임베디드 시스템을 위한 K 가상 머신 사운드 API 설계 및 구현*

전상호° 이철훈
충남대학교 컴퓨터공학과
(shjun°, clee)@cnu.ac.kr

Design and Implementation of K Virtual Machine Sound API for Embedded Systems

Shang-Ho Jeon°, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

최근 임베디드 디바이스에 여러 가지 장점을 제공하는 자바기술은 필수적인 요소가 되었다. 임베디드 디바이스에 적용되는 자바기술은 J2ME 플랫폼이며, 이는 K 가상 머신(K Virtual Machine)의 핵심인 CLDC(Connected Limited Device Configuration)와 그래픽 유저 인터페이스, 네트워크 API, 사운드 API 등을 명세하고 있는 MIDP(Mobile Information Device Profile)로 구성되어 있다. 이 중 그래픽 유저 인터페이스와 네트워크, 사운드 부분은 구현 시 시스템에 의존적인 부분을 따로 구현해야 하는데, 이는 네이티브(native)함수로 구현할 수 있다. 본 논문에서는 J2ME 플랫폼에서 정의된 사운드 API 의 기능들을 분석하여 임베디드 시스템에 적합한 사운드 API 의 네이티브 함수를 구현하였다.

1. 서론

최근 들어 자바의 플랫폼 독립성, 보안성, 네트워크 이동성, 실행코드의 재사용성, 작은 실행 파일 크기, 동적 적응성, 이식성, 개발의 용이성 등은 소형 기기에 적용 시 개발 비용 감소와 유지보수 같은 여러 가지 장점이 있기 때문에, 자바를 임베디드 장치와 같은 소형 기기에 적용하기 위한 연구들이 다방면에서 진행되어 왔다[1].

특히 핸드폰이나 PMP 등의 소형 기기에서는 영화, 게임 등의 미디어 관련 응용 프로그램을 실행 시키기 위해서 사운드 출력의 기능 구현이 필수적으로 이루어져야 한다. 이러한 소형 기기에 탑재되는 자바 가상 머신에서 자바 API 를 통해 사운드 출력을 제공하기 위해서는 시스템에 의존적인 부분을 따로 구현해야 하는데, 이는 네이티브 함수로 구현할 수 있다.

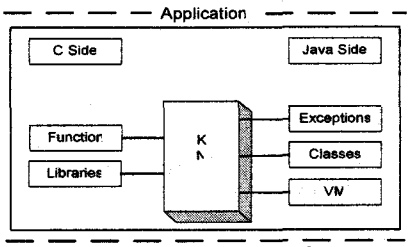
본 논문에서는 실시간 운영체제인 UbiFOS™상에서 K 가상 머신의 사운드 출력을 위한 네이티브 함수를 구현하였다.

본 논문의 2 장에서는 KNI 의 역할과 J2ME 플랫폼의 CLDC(Connected Limited Device Configuration), MIDP(Mobile Information Device Profile)에 대한 관련 연구, 3 장에서는 임베디드 시스템의 자바 사운드 API 설계 및 구현, 4 장에서는 실험 환경 및 결과, 5 장에서는 결론 및 향후 연구 과제에 대해 기술한다.

2. 관련 연구

2.1 KNI (K Native Interface)

KNI는 [그림2-1]에서 보는 바와 같이 자바와 자바 이외의 언어로 만들어진 애플리케이션이나 라이브러리가 상호 작동할 수 있도록 연결시켜주는 인터페이스로 자바에서 지원하지 못하는 플랫폼 종속적인 기능들과 이미 다른 언어로 만들어진 애플리케이션이나 라이브러리를 사용할 수 있도록 해주는 임베디드 시스템을 위한 인터페이스이다. 네이티브 언어를 이용한 자바 프로그램에서 자바로 된 부분은 여전히 플랫폼 독립적이지만 네이티브 언어로 된 부분은 호스트 환경에 맞게 다시 컴파일 되어야 한다[2].

