Android Real Target Porting Application Software Development*

Hong, Seon Hack** · Nam Gung, Il Joo***

안드로이드 리얼 타깃 포팅 응용 소프트웨어 개발

홍 선 학 · 남궁일주

<Abstract>

In this paper, we implemented the Android NDK porting application with Eclipse(JDK) ADT and TinyOS 2.0. TinyOS and Cygwin are component based embedded system and an open-source basis for interfacing with sensor application from H-mote. Cygwin is a collection of tools for using the Linux environment for commercially released with x86 32 bit and 64 bit versions of Windows. TinyOS-2. x is a component based embedded OS by UC Berkeley and is an open-source OS designed for interfacing the sensor application with specific C-language.

The results of Android porting experiment are described to show the improvement of sensor interfacing functionality under the PXA320 embedded RTOS platform. We will further more develop the software programming of Android porting under Embedded platform and enhance the functionality of the Android SDK with mobile gaming and kernel programming under sensor interfacing activity.

Key words : Froyo, Kernel, JAVA NDK, Android Platform, Cygwin

Ⅰ. Introduction

Now is an exciting time for us like embedded software developers. Mobile devices which are operated with Android have never been more popular, and powerful functions for them are an overwhelmingly popular choice for consumers. Stylish and versatile devices packing hardware features like GPS, accelerometers, and touch screens, combined with fixed-rate, provide an enticing platform upon which to create innovative mobile applications[1].

With much existing mobile development built on proprietary operating systems that restrict the development and deployment of third-party applications, Android offers an open alternative.

Without artificial barriers, like us as Android users are free to write applications that take full advantage of increasingly powerful mobile hardware and distribute them in an open society.

* 본 연구는 2010학년도 서일대학 교내학술연구비지원으로 수행되었음.

** 서일대학 컴퓨터전자과 교수

*** 서일대학 인터넷정보과 겸임교수

Built on an open source framework, and featuring powerful SDK libraries and an open philosophy, Android has opened mobile software development to thousands of developers who haven't had access to tools for building mobile applications. Experienced mobile software developers can now expand into the Android platform, leveraging the unique features to enhance existing functions or crate innovative new ones[2-3].

Ⅱ. Basic Theory

This chapter provides a fundamental overview of understanding the Android Software Stack. In Android, native third-party applications are written with the same APIs and executed on the same run time. These APIs feature hardware access, location-based services, map-based activities, inter-application communication, and 2D and 3D graphics.

Historically, like us developers, generally coding in low-level C or C++, have needed to understand the specific hardware they were coding for, generally a single device or possibly a range of devices from a single manufacturer. As hardware technology and mobile internet access has advanced, these closed approaches have become outmoded.

More recently, the outstanding advance in mobile development was the introduction of Java-hosted MIDlets. MIDlets are executed on a Java virtual machine, a process that abstracts the underlying hardware and lets developers create applications that run on the wide variety of devices that supports the Java run time. Unfortunately, this convenience comes at the price of restricted access to the device hardware. In this field, it was considered normal for third-party applications to receive different hardware access and execution rights from those given to native applications written by the phone makers, with MIDlets often receiving few of either.

The introduction of a Java MIDlets expanded developer's audiences, but the lack of low-level hardware access and sandboxed execution meant that most applications are regular desktop programs or web sites designed to render on a smaller screen, and do not take advantage of the inherent mobility of the handheld platform.

Android offers new possibilities for mobile applications by offering an open development environment built on an open-source Linux kernel. Hardware access is available to all applications through a series of APIs libraries, and application interaction, while carefully controlled, is fully supported. In Android, all applications have equal standing. Third-party and native Android applications are written with the same APIs and are executed on the same run time. Users can remove and replace any native application with a third-party developer alternative; even the dialer and home screen can be replaced.

2.1 Android Stack

In the proposed architecture of Fig 1, a Linux kernel and a collection of C/C++ libraries are exposed through an application framework that provides services for, and management of, the run time and applications.

Linux kernel is core services (including hardware drivers, process and memory management, security,

network, and power management) which are handled by a Linux 2.6 kernel. The kernel also provides an abstraction layer between the hardware and the remainder of the stack.

