멀티플랫폼상에 가상 기계 탑재를 위한 어댑터의 설계

오세중*, 고광만
상지대학교 컴퓨터정보공학부
e-mail:{na97oribal, kkman}@mail.sangji.ac.kr

Design of the Adapter for Virtual Machine Porting on the Multi-platform

Se-Jung Oh*, Kwang-Man Ko
School of Computer and Information Engineering,
Sang-Ji University

요 약

가상 기계는 다양한 종류의 플랫폼에 대한 독립성을 지원하는 프로그램 실행 환경으로서 로더/링커, 인터프리터 및 가상 기계를 특정 시스템에 탑재하기 위한 어댑터로 크게 구성되어 있다. 본 논문에서는 가상 기계를 특정 시스템에 효율적으로 탑재하기 위한 어댑터를 개발하기 위해 기존 가상 기계의 탑재 부분을 고찰한 후 멀티플랫폼에 효과적으로 탑재할 수 있는 어댑터를 설계하였으며 제안된 어댑터는 정의 부분, 주 어댑터, 네이티브 인터페이스 부분으로 구성되어 있다. 또한 어댑터 개발시에 장치 특성, 입출력 등을 위한 API 작성에 필요한 요소들을 제시하였다.

1. 서론

가상 기계는 언어에 대한 장치 독립성 또는 플랫폼 독립성(Write Once Run Anywhere; WORA)을 지원하는 프로그램 실행 환경이다. 즉, 가상 기계는 다양한 종류의 플랫폼에서 작동될 수 있어야 하므로 가상 기계의 역할은 프로그램을 특정 기계의 운영체제가 이해 할 수 있도록 변환 해주는 일종의 번역기라고 볼 수 있다. 현재까지 자바 언어를 위한 JVM, KVM 등이 다양한 환경에서 사용되고 있으며 유사한 가상 기계가 개발되어 사용되고 있다.

실제로 가상 기계를 특정 플랫폼에 원활히 탑재하기 위해서는 여러 가지의 제약 사항이 존재한다. 특히 가상 기계를 모바일 장치에 탑재하는 경우에는 윈도우즈나 유닉스 환경에서 고려하지 않아도 되었던 제약 사항이 존재한다. 즉, 크기도 작아야하고 파일 시스템이 없는 경우도 고려되어야 한다[3].

본 논문에서는 개발된 가상 기계를 특정 플랫폼에 탑재 시에 제기되는 고려 사항과 썬 마이크로시스템사의 JVM, KVM에 대한 분석과 WABA Soft의 WABA 가상 기계를 기반으로 용이한 탑재를 위한 어댑터를 설계하였다[1][6].

본 논문의 구성은 제 2장에서 본 연구에 관련된 기반 연구의 내용이며 제 3장에서는 멀티플랫폼상에 가상 기계를 탑재하기 위한 어댑터를 설계하였다. 4장에서는 장치 특성, 입출력 등을 위한 API 작성에 필요한 요소를 제시하였으며 5장에서는 결론 및 향후 연구 방향을 기술하였다.

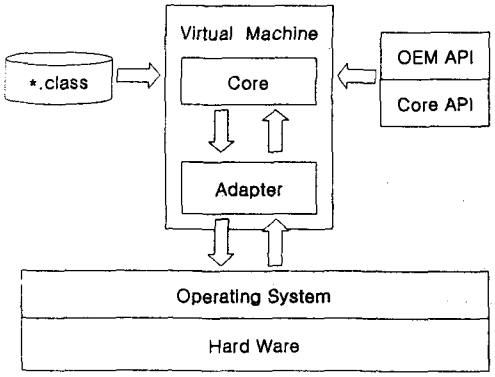
이 논문은 한국과학재단의 특정기초연구(과제번호: R01-2002-000-00041-0)지원에 의한 것임.

