Journal of Korea Multimedia Society Vol. 7, No. 7, July 2004(pp. 976-984)

# JVM 플랫폼에서 .NET 프로그램을 실행하기 위한 MSIL-to-Bytecode 번역기의 설계 및 구현

이양선[†], 황대훈[††], 나승원[†††]


## 요 약

마이크로소프트사는 .NET 플랫폼을 개발하면서 자바 언어에 대응하기 위해 C# 프로그래밍 언어를 만들었다. C#과 같은 .NET 언어로 작성된 프로그램은 컴파일 과정을 거치면서 MSIL 코드를 출력하기 때문에 .NET 플랫폼에서는 실행이 되지만 JVM 플랫폼에서는 실행이 되지 않는다. 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 하드웨어나 운영체제에 상관없이 실행이 가능한 플랫폼 독립적인 언어이다. 본 논문에서는 .NET 언어로 작성된 프로그램을 컴파일 하여 생성된 MSIL 코드를 자바의 중간 언어인 바이트코드 코드로 변환해 줌으로서 .NET 언어로 구현된 프로그램이 .NET 플랫폼 없이 자바의 플랫폼인 JVM에 의해 실행되도록 하는 MSIL-to-Bytecode 번역기를 설계하고 구현하였다. 이와 같은 작업이 프로그래머로 하여금 프로그래밍 언어의 제약 없이 응용 프로그램을 개발할 수 있는 환경을 제공한다.


# Design and Implementation of the MSIL-to-Bytecode Translator to Execute .NET Programs in JVM Platform

Yang-Sun Lee[†], Dae-Hoon Whang[††], Seungwon Na[†††]


## ABSTRACT

C# and .NET platform in Microsoft Corp. has been developed to meet the needs of programmers, and cope with Java and JVM platform of Sun Microsystems. After compiling, a program written in .NET language is converted to MSIL code, and also executed by .NET platform but not in JVM platform. Java, one of the most widely used programming languages recently, is the language invented by James Gosling at Sun Microsystems, which is the next generation language independent of operating systems and hardware platforms. Java source code is compiled into bytecode as intermediate code independent of each platform by compiler, and also executed by JVM. This paper presents the MSIL-to-Bytecode intermediate language translator which enables the execution of the program written in .NET language such as C or C# in JVM(Java Virtual Machine) environment, translating MSIL code produced by compiling .NET program into java bytecode. This work provides an environment for programmers to develop application programs without limitations of programming languages.


**Key words:** MSIL Code(MSIL 코드), Bytecode(바이트코드), Oolong Code(Oolong 코드), .NET Platform(닷넷 플랫폼), JVM Platform(JVM 플랫폼), MSIL-to-Bytecode Translator (MSIL-to-바이트코드 번역기)

※ 교신저자(Corresponding Author) : 이양선, 주소 : 서울시 성북구 정릉동 16-1(136-704), 전화 : 02)940-7292, E-mail : yslee@skuniv.ac.kr
접수일 : 2004년 4월 14일, 완료일 : 2004년 5월 31일
[†] 종신회원, 서경대학교 컴퓨터공학과
[††] 종신회원, 경원대학교 소프트웨어대학
(E-mail : hwangdh@mail.kyungwon.ac.kr)
[†††] SK 텔레콤, 플랫폼 R&D센터
(E-mail : nasw@dgu.ac.kr)
※ 본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0) 지원으로 수행되었음.

## 1. 서 론

마이크로소프트사의 .NET 플랫폼은 프로그래머들의 요구를 충족시키고 썬 마이크로시스템즈사의 JVM 환경에 대응하기 위해서 개발된 플랫폼으로 자바 언어에 대응하기 위해 C# 언어를 새로이 개발하였다. C# 은 자바처럼 문법적으로 깨끗하고, 비주얼 베이직처럼 쉽고, C++ 처럼 유연하고 강력한 언어이다. 또한, C# 은 다른 .NET 언어인 비주얼 베이직 .NET이나 비주얼 C++ .NET 등의 언어와 함께 하나의 응용 프로그램을 구성할 수 있다. 그 이유는 .NET 플랫폼에서의 모든 언어는 MSIL(Microsoft Intermediate Language) 코드를 중간 언어로 생성하기 때문이다. 이렇게 해서 생긴 MSIL 코드는 하드웨어에 독립적으로 특정 하드웨어 내에서 그 하드웨어에 맞는 .NET 플랫폼 환경에서의 런 타임 엔진에 의해 실행이 된다[1,3].

