

Java Bytecode에서 MSIL로의 번역기

민정현⁰ 오세만
동국대학교 컴퓨터공학과
salvatore@orgio.net smoh@dgu.ac.kr

Translator for Java Bytecode to MSIL

Jeonghyun Min⁰ Seman Oh
Dept. of Computer Engineering, Dongguk University

요 약

자바는 객체지향언어이고, 한번 자바로 작성된 프로그램은 자바가상기계가 있는 모든 곳에서 수정없이 실행될 수 있기 때문에 소프트웨어의 개발과 유지보수에 많은 장점을 가진 언어이다. 이러한 특징으로 인하여 현재 개발되는 제품들이 자바로 개발되는 경우가 많다. 그러나 이는 결국 주 운영체제에서 사용되는 실행파일 이외에 자바의 실행파일만을 위한 자바가상기계가 공존해야 하므로 시스템에 있어서는 이중부담을 안을 수 밖에 없고, 더욱이 아직 대다수의 소프트웨어 개발자들은 주로 C언어나 C++언어를 사용하고 있으며, 최근에는 C#이라는 언어를 사용하고 있다. 결국 자바가 플랫폼에 독립적이라는 장점은 가지고 있지만, 다수의 개발자가 마이크로소프트의 윈도우 운영체제를 사용하고 있다는 점을 감안하면, 커다란 장점만은 될 수 없다.

따라서, 본 논문에서는 자바의 클래스화일을 자바가상기계가 없이도, 마이크로소프트 윈도우의 .NET Framework에서 실행될 수 있는 실행파일로의 변환을 할 수 있는 중간언어인 MSIL(Microsoft Intermediate Language)로 바꿀 수 있는 번역기를 설계 및 구현 하였다. 이를 위한 방법으로는 자바 바이트코드와 MSIL의 어셈블리 형태에서의 명령어 매칭과정을 명령어들의 매핑 테이블을 이용하여 변환하였고, MSIL에서 자바 바이트코드의 함수와 같은 기능을 하는 메소드의 변환을 위하여 매크로변환 기법을 사용하였다.

1. 서 론

자바로 작성된 프로그램은 자바가상기계가 있는 곳에서는 프로그램의 수정과 재컴파일 과정 없이 실행가능하고, 객체 지향을 기반으로 하는 언어이기 때문에 최신 객체지향 기술을 쉽게 적용할 수 있다는 장점을 가진다. 그러나, 최근의 기술 동향을 살펴보면, 네트워크 환경이 점점 빠르게 발전함에 따라, COM을 기반으로 하는 컴포넌트 기반 기술들이 중점을 이루고 있는 것을 볼 수 있고, 또한, 실제로 자바를 사용하는 개발자들과 자바로 구현되는 라이브러리가 시간이 갈수록 늘어남에도 불구하고, C/C++를 사용하는 개발자가 훨씬 많은 것이 현실이다. 또한, 최근 마이크로소프트 사의 .NET Framework에서 제공하고 있는 C# 언어가 상당한 주목을 받고 있는 가운데, 이를 이용한 컴포넌트의 개발이 폭 넓게 적용되어 지고 있다. C# 언어는 마이크로소프트 사의 Visual Basic과 같은 코드작성 편리함과 유지보수의 용이성, 그리고 C++의 유연함과 강력함을 바탕으로 COM+와 웹 서비스를 더욱 쉽고 빠르게 개발할 수 있는 언어이다. 이는 기존의 마이크로소프트사에서 제공하는 컴포넌트를 모두 재사용할 수 있고, 윈도우즈 API를 80% 이상 사용하고

있다. 이와 같은 환경을 제공하는 .NET Framework에서는 개발언어에 상관없이 MSIL로 변환되어지기만 하면, 목적 플랫폼에서 최적화된 실행화일을 만들어 낼 수 있으므로, 자바 바이트코드가 MSIL로 변환되면, 자바가상기계가 없이도 자바의 실행화일을 윈도우의 .NET Framework에서 사용되는 실행파일로 만들어서 실행할 수 있다.

