모바일 단말기를 위한 J2ME 플랫폼

한국산업기술대학교 이대현

1. 서 론

휴대폰으로 대표되는 모바일 산업은 지난 십년간 글자그대로 폭발적인 성장을 해왔으며, 최근에는 DMB 및 와이브로 서비스가 시작되면서 새로운 전기를 맞고 있다. 모바일 서비스가 대중화되면서, 그에 따라 시장에는 각양각색의 모바일 단말기가 쏟아져 나오고 있으며, 대부분 무선 인터넷 접속 기능을 기본적으로 지원하고 있다.

초기의 무선 인터넷은 WAP 방식이 주로 이용되었으나, 다양한 부가 서비스의 제공을 통한 무선 인터넷의 활성화를 위해 WAP 방식의 문제점을 개선한 가상머신(Virtual Machine:VM) 플랫폼 방식이 현재는 주류로 자리잡고 있다.VM은 독립적인 환경에서 애플리케이션 등이 실행될 수 있는 환경을 의미한다. VM이라는 용어는 객체지향 언어인 스몰토크(SmallTalk)에서 처음으로 사용되었으며, 그 특징으로는 기종 간의 자유로운 이식성, 사용자의 추상화, API 제공 등을 들 수 있다. VM이 설치된 휴대 단말기는 동일한 실행환경을 가지게 되므로, VM에서 개발된 애플리케이션은 단말기의 종류과 상관없이 동일한 동작이 보장된다.

국내의 경우 이동통신 3사들이 GVM, SK-VM, KVM, BREW 등 다양한 종류의 VM 플랫폼을 도입하여 무선 인터넷 서비스를 제공하고 있으며, 최근에는 플랫폼 간의 호환성을 높이기 위해 WIPI 라고 불리는 통합 표준 플랫폼을 도입하고 있다.

본고에서는 VM 플랫폼 중 전세계적으로 가장 많이 사용되고 있는 대표적인 플랫폼인 J2ME[1] 플랫폼의 기능과 특징 및 구조 등을 자세히 살펴보도록 하겠다. 사실 앞서 언급한 국내 이동통신사들의 각기 다른 VM 플랫폼들도 실제로 내부적으로 J2ME에 기반한 것이 대부분이다.

1) 작년에 선마이크로시스템스가 '2'를 삭제한 'Java ME'로 이름을 변경했으나, 편의상 본고에서는 J2ME을 사용한다.

2. J2ME 플랫폼 개요

2.1 J2ME의 등장

J2ME는 1999년 6월 자바원(JavaOne) 컨퍼런스에서 선마이크로시스템스가 발표한 플랫폼으로써, Java2 Micro Edition의 약자이다. 이 플랫폼은 휴대폰이나 양방향 삐삐, 스마트폰과 같은 소형 기기에서 사용가능한 자바 애플리케이션을 목표로 제안되었다.

자바 언어가 처음 인터넷에서 성공할 수 있었던 가장 큰 요인은, 선마이크로시스템스에서 자바를 선보이면서 주장한 "Write once, run anywhere"라는 모토대로, 자바로 작성된 프로그램은 모든 컴퓨터에서 실행가능하고 서버에서 다운해서 실행될 수 있다는 점이었다. 그런데 휴대폰을 비롯한 각종 무선 정보기기들이 각자 서로 다른 사양을 가지고 쏟아져 나오기 시작하면서, 자바 프로그램이 모든 플랫폼에서 동일하게 동작한다는 원칙을 지키는 것이 사실상 불가능해졌다. 결국, 각 기기별 특성과 한계 때문에 하나의 크기로는 모른 것을 만족시킬 수 없다("one size doesn't fit all")라는 것을 깨닫고 J2ME가 새로 등장하게 된 것이다.

J2ME가 등장하기 이전부터 사실 임베디드 시장을 위해 만들어진 플랫폼으로 퍼스널 자바(Personal Java)와 임베디드 자바(Embedded Java)가 있었다.

퍼스널자바는 네트웍에 연결할 수 있는 소형 기기에 적합한 소규모의 자바 실행 환경을 정의한 것으로 CVM(Classic Virtual Machine)기반으로 구현된다. 기본적으로 퍼스널 자바는 JDK 1.1.8 API의 대부분의 기능을 수용하고 있으며 소형 휴대기기에 적합한 스펙이며, 후에 J2ME CDC의 모태가 되었다.

임베디드 자바는 퍼스널 자바 보다는 더 작은 자원을 가진 프로세스 컨트롤러, 네트웍 라우터, 휴대폰 등의 소형 디바이스를 위한 환경을 가지고 있으며, 자바 탄생 초기의 근본 목표였던 임베디드 기기들을 위한

스펙이다.

이러한 두가지의 플랫폼들이 기반이 되어, J2ME가 등장하게 되었다.

