논문 2007-44CI-4-2

다중 플랫폼을 지원하는 위피 실행 엔진 참조 구현

(Reference Implementation of WIPI Runtime Engine Supporting Multiple Platforms)

이 상 윤*, 최 병 욱**

(Sang-Yun Lee and Byung-Uk Choi)

요 약

본 논문에서는 REX OS, Qplus, 윈도우즈 등 다양한 플랫폼을 지원하는 위피 실행 엔진의 참조 구현을 제안한다. 각 플랫폼에 따른 위피 실행 엔진의 설계 방식을 기술하고, 중복 개발을 피하기 위한 방안을 제시한다. 또한, REX OS 상에서의 링커와 로더의 구현을 설명하고, 임베디드 리눅스인 Qplus에서의 실행 엔진 구조를 기술한다. 그리고 자바 가상 머신 기반의 Jlet/MIDlet 에뮬레이터와 윈도우즈 기반의 Clet 에뮬레이터 구현 방법에 대해서 소개한다. 마지막으로 호환성 인증 도구인 PCT 및 HCT의 검증 결과와 예제 프로그램의 정상적인 작동을 통해 제안된 참조 구현의 호환성 및 완성도를 검증한다.

Abstract

In this paper, we propose the reference implementation of WIPI runtime engine supporting various platforms such as REX OS, Qplus and Windows. We describe the architecture of WIPI runtime engine according to each platform, and introduce the method for avoiding repetitive development. And we explain the implementation of a linker and a loader on REX OS and describe the runtime engine structure on Qplus, a kind of embedded linux. And we introduce the implementation of the Jlet/MIDlet emulator based on a Java virtual machine and the Clet emulator based on Windows. Finally we verify the interoperability and the perfection of the proposed reference implementation through the result of the HCT and the PCT and the normal operation of the example programs.

Keywords : 위피, 실행 엔진, Clet, Jlet, 에뮬레이터

I. 서 론

국내 이동통신사들은 음성 통화와 문자 서비스 중심의 수익원의 성장세가 둔화되고 가입자 수가 포화 상태에 다다름에 따라 무선 콘텐츠라는 새로운 시장을 창출하게 되었다. 초창기에는 WAP(Wireless Application Protocol)이나 모바일 익스플로러와 같은 브라우저 방식의 서비스가 도입되었다. 이 방식은 유선 환경의 인터넷처럼 통신 채널을 통해 서버에 접속하여 정보를 읽어 온다. 하지만, 이 방식은 한번 읽어 온 정보를 저장할 수 없으며, 사용할 때마다 통신 채널에 연결해야 하므로 통신 요금 부담이 있고 게임과 같은 복잡한 프로그램을 실행시킬 수 없다. 이러한 단점을 극복하기 위해 도입된 것이 무선 인터넷 플랫폼이다. 무선 인터넷 플랫폼은 CDMA/GSM 통신 채널을 통해 응용 프로그램을 다운로드 받은 후 단말기에 저장하고, 필요할 때마다 저장된 프로그램을 반복하여 실행할 수 있다. 뿐만 아니라, 게임과 같은 복잡한 프로그램도 실행할 수 있다.

국내에서는 이동통신회사마다 도입한 무선 인터넷

* 정회원, 한국전자통신연구원 임베디드S/W연구단
(Embedded S/W Research Division, Electronics and Telecommunications Research Institute)
** 평생회원, 한양대학교 정보통신대학 정보통신학부
(Division of Information and Communications, Hanyang University)
※ 본 연구는 정보통신부 및 정보통신연구진흥원의 IT 신성장동력핵심기술개발사업의 일환으로 수행하였음. [2006-S-038-02, 모바일 컨버전스 컴퓨팅을 위한 단말적응형 임베디드 운영체제 기술]
접수일자: 2007년2월15일, 수정완료일: 2007년6월21일

플랫폼이 각기 달랐다. LGT는 자바를 지원하는 J2ME (Java2 Micro Edition)를, SKT는 자바를 지원하는 SK-VM과 C 언어를 지원하는 GVM을, KTF는 C 언어를 지원하는 MAP과 C/C++ 언어를 지원하는 BREW(Binary Runtime Environment for Wireless)를 도입하여 서비스 해오고 있다[1]. 콘텐츠 제공자와 단말기 제조업체들은 동일한 서비스를 다양한 플랫폼에 제공하기 위해 각기 포팅해야만 하는 중복 투자를 해야만 했고, 가입자들은 자신의 단말기에서 사용할 수 있는 콘텐츠가 제한적이었다. 이는 무선 인터넷 콘텐츠 시장이 성장하고 활성화되는데 큰 걸림돌이 되어 왔다. 이에, 무선 인터넷 플랫폼에 대한 표준화를 추진하게 되었다.

위피(WIPI; Wireless Internet Plaftorm for Interoperability)는 한국 무선 인터넷 표준화 포럼에서 제정하였으며[2] 한국 정보 통신 기술 협회에서 단체 표준 규격으로 채택된 대한민국 모바일 표준 플랫폼이다[3]. 2005년 4월부터는 한국에서 출시되는 모든 단말기에 의무적으로 탑재하도록 제도화하였다[4].

위피는 C언어, 자바 언어를 모두 지원하며 자바 언어로 작성된 프로그램을 바이너리 코드 형태로 실행함으로써 바이트 코드 실행 환경인 자바 가상 머신보다 빠르게 응용 프로그램을 실행시킬 수 있다. 또한 여러 개의 응용 프로그램을 동시에 실행할 수 있고 새로운 API를 추가하거나 갱신하는 기능을 제공한다. 이는 위피가 BREW나 J2ME 보다 우수한 기능을 제공함을 의미한다. 그림 1은 위피 시스템의 구조도를 보여준다.

위피 시스템은 REX OS, 리눅스, 그리고 윈도우즈와 같은 네이티브 시스템 소프트웨어에 탑재되며 플랫폼의 이식성을 높이기 위한 표준화된 하드웨어 추상화 계층인 HAL(Handset Adaptation Layer), 응용 프로그램 실행 환경인 실행 엔진, 다양한 응용 프로그램 및 서비스를 제공하는 위피 라이브러리(위피-C, 위피-Java, J2ME), 그리고 응용 프로그램을 다운로드하거나 저장 및 삭제 등을 관리하는 위피 응용 관리자(WAM; WIPI Application Manager)로 구성되어 있다. 위피 표준 규격은 위피-C/위피-Java/HAL API를 표준화 대상으로 하였으며 실행 엔진은 포함하고 있지 않다. 위피 2.0부터는 JSR 139인 CLDC 1.1과 JSR 118인 MIDP 2.0[5]을 수용하여 기존 J2ME 환경에서 서비스되던 자바 콘텐츠들도 위피에서 서비스가 가능하도록 했다.

