인터넷 단말기 플랫폼 BREW

KTF 임승혁
Qualcomm 박종렬

1. 인터넷 단말기 플랫폼 BREW

퀄컴의 BREW (Binary Runtime Environment for Wireless)는 세계 최초로 KTF에 의해 상용화된 무선 인터넷 단말 플랫폼으로서, 당시 열악한 성능의 단말 환경에서 자바와 같은 인터프리팅 방식의 플랫폼 보다 월등히 우수한 성능과 화려한 UI를 제공하는 최초 단말 플랫폼이라 해도 과언이 아니다.

BREW는 이러한 우수한 성능을 기반으로 하여 이동통신 가입자에게 자신의 개인적 취향에 맞는 화려하고도 편리한 UI를 지닌 다양한 서비스 환경을 제공하고 있으며 BREW를 채택한 이동통신사에게는 무선 인터넷 사용률을 증대시키는 지렛대 역할을 수행해 왔다.

또한 C/C++을 최고의 개발 언어로 생각하는 고집스런 서비스 개발자들에겐 더할 나위 없는 개발 환경을 제공하여 모바일 브루(www.mobilebrew.net)와 같은 인터넷 상의 커뮤니티들을 통해 BREW 개발자 간 커뮤케이션 뿐 아니라 CP(Content Provider)와의 다양한 연결고리를 만들어 가고 있다.

BREW는 이와 같이 이동통신사, 단말 사용자, 단말 서비스 개발자 등의 단말 플랫폼 유관 분야에 다양한 잇점을 제공할 뿐 아니라 CDMA 칩을 사용하는 전 세계 어디서나 통용이 되는 장점을 기반으로 세계 1,500만 이상의 단말에 탑재, 세계 유수 이동통신사에서 서비스 중에 있다.

현재 BREW의 서버 및 단말 환경은 최초 상용 당시에 비해 보다 풍부하고 안정적인 기능과 성능을 확보하기 위해 지속적으로 진화하고 있으며 본 논문은 BREW 서비스를 위한 서버 시스템 구성 현황 및 단말 상 BREW 구동원리, 멀티미디어 제공을 위한 architecture, 효율적인 메모리 사용 원리 등에 대해 서술하였다.

2. BREW 구성

QUALCOMM에서 제공하는 BREW는 단순히 단말기를 위한 플랫폼이나 플랫폼을 기술하는 사양만을 제공하는 것이 아니라 사업자가 BREW 서비스를 제공하는 데에 필요한 모든 솔루션의 집합체이다. 즉 BREW는 그림 1과 같이 서비스에 필요한 End-to-End 솔루션을 제공하고 있다.

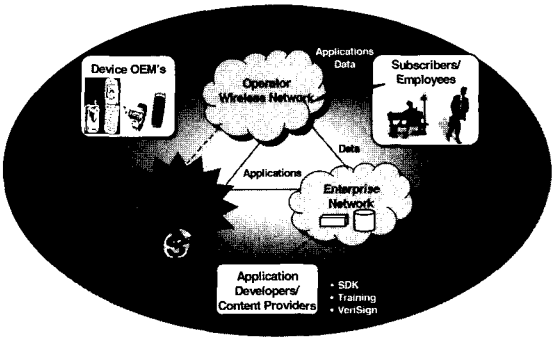


그림 1 BREW 서비스 솔루션 (Complete End-to-End Solution)

BREW 는 아래와 같이 크게 두가지 부분으로 나눌수 있다.

- BDE (BREW Development Environment)
- BDS (BREW Distribution System)

BDE는 Client 부분으로서 이의 구성 요소로는 단말기 플랫폼을 구성하는 Porting Kit, 단말기에 구현된 BREW 플랫폼을 검증하고 시험하는 Porting Evaluation Kit, 개발자들이 BREW application 개발을 위해 사용하는 SDK 및 개발자에 유용한 각종 Tool kit 등으로 구성되어 있다.

BDS는 간단히 서버 부분이라고 말할 수 있고 여기에는 그림 2와 같이 Application download server, Ap-plication manager, Transaction manager 및 빌링 시스템과 같은 물리적인 구성 요소 뿐 아니라 Application 배포 및 이와 관련된 각종 보안 시스템, Application 시험 및 관리 시스템, 사용자를 위한 extranet 등 많은 서비스 기능들을 내포하고 있다.

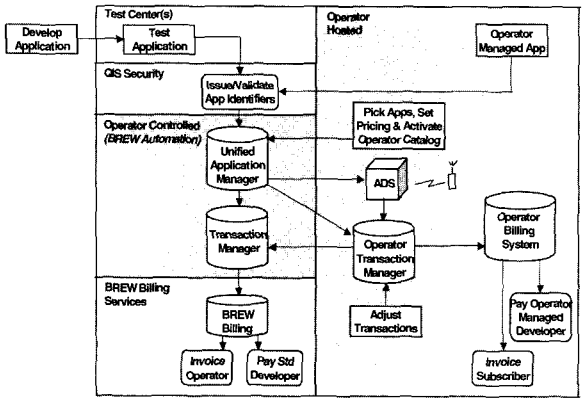


그림 2 BREW BDS 시스템

3. BREW 단말 Architecture

BREW Application들은 BDS 시스템으로부터 Handset Client 모듈중 하나인 Download engine에 의해서 Handset으로 다운로드 받을 수 있고 이를 위한 application catalog browsing이나 삭제와 같은 application 관리 등은 Mobileshop에 의해서 이루어진다. 그림 3은 이와 같은 구조를 나타낸 것이다.

