모바일 표준 플랫폼(WIPI) 검증 도구 설계 및 개발

이 상 윤[†] · 이 환 구[††] · 최 병 욱[†††]

요 약

WIPI는 한국무선인터넷표준화 포럼에서 개발하였으며 한국정보통신기술협회가 채택한 무선인터넷 표준 플랫폼이다. 상호 호환성을 보장받기 위해서는 개발한 WIPI 플랫폼이 표준을 준수하고 있는지 검증하는 절차가 필요하다. WIPI는 하드웨어 추상화 계층인 HAL, 실행 엔진, 그리고 API(WIPI-C, WIPI Java)로 나뉘어져 있다. WIPI 응용 프로그램은 WIPI API를 이용해 작성되며 WIPI API는 자체적으로 수행이 완료될 수도 있고 이벤트나 사용자 입력을 처리하기 위해 실행 엔진 또는 HAL 계층으로부터 필요한 기능을 제공받기도 한다. 따라서, 응용 프로그램에서 오류가 발생했을 때 어느 계층에서 문제점이 발생했는지 검증하는 것이 필요하다. 본 논문에서 WIPI 규격에 대한 검증 도구로서 플랫폼의 전체 기능과 API에 대한 검증 도구인 PCT와 하드웨어 추상화 계층인 HAL의 API 검증 도구인 HCT를 제안한다. PCT는 최종적으로 플랫폼에 대한 검증 결과만을 보고하기 때문에 검증에 실패하면 어느 계층에서 문제점이 발생했는지 알 수 가 없다. 따라서 문제점이 발생한 지점을 정확히 진단하고 정정하기 위해서는 전체 플랫폼에 대한 인증과는 별도로 HAL에 대한 인증이 필요하다.

키워드 : WIPI, HAL, PCT, HCT

The Design and Development of a WIPI Certification Toolkit

Sang-Yun Lee[†] · Hwan-Gu Lee[††] · Byung-Uk Choi[†††]

ABSTRACT

WIPI is developed by KWISF(Korea Wireless Internet Standardization Forum) and a wireless internet standard platform adopted by TTA. It needs the certification process for standard specification in order to confirm interoperability. The WIPI is composed of the HAL, the Runtime Engine, and APIs(WIPI-C, WIPI Java). WIPI applications are implemented through WIPI APIs that can be finished by themselves or provided essential functions from runtime engine or HAL. Therefore it needs to certify where the problems occur when errors occurred in a application.

In this paper we propose the PCT that certifies a WIPI platform's functionality and APIs and the HCT that certifies HAL APIs.

Because the PCT reports the final certification results for the platform it is impossible to know where the problems occur when it fails to certify platform. So, it needs to certify the HAL regardless of platform certification.

Key Words : WIPI, HAL, PCT, HCT

1. 서 론

WIPI(Wireless Internet Platform for Interoperability)는 한국무선인터넷 표준화 포럼(KWISF : Korea Wireless Internet Standardization Forum)에서 만든 모바일 표준 플랫폼 규격[1]으로 휴대폰에 탑재되어 무선 인터넷을 통해 다운로드 된 응용프로그램을 실행하기 위한 환경을 제공하는데 필요한 표준 규격이다. 2002년 WIPI 1.0 이 발표된 이후로 2006년 4월 현재 WIPI 2.1 이 발표되었다. 2005년 4월부터는 한국에서는 출시되는 휴대폰에는 위피 플랫폼을 탑재하도록 의무화되었으며, SK Telecom, KTF, LG Telecom 등 이동통신사를 통해 상용화되었다. WIPI와 비슷한 플랫폼으로는 Qualcom의 Brew와 SUN의 J2ME(MIDP/CLDC)가 있다. Brew는 C 언어를 지원하는 플랫폼으로 KTF를 통해 세계 최초로 상용화 되었으며 J2ME(MIDP/CLDC)는 Java 언어를 지원하는 플랫폼으로 LGT를 통해 세계 최초로 상용화 되었다.

WIPI 규격은 휴대폰 응용 프로그램 개발 언어로 널리 사용되는 C, Java 언어를 표준 규격 언어로 제공하며 WIPI 플랫폼은 이를 동시에 지원하여야 한다. 표준화된 규격을 지원하는 플랫폼을 개발한 후 출시하려면 상호 호환성을 위해 WIPI 규격을 준수하는지 공식적인 인증이 반드시 필요하다. 자바 표준 API를 제정하는 JCP(Java Community Process)에는 TCK(Technology Compatibility Kit)라고 하여 각

† 정 회 원 : 한국전자통신연구원 임베디드S W연구단 선임연구원
†† 정 회 원 : 한국전자통신연구원 임베디드S W연구단 연구원
††† 정 회 원 : 한양대학교 정보통신대학 정보통신학부 교수
논문접수 : 2006년 1월 11일, 심사완료 : 2006년 8월 11일

JSR(Java Specification Request) 의 스펙 주창자가 규격에 대한 검증 도구 제공을 의무화 하고 있다[3].

WIPI는 하드웨어 추상화 계층인 HAL, 실행 엔진, 그리고 API(WIPI-C, WIPI-Java)로 나뉘어져 있다. WIPI 응용 프로그램은 WIPI API를 이용해 작성되며 WIPI API는 자체적으로 수행이 완료될 수도 있고 이벤트나 사용자 입력을 처리하기 위해 실행 엔진 또는 HAL 계층으로부터 필요한 기능을 제공받기도 한다. 따라서, 응용 프로그램에서 오류가 발생했을 때 어느 계층에서 문제점이 발생했는지 검증하는 것이 필요하다.

본 논문에서는 WIPI 규격에 대한 검증 도구로서 WIPI 플랫폼 검증 도구인 PCT(Plaform Certification Toolkit)와 HAL (Hardware Adaptation Layter) API 검증 도구인 HCT(HAL Certification Toolkit)를 제안한다.

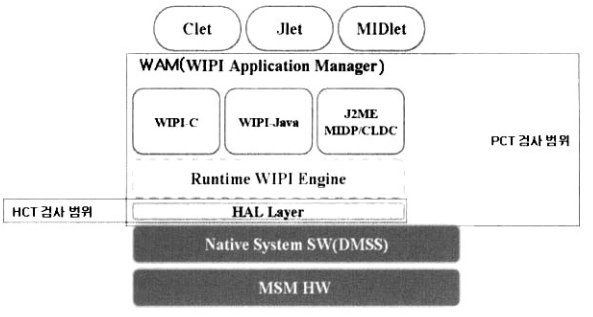
PCT는 구현된 WIPI 플랫폼이 규격에서 기술한 기능과 API를 제대로 지원하는지 검증하며, PCT 테스트를 통과한 WIPI 플랫폼은 상호 호환성을 확보한 것이다. 그런데, PCT 테스트 시 오류가 발견되면 그 오류 발생 지점이 응용 프로그램 인터페이스인지 실행 엔진인지 HAL 계층인지 정확히 진단할 수 없는 문제점이 있어 디버깅하는데 많은 어려움이 있다.