2. 기반 연구

2.1 어댑터(Adapter)

가상 기계의 탑재를 위해서는 가상 기계를 특정 시스템에 적합하도록 작성해 줘야 한다. 예를 들어, 자바 소스 프로그램은 플랫폼에 독립적이지만 소스 프로그램을 실행하는 JVM은 특정 플랫폼에 의존적이므로 특정 플랫폼에 적합하도록 작성되어야 한다. 이렇게 특정 시스템에 적합하도록 가상 기계를 개발하면 개발 과정에서 공통적인 부분을 발견할 수 있다. 인터프리터와 시스템 호출 함수를 사용하지 않고 작동되어지는 부분은 비슷한 수준의 시스템이라면 어디든 그대로 이식할 수 있다. 이런 공통된 부분을 따로 작성하게 되면 이것이 가상 기계의 코어

(Core)가 된다. 이 코어 부분을 제외한 나머지 부분이 코어와 운영체제를 연결하고 코어에서 내려오는 코드를 시스템의 고유 함수 즉, 네이티브 코드로 번역해 주는 일종의 어댑터가 된다. 이렇게 되면 코어 부분 역시 어댑터에 의해서 장치 독립성을 가지게 된다. 그림 1은 코어와 어댑터의 관계이다.



[그림 1] 코어와 어댑터의 관계

2.2 탑재를 위한 요소

가상 기계를 플랫폼에 탑재하는 주체는 어댑터이며 어댑터는 가상 기계를 장치로 이식했을 때 운영체제 위에서 아무 이상 없이 수행 될 수 있는 응용소프트웨어로 만들어 주는 기능을 수행한다. 어댑터는 기초적인 설정들을 위한 정의 부분(Define), 운영체제와 가상 기계의 연동을 위한 주 어댑터(Main Adapter), API를 처리하기 위한 네이티브 인터페이스 부분으로 표 1과 같이 구성되어 있다.

Adapter	Define	Main Adapter	Native Interface
구성요소	header file, Data type, Endian, System Resource, Flag	Main(), Loader, UI, VM Setup (Boot, Initialize)	Native Code Translater

[표 1] 탑재를 위한 어댑터의 구성요소

정의 부분은 자료형 등의 기초적인 설정 사항을 정의하는 부분으로 모바일 장치 등의 특성을 정의하게 된다. 이 부분은 가상 기계 제작과 탑재를 위한 기초가 되는 부분이다.

(1) 가상 기계를 탑재 할 특정 기계의 SDK에 정의되어 있는 헤더 파일들을 지정한다.
(2) 기본 자료 형을 지정한다. - 부동 소수점을 사용할 수 있는지 등등의 여부와 자료형의 크기 제한 등을 정의한다.
(3) BIG_ENDIAN과 LITTLE_ENDIAN과 같은 데이터의 저장 형태를 지정한다.
(4) 메모리 크기 등의 시스템 자원들의 상황을 얻어내거나 지정한다.
(5) 장치에서 사용되어질 여러 종류의 플래그들을 지정하고 값을 세팅한다.

주 어댑터는 정의 부분에서 지정된 값들을 받아서 코어를 실행하는 가상 기계와 어댑터의 중심부를 역할을 수행한다. 주 어댑터는 메인 UI(User Interface), Main() 함수, 로더, 초기화의 구성 요소로 다시 세분화 된다.

(1) 메인 UI는 모바일 장치에서 가상 기계 제어 윈도우등의 메인 인터페이스를 만들어서 사용자가 가상 기계를 제어할 수 있도록 해준다.
(2) Main() 함수는 장치마다 그 명칭이 다르거나 가상 기계 시작 등의 내용이 조금씩 다르기 때문에 장치마다 따로 작성되어지게 된다.
(3) 로더는 장치마다 다른 파일 관리 기법이 존재하기 때문에 (Palm의 경우 파일을 DB로 관리) 특정 기계의 파일 시스템에 맞춰서 제작해야 한다.
(4) 초기화 또한 기종별로 다르기 때문에 어댑터에 속하게 된다.

네이티브 인터페이스는 가상 기계에서 넘겨받은 함수들과 API에서 사용 되어지는 함수들을 네이티브 코드로 번역해서 운영체제로 넘겨주게 된다. 가상 기계의 핵심 부분에서 사용되어지는 함수들은 동일하게 작성되어야 이식성이 높아지므로 각각의 가상 기계 함수들을 특정 기계에 적합하게 처리해주는 기능을 가지고 있다.