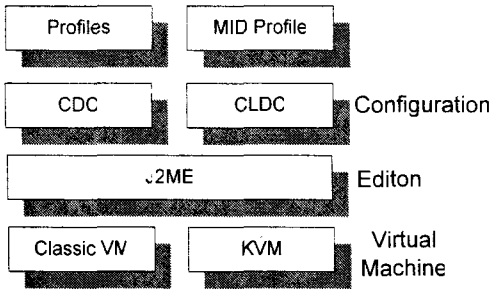


[그림 2-1] KNI의 구성도

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임

2.2 J2ME

J2ME 플랫폼은 다양한 제품과 임베디드 기기 시장을 목표로 예를 들어 셀룰러 폰, 팜, 인터넷 TV등을 위한 플랫폼을 규정하고 있다. J2ME 플랫폼은 실행 및 개발환경으로 KVM, 애플리케이션 프로그램 API Library, Deployment와 Configuration tool을 제공한다.



[그림 2-2] J2ME

[그림 2-2]에서 자바의 하이-레벨 구조를 보면 가상 머신(Virtual Machine) 위에 Configuration과 Profile이 수직적 구조로 이루어져 있다.

SUN사의 J2ME 플랫폼은 다양한 제품과 임베디드 디바이스 시장을 목표로 플랫폼을 규정하고 있다.

J2ME 플랫폼에서는 CLDC와 CDC(Connected Device Configuration)가 사용되고 있으며, Profile로는 MIDP만이 사용되고 있다[3].

2.3 MIDP

MIDP는 MID를 목표로 설계된 자바 클래스 라이브러리에 대한 명세로 CLDC를 기반으로 하고 있다. 그러므로 MIDP 명세는 CLDC의 명세를 구체화하거나, 확장 및 변경할 수 있다[2].

MIDP 명세는 애플리케이션의 모델, 유저 인터페이스와 이벤트 핸들링, 영속적인 저장공간, 네트워킹, 타이머 지원 등을 정의 하고 있다. 본 논문에서는 사운드를 구현하기 위해 Media API만 소개한다[2].

2.3.1 MIDP 2.0 Media API

MIDP 2.0 Media API는 직접 호환 가능한 Mobile Media API (JSR-135) 사양의 빌딩 블록이다. 이 빌딩 블록은 전체 Multimedia API와의 상위 버전 호환성을 유지하면서 해당 사양에 사운드 지원을 포함하려는 J2ME 프로파일(Profile)용으로 사용된다. 이러한 사양의 예로는 MIDP 2.0 (JSR-118)이 있다. 상호 운영되는 이러한 두 API의 개발로 인해 동일한 API 원리를 사용하는 J2ME 플랫폼 제품군에서 완벽한 사운드와 멀티미디어 내용 작성이 가능하다.

J2ME는 단순 톤(ton) 재생 기능을 가진 휴대폰에서 고급 오디오 및 비디오 렌더링 기능이 있는 PDA 및 웹 태블릿에 이르기까지 다양한 장치를 대상으로 한다. 다양한 구성과 멀티미디어 처리 기능을 수용하려면 추상화 수준이 높은 API 가 필요하다. MMAPI Expert Group 의 활동 목표는 광범위한 응용 프로그램 영역을 다루는 것이며 그 활동 결과로 만들어진 두 개의 API 집합을 제안한다.

- Mobile Media API (JSR 135)
- MIDP 2.0 Media API

Mobile Media API는 모바일 폰, PDA 및 셋톱 박스를 포함하여 고급 사운드 및 멀티미디어 기능이 있는 J2ME 플랫폼을 위한 것이다. MIDP 2.0 Media API는 직접 호환되는 Multimedia API의 하위 집합으로 범용 시장 모바일 장치(MIDP 2.0 실행)와 같이 자원이 제약된 장치를 위한 것이다. 또한 이 하위 집합 API는 사운드 지원을 필요로 하는 다른 J2ME 프로필에 적용될 수 있다.

일부 J2ME 플랫폼은 자원이 매우 제약되어 있으므로 일부 휴대폰의 비디오와 같은 모든 멀티미디어 유형을 지원하지 못할 수도 있다. 따라서 모든 장치가 사용자 정의 프로토콜과 내용 유형을 지원하는 확장 가능성과 같은 멀티미디어 API 의 모든 내용을 지원할 필요는 없다.

제시된 빌딩 블록 하위 집합 API 는 위의 제약 조건에 부합하도록 설계되어있다. 이 제시된 빌딩 블록은 MIDP 2.0 Expert Group 의 요구 사항을 충족한다. 이러한 요구 사항은 다음과 같다.

