자바가상기계 메모리 할당을 보여주는 비주얼 도구의 개발†

이 수 진*, 정 민 수*, 김 우 완*, 이 공 선**, 윤 기 송**

*경남대학교 컴퓨터공학과

**한국전자통신연구원 실시간시스템연구팀

Development Visual Tools of that shows memory allocation of JVM

Su-jin Lee*, Min-Soo Jung*, wu-woan Kim*, kong-seon Lee**, gi-song Yoon**

*Dept. of Computer Engineering, Kyungnam University

**Real-Time System Group, ETRI

요 약

본 논문에서 소개하고 있는 비주얼 자바가상기계 시뮬레이터는 자바 컴파일러에 의해 컴파일된 결과인 바이트코드를 분석하고, 그 분석된 결과가 자바가상기계 내부구조에 어떻게 할당되는지를 시각적으로 보여줌으로써 자바 소스 코드가 보여주지 못하는 가상기계 내부의 메소드 영역, 자바 스택 영역, 힙 영역에 할당되는 정보를 통해 자바 소스 프로그램의 보다 명확하고 쉬운 이해가 가능하도록 한다.

1. 서 론

지금 인터넷 브라우저 환경뿐만 아니라 데이터 베이스 연동, 내장 시스템, 전자 화폐에 이르기까지 다양한 분야에 적용되고 있는 자바는 분산 환경의 적합성, 코드 재사용 용이성, 프로그래밍의 단순성의 특징을 가진다. 특히 자바는 어떠한 프로세서 혹은 하드웨어에서도 동작할 수 있는 높은 이식성을 가진다. 자바로 작성된 프로그램의 경우 생성되는 목적코드가 특정한 프로세스 또는 하드웨어에 맞춰져 있지 않고 자바가상기계(java virtual machine)라는 가상의 명령어 집합과 실행환경에 맞춰져 있기 때문이다. 자바가상기계의 명령어 집합을 바이트코드(bytecode) 또는 클래스파일(.class)이라 한다.

객체지향언어(object-oriented language)중의 하나인 자바로 작성된 프로그램은 수행과정에 관련되어 있는 많은 정보가 숨겨져 있기 때문에 객체지향 프로그램을 분석하기가 쉽지 않다. 컴파일 후에 생성된 바이트코드에는 사용자가 정의한 객체뿐만 아니라 미리 정의되어 숨겨져 있는 객체에 대한 정보가 모두 저장된다. 자바 소스 프로그램(.java 파일)은 자바 컴파일러에 의해 바이트코드로 번역되고, 이 바이트코드는 자바가상기계에 의해서 실행된다.

본 논문에서는 현재 실행중인 자바 프로그램에 대한 정보가 자바가상기계상에서 각 구성요소(메소드 영역, 자바 스택 영역, 힙 영역)들 내에 할당되는 과정을 시각적으로 표현함으로써 자바 소스 프로그램에 대해 보다 명확하고 쉬운 이해가 가능하도록 한다.

† 본 연구는 한국학술진흥재단 우수연구소 과제 연구비 지원으로 수행되었습니다

본 논문의 구성은 다음과 같다. 2 장에서는 자바가상기계 내부적 구성요소에 대해서 설명하고, 3 장에서는 비주얼 자바가상기계 시뮬레이터의 구성 및 구현에 관해 기술하고 마지막 4 장에서는 결론 및 연구 결과의 활용에 관해서 기술한다.

2. 자바 가상 기계

2.1 자바 가상 기계의 구조

자바 가상 기계는 크게 클래스 로더 서브시스템(class loader subsystem), 실행 엔진(execution engine), 런타임 데이터 영역(runtime data areas)으로 구성되어 있다.

자바가상기계의 구성요소에 대한 설명은 다음과 같다.

클래스 로더 서브시스템

클래스(class)나 인터페이스(interface)의 타입을 위한 이진 데이터를 찾아 적재(loading)하고, 정확성을 검증한다. 또한 클래스 변수에 대한 메모리 할당과 디폴트 값으로 초기화를 수행하고, 그 타입에 대한 심볼릭 참조를 직접 참조로 변환한다. 그리고 적당한 시작값에 대한 클래스 변수를 초기화하는 자바 코드를 호출한다.

