유비쿼터스 게임 플랫폼을 위한 Java Bytecode-to-SAF 번역기의 설계 및 구현

박상훈*, 이양선*
*서경대학교 컴퓨터공학과
e-mail:{shpark*, yslee*}@skuniv.ac.kr

Design and Implementation of a Java Bytecode-to-SAF Translator for the Ubiquitous Game Platform

Sang-Hoon Park*, Yang-Sun Lee*
*Dept of Computer Engineering, SeoKyeong University

요 약

본 연구팀은 유비쿼터스 환경에서 다양한 분야의 콘텐츠를 보다 쉽게 개발하고 실행할 수 있는 통합 소프트웨어 개발 솔루션인 유비쿼터스 게임 플랫폼(Ubiquitous Game Platform)을 개발하였다. 유비쿼터스 게임 플랫폼은 C/C++ 언어와 자바 언어를 모두 지원하며, 가상기계 방식이기 때문에 여러 임베디드 기기에서 독립적으로 수행이 가능하다는 장점이 있다. 본 논문에서는 유비쿼터스 게임 플랫폼에서 자바 언어로 작성된 콘텐츠를 실행할 수 있도록 하기 위해 자바 바이트코드를 유비쿼터스 게임 플랫폼의 중간언어 형식인 SAF(Standard Assembly Format)로 변환해주는 Java Bytecode-to-SAF 번역기를 설계하고 구현하였다. Java Bytecode-to-SAF 번역기를 통해 C/C++ 콘텐츠뿐만 아니라 자바 콘텐츠도 하나의 플랫폼에서 실행이 가능해져서 개발자에게 효율적이며, 능률적인 개발 환경을 제공하고, 한번 개발한 콘텐츠를 타 기종으로 이식하기 위한 시간과 비용 소모를 줄여 생산성을 높일 수 있다.

1. 서론

유비쿼터스 환경의 임베디드 하드웨어 발전은 꾸준히 이루어졌다. 하지만 임베디드 기기에 필요한 소프트웨어, 즉 콘텐츠의 개발은 하드웨어에 비해 미약한 실정이다. 현재 국내 개발 환경에서는 콘텐츠를 해당 시스템에 맞게 다시 개발해 주어야 하는 단점이 있다[6,7]. 때문에 임베디드 시스템을 위한 콘텐츠 보급이 어려울 수밖에 없는 것이 사실이다. 본 연구팀은 임베디드 분야의 콘텐츠 보급 활성화를 도모하고자 콘텐츠를 보다 쉽게 개발하고 실행할 수 있는 환경 구축을 목표로 삼고, 다양한 언어를 지원하고 여러 임베디드 기기에서 독립적으로 수행이 가능한 통합 소프트웨어 개발 솔루션인 유비쿼터스 게임 플랫폼(Ubiquitous Game Platform)을 개발하였다. 본 논문에서는 유비쿼터스 게임 플랫폼에서 자바 콘텐츠 실행을 위해 자바 바이트코드를 유비쿼터스 게임 플랫폼의 중간언어 형식인 SAF(Standard Assembly Format)로 변환하는 Java Bytecode-to-SAF 번역기를 설계하고 구현하였다.

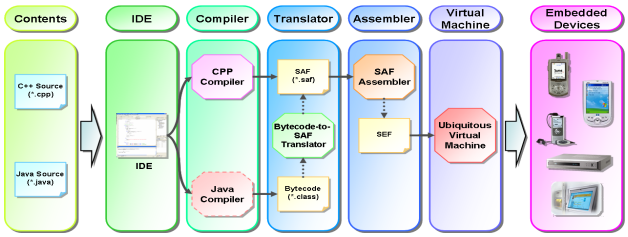
본 연구는 한국산업기술평가원의 '06 공통핵심 기술 개발 지원 사업의 연구 결과로 수행되었음.

2. 관련연구

2.1 유비쿼터스 게임 플랫폼(Ubiquitous Game Platform)

본 연구팀이 개발한 유비쿼터스 게임 플랫폼은 유비쿼터스 환경의 임베디드 기기를 위한 플랫폼이다. 유비쿼터스 게임 플랫폼에서는 현재 개발자들의 대다수가 사용하는 C/C++ 언어와 자바 언어를 모두 지원하기 위해, 순차적 언어와 객체지향 언어 모두를 수용할 수 있도록 설계된 중간언어 형식인 SAF(Standard Assembly Format)를 사용한다. 그리고 가상기계 방식을 채택하여 여러 임베디

드 기기에서 독립적인 수행이 가능하다[6,8]. (그림 1)은 유비쿼터스 게임 플랫폼의 구성도이다.

