

임베디드 시스템을 위한 HAL과 JAVA API 설계 및 구현*

이진관** · 이대형** · 정규철** · 이종찬** · 박기홍**

요 약

임베디드 시스템을 지원하는 JAVA API를 개발하면 임베디드 시스템을 개발자가 JAVA를 사용함으로써 코드의 재사용, 객체지향 개념의 시스템 개발들을 가능하게 한다. JAVA API를 구현하는데 있어 시스템에 의존적인 부분들이 존재하게 되는데, 이는 native 함수에서 구현한다. 본 논문에서는 리눅스 기반의 임베디드 시스템 디바이스를 제어하기 위한 JAVA API를 구현하는데 있어 플랫폼 독립적인 자바 부분과 의존적인 native 부분으로 나누어 설계 및 구현하였고, 임베디드 시스템 디바이스의 JAVA API를 통한 제어에 초점을 두었다.

Design and Implementation of HAL and Java API for Embedded System

Jin-Kwan Lee** · Dae-Hyung Lee**

Kyu-Cheol Jung** · Jong-Chan Lee** · Kihong Park**

ABSTRACT

The implement of JAVA API for supporting the embedded system is capable of developing the object oriented system with code reuse because the developer is able to use JAVA in the embedded system. The system dependant parts for implementing JAVA API are realized from a native function. In this paper, we divide JAVA API into the platform-independent JAVA part and platform-dependent native part for the implementation of JAVA API to manage the linux based embedded system devices, and focus on the control of the embedded system devices using JAVA API.

Key words : Embedded System, JAVA API, HAL

* 본 연구는 산업자원부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구 결과임.

** 군산대학교 컴퓨터정보과학과

1. 서 론

최근 들어 임베디드 시스템 기반의 홈네트워킹, 헬스케어, U-씨티 등 다양한 어플리케이션이 개발되고 있다. 이러한 임베디드 시스템에는 하드웨어와 소프트웨어가 결합된 고기능의 전자제어시스템을 내장하고 있다. 즉, 단순한 논리회로로 구성된 하드웨어가 아니라, 마이크로 프로세서 혹은 마이크로 컨트롤러를 내장하여 제작자가 의도한 다양한 기능을 수행하는 소프트웨어를 포함할 수 있는 시스템이다. 이렇게 임베디드 시스템이 소형화되고, 고성능으로 발전함에 따라 응용 범위도 넓어지고 있고, 많은 연구들이 진행되고 있다. 이러한 하드웨어적인 측면의 발전과 달리, 소프트웨어는 아직도 전문적인 기술과 하드웨어에 의존적인 프로그램을 벗어나지 못하고 있다. 즉, 임베디드 시스템 기반의 어플리케이션 개발은 대부분 C/C++로 이루어지고 있어 플랫폼에 의존적인 개발이 된다. 또한 객체지향 개념을 도입하는 것 또한 어려운 실정이다[7].

따라서, 임베디드 시스템을 지원하는 JAVA API를 개발하면 기존 JAVA 개발자와 현재 임베디드 시스템을 개발하는 개발자들도 JAVA를 사용함으로써 코드의 재사용, 객체지향 개념의 시스템 개발들을 가능하게 한다. JAVA API를 구현하는데 있어 시스템에 의존적인 부분들이 존재하게 되는데, 이는 native 함수에서 구현한다.