2.2 J2ME의 특징

J2ME 비록 휴대폰이나 임베디드 기기를 위한 플랫폼이지만, 기본적으로 자바가 가지고 있는 특징인, 객체지향방식의 프로그래밍, 코드의 높은 이식성, 안전한 네트워크 보안 지원 및 J2SE와 J2EE로의 상위 호환성은 그대로 유지하고 있다. 여기에 더하여, J2ME는 다음과 같은 특징을 지니고 있다:

다중플랫폼호환성(cross-platform compatibility): 100% 순수한 J2ME API를 사용하는 애플리케이션은 쉽게 다른 업체가 만든 모델과 광범위하게 호환될 수 있다. 단말기가 다르다고 해서 프로그램 개발자 측에서 같은 프로그램을 다시 개발하는 일은 상당히 번거로운 일이다. 하지만 J2ME 플랫폼은 어느 단말기에서든지 시간과 장소에 관계없는 강력한 호환성을 가진다.

보안성 : 무선 인터넷 환경에서의 보안은 아직도 해결되지 않는 문제점들이 많다. J2ME는 기존 자바의 보안 모델을 무선 쪽에 적용시킴으로써, 이를 상당부분 해결하고 있다. J2ME로 작성된 애플리케이션은 기본적으로 디바이스의 하드웨어나 다른 리소스에 직접 접근할 수 없기 때문에 바이러스나 다른 악성 프로그램을 만들어낼 수 없다.

동적인 애플리케이션 다운로드 : 무선 서비스를 통한 자바 애플리케이션들은 실시간적으로 동적으로 다운로드되므로 사용자들은 A/S 센터를 방문하거나, 단말기를 교체할 필요없이 소프트웨어들의 업그레이드를 쉽게 받을 수 있다.

한편, J2ME가 목표로 하고 있는 기기들을 좀 더 살펴보면, 그림 1에 나와 있는 바와 같이 크게 두 개

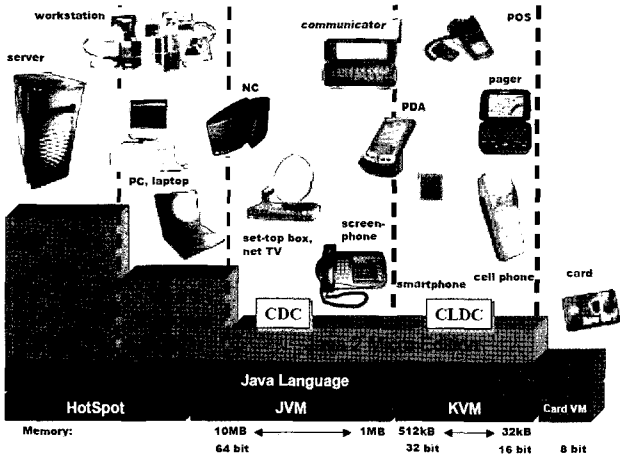

그림 1 자바 플랫폼들의 목표 시장(출처: http://java.sun.com)

의 부류로 요약할 수 있다:

공유되고 고정되고 연결된 정보 기기 : 그림에서 CDC라는 이름으로 대표되는 범주에 속하는 기기들이다. TV 셋톱박스, 인터넷 TV, 인터넷 화상폰, 네비게이션 시스템 등을 지칭한다. 이러한 기기들은 다양한 종류의 유저 인터페이스를 가지고 있으며, 메모리는 대개 2~16 메가바이트 정도이며, TCP/IP 프로토콜을 통해서 높은 대역폭의 네트워킹 연결이 가능하다.

개인 모바일 정보 기기 : 그림에서 CLDC라는 이름으로 대표되는 범주에 속하는 기기들로써, 휴대폰, 삐삐, PDA 등을 가리킨다. 이런 기기들은 대개 매우 단순한 유저 인터페이스를 가지고 있으며, 메모리 용량 역시 매우 작은 것이 특징이다. 그리고 네트워크 연결 역시 매우 제한적으로 지원된다.

사실 위와 같이 구분하는 것은 최근에 들어서 기술의 발전에 따른 기기들간의 컨버전스로 인해 그 경계가 모호해지고 있다. 따라서, 추후에는 위와 같이 기기들의 기능적인 측면에서 구분하는 대신, 메모리 용량, 네트워크 대역폭, 배터리 전력 소모, 기기의 화면 싸이즈와 같은 속성들을 통해서 구분하는 방향으로 나갈 것이다.

2.3 J2ME의 구성 요소: 컨피규레이션 (Configurations)

컨피규레이션은 자바 애플리케이션의 운용을 목표로 하는 기기의 특성 및 여건을 고려하여 공통특성을 갖는 것들을 그룹화한 뒤, 각 그룹에 대해 자바 가상 머신의 기능 범위, 자바 프로그래밍 언어 지원 범위 및 자바 클래스 라이브러리 API 등을 정의한 것이다. 따라서 사실상 하나의 컨피규레이션이 새로운 또 하나의 플랫폼과 맞먹는 것으로 볼 수도 있다. 그러나, J2ME 테두리 내에서 정의되기 때문에 컨피규레이션이라 명명된 것이다.

