자바 클래스 파일 실행 분석기

박상필[0] 고광만
상지대학교 컴퓨터정보공학부
{inpsp[0], kkman}@mail.sangji.ac.kr

Java Class File Execution Simulator

Park Sang-Pill[0] Ko Kwang-Man
School of Computer and Information Engineering, Sangji University

요 약

자바 언어에 대한 클래스 파일은 소스 프로그램의 의미를 자바 가상 기계에서 실행가능한 형식으로 변환된 형태이다. 이러한 클래스 파일의 구조 및 실질적인 실행 과정에 대한 분석은 디컴파일러 구성, 소스 프로그램의 디버깅 등에 편리성을 지원할 수 있다. 본 논문에서는 이러한 클래스 파일에 대한 분석 및 실제로 실행 과정을 보다 시각적으로 표현하기 위한 실행 과정 분석기에 관한 연구이다. 이를 위해 클래스 파일을 내용을 GUI 환경에서와 같이 접근 및 표현이 용이하도록 구현하였으며 이러한 클래스 파일의 실행 과정에서 핵심 정보를 저장하고 있는 메소드 영역 정보, 오퍼란드 스택 정보, 지역 변수의 정보를 시각적으로 표현하였다.

1. 서 론

자바의 클래스 파일은 바이너리 형태로서 자바 가상 기계에서 실행되어 실행 결과를 생성한다. 이러한 클래스 파일의 구조는 실행에 필요한 모든 정보를 저장하고 있으므로 그 내용 및 구조를 이해하는데는 많은 기초적인 지식을 필요로 한다. 즉, 클래스 파일을 분석하여 자바 가상 기계에서 수행의 기본 단위가 되는 명령어들이 자바 가상 기계에게 작업을 수행하게 하며 이러한 명령어들을 잘 조합해 원하는 결과를 얻을 수 있다. 자바 메소드의 구성 요소인 니모닉 코드들은 서로 조합되어 사용자가 원하는 작업들을 수행하게 되는데 이러한 작업의 수행을 위해서는 적은 수 또는 많은 명령어가 필요할 수도 있게 되며 메소드는 자신 내부에 있는 명령어들을 여러번 반복할 수도 있고, 특정 부분을 무시할 수도 있으며 특정 작업을 수행하기 위해 다른 메소드를 호출할 수도 있게 된다.

자바 가상 기계에서 니모닉 코드들을 분석하여 적절한 수행을 통해 실행하게 되는데 실제로 자바 가상 기계의 내부에서 니모닉 코드 수행되는 과정을 이해하는데는 많은 문제점을 가지고 있다, 본 논문은 자바 클래스 파일을 분석하여 클래스 파일의 내용 및 자바 가상 기계에서 실제로 수행되는 과정을 보다 시각적인 형식으로 구현하였다.

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

이를 위해 클래스 파일의 정보를 Constant_Pool, Class_file_info, Interface, Field, Method, Attribute 등의 6개의 분할된 정보 창으로 구성되어 나타내주며 특히, 자바 가상 기계에서 수행되는 명령어들을 분석하여 각각의 니모닉 코드들이 자바 가상 기계에서 수행되는 과정을 보다 편리하게 참조할 수 있도록 하였다.. 이러한 기능은 자바 가상 기계에서 니모닉 명령어들이 수행되는 과정을 보다 편리 참조할 수 있도록 시각적으로 구현하였다.

