 테스트이다. 컴포넌트 맞춤 테스트 기법[5,6]은, 그림2 (a)의 '컴포넌트 맞춤'에서 보듯이 컴포넌트 맞춤을 통해 변형된 컴포넌트, 즉 cBW의 오류를 테스트하기 위해, 그림2 (b)의 '오류 삽입'을 통해 cBW에 임의로 오류를 삽입하여 생성한 fBW와 cBW를 차별화하는 테스트 케이스를 선정한다. 선정된 테스트 케이스를 cBW에 적용시켜 그 결과가 예상 결과와 다를 경우, cBW에 오류가 있다고 판단한다.

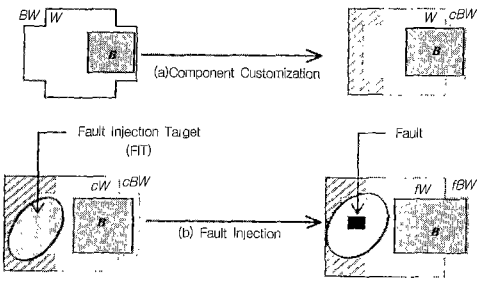

그림 2 컴포넌트 맞춤과 오류 삽입

컴포넌트 맞춤을 통해, BW를 cBW로 만드는 데에는 몇 가지 패턴이 있다. 이 패턴에 따라 cBW에 오류를 삽입하는 대상과 내용이 달라진다. 따라서 본 논문에서는 우선 EJB 컴포넌트의 맞춤 패턴을 정의하고, 그들에 따라 EJB 컴포넌트의 오류 삽입 대상을 선정한다.

3.1 EJB 컴포넌트의 맞춤 패턴

일반적으로 정의한 [5,6,7]의 컴포넌트 맞춤 패턴과 EJB 스팩을 참조하여, 다음의 4개의 EJB 컴포넌트의 맞춤 패턴을 다음과 같이 정의한다.

(1) EJB 맞춤 패턴 1

디플로이먼트 디스크립터의 <env-entry-value>의 값을 수정하여 EJB 컴포넌트의 속성 값을 변경하는 맞춤 패턴이다.

예.

다음은 TradeLimit이라는 속성의 값을 100으로 설정한 디플로이먼트 디스크립터이다. ■

```
....
<env-entry-name> TradeLimit </env-entry-name>
<env-entry-value> 100 </env-entry-value>
...
```

(2) EJB 맞춤 패턴 2

컴포넌트의 메소드들에 대한 접근 권한을 추가하거나 변경하는 맞춤 패턴이다. 디플로이먼트 디스크립터의 <security-role>부분을 수정하거나 추가하고, 이를 <role-link>로 설정하여 B가 참조하도록 맞춤한다.

예.

다음은 Deposit 메소드에만 접근 권한을 갖는 새로운 role인 X를 추가하고, 이를 client라는 role name에 연결시키는 맞춤이 이루어진 디플로이먼트 디스크립터이다. ■

```
....
<security-role>
    <role-name> X </role-name>
</security-role>
...
<method-permission>
    <role-name> X </role-name>
        <method>
        <ejb-name> statelessSession </ejb-name>
        <method-name> Deposit </method-name>
        </method>
</method-permission>
...
<security-role-ref>
    <role-name> client </role-name>
    <role-link> X </role-link>
</security-role>
...
```

(3) EJB 맞춤 패턴 3

EJB 컴포넌트 개발자에 의해 작성되어 컴포넌트와 함께 제공되는 여러 개의 EJB class들 가운데, 도메인 specific requirement에 따라 적당한 것을 application assembler나 deployer가 선택하여 해당 인터페이스와 연결시킨다. 이는 하나의 인터페이스에 대해 여러 개의 EJB class를 컴포넌트 개발자가 제공할 때 가능하다.

예.

다음은 Record 인터페이스가 사용하는 EJB class가 RecordBean이었는데, 이를 RecordBean2로 변경하기 위해, Deployment Descriptor의 해당 <ejb-class>를 수정하는 맞춤을 한 경우이다. 이때 RecordBean2 클래스가 이미 구현되어 EJB 컴포넌트안에 포함되어져 있다. ■

```
<session>
<ejb-name>RecordResult</ejb-name>
<home>example.ejb.basic.statelessSession.RecordHome</
home>
<remote>examples.ejb.basic.statelessSession.Record</re
mote>
<ejb-class>examples.ejb.basic.statelessSession.Recor
dBean2</ejb-class>
...
</session>
```

(4) EJB 맞춤 패턴 4

원하는 기능을 수행하는 또 다른 EJB 컴포넌트를 조립시킴으로써, EJB 컴포넌트를 맞춤하는 패턴이다.