2.3 탑재 시 고려사항

원하는 장치에 가상기계를 탑재하기 위해서는 그 장치에 대해서 상당한 기본 지식과 자료가 있어야 하고 그 장치에 대한 SDK로 프로그램을 작성할 수 있어야 한다. 가상기계도 일종의 응용프로그램이기 때문이다. 로더를 배제 했을 때 간단한 연산기능을 수행하는 가상 기계는 구현이 어느 정도 쉽다. 하지만 입출력과 GUI, 하드웨어 제어 등을 할 수 있는 가상 기계를 제작하기 위해서는 그 장치의 API를 이용해야 한다.

첫 번째 고려사항은 대상 장치에 대한 지식이다. 기계적으로는 장치의 자료 형과 사용가능한 자원 즉, 기억장치나 입출력 장치 등을 고려해야 한다. 소프트웨어 적으로는 사용되어지는 함수들과 프로그래밍 스타일, 파일 관리기법, 데이터의 전송방식 등을 알고 있어야 한다.

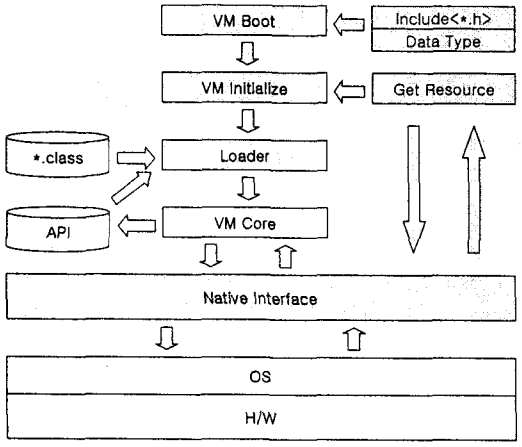
두 번째 고려사항은 제작한 가상 기계가 어느 정도까지 기능을 수행해야 하는가이다.

세 번째 고려사항은 어댑터의 구조이다. 네이티브 코드를 많이 써서 속도를 우선으로 해서 할 것인가 아니면 공통 함수를 만들고 네이티브 코드를 줄여서 크기가 늘고 속도가 약간 떨어지더라도 이식성을 높일 것 인가를 결정해야한다.

3. 어댑터 설계

3.1 어댑터 구조

어댑터 구조는 가상 기계의 크기, 이식성, 속도 차이를 결정할 수 있는 핵심 요소로서 네이티브 인터페이스를 위치를 고려해야 한다. 주 어댑터와 코어를 직접 그 장치의 네이티브 코드로 작성하게 되면 가상기계는 그 크기가 줄고 속도가 향상된다. 또 다른 방법은 주 어댑터와 코어에서 사용되어지는 시스템 함수들을 나중에도 쓸 수 있도록 가상으로 만드는 것이고 네이티브 코드로 작성되어진 함수들을 이용하되 최대한 적게 만들어서 가상함수들을 최대한 이용해서 작성하는 것이다. 그렇게 되면 다른 장치로 이식할 때도 몇 개의 네이티브 함수 부분만 작성한다. 이와 같은 방법으로 가상 기계를 개발하면 크기가 늘고 함수의 호출이 늘어나지만 이식성이 증가되어 다양한 장치로의 탑재가 훨씬 용이하게 된다. 따라서 그림 2와 같이 어두운 부분으로 표시된 부분처럼 어댑터는 코어를 감싸는 형태가 된다.



[그림 2] 어댑터의 개략 설계

3.2 정의 부분

정의 부분에서는 가상 기계에서 사용해야하는 필요한 모든 정보들을 가지고 있어야 한다. 컴파일 시 참조 할 파일들을 지정해야 하며 장치의 자원 정보를 알아내야 하며 장치의 ID, 기타 사용할 플래그 등을 정해야 한다. 지원할 자료 형, 자료형의 길이 제한, 데이터 저장 방식인 Endian, 저장 공간 용량, 기억장치 용량의 인식, 시스템 시간, 기본 입력 장치에 대한 정보를 알아내서 주 어댑터로 넘겨주어야 한다. 일종의 가상 BIOS이다.

