한국시뮬레이션학회 논문지
Vol. 20, No. 4, pp. 41-48 (2011. 12)

모형 객체 패턴을 이용한 Federation 통합시험 방법

심준용[1†] · 이용헌[1] · 이승영[1] · 김세환[1]

A Method of Integration Testing for Federation using Mock Object Patterns

Junyong Shim · Youngheon Lee · Seungyoung Lee · Sehhwan Kim

ABSTRACT

The act of writing a unit test is more an act of design than of verification. It is also more an act of documentation than of verification. The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function. Unit testing is a fundamental practice in Extreme Programming, but most non-trivial code is difficult to test in isolation. Normal unit testing is hard because It is trying to test the code from outside. On the other hand, developing unit tests with Mock Objects leads to stronger tests and to better structure of both domain and test code. In this paper, I first describe how Mock Objects are used for unit testing of federation integration. Then I describe the benefits and costs of Mock Objects when writing unit tests and code. Finally I describe a design of Mock federate for using Mock objects.

Key words : Mock Object, Test Framework, Federation Test, Unit Test

요 약

개발 기능에 대한 단위시험 모듈을 구현할 경우 도메인 구현부와 시험 구현부의 종속성이 높기 때문에 단위시험 모듈의 재사용이 어렵다. 특히, 동일한 구조나 기반 프레임워크를 재사용하는 시스템의 경우 구성 소프트웨어의 내부 인터페이스를 위한 단위시험 모듈의 중복이 불가피하며, 통합 시험 코드는 해당 모듈 간 연동 인터페이스 구현에 종속되기 때문에 각 모듈의 개발 일정에 따라 단위시험 수행이 제한될 수 있다. 이러한 문제를 해결하기 위해서 TDD 기법 중 하나인 모형 객체(Mock Objects) 패턴을 이용한 단위시험 방법이 제안되었다. 이 방법은 도메인 모듈과 시험 모듈을 분리할 수 있도록 도메인 모듈을 대리하는 모형 객체를 생성하고, 해당 모형 객체를 시험 모듈과 통합함으로써 단위시험 모듈의 구현을 용이하게 한다. 본 논문은 HLA 시뮬레이션 시스템 개발에 참여하는 Federate의 Federation 통합 및 연동 시험을 용이하게 하기 위해서 모형 객체를 적용한 모형 Federate를 설계하고, 모형 Federate의 구성 모듈을 위한 테스트 프레임워크를 제안한다. 제안 프레임워크는 RTI 서비스를 위한 시험 함수를 제공하며, 해당 함수들은 xUnit 패턴에 의해 자동화 된다.

주요어 : 모형 객체, 테스트 프레임워크, 페더레이션 테스트, 단위 시험

1. 서 론

시스템의 외부 환경이 급변하고 소프트웨어 기술 요소 또한 복잡하고 다양해지는 상황에서 요구 기능에 대한 체계적이고 효율적인 구현 방법을 찾는 문제는 시스템 개발의 성공 여부를 좌우할 만큼 중요하며, 특히 개발 시스템의 구성 모듈에 대한 지속적인 시험 및 통합은 신뢰성 있는 소프트웨어 개발을 위한 필수적인 요소로 인식되고 있다. 최근 개발자들로부터 각광받고 있는 테스트 주도 개발(Test Driven Development, 이하 TDD)은 구현과 동시에 시험 수행을 위한 방법론을 제시하며, 반복적인 단위시험을 통한 지속적 통합을 수행함으로써[1], 개발 시스템의 요구 기능에 대한 품질을 빠르게 확보할 수 있다.

한편, 기능에 대한 단위시험 모듈을 구현할 경우 도메

접수일(2011년 7월 13일), 심사일(1차 : 2011년 11월 8일), 게재 확정일(2011년 11월 8일)
[1] LIG넥스원(주) Maritime연구소
주 저 자 : 심준용
교신저자 : 심준용
E-mail; jyshim79@lignex1.com