WIPI 플랫폼 개발을 하는 업체들의 협력 관계를 보면 HAL을 개발하는 단말 제조사, 실행 엔진을 개발하는 플랫폼 업체, API를 개발하는 솔루션 업체, 응용 프로그램을 개발하는 컨텐츠 제공업자로 나뉘어져 있다. 따라서 어플리케이션을 개발하고 시험하는 과정에서 오류가 발생하면 그 원인을 찾기 위해 각 업체들이 긴밀히 협조해야 하고 경우에 따라서는 상호 보유하고 있는 소스 코드를 공유해야 할 필요가 있다. 하지만, 각 업체마다 소스 코드 공유를 꺼릴 뿐만 아니라 문제의 원인을 상대방으로 떠넘기려는 경향이 강해서 플랫폼 개발 기간이 길어지고 이에 따라 적시에 플랫폼을 출시하지 못하는 문제점이 있다. 본 논문에서는 이와 같은 문제점들을 해결하기 위해 HAL API 를 검증하는 HCT를 개발하고 제안한다.

우리가 개발한 PCT는 TTA(한국정보통신기술협회)로부터 소프트웨어 품질 인증을 획득하였으며 SK텔레콤, KTF, LG 텔레콤 등 이동통신 사업자들이 이를 공동으로 채택하여 각 사에서 개발한 플랫폼을 검증하는데 공식적인 도구로 사용하고 있다. 우리는 HCT 또한 공식적인 인증 도구로 발전하기 위해 지속적인 노력을 할 것이다. 본 논문은 다음과 같이 구성되어 있다. 2장에서는 WIPI 플랫폼의 구조에 대해서 설명한다. 3장에서는 본 논문에서 제안하는 PCT 의 아키텍처와 테스트 케이스, 인증 절차에 대해서 기술한다. 4장에서는 본 논문에서 제안하는 HCT의 아키텍처와 동작 모습을 보여준다. 5장에서는 본 연구에서 제안하는 검증 도구를 실제 플랫폼에 적용한 사례를 설명한다. 그리고 6장에서 본 논문을 요약하고 결론을 맺는다.

2. WIPI 플랫폼 소개

WIPI는 CPU, LCD, 메모리 등 단말기 하드웨어나 단말기



(그림 1) WIPI 시스템 구조

가 사용하는 운영 체제에 관계없이 실행과 이식이 용이하도록 구조화되었고 동일한 어플리케이션이 이동통신사업자 혹은 단말기에 상관없이 실행이 될 수 있도록 표준화한 모바일 플랫폼이다.

WIPI 플랫폼의 전체적인 시스템 구조는 (그림 1)과 같다.

(그림 1)에서 하단에 있는 단말기 기본 소프트웨어는 CDMA망에서는 퀄컴에서 제공하는 DMSS(Dual-Mode Subscriber Station) 소프트웨어를 지칭하는 것으로 REX라는 간단한 단말기 운영체제 기능과 통신 기능 및 각종 디바이스 드라이버가 포함된다. HAL은 단말기 제조사를 위한 API를 정의한 것으로 단말기 제조사마다 서로 다른 기기들을 지원하기 위해 HAL이라고 하는 추상화 계층을 도입한 것으로 Brew나 J2ME 에는 존재하지 않으며 WIPI의 아키텍처 중 획기적인 특징으로 받아들여지고 있다.

실행엔진과 WIPI-C API, WIPI-Java API 등은 포팅되는 대상에 상관 없이 동일한 코드로 개발될 수 있고 HAL은 탑재되는 대상에 따라 다르게 포팅되어야 한다. HAL이 휴대폰에 포팅되면 핸드폰용 WIPI 플랫폼이 되고 데스크탑 윈도우즈 환경에 포팅되면 WIPI 에뮬레이터가 된다[4, 5]. 실행 엔진은 다운로드 받은 WIPI 응용 프로그램을 실행시키는 미들웨어로서 링커 및 로더 기능, 메모리 관리, 리소스 관리, 가비지 컬렉션 기능 등을 수행한다. WAM(WIPI Application Manager)은 서버로부터 프로그램을 다운로드하고 저장하는 기능, 검색 등 위피 프로그램을 관리한다[6, 7]. WIPI-C API와 WIPI-Java API는 WIPI 응용 프로그램 개발자를 위한 C 및 Java API를 말하는 것으로 WIPI-C 로 개발한 응용 프로그램을 Clet, WIPI-Java로 개발한 응용 프로그램을 Jlet이라고 부른다. 그리고 WIPI 2.0 부터는 J2ME(CLDC1.1/MIDP2.0)가 필수 규격으로 포함되었다.

3. PCT

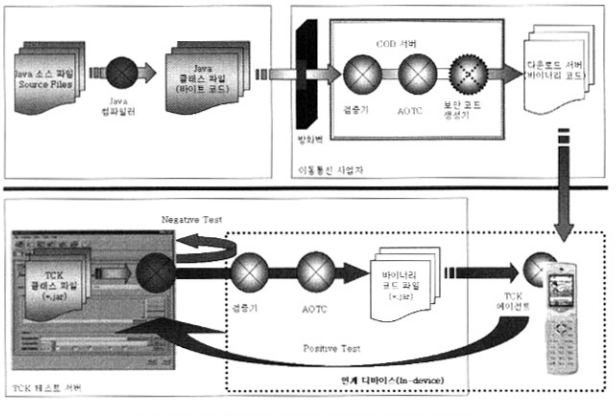
PCT는 개발된 WIPI 플랫폼이 WIPI의 표준 규격을 준수하는지 검증하는 도구이다. 검증은 표준 규격에서 정의한 API의 구현 여부와, 규격에서 정의한 기능이 정상적으로 동작하는지 확인하는 것이다. 본 논문에서 제안하는 PCT는 이 검증 외에 플랫폼 자체의 성능 벤치마크를 위한 성능 테스트 및 안정성 테스트를 포함한다[2]. WIPI는 초기 개발 당시

REX OS가 탑재된 휴대폰에서 구동되는 것을 목표로 삼았으나 현재는 리눅스, 윈도우 등 다양한 운영체제에 탑재되고 있다. 따라서 우리는 다양한 운영 체제를 지원할 수 있도록 PCT를 설계하고 개발하였다.

3.1 J2ME 인증 방안

WIPI2.0부터는 J2ME(MIDP2.0/CLDC1.1) 이 필수 규격으로 포함되었다. 따라서, WIPI 2.0에서 WIPI-C API와 WIPI-Java API는 PCT의 인증을 받아야 하고, J2ME (MIDP2.0/CLDC1.1) 은 TCK의 인증을 받도록 이원화되었다. 그런데, TCK는 자바 바이트 코드를 대상으로 검증을 수행하고 WIPI 플랫폼은 J2ME(MIDP2.0/CLDC1.1)을 바이너리 코드로 실행하기 때문에 TCK를 통과하기 위해서는 TCK의 바이트코드에 맞춘 검증 시나리오가 필요하다. WIPI는 바이트코드 변환기인 AOTC(Ahead Of Time Compiler)와 이를 수행하는 WIPI 단말기를 확장된 연계 디바이스(in-device)라는 개념으로 묶어서 이를 명쾌하게 해결하였다.

