Java ORB 시스템을 위한 IDL-to-Java 컴파일러 개발

이 진 호† · 이 근 영†† · 정 태 명†††

요 약

복잡한 분산 시스템 상에서 운용되는 소프트웨어들을 위해서 안정된 하부구조인 동시에 소프트웨어 아키텍처인 OMG의 CORBA는 효율적인 객체 지향 개발 환경을 제공한다. 인터넷, 개인용 컴퓨터, WWW의 사용 증가로 인해 이기종의 분산 시스템들 상에서 구동될 수 있는 응용프로그램들이 요구되고 있으며 이와 같은 이기종간의 이식성을 충족시키기 위해서 Java ORB에 대한 필요성이 증가되고 있는 추세이다.

본 논문에서는 IDL을 자바로 사상하는 규약을 분석하며 자바로 구현된 CORBA 시스템을 지원하기 위한 IDL-to-Java 컴파일러를 구현하였다. 구현된 컴파일러는 어휘 분석기, 파서, 목적 코드 생성기로 나누어지며 목적 코드 생성기에서는 IDL에서 자바로 사상된 파일들 이외에 투명성을 제공하게 되는 클라이언트 스텁 코드와 객체 구현 스켈레톤 코드를 생성한다.

An IDL-to-Java Compiler for Java ORB System

Jin-Ho Lee† · Geun-Young Lee†† · Tai-Myoung Chung†††

ABSTRACT

The CORBA which is a stable software infrastructure and architecture provides an effective object-oriented client/server development environment for complex distributed systems. The need for a development environment such as a Java ORB has been greatly increased for its portability, because the rapid growth of Internet and WWW has made the distributed computing environment more complex and sophisticated,

In this research, we designed and developed an IDL-to-Java compiler as a part of the Java ORB system that is designed to comply with the CORBA system standardized by OMG. It consists of lexical analyzer, parser, and code generator, and produces multiple target files such as client stub file, object implementation skeleton code, and multiple necessary Java source files to provide complete transparency to users.

1. 서 론

하드웨어나 소프트웨어에 상관없이 현대 사회 전 분야에 걸쳐서 진행되고 있는 정보통신 기술의 발달로 인해서 일반적인 네트워크에서 이용되는 서비스들은 데이터의 전송 등과 같은 기본적인 서비스에서 네트워크를 통한 데이터 공유, 분산 시스템이나 클라이언트/서버 환경을 바탕으로 하는 멀티미디어, 전자 결재 등과 같이 다양하면서 복잡한 네트워크 서비스들로 빠르게 확장되고 있다.

이와 같은 컴퓨터 네트워크 환경은 사용자들에게 다양한 서비스를 제공하는 장점이 있지만 개발자들에게는

† 준 회 원 : 성균관대학교 정보공학과 실시간 시스템 연구실
†† 종신회원 : 한국전자통신연구원 분산처리연구실
††† 종신회원 : 성균관대학교 정보공학과 실시간 시스템 연구실

논문접수 : 1998년 2월 11일, 심사완료 : 1998년 6월 22일

복잡적인 서버스 제공을 위해 개발 및 유지보수 비용이 많이 드는 소프트웨어들을 개발 또는 통합해야 하며 이 때 많은 문제점들이 야기된다.

소프트웨어 개발 및 통합 시에 분산된 정보로 인한 데이터의 일관성, 각각의 기종마다 특정 데이터 형식을 갖게 되는 데이터의 이질성 등과 같이 다양한 플랫폼에서 필요로 하는 요구 조건을 만족시키기 위해서 표준화된 개발 환경이 요구되었으며 그 중에서 OMG(Object Management Group)가 제안한 대표적인 표준안이 객체 지향 분산 환경을 제공하는 CORBA(Common Object Request Broker Architecture)이다[10].