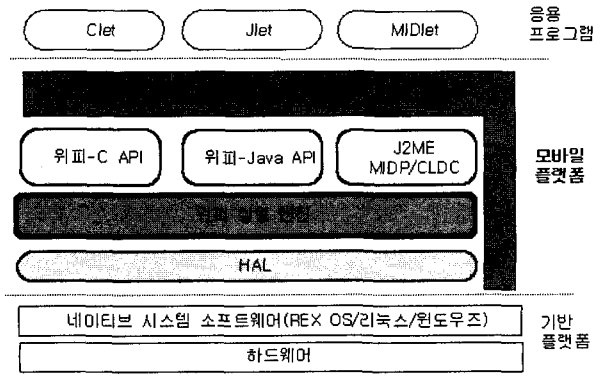


그림 1. 위피 시스템 구조도
Fig. 1. WIPI system architecture.

위피 플랫폼 기술은 아로마소프트, 지오텔, 이노에이스, XCE 등 국내의 몇 개 업체에 의해 독점 개발되고 있다. 그런데 경쟁이 치열해서 기술 공개가 되고 있지 않다 보니, 위피 플랫폼 연구에 대한 논문이나 기술 문서들을 접하기가 쉽지 않다. 특히, REX OS 상에서의 위피 실행 엔진에 대해서는 거의 알려진 바가 없으며, 단지 윈도우즈 상에서 동작하는 에뮬레이터가 공개되어 콘텐츠 개발자들에게 제공되고 있을 뿐이다.

본 논문에서는 위피 플랫폼의 핵심 기술인 실행 엔진의 개발 방법과 경험을 공유함으로써 위피 플랫폼 개발을 확산시키고 개발자들에게 실질적인 도움을 주고자 한다.

위피 플랫폼은 처음에는 REX OS 기반의 휴대폰을 대상으로 개발되었으나 스마트폰, DMB폰[6~7], 와이브로 단말기, 모바일 RFID[8], 그리고 텔레매틱스 단말기 등에서 이를 도입하려는 시도가 늘고 있다. 이는 위피 플랫폼의 우수성도 한 원인이지만, 콘텐츠를 새로 개발하지 않고 휴대폰용으로 개발된 위피 콘텐츠를 다른 단말기에서도 그대로 활용하고자 함이 큰 원인이라고 할 수 있다. 그런데, 이러한 단말기들에 임베디드 리눅스가 기반 플랫폼으로 많이 채택되고 있다.

본 논문에서는 한국전자통신 연구원에서 개발한 임베디드 리눅스 중의 한 종류인 Qplus에 기반한 위피 플랫폼을 제안한다. 또한, Java2C 컴파일러를 통해 자바 프로그램을 고속으로 실행시키는 방안을 제안하고, Clet, Jlet, MIDlet 등을 미리 데스크탑 PC에서 작동시켜볼 수 있는 에뮬레이터를 제안한다.

본 논문의 구성은 다음과 같다. II장에서는 다양한 플랫폼에서의 Clet 실행 엔진 설계 방법을 기술한다. III장에서는 Jlet 실행을 위한 Java2C 컴파일러와 구현해야할 네이티브 API에 대해서 논의하고 Jlet/MIDlet 에뮬레이터를 소개한다. IV장에서는 REX OS, Qplus, 그리고 윈도우즈 상에서 구현된 위피 참조 구현의 실제

동작 모습을 보인다. 마지막으로 V장에서는 본 논문을 요약하고 결론을 맺는다.

Ⅱ. Clet 실행 엔진 설계

1. REX OS 지원 엔진 설계

CDMA 방식의 통신을 서비스 하는 단말기는 퀄컴의 MSM 계열의 칩을 탑재하며, REX 라는 태스크 기반의 OS 상에서 소프트웨어가 동작한다. REX OS는 DMSS API라고 하는 어플리케이션 개발용 API를 제공한다. 그런데, DMSS API를 사용해서 개발한 소프트웨어는 대부분 단말기에 기본적으로 탑재되어 출시되며 REX OS는 독립된 동작을 할 수 있는 태스크를 스케쥴링 하는 기능을 주로 한다.

그런데, 윈도우즈나 리눅스 같은 PC용 운영체제에서 당연히 제공하는 메모리 관리, 링커, 로더, 그리고 프로그램 관리 기능 등이 없기 때문에 위피처럼 응용 프로그램을 다운로드하여 실행시키려면 이러한 기능들을 제공하는 위피 실행 엔진 개발이 필요하다. 이 절에서는 REX OS 상에서의 실행 엔진의 구조를 제안하고 링커와 로더에 대해서 설명한다.

가. 엔진 구조

위피 실행 엔진을 기존 REX OS 태스크 기반 환경에서 작동시키는 방법은 두 가지를 생각해 볼 수 있다. 가장 안전하고 쉬운 방법은 UI 태스크에 엔진을 끼워 넣는 방식이나 이는 사용자의 키 입력이나 외부로부터의 이벤트를 처리하기가 용이하지 않다.

두 번째 방법은 새로운 위피 태스크를 생성하는 방식인데, 실행 엔진을 독립적으로 관리할 수 있어 이벤트 처리가 용이하고 다른 태스크 동작에 영향을 주지 않아 좀 더 유연하게 REX OS에서 동작시킬 수 있다. 본 논문에서는 두 번째 방법을 채택했다.

실행 엔진은 Clet 응용 프로그램을 메모리에 적재하고 필요한 라이브러리를 연결해 주는 링커와 로더, 동적으로 메모리를 할당하고 자동으로 해제해 주는 메모리 관리자[9], Clet의 라이프 사이클에 따라 동작을 관리해주는 스케줄러[10], UI 태스크로부터 받은 이벤트를 처리해 주는 이벤트 핸들러로 구성되어 있다.

그림 2는 본 논문에서 제안하는 위피 실행 엔진의 내부 구조를 나타낸다.

위피-C 라이브러리와 WAM, Clet 응용 프로그램은 모두 위피 태스크라는 새로운 태스크에서 동작한다. 실

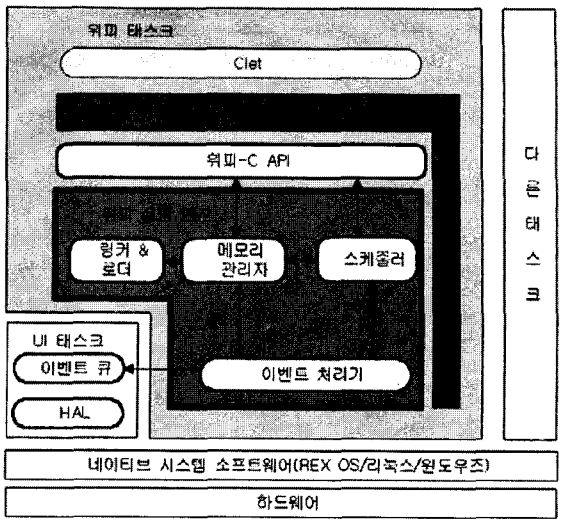


그림 2. 위피 실행 엔진 구조
Fig. 2. WIPI runtime engine architecture.