런타임 데이터 영역

자바가상기계가 프로그램을 실행시키는데 필요한 메모리를 구성하고, 적재된 클래스 파일로부터 추출된 바이트 코드를 포함한 많은 정보들이 저장되는 메모리 영역이다. 세부적으로 메소드 영역(method area), 자바 스택 영역(java stack area), 힙 영역(heap), PC 레지스터(PC register), 원시 메소드 스택(native method stack)으로 구성된다.

– 메소드 영역

적재된 이진 데이터의 타입에 관한 정보를 저장한다. 즉, 이진 데이터에서 타입에 관련된 정보를 추출해서 저장하는 영역이다. 가상 기계는 호스팅하는 응용 프로그램을 실행시 이 영역에 저장된 타입정보를 찾아 사용한다. 이 영역에서의 기본적인 타입정보는 표 1에 나타나 있다.

표 1. 메소드 영역에 저장되는 정보

타입 정보(type information)
타입의 완전한 이름 타입의 슈퍼클래스의 완전한 이름 타입 식별 정보 클래스 또는 인터페이스 타입의 식별자 슈퍼인터페이스의 완전한 이름에 대한 정렬 리스트
상수 풀(constant pool)
필드 정보(field information)
필드의 이름 필드의 타입 필드의 수식어
메소드 정보(method information)
메소드 이름 메소드 반환 타입(또는 void) 메소드 파라메터의 타입과 개수 메소드의 수식어 메소드의 바이트코드 피연산자 스택, 지역변수 섹션 크기 예외 테이블
클래스 변수(class variable)
클래스 로더 클래스 클래스 클래스

– 힙

프로그램이 실행될 때 자바 가상 기계는 프로그램이 인스턴스화한 모든 객체를 힙에 할당한다. 자바 가상기계 인스턴스 내부에는 단 하나의 힙이 존재하기 때문에 런타임시 모든 쓰레드는 힙을 공유하고, 이러한 객체(힙 데이터)에 접근하기 위해서 멀티쓰레드의 적당한 동기화와 관계한다.

– 프로그램 카운터

실행되는 프로그램의 각 쓰레드는 자신의 PC 레지스터와 자바 스택을 가지는데 이것은 쓰레드가 시작될 때 생성되고, PC 레지스터의 값은 실행할 다음 명령어를 가리킨다.

– 자바 스택

새로운 쓰레드가 생성되면 자바 가상 기계는 그 쓰레드를 위한 자바 메소드 호출의 상태(지역변수, 매개변수, 리턴 값, 중간계산)을 저장하고, 현재 클래스와 힙 재 상수 풀의 트랙을 유지한다. 또한 쓰레드의 스택에 새로운 프레임(frame)을 형성해 푸시(push)와 팝(pop) 두 가지 연산을 수행한다.

– 스택 프레임

스택 프레임은 지역 변수(local variable), 오퍼란드 스택(operand stack), 프레임 데이터(frame data)를 구성한다.

자바 가상 기계가 자바 메소드를 생성할 때 스택 프레임은 지역 변수와 오퍼란드 스택에서 메소드에 의해서 요구되는 워드 수를 결정하기 위해서 클래스 데이터를 검사한다 자바 스택 프레임의 지역 변수는 워드의 배열로 구성되고, 메소드의 매개변수와 지역 변수를 포함한다. 컴파일러는 선언되어진 순서에 따라 지역 변수 배열에 매개변수와 지역 변수를 배치한다.

오퍼란드 스택도 워드의 배열로 구성되어 있다. 하지만 지역 변수와는 달리 배열 인덱스를 통해서 접근된다. 오퍼란드 스택은 값을 푸시하고 팝하므로써 접근되어지고, 오퍼란드 스택에 값을 푸시했다면, 이후의 명령어는 팝해서 그 값을 사용한다.

– 원시 메소드 스택

쓰레드가 자바로 작성된 메소드가 아닌 다른 언어로 작성된 메소드를 호출하면 이 메소드에 대한 데이터 영역이 생성된다.

실행 엔진

적재된 클래스의 메소드들에 포함된 명령어들을 실행한다. 실행 엔진이 명령어들을 실행할 때 자바 가상기계는 현재의 클래스의 상수 풀, 현재 프레임의 지역 변수, 현재 프레임의 오퍼란드 스택의 탑(top)에 놓인 값들을 사용한다

3. 비주얼 자바 가상 기계 시뮬레이터

이 장에서는 자바 소스 프로그램이 컴파일과 자바 디버거를 통해서 출력되는 코드를 입력으로 받아 실행되는 비주얼 자바가상기계 시뮬레이터를 기술한다.

