유비쿼터스 게임 플랫폼을 위한 SAF 어셈블러의 설계 및 구현

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

Design and Implementation of a SAF Assembler for the Ubiquitous Game Platform

Hong-Suck Choi*, 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 유비쿼터스 환경의 기기에 탑재되어 콘텐츠들을 C/C++/Java 언어 모두를 수용하며 가상기계 방식이어서 유비쿼터스 환경의 기기에 독립적으로 실행이 가능하다는 장점을 가지고 있다.

본 논문에서는 유비쿼터스 게임 플랫폼의 가상기계인 유비쿼터스 가상기계(u-VM)의 입력인 SEF(Standard Executable Format) 실행파일을 생성하기 위한 SAF 어셈블러를 설계 및 구현 하였다.

SAF 어셈블러는 중간언어 형식인 SAF(Standard Assembly Format)를 입력으로 받아 u-VM이 실행 가능한 파일 형식인 SEF를 생성한다. SEF 파일을 생성하면서 SAF 어셈블러는 여러개의 SAF파일을 한 개의 SEF로 묶어주는 링커의 역할을 하여 주며, 실행 파일을 텍스트 형식의 SAF에서 바이너리 형식의 SEF로 바꿈으로써 보안상의 문제를 해결해 준다. 또한 SEF는 SAF에 비해 가볍게 변환이 되어 가상기계에서의 실행 속도를 개선하는 역할을 하여 준다.

1. 서론

현 사회에서 핸드폰은 물론 PMP와 PDA 같은 임베디드 기기들을 가지고 다니는 현대인들이 부지기수로 늘고 있다.

이런 유비쿼터스 환경의 임베디드 기기들의 선풍적인 인기에 따라 임베디드 기기 내에서 구동되는 각종 S/W 콘텐츠들 또한 각광을 받고 있다.

그러나 현재 임베디드 콘텐츠들의 질이나 양은 PC환경의 그것에 비해 현저히 떨어지는 것이 현실이며 그것의 개발 환경도 매우 열약하다.

각 콘텐츠들은 목적 기기에 따라 재개발되어야 하는 비효율 적인 방식의 개발 환경이 구축 되어 있으며 각 기기들에 따라 개발 언어를 마음대로 고를 수 없다.

본 연구는 문화관광부 및 한국문화콘텐츠진흥원의 '06문화콘텐츠기술(CT)개발지원사업의 연구결과로 수행되었음

핸드폰의 개발환경에 경우 각 개발 언어의 서브셋만을 지원하는 개발 환경으로 인해 품질 좋은 콘텐츠의 제작이 힘들기도 한다.

본 논문에서 제안하는 유비쿼터스 게임 플랫폼은 유비쿼터스 환경의 임베디드 기기들을 위한 게임 플랫폼으로써 위에 열거한 열약한 임베디드 콘텐츠 환경을 크게 개선할 수 있게 하여 준다.

유비쿼터스 게임 플랫폼은 가상 기계 기반의 방식을 사용하는데 IDE 에뮬레이터나 각종 임베디드 기기에 탑재되는 이 가상 기계들은 SEF(Standard Executable Format)라는 실행 파일 포맷을 가진 파일들을 입력으로 콘텐츠를 실행한다. SAF 어셈블러는 본 연구팀이 개발한 컴파일러나 번역기 시스템을 통해 나오는 SAF(Standard Assembly Format)를 입력으로 받아 유비쿼터스 게임 플랫폼의 가상기계인 u-VM의 입력으로 사용하는 SEF를 생성하기 위

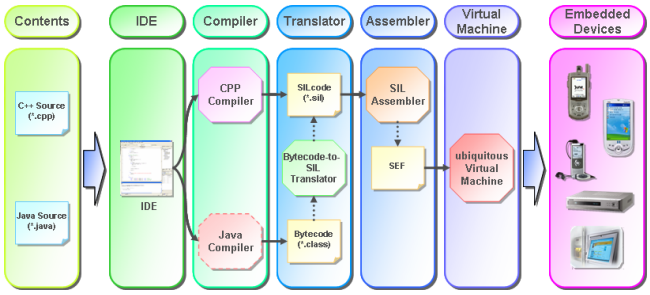
해 제작되어진 모듈이며 유비쿼터스 게임 플랫폼에서 로딩 과정과 링킹 과정을 수행하고 실행파일의 보안을 위해서 실행되는 모듈이다.

