EVM 파일 포맷의 정의와 커버링 문제*

정한종° 오세만
동국대학교 컴퓨터공학과
{prana, smoh}@dongguk.edu

Definition of EVM File Format and Covering Problem

Hanjong Cheong° , Seman Oh
Dept. of Computer Engineering, Dongguk University

요 약

가상 기계란 하드웨어로 이루어진 물리적 시스템과 달리 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 개념적인 프로세서이다. 가상 기계 기술은 프로세서나 운영체제가 바뀌더라도 응용프로그램을 변경하지 않고 사용할 수 있는 장점이 있다. 임베디드 시스템을 위한 가상 기계 기술은 모바일 장치나 디지털-TV 등에 탑재할 수 있는 핵심기술로서 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

현재 EVM 이라 명명된 임베디드 시스템을 위한 가상 기계에 대하여 연구가 진행 중이다. 이러한 연구의 일환으로 본 논문에서는 기존의 가상 기계를 위한 실행 파일 포맷들의 분석을 기반으로 하여 임베디드 시스템을 위한 실행 파일 포맷인 EVM 파일 포맷(EFF)을 정의한다. 또한 제안한 EFF 의 완전성을 증명하기 위하여, 기존에 널리 사용되고 있는 실행 파일인 클래스 파일을 이용해서 구조적으로 증명한다.

1. 서론

가상 기계 개념은 목적 기계에 영향을 받지 않는 컴파일러로부터 시작되었다. 즉, 기존에는 응용 프로그램들이 하드웨어와 운영체제에 종속적이었으나, 가상 기계는 플랫폼 독립을 가능하게 한다. 대표적인 가상 기계로서 클래스 파일을 입력으로 받아 실행하는 JVM(Java Virtual Machine)있다. 최근에는 GVM, KVM 등 모바일 단말기를 위한 가상 기계들이 개발되면서 그 중요성이 더욱 부각되고 있다. 특히, 임베디드 시스템을 위한 가상 기계 기술은 모바일 장치와 디지털-TV 등에 탑재할 수 있는 핵심기술로서, 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

이러한 각각의 가상 기계는 입력으로 받는 실행 파일을 갖고 있다. 예를 들면, JVM 의 클래스 파일, .NET CLR 의 PE 파일, GVM 의 SGS 파일, EVM 의 EVM 파일이 있다.

EVM(Embedded Virtual Machine)이라 명명된 이 가상 기계 솔루션은 객체지향 언어뿐만 아니라, 순차적인 언어로 작성된 프로그램들을 가상 기계를 위한 코드(*.sil)를 거쳐서 EVM 파일(*.evm) 포맷으로 변환하여 임베디드 시스템에 탑재된 가상 기계에서 실행할 수 있도록 한다.

이러한 연구의 일환으로 본 논문에서는 기존의 가상 기계를 위한 파일 포맷들의 분석을 기반으로 하여 임베디드 시스템을 위한 실행 파일 포맷인 EVM 파일 포맷(EFF)을 정의한다. 또한 제안한 EFF 의 완전성을 증명하기 위하여 기존에 널리 사용되고 있는 실행 파일인 클래스 파일을 이용해서 구조적으로 증명한다. 클래스 파일 포맷의 구조는 SUN 사에서 제공하는 자료를 기준으로 한다[3].

2. 관련 연구

2.1 PE 파일 포맷

.NET 실행 파일은 마이크로소프트 PE(Portable Executable)와 COFF(Common Object File Format)의 기준을 토대로 확장한 형태로써, CLR(Common Language Runtime) 환경에서 동작하기 위한 메타데이터와 IL 을 포함하고 있다. .NET 실행 파일은 기존에 실행 파일과 동일하게 확장자가 EXE 또는 DLL 이며 리틀 엔디언(Little-Endian)의 순서로 저장된다[1].

2.2 클래스 파일 포맷

클래스 파일은 자바 소스 코드가 자바 컴파일러에 의해 생성된 파일로서 확장자가 class 이며 자바 가상 기계(JVM)에 입력되어 실행된다. 8 비트 단위의 스트림으로 구성되어 있어 16 비트, 32 비트, 64 비트 크기를 가진 데이터들은 8 비트 단위로 나누어져 높은 비트가 먼저 나오는 빅 엔디언(Big-Endian)의 순서로 저장된다[3].