(그림 2)는 WIPI 2.0에서의 인증 절차를 나타내고 있다. (그림 2)에서 볼 수 있듯이 컨테츠 제공자는 개발 도구(SDK)를 통해 자바 소스 파일을 컴파일하여 클래스 파일을 이동통신사업자에게 전달한다. 이동통신사업자는 COD (Compile On Demand) 서버를 통해 클래스 파일을 검증하고 AOTC를 통해 바이너리 코드로 변환한다. 그리고 이 바이너리 코드는 단말기에 전송되어 실행된다. 이 바이너리 코드를 검증하기 위해 TCK 에이전트가 단말기에서 기동되고 TCK 서버로부터 테스트 클래스 파일을 다운로드 받아 실행한다. 이 때 TCK 테스트 서버에서 제공하는 클래스 파일은 WIPI 단말기에서 작동되기 위해서 바이너리 코드로 변환되어야 하는데 이 때 AOTC 를 거친다. AOTC와 TCK 에이전트가 연계 디바이스에서 작동되는 한 컴포넌트로 간주되기 때문에 TCT 서버는 전달한 테스트용 클래스 파일이 바이너리로 변환되는지 알지도 못한채 그 에이전트로부터의 결과를 받아서 이를 비교해 검증하기 때문에 TCK의 인증 방안을 그대로 따를 수 있다.



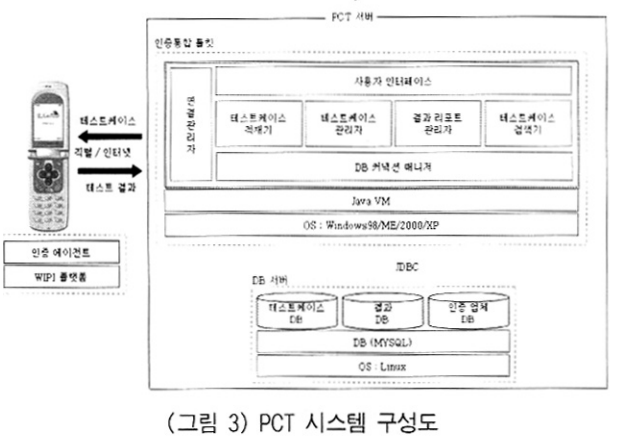
(그림 2) WIPI 2.0의 TCK 인증 절차

3.2 PCT 시스템 구성도

PCT는 휴대폰에 탑재되어 WIPI 플랫폼 상에서 구동되는 인증 에이전트와 PC에 탑재되어 인증 통합 환경을 제공하는 인증 통합 툴킷, 그리고 인증에 필요한 다양한 데이터 및 인증 결과를 저장할 수 있는 인증 DB서버로 구성되어 있다. 인증 통합 툴킷과 DB 서버를 합쳐 PCT 서버가 된다. (그림 3)은 이와 같은 PCT 시스템의 구성도를 나타낸다.

휴대 단말기는 메모리나 CPU 성능 등 하드웨어 사양이 열악하기 때문에 테스트에 필요한 많은 기능들을 전부 탑재시킬 수는 없다. 본 논문에서는 인증 에이전트를 도입해 이를 해결하고자 한다. 인증 에이전트는 휴대폰의 WIPI 플랫폼에 탑재되어 동작하는 프로그램이다. 인증 에이전트는 PCT 서버로부터 테스트케이스를 전송받아 WIPI 플랫폼 상에서 구동한 후 테스트케이스 수행의 결과를 통합 툴킷으로 전송한다.

우리가 개발한 인증 에이전트는 WIPI 어플리케이션의 하나인 Jlet으로 구현하였으며 플랫폼 기능 중의 하나인 애플리케이션이 다른 애플리케이션을 호출할 수 있는 기능과 애플리케이션간의 통신 기능을 이용하여 구현하였다. 서버와 통신할 때는 시리얼 케이블 혹은 TCP/IP를 이용한 네트워크 통신을 이용한다. 본 논문에서는 휴대폰용, Windows 에뮬레이터용, Qplus(한국전자통신연구원에서 개발한 실시간 임베디드 리눅스) 용의 인증 에이전트를 각기 개발하였다.

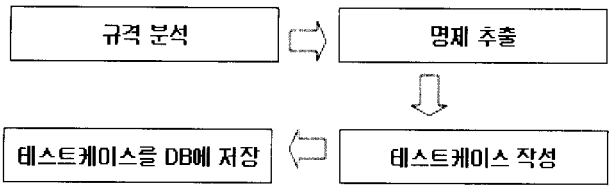


(그림 3) PCT 시스템 구성도

3.3 테스트 케이스

테스트케이스는 플랫폼 인증 요소 중 가장 중요한 요소이다. 인증을 위한 테스트는 블랙박스 테스트 형태를 가지게 되므로 테스트케이스는 미리 결과 값을 알고 있는 테스트를 위한 단위 프로그램으로써 (그림 4)와 같은 절차를 거쳐 생성되고 인증 DB에 저장된다. 저장된 테스트케이스는 인증자의 요구에 따라 폰에 탑재되고 구동된 후 인증 결과를 생성한다.

명제 추출단계에서는 각 API와 기능을 규정한 규격을 바탕으로 그 규격이 준수해야 하는 명제를 정형적인 방법으로



(그림 4) 테스트 케이스 추출 절차

추출한다. 정형적인 방법이란, 테스트케이스 프로그램에서 테스트케이스를 블랙박스로 생각할 때 구체적인 테스트케이스의 입력 매개변수와 출력 매개변수를 규정짓는 것이다. 이러한 명제가 추출되면 이를 바탕으로 테스트케이스를 작성한다. 테스트케이스들이 작성되면 관련 있는 테스트케이스들은 패키지화되고 데이터베이스에 저장된다.

3.4 PCT의 테스트 기능

PCT는 다음과 같이 4가지의 테스트 기능을 제공한다.

첫 번째, WIPI 플랫폼에서 제공하는 API가 규격에서 정의한 사항을 준수하는지 검증하는 규격성 테스트를 제공한다. 예를 들어 Java API중 ListComponent의 int getSelected Index(int index)라는 함수에 대해 ListComponent가 생성된 후 getSelectedIndex 함수의 반환값이 1인지를 확인하는 테스트케이스를 작성한 뒤 휴대폰에서 구동하여 그 결과 값을 비교한다. 검증 판단 방식에는 결과 값을 미리 예상한 값과 비교하여 그 결과의 옳고 그름을 자동으로 판단하는 일괄 테스트와 사람이 개입하여 화면에 출력된 결과값을 확인한 후 사람이 직접 테스트 결과를 검증하는 상호작용 테스트가 있다.

두번째 플랫폼이 기능적으로 지원해야 하는 규격 준수 여부를 검증하는 기능성 테스트를 제공한다. 기능성 테스트에는 API 추가 및 갱신 기능, 응용 프로그램간 통신 지원 기능, 보안관련 기능 등이 있다.