행 엔진을 구동시키기 위해서는 표준 규격에서 정의한 다음과 같은 두 개의 함수를 구현해줘야 한다.

M_Int32 MH_pltStart(M_Int32 JavaC, M_Char programID, M_Char* path, M_Char* args);*

이 함수는 실행 엔진을 구동하는 함수로서 인자로 지정된 응용 프로그램 관리자 프로그램을 실행시킨다. 이 함수는 응용 프로그램 관리자가 종료하여 엔진이 종료될 때까지 반환되지 않는다.

*M_Boolean MH_pltEvent(MH_Event event, void *param);*

이 함수는 응용 프로그램 수행 중에 발생하는 키 입력 이벤트나 타이머 이벤트 등 다양한 이벤트를 실행 엔진에 전달하기 위해 호출된다. REX OS는 단말기에서 발생하는 모든 이벤트를 UI 태스크에 넘겨주는데, 발생한 이벤트는 UI 태스크의 이벤트 큐에 저장된다. 저장된 이벤트가 위피 실행 엔진에서 사용되어야 할 이벤트라면 위피 실행 엔진의 이벤트 핸들러가 처리한다. 실행 엔진은 응용 프로그램의*handleCletEvent()* 함수를 호출하여 이벤트를 처리한다.

나. Clet 응용 프로그램 구동 시나리오

단말기가 부팅되면 위피 태스크는 대기 상태로 들어간다. 메뉴에서 위피 시작 버튼을 누르면 UI 태스크가 사용자의 입력을 처리하여 위피 태스크에 플랫폼을 시

작하도록 명령한다. 위피 태스크는 대기 상태에서 깨어나 WAM을 구동한다. WAM은 EFS(Embedded File System)에 있는 Clet 응용 프로그램 목록을 사용자에게 보여주고 UI 태스크는 사용자의 입력을 받아 *MH_pltEvent()* 함수를 호출하여 실행 엔진의 이벤트 큐에 이를 저장한다.

위피 태스크는 실행 엔진의 이벤트 큐에 새로운 이벤트가 들어오면 WAM의 *handleCletEvent()* 함수로 이벤트를 전달한다. WAM은 이벤트 처리를 통해 사용자가 선택한 Clet을 *MC_knlExecute()* 함수를 호출하여 구동시키고 이후 사용자의 입력은 활성화된 Clet으로 전달된다. 사용자가 Clet을 종료하면 WAM이 다시 활성화되고 WAM이 종료되면 실행 엔진에 MH_EXIT_EVENT가 전달되고 *MH_pltStart()* 함수로부터 반환된다. 이 후 위피 태스크는 UI 태스크에 위피 플랫폼이 종료되었음을 알리고 다시 대기 상태로 들어간다. UI 태스크는 초기 메뉴 상태를 출력하고 사용자의 입력을 기다린다.

다. 동적 링커 및 로더 설계

그림 3은 위피 시스템 환경에서 콘텐츠 제공자가 개발한 프로그램이 COD(Comfile On Demand) 서버를 거쳐 사용자의 단말기에 다운로드된 후 실행되는 일련의 과정을 나타내고 있다.

COD 서버에서 생성된 hello.elf는 이 프로그램 실행에 필요한 함수들을 정적으로 포함하고 있지 않고, 필요한 함수들에 대한 주소 정보를 저장하고 있다. 실행에 필요한 모든 함수들을 정적으로 포함하고 있으면, 다른 프로그램들에서도 사용되는 함수들을 중복으로 저장할 수 있으므로 메모리가 제한되어 있는 단말기 환경에는 적합하지 않다. 따라서 단말기에서는 필요한 함수들을 동적으로 라이브러리(libwipi.so)와 링크해서 실행시키는 구조를 가져야 한다.

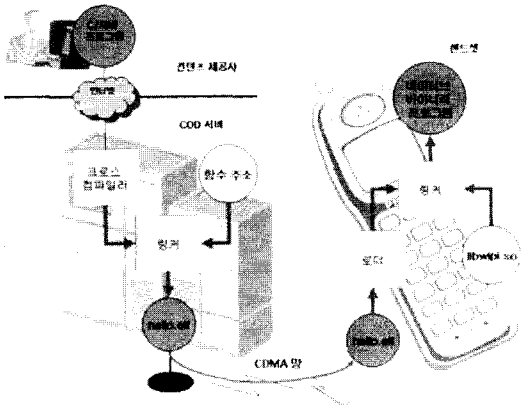


그림 3. 위피 프로그램 수행 절차
Fig. 3. WIPI program execution process.

로더는 실행 파일을 해석하여 코드와 데이터를 메모리에 적재하는 작업을 수행하는 프로그램을 말한다. 동적 링커는 코드에서 공유 라이브러리를 호출하는 경우에 공유 라이브러리를 메모리에 적재하고 코드 안의 공유 라이브러리 함수 호출이 실제 공유 라이브러리가 적재된 메모리 주소와 일치하도록 코드를 수정하는 작업을 담당하는 프로그램을 말한다[11].

REX OS는 로더와 링커가 없기 때문에 본 논문에서는 이를 개발하였다. 실행 파일의 형식에 따라 로더와 링크의 구현 방법이 달라진다. 본 논문에서는 리눅스 환경에서 가장 많이 사용되는 ELF(Executable and Linking Format)[12]을 선택하였다. ELF는 매우 유연하여 불필요한 정보를 제거하거나 필요한 정보를 추가할 수 있다. 그림 4는 본 논문에서 구현한 링커와 로더의 동작을 나타낸다[13].

로더는 실행 파일의 프로그램 헤더 테이블을 읽어 세그먼트를 메모리에 적재한다. 실행 파일에 동적 세그먼트가 존재하면 링커는 필요한 공유 라이브러리를 메모리에 적재하고 동적 세그먼트 내부의 심볼 테이블, 해시 테이블, 재배치 테이블 등을 사용하기 쉽도록 포인터를 적절히 설정하는 자료 구조 준비 과정을 거친다. 이 후 심볼 해석을 통해 응용 프로그램과 공유 라이브러리의 의존성 관계를 파악한 후 재배치 과정을 거친다. 동적 세그먼트가 존재하지 않는 경우 GOT(Global Offset Table) 각 항목에 적재된 코드의 기반 주소를

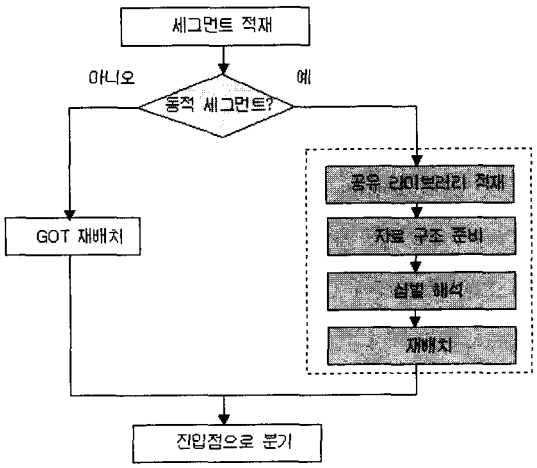


그림 4. 로더 및 동적 링커의 동작 절차
Fig. 4. The procedure of a loader and a dynamic linker.