예.

RecordResult라는 EJB 컴포넌트를 사용하고 있는 Trader라는 EJB 컴포넌트가 RecordResult2라는 EJB

컴포넌트를 사용하도록 맞춤할 수 있다. 다음은 이 맞춤에 대한 디플로이먼트 디스크립터의 일부이다. 이때 RecordResult2 컴포넌트는 맞춤을 위해 새로 작성될 수도 있고, 기존에 작성된 EJB 컴포넌트일 수도 있다. 즉, 조립된 컴포넌트를 다른 것으로 바꾸어줌으로써, 컴포넌트 맞춤을 수행한다. ■

```
<ejb-ref>
   <ejb-ref-name>ejb/Record</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
<home>examples.ejb.basic.statelessSession.Record2Home
</home>
<remote>examples.ejb.basic.statelessSession.Record2</re
mote>
   <ejb-link>  RecordResult2 </ejb-link>
</ejb-ref>
....
<session>
   <ejb-name>RecordResult2</ejb-name>
<home>example.ejb.basic.statelessSession.RecordHome</
home>

<remote>examples.ejb.basic.statelessSession.Record</rem
ote>
<ejb-class>examples.ejb.basic.statelessSession.TraderBea
n</ejb-class>
...
</session>
```

EJB 맞춤 패턴 1과 2는 컴포넌트 개발자에 의해 제공되는 내용을 사용자가 단지 선택하거나 설정하는 수준의 맞춤 패턴이므로, 컴포넌트 맞춤 패턴[5,6,7]의 $Pattern.syn_1$와 $Pattern.sem_1$에 해당한다. EJB 맞춤 패턴 3과 4는 컴포넌트의 기능을 추가하거나 변화를 주는 맞춤 패턴이므로 [5,6,7]에서 정의한 컴포넌트 맞춤 패턴의 $Pattern.syn_2$, $Pattern.syn_3$과 $Pattern.sem_2$에 해당한다.

3.2 EJB 컴포넌트의 오류 삽입 대상과 오류 삽입 연산자

본 논문에서는 오류를 단순히 cW 전체 혹은 맞춤을 위해 수정된 코드, 즉 맞춤 코드에만 삽입하지 않는다. 물론 그곳에 오류를 삽입하여 선정된 테스트 케이스들도 컴포넌트 맞춤으로 인한 오류를 감지할 수 있을지도 모른다. 하지만 효율적인 테스트 케이스를 선정하기 위해 CBSD를 통해 생성되는 cBW의 cW와 B의 상호작용을 분석하고, 그를 토대로 효율적인 오류 삽입 대상을 선정해야 한다. [5,6,7]에서 선정한 오류 삽입 대상과 연산자는 인터페이스를 객체지향프로그램으로 보고 선정하였으므로, EJB에 직접 적용할 수 없다. EJB 인터페이스는 XML로 구현되어져있다. 따라서 본 논문에서는 EJB 컴포넌트를 위한 오류 삽입 대상과 연산자를 새롭게 정의한다.

EJB 컴포넌트의 B와 cW사이의 상호작용은 cW의 특정 항목 (cW를 이루는 구성 단위, 예를 들어 EJB의 경우 하나의 XML element) 하나를 중심으로 한다. B는 cW의 항목들을 참조하여 새로운 요구사항에 맞는 cBW로서 작동한다. 이때 B는 cW의 모든 항목들을 직접 참조하지는 않는다. B는 cW의 항목들 가운데 특정 항목 하나인 'x'를 참조함으로써, x와 관련된 다른 항목들을 간접적으로 참조하게 된다. 본 논문에서는 x를 직접 참조 항목으로 정의하고, x와 관련되어 x를 통해 B에 참조되는 cW의 또 다른 항목들을 x에 대한 간접 참조 항목으로 정의한다.

정의 4. 직접 참조 항목 (DRE)

cW를 구성하는 항목으로서, B에 의해 직접 참조되는 항목을 말한다. 이를 DRE로 지칭한다.

예 4.

EJB의 컴포넌트 맞춤 패턴2으로 생성되는 cBW에서, B는 cW의 <role-link>항목을 참조함으로써, 관련된 <role-name>,<method-permission>등의 항목에 영향을 받는다. 이 경우 <role-link>가 DRE이다. ■

정의 5. 간접 참조 항목 (IRE)

cW를 구성하는 항목으로서, DRE에 자신의 맞춤으로 인한 수정 결과를 반영하여, DRE를 통해 B에 간접적으로 참조되는 항목을 말한다. 이를 IRE로 지칭한다.

예 5.