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 바이트코드(Bytecode)로 변환되며 바이트코드는 클래스 파일로부터 추출된다. 하지만 바이너리 코드로 되어있는 클래스 파일에서 바이트코드를 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 바이트코드와 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[2,4,7].

자바 언어로 작성된 프로그램은 JVM 플랫폼에서 실행이 되지만 .NET 플랫폼에서 실행이 안되고, .NET 언어로 작성된 프로그램은 반대로 .NET 플랫폼에서 실행이 되지만 JVM 플랫폼에서 실행이 되지 않는다. 이런 이유로 본 논문에서는 .NET 언어로 작성된 프로그램을 컴파일 하여 생성된 MSIL 코드를 바이트코드로 변환하여 .NET 플랫폼에서 구현된 프로그램이 JVM 환경에서 실행되도록 하는 MSIL-to-Bytecode 번역기 시스템을 구현하였다. 이와 같은 노력이 프로그래머가 응용 프로그램을 개발할 때 언어의 제약을 받지 않고 프로그램을 개발할 수 있는 환경을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 기존의 번역기의 종류와 특징을 비교하였고, 중간언어인 MSIL 코드와 바이트코드의 특징을 기술하였으며, 코드생성 방법과 코드변환 방법을 제시하였다. 3장에서는 MSIL 코드를 바이트코드로 번역하기 위한 시스템의 개요와 번역기 시스템의 구조 및 구현 내용을 기술하였다. 4장에서는 각 구문구조에 대한 번역과정과 실험결과 및 분석내용을 제시하였다. 끝으로 5장에서는 본 연구에 대한 요약 및 제언을 하면서 결론을 맺는다.

## 2. 관련연구

### 2.1 번역기

하나의 프로그래밍언어로 작성된 프로그램을 운영체제나 아키텍처와 같은 플랫폼이 다른 환경에서 실행하기 위한 번역기(Translator)에 관한 연구로는 자바 응용 프로그램의 실행속도를 개선하기 위해 바이트코드로 부터 목적기계(Target Machine)의 네이티브 코드(native code)를 생성하는 중간코드 변환기(Intermediate Language Translator)가 있다. 그러나, 실행속도의 개선만을 염두해 두고 개발되었기 때문에 지나치게 목적기계에 의존적이고 다른 컴퓨팅 환경에서 사용할 수 없는 단점을 갖는다[15-18].

한편, 마이크로소프트는 원시언어(Source Language) 레벨에서 자바 프로그램을 C# 프로그램으로 변환하는 JLCA(Java Language Conversion Assistant) 번역기[20]를 개발하였다. 또한, 핼시온 소프트(Halcyon Soft)는 .NET의 중간언어인 MSIL 코드를 자바 소스 프로그램으로 번역하는 iNET 번역기[20]을 개발하였으며, 역으로 리모트 소프트(Remote Soft)는 자바 소스 프로그램을 .NET의 MSIL 코드로 변환하는 Java.NET 번역기[21]를 개발하였다. JCLA나 Java.NET과 같이 원시 프로그램을 또 다른 원시 프로그램으로 번역하거나 중간언어로 번역하는 것은 소스 파일을 가지고 있어야만 번역하여 목적기계 환경에서 실행시킬 수 있기 때문에 프로그램 소스의 알고리즘에 대한 보안을 유지할 수 없다. 또한, iNET

과 같이 중간언어를 원시 프로그램으로 번역하는 것은 여러 단계의 번역과정을 거치기 때문에 번역시간이 오래 걸리고 번역 성공율이 상대적으로 높지 않은 단점이 있다.

이런 이유로 본 논문에서는 소스 파일에 대한 보안을 유지할 수 있고, 여러 단계의 번역과정을 감소시켜 번역시간을 줄일 수 있으며, 번역 성공율이 상대적으로 높은 중간언어 번역과정을 채택하였다.

### 2.2 MSIL 코드와 Bytecode

#### 2.2.1 MSIL 코드

MSIL(Microsoft Intermediate Language)[1,3,6]은 C#을 포함한 .NET 언어들의 중간언어로 .NET 언어로 작성된 소스 코드가 컴파일되면 MSIL로 구성된 코드가 생성된다. 또한, MSIL은 오퍼런드 스택을 이용하는 스택 기반의 명령어 집합으로 구성된다. MSIL 코드는 어떤 .NET 언어로부터 만들어지든지 동일한 형태를 가지므로 서로 다른 .NET 언어로부터 만들어진 IL 코드와 함께 하나의 응용 프로그램을 구성할 수 있다. 그리고, MSIL 코드는 CLR 인터프리터와 베이스 클래스 라이브러리가 장착된 컴퓨터면 어디에서든지 실행될 수 있다[5,6,8-10].

