ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

컨테이너 내부 테스팅 전략 기반의 EJB 컴포넌트 테스트 자동화 방법

An Automatic Testing Method for EJB Components based on In-Container Testing Strategy

국 승 학[1] 김 현 수[2*]
Seung-hak Kuk Hyeon Soo Kim

요 약

컴포넌트 기술은 소프트웨어를 신속하고, 효과적으로 개발할 수 있는 대안으로 90년대 초반부터 각광 받기 시작하였으며, 현재 컴포넌트 중심의 애플리케이션은 대부분 J2EE/EJB 컴포넌트 환경을 이용하여 개발되고 있다. 그러나 EJB 컴포넌트를 테스트하는 것은 기존의 자바 클래스의 테스트보다 더 많은 노력이 필요하며, 어려운 작업이기 때문에 대부분 충분한 테스트가 수행되지 않는 문제점이 있다. 이에 본 논문에서는 컨테이너 내부 테스팅 전략 기반의 EJB 컴포넌트 테스트 자동화 방법을 제안하고, 도구를 구현한다. 본 논문에서 제안하는 방법은 EJB 컴포넌트의 테스트 환경을 자동으로 구축함으로써 테스트 과정에서 개발자 또는 사용자의 시간과 노력을 많이 줄여줄 수 있다. 이를 통해 EJB 컴포넌트 개발 과정에서 보다 더 많은 그리고 다양한 테스트의 수행을 가능하게 함으로써 컴포넌트의 신뢰도를 높일 수 있다.

☞ 주제어 : EJB 컴포넌트, 컴포넌트 테스트, 테스트 자동화, 테스트 환경

ABSTRACT

Component technologies which enable quickly and effectively to develop software have begun to come into the spotlight since early 1990s. Currently, a number of software development works are performed on the J2EE/EJB environment. However component testing is a very complicated task, in addition it requires more efforts than the previous Java class testing. Thus many developers do not perform sufficiently testing works. In this paper we propose an automatic testing method for EJB components based on the in-container testing strategy and implement a testing tool. Since our method builds automatically the test environment for EJB components, it is possible for developers or testers to save their time and efforts at the test preparation phase. Therefore we are convinced that the reliability of EJB components can be increased through sufficient testings with our method.

☞ keyword : EJB Component, Component Test, Test Automation, Test Environment

1. 서 론

컴포넌트 기술은 소프트웨어를 신속하고, 효과적으로 개발할 수 있는 대안으로 90년대 초반부터 각광 받기 시작하였다. 컴포넌트 기술을 사용하여 애플리케이션을 개발할 경우에는 컴포넌트 기술이 약속하고 있는 재사용성을 기반으로 애플리케이션 개발 생산성을 높일 수 있어서, 기하급수적으로 증가하고 있는 소프트웨어의 수요를 충당할 수 있게 된다. 따라서 컴포넌트 기술을 활용하는 컴포넌트 기반 개발 방법론(Component Based Development)이라는 새로운 패러다임이 미래 소프트웨어 산업의 새로운 대안으로 인식되며 많은 사람들의 관심을 모아 왔다[1, 2].

J2EE(Java 2 Platform, Enterprise Edition)는 다층 엔터프라이즈 애플리케이션의 개발을 위한 엔터프라이즈 표준 환경이다[3,4]. 최근 산업계는 급변하는 사용자의 요구사항을 반영하면서, 경제적이고, 경쟁력 있는 소프트웨어를 개발하기 위해 J2EE/EJB(Enterprise java beans) 컴포넌트 개발 방법을 채택하고 있다. 이는 J2EE/EJB의 스펙을 따르는 컴포넌트는 어느 곳에서나 동작한다는 것과, 컨테이너에서 보안과 트랜잭션 처리와 같은 많은 서비스를

[1] 2-TICN, Agency for Defense Development, Daejeon, 305-600, Korea
[2] Dept. of Computer Science & Engineering, Chungnam National University, Daejeon, 305-764, Korea
* Corresponding author (hskim401@cnu.ac.kr)
[Received 5 February 2015, Reviewed 9 February 2014, Accepted 13 March 2015]

제공하기 때문에 개발자는 비즈니스 로직 개발에만 집중할 수 있기 때문이다. 이와 같이 J2EE/EJB 기반의 소프트웨어 개발은 다양한 이점을 제공해주기 때문에 외국뿐만 아니라 국내에서도 빠르게 성장하고 있다[2].

이러한 J2EE/EJB 환경에서는 도메인 테스팅, 흐름 기반 테스팅 기법 등과 같은 기존의 테스트 기법뿐만 아니라 객체지향 테스트 기법도 곧바로 적용하기 어렵다. 이는 실행 시 EJB의 생명주기가 EJB 컨테이너에 의해 관리되고, 컨테이너가 EJB 기능에 대한 접근을 통제하기 때문이다. 따라서 EJB 컴포넌트를 테스트하기 위해서는 J2EE/EJB 환경에 맞는 테스팅 전략이 필요하다. 현재 EJB 컴포넌트의 특성을 고려한 고립 테스팅(Testing in Isolation), 컨테이너 외부 테스팅(Outside the Container Testing), 컨테이너 내부 테스팅(In-Container Testing)의 전략이 널리 알려져 있으며, MockEJB[5], JUnit[6], Cactus[7], JUnitEE[8] 등과 같은 프레임워크 형태의 테스트 도구가 사용되고 있다. 그러나 많은 개발자들이 EJB 컴포넌트의 비즈니스 로직을 개발하는데 관심이 있을 뿐 테스트 환경에 관해 고려해야할 사항과 채택해야 하는 전략에 관해서 알지 못하며, 대부분 수동 테스트를 수행한다. 또한 위와 같은 도구들은 각 전략을 손쉽게 적용할 수 있도록 지원하고 있을 뿐, 테스트 코드의 생성부터 실행까지 개발자 혹은 테스터가 처리해주어야 할 사항이 많기 때문에 적용하기 쉽지 않다. 따라서 이 도구를 활용해서는 사용자가 충분한 테스트를 수행하기 어려울 뿐만 아니라 테스트 과정에서 많은 부담을 준다[1, 9, 10].

이러한 어려움을 해결하고자 본 논문에서는 컨테이너 내부 테스팅 전략 기반의 EJB 컴포넌트의 기능 테스트를 자동화 하는 방법을 제안하고 도구로 구현한다. 이를 위해 소스코드와 배치 지시자 파일(Deployment Descriptor File)을 분석하여 테스트 자동화에 필요한 정보를 추출하고, 추출된 정보로부터 테스트 드라이버 및 테스트 데이터를 자동으로 생성하는 방법과 배치의 자동화를 통해 테스트 환경을 자동으로 생성하는 방법 및 테스트 수행 자동화 방법을 제안한다.

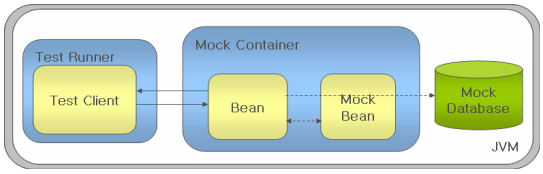
본 논문의 구성은 다음과 같다. 2장에서는 기존의 EJB 컴포넌트 테스트 전략과 방법들에 대해서 살펴본다. 3장에서는 본 논문이 제시하는 EJB 컴포넌트 테스트 자동화 방법을 기술한다. 4장에서는 EJB 컴포넌트 테스트 자동화 시스템의 구현 사항과 사례적용에 대해 기술한다. 마지막으로 5장에서는 결론과 앞으로 수행할 연구 내용에 대해서 언급한다.

2. 관련연구

2.1 EJB 컴포넌트 테스트 전략

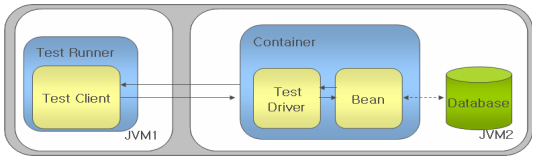
EJB 컴포넌트는 실행 시 생명주기가 EJB 컨테이너에 의해 관리되고, 컨테이너가 EJB 기능에 대한 접근을 통제하기 때문에 기존의 테스트 방법을 그대로 적용할 수 없다. 따라서 EJB컴포넌트의 동작 환경을 고려한 테스트 방법이 적용되어야 한다. 이러한 점을 고려한 EJB 컴포넌트 테스트 전략은 크게 세 가지로 나눌 수 있다[1,11].