더해주는 GOT 재배치 작업이 진행되고 진입점으로 분기한다[13].

2. Qplus 지원 위피 플랫폼 설계

본 논문에서 제안한 Qplus 기반 위피 플랫폼의 구조는 그림 5와 같다. 위피I-C, 위피I-Java, 그리고 J2ME API는 REX OS에서 구현한 라이브러리를 그대로 사용한다. REX OS와 가장 큰 차이점은 Qplus 에서는 기본적으로 링커와 로더, 메모리 관리 기능 등을 제공하므로 이에 대한 개발이 필요 없다는 것이다.

HAL은 REX OS의 DMSS API 대신 Tiny-X/GTK와 Glibc 라이브러리를 이용해 구현하였다. TAPI는 CDMA 모듈을 이용할 수 있는 텔레포니 API이다. 이 API를 이용해 통신 관련 기능을 제공할 수 있다[14]. HCT(HAL Certificate Toolkit) 에이전트는 HCT 시스템의 일부분으로 HCT 서버와 통신하면서 구현된 HAL이 표준 규격을 준수하는지 검증한다[15].

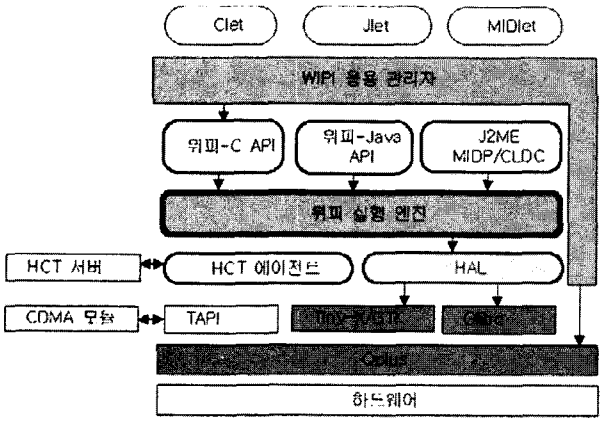


그림 5. Qplus 상에서의 위피 플랫폼 구조
Fig. 5. The architecture of WIPI platform on Qplus.

3. Clet 에뮬레이터 설계

Clet 에뮬레이터는 콘텐츠 제공자가 Clet을 데스크탑 PC에서 미리 동작시켜 보는 목적으로 만들지만, 본 논문에서는 이 목적 이외에 위피-C API를 편리하게 개발하려는 목적도 있다. 이는 윈도우즈 환경이 개발 환경과 디버깅 환경이 다른 플랫폼 보다 우수하기 때문이다. 윈도우즈 환경에서 구현된 위피-C API는 그대로 REX OS 환경이나 다른 운영 체제 환경에서 수정 없이 컴파일되고 동작될 수 있도록 설계하였다. 플랫폼에 따라 수정해야 될 부분들은 모두 HAL 계층에서 발생할 수 있도록 설계함으로써 다른 플랫폼으로 포팅할 때 수정 작업을 최소화 할 수 있도록 설계하였다.

Clet 에뮬레이터 시스템의 구조는 기본적으로 REX OS 기반 엔진과 크게 다르지 않다. 그림 6에서 볼 수 있듯이, 스케쥴러, 메모리 관리자, 이벤트 처리기는 REX OS에서 동작하는 코드를 그대로 사용할 수 있도록 설계하였으며 링커와 로더는 운영 체제에서 제공하는 것을 사용한다. 에뮬레이터가 동작하는 운영 체제에 맞게 HAL을 포팅하면 이 에뮬레이터는 바로 그 운영 체제에서 동작이 가능하다.

동시에 여러 개의 Clet 프로그램을 동작시키기 위해서는 이를 관리할 수 있는 스케쥴러가 필요하다. 본 논문에서는 플랫폼의 실행엔진이 Clet 내의 함수들을 전체적으로 관리하고 호출함으로써 응용프로그램의 실행을 스케쥴링하도록 설계하였다. 즉, 다중 응용프로그램 실행에 있어 새로운 응용프로그램이 실행되면 이전에 실행되던 응용프로그램은 *pauseClet()* 함수를 실행함으로써 일시 정지하도록 하였다[10]. 운영 체제가 제공하는 쓰레드 기능을 이용하면 구현은 쉽게 할 수 있겠으나, REX OS처럼 쓰레드를 지원하지 않는 운영 체제에서는 이를 사용할 수 없으므로 포팅의 부담을 덜기 위해서는 독립적인 스케쥴러 구현이 필요하다.

위피-C API 중 메모리 할당 및 해제와 관련된 API는 *MC_knlAlloc(), MC_knlCalloc(), MC_knlFree()* 이다. 이를 지원하기 위해 운영체제에서 제공하는 *malloc(), calloc(), free()* 함수를 이용할 수 있으나, REX OS는 이런 함수들을 제공하지 않는다. 따라서 실행 엔진 내부에서 이 기능을 담당할 별도의 메모리 관리기가 필요하다. 본 논문에서는 자유 리스트 방식의 메모리 할당과 해제 방식을 이용하였다.

이벤트 처리기는 Clet 응용프로그램을 실행하는 동안

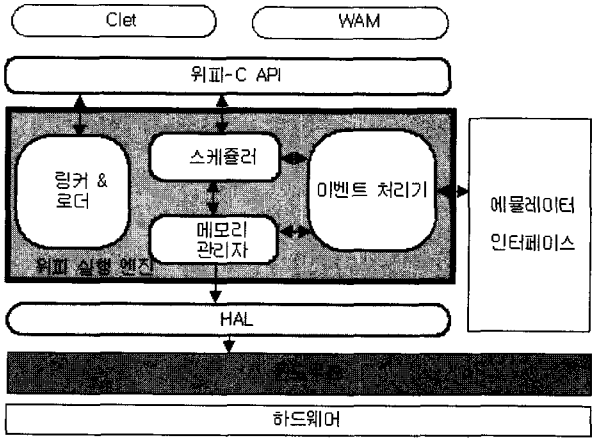


그림 6. Clet 에뮬레이터의 시스템 구조
Fig. 6. Architecture of Clet emulator.

에뮬레이터 인터페이스에서 생성된 이벤트나 각종 위피 관련 이벤트들을 처리한다.

