구문 트리를 이용한 자바 바이트코드에서 SIL로의 번역기

김영근°, 권혁주, 이양선
서경대학교 컴퓨터공학과
e-mail:{ykkim, hjkwon, yslee}@pl.skuniv.ac.kr

Java Bytecode-to-SIL Translator using an Abstract Syntax Tree

Young-Koun Kim°, Hyeok-Ju Kwon, Yang-Sun Lee
*Dept of Computer Engineering, SeoKyeong University

요 약

자바는 현재 가장 널리 사용되는 범용 프로그래밍 언어중 하나로 컴파일러에 의해 중간언어인 바이트코드로 변환되며 JVM(Java Virtual Machine)에 의해 실행되는 플랫폼 독립적인 언어이다. SIL (Standard Intermediate Language)은 Microsoft사의 .NET 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 시스템을 위한 중간언어로 가상기계인 EVM(Embedded Virtual Machine)에서 실행된다.

본 논문에서는 자바 프로그램을 컴파일하여 생성된 클래스 파일에서 Oolong 코드를 추출하고 추출된 Oolong 코드를 EVM의 SIL 코드로 변환하여 자바로 구현된 프로그램이 EVM에서 실행되도록 하는 Bytecode-to-SIL 번역기 시스템을 구현하였다. 그리고 본 번역기 시스템을 다른 플랫폼에 용이하게 설치하기 위한 재목적성(retargetability)을 위해 단일패스(one-pass)을 사용하는 기존의 번역기들과 달리 AST를 이용한 컴파일러 기법을 사용하여 AST가 가지고 있는 정보에 대해 최적화 작업을 수행하여 보다 효과적인 코드 변환을 할 수 있도록 설계하였다.

1. 서론

가상기계는 하드웨어를 대신할 수 있는 소프트웨어이며, 하드웨어-소프트웨어간에 인터페이스 역할을 하는 일종의 미들웨어로 볼 수 있다. 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스, 디지털 TV, 홈 관리 시스템 등에 탑재할 수 있는 핵심 기술로서 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

EVM(Embedded Virtual Machine)은 Microsoft사의 .NET 언어와 SUN사의 Java 언어 등을 모두 수용할 수 있는 임베디드 가상 기계이고, SIL (Standard Intermediate Language)은 EVM의 중간언어로 .NET 언어, Java 언어 등으로 작성된 프로그램을 어셈블리 언어 형태인 *.sil 파일로 변환한 후에, EVM에서 실행한다[8,9,10].

SIL은 다양한 프로그래밍 언어를 수용하기 위해서 기존의 가상기계 어셈블리 언어들의 분석을 토대로 정의 되었으며, 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합을 갖고 있다[1,2,4,6,8].

본 논문에서는 자바 프로그램을 컴파일 하여 생성된 클래스 파일에서 Oolong 코드를 추출하고 추출된 Oolong 코드를 EVM의 가상기계 코드인 SIL 코드로 변환하여 자바로 구현된 프로그램이 EVM에서 실행할 수 있도록 하는 Bytecode-to-SIL 번역기 시스템을 구현하였다.

2. 바이트코드와 SIL코드

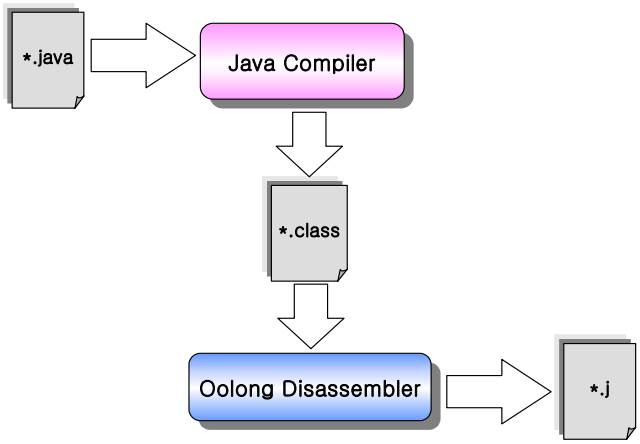
2.1 바이트코드

JVM(Java Virtual Machine)은 자바로 작성된 프

본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0) 지원으로 수행되었음.