고립 테스팅(Testing in Isolation): 이 전략은 EJB 컴포넌트 테스트를 위해 컨테이너나 의존적인 다른 EJB 컴포넌트의 스텁(Stub)을 생성해 완벽하게 독립적인 환경에서 테스트를 수행하는 전략이다. 이를 위해 그림 1과 같이 테스트 대상이 되는 빈을 제외한 모든 요소를 모의 객체(Mock Object)로 대체해 테스트를 수행한다[12].



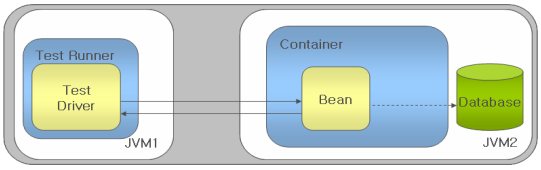
(그림 1) 고립 테스팅 전략
(Figure 1) Testing in Isolation Strategy

컨테이너 내부 테스팅(In-Container Testing): 이 전략은 컨테이너 내부 혹은 애플리케이션 서버 내에 테스트를 위해 작성된 테스트 드라이버를 배치하여 테스트를 수행하는 방법이다. 테스트를 수행하는 테스트 드라이버가 테스트 대상이 되는 빈과 같은 컨테이너 내부에 배치되어 동작한다. 이 방법은 로컬 인터페이스를 갖는 EJB를 테스트하는데 알맞은 방법이다. 그러나 이 전략은 컨테이너 내에서 테스트를 수행할 수 있도록 도와주는 추가적인 테스트 프레임워크가 필요하며, 구현, 배치, 그리고 테스트를 위한 호출이 좀 더 복잡하다.



(그림 2) 컨테이너 내부 테스팅 전략
(Figure 2) In-Container Testing Strategy

컨테이너 외부 테스팅(Outside the Container Testing): EJB가 원격 인터페이스를 갖는 경우에 원격 클라이언트를 작성함으로써 테스트 할 수 있는 가장 간단한 방법이다. 이 전략은 테스트의 작성과 실행이 쉽고, JUnit과 같은 테스트 인프라를 쉽게 이용할 수 있다. 분산 환경에서 원격 인터페이스를 통해 비즈니스 로직을 접근할 경우에 적합한 테스트 방법이다.



(그림 3) 컨테이너 외부 테스팅 전략
(Figure 3) Outside the Container Testing Strategy

세 가지 전략은 각각 장단점이 있다. 그러나 가장 효과적인 방법은 컨테이너 내부 테스팅 전략이다. 첫 번째 전략의 경우 컨테이너나 EJB 컴포넌트, DB와 같은 자원에 대해 많은 모의 객체를 생성해야 하며, 독립적인 환경에서 테스트를 성공적으로 수행했다고 하더라도 실제 컨테이너에서 올바로 동작한다고 보장할 수 없는 단점이 있다. 세 번째 전략의 경우 외부에 노출된 리모트 인터페이스에 대해서 간단하게 테스트를 수행하는 것은 가능하지만 EJB 컴포넌트가 로컬 인터페이스를 갖는 경우에 이를 전혀 테스트 할 수 없다는 단점이 있다. 그러나 컨테이너 내부 테스팅 전략의 경우 실제 컨테이너 내부에서 동작하기 때문에 테스트 결과를 신뢰할 수 있고, 로컬 인터페이스에 대해서도 테스트 할 수 있다. 이 전략의 경우 컨테이너 내에서 테스트를 수행할 수 있도록 도와주는 추가적인 테스트 프레임워크가 필요하며, 구현, 배치, 테스트를 위한 호출이 다소 복잡하다는 단점이 있다. 본 논문에서는 컨테이너 내부 테스팅 전략의 단점을 극복하기 위해 테스트 환경을 자동으로 구축하여 사용자의 개입을 최소화하는 방법을 제안하고 도구로 구현한다.

2.2 기존 연구 및 도구

2.2.1 기존의 EJB 컴포넌트 테스트 관련 연구

논문 [13]의 경우 EJB 컴포넌트의 메소드 시그너처 정보를 자바의 리플렉션(Reflection) 기능을 이용해 추출하고 이를 기반으로 성능테스트를 자동으로 수행하는 시스템을 설계하였다. 그러나 이 연구는 외부의 클라이언트를 이용해 테스트를 수행하기 때문에 정확한 결과를 측정하기 어렵다. 논문 [14]의 경우 일반적인 컴포넌트에서 제공되는 명세를 기반으로 테스트 랩퍼(wrapper)를 생성하고 이를 이용한 테스트 방법을 제안하고 있다. 그러나 이 방법은 컴포넌트의 명세가 표준화 되어있지 않은 상태에서 모든 컴포넌트에 적용하기 어렵다는 단점이 있다. EJB 컴포넌트의 경우 배치 지시자에 각 인터페이스에 대한 사항이 기술된다. 배치 지시자의 정보가 논문 [14]에서 요구하는 정보에 비해 제약적인 정보만을 제공하기 때문에 곧 바로 적용하기 어렵다. 논문 [15]의 경우 EJB 컴포넌트의 조립 과정에서 에러 가능성이 있는 부분에 대해 에러 코드를 삽입하고 뮤테이션(mutation) 테스트 기법을 적용한 연구다. 이 연구는 EJB 컴포넌트 그 자체의 기능과 관련된 테스트라기보다는 컴포넌트의 조립 과정에서 다양한 설정을 변형하고 그것으로 인해 미치는 영향을 분석하는 연구로서, 자동화된 방법을 제시하지 않는다. 논문 [16]의 경우 XML로 표현된 UML 모델로부터 테스트 데이터를 자동으로 생성하고 이를 이용해 EJB 컴포넌트의 기능을 테스트하는 CTM(Component Test Manager)라는 도구를 구현한 사례를 보여준다. 그러나 컨테이너 외부 테스팅 전략을 이용하기 때문에 로컬 인터페이스를 갖는 EJB 컴포넌트에 대해서는 테스트를 수행할 수 없다. 논문 [17]의 경우 EJB 컴포넌트의 메타컨텐트(Metacontent)를 이용하여 회귀 테스트(regression test)를 수행하는 기법을 설명한다. 이 논문에서는 EJB 컴포넌트의 소스코드를 분석하여 제어 흐름을 파악하고, 메타컨텐트를 추출하는 방법과 명세에 테스트와 관련된 메타컨텐트를 기술하는 방법을 소개한다. 그러나 소스코드로부터 메타컨텐트를 추출하거나 명세에 다양한 정보를 기술하는 것은 사용자에게 많은 부담을 준다. 논문 [18]은 [17]의 연구를 확장하여 메타컨텐트를 이용해 기능적인 측면뿐만 아니라 다양한 품질속성을 기술하여 EJB 컴포넌트를 검증하는 자동화된 방법을 제시하는 논문이다. 그러나 EJB 컴포넌트를 테스트하기 위해 직접 다양한 명세를 기술해야 하므로 사용자에게 많은 부담을 준다. 논문 [19]는 유스케이스를 기반으로 테스트 케이스를 기술하고 이를 이용한 테스트 방법을 제시한다. 여기서 유스케이스는 테스트 수행 시 일련의 시나리오를 기술한 것으로 보고 이를 이용하여 시나리오 기반의 테스트가 가능하다. 그러나 이 논문 역시 기존의 컴포넌트의 명세를 테스트에 활용하는 방법과 유사하다. 따라서 사용자가 직접 테스트 케이스 메타모델을 기술해줘야 하며, 자동

화된 방법을 제시하지 못하고 있다. 마지막으로 논문[20]의 경우 일반적인 컴포넌트 기반 소프트웨어의 블랙박스 테스트 자동화 프레임워크에 관한 연구를 소개한다. 이 연구는 컴포넌트의 인터페이스를 분석해 각각의 메소드를 추출한 후 테스트 드라이버와 데이터 그리고 테스트 오라클을 위한 컴포넌트 랩퍼를 자동으로 생성하여 테스트 프레임워크를 구축하는 방법을 소개한다. 그러나 이 방법은 실제 컴포넌트가 동작하는 환경을 고려하지 않았다.