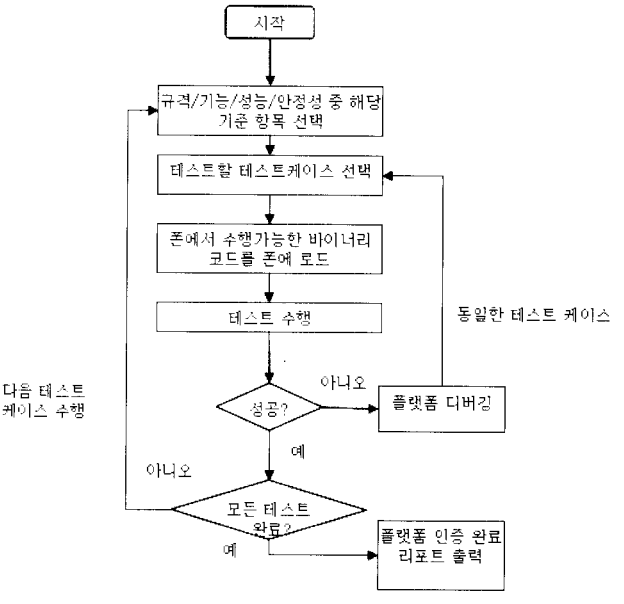
세번째 모바일 디바이스에 탑재된 플랫폼의 성능을 측정하는 효율성 테스트를 제공한다. 효율성 테스트에는 산술 연산, 비트 연산, 파일시스템 접근, 데이터베이스 접근, 배열 연산, 함수 호출과 같은 기본적인 항목과 화면 출력, 임의의 크기의 원 그리기, 폰트 출력 등 사용자 인터페이스 항목, 수학 연산, 메모리 할당 등의 고급 항목이 있다.

네번째 모바일 플랫폼의 안정성과 밀접한 관련이 있는 쓰레드, 네트워크, 응용 어플리케이션 간의 통신 및 성능테스트에 관련된 항목들을 무작위로 조합하여 규격에서 정한 시간 동안 휴대폰 상에서 연속적으로 구동시킨 후 휴대폰에 탑재된 모바일 플랫폼이 안정적으로 작동하는지의 여부를 검증하는 안정성 테스트를 제공한다.

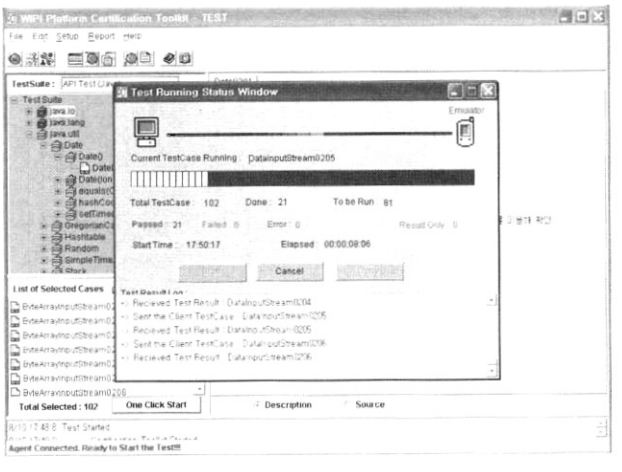
3.5 인증 절차

PCT는 앞에서 제시된 네 가지 인증 기준에 따라 (그림 5)와 같은 인증 절차를 거친다.

인증 툴킷을 이용하여 데이타베이스에 저장된 테스트케이스를 하나씩 휴대폰으로 업로드하여 테스트한 후 모든 테스트케이스의 정확한 동작 여부에 따라 인증 여부를 결정짓는다. 만약 테스트케이스를 구동하여 인증에 실패한 경우 다시 플랫폼을 디버깅한 후 다시 동일한 테스트케이스로 테스트하여 플랫폼의 정확한 동작 여부를 다시 인증하게 된다. 모든 테스트케이스의 테스트가 성공적으로 끝나게 되면 탑재된 플랫폼은 모든 인증 기준을 통과하게 되었다고 말할 수 있으며, 플랫폼은 상호 호환성을 인증 받은 것이다.



(그림 5) 인증 절차

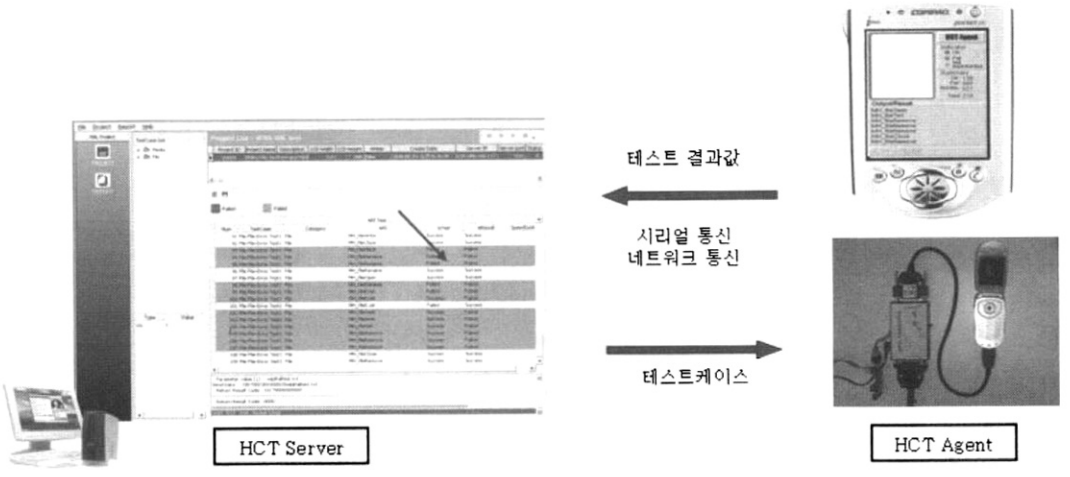


(그림 6) PCT 동작 화면

4. HCT

4.1 HCT 시스템의 구성

HCT 시스템은 (그림 7)에서 볼 수 있듯이 PCT 처럼 테스트케이스와 테스트 프로그램을 관리하는 HCT 서버와 휴대



(그림 7) HCT 서버와 에이전트

단말기에 탑재되어 테스트를 수행해 주는 HCT 에이전트로 구성되어 있다.

HCT는 테스트 대상 API가 PCT와 틀리고, 단말 제조사가 주로 사용할 도구이기 때문에 별도의 프로그램으로 개발하였다.

HCT 에이전트는 이더넷 혹은 직렬 케이블을 통해 서버와 연결할 수 있다. 에이전트가 서버로 인증을 요청하면 서버는 테스트케이스를 선택하여 에이전트로 내려보낸다. 에이전트는 서버로부터 받은 테스트케이스를 실행하고 그 결과를 서버로 전송한다. 이러한 과정은 PCT와 비슷하다.

4.2 HCT 에이전트

(그림 8)은 HCT 에이전트의 구성도이다.

