DTV 콘텐츠 검증 시스템을 위한 Java 컴파일러의 개발

손민성*, 박진기*, 이양선**
*넥솔 텔레콤 부설 연구소
**서경대학교 컴퓨터공학과
e-mail: {kkabunda, jkpk2000}@nexoltelecom.com*, yslee@skuniv.ac.kr**

Development of a Java Compiler for Verification System of DTV Contents

Min-Sung Son*, Jin-Ki Park*, Yang-Sun Lee**
*Nexoltelecom Research Institute
**Dept of Computer Engineering, Seokyeong University

요 약

디지털 위성방송의 시작과 더불어 본격적인 데이터 방송의 시대가 열렸다. 데이터방송이 시작 되면서 데이터방송용 양방향 콘텐츠에 대한 수요가 급속하게 증가하고 있다. 하지만 양방향 콘텐츠 개발에 필요한 저작 도구 및 검증 시스템은 아주 초보적인 수준에 머물러 있는 것이 현실이다. 그러나 방송의 특성상 콘텐츠 상에서의 오류는 방송 사고에까지 이를 수 있는 심각한 상황이 연출 될 수 있다. 본 연구 팀은 이러한 DTV 콘텐츠 개발 요구에 부응하여, 개발자의 콘텐츠 개발 및 사업자 또는 기관에서의 콘텐츠 검증이 원활이 이루어 질수 있도록 하는 양방향 콘텐츠 검증 시스템을 개발 중이다. 양방향 콘텐츠 검증 시스템은 Java 컴파일러, 디버거, 미들웨어, 가상머신, 그리고 IDE 등으로 구성된다.

본 논문에서 제시한 자바 컴파일러는 양방향 콘텐츠 검증 시스템에서 데이터 방송용 자바 애플리케이션(Xlet)을 컴파일하여 에뮬레이팅 하거나 런타임 상에서 디버깅이 가능하도록 하는 바이너리형태의 class 파일을 생성한다. 이를 위해 Java 컴파일러는 *.java 파일을 입력으로 받아 어휘 분석과 구문 분석 과정을 거친 후 SDT(syntax-directed translation)에 의해 AST(Abstract Syntax Tree)를 생성한다. 클래스 링커는 생성된 AST를 탐색하여 동적으로 로딩 되는 파일들을 연결하여 AST를 확장한다. 의미 분석과정에서는 확장된 AST를 입력으로 받아 참조된 명칭의 사용이 타당한지 등을 검사하고 코드 생성이 용이하도록 AST를 변형하고 부가적인 정보를 삽입하여 ST(Semantic Tree)를 생성한다. 코드 생성 단계에서는 ST를 입력으로 받아 이미 정해 놓은 패턴에 맞추어 Bytecode를 출력한다.

1. 서론

2002년 3월 국내에서는 SkyLife의 디지털 위성방송의 시작과 더불어 본격적인 데이터 방송의 시대가 열렸다. 데이터방송이 시작되면서 데이터방송 전용 채널이 등장하고 양방향 콘텐츠에 대한 수요가 급속하게 증가하고 있다. 하지만 양방향 콘텐츠 개발에 필요한 저작 툴 및 검증시스템은 아주 초보적인 수준이어서 개발자들이 외면하고 있는 것이 현실이다. 하지만 방송의 특성상 기존의 웹이나 모바일 환경과는 달리 콘텐츠 상에 오류가 발생했을 시에는 TV를 보는 시청자들에게까지 영향을 미쳐 방송 사고에 까지 이를 수 있는 심각한 상황이 연출 될 수 있다. 따라서 이러한 양방향 디지털 콘텐츠들은 기존의 검증 방법과는 차별화된 철저한 검증이 필요 하다. 본 연구팀은 늘어나는 DTV 콘텐츠 개발 요구에 부응하고, 개발자 또는 콘텐츠의 검증을 원하는 사업자 및 기관에게 양질의 개발환경 및 검증 환경을 제공하기 위한 양방향 콘텐츠 검증 시스템을 개발 중에 있다. 양방향 콘텐츠 검증 시스템은 Java 컴파일러, 디버거, 미들웨어, 가상머신, 그리고 IDE(통합개발환경) 등으로 구성되어다.