EJB 컴포넌트 테스트와 관련된 기존의 연구들은 대부분 컨테이너 외부 테스팅 전략을 적용하고 있으며, 컴포넌트의 메타 데이터를 이용한 방법이 주류를 이룬다. 컨테이너 외부 테스팅 전략은 외부로 드러난 리모트 인터페이스에 대해서 테스트를 수행하기 때문에 정확한 에러의 원인을 찾아낼 수 없다. 또한 메타 데이터를 이용한 테스트 방법은 개발자 혹은 테스터가 컴포넌트에 대한 메타 데이터를 추가적으로 기술해야하기 때문에 많은 노력이 요구된다. 그러나 우리 연구에서는 컨테이너 내부 테스팅 전략을 이용하기 때문에 모든 인터페이스에 대한 테스트 수행이 가능하며, 테스트 준비 과정과 수행 과정을 자동화하기 때문에 짧은 시간에 많은 테스트의 수행이 가능하다는 장점이 있다.

2.2.2 EJB 컴포넌트 테스트 도구

이 절에서는 테스트 전략에서 언급한 MockEJB, Cactus, JUnit, JUnitEE를 대상으로 장단점을 분석한다. 각 도구는 비상업용 도구로써 프레임워크 형태로 제공되고, 현재 EJB 컴포넌트 테스트에 사용되고 있다. 표 1은 기존 도구의 장점과 단점을 요약한 것이다.

(표 1) 기존 도구의 장·단점

(Table 1) Strong and Weak Points of the Existing Tools

도구	사용전략	장점	단점
Mock EJB	Testing in Isolation	가상의 컨테이너 제공 가상의 EJB 제공테스트 환경 구축이 쉬움	완벽한 구현이 아님 사용자의 테스트 환경 구축 필요 테스트 수행 결과가 완벽하다고 보장할 수 없음
JUnit	Outside the Container Testing	JUnit을 이용해 테스트를 수행하기 때문에 비교적 간단함	로컬 인터페이스에 대한 테스트 수행 불가 사용자의 테스트 환경 구축 필요
Cactus	In-Container Testing	모든 인터페이스 메소드에 대한 테스트 수행가능 테스트 클라이언트/브라우저를 통해 테스트 수행이 가능함	Cactus프레임워크 설치가 복잡함 테스트 드라이버 구현이 복잡함 사용자의 테스트 환경 구축 필요 배치와 관련된 작업을 사용자가 직접 수행
JUnite e	In-Container Testing	모든 인터페이스에 대한 테스트 수행가능 브라우저를 통해 테스트 수행이 가능	테스트 자동화가 어려움 사용자의 테스트 환경 구축 필요 배치와 관련된 작업을 사용자가 직접 수행

3. EJB 컴포넌트의 테스트 자동화 방법

본 논문에서는 EJB 컴포넌트 테스트 자동화를 위해 테스트 환경을 자동으로 구축하고, 테스트 수행을 자동화하는 방법을 제시한다.

3.1. 테스트 환경 자동 구축

본 논문에서 제안하는 테스트 환경 자동 구축 단계는 크게 EJB 컴포넌트 분석 단계, 테스트 드라이버 및 테스트 데이터 생성 단계, 배치 단계로 나뉜다. 이번 절에서는 각 단계의 방법을 설명한다.

3.1.1. EJB 컴포넌트 분석 단계

EJB 컴포넌트 분석 단계는 테스트 대상 EJB 컴포넌트를 분석하여 테스트 자동화에 필요한 정보를 추출하는 단계이다. EJB 컴포넌트 테스트 환경 자동 구축을 위해서 다음과 같은 정보가 필요하다.

- 애플리케이션 구성정보: 애플리케이션을 구성하는 EJB 컴포넌트에 대한 정보
- EJB 컴포넌트 구성 정보: EJB 컴포넌트를 구성하는 홈 인터페이스, 컴포넌트 인터페이스(리모트 및 로컬 인터페이스), 그리고 EJB 구현 클래스에 대한 정보
- EJB 컴포넌트 종류: 해당 EJB 컴포넌트가 엔티티 빈, 세션 빈, 혹은 메시지 드리븐 빈인지에 대한 정보
- 다른 컴포넌트와의 연동 정보: 다른 EJB 컴포넌트와 연동하고 있을 경우 연동되는 컴포넌트에 대한 정보

- JNDI 설정 정보: 원격 객체를 찾기 위한 방법인 JNDI 정보. 이를 알고 있어야 대상 EJB 컴포넌트의 테스트가 가능
- 인터페이스 메소드 정보: EJB 컴포넌트의 단위 테스트를 위해 각각의 메소드에 대한 정보

본 논문에서는 이러한 정보를 추출하기 위해 EJB 컴포넌트의 배치 지시자와 소스코드를 분석한다.

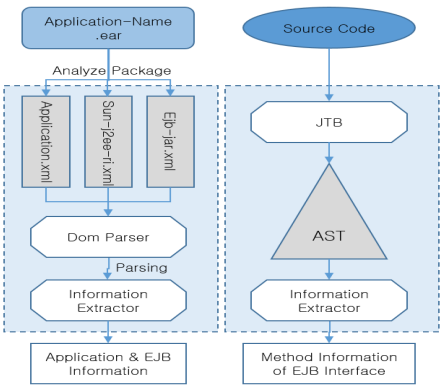
(1) 배치 지시자로부터 정보 추출

배치 지시자는 EJB 컴포넌트 기반 애플리케이션에 대한 선언적 정보를 포함하는 XML파일이다. 이러한 배치 지시자에는 애플리케이션의 구성정보, 개별 EJB 컴포넌트들의 인터페이스 정보, 컨테이너로부터 제공받는 서비스, 다른 EJB와 컨테이너들 사이의 자원관리에 대한 의존관계 등이 명시된다. EJB 기반 애플리케이션에는 전체 애플리케이션을 명세하는 배치 지시자(application.xml), 애플리케이션의 실행 시 컨테이너 서비스와 관련된 설정 정보를 포함하는 배치 지시자(sun-j2ee-ri.xml), 그리고 각 구성 EJB 컴포넌트의 정보를 포함하는 배치 지시자(ejb-jar.xml)를 포함하고 있다. application.xml의 분석을 통해 전체 애플리케이션의 구성 정보를 추출한다. sun-j2ee-ri.xml을 분석하여 실제 구현되어있는 컴포넌트 이름과 JNDI 설정 정보를 추출한다. 그리고 EJB 컴포넌트의 이름을 이용해 각 EJB 컴포넌트의 배치 지시자인 ejb-jar.xml을 분석하여 EJB 컴포넌트의 종류, 구성 정보, 인터페이스 정보 등을 추출한다.

배치 지시자로부터 필요한 정보를 추출하는 과정은 전체 애플리케이션의 패키지인 EAR(Enterprise Archive) 파일이 입력으로 주어지면 이 패키지를 분석해 배치 지시자 파일을 추출하고, 각 배치 지시자를 DOM 파서를 이용해 파싱한 후 필요한 정보를 추출한다. 이 과정은 그림 4의 왼쪽 부분에 해당한다.

(2) 소스코드로부터 정보 추출

EJB 컴포넌트의 소스코드 분석을 통해 인터페이스 메소드에 관한 정보를 추출한다. EJB 컴포넌트의 인터페이스 정보는 배치 지시자에 기술되지 않는 경우도 있다. 대표적으로 EJB 컴포넌트의 로컬 인터페이스 정보는 일반적으로 배치 지시자에 기술되지 않는다. 우리는 EJB 컴포넌트의 모든 기능을 테스트하는 것이 목적이기 때문에 소스코드 분석을 통해 EJB 컴포넌트의 인터페이스 메소드 정보를 추출한다. 이러한 인터페이스 정보는 EJB 컴포넌트의 인터페이스 클래스에 명시된다.