인 구현부와 시험 구현부의 종속성이 높기 때문에 단위시험 모듈의 재사용이 어렵다. 특히, 동일한 구조나 기반 프레임워크를 공유하는 시스템의 경우 구성 소프트웨어의 내부 인터페이스를 위한 단위시험 모듈의 중복이 불가피하며, 상호작용 모듈을 위한 시험 코드는 해당 모듈 간 인터페이스 구현에 종속되기 때문에 각 모듈의 개발 일정에 따라 단위시험 수행이 제한될 수 있다. 이러한 문제를 해결하기 위해서 TDD 기법 중 하나인 모형 객체(Mock Objects) 패턴을 이용한 단위시험 방법이 제안되었다[2]. 이 방법은 도메인 모듈과 시험 모듈을 분리할 수 있도록 도메인 모듈을 대리하는 모형 객체를 생성하고, 해당 모형 객체를 시험 모듈과 통합함으로써 단위시험 모듈의 구현을 용이하게 한다.

본 논문은 모형 객체 패턴을 적용한 HLA(High Level Architecture) 시뮬레이션 시스템의 단위시험 모듈을 설계하고, 모형 객체 기반의 Federation 통합시험 방법을 제시한다. 특히, 모형 객체를 구현한 모형 Federate의 구조와 적용 방법을 보여준다. 본 논문의 구성은 다음과 같다. 2장은 단위시험 패턴과 모형 객체를 이용한 TDD 기법을 살펴보고, 3장에서 HLA 시뮬레이션 시스템의 Federation 통합시험 구조와 모형 Federate의 필요성을 보여준다. 4장은 모형 객체를 구현한 모형 Federate의 구조와 모형 Federate를 활용한 Federation 통합시험 방법을 기술한다. 끝으로 5장에서 결론 및 향후 연구 방향을 제시한다.

2. 단위시험 패턴과 모형 객체

2.1 xUnit 패턴

개발 소프트웨어의 요구 기능에 대한 시험을 자동화한다면 단위시험 모듈이 늘어나거나 지속적인 회귀 시험(Regression Test)이 필요한 경우 상당한 비용을 줄일 수가 있다. xUnit은 xUnit 계열의 시험 프레임워크를 위한 단위시험 패턴이며[3], 시험 자동화(Test Automation) 구현을 위해 단언(Assertion), 픽스처(Fixture), 외부 픽스처, 테스트 메서드, 예외 테스트, 전체 테스트 등의 구성 요소를 정의하고 있다. xUnit 패턴을 구현한 시험 도구는 SUnit(Smalltalk), JUnit(Java), CppUnit(C++), NUnit (.NET) 등이 사용된다[4]. 구성 요소를 살펴보면 다음과 같다.

- 단언(Assertion): 테스트의 정상적인 작동 여부에 대한 검사 방법을 제시하며, 불리언(boolean) 수식을 작성하여 프로그램이 자동으로 코드의 동작 여부를 판단할 수 있도록 assertTrue(arg), assertEquals(arg) 등의 서비스를 구현한다.

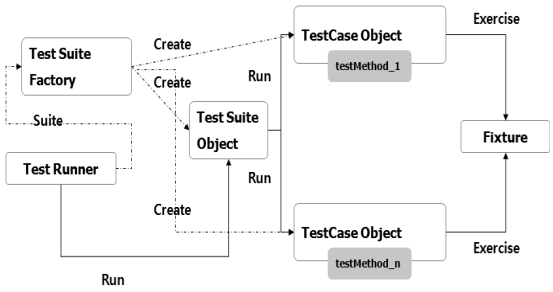

그림 1. xUnit 구조

- 픽스처(Fixture): 다수의 테스트 케이스에서 공통으로 사용하는 객체들을 생성하는 방법을 제시하며, setUp() 함수를 통해 객체 초기화를 구현한다.
- 외부 픽스처: 테스트가 실행된 후 실행되기 전 상태와 동일하게 유지하도록 객체의 설정을 해제(Release)하는 방법을 제시하며, tearDown() 함수에서 구현된다.
- 테스트 메서드: 수행되는 테스트 케이스 단위를 표현하는 방법을 제시하며, ‘test’로 시작하는 이름의 함수로 구현된다.
- 예외 테스트: 예외가 발생하는 상황이 정상적인 경우의 테스트 작성 방법을 제시하며, 의도한 예외가 발생하지 않은 경우에 테스트를 실패하도록 fail() 함수에 구현한다.
- 전체 테스트: 작성된 모든 테스트를 한 번에 실행할 수 있는 방법을 제시하며, 테스트 케이스에 대한 테스트 슈트(Test Suite)를 구현한다.

xUnit의 구조를 정리하면 그림 1과 같으며 단위시험 객체인 TestCase Object에 픽스처, 외부픽스처, 테스트 메서드를 구현하고, TestSuite 객체에 등록함으로써 반복 수행하는 구조를 보여준다.