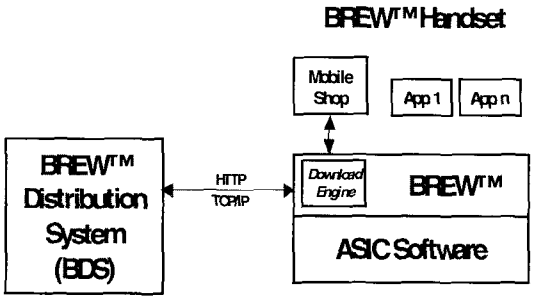


그림 3 BREW 시스템 구조

BREW Handset Client는 API Platform과 application 실행을 관리하는 부분으로 구성되어 있으며 이를 AEE (BREW Application Environment)라 부른다. 그림 4.와 같이 BREW의 경우 Native User Interface와 함께 있을 수 있기 때문에 제조업체는 기존의 code를 최대한 이용하고 보호하면서 Handset UI를 BREW application으로 만들 수 있다. 이 경우 제조업체에서는 Handset UI의 이식성 및 호환성 그리고 무선망을 통한 Dynamic upgrade 및 recall이 가능하여 Handset 개발에 많은 잇점을 가질 수 있다. 또한 ASICI에서 제공하는 TCP/IP와 같은 Native data service나 MIDI, CMX와 같은 Multimedia 그리고 BlueTooth, gpsOne 및 MSM6100의 3D engine, Qtv Qcamera 등과 같은 보다 진보된 기능을 BREW 통해 쉽게 사용할 수 있다.

BREW Client는 객체지향 모델을 채택하고 있어 API들이 object class 내에 캡슐화 되어 있으며 모든 Object들은 32bit의 AEECLSID에 의해 구분된다. 또한 File, socket, database, viewer, sound player, control 등 많은 기능들을 이미 포함하고 있어 개발자가 충분히 상위 레이어에서 application을 개발할 수 있는 OS-like 기능들을 제공하고 있다.

BREW는 그림 5와 같은 Event-driven 방식의 application model을 채택하고 있다. 이와 같은 event-driven 방식에서는 모든 callback이나 event들이 application의 task state로 post 되어 application의 복잡도를 크게 줄일 수 있고 개발자는 task 간의 복잡한 dispatching 관계를 고려할 필요가 없어 효율적인 개발이 가능하다.

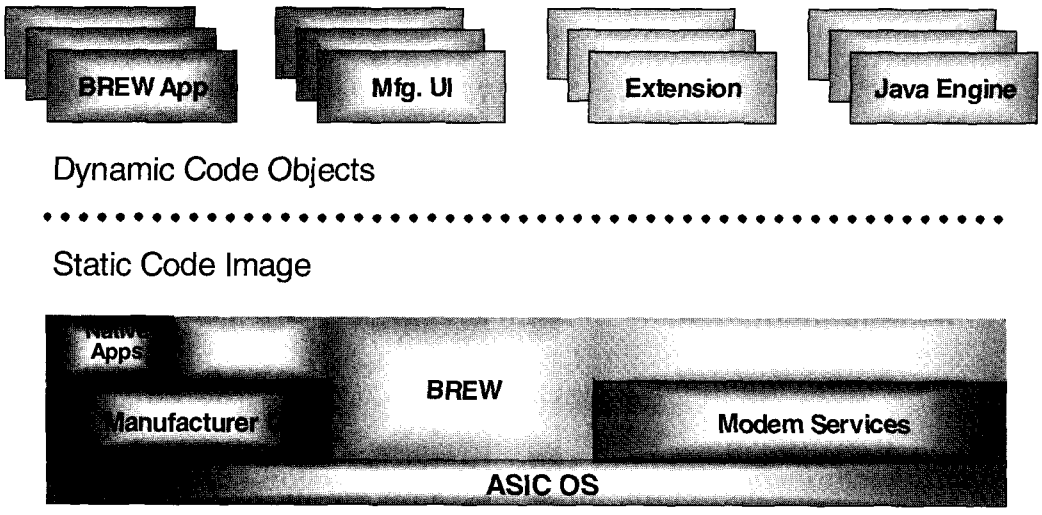


그림 4 Handset Client Architecture

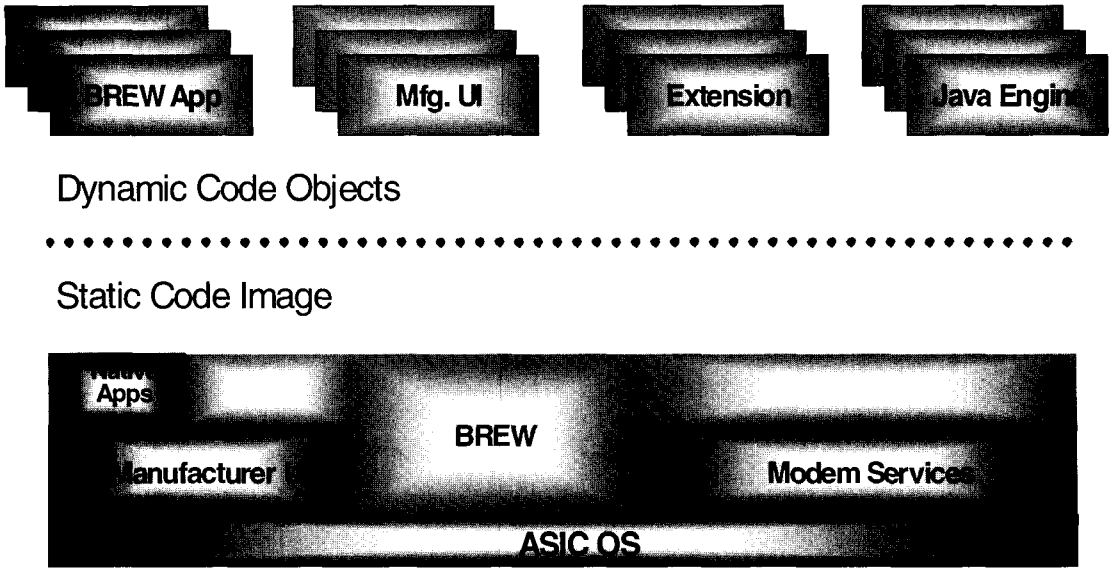


그림 5 Application Model - Events & Callbacks

BREW 는 다양한 Shell service와 API들을 개발자들에게 제공한다. Shell service로는 다음과 같은 서비스가 제공되고 있다.