로그램을 자바 컴파일러에 의해 클래스 파일 형태의 중간언어를 생성하고 실행하는데, 클래스 파일이 바이너리 형식으로 되어 있기 때문에 분석하거나 수정하기가 매우 어렵다. 이에 비해 또 다른 형태의 자바 중간언어인 Oolong 코드로 작성된 파일은 클래스 파일 내에 있는 바이트코드와 유사한 형태의 실제 프로그램 로직 부분을 텍스트 형식으로 저장하므로 프로그래머 입장에서 좀 더 쉽게 접근할 수 있기 때문에 코드의 이해와 프로그램의 작성 및 수정을 용이하게 한다. Oolong 코드는 존 메이어(John Meyer)의 Jasmin 언어를 기반으로 만들어 졌으며 프로그래머가 바이트코드 수준에서 프로그램을 작성할 수 있도록 설계되어 있다[1,2,4,7].

본 논문에서 개발한 번역기 시스템에 입력으로 들어갈 Oolong 코드는 자바 클래스 파일로부터 Oolong 코드를 추출하는 과정이 필요하다. [그림1]은 클래스 파일로부터 Oolong 코드를 추출하는 과정이다.

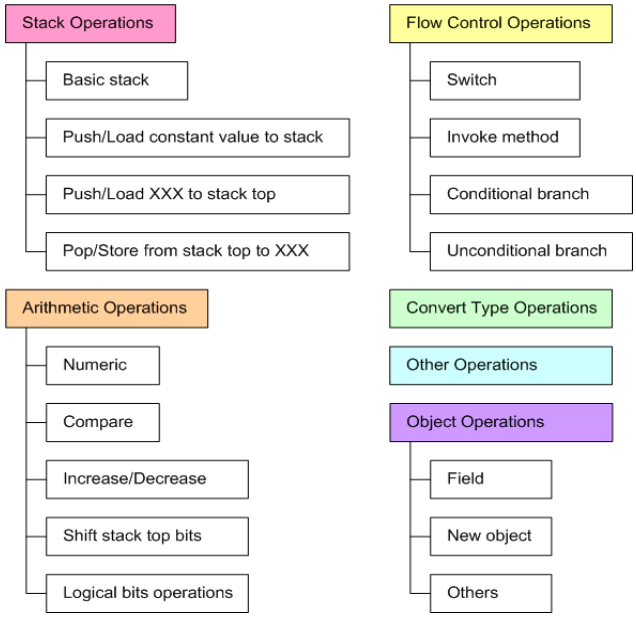


[그림1] Oolong 코드 추출과정

2.2 SIL 코드

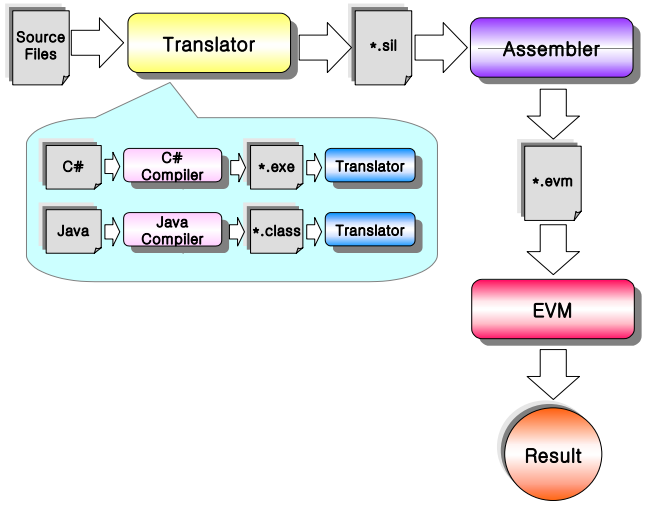
SIL(Standard Intermediate Language)은 EVM (Embedded Virtual Machine)의 중간언어로 객체 지향 언어와 순차적인 언어를 모두 수용할 수 있는 코드로 설계되었다.