본 논문에서는 리눅스 기반의 임베디드 시스템 디바이스를 제어하기 위한 JAVA API를 구현하는데 있어 플랫폼 독립적인 자바 부분과 의존적인 native 부분으로 나누어 설계 및 구현하였고, 임베디드 시스템의 디바이스를 JAVA API를 통해 제어하는데 초점을 두었다. 플랫폼에 의존적인 native 부분은 JNI(Java Native Interface)를 통해 구현하였다. JNI란 자바와 자바 이외의 언어로 만들어진 어플리케이션이나 라이브러리가 상호 작동할 수 있도록 연결시켜주는 인터페이스로 자바에

서 지원하지 못하는 플랫폼 종속적인 기능들과 이미 다른 언어로 만들어진 어플리케이션이나 라이브러리를 사용할 수 있게 한다. JNI를 통한 자바 네이티브 프로그램은 자바가 아닌 다른 프로그래밍 언어들로 자바 객체를 생성하고, 검사하고, 갱신하는 것과 다른 언어에서 자바 메소드 호출을 가능하게 하고 클래스를 로드하고 클래스 정보를 얻을 수 있도록 하는 것 그리고 런타임 타입 체크를 수행하는 것 등을 가능하게 해준다[3, 4].

JNI와 어플리케이션 자바 API를 연결해주기 위해 HAL(Hardware Abstraction Layer)개념이 필요하다. HAL은 컴퓨터 운영체계가 자세한 하드웨어 계층이 아닌 일반적이거나, 추상적인 계층에서 하드웨어 서비스와 상호 작용할 수 있게 해주는 프로그램 계층이다.

본 논문에서는 임베디드 시스템의 플랫폼에 상관없이 어플리케이션에서 이용할 수 있는 HAL 관련 JAVA API를 추가하였다.

2. 관련 연구

2.1 임베디드 시스템

임베디드 시스템은 범용 컴퓨팅 시스템보다 요구사항이 까다롭다. 예를 들면, 휴대폰이 무겁고 수시로 충전해야 한다면 고유의 기능을 수행할 수 없고, 특정 계절에 자동차의 엔진제어가 불가능하다면 상품성이 전혀 없다. 임베디드 시스템의 주요한 특징을 정리하면 다음과 같다[7].

- 특정기능 수행 : 다양한 작업보다 특별한 작업을 수행하기 위해 설계된다.
- 강한 제약성 : 모바일 기기나 대형 시스템의 일부 기능을 수행하기 위해 소형이며, 전력 소모량이 작고, 실시간 반응을 요구한다.
- 제한된 시스템 자원 : 가격 및 휴대성과 연관되다보니 최적화된 상품을 위해 하드웨어와 소프

트웨어를 병행하여 개발한다.

- 호환성: 무수한 종류의 임베디드시스템에 최적화된 코드, 특히 어셈블리 언어의 사용으로 인하여 이식성이 매우 적다.
- 깔끔한 오류처리: 임베디드 시스템은 대부분 비전문가들이 사용하므로 가능한 사용자의 개입을 최소화해야 한다. 이를 위해 오류가 발생할 경우 깔끔한 처리가 요구된다.

2.2 자바 플랫폼

자바는 인터넷의 빠른 확산과 더불어 차세대 컴퓨팅 플랫폼으로 인식되고 있는 기술이다. 자바의 장점은 플랫폼 독립성과 연결성, 동적 다운로드 그리고 보안성이 있다.

기존의 프로그래밍 언어들은 해당 플랫폼에 의존적인 바이너리 파일들로 컴파일 해야만, 해당 시스템에서 실행할 수 있다. 그러나 자바는 플랫폼에 관계없이 단 한번의 컴파일을 통해서 얻은 바이트 코드로 모든 환경에서 동일하게 실행 될 수 있다. 이러한 특징은 서비스 개발자와 제공자, 그리고 사용자에게 편리함과 효율성을 제공해준다. 서비스 개발자는 모든 플랫폼에 맞추어 프로그램을 작성하지 않아도 되며, 서비스 제공자는 동일한 프로그램을 서로 다른 플랫폼들에 제공하기 위해 여러 종류의 바이너리 파일들을 제공해야 할 필요가 없다. 또한 사용자는 임의의 프로그램을 설치하고자 할 때, 자신의 플랫폼에 알맞도록 프로그램을 찾아야 할 필요가 없으며, 자바로 작성된 프로

그램은 무엇이든 필요할 때 네트워크를 통해 다운로드 받아서 사용할 수 있다.