CORBA는 일반적인 RPC(Remote Procedure Call)들과는 달리 분산 환경에서 많이 사용되고 있는 객체 지향 개념으로 구성되어 있어서 클라이언트들에게 서버 객체들과 통신하는 부분, 서버 객체를 활성화시키는 부분, 서버 객체를 저장하는 메커니즘 등을 숨기어 해당 객체들의 위치에 상관없이 투명하게 서버 객체들의 서비스를 요구하거나 응답을 받을 수 있도록 하는 환경과 인터페이스를 제공해준다[7,8,12].

이와 같은 특징들과 더불어 CORBA는 개발자들에게는 객체 구현의 투명성을 통하여 높은 생산성과 소프트웨어 재사용 등의 이점을 제공해 주며 사용자들에게는 분산 소프트웨어의 요소들인 객체들의 관리를 통하여 여러 회사의 제품들을 동적으로 통합할 수 있는 기능을 제공하기 때문에 분산 환경에서 대표적인 솔루션으로 자리잡고 있다.

OMG의 CORBA 표준안에서 제안하고 있는 IDL(Interface Definition Language)은 서버 객체의 서비스들에 해당하는 인터페이스를 정의하는 기능을 가지고 있는 언어일 뿐만 아니라 IDL 컴파일러를 사용하여 소프트웨어 설계와 그것을 구현한 코드 사이의 직접적인 경로를 제공해줄 수 있다[8,14]. 또한 어떠한 절차적 구조(procedural structures)나 변수(variable)들을 포함하지 않고 소프트웨어 설계와 구현을 독립적으로 하였기 때문에 CORBA에서 소프트웨어 표기를 위한 방법으로 사용된다.

서버 객체의 서비스들에 대한 인터페이스를 제공해주는 클라이언트 환경에서의 스텁과 서버 환경에서의 객체 구현 스켈레톤(Object Implementation Skeleton) 코드 생성의 편의를 위하여 IDL을 사용하는 것 이외에 CORBA에서는 인터페이스 저장소(IR:Interface Repository)에 모든 객체들에 대한 IDL 정의를 ORB가 저장해준다.

이와 같이 인터페이스 정의들에 관한 정보들은 객체형 분산 시스템에서 중요한 자원이 되기 때문에 ORB에서 뿐만 아니라 클라이언트와 객체 구현, 객체 계층 브라우저와 디버거와 같은 유틸리티들에서도 유용하게 사용될 수 있다[14].

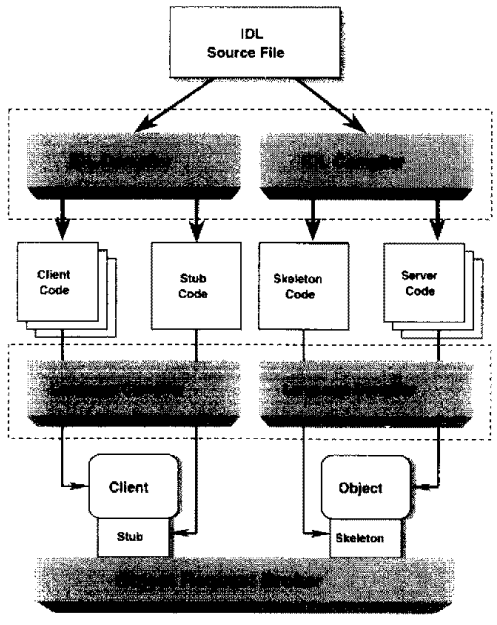
본 논문에서는 이러한 환경 개발을 배경으로 CORBA에서 사용되고 있는 IDL과 IDL로부터 자바 프로그래밍 언어로의 사상에 관한 분석을 바탕으로 자바를 기반으로 하는 CORBA 시스템과 연동되는 효율적이며 안정된 IDL-to-Java 컴파일러 시스템을 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서 IDL 컴파일러의 필요성을 설명하고 3장에서는 본 논문에서 구현한 IDL-to-Java 컴파일러의 구조를 어휘 분석기, 파서, 목적 코드 생성기로 나누어 설명하며 4장에서는 결론과 향후 계획을 제시한다.