2.2 모형 객체(Mock Object)

단위 시험은 한 번에 하나의 기능을 시험하기 위한 방법을 제시한다. 즉, 도메인 코드의 요구 기능에 대한 만족 여부를 판단하기 위한 단일 모듈 또는 함수에 대한 시험 코드를 작성한다. 모형 객체는 개발 모듈과 동일한 인터페이스를 갖는 객체로서 의존성을 갖는 모듈 간 단위 시험을 위해 사용되며, 시험 모듈 작성을 위한 환경 구축이 어렵거나 특정 경우나 순간에 의존적인 경우 모형 객체로 구현될 수 있다. 실 객체를 대신해서 시험을 진행할 수 있도록 생성되는 객체를 통칭하여 시험 대역(Test Double)이라고 하는데 모형 객체는 그림 2와 같이 시험 대역의

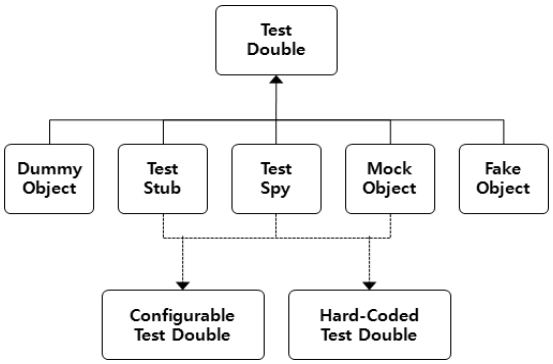

그림 2. Test Double 구성요소

하위로 분류된다[5].

시험 대역 분류에 따라 적용되는 방법은 다음과 같다.

- 더미 객체(Dummy Object): 시험 모듈에 대한 Instance 생성 수준을 제공하는 객체로서 해당 객체에서 제공되는 함수는 널(Null)로 구현된다. 기능에 대한 구현을 포함하지 않기 때문에 시험 대상이 되는 객체로부터 호출될 경우 정상 동작은 보장하지 않는다.
- 테스트 스텁(Test Stub): 더미 객체와 마찬가지로 Instance 생성 수준을 제공하지만 특정 상태를 가정하여 해당 객체의 함수를 구현한다. 시험 대상이 필요로 하는 정보를 반환하거나 메시지를 출력하는 등의 기능을 수행할 수 있다.
- 테스트 스파이(Test Spy): 시험 대상이 되는 객체로부터 특정 객체의 사용 여부 또는 의도한 함수의 정상적인 호출 여부를 확인하기 위해서 기록 정보를 전달하는 함수로 구현된다. 일반적으로 대상 모듈의 기능과 관계없이 시험 전용의 함수로 제공된다.
- 페이크 객체(Fake Object): 테스트 스텁 객체와 비슷한 기능을 수행하지만 생성 Instance가 다수의 객체를 대표하거나 실 객체에서 사용되는 로직을 단순화하여 함수를 구현한다.
- 모형 객체(Mock Object): 모형 객체를 제외한 시험 대역은 시험 모듈의 상태(State)를 기반으로 시험을 수행하지만 모형 객체는 행위(Behavior)를 기반으로 시험을 수행한다. 즉, 시험 모듈에 필요한 다수 객체의 생성 및 호출의 절차를 정의하고, 선행 및 후행 조건의 만족 여부를 검증할 수 있도록 구현한다.

그림 3에서 볼 수 있듯이 시험 모듈 개발자는 시험 기능을 SUT(System Under Test)에 작성하고, 시험 대역과 픽스처를 통해 시험을 수행하게 된다.