그림 1은 .NET 플랫폼의 어셈블리인 MSIL 코드를 추출해내는 과정을 나타낸 것이다.

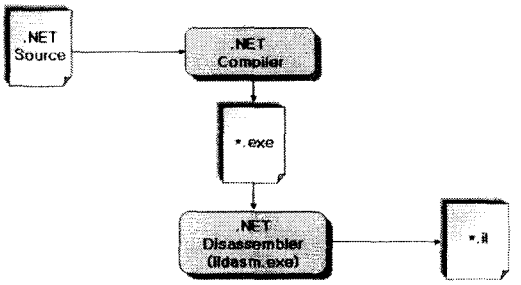



그림 1. MSIL 코드 추출과정

#### 2.2.2 바이트코드(Bytecode)

바이트코드는 자바의 중간언어로 명령어 코드(Opcode)가 한 바이트로 구성되어 있기 때문에 바이트코드라고 부른다. 바이트코드는 목적기계(target machine)에 독립적이기 때문에 플랫폼 독립적으로 JVM(Java Virtual Machine)에 의해 실행된다. 따라서, 새로운 프로그램을 개발 할 때 이식성(portability)에 따른 프로그램의 개발 노력을 줄일 수 있고 소프트웨어의 확산을 증가시킬 수 있다

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 자바 중간언어의 또 다른 형태인 Oolong 코드는 읽고 쓰기가 클래스 파일 형식에 비해서 훨씬 쉽다. Oolong 코드는 존 메이어(John Meyer)의 자스민(Jasmin) 언어를 기반으로 만들어 졌으며 프로그래머가 바이트 코드 수준에서 프로그램을 작성할 수 있도록 설계되어 있다[2,4,7,11,12].

클래스 파일로부터 Oolong 코드를 추출해내는 과정은 그림 2와 같다.

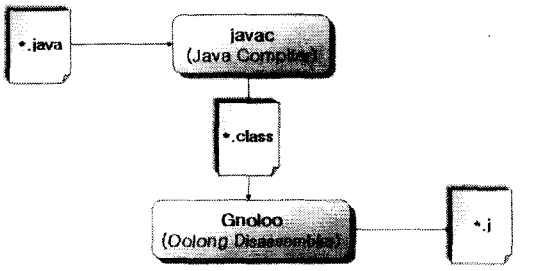



그림 2. Oolong 코드 추출과정

## 3. MSIL-to-Bytecode 번역기 시스템

### 3.1 번역기 시스템의 개요

MSIL-to-Bytecode 중간 언어 번역기 시스템은 실행파일(*.exe)로부터 MSIL 코드 추출과정에 의해서 생성된 *.il 파일을 입력으로 받아서 자바 플랫폼에서 실행될 수 있도록 Oolong 코드 즉, *.j를 생성한다. 그림 3은 MSIL 파일이 번역기 시스템에 입력으로 들어가 출력으로 Oolong 코드가 생성되면 그것을 다시 Oolong 어셈블러를 통해 클래스 파일을 생성하여 JVM을 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 생성되는데, 이것을 JVM으로 실행하면 .NET 언어

로 작성된 프로그램의 실행 결과와 같은 결과를 얻을 수 있다.

