동적 컴파일링 기법을 이용한 소형 자바 가상 기계의 설계†

○
김 도 우*, 정 민 수*, 진 민*, 이 은 정**
*경남대학교 컴퓨터공학과
**한국전자통신연구원 실시간시스템연구팀

Design of A Miniature Java Virtual Machine Using Dynamic Compiling

Do-Woo Kim*, Min-Soo Jung*, Jin Min*, Eunjung Lee**
*Dept. of Computer Engineering, Kyungnam University
**Real-Time System Group, ETRI

요 약

썬에 의해 자바 플랫폼이 소개된 후, 자바 기술은 네트워크 상에 연결된 장치들이 선택할 수 있는 하나의 플랫폼으로 자리를 잡았다. 그래서 자바 기술을 자원 제약적인 네트워크 디바이스 및 디지털 전자기기에 적용하고자 하는 연구가 계속해서 진행 중이다. 본 논문에서는 자바 가상 기계를 소형화할 수 있는 기법들을 연구 분석하고, 처리 능력이나 메모리의 크기에 제약이 있는 디바이스들에 사용 가능하고, 동적 컴파일링 기법을 적용하여 실행 효율을 높일 수 있는 소형 자바 가상 기계를 설계하고자 한다.

1. 서 론

자바(JAVA)는 플랫폼 독립적인 특성을 지니고 있다. 자바는 네트워크를 통하여 자바 바이트코드라 불리는 중간 코드가 로컬로 다운로드 되어 실행되거나, 로컬 환경에서 보통의 응용 프로그램들처럼 동작하기도 한다. 자바 바이트코드는 자바 가상 기계가 구현된 어떠한 플랫폼에서도 소스 코드의 수정 없이 잘 동작되도록 설계되었다. 그래서 네트워크 상에 연결된 많은 장치들에 의해 활용되고 있다. 또한 네트워크에 접속 가능한 디바이스(device) 및 디지털 전자기기의 개발도 아주 빠른 속도로 진행되고 있다.

썬은 처리 능력이나 메모리와 같은 부분에서 상당히 많은 제약 사항을 가지고 있는 화상 전화, 셋톱박스, 내장 장치에 자바 기술을 적용하기 위해 PersonalJava™, EmbededJava™, JavaCard 와 같은 플랫폼을 제시했고, 셀룰러 폰, PDA, TV, VCR, CD 플레이어, 게임기에 적용 가능한 플랫폼도 제시했다. 실제, 자원 제약적인 네트워크 디바이스나 디지털 전자기기에 자바 기술을 적용하기 위해서는 소형의 자바 가상 기계와 자바 API 런타임 라이브러리의 축소가 필수적인 요소이다. 자바 가상 기계를 소형화할 수 있는 기법들이 계속해서 제시되고 있고 이를 적용한 가상 기계가 연구되고 있다[12].

지금까지 제시된 소형 자바 가상 기계는 해석기 방식을 채택하고 있어 실행 시간면에서 비효율성을 내포하고 있다[2,3].

본 논문에서는 자바 가상 기계를 소형화할 수 있는 기법들을 적용하고 실행 시간의 효율성을 높이기 위해 동적 컴파일링 기법을 적용하여 자원 제약적인 디바이스나 디지털 전자기기에 적합한 소형 자바 가상 기계를 설계하고자 한다.

본 논문의 구성은 2 장에서는 소형화된 자바 플랫폼 및 가상 기계 소형화 기법을 연구 분석하고, 3 장에서는 자바 가상 기계의 소형화 기법과 동적 컴파일링 기법을 이용하여 설계한 자바 가상 기계를 설명한다. 그리고, 4 장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 JavaCard 플랫폼

JavaCard 기술은 자바 프로그래밍 언어로 작성된 프로그램을 스마트 카드나 자원 제약적인 디바이스에서 운용할 수 있도록 한다. 표준 자바 소프트웨어 개발 도구와 개발 환경을 이용하여 프로그램을 개발하고 테스트한 후, JavaCard 기술이 사용 가능한 디바이스에 설치 가능한 형태로 그 프로그램을 변환한다. JavaCard 플랫폼에 대한 응용 소프트웨어를 JavaCard 애플릿(applet)이라 한다.