본 논문에서는 양방향 콘텐츠 검증 시스템에서 데이터방송용 자바 애플리케이션(Xlet)을 컴파일 하여 에뮬레이터에서 실행 하거나, Runtime 상에서 디버거를 통하여 디버깅이 가능하도록 하는 바이너리형

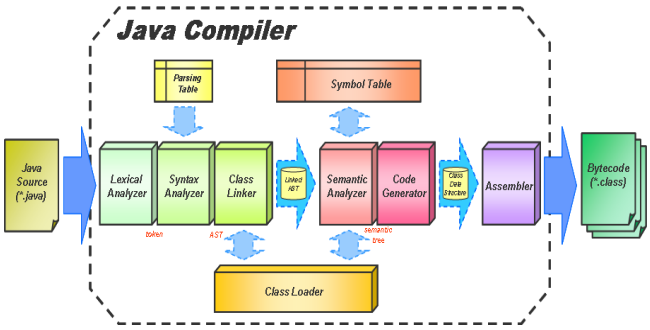
본 연구는 한국산업기술평가원의 '06 공통핵심기술 개발 지원 사업의 연구 결과로 수행되었음.

태의 Class 파일을 생성하는 Java 컴파일러를 설계하고 구현하였다.

2. Java 컴파일러의 개발

2.1 자바 컴파일러의 개요

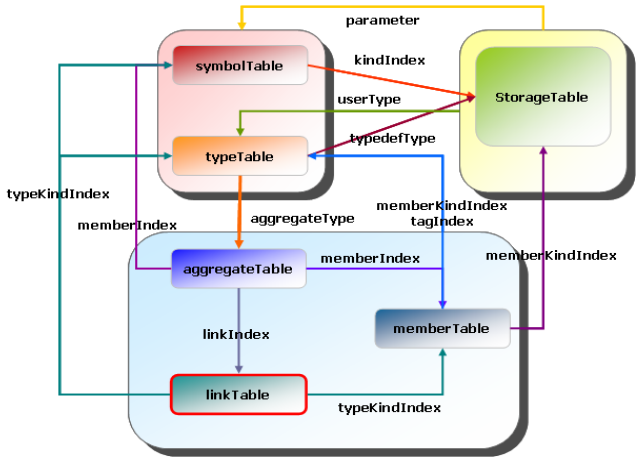
자바 컴파일러는 자료구조를 정의하는 부분인 심벌 테이블과, 프로그램의 문제 해결 과정을 기술한 부분으로 어휘 분석기, 구문 분석기, 클래스 로더, 클래스 링커, 의미 분석기, 그리고 코드 생성기로 구성된다.[3]



(그림 1) Java 컴파일러의 구성도

2.2 심벌 테이블

심벌 테이블이란 심벌(또는 명칭)들에 대한 바인딩 정보와 영향 범위를 관리하고 운영하는데 사용되는 자료구조이다.[1] 컴파일러에서는 어휘 분석과 구문분석을 거친 후에 SDT(Syntax Directed Translator)에 의해 생성된 추상 구문 트리를 분석하여 인식되는 명칭에 대해서 그 속성들을 수집하고 참조하기 위해서 사용한다. 다음 (그림 2)는 구성된 심벌 테이블과 테이블간의 연관 관계를 도식한 것이다.



(그림 2) 심벌 테이블 관계도

윈도우 테이블은 Symbol Table과 Type Table로 구성되어 있다. Symbol Table은 지역 스코프를 가지는 변수를 저장하기 위한 테이블이며 Type Table은 클래스와 인터페이스를 저장하기 위한 테이블 이다. Storage Table은 변수의 타입이나 함수의 리턴 타입 및 관련 정보를 저장하기위한 테이블이다. Aggregate Table과 Member Table, Link Table은 Type Table의 서브 테이블로서 클래스에 관련된 정보를 저장하기 위한 공간이다.[6]

2.3 어휘 분석기와 구문 분석기

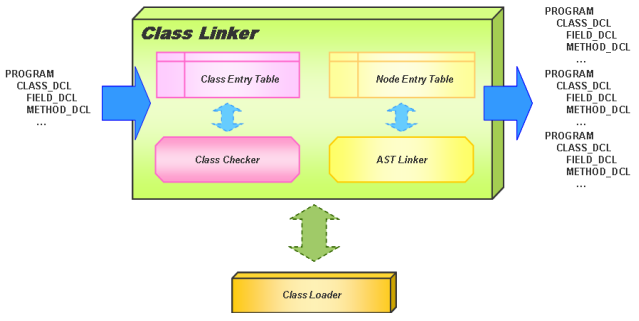
어휘 분석기는 토큰 자료구조를 이용하여 Java의 어휘를 인식한다. 인식 과정은 공백 제거, 명칭, 상수, 연산자, 키워드 분류 등이 있다. 키워드, 명칭, 상수의 형태는 Java 규격을 따르며 이를 인식 가능하도록 설계되었다.