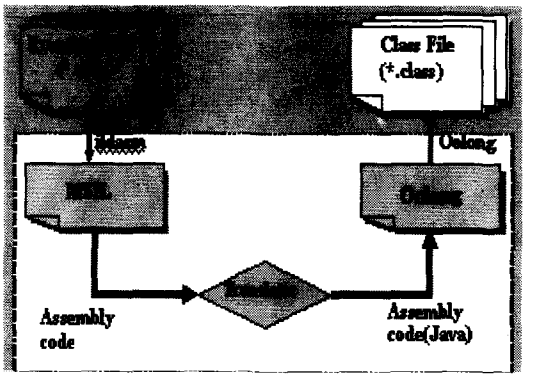



그림 3. MSIL 코드의 Oolong 코드 번역과정

### 3.2 번역기 시스템의 구조

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

번역기 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

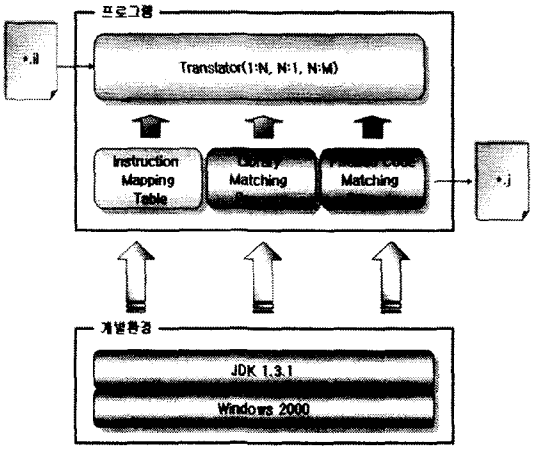



그림 4. 번역기 시스템 구성도

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. 변환과정에서 MSIL 코드와 Oolong 코드가 1:1로 매핑이 되지 않는 부분은 1:M이나 N:1 또는 N:M으로 매핑 코드를 만들어 의미적으로 동등한 코드가 되게 번역을 하였다. 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.

### 3.3 번역기 시스템의 구현

#### 3.3.1 번역기 시스템의 개발환경

번역기 시스템의 개발환경과 사용된 소프트웨어는 다음과 같다.

JDK 1.3.1
Java Disassemble Utility
    D-java Decafe pro
Visual Studio .NET (Version 7.0)
IE 6.0
Windows Component Update
    bootstrap.msi
    mso9.msi
    msxml3.msi
Microsoft Data Access Components
Microsoft .NET SDK(W2K)
O/S - Windows 2000

그림 5. 번역기 시스템의 개발환경

#### 3.3.2 데이터 타입 매핑 테이블

표 1은 번역과정에서 MSIL 코드와 Oolong 코드의 데이터 타입을 비교하기 위해 만든 데이터 타입 매핑 테이블(data type mapping table)이다. 번역기 시스템은 표 1를 기반으로 구현되었다.

#### 3.3.3 명령어 매핑 테이블

표 2는 MSIL 코드와 Oolong 코드의 명령어를 비교하여 나타낸 명령어 매핑 테이블(instruction map-

표 1. 데이터 타입 매핑 테이블

| MSIL | Oolong | Description |
|---|---|---|
| bool | bool | true/false value |
| char | char | unicode 16-bit char. |
| object | oject | object or boxed value type |
| string | String | unicode string |
| float32 | float | IEC 60559:1989 32-bit float |
| float64 | double | IEC 60559:1989 64-bit float |
| int8 | byte | signed 8-bit integer |
| unsigned int8 | | unsigned 8-bit integer |
| int16 | short | unsigned 16-bit integer |
| unsigned int16 | | unsSigned 16-bit integer |
| int32 | int | signed 32-bit integer |
| unsigned int32 | | unsigned 32-bit integer |
| int64 | long | signed 64-bit integer |
| unsigned int64 | | unsigned 64-bit integer |
| ... | ... | ... |

표 2. 명령어 매핑 테이블

| offset | MSIL | Oolong | Description |
|---|---|---|---|
| 0x00 | nop | nop | No Operation |
| : | : | : | : |
| 0x06 | ldloc.0 | iload_1 | Load First local variable onto the stack |
| 0x07 | ldloc.1 | iload_2 | Load Second local variable onto the stack |
| : | : | : | : |
| 0x0a | stloc.0 | istore_1 | Pop value from stack into first local variable |
| 0x0b | stloc.1 | istore_2 | Pop value from stack into second local variable |
| : | : | : | : |
| 0x17 | ldc.i4.1 | iconst_1 | Push 1 onto the stack as i4 |
| 0x18 | ldc.i4.2 | iconst_2 | Push 2 onto the stack as i4 |
| : | : | : | : |
| 0x2a | ret | return | Return from method |
| : | : | : | : |
| 0x58 | add | iadd | Add values, returning a new value |
| 0x59 | sub | isub | Subtract values, returning a new value |
| 0x5a | mul | imul | Multiply values, returning a new value |
| 0x5b | div | idiv | Divide values, returning a new value |
| : | : | : | : |

ping table)이다. 번역기 시스템은 표 2를 기반으로 구현되었다.

3.3.4 번역 알고리즘

다음은 MSIL 코드를 Oolong 코드로 번역하기 위한 알고리즘의 개요이다.