2. IDL 컴파일러의 필요성

2.1 IDL 컴파일러의 역할

IDL 컴파일러는 일반적으로 기계어 코드를 생성하는 범용 컴파일러들[1]과는 달리 서버의 객체가 클라이



(그림 1) IDL 컴파일러의 역할
(Fig. 1) A role of IDL compiler

언트들에게 제공하고자 하는 서비스들에 대한 표현을 포함하고 있는 IDL 파일을 입력받아서 특정 프로그래밍 언어로의 사상 규약을 통한 스텁이나 스켈레톤 코드들을 생성해주는 소프트웨어이다.

CORBA에서 정의된 IDL에서 특정 프로그램 언어로 사상하는 IDL 컴파일러의 사용으로 인해서 시스템 개발자나 사용자들은 공통된 인터페이스를 통해 복잡한 프로그램의 작업 분담 시에 편의를 제공받을 뿐만 아니라 IDL의 독립적인 특성으로 인해 작성 후에도 소프트웨어의 유지 보수에 상당한 도움을 받게 된다.

한편 IDL 컴파일러에 의해 생성되는 스텁이나 스켈레톤 코드들은 클라이언트와 객체 구현간의 ORB를 통한 세부적인 통신 사항 등을 담당하게 되며 서버 측 객체들이 마치 클라이언트 측에 있는 것처럼 보여주는 대리 객체(proxy object)들을 통하여 프로그램 개발자들에게 하나의 고정 호스트에서의 프로그램을 작성하는 것과 같은 환경을 제공해 준다.

(그림 1)에서 보는 바와 같이 IDL 컴파일러가 IDL 파일을 입력받아서 컴파일한 결과로서 클라이언트와 객체 구현(서버) 프로그램 이외에 특정 프로그래밍 언어로 작성된 클라이언트 스텁 코드와 객체 구현 스켈레톤 코드가 생성된다. 최종적으로 IDL 컴파일러에서 출력된 스텁 코드와 클라이언트 프로그램, 스켈레톤 코드와 객체 구현 프로그램을 함께 특정 언어 컴파일러로 컴파일하여 실질적으로 ORB에서 수행되는 프로그램들을 생성하게 된다. 특히 사상되는 프로그래밍 언어의 종류에 따라 스텁과 스켈레톤이 다른 프로그래밍 언어로 생성될 수도 있는데 이러한 점은 클라이언트와 객체 구현이 서로 다른 프로그래밍 언어로 작성되는 것을 가능하게 하여 기존에 구현되었던 프로그램들이 다른 언어로 구현된 프로그램과 호환될 수 있는 장점이 있다.

2.2 IDL-to-Java 컴파일러의 필요성

자바는 일반적으로 다른 객체 지향 언어와 비슷한 원리들을 가지고 있는 동시에 다른 언어들에 비해서 플랫폼에 대해 독립적인 점이 특히 주목할 만하다. 이와 같이 객체 지향이라는 점과 플랫폼에 대해 독립적인 점은 다른 언어들보다도 객체 지향 분산 환경을 구축하고자 하는 CORBA의 목적에 근접시키게 된다.

비록 자바 언어가 인터프리터 언어이기 때문에 속도면에서 단점이 있기도 하지만 점점 고속화되는 하드웨어, 현재 개발되고 있는 자바칩 등으로 속도 문제는 장차 개선될 것으로 전망된다. 자바 기반의 CORBA 환경 구축의 당위성에 관해 기술하면 다음과 같다.

- 자바는 객체 지향 분산 환경을 제공하는 CORBA가 원하는 객체 지향 언어이다.
- 자바는 이기종의 환경 하에서 작동이 가능한 언어로서 이식성이 강하다.
- C++, Smalltalk 등과 같은 타 객체 지향 언어에 비해 단순하여 배우기 쉬우며 IDL로부터 사상(mapping)이 용이하다.
- 인터넷의 발달과 함께 WWW(World Wide Web)에 적합한 언어로 인정받고 있으며 전세계적으로 사용이 증가하는 추세에 있다.