EJB의 컴포넌트 맞춤 패턴 2로 생성되는 cBW에서, B는 cW의 <role-link>항목을 참조함으로써, 관련된 <role-name>, <method-permission>등의 항목에 영향을 받는다. 이 경우 <role-link>가 DRE라고 하면, <role-link>와 관련된 <role-name>, <method-permission>등 항목들이 IRE이다. ■

컴포넌트 맞춤을 위해 수정된 IRE에 오류가 있을 경우, 그 오류의 결과는 DRE를 통해 B에 영향을 미친다. 따라서 DRE에 오류를 심음으로써 생성한 테스트 케이스는 DRE뿐 아니라 이와 관련된 IRE에 존재하는 오류까지 감지할 수 있을 것이다. 따라서 본 논문은 앞서 기술한 B와 cW의 상호 작용을 기반으로 오류가 삽입되는 항목을 정의 6.으로 정의한다.

정의 6. 오류 삽입 대상 (FIT)

cW의 항목가운데 오류를 삽입하는 대상이 되는 항목으로서, 이를 FIT로 지칭한다. 본 논문에서는 cW의 DRE를 FIT로 한다.

예 6.

예 4.에서 보았듯이, EJB의 컴포넌트 맞춤 패턴 2에서, <role-link>항목이 DRE가 된다. 따라서 EJB 컴포넌트 맞춤 패턴 2의 FIT는 <role-link>가 된다. ■

오류를 삽입하기 위해서는 오류가 삽입되는 장소뿐

아니라, 오류를 어떻게 삽입하느냐를 결정해야 한다. *FIT*에 오류를 삽입하는 방법은 컴포넌트 아키텍처가 *W*를 어떻게 구현하였느냐에 따라 달라진다. EJB의 경우 *W*는 XML로 구현된다. 따라서 XML의 DTD에 어긋나지 않도록 *FIT*가 되는 항목의 내용을 변경한다. 오류가 삽입된 *fBW*가 정적분석에서 오류없이 구축되어져야 하므로 각 컴포넌트 아키텍처의 *W*의 문법에 어긋나지 않도록 오류를 삽입해야 한다. 따라서 *W*를 이루는 항목들의 특성에 따라 오류를 삽입하는 구체적인 방법이 달라진다. 본 논문은 일반적으로 오류를 삽입하는 방법을 정의 7.과 같이 정의하고, 이를 오류 삽입 연산자로 지칭한다. 정의 7.은 오류 삽입 연산자가 가져야 하는 원칙으로서, 이를 기반으로 각 컴포넌트 아키텍처에서 구체적인 연산자를 정의해야 한다.

정의 7. 오류 삽입 연산자 (*FIO*)

오류 삽입 연산자는 *FIT*에 오류를 삽입하는 방법을 표현한 것이다. 오류 삽입은 *fBW*에서 문법적 오류가 발생하지 않도록 이루어져야 한다. 본 논문은 이를 *FIO*로 표현하고, 이는 각 컴포넌트 아키텍처의 *W*의 구현에 따라 달라진다.

예 7.

예 6.의 *FIT*, <role-link>항목에 오류를 삽입하는 방법은 오류가 삽입된 후에도 문법적 오류가 없어야 한다. <role-link>항목은 role의 이름을 갖는 항목이므로, role의 이름이 아닌 다른 값이 올 경우, *fBW*는 문법적 오류를 갖게 된다. 따라서 <role-link>항목의 값을 또 다른 role의 이름으로 대치하는 연산자를 *FIO*로 한다. ■

3.1절에서 정의한 EJB 맞춤 패턴에서의 *FIT*와 *FIO*는 정의 6.과 정의 7.에 따라, 표1과 같이 추출할 수 있다.

표1에서 *FIT*에 해당하는 부분은 디플로이먼트 디스크립터의 항목들이고, *FIO*는 그곳에 정의7을 적용하여 만들어진 오류 삽입 연산자이다. REV는 <env-entry-value>항목의 값을 다른 값으로 바꾸어주는 연산자로서, 각 <env-entry-type>에 명시된 형식내의 값으로 바꾸어준다. RRL은 <role-link>항목의 값을 다른 값으로 바꾸어주는 것으로서, <method- permission>내에 설정된 <role-name>의 값들 중에 선택한 값으로 바꾸어주는 연산자이다. REC는 <ejb- class>항목의 값을 다른 값으로 바꾸어주는 연산자로서, 컴포넌트 제공자에 의해 JAR파일내에 포함된 bean class파일 이름 중에 선택한 이름으로 바꾸어준다. REL은 <ejb-link>항목의 값을 다른 값으로 바꾸어주는 것으로서, 디플로이먼트 디스크립터에 명시된 다른 EJB 컴포넌트 이름으로 바꾸어주는 연산자이다.