〈BREW API - Shell services〉

Interface/Application Enumeration, Query, Creation
Application Support (Starting, Closing, Events, etc.)
Resource Files
Preferences
Registry Services
Timers/Alarms
Dispatching (Callbacks)

BREW에서 제공하는 기본적인 API들은 아래와 같다.

Network	HTTP/HTTPS
- Sockets	HTML/CHTML/XHTML
- DNS	Graphics
- Dormancy	- DDB/DIB
Files/Directories	- 2D Graphics
Databases	- Sprites
Compression	- 3D
Security	Media
User Interface	- PNG, BMP, JPEG, BCI
- Menus, Lists, Icon Views	- MPEG4
- Text Input	Camera/Camcorder
- Date/Time/Clock	Vocoder (voice memos, etc.)
Position Location	TAPI
Clipboard	Sounds
BlueTooth	- MIDI, MP3, QCELP

4. 멀티미디어 기능 지원

BREW에서는 멀티미디어 기능을 위하여 다양한 Multimedia API들을 제공하고 있다. 멀티미디어 어플리케이션을 지원하기 위해서는 플랫폼이 MIDI, MP3, QCP, MPEG4, AAC 등과 같은 다양한 미디어 포맷을 지원하여야 할 뿐 아니라, 다양한 포맷의 미디어들이 여러 개가 동시에 재생될 수 있어야만 한다. 또한 recording 및 encoding도 가능하여야 한다. 그리고 application 개발자들을 위해서는 멀티미디어 컨텐츠 처리가 추상화되어야 하고 복잡한 application 개발을 위한 building block들을 제공해야 한다. 컨텐츠 handler 개발자들을 위해서는 새로운 미디어 컨텐츠 handler가 쉽게 플랫폼에 plug-in 및 등록되어 플랫폼의 interface로 제공되어야 하고 application이 새로운 미디어 포맷을 즉시 인지할 수 있어야만 한다.

이와 같은 다양한 멀티미디어 기능 지원 요구를 만족시키기 위해 BREW에서는 미디어 데이터 처리를 위한 Imedia, 디바이스의 Camera 엑세스를 위한Icamera, low-level voice frame 엑세스를 위한 Ivcoder와 사운드 서비스를 위한 Isound 등과 같은 API들을 제공하고 있다. 그림 6과 7은 BREW에서 제공하는 multimedia 기능의 기본 구조이다. 여기에서 보는 바와 같이 Imedia는 미디어 컨텐츠의 추상화된 인터페이스이다. 즉 개발자는 미디어에 대한 모든 조작을 Imedia API를 통해서 할수 있다. Imedia의 파생 class들은 특정 미디어 포맷을 처리하는 컨텐츠 handler가 된다. 그리고 AEEMediaData는 미디어 데이타를 캡슐화하며 파일, 버퍼 또는 스트림으로의 입출력을 지원한다.

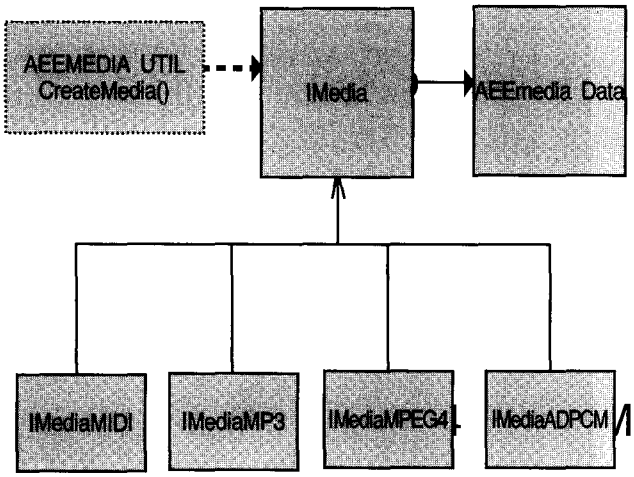


그림 6 BREW Imedia Architecture

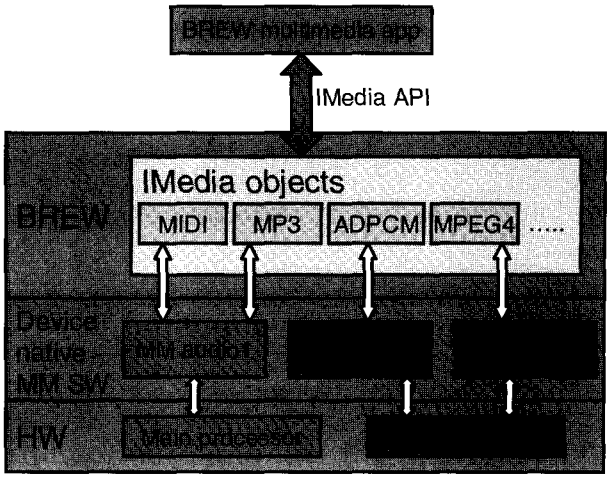


그림 7 BREW Multimedia Architecture

AEEMediaUtil_CreateMedia()는 미디어 데이터를 분석하고 미디어 타입을 알아내며 Imedia 오브젝트를 생성한다. 이때 동일 미디어 타입의 Imedia 오브젝트를 여러 개 생성할 수 있다.

그리고 BREW에서 Graphic 지원 기능으로 3D 엔진을 제공하고 있다. 현재 제공하고 있는 3D 기능은 다음과 같다.

