제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

퍼베이시브 컴퓨팅을 위한 가상기계의 어셈블러

전병준[0], 이창환, 오세만
동국대학교 컴퓨터공학과
e-mail : {junniya[0], yich, smoh}@dongguk.edu

Assembler of Virtual Machine for Pervasive Computing

Byung-Jun Jeon[0], Chang-Hwan Yi, Se-Man Oh
Dept of Computer Engineering, Dongguk University

요 약

최근 유·무선 통신 기술이 발전하고 소규모 장치의 개발 기술이 향상되면서 퍼베이시브 컴퓨팅 환경에 대한 관심이 높아지고 있다. 퍼베이시브 컴퓨팅 환경에서는 다양한 생활용품에 장착 가능한 소규모 장치와 센서 등을 사용한다. 소규모 장치와 센서들은 다양한 기기로 구성되어 있기 때문에 개발 환경과 실행 환경의 호환성이 낮은 문제점을 가지고 있다. 이는 가상기계 플랫폼을 적용하여 해결할 수 있다. 기존에 개발된 가상기계는 규모가 크고 높은 컴퓨팅 파워를 요구하기 때문에 퍼베이시브 환경에는 사용할 수 없다. 그러므로 퍼베이시브 환경에 적합한 가상기계의 어셈블러가 필요하다.

본 논문에서는 임베디드 시스템을 위한 가상기계인 EVM의 어셈블러를 수정하여 퍼베이시브 컴퓨팅 환경에 적합한 어셈블러를 설계하고 구현한다. 적은 리소스만을 제공하는 소규모 가상기계에 적합하도록 EVM의 객체 지향 특성과 불필요한 명령어를 제거한다. 수정된 새로운 가상기계 플랫폼을 위한 어셈블러를 통해서 가상기계에서 실행 가능한 실행 파일 포맷을 생성할 수 있다.

1. 서론

최근 유·무선 통신 기술이 발전하고 소규모 장치의 개발 기술이 향상되면서 퍼베이시브 컴퓨팅 (Pervasive Computing) 환경에 대한 관심이 높아지고 있다. 퍼베이시브 컴퓨팅이란 언제 어디서나 아무런 제약 없이 각종 기기들을 사용할 수 있는 환경을 말한다[1].

퍼베이시브 컴퓨팅 환경 구축에 필요한 소규모 장치들은 서로 다른 프로세서와 다양한 크기의 메모리로 구성되어 개발 환경이나 실행 환경의 호환성이 낮다. 이런 단점은 가상기계 플랫폼을 적용하여 해결할 수 있다.

기존에 구현된 가상기계인 EVM (Embedded Virtual Machine) 이나 JVM (Java Virtual Machine) 같은 가상기계 플랫폼은 크기가 크고, 높은 컴퓨팅 파워를 요구하기 때문에 퍼베이시브 컴퓨팅 환경에 적합하지 않다. 따라서 퍼베이시브 컴퓨팅 환경에 적합한 가상기계 플랫폼이 필요하다.

본 논문에서는 기존에 개발된 임베디드 시스템을 위한 가상기계 환경인 EVM을 수정하여 퍼베이시브 컴퓨팅 환경에 적합한 가상기계 플랫폼을 위한 어셈블러를 설계하고 구현한다[1][2].

소규모 환경에 적합한 가상기계 플랫폼을 구현하기 위하여 EVM의 객체 지향 특성을 제거하고 불필요한 명령어를 축약하여 소규모 가상기계에 적합하도록 수정한다.