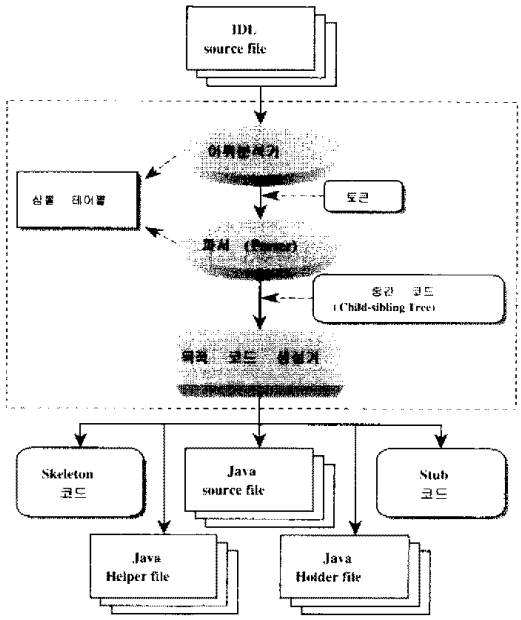
위와 같은 자바 기반의 CORBA 환경 구축의 당위성으로 인해 IDL-to-Java 컴파일러의 개발은 필수적이며 단순히 사용자가 정의한 클라이언트 스텁과 객체 구현 스켈레톤을 생성하는 다른 IDL 컴파일러의 기능뿐만 아니라 생성된 코드가 자바 언어이기 때문에 자바 ORB와 함께 이질적인 시스템에서 사용될 수 있는 이식성이 강한 스텁과 스켈레톤을 생성하게 된다. 한편 자바 ORB가 웹 브라우저에 내장될 경우에는 실질적인 클라이언트나 서버 코드뿐만 아니라 해당하는 스텁이나 스켈레톤 코드가 함께 오게 되기 때문에 IDL-to-Java 컴파일러에서 생성되는 스텁 및 스켈레톤 코드의 양을 최소화함으로써 CORBA 응용 프로그램들이 웹 브라우저에서 빠른 로드 속도를 유지할 수도 있다. 이와 같이 생성된 코드들은 다른 언어를 기반으로 하는 코드들보다 인터페이스만 같다면 재컴파일없이 다른 시스템에서 이용될 수 있다는 장점을 가진다.

국내에서는 Ada95를 생성하는 IDL 컴파일러[17]가 개발되었지만 아직 IDL-to-Java 컴파일러는 개발되지 않고 있으며 현재 개발된 IDL-to-Java 컴파일러들로는 자바로 구현된 CORBA인 JacORB에서 제공하는 컴파일러, Visigenic사의 Visibroker for Java에서 제공하는 idl2java, Orbix사의 OrbixWeb에서 제공하는 idl 등이 있다[2,3,4,5,15,16]. JacORB에서 제공하는 컴파일러는 JacORB 자체가 교육용으로 제작되었으며 [11]에서 제안한 IDL-to-Java 사상이 제안되기 전에 개발되어 기능상의 미약한 점이 있으며 idl2java와 idl은 상업용으로 개발된 IDL-to-Java 컴

파일러입니다.

본 논문의 IDL to-Java 컴파일러에서는 OMG에서 제안한 표준을 충분히 수용하였으며 lex와 yacc[6]을 사용하여 다 컴파일러들에 비해 사상하기 쉬운 중간 코드를 사용하여 간결하게 구현된 특징들이 있다.