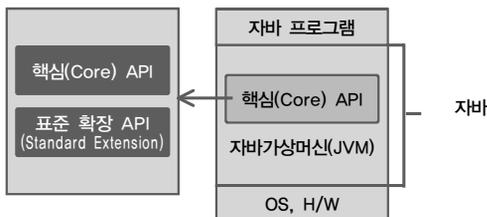
위의 특징들을 제공하기 위해 시스템 개발자들은 해당 시스템에 자바 플랫폼을 탑재해야 한다. 자바 플랫폼은 (그림 1)에서 볼 수 있듯이 자바 가상머신과 자바 API로 구성된다.

보통 자바 플랫폼은 자바 프로그램이 실행되는 하드웨어와 OS 및 소프트웨어 환경에 따라 나뉜다. 그러나 보통 하드웨어나 소프트웨어적인 것보다는 지원되는 API의 기능에 따라서 구별된다. 즉, 자바가상머신이 동작하는 곳이 브라우저인지 자바 OS인지 보다는 지원되는 API가 자바 프로그램에 있어서는 더 중요하다.

자바 제반 기술의 핵심인 자바 가상 머신은 일종의 소프트웨어 CPU로, 자바 번역기라고도 불린다. 자바 가상머신은 컴파일 된 자바 바이트 코드와 컴퓨터 운영 시스템 간의 번역기 역할을 하며, 자바의 플랫폼 독립성을 보장한다.

임베디드 시스템을 지원하는 자바가상머신으로 JamVM이 있다. JamVM은 아래와 같은 특성을 갖는다[1].

- 많은 최적화 단계를 지원: 단순 JIT에 가까운 성능을 낸다.
- 폴쓰래드 구현: 네이티브 쓰레딩(POSIX threads)을 지원한다.
- 클래스 로더를 지원
- 실행 엔진은 디스패치 오버헤드를 최소화하기 위하여 기본 교환 인터프리터(basic switched interpreter)와 쓰레드 인터프리터(threaded interpreter)를 지원한다.
- 쓰레드 서스펜션(Thread suspension)은 서스펜드 레이턴시(suspend latency)와 성능 향상을 위해 시그널들을 사용한다.
- 가비지 컬렉터안에서 풀 오브젝트 파이널라이즈(Full object finalisation)를 제공한다.
- 가비지 컬렉터는 자신의 쓰레드에서 동기나 비동기 방식으로 실행될 수 있다.



(그림 1) 자바 플랫폼

- 표준 라이브러리를 위한 JNI와 동적 로딩을 지원한다.
- JNI 오버헤드 없이 내부 네이티브 메소드들을 위하여 자신의 라이트웨이트 네이티브 인터페이스를 사용한다.
- JamVM은 C로 작성되어 있으며, 적은양의 플랫폼 종속적인 어셈블러로 되어 있다. 그리고 다른 아키텍처로 쉽게 포팅될 수 있다.

2.3 관련기술 분석

현재 급부상 중인 임베디드 시스템을 객체 기반 자바로 구현하는 것은 플랫폼 독립성, 네트워크 연결성, 안전성 등의 자바의 장점을 제공할 수 있다. 기존 임베디드 시스템 개발은 임베디드 리눅스 상에서 GTK나 C/C++을 이용한다. 그러므로 기술 구현의 난이도가 높을 뿐만 아니라, 코드의 재사용성 또한 떨어진다.