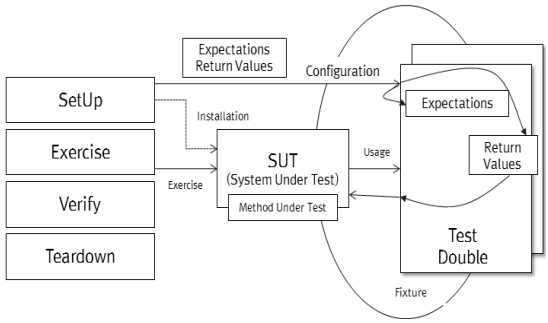

그림 3. 시험 대역(Test Double) 구성도

3. Federation 통합 및 모형 Federate

3.1 HLA / RTI

최근 국방 분야는 모델링 및 시뮬레이션(Modeling and Simulation)을 기반으로 하는 시뮬레이션 시스템 개발을 늘리고 있으며, 특히 분산 시뮬레이션 표준 프레임워크인 HLA를 적용함으로써 개발 소프트웨어의 재사용 및 상호운용성 확보를 위해 노력하고 있다. HLA는 규칙(Rules) [6], 인터페이스 명세서(Interface Specification)[7] 및 객체모델 템플릿(Object Model Template)[8]의 주요 컴포넌트들로 구성되며, 모의 단위인 Federate와 연동 환경인 Federation을 정의한다. 특히, 연동객체 모델(Federation Object Model, 이하 FOM)을 생성하여 교환 정보를 기술하고, 인터페이스 명세서를 구현한 RTI(Run-Time Infrastructure)를 통해 서비스를 제공한다.

RTI는 연합 관리, 선언 관리, 객체 관리, 시간 관리, 소유권 관리 그리고 데이터분산 관리를 제공하기 위한 API를 제공하며, RTIambassador 및 FederateAmbassador 객체를 통해 Federate 간 정보를 교환할 수 있다.

3.2 Federation 개발 및 통합시험 절차

HLA 시뮬레이션 시스템은 시뮬레이션 대상 소프트웨어인 Federate를 개발하고, 동일 Federation 환경을 구성하여 Federate의 통합 및 정보를 교환한다. 각 Federate는 연동 모델인 FOM를 참조하여 상호교환을 위한 인터페이스를 구현한다. 한편, Federation 통합을 위한 Federate 간 인터페이스 시험은 개별 Federate의 개발 일정에 의존한다. HLA Federation의 개발 모델인 FEDEP(Federation Development and Execution Process)[9]를 살펴보면, 1단계 Federation 목적 정의, 2단계 Federation 개념모델 개발, 3단계 Federation 설계, 4단계 Federation 개발, 5단계

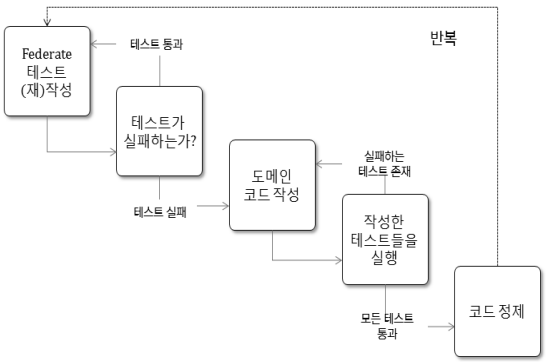

그림 4. TDD기반 시험 절차

Federation 통합 및 시험 그리고 6단계 Federation 실행 및 결과의 절차를 수행하는데 Federation 통합 및 시험 단계에 이르러 FOM을 적용하여 Federate 간 연동 시험을 수행하게 된다. 즉, Federate의 연동 인터페이스를 위한 단위 시험은 Federate 간 의존성으로 인해 통합 시험 단계까지 지연된다.

한편, Federate가 개발되는 Federation 개발 단계는 TDD 기법을 적용하여 통합 단계 전까지 연동 인터페이스 시험을 수행할 수 있다. 즉, Federate 개발과 동시에 연동 인터페이스에 대한 단위 시험을 수행함으로써 통합 시험의 오류 및 비용을 감소시킬 수 있다. 그림 4는 TDD 기반의 Federate 시험 절차를 보여준다.

TDD 적용을 위해서 시험 대역의 시험 패턴을 선택하고 해당 객체를 구현하기 위한 단위 시험 모듈들의 설계 및 구현이 이루어져야 한다.