3. EVM 파일 포맷 (EFF)

3.1 설계 목표

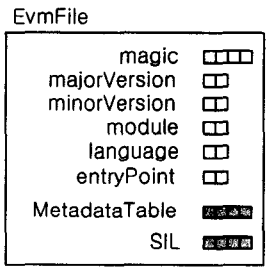
EFF 설계 목표는 크게 4 가지가 있다. 첫 째, 통합언어 지원이다. 순차적 언어인 C 와 Pascal 등을 지원하며 C#과 Java 등 객체 지향 언어도 지원 한다. 둘 째, 확장이 가능하고 융통성을 갖

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

는 것이다. 셋 째, 메타데이터와 중간 언어가 서로 독립적으로 구성하여 분석이 쉽고 타입 체크가 편리한 구조로 설계한다. 넷 째, 임베디드 시스템을 고려하고 있기 때문에 불필요한 정보를 제거하고 실행에 필요한 최소의 정보만 저장한다.

3.2 EVM 파일 포맷의 구조

EFF(*.evm)는 가상 기계를 위한 표준 중간언어(*.sil)를 입력으로 받는 EFF Builder(어셈블러)의 출력이다. EFF 는 EVM 에 입력 되어 실행 결과를 만들어 낸다. 메타데이터(MDT)와 중간언어(SIL)를 분리한 형태이다. 메타데이터 간에 참조는 MDT index 를 이용한다. MDT index 는 해당 테이블의 tag([그림 2] 참조)와 레코드 번호로 이루어져 있어 MDT index 만 보아도 어떤 테이블를 참조하고 있는지 알 수 있다. 예를 들면, MDT index 의 값이 0x2004 이면 MDTString(0x20)의 다섯 번째 레코드를 참조하는 것이다. 또한 참조하는 엔트리들이 존재하지 않으면 none(0x00) tag 를 채워준다. EFF 는 클래스가 기본 단위이며 여러 클래스를 1 개의 파일에 저장한다. 그러므로 순차적 언어를 지원하기 위하여 더미 클래스를 생성하여 저장하게 된다. [그림 1]은 EFF 의 전체 구조를 나타낸 것이다.



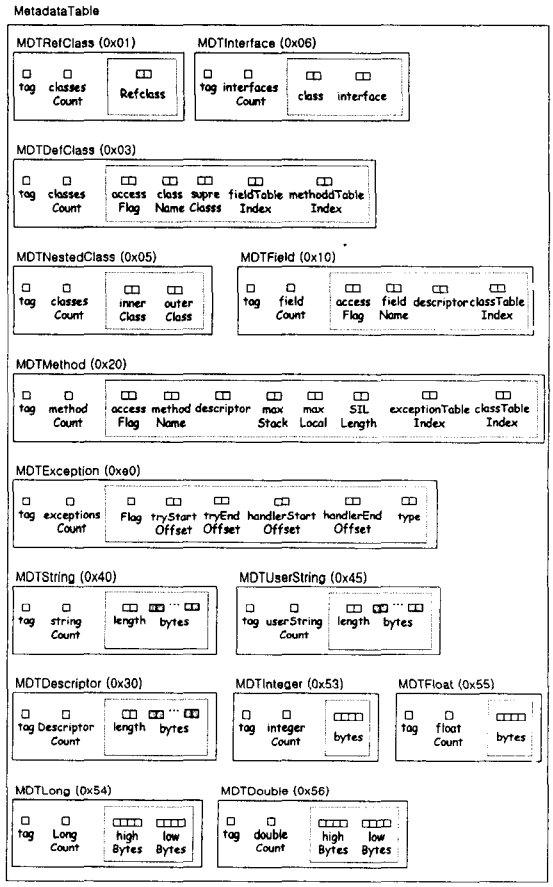
[그림 1] EFF 의 구조도

EFF 을 식별하기 위한 magic 은 0x0E054DFF 값을 가진다. majorVersion 과 minorVersion 은 EVM 의 주 버전과 부 버전이다. module 은 소스 파일의 이름이며 MDTString index 의 값을 가진다. language 는 소스 코드의 언어를 나타내며 MDTString index 의 값을 가진다. 프로그램의 시작 메소드를 가리키는 entryPoint 는 MDTMethod index 의 값을 가진다.

