.NET 언어를 위한 중간 언어 번역기

최성규*, 박진기*, 이양선*
*서경대학교 컴퓨터공학과
e-mail: {skche, jkpark, java}@nit.skuniv.ac.kr

An Intemediate Language Translator for .NET Languages

Sung-Kyou Che*, Jin-Ki Park*, Yang-Sun Lee*
*Dept. of Computer Engineering, Seokyeong University

요 약

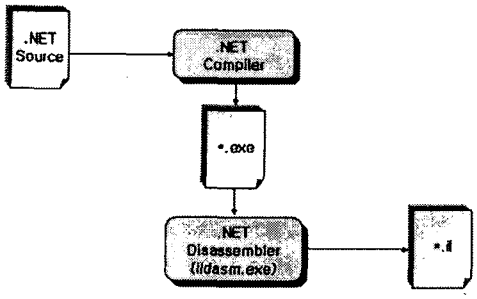
자바는 썬 마이크로시스템즈사가 개발한 언어로서 현재 가장 널리 사용되는 프로그래밍 언어 중 하나이며, 컴파일러에 의해 플랫폼에 독립적인 바이트코드를 바이너리 형태로 가지고 있는 클래스 파일을 생성하면 JVM에 의해 어떠한 하드웨어나 운영체제에 상관없이 수행이 가능한 플랫폼 독립적인 언어이다. 마이크로소프트사는 .NET 플랫폼을 개발하면서 자바 언어에 대응하기 위해 C# 프로그래밍 언어를 만들었다. C#은 C/C++의 강력함과 자바나 비주얼 베이직의 높은 생산성을 매우 효과적으로 결합한 프로그래밍 언어이다. C#은 컴파일 과정을 거치면 자바의 바이트코드와 같은 중간 언어인 MSIL 코드를 갖는 파일을 출력하게 되는데, C# 이외에도 모든 .NET 언어들은 컴파일과정을 거치면서 MSIL 코드를 얻기 때문에 여러 .NET 언어들을 이용해서 하나의 응용 프로그램을 만들 수 있다. 본 논문에서는 임베디드 시스템에서의 적용을 위해 C#을 컴파일 하여 나오는 중간 언어 형태인 MSIL 코드를 자바의 중간 언어의 한 종류인 Oolong 코드로 변환해 줌으로서 C#으로 구현된 프로그램이 자바 플랫폼에서 JVM에 의해 실행되도록 하는 중간 언어 번역기를 구현하였다.

1. 서론

썬 마이크로시스템즈사의 자바는 제임스 고슬링(James Gosling)에 의해 고안된 언어로서 운영체제 및 하드웨어 플랫폼에 독립적인 차세대 언어로 최근에 가장 널리 사용하는 범용 프로그래밍 언어 중 하나이다. 자바는 컴파일러에 의해 각 플랫폼에 독립적인 중간 코드 형태의 바이트 코드로 변환되며 바이트 코드는 클래스 파일로부터 추출된다. 하지만 바이너리 코드로 되어있는 클래스 파일에서 바이트 코드를 읽는다는 것은 상당히 어렵고 복잡한 일이다. 반면에 클래스 파일(*.class)로부터 Oolong 역어셈블러(Gnoloo.class)를 이용하여 추출하는 또 다른 형태의 자바 중간 언어인 Oolong 코드로 작성된 파일은 클래스 파일 내에 있는 바이트 코드와 유사한 형태의 실제 프로그램 로직 부분을 텍스트 형식으로 저장하므로 프로그래머 입장에서 좀 더 쉽게 접근할 수 있고 코드의 이해와 프로그램의 작성 및 수정을 용이하게 한다[2,5,8].