3.3 HLA 시뮬레이션 프레임워크

Federate 개발을 위해 HLA 기반 시뮬레이션 프레임워크가 개발되었으며[10], HLA 서비스 제공을 위해 RTI 기능을 구현하고 있다. 개발 프레임워크는 크게 사용자 GUI 통신을 위한 UI관리자, 구현 모델 정보를 관리하는 모델 관리자, 원격 개체 제어를 위한 통제 관리자, 모든 관리자를 통제하는 설정통제 관리자, 링크 모의 개체 정보를 관리하는 파라미터 관리자, 시뮬레이션 시나리오를 관리하는 시나리오 관리자 그리고 RTI 서비스를 구현한 시뮬레이션 네트워크 관리자로 구성되어 있다(그림 5).

각 관리자는 메시지 디스패처 프로토콜을 사용하여 메시지를 교환하고, Federate 간 객체 정보는 RTI를 통해서 교환한다(그림 6).

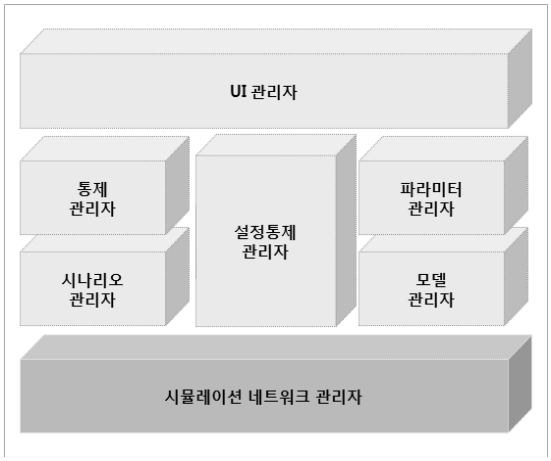

그림 5. HLA 시뮬레이션 프레임워크

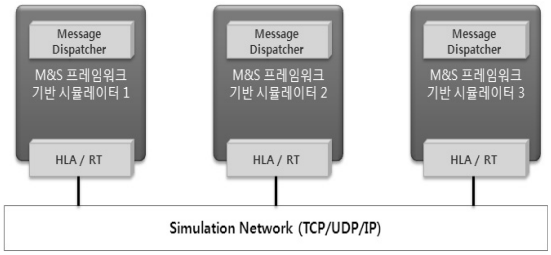

그림 6. 프레임워크 상호작용 구조

프레임워크 구조에서 볼 수 있듯이 각 Federate의 내부 관리자에 대한 기능은 개발자가 직접 구현하기 때문에 동시에 단위 시험 모듈 구현이 가능하지만, 외부 연동 인터페이스를 구현하는 시뮬레이션 네트워크 관리자는 상대 관리자가 제공되어야 한다. 즉, 각 Federate의 개발 완료 시점에 이르러 RTI 기능 시험을 수행할 수 있다.

3.4 모형 Federate의 필요성

Federation에 참여하는 Federate가 많을수록 상호작용하는 메시지가 늘어나므로 인터페이스에 대한 시험 비용은 높아진다. 또한 각 Federate의 개발 시점이 다를 경우 통합 연동을 위한 시험이 지연될 수 있다. 물론 개별적으로 연동 시험을 위한 모듈을 작성할 수 있지만 재사용에 대한 제약을 받는다. 따라서 Federate의 연동 인터페이스 시험에 필요한 모형 Federate의 제공은 Federation 통합 및 시험 단계에서 오류를 줄이기 위한 좋은 방안이 될 수 있다. 모형 Federate는 RTI의 기능을 제공하는 RTIambassador와 FederateAmbassador 객체에 대한 단위 시험 모듈 구

현을 위해서 xUnit 도구와 모형 객체 패턴을 적용할 수 있다.

4. 모형 객체를 이용한 Federation 통합시험

4.1 모형 Federate 프레임워크

시험 대역(Test Double)은 다양한 모형 패턴을 포함하지만 모형 객체(Mock Object)는 시험 대상의 상태뿐만 아니라 행위를 시험할 수 있기 때문에 RTI 서비스와 같이 전후 관계에 의존적인 시험 모듈 구현을 위한 구조를 제공할 수 있다. 본 논문에서 제안하는 모형 Federate의 구조는 그림 7과 같다. xUnit 구현물인 CppUnit 도구가 적용되었다.