애플릿을 구성하는 클래스 파일을 JavaCard 변환기를 사용하여 CAP(converted applet)파일로 변환한다. JavaCard 변환기는 클래스 파일뿐만 아니라, 변환될 클래스에 의해 참조되는 패키지의 내용에 대한 링크 정보를 포함한

† 본 연구는 한국학술진흥재단 우수 연구소 과제연구비 지원으로 수행되었습니다

하나 이상의 export 파일을 입력으로 받는다. 변환된 CAP 파일이 카드 판독 장치를 내장한 컴퓨터인 카드 터미널로 복사되고, 터미널의 설치 도구가 CAP 파일을 적재하고 JavaCard 기술이 사용 가능한 장치로 전송한다. 디바이스 내의 설치 프로그램은 CAP 파일의 내용을 받고 JavaCard 가상 기계에 의해 이 CAP 파일이 실행될 수 있도록 준비를 한다. 가상 기계는 CAP 파일을 적재하거나 조작할 필요 없이 설치 프로그램에 의해 디바이스에 적재된 CAP 파일 내의 코드를 실행하는 역할만 담당한다.

JavaCard 기술은 가상 기계의 크기 및 기능을 전체적으로 축소한 것뿐만 아니라 설치 프로그램에서 적재 기능을 수행할 수 있도록 기능 분리를 이루었다[2,12].

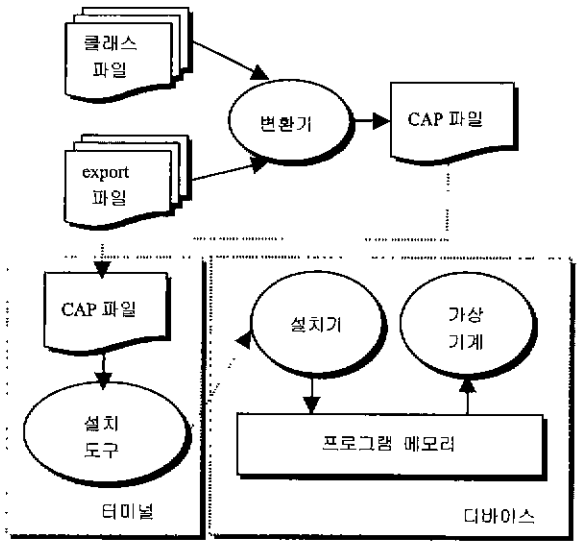

그림 1 JavaCard 플랫폼

2.2 K 가상 기계

자바 가상 기계는 자바 프로그래밍 언어로 작성된 응용 프로그램을 다른 하드웨어 환경이나 운영체제로 이식할 수 있도록 하는 자바 기술의 핵심 요소이다. 자바 가상 기계는 단순히 응용 프로그램의 바이트코드만을 실행하는 것은 아니고, 시스템의 메모리를 관리하고, 유해한 코드에 대한 보안을 제공하고, 다중 스레드를 관리하는 등의 바이트 코드 실행과 관련된 테스크(task)들을 수행한다[5,6,7,8].

K 가상 기계는 셀룰러 폰, PDA 와 같은 메모리 절약형 디바이스에 자바 기술을 적용하기 위해 설계되어졌다. K 가상 기계의 설계 목표는 성능 저하 없이 가상 기계의 크기를 줄이고, 실행동안 가상 기계에 의해서 사용되는 메모리의 크기를 줄이고, 가상 기계의 구성 요소들이 특정 디바이스에 적합하게 구성될 수 있도록 하는 것이다.

K 가상 기계 설계 시 중요한 또 하나의 고려사항은 이식성(portability)이다. K 가상 기계는 시스템 의존성을 최대한 줄이는 높은 이식 가능한 구조를 가진다. 바이트코드 인터프리터(bytecode interpreter), 멀티 스레딩과 쓰레기 수집기(garbage collector)도 임의의 플랫폼에 빠르게 이식할 수 있도록 완전히 시스템 독립적으로 구현되었다. 또한 가상 기계의 기능 분리가 가능하도록 설계되어졌다.

K 가상 기계의 크기는 40K 정도이며 효율적인 실행을 위해 수십 K 의 동적 메모리가 필요하다. 가상 기계 크기와 실행 시 필요한 동적 메모리 크기의 합이 128K 정도로 축소됨으로써 자원 제약적인 디바이스에 자바 기술을 적용할 수 있게 된 것이다. 그러나, K 가상 기계는 완전한 소프트웨어 스택이 존재하지 않아 운영 체제에 종속적이다[3,12].

2.3 자바 가상 기계 소형화 기법

자바 가상 기계는 스택을 기반으로 한 기계이며, 가상의 명령어 집합과 실행 환경을 제공한다. 자바 가상 기계를 소형화하는 기법은 크게 두 가지로 나눌 수 있다. 첫째는 자바 API 런타임 라이브러리를 축소하는 방법이고, 둘째는 자바 가상 기계의 구성요소의 크기를 줄이거나 없애는 방법이다.