(그림 4) 정보 추출 과정

(Figure 4) Information Extraction Process

EJB 컴포넌트의 메소드 정보를 알기 위해서는 소스코드가 입력으로 주어지면 EJB 컴포넌트의 인터페이스 클래스를 파싱하여 AST(Abstract Syntax Tree)를 생성하고, 그로부터 각각의 메소드 정보를 추출한다. 소스코드로부터 정보를 추출하는 과정은 그림 4의 오른쪽 부분과 같다. 본 연구에서는 JTB(Java Tree Builder)[21]를 이용하여 AST를 생성한다. JTB는 JavaCC 파서 생성기를 이용하여 AST를 생성하는 도구이다.

이와 같이 소스코드와 배치 지시자를 분석해 테스트 자동화를 위해 필요한 모든 데이터를 추출한다. 표 2는 필요한 정보와 정보원을 요약해서 정리한 것이다.

(표 2) 테스트 프레임워크 구축 시 필요한 정보와 정보원

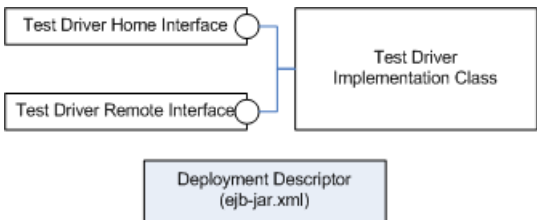
(Table 2) Information and Its Resources for Constructing Test Framework

정보	내용	정보원
애플리케이션 구성정보	애플리케이션을 구성하는 컴포넌트들	배치지시자 (Application.xml)
EJB 컴포넌트 구성정보	홈 인터페이스 컴포넌트 인터페이스(리모트, 로컬 인터페이스) 빈 구현 클래스	배치지시자 (Sun-j2ee-ri.xml, Ejb-jar.xml)
EJB 컴포넌트 종류	엔티티 빈 /세션 빈 메시지 드리븐 빈	배치지시자 (Sun-j2ee-ri.xml, Ejb-jar.xml)
EJB 컴포넌트 연동정보	다른 컴포넌트와의 연동 정보	배치지시자 (Sun-j2ee-ri.xml, Ejb-jar.xml)
EJB 컴포넌트 JNDI 설정 정보	JNDI 설정 정보	배치지시자 (Sun-j2ee-ri.xml, Ejb-jar.xml)
EJB 컴포넌트 메소드 정보	메소드 이름 메소드 입력 파라메터 메소드 반환 타입	소스코드

3.1.2. 테스트 드라이버 및 테스트 데이터 생성 단계

(1) 테스트 드라이버 생성

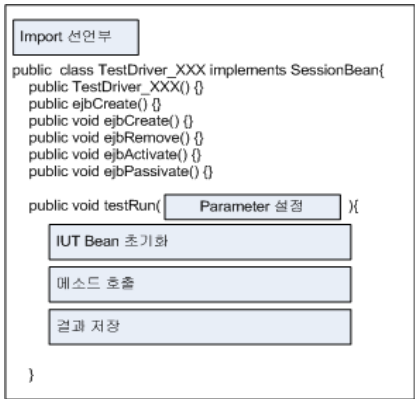
컨테이너 내부 테스팅 전략을 기반으로 테스트 환경을 구축하기 위해서는 테스트 드라이버가 컨테이너 내부에 배치되어야 한다. 그러므로 테스트 드라이버 역시 EJB 컴포넌트로 생성되어야 한다. 이를 위해서는 그림 5와 같이 테스트 드라이버 구현 클래스, 홈 인터페이스, 리모트 인터페이스가 생성되어야하며, 배치 지시자 역시 생성되어야한다.



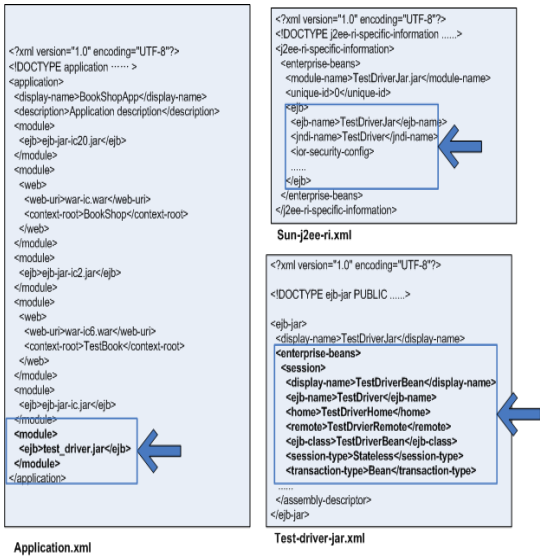
(그림 5) 테스트 드라이버의 구성 요소
(Figure 5) Elements of Test Driver

테스트 드라이버 홈 인터페이스는 테스트 드라이버의 생성/소멸에 관련된 메소드를 포함하고 있으며, 테스트 드라이버 리모트 인터페이스에는 실제 테스트를 수행할 때 테스트 수행 엔진이 호출하는 메소드가 포함된다. 테스트 드라이버 구현 클래스는 테스트 대상 인터페이스 메소드를 호출하고, 그 결과를 기록하는 역할을 수행한다. 그림 6은 테스트 드라이버 구현 클래스의 구조를 도식화한 것이다.

Import 선언부는 EJB 컴포넌트의 동작과 관련된 기본적인 패키지를 포함하도록 하고, 테스트 대상 EJB 컴포넌트 분석 단계에서 추출한 테스트 대상 컴포넌트의 홈 인터페이스 및 컴포넌트 인터페이스와 관련 EJB 컴포넌트의 인터페이스를 포함한다. 파라메터 설정 부분은 테스트 수행 엔진에서 테스트 대상 EJB 컴포넌트의 메소드에 전달하고자하는 데이터 값을 넘겨받을 수 있도록 구현되며, 또한 각각의 테스트 수행을 기록하기 위한 테스트 ID도 전달 받는다. IUT 빈 초기화 부분은 실제로 테스트 대상 메소드를 가진 빈의 객체를 생성하는 부분이다. 여기서는 EJB를 분석해 추출한 JNDI 설정 정보를 이용한다. 메소드 호출 부분은 실제로 테스트 대상 메소드를 호출하는 부분으로 리모트 인터페이스의 메소드와 로컬 인터페이스의 메소드를 호출한다. 이 과정에서 파라메터 설정 부분에서 넘겨받은 테스트 데이터를 이용해 대상 메소드를 호출한다. 호출된 메소드의 수행 결과는 결과 저장 부분을 통해 저장되며 추후 테스트 결과 분석을 위해 사용된다.



(그림 6) 테스트 드라이버 구현 클래스
(Figure 6) Implementation Class of Test Driver



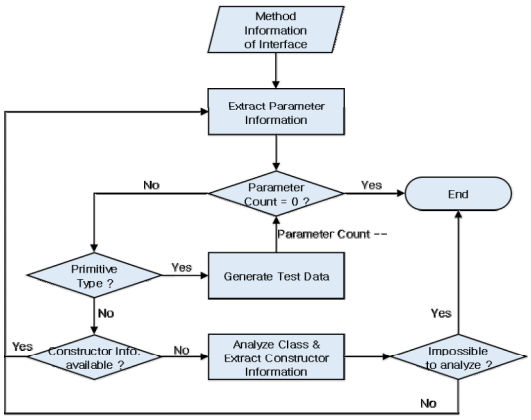
(그림 7) 애플리케이션 배치 지시자의 확장
(Figure 7) Extension of Application Deployment Descriptor

한편, 생성된 테스트 드라이버는 대상 EJB 컴포넌트와 같은 애플리케이션의 구성 컴포넌트로 포함되어야하기 때문에 애플리케이션 구성 정보 배치 지시자(예를 들어, application.xml, sun-j2ee-ri.xml)에 이런 사항을 반영해

야 한다. 그림 7과 같이 원래의 배치 지시자에 테스트 드라이버에 관한 정보를 포함시켜야 하며, 추가로 테스트 드라이버에 대한 배치 지시자도 기술되어야 한다.

