자바 언어를 위한 중간 언어 번역기

정지훈, 박진기, 이양선
서경대학교 컴퓨터 공학과

Intermediate Language Translator for Java Language

Ji-Hoon Jung, Jin-Ki Park Yang-Sun Lee
*Dept. of Computer Engineering, Seokyeong University

요 약

자바와 .NET 언어는 프로그래머들이 프로그램을 개발하는데 가장 널리 사용되고 있는 언어이다. 프로그래머가 작성한 자바 프로그램은 JVM 플랫폼에서는 실행이 되지만 .NET 플랫폼에서 실행이 되지 않고, 반대로 C#과 같은 .NET 언어로 작성한 프로그램은 .NET 플랫폼에서는 실행이 되지만 JVM 플랫폼에서 실행이 되지 않는다. 그러므로 자바 프로그래머는 .NET 플랫폼 환경에 맞추어 프로그래밍하지 못하고, .NET 프로그래머는 JVM 플랫폼 환경에 맞추어 프로그래밍하지 못하는 단점이 있다. 본 논문에서 연구한 Bytecode-to-MSIL 번역기는 위의 단점을 해결한 것이다. 먼저 자바로 작성된 프로그램을 컴파일하여 생성된 클래스 파일(*.class)로부터 Oolong 역어셈블러를 이용하여 Oolong 코드를 추출한다. 추출된 Oolong 코드를 본 논문에서 연구한 Bytecode-to-MSIL 번역기로 .NET의 중간언어인 MSIL 코드로 변환하여 .NET 어셈블러로 실행파일을 만들어 자바 언어로 구현된 프로그램이 .NET 환경에서도 실행될 수 있도록 한다. 따라서, 자바 프로그래머나 .NET 프로그래머는 JVM 이나 .NET 플랫폼 환경에 관계없이 프로그램을 작성하여 실행시킬 수 있다.

1. 서론

자바는 썬 마이크로시스템즈사의 제임스 고슬링(James Gosling)에 의해 고안된 언어로 운영체제 및 하드웨어 플랫폼에 독립적인 차세대 언어로 최근에 가장 널리 사용하는 범용 프로그래밍 언어 중 하나이다. 자바 프로그램은 컴파일러에 의해 각 플랫폼에 독립적인 중간 코드 형태의 바이트코드로 변환된 클래스 파일이 생성되면 JVM(Java Virtual Machine)에 의해 실행된다. 마이크로소프트사의 .NET 언어는 프로그래머들의 요구를 충족시키고 썬사의 JVM 환경과 자바 언어에 대응하기 위해서 개발된 언어이다[2,5,8,11,12]. C#과 같은 .NET 언어는 컴파일러에 의해 MSIL(MicroSoft Intermediate Language) 코드로 번역되며 .NET 플랫폼 환경에서 런타임 엔진인 CLR(Common Language Runtime)에 의해 실행이 된다.[1,3,4,7,13].

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

자바와 .NET 언어는 프로그래머들이 프로그램을 개발하는데 가장 널리 사용되고 있는 언어이다. 프로그래머가 작성한 자바 프로그램은 JVM 플랫폼에서는 실행이 되지만 .NET 플랫폼에서 실행이 되지 않고, 반대로 C#과 같은 .NET 언어로 작성한 프로그램은 .NET 플랫폼에서는 실행이 되지만 JVM 플랫폼에서 실행이 되지 않는다. 그러므로 자바 프로그래머는 .NET 플랫폼 환경에 맞추어 프로그래밍하지 못하고, .NET 프로그래머는 JVM 플랫폼 환경에 맞추어 프로그래밍하지 못하는 단점이 있다.

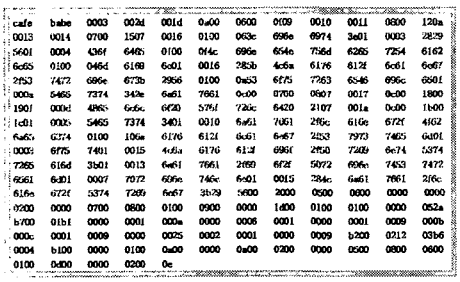
본 논문에서 연구한 Bytecode-to-MSIL 번역기는 위의 단점을 해결한 것이다. 먼저 자바로 작성된 프로그램을 컴파일하여 생성된 클래스 파일(*.class)로부터 Oolong 역어셈블러(Gnoloo.class)를 이용하여 Oolong 코드를 추출한다. 추출된 Oolong 코드를 본 논문에서 연구한 Bytecode-to-MSIL 번역기로 .NET의 중간언어인 MSIL 코드로 변환하여 .NET 어셈블러(ilasm.exe)로 실행

파일을 만들어 자바 언어로 구현된 프로그램이 .NET 환경에서도 실행될 수 있도록 한다. 따라서, 자바 프로그래머나 .NET 프로그래머는 JVM 이나 .NET 플랫폼 환경에 관계없이 프로그램을 작성하여 실행시킬 수 있다.