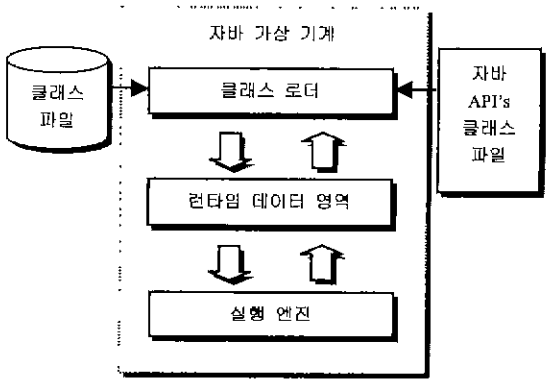

그림 2 가상 기계 내부 구성

후자의 경우, JavaCard 와 같이 설치 프로그램을 통해 변형된 클래스 파일을 적재할 수 있기 때문에 클래스 로더를 생략할 수 있고, 클래스 파일의 축소/압축이나 동적 메모리 사용의 제한을 통해 런타임 데이터 영역의 크기를 줄일 수 있고, 실행 엔진을 구성하는 기능 중에서 불필요한 기능을 삭제함으로써 크기를 줄일 수 있다. 예를 들면, 쓰레기 수집기도 필요성 유무에 따라 삭제 가능하고 클래스 검증(verification)도 생략 가능하다. 또한 자바 기술을 적용할 디바이스의 종류에 따라 불필요한 가상 기계 명령어 집합을 줄임으로써 가상 기계의 크기를 줄일 수 있다[2,3,5].

3. 동적 컴파일링 기법의 소형 자바 가상 기계

이전 장에서 설명된 JavaCard 가상 기계나 K 가상 기계는 해석기 방식의 자바 가상 기계이다. 해석기 방식은 바이트코드 하나 하나를 읽어 해석하여 직접 실행하는 방식으로써, 프로그램의 수행이 일률적으로 이루어지므로 프로그램 수행의 중단이 없고, 전체적인 응답 시간이 상대적으로 짧아진다. 하지만, 각각의 바이트코드를 읽고 해석하여 동작을 수행하는 과정에서 프로그램의 전체 수행 시간이 비교적 길어지게 되는 비효율성

을 내포하고 있다.

일반적으로 프로그램의 코드 중에서 10~20% 정도가 프로그램 수행 속도에 있어서 80~90% 정도를 소모하는 것으로 보고되고 있다. 자원 제약적인 디바이스나 디지털 전자기기의 경우는 특정 기능에 의존적일 수 있으므로 이 특정 기능 코드에 대한 최적화에 중점적인 노력을 기울인다면 해석기 방식보다 빠른 수행 속도를 기대할 수 있을 것이다.

본 논문에서 제안하는 '자바 가상 기계'는 자원 제약적인 디바이스에 적용 가능하도록 소형화하고 해석기 방식의 느린 수행 속도를 동적 컴파일링 기법을 적용하여 가상 기계의 효율성을 높이는데 중점을 두고 있다 [1].

3.1 시스템 구조

본 논문에서 제안하는 '소형 자바 가상 기계'는 해석기와 컴파일러 모두를 가짐으로써, 조건에 따라 해석과 컴파일을 자유롭게 할 수 있도록 설계되었다.