알고리즘 1. MSIL-to-Oolong 코드 번역 알고리즘

```
public static void main()
{
  String[] classNames = getClassNames();
    //the names for classes
  numClasses = getNumClasses();
    //the number of classes
  for (int i = 0; i < numClasses; i++)
  {
    if (has an extended class)
       add '.super' and a super class' id
       to the class declaration part;
    if (has an interface)
       add '.interface' and a class' id
       to the class declaration part;
     String strLine;
   while ((strLine =className.readLine()) != EOF)
    {
      if (has(strLine, ".class") == true)
          translate the class declaration part;
      else if (has(strLine, ".method") == true)
          translate the method declaration part;
      else if (is an instruction)
          translate the MSIL instruction
          into Oolong instructions;
    } // translate a MSIL file and make Oolong files
       as many as classes
  }
}
```

MSIL 파일은 한 개의 *.il 파일에 여러 개의 클래스가 정의 되어 있을 수 있는 반면에, Oolong 파일은 한 개의 *.j 파일에 오직 한 개의 클래스만 존재하기 때문에 *.il 파일을 입력으로 받아서 클래스의 개수와 이름을 알아내 그 개수만큼 *.j 파일을 만들어 주어야 한다. 이를 위한 초기화 단계로 *.il 파일을 읽어 들여 각 클래스의 시작부분과 끝부분을 탐색하여 Oolong 코드로 변환하고, 다음으로 마찬가지 방법을 이용하여 *.il 파일을 읽어 들여 각 클래스 내의 메소드들의 시작부분과 끝부분을 탐색하여 Oolong 코드로 변환을 하여준다.

## 4. 실험결과 및 분석

### 4.1 구문의 번역

MSIL 코드의 클래스 선언부에서는 구조 선언부와 멤버 선언부로 나누어져 있지만, Oolong에서는 .class 지시어와 .super 지시어로 이루어져 있다. 따라서, MSIL 코드의 클래스 선언부를 .class 지시어와 .super 지시어로 번역한다. 표 3은 클래스 시작부분

표 3. 클래스 시작부분의 번역

| public class Counter { ... } | |
|---|---|
| MSIL code | Oolong code |
| .class public auto ansi beforefieldinit<br>Counter extends [mscorlib]<br>System.Object<br>{<br>IL_0017: newobj instance void [mscorlib]<br>System.Threading.Thread::.ctor<br>(class[mscorlib] System.<br>Threading.ThreadStart)<br>IL_001c: stloc.1 | .class public super Counter<br>.super java/lang/Object |

의 번역과정을 나타낸 것이다. 클래스의 필드 부분을 MSIL 코드에서 Oolong 코드로 번역 할 때 1 : 1로 매핑이 되기 때문에 형식은 그대로 이용하고 명령어만 Oolong 코드로 번역하면 된다. 표 4는 클래스 필드의 번역과정을 나타낸 것이다.

메소드 선언부는 MSIL코드와 Oolong 코드가 N:1로 매핑이 되므로 필요 없는 부분은 무시하고 필요한 정보만 Oolong 코드로 번역하면 된다. 표 5는 메소드 시작부분의 번역과정을 나타낸 것이다.

배열과 제어문은 MSIL 코드에서 Oolong 코드로 번역 할 때 1:1로 매핑이 되기 때문에 형식은 그대로 이용하고 명령어만 Oolong 코드로 번역하면 된다. 표 6은 배열의 번역과정을 나타낸 것이다.

MSIL 코드에서는 객체를 생성하는 작업을 하지 않고 local 변수에 클래스 객체를 선언하여 사용하고, newobj 명령어를 사용하여 객체를 초기화 한다. 그러나, Oolong 코드에서는 new 명령어와 dup 명령어를 이용하여 객체를 생성하여 복사한 후에 인자에

표 4. 필드의 선언문과 출력문의 번역

| static int i = 3; ...<br>System.Console.WriteLine(i); | |
|---|---|
| MSIL code | Oolong code |
| .field private static int32 I<br>...<br>IL_0000: ldsfld int32 FieldTest::i<br>IL_0005: call void [mscorlib]<br>System.Console::WriteLine(int32) | .field private static i I<br>...<br>IL_0000: getstatic java/lang/System/out<br>Ljava/io/PrintStream<br>getstatic FieldTest/i I<br>IL_0005: invokevirtual<br>java/io/PrintStream/println (I)V |

표 5. 메소드 시작부분의 번역

| public int sum(int a, int b) { ... } | |
|---|---|
| MSIL code | Oolong code |
| .method public hidebysig instance int32<br>sum(int32 a, int32 b) cil managed | .method public sum (II)I |

표 6. 배열의 번역

| string[] s = new string[4]; | |
|---|---|
| MSIL code | Oolong code |
| IL_0000: ldc.i4.4<br>IL_0001: newarr [mscorlib]System.String<br>IL_0006: stloc.0 | IL_0000: iconst_4<br>IL_0001: anewarray java/lang/String<br>IL_0006: astore_1 |

넣는 작업을 한다.