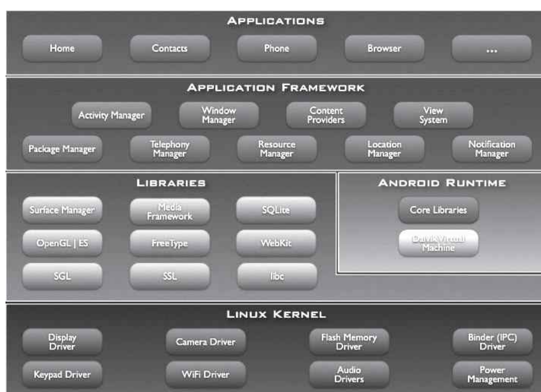

Fig 1. Android Software Stack

Android contains various C/C++ core libraries (including a media library, surface manager, Graphics libraries, SQLite database, and SSL and WebKit for integrated web browser and Internet security) such as libc and SSL which are running on top of the kernel.

Android run time is what makes an Android phone rather than a mobile Linux implementation. Including the core libraries and the Dalvik virtual machine, the Android run time is the engine that powers your applications and, along with the libraries, forms the basis for the application framework.

Especially, the core libraries provide most of the functionality available in the core Java libraries as well as the Android-specific libraries. Dalvik is a register-based virtual machine that's been optimized to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.

The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources.

All applications, both native and third-party, are built on the application layer by means of the same API libraries. The application layer runs within the Android run time, using the classes and services made available from the application framework[4-6].

2.2 Dalvik Virtual Machine

There are very outstanding three items in the Android architecture such as ; Dalvik virtual machine, Application and libraries. One of the key elements of Android is the Dalvik virtual machine. Rather than use a traditional Java virtual machine(VM) such as Java ME (Java Mobile Edition), Android uses its own custom VM designed to ensure that multiple instances on a single device.

The Dalvik VM uses the device's underlying Linux kernel to handle low-level functionality including security, threading, and process and memory management. It's also possible to write C/C++ applications that run directly on the underlying Linux OS. If the speed and efficiency of C/C++ is required for your applications, Android now provides a Native Development Kit(NDK, r5c). The NDK is designed to enable you to create C++ libraries using the libc and libm libraries, along with native access to OpenGL.

All Android hardware and system service access is managed using Dalvik as a middle tier. By using a VM to host application execution, developers have an abstraction layer that ensures they never have to worry

about a particular hardware implementation. The Dalvik VM executes Dalvik executable files, a format optimized to ensure minimal memory footprint. We create. dex executables by transforming Java language compiled classes using the tools supplied within the SDK.

2.3 Application Architecture

Android's architecture encourages the concepts of component reuses, enabling you to publish and share Activities, Services, and data with other applications, with access managed by the security restrictions we put in place. The same mechanism that lets you produce a replacement contact manager or phone dialer can let us expose our application components to let other developers create new UI front ends and functionality extension, or otherwise build on them.

There are several application services in the application services such as ; Activity Manager which controls the life cycle of our Activities, Views which are used to construct the user interfaces for our Activities, Notification Manager which provides a consistent and nonintrusive mechanism for signaling users, Content Providers which let our applications share data, and Resource manager which supports non-code resources like string and graphics to be externalized.

In the final stage, Android offers a number of APIs for developing our applications. Rather than list them all here, we are referred to the documentation at the web site of the android developing reference package which gives a complete list of packages included in the Android SDK. Although Android is intended to target a wide range of mobile hardware, so be aware that the suitability and implementation of some of the advanced or optional APIs may vary depending on the host devices[7-8].

Ⅲ. JNI Development Enviromnet

In this paper, We used the JNI(Java Native Interface) which implement the C/C++ language code under Java environment. And also we need the Android NDK(r5c) and ADT(11.0.0) plugin for Eclipse.

3.1 Android NDK

The Android NDK is a toolset that lets us embed components that make use of native code in our applications.

Android applications run in the Dalvik VM. The NDK allows us to implement parts of our applications using native-code languages such as C and C++. This can provide benefits to certain classes of applications, in the form of reuse of existing code and in some cases increased speed.

The latest release of the NDK supports ARMv5TE and ARMv7-A instruction sets. ARMv5TE machine code will run on all ARM-based Android devices. ARMv7-A will run only on devices such as the Verizon Droid or Google Nexus One that have a compatible CPU. The main difference between the two instruction sets is that ARMv7-A supports hardware FPU, Thumb-2, and NEON instructions.

The NDK provides stable headers for libc (the C

library), libm (the Math library), OpenGL ES (3D graphics library), the JNI interface, and other libraries. The NDK will not benefit most applications, so we need to balance its benefits against its drawbacks; notably, using native code does not result in an automatic performance increase, but always increases application complexity. And therefore, we should only use native code if it is essential to our application, not just because we prefer to program in C/C++.

The NDK includes a set of cross-toolchains (compilers, linkers, etc) that can generate native ARM binaries on Linux, OS X, and Windows(with Cygwin) platforms[9-10].