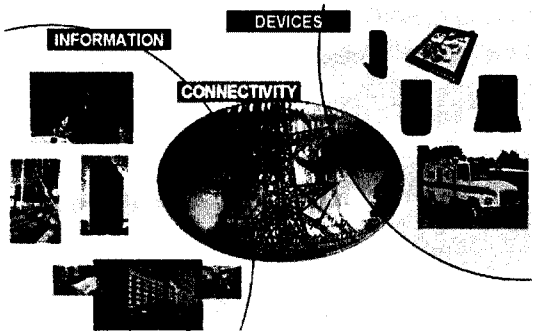
수정된 새로운 가상기계의 플랫폼을 위한 어셈블러를 통해서 가상기계에서 실행 가능한 실행 파일 포맷을 생성할 수 있다.

2. 배경연구

2.1 퍼베이시브 컴퓨팅 (Pervasive Computing)

퍼베이시브 컴퓨팅이란 언제 어디서나 아무런 제약 없이 각종 기기들을 사용할 수 있는 환경을 말한다. 이런 환경은 시간과 장소에 상관없이 원하는 정보에 간편하게 접근할 수 있게 된다[1].

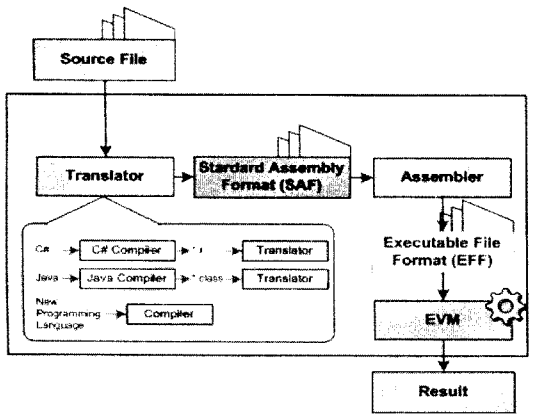
퍼베이시브 컴퓨팅 환경 구축에 필요한 장비들은 환경에 맞는 프로세서나 메모리 등을 가지고 있다. 이런 환경은 일상생활과 개방형 표준에 기반을 둔 애플리케이션을 연결하여 생활을 더욱 편리하게 하는 것을 말한다. (그림 1)은 퍼베이시브 컴퓨팅 환경에 대한 그림이다[3].



(그림 1) 퍼베이시브 컴퓨팅 환경

2.2 EVM (Embedded Virtual Machine)

임베디드 시스템을 위한 가상기계인 EVM은 모바일 장비, 셋톱박스, 디지털 TV등에 탑재되어 동적 응용 프로그램을 다운로드 하여 실행 할 수 있는 가상기계 솔루션이다. (그림 2)는 EVM의 구조를 나타낸다[3].



(그림 2) EVM의 구조

EVM의 표준 중간 언어인 SIL (Standard Intermediate Language) 은 스택 기반의 명령어 집합이다. 언어의 독립성과 하드웨어, 플랫폼 독립성을 가지고 있으며 다양한 프로그래밍 언어를 수용하기 위하여 바이트코드, .NET IL 등 기존의 가상기계 코드들의 분석을 토대로 정의하였다. SIL은 객체지향 언어와 순차지향 언어를 모두 수용하기 위한 연산코드 집합을 가지고 있다[4].

EVM의 표준 어셈블리 포맷인 SAF (Standard Assembly Format) 는 의사코드와 연산코드로 구성된다. SAF는 클래스 기반 구조를 가지며 SAF 어셈블러의 입력 파일이다. 어셈블러를 통해서 EVM에서 실행되는 EFF (Executable File Format) 를 생성한다[4].

EVM을 위한 실행 파일 포맷인 EFF는 8비트 스트림으로 구성되어 있다. 하위 바이트들이 먼저 저장되는 스몰-엔디언 (small-endian) 방식으로 저장된다. EFF 파일은 헤더, 코드, 메타데이터의 3가지 부분으로 나뉜다. 헤더 부분은 EFF 파일의 버전, 프로그램의 이름, 시작점 등의 정보를 포함한다. 코드 부분은 실질적인 명령어들을 포함하고 있다. 메타데이터는 실행에 필요한 모든 정보를 저장한다[5][6].