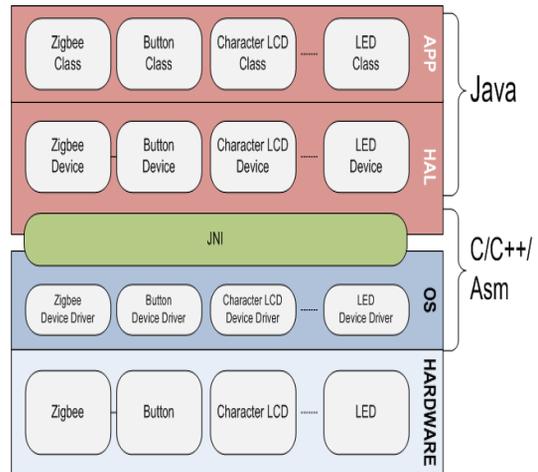
본 논문의 실행환경은 임베디드 리눅스 상의 자바 플랫폼 환경이다. JVM은 JamVM을 이용하였고, 자바 API 개발은 J2SDK 1.4버전을 이용하였다. 자바 API는 자바 플랫폼의 구성요소이므로, 다른 요소인 JVM과 밀접한 연관이 있다. 이들 사이의 연결자 역할은 JNI를 사용함으로써 가능하다. JNI를 통하여, API들은 JVM의 특성을 이용하고 원하는 값을 얻으며, C 라이브러리와 리눅스 상에서 제공하는 시스템 호출을 이용할 수 있다.

3. 자바 API 설계 및 구현

3.1 임베디드 시스템을 위한 자바 API와 고려사항

본 논문에서 제안한 자바 API의 구조는 두 부분으로 구성된다((그림 2) 참조). 하나는 API를 구성하는 인터페이스와 클래스들을 제공하는 자바 부분(*.java)이고 다른 하나는 각각의 디바이스들

을 제어하기 위한 플랫폼 의존적 시스템 호출을 위해 JNI 형식을 사용하여 C나 C++로 구현한 네이티브 라이브러리이다. HAL은 하위단인 디바이스와 상위단인 자바 어플리케이션의 인터페이스를 수행함으로써 상위 계층이 하위 계층과 독립적인 프로그래밍이 가능하도록 지원한다.



(그림 2) JAVA API 구조

3.2 JNI와 디바이스 드라이버

리눅스 기반의 디바이스를 제어하기 위하여 디바이스를 커널에 포팅시키기 위한 드라이버를 제작해야 한다. 이를 위하여 디바이스 드라이버를

```
static const char dev[] = "/dev/CC2420";
JNIEXPORT jboolean JNICALL Java_kr_ac_kun-
san_usn_de-
vices_hybus_1fpga_CC2420_nativeInitialize(...){...}
JNIEXPORT jbyteArray JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_CC2420_nativeReceive-
Packet(...){...}
JNIEXPORT jint JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_CC2420_nativeSendPacket(...){...}
JNIEXPORT jboolean JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_CC2420_nativeRe-
lease(...){...}
```

(그림 3) CC2420 관련 JNI

모듈방식으로 제공함으로써 커널의 유연한 확장성을 제공할 필요가 있다[5].

본 논문에서는 크로스 컴파일러로 임베디드용 CLCD, CC2420, 버튼 드라이버를 제작하였다. CLCD를 통해 문자열을 출력하고((그림 3) 참조), CC2420을 통해 지그비 통신을 가능케 하였다((그림 4) 참조). 또한 버튼 클릭 시, 특정 기능을 처리할 수 있도록 버튼 이벤트를 추가하였다.

```
static char lcdDev[] = "/dev/clcd";
JNIEXPORT jboolean JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_LCD_11I_nativeInitialize
(...){...}
JNIEXPORT void JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_LCD_11I_nativeMove-
Cursor (...){...}
JNIEXPORT void JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_LCD_11I_nativeDisplay-
Control(...){...}
JNIEXPORT void JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_LCD_11I_nativeClear
(...){...}
JNIEXPORT void JNICALL Java_kr_ac_kun-
san_usn_devices_hybus_1fpga_LCD_11I_nativeWrite
(...){...}
```

(그림 4) CLCD 관련 JNI

3.3 HAL 관련 API의 구현

어플리케이션 자바 API에서 하드웨어에 종속적인 JNI에 상관없이 JNI를 이용하기 위해 HAL 자바 API를 설계하였다.