Mock_TestFederate는 Mock_RTIambassador와 Mock_FederateAmbassador 객체를 구현하고, xUnit의 구성 요소인 setUp(), tearDown() 및 TestMethod()를 생성한다. TestMethod()는 실제 시험 대상이 요구하는 도메인 코드가 삽입된다. TestMethod() 실행에 필요한 RTI 서비스는 Mock_TestFederate로부터 제공된다.

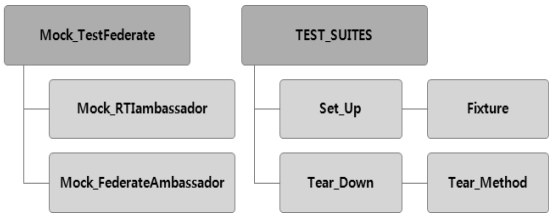

그림 7. 모형 Federate 구조

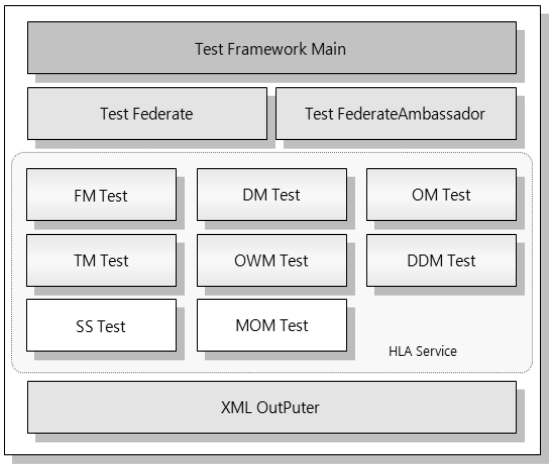

그림 8. 테스트 프레임워크 구조

Mock_RTIambassador 객체는 RTI의 6가지 서비스에 대한 모형 함수를 제공하며, Test Federate로부터 관리된다. Mock_FederateAmbassador 객체는 Callback 함수를 제공하며, Test FederateAmbassador로부터 관리된다. 모형 Federate 구현을 위한 테스트 프레임워크 구조는 그림 8과 같다.

제안 프레임워크의 각 컴포넌트 기능은 다음과 같다.

• TestFederate 모듈

Federation 생성 및 삭제, Federate의 가입 및 탈퇴, FOM 정보에 대한 (Un)Publish 및 (Un)Subscribe, 그리고 업데이트 기능과 같이 단위 시험 작성 시 공통으로 사용되는 RTIambassador 객체에 대한 API를 제공한다. 가령, Federate 가입에 해당하는 joinFederationExecution() 기능을 시험할 경우 Federation 생성에 대한 모형 함수로 quickCreate() 기능을 수행할 수 있다. RTI의 6가지 서비스에 대한 기능들 또한 quickCreate(), quickPublish(), quickUpdate()와 같이 quickXXX() 형태로 제공된다.

• TestFederateAmbassador 모듈

FederateAmbassador 인터페이스를 상속받아 구현한 모듈로서 RTI로부터 수신되는 Callback 함수를 구현하고 있다. 서비스 별 수행 절차에 대한 검증이 필요할 경우 사용자로부터 구현되어야 한다.

• HLAService 모듈

RTI의 FM, DM, OM, OWM, TM 그리고 DDM Test의 6가지 서비스와 지원 및 관리 서비스에 대한 시험 케이스가 작성되어 있다. 특히, CppUnit의 TestFixture 클래스를 상속받아 setUp()과 tearDown()을 구현하며, TEST_SUITE에 해당 서비스들을 등록하여 시험 자동화를 위한 환경을 구성한다.

• TestFrameworkMain 모듈

HLAService 모듈에 포함된 각 시험 함수의 구현 객체를 시험 실행기(Test Runner)에 등록하고, 수행된 결과를 XMLOutPuter에 전달한다.

• XMLOutPuter 모듈

시험 결과를 XML로 생성하고 스키마 파일을 통해서 클라이언트 프로그램(브라우저)으로 출력한다.

4.2 Federation 통합시험

시험 대상 Federate는 모형 Federate와 연동 시험을 수행하기 위해서 Federation을 생성하고, 시험 대상과 상호작용하는 교환 메시지를 FOM으로부터 추출한 후 모형 Federate 객체와 HLAServices 모듈을 사용하여 구현한다.