2. 관련 연구

2.1 유비쿼터스 게임 플랫폼

유비쿼터스 게임 플랫폼은 본 연구팀이 개발한 유비쿼터스 환경의 임베디드 기기들을 위한 게임 플랫폼 이다.

중간 언어를 사용하는 가상 기계 기반의 플랫폼이어서 여러 임베디드 기기에 독립적으로 수행이 가능하며 중간언어인 SAF는 순차적 언어인 C와 객체지향 언어인 C++, Java를 모두 수용할 수 있어서 세 가지 언어를 모두 사용할 수 있는 장점을 가진다.



(그림 1) 유비쿼터스 게임 플랫폼의 구조

유비쿼터스 게임 플랫폼의 구조도는 (그림 1)과 같다. 유비쿼터스 게임 플랫폼은 컴파일 부인 전단부와 실행 부인 후단부로 나뉜다.

전단 부는 u-VM이 실행 가능한 SEF(Standard Executable Format)를 출력하는 부분으로써 IDE를 이용하여 C, C++, Java 언어를 이용해 콘텐츠를 제작할 수 있는 환경을 제공해 준다.

제작된 콘텐츠들은 C, C++의 경우 CPP 컴파일러를 통해 중간언어인 SAF를 출력으로 내고, Java의 경우 Sun사의 Java 컴파일러를 통해 바이트 코드를 낸 후 SAF로 출력하기 위해 Bytecode-to-SAF 번역기를 이용하여 SAF를 출력한다.

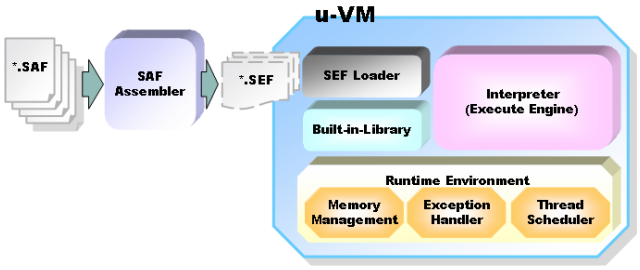
출력된 SAF는 로더와 링커의 역할을 하는 SAF 어셈블러를 통해 u-VM이 실행 가능한 SEF으로 변환된다.

이렇게 출력된 SEF는 후단부에 있는 u-VM에서 실행을 하게 된다. VM이 탑재된 IDE의 경우 에뮬레이터에서의 실행을 담당하게 되며 기기 내 탑재된 u-VM의 경우 실제 임베디드 기기 내에서 실행이 된다.[5]

2.2 u-VM (ubiquitous-Virtual Machine)

u-VM은 유비쿼터스 게임 플랫폼의 실행을 담당하며 H/W의 기능을 제공하는 S/W로써 u-VM이 탑재되는 플랫폼이라면 어디서든지 u-VM의 콘텐츠를 실행할 수 있게 하는 플랫폼에 독립적인 특징을 가진다.

u-VM은 순차적 언어와 객체지향 언어를 모두 수용하는 중간 언어인 SAF를 SAF 어셈블러에 의해 변환한 SEF를 입력으로 받아 사용함으로써 개발 언어에 구애받지 않는 장점을 가지게 하여 준다.



(그림 2) u-VM의 구조

u-VM의 구성은 (그림2)와 같이 SEF 로더와 실제 실행 엔진인 인터프리터, 내장 라이브러리, 실행 환경으로 구성되어 있다.[5]

내장 라이브러리에는 그래픽 라이브러리와 ANSI C 라이브러리, Java 라이브러리 등을 탑재하여 사용하고 있으며 언어에 상관없이 각자의 라이브러리를 서로 공유할 수도 있다.

실행환경에서는 메모리를 관리하여 주고 예외처리기를 통해 예외처리를 가능하게 하여 주며 스레드 스케줄러를 통해 멀티태스킹을 가능하게 하는 개발 환경을 제공하여 준다.

