# 모바일 서비스 회귀성 시험 환경 구축을 위한 네트워크 시뮬레이터

김주형*, 김재훈**, 김응구***
*아주대학교 지식정보보안학과
**아주대학교 정보통신전문대학원
***(주)이루온 QA팀
e-mail:noel082@ajou.ac.kr

# Network Simulator for Regression Testing Environment on Mobile Services

Ju-Hyung Kim*, Jai-Hoon Kim**, Oung-Gu Kim***
*Department of Knowledge Information Security, Ajou University
**Graduate School of Information & Communication, Ajou University
*** QA, Eluon, Inc

요 약

여러 장치들과 연동하여서 동작하는 네트워크 시스템 개발 시, 회귀테스트는 전체 시스템의 안정성을 보장하기 위한 가장 필요한 테스트이다. 그러나 서비스가 진행 중이거나 연동을 위한 추가적인 장비가 필요할 경우 테스트의 제약을 받게 된다. 본 논문에서는 회귀성 테스트에서 발생하는 시간과 비용을 줄이기 위한 가상 시스템에서 네트워크 시뮬레이터를 제안한다. 네트워크 시뮬레이터는 테스트에 대한 시나리오를 분석하여 테스트에 따른 실제 장비에서의 메시지들을 구성하며, 테스트 시나리오에 맞게 이벤트를 발생시킴으로서 가상으로 회귀성 테스트를 가능하게 한다. 설계된 네트워크 시뮬레이터는 우선적으로 모바일 환경에서 테스트를 시행하여, 가상 이벤트 구성과 동작의 기능을 검증하는데 사용되었다.

## 1. 서론

현대에 개발되는 많은 네트워크 장비들은 여러 가지 종류의 네트워크 장비들과 연동하여 동작하는 것이 일반적이다. 이처럼 연동하여 동작하는 장비들을 테스트하기 위해서는 많은 장비들이 필요하게 되고, 이러한 연동 테스트 환경을 구축하는 것은 많은 비용을 소비하게 된다. 이러한 원인으로 장비의 충분한 테스트가 이루어지지 못하면 실제 시스템 운용 시, 이상 동작으로 인한 유지보수비용이 많이 들어가게 되는 등, 소프트웨어 품질이 나빠지게 된다. 그러나 아직까지도 단순한 하나의 장비 기능에 대한 테스트 프로그램들 만이 개발되어 왔고, 연동 테스트를 위한 시뮬레이터를 개발함에 있어서도, 연동 장비에 대한 모든 기능들을 구현하여 하나의 시스템을 만들어 테스트를 하려는 시도만이 이루어져 왔다. 연동장비의 모든 기능을 수행하는 시뮬레이터 개발은 몇 가지 문제점이 존재하는데, 시뮬레이터를 만드는 비용이 많이 들게 되고, 다른 장비에 대한 시뮬레이터를 만들기 위해서는 또 같은 비용이 들어가게 되어 실제 장비보다 테스트장비 개발의 비용이 더 많이 들어가게 되는 상황에 이른다는 것이다. 따라서 이러한 문제점을 해결 할 수 있는 단순하면서도 효율적인 네트워크 시뮬레이터 개발이 필요하다.

**• 장비에 독립적인 회귀성 네트워크 시뮬레이터:** 연동 장비의 특정한 메시지들을 만들어 두고 사용하여 복잡한 시스템을 모두 구현 하지 않고 테스트를 시행 할 수 있도록 한다. 따라서 연동 장비의 사용되는 메시지만 분석되면 해당 장비의 기능을 하는 가상 장비를 시뮬레이터로 개발할 수 있다.

**• 프로토콜에 독립적인 회귀성 네트워크 시뮬레이터:** 네트워크에서 동작하는 장비들은 특성에 맞는 프로토콜을 사용하여 동작하게 된다. 이러한 여러 프로토콜들을 지원하기 위해 프로토콜과 관련된 부분은 독립적으로 구성하고 구현하여, 필요한 경우 사용 할 수 있도록 해야 한다.

## 2. 관련연구

### 2. 1 회귀테스트[1]

회귀 버그를 찾는 모든 소프트웨어 테스트 방식은 회귀 테스트라 할 수 있다. 회귀 버그는 이전에 제대로 작동하던 소프트웨어 기능에 문제가 생기는 것을 가리킨다. 일반적으로 회귀 버그는 프로그램 변경 중 뜻하지 않게 발생한다. 회귀 테스트로는 이전의 실행 테스트를 재실행하며 이전에 고쳐졌던 오류가 재현되는지 검사하는 방법이 많이 사용된다.