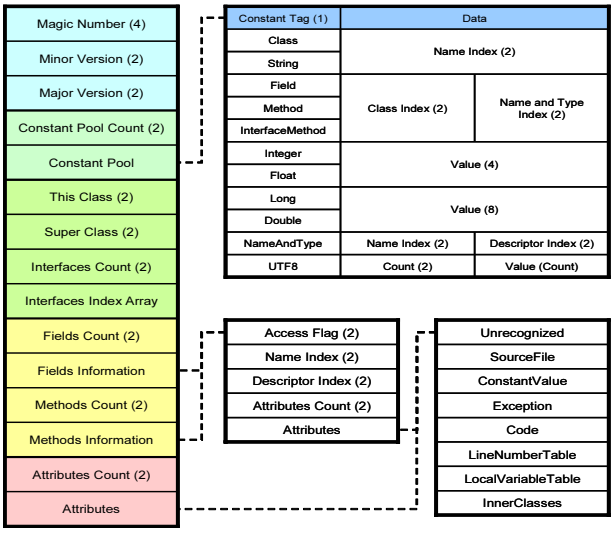


(그림 1) 유비쿼터스 게임 플랫폼 구성도

유비쿼터스 게임 플랫폼은 자체 C++ 컴파일러를 통해 C/C++ 프로그램을 SAF로 컴파일 한다. 그리고 어셈블러를 통해 가상기계의 실행 형식인 SEF(Standard Executable Format)로 변환한 후에 유비쿼터스 게임 플랫폼의 가상기계인 u-VM(Ubiquitous Virtual Machine)에서 실행시킨다[6]. 자바 프로그램은 본 논문에서 구현한 Java Bytecode-to-SAF 번역기를 통해 클래스 파일을 SAF로 변환하여 실행한다.

2.2 자바 바이트코드(Java Bytecode)

자바 바이트코드는 JVM(Java Virtual Machine)의 중간코드이다. 일반적으로 자바 언어를 컴파일 하면 생성되는 것으로, 명령어(Instruction)가 한 바이트로 구성되어 있기 때문에 바이트코드라고 부른다. 바이트코드는 하나의 클래스 단위로 이루어져있다[5,7]. (그림 2)는 클래스 파일의 구조이다.

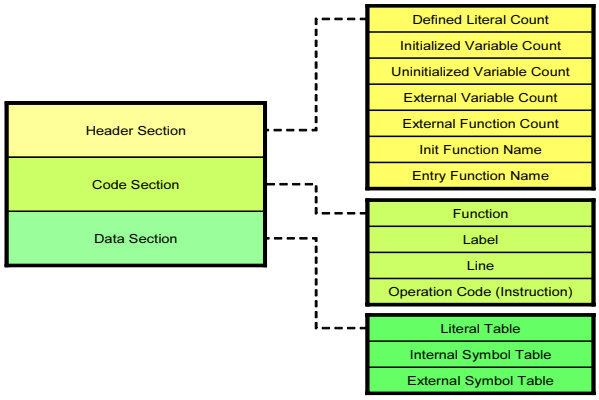


(그림 2) 클래스 파일 구조

자바 클래스 파일임을 나타내주는 Magic Number를 시작으로 클래스 파일 정보가 구성된다. 상수 풀(Constant Pool)은 그 클래스에서 사용되는 상수를 담아 놓은 일종의 메모리 공간이다. 리터럴(Literal), 클래스 이름, 메소드 이름 등이 포함된다. 다음으로 This 클래스, Super 클래스, 인터페이스 정보를 담고 있고, 이어서 필드와 메소드 정보를 담는다. 필드, 메소드, 클래스의 정보는 속성(Attribute)을 통해 정의된다[1,2,3,4].

