제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

테스트 기능 내장 컴포넌트의 설계와 구현

송호진, 최은만
동국대학교 컴퓨터멀티미디어공학과
e-mail : nemoz@dongguk.edu, emchoi@dgu.ac.kr

Design and Implementation of Built-In Test Component

Ho-Jin Song, Eun Man Choi
Department of Computer Science, Dongguk University

요 약

최근 소프트웨어 개발을 위한 각 분야에서 컴포넌트 기반 개발(Component Based Development)에 초점을 맞추고 많은 연구와 개발이 이루어지고 있다. 소프트웨어는 컴포넌트의 조립을 통해 완성되며 이는 비용과 시간의 절감, 검증된 컴포넌트 사용으로 인한 소프트웨어 신뢰성의 증가, 컴포넌트 개발을 통한 자산으로써의 가치 등을 고려해 봤을 때, 컴포넌트 기반 개발은 중요한 의미를 지니고 있다. 이러한 컴포넌트들은 컴포넌트가 지닌 기능이나, 성능을 테스트 하여 검증하는 과정이 매우 중요하다. 본 연구에서는 컴포넌트 테스트를 위한 BIT(Built-in Test)의 구현을 통해 컴포넌트 테스트를 수행하고 평가하기 위한 기초연구를 수행하였다.

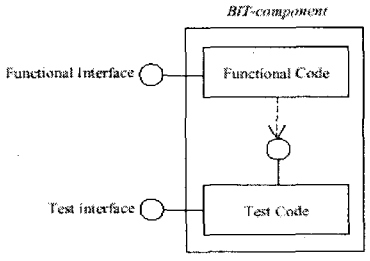
1. 서론

최근 소프트웨어 개발을 위한 각 분야에서 컴포넌트 기반 개발(Component Based Development)에 초점을 맞추고 많은 연구와 개발이 이루어지고 있다. 컴포넌트 조립을 통한 소프트웨어의 개발은 비용과 시간의 절감 등 많은 이점을 얻을 수 있으며, 이에 대한 컴포넌트의 기능 및 성능, 신뢰성을 검증하기 위한 과정은 매우 중요하다고 할 수 있다. 특히 서드파티(third-party)에서 제공하는 컴포넌트는 소스코드 형태로 배포되지 않으며, 이를 검증하기 위한 테스트 방법도 블랙박스(Black Box)형태의 테스트로 제한된다.

이로 인해 컴포넌트 기반의 소프트웨어 개발은 낮은 테스트 능력(testability)과 CBSE(Component Based Software Engineering) 행위자에 의한 낮은 유지 보수성, 런타임(run-time) 테스팅이 지원되지 않는 제약을 가지고 있다[5].

이는 BIT(Built-in Test)를 이용하여 컴포넌트의 내부에 테스트를 위한 기능들을 내장하여 컴포넌트의 본래 기능을 제공하는 기능적 인터페이스(functional interface)외에 시험에 필요한 테스트 데이터와 제어의 입출력을 위한 테스트 인터페이스를 추가로 제공하는 BIT 컴포넌트를 제작함으로써 테스트의 제약을 극복할 수 있다.

또한 컴포넌트에 내장된 BIT 는 소프트웨어의 라이프사이클(life cycle)전반에 걸쳐 재사용될 수 있으며 이를 통해 소프트웨어의 유지보수성을 증가시킬 수 있다[1].



[그림 1] BIT 컴포넌트의 개념[2]

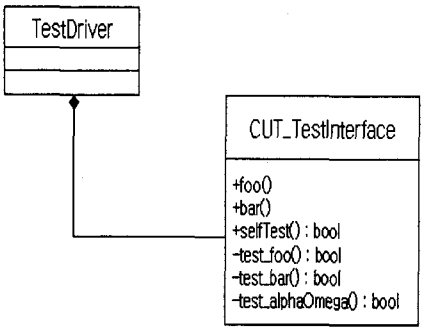
BIT 컴포넌트(그림 1)는 컴포넌트의 본래 기능을 수행하는 기능 인터페이스와(Functional Interface)와 하나 이상의 테스트 인터페이스(Test interface)가 합쳐진 형태로 되어있다. 컴포넌트 개발자는 이러한 인터페이스의 접근을 통해 컴포넌트의 통합 테스트(Integration

test)를 비롯한 계속적인 검증 작업을 수행할 수 있다.