3.3 MetadataTable (MDT)

MDT 의 각 테이블들은 tag 와 엔트리들로 이루어져 있다. 테이블들의 순서가 정해져 있지 않으며 테이블은 필요할 때 마다 삽입 하면 된다. 다시 말하면, 해당 테이블 의 정보가 없으면 파일 내에 테이블이 존재하지 않는다. tag 의 숫자는 확장을 고려하여 설계하였다. 테이블의 레코드 수가 256 개를 초과하면 다음 tag(숫자)를 사용하게 된다.

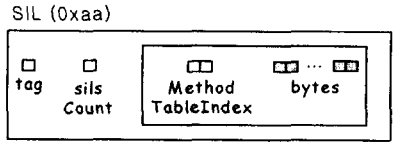
[그림 2]는 MDT 의 종류와 각각의 엔트리들을 나타낸 것이다. MDTRefClass 는 참조하는 클래스의 정보를 저장하며, MDTInterface 는 인터페이스 정보와 이를 상속하는 클래스 또는 인터페이스 정보를 저장한다. MDTNestedClass 는 중첩되는 외부와 내부 클래스의 정보를 저장한다. MDTDefClass 는 정의한 클래스의 정보를 저장한다. MDTField 와 MDTMethod 는 필드와 메소드의 정보를 저장한다. MDTException 는 예외 처리를 위한 정보를 저장한다. MDTInteger, MDTFloat, MDTLong, 그리고 MDTDouble 은 각 타입의 상수 값을 저장한다. MDTString 은 시스템에서 사용하는 스트링을 저장하며 MDTUserString 은 프로그래머가 정의한 스트링을 저장한다. MDTDescriptor 는 필드의 타입과 메소드의 시그네춰 정보를 저장한다.



[그림 2] MDT의 종류 및 각 속성들

3.4 SIL

SIL 은 프로그램의 실행을 위한 명령어들의 집합이다. MDT 와 SIL 의 경계는 tag 를 보고 알 수 있다.

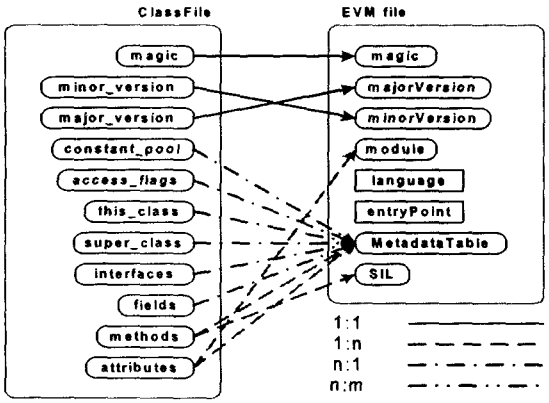


[그림 3] SIL 의 구조

[그림 3]은 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.

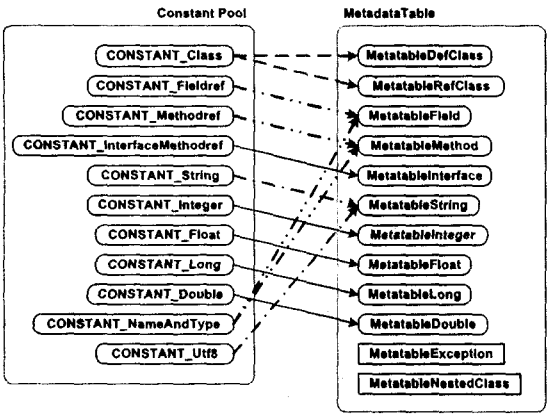
4. EFF의 완전성

임베디드 시스템을 위한 실행 파일 포맷(EFF)을 설계 목표에 따라 정의하였으나 제안한 EFF의 완전성 증명이 요구된다. 따라서 EFF의 완전성을 증명하기 위하여 클래스 파일을 이용해서 구조적으로 증명한다. 클래스 파일 포맷의 구조는 썬사에서 제공하는 자료를 기준으로 한다. 매핑이 이루어지지 않는 부분은 직사각형으로 표현한다.