현재까지 나온 J2ME에서는 디바이스를 크기와 성능에 따라 다음과 같이 두가지로 구분하고, 각각의 디바이스 군을 위해 CLDC(Connected Limited Device Configuration) 컨피규레이션과 CDC(Connected Device Configuration) 컨피규레이션을 정의하고 있다.

CLDC는 네트워크 연결이 가능하고, 비교적 적은 리소를 가진 소형 디바이스들에게 적합하도록 만든 컨피규레이션이다. 디바이스에 있어 중요한 메모리, 전원, 네트워크 대역폭 등이 극히 제약적인 디바이스를 위한 컨피큐레이션이라고 할 수 있으며, 상세한 내용은 3절에서 다시 다룬다.

CDC는 CLDC가 타켓으로 하는 디바이스보다 덜

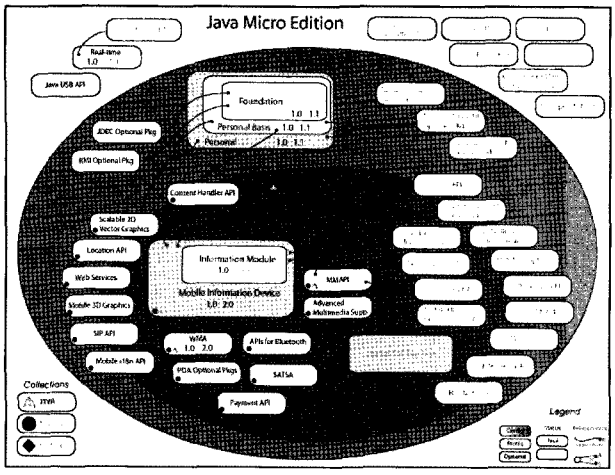

그림 2 J2ME의 컨피규레이션 구조(출처: http://java.sun.com)

제한적인 환경을 가진 디바이스를 위한 컨피규레이션이다. CLDC에 비해 디바이스의 제약 사항이 적기 때문에, CLDC의 경우 제한적으로 자바 언어를 지원하지만, CDC는 자바 스펙에 정의된 자바 언어를 완벽히 지원하며, 따라서 J2SE의 API의 대부분을 사용할 수 있다. 그리고 가상 머신으로 CVM이라는 것을 사용하는데, CVM은 기본 자바 가상 머신인 JVM의 스펙을 그대로 따른다.

2.4 J2ME의 구성 요소: 프로파일(Profiles)

J2ME가 이미 컨피규레이션을 통해 세분화되어 있긴 하지만, 컨피규레이션은 소형 기기의 특성을 반영한 기반 환경을 정의한 것이고, 각각의 기기 유형에 따라 달라질 수 있는 부분을 위해 특수한 API를 정의할 필요가 있으며, 이렇게 정의되는 API들의 집합을 프로파일이라고 한다. 예를 들어, 휴대폰은 숫자 버튼과 통화 버튼 등 특수한 사용자 인터페이스를 가지고 있는데, 이를 지원하기 위해 공통된 라이브러리가 있다면 개발자들은 이 라이브러리를 사용하여 휴대폰을 위한 애플리케이션을 좀 더 쉽게 개발할 수 있을 것이다. 이러한 라이브러리가 최소한 가지고 있어야 할 표준을 정의하는 것이 바로 프로파일이다.

이렇게 컨피규레이션과 프로파일의 개념적인 분할이 필요한 이유는 메모리와 CPU 등의 크기와 성능이라는 측면에서의 요구사항이 동일한 디바이스들의 집합을 하나로 묶어 컨피규레이션을 정의하고, 이러한 컨피규레이션을 바탕으로 각 디바이스의 기능, 혹은 시장의 요구 사항에 맞추어 프로파일을 정의함으로써, 플랫폼의 통일성과 다양성을 동시에 만족시킬 수 있기 때문이다.

프로파일은 애플리케이션 라이프 싸이클 모델을 정의하는 API, 유저 인터페이스 관련 API, 영구 기억 장치를 사용하는 방법을 정의하는 API 등으로 구성된다. 대표적인 프로파일로써 CLDC에 기반하며, 휴대폰을 위해 정의된 MIDP(Mobile Inforation Device Profile)가 있는데, 다음 3절에서 자세히 살펴보도록 하겠다.

한편, CDC에서 정의되고 있는 프로파일에는 CDC 상에서 어떤 J2ME 애플리케이션을 작성하더라도 공통적으로 필요한 API를 정의하고 있는 파운데이션 프로파일, 그리고 CDC 디바이스 군에 해당하는 각각의 디바이스를 위한 TV 프로파일, 스크린폰 프로파일, 자동차 프로파일 등이 있다.

3. CDLC와 MIDP의 이해

3.1 CLDC(Connected Limited Device Configuration) 컨피규레이션

3.1.1 CLDC의 가상 머신 KVM(K Virtual Machine)

KVM은 CLDC 컨피규레이션이 지정하는 자바 가상 머신으로써, CLDC를 이용하며 개발된 애플리케이션의 실행 환경이 된다. CLDC가 사용될 하드웨어들은 성능 및 용량 제약이 매우 커서 기존의 JVM의 모든 사항을 구현하기에는 무리가 따르기 때문에, KVM은 기존의 JVM에서 몇가지 기능적인 사항을 축소했다. 이에 따라 KVM은 JVM에 비해 아래와 같은 차이를 갖는다:

① 부동소수점 자료형(float이나 double형)을 지원하지 않는다.