SIL 코드에는 크게 의사코드(Pseudo code)와 연산코드(Operation code)가 있다. 의사코드는 가독성을 높이기 위해서 원시코드 수준의 키워드를 사용하였다. 연산코드는 크게 6개의 카테고리로 되어있으며, 니모닉(Mnemonic)은 연산코드가 수행하는 일을 약자로 구성하였다. 연산코드는 매개변수를 가질 수 있으며 매개변수의 수는 코드마다 다르다. 또한, 연산코드의 다형성(Polymorphism)을 위해서 피연산자가 타입을 유지하도록 설계되었다. [그림2]는 SIL 연산코드의 카테고리이다[8,10].



[그림2] SIL 연산 카테고리

번역기(Translator)는 Java 프로그램과 .NET 언어로 작성된 프로그램을 입력으로 받아 컴파일된 파일을 SIL로 번역하고, 번역된 SIL은 SIL 어셈블러에 의해 EVM 실행파일(*.evm)로 변환되며, *.evm 파일은 가상기계인 EVM에서 실행된다. [그림3]은 전체 EVM 시스템의 구성도이다[10].



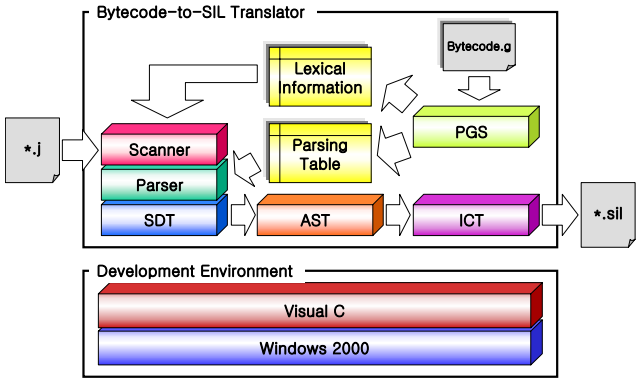
[그림3] EVM 시스템 구성도

3. Bytecode-to-SIL 번역기 시스템

Bytecode-to-SIL 번역기 시스템은 클래스 파일로부터 추출된 Oolong 코드를 입력으로 받아 EVM의 중간언어인 SIL을 생성하는 번역기이다. 번역기는 다른 플랫폼에 용이하게 설치하기 위한 재목적성

을 위해 단일패스 방식을 사용하는 기존의 번역기들과 달리 AST를 이용한 컴파일러 기법을 사용하여 AST가 가지고 있는 정보에 대해 최적화 작업을 수행하여 보다 효과적인 코드 변환을 할 수 있도록 설계하였다[9].

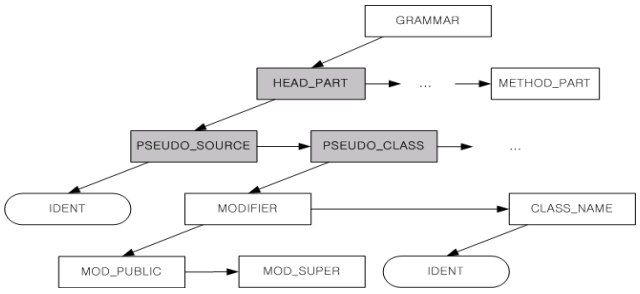
시스템 구현 환경은 Windows 2000에서 Visual C를 사용하였다. 번역기 시스템의 구성은 스캐너, 파서, SDT(Syntax Directed Translation), AST (Abstract Syntax Tree) 그리고 ICT(Intermediate Code Translator)의 5부분으로 구성된다. [그림4]는 번역기 프로그램의 구성도이다.



