Journal of Korea Multimedia Society Vol. 8, No. 9, September 2005(pp. 1282-1291)

임베디드 시스템을 위한 가상기계의 설계 및 구현

오세만[†], 이양선[††], 고광만[†††]

요 약

본 논문에서는 모바일 디바이스(휴대폰, PDA), 셋톱박스, 디지털 TV 등의 임베디드 시스템에 탑재되어 수많은 동적 애플리케이션을 다운로드하여 실행할 수 있는 임베디드 가상기계(Embedded Virtual Machine)를 설계하고 구현하였다. 이를 위해 표준 중간 언어인 SIL(Standard Intermediate Language)을 정의하고, 자바와 .NET 언어로 구현된 프로그램이 EVM에서 실행될 수 있도록 SIL 코드로 번역해 주는 MSIL-to-SIL 번역기와 Bytecode-to-SIL 번역기를 구현하였다. 또한, SIL 코드를 EVM의 실행파일인 *.evm으로 변환해주는 어셈블러인 EFF 빌더를 개발하였으며, EFF 빌더가 생성한 *.evm 파일을 입력으로 받아 실행하는 가상기계(EVM)를 구현하였다. 본 연구에서 구현한 가상기계는 플랫폼이 변경되더라도 컨텐츠의 수정없이 실행할 수 있는 소프트웨어 기술이다. 실제로, 제안된 가상기계는 기존에 존재하는 가상기계들의 표준화 모델로 사용될 수 있을 뿐만 아니라 모바일 디바이스, 디지털 TV, 셋톱박스 등과 같은 임베디드 시스템에 내장되어 응용프로그램을 효율적으로 실행시켜 줄 수 있다.

Design and Implementation of a Virtual Machine for Embedded Systems

SeMan Oh[†], YangSun Lee[††], KwangMan Ko[†††]

ABSTRACT

This paper presents the EVM(Embedded Virtual Machine) which enables the execution of dynamic applications loaded in the embedded systems such as Mobile Devices(mobile phone, PDA), Set-Top Box, and Digital TV using downloading techniques. To accomplish this goal, we defined a SIL(Standard Intermediate Language) code, and implemented a Bytecode-to-SIL translator which enables the execution of programs written in java language in the EVM platform without JVM, and a MSIL-to-SIL translator which enables for programs written in .NET language to be executed in the EVM platform without .NET platform. Also, we developed a EFF(Executable File Format) builder as an assembler which translates SIL codes into an executable file, *.evm, and implemented the EVM which reads the *.evm file and executes it. The virtual machine for embedded systems developed in this paper is the software technologies that enable the execution of applications or contents without changes to contents even when the platforms change. In fact, the virtual machine suggested here is not only usable as a standard model for existing virtual machines but also aid in more efficient execution of applications loaded in the embedded systems such as Mobile Devices, Digital TV, and Set-Top Box.

Key words: EVM(임베디드 가상기계), SIL(표준중간언어), EFF 빌더(실행파일포맷 생성기), Intermediate Language Translator(중간언어 번역기)

※ 교신저자(Corresponding Author) : 이양선, 주소 : 서울시 성북구 정릉4동(136-704), 전화 : 02)940-7292, FAX : 02)919-0345, E-mail : yslee@skuniv.ac.kr
접수일 : 2005년 4월 7일, 완료일 : 2005년 4월 26일
[†] 정회원, 동국대학교 컴퓨터공학과 교수
(E-mail : smoh@dgu.ac.kr)
[††] 종신회원, 서경대학교 컴퓨터공학과 교수
[†††] 정회원, 상지대학교 컴퓨터정보공학부 조교수
(E-mail : kkman@mail.sangji.ac.kr)
※본 연구는 한국과학재단 목적기초연구(R01-2002-000-00041-0) 지원으로 수행되었음.

1. 서 론

최근에 프로그래머가 프로그램을 한번 작성하면 프로세서나 운영체제와 같은 플랫폼에 의존하지 않고 어느 시스템에서나 실행할 수 있는 컴파일러 기술에 대한 연구와 가상기계를 이용하여 모바일 디바이스 상에서 동적인 애플리케이션을 실행할 수 있는 기술이 국내외적으로 급부상하고 있다. 또한, 무선 네트워크를 통해 단말기로 동적인 애플리케이션을 다운로드할 수 있다는 점에서 가상기계 기술에 관한 연구가 폭넓게 진행되고 있으나 휴대폰의 모바일 플랫폼 분야를 제외하곤 PDA, 디지털 TV, 셋톱박스 등과 같은 임베디드 시스템 분야에서는 가상기계 기술에 관한 연구가 현재까지는 매우 미흡한 실정이다.