3. IDL-to-Java 컴파일러 구조



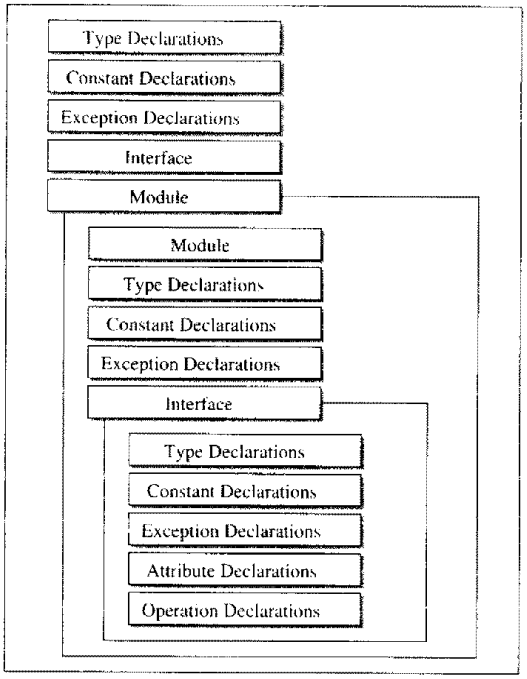
(그림 2) IDL-to-Java 컴파일러의 구조
(Fig. 2) An architecture of IDL-to-Java compiler

본 논문에서 구현한 컴파일러의 구조는 (그림 2)에서 보는 바와 같이 IDL 소스 파일을 입력받아서 해당 자바 사상 코드를 생성하는데 3단계 과정을 거친다. 1단계는 IDL 파일을 토큰 스트림으로 분리하는 어휘 분석 단계이고 2단계인 파서에서는 어휘 분석기에서 전달된 토큰 스트림과 IDL 문법을 바탕으로 하는 구문 분석을 하며 그에 따른 중간 코드를 생성한다. 마지막 3단계는 파서에서 생성된 중간 코드에서 원하는 자바 코드를 생성해 주는 코드 생성 단계이다.

본 장에서는 입력 파일인 IDL 파일의 구조, 어휘 분석기, 파서, 목적 코드 생성기의 순으로 설명하고자 한다.

3.1 IDL 파일 구조

IDL 문법을 분석한 결과 IDL-to-Java 컴파일러의 입력 파일이 되는 IDL 소스 파일의 구조는 (그림 3)과 같다.



(그림 3) IDL 파일의 구조
(Fig. 3) An architecture of IDL file

사용되는 범위를 한정하는 기능을 가진 모듈은 nest가 가능하며 모듈 내에는 서버 객체에서 사용되는 데이터 형 및 상수 선언문, 예외 상황이 일어났을 때 수행되는 예외 선언문, 서버 객체에서 수행되는 오퍼레이션들을 포함하고 있는 인터페이스 정의들이 있다. 그 중에서 모듈 외부에 전역적으로 선언될 수 있는 데이터 형 및 상수 선언문, 예외 선언문, 인터페이스 정의들은 CORBA 환경 내에서 어떠한 곳에서도 참조가 가능하게 된다.

실질적으로 서버 객체들을 명시하게 되는 인터페이스는 서버 객체에서 사용하는 데이터 형 및 상수 선언문, 서버 객체 수행 시에 발생되는 예외에 관한 예외 선언문, 서버 객체의 속성들인 속성 선언문, 서버 객체의 서비스를 수행하는 오퍼레이션들로 구성된다.

본 논문에서의 IDL-to-Java 컴파일러에서는 이와 같이 해당 IDL의 각각의 선언문들을 올바른 사상 규약에 따라 자바 기반의 ORB에서 수행될 자바 코드로 바꾸어 주며 이후 컴파일러의 기능을 쉽게 설명하기 위하여 (그림 4)에 나타난 실제 IDL 코드를 예로 들어 어휘 분석기, 파서, 목적 코드 생성기 등을 설명할 것이

다. (그림 4)에 있는 예제의 의미는 다음과 같다.

- 서버 객체의 기능을 하는 인터페이스 paper는 논문 검색을 위한 오퍼레이션인 getstatus를 서비스로 제공한다.
- getstatus 오퍼레이션은 논문 제목과 논문 연도를 입력으로 하여 해당 검색 논문의 고유 번호와 사용 여부를 출력 값으로 한다.
- 오퍼레이션의 입력 매개 변수로서 title은 논문 제목이고 입출력 매개 변수인 pID는 입력 시에는 논문 년도, 출력시에는 논문 번호를 의미한다.
- 오퍼레이션의 리턴 값에서는 해당 논문의 사용 여부를 boolean형으로 알려준다.

```
interface paper {
   boolean getstatus(in string title, inout long pID);
};
```

(그림 4) 논문 검색을 위한 IDL의 예제
(Fig. 4) An IDL example for paper search engine

3.2 어휘 분석기

실제 IDL 파일이 IDL-to-Java 컴파일러의 어휘 분석기에 전달되기에 앞서 일반적인 C나 C++언어에서 사용되는 #include, #define과 같은 전처리 문을 처리하는 전처리기(preprocessor)를 거치게 된다.