표 1 EJB의 *FIT*와 *FIO*

EJB 맞춤패턴	FIT	FIO
1	<env-entry -value>	REV (Replace to another Environment entry Value)
2	<role-link>	RRL (Replace to another Role Link)
3	<ejb-class >	REC (Replace to another Ejb Class)
4	<ejb-link>	REL (Replace to another Ejb Link)

3.3 Test Case Selection

컴포넌트 기반 소프트웨어의 테스트 케이스는 메소드 시퀀스 형태이다. 컴포넌트 맞춤 패턴에 따른 다양한 *FIO*들을 적용하여 생성된 *fBW*와, 테스트 대상이 되는 *cBW*를 차별화할 수 있는 메소드 시퀀스 형태의 테스트 케이스를 추출한다. 컴포넌트 기반 소프트웨어 테스트 케이스 선정 기법은 다음과 같이 [5,6,7]에서 정의한 것을 그대로 사용하였다.

정의 8. 컴포넌트 기반 소프트웨어 테스트 케이스 선정 기법

컴포넌트 기반 소프트웨어를 개발하기 위해 CBSD에서 맞춤된 *BW*를 *cBW*라고 하고, *FIO*를 사용하여 *cBW*의 *FIT*에 오류가 삽입된 *cBW*를 *fBW*라고 할 때, 컴포넌트 기반 소프트웨어 테스트 기법에 의해 선정되는 테스트 케이스, *TC*는 *cBW*와 *fBW*를 차별화하는 일련의 메소드 순서들로서 아래와 같다.

$TC = \{ x \mid cBW(x) \neq fBW(x),$ where x is a method sequence.$\}$

4. 적용 및 분석

본 논문에서 개발한 EJB 컴포넌트의 맞춤 테스트 기법을 적용하는 예를 보임으로써, 본 기법을 구체적으로 이해할 수 있도록 하고, 본 기법의 테스트 케이스가 EJB 컴포넌트 맞춤으로 발생한 오류를 감지함을 보인다.

4.1 대상 컴포넌트

주식 트레이딩을 수행하는 EJB 컴포넌트인 'Trading 컴포넌트'에 본 기법을 적용한다. 이 Trading 컴포넌트는 EJB 아키텍처에 따라 하나의 JAR파일로 제공되며, JAR파일 내에는 컴포넌트의 블랙박스클래스에 해당하는 리모트 인터페이스, 홈 인터페이스, 빈 클래스가 소스파일 없이 자바 클래스 파일 형태로 존재하고, 화이트박스클래스에 해당하는 디플로이먼트 디스크립터가

XML 소스파일로 존재한다. 본 적용사례는 EJB 컨테이너와 서버로 가장 많이 사용되는 Weblogic을 이용하고, 특히 최근 EJB 1.1 버전을 구현한 Weblogic 버전 5.1을 설치하여 그가 제공하는 EJB Deployer를 이용한다.

(1) Trading 컴포넌트 : *BW*

그림3은 Trading 컴포넌트를 나타내고 있다.

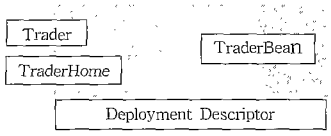

그림 3 Trading 컴포넌트 구성요소

이는 Trader_Session.jar라는 JAR 파일로서 제공되며, 그 안에는 리모트 인터페이스, 홈 인터페이스, 빈 클래스, 디플로이먼트 디스크립터 역할을 하는 4개의 파일들이 존재한다. 즉 Trading 컴포넌트는 각각 Trader.class, TraderHome.class, TraderBean.class, ejb-jar.xml로 구성된다.

(2) Trading 컴포넌트의 *B*

정의1.에 따라 *B*에 해당하는 부분은 컴포넌트 제공자가 컴포넌트 사용자에게 공개하지 않는 부분으로서 수정 불가능한 부분이다. EJB 컴포넌트 제공자는 디플로이먼트 디스크립터를 제외한 부분에 대한 수정을 컴포넌트 사용자에게 허락하지 않는다. 따라서 그림3에서 Trading 컴포넌트의 *B*는 리모트 인터페이스, 홈 인터페이스, 빈 클래스에 해당하는 Trader.class, TraderHome.class, TraderBean.class 파일들이다. EJB 컴포넌트는 컴포넌트의 구현 내용을 Java의 클래스 파일 형태로 갖고 있어서, 그 내용을 블랙박스화한다

