모바일 기기를 위한 자바가상머신 설계

유용선[0] , 성영락*, 이철훈
충남대학교 컴퓨터공학과
(ysryu[0], chlee)@ce.cnu.ac.kr, *yeong@mail.kookmin.ac.kr

Design of the Java Virtual Machine for Mobile devices

Yong-Sun Ryu[0], Yeong Rak Seong*, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
*School of Electrical Engineering, Kookmin Univ.

최근들어 인터넷의 발달로 인해 모바일 기기들은 기존의 음성 서비스 외에 네트워크를 통한 온라인 컨텐츠 서비스를 제공하고 있다. 그러나, 모바일 기기에 제공되는 동일한 컨텐츠들이 모바일 기기의 플랫폼에 맞게 따로 작성되어야 하는 단점이 있다. 그래서, 네트워크 기반의 플랫폼 독립성, 보안성, 이동성의 장점을 가진 자바기술을 모바일 기기에 적용하려는 연구가 계속되고 있다. 그러나, 기존의 자바가상머신을 임베디드 시스템이나 모바일 시스템과 같이 작고, 자원이 제한적인 장치에 탑재하게 되면 메모리 부족 및 성능저하 등의 여러가지 문제가 발생할 수 있다. 이에 본 논문에서는 CLDC에서 채택한 KVM(Kilo Virtual Machine)을 분석하여 작은 footprint Java™ platform에 속하는 자바가상머신을 설계한다.

1. 서 론

최근들어 인터넷의 발달로 인해 모바일 시장은 기존의 음성 데이터 서비스 기능외에 정보 검색이나 컨텐츠 쇼핑과 같은 무선 데이터 서비스에 많은 비중을 두고 있다. 그러나, 모바일 기기에 제공되는 동일한 컨텐츠들이 모바일 기기의 플랫폼에 맞게 따로 작성되어야 하는 단점이 있다. 그래서, 모바일 기기들에 플랫폼 독립성, 보안성, 이동성 등의 장점 을 포함하는 자바 환경을 모바일 기기에 적용하려는 연구가 계속되고 있다.

이러한 자바 기술을 적용한 모바일 기기들은 네트워크 인터페이스를 통해 사용자가 요청할 때마다 다운로드된 3rd party application software를 이용 할 수 있기 때문에, 다양한 애플리케이션 프로그램을 수행할 수 있는 장점을 가진다.

한편, 모바일 기기들에 기존의 자바가상머신을 탑재하게 되면 많은 메모리가 필요할 뿐만 아니라, 성능저하의 문제도 발생할 수 있다. 따라서, 본 논문에서는 JSR-30 문서에서 기술된 CLDC에 적합한 KVM 명세서를 토대로 보다 효율적인 자바가상머신을 분석 및 설계에 대해 기술한다.

본 논문의 2장에서는 관련연구로 CLDC에서 제시된 KVM을 설명하고, 3장에서는 자바가상머신을 분석 및 설계하며, 마지막으로 4장에서 향후 과제에 대하여 기술한다.

2. 관련 연구

2.1. CLDC에서 명시된 KVM

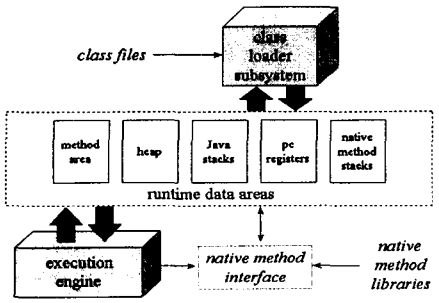
KVM은 작고 자원이 한정된 기계장치를 위해 설계된 소형이며, 휴대용 기기에 적합한 자바가상머신이다. KVM은 자바언어의 기능을 유지하며 몇백킬로바이트로 제한된 기계장치에서 실행되어야 하는것을 전제로 하는 완전한 자바가상머신을 목표로 최대한 작게 설계되었다. CLDC의 스펙에는 KVM의 설계에 관해 " ①고정된 메모리 용량을 40-80KB정도로 작아야 하다. ② 이식성이 좋아야 한다 ③ 모듈화와 최적화가 되어 있어야 한다 ④ 가능한 완전하고 빨라야 한다" 고 명시되어 있다

한편 KVM을 설계시 요구되는 CLDC에서 특징으로는 몇 가지가 있다.