② 자바 네이티브 인터페이스(Java Native Interface: JNI)를 지원하지 않는다. JNI는 덩치가 크기도 하지만 CLDC의 제한적인 보안 모델로 인해, 네이티브 함수 호출 자체가 위험할 수 있기 때문에 지원하지 않는다.

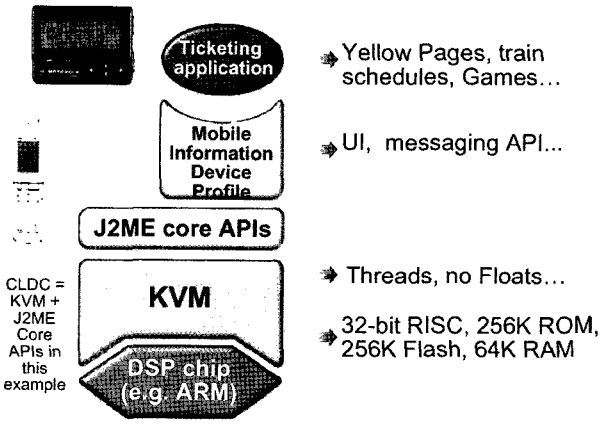

그림 3 J2ME의 KVM(출처: http://java.sun.com)

③사용자정의 클래스 로더를 사용할 수 없다.

④리플렉션(reflection) 기능을 지원하지 않는다. 리플렉션은 자바 프로그램이 가상 머신 내부의 클래스, 인터페이스, 객체 인스턴스를 조사할 수 있는 자바 가상 머신의 기능이지만, KVM에서는 프로그램의 실행 속도 저하와 메모리 사용량의 부담으로 인해 지원되지 않는다. 따라서, 리플렉션 기능을 이용하는 RMI, 객체 직렬화, JVMDI(디버깅 인터페이스), JVMPI(프로파일러 인터페이스) 등의 기능 역시 지원할 수 없게 되었다.

⑤에러처리가 제한적으로 지원된다.

3.1.2 CLDC의 특징

CLDC는 휴대폰과 같은 모바일 단말기를 위한 J2ME의 컨피규레이션으로써, 다음과 같은 특징을 가지고 있다:

①자바 언어의 기본 특징을 그대로 가지고 있다. CDLC가 타켓으로 하고 있는 디바이스의 한계로 인해 지원되지 않는 부분이 있으나, 기본적으로 자바 언어 스펙에 정의된 특징은 거의 모두 적용된다.

②모래 상자(Sandbox) 보안 모델을 사용한다. 애플리케이션들은 각자 폐쇄적인 실행 환경 상에서 동작하므로 서로 방해받지 않는다.

③J2SE에서 가져온 핵심 패키지들을 보유하고 있다. 여기는 java.lang.*, java.io.*, java.util.* 과 같은 클래스들이 포함된다.

④네트워킹과 입출력을 위해 GCF(General Connection Framework)라는 패키지가 별도로 제공된다.

한편, 개발 언어 측면에서 CDLC는 표준 자바 언어와 비교해서, 다음과 같은 제약 사항이 존재한다:

①부동소수점을 지원하지 않는다. 휴대폰과 같은 모바일 기기들에 사용되는 CPU들은 일반적으로 부동소수점 연산을 처리하는 하드웨어를 가지고 있기 않기 때문이다. 물론 소프트웨어적으로 부동소수점을 처리할 수 있겠지만, 속도 문제로 인해 부동소수점의 사용은 사실상 어렵다고 볼 수 있다.

②CLDC 라이브러리의 Object 클래스에서는 finalize() 메소드를 지원하지 않는다. 일반적으로 자바 언어는 finalize() 메소드가 있어, 생성된 인스턴스들에 대한 가비지 콜렉션 알고리즘을 자동적으로 수행하며 효율적인 메모리 관리를 꾀한다. 하지만, CLDC에서는 복잡한 가비지 콜렉션 알고리즘의 수행에 따른 프로그램 실행 속도의 저하 및 메모리의 부족문제로 인해 finalize() 메소드를 지원하지 않는다.