2. 테스트 기능 내장의 개념

BIT 는 컴포넌트에만 국한된 테스트 방법은 아니다. Java 나 C++과 같은 객체지향 언어에서 테스트 하고자 할 때 클래스 내에 Private 으로 선언된 메소드나 인스턴스 변수를 외부에서 접근 할 수 있는 방법이 없기 때문에, 클래스 내부에 BIT 를 포함시키고 BIT 테스트 인터페이스를 통해 테스트를 수행할 수 있다.

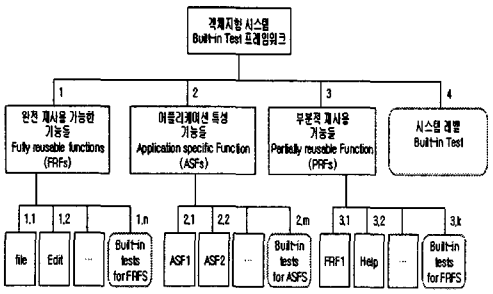
다음 그림 2 는 BIT Test Driver 을 이용한 테스트 수행 구조를 보여주고 있다.



[그림 2] Built-in Test Driver[3]

이러한 프로그램 언어상의 접근 제약에 대한 테스트 한계를 극복할 수 있는 유일한 방법으로써 Built-in Test 를 사용한다. Java 에서는 BIT 를 클래스의 main 부에 내장하는 경우와, BIT 를 내부 클래스(inner class)로 작성하여 외부 클래스(outer class)의 메소드와 변수를 직접적으로 접근하여 테스트 할 수 있는 방법이 있다. 또한 JUnit 과 같은 테스팅 프레임워크를 상속 받아 테스트에 이용할 수 있다.

단순한 하나의 객체 모듈에 대한 테스트를 진행할 수 도 있지만, 대부분 여러 개의 객체로 구성된, 객체지향 시스템 프레임워크에 대한 테스트가 필요하다. 이러한 테스트를 위하여 소프트웨어 시스템은 다음 그림 3 과 같은 BIT 클래스와 서브시스템으로 구성된다.

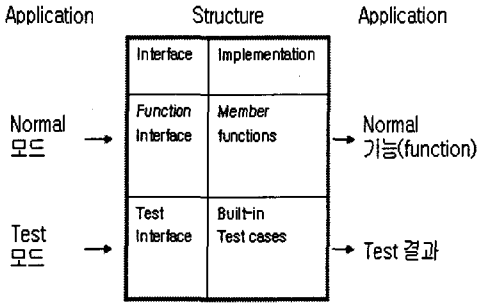


[그림 3] OO 시스템 Built-in Test 프레임워크[1]

시스템은 완전 재사용 가능 기능, 어플리케이션 특성 기능, 부분적 재사용 기능으로 분류하여 각각의 특성에 따라 BIT 를 이용한 개별적인 테스트를 수행하게 된다.

기능별 테스트가 완료되면, 시스템 프레임워크 전체의 시스템 수준의 BIT 를 이용한 시스템 레벨 테스트를 진행하게 된다.

이러한 BIT 오브젝트(그림 4)들은 Normal 과 Test 의 두 가지 모드로 나누어 진다. BIT 는 평상시 Normal 모드 상태로 동작하게 되며, 테스트 필요 시 Test 모드로 활성화되어 내장된 BIT 를 이용한 테스트를 수행하게 된다[4].



[그림 4] BIT Normal 모드와 Test 모드

최종 사용자는 Test 모드 상의 모든 BIT 멤버함수들을 재사용 할 수 있다. 이러한 BIT 방법을 사용함으로써 블랙박스(기능) 또는 화이트박스(구조)에 의해 생성된 테스트 케이스를 모두 시험해 볼 수 있다.