(3) Trading 컴포넌트의 *W*

정의2.에 의해, *W*에 해당하는 부분은 디플로이먼트 디스크립터이다. 이는 컴포넌트의 인터페이스에 해당하는 부분으로서, EJB 컴포넌트 제공자가 컴포넌트 사용자에게 변경을 허락하는 부분으로서 소스코드가 공개되는 부분이다. 그림3에서 Trading 컴포넌트의 *W*는 디플로이먼트 디스크립터에 해당하는 ejb-jar.xml파일이다. EJB 컴포넌트는 컴포넌트의 인터페이스를 XML파일의 소스코드 형태로 갖고 있어서, 그 내용을 수정하여 컴포넌트를 변형시킬 수 있도록 한다

(4) 맞춤된 Trading 컴포넌트 : ***cBW***

본 기법은 *cBW*에 적용된다. 따라서 사례 연구를 위해 Trading 컴포넌트에 EJB 컴포넌트 맞춤 패턴 2에 해당하는 맞춤을 수행하여 *cBW*를 생성한다. 도메인의 특정 요구사항에 따라, Trading 컴포넌트의 W에 buy와 sell만을 수행하도록 제한된 자격을 갖는 role을 추가하여 *cW*를 만든다.

본 사례연구에서는 테스트 결과를 보이기 위해 *cBW*가 오류를 갖도록 하였다. 그림4는 Weblogic 5.1의 EJB Deployer를 사용하여 이 맞춤을 수행하는 모습이다. 그림 4에서 보듯이 컴포넌트 맞춤을 통해, 새로 정의한 role Y에 buy와 sell 메소드에 대한 권한을 부여하고자 했으나, buy 메소드 권한만을 부여하는 오류를 범했다. 이 오류는 *cW*에 대한 단위 테스트에서는 감지되지 않는 오류이다. EJB Deployer는 맞춤으로 생성되는 *cW*에 대한 컴파일를 수행하고, 그 결과 오류가 없을 때, *cBW*를 생성한다. 즉, *cW*에 대한 단위테스트는 EJB Deployer에 의해 수행되는 것이다. 이 오류를 갖는 *cW*는 EJB Deployer내에서 아무런 문제없이 *cBW*로 생성된다. 그러나 이 오류는 *B*와의 상호작용을 통해 CBSD로 구현된 컴포넌트 기반 소프트웨어에서 치명적인 오류를 일으킬 수 있다. 본 논문에서 제안한 테스트 기법은 이와 같은 오류를 테스트한다.

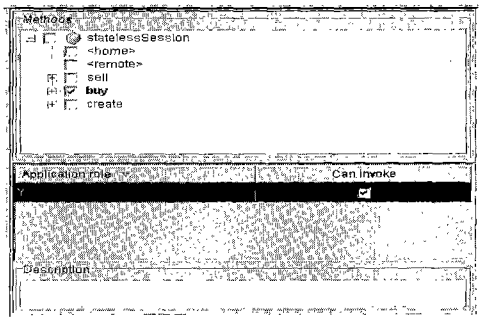

그림 4 Weblogic의 Deployer를 통한 맞춤

그림5는 그림4로 맞춤된 Trading 컴포넌트의 *cW*코드이다. 특히 굵은체로 표현된 것이 그림 4 맞춤으로 생성되거나 변경된 맞춤 코드이다. 그림 5의 a는 새로운 role Y를 선언하는 부분이고, b는 선언한 role Y가 수행할 수 있는 method들을 명시하는 부분이다. 그림 5의 c는 lowQual이라는 role을 수행할 때, role Y가 적용되도록 하는 것이다. 이 a,b,c로 인해 lowQual이라는 role을 갖는 사람은 b에 명시한 메소드만을 수행할 권한이 주어지는 것이다.

그림5와 Trading 컴포넌트의 *B*로 이루어진 *cBW*에 본 기법을 적용한다.