- IDevice.class

디바이스 장치를 나타내는 클래스로 인터페이스로 구현하였다. 모든 디바이스는 IDevice를 구현해야 한다.

- AbstractCharacterLCDDevice.class

AbstractCharacterLCDDevice 클래스는 칩셋 이름, CLCD의 라인수, 라인 당 표시할 수 있는 문자 수 지정 속성과 CLCD에 출력할 수 기능을 갖는다.

- IPacket.class

IPacket 클래스는 패킷의 사이즈를 반환하는 메서드를 제공하는 인터페이스이다. 통신 디바이스 관련 클래스는 IPacket 클래스를 상속 받아야 하며, 어플리케이션에서 IPacket 클래스를 구현하여 필요한 데이터를 추출해야 한다.

- ICommunicationDevice.class

통신 디바이스는 OSI 7계층의 데이터 링크 계층과 동일한 기능을 수행한다. Packet 단위로 통신을 함으로 IPacket이라는 인터페이스를 사용하여 클래스를 운용한다. 또한 통신 디바이스 클래스는 패킷의 송수신을 여부를 파악하기 위하여 Runnable 인터페이스를 상속하여 이를 감시토록 한다.

- ICommand.class

디바이스 제어를 위한 명령 인터페이스로 디바이스의 기능을 실행하는 메서드를 제공한다. 디바이스의 제어 명령을 하나의 인터페이스로 묶기 위한 인터페이스로 각각의 제어 명령 하나하나를 이 인터페이스를 통하여 구현한다.

- ITranslator.class

하위 레이어인 데이터통신계층에 쓰이는 Packet과 그 위 계층에서 사용하는 명령 사이를 변환시켜주는 기능을 정의한 인터페이스이다. ICommand를 IPacket형태로 변환시키는 메서드가 제공된다.

3.4 문자 디스플레이(CLCD) 관련 API의 구현

CLCD는 제조사마다 다른 명령어 셋을 사용하여 디바이스를 제어하므로 디바이스가 변경될 때마다 서로 다른 명령어셋을 사용해야 한다. 하지만 명령어셋을 따로 분리하여 인터페이스를 만들 경우, 이중의 디바이스를 위한 JNI 작업 시간이 상당히 단축될 수 있다. 또한 AbstractCharacterLCD 형식에 맞추어 구현한다면 서로 다른 디바이스를 사용

하여 프로그래밍할지라도 장치관련 클래스(예, LCD_II)를 인스턴스화하여 CharacterLCD 생성 시 인자(Parameter)로 넘겨주어 응용프로그램의 기능에 전혀 영향을 미치지 않고 그대로 사용할 수 있게 된다.

○ LCD_II.class

AbstractCharacterLCDDevice 클래스를 상속받아 어플리케이션에 사용할 수 있는 클래스로 CLCD의 On/Off 및 초기화, 출력 관련 메소드를 제공한다.

○ ILCD_IICommandSet.class

LCD II Command Set 관련 상수를 정의한 인터페이스이다[2].

<표 1> LCD Command Set

상수명	값	명령
CLEAR_SCREEN	0x01	화면지우기
CURSOR_HOME	0x02	홈으로 돌림
ENTRY_MODE	0x04	
DISPLAY_OFF	0x08	Display On/Off
DISPLAY_ON	0x0C	
CURSOR_ON	0x0E	커서 On/Off
CURSOR_OFF	0x0C	
BLINK_ON	0x0F	Blink On/Off
BLINK_OFF	0x0E	
CURSOR_SHIFT_LEFT	0x0010	커서이동
CURSOR_SHIFT_RIGHT	0x0014	
DISPLAY_SHIFT_LEFT	0x0018	Display 이동
DISPLAY_SHIFT_RIGHT	0x001C	
WRITE_CONTENTS	0x0200	출력

○ CharacterLCD.class

CharacterLCD 클래스는 총 슬롯 개수, 슬롯 추가 삭제 시 사용되는 카운터, 해시 테이블을 이용한 멤버 그룹 저장소, AbstractCharacterLCD Device 객체를 속성으로 갖는다. 모든 콘텐츠는 member

형태로 존재하며 memberGroup에 속한다. member, memberGroup의 추가/삭제 메소드, member의 CLCD 출력 메소드를 제공한다.