2. 중간 언어

2.1 바이트코드

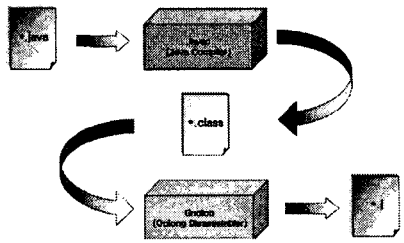
일반적으로 자바 언어를 컴파일하면 생성되는 것으로서, Opcode가 한 바이트로 구성되어 있기 때문에 바이트코드(Bytecode)라고 부른다. 한 명령은 한 바이트 이상으로 구성되는데 명령마다 다른 길이를 가진다. 이 바이트코드 해석은 JVM 또는 자바 API를 통해 운영체제에 연결되어 원하는 작업을 할 수 있다. 바이트코드는 플랫폼 독립적으로 수행된다. 따라서, 새로운 프로그램을 개발 할 때 이식성(portability)에 따른 개발노력의 단축뿐만 아니라 소프트웨어의 확산이 증가할 것이다. [그림1]은 바이트코드가 자바 클래스 파일 내에 바이너리 형식으로 저장되어 있는 것을 보여주는 예이다[2,5,11,12].



[그림1] 클래스 파일의 내부

2.2 Oolong 코드

JVM은 자바 프로그래밍 언어로 작성된 소스코드를 클래스 파일 형식을 사용해서 저장하고 실행하는데, 클래스 파일이 바이너리 형식으로 되어 있기 때문에 분석하거나 수정하기가 매우 어렵다. 이에 비해 또 다른 형태의 자바 중간언어인 Oolong 코드는 읽고 쓰기가 클래스 파일 형식에 비해서 훨씬 쉽다. 이 Oolong 코드는 존 메이어(John Meyer)의 자스민(Jasmin)언어를 기반으로 만들어 졌으며 프로그래머가 바이트 코드 수준에서 프로그램을 작성할 수 있도록 설계되어 있다. [그림2]는 자바 클래스 파일로부터 역어셈블하여 Oolong 코드를 추출하는 과정이다. 이렇게 추출된 Oolong 코드를 중간언어 번역기 시스템의 입력으로 사용할 것이다[2,5,11].



[그림2] Oolong 코드 추출과정

2.3 MSIL 코드

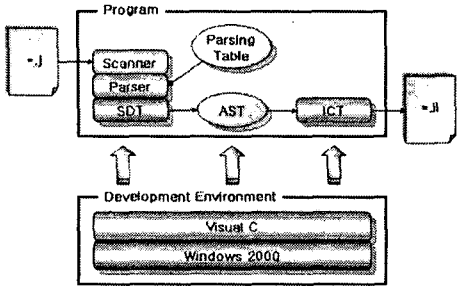
MSIL(MicroSoft Intermediate Language)은 C#을 포함한 .NET 언어들의 중간언어로 .NET 언어로 작성된 소스코드가 컴파일되면 MSIL로 작성된 코드가 생성된다. MSIL은 오퍼런드 스택을 이용하는 스택 기반의 명령어 집합으로 언어 상호 운용성과 플랫폼 독립성의 두 가지 큰 특징을 가지는 언어이다. 또한, 처음부터 JIT(Just In Time)를 목적으로 설계되었으며, 자바 언어와는 다르게 처음부터 언어 독립적으로 설계되어 포괄적인 프로그래밍을 목적으로 하기 때문에 프로그램의 기능 및 구조의 변화에 잘 적응하는 언어이다. [그림3]은 .NET의 실행 파일로부터 역어셈블하여 MSIL 코드를 추출하는 과정이다. 이렇게 추출된 MSIL 코드로 중간언어 번역기 시스템이 번역한 MSIL 코드의 결과와 비교 분석하여 더욱 정확한 번역을 할 수 있다[1,3,6,13].

[그림] MSIL 코드 추출과정

3. 자바 언어를 위한 중간 언어 번역기

3.1 시스템 구조

본 번역기 시스템은 윈도우즈 2000 환경 하에서 비주얼 C를 이용하여 구현하였으며, 스캐너, 파서, SDT(Syntax Directed Translation), AST (Abstract Syntax Tree) 그리고 ICT(Intermediate Code Translator)로 구성된다. [그림4]는 시스템 구성도를 나타낸 것이다.