(2) 테스트 데이터 생성

EJB 컴포넌트의 기능 테스트를 위해 테스트 데이터는 컴포넌트의 메소드 정보를 기반으로 자동으로 생성한다. 메소드의 파라메터 타입이 단순(Primitive) 타입이거나 문자열 타입일 경우에는 동치 분할(Equivalence Partitioning)과 경계 값 분석(Boundary Value Analysis)을 적용하여 정상적인 상황과 비정상적인 상황을 테스트 할 수 있도록 생성한다. 예를 들어, 파라메터의 타입이 int이고, 파라메터의 입력 범위가 양수라면 동치 분할을 이용하여 양의 정수(정상 범위), 0과 음의 정수(비정상 범위)에서 각각 대푯값을 선택하여 테스트 데이터로 사용한다. 파라메터 타입이 객체인 경우 객체를 생성하여 전달한다. 이 경우 EJB 컴포넌트를 분석하는 과정에서 추출한 생성자 정보와 생성자에서 사용되는 관련 클래스의 생성자 정보를 추출하여 재귀적으로 객체를 생성한다. 그림 8은 테스트 데이터를 생성하기 위한 일련의 절차이다.



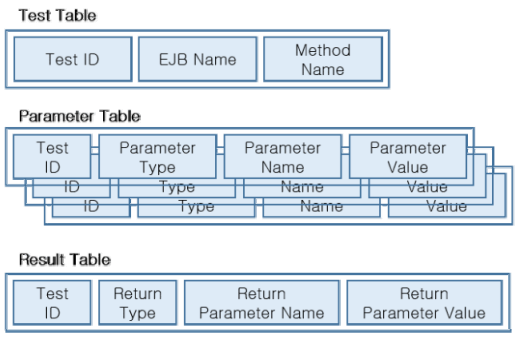
(그림 8) 테스트 데이터 생성 절차
(Figure 8) Test Data Generation Procedure

3.1.3. 배치 단계

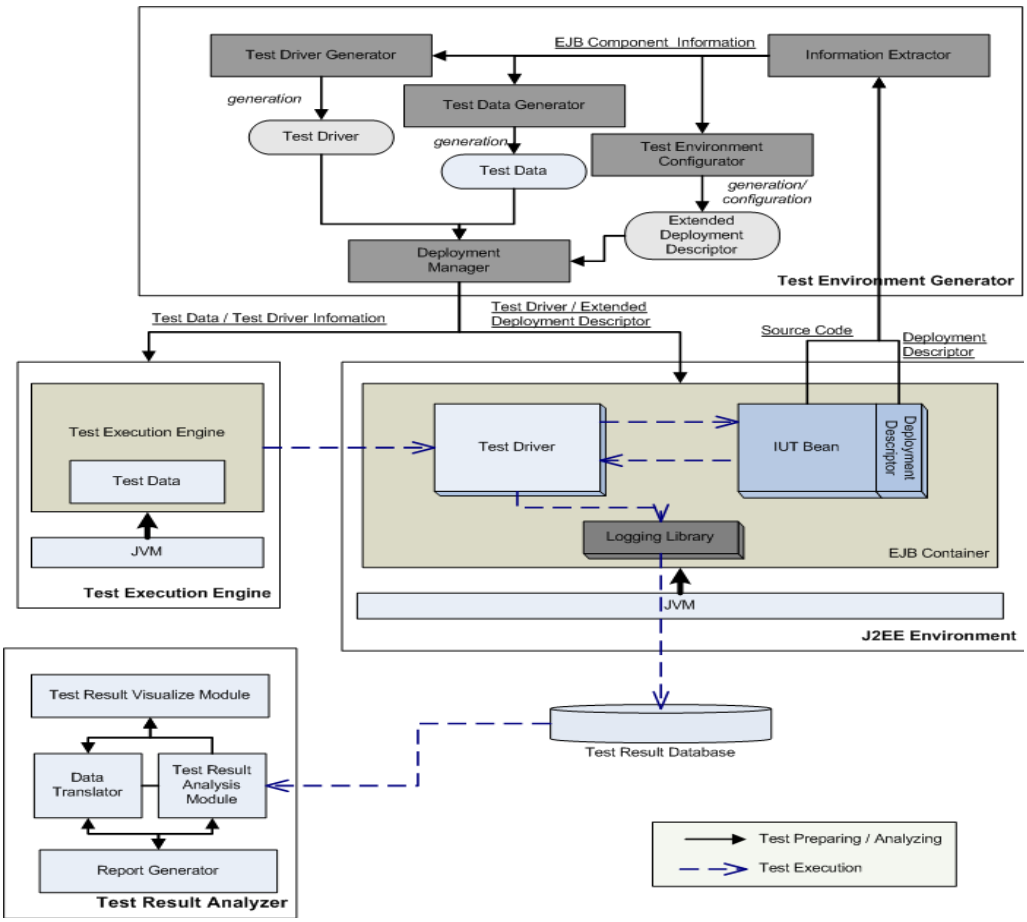
배치란 소프트웨어를 운영 환경에 설치하는 과정을 말한다. J2EE 애플리케이션 서버에는 복수 개의 애플리케이션이 배치되어 운영된다. 이 때문에 애플리케이션 서버에 각각의 애플리케이션에 대한 정보를 알려주어야 한다. J2EE에서 배치라는 말은 애플리케이션이나 그 구성 요소를 애플리케이션 서버에서 운영될 수 있도록 배치하고 초기화하는 과정을 말한다. 컨테이너 내부 테스팅 전략을 이용하는 EJB 컴포넌트 테스트 방법에서 배치 과정은 필수적이다. 배치의 자동화는 테스트 대상이 되는 애플리케이션에 테스트 코드를 추가한 다음 그것을 배치하는 과정과 테스트 수행 이후 테스트 코드를 제거한 다음 원래의 애플리케이션 코드를 재배치하는 과정으로 구성된다. 컨테이너 내부 테스팅 전략을 사용하는 기존 도구들의 경우 이러한 일련의 배치 과정을 사용자가 직접 수행해야 하는 번거로움이 있었다. 배치 자동화를 위해 우리가 채택한 방법은 배치 스크립트(Deployment Script)를 사용하는 것이다. 배치 스크립트에 대한 상세한 내용은 4.4절에서 기술한다.

3.2 테스트 수행 및 결과 분석

본 논문의 테스트 드라이버는 테스트 대상 EJB 컴포넌트와 함께 동일 컨테이너 내부에 배치되어 동작하기 때문에 리모트뿐만 아니라 로컬에 해당하는 모든 인터페이스 메소드를 테스트 할 수 있다. 그러나 테스트 드라이버가 EJB 컴포넌트 형태이기 때문에 외부에 존재하는 테스트 수행 엔진의 지시를 받는다. 테스트 수행 엔진은 테스트 드라이버 정보와 생성된 테스트 데이터를 이용해 테스트를 수행할 수 있도록 구현된다. 테스트 수행 결과는 호출한 인터페이스 메소드의 반환 값으로서 데이터베이스에 저장된다. 반환 값이 단순 타입일 경우 그 결과 값을 직접 저장하고, 객체 타입일 경우 직렬화(serialization)하여 저장한다. 또한 수행 도중 에러가 발생하는 경우 예외 상황 메시지(exception message)를 기록하여 추후에 에러의 원인을 파악한다. 그림 9는 테스트 결과를 저장하는 데이터베이스의 테이블 구조이다.



(그림 9) 테스트 결과 저장을 위한 테이블 구조
(Figure 9) Table Structure for Storing the Test Result



(그림 10) 테스트 자동화 도구 구조
(Figure 10) Architecture of Test Automation Tool

테스트 결과는 테스트 대상 EJB, 메소드, 파라메터 정보와 결과 값을 기반으로 사용자에게 의미 있는 형태로 전달되어야한다. 따라서 결과 분석 기능 및 데이터 변환기가 포함되어야하고 보고서 작성 도구가 필요하다.