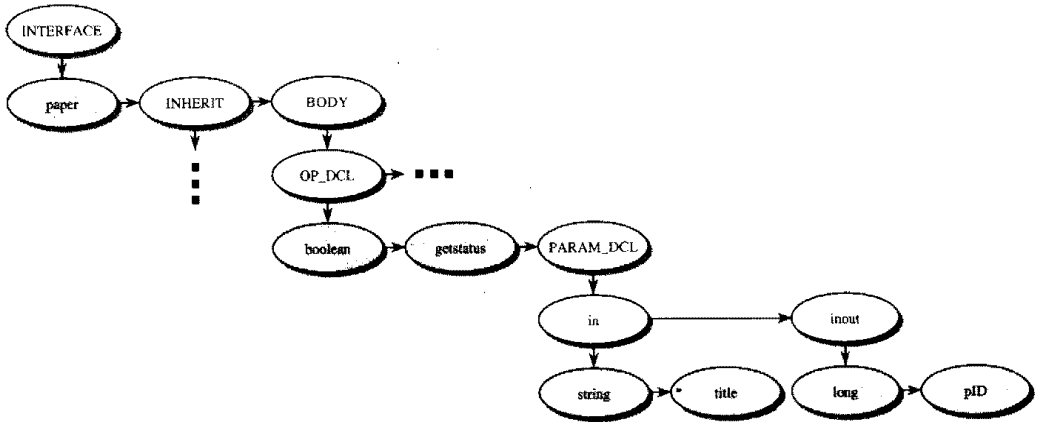
해당 전처리기를 거친 IDL 소스 파일(source file)은 어휘 분석기에서 정규 표현식으로 표현한 패턴에 맞는 토큰으로 분리되어 문법을 검사하는 파서로 전달된다. 여기서 해당하는 토큰은 키워드, 연산자(operator), 식별자(identifier) 등을 예로 들 수 있으며 (그림 4)의 IDL 예제에서 키워드에 해당하는 토큰으로는 interface, boolean, in, string, inout, long이 있으며 연산자에 해당하는 토큰으로는 { } ( ) , ;와 같은 것들이 있다. 또한 식별자에 해당하는 토큰으로는 paper, getstatus, title, pID가 된다.

어휘 분석기에서는 이와 같이 IDL 문법에서 정의된 토큰들을 구문 분석기에 순서대로 전달하는 동시에 전달할 토큰이 해당 정규 표현식에 맞지 않을 경우에는 에러를 출력한다.

3.3 파서(Parser)

IDL-to-Java 컴파일러에서의 파서는 구문 분석 단계에서 어휘 분석기로부터 토큰 스트림을 전달받아 입력 파일 구문이 IDL 문법에서 허용하는 올바른 문장인가를 검사하며 의미 분석 단계에서는 각각의 식별자에 대해서 생성된 심벌 테이블을 참조하여 상수와 형 정의문에서 사용되는 각각의 데이터형과 범위 규칙(scope rule)을 올바르게 지키는지 검사하는 동시에 자바 언어에서는 없는 typedef문에 대한 사상을 수행한다.

최종적으로는 구문 분석과 의미 분석을 거친 뒤에 목적 코드 생성기의 입력이 되는 중간 코드를 생성하게 된다. 중간 코드는 일반적으로 구문 트리를 많이 사용하는데 그 중에서 child-sibling 트리는 자신의 범위(scope) 내에 있는 선언문들을 대등한 형제 노드로 연



(그림 5) 파서에서 생성된 child-sibling 트리
(Fig. 5) A child-sibling tree generated by the parser

결하는 자료 구조로서 본 논문에서 파서가 생성하는 중간 코드이다. child-sibling 트리는 일반적으로 n-ary 트리보다 구현하기에 간단하며 코드 생성 시에 IDL 문법 및 생성되는 자바 코드 사이의 사상 규약이 선언문별로 정의되어 있는 특성상 사상하기에 용이하다[9,13].