이를 위하여 본 논문에서는 자바의 중간 언어인 바이트코드를 .NET Framework의 중간 언어인 MSIL로 바꿀 수 있는 번역기를 설계 및 구현하여, 궁극적으로 자바가상기계가 없이 자바의 실행파일인 클래스화일을 .NET Framework 환경 하에서 실행시킬 수 있도록 하여, 기존의 자바 개발자와 C언어, C++언어, C#언어 등의 개발언어에 구애받지 않고 프로그래밍을 할 수 있도록 하는데 목적을 두고 있다.

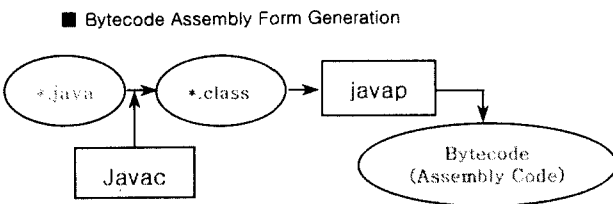
2. 관련 연구

2.1 Bytecode 개요

바이트코드는 자바가상기계의 기계언어형태이며, 클래스 파일을 로드할 때, 클래스 내에서 각각의 메소드에 대하여 스트림 형태로 얻어지며, 이 때의 스트림은 8-bit의 바이트로 구성된 이진스트림 형태이다.

본 연구는 한국과학재단의 특정기초연구(과제번호: 1999-1-30 300-3) 지원에 의한 것임.

또한, Bytecode 는 기본적으로 스택 지향 구조를 가지고 있고, 처음부터 인터프리터를 목적으로 설계 되었으며(JVM 에 의해 인터프리트 되든지, 클래스 로딩시 컴파일된다), 한편으로 Bytecode 는 프로그램의 기능추가 및 성능향상을 위하여 Bytecode 의 변경을 해야 하고, 클래스 및 객체는 반드시 자바언어로 구성하여야 한다는 제약성도 가지고 있다. 다음은 자바 원시코드에서 Bytecode 어셈블리 파일이 얻어지는 과정을 도식화한 것이다. Javac에 의해서 얻어진 클래스 파일을 번역기의 입력인 어셈블리 형태의 파일로 얻어내기 위하여 JDK1.3의 javap -c Option 을 사용한다. javap는 클래스 파일안에 있는 여러 정보들 중 에서 원시코드에 대한 내용들만을 추출하여 주는 기능을 수행한다.

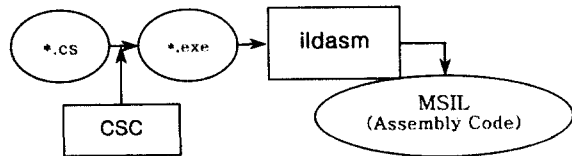


[그림1] Bytecode 시스템 구성도

2.2 MSIL 개요

MSIL은Microsoft Intermediate Language의 약자로 .NET Framework의 중간 언어이며, 컴파일러나 다른 도구에 의해 원시코드로부터 쉽게 생성될 수 있도록 설계된 스택 기반의 명령어 집합이다. MSIL 명령어 집합은 스택상에서 데이터 타입들을 탐색하고, 직접 인터프리트된다. 또한, MSIL은 하드웨어 및 플랫폼에 독립적이고, 처음부터 JIT 를 목적으로 설계 되었으며, 자바언어와는 다르게 처음부터 언어독립적으로 설계되어, 포괄적인 프로그래밍(Generic Programming)을 목적으로 하기 때문에 프로그램의 기능 및 구조의 변화에 잘 적용하는 언어이다. 다음은 번역기의 결과 형태를 알아보기 위하여 역으로 MSIL의 코드를 다음의 방법으로 추출하여 변환될 부분의 형태를 예측하도록 한다. 이 부분에서 생성되는 코드는 자바언어의 클래스 파일에서와 같이 다양한 정보들을 포함하고 있으며 본 논문에서는 프로그램의 코드 부분에 대한 정보들만을 기초로 정보를 추출하기로 한다.