2.3 SAF(Standard Assembly Format)

SAF는 유비쿼터스 게임 플랫폼을 위한 중간언어 형식으로서 순차적인 언어와 객체지향 언어를 모두 수용할 수 있도록 설계되었다. SAF는 헤더 영역, 코드 영역, 데이터 영역의 3가지 영역으로 구성되어 있다. 헤더 영역은 SAF 파일을 구성하는 주요 정보를 표현한 부분이다. 코드 영역은 함수들이 모여 구성된다. 각각의 함수는 지역공간과 연산코드로 구성된다. 클래스 메소드도 함수와 동일하게 정의한다. 데이터 영역은 프로그램의 데이터를 표현하는 부분으로서 코드 영역에서 참조된 전역 변수와 리터럴(Literal)을 표현한다[8]. (그림 3)은 SAF의 구조이다.



(그림 3) SAF의 구조

3. Java Bytecode-to-SAF 번역기 시스템

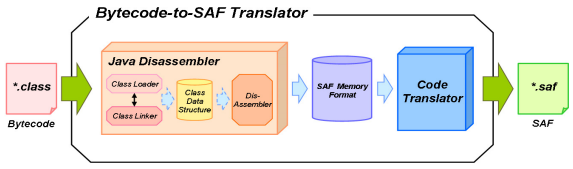
3.1 시스템 개요

Java Bytecode-to-SAF 번역기는 유비쿼터스 게임 플랫폼에서 자바 프로그램을 실행시키기 위해 JVM(Java Virtual Machine)의 중간코드인 바이트코드를 입력으로 받아 유비쿼터스 게임 플랫폼의 가상기계, u-VM(Ubiquitous Virtual Machine)의 중간언어 형식인 SAF(Standard Assembly Format)로 변환한다.

3.2 시스템 구성

Java Bytecode-to-SAF 번역기 시스템은 자바 디스어셈블러(Java Disassember)와 코드 변환기(Code Translator), 그리고 SAF의 메모리 구조로 구성된다. 자바 디스어셈블러는 바이너리 형태의 클래스 파일을 분석하여 클래스 자

료 구조를 생성하고, 이를 SAF의 메모리 구조로 변환한다. 이 과정에서 SAF의 헤더 영역(Header Section)과 데이터 영역(Data Section)이 구성된다. 코드 변환기에서는 바이트코드의 명령어(Instruction)를 SIL(Standard Intermediate Language) 코드 형식으로 변환하여 SAF의 코드 영역(Code Section)을 구성한다. (그림 4)는 Java Bytecode-to-SAF 번역기의 구성도이다.



(그림 4) Java Bytecode-to-SAF 번역기 구성도

3.3 시스템 구현

자바 디스어셈블러는 클래스 로딩(Class Loading), 클래스 링킹(Class Linking), 그리고 디스어셈블링(Disassembling) 과정을 거쳐 바이트코드의 번역을 수행한다. 클래스 로딩 과정에서는 바이너리 형식의 *.class 파일을 읽어 클래스 파일 구조에 맞추어 정보를 추출한다. 그리고 클래스 링킹 과정에서는 읽어 들인 클래스에서 다른 클래스가 사용되었는지 조사하고, 필요한 클래스를 클래스 로더를 통해 읽어 들여 변환에 필요한 자료 구조를 생성한다. 디스어셈블링 과정에서는 링킹 과정을 통해 생성된 클래스 구조를 SAF 형식으로 변환한다. 클래스 구조의 상수 풀(Constant Pool), 필드 정보 등을 이용해 SAF의 헤더 영역, 데이터 영역을 구성한다. 또한, 코드 변환 과정에서 사용할 필드의 오프셋(Offset) 값을 구한다. 바이트코드의 경우 필드를 사용할 때 상수 풀의 인덱스를 통해, 이름과 기술자(Descriptor)로 접근하지만 SAF의 경우 오프셋을 통해 접근한다.

클래스 파일의 명령어는 메소드 정보(Method Information)의 코드 속성(Code Attribute) 부분에 정의되어 있다. 코드 변환기는 이 코드 속성을 분석해 SAF의 명령어 코드인 SIL 코드로 변환한다. 코드 변환은 2번의 변환 과정을 거친다. 1차 변환 과정에서는 일대일 맵핑(Mapping)이 가능한 바이트코드의 명령어를 SIL 코드로 변환한다. 대부분의 연산 명령어와 형 변환 명령어가 이 과정에서 변환된다. 또한, 지역 변수의 베이스-오프셋(Base-Offset) 테이블과 명령어 수행 후의 스택 값을 구성하여, 2차 변환에서 사용되도록 한다. 2차 변환 과정에서는 일대일 맵핑으로는 변환할 수 없는 바이트코드 명령어를 의미적으로 동등한 형태의 SIL 코드로 변환한다. 슬롯 형식의 Load, Store 명령어를 베이스-오프셋 형식의 SIL 코드로 변환하고, 분기 명령어, 객체 관련 명령어 등을 변환한다. <표 1>은 코드 변환 테이블의 일부분이다.