2. 기반 연구

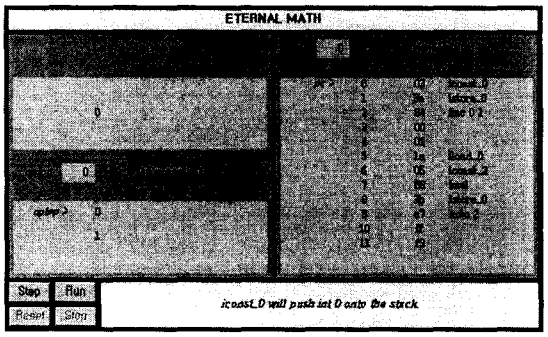
2.1 클래스 파일

자바 클래스 파일은 8비트 이진 스트림으로 구성되어 있다. 자바 클래스 파일의 구성은 올바른 클래스 파일인지 구분해 주는 Magic Number, 클래스 파일의 버전을 알려주는 Minor Version 과 Major Version, 클래스 파일의 상수 정보를 알 수 있는 Constant Pool Count, Constant Pool, 클래스 파일에 대한 정보를 가지고 있는 Access Flag, This Class, Super Class, 인터페이스의 정보를 갖는 Interface Count, Interfaces, 필드의 정보를 갖는 Field Count, Fields, 메소드에 정보를 갖는 Method Count, Methods, 클래스 파일의 속성에 대한 정보를 갖는 Attribute Count, Attributes등의 정보로 구성되어 있다..

본 논문에서는 클래스 파일에 대한 분석시에 Magic Number, Minor Version, Major Version, Constant Pool Count, Constant Pool의 정보를 constant 그룹으로 정보를 표시하였으며, Access Flag, This Class, Super Class를 합쳐 Class_file information 그룹으로 표시하였다.

2.2 Artima Software의 EteralMath

EteralMath[8]는 자바 애플릿을 통해 자바 가상 기계가 실행하는 과정을 보여준다. 이 프로그램은 하나의 메소드에 대한 자바 가상 기계의 실행 과정을 그림 1과 같이 표현한다.



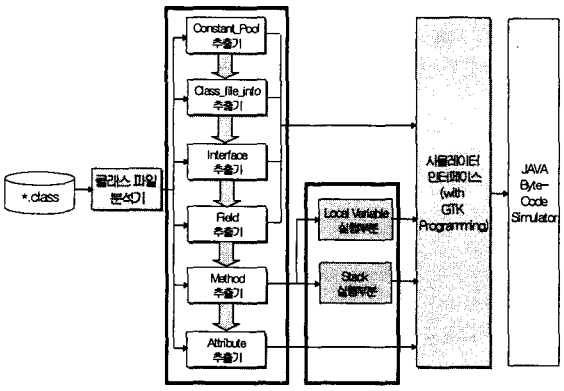
[그림 1] EteralMath

2.3 자바 프로그램을 위한 바이트코드 시뮬레이터

EteralMath와는 달리 클래스 파일을 입력으로 읽어들여서 분석된 메소드 정보를 가지고 자바 가상기계가 실행되는 과정을 보여주고 있다. 하지만 클래스 파일에 대한 분석 정보를 상세한 이해할 수 없는 단점을 가지고 있다. 실질적인 구현에서는 메소드 부분에 중점을 두어 시뮬레이터를 구현하였다[9].