3.2 ADT Plugin for Eclipse

Android Development Tools (ADT) is a plugin for the Eclipse IDE that is designed to give us a powerful, integrated environment in which to build Android applications.

ADT extends the capabilities of Eclipse to let us quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug our applications using the Android SDK tools, and even export signed (or unsigned). apk files in order to distribute your application.

Developing in Eclipse with ADT is highly recommended and is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives you an incredible boost in developing Android applications. Figure 2 displays the layout of overall system.

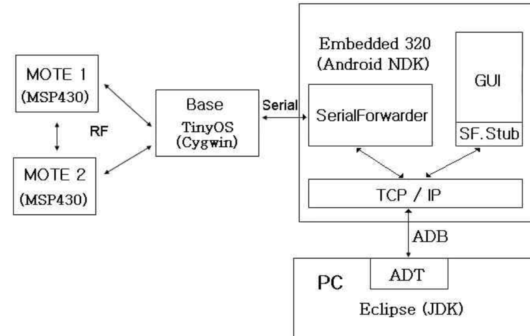

Fig 2. Layout of Overall System

3.3 Communication Interface with Cygwin

In this paper, we could use a number of interfaces to abstract the underlying communications services and a number of components that provide these interfaces. All of these interfaces and components use a common message buffer abstraction, called message_t, which is implemented as a nesC struct. There are a number of interfaces and components that use message_t as the underlying data structure.

- Packet : provides the basic accessors for message_t abstract data type. This interface provides commands for clearing a message's content's, getting its payload length, and getting a pointer to its payload area.
- Send : provides the basic address-free message sending interface. This interface provides commands and cancelling a pending message send, and an event to indicate whether a message was sent successfully or not.
- Receive : provides the basic message reception interface. This interface provides an event for receiving messages.

Since it is very common to have multiple services

using the same radio to communicate, TinyOS provide the Active Message(AM) layer to multiplex access to the radio. Am type are similar in function to the Ethernet frame type filed, IP protocol filed, and the UDP port in that all of them are used to multiplex access to a communication service. AM packets also include a destination filed, which stores an "AM address" to address packets to particular motes[11-13].

In this paper, We have defined a message type for our application. We want a timer-driven system in which every firing of the timer results in (i) incrementing a counter, (ii) displaying a number of bits of the counter, and (iii) transmitting the node's id and counter value over the radio. To implement this program, we follow a number of simple steps.

First, we need to identify the interfaces that provide access to the radio and allow us to manipulate the message_t type. Second, we must update the module block in the application program by adding uses statements for the interfaces we need. Third, we need to declare new variables and add any initialization and start/stop code that is needed by the interfaces and components. Fourth, we must add any calls to the component interface we need for our application. Fifth, we need to implement any events specified in the interfaces we plan on using. Sixth, the application configuration interface must be updated by adding a components statement for each component we sue that provides one of the interface we choose earlier. Finally, we need to wire the interface used by the application to the components which provide those interfaces [14-17].

Ⅳ. Embedded Controller Design

In this chapter, we describe the characteristics of embedded RTOS controller, including MSP430CPU, cc2420-RF chip and nesC.

4.1 Mixed Signal Controller

The MSP430 has a 16-bit RISC architecture that is highly transparent to the application. All operations, other than program-flow instructions, are performed as register operations in conjunction with seven addressing modes for source operand and four addressing modes for destination operand. The CPU is integrated with 16 registers that provide reduced instruction execution time. The register-to-register operation execution time is one cycle of the clock.

The digitally controlled oscillator(DOC) allows wake-up from low-power modes to active mode in less than $6\mu s$. This device has two built-in 16-bit timers, a fast 12-bit A/D converter, dual 12-bit D/A converter, one or two universal serial synchronous/ asynchronous communication interfaces (USART), I^2C, DMA, and 48 I/O pins, In addition, it offers extended RAM addressing for memory-intensive applications and large C-stack requirements.

The RF chip, cc2420 has high performance and low power 8051 micro-controller core, and 2.4GHz IEEE 802.1.5.4 compliant RF transceiver. This device has 32, 64, or 128 KB in-system programmable flash, and 8KB SRAM, 4KB with data retention in all power modes. We use only a single crystal needed for mesh network systems and two powerful USARTs with support for

several serial protocols[18-19].

4.2 NesC Language

In this paper, we implement controller with nesC which is a new language for programming structured component-based applications. The nesC language is primarily intended for embedded systems such as sensor networks. nesC has a C-like syntax, but supports the TinyOS concurrency model, as well as mechanisms for structuring, naming, and linking together software components into robust network embedded systems. The nesC has a several features. First, nesC applications are built out of components with well-defined, bidirectional interfaces. Second, nesC defines a concurrency model, based on tasks and hardware event handlers, and detects data races at compile time.