■ MSIL Assembly Form Generation



[그림2] MSIL 시스템 구성도

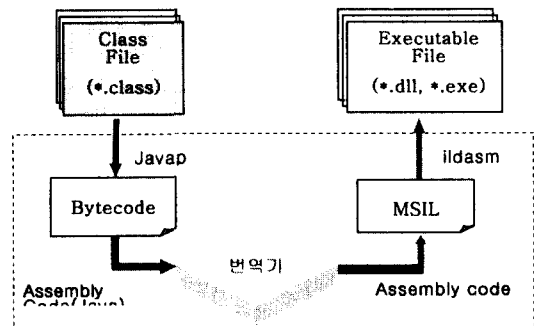
3. 번역기의 설계

3.1 번역기의 기본 개념

이 절은 번역기의 설계에 대한 부분으로 매크로 변환 기법을 이용하여 Bytecode를 MSIL로 변환하는 코드 번역기를 설계하도록 한다. 이는 아래 [그림4]에서와 같이 번역기를 이루는 주요 부분인 명령어 매핑 부분과 함수형태 구조변환 부분이 Bytecode와 MSIL간에 기능적으로 동일한 부분이 될 수 있도록 코드간의 매칭을 매핑테이블을 이용하여 변환할 수 있다는 가정 하에서 이루어진 것이다. 또한, 이를 위하여 실제 구현에서는 매크로 변환 기법을 이용하여 코드간에 각각에 해당하는 테이블을 참조하기로 한다. 따라서, 그 결과 코드와 입력 코드의 기능을 같게 하고, 궁극적으로 전체 프로그램의 기능을 같게 하여, 자바언어로 짜여진 프로그램을 .NET Framework에서 사용 가능하도록 구현하는 것을 목적으로 한다.

3.2 번역기의 구조 및 동작원리

다음의 [그림 3]은 번역기의 시스템 구성도로 전체적으로 Bytecode에서 MSIL까지 변환하는 과정을 도식화한 것이다. 먼저, 변환 하고자 하는 프로그램의 소스파일인 *.java를 Javac에 의해서 *.class 로 바꾸고 이를 어셈블리 형태로 얻어내기 위해서 Javap -c 옵션을 이용하여 코드 부분의 어셈블리 형태를 얻어 낸다. 본문에서 해야 할 것은 앞서 만들어 낸 코드부분의 어셈블리 형태를 입력으로 받아서 구현하는 번역기에 의해 MSIL 코드 부분의 어셈블리 형태로 바꾸는 것을 구체적으로 설계 및 구현하는 것이다.



[그림3] IL Translator 시스템 구성도

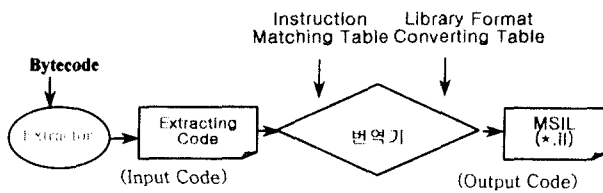
3.3 Translator 제공 기능

.NET Framework에서 자바 Bytecode를 사용하기 위해서 번역기는 다음과 같은 기능을 제공한다.

- Instruction Code Translation
- Library Format Converting
- Code Pattern Customizing

4. 번역기의 구현

자바 Bytecode를 .NET Framework의 중간 언어인 MSIL로 바꾸는 번역기를 개발하기 위해서는 구체적으로 아래 [그림 4]의 단계로 세부 구현 사항을 진행해 나가기로 한다. 이를 위해서 먼저, 생성된 클래스 파일을 어셈블리 형태인 Bytecode로 얻어야 하고, 이 부분 중에서 소스 코드 부분만의 정보들을 추출해야 하는데, 그러기 위해서 추출기 역할을 하는 javap -c를 이용한다. 다음 단계로는 분리된 코드를 번역기의 입력으로 하여 명령어 매핑 테이블과 함수형태 구조변환 테이블을 이용하여 번역기의 결과 코드인 MSIL형태로 구성하도록 한다. 아래의 그림에서 보면, Bytecode에서 MSIL로의 결과를 얻어내기 위해, Extracting과정을 통해서 코드를 추출하고, 두 가지의 매핑 및 변환과정을 통해서 결과가 나오는 형태를 취한다.