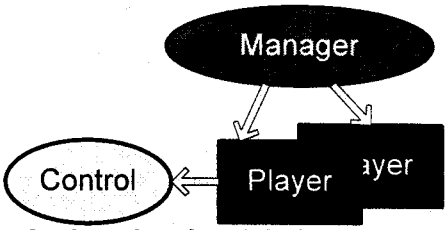
- 낮은 풋프린트(foot print) 오디오 재생
- 알 수 없는 프로토콜 및 내용 형식
- 톤 재생 지원
- 일반 미디어 흐름 제어지원 : 시작, 정지 등
- 미디어별 유형 제어 지원 : 볼륨 등
- 기능 쿼리 지원

이 하위 집합은 다음과 같은 면에서 전체 Mobile Media API 와 다르다.

- 오디오 전용
- 비디오나 그래픽 관련 제어를 모두 제외
- 사용자 정의 DataSources 를 통한 사용자 정의 프로토콜을 지원하지 않음

MIDP 2.0 에 사용된 빌딩 블록 하위 집합은 전체 Mobile Media API 에 적합한 하위 집합으로 향후 버전과 완전 호환된다. 전체 Mobile Media API 기능을 MIDP 2.0 에서 사용하려면 해당 API 에서 추가 클래스와 메소드를 구현하기만 하면 된다.

오디오 빌딩 블록 시스템은 [그림 2-3]과 같이 세 가지 주요 부분으로 구성되어 있다.



[그림 2-3] 오디오 빌딩 블록 시스템

Manager 는 오디오 자원의 최상위 컨트롤이다. 응용 프로그램은 Manager 를 사용하여 Player 를 요청하고 등록 정보, 지원되는 내용 유형 및 지원되는 프로토콜을 요청한다.

Player 는 멀티미디어 내용을 재생한다. 응용 프로그램은 Manager 에 위치 정보가 있는 문자열을 제공하여 Player 를 얻는다.

Control 은 Player 가 가질 수 있는 서로 다른 제어를 모두 구현하는 데 사용되는 인터페이스이다. 응용 프로그램은 지원하는 제어의 Player 를 요청한 다음 VolumeControl 과 같은 볼륨을 제어할 특정 Control 을 요청할 수 있다[4].

3. 설계 및 구현

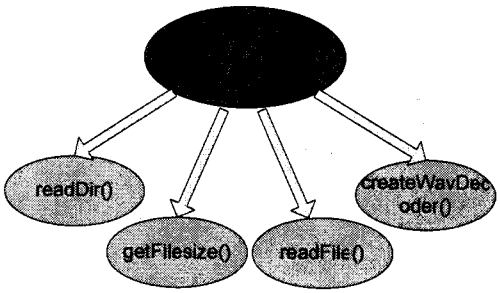
본 논문에서는 실시간 운영체제인 UbiFOS™상에서 KVM 의 사운드 출력을 위한 네이티브 함수들을 구현하였다. 이 네이티브 함수들은 크게 두 가지의 모듈로 구성된다. 사운드 출력을 위해 사운드 파일을 오픈하는 모듈과 오픈된 파일을 KVM 이 탑재된 임베디드 사운드 시스템에 출력하는 모듈로 구성된다.

3.1 사운드 파일 오픈 모듈 구현

사운드 파일 오픈 모듈은 사운드 출력에 앞서 출력될 사운드 파일의 정보를 얻어오며, 파일을 읽어 배열에 저장하는 역할을 한다.

또한 디지털 코드화된 사운드 파일을 아날로그 코드로 변경시켜주기 위한 decoder 생성의 역할도 담당한다.

[그림 3-1]은 파일 오픈을 위해 구현한 함수들의 호출 관계이다.



[그림 3-1] 파일 오픈 모듈

readDir() 함수는 재생할 사운드 파일의 정보를 저장하고 있는 구조체의 정보를 포인터를 리턴해 준다. getFilesize() 함수는 사운드 파일의 크기를 얻어오며, readFile() 함수는 사운드 파일을 읽어 배열에 저장하는 역할을 한다. createWavDecoder() 함수는 decoder 을 생성한다. audioOpen() 함수는 사운드 파일의 정보에 따라 위의 4 개의 함수의 파라미터를 제어하며 호출한다.

[그림 3-2] 는 audioOpen()함수의 일부이다.