There are two types of components in nesC: modules and configurations. Modules provide application code, implementing one or more interface. Configurations are used to assemble other components together, connecting interfaces used by components to interfaces provided by others. This is called wiring. Every nesC application is described by a top-level configuration that wires together the components inside. In nesC, downcalls are generally commands, while upcalls are events. An interface specifies both sides of this relationship. Figure 3 represents the structure of TinyOS Application Layer.

4.3 Sensor Interfaces and Applications

We implemented the embedded sensor interfaces with TinyOS. Because sensor nodes have a broad

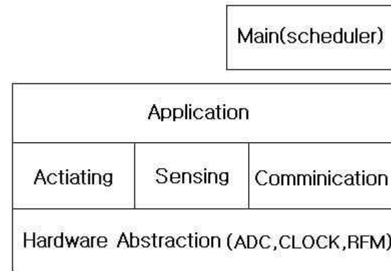

Fig 3. TinyOS Application Layer

range of hardware capabilities, one of the goals of TinyOS is to have a flexible hardware/software boundary. Our experiments that encrypts packets should be able to interchangeably use hardware or software implementations. Hardware, however, is almost always split-phase rather than blocking.

It is split-phase in that completion of a request is a callback. In our experiment, to acquire a sensor reading with an A/D converter, software writes to a few configuration registers to start a sample. When the ADC sample completes, the hardware issues an interrupt, and the software reads the value out of a data register[20-21].

The problem with threads in embedded systems is that they require a good deal of RAM. Each thread has its own private stack which has to be stored when a thread is waiting or idle. When a thread samples a blocking ADC and is put on the wait queue, the memory of its entire call stack has to remain untouched so that when it resumes it can continue execution. TinyOS therefore takes the opposite approach. Rather than make everything synchronous through threads, operations that are split-phase in software as well. This means that many common operations, such as sampling sensors and sending packets, are split-phase.

An important characteristics of split-phase interfaces is that they are bidirectional: there is a downcall to start the operation, and an upcall that signifies the operation is complete. In nesC, downcalls are generally commands, while upcalls are events. An interface specifies both sides of this relationship.

Figure 4 represents the structure of dispatcher pattern.

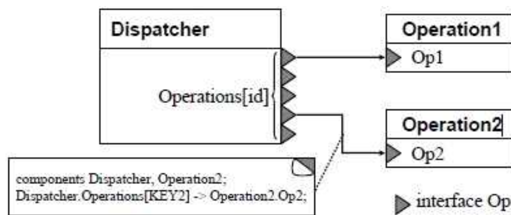

Fig 4. Structure of Dispatcher Pattern

A better approach in TinyOS is to use the dispatcher pattern. A Dispatcher invokes operations using a parameterised interface based on a data identifier. Operation implements the desired functionality and wires it to the dispatcher. The compile-time binding of the operation simplifies program analysis and puts dispatch tables in the compiled code, saving RAM. Dispatching provides a simple way to develop programs that executes in reaction to their environment[22-23].

Figure 5 shows the internal procsessing flow of Embedded Android platform. In this flow, we use the JNI object file which should generate the native code(libhello-jni) for Android and JNI source file which control the TinyOS sensor network with RF communications.

Commands init(), start() and stop() are the initial state of the Android program. Usb port open() function setups the usb port. The void onCreate() function generates the thread which starts the Android application program. System loadlibrary, "Hello-jni" callup the API function. DataByteStr receives the sensor data from H-mote. Widget function displays the sensor data to the LCD of the Embedded TKU 320.

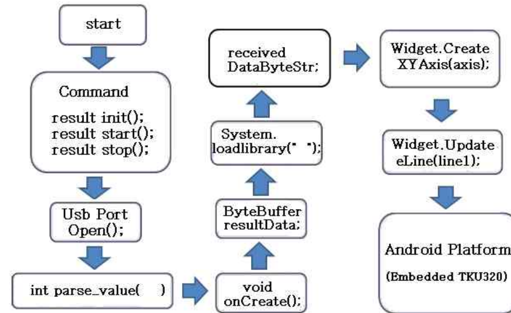

Fig 5. Structure of Software Flow

Figure 6 describes the data-logging result of temperature in H-mote sensor with the Android platform embedded TKU320 interfacing USN via RF communications. First we used the log-cat functions with Eclipse for verifying the sensing data, and after setup the GUI with interfacing JDK program for Android platform.