3. 가상기계를 위한 어셈블러

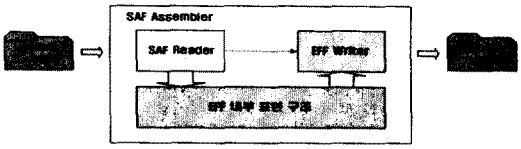
본 논문에서는 기존에 구현된 EVM의 어셈블러를 수정하여 퍼베이시브 컴퓨팅 환경에 적합한 어셈블러를 설계하고 구현한다. 기존 EVM의 객체 지향 특성을 제거하고 불필요한 명령어를 제거한다.

그러나 EVM과 호환을 위해서 객체 지향 기능은 삭제하되 프로그램의 단위로서 클래스 기능은 유지하도록 한다. 이 장에서는 어셈블러의 전체적인 구조에 대하여 설명한다.

3.1 어셈블러의 구성

퍼베이시브 컴퓨팅 환경을 위한 어셈블러는 SAF 리더, EFF 내부 표현 구조, EFF 라이터로 구성된다. 어셈블리 파일 (SAF) 을 입력으로 받은 SAF 리더는 SAF 파일의 데이터를 실행 파일 포맷의 데이터로 변경하여 EFF 내부 표현 구조에 저장한다. EFF 라이터는 EFF 내부 표현 구조에서 필요한 데이터를 가지고 온 후, 바이너리 파일을 생성한다. 생성된 이진 파일은 가상기계에서 실행되는 실행 파일이다.

퍼베이시브 컴퓨팅을 위한 가상기계의 어셈블러는 (그림 3)과 같은 구조를 가진다.



(그림 3) 가상기계를 위한 어셈블러의 구조

3.2 SAF 문법 축약

기존에 구현된 EVM을 소규모 가상기계 플랫폼에 맞도록 수정하면서 객체 지향 기능과 불필요한 기능을 삭제하였다. 그러나 기존 EVM과의 호환성을 위해서 객체 지향 기능은 삭제하되 클래스 기능은 유지하도록 하였다.

수정된 부분을 토대로 어셈블리 문법을 축약하여 소규모 가상기계 플랫폼에서 실행되는 실행 파일 포맷을 만들 수 있다.

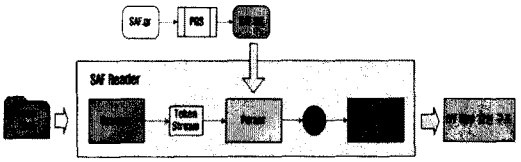
(그림 4)는 축약된 SAF 문법을 나타낸다.

```
<saf_program>       ::= { <class_dcl> }
<class_dcl>         ::= '.class' <modifiers> <class_name>
                          '.bgn'
                            <class_body>
                          '.end'
<class_body>        ::= <member_dcl> { <member_dcl> }
<member_dcl>        ::= <field_dcl> | <method_dcl> |
                        <class_dcl>
<field_dcl>         ::= '.field' <modifiers>
                        <type_specifier> <filed_name>
<method_dcl>        ::= '.method' <modifiers>
                        <type_specifier> <method_name>
                       '(' [<formal_param>] ')'
                          '.bgn'
                              <method_body>
                          '.end'
<formal_param>      ::= <type_specifier>
                        <formal_param_name> {',' <type_specifier>
                        <formal_param_name>}
<method_body>       ::= { <dcl_statement> } {
                          <instr_statement> }
<dcl_statement>     ::= <stack_dcl_st> |
                        <local_var_dcl_st> |
<stack_dcl_st>      ::= '.maxstack' <number>
<local_var_dcl_st>  ::= '.locals' '(' [<local_var_list>] ')'
<local_var_list>    ::= <local_var> {',' <local_var> }
<local_var>         ::= <type_specifier> <local_var_name>
                          {',' <local_var_name> }
<instr_statement>   ::= [ <label_name> ':'] ( <stack_op>
                        | <arithmetic_op> |
                          <flow_control_op>
                        | <object_op> |
                          <typ_conversion_op>)
<modifiers>         ::= 'public' | 'private' | 'static' |
                        'terminal' | 'guarded' |
                        'interface' | 'concept'
<type_specifier>    ::= 'byte' | 'integer' | 'long' |
                        'float' | 'double' | 'short' |
                        'char' |'reference' | 'boolean'
<*_name>           ::= 'xident'
<*_number>          ::= 'xnumber'
<*_op>              ::= <opcode_name> [ <param_name> ]
```