4. EJB 컴포넌트 테스트 자동화 도구 구현 및 사례 적용

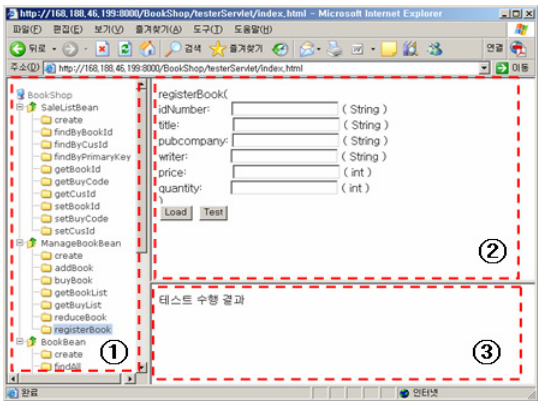
EJB 컴포넌트 테스트 자동화를 위해 그림 10과 같이 테스트 대상이 되는 EJB 컴포넌트의 소스코드나 배치 지시자로부터 정보를 추출하는 정보 추출(Information Extractor)모듈, 정보 추출 모듈로부터 테스트 드라이버를 생성하는 테스트 드라이버 생성(Test Driver Generator)모듈, 그리고 테스트 데이터를 자동으로 생성하는 모듈(Test Data Generator), 또한 생성된 테스트 드라이버의 배치과정을 자동화하기 위한 배치 관리자(Deployment Manager)와 배치에 관련된 배치 지시자의 설정 작업을 수행하는 테스트 환경 설정기(Test Environment Configurator)를 설계하고 프로토타입 시스템을 구현하였다.

4.1 정보 추출 모듈

정보 추출 모듈은 크게 두 가지인데, 하나는 배치 지시자로부터 정보를 추출하는 모듈이고 다른 하나는 소스코드로부터 정보를 추출하는 모듈이다. 추출된 정보는 테스트 드라이버와 테스트 데이터를 생성하기 위해 저장된다.

4.2. 테스트 드라이버 생성 모듈 및 테스트 수행 엔진

테스트 드라이버 생성 모듈에서는 그림 5와 같은 EJB 컴포넌트 형태의 테스트 드라이버를 생성한다. 하나의 EJB에 대해 하나의 테스트 드라이버를 생성하여 해당 EJB의 모든 인터페이스 메소드를 테스트 할 수 있도록 생성한다. 테스트 결과는 테스트 결과 데이터베이스에 저장한다. 이를 위해 테스트 드라이버와 같은 위치에 배치되는 로깅 라이브러리(그림 10 참조)를 구현하여, 테스트 결과 반환 시점에 데이터를 기록한다.



(그림 11) 테스트 드라이버 UI
(Figure 11) User Interface of Test Driver

테스트 수행 엔진은 테스트 데이터 생성 모듈에서 생성한 데이터를 기반으로 반복적으로 테스트를 수행한다. 테스트 수행 중에 사용자의 개입이 용이하도록 그림 11과 같이 JSP 기반의 UI를 갖는 테스트 클라이언트가 제공된다. 그림 11의 페이지 구성은 다음과 같다.

① 사용자가 테스트 대상 컴포넌트와 메소드를 선택할 수 있도록 애플리케이션의 구조를 보여주는 페이지
② 테스트 대상 메소드의 파라메터를 설정하거나 데이터 기반 테스트 수행을 명령하는 페이지
③ 테스트 수행 후 결과 분석 모듈에서 분석한 데이터를 사용자에게 보고하는 페이지

4.3. 테스트 데이터 생성 모듈 및 결과 분석 모듈

테스트 데이터는 테스트 대상 컴포넌트의 메소드 정보를 이용하여 생성한다. 제안하는 도구는 메소드의 파라메터 정보를 이용하여 다양한 조합의 테스트 데이터를 생성한다. 예를 들어, 그림 11의 테스트 드라이버에 나타난 registerBook()이라는 메소드는 4개의 문자열 파라메터와 2개의 정수형 파라메터를 갖는다. 메소드의 파라메터 타입이 단순 타입 또는 문자열 타입이기 때문에 동치 분할을 적용하여 정상적인 상황과 비정상적인 상황을 테스트 할 수 있도록 생성하였다. 이를 위한 테스트 데이터는 표 3과 같다.

(표 3) registerBook() 메소드를 위한 테스트 데이터
(Table 3) Test Data for registerBook() Method

테스트 대상 메소드	파라메터	테스트 데이터				
		1	2	...	N-1	N
registerBook()	idNumber	"abced"	NULL	...	D0 CF	"가나다"
	title	"!@$%"	D0 CF	...	NULL	"C:\"
	pubcompany	NULL	"C:\"	...	"12345"	"abced"
	writer	"가나다"	"abced"	...	"!@$%"	"C:\"
	price	NULL	-887	...	'a'	258
	quantity	-578	-94	...	786	NULL

파라메터가 객체 타입일 경우 해당 객체를 생성하여 테스트를 수행해야한다. 예를 들어, addBookList(BookList list) 메소드는 BookList라는 클래스 타입을 파라메터로 갖는다. 따라서 테스트 데이터를 생성하기 위해서는 BookList 객체를 생성해야 한다. 이를 위해 관련된 BookList 클래스를 분석하여 생성자 정보를 추출하고, 생성자가 필요로 하는 데이터를 다시 생성하는 재귀적인 방법을 사용한다.

결과 분석 모듈은 테스트 수행 정보와 결과를 사용자에게 의미 있게 전달하기 위해 보고(report) 기능을 포함한다. 그림 10에서 테스트 결과 분석 모듈이 포함된 것을 볼 수 있다. 결과 분석 모듈에서는 데이터베이스에 저장된 테스트 결과를 가져와 분석하는 기능과 사용자가 쉽게 그 결과를 판단할 수 있도록 보고하는 기능이 구현된다. 이를 위해 데이터 변환 모듈을 추가하였다. 이 모듈은 객체 타입의 파라메터나 반환 값의 상태를 분석하여 사용자에게 보고할 수 있는 형태로 변환한다. 본 논문에서 구현한 결과 분석 모듈은 다음과 같은 방법으로 반환 값을 분석하여 보고한다.

(표 4) 반환 값에 따른 처리 방법
(Table 4) Processing Method for a Return Value

타입	출력 방법
void	'Return type is void'라는 메시지 출력
Primitive type	결과 값 출력
String	결과 값 출력
Object	빈의 모든 get 메소드를 이용하여 상태를 출력
Vector	Vector의 Element 종류에 따라 Primitive type, String일 경우는 결과 값을 출력하고, 객체일 경우에는 get을 이용하여 상태 출력
Collection	Collection의 Element 종류에 따라 Primitive type, String일 경우는 결과 값을 출력하고, 객체일 경우 get 메소드를 이용하여 상태 출력

4.4. 테스트 환경 설정기 및 배치 관리자

컨테이너 내부 테스팅 전략에서는 테스트 드라이버가 실제 컨테이너 내부에 배치된다. 따라서 배치되는 드라이버에 관한 정보가 컨테이너에 반영되어야 한다. 이를 위해 우리는 테스트 환경 설정기에서 기존의 배치 지시자를 수정하여 추가되는 테스트 드라이버에 대한 정보를 포함하도록 한다. 또한 테스트 드라이버도 EJB 컴포넌트 형태이기 때문에 테스트 드라이버 컴포넌트에 대한 배치 지시자도 생성하여야 한다. 테스트 환경 설정기에서는 이러한 일련의 작업을 자동으로 수행하도록 구현된다. 확장된 배치 지시자의 예는 그림 7에서 볼 수 있다.

테스트 드라이버와 배치 지시자가 자동으로 생성된 후에 그것을 컨테이너 내부에 배치하는 과정이 필요하다. 또한 테스트가 수행된 이후에 컨테이너에서 테스트 드라이버 컴포넌트를 제거하는 작업도 필요하다. 이러한 일련의 작업은 배치 스크립트를 통해 이루어진다. 배치 스크립트는 EJB 서버의 종류에 따라 다르다. 왜냐하면 각 EJB 서버를 제공하는 업체마다 제공하는 배치 도구가 다르고, 이 도구를 이용하는 방법이 다르기 때문이다. 우리는 선사의 EJB 서버를 이용하기 때문에 그에 해당하는 배치 스크립트를 작성하였다. 배치 스크립트는 실제 배치하는 과정뿐만 아니라 배치를 위해 준비하는 과정을 포함한다. 아래의 배치 스크립트에서 패키징 과정이 배치를 위해 준비하는 과정에 해당한다. 전체 배치 과정은 표 5와 같다.