[그림4] 번역기 시스템 구성도

첫번째 단계인 스캐너와 두번째 단계인 파서를 구현하기 위해서 분석한 바이트코드를 기반으로 하여 context-free 문법을 고안하였다. 그리고 문법을 입력으로 받아서 PGS(Parser Generator System)를 이용해 어휘정보와 파싱테이블을 얻는다. 어휘정보를 가지고 스캐너에서 Oolong 코드(*.j)를 토큰단위로 파서에 전달하고 파싱테이블을 참조하여 파서에서는 스캐너로부터 토큰을 읽으면서 파싱을 한다.

세번째 단계로 파싱한 노드와 트리구조를 받아서 입력문법의 구조에 따라 생성규칙에 대한 의미 수행코드를 작성하여 AST를 생성한다. [그림5]는 AST의 기본구조를 보여주는 것이다.



[그림5] AST의 기본구조

마지막 단계로 코드 변환 부분인 ICT가 생성한 AST를 입력으로 받아 바이트코드와 의미적으로 동일한 SIL 코드(*.sil)를 생성한다. [표1]은 코드를 변환하기 위해 Oolong 코드와 SIL 코드의 니모닉을 매핑한 것이다.

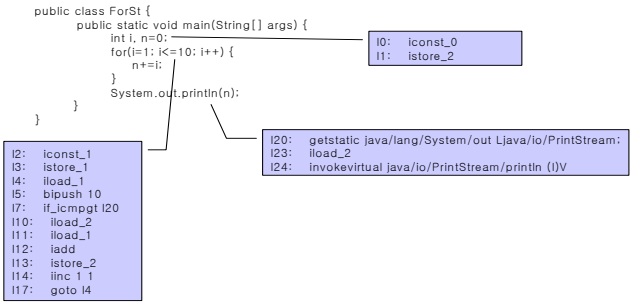
[표1] Bytecode-to-SIL 니모닉 매핑 테이블

Oolong	SIL
iload	ldl
istore	str
.....	.....
iadd, isub, imul, idiv	add, sub, mul, div
if_icmpeq	eq
.....	.....
l2i, f2i, d2i	convi
.....	.....

이와같은 과정을 거쳐서 변환된 SIL 코드는 SIL 어셈블러를 통해 EVM 실행파일(*.evm)로 생성되고, 가상기계인 EVM에서 실행된다.

4. 실행결과 및 분석

다음은 제어문에 대한 번역하는 과정을 보여주는 예제이다.



[예제1] 자바 프로그램과 Oolong 코드

```
Nonterminal: PROGRAM
    Nonterminal: HEAD_PART
        Nonterminal: PSEUDO_SOURCE
            Nonterminal: FILE_NAME
                Terminal: ForSt
.....중간생략
        Nonterminal: METHOD_START
            Nonterminal: MODIFIER_METHOD
                Nonterminal: MOD_PUBLIC
                Nonterminal: MOD_STATIC
            Nonterminal: METHOD_NAME
                Terminal: main
            Nonterminal: DESCRIPTOR
                Nonterminal: ARGUMENT
                    Nonterminal: DESC_ARRAY
                    Nonterminal: DESC_REFERENCE
                        Nonterminal: CLASS_PATH
                            Terminal: java
                            Terminal: lang
.....중간생략
```

```
                    Nonterminal: LABEL_NAME
                    Terminal: 127
                Nonterminal: RETURN
            Nonterminal: METHOD_END
```