구문 분석기는 어휘 분석기에서 얻어진 토큰 정보를 이용하여 입력으로 받은 프로그램의 구문 구조를 분석하는 과정을 담당한다. LALR(1)에서 인식 가능한 문법 형태로 기술된 Java 문법을 입력으로 하여 PGS(Parser Generating System)를 통하여 얻어진 파싱테이블은 구문 분석기의 행동을 미리 결정하며, 이에 따라 구문 분석된 결과는 SDT를 통하여 AST(Abstract Syntax Tree) 형태로 변환된다.

2.4 클래스 링커와 클래스 로더

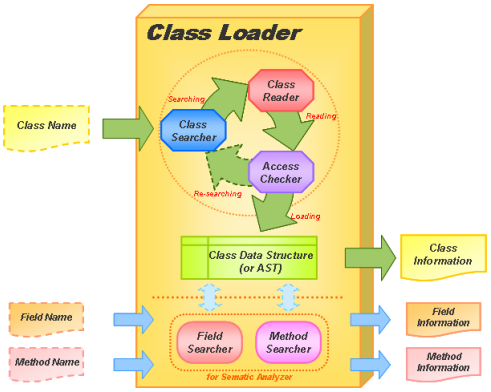
클래스 링커와 클래스 로더는 자바의 가장 큰 특징인 동적 로딩을 정적으로 처리하기 위한 방법이다.

클래스 링커는 SDT를 탐색하여 현재 파일에서 동적으로 로딩 해야 할 파일들을 판단하여 어휘 분석과 구문 분석을 하여 SDT를 생성하여 현재 파일과 연결시켜 의미 분석과 코드 생성과정에서 정적으로 처리할 수 있도록 한다. 다음은 클래스 링커에 대한 구조도 이다.



(그림 3) 클래스 링커의 구조도

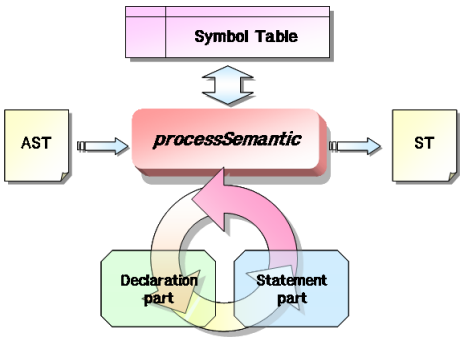
클래스 로더는 이미 생성되어 있는 class 파일에서 필드와 메소드 등의 정보를 추출하여 컴파일러에서 컴파일 타임에 사용할 수 있도록 해주는 모듈이다.



(그림 4) 클래스 로더의 구조도

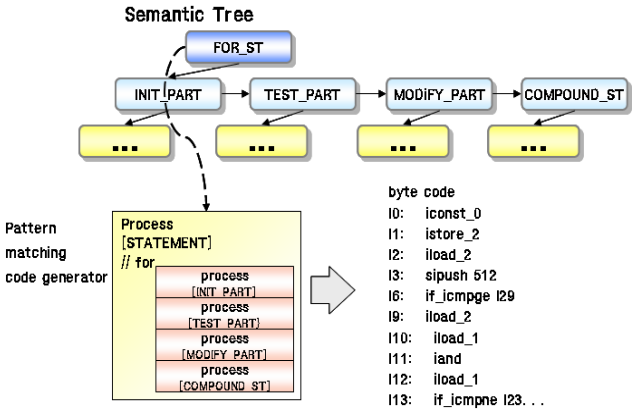
2.5 의미 분석기와 코드 생성기

의미 분석과정은 다음 그림과 같이 추상 구문 트리를 입력으로 받아 심벌 테이블의 정보와 의미 분석을 수행한수 시맨틱 트리(Semantic Tree)를 생성하게 된다. 시맨틱 트리는 추상 구문 트리에 의미정보를 부가한 자료구조로써, 구문 분석과정에서 처리하지 못한 의미 분석을 처리한 후 코드 생성에 용이한 형태로 변환된 구조이다. 다음 그림은 의미 분석과정을 도식화 한 것이다.