• Basics - 16-bit z-buffering - Culling - 16-bit color depth • Shading - Smooth (Gouraud) - Flat • Viewing - Clipping planes - Zoom features - Orthographic and perspective projection	• Animation - Arbitrary matrix transformations - Scene, model and segment animation • Lighting - Ambient and diffused - Specular lighting with color - Directional lighting • Scenes - Textured backgrounds	• Textures - Support for multiple textures - Texture mapping - Texture shading - 8 bpp 128x128 and 64x256 support

5. 메모리 보호 기능

최근에 발표된 BREW 3.0에서 BREW application으로 부터 BREW나 ASIC OS (DMSS)의 코드와 데이타를 보호할 수 있는 BRIDLE (BREW Isolated Domain for Legitimate Execution)이라는 메모리 보호 기능을 도입하였다. 이 기능은 ARM 9 Core의 MSM chipset 상에서 가능한 Memory Management Unit(MMU) 서비스 기능을 이용하고 있다. MMU가 없는 ARM7 기반의 MSM 6000 계열 chipset의 경우에는 MPU를 사용해서 BREW application을 보호하는 기능을 제공한다. 메모리 보호 기능은 기본적으로 non-BREW 코드 및 데이터 메모리 영역을 보호하고 BREW application 관리 기능을 강화하기 위해서 도입되었다.

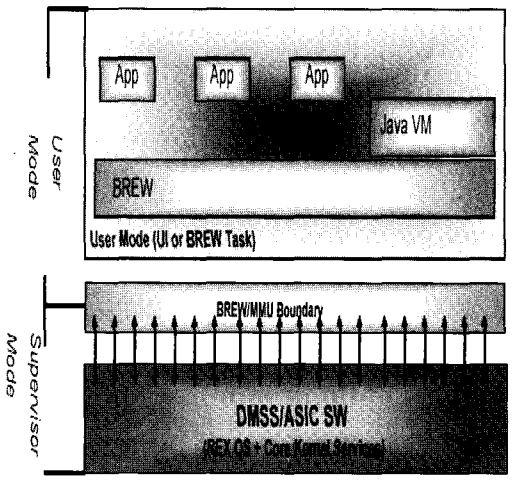


그림 8 BREW MMU Architecture

그림에서 보는 바와 같이 BREW와 ASCI OS(DMSS) 사이는 MMU에 의해 보호되고 있으며, BREW와 application 들은 유저 모드에서 그리고 non-BREW는 슈퍼바이저 모드에서 실행된다.

6. BREW Simulator

BREW는 기존의 BREW Emulator를 개선하여 실제 단말기의 기능과 성능을 보다 정확하게 emulation 할 수 있는 BREW Simulator를 버전 3.0부터 제공하고 있다. 이 Simulator는 application 시험때 실제 단말기의 필요성을 최소한으로 줄이고자 하는데 목적이 있다. 이를 위해 Simulator는 정확한 단말기 emulation을 위하여 Device pack이라 불리우는 단말기 정보 데이타들을 이용한다. 이 Device pack은 실제 단말기 정보로서 Device data 파일, 소프트웨어 emulation components와 Device의 성능 데이터 등으로 구성되어 있다. 이와 같은 단말기 정보들을 실제 단말기의 성능 시험 등으로부터 얻어진다.

Device data 파일은 Keypad mapping, Display 특성, 파일 시스템 특성, 네트웍 특성과 미디어 및 지원되는 부가장치 등에 대한 정보로 구성된다. 소프트웨어 emulation components란 단말기에서 지원되는 extension과 Imedia와 같은 단말기의 BREW Interface 들에 대한 simulation을 말한다. 그리고 Device 성능 데이타는 단말기 성능 시험툴에 의해 나온 실제 단말기의 성능 데이타로서 Display, File-system과 Database의 성능 뿐 아니라 실행 속도 성능에 대한 데이타등으로 구성되어 있다.

7. BREW Application 보안

Application이 최종 사용자에게 전달되기까지는 Application 개발 과정에서부터 시험 및 인증 그리고 application 배포 시스템을 통한 단말기로의 다운로드 등 많은 과정을 거쳐야만 한다. 이 과정에서 Hacker 등과 같은 악의를 가진 접근자에 의한 바이러스 침입이나 application 변경 등과 같은 코드 변경을 막기 위해서는 개발 과정에서 시작하여 최종 배포까지의 모든 단계에서 엄밀한 보안 정책이 수립되어 시스템으로 구현되어야 하며 예측 가능한 모든 Security hole에 대하여 방어책이 마련되어야만 한다.

BREW의 BDS는 그림 9에서 보는 바와 같이 개발자에 의한 개발 과정에서부터 application의 시험 및 인증 과정과 단말기로의 다운로드 과정에 digital key에 의한 application signing 과정을 거치도록 하여 인증 받지 못한 application이 배포 시스템을 통과하지 못하도록 하는 보안 체계를 사용하고 있다. 이에 따라 인증 받지 못한 application이나 배포 과정 내에서의 불법적인 코드 변경 등은 각 단계에서 검사되어 BDS 통하여 악의적인 코드가 최종 사용자에게 전달되는 것을 막고 있다.