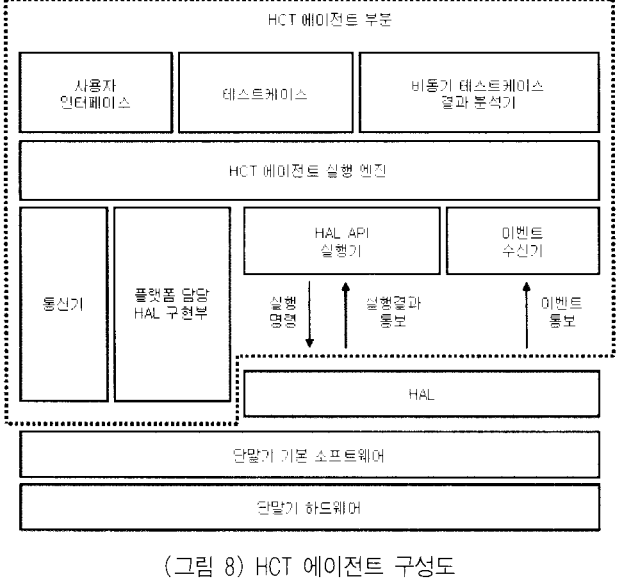
통신기는 HCT 에이전트와 HCT 서버와의 통신을 단말에 맞게끔 시리얼이나 이더넷을 통하여 수행한다. 플랫폼 담당 HAL 구현부는 HAL 규격에서 MH_pltStart()와 MH_pltEvent() 등과 같이 플랫폼에서 제공해야 할 API를 구현한 모듈로써 HCT 에이전트를 실행할 수 있고 단말기 기본 소프트웨어로부터의 이벤트 수신을 처리한다. 사용자 인터페이스는 각 테스트케이스가 진행될 때 마다 사용자에게 결과값에 대한 확인을 요청하는 상호 작용 테스트에 사용된다.

이벤트 수신기는 HAL 계층으로부터 전달되는 이벤트를 처리한다. HCT 에이전트 실행 엔진은 통신기를 통하여 호스트 검증 시스템 등으로부터 테스트케이스를 수신하여 HAL API 실행기를 통하여 테스트를 실행한다. 테스트 실행 결과는 HAL API 실행기나 사용자 인터페이스, 또는 비동기 테스트케이스 결과 분석기를 통해 그 결과를 통보받는다. 테스트 결과는 통신기를 통하여 HCT 서버에 전달한다.

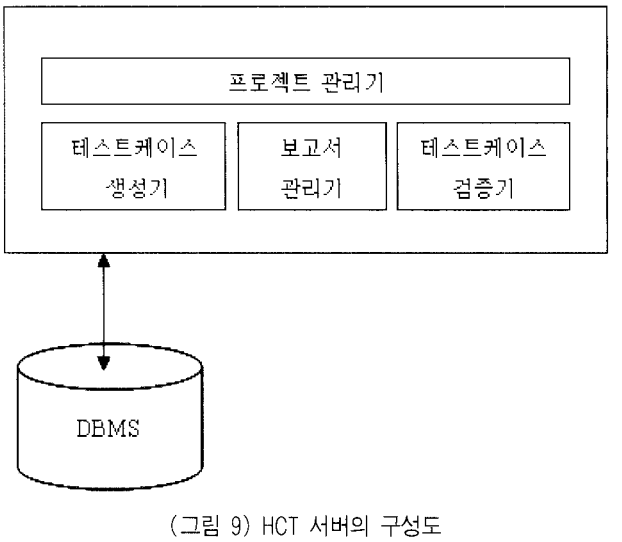
4.3 HCT 서버

(그림 9)는 HCT 서버의 구성도이다.

프로젝트 관리기는 WIPI가 탑재된 단말기의 정보, HAL 검증 진행 여부, 검증 결과 정보 등을 관리한다. 테스트케이스 생성기는 통합 테스트케이스를 등록하는 모듈로서 각 HAL



(그림 8) HCT 에이전트 구성도



(그림 9) HCT 서버의 구성도

API가 통합 테스트케이스로 등록되어 있는 항목을 검증할 단말기의 환경에 맞게 조합하고 파라미터 입력값을 입력하거나 기대값을 입력한다. 보고서 관리기는 테스트 결과를 웹문서로 배포하거나 인쇄하는 기능을 담당한다. 테스트케이스 검증기는 이더넷을 통한 TCP/IP 통신이나 시리얼 케이블을 통한 통신을 담당한다. 그리고 DBMS에는 테스트케이스가 저장된다.

4.4 HCT 테스트케이스

테스트하는 방법에는 저장된 모든 테스트케이스가 자동으로 HCT 에이전트와 연동되어 테스트가 진행되는 일괄 테스트와 각 테스트케이스가 진행될 때 마다 사용자에게 결과값에 대한 확인을 요청하는 상호 작용 테스트가 있다.

각각의 HAL API에 대해서는 다양한 인자값과 그에 따른 예상되는 반환값으로 테스트케이스가 작성된다. 그리고 연관된 테스트케이스끼리 묶어 놓은 통합 테스트 케이스가 있다. 이러한 테스트케이스의 규모는 WIPI 1.2 인 경우 통합테스트케이스가 14개 일반테스트케이스가 109개로 구성되어 있고, WIPI 2.0 인 경우 통합테스트케이스가 28개 일반테스트케이스가 273개로 구성되어 있다.

4.5 HCT 동작

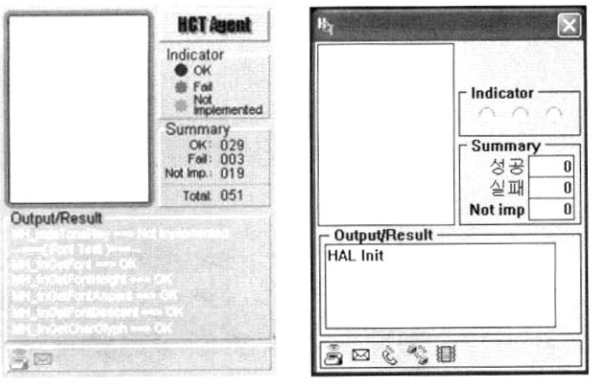
4.5.1 시스템 환경

한국전자통신연구원에서는 위피 실행 엔진으로서 QPlus (한국전자통신연구원에서 개발한 임베디드 리눅스) 기반의 실행 엔진과 윈도우 기반의 에뮬레이터를 개발한 바 있다. 이 엔진들은 WIPI 1.1, WIPI 2.0, WIPI 2.1 모두 지원한다. HAL은 각기 윈도우용과 QPlus용으로 포팅되었다. QPlus 용 HAL은 SMDK 2410 보드 용 커널(커널 버전 : 2.4.18)을 사용하였으며 tinyX와 GTK 1.2 설치가 필요하다. QPlus의 타겟빌더가 제공하는 라이브러리 이외에 audiofile 0.2.5와 esound 0.2.32, event 0.7c 라이브러가 있어야 컴파일이 가능하다. GCC를 이용해 컴파일 하였으며 ARM 계열에서 동작할 수 있도록 크로스 컴파일을 하였다. 윈도우즈용 HAL은 윈도우즈98 이상에서는 모두 사용이 가능하며 ws2_32.lib와 Winmm.lib, vfw32.lib 라이브러리가 추가적으로 필요하다. Visual Studio 6.0을 이용해 컴파일 하였다. HCT 서버는 윈도우98이상에서 동작할 수 있도록 개발되었다. 테스트케이스를 관리하기 위해 데이터베이스로서 Access DB를 사용하였다.