모형 Federate의 구현 예제는 아래와 같다.

```
class MockTest : public TestFixture
{
public:
    void setUp();
    void tearDown();

    void testUpdateInstance();
    ..... // Test Function

    CPPUNIT_TEST_SUITE( MockTest );
    CPPUNIT_TEST( testUpdateInstance );
    ..... // Register Test
    CPPUNIT_TEST_SUITE_END();
```

```
void MockTest::setUp()
{
    testFederate->quickCreate();
    testFederate->quickJoin();
    testFederate->quickPublish();
    ..... // Initialize
}
void MockTest::tearDown()
{
    testFederate->quickResign();
    testFederate->quickDestroy();
}
void MockTest::testUpdateInstance()
{
    ..... // Impl
    rtiAmb->updateAttributeValues();
}
```

구현된 모형 Federate는 시험 대상이 되는 시뮬레이션 네트워크 관리자의 인터페이스에 맞게 시험 케이스를 작

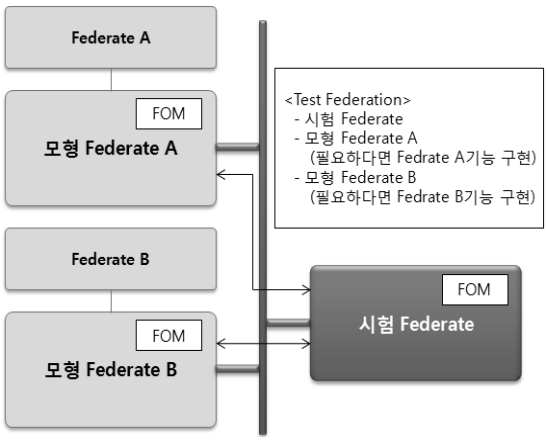

그림 9. 모형 Federate를 활용한 통합시험

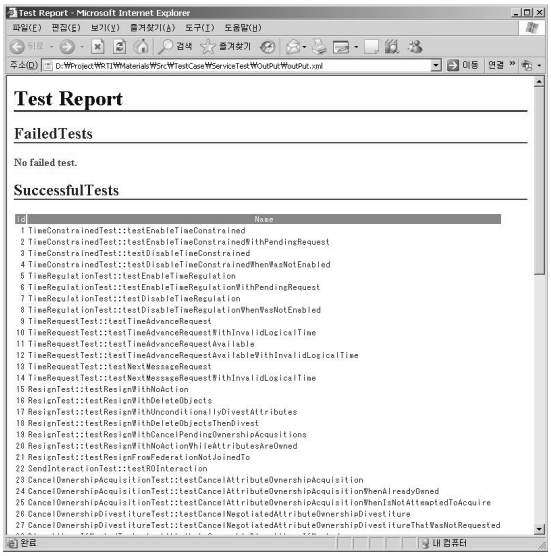

그림 10. 모형 Federate 시험 결과 출력

성한 후 시험 실행기에 등록함으로써 시험을 수행할 수 있다. 만약 다수 Federate와 연동이 필요한 시험 Federate인 경우 모형 Federate를 해당 개수만큼 생성하여 시험할 수 있다(그림 9).

그림 10은 모형 Federate의 시험 결과에 대한 출력 예시를 보여준다.

4.3 모형 Federate의 특징

앞에서 살펴본 바와 같이 모형 객체는 시험 대상 객체의 상태 및 행위(또는 절차)를 시험할 경우 사용되는 시험 대역으로써 시험 대상인 시뮬레이션 네트워크 관리자 기능을 대신할 수 있는 모형 Federate의 설계를 제공하며, xUnit 패턴을 적용함으로써 구성 요소에 대한 재사용 구조와 자동화 시험 환경을 제공한다. 제안된 모형 Federate를 통해서 개발 Federate의 일정에 관계없이 연동 인터페이스 시험이 가능하며, 도메인 코드와 시험 코드를 독립적으로 구현할 수 있기 때문에 시험 모듈에 대한 재사용성을 높일 수 있다. 특히, 개발 Federation에 참여하는 모든 Federate들에게 정형화된 시험 방식을 제공하기 때문에 시험 결과에 대한 신뢰성을 확보할 수 있다.