○ IMemberContent.class

member에 포함되는 콘텐츠 인터페이스로 문자열이 저장된다.

○ CharacterLCDMember.class

CharacterLCDMember 클래스는 CLCD에 들어갈 문자열을 가지는 클래스로 시작 컬럼, IMemberContent 인스턴스를 속성으로 갖는다.

○ CharacterLCDMemberGroup.class

CharacterLCDMemberGroup는 CLCD의 라인수와 CharacterLCDMember 인스턴스를 갖는다.

3.5 지그비 관련 API의 구현

이 클래스를 구현하기 위해서는 먼저 지그비 기반의 패킷 전송을 위한 패킷 분석 방법과 상위단인 자바에서 이를 제어하기 위한 최소한의 JNI로직이 필요하다. 또한 상호 호환성을 갖는 통신을 위하여, 서로 다른 지그비 통신 디바이스간의 표준에 근거하여 작성한다[6].

○ CC2420.class

CC2420 클래스는 ICommunicationDevice 클래스를 구현해야 한다. cc2420 지그비 컨트롤러를 초기화하고, JNI를 통하여 데이터를 송수신 할 수 있는 메소드를 제공한다.

3.6 버튼 이벤트 관련 API의 구현

버튼 장치의 경우는 OS에서 버튼 클릭 시, Event를 발생시켜 자바의 다른 EventListener와 동일하게 구동해야 하지만 디바이스 특성상 완전한 EventListener

형태로 구현하기는 어렵다. 따라서 특정 읍저버 패턴을 사용하여 주체(Subject)역할을 하는 ButtonHook 클래스를 구축하였고, 실제 디바이스를 감시하는 Poll Thread를 사용하여 디바이스 감지 Button Event를 발생시킨다.

○ ButtonEvent.class

버튼의 클릭 시, 발생하는 이벤트를 나타내는 클래스로 EventObject를 상속받는다.

○ ButtonEventListener.class

버튼 이벤트를 수신하는 버튼 이벤트 리스너 인터페이스로 EventListener를 상속 받는다. 버튼 클릭 시, 수행되는 메서드인 buttonPressed()와 버튼의 클릭 해제 시 수행되는 메서드인 buttonReleased()가 제공된다.

○ ButtonHook.class

ButtonHook 클래스는 버튼의 클릭 여부 감시자로 버튼 클릭을 감지하고자 하는 인스턴스이다. 이 클래스를 생성 후 addEventListener로 자신을 추가해야 한다. 그렇지 않을 경우 이벤트 메시지를 탐지하지 못한다.

○ PollThread.class

PollThread 클래스는 Thread를 통한 지속적인 버튼 감시를 통하여 버튼 상태를 체크하는 클래스이다. 장치를 감시하기 위한 JNI 루틴이 포함된 메서드로 다른 장치를 사용할 경우 이 클래스를 참조해야 한다.

4. 시스템 실험

실험 환경은 호스트 컴퓨터와 타겟 컴퓨터로 구성된다. 호스트 컴퓨터는 RedHat FC4 운영체제에서 gcc와 J2SDK 1.4버전의 개발툴을 이용하였다.

타겟 컴퓨터는 하이버스의 XHyper270-TKU를 이용하여 임베디드 시스템용 JAVA API를 개발하였고, 임베디드 시스템의 JVM은 JamVM을 이용하였다. JamVM은 GNU Classpath 자바 클래스 라이브러리를 이용하게 설계되었다[1].