예외처리 부분의 번역과정은 다음과 같다. MSIL 코드에서는 예외를 C# 언어와 같이 .try 지시자와 catch 명령어를 이용해서 문장들을 블록으로 만든다. 반면에, Oolong 코드에서는 예외 처리기를 .catch 지시자를 사용하여 나타낸다. 번역할 때 .try 명령어를 보고서 Oolong 코드에서 .catch 지시자를 사용하여 번역한다.

스레드의 번역과정은 다음과 같다. C#에서는 System.Threading이라는 네임스페이스를 지정해 줌으로서 스레드를 상속받는다. 그리고 ThreadStart 델리게이트를 통해서 스레드로 실행할 메소드를 지정한다.

이에 반해서, 자바에서는 스레드를 만들려는 클래스에 Thread 클래스를 상속 받고, 실행될 메소드로 run() 메소드가 지정된다. 번역할 때는 MSIL 코드의 ThreadStart 델리게이트까지 보고 Oolong 코드의 스레드 객체를 초기화 한다.

### 4.2 확장 클래스 프로그램의 번역과 실행 예제

다음은 C#으로 작성된 확장 클래스 프로그램을 구현한 MSIL-to-Bytecode 번역기를 통해 번역하여 실행하는 예제이다. 프로그램 1은 C# 프로그램를 나타내고, 프로그램 2는 추출된 MSIL 코드와 번역기를 통해 생성된 Oolong 코드를 나타낸다.

```
using System;
class SuperClass {
        public int a = 1;
}
class SubClass : SuperClass {
        public new double a = 2.4;
        public void output() {
              Console.WriteLine("Base class: a = " + base.a);
              Console.WriteLine("Extended class: a = " + a);
        }
        public bool isPositive(int a) {
              if (a >= 0) return true;
              else return false;
        }
        public int sum(int a, int b)
        int j = 0;
        for (int i = a; a <= b; a++) j = j + a;
        return j;
}
public class Counter {
    public static void Main() {
        SubClass obj = new SubClass();
        obj.output();
        Console.WriteLine("result: " + obj.isPositive(5, 9));
        Console.WriteLine("Sum 1 through 10: " + obj.sum(1, 10));
    }
}
```

프로그램 1. C# 프로그램

그림 6은 C# 프로그램을 컴파일한 MSIL 코드와 본 논문에서 구현한 번역기를 통해 이를 번역하여 출력한 Oolong 코드를 각각 .NET 플랫폼과 JVM 플랫폼에서 실행한 결과이다.