3. 설계 및 구현

3.1 설계

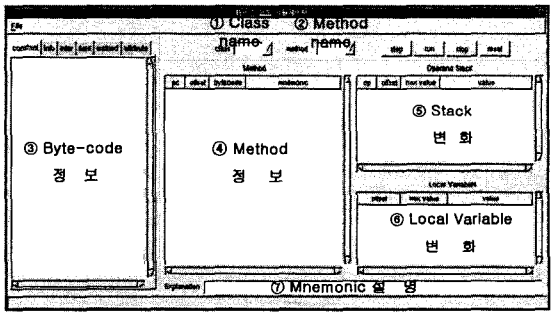


[그림 2] 자바 클래스 파일 실행 분석기 구성

그림 2는 자바 클래스 파일 실행 분석기의 전체 구성도이다. *.class 파일이 입력으로 들어오게 되면 클래스 파일 분석기가 클래스 파일을 Constant_Pool, Class_file_info, Interface, Field, Method, Attribute 의 6단계로 나누어서 차례대로 그 내용을 분석하게 된다. 분석된 내용중 자바 가상 기계의 동작 과정 분석을 위해서 필요한 Method 부분을 따로 읽어 들여 Local Variable와 Stack 영역으로 나누어 Local Variable 영역 정보와 Stack 영역 정보에 어떠한 내용들이 저장되고 어떻게 변화하는지에 대해 분석하게 된다. 이 과정을 클래스 파일의 나머지 부분들과 함께 좀더 보기 쉽게 GTK(GIMP Tool Kit)를 이용하여 작성된 인터페이스상에 표현되게 되어 최종의 자바 클래스 파일 실행 분석기가 완성되는 것이다.

3.2 구현

본 논문은 리눅스 환경에서 작동하도록 프로그램 되었으며 리눅스상의 인터페이스를 표현하게 위해 GTK(GIMP Tool Kit)를 이용하여 작성되었다.



[그림 3] 자바 클래스 파일 실행 분석기 화면 구성

그림 3은 자바 클래스 파일 실행 분석기의 화면구성에 대해 나타낸 것이다. File 메뉴를 이용하여 알고자 하는 클래스 파일을 선택한 후 해당 클래스 파일에 대한 내용을 보여주게 된다.

① Class name 부분은 현재 입력으로 받은 클래스의 이름을 나타낸다.

② Method Name 은 사용자가 사용하고자 하는 Method Name을 선택하는 부분이다. 사용자는 여러 개의 method 중에서 사용하고 싶은 method를 선택적으로 사용할 수 있게된다.

③ Byte-Code 정보 부분에는 입력된 *.class 파일에 대해 분석된 정보들이 들어가게 된다. Constant_Pool부터 Attribute에 이르는 6개의 창들이 보고자 하는 부분을 선택하여 그 내용들을 자세히 살펴볼 수 있는 형태로 나타내 주고 있다.

④ Method 정보 부분은 사용자가 선택한 Method Name 에 대해 어떠한 형태들의 명령어들이 사용되고 있는지를 나타내 주는 부분이다. pc는 현재 읽혀지고 있는 카운터의 위치이고, offset는 명령어들의 위치, 바이트 코드는 명령어들이 본래 어떤 바이트 코드로부터 왔는지를 알 수 있게 해주며,

mnemonic은 명령어들을 보여주게 된다.

⑤ Stack의 변화 부분은 명령어가 수행될 때 Stack 부분에서 일어나는 동작 상태를 표시한다.

⑥ Local Variable 부분은 지역변수들의 변화에 대해 보여주게 된다.

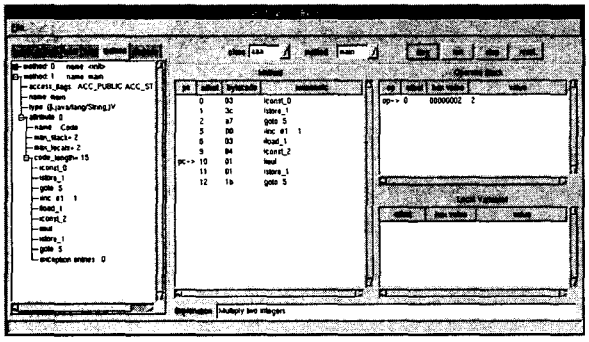
⑦ Mnemonic 설명 부분은 해당 명령어가 수행되면 어떠한 의미를 갖는지에 대해 설명을 나타내주게 된다.

바이트 코드 정보와 메소드에 대해 알고자 하는 클래스 파일을 열 경우 기본적으로 해당 클래스 파일을 읽어들여 나타내 주게 되고, Stack 정보와 Local Variable 정보, Mnemonic 설명 부분은 주어진 step, run, stop, reset 버튼을 통해 Method로부터 하나씩 또는 여러 개씩 읽어들이면서 변화되는 모양을 나타내 주게 된다.