<표 1> 코드 변환 테이블

Bytecode			SIL code		
pop			pop		
iadd			add.i		
ixor			bxor.i		
i2l			cvi.l		
iload	slot		lod.i	base	offset
istore	slot		str.i	base	offset
ifeq	offset		ldc.i eq.i tjp	0 ##label	
객체 참조자 load *저장 값 load* putfield	index		*객체 참조자 load* ldc.p add.p *저장 값 load* sti.i	offset	
객체 참조자 load *argument load* invokevirtual	index		ldp *객체 참조자 load* *argument load* call	 &name	

4. 실행 결과 및 분석

자바 언어로 작성된 "테트리스" 게임 프로그램을 본 논문에서 구현한 Java Bytecode-to-SAF 번역기를 통해 변환한 후 유비쿼터스 게임 플랫폼의 가상기계인 u-VM(Ubiquitous Virtual Machine)에서 실행해 보았다. (그림 5)는 자바 언어로 작성된 프로그램 소스이다. 본 예제 프로그램에서 사용된 라이브러리는 u-VM에서 지원하는 내장 라이브러리이다.

```
class Game {
    int board[][] = new int[16][9];
    int block_x;
    int block_y;
    int down_speed;
                        (중간생략)

    public void main() {
        x= bc[1 + i] + block_x;
        y= bc[1 + i + 1] + iy;
        board[x][y] = bc_cur + 1;
        board[iy - 1][ix] = board[iy][ix];
        board[16 - 1][ix] = Rand(1, (7 + 1));
        initWindow();
        ClearWhite();
        _DrawText(20, 10, "테트리스 게임");
        _DrawStr(10, 100, "- ←,↑/↓ C,→ +");
        _DrawText(20, 120, "SPEED:");
        _SetTimer(80, 1);
                        (중간생략)

    void block_down() {
        flag = 0;
        for (i = 0; i < 8; /*i += 2*/ i = i + 2) {
            x = bc[1 + i] + block_x;
            y = bc[1 + i + 1] + block_y + 1;
            if (y < 0)
                continue;
            if (y >= 16) {
                flag = 1;
                break;
            }
                        (중간생략)
```

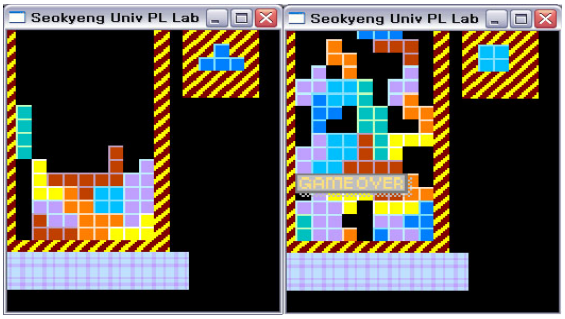
(그림 5) 자바 언어로 작성된 테트리스 소스

(그림 6)은 자바 컴파일러로 컴파일 한 후 Java Bytecode-to-SAF 번역기를 통해 변환한 SAF 코드이다.

```
%%HeaderSectionStart
    %DefinedLiteralCount            4
    %InitializedVariableCount       0
    %UninitializedVariableCount     0
    %ExternalVariableCount          0
    %ExternalFunctionCount          0
    %InitFunctionName               0
    %EntryFunctionName              &Game::main_()V
%%HeaderSectionEnd
%%CodeSectionStart
    %FunctionStart
        .func_name      &Game::main_()V
        .func_type      0
        .param_count    1
        .opcode_start
            proc        20          1           1
    %Line 170
            lod.p       1           0
            ldc.p       96
            add.p
            lod.p       1           0
            ldc.p       12
            add.p
            ldi.p
                        (중간생략)
    %FunctionStart
        .func_name      &Game::block_down_()V
        .func_type      0
        .param_count    1
        .opcode_start
            proc        8           1           1
            str.p       1           0
    %Line 728
            lod.p       1           0
            ldc.p       112
            add.p
            ldc.i       0
            sti.i
                        (중간생략)
%%CodeSectionEnd
%%DataSectionStart
    %LiteralTableStart
        .literal_start  @0      1       20
            0x0FFFFFFED, 0x0FFFFFF85, ,,,,,,(생략)
        .literal_end
                        (중간생략)
    %LiteralTableEnd
    %InternalSymbolTableStart
    %InternalSymbolTableEnd
    %ExternalSymbolTableStart
    %ExternalSymbolTableEnd
%%DataSectionEnd
```