경험적으로 이러한 오류 재현 방법은 꽤 효과적이다. 부실한 버전 관리로 인해 이전의 버그 픽스를 유실하고

이로 인해 버그가 재발하는 경우가 종종 있으며, 또한 그저 자주 픽스하는 방법은 프로그램을 지저분하게 하는 임시방편일 뿐 근본적인 해결은 되지 못하기 때문이다. 임시적인 픽스들은 프로그램의 다른 부분을 변경할 경우 무용지물이 되는 경우가 많다. 즉, 리팩토링을 통해서 몇몇 기능을 재디자인 할 경우 동일한 문제가 이전에 고쳐진 부분에서 다시 발생하는 경우가 많다.

## 3. 본 론

### 3. 1 요구사항

여러 장비와의 연동이 필요한 테스트에서 각각의 장비들은 독립적으로 존재하게 되며, 특정 인터페이스를 통하여 데이터를 주고받게 된다. 이를 위해 각각의 장비는 다른 프로세스로 동작해야 하며 서로 영향을 주어서는 안된다. 또한 이러한 프로세스들을 관리하기 위한 모니터링 프로세스도 존재하여야 한다. 연동장비가 사용하는 프로토콜을 위해 별도의 프로토콜 스택이 존재해야 하며 이는 선택 가능해야 한다. 연동장비의 메시지 흐름을 관리하기 위해 FSM(Finite Satae Machine)이 존재하여야 하며, 이상 메시지에 실제 장비가 반응하지 않는 것처럼 메시지의 비교에 대한 메커니즘이 존재하여야 한다.

### 3. 2 설계 및 구현

#### 3. 2. 1 설계

코드 종속성을 최소화하기 위해 외부 인터페이스를 담당하는 부분과 내부 로직을 수행하는 부분을 구분하였다. UI와의 공유 DB를 두어 데이터 전달에서 발생하는 트래픽을 최소화하였고, 내부에서 전달되는 데이터는 모두 XML로 구성하여, 이기종간의 동기화와 데이터의 검증을 편하게 할 수 있도록 하였다.

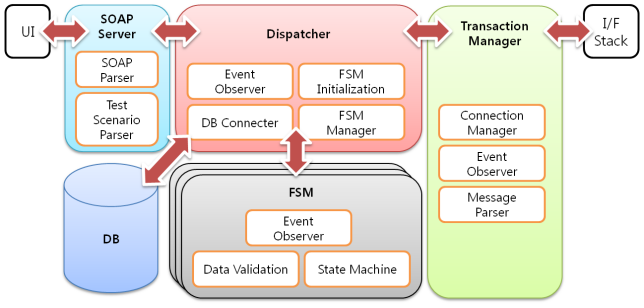


그림 1. 네트워크 시뮬레이터 구조도

• **SOAP Server**는 UI와의 인터페이스를 담당한다. SOAP Parser를 통해 UI에서 엔진에 실행하기를 원하는 테스트 시나리오 정보들을 생성한다. Test Scenario Parser는 Parsing된 SOAP 메시지 정보를 활용하여, 공유 DB로부터 정보를 얻어오고, Test Scenario를 재구성 하여 Dispatcher로 전송 하고, 결과를 받아 UI로 알려준다.

• **Dispatcher**는 SOAP Server를 통해서 동적으로 생성되며, 실행할 시나리오가 존재 하는 경우에 동작하게 된다. Event Observer를 통해 양쪽으로 연결되어 있는 SOAP Server와 Transaction Manager의 이벤트를 감시하여 발생하는 이벤트를 올바르게 처리한다. SOAP Server에서 전달된 이벤트에 대해서는 FSM Initialization을 통해 가상장비에 해당하는 FSM을 Setting 하고, 실행시킨다. 이렇게 실행시킨 FSM들의 관리하기 위해 FSM Manager를 두었고 이는 FSM의 오류와 종료를 확인한다. Transaction Manager로부터 전달된 이벤트에 대해서는 FSM Setting시에 별로로 생성된 ID값을 확인하여, 올바른 FSM으로 메시지를 전달한다. 또한 UI에서 해당 시나리오의 절차 및 결과를 확인할 수 있도록 DB Connection을 통해 Test Scenario의 결과를 저장한다.