본 논문에서 구현한 파서에서 생성하는 중간 코드들은 (그림 3)의 IDL 파일의 요소들인 형 선언문, 상수 선언문, 예외 선언문, 인터페이스과 같은 선언문들에 대한 child-sibling 트리이다. 그 중에서 IDL의 인터페이스 예제인 (그림 4)에 대해서 파서가 생성하는 트리는 (그림 5)와 같으며 특히 어둡게 칠해진 부분은 문자열 형태로 저장되어 자바 코드로 사상할 때 실질적으로 이용되는 부분들이다. 인터페이스 선언문에서는 인터페이스의 헤더에 해당하는 인터페이스 이름과 상속 관계, 인터페이스 몸체(body) 부분이 형제(sibling) 노드로 연결되고 인터페이스의 몸체 부분에는 인터페이스 선언문 안에 들어 올 수 있는 형 선언문, 상수 선언문, 예외 선언문, 속성 선언문, 오퍼레이션들이 자식(child) 노드로 연결된다.

(그림 5)에서 인터페이스 선언문은 상속 관계는 없으며 몸체 부분에는 오퍼레이션인 getstatus가 자식 노드로 연결된다. 오퍼레이션 노드인 getstatus의 자식 노드로 리턴값, 오퍼레이션 이름, 매개 변수 선언이 형제 관계로 연결되고 매개 변수 선언에는 입력 매개 변수인 title, 입출력 변수인 pID가 자식 노드로 연결된다.

3.4 목적 코드 생성기

파서에서 생성된 중간 코드인 child-sibling 트리에서 실제 컴퓨터에서 수행될 수 있는 기계어를 생성하는 범용 컴파일러와는 달리 ORB에서 수행될 자바 코드로 사상해 주는 부분이 IDL-to-Java 컴파일러의 목적 코드 생성기이다.

목적 코드 생성기에서는 파서에서 생성된 해당 선언문의 중간 코드별로 기본형에 의한 사상 규약과 선언문 사상 규약을 통하여 IDL 파일의 내용들을 자바 언어로 사상하게 되는데 기본형에 의한 사상 규약은 〈표 1〉과 같으며 선언문 사상 규약은 다음과 같다[11].

- 모듈 : 자바 언어의 패키지로 사상된다.
- 상수 선언문 : IDL의 인터페이스 내에 있는 상수 선언문은 IDL의 인터페이스가 사상되는 자바 인터페이스 내에 public static final field로 사상되며, 인터페이스 외부에 있는 상수 선언문은 상수와 같은 이름을 가진 자바 언어의 public interface를 생성하고 해당 인터페이스는 value라는 이름의 public static final 필드를 포함하게 된다.
- 예외 선언문 : org.omg.CORBA.UserException 클래스에서 상속되는 final 자바 클래스로 사상된다.
- 형 선언문 : 기본형을 제외한 IDL의 형 선언문으로는 시퀀스형, 배열, 열거형, 구조체, 공용체 등이 있는데 시퀀스형과 배열은 자바의 배열로 사상되며 열거형, 구조체, 공용체와 같은 사용자 정의 형 선언문들은 IDL에서 선언된 형과 같은 이름을 가진 final 자바 클래스로 사상된다.
- 인터페이스 : IDL의 인터페이스와 같은 이름을 가진 자바 언어의 public interface로 사상된다. 사상되는 자바 인터페이스 내에는 IDL의 인터페이스 안에 있던 형 선언문, 상수 선언문, 예외 선언문, 속성 선언문, 오퍼레이션들이 사상 규약에 따라 사상되어 들어간다. 또한 IDL의 인터페이스마다 자바 기반의 ORB를 통하여 세부적인 통신을 담당하게 될 클라이언트 스텁과 객체 구현 스켈레톤 파일들이 생성된다.