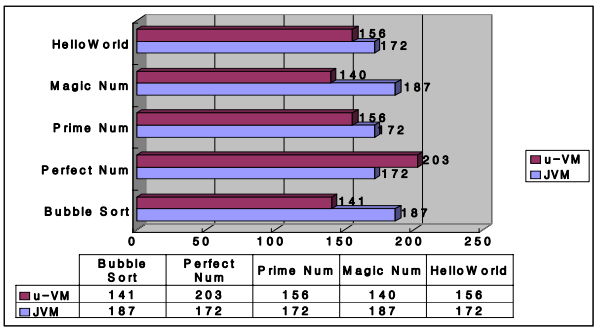
(그림 6) 변환된 SAF 코드

자바 콘텐츠를 SAF 어셈블러로 어셈블한 후 생성된 SEF를 u-VM을 통해 실행해 보았다. (그림 7)과 같이 자바 언어로 작성된 프로그램이 클래스 파일로 생성된 후 SAF로 변환되어 u-VM에서 정상적으로 실행되는 것을 확인하였다.



(그림 7) u-VM 실행화면

(그림 8)은 자바 바이트코드와 이를 Java Bytecode-to-SAF 번역기를 통해 변환한 SAF에 대한 실행속도를 비교한 자료이다. 전반적으로 자바 바이트코드를 실행하는 것보다 빠른 실행 속도를 보이는 것을 알 수 있다.



(그림 8) 바이트코드와 SAF의 실행속도 비교

5. 결론 및 향후연구

본 논문에서는 유비쿼터스 게임 플랫폼에서 C/C++ 콘텐츠뿐만 아니라 자바 콘텐츠도 지원하기 위해 Java Bytecode-to-SAF 번역기를 설계하고 구현하였다. Java Bytecode-to-SAF 번역기를 통해 자바 바이트코드를 SAF로 변환하여, 유비쿼터스 게임 플랫폼의 가상기계인 u-VM에서 실행이 가능해졌다. 이로 인해 개발자에게 효율적이고 능률적인 개발 환경을 제공하고, 콘텐츠의 생산성을 높일 수 있다.

앞으로 자바 플랫폼 프로그래밍 언어의 API 지원을 위해 번역기를 확장하고, 보다 효과적인 코드를 낼 수 있는 코드 최적화를 위한 연구를 수행할 예정이다.

참고문헌

[1] Bill Venners, "Inside JAVA Virtual Machine", Second Edition, McGraw-Hill, 1999.

[2] Hoshua Engel, "Programming for the Java Virtual Machine", Addison-Wesley, 1999.

[3] John Meyer & Troy Downing, "Java Virtual Machine", O'REILLY, 1997.

[4] Tim Lindholm & Frank Yellin, "The Java™ Virtual Machine Specification", Second Edition, Addision-Wesley, 1997.

[5] Yang-Sun Lee, "Java Bytecode-to-.NET MSIL Translator for Construction of Platform Independent Information Systems," LNAI 3215, Vol.3, ISSN 0302-9743, Springer, pp.726-732, Sep 2004.

[6] 최홍석, 김영근, 권혁주, 이양선, "유비쿼터스 게임 플랫폼을 위한 임베디드 가상기계의 설계 및 구현", 한국정보처리학회 2006 추계학술발표논문집, Vol.14, No.2, pp.925-928, Nov, 2006.

[7] 이양선, 박진기, "Translation Java Bytecode to EVM SIL Code for Embedded Virtual Machine," 한국멀티미디어학회 논문지, Vol.8, No.12, pp.1658-1668, Dec 2005.

[8] 서경대학교 프로그래밍 언어 연구실, Standard Assembly Format 규격, Nov, 2006.