• FSM(Finite State Machine)은 각각의 가상장비를 가리키는 것으로 각 가상장비가 상태를 기반으로 동작하므로 이렇게 명명하였다. 해당 모듈은 Dispatcher에 의해서 초기화 및 생성되며, 테스트 시나리오에 따라 개수와 상태가 동적으로 결정된다. Event Observer를 통해 발생되는 이벤트를 감시하고, 이를 State Machine으로 전달하여 특정 이벤트 발생에 따른 상태 변경 및 상태 변경에 따른 동작을 수행한다. 이 때 메시지 비교 이벤트가 발생하게 되면 Data Validation로 메시지를 전달하여, 메시지 비교로직을 수행한다. 여기에서 메시지 비교를 위한 몇몇의 API를 제공하며, 이를 사용한 UI쪽의 스크립트를 실행하여 비교를 실행하고, 결과를 알려주게 된다.

• Transaction Manager는 연결된 프로토콜 스택과의 인터페이스 역할을 한다. 여러 프로토콜 스택을 연결하여 선택, 사용할 수 있도록 Connection Manager을 통하여 프로토콜스택의 연결을 관리한다. 각각의 프로토콜 스택은 하나씩 연결될 수 있다. Event Observer에서 특정 프로토콜 스택에서 전달된 이벤트와 Dispatcher에서 전달된 이벤트를 확인하여, Message Parser를 통해 XML메시지를 내부 구조체로 변경하거나, 동적으로 XML을 구성하여서 전달한다.

#### 3. 2. 2 구현 및 구현에서의 이슈

##### 3. 2. 2. 1 구현

구현에 있어서의 언어는 Python[2]을 사용하였다. Python은 스크립트 언어로서 여러 가지 유용한 모듈들을 제공하며, 여기에서는 SOAP Server 모듈과 XML 파싱 모듈을 사용하여, 코드작성의 시간을 줄였다. 특히 스트링을 비교하는 모듈을 이용하면 테스트 메시지들의 비교를 쉽게 할 수 있다. Python의 버전은 2.7을 사용하였다.

모듈은 크게 SOAP Server와 Dispatcher, FSM, Transaction Manager, 로 나누어 구현하였으며, 쓰레드로 생성 및 동작하도록 하였다. 각 모듈은 큐를 이용하여 통신하며, 큐에 들어온 데이터의 헤더를 확인하여 이벤트 종류를 확인 할 수 있게 하였다.

• **SOAP Server**는 Zolera SOAP Infrastructure(ZSI)[3]를 이용하여 구현하였고, WSDL[6]을 사용하여, UI에게 S

OAP 메시지의 구조를 알려주도록 하였다. 또한 Test Scenario Parser에서 SOAP 메시지의 내용을 토대로 DB에서 데이터를 받아오는 등의 작업을 하여 온전한 하나의 시나리오로 만들게 된다. 이렇게 생성된 메시지는 Dispatcher 모듈로 큐를 통해 전달하고, 결과를 기다리게 된다.

• **Dispatcher**는 SOAP서버, FSM, Transaction Manager로 부터의 이벤트에 대해 처리 하며, 각각의 이벤트에 대해 SOAP Server 이벤트는 전달된 시나리오를 이용하여 FSM을 생성하고 실행하며, FSM 이벤트는 전송하려는 실제 메시지에 FSM을 식별할 수 있는 별도의 ID를 부여하여 Transaction Manager로 전달한다. Transaction Manager 이벤트는 해당 메시지의 ID를 확인하여 올바른 FSM으로 전송한다. 또한 FSM과 Transaction Manager로 부터의 이벤트가 발생하거나 일정시간 반응이 없으면, FSM의 상태를 확인하여, 시나리오의 상태를 확인하는 모니터링 기능도 수행하도록 하였다.

• **FSM**은 테스트 시나리오에 맞추어 Idle, Send, Receive, Error 상태를 가지며, 시나리오에 맞게 상태를 변경한다. Send 상태에서는 메시지 전송, Receive 상태에서는 메시지를 수신한다. 특히 Receive 상태에서 수신한 메시지는 UI에서 작성된 스크립트를 이용하여, 메시지 비교를 하고, 그 결과를 UI에게 알려준다. 또한 이 스크립트를 간단하게 이용 할 수 있도록 별도의 API를 제공한다.