[그림4] 번역기 시스템 구성도

3.2 Bytecode-to-MSIL 번역기

Bytecode-to-MSIL 번역기를 정형화한 형태로 구성하기 위해 Oolong 코드의 명령어 집합을 context -free 문법(Oolong.g)을 이용하여 설계하였고, 설계된 문법을 가지고 파서 생성기(PGS)를 이용하여 문법에 대한 어휘 정보와 파싱 테이블을 얻는다. 여기서 얻어진 어휘 정보를 이용하여 스캐너를 작성하고, 파싱 테이블을 이용하여 파서를 구성하였다. Bytecode-to -MSIL 번역기가 자바의 클래스 파일에서 추출한 Oolong 코드를 입력으로 받으면 파서에서 스캐너를 호출하여 토큰을 인식한 후 파싱 테이블을 참조하여 LR 구문 분석을 한다. 파싱을 성공하면 입력 문법의 구조에 따라 그 생성 규칙에 대한 의미 수행 코드를 작성하여 AST를 생성하고 이 AST를 탐색하면서 코드 변환을 하는 ICT를 작성하였다. ICT에서는 Oolong 코드와 MSIL 코드간에 서로 기능적으로 동일한 부분이 될 수 있도록 명령어를 번역하고, Oolong 코드에 포함되어 있는 라이브러리 함수들이 .NET의 MSIL 코드에 포함되어 있는 라이브러리 함수와 대응될 수 있도록 매크로 처리를 하며, Oolong 코드가 가지고 있는 의사 코드도 MSIL 코드가 가지고 있는 의사 코드와 동일한 기능을 수행 할 수 있도록 변환을 하였다. 다음 [표1]과 [표2]는 Oolong 코드와 MSIL 코드를 비교 할 수 있도록 매칭시킨 테이블이다. ICT에서는 이것을 바탕으로 코드 변환을 한다.

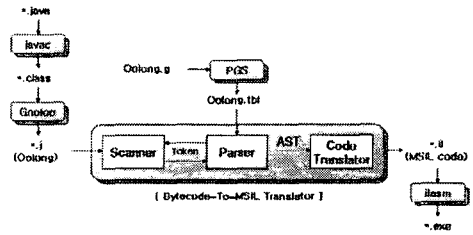
Oolong	MSIL
String	string
...	...
byte	int8
short	int16
int	int32
long	int64
...	...
float	float32

[표1] 데이터 타입 매핑표

Oolong	MSIL
nop	nop
...	...
iload_num	ldloc.num
istore_num	stloc.num
iadd, isub	add, sub
imul, idiv	mul, div
...	...
return	ret

[표2] 명령어 집합 매핑표

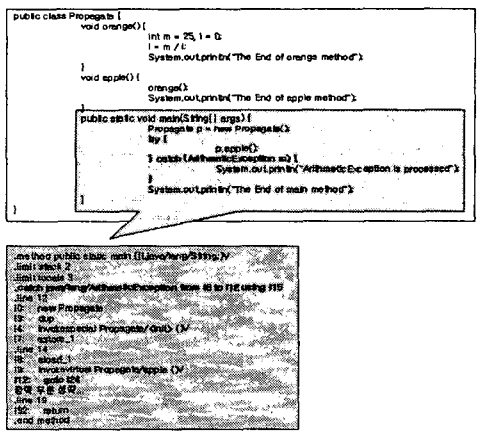
이렇게 함으로써, 입력 코드인 Oolong 코드의 기능과 출력 코드인 MSIL 코드의 기능을 같게 하고, 결과적으로 Oolong 코드와 의미적으로 동등한 MSIL 코드를 생성함으로써 자바 프로그램이 .NET 플랫폼 환경에서 실행될 수 있다. [그림5]는 Oolong 파일이 번역기 시스템에 입력으로 들어가 출력으로 MSIL 코드가 나오면 그것을 다시 MSIL IL 어셈블러를 이용해 실행이 되는 과정을 나타낸 것이다.



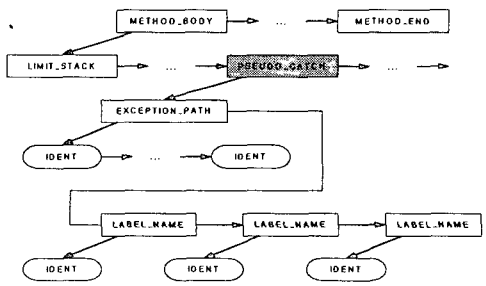
[그림5] Bytecode-to-MSIL 번역기

3.3 실험 결과 및 분석

다음 [예제1]부터 [예제3]까지는 Bytecode-to-MSIL 번역기를 사용하여 하여 예외 처리 소스를 번역하고 그 결과를 살펴본 것이다.