The GUI of the LCD in the TKU320 describes the temperature signal from H-mote under the Cygwin and TinyOS environments. It is possible to show the

Fig. 6 Experimental Results

several sensing data in this Android system, using multi-processing or multi-threading of Dalvik virtual machine.

Ⅴ. Conclusions

In this paper, we implement the Android NDK porting application with Eclipse(JDK) ADT and TinyOS 2.0. TinyOS and Cygwin are component based embedded system and an open-source basis for interfacing with sensor application from H-mote.

This experiment describes the improvement of sensor interfacing functionality under the Android PXA320 embedded RTOS platform.

The results of experiment are described to show the flexibility of mobile devices, such as smart phone or portable electronics devices. We will experiment the results of this paper in detail to be broaden the scope of sensor interface and design considerations.

참고문헌

[1] 김명호, Cygwin과 함께 배우는 C 프로그래밍, 홍릉 출판사, 02. 2010

[2] Levis, Philip, TinyOS Programming, Cambridge University Press, 04. 2009

[3] Matischek, Rainer, A TinyOS-Based Ad Hoc Wireless Sensor network, VDM Verlag, 07. 2008

[4] Marvell PXA320 Processor, Graphics and Input Controller, December 14, 2006

[5] 김상형, 안드로이드 프로그래밍 정복, 한빛미디어, 5, 10, 2010.

[6] Mary Campione, Kathy Walrath, The Java Tutorial, Object-Oriented programming for the Internet, Addison-Wesley, 2001.

[7] Downey, Allen B, Python for software Design, Cambridge, 03, 2009.

[8] http://developer.android.com/refernece/packages.html

[9] 노성동, 이명의, 임베디드 리눅스 구조와 응용, GS 인터비전, 5, 2009.

[10] Philip Levis, TinyOS programming Guide, Computer Systems Laboratory Stanford University, 10. 2006.

[11] 홍선학, "UML기반의 창의공학용 로봇설계, 한국통신학회," 제33권, 제10호, 2008, pp. 343-349.

[12] 홍선학, "GUI환경을 갖는 퍼지기반 이동로봇제어," 대한전자공학회, 제43권, IE편, 제4호, 2006, pp. 340-347.

[13] 홍선학, "센서결합을 이용한 이동로봇제어," 대한전자공학회, 제42권, TE편, 제2호, 2005, pp. 91-98.

[14] 홍선학, "영상 추적을 이용한 이동로봇제어," 대한전자공학회, 제42권, TE편, 제4호, 2005, pp. 201-208.

[15] Philip Levis, Component Composition and Radio Communication, Computer Systems Laboratory Stanford University, 07. 2003.

[16] 김정원, 신진철, 박형근, "Zigbee를 이용한 사용자 인식기반의 헬스케어시스템구현," 디지털산업정보학회, 제4권,3호, 2008.

[17] Philip Levis, Simulating TinyOS Applications in TOSSIM, Computer Systems Laboratory Stanford University, 08. 2003.

[18] Nagy, Chris, Embedded Systems Design Using the Ti Msp430 Series, Newnes, 09. 2003.

[19] 홍선학, 윤진섭, "임베디드 RTOS기반의 로봇 컨트롤러 설계," 디지털 산업정보학회, 제6권 4호, 2010.

[20] 조나단 코벳, 리눅스 디바이스 드라이버, 한빛미디

어, 1999.

[21] PXA320_Monahans_P_DM_Vol[1]+Rev_0.95 System and Timer Developers, 11, 7, 2006

[22] Reto Meyer, Professional Android 2 Application Development, Wrox, 02. 2010.

[23] 홍선학, “MCU플랫폼창의공학용 로봇 제어기 설계,” 디지털 산업정보학회, 제5권 4호, 2009.

■ 저자소개 ■

홍 선 학
Hong, Seon Hack

1992년~현재 서일대학 컴퓨터전자과 교수
1994년 광운대학교 대학원 박사 졸업
1988년 광운대학교 전기공학과 석사졸업
1985년 광운대학교 전기공학과 학사졸업

관심분야 : 제어, 컴퓨터응용, 로봇 분야 등
E-mail : hongsh@seoil.ac.kr

남궁 일 주
Nam Gung, Il Joo

2008년~현재 서일대학 인터넷정보과 겸임교수
2002년 EBS 교육방송 컴퓨터강좌 강사

관심분야 : 멀티미디어
E-mail : allcom@allcom.co.kr

논문접수일 : 2011년 6 월 27 일
수 정 일 : 2011년 8 월 12 일
게재확정일 : 2011년 8 월 23 일