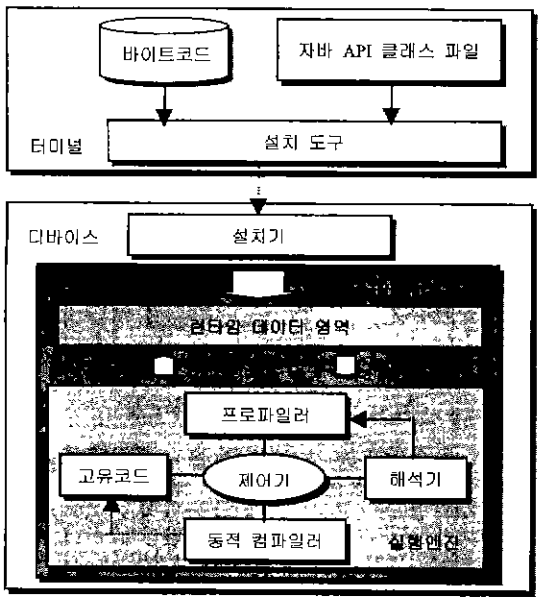

그림 3 동적 컴파일링 기법을 이용한 소형 자바 가상 기계의 구성

이 시스템은 자바 컴파일러에 의해 생성된 바이트코드와 실행에 필요한 자바 API 를 먼저 클래스 로더 기능을 가진 컴퓨터인 터미널의 설치 도구를 통해 적재하고 이 적재된 내용을 설치기로 전송한다. 디바이스 내의 설치기는 전송된 내용을 런타임 데이터 영역에 복사한다. 런타임 데이터 영역에 적재된 내용이 먼저 해석기를 통하여 번역되어 실행된다. 실행될 때 프로파일러는 해석되어 실행되는 코드에 대한 정보를 기억하고, 컴파일레이션 방법을 선택한다. 컴파일 방법은 백그라운드 컴파일레이션을 통해 고유코드의 캐시에 저장된다. 메소드가 호출될 때, 이 프로파일러에 의해 해석이 되기도 하고 컴파일된 고유코드가 직접 실행되기도 한다.

3.2 소형화를 고려한 설계

본 논문에서 자바 가상 기계를 설계하는데 있어 가장 중점적으로 고려한 사항은 소형화이다.

먼저 자바 API 런타임 라이브러리를 가상 기계의 동작에 기본적으로 필요한 자바 API 와 디바이스의 동작을 구현하는데 필요한 자바 API 를 디바이스에 따라 비례축소 가능하도록 구성하였다. 다음으로 가상 기계 내의 클래스 로더를 적재 기능을 갖춘 터미널 내의 설치 도구를 통해 수행되도록 대체하였고, 클래스 내의 필드 수와 메소드 수, 배열내의 원소 수, 메소드 내의 바이트코드 수, 메소드가 사용하는 스택의 최대크기, 메소드가 사용하는 로컬 변수의 최대 수 등의 제한을 통해 런타임 데이터 영역의 크기를 줄일 수 있도록 설계하였다.

차기 연구를 통해 클래스 내의 필드를 해석 과정을 거치지않고 직접 참조 가능하도록 하여 런타임 데이터 영역의 크기를 줄이려 한다 .

4. 결론 및 연구 결과의 활용

본 논문에서 제안한 '자바 가상 기계'는 소형화 기법들을 적용하여 소형화하고 해석기 방식의 느린 수행 속도를 동적 컴파일링 기법을 적용하여 실행 효율을 높일 수 있도록 설계하였다.

동적 컴파일레이션 기술은 실행 시 프로파일링과 컴파일레이션에 대한 오버헤드 등의 문제점을 안고 있지만 자원 제약적인 디바이스가 특정 기능들에 의존적일 수 있기 때문에 성능 향상을 가져올 수 있다.

앞으로는 좀 더 향상된 '소형 자바 가상 기계'를 위하여 본 논문에서 고려하지 않은 최적화 기법을 적용하여 제한적인 자원 특히, 메모리를 효율적으로 활용할 수 있는 방향으로 연구를 진행하고자 한다. 표준 자바 API 런타임 라이브러리와 호환되고 더 작은 메모리를 필요로 하는 라이브러리를 연구하는데 주력하고자 한다.

참 고 문 헌

[1] 이종동, 정민수, 이수진, 진민, "동적 컴파일링 기법을 이용한 자바 가상 기계 설계", 한국 정보과학회 제 25 회 추계발표대회, p425-427(1998)
[2] JavaCard™ 2.1 Virtual machine Specification, SUN(1999)
[3] The K virtual machine(KVM), SUN(1999)
[4] A. Krall and R. Grafl, CACAO-A 64 bit Java VM Just-in-Time Compiler, (1997)
[5] B. Venners, "Inside the Java Virtual Machine", McGraw-Hill, (1998)
[6] G. McGraw and Felten, "Java™ Security", Wiley, (1997)
[7] J. Gosling, "The Java Language Specification" , Addison-Wesley, (1996)
[8] J. Meyer and T. Downing, "Java™ Virtual Machine", O'Reilly, (1997)
[9] M. Campione and K. Walrath, "The Java™ Tutorial, Object-Oriented Programming for the Internet", Addison-Wesley, (1996)
[10] P. van der Lindon, "just Java, 2/E", SunSoft Press, (1997)
[11] T. Lindholm and F. Yellin, "The Java™ Virtual Machine Specification" , Addison-Wesley, (1996)
[12] http://java.sun.com/, Sun Microsystems, Java Home Page