[예제1] 자바 프로그램 및 추출된 Oolong 코드

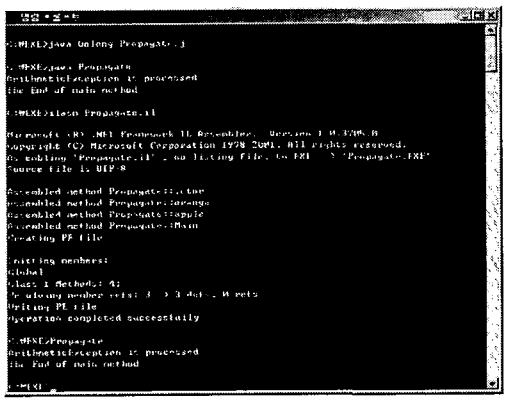


[예제2] 예외 처리에 대한 AST 형태

```
헤드 부분과 생성자 메소드 생략...
.method private hidebysig instance void orange() cil managed {
    ...
}
.method private hidebysig instance void apple() cil managed {
  ...
}
.method public hidebysig static void  Main() cil managed {
  .entrypoint
  .maxstack  2
  .locals init (int32 V_0, class Propagate V_1,
           class [mscorlib]System.ArithmeticException V_2)
  IL0:  newobj     instance void Propagate::.ctor()
  IL3:  nop
  IL4:  nop
  IL7:  stloc.1
  .try {
    IL8:  ldloc.1
    IL9:  callvirt   instance void Propagate::apple()
    IL12:  leave.s    IL24
  }
  catch [mscorlib]System.ArithmeticException {
    IL15:  stloc.2
    IL16:  nop
    IL19:  ldstr      "ArithmeticException is processed."
    IL21:  call       void [mscorlib]System.Console::WriteLine(string)
           leave.s    IL24
  }
  출력 부분 생략...
  IL32:  ret
}
```

[예제3] 생성된 MSIL 코드

[그림6]은 번역기를 통해 생성된 MSIL 코드를 실행파일로 변환해서 실행한 결과를 자바 프로그램의 출력결과와 비교한 화면이다.



[그림6] 실행 결과 비교

4. 결론

자바로 작성된 프로그램은 JVM 플랫폼에서는 실행이 되지만 .NET 플랫폼에서 실행이 되지 않고, 반대로 C#과 같은 .NET 언어로 작성된 프로그램은 .NET 플랫폼에서는 실행이 되지만 JVM 플랫폼에서 실행이 되지 않는다. 이런 이유로 본 논문에서는 자바소스를 컴파일하여 생성된 클래스 파일에서 Oolong 코드를 생성하고 생성된 Oolong 코드를 .NET의 MSIL 코드로 변환하여 자바로 구현된 프로그램이 .NET 환경에서 실행되도록 하는 Bytecode-to-MSIL 번역기 시스템을 구현하였다. 따라서, 자바 프로그래머는 JVM이나 .NET 플랫폼 환경에 관계없이 프로그램을 작성하여 실행시킬 수 있다. 앞으로 자바 플랫폼 프로그래밍 언어에서 지원하는 기능 및 모듈들을 .NET 플랫폼 환경에서도 동일하게 사용할 수 있도록 번역기를 확장하고, Oolong 코드와 MSIL 코드 자체를 분석하여 실험을 통해 보다 나은 코드를 낼 수 있는 코드 최적화를 위한 연구를 수행할 예정이다.

[참고문헌]

[1] Andrew Troelsen, C# and the .NET Platform, 2nd ed, APRESS, 2001.

[2] Bill Venners, Inside the JAVA Virtual Machine, 2nd ed., McGraw-Hill, 2000.

[3] Don Box & Chris Sells, Essential .NET Volume 1 The Common Language Runtime, Addison Wesley, 2002.

[4] Arthur Gittleman, Computing With C# and the .Net Framework, Jones & Bartlett Publishers, 2003.

[5] Hoshua Engel, Programming for the Java Virtual Machine, Addison Wesley, 1999.

[6] Jeff Prosise, Programming Microsoft .NET, Microsoft Press, 2002.

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

[8] Ken Arnold, James Gosling & David Holmes, The Javatm Programming Language, 3rd ed., Addison Wesley, 2000.

[9] Microsoft Corporation, Common Language Infrastructure(CLI), Dec. 2001.

[10] Serge Lindin, Inside Microsoft .NET IL Assembler, Microsoft Press, 2002.

[11] Troy Downing & Jon Meyer, Java Virtual Machine, O'REILLY, 1997.

[12] Tim Lindholm & Frank Yellin, The Java^{tm} Virtual Machine Specification, 2nd ed, Addison Wesley, 1999.

[13] Tom Archer, INSIDE C#, 2nd ed, 정보문화사, 2002.

[14] 오세만, 컴파일러 입문, 정익사, 2000.

[15] 오세만 & 이양선 & 김상훈 & 고광만, 자바 입문, 생능출판사, 1998.