III. Jlet/MIDlet 실행 엔진 설계

1. Java2C

자바로 작성된 프로그램을 자바 가상 머신으로 실행시키면 속도가 느리다는 단점이 있다. 위피 표준 규격에서는 이 단점을 극복하기 위해 자바를 바이너리 코드로 실행시키도록 규정하고 있다. 본 논문에서는 AOTC (Ahead-Of Time Compiler)[16] 방식을 이용한 Java2C를 제안한다. Java2C는 컴파일 된 클래스 파일(.class)을 입력받아 분석한 후 자바 프로그램과 동일한 동작을 수행할 수 있는 C 소스 파일과 헤더 파일을 생성한다. 이 파일들은 타겟 컴파일러에 의해 타겟에 맞는 바이너리 코드로 컴파일된다. 이 일련의 과정들이 그림 7에 나타나 있다.

AOTC는 COD 서버에서 제공되므로 콘텐츠 개발자들은 자바 프로그램 개발에만 집중을 하면 된다.

자바는 객체 지향 언어로서 C 언어에서는 지원되지 않는 클래스의 상속 관계, 함수의 다형성, 예외 처리 등의 특징을 가지므로 코드 변환 시 이에 대한 처리 방안이 마련되어 있어야 한다.

본 논문에서는 클래스를 구조체로 정의하고 이 구조체는 모든 클래스가 공통적으로 포함하는 정보들을 공통 필드를 갖도록 했다. 공통 필드에는 클래스 이름, 상위 클래스의 자료 구조를 가리키는 포인터, 멤버 함수들의 주소 정보 등이 포함된다.

함수 포인터는 상속 받은 클래스의 함수 포인터와 해당 클래스의 함수 포인터를 모두 구조체에 포함하도록 했다. 이렇게 함으로써 상위 클래스와 동일한 이름을 갖는 멤버 함수의 중복도 처리가 가능하다. 예외 처리는 C 언어의 *setjump()*와 *longjump()* 함수를 이용하여 처리하였다. 표 1은 본 논문에서 제안한 Java2C를 이용해 코드를 변환한 간단한 예제를 보인다.

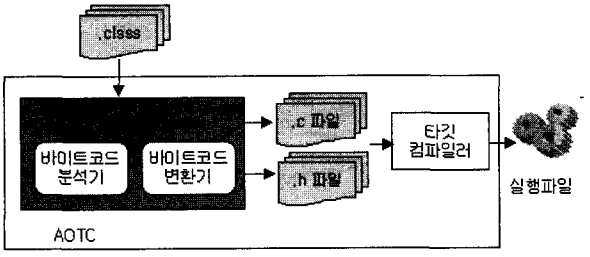


그림 7. Java2C를 통한 코드 변환
Fig. 7. Java2C Transcoder.

표 1. 바이트코드 변환 예제
Table 1. Example of Java2C Translator.

.java	바이트코드	.c

a = b + c;	iload_2
	iload_3
	iadd
	istore_1
		i1 = iv2;
		i2 = iv3;
		i1 = i1 + i2;
		iv1 = i1;

표 2. 구현된 네이티브 API 일례
Table 2. Example of implemented native APIs.

클래스명	네이티브 함수명
Runtime	public native long freeMeory(); public native void gc();
Thread	public static native int activeCount(); public final native boolean isAlive(); public synchronized native void start();
String	native public String toString(); native public char[] toCharArray();
System	public static native long currentTimeMillis();
Resource	static native synchronized public byte[] getResource(String name);
StringBuffer	native public String toString(); native public synchronized StringBuffer append (String str);

자바는 성능 개선이나 기존에 개발된 라이브러리를 활용하기 위해 JNI(Java Native Interface)를 지원한다.

표 2는 본 논문에서 구현한 네이티브 API들의 일부를 보여준다.

위피-Java 혹은 MIDP의 클래스를 변환하다 보면 JNI를 이용하는 네이티브 API를 만나게 되는데 타겟에 맞는 네이티브 API 가 구현되어 있어야 C 코드로 변환된 프로그램이 정상적으로 동작할 수 있다.

2. Jlet/MIDlet 에뮬레이터 설계

Jlet/MIDlet 에뮬레이터는 데스크탑 PC상에서 Jlet/MIDlet을 미리 동작시켜 볼 수 있는 도구이다. 본 절에서는 Jlet/MIDlet 에뮬레이터의 시스템 구조, 동작 과정, 그리고 기능에 대해서 소개한다.

가. 시스템 구조

REX OS에서 Jlet/MIDlet을 실행시킬 때는 Java2C 컴파일러를 이용해 자바 클래스를 바이너리 코드로 변환한 후 이를 수행하는 실행 엔진이 필요하지만, 에뮬레이터는 이러한 복잡한 절차를 굳이 따를 필요는 없다. 에뮬레이터는 Jlet/MIDlet 응용 개발자가 구현한 소스 코드를 핸드셋에서 동작하는 모습 그대로 데스크탑 PC에 실행시켜 주면 되는 것이다.

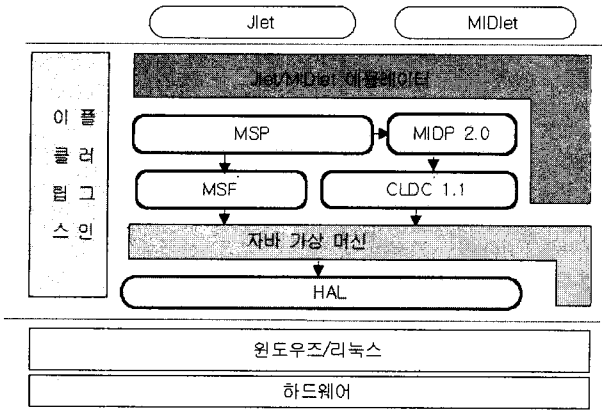


그림 8. Jlet/MIDlet 에뮬레이터의 시스템 구조
Fig. 8. Architecture of the Jlet/MIDlet emulator.

본 논문에서는 위피 실행 엔진 대신 자바 가상 머신을 도입해서 Jlet/MIDlet 클래스 파일을 실행시킬 수 있도록 설계하였다. 이렇게 함으로써, 복잡한 단계를 거치지 않고 Jlet/MIDlet을 실행시킬 수 있다.

또한, 자바 가상 머신이 탑재된 운영 체제에서는 이 에뮬레이터 사용이 가능하다. Jlet/MIDlet 에뮬레이터의 시스템 구조는 그림 8과 같다.

에뮬레이터는 독립적인 프로그램으로도 실행이 가능할 뿐만 아니라, 이클립스와 연동하여 소스 코드 작성 및 디버깅이 가능하도록 설계하였다.

나. 에뮬레이터 내부 모듈 구조 및 동작 과정

그림 9는 에뮬레이터의 내부 구조와 Jlet/MIDlet을 실행시키는 과정을 나타낸다.