• **Transaction Manager**는 총 3개의 쓰레드로 구성되며, 프로토콜 스택의 연결을 관리하는 Connection Manager 쓰레드, 메시지 전송 쓰레드, 메시지 수신 쓰레드로 구분된다. Connection Manager 쓰레드는 프로토콜 스택을 포트로 구분하여 연결을 대기하며, 연결된 프로토콜 스택을 활성화 하여 메시지 전송 쓰레드와, 메시지 수신 쓰레드에서 사용 할 수 있도록 한다. 각 프로토콜 스택에 대해서는 하나의 연결만을 허용하다. 메시지 전송 쓰레드는 큐를 통해 이벤트가 발생하면 메시지의 내용을 XML로 만들어서 프로토콜 스택으로 전송한다. 메시지 수신 쓰레드는 수신된 메시지를 Dispatcher에서 사용하는 구조체로 변경하여 Dispatcher로 전달한다. 연결 종료 메시지가 오면 해당 프로토콜 스택을 비활성화 하고, Connection Manager에게 알려 연결을 다시 맺을 수 있도록 한다.

#### 3. 2. 2. 2 구현에서의 이슈

회귀성 테스트 시뮬레이터의 경우 내부적으로 여러 개의 장비가 동작하고, 외부로의 인터페이스는 하나이므로 외부에서 들어오는 메시지에 대해서 어느 장비로 전달될 메시지인지 확인하는 별도의 과정이 필요하게 된다.

그림 2. 에서 볼 수 있듯이 여기에서는 프로토콜 스택과 교환하는 XML 데이터에 ID필드와 OPNAME 필드를 추가함으로서 오가는 메시지에 대한 장비 식별을 할 수 있도록 하였다. 하지만 여기에서 OPNAME은 프로토콜 스택에 종속적인 부분으로 프로토콜 스택에 따라 중복 될 수 있어 동시에 여러 프로토콜이 스택이 동작할 경우 ID 값으로 적합하지 않게 되는 문제점이 있다. 이를 위해 초기 프로토콜 스택과의 Session 연결 시에 실행될 Test Scenario의 정보를 알려 주어 프로토콜 스택에서 이 정보를 토대로 ID를 생성 할 수 있도록 하거나, 별도의 ID로 사용 할 수 있는 Unique한 값을 찾아야 한다. 이 부분에 대해서는 연구를 통해 차후에 구현 예정이다.

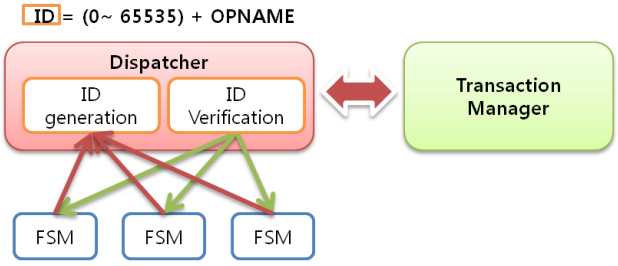


그림 3. ID값을 이용한 가상 장비 식별

### 3. 3 시험 및 결과

#### 3. 3. 1 시험 환경

시험 환경은 2대의 컴퓨터에서 행해졌다. 그림4는 모바일 서비스 중 하나인 HLR장비의 메시지 전달을 시험하기 위한 환경으로 각각의 컴퓨터에 시스템은 똑같이 구성 된다. 한쪽에서는 HLR장비의 응답 메시지들을 시나리오로 한쪽에서는 실제 테스트 시나리오를 입력하고, 메시지가 올바르게 오가는지 확인한다. 이기종간의 올바른 동작을 확인하기 위하여 각각의 컴퓨터 시스템을 Linux와 Unix로 다르게 구성하였다.