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enter
JavaBeans 1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
 <small-icon>images/green-cube.gif</small-icon>
 <assembly-descriptor>
 <security-role>
 <role-name> X </role-name>
 </security-role>
 <security-role>
 <role-name> Y </role-name>
 </security-role>                                  } a
 <method-permission>
 <role-name> X </role-name>
 <method>
 <ejb-name> statelessSession </ejb-name>
 <method-name> * </method-name>
 </method>
 </method-permission>
 <method-permission>
 <role-name> Y </role-name>
 <method>
 <ejb-name> statelessSession </ejb-name>           } b
 <method-name> buy </method-name>
 </method>
 </method-permission>
 <container-transaction>
 <method>
 <ejb-name>statelessSession</ejb-name>
 <method-intf>Remote</method-intf>
 <method-name>*</method-name>
 </method>
 <trans-attribute>Required</trans-attribute>
 </container-transaction>
 </assembly-descriptor>
 <enterprise-beans>
 <session>
 <small-icon>images/orange-cube.gif</small-icon>
 <ejb-name>statelessSession</ejb-name>
 <home>examples.ejb.basic.statelessSession.TraderHome</home>
 <remote>examples.ejb.basic.statelessSession.Trader</remote>
<ejb-class>examples.ejb.basic.statelessSession.TraderBean</ejb-class>
 <session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
<env-entry>
 <env-entry-name>tradeLimit</env-entry-name>
 <env-entry-type>java.lang.Integer</env-entry-type>
 <env-entry-value>500</env-entry-value>
 </env-entry>
 <security-role-ref>
 <role-name> lowQual </role-name>                  } c
 <role-link> Y </role-link>
 </security-role-ref>
 /session>
 </enterprise-beans>
 </ejb-jar>
```

그림 5 Trading 컴포넌트의 맞춤 후의 코드 (cW)

4.2 EJB 컴포넌트의 맞춤 테스트

CBSD에서 Trading 컴포넌트를 재사용하여 컴포넌트 기반 소프트웨어를 개발할 때 발생하는 오류, 즉 맞춤된 컴포넌트인 cBW에 존재하는 오류를 테스트하기 위해, 그림5에 본 기법을 적용한다.

(1) 오류 삽입 대상 : ***FIT***

EJB 맞춤 패턴 2에 의해 컴포넌트 맞춤된 Trading 컴포넌트, 즉 cBW의 FIT는, 표1에 따라 그림5 cW의 <security-role-ref><role-name>lowQual</role-name><role-link> Y </role-link></security-role-ref>에서 <role-link> Y </role-link>이다.

(2) 오류 삽입 연산자 : ***FIO***

표1에 따라 해당 FIT의 FIO는 RRL로서, 다른 role link로 대치하는 연산자이다. 따라서 FIT, <role-link> Y </role-link>에 RRL을 적용하면, FIT <role-link> Y </role-link>에 다른 role X를 오류로 삽입하여 <role-link> X </role-link>로 만든다. 이렇게 오류가 삽입된 cBW는 fBW가 된다. 오류 삽입에 의해, cW의 <role-link> Y </role-link>이 fW에서 <role-link> X </role-link>로 된다.

표 2 그림5 cBW에 대한 테스트 케이스의 일부

테스트 케이스, x	$cBW(x)$	$fBW(x)$
lowQual : sell("BEAS",100)	permission denied	"Selling 100 shares of BEAS"
lowQual : setLimit(100)	permission denied	"Setting 100 to tradeLimit"
lowQual : buy("MS",300),lowQual:sell ("MS",300)	permission denied	"Attempt to sell 300 is greater than limit of 175"
highQual : sell("BEAS",300), lowQual:sell("BEAS",300)	permission denied	"Attempt to sell 300 is greater than limit of 250"
admin:buy("MS",200), admin:buy("BEAS",300), lowQual:setLimit(150)	permission denied	"Setting 500 to tradeLimit"
highQual:sell("MS",500),low Qual:sell("MS"500)	permission denied	"Attempt to sell 500 is greater than limit of 500"