[예제2] Oolong 코드와 동일한 의미를 가진 AST

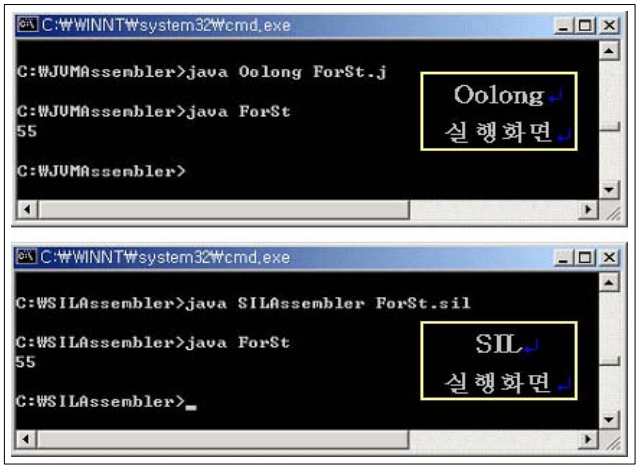
```
.class public super ForSt
.super java/lang/Object

.method public V <init> ()
.locals o
l0:     ldl 0
l1:     call V java/lang/Object/<init> ()
l4:     ret
.end

.method public static V main ([Ljava/lang/String;)
.locals o i i
l0:     ldc 0
l1:     str 2
l2:     ldc 1
l3:     str 1
l4:     ujp l14
l7:     ldl 2
l8:     ldl 1
l9:     add
l10:    str 2
l11:    inc 1 1
l14:    ldl 1
l15:    ldc 10
l17:    le l7
l20:    ldsfld Ljava/io/PrintStream; java/lang/System/out
l23:    ldl 2
l24:    calli V java/io/PrintStream/println (I)
l27:    ret
.end
```

[예제3] 번역기에 의해 생성된 SIL 코드

[예제4]는 Oolong 코드와 번역기로 변환되어 생성된 SIL 코드를 어셈블해서 실행한 결과화면을 나타낸 것이다.



[예제4] Oolong 코드와 SIL 코드의 실행화면

5. 결론 및 향후연구

SIL은 임베디드 시스템을 위한 가상 기계 코드로 객체 지향 언어와 순차 언어를 모두 수용할 수 있는 스택 기반의 연산코드 집합을 가지고 있다. 본 논문에서는 자바 컴파일러에 의해 생성된 바이트코드의 중간 언어로 작성된 클래스 파일에서 Oolong 코드를 추출한 뒤, 추출된 Oolong 코드를 SIL 코드로 변환하여 EVM에서 실행할 수 있는 Bytecode-to-SIL 번역기 시스템을 구현하였다. 또한, 다른 플랫폼에 용이하게 설치하기 위한 재목적성을 위해 단일패스 방식을 사용하는 기존의 번역기들과 달리 AST를 이용한 컴파일러 기법을 사용하여 AST가 가지고 있는 정보에 대해 최적화 작업을 하여 보다 효과적인 코드 변환을 할 수 있도록 설계하였다.

앞으로 자바 플랫폼 프로그래밍 언어에서 지원하는 기능 및 모듈들을 EVM 플랫폼 환경에서도 동일하게 사용할 수 있도록 번역기를 확장하고, Oolong 코드와 SIL 코드의 분석을 통해 보다 효과적인 코드를 낼 수 있는 코드 최적화를 위한 연구를 수행할 예정이다.

참고문헌

[1] Bill Venners, "Inside JAVA Virtual Machine SE", McGraw-Hill, 1999.

[2] Hoshua Engel, "Programming for the Java Virtual Machine", Addison-Wesley, 1999.

[3] John Gough, "Compiling for the .NET Common Language Runtime(CLR)", Prentice Hall, 2002.

[4] John Meyer & Troy Downing, "Java Virtual Machine", O'REILLY, 1997.

[5] Microsoft Corporation, "Common Language Infrastructure(CLI)", 2001.

[6] Ser Lidin, ".NET IL ASSEMBLER", Microsoft Press, 2002.

[7] Tim Lindholm & Frank Yellin, "The Java^{tm} Virtual Machine Specification Second Edition", Addision-Wesley, 1997.

[8] 남동근・윤성림・오세만, "가상기계를 위한 어셈블리 언어", 정보처리학회 2003 춘계학술발표논문집, Vol.10, No.1, pp.783-786, Mar. 2003.

[9] 정지훈・박진기・이양선, "자바 바이트코드의 .NET MSIL 중간언어 번역기", 정보처리학회 2003 추계학술발표논문집, Vol.10, No.2, pp.663-666, Nov. 2003.

[10] 정한종・윤성림・오세만, "가상 기계를 위한 실행 파일 포맷", 정보처리학회 2003 추계학술발표논문집, Vol.10, No.2, pp.647-650, Nov. 2003.