비주얼 자바가상기계 시뮬레이터는 그림 1에서 보는 바와 같이 크게 바이트코드를 입력으로 한 바이트 코드 분석부와 자바디버거 분석부, 또한 각각의 분석부를 통해 나온 결과를 시각화하는 메소드 영역 실행부와 힙과 자바 스택영역 실행부 네부분으로 구성된다.

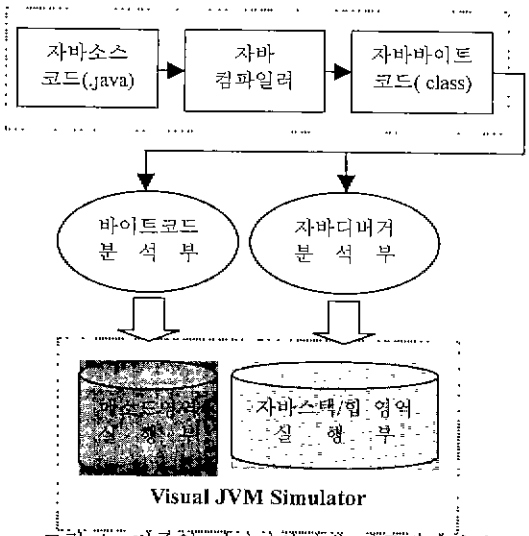

그림 1 비주얼 자바가상기계 시뮬레이터의 전체 구성도

3.1 비주얼 자바가상기계 시뮬레이터의 구성

먼저, 비주얼 자바가상기계 시뮬레이터는 자바 컴파일러를 통해 생성된 바이트코드를 입력으로 받고, 자바 바이트코드 분석부에서는 그 입력된 바이트코드를 분석하여 필요한 메소드 관련정보를 분리한다 분리된 이 정보는 다음 단계인 메소드 영역 실행부로 전달되고 시각화된다. 그러나 실제 바이트코드 분석부에서는 자바 스택영역의 지역 변수부의 변수와 힙 영역에 대한 정보는 알 수가 없다. 이러한 이유로 자바 디버거를 접목하여 자바 디버거 분석부에서는 자바 스택영역과 힙영역에 저장되는 자료를 분석하고, 그 영역에 헤당하는 자료는 분리되어 자바 스택과 힙 영역 실행부에 의해 출

릭되는 과정을 시각화하는 것이다.

자바가상기계는 해당 바이트코드를 적재한 후 가상기계에 대한 배치와 초기화, 실행의 순서로 이루어지는데, 비주얼 자바가상기계 시뮬레이터는 적재에서 초기화과정을 가상기계의 동작 그대로를 이용하고, 마지막 단계인 실행을 비주얼하게 시뮬레이션한다.

3.2 비주얼 자바가상기계 시뮬레이터의 구현

비주얼 자바가상기계 시뮬레이터는 자바컴파일러에 의해 적재되는 바이트코드에 대한 클래스의 메소드 영역과 자바 디버거(jdb)를 통해 지역변수가 할당되는 자바 스택부분과 인스턴스 호출시 생성되는 힙을 시각화한다. 본 논문에서 구현한 비주얼 자바가상기계 시뮬레이터는 윈도우 환경하에서 썬사가 제공하는 JDK 1.2 버전을 사용하고, 전체적인 화면 구성은 Symantec 의 Café Pro3.0 을 이용하여 구성하였다.

그림 2 는 Test.java 라는 예제 프로그램을 통해 Test class, Parent.class, Child.class 를 생성하는 과정을 비주얼 자바가상기계 시뮬레이터를 통해 보인 것이다.

```
class Parent {
    int p;
    static int sp;
    void mp() {
        System.out.println("Here:Parent"); }
    }
class Child extends Parent {
    int c;
    static int sc;
    void mc() {
        System.out.println("Here:Child"); }
    }
public class Test {
    public static void main(String[ ] args) {
        int lv;
        Child cc=new Child ();
        Parent pp=new Parent() ;
        cc mc();
        pp.mp();   }
    }
```

그림 2. 자바 예제 프로그램

그림 3 은 컴파일 후에 생성된 Test.class 를 입력으로 바이트코드가 가상기계를 통해 시뮬레이션되고, 자바 디버거를 통해 자바 스택영역과 힙 영역에 할당되는 것을 보이고 있다. 그리고 동시에 보다 쉬운 이해를 돕기 위해 프로그램 소스 리스트도 시각화한다.