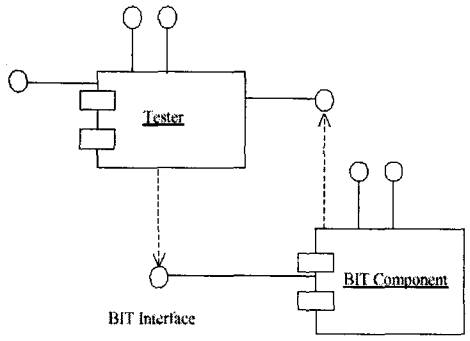
3. 테스트 내장형 컴포넌트 설계 방법

소프트웨어의 조각으로 개발된 컴포넌트의 테스팅은 기존의 테스트 방법과는 다른 성향을 나타낸다. 보편적으로 컴포넌트의 사용자는 컴포넌트의 내부에 대해서 잘 알지 못한다. 컴포넌트 사용자는 오직 외부에 보여지는 컴포넌트의 인터페이스와 행위만을 관찰할 수 있을 뿐이다. 이에 따라 BIT 메커니즘을 이용하여 외부로부터 소프트웨어 컴포넌트의 내부에 접근하는 방법은 반드시 필요하다.

소프트웨어 컴포넌트 BIT 는 두 가지 주요한 측면으로 분류될 수 있다[5].

- 계약 테스팅(Contract testing): 컴포넌트 간에 상호동작에 대한 관점에서, 컴포넌트가 명시된 계약사항에 맞게 동작하는지를 검증하는 것.
- 서비스 품질 테스팅(Quality of service testing): 서비스 품질 테스팅의 주요 초점은 운용 환경에서 소프트웨어 컴포넌트가 계속적으로 올바른 서비스를 제공하는지를 검증하는 것이다. 서비스 품질 테스팅은 컴포넌트와 컴포넌트가 전개되는 시스템 사이의 상호작용에 대한 테스트를 말한다.

BIT 컴포넌트는 BIT 인터페이스를 통해 연결된 테스터(tester)을 이용해 테스트를 수행하게 되며, 테스터와 BIT 컴포넌트와의 관계는 그림 5 와 같은 클라이언트/서버 관계의 형태로 나타낼 수 있다.



[그림 5] BIT 컴포넌트와 Tester 와의 관계

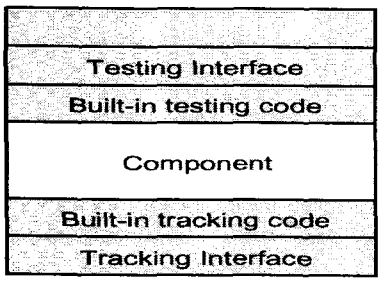
테스터는 BIT 컴포넌트의 BIT 인터페이스를 통해 테스트 오퍼레이션을 호출하여 테스트를 수행하고 수행 결과를 테스터의 인터페이스를 통하여 전달받게 된다. 테스터는 소프트웨어 컴포넌트 라이프사이클 전반에 걸쳐 재사용되며, 실행 환경 및 실행 시간에 영향을 받지 않고 테스트를 수행할 수 있는 장점을 지닌다.

4. 사례 연구 및 실험

EJB(Enterprise Java Beans)는 현재 컴포넌트 개발을 위한 컴포넌트 표준으로 자리잡고 있다. 현재 엔터프라이즈 급 컴포넌트 기반 소프트웨어 개발을 위한 표준으로써 가장 많이 사용되고 있다.

EJB 를 통한 BIT 컴포넌트 개발을 통해 컴포넌트의 테스트 능력(testability)를 높이고 플러그 인 테스트(plug-in-and-test) 환경을 제공하며 컴포넌트 테스트 자동화를 이룰 수 있다[6].

테스트 가능한 BIT 빈(Bean)의 구조는 그림 6 에 나타나 있다.



[그림 6] BIT 빈의 구조[6]

BIT 빈은 테스팅 인터페이스를 통해 컴포넌트 내부의 BIT 의 테스팅 코드를 수행하여 Built-in tracking 코드를 통해 테스팅 결과나 테스팅 수행내용을 기록(tracking)하게 된다. 기록 결과는 외부의 파일이나, 데이터베이스에 저장하게 된다.