한편, 마이크로소프트사의 .NET 플랫폼은 프로그래머들의 요구를 충족시키고 썬사의 JVM 환경에 대응하기 위해서 개발된 플랫폼으로 자바 언어에 대응하기 위해 C# 언어를 새로이 개발하였다. C# 프로그래밍에서는 C/C++에서와 같이 직접적으로 포인터를 조작할 필요가 없다. C# 프로그래밍에서는 자바의 가비지 콜렉션과 같이 메모리 관리가 자동으로 되며, 비주얼 베이직에서처럼 클래스 프로퍼티라는 개념이 있고, C++ 처럼 클래스에서 연산자를 오버로드할 수도 있다. C# 은 자바처럼 문법적으로 깨끗하고, 비주얼 베이직처럼 쉽고, C++ 처럼 유연하고 강력한 언어이다. 또한 C# 은 다른 .NET 언어인 비주얼 베이직 닷넷이나 비주얼 C++ 닷넷 등의 언어와

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



[그림 1] MSIL 코드 추출과정

공동으로 하나의 응용 프로그램을 개발할 수 있다. 그 이유는 닷넷 플랫폼에서의 모든 언어는 MSIL 코드를 중간 언어로 생성하기 때문이다. 이렇게 해서 생긴 MSIL 코드는 하드웨어에 독립적으로 특정 하드웨어 내에서 그 하드웨어에 맞는 .NET 플랫폼 환경에서의 런타임 엔진에 의해 실행이 된다[1,3,4].

자바 언어로 작성된 프로그램은 JVM 플랫폼에서 실행이 되지만 닷넷 플랫폼에서 실행이 안되고, C#으로 작성된 프로그램은 반대로 .NET 플랫폼에서 실행이 되지만 JVM 플랫폼에서 실행이 되지 않는다. 이런 이유로 본 논문에서는 임베디드 시스템에서의 적용을 위해 C#을 컴파일 하여 생성된 MSIL 코드를 Oolong 코드로 변환하여 .NET 플랫폼에서 구현된 프로그램이 JVM 환경에서 실행되도록 하는 MSIL-to-Oolong 번역기 시스템을 구현하였다.

2. MSIL 코드

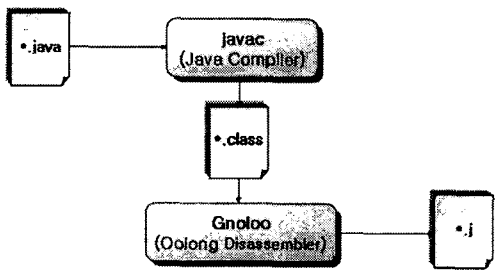
MSIL(Microsoft Intermediate Language) [1,3,7]은 C#을 포함한 .NET 언어들의 중간언어로 .NET 언어로 작성된 소스 코드가 컴파일되면 MSIL로 작성된 코드가 생성된다. 또한 MSIL은 오퍼런드 스택을 이용하는 스택 기반의 명령어 집합으로 두 가지 큰 특징을 가지는데, 첫 째, 언어 상호 운용성(Language Interoperability)이다. C#을 포함한 모든 .NET 언어로 작성된 소스 코드는 컴파일러에 의해 표준화된 MSIL 코드로 변환되는데, MSIL 코드는 어떤 .NET 언어로부터 만들어지던지 동일한 형태를 가진다. 따라서 서로 다른 언어로부터 만들어진 IL 코드를 함께 붙여서 어플리케이션을 구축할 수 있다. 둘째, 플랫폼 독립성이다. 단, 아직까지는 윈도우즈용만 개발되어 있고 향후에 개발되어질 계획이다[4,6,7,11].

본 논문에서 개발한 번역기 시스템에 입력으로 들어갈 MSIL 코드는 컴파일러에 의해 생성된 *.exe나 *.dll에 포함되어 있으므로 MSIL 코드를 추출하는 과정이 필요하다. [그림 1]은 .NET 플랫폼의 어셈블리인 MSIL 코드를 추출해내는 과정을 나타낸 것이다.

3. Oolong 코드

JVM(Java Virtual Machine)은 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 중간 언어인 Oolong 코드는 읽고 쓰기가 클래스 파일 형식에 비해서 훨씬 쉽다. 이 Oolong 코드는 존 메이어(John Meyer)의 Jasmin 언어를 기반으로 만들어졌으며 프로그래머가 바이트 코드 수준에서 프로그램을 작성할 수 있도록 설계되어 있다[2,5,6,12].

클래스 파일로부터 Oolong 코드를 추출해내는 과정은 [그림2]와 같다.