가상기계란 프로세서나 운영체제 등이 바뀌더라도 응용프로그램을 변경하지 않고 사용할 수 있는 기술로 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스와 디지털 TV 등에 탑재할 수 있는 핵심기술로 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다. 또한, 응용프로그램의 개발 방법 및 실행방법은 크게 네이티브 애플리케이션과 가상기계 애플리케이션으로 나눌 수 있으며 전자는 이제까지 사용하던 방법으로 실행속도 면에서는 탁월한 장점을 갖는다. 그러나, 플랫폼이 바뀌면 모든 응용프로그램을 변경해야 할뿐만 아니라 심지어는 사용할 수 없게 된다. 이러한 단점을 극복하기 위해서 가상기계를 탑재하여 응용프로그램을 실행시켜주는 가상기계 솔루션이 등장하였으며 특히, 임베디드 시스템에서는 프로세서 및 운영체제의 다양성과 잦은 변경으로 인하여 가상기계를 이용하는 것이 적합하고 더욱이 다운로드 솔루션에서는 가상기계 애플리케이션이 타당한 방법이라 판단된다.

임베디드 시스템을 위한 가상기계 개발은 전 세계적으로 무한한 시장 규모를 가지고 있다. 현재까지 대부분의 애플리케이션은 프로세서 및 운영체제의 종류에 의존적인 네이티브 애플리케이션(Native Application) 형태에서 앞으로 SUN 사의 자바 솔루션, 마이크로소프트사의 .NET 솔루션과 같이 프로세서 및 운영체제의 종류에 독립적인(이식성 및 호환성) 가상기계 애플리케이션 형태로 변환될 것이다.

이런 이유로, 본 논문에서는 모바일 디바이스(휴대폰, PDA), 셋톱 박스, 디지털 TV 등에 탑재되어 수많은 동적 애플리케이션을 다운로드하여 실행할 수 있는 가상기계를 설계하고 구현하였다. 본 연구에서 구현한 가상기계는 플랫폼이 변경되더라도 컨텐츠의 수정 없이 실행할 수 있으며, 개발 언어인 C 언어나 Java 언어에 관계없이 모두 수용하여 실행할 수 있는 소프트웨어 기술이다. 실제로, 제안된 가상기계는 기존에 존재하는 가상기계들의 표준화 모델로 사용될 수 있을 뿐 아니라 모바일 디바이스, 디지털 TV, 셋톱 박스 등과 같은 임베디드 시스템에 효과적으로 내장되어 응용프로그램을 효율적으로 실행시켜 줄 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 임베디드 시스템을 위한 가상기계 코드인 SIL을 설계하고 의사코드와 연산코드를 기술하였으며, SIL을 실행파일포맷으로 변환하는 어셈블러인 EFF 생성기를 서술하였다. 3장에서는 임베디드 가상기계의 구현모델과 SIL 코드 번역기 시스템에 대해 기술하였고, 4장에서는 임베디드 가상기계(EVM)의 구현내용을 기술하였다. 끝으로 5장에서는 본 연구에 대한 요약 및 제언을 하면서 결론을 맺는다.

2. 관련연구

2.1 가상기계 코드(SIL) 설계

EVM의 가상기계 코드인 SIL(Standard Intermediate Language)은 일반적인 임베디드 시스템을 위한 가상기계 코드의 표준화 모델로 설계 되었다. SIL은 스택 기반의 명령어 집합이며 언어 독립성과 하드웨어 및 플랫폼 독립성을 갖고 있다. SIL은 다양한 프로그래밍 언어를 수용하기 위해서 바이트코드, .NET IL 등 기존의 가상기계 코드들의 분석을 토대로 정의 되었으며, 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합이다. 또한 어셈블리 언어 수준의 디버깅을 용이하게 하기 위해 일관성 있는 이름 규칙을 적용하여 가독성 높은 니모닉을 정의하였다[14,16].

C#, 자바 등의 고급 언어로 작성된 프로그램은 번역기를 통해 표준중간언어인 SIL로 번역된다. SIL은 어셈블러(EFF Builder)에 의해 *.evm 형태로 변환되어 시스템의 운영체제나 아키텍처에 상관없이 EVM에 의해 실행된다. *.evm 파일은 디스어셈블러(SIL Extractor)를 통해 *.sil로의 역변환이 가능하다.

1) 의사코드

의사코드(pseudo code)는 클래스 선언 등 특정 작업의 수행을 나타내는 코드이다. SIL의 의사코드는 의사코드임을 나타내는 특수문자인 '.'과 수행하는 작업을 의미하는 니모닉의 조합으로 정의하였다. 의사코드의 니모닉은 원시코드 수준의 키워드를 그대로 사용하여 가독성을 높였다. SIL의 의사코드는 총 21개이며 각 의사코드와 그 의미를 표 1에 기술하였다. 주석은 '//'으로 시작하며 해당 행의 끝까지를 주석으로 간주한다.

표 1. SIL의 의사 코드

지시자	의 미
.class	생성되는 클래스 이름과 접근수정자를 정의
.field	클래스의 필드를 선언
.method	클래스의 메소드를 선언
.maxstack	해당 메소드의 최대 스택의 개수를 제한
.throws	해당 메소드에서 발생하는 예외를 선언
.catch	해당 메소드에서 예외 검사를 위한 구역 설정
.end	블럭의 끝을 나타내는 지시자
.locals	지역 변수를 선언

2) 연산코드

연산코드는 6개의 카테고리로 분류할 수 있으며 각각의 카테고리는 서브 카테고리를 갖는다. Stack operations 카테고리는 기본적인 스택 제어 및 저장 장소에 따른 push/load와 pop/store 연산 코드를 포함한다. Arithmetic operations 카테고리는 수학적 연산과 비교 연산, 비트 연산, 그리고 논리 연산에 관련된 연산 코드를 포함한다. Flow control operations 카테고리는 branch와 메소드 호출에 관한 연산 코드를 포함한다. Type conversion operations 카테고리는 스택 탑에 있는 데이터의 타입을 변환하는 연산 코드를 포함한다. Object operations 카테고리는 객체 생성과 필드에 관련된 연산 코드를 포함한다. 그 이외의 연산 코드는 Other operations 카테고리 안에 포함된다.