(표 5) 배치 스크립트
(Table 5) Script for Deployment

① Deploytool uninstall <애플리케이션 이름> : 선택된 애플리케이션의 배치를 해제한다.

② Jar xvf <애플리케이션.ear> : 애플리케이션.ear 파일의 패키지를 푼다.

A. 애플리케이션을 구성하는 EJB 컴포넌트의 정보를 추출한다.

B. EJB 컴포넌트 정보를 기반으로 테스트 드라이버를 생성한다.

C. 테스트 드라이버를 애플리케이션에 추가한다.

D. application.xml에 추가된 테스트 드라이버의 정보를 추가한다.

③ Jar cvf <애플리케이션.ear> : 테스트 드라이버가 추가된 애플리케이션을 다시 패키징한다.

④ Deploytool deploy <애플리케이션.ear> <server name>: 서버에 테스트 드라이버가 추가된 애플리케이션을 배치한다.

A.테스트를 수행한다.

⑤ Deploytool uninstall <애플리케이션 이름> <server name>: 테스트 수행 후 서버에서 애플리케이션의 배치를 해제한다.

⑥ Jar xvf <애플리케이션.ear> :애플리케이션.ear 파일의 패키지를 푼다.

A. 애플리케이션으로부터 테스트 드라이버를 제거한다.

B. application.xml에서 테스트 드라이버에 대한 정보를 제거한다.

⑦ Jar cvf <애플리케이션.ear> : 테스트 드라이버가 제거된 애플리케이션을 다시 패키징한다.

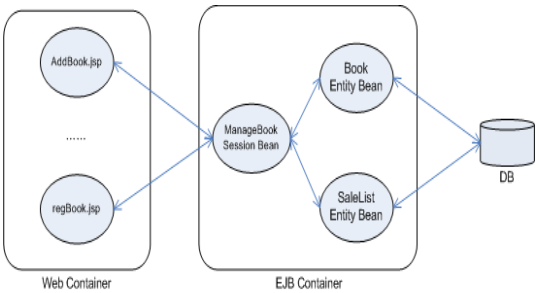
⑧ Deploytool deploy <애플리케이션.ear> <server name> : 서버에 테스트 드라이버가 제거된 애플리케이션을 다시 배치한다.

4.5. 사례적용

본 논문에서는 인터넷 도서 구매 사이트인 BookShop이라는 간단한 애플리케이션의 EJB 컴포넌트에 대해 테

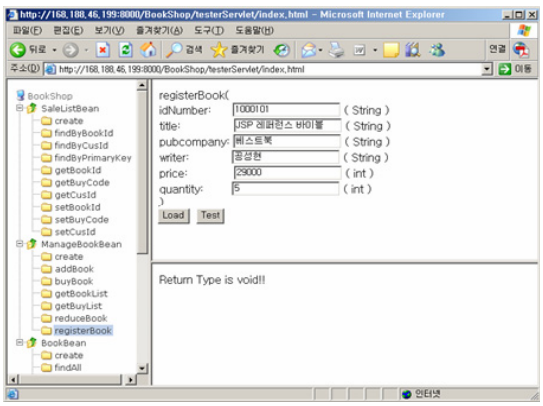
스트를 수행하였다. BookShop 애플리케이션의 구성은 다음과 같다.

- 웹 컨테이너: BookShop 애플리케이션의 사용자 UI를 담당하는 웹 페이지를 포함
- Book 엔티티 빈: 책 정보를 표현하기 위한 엔티티 빈
- SaleList 엔티티 빈: 고객들이 구매한 책 목록을 표현하기 위한 엔티티 빈
- ManageBook 세션 빈: 고객과 관리자가 책 등록, 구매, 검색 등을 수행하기 위한 세션 빈

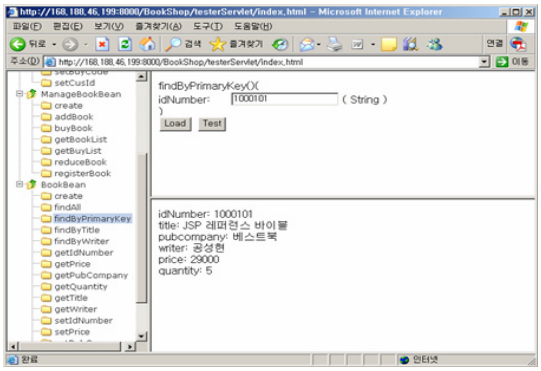


(그림 12) BookShop 애플리케이션의 구성
(Figure 12) Structure of BookShop Application

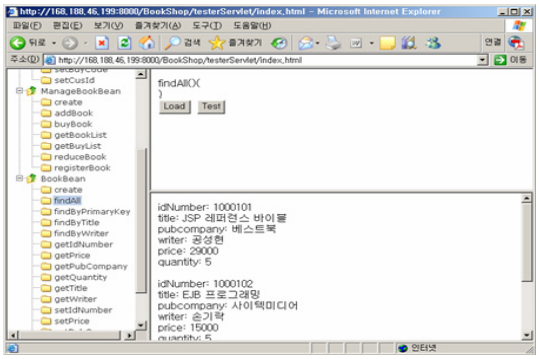
실제로 각 빈의 메소드에 대한 테스트 수행 결과는 다음과 같다.



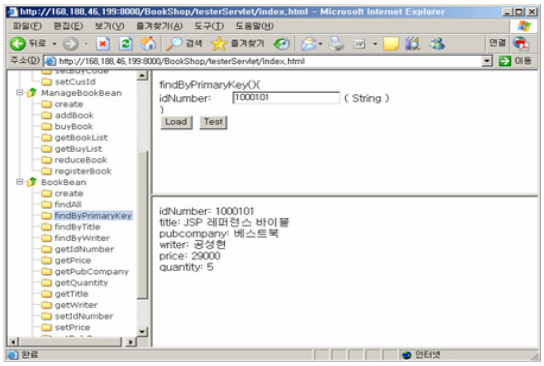
(그림 13) ManageBook의 registerBook() 테스트 수행 결과
(Figure 13) Test Result of registerBook() in ManageBook



(그림 14) Book의 findByPrimaryKey() 테스트 수행 결과
(Figure 14) Test Result of findByPrimaryKey() in Book



(그림 15) Book의 findAll()테스트 수행 결과
(Figure 15) Test Result of findAll() in Book



(그림 16) Book의 findByPrimaryKey() 테스트 수행 결과
(Figure 16) Test Result of findByPrimaryKey() in Book

4.6. 기존 도구와의 비교

본 논문에서는 테스트 환경을 자동으로 구축함으로써 테스트 준비과정을 완전 자동화하기 때문에 기존의 도구에 비해 많은 장점을 갖는다. 이 절에서는 Cactus, JUnitEE와 우리 도구를 비교하였다. Cactus와 JUnitEE는 우리 도구와 마찬가지로 컨테이너 내부 테스팅 전략을 사용하고 있으며, 현재 EJB 컴포넌트 테스트에 많이 이용되고 있다.