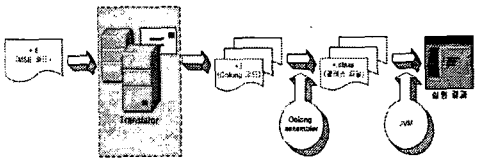


[그림 2] Oolong 코드 추출과정

4. 중간 언어 번역기 시스템 개요

MSIL 코드의 Oolong로의 중간 언어 번역기 시스템은 MSIL 코드 추출과정에 의해서 생성된 *.il 파일을 입력으로 받아서 자바 플랫폼에서 실행될 수 있도록 Oolong 코드 즉 *.j를 생성한다. [그림 3]은 MSIL 파일이 번역기 시스템에 입력으로 들어가 출력으로 Oolong 코드가 나오면 그것을 다시 Oolong 어셈블러와 JVM(Java Virtual Machine)을 이용해 실행이 되는 과정을 나타낸 것이다.

역어셈블러(ildasm.exe)를 통해 추출된 MSIL 코드를 담고 있는 *.il 파일은 번역기 시스템에 의해 Oolong 코드를 담고있는 *.j 파일로 바뀐다. 이 파일이 다시 Oolong 어셈블러를 거치면 클래스 파일이

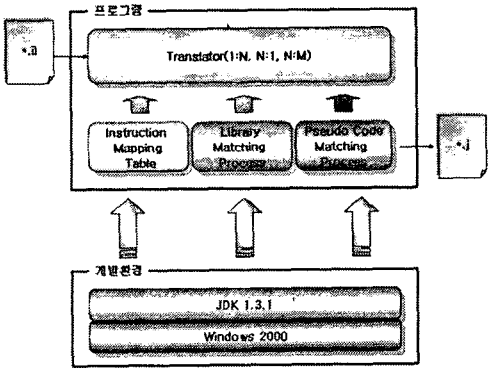


[그림 3] MSIL 코드의 번역과 실행과정

생성되는데, 이것을 JVM으로 수행하면 .NET 언어로 작성된 프로그램의 실행 결과와 똑같은 결과를 얻을 수 있다.

5. 중간 언어 번역기 시스템의 구조 및 구현

본 번역기 시스템은 윈도우즈 2000 환경 하에서 JDK 1.3.1을 이용하여 구현하였으며, 크게 명령어 매핑 테이블, 라이브러리 매칭 프로세스, 그리고 의사 코드(pseudo code) 매칭 프로세스로 구성된다. [그림 4]는 시스템의 구조를 나타낸 것이다.



[그림 4] 번역기 시스템 구성도

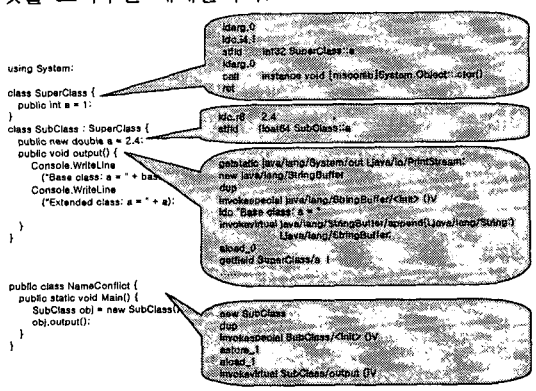
번역 시스템이 MSIL 코드와 메타 데이터로 구성된 *.il 파일을 입력으로 받으면 일단 메타 데이터는 번역과정에서 필요가 없으므로 무시하고 MSIL 코드를 번역하게 된다. 번역기는 매핑 테이블을 이용하여 명령어 매핑 부분이 MSIL 코드와 Oolong 코드 간에 서로 기능적으로 동일한 부분이 될 수 있도록 변환을 하고 MSIL 코드에 포함되어 있는 라이브러리 함수들이 자바의 라이브러리 함수와 대응될 수 있도록 매크로 처리를 하며, MSIL 코드가 가지고 있는 의사 코드도 Oolong 코드가 가지고 있는 의사 코드와 동일한 기능을 수행 할 수 있도록 변환을 한다. 이렇게 함으로 해서, 입력 코드인 MSIL 코드의 기능과 출력 코드인 Oolong 코드의 기능을 같게 하고, 결과적으로 MSIL 코드와 의미적으로 동등한 Oolong 코드를 생성함으로써 .NET 언어를 사용하여 작성된 프로그램이 자바 플랫폼에서 실행 가능하게 된다.