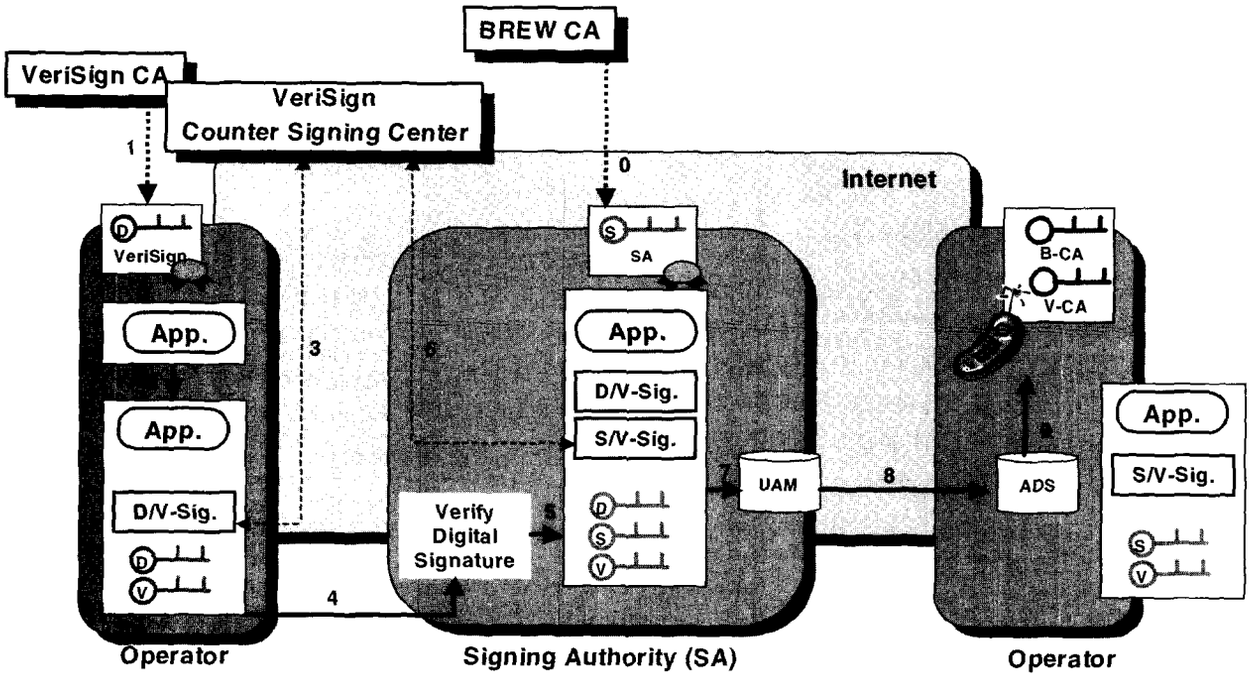
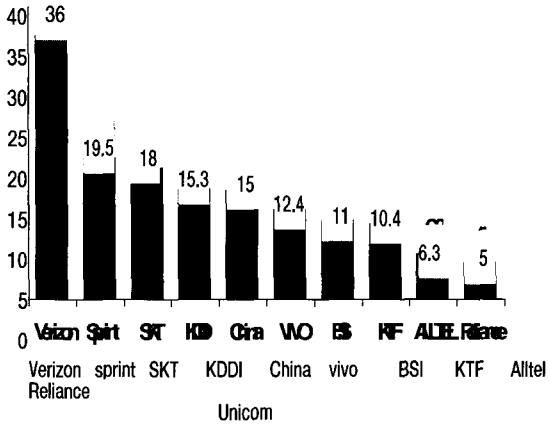


그림 9 BREW Application Security Mechanism

8. BREW 시장 동향

현재 전 세계 브루 가입자는 약 1500만 이상으로 전 세계 17개 제조업체가 80개의 브루폰 모델을 공급하고 있고 4천만 개의 애플리케이션이 다운로드 되고 있다. 세계적으로 브루를 채택한 사업자는 한국의 KTF와 미국의 버라이존 와이어리스(Verizon Wireless), 알텔(Alltel), US셀룰러(US cellular), 미드웨스트 와이어리스(Midwest Wireless), 일본의 KDDI, 중국의 차이나유니콤(China Unicom), 브라질의 (ViVo), 콜롬비아의 벨사우스(Bellsouth), 호주의 텔스트라(Telstra) 그리고 태국의 허치슨 등이 브루를 채택하여 서비스 중에 있으며 그 외에 인도의 릴라이언스, 이스라엘의 펠레폰 등이 브루를 채택해 서비스 할 예정이다. 그림 10에서 보는 바와 같이 세계 상위 10대 CDMA 사업자 중 8개 사업자가 BREW를 채택하고 있다.



Source : EMC September 2003 and company press releases

그림 10 상위 10대 CDMA 사업자

현재 한국에는 약 400여 BREW 개발업체가 있고 이들 중 200여 개 업체가 KTF에 BREW 애플리케이션을 공급, 현재 약 1000여 개의 BREW 애플리케이션이 KTF를 통해 서비스되고 있다. 한국의 BREW 개발자 중 해외 사업자에 한국의 애플리케이션을 공급하는 회사는 20여 개 업체가 넘으며, 이외에도 많은 국내 BREW 개발자들이 해외로 수출하기 위하여 준비 중이거나 추진중에 있다. 무료로 다운로드 받은 BREW SDK를 이용해 제작한 BREW 애플리케이션을 해외 각지로 수출, 수익을 창출하고, 이를 통해 회사의 가치를 높일 수 있는 현 BREW 비지니스 모델은 다른 어떤 플랫폼보다도 개발자들에게 많은 시장성과 이익을 제공하고 있다.

세계 각국의 우수 애플리케이션들이 자국 시장에 머물지 않고 전세계로 확장돼 더 많은 이용자들이 우수한 컨텐츠를 즐길 수 있도록 하고, 세계 어디서나 같은 수준의 무선 인터넷을 이용하도록 하는 것이 BREW의 본래 취지이다.

9. BREW Application 개발

9.1 BREW 개발툴

BREW application을 개발하기 위해서는 먼저 QUALCOMM의 BREW 홈페이지 (http://www.qualcomm.com/brew)로부터 SDK와 같은 개발에 필요한 툴들을 다운로드 받아야 한다. BREW SDK는 간단한 사용자 정보 등록으로 BREW 홈페이지로부터 무료로 다운로드하여 사용할 수 있다.

BREW SDK의 구성 요소는 다음과 같다.