4.2 HCT 시스템 동작

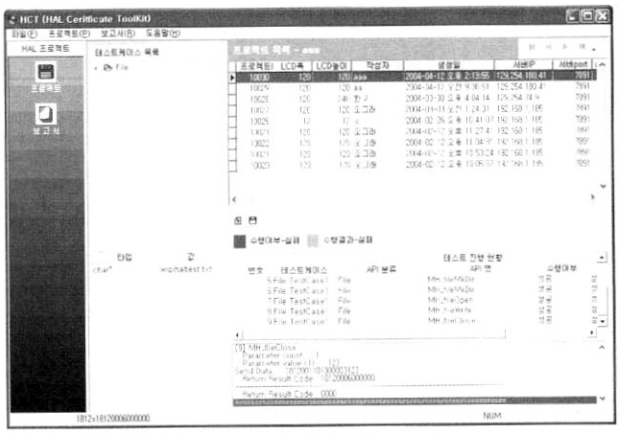
(그림 10)은 본 논문에서 개발한 HCT 에이전트의 실행 모습이다. (그림 10-a)는 QPlus 기반의 타겟 단말기에 동작하는 실제 모습이고, (그림 8-b)는 윈도우 에서 동작하는 실제 모습이다.

HCT 에이전트의 실행 화면에서 'Indicator' 중 'OK'와 'Fail'은 타겟 시스템에 해당 API가 구현되어 있을 때 테스트가 실행되어 그 결과값이 성공 혹은 실패했음을 나타내며, 'Not Implemented'는 타겟 시스템에 해당 API가 구현되어 있지 않아 테스트 실행이 불가능함을 나타낸다. 'Output/Result' 창은 현재 실행되고 있는 테스트케이스에 대한 간략한 상태를 표시하는데 사용된다. (그림 11)은 HCT의 서버 프로그램의 실행 모습이다.



(a) QPlus에서 작하는 에이전트 (b) 윈도우에서 동작하는 에이전트

(그림 10) HCT 에이전트 실행 화면



(그림 11) HCT 서버 실행 모습

프로젝트 기능바는 선택된 테스트 케이스 목록을 해당 프로젝트와 연결시켜 저장하는 저장 메뉴, 선택된 테스트 케이스 목록에 표시된 내용을 화면상에서만 지우는 초기화 메뉴, 해당하는 프로젝트와 관련된 모든 정보를 삭제하는 삭제 메뉴, 프로젝트 목록 이동 메뉴로 구성되어 있다.

프로젝트 목록 리스트 화면은 등록된 모든 프로젝트를 보여준다. 테스트케이스 목록 화면은 각 패키지에 등록된 다양한 테스트케이스를 보여준다.

선택된 테스트케이스 목록 화면은 테스트케이스 목록 화면에서 마우스로 클릭하거나 드래그해서 선택한 테스트케이스 목록을 보여준다. 파라미터 값 화면은 해당 API를 테스트하기 위한 파라미터값을 입력하는 화면이다. 이 값은 내정된 값이 지정되어 있으며 필요에 따라 다른 값으로 변경이 가능하다. 기대값 화면은 해당 API가 처리된 후 예상 되어지는 결과코드나 반환값을 입력 할 수 있는 화면이다.

HCT 서버의 메뉴는 파일, 프로젝트, 보고서, 도움말로 구성되어 있다. 여기서 프로젝트란 인증 대상이 되는 단말기의 등록 정보와 실행할 테스트케이스 정보, 그리고 테스트 결과

를 묶어 놓은 것이다. 파일 메뉴에는 이러한 프로젝트의 생성과 종료 메뉴가 있으며, 프로젝트 메뉴는 프로젝트 설정 변경, 보고서 메뉴에서는 설정된 테스트케이스 실행 결과에 따른 결과 화면 보기, 파일로 저장하기 및 인쇄하기가 있으며, 도움말 메뉴는 이 도구의 사용법에 대해서 기술하고 있다.

5. WIPI 플랫폼 적용 사례

본 논문에서 제안하는 검증 도구를 적용하려면 실제 WIPI 플랫폼이 필요하다. 우리는 WIPI 에뮬레이터를 개발하여 [8-13] 이를 대상으로 검증 도구를 적용하였다. WIPI 에뮬레이터는 WIPI 응용 프로그램을 개발 한 후 폰에 탑재하여 실행하는 것이 시간도 많이 걸리고 디버깅의 어려움이 있기 때문에 이를 해소하기 위해 미리 데스크탑용 컴퓨터에서 실행해 볼 수 있도록 한 것이다. (그림 12)는 우리가 개발한 WIPI 에뮬레이터의 실행 모습이다. Clet으로 구현한 테트리스 게임이 실행되고 있다.

PCT 도구를 이용해 에뮬레이터를 검증하는데 걸리는 시간은 테스트가 일괄 테스트인지 상호 작용 테스트인지에 따라 구별해야 한다. 일괄 테스트인 경우는 컴퓨터가 자동으로 수행하기 때문에 검증자에 상관없이 동일하게 나타나지만 상호 작용 테스트인 경우는 검증자의 숙련도에 따라 달라진다. **MC_fsOpen** 함수는 일괄 테스트로 수행하는데 이 경우 펜티엄-IV, 윈도우즈XP, 1GB 램 환경에서 1초 이내에 끝났다. 다른 함수들도 크게 다르지 않았다. 그러나 검증자의 판단이 개입이 되어야 하는 **MC_grpDrawLine** 함수는 검증자가 화면에 출력되는 모습을 보고 선이 제대로 그려졌는지, 점선 형태로 잘 그려졌는지 눈으로 확인하면서 주관적으로 판단해야 되기 때문에 시간이 많이 걸리고 지루한 작업이다. 이는 HCT도 마찬가지이다. HCT도 각 함수별로 일괄 테스트를 수행한 경우 대부분 1초 이내에서 수행이 완료되었으나 상호 작용 테스트 인 경우는 마찬가지로 애로 사항이 나타났다.

(그림 12) WIPI 에뮬레이터

PCT를 WIPI 플랫폼이 탑재되는 실제 이동통신 단말기에 적용한 사례 보고에 의하면, PCT가 수행되는 시간 보다도 PCT 테스트케이스를 단말기로 전송되는데 걸리는 시간이 훨씬 더 걸렸다. 전송 장비에 따라 시간의 차이가 많이 나기도 하는데, 고가의 장비인 경우 5분 정도 걸리나 일반적인 장비를 사용하는 경우 20분에서 30분 정도가 걸려, 실제로 검증 작업하는데 상당한 애로 사항이 있음이 보고 되고 있다.

또한, 이동통신사마다 정책이 달라서, 전송 장비를 이용한 테스트를 허용하는 회사가 있는 반면, 이를 허용치 않아 CDMA 통신망을 통해 요금을 지불하면서 테스트를 해야 하는 경우도 있어 문제점으로 지적되고 있다.

PCT와 HCT의 구현 과정에서 어려웠던 점은 검증이 실패한 경우, 플랫폼 혹은 HAL 계층이 실제로 문제점이 있었는지와 테스트케이스 자체에 문제점이 있었는지를 밝히는 것이었다. 실제로 이러한 문제점들이 상당히 발견되었다. 오류의 책임 소재를 밝히는 과정에서, 플랫폼 혹은 HAL의 구현이 잘못된 경우도 있었고, 테스트케이스의 오류도 있었다. 또한 정작 심각한 규격의 문제점도 발견되었다. 결과적으로 이러한 책임 소재를 밝히고 해결함으로써 플랫폼과 PCT, HCT를 상호 검증하게 되었고, 규격도 완성도가 높아졌다.