- 부동 소수점을 지원하지 않는다.
- finalization을 지원하지 않는다.
- JNI를 지원하지 않는다.
- 리플렉션을 지원하지 않는다.
- 쓰레드 그룹과 데몬 쓰레드를 지원하지 않는다.
- 클래스 검증과정이 on-device/off-device로 2단계로 나뉘어 졌다.
- Sand-Box 보안 모델을 사용한다.
- 클래스 Lookup, 클래스 로딩과 링킹 방법이 다르다.

3. 모바일 기기를 위한 자바가상머신 분석

네트워크를 기반한 구조로 플랫폼 독립성, 보안, 네트워크 이동을 지원하는 자바가상머신은 자바 제반 기술의 핵심으로 주요 작업은 클래스 파일을 로딩해서 바이트 코드로 실행하는 것이며, 전체적인 구조를 살펴보면 [그림1]과 같다

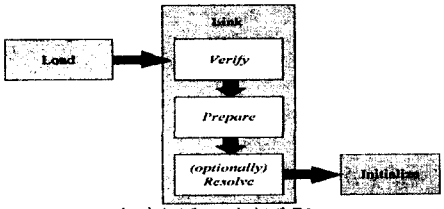


[그림 1] 자바가상머신의 구조

본 논문에서는 자바가상머신을 클래스 로더부분과 클래스 화일을 얻어와서 가상머신의 실행상태로 만들어주는 클래스 런타임 데이터 영역, 그리고, 바이트 코드의 인스터럭션을 수행하는 실행엔진으로 세분화하여 기술한다.

3.1. 클래스 로더 시스템에 관한 분석 / 설계

클래스 로더 시스템은 클래스 화일을 얻어와서 가상머신의 실행상태로 만들어주는 부분으로, 클래스 변수를 저장하기 위해 메모리를 할당하고 초기화하며, 심볼릭 레퍼런스의 레솔류션(resolution)을 돕는다. 이러한 과정은 로딩, 링킹, 초기화 과정을 통해 수행된다.



[그림2] 클래스 로더 시스템 구조

로딩은 특정한 이름을 사용해 클래스나 인터페이스의 바이너리 형태를 찾는 과정으로 세개의 기본적인 activity로 구성된다.
- 그 타입을 표현하는 이진데이타의 스트림을 생성
- 이진데이타의 스트림을 메소드 영역안에서 내부 데이터구조로 파싱
- 그 타입을 표현하는 java.lang.Class 클래스의 인스턴스 생성 및 내부데이타 구조의 인터페이스의 역할을 담당.

링킹은 verification, preparation, resolution의 3단계로 수행된다. KVM에서의 verification은 단순히 바이트코드를 순차적으로 검증하며 Off-device pre-verification과 스택 맵을 이용한 런타임 검증이 있다. Preparation은 클래스 변수를 위한 메모리, 실행중인 프로그램의 수행을 향상시키기 위해 데이터 구조를 위한 메모리(클래스 메소드를 위한 데이터를 포인트하는 메소드 테이블)를 할당한다. resolution과정은 런타임시 constant pool에 있는 심볼릭 레퍼런스로부터 동적으로 구체적인 값을 결정하는 과정이다

초기화 과정은 클래스 변수에 적절한(proper) 초기치를 세팅시키는 작업으로 자바 코드에서 적당한 초기값은 클래스 variable initializer나 static initializer에 의해 기술되고, 타입의 클래스 variable initializers와 static initializers은 자바 컴파일러에 의해 하나의 특별한 메소드(<clinit>())에 위치한다

3.2. 런타임 데이터 영역에 관한 분석 / 설계

런타임 영역은 프로그램이 실행되는 동안 메모리를 구성하는 부분으로 메소드, 힙, 자바 스택, PC 레지스터, 네이티브 메소드 스택으로 구성되며, 자바가상머신의 인스턴스들은 하나의 메소드영역과 힙영역을 가지며, 이러한 런타임 데이터 영역은 쓰레드들간 서로 공유한다.