3. SAF Assembler

3.1 SEF(Standard Executable Format)

SEF는 본 연구팀에서 고안한 실행 파일 포맷 이다.

유비쿼터스 게임 플랫폼의 가상기계인 u-VM의 입력으로 들어가서 실행되며 SAF 어셈블러의 출력으로 생성된다.

SEF는 임베디드 콘텐츠의 사용자가 소스의 정보를 알아 볼 수 없도록 바이너리 형식으로 되어 있으며 형식은 (그림 3)과 같다.

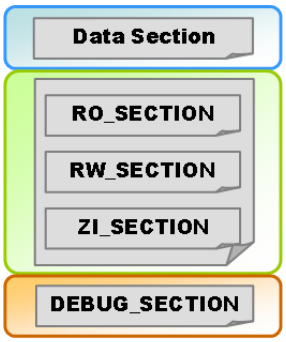
Data_Section에는 프로그램의 엔트리 정보와 초기화 함수의 Index정보 그리고 각 Section의 크기를 정의하고 있으며 RO_Section은 각 연산코드들과 연산코드에 대응되는 파라메터, 리터럴 등의 정보를 가지고 있다.

RW_Section은 초기화 값을 가지고 있는 전역 변수에 대한 정보들을 가지고 있으며 ZI_Section은 초기화 값을

가지지 않는 전역 변수에 대한 정보들을 가지고 있다.

DEBUG_SECTION은 실제 u-VM의 실행에는 직접적인 관여를 하지 않으며 디버거의 실행에 연관이 있는 영역이다.

DEBUG_SECTION에는 기존 소스코드의 디버깅에 관한 정보를 담고 있으며 본 논문에서 구현될 SAF 어셈블러는 이 정보를 따로 파일로 분류하여 출력할 것이다.[7]



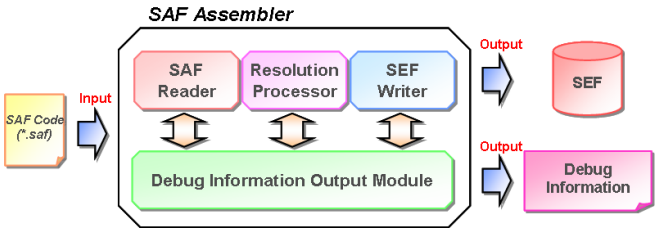
(그림 3) SEF의 형식

3.2 SAF 어셈블러의 구성

SAF 어셈블러는 u-VM의 실행을 위한 파일 포맷인 SEF를 출력으로 내기 위한 모듈이다.

SAF 어셈블러는 크게 텍스트형식인 SAF를 로딩 하여 바이너리 형식인 SEF로 바꾸는 기능과 여러 개의 SAF를 묶어서 각 심벌들을 바인딩 하여주는 링킹의 기능, 디버깅 정보를 출력하는 기능을 가지고 있다.

SAF 어셈블러의 구조는 (그림 4)와 같다.



(그림 4) SAF 어셈블러의 구조

SAF 어셈블러는 SAF Reader와 Resolution Processor, SEF Writer, Debug Information Output Module로 구성되어 있다.

SAF Reader는 한 개 이상의 SAF 파일을 읽어서 SAF 어셈블러의 자료구조에 적재하는 역할을 하여 주며 자료구조에 적재된 연산코드와 의사코드들의 정보는 Resolution Processor에 의해 같은 이름을 가진 심벌이나 레이블을 독립적인 이름으로 변경시켜 주는 등의 작업을 거쳐 하나로 된 SEF의 형식으로 합쳐 준다.[1,6]

그리고 함수 호출, 흐름 제어 명령어에 쓰이는 레이블 정보는 실제 메모리 상의 주소를 계산하여 각 레이블에 해당하는 인덱스로 변환하여 준다.

또한 함수 호출의 경우는 보통 함수 호출이나 스레드 함수 호출의 정보를 VM에서 구별할 수 있는 정보를 표현하여 준다.