테스트 인터페이스는 다음과 같은 구조를 통해 3 가지 기본 기능을 제공하게 된다.

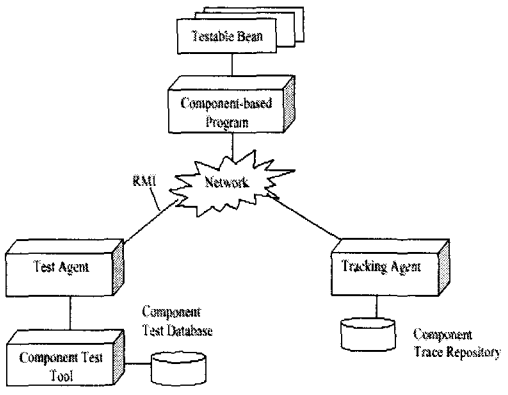
- 테스트 셋업(테스트 케이스 또는 데이터)
  : setParameters 를 통해 오퍼레이션의 파라메터를 셋팅한다.
- 명세된 기능에 대한 테스트 수행
  : runMethod 를 통해 주어진 오퍼레이션을 수행
- 테스트 결과 보고와 확인
  :validateTestResult 를 통해 테스트 결과를 확인한다.

J2EE 환경에서의 컴포넌트들은 여러 시스템에서 분산되어 전개될 확률이 매우 높다고 할 수 있다. 이로 인해 단순한 객체나 객체 지향 시스템 프레임워크와는 다른 형태의 테스트가 필요하다.

분산환경의 BIT 컴포넌트 테스팅을 위해서 다음과 같은 기본 환경이 갖춰져야 한다[6].

- BIT 빈(테스트 가능한 BIT 컴포넌트)
- 테스트 에이전트(Test agent)
- 컴포넌트 테스트 저장소(Component test repository)
- 기록 에이전트(Tracking agent)
- 컴포넌트 기록 저장소(Component tracking repository)

J2EE(Java 2 Enterprise Edition)에서의 분산환경은 네트워크를 통해 각 분산 컴포넌트 객체들이 RMI 프로토콜을 이용하여 데이터를 주고 받게 된다. 다음 그림 7 은 분산환경에서의 BIT 컴포넌트 테스팅을 위한 환경을 나타내고 있다.

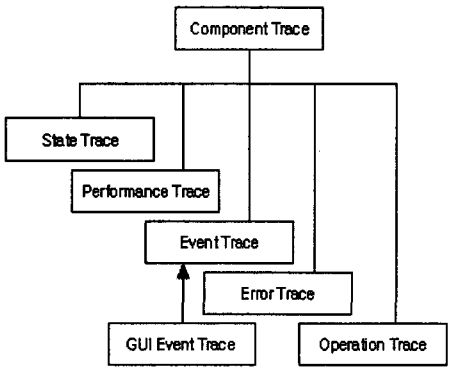


[그림 7] 분산환경에서의 BIT 빈 테스트[6]

각각의 분산된 BIT 를 내장한 빈들은 컴포넌트 테스트 툴에 관련된 테스트 에이전트를 통해 네트워크를 통한 RMI 프로토콜을 이용해 각각의 테스트 오퍼레이

션을 호출하여 수행하게 된다. 또한 기록 에이전트를 통해 테스트가 수행된 내용과 결과를 컴포넌트 기록 저장소에 저장하게 되는 구조를 가지고 있다.

기록 에이전트에 의해 기록되는 내용은 다음 그림 8 과 같은 구조로 되어 있다.



[그림 8] BIT 컴포넌트 테스트 기록 구조

컴포넌트 기록(Component trace)는 다음과 같은 다섯 유형으로 분류된다.

- Operational trace : 컴포넌트 오퍼레이션 간의 상호작용(interaction)을 기록.
- Performance trace : 컴포넌트가 전개된 환경에서의 각 컴포넌트 기능에 대한 벤치마크와 성능 데이터를 기록.
- State trace : 컴포넌트 내의 데이터의 상태 또는 객체의 상태를 기록.
- Event trace : 컴포넌트 내에서 발생한 이벤트를 기록. GUI 컴포넌트의 이벤트기록을 통해 GUI 수행 시나리오를 재연할 때 유용.
- Error trace : 컴포넌트에 의해 생성된 에러 메시지나 예외상황에 대한 정보를 기록.