(그림 5) 의미 분석과정

코드생성기는 모든 분석이 끝난 시맨틱 트리를 입력으로 받아서 소스(*.java)와 의미적으로 동등한 bytecode를 생성한다. 코드 생성기는 프로그램의 구문 구조를 일관성 있게 표현하는 AST와 시멘틱 트리의 형태를 분석하여 설계하였기 때문에 생성 가능한 모든 의미 있는 구조와 매칭이 가능하다. 다만 패턴 매칭 방법을 이용하기 때문에 생성된 코드의 질이 저하되는 단점을 가진다. 이는 코드 최적화기를 통하여 충분한 향상이 가능 하다. 다음은 패턴 매칭 코드 생성기 모델이다.[2][4][5]



(그림 6) 패턴 매칭 코드 생성기 모델

코드생성 부분에서는 bytecode를 심볼릭화 하여 코드 생성 및 처리에 용이하도록 하였으며 형 변환 코드 리스트와 같은 자료구조를 두어 효율적인 코드 생성이 이루어지도록 하였다. 형 변환 코드 리스트는 코드 생성시 형 변환 시맨틱 노드를 bytecode로의 변환 과정을 미리 계산하여 구성한 자료 구조이다.

[표 1] 형 변환 코드 리스트

```
int typeConvertCodeList[typeConversionNodeNumber][2] = {
/*CVS_B*/        { opc_I2i,        opc_i2b},
/*CVL_C*/        { opc_I2i,        opc_i2c},
/*CVL_S*/        { opc_I2i,        opc_i2s},
/*CVL_I*/        { opc_I2i,        cv_fin},
/*CVL_F*/        { opc_I2f,        cv_fin},
/*CVL_D*/        { opc_I2d,        cv_fin},
...}
```

3. 실행 결과 및 성능 평가

다음은 구현된 Java 컴파일러에 입력된 소스코드와 컴파일러에서 컴파일된 class 파일의 내용을 확인 할 수 있도록 덤프한 것이다.

[표 2] TicTacToe.java 의 일부

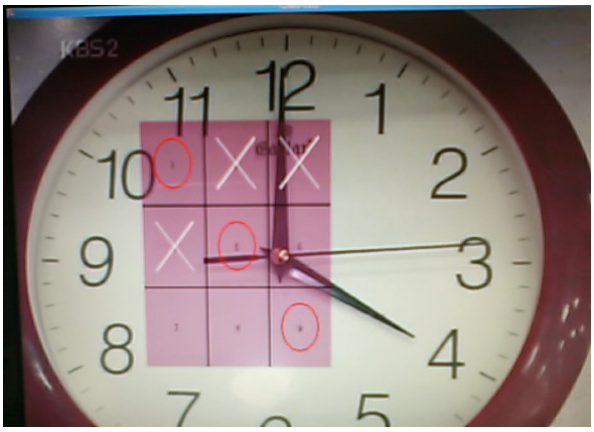
```
public class TicTacToe extends Component {
    private int player;
    private int computer;
    private int[] hasWon = new int[1<<9];
    final static int[] PRIOR_FIELD = {4, 0, 2, 6, 8, 1, 3, 5, 7};
    private Color gridColor;
    private Color signColor;
    private Color winColor;

    public TicTacToe() {
        this.player = 0;
        this.computer = 0;
      setWon( (1<<0) | (1<<1) | (1<<2));
```

```
                    •
                중간생략
                    •
    public void paint( Graphics g) {
            g.setColor( getBackground());
            g.fillRect( 0, 0, getSize().width, getSize().height);
            g.setColor( gridColor);
            int fieldSize = this.getFieldSize();
                    •
                중간생략
                    •
```