(표 6) 기존 도구와의 비교
(Table 6) Comparison with the Existing Tools

	Cactus	JUnitEE	본 연구
사용 전략	In-Container Testing		
테스트 대상	EJB, 서블릿	EJB, 서블릿	EJB
테스트 대상 분석	수동	수동	자동분석
테스트 코드작성	수동	수동	자동생성
테스트 코드형태	서블릿	서블릿	EJB/서블릿
테스트 코드 배치	수동	수동	자동배치
테스트 케이스	수동	수동	수동/자동생성
테스트 오라클	수동	수동	수동
테스트 수행 엔진	클라이언트 프로그램 / 브라우저	브라우저	클라이언트 프로그램 / 브라우저
테스트 수행	수동	수동	수동/자동수행

표 6에서 보는 것처럼 기존의 도구들은 테스트 준비 및 실행 과정에서 사용자가 수동으로 처리해야 하는 일들이 많다. 그러나 우리 도구는 많은 부분을 자동화함으로써 사용자들의 부담을 경감시킨다. 예를 들어, 본 논문에서 개발한 시스템을 이용하여 그림 12의 도서구매 애플리케이션의 23개의 인터페이스 메소드를 테스트할 경우 532LOC(Line of Code)의 테스트 드라이버와 추가적인 테스트수행 엔진 코드가 생성되었다. 또한 컨테이너 내부 테스팅 전략을 적용하기 위해 5개의 배치 지시자 파일이 수정되었으며, 3개의 추가적인 배치 지시자를 생성하였다. 그리고 테스트 수행을 위해 테스트 드라이버의 배치와 테스트 수행 이후 배치 해제 과정이 추가로 수행되었다. 이러한 모든 과정을 사용자가 수동으로 처리할 경우 많은 시간과 노력이 필요할 것이며, 컴포넌트의 구현이 복잡할수록 보다 더 많은 노력이 요구될 것이다. JUnitEE나 Cactus 등의 도구를 이용하면 테스트 대상 EJB 컴포넌트의 분석, 테스트 드라이버의 생성, 테스트 데이터의 생성 및 컨테이너 내부에 배치 등 일련의 테스트 준비 과정을 사용자가 모두 수행해야하기 때문에 수동 테스트와 비슷한 노력이 요구된다. 그러나 우리의 방법은 이러한 테스트 준비과정 및 수행 과정을 자동화하기 때문에 테스트 수행에 있어 많은 시간과 노력을 줄여줄 수 있다는 강점이 있다.

5. 결론 및 향후 연구 방향

EJB 컴포넌트를 테스트하기 위해서 기존의 자바 애플리케이션 테스트 방법을 그대로 적용하기 어렵다. 이는 EJB 컴포넌트가 컨테이너 내부에서 동작하고 컨테이너 서비스에 의존적이기 때문이다. 또한 EJB 컴포넌트 테스트를 위해서는 컴포넌트 분석, 테스트 드라이버 및 테스트 데이터 생성, 배치와 같은 많은 노력이 요구되기 때문에 자동화 방법이 필수적이다. 이에 본 논문에서는 컨테이너 내부 테스팅 전략을 기반으로 EJB 컴포넌트 테스트를 자동화하기 위한 방법에 초점을 맞춰 연구를 수행하였다.

EJB 컴포넌트 테스트 자동화를 위해 본 논문에서는 테스트 대상 컴포넌트 분석의 자동화, 테스트 드라이버 및 테스트 데이터 자동 생성과 생성된 테스트 드라이버를 자동으로 배치하기 위한 방법을 제시하고 도구로 구현하였다.

현재 구현된 도구는 EJB 컴포넌트의 인터페이스 메소드의 기능 테스트를 수행할 수 있다. 하지만 EJB 컴포넌트 테스트를 위해서는 인터페이스 메소드의 기능 테스트뿐만 아니라 배치할 때의 설정이나 다른 컴포넌트와의 연동 과정을 테스트하는 연동 테스트도 필요하다. 따라서 향후에는 구현된 컴포넌트와 다른 컴포넌트와의 연동 과정에 대한 연동 테스트 방법에 관해 연구할 것이다.

참 고 문 헌 (Reference)

[1] E. J. Weyuker, "Testing Component-Based Software: A Cautionary Tale", IEEE Software, vol. 15, issue. 5, 1998, http://dx.doi.org/10.1109/52.714817

[2] Y. S. Ma, et. al., "Framework for Third Party Testing of EJB Component Software", Proc. of 8th Asia Pacific Software Engineering Conference, pp.431-434, 2001, http://dx.doi.org/10.1109/APSEC.2001.991511

[3] Sun Microsystems Inc., The J2EE™ Tutorial for the Sun ™ ONE Platform, http://java.sun.com/j2ee/1.3/docs/tutorial/doc/

[4] Sun Microsystems Inc., Enterprise JavaBeans™ Specification, Version 2.0, http://download.oracle.com/otndocs/jcp/7294-ejb-2.0-fr2-spec-oth-JSpec/

[5] MockEJB, http://mockejb.sourceforge.net/

[6] JUnit, http://www.junit.org/

[7] Cactus, http://attic.apache.org/projects/jakarta-cactus.html

[8] JunitEE, http://sourceforge.net/projects/junitee/

[9] Scott W. Ambler, "A J2EE Testing Primer", http://www.drdobbs.com/a-j2ee-testing-primer/184414736

[10] Scott Stirling, "Testing J2EE applications", JavaWorld, Aug. 2004, http://www.javaworld.com/article/2072923/testing-debugging/testing-j2ee-applications.html

[11] Rod Johnson, "Expert One-on-One J2EE Design and Development", Wrox, 2002, http://www.wrox.com/WileyCDA/WroxTitle/Expert-One-on-One-J2EE-Design-and-Development.productCd-0764543857.html

[12] T. Mackinnon, S. Freeman, P. Craig, "Endo-Testing: Unit Testing with Mock Objects", in Extreme Programming Examined, Addison-Wesley, 2001, http://www.ccs.neu.edu/research/demeter/related-work/extreme-programming/MockObjectsFinal.PDF

[13] C. N. Oh, K. H. Lee, "Performance Measuring Method of Enterprise JavaBeans(EJB)", Proc. of KIISE Fall Conference, 2000, http://www.dbpia.co.kr/Article/450355

[14] H. J. Song, E. M. Choi, "Automated Generation of Wrapper to Test Components", Journal of KIISE: Software and Applications, vol. 32, no. 8, 2005, http://www.dbpia.co.kr/Article/505858

[15] H. J. Yoon, B. J. Choi, "A Testing Technique for Customized EJB Component", Journal of KIISE: Software and Applications, vol. 28, no. 3, 2001, http://www.dbpia.co.kr/Article/452067

[16] J. S. Kim, J. S. Kang, Y. S. Ma, Y. R. Kwon, "Component-based Software Testing Tool Using Test Data Generated From UML Models in XMI", Proc. of KIISE Spring Conference, 2002, http://www.dbpia.co.kr/Article/453905

[17] A. Orso, M. J. Harrold, D. Rosenblum, G. Rothermel, M. L. Soffa, H. S. Do, "Using Component Metacontent to Support the Regression Testing of Component-Based Software", Proc. of the IEEE Int'l Conference on Software Maintenance, Nov 2001, http://cse.unl.edu/~grother/papers/icsm01b.pdf

[18] J. Grundy, G. Ding, "Automatic Validation of Deployed J2EE Components Using Aspect", Proc. of the IEEE Int'l Conference on Automated Software Engineering, 2002, http://dx.doi.org/10.1109/ASE.2002.1114993

[19] M. Strembeck, U. Zdun, "Scenario-based Component Testing Using Embedded Metadata", Proc. of the Workshop on Testing of Component-based Systems, 2004, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.9220&rep=rep1&type=pdf

[20] S. H. Edwards, "A framework for practical, automated black-box testing of component-based software", Journal of Software Testing, Verification and Reliability, 2001, http://dx.doi.org/10.1002/stvr.224

[21] JTB: Java Tree Builder, https://java.net/projects/jtb

◐ 저 자 소 개 ◑

국 승 학 (Seung-hak Kuk)
2004년 충남대학교 컴퓨터과학과(이학사)
2006년 충남대학교 대학원 컴퓨터공학과(공학석사)
2012년 충남대학교 대학원 컴퓨터공학과(공학박사)
2012년~현재 국방과학연구소 연구원
관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 소프트웨어 아키텍처, etc
E-mail : shkuk@add.re.kr

김 현 수 (Hyeon Soo Kim)
1988년 서울대학교 계산통계학과(이학사)
1991년 한국과학기술원 전산학과(공학석사)
1995년 한국과학기술원 전산학과(공학박사)
1995년~1995년 한국전자통신연구원 Post Doc.
1996년~2001년 금오공과대학교 조교수
2001년~현재 충남대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 분산컴퓨팅, etc.
E-mail : hskim401@cnu.ac.kr