그림 1은 SIL 연산코드의 카테고리를 나타낸 것이다.

3) SIL의 완전성 증명

SIL의 완전성을 증명하기 위해서 기존에 이미 널리 사용되고 있는 가상기계 코드를 SIL 카테고리에 매핑

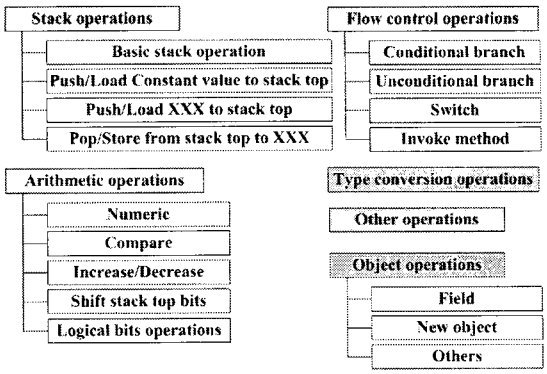


그림 1. SIL 연산코드의 카테고리

시킴으로써 구조적으로 SIL의 완벽성을 증명하였다 [13,16,21].

1) SIL 카테고리에 대한 Oolong 카테고리 매핑

Oolong의 카테고리는 메인 카테고리 수준에서 SIL 카테고리에 명백하게 매핑이 이루어진다. 스택에서 값을 가져오거나 스택 탑에 특정 값을 저장하는 코드들의 카테고리는 SIL의 Stack operations 카테고리에 매핑된다. 수학적 연산 및 비트 연산, 논리 연산 등의 카테고리는 SIL의 Arithmetic operations 카테고리에 매핑된다. 그 이외의 카테고리는 모두 각각 해당되는 SIL 카테고리에 1대1 매핑이 이루어진다. 그림 2는 Oolong에서 SIL로의 매핑을 나타낸 것이다.

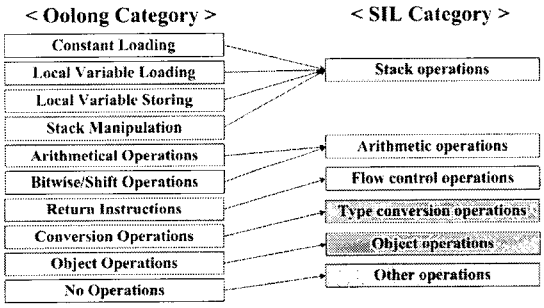


그림 2. Oolong에서 SIL로의 매핑

2) SIL 카테고리에 대한 .NET IL 카테고리 매핑

.NET IL 카테고리에서 SIL 카테고리의 매핑은 Oolong의 경우 보다 좀 더 복잡하다. 메인 카테고리 수준에서의 매핑은 1대 1 매핑뿐만 아니라 1대 다의 매핑이 발생한다. 따라서 서브 카테고리 수준의 매핑을 통한 증명이 필요하다. 서브 카테고리 수준에서는

.NET IL의 카테고리가 SIL 카테고리에 명백하게 매핑된다. 메인 카테고리 매핑에서 가장 복잡한 매핑을 나타낸 두 카테고리의 서브 카테고리 매핑을 그림 3에 나타내었다.

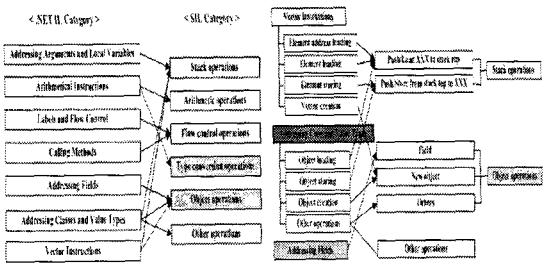


그림 3. .NET IL에서 SIL로의 서브 카테고리 매핑

2.2 EFF 생성기

EFF(*.evm)는 가상기계를 위한 표준 중간언어(*.sil)를 입력으로 받는 EFF Builder(어셈블러)의 출력이다. EFF는 EVM에 입력되어 실행 결과를 만들어 낸다. EFF는 다음과 같은 설계목표에 따라 정의 하였다[28]. 첫째, 통합언어 지원이다. 순차적 언어인 C와 Pascal 등을 지원하며 C#과 Java 등 객체 지향 언어를 지원 한다. 둘째, 확장이 가능하고 융통성을 갖는 것이다. 셋째, 메타데이터와 중간 언어가 서로 독립적으로 구성하여 분석이 쉽고 타입 체크가 편리한 구조로 설계하였다. 넷째, 임베디드 시스템을 고려하고 있기 때문에 불필요한 정보를 제거하고 실행에 필요한 최소의 정보만 저장한다.