③에러처리가 제한적으로 지원된다.그 이유는 CLDC가 지원하는 휴대폰과 같은 기기들의 종류가 매우 다양해서 에러 처리를 동일하게 하는 것이 사실상 불가능하기 때문이다. 많은 경우 디바이스들은 에러가 발생했을 때, 그냥 리셋을 해버리는 경우가 대부분이다. 따라서 CLDC는 다음과 같은 세가지의 에러클래스만 지원한다:

java.lang.Error
java.lang.VirtualMachineError
java.lang.OutOfMemeoryError

3.1.3 CLDC의 클래스 검증 모델

자바 클래스 파일은 클래스 파일 검증기에 의해 검증되어야 한다. PC에서 자바 애플리케이션을 수행할 때는 프로그램을 수행시키는 PC에서 클래스의 검증과정을 수행했지만, CLDC에서는 클래스 검증과정을 실제 응용 프로그램이 수행되는 단말기에서 모두 전담하지 않는다. 즉, 사전 검증(Preverify)이라는 방법을 사용하여 애플리케이션을 배포하는 서버 측에서 검증을 일차적으로 거치게 함으로써, 실제 애플리케이션이 수행되는 디바이스의 부담을 줄이고 있다. 일단 서버측에서(프로그램 작성자 측)에서 사전 검증을 거친 클래스 파일은, 디바이스에 다운로드된 후 다시 디바이스 상에서 검증을 거친다. 즉, 디바이스 상의 가상 머신은 미리 검증된 클래스파일을 받게 되고, 이를 재차 확인함으로써 안전한 자바 애플리케이션의 수행을 보장하고 있다. 이러한 두단계의 검증은 사전 검증으로 인해 실제 클래스의 크기가 원래보다 커지는 단점은 있으나, 애플리케이션 작성자 측에서 사전 검증을 수행함으로써 그만큼 최종 디바이스 상에서의 부담을 줄일 수 있는 장점을 지닌다.

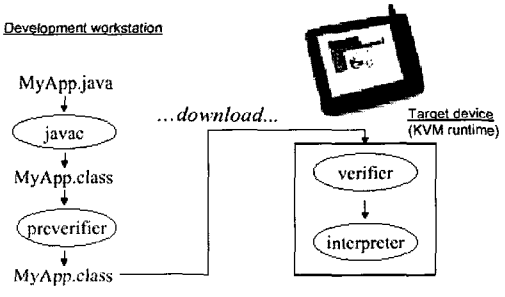

그림 4 클래스 검증 모델(출처: http://java.sun.com)

3.2 MIDP(Mobile Information Device Profile)

3.2.1 MIDP 개요

MIDP는 CLDC 컨피규레이션을 기반으로 하는 프로파일로, 휴대폰 상에서 애플리케이션의 개발을 지원

하기 위해 만들어진 프로파일이며 현재 버전 2.0까지 공개되어있다. MIDP 2.0 에서 요구하는 최소 하드웨어 사양은 표1과 같다.

표 1 MIDP 2.0의 최소 하드웨어 요구 사양

디스플레이	화면 크기: 96x54 디스플레이 깊이: 1비트 픽셀 모양(가로 세로비): 약 1:1
입력	사용자 입력 기법(한손 키보드, 양손 키보드 또는 터치 스크린) 중 하나 이상
메모리	MIDP 구현에 256KB의 비휘발성 메모리(CLDC 요구 사항보다 큼) 응용 프로그램에서 작성된 영구 데이터에 8KB의 비휘발성 메모리 Java 런타임(예: Java 힙)에 128KB의 휘발성 메모리
네트워킹	제한된 대역폭을 사용한 단속적(가능한 경우) 양방향 무선
사운드	전용 하드웨어나 소프트웨어 알고리즘을 통한 톤 재생 기능

한편, MIDP의 API는 CLDC 라이브러리에 유저인터페이스 관련 API 및 I/O 관련 API 등을 추가한 것으로써, 표2과 같이 다양한 라이브러리 패키지로 구성된다:

표 2 MIDP 2.0의 표준 패키지

유저 인터페이스 패키지(javax.microedition.lcdui)
게임 패키지(javax.microedition.lcdui.game)
응용 프로그램 라이프 싸이클 패키지 (javax.microedition.midlet)
지속성 패키지(javax.microedition.rms)
네트워킹 패키지(javax.microedition.io)
공개키 패키지(javax.microedition.pki)
사운드 패키지(javax.microedition.media)
핵심 패키지(java.lang, java.util)

모바일 기기에서 MIDP 플랫폼의 구조를 보면 그림 5와 같다. 플랫폼의 가장 하단에 MID, 즉 디바이스에 대한 부분이 있고 그 위에는 네이티브 시스템 소프트웨어가 존재하며 그 상위에 CLDC가, 그리고 다시 그 위에서 MIDP가 위치하게 된다. MIDP 애플리케이션은 MIDP에서 정의하는 API와 CLDC에서 정의하는 API를 이용하여 작성되는 것이 일반적이지만, 그림에 보이는 바와 같이 OEM API 클래스를 이용한 애플리케이션도 개발할 수 있다. OEM API라는 것은 다른 디바이스에서는 사용할 수 없고, 자신의 디바이스에서만 독립적으로 수행될 수 있도록 제작된 API이다. 이는 자신의 디바이스의 특성을 잘 나타내거나 또는 MIDP에서는 제공하지 않는 특성화된 클래스들로 이루어져 있다. 지금까지 나와있는 많은 MIDP API들에도 이런 예는 많이 보이고 있으며, 우리나라의 이동통신사에서도 GVM, SK-VM 등 이통사 고유의 VM을 제공하고 있는데, 이런 VM들은 휴대폰의 특성에 맞는 기능, 예를 들어 진동 기능이나, SMS를 컨트롤하는 기능 등을 OEM API를 이용하여 구현하고 있다.