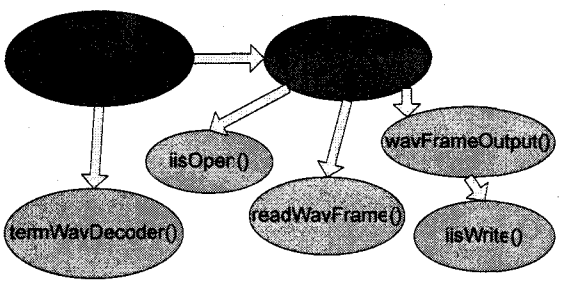
```
int audioOpen(int sRate, int bits, int channels, int isSigned, int
        endianess)
{
        int ret;
        char* fname;
        fs_t* pFs;

        pFs= read_dir(MUSIC_PROGRAM_TYPE);
        ...
        strcpy(curFile,fname);
        curDataLen= getFilesize(curFile);

        ret=readFile(curFile,curData,curDataLen);

        pDec= createWavDecoder();

}
```

[그림 3-2] audioOpen() 함수 일부

3.2 사운드 출력 모듈 구현

사운드 출력 모듈은 배열로 읽어온 사운드 정보를 Inter-IC SOUND(IIS) BUS INTERFACE 를 통해 스피커로 전달함으로써 사운드 출력을 담당한다. 스피커로 전달되는 입력은 아날로그의 형태여야 하는데, 파일 오픈 모듈에서 생성한 decoder 를 이용해서 디지털 코드를 아날로그 코드로 변경해준다.

[그림 3-3]은 사운드 출력을 위해 구현한 함수들의 호출 관계이다.



[그림 3-3] 사운드 출력 모듈

decodeWavDecoder() 함수는 배열에 저장된 디지털 코드로 된 사운드 정보를 decoder 를 통해 아날로그 코드로 변경시켜 주는 역할을 한다. 또한 iis_open() 함수를 호출하여 IIS 버스를 오픈 해 준다.

readWavFrame() 함수는 사운드 출력 단위인 프레임을 하나씩 읽어오며, wavFrameOutput() 함수는

읽어온 프레임을 IIS BUS 를 통해 스피커로 보내는 역할을 하는데, 이 때 IIS BUS 를 통해 사운드 정보를 전달하는 함수인 iisWrite() 함수를 호출한다. readWavFrame() 함수와 wavFrameOutput() 함수가 하나의 프레임을 처리할 때마다 계속 호출이 일어나면서 스피커를 통해 사운드가 출력된다.

프레임을 모두 출력한 후에는 termWavDecoder() 함수를 호출하여, decoder 가 사용한 메모리는 반납한다.

[그림 3-4] 는 audioOpen()함수의 일부이다.

```
int decodeWavDecoder(void* p,char* playfile,void* rwOps,
        playercb_t playercb,void* arg)
{
        wav_decoder_t* pdec= (wav_decoder_t*)p;
        int ret= 0;
        int bitdepth,samplerate;
        int channels,framecount;
        ..
        pdec->filelen= getWavFileinfo(pdec->fd);
        ..
        getWaveFileinfo(pdec->fd,&bitdepth,
                        &samplerate,&channels);
        ..
        iisOpen(channels,bitdepth,samplerate);

        do {

                continue;

                ret= readWavFrame(pdec->fd,pdec);

                ret= wavFrameOutput(pdec,playercb);

        } while(1);
        ..
}
```

[그림 3-4] audioOpen() 함수 일부

3.3 사운드 컨트롤 구현

사운드 출력 도중에 출력을 컨트롤하는 부분이 있어야 함으로, memSeek() 함수와 memRead()함수, memWrite() 함수, memClose() 함수 등을 구현하여 사운드 출력 중에 컨트롤이 가능하도록 하였다. 사운드 출력 애플리케이션의 실행 도중에 위 함수들을 호출하기 위해서 함수 포인터를 사용하였다. 이를 위해 위 함수들의 포인터를 저장하기 위한 MG_RWops 구조체를 선언하였다.

(그림 3-5)는 함수 포인터 저장을 위해 선언된 MG_RWops 구조체이다.