[표 3] TicTacToe.class의 일부

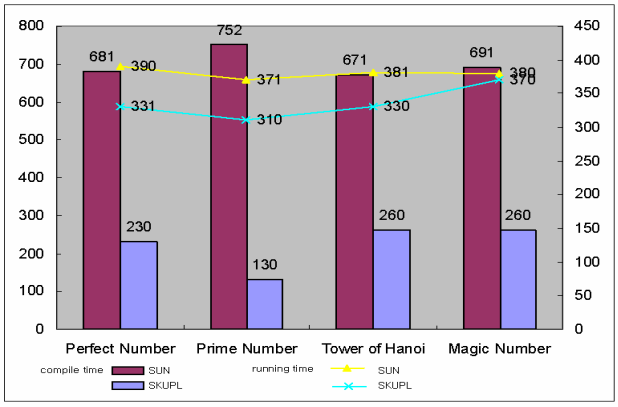
```
000000 cafebabe              magic = ca fe ba be
000004 0003                  minor version = 3
000006 002d                  major version = 45
000008 0093                  147 constants
00000a 010009                   1. UTF length=9
00000d 546963546163546f65                  TicTacToe
000016 070001                2. Class name #1
000019 01000a                3. UTF length=10
00001c 536f7572636546696c65                SourceFile
000026 01000e                4. UTF length=14
000029 546963546163546f652e6a617661        TicTacToe.java
                    •
                중간생략
                    •
                             field/method attribute 0
00058a 0013                     name = #19<Code>
00058c 000000bc                 length = 188
000590 0045                     max stack: 69
000592 0001                     max locals: 1
000594 000000b0                 code length: 176
000598 2a                       0 aload_0
000599 b70015                   1 invokespecial #21
00059c 2a                       4 aload_0
00059d 04                       5 iconst_1
00059e 1009                     6 bipush 9
0005a0 78                       8 ishl
0005a1 c5001601                 9 multianewarray #22 1
0005a5 b50018                   13 putfield #24
0005a8 2a                       16 aload_0
0005a9 03                       17 iconst_0
0005aa b5001a                   18 putfield #26
0005ad 2a                       21 aload_0
0005ae 03                       22 iconst_0
0005af b5001c                   23 putfield #28
                    •
                중간생략
                    •
```

다음 (그림 7)은 본 컴파일러에서 생성된 class 파일을 본 검증 시스템의 미들웨어 상에서 수행되는 모습을 캡처 한 것이다.



(그림 7) 미들웨어 상에서 실행중인 TicTacToe.class

다음은 본 연구팀이 개발한 Java 컴파일러와 Sun 사의 Java 컴파일러의 성능 평가를 위한 도표이며 컴파일시간과 JVM에서의 수행시간을 비교하였다.



(그림 8) 컴파일러의 성능 비교

4. 결론 및 향후 연구

본 논문에서는 양방향 콘텐츠 검증 시스템에서 자바 애플리케이션(Xlet)을 컴파일 하여 에뮬레이터에서 실행 하거나, Runtime 상에서 디버거를 통하여 디버깅이 가능하도록 하는 바이너리형태의 Class 파일을 생성하는 Java 컴파일러를 설계하고 구현하였다. 본 Java 컴파일러는 순수 원천 기술을 바탕으로 정형화된 최신 컴파일러 개발 기법을 적용하여 개발 하였다.

향후, 패턴 매칭으로 통하여 생성된 코드를 보다 최적화된 코드로 생성하도록 하는 코드 최적화 작업과 컴파일러에 이상이 없는지에 대한 검증 작업을 진행 할 것이다.

참고문헌

[1] A.V. Aho, M.S. Lam, R. Sethi and J.D. Ullman, Compilers : Principles, Techniques and Tools, 2nd Edition, Addison Wesley, 2007

[2] Bill Venners, Inside the JAVA Virtual Machine, 2nd Edition, McGraw-Hill, 2000.

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

[4] Joshua Engel, "Programming for the Java Virtual Machine", Addison Wesley, 1999.

[5] 이양선, 임베디드 시스템을 위한 ANSI C 컴파일러 개발, 서경대학교 산학협력단, 2005.

[6] 이양선, Retargetable C 컴파일러를 위한 코드 최적화기 개발, 서경대학교 산학협력단, 2005.

[7] 이양선, 임베디드 시스템을 위한 C++ 컴파일러 개발, 서경대학교 산학협력단, 2006.