3.3 실험 결과 분석

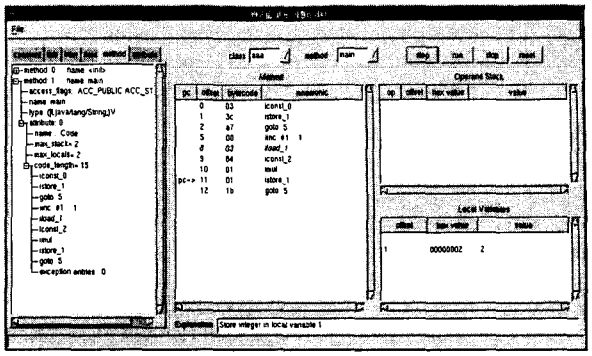
자바 언어로 작성된 임의의 예제 프로그램을 실행시킨 클래스 파일에 대해 실행 결과는 그림 4와 같다.

```
1 public class aaa {
2     public static void main(String args[]) {
3         int i = 0;
4         for (;;) {
5             i += 1;
6             i *= 2;
7         }
8     }
9 }
```



[그림 4] pc가 offset 10에 있을 때의 결과

javac를 이용하여 클래스 파일을 생성한 후 만들어진 aaa.class 라는 이름의 클래스 파일을 읽어들이면 그림4와 같은 형태로 클래스 파일에 대해 분석된 결과들이 나타나게 된다. 현재 pc는 offset이 10에 가있으므로 imul 이라는 명령어를 수행하게 된다. imul에 대한 설명은 Multuply two integers 라고 설명 부분에 표시되며 Operand Stack와 Local Variables 에 변화된 값이 나타난다. 현재에는 Operand Stack의 0번째 offset에 연산된 결과가 저장된 형태를 보여주게 된다.



[그림 5] pc가 offset 11로 step 된 결과

그림 5는 그림 4의 상태에서 step 버튼을 클릭 함으로서 다음 명령어를 수행하게 되는 것을 보여주고 있다. istore_1을 수행하여 현재 Stack에 있는 값을 Local Variables의 첫 번째 슬롯에 넣어주게 되므로 그림 4와 같은 실행결과를 얻을 수 있게 된다.

4. 결론 및 향후 연구

본 논문은 자바 클래스 파일이 가지고 있는 정보를 이용하여 클래스 파일이 어떻게 구성되어 있는지를 단순한 텍스트 형태가 아닌 시각화 형태로 보여주어 사용자가 필요한 정보에 대한 접근을 용이하게 함은 물론이고, 자바 가상 기계가 클래스 파일을 읽어들여 어떠한 명령들을 수행하여 스택과 지역 변수들이 어떻게 동작되는지에 대해 직접 실행시키며 살펴볼 수 있게 작성되었다. 이러한 연구는 차후 클래스 파일을 이용하여 다른 분야에 도움을 줄 수도 있으며 자바 가상 기계를 실행 분석기를 통해 실행해 봄으로 인해 좀더 나은 프로그램의 최적화를 이룰 수 있을 것이다.

또한 본 논문 결과는 현재 다양한 분야에서 활용되고 있는 Java 언어를 위한 보다 효과적인 실행 환경 구축 및 개발 환경 개선에 활용될 수 있다.

5. 참고 문헌

[1] Jon Meyer and Troy Downing, JAVA Virtual Machine, O'REYLLY, 1997.

[2] Ken Arnold and James Gosling, The Java Programming Language, Sun Microsystems, 1996.

[3] Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, O'REYLLY, 1999.

[4] Joshua Engel, Java Virtual Machine Programming, Info-book, 2000.

[5] http://members.fortunecity.com

[6] http://www.tip.net.au

[7] http://www,gtk.org/tutorial

[8] http://www.artima.com/insidejvm/applets/EternalMath.html

[9] 김도우, 정민수 "자바 프로그램 분석을 위한 바이트 코드 시뮬레이터" 한국 정보처리학회지, 제7권 제7호, 2000.