우리가 개발한 PCT 검증도구는 업체에 기술 이전되어 한국정보통신기술 협회로부터 품질 인증을 받았으며, SK Telecom, KTF, LG Telecom 의 공식적인 WIPI 플랫폼 검증 도구로 사용되고 있다. HCT 또한 PCT처럼 공식적인 검증 도구로 인정받도록 지속적인 노력을 할 것이다.

6. 결 론

본 논문에서는 무선인터넷 표준 플랫폼인 WIPI의 플랫폼 검증 도구로서 PCT와 HAL 계층을 검증하기 위한 도구인 HCT를 제안하였다.

PCT는 PCT 서버와 에이전트로 구성되어 있는데, WIPI 플랫폼이 탑재되는 단말기가 하드웨어적으로 열악한 환경임을 고려하여, 테스트케이스를 일반 PC인 서버에 저장하여 에이전트에서 테스트케이스를 다운로드 받아 실행한 후 그 결과를 서버에 보고하는 방식을 취하도록 설계하였다. PCT의 모든 검사 항목을 통과한 플랫폼은 호환성을 확보했다고 인증 받을 수 있다.

HCT는 오류 발생시 오류의 원인이 HAL 계층에서 발생했는지 아니면 다른 곳에서 발생했는지를 알 수 없는 PCT의 한계를 극복하고자 개발되었다. HAL API에 대한 테스트케이스를 따로 작성하여 이를 가지고 검증함으로써 상위 계층에 대한 HAL 계층의 적합성과 안정성을 보장할 수 있다. HCT는 HCT 에이전트와 HCT 서버로 구성되어 있어서 테스트케

이스를 서버가 관리하도록 함으로써 메모리가 부족한 단말기에서도 충분히 테스트실행이 가능한 구조로 설계하였다. 이는 PCT와 유사한 방식이다.

본 논문에서 제안한 PCT는 WIPI 에뮬레이터에서 동작 및 테스트 실험을 완수하였으며 HCT 인증도구는 QPlus 기반의 WIPI HAL에서 동작 실험을 수행하여 그 확장성과 유용성을 확인하였다. 본 논문에서 제안한 HCT를 이용하면 제조사와 플랫폼 업체간에 문제가 발생시 문제의 출처를 명확히 함으로써 디버깅을 도울 수 있고 개발 기간을 단축시킬 수 있으리라 기대한다. 향후에는 WIPI 규격의 버전 업그레이드 따른 지원이 있어야 할 것이며, 테스트 시간을 단축시킬 수 있는 방안에 대한 연구가 더 이루어져야 할 것이다.

참 고 문 헌

[1] 모바일 표준 플랫폼 규격 2.0, 한국정보통신기술협회, 표준번호 TTAS.KO-06.0036/R3

[2] 이환구, 김우식, 이상윤, 이재호, 김선자, "WIPI 플랫폼 인증 툴킷 개발", 한국정보처리학회 추계학술발표대회, 제 11권 제2호, pp.1539-1542, 2004.

[3] JCP, www.jcp.org

[4] 이상윤, 김선자, 김홍남, "한국 무선 인터넷 표준 플랫폼(WIPI)의 표준화 현황 및 발전 전망", 한국정보과학회학회지, 제 22권 제 1호 통권 제 176호, pp.16-23, 2004.

[5] 이상윤, 김선자, 김홍남, "무선인터넷 표준 플랫폼 WIPI 2.0", TTA Journal 통권 92호, pp.97-102, 2004.

[6] Jaeho Lee, Sunja Kim, Sangyun Lee, Woosik Kim, Hwangu Lee, "Implementation WIPI for Linux-based Smartphone", ICACT 2005.

[7] Jaeho Lee, Sunja Kim, Sangyun Lee, "Embedded Linux-based smartphone platform for sharing WIPI contents", IT-SOC 2004, pp.550-553, 2004.

[8] 유용덕, 박충범, 최훈, 김우식, "위피 응용프로그램 개발환경 설계 및 구현", 한국정보처리학회, 제12-C권 제5호(통권 제101호), pp.749-756, 2005.

[9] 김유일, 이원재, 한환수, 이재호, 김선자, "이동단말기 환경에서 응용프로그램 로더와 동적 링커 개발", 한국정보과학회 2004년 춘계학술대회 제 31권 제 1호, pp.841-843, 2004.

[10] 김연수, 강민철, 유용덕, 최훈, "무선인터넷 플랫폼에서 다중 응용프로그램 수행을 위한 스케쥴러 설계", 한국정보처리학회 2004년 추계학술대회, 제 11권 제 2호, pp.1759-1762, 2004.

[11] 임형택, 장준, 최훈, "무선인터넷 플랫폼에서의 API 관리자 설계 및 구현", 한국정보처리학회 2004년 추계학술대회 제11권 제2호, pp.1763-1766, 2004.

[12] 유용덕, 김연수, 임형택, 강민구, 최훈, "무선인터넷 플랫폼을 위한 메모리 관리 모듈 설계 및 구현", 한국정보처리학회 2004년 추계학술대회, 제11권 제2호, pp.1783-1786, 2004.

[13] 박충범, 김연수, 연대진, 유용덕, 최훈, "무선인터넷 플랫폼 에뮬레이터를 위한 HTTP API 설계 및 구현", 한국정보처리학회 2004년 추계 학술 대회, 제11권 제2호, pp.1791-1794, 2004.

부 록

1) 규격성 테스트케이스 예제

〈표 1〉 MC_fsOpen 함수의 테스트케이스 일례

```
void MC_fsOpen1002(void)
{
        M_Int32      fs;
        M_Char *fsName="(|*&^testFS";
  memset(gTestCaseTitle, 0x00, EXE_LINE_BUF);
  strcpy(gTestCaseTitle, "MC_fsOpen1002");

        // 수행 결과를 버퍼에 넣는다.
        fs=MC_fsOpen(fsName, MC_FILE_OPEN_RDWR,
           MC_DIR_PRIVATE_ACCESS);
        if(fs==M_E_BADFILENAME)
        {
                EXE_outStatus(EXE_PASS,
                "File name found wrong");
        }
        else
        {
                EXE_outStatus(EXE_FAIL,
                "Result is not M_E_BADFILENAME");
        }
        MC_fsClose(fs);
        MC_fsRemove(fsName, MC_DIR_PRIVATE_ACCESS);
}
void MC_fsOpen1003(void)
{
        M_Int32fs;
        M_Char *fsName="test01234567890123456789012345678901234567890";
        memset(gTestCaseTitle, 0x00, EXE_LINE_BUF);
        strcpy(gTestCaseTitle, "MC_fsOpen1003");

        // 수행 결과를 버퍼에 넣는다.
        fs=MC_fsOpen(fsName,
        MC_FILE_OPEN_RDWR, MC_DIR_PRIVATE_ACCESS);
        if (fs==M_E_LONGNAME)
        {
                EXE_outStatus(EXE_PASS,
                "returns M_E_LONGNAME");
        }
        else
        {
                EXE_outStatus(EXE_FAIL,
                "Result is not M_E_LONGNAME");
        }
        MC_fsClose(fs);
        MC_fsRemove(fsName, MC_DIR_PRIVATE_ACCESS);
}
```