[그림4] 번역기의 내부 시스템 구성도

4.1 상세 구현

위의 [그림 4]에서 Bytecode의 CLR Data의 명령어를 MSIL의 출력형태로 얻어 내기 위해서 본 논문에서는 Instruction Mapping Table을 작성하여 입력 코드인 Bytecode의 해당 라인에 해당하는 명령어를 테이블 참조를 통하여 MSIL 명령어 형태로 만들어 주는 기능을 구현한다. 다음의 [표1-1]은 간단한 예제에 대한 명령어 매핑 테이블을 도시한 것이다.

Ref_num	Bytecode Instruction	MSIL Instruction
000	nop	Nop
:	:	:
004	iconst_1	ldc.i4.1
:	:	:
027	iload_1	ldloc.0
:	:	:
070	istore_1	stloc.0
:	:	:
094	iadd	add
:	:	:
098	isub	sub
:	:	:
102	imul	mul
:	:	:
108	idiv	div
:	:	:
177	return	ret
:	:	:

[표 1-1]. Instruction Mapping Table

5. 실험결과 및 분석

Bytecode	MSIL
iconst_1	ldc.i4.1
istore_1	stloc.0
iconst_2	ldc.i4.2
istore_2	stloc.1
getstatic #2 <Field java.lang.Object.>	instance void [mscorlib]System.Object::.ctor()
iload_1	ldloc.0
iload_2	ldloc.1
iadd	add
invokevirtual #3 <Method void print(int)>	call void [mscorlib]System.Console::Write(int32)
return	ret

[표 1-2]. Bytecode to MSIL 변환 결과

6. 결론 및 향후 연구방향

본 논문에서는 .NET Framework의 환경 하에서 자바 바이트코드를 사용할 수 있게 하기 위해서 .NET Framework의 중간 언어인 MSIL과 자바가상기계의 중간언어인 바이트코드를 비교 분석하여 바이트코드를 MSIL로 변환할 수 있는 번역기를 설계하고 구현하였다. 현재 연구의 진행정도는 Bytecode를 MSIL로 변환하는 과정에서 어셈블리 명령어 매핑 테이블과 함수형태 구조 변환 테이블을 구성하였고, 전체적인 코드 형태를 대형 프로그램에 대해서도 변환할 수 있도록 최적화 연구를 수행하고 있다.

향후 연구방향은 자바 언어에서 지원하는 기능 및 모듈들을 MSIL에서도 동일하게 사용할 수 있는 개발 환경을 구축할 뿐만 아니라, MSIL에서 사용할 수 있는 언어적인 장점을 자바에서도 사용 가능하게 만드는 것을 목표로 하고 있다. 또한, 이것이 단순히 중간언어의 일대일 변환에서 그치는 것이 아닌, 코드 형태의 분석에 의한 통계적인 코드 최적화로 이어지기 위한 연구를 지속적으로 해 나갈 것이다.

참고 문헌

- [1]. Bill Venners, Bytecode basics, A first look at the bytecodes of the Java virtual machine
- [2]. Eric Gunnerson, A Programmer's Introduction to C#, Apress™, 2000
- [3]. Java Bytecode Assembler, <http://www.eg.bucknell.edu/~cs360/java-assembler/>
- [4]. Ken Arnold & James Gosling, The Java™ Programming Language, 1997
- [5]. Microsoft.NET, C# & ASP.NET Programming, .net press
- [6]. O'REILLY, Jon Meyer & Troy Downing, Java Virtual Machine, Mar. 1997
- [7]. Wrox, Simon Robinson & Ollie Cornes & Jay Glynn ... , Professional C#, 2001
- [8]. Microsoft press, Tom Archer, Inside C#, 2001