각 자료구조에 적재되어 있는 SEF구조는 SEF Writer에 의해 실제 SEF 파일을 만들게 되며 이는 SEF의 포맷에 맞추어 바이너리 형식으로 쓰여 진다.

Debug Information Output Module은 입력된 SAF파일을 SAF Reader, Resolution Processor, SEF Writer가 수행되는 동안 수행하여 Line정보나 Label 정보 등을 메모리에 적재하는 역할을 한다.

Debug Information Output Module은 아직 완전히 구현이 되어 있지 않으며 이는 디버거의 구현과 함께 디버깅 정보의 확장 등을 더 고려하면서 구현을 하려 한다.

4. 실험 결과 및 분석

SAF 어셈블러는 C++언어로 Window XP환경의 Visual Studio .NET 2003 환경에서 제작되었다.

제작된 SAF 어셈블러의 제대로 된 검증을 위한 실험으로 본 논문에서는 서경대학교 PL 연구실에서 제작된 CPP 컴파일러와 u-VM, u-VM의 라이브러리를 통해 C++언어로 제작된 임베디드 게임 엘리멘탈 포스를 사용 하였다.

다음은 엘리멘탈 포스라는 게임의 소스코드를 입력으로 넣어 CPP 컴파일러를 통해 출력된 SAF 중간언어의 모습이다.

<표1> ElementalForce.cpp.saf

```
%%HeaderSectionStart
   %DefinedLiteralCount          59
   %InitializedVariableCount     103
   %UninitializedVariableCount   53
   %ExternalVariableCount        2
   %ExternalFunctionCount        0
   %InitFunctionName             0
   %EntryFunctionName            &main
   %SourceFileName  ex\elemental\Elemental.cpp
%%HeaderSectionEnd
%%CodeSectionStart
   %FunctionStart
      .func_name        &GetScore
      .func_type        2
      .param_count      1
      .opcode_start
            proc        4           1           1
            str.i       1           0
            lod.i       1           0
            ldc.i       0
            ge.i
            fjp         ##0
            lda         0           $g_Score
            ldc.i       0
중간생략
            ujp         ##1056
      %Label            ##1057
            ret
      .opcode_end
   %FunctionEnd
%%CodeSectionEnd
%%DataSectionStart
   %LiteralTableStart
      .literal_start    @0          0           20
0x50,0x45,0x54,0x5f,0x53,0x54,0x41,0x54,0x55,0x53,0x5f,0x44,0
x45,0x4d,0x41,0x47,0x45,0x5c,0x6e,0x00
      .literal_end
중간생략
      .var_start        $GMenuName3         0           9
```

```
            9           1           (1)
            0x53,0x43,0x45,0x4e,0x41,0x52,0x49,0x4f,0x00
        .var_end
        .var_start      $GMenuName4     0           8
            8           1           (1)
            0x52,0x41,0x4e,0x4b,0x49,0x4e,0x47,0x00
        .var_end
        .var_start      $GMenuName5     0           7
            7           1           (1)
            0x4f,0x50,0x54,0x49,0x4f,0x4e,0x00
        .var_end
        .var_start      $swData   0           4           0
        .var_end
        .var_start      $frameCnt 0           4           1
            1           (4)
            0x00000000
        .var_end
        .var_start      $swFrame2       0           4
            0
        .var_end
    %InternalSymbolTableEnd
    %ExternalSymbolTableStart
        .evar_decl      $bgiemu_handle_redraw
        .evar_decl      $bgiemu_default_mode
    %ExternalSymbolTableEnd
%%DataSectionEnd
```

```
00000000h: 72 47 01 00 5A 4A 01 00 D7 4E 01 00 82 C0 FE FF ; rG..ZJ..?..弼?
00000010h: FF FF FF FF 7A 45 01 00 A4 00 04 00 00 00 01 00 ;         zE..?.....
00000020h: 01 00 2B 00 01 00 00 00 00 00 13 00 01 00 00 00 ; ..+.............
00000030h: 00 00 09 00 00 00 00 00 60 00 93 00 EE 00 00 00 ; ........`.??..
00000040h: 27 00 00 00 11 0D 00 00 09 00 00 00 00 00 AB 00 ; '.............?
00000050h: B1 00 3E 00 27 00 00 00 11 0D 00 00 09 00 00 00 ; ?>.'...........
00000060h: 00 00 AB 00 B1 00 3E 00 1F 00 13 00 01 00 00 00 ; ..??>.........
00000070h: 00 00 3B 00 34 00 27 00 00 00 11 0D 00 00 09 00 ; ..;.4.'.........
00000080h: 00 00 00 00 AB 00 B1 00 3E 00 1F 00 09 00 10 27 ; ....??>......'
00000090h: 00 00 60 00 93 00 EE 00 00 00 27 00 00 00 11 0D ; ..`.??..'.....
000000a0h: 00 00 09 00 04 00 00 00 AB 00 B1 00 3E 00 1F 00 ; ........??>...
000000b0h: 03 00 09 00 01 00 00 00 3B 00 27 00 00 00 11 0D ; ........;.'.....
000000c0h: 00 00 09 00 04 00 00 00 AB 00 B1 00 3E 00 05 00 ; ........??>...
000000d0h: 34 00 27 00 00 00 11 0D 00 00 09 00 00 00 00 00 ; 4.'.............
000000e0h: AB 00 B1 00 3E 00 27 00 00 00 11 0D 00 00 09 00 ; ??>.'.........
000000f0h: 00 00 00 00 AB 00 B1 00 3E 00 1F 00 09 00 10 27 ; ....??>......'
00000100h: 00 00 41 00 34 00 95 00 A4 00 14 00 00 00 01 00 ; ..A.4.??......
00000110h: 01 00 2B 00 01 00 10 00 00 00 2B 00 01 00 0C 00 ; ..+.......+.....
```