2) 기능성 테스트케이스 예제

〈표 2〉 기능성 테스트케이스 일례

```
public class TAPITest extends InteractiveTest {


        /**
         * Assertion testing for TAPITest.
         * 플랫폼의 응용 프로그램이 실행 도중에 전화가 걸려 오거나, SMS 메시지가
           전달 되었을 경우
         * 플랫폼은 응용 프로그램에게 이 사실을 전달해야 한다. 실제로 SMS 메시지
           확인을 하거나
         * 전화 통화를 하는 것은 사용자가 선택하는 것이 가능해야 한다. 어플리케이션
           을 실행하고 전화나 SMS를 걸어 이를 확인
         */
        public Status tapiTest() {
                String instruction    "어플리케이션 실행중에 걸려온 전화 통화나
                                       SMS 확인이 가능한지 확인하세요.";
                String question           "어플리케이션 실행중에 걸려온 전화
                                           통화나 SMS 확인이 가능합니까?";
                String passMsg = "OK";
                String failMsg   "Fail";

                addInfo(instruction);
                tc    new Displayable();
                setTestDisplay(tc);
                card    new Card(1);
                command = new Command[1];
                card[0]    new Card(0, 0, 100, 100) {
                        int y    10;

                        public void paint(Graphics g){
                                g.setColor(0xff,0xff,0xff);
                                g.fillRect(0,0,this.getWidth(),
                                this.getHeight());
                                g.setColor(0,0,0);
                                g.drawString
                                ("Testing...", 10, y, g.TOP|g.LEFT);
                                y = 10;
                                if( y > this.getHeight())
                                        y = 0;
                                repaint();
                                try {
                                        Thread.sleep(100);
                                } catch(InterruptedException e){
                                }
                        }
                };

                command[0]    new Command("Start", null);
                tc.addCommand(command[0]);
                ml.addListener(new CommandListener() {
                        boolean called    false;
                        public void commandAction(Command
                        c, int type, Object s) {
                                if( s    tc) {
                                        if (called) {
                                                called=
                                                false;
                                                disp.re
                                                move
                                                Card
                                                (card[0]);
                                        }
                                }
                                if(s    tc && c
                                command[0]
                                && type
                                CommandListener.
                                SELECT) {
                                        if (! called) {
                                                called
                                                true;
                                                disp.push
                                                Card
                                                (card[0]);
                                        }
                                }
                        });


        }
```

3) 효율성 테스트케이스 예제

〈표 3〉 파일 시스템의 효율성 테스트케이스 일례

```
/*
 * File System 접근 테스트 : 파일을 생성하고 문자열을 저장하고 파일을
   읽고 지우는 동작을 100회 수행한다.
 */
void FileTestC(void)
{
        M_Int64 startTime, stopTime;
        M_Int32 count = 100;
        M_Int32 fs;
        M_Char *fsName = "test_c.txt";
    M_Char buf[EXE_LINE_BUF * 2];

        M_Char *buf1 = "This is test file.";
    M_Char *buf2 = "Hello World";

        M_Char t_text[EXE_LINE_BUF * 2];
        M_Int32 i;

    memset(gTestCaseTitle, 0x00, EXE_LINE_BUF);
    strcpy(gTestCaseTitle, "FileTestC");

        startTime = (M_Int64) MC_knlCurrentTime();

        for (i=0; i<count ; i++ )
        {
        fs = MC_fsOpen(fsName, MC_FILE_OPEN_RDWR,
              MC_DIR_PRIVATE_ACCESS);
                MC_fsWrite(fs, buf1, strlen(buf1));
                MC_fsSeek(fs, 0, MC_FILE_SEEK_SET);

                memset(buf, 0x00, EXE_LINE_BUF * 2);
            MC_fsRead(fs, (char *) &buf, EXE_LINE_BUF * 2);

        MC_fsSeek(fs, 5, MC_FILE_SEEK_SET);
        MC_fsWrite(fs, buf2, strlen(buf2));
        MC_fsSeek(fs, 0, MC_FILE_SEEK_SET);

                memset(buf, 0x00, EXE_LINE_BUF * 2);
            MC_fsRead(fs, (char *) &buf, EXE_LINE_BUF * 2);

        MC_fsClose(fs);
                MC_fsRemove(fsName, MC_DIR_PRIVATE_ACCESS);

        }

        stopTime = (M_Int64) MC_knlCurrentTime();

        memset(t_text, 0x00, EXE_LINE_BUF * 2);
        MC_knlSprintk(t_text, "takes %d ms", s
        topTime    startTime);
        EXE_outStatus(EXE_RESULTONLY, t_text);
}
```

4) 안정성 테스트케이스 예제

〈표 4〉 안정성 테스트케이스 일례

```
/*
 *  Network Stress Test : 네트워크 연결과 종료를 50번 반복한다.
 */

void EXE_netConnectCallBack(int error, void *param)
{
}

void NetworkStressTestC(void)
{
        M_Int32 count = 50;
        M_Int64 startTime, stopTime;

        M_Char t_text[EXE_LINE_BUF * 2];
        M_Int32 i;
        M_Int32 net_connect, socket;

    memset(gTestCaseTitle, 0x00, EXE_LINE_BUF);
    strcpy(gTestCaseTitle, "NetworkStressTestC");

        startTime = (M_Int64) MC_knlCurrentTime();

        for (i = 0; i < count; i++ )
        {
            net_connect = MC_netConnect(EXE_netConnectCallBack, NULL);

            socket = MC_netSocket(MC_AF_INET, MC_SOCKET_STREAM);

            MC_netSocketClose(socket);
#ifndef EXE_NETWORK_TEST
            MC_netClose();
#endif /* EXE_NETWORK_TEST */
        }

        stopTime = (M_Int64) MC_knlCurrentTime();

        memset(t_text, 0x00, EXE_LINE_BUF * 2);
        MC_knlSprintk(t_text, "takes %d ms", stopTime - startTime);
        EXE_outStatus(EXE_RESULTONLY, t_text);
}
```


이 상 윤

e-mail : syllee@etri.re.kr

1994년 2월 한양대학교 전자통신공학과 (학사)

1996년 2월 한양대학교 전자통신공학과 (석사)

1999년 9월~현재 한국전자통신연구원 임베디드S/W연구단 선임연구원

관심분야 : 무선인터넷플랫폼, JVM, 임베디드 시스템

이 환 구

e-mail : hgllee@etri.re.kr

2002년 2월 한양대학교 기계공학(학사)

2004년 2월 한양대학교 정보통신대학원 (석사)

2004년 3월~현재 한국전자통신연구원 임베디드S/W연구단 연구원

관심분야 : 임베디드 시스템, Wibro, WIPI

최 병 욱

e-mail : buchoi@hanyang.ac.kr

1973년 한양대학교 전자공학과(학사)

1978년 일본 경응의숙(KEIO) 대학 전기공학과(석사)

1981년 일본 경응의숙(KEIO) 대학 전기공학과(박사)

현 재 한양대학교 정보통신대학 정보통신학부 교수

관심분야 : 영상처리, 멀티미디어 공학