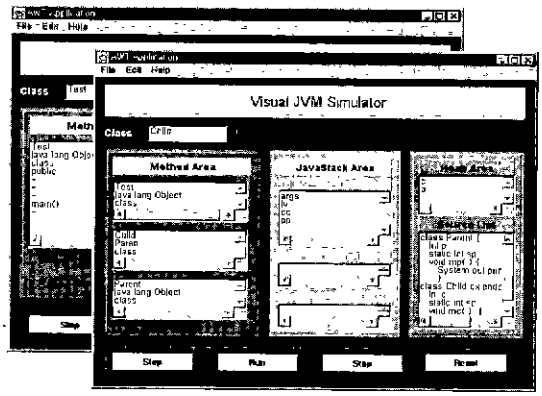

그림 3. 비주얼 자바가상기계 시뮬레이터의 실행 예

위 예에서 실행되고 있는 클래스는 메인 클래스를 나타내고, 현재 실행 중인 메소드를 보이고 있다. 이 메소드 영역은 메소드의 실제 코드와 그에 해당하는 모든 정보(타입, 상수 풀, 필드, 메소드, 클래스 변수)를 보여주고 있다. 그리고 초기의 값들은 자바 스택의 지역 변수에 나타난다. 위 그림에는 네 개의 버튼이 있는데, Step 버튼은 메인부터 상속받은 클래스를 단계별로 수행하고, Run 버튼은 계속적으로 모든 클래스를 수행하고, Stop 버튼은 수행을 중단하고, Reset 버튼은 메인 클래스부터 다시 수행하게 한다.

4. 결론 및 연구 결과의 활용

본 논문에서 소개한 '비주얼 자바가상기계 시뮬레이터'는 객체지향언어인 자바 프로그램을 컴파일하여 생성되는 바이트코드가 자바가상기계 내부의 메소드 영역에 어떻게 할당되는지, 그리고 자바 디버거를 통한 실질적인 자료의 추출로 자바 스택 영역, 힙영역에 어떻게 구성되는지를 시각적으로 표현한다. 또한, 소스 프로그램 리스트를 함께 출력함으로써 프로그램 소스 코드의 복잡성을 보다 쉽게 알 수 있게 하고, 상속관계를 쉽게 이해할 수 있도록 한다.

따라서 자바를 공부하는 사람뿐만 아니라 자바를 이용하여 프로그램을 구현하는 프로그래미에게도 자바로 작성된 프로그램을 체계적으로 분석하여 자바 프로그래밍 개발 환경으로 쉽게 활용 할 수 있도록 도움을 줄 것이다.

참 고 문 헌

[1] A. Krall and R. Grafl, CACAO-A 64 bit Java VM Just-in-Time Compiler, (1997)

[2] B. Venners, "Inside the Java Virtual Machine", McGraw-Hill, (1998)

[3] D Flanagan. "Java in a Nutshell, 2/E", O' Reilly. (1997)

[4] G. Cornell and C. S. Horstmann. "Core Java. 2/E", SunSoft Press. (1977)

[5] G. McGraw and E. Felten. "Java™ Security', Wiley, (1997)

[6] J. Gosling, "The Java Language Specification" , Addison-Wesley, (1996)

[7] J. Meyer and T Downing, "Java™ Virtual Machine", O'Reilly, (1997)

[8] M Campione and K. Walrath, "The Java™ Tutorial. Object-Oriented Programming for the Internet", Addison-Wesley, (1996)

[9] P. van der Lindon, "just Java. 2/E", SunSoft Press, (1997)

[10] T. Lindholm and F. Yellin, "The Java™ Virtual Machine Specification" , Addison-Wesley, (1996)

[11] http://java.sun.com/, Sun Microsystems, Java Home Page

[12] 류동항, 정민수, "바이트코드 분석기의 설계 및 구현", 한국정보과학회 제 25 회 춘계학술발표논문집, (1998)

[13] 김도우, 정민수, "자바 가상 기계 시뮬레이터의 설계 및 구현", 한국정보과학회 제 25 회 추계학술발표논문집, (1998)