3.2.1 메소드 영역

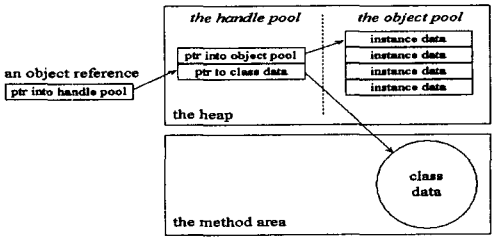
로드된 타입들에 대한 정보는 메소드 영역이라 불리는 논리적 메모리 영역에 저장된다. 클래스 로더는 클래스 파일에 포함된 바이너리 데이터로부터의 타입 정보을 읽어 자바가상머신에 전달하면, 자바가상머신은 바이너리 데이터로부터 타입에 대한 정보를 추출하여 메소드 영역에 정보를 저장하며, 타입정보는 다음과 같다
- fully qualified name에 관한 정보
- fully qualified name 의 직계 슈퍼클래스에 관한 정보
- modifier에 관한 정보
- constant pool의 타입 정보
- 필드 정보, 메소드 정보
- 클래스로더의 레퍼런스에 관한 정보
- 상수를 제외한 선언된 모든 클래스 변수의 관한 정보

한편, 모든 쓰레드들이 같은 메소드 영역을 공유하게 되기 때문에 메소드 영역의 데이터 구조는 쓰레드에 안전하게 디자인되어야 하며, 이 영역은 가비지 콜렉터에 의해 관리된다.

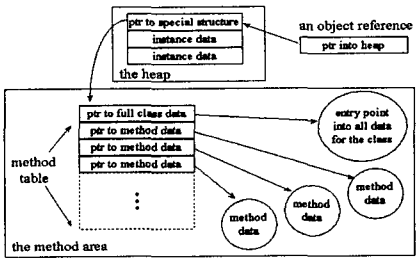
3.2.2. 힙(Heap) 영역

힙영역은 초기화된 자바 애플리케이션의 모든 객체들을 적재하는 공간으로, 모든 쓰레드들이 공유하기 때문에 객체(힙 데이터)에 접근하기 위한 적당한 동기화가 필요하며, 자바가상머신 구현은 힙을 다루기 위하여 가비지 컬렉터를 사용한다.

힙에서의 객체 표현(object representation)은 핸들풀과 객체풀을 분리하여 힙의 단편화를 최소화 하는 방법[그림3]과, 메소드 테이블을 사용하여 인스턴스 메소드 호출을 향상시키는 방법[그림4] 등이 있다.



[그림3] 핸들 풀과 객체 풀을 분리한 객체 표현



[그림4] 메소드 테이블을 사용한 객체 표현

3.2.3. PC registers

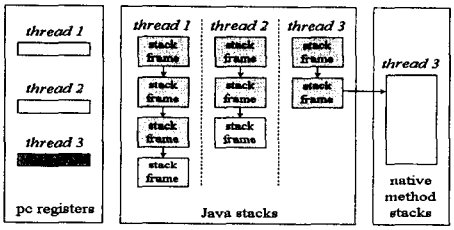
새로운 쓰레드가 생성되면, 그 쓰레드는 자신만의 PC레지스터와 자바스택을 할당받는다. PC레지스터는 1워드 크기로 네이티브 포인터와 리턴값을 갖는다.

그리고 이 쓰레드가 자바 메소드를 실행하면, 쓰레드에 의해 실행되는 현재 명령에 대한 주소(메쏘드 바이트 코드 시작위치로 부터의 옵셋)를 포함한다..