admin:setLimit(200), lowQual:sell("BEAS",150)	permission denied	"Attempt to sell 150 is greater than limit of 100"
highQual:setLimit(100), highQual:sell("BEAS",100), lowQual:setLimit(500)	permission denied	"Setting 500 to tradeLimit"
highQual:setLimit(100),low Qual:setLimit(500)	permission denied	"Setting 500 to tradeLimit"
lowQual:buy("BEAS",200), lowQual:sell("BEAS",100)	permission denied	"Selling 100 shares of BEAS"
highQual:setLimit(1000), lowQual:sell("AMZN",300)	permission denied	"Selling 300 shares of AMZN"
lowQual : sell("MS",500)	permission denied	"Attempt to sell 500 is greater than limit of 500"
lowQual : setLimit(1000)	permission denied	"Setting 1000 to tradeLimit"
lowQual : buy("BEAS",200),lowQual:s ell("MS",300)	permission denied	"Attempt to sell 300 is greater than limit of 175"
highQual : sell("BEAS",300), lowQual:sell("MS",300)	permission denied	"Attempt to sell 300 is greater than limit of 250"
admin:buy("MS",200), admin:buy("BEAS",300), lowQual:setLimit(200)	permission denied	"Setting 200 to tradeLimit"
highQual:sell("MS",500),low Qual:sell("MS"500)	permission denied	"Attempt to sell 500 is greater than limit of 500"
Admin:setLimit(200), lowQual:sell("BEAS",150)	permission denied	"Attempt to sell 150 is greater than limit of 100"
admin:buy("MS",200), admin:buy("BEAS",300), lowQual:setLimit(200)	permission denied	"Setting 200 to tradeLimit"
highQual:sell("MS",500),low Qual:sell("MS",500)	permission denied	"Attempt to sell 500 is greater than limit of 500""Setting 200 to tradeLimit"

(3) 테스트 케이스 : *TC*

그림5에 나타낸 *cW*를 갖는 *cBW*를 테스트하기 위해, 정의 8.에 따라 *fBW*와 *cBW*를 차별화하는 테스트 케이스를 선정한다. 예를 들어 lowQual:sell(BEAS,100)를 *cBW*에 적용한 결과가 'permission denied'이고, lowQual:sell(BEAS,100)를 *fBW*에 적용한 결과가 'Selling 100 shares of BEAS'이므로, 정의 8.에 따라 lowQual:sell(BEAS,100)는 테스트 케이스로 선정된다. 표2는 이렇게 선정된 많은 테스트 케이스들의 일부만을 나타낸다. 표2의 테스트 케이스들 가운데 본 사례연구에서는 임의로 테스트 케이스 lowQual:sell(BEAS,100)을 *cBW*에 적용하여 테스트하는 것을 보인다.

4.3 결과 분석

테스트 케이스를 *cBW*에 적용한 결과와 그에 대한 컴포넌트 사용자의 기대 결과가 다르다면 수행한 맞춤에 오류가 있는 것이다. 표2와 같이 선정한 테스트 케이스 가운데 lowQual:sell(BEAS ,100)을 임의로 선택하여 *cBW*에 적용한 결과는 'permission denied' 이었고, lowQual:sell(BEAS ,100)에 대해 컴포넌트 사용자가 *cBW*에게 기대하는 결과는 'Selling 100 shares of BEAS'이다. 따라서 컴포넌트 사용자가 맞춤을 수행한 *cBW*에는 잘못된 맞춤이 이루어진 것임을 알 수 있다. 즉 본 기법으로 선정한 테스트 케이스로 *cBW*에 오류가 있음을 감지할 수 있다. *cBW*에 존재하는 오류는 4.1절에서 설명한대로 <method-name>이 누락된 것으로서 <method-permission> 항목 내에 존재한다.

또한 본 기법에서의 f*BW*를 만들기 위해 임의로 오류를 <role-link>에 심었다. 즉 오류가 존재하는 곳과 *FIT*는 전혀 다른 곳이다. 본 기법은 해당 패턴의 맞춤 코드 어느 곳에 오류가 있던지, 각 *FIT*에만 오류를 심어서 추출한 테스트 케이스로 이를 테스트할 수 있다. 이에 대한 체계적인 분석을 통해 본 기법의 *FIT*에 대한 정당화 작업을 진행하고 있다.

5. 결론 및 향후 연구 과제

컴포넌트 사용자는, 컴포넌트의 소스코드를 사용할 수 없기 때문에 컴포넌트 사용자 입장에서 수행하는 테스팅에 제약이 있다[1]. 따라서 특히 컴포넌트 사용자의 입장, 즉 컴포넌트 제공자에 의해 제공받은 컴포넌트를 사용하여 CBS를 구축할 때 적용할 수 있는 새로운 테스트 기법 연구가 필요하다. 컴포넌트 사용자는 컴포넌트 맞춤, 즉 컴포넌트를 특정 도메인 요구사항에 맞추는 작업, 을 반복적으로 수행함으로써, 컴포넌트 기반 소프트웨어를 개발한다. 따라서 컴포넌트 기반 소프트웨어 테스트의 주요 대상은 맞춤으로 변형된 컴포넌트가 된다. 컴포넌트 맞춤으로 변형된 컴포넌트를 테스트하기 위한 컴포넌트 맞춤 테스트 기법이 필요하다. 이미 컴포넌트의 일반적인 특성을 기반으로 컴포넌트 맞춤 테스트 기법[5,6]이 제안되었다. 이는 컴포넌트 맞춤에 효과적인 테스트 케이스를 선정하는 기법이긴 하지만, 구체적이지 못하여, 실제 CBSD에 적용하기 어렵다. 본 논문에서는 EJB 컴포넌트 아키텍처에 적용할 수 있는 EJB 컴포넌트 맞춤 테스트 기법을 개발하였다.