이렇게 저장된 정보는 컴포넌트 유지보수 측면이나 컴포넌트 재사용 측면에서 매우 중요한 정보로 이용될 것이다. 이러한 구조는 .NET 과 같은 표준 컴포넌트 프레임워크에도 같이 적용될 수 있다. 이러한 테스트를 수행함으로써 단순한 컴포넌트 결함을 찾아내는 것뿐만 아니라, 컴포넌트간의 행위 및 컴포넌트의 수행 능력을 분석, 측정할 수 있다.

5. 결론 및 향후 연구

본 연구에서는 객체단위 및 객체 지향 프레임워크에 관련된 BIT 에 대한 조사와, 객체지향 소프트웨어 컴포넌트 단위의 BIT 컴포넌트에 대한 설계와 구현을 위한 기초 연구를 수행하였다.

객체지향 소프트웨어 컴포넌트는 캡슐화의 특성으로 인해 외부에서 내부의 행위나 구조를 알 수 없어, 테스트에 어려움이 따르게 된다. 이러한 문제점을 BIT 를 이용하여 해결할 수 있다. BIT 재사용의 측면에서 소프트웨어 테스트에 소요되는 비용과 수고를 줄일 수 있으며, 소프트웨어 품질을 높일 수 있다. 또한 안정적인 테스트 모델을 제공하고, 서드파티(third-party) 컴포넌트들의 외부 행위를 모니터링 할 수 있으며, 유연성 있는 테스트 제어 능력을 제공하게 된다.

향후 연구로서 BIT 가 가능한 분산 컴포넌트의 테스트를 수행할 수 있는 테스트 툴과 이에 관련된 컴포넌트 테스트 에이전트의 실제 구현이 필요하다. 또한 분산된 웹 컴포넌트가 BIT 로 구성되어 있다면 단순히 같은 컴포넌트 플랫폼 상의 컴포넌트 테스트뿐만이 아니라, 이 기종 플랫폼간의 상호 연동이 가능한 웹 서비스와 같은 기술을 이용하여 서로 다른 소프트웨어 컴포넌트 프레임워크간의 컴포넌트 테스트도 가능 할 것이다.

참고문헌

[1] Yingxu Wang ,"A Method for Built-in Tests in Component-based Software Maintenance", Centre for Software Engineering.

[2] J Vincent, "Built-In-Test Vade Mecum – Part1, A Common BIT Archicture", Southampton Institute/Bournemouth University, UK.

[3] Robert B.Binder, "Testing Object-Oriented Systems, Models, Patterns, And Tools", ADDISON-WESLEY.

[4] Yingxu Wang, "On Built-in Test Reuse in Object-Oriented Framework Design", South Bank University.

[5] Hakan Edler, "BIT in software component", EC IST 5th Framework.

[6] Jerry Gao, "On Building Testable Software Components", San Jose State University.

[7] Jerry Gao, "Monitoring Software Components and Component-Based Software", San Jose State University.

[8] Jerry Gao, "Component Testability and Component Testing Challenges", San Jose State University

[9] Hans-Gerhard Gross, "Built-in Contract Testing in Component-based Application Engineering", Fraunhofer Institute for Experimental Software Engineering.

[10] Stephen H. Edwards, "A Framework for Practical, Automated Black-Box Testing of Component-Based Software", Virginia Tech, Dept. of Computer Science.

[11] Jonas Hornstein, "Test Reuse in CBSE Using Built-in Tests", IVF Industrial Research and Development Corporation, Sweden

[12] Hans-Gerhard Gross, "Built-in testing for Component-based Development", Fraunhofer Institute for Experimental Software Engineering.

[13] J Vincent, "Built-In-Test Vade Mecum – Part3, Quality-of-Service Testing", Southampton Institute/Bournemouth University, UK.