우선 이들을 통합하기 전에 각각의 요소의 개별적인 동작을 테스트해야 하므로, 구현한 자바 API의 테스트를 위해 다음과 같은 테스트 환경을 설정하였다.

○ 타겟 시스템 : XHyper270-TKU

- 커널 : Linux 2.6
- JVM : JamVM 1.4.5
- Classpath : GNU Classpath 0.9.3

○ 호스트 컴퓨터 : IBM 호환 PC

- OS : Redhat FC4
- 자바 컴파일러 : J2SDK 1.4
- IDE : Eclipse 3.3

자바 API를 구현하였으므로, 이들 API가 올바르게 동작하는지를 알아보기 위하여 간단한 임베디드 어플리케이션으로 테스트를 한다.

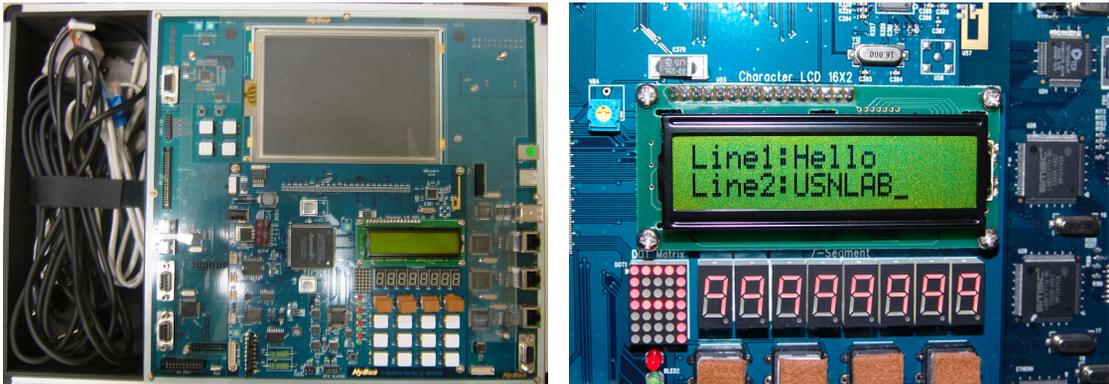
(그림 5)에서 CLCDTest는 테스트용 클래스로 main 메소드만을 가진다. ①은 HAL 관련 자바 API인 AbstractCharacterLCDDevice 클래스를 상속받아 어플리케이션에 사용할 수 있는 네이티브 LCD_II 객체를 생성한다. LCD_II 클래스는 CLCD의 On/Off 및 초기화, 출력 관련 메소드를 제공한다. ②는 네이티브 LCD 객체를 인수로 하여 어플리케이션 영역에서 사용할 수 있는 CLCD 객체를 생성한다. ③은 CLCD 화면에 출력할 2줄의 콘텐츠를 생성한다. ④⑤⑥에서는 멤버그룹을 등록하고, 생성된 콘텐츠를 멤버그룹에 등록한다. ⑦은 CLCD에 출력하는 메소드이다.

(그림 6)은 XHyper270TKU에 임베디드용 자바 API를 이용하여 제작한 CLCDTest를 실행한 화면이다.

```

public class CLCDTest {
    public static void main(String[] args){
        AbstractCharacterLCDDevice nativeLCD = new LCD_II( ); ---①
        CharacterLCD clcd = new CharacterLCD(2, nativeLCD); ---②
        IMemberContent line1Content = new LineContent("Line1: Hello World"); ---③
        IMemberContent line2Content = new LineContent("Line2: USNLAB");
        CharacterLCDMemberGroup Line1 = new CharacterLCDMemberGroup(0, 16, " "); ---④
        CharacterLCDMemberGroup Line2 = new CharacterLCDMemberGroup(1, 16, " ");
        Line1.addMember(new CharacterLCDMember(0,line1Content)); ---⑤
        Line2.addMember(new CharacterLCDMember(0, line2Content));
        clcd.addMemberGroup(Line1); ---⑥
        clcd.addMemberGroup(Line2);
        clcd.displayAll( ); ---⑦
        try { Thread.sleep(1000); }
        catch (InterruptedException e) { e.printStackTrace( ); }
        clcd.cursorOnOff(true);
    }
}
    
```

(그림 5) CLCDTest.java



(그림 6) 임베디드 시스템 결과 화면

본 논문에서 연구한 임베디드용 자바 API를 활용하면 다양한 임베디드용 어플리케이션을 개발할 수 있다.