1) EVM 파일 포맷의 구조

EFF는 메타데이터(MDT)와 중간언어(SIL)를 분리한 형태이다. 메타데이터 간에 참조는 테이블 인덱스를 이용한다. 인덱스는 해당 테이블의 태그와 레코드 번호로 이루어져 있어 인덱스만 보아도 어떤 테이블을 참조하고 있는지 알 수 있다. EFF는 클래스가 기본 단위이며 여러 클래스를 1개의 파일에 저장한다. 그러므로 순차적 언어를 지원하기 위하여 더미 클래스를 생성하여 저장하게 된다[21,22].

그림 4는 EFF의 전체 구조를 나타낸 것이다. EFF을 식별하기 위한 magic은 0x0E054DFF 값을 가진다. majorVersion과 minorVersion은 EVM의 주 버전과 부 버전이다. module은 소스 파일의 이름이며 MDTString index의 값을 가진다. language는 소스 코드의 언어를 나타내며 MDTString index의 값을 가진다. 프로그램의 시작 메소드를 가리키는 entryPoint는 MDTMethod index의 값을 가진다.

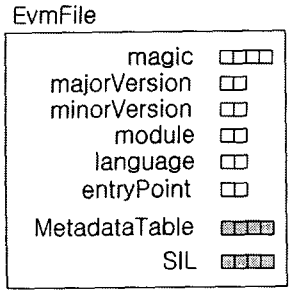


그림 4. EFF의 전체구조

2) SIL

SIL은 프로그램의 실행을 위한 명령어들의 집합이다. MDT와 SIL의 경계는 tag를 보고 알 수 있다. 그림 5는 SIL의 구조를 나타낸 것이다. silsCount는 SIL의 레코드 개수이다. 이 개수만큼 MDTMethod Index와 해당 바이트가 저장된다. 바이트들의 길이는 해당 MDTMethod의 SIL Length에 저장되어 있다. 바이트들은 명령어와 인수 또는 MDT를 포함한다.

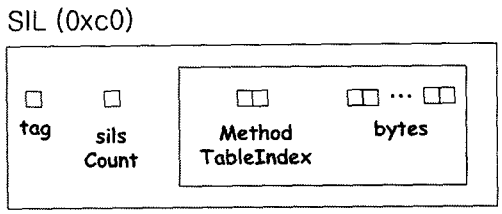


그림 5. SIL의 구조

3) EFF Builder의 세부구성

EFF Builder는 SIL Translator를 통해 얻어진 가상기계의 표준 중간언어인 *.sil 파일을 입력으로 받아서 이를 EVM에서 실행할 수 있는 *.evm 파일을 생성해주는 실행파일포맷 생성기이다. EFF Builder는 크게 어휘 분석기, 구문 분석기, 포맷 생성기로 구성되며 그 구조도는 그림 6과 같다[21,22].

여기서, 포맷 생성기는 EFF Builder의 입력인 SIL 프로그램이 어휘 분석과 구문 분석 과정을 거쳐 변환된 AST(Abstract Syntax Tree)를 순행하면서 실행파일인 *.evm을 생성하게 된다.

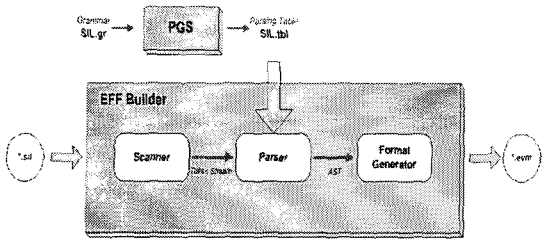


그림 6. EFF Builder 구조도

3. SIL 코드 번역기

3.1 EVM의 구현 모델

임베디드 시스템을 위한 가상기계 모델은 그림 7과 같다.

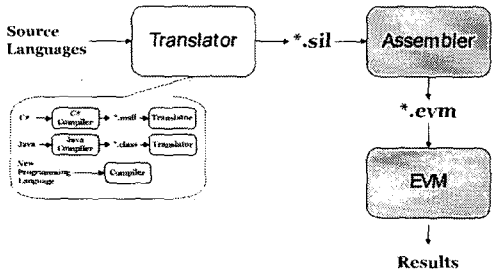


그림 7. 임베디드 시스템을 위한 가상기계 모델

임베디드 시스템을 위한 가상기계를 개발하기 위해 가상기계 개발 방법을 설계부분과 구현부분으로 분리하여 재목적성을 증가시킬 수 있도록 연구하는 방법을 사용하였다. 설계부분은 첫째, 프로그래밍언어 의존적인 부분, 둘째, 목적기계 의존적인 부분, 셋째, 프로그래밍언어와 목적기계에 독립적인 부분으로 구성되어 있다. 구현부분은 다양한 프로세서와 운영체제 등에 탑재를 용이하게 하기 위해 목적기계 의존적인 부분과 목적기계 독립적인 부분으로 나누어 개발하였다[9,14,16].

3.2 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 재목적성을 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[9,18-20].