- BREW 에뮬레이터(BREW Simulator) - BREW 에뮬레이터는 사용자에게 익숙한 MS-Windows 기반의 어플리케이션을 로드해서 시험할 수 있는 환경이다. 또한 이미 출시된 여러 개의 스킨(에뮬레이터에서의 핸드폰의 디자인을 그대로 사용할 수 있는 정보, 자세한 사항은 BREW 장치 구성자 가이드를 참조하자)을 포함하고 있다.
- BREW 장치 구성자 - BREW 장치 구성자는 BREW 에뮬레이터에서 사용되는 스킨 파일을 만드는 툴이다. 스킨 파일은 해당하는 단말기 모델에 대한 스크린 정보, 키 정보값, 메모리 크기를 가진다.
- BREW 리소스 편집기 - BREW 리소스 편집기는 응용 프로그램을 위한 리소스 데이타를 만들어 낸다. 여기서 만들어진 리소스 파일은 BREW 에뮬레이터에서 사용될 수도 있고 BREW가 탑재된 핸드셋에서 사용될 수도 있다.
- BREW MIF 편집기 - BREW MIF 편집기는 BREW에서 사용되는 모듈 정보 파일을 만들기 위한 툴이다. MIF 파일은 모듈에 대한 정보를 갖는다.
- BREW 압축 이미지 저작 도구 - 하나 이상의 작은 그래픽 이미지를 BREW 압축 이미지 파일(BCI)로 결합하여 압축할 수 있는 툴이다.
- BREW DLL들 - BREW는 Microsoft Windows 환경의 BREW 클래스를 이루는 DLL 파일을 지원한다. 이 DLL들은 Windows 환경에서 BREW 코어를 위해 사용되기 위해서 만들어진 것으로 단말기 개발 환경에서의 대응되는 라이브러리와는 차이를 갖는다.
- Win-OEM DLLs - BREW 에뮬레이터에서 실

제 단말기와 흡사한 모습을 구현하기 위해서 제공되는 하위 계층의 DLL이다.

- BREW 헤더 파일 - BREW 응용 프로그램을 작성하기 위한 C/C++ 언어용 헤더 파일이다.
- Visual Studio add-ins - Microsoft Visual Studio에서 BREW 응용 프로그램 프로젝트를 만들수 있는 프로젝트 마법사 기능을 제공한다.
- 예제 프로그램 - 예제 프로그램에는 BREW 에뮬레이터용 DLL은 물론 학습을 위하여 소스코드를 제공하며 다양한 응용 프로그래들이 포함되어 있다.
- BREW 설명서 - BREW SDK 가이드, 리소스 편집기 가이드, BREW 장치 구성자 가이드, BREW MIF 편집기 가이드 등 BREW SDK를 사용하는 데에 필요한 모든 설명서들이 포함되어 있다.

BREW SDK 이외에 QUALCOMM에서는 application 시험을 다양한 시나리오에 따라 자동적으로 수행할 수 있는 Grinder와 단말기의 실행 환경을 제공할 수 있는 Shaker란 툴들을 지원하여 개발 과정에서의 application 시험을 지원하고 있다. 이와 같은 개발 툴들에 의해 개발된 BREW application은 최종적으로 ARM Compiler인 BREW Builder에 의해서 실행 바이너리로 만들어진다. 그리고 상용화를 위해서는 아래와 같은 상용화 툴들이 필요하다.

- BREW Class ID 생성기 - Application을 위한 고유의 Class ID를 생성하는 툴
- BREW TestSig 생성기 - 개발 과정 중 시험을 위하여 applicatin이 특정한 핸드셋에서 동작할 수 있게 시험용 전자서명을 생성하는 툴
- BREW AppLoadser - 개발자가 application을 PC로부터 핸드셋으로 전송하는 툴
- BREW AppSigner - VeriSign Class 3 인증을 이용하여 application을 전자서명하는 툴

이와 같은 상용화 툴 들은 등록된 개발자들에게 BREW 웹 페이지를 통하여 무료로 배포된다.

9.2 BREW application 개발

BREW application은 아래와 같은 구성 요소로 만들어 진다.

- MIF 파일 (.mif) - BREW 모듈의 애플릿과 클래스에 대한 정보를 저장
- Application 바이너리 (.dll) - PC/BREW 에뮬레이터(Simulator)를 위한 Window DLL 형태의 바이너리
- ARM 바이너리 (.mod) - ARM compiler에 의해 컴파일되어 실제 핸드셋에서 수행 가능한 ARM용 바이너리
- 리소스 파일 (.bar)

아래 코드는 BREW application 코드의 기본 형태이고 이에 바탕을 둔 예제 프로그램을 자세한 이해를 위하여 첨부하였다.