3.2.4 자바 스택

새로운 쓰레드가 실행될 때 자바가상머신은 쓰레드를 위한 자바 스택을 생성하며, 자바 스택에서는 스택 프레임에 자바 메소드의 호출 상태(지역변수, 파라미터, 리턴값, 중간연산값)를 저장한다. 자바가상머신은 자바스택을 push와 pop 연산을 사용하여 스택프레임을 pushing하거나 poping하며, 스택 프레임은 local variable, operand stack, frame data으로 구성되어 있다.

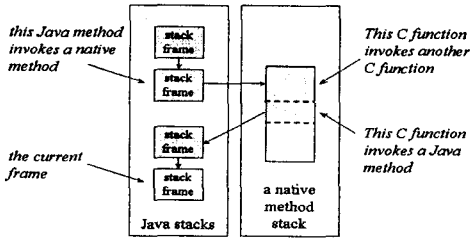
한편, 자바스택 영역에서는 [그림5]처럼 각각의 쓰레들간의 영역이 자바스택에 있는 모든 데이터는 개별적이다.



[그림5] 쓰레드에 할당되는 자바스택 영역

3.2.5 Native 메소드 스택

쓰레드가 native 메소드를 호출할 때 생성되는 스택으로 native 메소드의 호출 상태를 저장한다 . 네이리브 프로세서 내부의 레지스터를 사용하며, 네이트브 힙의 메모리 할당한다. 한편, 자바가상머신은 네이트브 코드로 어떤 제약없이 동적으로 확장 할 수 있다.



[그림6] java와 native 메소드를 호출하는 쓰레드에서의 스택

3.3 Execution engine

자바가상머신 구현의 핵심부분으로 execution engine의 수행은 명령어 집합(Instruction Set)으로 구성된다. 각각의 인스트럭션 명세서에는 해당하는 명령어에 따른 동작을 자세하게 정의한다.

그리고 메소드의 바이트코드 스트림은 자바가상머신을 수행하기 위한 명령 순서로 Execution engine에서 바이트코드를 한번에 하나씩 수행한다.

3.4. 자바가상머신 설계시 기타 고려사항

- CLDC에서는 JNI를 사용하게 되면 제한적인 보안 모델 정책으로 네이리브 함수의 사용이 위험해지기 때문에 네이리브 함수와 자바 메소드간의 랩퍼를 위한 설계가 필요하다
- 가비지 컬렉션을 수행시 finalization을 지원하지 않도록 설계해야 한다.
- CLDC에서는 jar 파일 포맷을 네트워크를 통해 로더 시스템에 로드되므로, 사용자 정의 클래스 로더를 만들수 없도록 고려해야 한다.
- 대부분의 CLDC 디바이스들은 파일 시스템이 없기 때문에 자바가상머신 설계시 ROM이나 RAM, 또는 플래시 메모리에 관한 사항도 고려해야 한다

4. 향후 과제

앞으로 리소스가 적은 소형 통신디바이스나, 임베디드 시스템에 네트워크 장점을 갖는 자바기술의 적용이 시도 될 것이다. 그러나, 이러한 디바이스에 기존의 자바가상머신을 탑재할 수 없기 때문에, foot print가 작은 자바가상머신의 개발이 필요하다. 따라서, 향후과제로써 KVM의 명세에 맞는 적은 메모리의 자바가상머신을 구현한다.

한편, 최근 추세가 모바일 디바이스에 Jini기술을 적용하기 위해 노력중인데, 이를 위해서 앞으로 RMI기술을 이용할 수 있도록 고려해야 한다.

참고문헌

[1] Sun Microsystems, " The Java™ Virtual Machine Specification second edition
[2] Bill Veners, " Inside the JAVA2 Virtual Machine second edition"
[3] Sun Microsystems, " JSR-30 J2ME Connected, Limited Device Configuration(final Release)"
[4] Sun Microsystems, " Java™ 2 Platform Micro Edition (J2ME™) Technology for Creating Mobile Devices White Paper"