(그림 4) 축약된 SAF 문법

3.3 SAF 리더 (SAF Reader)

어셈블러의 내부 구조중 하나인 SAF 리더의 구조는 (그림 5) 와 같다.



(그림 4) SAF 리더의 구조

스캐너는 입력파일의 어휘 분석을 하여 토큰 스트림 (토큰번호, 토큰값)을 생성한다. 그리고 문법파일인 SAF.gr에서 명시하지 않은 명령어들에 대한 분석을 한다.

파서는 스캐너에서 생성된 토큰 정보와 PGS (Parser Generating System) 에서 얻어진 파싱 테이블 (Parsing Table) 을 가지고 SAF 파일에 대한 구분분석을 수행한다.

구분 분석 과정에서 SDT (Syntax-Directed Translator) 방식으로 SAF 프로그램의 구조를 AST (Abstract Syntax Tree) 로 변환하여 나타낸다. AST에는 EFF 파일의 생성에 필요한 정보를 가진다.

AST 탐색기는 파서를 통해서 생성된 AST를 탐색해서 EFF 데이터를 생성한다. 생성한 데이터는 EFF 내부 표현 구조에 저장이 된다[5][6][7].

3.4 EFF 라이터 (EFF Writer)

EFF 라이터는 EFF 내부 표현 구조에 저장된 데이터를 실행 파일 포맷의 구조에 맞춰서 바이너리 파일로 기록하도록 한다.

생성된 EFF 파일은 코드 부분과 메타데이터 부분으로 나뉜다. 코드 부분은 실행에 필요한 명령어로 구성되며 메타데이터 부분은 상수 정보, 스트링 정보로 구성된다.

3.5 EFF 내부 표현 구조 (EFF Representation)

EFF 내부 표현 구조는 실행 파일 포맷의 자료가 저장되는 부분이다. 내부 표현 구조는 명령어 부분과 메타데이터 부분으로 구성된다.

명령어 부분은 프로그램의 실행을 위한 SIL 명령어들과 메소드 정보가 저장되며, 메타데이터 부분은 실행에 필요한 정보가 저장된다. 메타데이터의 각 테이블은 태그와 해당 엔트리들로 이루어져 있

다. 테이블의 순서는 정해져 있지 않으며 필요할 때 마다 테이블을 삽입하면 된다.

4. 실험 및 결과

4.1 구현환경

본 논문에서는 어셈블러를 구현하기 위하여 구현 환경은 MS Windows XP Professional SP2 를 사용하였다. 개발도구는 Visual C++ 6.0을 사용하여 구현하였다.

4.2 실험결과

본 논문에서 구현한 어셈블러의 실험을 하기 위하여 사용한 실험 파일은 프로그램을 사용하였다.