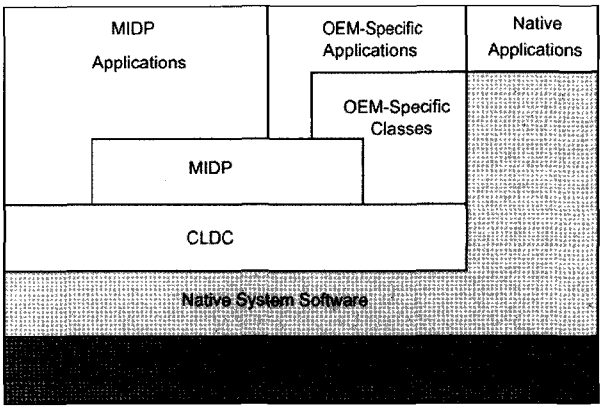

그림 5 MIDP 플랫폼의 구조(출처: http://java.sun.com)

3.2.2 미들릿(MIDlet)의 이해

MIDP 애플리케이션을 작성하기 위해선 일단 javax.microedition.midlet이라는 추상 클래스를 상속받은 클래스를 작성해야 한다. 즉, MIDP 상에서 수행되는 애플리케이션은 모두 이 클래스를 상속받은 클래스들이며, 이런 클래스들을 통틀어서 미들릿(MIDlet)이라고 부른다. 표준 자바에서 J2SE 기반의 애플릿(applet)을 작성하기 위해 무조건 'Applet'이라는 클래스를 상속받아 작성하는 것과 유사하다.

미들릿을 휴대폰에 설치해서 실행하거나, 삭제하는 동작은 휴대폰의 시스템 소프트웨어로 미리 설치되어 있는 JAM(Java Application Manager)에 의해 동작하게 된다.

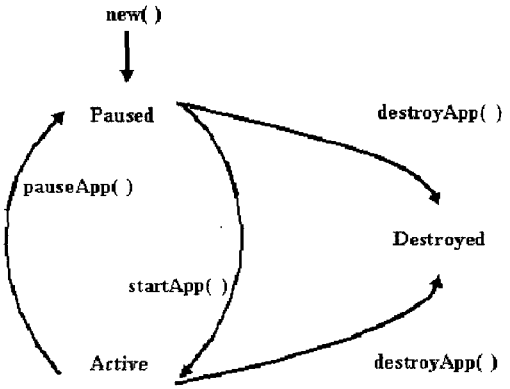

그림 6 미들릿의 라이프 싸이클 모델
(출처: http://java.sun.com)

미들릿을 개발하기 위해서는 먼저 미들릿의 라이프 싸이클 모델을 이해할 필요가 있다. 그림 6에서 보여지듯이, 미들릿은 모두 세 개의 상태를 가지게 된다:

정지(Paused) 상태 : 미들릿이 (1) 초기화되었거나 (2) 멈추고 있는 상태. (1)의 경우는 미들릿이 방금 new에 의해 만들어진 경우임. (2)의 경우는 JAM이 pauseApp() 메소드로 미들릿을 멈추게 한 경우, 혹은 미들릿이 스스로 notifyPause() 메소드로 JAM에게 멈추기를 요청하는 경우임.

활성(Active) 상태 : 미들릿이 실행중인 상태. 정지 상태에서 JAM이 미들릿의 startApp() 메소드를 호출하는 경우, 혹은 미들릿이 JAM에게 resumeRequest()를 요청해서 허락을 받고 startApp() 가 호출되는 경우에 활성 상태가 됨.

소멸(Destroyed) 상태 : 미들릿이 사용하는 모든 자원(메모리 등)을 해제하는 상태. JAM이 미들릿의 destroyApp()를 호출하여 강제로 미들릿을 해제하는 경우(더 중요한 다른 미들릿을 실행해야 하는데 자원이 없는 경우와 같은 경우), 혹은 미들릿 스스로 notifyDestroyed() 를 호출하여 미들릿의 수행을 종료하는 경우에 해당됨.

다음의 코드는 위에서 설명한 라이프 싸이클 모델을 기반으로 작성된 프로그램으로써, 'MIDlet' 클래스를 상속받아 'HelloMIDlet'이라는 이름을 갖는 미들릿을 작성하는 MIDP 프로그램의 전형적인 구조를 보여주고 있다.

```
import javax.microedition.midlet.*;

public class HelloMIDlet extends MIDlet {
  public HelloMIDlet() {}
  public void startApp() {}
  public void pauseApp() {}
  public void destroyApp(boolean unconditional)
{}
}
```

그림 7 MIDP 프로그램의 구조

위의 MIDP 프로그램을 보면, 'HelloMIDlet' 미들릿은 생성자 함수 외에 크게 세 개의 메소드로 구성된다. 이 메소드들은 미들릿의 핵심적인 함수로써, JAM이라고 불리는 상위의 소프트웨어에서 이 메소드들을 각각 호출함으로써 미들릿이 실제로 동작하게 된다. 각 메소드의 기능은 다음과 같다.