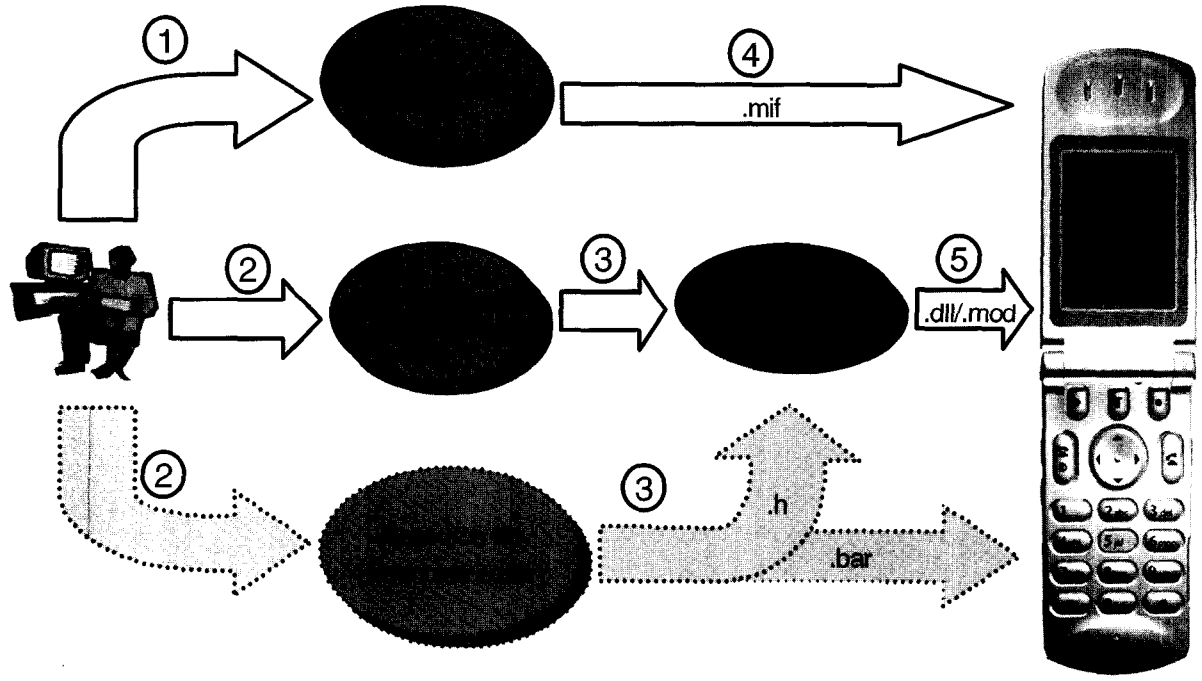


그림 11 BREW SDK 구성 요소와 application 개발

참고문헌

[1] BREW White Paper, BREW and J2ME-A Complete Wireless Solution for Operators Committed to Java, QUALCOMM.

[2] BREW White Paper, BREW Distribution System(BDS) Overview, QUALCOMM.

[3] BREW White Paper, The Road to Profit is Paved with Data Revenue, QUALCOMM.

[4] ARM Architecture Reference Manual, Advanced RISC Machines Ltd., Cambridge, UK.

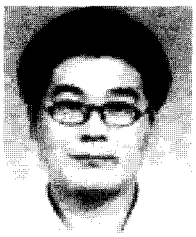
[5] 천귀호, BREW 모바일 프로그래밍, 한빛미디어, 2002.05.

[6] 송철민, 브루(BREW) 2.0, 프로그램세계, 2003.03.

임 승 혁

1999 KTF 위치 추적 서비스 개발
2000 KTF 무선인터넷 단말 웹브라이저 ME 개발
2001 KTF 멀티팩 서비스 단말 플랫폼 BREW 개발참여
2001~현재 KTF 멀티팩 서비스 단말 플랫폼 WIPI 개발
관심분야 : 단말플랫폼 기술동향
E-mail : perhaps@ktf.com

박 종 렬



1989. 2 경상대학교 전기전자공학과 (학사)
1991. 2 경상대학교 전기저자공학과 (석사)
1991~1996 기아정보시스템(주) 연구소
1997~2001 Ericsson Korea Ltd. 기술지원 및 Tecnhincal Sales
2001~현재 QUALCOMM Korea Ltd. 기술지원팀 부장
관심분야 : 단말기 플랫폼, 지능망, 데이터 통신 네트웍 구조, VoIP
E-mail : perhaps@ktf.com

BREW application 코드의 기본 형태

```
#include "AEEAppGen.h"    // Applet helper file
#include "helloworld.bid"    // Applet-specific header that contains class ID
static boolean MyApp_HandleEvent(AEEApplet * pme, AEEEvent eCode,uint16 wParam, uint32 dwParam);
int AEEClsCreateInstance(AEECLSID ClsId,IShell * pIShell,IModule * pMod,void ** ppObj)
{
   *ppObj = NULL;

   if(AEEApplet_New( sizeof(AEEApplet),              // Size of our private class
               ClsId,                    // Our class ID
               pIShell,                  // Shell interface
               pMod,                     // Module instance
               (IApplet**)ppObj,     // Return object
               (AEEHANDLER)MyApp_HandleEvent, // Our event handler
               NULL))                // No special "cleanup" function
    return(AEE_SUCCESS);
         return (EFAILED);
}
static boolean MyApp_HandleEvent(AEEApplet * pMe, AEEEvent eCode, uint16 wParam, uint32 dwParam)
{
  switch (eCode){
   case EVT_APP_START:
     return(TRUE);
  }
  return(FALSE);
}
```

〈BREW application 예제 코드〉

```
/*=============================================================================
FILE: Helloworld.c

SERVICES: Sample applet using AEE

DESCRIPTION
  This file contains a very simple sample application that displays the
  "Hello World" on the display.

PUBLIC CLASSES:
  N/A

INITIALIZATION AND SEQUENCING REQUIREMENTS:
   The following explanation applies to this sample containing one applet, which serves
   as a base for app developers to create their own applets using AEE Services:

   In the applet source file (like this one), include AEEAppGen.h.
```

Mandatory Sections in Applet Source (this file):

Following Mandatory Sections are required for each applet source file.
(Search for "Mandatory" to identify these sections)

Includes:
Copy this section as-is from the sample applet. It contains:
AEEAppGen.h: For AEEApplet declaration

Type Declarations:
A data structure is usually defined to hold the app specific data. In this structure, the first element must be of type AEEApplet. Note that this simple example does not require any additional data, so this step has been removed.

Functions: (For more details, see corresponding function description in this applet)

App_HandleEvent(): This function is the Event Handler for the applet.
Copy the function outline from the sample applet and add app specific code.

AEEClsCreateInstance(): This function is called to create an instance of the applet.
It is called by AEEModGen when applet is being created.

Important Notes:

1. DO NOT use any "static data" in the applet. Always use the functions exported by AEEStdlib or by IHeap services to dynamically allocate data and make it a member of the applet structure.
2. DO NOT include and link "standard C library". Use AEE Memory Services (in AEEHeap.h) and Standard Library macros(in AEEStdLib.h).
   For example, use MALLOC() to allocate memory, WSTRCPY() to make a copy of Unicode (wide) string.
3. BREW is Unicode(wide string) compliant ONLY (no ISOLATIN1/ANSI) except for file names which are ISOLATIN1/ANSI.
   ALWAYS USE AECHAR instead of "char". Use string services provided in AEEStdLib.h for string manipulation.
4. It is always a good idea to DEFINE RESOURCES using BREW ResourceEditor. Make Strings, Bitmaps, Dialogs, etc.
   as resources. ResourceEditor saves resources as .bri file, generates resource header file and compiles .bri into a .bar binary file, which can be used by the applet.