(그림 6)은 어셈블러에 입력되는 입력파일에는 500이하의 완전수를 구하는 소스인 Perfect.saf를 사용하였다. (그림 7)은 어셈블러를 통해서 나온 결과인 Perfect.eff이다.

```
.class public CPerfect
   .bgn
      .method public integer main()
         .bgn
            .locals (integer i, max, sum)
               ldc.i   0
               str     i
               ldl     max
               ldc.i   500
               le
               brf     $$0
               ldc.i   0
               str     i
               ldc.i   1
               str     i
      $$3 :    ldl     i
               ldl     max
               le
               brf     $$1
               ldl     max
               ldl     i
               mod
               ldc.i   0
               eq
               brf     $$2
               ldl     sum
               ldl     i
               add
               str     sum
      $$2:     ldl     i
               ldc.i   1
               add
               str     i
               br      $$3
      $$1:     ldl     max
               ldl     sum
               eq
               brf     $$4
               ldl     max
               calli   write .......
```

(그림 6) Perfect.saf

```
00000000h: AB 4C 4C 05 01 00 00 00 00 00 00 04 01 00 00 04
00000010h: 00 00 00 20 01 00 00 00 00 00 00 20 A4 00 06 00
00000020h: 00 00 00 12 00 00 00 00 00 0B 01 00 00 00 00 06 02 00
00000030h: 00 00 3E 52 B5 00 00 00 06 00 00 00 00 12 01 00
00000040h: 00 00 06 02 00 00 00 12 01 00 00 00 0B 00 00 00
00000050h: 00 06 02 00 00 00 3E 52 BB 00 00 00 0B 02 00 00
00000060h: 00 0B 00 00 00 00 35 06 00 00 00 00 3C 52 76 00
00000070h: 00 00 0B 02 00 00 00 0B 00 00 00 00 30 12 02 00
00000080h: 00 00 0B 00 00 00 00 06 02 00 00 00 30 12 00 00
00000090h: 00 00 50 3E 00 00 00 0B 01 00 00 00 0B 02 00 00
000000a0h: 00 3C 52 A5 00 00 00 0B 01 00 00 00 65 FE FF FF
000000b0h: FF 0B 01 00 00 00 06 02 00 00 00 30 12 01 00 00
```

(그림 7) Perfect.eff

5. 결론 및 향후 연구

본 논문에서는 임베디드 시스템을 위한 가상기계인 EVM의 어셈블러를 수정하여 퍼베이시브 컴퓨팅 환경을 위한 가상기계 플랫폼에서 사용할 수 있는 어셈블러를 설계하고 구현하였다.

퍼베이시브 컴퓨팅 환경에서는 매우 제한된 리소스만을 제공하는 장치를 사용하기 때문에 소규모 가상기계에 적합하도록 EVM의 객체 지향 특성과 불필요한 명령어를 제거 하였다. 또한 수정된 어셈블러를 통하여 SAF 를 EFF 형식으로 출력하였다.

본 논문의 향후 연구 과제로는 어셈블러와, 실행엔진, 디버거, 최적화기 등을 포함한 퍼베이시브 환경에 적합한 가상기계 플랫폼에 대한 연구가 필요하다. 그리고 구현된 어셈블러를 새로운 가상기계 플랫폼에서 사용할 수 있도록 모듈화 하는 작업이 필요하다.

참고문헌

[1] 김신덕, 박기호, 맹혜선, 강두진, 이영미, "Embedded System용 JVM 개발 및 기술 분석에 관한 연구 보고서", 한국전자통신연구원, 1998.

[2] 오세만, 이양선, 고광만, "임베디드 시스템을 위한 가상기계의 설계 및 구현", 멀티미디어학회 논문지, 제 8권, 제9호, pp1282~1291, 2005.

[3] "퍼베이시브 컴퓨팅", www.nexio.com/images/pervasive_computing1.JPG

[4] 남동근, 오세만, "가상기계를 위한 어셈블리 언어의 설계", 동국대학교 석사학위논문, 2003.

[5] 정한종, 오세만, "임베디드 가상 기계를 위한 실행 파일 포맷", 동국대학교 석사학위논문, 2004.

[6] 손윤식, 오세만, "실행 파일 포맷 생성기의 설계 및 구현", 한국정보처리학회 추계학술발표대회 논문집, 제 11권, 제 2호, pp623~626, 2004.

[7] 오세만, 컴파일러 입문(개정판), 정익사, 2006.