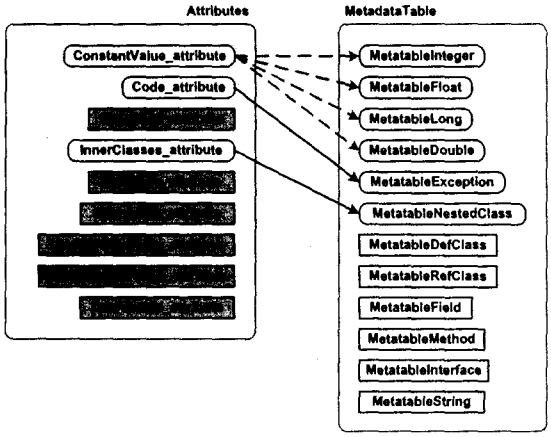
[그림 4] 클래스 파일에서 EFF로의 매핑

[그림 4]는 클래스 파일로부터 EFF로의 매핑을 나타낸 것이다. 클래스 파일 경우 1개의 클래스 단위로 구성 되나 EFF의 경우에는 1개 이상의 클래스를 포함하기 때문에 클래스 파일의 많은 부분들이 EFF의 MDT로 매핑 된다. 클래스 파일에서 EFF로의 매핑이 완전히 이루어지는 것을 [그림 4]를 통해서 알 수 있다.



[그림 5] 상수 풀에서 MDT로 매핑

[그림 5]는 클래스 파일의 상수 풀에서 EFF의 MDT로의 매핑을 나타낸 것이다. EFF의 MDT로 매핑이 되지 않는 부분은 클래스의 속성 부분에서 매핑이 이루어진다. 클래스의 상수풀이 EFF의 MDT로 완전하게 매핑되는 것을 [그림 5]에서 볼 수 있다.



[그림 6] 속성에서 MDT로 매핑

[그림 6]은 클래스 파일의 속성에서 EFF의 MDT로의 매핑을 나타낸 것이다. 클래스 파일의 속성에서 매핑이 이루어지지 않는 부분은 디버깅 정보, 불필요한 정보들, 그리고 상위 구조에서 매핑이 이루어진 부분들이다.

5. 결론 및 향후 연구

임베디드 시스템을 위한 실행 파일 포맷인 EFF를 설계 목표에 따라 정의하였다. 제안한 EFF에 대하여 완전성 증명이 요구되어 EFF를 구조적으로 증명하기 위하여, 가장 널리 알려진 가상 기계를 위한 실행 파일인 클래스 파일을 매핑 대상으로 하였다. 본 논문에서 불필요한 정보와 디버깅 정보를 제외한 모든 부분이 EFF로 완전히 매핑 됨을 증명하였다. 구조적으로 매핑이 이뤄졌으나 연구 과정에서 각각의 내부 데이터에 대한 매핑을 증명하였다. EFF는 가상 기계를 위한 실행 파일 포맷으로 완전하다고 판단된다.

향후 과제로는 .NET PE 파일의 커버링 문제를 해결하기 위하여 EFF의 보완과 수정이 필요하겠다. 더불어 EFF를 직접 실행할 수 있는 가상 기계 구현에 대한 연구가 계속 필요하겠다.

참고 문헌

[1] James S. Miller, The Common Language Infrastructure Annotated Standard, Addison Wesley, 2003.

[2] Serge Lidin, .NET IL Assembler, Microsoft Press, 2002.

[3] Tim Lindholm, The Java Virtual Machine Specification, SUN, 1999.

[4] 남동근, 윤성림, 오세만, " 가상기계의 어셈블리 언어" , 정보처리학회 춘계학술발표논문집(중), 제10권 제1호, pp.783~786, 2003.