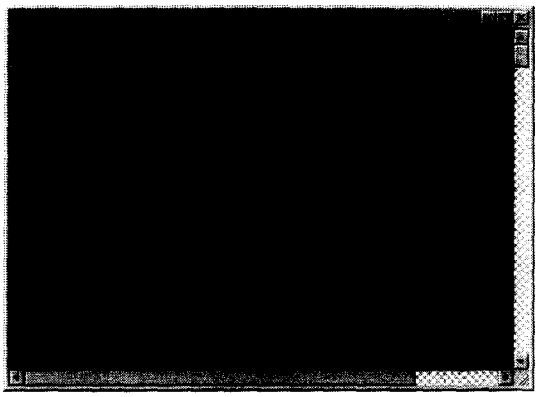


그림 6. MSIL과 번역된 Oolong 코드의 실행 결과

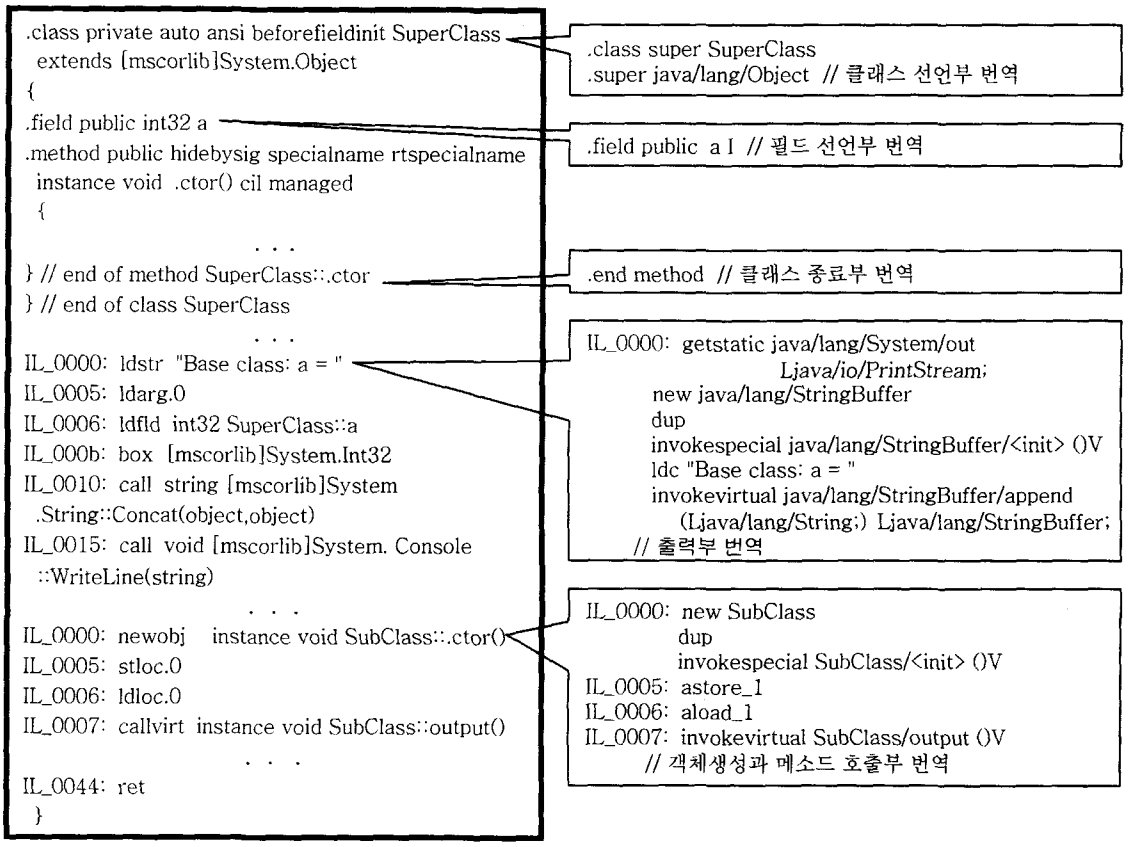
MSIL 코드로 구성된 NameConflict.exe의 실행결과와 번역된 NameConflict.j를 Oolong 어셈블러로 클래스 파일을 만들어 JVM 플랫폼에서 실행한 결과가 같음을 보여주고 있다.

## 5. 결 론

본 논문에서는 MSIL 코드를 바이트코드로 번역하여 .NET 언어로 작성된 프로그램이 .NET 플랫폼에서뿐만 아니라 자바 플랫폼에서 실행될 수 있는 중간언어 번역기 시스템을 구현하였다. 번역기는 명령어 매핑 테이블, 라이브러리 매칭 프로세스 등을 이용하여 구현하였으며, 매크로 변환기법을 사용하였다.

번역기가 MSIL 코드를 바이트코드로 번역하기 위해 .NET 언어로 작성된 프로그램을 컴파일하여 생성된 실행 파일(*.exe)로부터 역어셈블러(ildasm.exe)를 이용하여 MSIL 코드를 추출하고 추출된 MSIL 코드를 자바의 중간언어인 바이트코드로 변환하여 .NET 언어로 구현된 프로그램이 JVM 환경에서도 실행되도록 하였다.

따라서, 자바 프로그래머나 .NET 프로그래머는 JVM 이나 .NET 플랫폼 환경에 관계없이 프로그램을 작성하여 실행시킬 수 있다. 또한, 프로그래머는 응용 프로그램을 개발할 때 언어의 제약을 받지 않고




프로그램 2. 추출된 MSIL 코드와 번역기를 통해 생성된 Oolong 코드

프로그램을 개발할 수 있다.

앞으로 .NET 플랫폼 프로그래밍 언어에서 지원하는 기능 및 모듈들을 자바 플랫폼 환경에서도 동일하게 사용할 수 있도록 번역기를 확장하고, MSIL 코드와 바이트코드를 분석하여 실험을 통해 보다 나은 코드를 낼 수 있는 코드 최적화에 위한 연구를 수행할 예정이다.