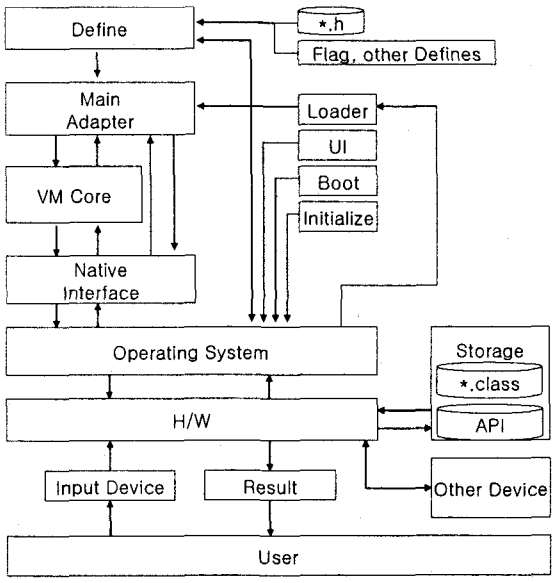
3.3 주 어댑터

주 어댑터를 설계하는 것이 어댑터 설계의 핵심이 된다. 대부분의 모바일 SDK는 UI를 위한 라이브러리들을 제공한다. 주 어댑터에서는 이러한 라이브러리를 이용해서 UI를 작성해주고 모바일 장치의 이벤트 키 값을 지정해준다. Main() 함수 부분은 가상 기계의 시작 과정과 초기화, 로더의 실행과 코어와의 연동을 통한 인터프리터의 실행을 위한 코드를 가지고 있어야 한다. 로더는 장치의 파일 관리기법에 따라서 클래스 파일을 받고 인터프리터의 요청에 의해서 API들을 읽어 들일 수 있어야 한다. 초기화는 장치의 자원 상황과 플래그의 기본 값들을 정의 부분으로부터 받아서 가상 기계가 실행 될 수 있는 상태를 만들어 준다. 또한 네이티브 인터페이스를 호출하기 위해서 네이티브 함수들의 인덱스 값을 지정해야 한다.

3.4 네이티브 인터페이스

네이티브 인터페이스는 가상 기계의 코어와 주 어댑터에서 사용되어지는 가상 함수들을 실제로 구현하는 부분으로서 단순한 변환기의 구조를 가지고 있다. 가상함수 -> 네이티브 코드 -> 운영체제의 구조로 되어있다. 가상 함수는 공통적인 것이라서 특정 시스템에서는 추가적인 정보를 요청할 수도 있다. 이 경우에는 네이티브 함수를 구현할 때 추가적인 구현을 해주어야 한다. 주 어댑터가 잘 설계되어 있다면 인덱스 값과 가상함수의 추가와 네이티브 함수의 추가로 가상 기계의 기능을 계속 확장 시킬 수 있다.

위와 같은 고려 사항 등을 기반으로 하여 본 논문에서는 그림 3과 같은 어댑터를 설계하였다.



[그림 3] 어댑터의 세부 설계

정의 부분에서는 운영체제로부터 입출력 장치와 저장 장치에 대한 정보와 사용 가능한 시스템 자원을 얻는다. 그리고 주 어댑터는 그 정보를 이용해서 가상 기계의 실행 준비와 UI를 만들고 클래스 파일

의 위치를 알아내고 로더를 불러서 클래스 파일을 얽어온다. 그러고 인터프리터를 실행 시킨다. 인터프리터가 바이트 코드를 실행 하다가 출력을 위한 API를 호출하게 되면 로더는 API를 가져오고 네이티브 인터페이스는 해당되는 네이티브 함수를 사용하여 LCD나 CRT 등의 출력 장치를 통해서 문자를 출력한다.