워크스페이스 모듈은 등록된 Jlet/MIDlet 프로젝트를 보여준다. 또한, 새로운 프로젝트의 생성, 에뮬레이터 초기화, Jlet이 실행될 폰 윈도우를 구동시키는 기능 등을 수행한다. 설정 관리자 모듈은 프로젝트의 속성을

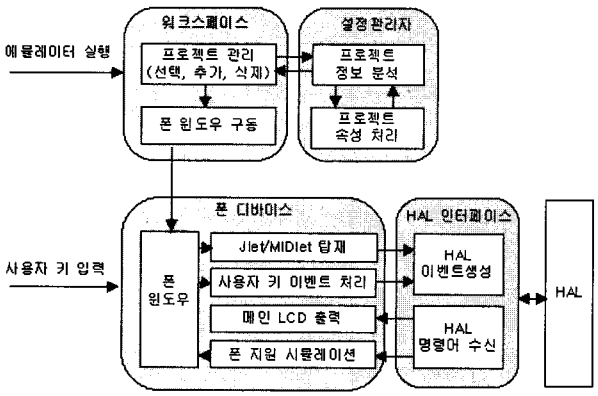


그림 9. 에뮬레이터의 실행 절차
Fig. 9. Execution procedure of the emulator.

파싱하는 기능, 프로젝트의 내용을 DB에 저장하거나 DB로부터 읽어오는 기능, Jlet/MIDlet 실행에 필요한 클래스 파일이나 자원 파일들을 워크스페이스로 읽어오는 기능 등을 수행한다.

폰 디바이스 모듈은 폰 윈도우의 메인 프레임 윈도우에 전체 스킨을 그려주는 기능, 사용자가 입력한 마우스 혹은 키보드 입력을 받아서 HAL 인터페이스에 전달하는 기능, 실제의 폰처럼 Jlet/MIDlet을 실행시키는 기능, LED나 미디어 플레이어 같은 폰 자원들을 실제 폰과 유사하게 데스크탑 PC 화면과 스피커에 출력하는 기능 등을 수행한다.

HAL 인터페이스 모듈은 폰 디바이스 모듈로부터 입력 받은 HAL 이벤트를 HAL 계층에 전달하는 역할을 한다. 이를 통해 LCD 화면 갱신, 사용자 키 입력, 이벤트 전달, 그리고 단말기 자원 사용이 가능하다.

에뮬레이터를 실행시키면, 워크스페이스 모듈과 컨피규레이션 모듈은 상호 통신하면서 Jlet/MIDlet 실행에 필요한 기본 정보를 수집한다. 이 정보를 바탕으로 폰 윈도우가 구동되며, 에뮬레이터 모듈과 HAL 인터페이스가 상호 작용하면서 프로그램 동작을 화면에 보여준다.

LCD 화면의 기본 크기는 143x175 이며 구동 초기에 설정이 가능하다. 실제 폰 단말기에서 제공하는 LCD, 백라이트, LED, 진동, 그리고 미디어 플레이어와 같은 단말기 리소스들의 작동은 애니메이션과 PC 스피커를 통해 시뮬레이션으로 표현되도록 설계하였다.

에뮬레이터는 독립적인 프로그램으로 동작이 가능하다. 즉, 제 3의 개발 툴을 이용해 Jlet/MIDlet을 구현한 후 에뮬레이터로 수행해 볼 수 있다. 하지만, 개발 과정에서 수시로 코드를 수정하면서 그 결과를 바로 보고자 한다면, 개발 툴과 에뮬레이터가 통합되는 것이 매우 편리하다. 제안된 에뮬레이터는 개방형 통합 개발 도구인 이클립스에 플러그-인으로 통합시켜 사용자의 편의를 도모하였다.

IV. 구현 및 실험

위피 플랫폼을 개발하려면 크게 HAL, 실행 엔진, 위피 라이브러리(위피-C, 위피-Java, J2ME)를 구현하여야 한다. 그리고, REX OS, 리눅스, 윈도우즈 등 다양한 플랫폼을 지원하도록 개발 할 때, 플랫폼에 따른 포팅이 최소화 될 수 있도록 설계하는 것이 중요하다.

본 논문에서는 위피 라이브러리는 수정하지 않고,

HAL과 실행 엔진의 수정만으로 위피 플랫폼을 포팅할 수 있도록 설계하였고 이 세 플랫폼을 지원하는 위피 플랫폼을 구현하였다.

1. REX OS 지원 위피 플랫폼

REX OS 상에서 구현된 링커와 로더의 동작 확인을 위해 "Hello world" 라는 문자열을 화면에 출력하는 간단한 프로그램을 실험하였다.

데스크탑 PC에서 ARM 컴파일러를 이용해 크로스-컴파일을 하였고, hello.elf 라는 파일을 생성하였다. 이 파일을 JTAG 툴을 이용해 휴대 단말기로 전송하여 실행하였다. 그림 10은 문자열이 화면에 제대로 출력되는 것을 보여준다.

링커와 로더의 정상 동작을 성공한 후 화면에 이미지를 출력하는 Clet 프로그램을 시험해 보았다. 그림 11은 단말기에 탑재된 위피가 Clet 프로그램을 제대로 동작시키고 있음을 보여준다.

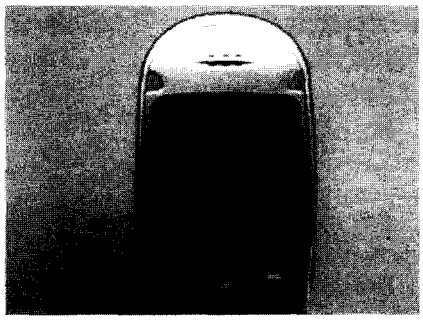


그림 10. 로더 및 동적 링커 동작 실험
Fig. 10. Operation experiment of a loader and a dynamic linker.

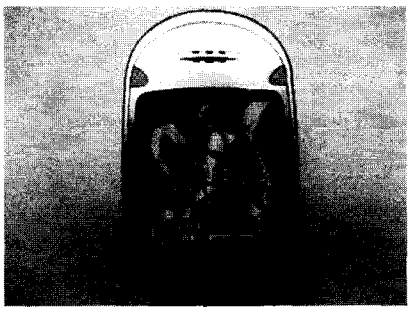
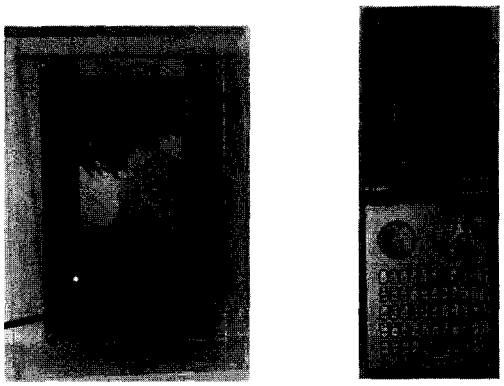


그림 11. Clet 예제
Fig. 11. Clet example on REX OS.