번역기 시스템은 스캐너(Scanner), 파서(Parser), SDT(Syntax Directed Translation), AST(Abstract Syntax Tree) 그리고 ICT(Intermediate Code Translator)의 5부분으로 구성된다. 그림 8은 번역기 프로그램의 구성도이다.

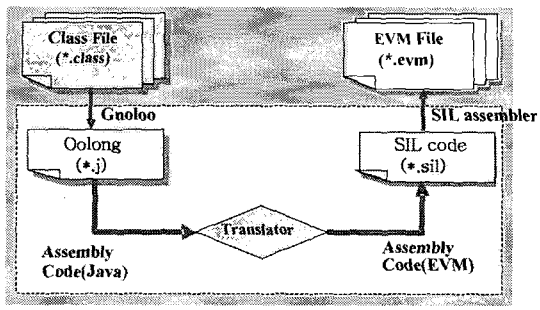


그림 8. Bytecode-to-SIL 번역기 시스템 구성도

스캐너와 파서를 구현하기 위해서 분석한 바이트코드 명령어를 가지고 context-free 문법을 설계하였다. 그리고 PGS(Parser Generating System)를 이용해 문법을 입력으로 어휘정보와 파싱 테이블을 생성한다. 코드 변환 부분인 ICT는 파서가 생성한 AST를 입력으로 받아 트리의 노드를 탐색하면서 매칭되는 노드에 대해 변환하고, Opcode 노드에서는 매핑 테이블을 참조하여 Oolong 코드와 동일한 의미를 가지는 SIL 코드로 변환한다. 이와 같이 변환함으로서 바이트코드 프로그램을 의미적으로 동등한 SIL 코드 프로그램으로 번역할 수 있다. 표 2는 바이트코드와 SIL 명령어 매핑 테이블을 나타낸 것이다.

표 2. 바이트코드와 SIL 명령어 매핑 테이블

분류	Oolong Code	매핑	SIL Code
산술 명령어	*add (* - Type)	N:1	add
	mul (- Type)		mul
	and (- Type)		and
	or (- Type)		or
	…	…	…
제어 명령어	Ifne	1:N	ldc 0, ne
	ifeq		ldc 0, eq
	if_icmple	1:1	le
	if_icmpge		ge
	…	…	…

분류	Oolong Code	매핑	SIL Code
객체 명령어	getfield	1:1	ldfld
	putstatic		strsfld
	invokespecial		call
	invokevirtual		calli
	...	...	...
변환 명령어	d2l	1:1	convl
	l2i		convi
	...	...	...
스택 명령어	dup2	1:N	dup, dup
	...	...	...
변수 명령어	*load (* - Type)	N:1	ldl <parameter>
	store (- Type)		str <parameter>
	...	...	...

3.3 MSIL-to-SIL 번역기 시스템

MSIL-to-SIL 중간 언어 번역기 시스템은 실행파일(*.exe)로부터 MSIL 코드를 추출하여 생성된 *.il 파일을 입력으로 받아서 EVM에서 실행될 수 있도록 SIL 코드 즉, *.sil을 생성한다. 그림 9는 번역기 시스템이 입력으로 MSIL 파일을 받아 출력으로 SIL 코드를 생성하는 과정을 나타낸 것이다[17,23].

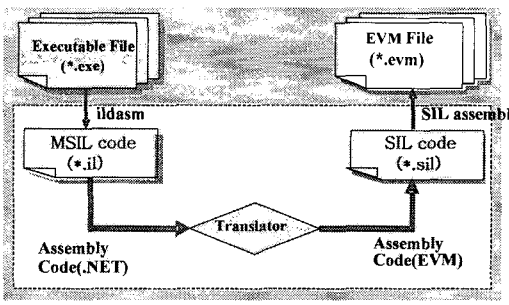


그림 9. MSIL 코드의 SIL 코드 번역과정

역어셈블러(ildasm.exe)를 통해 추출된 MSIL 코드로 구성된 *.il 파일은 번역기 시스템에 의해 SIL 코드로 구성된 *.sil 파일로 변환된다. 생성된 *.sil 파일은 SIL 어셈블러에 의해 실행파일인 *.evm 파일로 변환되고, 이것을 EVM으로 실행하면 .NET 언어로 작성된 프로그램의 실행 결과와 같은 결과를 얻을 수 있다. 표 3은 MSIL 코드와 SIL 명령어 매핑 테이블을 나타낸 것이다.

표 3. MSIL 코드와 SIL 명령어 매핑 테이블

offset	MSIL	SIL	Description
0×00	nop	nop	No Operation
:	:	:	:
0×06	ldloc.0	ldl	Load First local variable onto the stack
:	:	:	:
0×0a	stloc.0	str	Pop value from stack into first local variable
:	:	:	:
0×17	ldc.i4.1	ldc	Push 1 onto the stack as i4
:	:	:	:
0×2a	ret	ret	Return from method
:	:	:	:
0×58	add	add	Add values, returning a new value
0×59	sub	sub	Subtract values, returning a new value
0×5a	mul	mul	Multiply values, returning a new value
0×5b	div	div	Divide values, returning a new value
:	:	:	:

4. 임베디드 가상기계(EVM)

EVM(Embedded Virtual Machine)은 ANSI C 언어와 SUN사의 자바 언어 등을 모두 수용할 수 있는 임베디드 시스템을 위한 가상기계로 모바일 디바이스, 셋톱박스, 디지털 TV 등에 탑재되어 동적 응용 프로그램을 다운로드하여 실행할 수 있는 가상기계 솔루션이다. EVM은 크게 컴파일러, 어셈블러, 가상기계의 세부분으로 구성되며, 계층적인 구조로 설계되어 목적기계에 설치하는 과정의 부담을 최소화 한다[14].