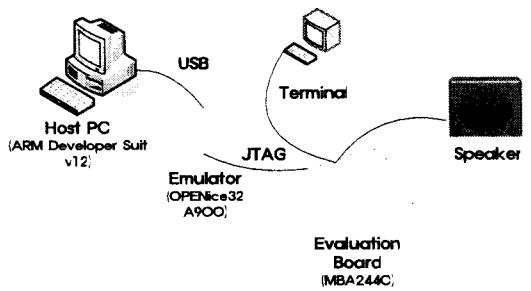
```
typedef struct MG_RWops {
    int (*seek)(struct MG_RWops *context, int offset, int whence);
    int (*read)(struct MG_RWops *context, void *ptr, int objsize,
            int num);
    int (*write)(struct MG_RWops *context, const void *ptr, int
            objsize, int num);
    int (*close)(struct MG_RWops *context);
    int (*eof)(struct MG_RWops *context);
    Uint32 type;
    union {
        struct {
            int autoclose;
            FILE *fp;
        } stdio;
        struct {
            Uint8 *base;
            Uint8 *here;
            Uint8 *stop;
        } mem;
        struct {
            void *data1;
        } unknown;
    } hidden;
} MG_RWops;
```

(그림 3-5) MG_RWoops 구조체

4. 테스트 환경 및 결과

본 논문의 테스트 환경은 MBA2440 보드상에 실시간 운영체제로 UbiFOS™를 사용하였고, 개발도구로 ARM SDT v2.51 을 사용하였다. 또한 OPENice-A900 을 사용하여 디버깅 하였다.

[그림 4-1]은 테스트 환경을 나타내고 있다.

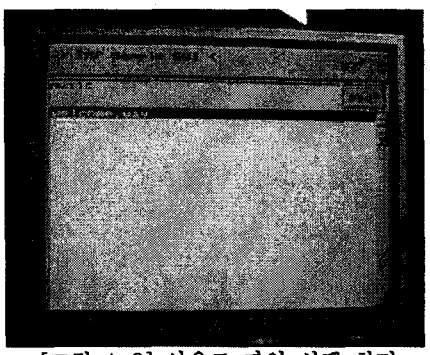


[그림 4-1] 테스트 환경

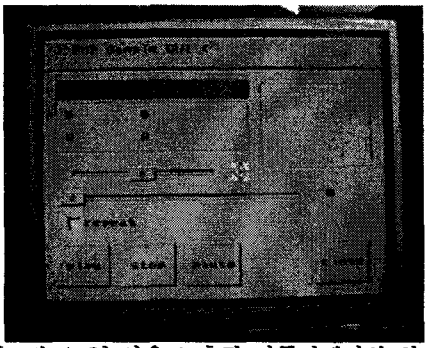
테스트는 하나의 WAV 파일을 스피커를 통해 출력하여 확인하는 방법으로 수행하였다.

MBA2440 보드에서 KVM 을 실행 한 후, 보드 LCD 에 보여지는 파일을 선택하여 사운드 파일을 재생한다. UbiFOS™가 아직 파일 시스템을 지원하지 않아 사운드 파일은 보드 메모리의 특정 주소에 직접 올리는 퓨징(fusing) 기법을 사용하여 처리되었다.

[그림 4-2]는 사운드 파일을 선택하는 화면이며, [그림 4-3]은 MBA2440 보드에서 사운드 출력 애플리케이션 실행 화면이다.



[그림 4-2] 사운드 파일 선택 화면



[그림 4-3] 사운드 출력 어플리케이션 화면

play 버튼을 누르면 퓨징 기법으로 이미 메모리에 올라가있는 사운드 파일이 출력이 된다. 본 논문의 테스트에서는 사운드 출력과 일시 정지, 정지 기능 모두 제대로 작동되는 것을 확인할 수 있었다.

5. 결론 및 향후 연구과제

본 논문에서는 제한된 리소스를 사용하는 임베디드 시스템의 사운드 기능을 제공하기 위하여 UbiFOS™상에서 KVM 의 사운드 API 와의 연동을 위한 네이티브 함수들을 구현하였으며, 테스트 결과 사운드 출력이 제대로 되는지 확인하였다.

하지만 본 논문에서 구현한 네이티브 함수는 WAV 포맷의 파일만을 지원하며, 톤 재생 기능은 지원하지 않는다. 그러므로 향후 연구과제는 WAV 포맷 파일 뿐만 아니라, MP3, MIDI 포맷등 다양한 포맷의 사운드 파일도 지원되도록 하고, 톤 재생이 가능하도록 하는 것이다.

참고 문헌

[1] 전상호 정근재 이정원 이철훈, *The Design and Implem entation of KVM Network for Embedded Systems*" 정보과학회, 2006

[2] sun Microsystems, Inc., K Native Interface Specification, 2002.

[3] Sun Microsystems, " *Mobile Information Device Profile(JSR-37)*" , 2000

[4] Sun Microsystems, " *Mobile Media API (JSR-135)*" , 2002