2. Qplus 지원 위피 플랫폼

본 논문에서는 개발 보드인 Kingfish 보드와 상용 단말기인 YOPY-3700 단말기에 Qplus와 위피 플랫폼을 탑재하여 검증된 Clet 프로그램을 실행시켜보았다. 그림 12에서 볼 수 있듯이 이 게임들이 Qplus 상에서도 제대로 동작함을 확인할 수 있었다.



(a) Kingfish 보드 (b) YOPY-3700 단말기

그림 12. Qplus 상에서의 Clet 게임
(a) Kingfish board (b) YOPY-3700 terminal
Fig. 12. Clet games on Qplus.

3. 윈도우즈 지원 에뮬레이터

(1) Clet 에뮬레이터

Clet 에뮬레이터의 동작 확인을 위해 테트리스 게임을 Clet으로 구현한 후 이를 실행시켜 보았다. 그림 13은 테트리스 게임이 정상적으로 동작함을 보여준다.

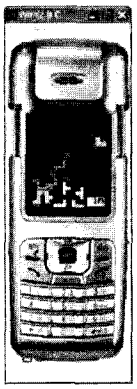
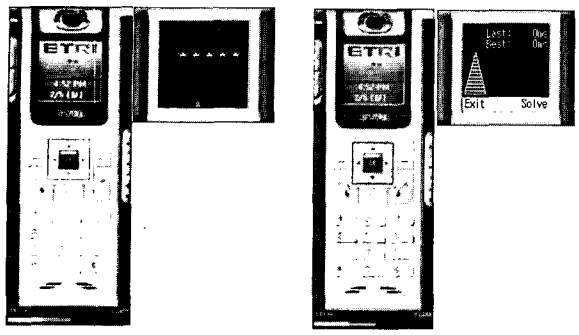


그림 13. Clet 예제
Fig. 13. Clet example.

(2) Jlet/MIDlet 에뮬레이터

위피-Java API는 MSF(Mobile Standard Foundation)와 MSP(Mobile Standard Profile)로 구성되어 있고, J2ME는 CLDC와 MIDP로 구성되어 있다. MSF는 CLDC와 대응이 되고 MSP는 MIDP와 대응된다고 할 수 있다. 데스크탑 PC에서는 J2ME 대신 J2SE (Java2 Standard Edition)을 사용해야 하므로 J2SE의 CDC(Connected Device Configuration)를 이용해 CLDC



(a) Jlet (Spcae Invader 게임) (b) MIDlet (Hanoi 게임)

그림 14. 게임 예제
(a) Jlet (Spcae Invader game) (b) MIDlet (Hanoi game)

Fig. 14. Game example.

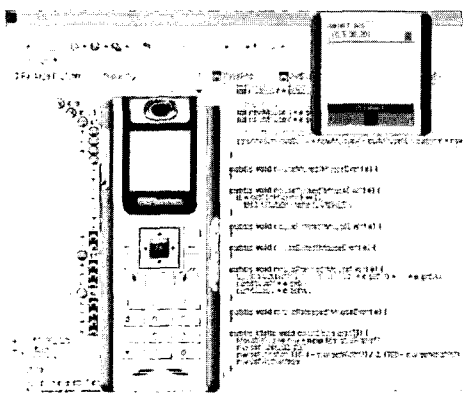


그림 15. 어클립스에 플러그인 된 Jlet/MIDlet 에뮬레이터

Fig. 15. Jlet/MIDlet emulator plugged-in the Eclipse.

를 구현했다. MSF와 CLDC는 많은 부분들이 일치하므로 CLDC를 이용해 MSF를 구현했으며 MSP와 MIDP는 새로 구현을 했다.

에뮬레이터의 동작 확인을 위해 검증된 MIDlet 게임을 구해서 Jlet으로 포팅한 후 실행시켜 보았다. 그림 14(a)는 Jlet으로 포팅된 Space Invader 라는 게임이 정상적으로 동작하고 있음을 보여준다. 그림 14(b) 는 Hanoi 라는 MIDlet 게임이 정상적으로 동작함을 보이고 있다.

그림 15는 이 에뮬레이터가 이클립스에 플러그인 되어 실행되고 있는 모습을 보여준다. 이클립스에서 Jlet/MIDlet 프로그램을 편집, 컴파일, 디버깅, 그리고 실행이 모두 가능하다.

4. Java2C 성능 실험

Java2C로 변환된 바이너리 코드가 얼마나 성능을 향상시켰는지 측정해 보았다. 그림 16은 변환되기 전의 클래스의 수행 시간과 변환된 후의 바이너리 코드의 수행 시간을 비교한 것이다.

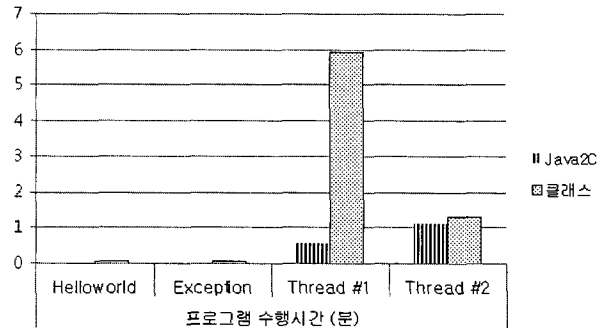


그림 16. Java2C 성능 시험

Fig. 16. Performance evaluation of the Java2C.

표 3. Java2C에 의해 변환된 코드의 크기

Table 3. Size of the code translated by the Java2C.

	Helloworld	Exception	Thread #1	Thread #2
Java2C	446,339	418,342	421,182	421,540
클래스	736	1,774	2,870	3,816

Helloworld는 화면에 간단히 "Hello World"라는 문자를 출력하는 프로그램이다. Exception은 네트워크와 관련된 예외를 발생시키는 프로그램이다. Thread #1은 쓰레드를 발생시켜 동기화를 수행하는 프로그램이다. Thread #2는 다수의 쓰레드를 생성해 이를 수행시키는 프로그램이다. 그림에서 볼 수 있듯이 Java2C에 의해 변환된 프로그램이 1.2배에서 32.5배까지 수행 시간이 빨랐다.

Java2C로 변환하면 수행 시간은 빨라지지만 코드량이 증가하는 문제점이 있다. 실제로 얼마나 증가하는지 표 3으로 나타냈다.

클래스의 파일 크기가 증가하더라도 변환된 코드의 양은 크게 변함이 없음을 알 수 있다. 하지만, 코드 크기가 클래스에 비해 절대적으로 크기 때문에 실제 상용화하여 서비스하기에는 많은 문제점을 일으킬 수 있다.

5. 검증

본 논문에서는 HAL, 실행 엔진, 그리고 라이브러리의 호환성을 검증하기 위해 검증 도구인 HCT와 PCT를 이용하였다[17]. PCT는 한국통신기술협회에서 품질인증을 받았을 뿐만 아니라 위피의 공식적인 인증툴로 인정되어 사용되고 있다. 또한, HCT는 PCT로서는 검증할 수 없는 오류 위치를 찾아내는 툴로서 HAL 개발 및 포팅에 많은 도움을 준다[18].