표 3 미들릿의 기본 메소드

displayApp()	미들릿이 종료 상태로 되기 전에 호출되며, 프로그램의 모든 자원을 해제하고, 그 상태를 저장한다.
startApp()	미들릿이 실행 상태가 되기 전에 호출되며, 모든 필요한 자원을 얻는다.
pauseApp()	미들릿이 대기 상태로 가기 전에 호출되며, 일시적인 자원이라도 해제하고 미들릿은 정지상태로 변환한다.

3.2.3 미들릿(MIDlet)의 관리

미들릿을 실제로 휴대폰에 다운로드하고 실행하기 위해서는 단순히 미들릿 클래스 파일만 있어서는 되지 않고, 여러 가지의 과정을 거쳐야 한다. 이 과정을 이해하기 위해서 JAM, JAR, 미들릿 슈트 및 JAD의 개념을 알아보고, 이를 바탕으로 애플케이션의 개발에서 배포하는 과정을 살펴보자.

JAM(Java Application Manager) : JAM은 애플리케이션 관리 소프트웨어로써, 서버로부터 MIDP 응용 프로그램을 다운받거나, 다운받은 프로그램을 디바이스 설치하기, 미들릿의 실행 및 삭제, 미들릿의 버전 업그레이드 등을 담당한다.

JAM이 도입된 이유는 고객이 굳이 소프트웨어 업그레이드를 받기 위해 A/S 센터를 찾는 번거로움을 없애고, 취향에 따라 다양한 선택의 폭을 가진 애플리케이션을 직접 고를 수 있고, 이렇게 설치한 프로그램이 업그레이드되었을 경우, 자동적으로 업그레이드를 수행할 수 있도록 하기 위한 것이다.

미들릿 슈트(MIDlet Suite)와 JAR : MIDP 애플리케이션이 설치되고 실행되기 위해서는 단순히 실행파일만 있어서는 안된다. 윈도우 애플리케이션을 설치할 때도 설치 관련 정보나 각종 리소스 파일들이 함께 설치되어야 하듯이, MIDP에서도 미들릿이 JAM에 의해서 동작하기 위해서는 미들릿을 구성하는 클래스 파일 외에도 추가적인 정보와 함께 적절한 형태로 패키징이 되어야 한다.

패키징된 파일은 확장자가 .jar인 JAR 파일이며, 자바 SDK의 jar.exe 프로그램을 이용하여 압축파일을 만들게 된다. JAR 파일 안에는 미들릿 클래스들이 들어가게 되며, 이미지 파일과 같은 리소스 파일들과 메니페스트(Manifest)라는 텍스트 파일도 함께 들어가게 된다. 이 때 여러개의 미들릿들이 JAR 파일 속에 들어갈 수 있는데, 사용자는 JAM을 이용하여 미들릿 슈트에 포함된 미들릿 중 하나를 선택하여 실행할 수 있다. JAR 압축 파일이 생성되었으면, JAM에 의해 참조되는 파일인 JAD 파일을 함께 작성해야 한다. 이 두 개의 파일을 묶어서 미들릿 슈트라고 부르게 되며, 이것이 실제로 서버에 올라간 후, 사용자는 해당 미들릿 슈트를 다운받아 실행할 수 있게 된다.

JAD(Java Application Descriptor)와 메니페스트(Manifest) 파일 : JAM이 애플리케이션을 다운로드하여 설치 및 실행하기 위해서는 미들릿의 이름, 버전, 크기와 같은 속성들이 필요하게 된다. 이러한 정보를 제공해 주는 역할을 JAR 파일 속에 포함된 메니페스트 파일이 맡게 된다. 다시 말해 메니페스트는 다운로

드된 JAR 파일의 명세서라고 생각할 수 있다. JAR 파일내의 정보가 메니페스트 파일에 담겨있기 때문에 나타나는 문제점이 있다. 예를 들어, 이미 다운로드하여 설치한 애플리케이션의 JAR 파일을 다시 다운로드하여 설치하는 경우 사용자에게 해당 사항을 알려주어야 할 것이다. 그런데 JAM은 JAR 파일 내에 있는 메니페스트 정보를 통해서 애플리케이션에 대한 정보를 알 수 있으므로, 일단 JAR 파일을 먼저 다운로드한 후에 비로소 이미 설치했던 프로그램임을 알 수 있게 된다. 휴대폰을 이용한 다운로드에 지불하는 비용을 생각한다면, 무척 비효율적인 방법이다. 이에 대한 대안으로 생긴 것이 JAD이다. 즉, JAM은 미리 JAD를 다운로드하고 이를 참조하여 다운로드할 JAR 파일안의 미들릿이 이미 단말기에 최신 버전으로 깔려있는지, 다운로드 크기는 적당하지 등의 검사를 미리 수행하여 결국 적합한 JAR 파일만 효율적으로 다운로드할 수 있는 것이다.