(그림 5) ElementalForce.sef

이 SAF를 이용하여 SAF 어셈블러를 구동시키면 (그림 5)와 같은 바이너리 형식의 SEF 파일이 생성된다. 바이너리 형식의 SEF파일을 눈으로 검증하기는 힘이 들기 때문에 u-VM의 입력으로 이 SEF파일을 입력하여 제대로 된 게임이 동작 되는지 확인하여 보았다. (그림 6)과 같이 엘리멘탈 포스라는 게임이 u-VM에서 제대로 동작하는 것을 확인할 수 있었다.

(그림 6) Elemental Force의 실행 모습

5. 결론 및 향후 연구

본 논문에서는 SAF 어셈블러를 설계하고 구현하였다. SAF 어셈블러의 구현으로 바이너리 형식인 SEF를 실행파일로 사용함으로써 텍스트 형식으로 되어 있는 SAF를 바로 실행파일로 사용하던 기존의 형식보다 보안이 개선되었으며 여러 개의 SAF 파일을 읽어 하나의 SEF로 구성하는 링커의 기능을 수행할 수 있게 되었다. 또한 SAF에 비해 상대적으로 가벼운 SEF를 실행파일로 쓰게 됨으로써 u-VM의 실행속도가 더 빨라졌으며 저장 공간을 덜 차지하게 되는 이점을 갖게 되었다.

향후 SAF 어셈블러에서 보안을 더 강화하기 위해 SAF 어셈블러에서의 조작을 통해 암호화를 할 수 있게 할 것이며 아직 부족한 디버깅 정보에 관한 모듈 또한 재설계하고 구현해야 할 것이다.

참고문헌

[1] John R. Levine "Linkers and Loaders", Morgan Kaufmann Publishers, 2000

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

[3] Joshua Engel, "Programming for the Java™ Virtual Machine", Addison-Wesley, 1999

[4] Alfred V.Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers Principles, Techniques, and Tools", Addison-Wesley, 1986

[5] 최홍석, 김영근, 권혁주, 이양선, "유비쿼터스 게임 플랫폼을 위한 임베디드 가상기계의 설계 및 구현", 한국정보처리학회 추계학술발표대회 논문집(하), 제13권 제2호 pp.925-928, 2006년 11월

[6] 서경대학교 Programming Language Lab., Standard Assembly Format 규격, 2006.

[7] 서경대학교 Programming Language Lab., Standard Executable Format 규격, 2006.

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