EJB의 *W*를 분석하여 *W*의 각 항목들의 특성에 따라 본 논문에서 정의한 DRE와 IRE를 선정하여, EJB만의 *FIT*를 추출하였고, 그에 대한 *FIO*도 정의하였다. 이를 weblogic 5.1을 이용한 EJB 컴포넌트에 적용하여, 실제 CBSD 개발 환경에 본 기법이 적용될 수 있음을 보이고, 본 기법에 대한 보다 구체적인 이해를 가능하게 하였다. 본 논문은 실제 개발 현장에 적용할 수 있는 기법을 제시하기 위해 EJB를 위한 *FIT*와 *FIO*를 선정하여 EJB에서의 테스트 사례를 보였다. 본 논문에서 정의한 *FIT*와 *FIO*는 일반적인 컴포넌트의 특성을 기반으로 하였으므로, EJB외의 다른 컴포넌트 아키텍처에도 적용되어, 각각의 특정화된 *FIT*와 *FIO*를 선정할 수 있다.

CBSD에서의 테스트의 중요성은 인식되고 있으나, 구체적인 테스트 기법에 대한 연구가 미흡한 현 시점에서, 본 기법은 반드시 필요한 연구 결과라고 할 수 있다. 향후 본 기법의 효율성을 증명하는 실험을 수행하여, 본 기법이 개발 비용 절감 효과를 기대하는 CBSD에 충분히 적용 할 수 있음을 보일 것이다. 컴포넌트 기반 소프트웨어 개발이 추구하는 근본적인 목적 -즉 개발 비용 감소-를 역행하지 않아야 하므로, 테스트 기법이 선정하는 테스트 케이스의 효율성도 중요한 이슈가 된다. 효율적인 테스트 기법은 보다 적은 수의 테스트 케이스를 이용하여 보다 많은 수의 오류를 감지할 수 있어야 한다. 테스트 케이스가 많을수록 상식적으로 많은 수의 오류를 감지할 수 있으나, 테스트 비용을 고려했을 때, 이러한 테스트 케이스 선정은 비효율적이다. 즉, 효율적인 테스트 기법은 적절한 수의 테스트 케이스를 선정하여 최대의 오류 감지 효과를 본다. 향후 본 논문에서 제안하는 테스트 기법의 효율성을 실험을 통해 증명할 것이다. 실험은 다양한 실험 집합들을 대상으로 수행하여, 실험의 객관성을 유지한다. 나아가 본 기법에 대한 다양한 사용자

검증을 수행한 후, 본 기법을 자동으로 수행할 수 있는 자동화 도구를 구현할 계획이다.

참 고 문 헌

[1] Mary Jean Harrold, "Testing: A Roadmap," In Future of Software Engineering, 22nd International Conference on *Software Engineering,* June 2000.

[2] Paul Allen, "Practical Strategies for Migration to CBD," IT Journal Distributed Component Systems, 1999.

[3] Desmon F.D'Souza and A.C.Wills, *Object, Components, and Frameworks with UML,* Addison-Wesley, 1998

[4] Sun Microsystems, Enterprise JavaBeans Specification 1.1, at URL: http://www.javasoft.com

[5] Hoijin Yoon, Byoungju Choi, "A Test Technique for Component Customization Failure," Joutnal of Korea Information Science Society, 27(2):148-156, 2000

[6] Hoijin Yoon and Byoungju Choi, "Inter-class Test Technique between Black-box-class and White-box-class for Component Customization Failures," in *Proc. 6th APSEC*, page162-165, Japan, Dec 8-10, 1999

[7] Hoijin Yoon and Byounju Choi, "Component Customization Testing Technique Using Fault Injection Technique and Mutation Test Criteria," in *Proc. Mutation2000*, USA, Oct.6-7, 2000

[8] Marcio Eduardo Delamaro, Jose' Carlos Maldonado, and Aditya Mathur, "Integration Testing Using Interface Mutations," SERC-TR-169-P, Apr. 1996

[9] R.A.DeMillo, R.J.Lipton, and F.G.Sayward, "Hints on Test Data Selection : Help for the Practicing Programmer," IEEE Computer, Vol.11, No.4, pp.34-41, Apr 1978

윤 희 진

1993년 2월 이화여자대학교 전자계산학과 학사. 1998년 2월 이화여자대학교 대학원 컴퓨터학과 석사. 1998년 3월 ~ 현재 이화여자대학교 대학원 컴퓨터학과 박사과정. 관심분야는 소프트웨어공학, 분산 컴포넌트 테스팅, 테스트 프로세스, 객체지향 소프트웨어 테스팅.

최 병 주

1979년 ~ 1983년 이화여대 수학과 학사. 1986년 ~ 1988년 Purdue Univ. 전산석사. 1987년 ~ 1990년 Purdue Univ. 전산학(소프트웨어공학 전공) 박사. 1991년 ~ 1992년 삼성종합기술원 선임연구원. 1992년 ~ 1995년 용인대학교 전자계산학과 전임강사. 1995년 ~ 현재 이화여자대학교 컴퓨터학과 조교수. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 소프트웨어 품질 측정, 테스트 케이스 적정성 측정 알고리즘임.