5. 결 론

본 논문은 HLA 시뮬레이션 시스템을 개발하는 과정에서 Federation 통합 및 연동 시험의 오류를 줄이기 위한

방법으로 참여 Federate의 단위 시험에 대한 필요성을 기술했다. 특히, 시험 대역(Test Double)의 모형 객체(Mock Object)를 적용하여 개발과 동시에 시험을 수행할 수 있는 TDD 기법을 적용하고, 도메인 코드와 시험 코드를 분리할 수 있도록 xUnit 패턴을 적용하여 모형 Federate를 설계했다. 또한 모형 Federate의 구성 모듈을 재사용할 수 있도록 테스트 프레임워크를 제안했다.

모형 Federate는 시험 Federate 간 연동 인터페이스 구현을 위해 FOM으로부터 객체 모델 정보를 추출하고, 단위 시험 구현을 위해서 Mock_RTIAmbassador와 Mock_FederateAmbassador 객체로부터 제공되는 기능을 사용하면 된다. 제안된 테스트 프레임워크는 이러한 모듈을 제공함으로써 다수의 모형 Federate 구현을 용이하게 한다.

한편, 모형 Federate는 모형 객체 패턴의 장점뿐만 아니라 단점도 갖는다. 예를 들어, 시험 대상 규모가 커짐에 따라 복잡한 절차 및 로직을 구현할 수 도 있으며, 이는 요구 기능뿐만 아니라 시험 기능에 대한 개발 일정이 추가로 요구될 수 있다. 따라서 Federation 설계자는 Federation에 참여하는 Federate 개수, FOM의 복잡도에 따라 모형 Federate의 적용 여부를 판단해야 한다.

참 고 문 헌

1. Patton, Ron, “Software Testing 2ND”, Macmillan Computer Pub, 2005, ISBN-10 0672327988.
2. Tim Machinnon, Steve Freeman, and Philip Craig. Endo-testing: unit testing with mock objects. pages 287-301, 2001.
3. Kent back, “Test-Driven Development: By Example”, Addison Wesley, 2002, ISBN 0321146530.
4. Elfriede Dustin, Thom Garrett, Bernie Gauf, “Implementing Automated Software Testing”, Addison Wesley, 2009, ISBN-10 0321580516.
5. Gerard Meszaros, “xUnit Test Patterns: Refactoring Test Code”, Addison Wesley, 2007, ISBN 0-13-149505-4.
6. IEEE, “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules” IEEE Std. No.: 1516 - 2000.
7. IEEE, “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification.” IEEE Std. No.: 1516.1 - 2000.
8. IEEE, “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template(OMT).” IEEE Standard No.: 1516.2-2000.
9. IEEE, “IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process” IEEE Std. No.: 1516.3 - 2003.
10. 심준용, 진정훈, 김세환, “M&S 프레임워크를 적용한 효율적인 분산객체 통신 모듈 설계”, 한국소프트웨어공학회 학술대회 논문집 제10권 1호, 2008.

심 준 용 (caesar@lignex1.com)

2005 우석대학교 컴퓨터공학과 학사
2007 한양대학교 컴퓨터공학과 석사
2007～현재 LIG넥스원 VM팀 선임연구원

관심분야 : 모델링&시뮬레이션, HLA/RTI, 분산 컴퓨팅

이 용 헌 (yhlee80@lignex1.com)

2007 포항공과대학교 학사
2007～현재 LIG넥스원 VM팀 선임연구원

관심분야 : HLA/RTI, High Performance Computing, 병렬 프로그래밍

이 승 영 (seungyoung.lee@lignex1.com)

1999 인하대학교 전자공학과 학사
2002 인하대학교 컴퓨터공학과 석사
2002～현재 LIG넥스원 VM팀 수석연구원
2009～현재 국방과학기술조사 Simulation분야 자문위원
국방과학기술수준조사 기술전문가

관심분야 : 모델링&시뮬레이션, SBA, LVC

김 세 환 (saehwan.kim@lignex1.com)

1985 경북대학교 전자공학과 학사
1987 경북대학교 전자공학과 석사
2005～현재 LIG넥스원 VM팀 수석연구원(팀장)
2006～현재 국방과학기술조사 M&S자문위원
국방과학기술수준조사 기술전문가

관심분야 : 모델링&시뮬레이션, SBA, LVC