제안된 참조 구현은 대부분의 테스트케이스를 통과하였으나, 하드웨어가 지원 되지 않아 구현을 못한 함

수들은 실패했다. 그러나, 그런 하드웨어를 사용하지 않는 프로그램을 동작시키는 데는 아무런 문제가 없음을 알 수 있었다[19].

V. 결 론

본 논문에서는 다양한 플랫폼을 지원하는 위피 실행 엔진의 참조 구현을 제안하였다. 플랫폼에 따른 포팅을 최소화하기 위해 위피 라이브러리 수정 없이 HAL과 실행 엔진만을 수정하여 위피 플랫폼을 이식할 수 있도록 설계하였다. 현재 지원하는 플랫폼으로서는 REX OS, Qplus, 그리고 윈도우즈이다. REX OS를 지원하기 위해 실행 엔진의 내부 구조와 동작 방식을 제안하였으며, 링커와 로더의 구현 방법을 기술하였다. 그리고 이를 Qplus에 포팅해 봄으로써 다른 플랫폼에 쉽게 이식할 수 있는지 확인하였다. 그리고 윈도우즈 상에서 Jlet/MIDlet을 동작시켜 볼 수 있는 에뮬레이터를 제안하였다. 이 에뮬레이터의 주요 특징은 실행 엔진 대신 자바 가상 머신을 사용하고 이클립스에 플러그인 된다는 점이다. 또한 자바 프로그램을 바이너리 코드로 변환해 주는 Java2C 컴파일러를 제안하였다.

제안된 위피 플랫폼은 호환성 검증 도구인 PCT와 HCT의 테스트케이스를 대부분 통과하였으며 예제 프로그램을 성공적으로 실행해 봄으로써 구현을 검증할 수 있었다.

향후에는 Java2C 컴파일러의 최적화를 통해 바이너리 코드를 줄이고 실행 속도를 향상시키는 연구를 진행할 예정이다. 그리고 CDMA 망을 통해 실제로 위피 프로그램을 다운로드 받아 실행할 수 있는 환경을 구축할 계획이다.

참 고 문 헌

[1] 이상윤, 김선자, 김홍남, "무선인터넷 표준 플랫폼 위피 2.0", *TTA저널*, 통권92호, 97-102쪽, 2004년

[2] KWISF, Wireless Internet Platform for Interoperability, www.wipi.or.kr, 2004년

[3] 한국정보통신기술협회, 모바일 표준 플랫폼 규격 2.0, 표준번호 TTAS.KO-06.0036/R3, 2004.

[4] 이상윤, 김선자, 김홍남, "한국 무선 인터넷 표준 플랫폼(위피)의 표준화 현황 및 발전 전망", *한국정보과학회 학회지*, 제22권 제1호 통권 제176호, 16- 23쪽, 2004년

[5] JCP, www.jcp.org, 2007.

[6] B. G. Bae, W. S. Kim, J. G. Yun, C. H. Ahn, S. I. Lee, and K. I. Sohng, "Verification of WIPI-based T-DMB Platform for Interactive Mobile Multimedia Services," in Proc. of SPIE-IS&T Vol. 6074, pp. 60740W-1-60740W-8, 2006.

[7] B. G. Bae, W. S. Kim, C. H. Ahn, S. I. Lee, and K. I. Sohng, "Development of extended T-DMB platform based on WIPI for interactive mobile broadcasting services," Digest of Technical Papers-IEEE International Conference on Consumer Electronics 2006, pp. 259-260, 2006.

[8] N. M. Park, J. Kwak, S. J. Kim, D. H. Won, and H. W. Kim, "WIPI Mobile Platform with Secure Service for Mobile RFID Network Environment," *Lecture Notes in Computer Science* 3842 pp. 741-748, 2006.

[9] 유용덕, 박충범, 최훈, 김우식, "위피 응용프로그램 개발환경 설계 및 구현", *한국정보처리학회논문지*, 제12-C권 제5호(통권 제101호), 749-756쪽, 2005년

[10] 김연수, 강민철, 유용덕, 최훈, "무선인터넷 플랫폼에서 다중 응용프로그램 수행을 위한 스케쥴러 설계", 한국정보처리학회 2004년 추계학술대회, 제11권 제2호, 1759-1762쪽, 2004년

[11] John R. Levine, *Linkers and Loaders*, Morgan Kaufmann Publishers, 2000.

[12] ARM, *ARM ELF Specification Issue A-08*, 1999.

[13] 김유일, 이원재, 한환수, 이재호, 김선자, "이동단말기 환경에서 응용프로그램 로더와 동적 링커 개발", 한국정보과학회 2004년 춘계학술대회 제31권 제1호, 841-843쪽, 2004년

[14] Jaeho Lee, Sunja Kim, Sangyun Lee, Woosik Kim, and Hwangu Lee, "Implementation WIPI for Linux-based Smartphone," in Proc. of the 7th ICACT, pp. 692-696, 2005.

[15] Jaeho Lee, Sunja Kim, and Sangyun Lee, "Embedded Linux-based smartphone platform for sharing WIPI contents," IT-SOC 2004, pp. 50-553, 2004.

[16] Todd A. Proebsting, Gregg TownSend, Patrick Bridge, John H. Hartman, Tim Newsham, and Scott A. Watterson, "Toba: Java for applications: A way ahead of time(WAT) compiler", In Proceeding of the 3rd USENIX Conference on Object-Oriented Technologies and Systems (COOTS97), 1997.

[17] 이환구, 김우식, 이상윤, 이재호, 김선자, "위피 플랫폼 인증 툴킷 개발", 한국정보처리학회 추계학술발표대회, 제11권 제2호, 1539-1542쪽, 2004년

[18] 이상윤, 이환구, 최병욱, "모바일 표준 플랫폼(위피) 검증 도구 설계 및 개발", 한국정보처리학회, 제13-D권 제5호, 731-740쪽, 2006년

[19] S. Y. Lee and B. U. Choi, "Design and Implementation of WIPI Runtime Engine," 2006 International Conference on Hybrid Information Technology, IEEE Computer Society, pp. 19-23, 2006.

저 자 소 개

이 상 윤(정회원)
1994년 한양대학교 전자통신 공학과 학사 졸업.
1996년 한양대학교 전자통신 공학과 석사 졸업.
1999년~현재 한국전자통신연구원 임베디드S/W연구단 선임 연구원
<주관심분야 : 모바일 플랫폼, 임베디드 S/W>

최 병 욱(평생회원)
1973년 한양대학교 전자공학과 학사 졸업.
1978년 일본 경응의숙(KEIO)대학 전기공학과 석사 졸업
1981년 일본 경응의숙(KEIO)대학 전기공학과 박사 졸업.
1981년~현재 한양대학교 정보통신대학 정보통신학부 교수
<주관심분야 : 영상처리, 멀티미디어 공학>