4.1 EVM을 위한 로더의 설계 및 구현

EVM을 위한 로더는 그림 10과 같이 main() 함수를 포함하는 ⓐ 부분, 다양한 자료 저장을 위해 구현 스택 및 힙을 초기화하기 위한 함수(VmInit( )) 및 로더 호출함수(VmStartApp( ))를 포함하는 ⓑ, 마지막으로 기본 자료형 및 라이브러리 함수가 포함되어

있는 ⓒ 부분으로 구성되어 있다.

로더는 *.evm 내에서 여러 가지 정보를 추출하여 힙 공간(evmHashList, evmHeap)과 하나의 구조체(EvmStruct* evm)에 저장된다. evmHashList는 전역 포인터 배열로 선언되어 있으며 evmHeap에는 저장되어 있는 클래스 정보의 시작 주소를 가지고 있다. evmHeap은 전역 포인터 변수로 선언되어 있으며, 중간코드인 *.evm의 모든 정보를 저장하게 된다. 이때 저장 되는 내용은 대부분 주소 값을 저장하여 나중에 *.evm내에서 필요한 정보를 빠르게 찾을 수 있도록 한다[11].

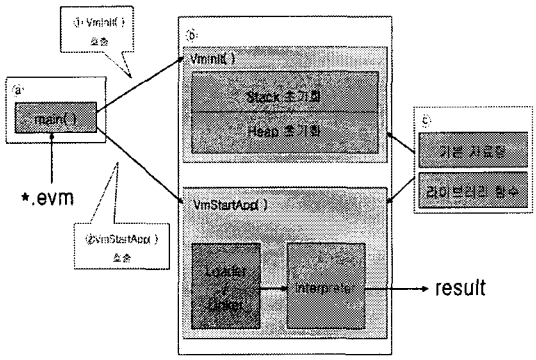


그림 10. 로더의 세부 구성도

4.2 EVM의 설계 및 구현

EVM은 EFF 빌더가 생성한 *.evm을 입력으로 받아 실행하는 가상기계 루틴이다. 가상 기계는 크게 정의 부분, 초기화 부분, 실행 부분으로 나누어져 있으며, 1개의 핵심 파일과 두 개의 포팅 관련 소스로 나누어 개발하였다. 인터프리터는 *.evm 파일의 SIL 코드들을 실제적으로 실행하는 루틴으로 각각의 SIL 코드에 대해 실제 처리 동작을 작성하였다. 인터프리터는 로더의해 저장된 클래스 정보들을 클래스 영역에 저장하여 유지하며 인터프리터 실행시에 참조되어진다. 처리중 발생되어지는 데이터들은 스택과 힙에 저장되어 관리되며 수행중에 에러 발생시 해당 에러 메시지를 출력하고 종료하도록 구현되었다. 각 메소드의 소멸은 호출의 역순서로 진행되기 때문에 스택에 프레임(frame) 정보가 없으면 이를 종료상태로 보고 인터프리터의 수행을 종료한다[10,15].

그림 11은 인터프리터의 구현을 위한 세부 설계도이다.

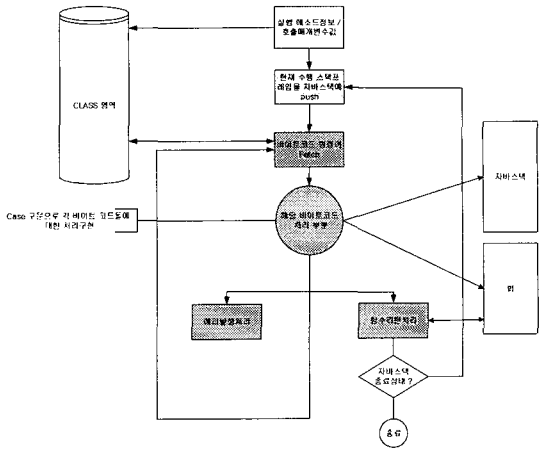


그림 11. 인터프리터 구현 내용

4.3 실험결과 및 분석

다음은 완전수(perfect number)를 구하는 프로그램을 EVM의 중간언어인 SIL로 번역하여 가상기계인 evm으로 실행하는 예제이다.