## 참 고 문 헌

[1] Andrew Troelsen, C# and the .NET Platform, APRESS, Berkeley, CA., 2001.

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

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

[4] Hoshua Engel, Programming for the Java Virtual Machine, Addison-Wesley, Boston, MA., 1999.

[5] Jeff Prosise, Programming Microsoft .NET, Microsoft Press, Redmond, WA., 2002.

[6] John Gough, Compiling for the .NET Common Language Runtime(CLR), Prentice Hall, Upper Saddle River, NJ., 2002.

[7] Ken Arnold and James Gosling, The Java[tm] Programming Language, Addision Wesley, Boston, MA., 1996.

[8] Microsoft Corporation, Common Language Infrastructure(CLI), Microsoft Corporation, Redmond, WA., 2001.

[9] Serge Lindin, Inside Microsoft .NET IL Assembler, Microsoft Press, Redmond, WA., 2002.

[10] Simmon Robinson, Professional C#, Wrox, Hoboken, NJ., 2002.

[11] Tim Lindholm and Frank Yellin, The Java[tm]

Virtual Machine Specification Second Edition, Addision-Wesley, Boston, MA., 1999.

[12] Troy Downing and John Meyer, Java Virtual Machine, O'REILLY, Sebastopol, CA., 1997.

[13] Susan L. Graham, "Table Driven Code Generation," IEEE Computer, Vol.13, No.8, pp. 25-34, 1980.

[14] M.Ganapathi, C.N.Fisher and J.L.Hennesy, "Retargetable Compiler Code Generation," ACM Computing Sur-veys, Vol.14, No.4, pp. 573-592, 1982.

[15] C.A.Hsieh, M.T.Conte, and T.L.Johnson, "Java Bytecode to Native Code Translation: the Caffeine Prototype and Prelimi-nary Results", Proc. of the IEEE 29th Annual International Symposium on Microarchitecture, Dec 1996.

[16] Harlan McGhan and Mike O'Conner, "PicoJava: A Direct Execution Engine for Java Bytecode", IEEE Computer, pp. 22-30, 1998.

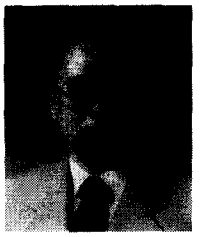
[17] Ronald Veldema, "Jcc, A Native Java Compiler", Vrije Universiteit Amsterdam, 1998.

[18] A.Krall and R.Grafl, "CACAO: A 64 bit Java VM Just-in-Time Compiler", Concurrency: Practice and Experience, 1997. http://www.complang. tuwien.ac.at/~andi

[19] JLCA; Java-Language-to-C# Conversion Assistant, http://www.microsoft.com/korea/press/pressroom/2002/02/02.htm

[20] Halcyon Soft, iNET, http://www.halcyonsoft.com/

[21] Remotesoft, Java.NET, http://www.remotesoft.com/

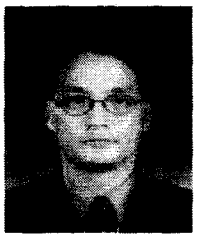


**이 양 선**

1985년 동국대학교 전자계산학과 (공학사)
1987년 동국대학교 대학원 컴퓨터공학과 (공학석사)
1993년 동국대학교 대학원 컴퓨터공학과 (공학박사)
1994년 3월~현재 서경대학교 컴퓨터공학과 교수
2000년 2월~현재 멀티미디어학회 이사
2001년 2월~현재 프로그래밍언어연구회 이사
관심분야 : 프로그래밍 언어, 임베디드 시스템, 모바일 컴퓨팅

**황 대 훈**

1977년 동국대학교 수학과(이학사)
1983년 중앙대학교 대학원 전자계산학과(공학석사)
1991년 중앙대학교 대학원 전자계산학과(공학박사)
1987년 3월~현재 경원대학교 소프트웨어대학 교수
2003년 3월~현재 한국멀티미디어학회 논문지 편집위원장
관심분야 : XML 문서처리, e-Learning, VRML 등 인터넷 응용



**나 승 원**

1993년 단국대학교 농경제학과 (학사)
1996년 단국대학교 대학원 전자정보관리 전공(석사)
2004년 동국대학교 대학원 컴퓨터공학 전공(박사)
1997년~현재 SK텔레콤 플랫폼 R&D센터 선임 연구원
관심분야 : 모바일 컴퓨팅, 이동 에이전트, 유비쿼터스 컴퓨팅, 프로그래밍 언어