4. API

4.1 가상 기계와 API

가상 기계는 같은 언어를 지향한다 하더라도 그 언어의 구조가 조금씩 다르게 된다. 그래서 API들을 다시 작성해 주어야 하는 경우가 있다. 그것의 요인은 장치의 특성과 밀접한 연관을 가지고 있고 가상 기계가 얼마만큼의 기능을 가지도록 작성할 것인지에 따라서 결정 된다. 그 외에도 작성자의 구현의도에 따라 다르게 개발될 수 있다. JVM과 KVM, WABA 가상 기계와 같이 자바를 지향하는 가상 기계이지만 가상 기계들의 API는 다르다. JVM은 PC급 이상의 컴퓨터를 위해서 만들어져 있기 때문에 다른 두 가상 기계와 비교했을 때 크기가 크고 기능이 많다. KVM과 WABA는 저장 장치가 작고 저전력, 적은 기능을 위해 개발되었기 때문에 JVM의 API를 그대로 사용 할 수없다. KVM의 경우 JVM의 API를 약간 수정 하거나 그대로 가져오긴 했지만 모바일의 특성 때문에 몇몇 API들을 새로 만들어 줘야 했다. WABA의 경우 KVM과 같은 모바일 장치를 지향하며 제작 했지만 설계 방법이 다르고 기능 수행의 정도가 다르기 때문에 KVM의 API를 사용 할 수 없게 되었다.

4.2 API의 제작

API들은 탑재 영역에서 핵심 요소는 아니다. 하지만 가상 기계 환경에서 특정 언어에 대한 프로그램을 실행하기 위해서는 반드시 필요한 부분이다. 특히 시스템 API들은 네이티브 인터페이스를 사용하는 주체가 된다. API들은 자바로 작성되어있고 시스템 호출을 사용하는 부분은 가상 함수 또는 직접적인 네이티브 함수를 호출하게 된다. 물론 이 부분은 가상 기계가 API를 실행 할 때 작동되는 부분이므로 API 자체가 호출하는 것은 아니고 인터프리터에서 호출되는 것이다.

또한 API들은 장치의 특성에 맞게 작성되어야 한다. API들 중에서 시스템 API들은 네이티브 인터페이스를 사용할 수 있도록 만들어야 한다. 시스템 API들을 세부적으로 만들어 놓으면 추가적인 API들은 네이티브 인터페이스들을 몰라도 시스템 API들의 사용법만 안다면 쉽게 작성 될 수 있다. 그리고 언어의 특성을 살리고 다양한 기능의 API들을 구현할 수 있게 된다.

5. 결론

가상 기계를 멀티플랫폼 환경에 원활한 탑재를 위해 어댑터와 코어의 분리된 설계는 개발자에게는 어댑터 수정의 간소화와 구매자에게는 시간 절약의 이익을 주게 된다. 하지만 어댑터의 설계는 대상 장치에 대한 지식이 충분한 상태에서만 가능하다. 대부분의 경우 기종별로 사용되는 함수는 같은 의미일지라도 사용법과 명칭이 다르다. 기타의 환경도 다르고 또한 그것에 대한 정보를 구한다는 것이 쉬운 일도 아니다. 이것이 어댑터 제작의 난제이다. 이것은 모바일 장치 개발 회사들 간에 어느 정도의 표준화가 이루어지기 전까지는 해결하기 힘들 것으로 생각된다.

가상 기계를 다양한 장치에 탑재 할 수 있게 된다면 인터프리터 언어의 가치가 높아지게 된다. 특히 모바일 분야에서 하나의 프로그램이 여러 가지의 기종과 거기에 따른 다양한 운영 체제 속에서 수정을 하지 않고도 실행 될 수 있다면 더욱 그렇다. 이미 PC급 이상의 컴퓨터들은 어느 정도의 표준화가 되어 있기 때문에 가상 기계를 몇 가지의 플랫폼에 대해서만 탑재하면 되지만 차세대 정보 기기인 모바일 분야에서 가상 기계의 탑재 분야는 모바일 장치들의 표준화가 이루어지기 전까지는 큰 비중을 차지하게 될 것이다.

6. 참고문헌

[1] Joshua Engel, Programing for the Java Virtual Machine, Info-Book, 2000.

[2] James Gosling, Bill Joy, Guy Steele, The Java Language Specification Edition 1.0 & 2.0, Sun Microsystems, 1996

[3] Bill Blunden, Virtual Machine Design and Implementation in C/C++, Wordware publishing, Inc, 2002

[4] Jon Meyer & Troy Downing, JAVA Virtual Machine, O'REILLY, 1997

[5] Tin Lindholm & Frank Yellin, The Java Virtual Machine Specification, O'REILLY, 1999

[6] http://www.mobilejava.co.kr

[7] http://www.PalmOs.com

[8] http://www.waba.com

[9] http://www.kpug.net

[10] http://www.jmjeong.com