1) perfect.c

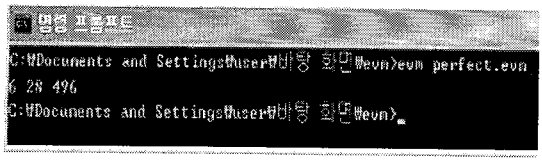
```
void main()
{ int i,j,k;
  int rem,sum;
  int max=500;

  for (i=2; i<=max; i++) {
      sum=0;
      k=i/2;
      for (j=1; j<=k; j++) {
          rem = i%j;
          if (rem == 0) sum += j;
      }
      if (i==sum) printf("%d ", i);
  }
}
```

2) perfect.sil

```
%%Data Section
:
%%Code Section
%File: perfect.c
main:
proc 2 4 1 1
.sym i      i  0 0 302 0 0
.sym j      i  0 0 303 0 0
.sym k      i  0 0 304 0 0
.sym rem i  0 0 305 0 0
.sym sum i  0 0 306 0 0
.sym max i  0 0 307 0 0
%Line 7: int max = 500;
ldc.i 500
str.i 1 20
%Line 130: for(j=2; i<=max; i++){
ldc.i 2
str.i 1 0
```

```
$$0: nop
lod.i 1 0
lod.i 1 20
le.i
fjp $$1
%Line 131: sum = 0;
ldc.i 0
str.i 1 16
%Line 132: k = i/2;
lod.i 1 0
ldc.i 2
div.i
str.i 1 8
%Line 133: for(j=1; j<=k; j++){
ldc.i 1
str.i 1 4
$$3: nop
lod.i 1 4
lod.i 1 8
le.i
fjp $$4
%Line 134: rem = i%j;
lod.i 1 0
lod.i 1 4
mod.i
str.i 1 12
%Line 135: if (rem == 0)sum += j;
lod.i 1 12
ldc.i 0
eq.i
fjp $$6
%Line 135: if (rem == 0)sum += j;
lod.i 1 16
lod.i 1 4
add.i
str.i 1 16
$$6: nop
$$5: nop
lod.i 1 4
dup
ldc.i 1
add.i
str.i 1 4
ujp $$3
$$4: nop
%Line 137: if(i==sum) printf("%d ", i);
lod.i 1 0
lod.i 1 16
eq.i
fjp $$7
%Line 137: if(i==sum) printf("%d ", i);
ldp
ldc.p %d
lod.i 1 0
call printf
$$7: nop
$$2: nop
lod.i 1 0
dup
ldc.i 1
add.i
str.i 1 0
ujp $$0
$$1: nop
```

3) 결과



5. 결 론

가상기계란 프로세서나 운영체제 등이 바뀌더라도 응용프로그램을 변경하지 않고 사용할 수 있는 기술로 특히, 임베디드 시스템을 위한 가상기계 기술은 모바일 디바이스와 디지털 TV 등에 탑재할 수 있는 핵심기술로 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

또한, 응용프로그램의 개발 방법 및 실행방법은 크게 네이티브 애플리케이션과 가상기계 애플리케이션으로 나눌 수 있으며 전자는 이제까지 사용하던 방법으로 실행속도 면에서는 탁월한 장점을 갖는다. 그러나, 플랫폼이 바뀌면 모든 응용프로그램을 변경해야 할뿐만 아니라 심지어는 사용할 수 없게 된다. 이러한 단점을 극복하기 위해서 가상기계를 탑재하여 응용프로그램을 실행시켜주는 가상기계 솔루션이 등장하였으며 특히, 임베디드 시스템에서는 프로세서 및 운영체제의 다양성과 잦은 변경으로 인하여 가상기계를 이용하는 것이 적합하고 더욱이 다운로드 솔루션에서는 가상기계 애플리케이션이 타당한 방법이라 판단된다.

이런 이유로, 본 논문에서는 모바일 디바이스(휴대폰, PDA), 셋톱박스, 디지털 TV 등에 탑재되어 수많은 동적 애플리케이션을 다운로드하여 실행할 수 있는 가상기계를 설계하고 구현하였다. 본 연구에서 구현한 가상기계는 플랫폼이 변경되더라도 컨텐츠의 수정 없이 실행할 수 있으며, 개발 언어인 C 언어나 자바 언어에 관계없이 모두 수용하여 실행할 수 있는 소프트웨어 기술이다. 따라서, 자바 프로그래머나 .NET 프로그래머는 플랫폼 환경에 관계없이 프로그램을 작성하여 실행시킬 수 있다. 또한, 프로그래머는 응용 프로그램을 개발할 때 언어의 제약을 받지 않고 프로그램을 개발할 수 있다. 실제로, 제안된 가상기계는 기존에 존재하는 가상기계들의 표준화 모델로 사용될 수 있을 뿐 아니라 모바일 디바이스, 디지털 TV, 셋톱박스 등과 같은 임베디드 시스템에 효과적으로 내장되어 응용프로그램을 효율적으로 실행시켜 줄 수 있다.

참 고 문 헌

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

[2] Don Box and Chris Sells, *Essential .NET Volume 1 The Common Language Runtime*, Addison Wesley, 2002.

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

[4] John Meyer and Troy Downing, *JAVA Virtual Machine*, O'REYLLY, 1997.

[5] Joshua Engel, *Programming for the Java Virtual Machine*, Addison-Wesley, 1999.

[6] Ken Arnold and James Gosling, *The Java^{tm} Programming Language*, Addison-Wesley, 1996.

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

[8] Tim Lindholm and Frank Yellin, *The Java^{tm} Virtual Machine Specification*, 2nd Ed., Addison-Wesley, 1997.