[표 1]은 MSIL 코드와 Oolong 코드의 명령어를 비교 할 수 있는 명령어 매핑 테이블이다.

MSIL	Oolong	Description
and	iand land :	Bitwise AND
or	ior lor :	Bitwise OR
:	:	:
beq	ifeq :	Goto if equal
brnull	ifnull	Goto if null
:	:	:
conv.i4	f2i :	Convert to an Integer value
:	:	:
ldloc	iload :	load a value
:	:	:

[표1] 명령어 매핑 테이블

다음은 본 번역기가 예외처리 부분을 번역하는 것을 보여주는 예제들이다.



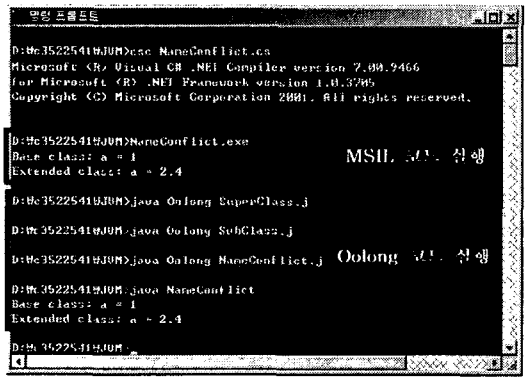
[예제3] C#소스와 추출된 MSIL 코드

```
.class super SuperClass
.super java/lang/Object

.field public  a I

.method public <init> ()V
...
return
.end method //SuperClass.j
```

```
.class super SubClass
.super SuperClass

.field public  a D
.method public output ()V
getstatic java/lang/System/out Ljava/io/PrintStream;
new java/lang/StringBuffer
dup
invokespecial java/lang/StringBuffer/<init> ()V
ldc "Base class: a = "
invokevirtual
java/lang/StringBuffer/append(Ljava/lang/String;)
        Ljava/lang/StringBuffer;
aload_0
getfield SuperClass/a  I
invokevirtual
java/lang/StringBuffer/append(I)Ljava/lang/StringBuffer;
invokevirtual java/lang/StringBuffer/toString
()Ljava/lang/String;
invokevirtual java/io/PrintStream/println
(Ljava/lang/String;)V
...
return
.end method

.method public <init> ()V
...
return
.end method //SubClass.j
```

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

.method public static main ([Ljava/lang/String;)V
new SubClass
dup
invokespecial SubClass/<init> ()V
astore_1
aload_1
invokevirtual SubClass/output ()V
return
.end method
... //NameConflict.j
```

[예제4] 번역기를 통해 생성된 Oolong 코드



[그림6] 실행 화면

6. 결론

본 논문에서는 MSIL 코드를 Oolong 코드로 번역하여 자바 플랫폼에서 실행할 수 있는 번역기 시스템을 구현하였다. 본 번역기는 명령어 매핑 테이블, 라이브러리 매칭 프로세스 등을 이용하여 구현하였으며, 매크로 변환기법을 사용하였다.

앞으로 .NET 플랫폼 프로그래밍 언어에서 지원하는 기능 및 모듈들을 자바 플랫폼 환경에서도 동일하게 사용할 수 있도록 번역기를 확장하고, MSIL 코드와 Oolong 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", APRESS, 2001

[2] Bill Venners, "Inside the JAVA Virtual Machine Second Edition", McGraw-Hill, Dec. 1998

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

[4] Eric Gunnerson, "A Programmer's Introduction to C#", APRESS, 2001

[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, "The Java™ Programming Language", Addision Wesley", 1996

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

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

[11] Simmon Robinson, "Professional C#", Wrox, 2002

[12] Tim Lindholm·Frank Yellin, "The Java™ Virtual Machine Specification Second Edition, Addision Wesley, 1997