미들릿의 개발 및 배포는 위에 설명된 방법들을 이용하여 체계적으로 이루어지게 된다. 아래의 그림은 실제 미들릿의 작성에서 배포까지의 과정을 나타낸다.

1. 미들릿 클래스의 작성.
2. 컴파일(javac를 이용)
3. 사전검증(preverify를 이용).
4. 미들릿 클래스와 리소스가 포함된 JAR 파일 생성.
5. JAD 파일 작성.
6. 서버에 미들릿 슈트 업로드.

그림 8 MIDP 프로그램의 개발 및 배포 과정

4. 부가 패키지(Optional Packages)

부가 패키지는 J2ME 플랫폼을 확장하는 것으로써, CDLC 또는 CDC와 관련된 프로파일들에 추가적이 기능을 더하는 것이다. 아주 특별한 애플리케이션의 요구에 대응하기 위해서 만들어진 것으로써, 부가 패키지는 데이터베이스 연결, 무선 메시지, 멀티미디어, 3D 그래픽, 웹써비스와 같이 새로이 출현하는 기술을 지원하기 위한 표준 API를 제공한다. 이러한 부가 패키지는 모듈 구조를 가지고 있기 때문에, 디바이스 생산회사는 필수적으로 요구되는 패키지만을 포함시킴으로써, 불필요한 기능을 가지고 가는 부담을 덜 수 있다. 부가 패키지는 어떤 컨피규레이션과 프로파일의 조합과도 함께 구현될 수 있다.

현재까지 공개된 주요한 부가 패키지는 다음과 같은 것들이 있다:

- **CHAPI(Contents Handler API)**: 네이티브 애플리케션 프로그램으로부터 J2ME 애플리케이션을 호출하는 것을 지원하기 위한 API.
- **보안 및 신뢰 서비스(Security and Trust Services)**: 보안 요소를 강화함으로써 좀 더 향상된 보안 기능을 제공하는 API.
- **웹 서비스(Web Services)** : J2ME 애플리케이션으로부터 웹 서비스를 표준 액세스하는데 필요한 API.
- **모바일 미디어 API(Mobile Media API)** : 모바일 기기들로 하여금 J2ME 상에서 다양한 그래픽 및 사운드 처리를 구현할 수 있게 지원하는 API.

5. 맺음말

지금까지 모바일 단말기를 위한 대표적인 가상 머신 플랫폼인 J2ME에 대해서 상세히 살펴보았다. 1999년 J2ME 플랫폼이 소개된 이후로, J2ME 플랫폼은 계속 진화를 거듭하고 있다. CLDC 컨피규레이션은 현재 버전 1.1까지, MIDP는 2.0까지 개정되었으며, 그 외 신규 기기들을 효과적으로 지원하기 위해 각종 부가 패키지들이 계속 갱신되고 있다.

사실, J2ME 플랫폼은 현재 큰 변화의 시기를 맞이하고 있다. 근본적인 이유는 점차로 모바일 단말기들의 성능이 기존 PC의 성능에 필적하게 되면서, J2ME 플랫폼의 존재에 대한 의문이 일각에서 제기되고 있기 때문이다. 이 때문에 이제 J2ME이 J2SE 플랫폼으로 통합 흡수되어야 한다는 목소리도 나오고 있는 상황이다.

하지만 J2ME가 모바일 단말기를 위한 애플리케이션의 확대 보급에 결정적인 기여를 해온 것은 부인할 수 없는 사실이며, 여전히 전세계의 수많은 개발자들이 J2ME 플랫폼을 이용하여 휴대폰용 애플리케이션을 개발하고 있는 것도 또한 사실이다.

여러분들도 J2ME 플랫폼을 이용하여, 모바일 기기용 애플리케이션 개발에 참여해보길 기대해본다. 일단 개발하면, 전세계의 수억대의 휴대폰에서 확실한 동작하는 것을 J2ME가 보증한다!

참고문헌

[1] 권기경, 박용우, "모바일 자바 프로그래밍", p. 527, 한빛미디어, 2002.9.25

[2] 강성윤, 이경범, 홍성인, "자바 모바일 프로그래밍", p. 714, 도서출판 대림, 2002.1.10

[3] 김성환, 양석호, "모바일 자바 프로그래밍:J2ME 및 WAP 프로그래밍", p. 618, 피어슨에듀케이션 코리아, 2002.5.10

[4] http://java.sun.com/javame/index.jsp

이 대 현

2006~현재 시사게임즈(주) 대표이사
2005~현재 한국산업기술대학교 게임공학과 전임강사
2001~2004 삼성전자 책임연구원
2001 한국과학기술원 전기및전자공학과 박사
1995 한국과학기술원 전기및전자공학과 석사
1993 서울대학교 제어계측공학과 학사

관심분야 : 모바일 콘솔 게임 SW/HW, 게임 임베디드 시스템
E-mail : dustinlee@kpu.ac.kr