[9] 권혁주, 김영근, 이양선, "재목적 Oolong-to-SIl 중간 언어 번역기," *한국멀티미디어학회 춘계학술발표논문집*, 제7권, 제1호, pp. 310-313, 2004 5월.

[10] 김선귀, 고광만, "소규모 장치를 위한 가상기계의 설계에 관한 연구," *한국정보처리학회 추계학술발표논문집(중)*, 제10권 제2호 pp. 805-808, 2003년 11월.

[11] 김성진, 고광만, "EVM을 위한 로더의 설계 및 구현," *한국정보과학회 봄 학술발표논문집(B)*, 제31권 제1호 pp. 877-879, 2004년 4월.

[12] 김영근, 권혁주, 이양선, "구문 트리를 이용한 자바 바이트코드에서 SIL로의 번역기," *한국정보처리학회 춘계학술발표논문집*, 제11권, 제1호, pp. 519-522, 2004년 5월.

[13] 남동근, 오세만, "가상기계 코드의 커버링 문제," *한국정보과학회 가을 학술발표논문집(I)*, 제30권 제2호 pp. 247-249, 2003년 10월.

[14] 남동근, 윤성림, 오세만, "가상기계를 위한 어셈블리 언어," *한국정보처리학회 춘계학술발표 논문집*, 제10권, 제1호, pp. 783-786, 2003년 5월.

[15] 오세중, 고광만, "멀티 플랫폼 상에 가상기계 탑재를 위한 어댑터의 설계," *한국정보처리학회 추계학술발표논문집(중)*, 제10권 제2호 pp. 809-812, 2003년 11월.

[16] 윤성림, 남동근, 오세만, 김정숙, "Virtual Machine Code for Embedded Systems," International Conference on CIMCA' 2004년 7월.

[17] 이양선, 황대훈, 나승원, "JVM 플랫폼에서 .NET 프로그램을 실행하기 위한 MSIL-to-Bytecode 번역기의 설계 및 구현," *한국멀티미디어학회 논문지*, Vol. 7, No. 2, 2004년 6월.

[18] 이양선, 나승원, 황대훈, "Intermediate Language Translator for Execution of Java Programs in .NET Platform," *한국멀티미디어학회 논문지*, Vol. 7, No. 2, 2004년 6월.

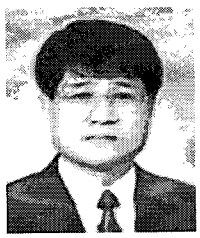
[19] 정지훈, 박진기, 이양선, "자바 바이트코드의 .NET MSIL 중간언어 번역기," *한국정보처리학회 추계학술발표논문집*, 제10권, 제2호, pp. 663-666, 2003년 11월.

[20] 정지훈, 이양선, "Design and Implementation of the Java Oolong-to-.NET MSIL IL Translator," SNPD'2004 International. Conference. 2004년 6월.

[21] 정한종, 오세만, "EVM 파일 포맷의 정의와 커버링 문제," *한국정보과학회 봄 학술발표논문집(B)*, 제31권 제1호 pp. 844-846, 2004년 4월.

[22] 정한종, 윤성림, 오세만, "가상 기계를 위한 실행 파일 포맷," *한국정보처리학회 추계학술발표논문집(중)*, 제10권 제2호 pp. 647-650, 2003년 11월.

[23] 최성규, 정지훈, 이양선, "임베디드 시스템을 위한 C# MSIL 코드의 Oolong 코드 번역에 관한 연구," *한국정보처리학회 춘계학술발표논문집*, 제10권, 제1호, pp. 983-986, 2003년 5월.



오 세 만

1977년 서울대학교 수학교육과 (이학사)
1979년 한국과학기술원 전산학과(이학석사)
1985년 한국과학기술원 전산학과(이학박사)
1985년 3월~현재 동국대학교 컴퓨터공학과 교수
2001년 11월~2003년 11월 한국정보과학회 SIGPL 위원장
2004년 4월~현재 한국정보처리학회 게임연구회 위원장
관심분야 : 컴파일러, 프로그래밍언어, 임베디드 시스템

이 양 선

1985년 동국대학교 전자계산학과(공학사)
1987년 동국대학교 대학원 컴퓨터공학과(공학석사)
1993년 동국대학교 대학원 컴퓨터공학과(공학박사)
1994년 3월~현재 서경대학교 컴퓨터공학과 교수
1996년 3월~2000년 2월 서경대학교 전자계산소 소장
2000년 2월~현재 멀티미디어학회 이사
2004년 4월~현재 한국정보처리학회 게임연구회 부위원장
관심분야 : 프로그래밍언어, 임베디드 시스템, 모바일 컴퓨팅

고 광 만

1991년 원광대학교 컴퓨터공학과(공학사)
1993년 동국대학교 대학원 컴퓨터공학과(공학석사)
1998년 동국대학교 대학원 컴퓨터공학과(공학박사)
1998년~2001년 광주여자대학교 컴퓨터학부 전임강사
2002년~2003년 Queensland University of Technology 연구교수
2001년~현재 상지대학교 컴퓨터정보공학부 조교수
관심분야 : 프로그래밍언어 설계 및 구현