Miscellanoeus Notes:

1. Make sure that the class ID used for the app is the same as that defined corresponding in .MIF file
2. Always make sure that compiled resource (.bar) file and corresponding resource header (a) reside in app directory and (b) are included in the applet code.
   Define a constant APP_RES_FILE containing the name of the compiled resource file (with .bar extension).

More than one applet:

If more than one applet needs to be defined, then do the following

(1) Follow the above description for each applet

```
        Copyright ?2000-2002 QUALCOMM Incorporated.
                All Rights Reserved.
          QUALCOMM Proprietary/GTDR
=============================================================*/

/*=============================================================
INCLUDES AND VARIABLE DEFINITIONS
=============================================================*/
#include "AEEAppGen.h"        // Applet helper file
#include "helloworld.bid"          // Applet-specific header that contains class ID

/*-------------------------------------
Static function prototypes
-------------------------------------*/
static boolean HelloWorld_HandleEvent(AEEApplet * pme, AEEEvent eCode,uint16 wParam, uint32 dwPar-
am);

/*=============================================================
FUNCTION DEFINITIONS
=============================================================*/
/*=============================================================
```

FUNCTION: AEEClsCreateInstance

DESCRIPTION

This function is invoked while the app is being loaded. All Modules must provide this function. Ensure to retain the same name and parameters for this function.
In here, the module must verify the ClassID and then invoke the AEEApplet_New() function that has been provided in AEEAppGen.c.

After invoking AEEApplet_New(), this function can do app specific initialization. In this example, a generic structure is provided so that app developers need not change app specific initialization section every time except for a call to InitAppData(). This is done as follows: InitAppData() is called to initialize AppletData instance. It is app developers responsibility to fill-in app data initialization code of InitAppData(). App developer is also responsible to release memory allocated for data contained in AppletData — this can be done in FreeAppData().

PROTOTYPE:

int AEEAppCreateInstance(AEECLSID clsID, IShell* pIShell, IModule* pIModule,IApplet** ppApplet)

PARAMETERS:

clsID: [in]: Specifies the ClassID of the applet which is being loaded

pIShell: [in]: Contains pointer to the IShell interface.

pIModule: pin]: Contains pointer to the IModule interface to the current module to which this app belongs

ppApplet: [out]: On return, *ppApplet must point to a valid AEEApplet structure. Allocation

of memory for this structure and initializing the base data members is done by AEEApplet_New().

DEPENDENCIES
none

RETURN VALUE
SUCCESS: If the app needs to be loaded and if AEEApplet_New() invocation was successful
EFAILED: If the app does not need to be loaded or if errors occurred in AEEApplet_New().
If this function returns FALSE, the app will not be loaded.

SIDE EFFECTS
none
==========*/
```
int AEEClsCreateInstance(AEECLSID ClsId,IShell * pIShell,IModule * pMod,void ** ppObj)
{
   *ppObj = NULL;

   if(AEEApplet_New( sizeof(AEEApplet),    // Size of our private class
            ClsId,                         // Our class ID
            pIShell,                       // Shell interface
            pMod,                          // Module instance
            (IApplet**)ppObj,              // Return object
            (AEEHANDLER)HelloWorld_HandleEvent, // Our event handler
            NULL))                         // No special "cleanup" function
     return(AEE_SUCCESS);

         return (EFAILED);
}
```

/*==========

FUNCTION HelloWorld_HandleEvent

DESCRIPTION
This is the EventHandler for this app. All events to this app are handled in this function. All APPs must supply an Event Handler.

Note - The switch statement in the routine is to demonstrate how event handlers are generally structured. However, realizing the simplicity of the example, the code could have been reduced as follows:

```
if(eCode == EVT_APP_START){
   IDISPLAY_DrawText();
   IDISPLAY_Update();
   return(TRUE);
}
return(FALSE);
```

However, while doing so would have demonstrated how BREW apps can be written in about 8 lines of code (including the app creation function), it might have confused those who wanted a bit more practical example.

Also note that the use of "szText" below is provided only for demonstration purposes. As indicated in the documentation, a more practical approach is to load text resources from the applicaton's resource file.

Finally, note that the ONLY event that an applet must process is EVT_APP_START. Failure to return TRUE from this event will indicate that the app cannot be started and BREW will close the applet.

PROTOTYPE:
boolean HelloWorld_HandleEvent(IApplet * pi, AEEEvent eCode, uint16 wParam, uint32 dwParam)

PARAMETERS:
pi: Pointer to the AEEApplet structure. This structure contains information specific to this applet. It was initialized during the AppCreateInstance() function.

ecode: Specifies the Event sent to this applet

wParam, dwParam: Event specific data.

DEPENDENCIES
none

RETURN VALUE
TRUE: If the app has processed the event
FALSE: If the app did not process the event

SIDE EFFECTS
none
==========*/

```
static boolean HelloWorld_HandleEvent(AEEApplet * pMe, AEEEvent eCode, uint16 wParam, uint32 dwParam)
{
  AECHAR szText[] = {'H','e','l','l','o',' ','W','o', 'r', 'l', 'd', '\0'};

  switch (eCode){
   case EVT_APP_START:
    IDISPLAY_DrawText(pMe->m_pIDisplay,  // Display instance
            AEE_FONT_BOLD,   // Use BOLD font
            szText,       // Text - Normally comes from resource
            -1,           // -1 = Use full string length
            0,            // Ignored - IDF_ALIGN_CENTER
            0,            // Ignored - IDF_ALIGN_MIDDLE
            NULL,       // No clipping
            IDF_ALIGN_CENTER | IDF_ALIGN_MIDDLE);
    IDISPLAY_Update (pMe->m_pIDisplay);
                        return(TRUE);
    case EVT_APP_STOP:
     return(TRUE);


    default:
      break;
   }
   return(FALSE);
}
```