5. 결론 및 향후 과제

본 논문에서 구현한 자바 API 들은 네트워크 연결성, 플랫폼 독립성, 클래스 동적로딩, 안전성 등의 자바 장점을 임베디드 시스템에 제공함으로써 홈 네트워크, U-씨티, 헬스케어 구축에 큰 기여를 할 것으로 예측된다.

본 연구에서 구현한 JAVA API는 임베디드 시스템 디바이스 중 주요 기능 만을 구현 한 것으로서 추후 다양한 임베디드 시스템 디바이스를 지원할 수 있는 JNI와 HAL을 추가할 필요가 있다. 위에서 언급한 바와 같이 버튼 디바이스 경우 상위 단계인 자바에서 직접 디바이스를 감시하는 클래스를 인스턴스화하여 감지하지만, 추후 독자적인 태몬 형태의 JNI기반 네이티브 라이브러리를 생성하여 ButtonEventListener가 PollThread와 ButtonHook의 지원 없이도 ButtonEvent가 발생하도록 하려한다.

참 고 문 헌

- [1] <http://jamvm.sourceforge.net/>.
- [2] http://www.stanford.edu/class/ee281/handouts/hd44780_lcd_controller_datasheet.pdf.
- [3] <http://java.sun.com/javase/6/docs/tech-notes/guides/jni/spec/jniTOC.html>.
- [4] Java(TM) Native Interface: Programmer's Guide and Specification by Sheng Liang, Sun Microsystems.
- [5] <http://www.xml.com/ldd/chapter/book/>.
- [6] <http://www.waset.org/pwaset/v26/v26-66.pdf>.
- [7] 남상엽, PXA270을 이용한 임베디드 시스템 구조 및 응용, 상학당, 2007.
- [8] Bill Venner, Inside the Java Virtual Machine 2nd, McGraw-Hill, 1997.



이진관

- 1996년 군산대학교 컴퓨터학과 (이학사)
- 2002년 군산대학교 컴퓨터학과 (이학석사)
- 2007년 군산대학교 컴퓨터학과 (이학박사)

2006년~현재 군산대학교 컴퓨터학과 강사



이대형

- 1996년 군산대학교 컴퓨터학과 (이학사)
- 2007년 군산대학교 컴퓨터정보학과 (이학석사)
- 2008년 군산대학교 컴퓨터정보학과 (박사 과정)



정규철

- 1996년 군산대학교 컴퓨터학과 (이학사)
- 1999년 군산대학교 컴퓨터학과 (이학석사)
- 2006년 군산대학교 컴퓨터학과 (이학박사)

1999년~현재 군산대학교 컴퓨터학과 강사



이종찬

- 1994년 군산대학교 컴퓨터학과 (이학사)
- 1996년 숭실대학교 컴퓨터학과 (이학석사)
- 2000년 숭실대학교 컴퓨터학과 (공학박사)

2000년~2005년 한국전자통신연구원 선임연구원

2005년~현재 군산대학교 컴퓨터학과 조교수



박기흥

- 1986년 숭실대학교 전자계산학과 (이학사)
- 1986년 숭실대학교 전자계산학과 (공학석사)
- 1995년 일본 토쿠시마대학교 지능정보학과(공학박사)

1997년~1998년 영국 Middlesex Univ 객원 교수

1987년~현재 군산대학교 컴퓨터학과 교수

2004년~현재 NURi 사업 텔레메틱스 인력양성 사업단 (군산대) 단장