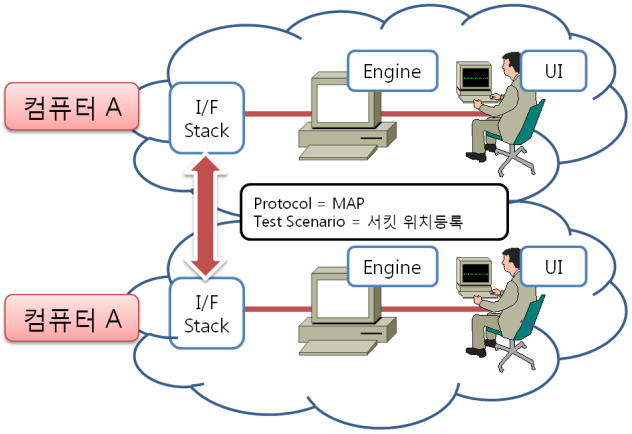


그림 4. 데모 환경 도식

#### 3. 3. 2 시험 결과

그림 6은 테스트에 사용된 시나리오의 흐름이다. 여기에서 교환기A와 교환기B는 그림 4에서 컴퓨터A의 가상장비로 동작하며 HLR은 그림 4에서 컴퓨터B의 가상장비로 동작하여 해당 메시지들을 전송 할 수 있게 만들어 진다. 하나의 데이터의 흐름을 모니터링 하기 위한 별도의 GUI툴은 존재하지 않고, 실행에 따른 로그파일을 통해 올바르게 시나리오가 실행되었다는 것을 확인할 수 있으며,

UI로 전송되는 로그를 통해 시나리오의 흐름을 확인할 수 있었다.

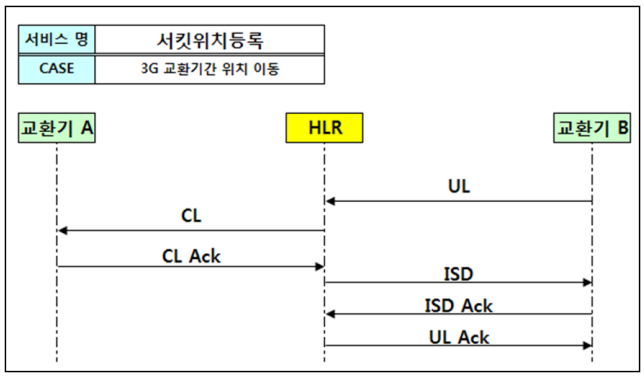


그림 5. 테스트 시나리오

연동 테스트를 할 경우 장비들의 메시지흐름을 알 수 있다면, 실제 장비가 아닌 가상 장비를 통해서도 충분히 회귀성으로 테스트를 할 수 있는 환경을 구축할 수 있다는 것을 확인할 수 있다.

## 5. 결론

장비 개발에 있어서 테스트를 통한 품질 검증은 매우 중요한 일임에 틀림없다. 충분한 품질 테스트를 통해서만이 양질의 장비가 개발된다. 우리나라에서는 아직까지 이러한 사실을 잘 알고 있지만 실제로 충분한 테스트를 시행하지는 못하고 있다. 가장 큰 이유로는 역시 비용이 많이 들기 때문이라고 할 수 있다. 연동 장비를 테스트하기 위한 환경은 쉽게 구축할 수 없고 구축 하더라도 많은 비용과 시간을 투자해야 하기 때문이다. 이러한 비용 문제를 해결하기 위해서 본 논문에서는 회귀성 테스트를 위한 네트워크 시뮬레이터를 제안했다.

웹 환경에서의 쉬운 접근을 위해 네트워크 시뮬레이터는 SOAP을 사용하여 동작시킬 수 있도록 하였고, 프로토콜에 독립적으로 동작할 수 있도록 프로토콜 스택을 분리시켰다. 그리고 XML을 사용하고, 내부적인 비교 API를 제공하여, 사용자가 동적으로 비교 스크립트를 작성할 수 있도록 함으로서 테스트 메시지의 단순한 비교에서 벗어나 테스트 메시지의 자세한 부분까지 확인할 수 있도록 하였다. 향후 회귀성 테스트에 따른 기능 측정뿐만이 아니라 성능까지도 측정 가능 하도록 하는 네트워크 시뮬레이터를 연구할 것이다.

## 참고문헌

[1] Wikipedia, 회귀테스트, http://ko.wikipedia.org/wiki/%ED%9A%8C%EA%B7%80_%ED%85%8C%EC%8A%A4%ED%8A%B8/

[2] Python v2.7 documentation, http://docs.python.org/

[3] ZSI : The Zolera Soap Infrastructure, http://pywebsvcs.sourceforge.net/zsi.html

[4] ibm, developerWorks, http://www.ibm.com/developerworks/

[5] Python 4Suite-XML, http://pypi.python.org/pypi/4Suite-XML

[6] Web Services Description Language(WSDL), http://www.w3.org/TR/wsdl

[7] 3GPP TS 23.002 Network architecture, http://www.3gpp.org/ftp/Specs/html-info/23002.htm