〈표 1〉 기본형에 대한 IDL-to-Java 사상 규약
〈Table 1〉 IDL-to-Java mapping rule for basic types

IDL Type	Java Type
boolean	boolean
char / wchar	char
octet	byte
short / unsigned short	short
long / unsigned long	int
long long / unsigned long long	long
float	float
double	double

특히 패키지로 사상되는 IDL의 모듈은 자바 언어의 특성상 디렉토리를 생성하게 되며 사상된 클래스나 인

터페이스에 대한 자바 파일들은 해당 패키지의 이름과 같은 디렉토리에 생성된다.

추가적으로 오퍼레이션의 출력 및 입출력 매개 변수 전달 등과 같은 경우에 사용되는 Holder 클래스와 모든 IDL 사용자 정의형을 다루기 위한 정적 메소드들을 제공해주는 Helper 클래스들이 생성되는데 다른 프로그래밍 언어에서는 보기 힘든 IDL-to-Java 사상에서만 특징적으로 사용되는 경우이다.

(그림 4)의 논문 검색 IDL 예제로부터 파서에서 생성한 (그림 5)의 중간 코드를 이용하여 본 논문에서 구현한 컴파일러의 코드 생성기가 생성하는 자바 코드들로는 (그림 6)부터 (그림 10)과 같으며 특히 (그림 7)과 (그림 8)은 앞에서 설명한 Holder 클래스와 Helper 클래스이다.

```
public interface paper
  extends org.omg.CORBA.Object {
    public boolean getstatus(
      java.lang.String title, IntHolder pID);
}
```

(그림 6) 사상된 자바 인터페이스
(Fig. 6) Mapped Java interface

(그림 6)은 논문 검색 IDL예제에서의 IDL 인터페이스인 paper에 대해서 사상되는 자바 인터페이스 코드로서 인터페이스 내에 getstatus 오퍼레이션을 가지고 있으며 각각의 데이터형에 대한 사상은 <표 1>과 같은 사상 규약을 따른다. 또한 CORBA에서 IDL의 모든 인터페이스들은 CORBA 정의 상 암시적으로 CORBA의 Object 클래스에서 상속되기 때문에 (그림 6)에서 보는 바와 같이 사상된 자바 코드에서 자바의 object 클래스와 구별되게 CORBA의 Object 클래스에 대한 상속 관계를 표기한다.

```
final public class paperHolder implements
 org.omg.CORBA.portable.Streamable {
  public paper value;
  public paperHolder() {}
  public paperHolder(paper initial) {...}
  public void _read
    (org.omg.CORBA.portable.InputStream i) {...}
  public void _write
  (org.omg.CORBA.portable.OutputStream o) {...}
  public org.omg.CORBA.TypeCode _type() {...}
}
```

(그림 7) IDL 인터페이스에 대한 홀더 클래스
(Fig. 7) A Holder class for an IDL interface

(그림 7)은 paper 인터페이스가 오퍼레이션의 출력 및 입출력 매개 변수로 사용될 때 사용되는 Holder 클래스로서 ORB의 이식성(portability) 기능을 제공해주는 portable 패키지의 Streamable 인터페이스를 구현한다. 일반적으로 typedef문에 의해 정의된 것을 제외한 모든 사용자 정의 데이터형에 대해서는 해당 Holder 클래스가 생성되며 기본형에 대해서는 각각의 Holder 클래스들이 omg.org.CORBA 패키지 안에 정의되어 있다.

```
public class paperHelper {
  public static void insert(
    org.omg.CORBA.Any a, paper t) {...}
  public static paper extract(Any a) {...}
  public static org.omg.CORBA.TypeCode type() {...}
  public static String id() {...}
  public static paper read(
   org.omg.CORBA.portable.InputStream istream) {...}
  public static void write(
   org.omg.CORBA.portable.OutputStream ostream,
   paper value) {...}
  public static paper narrow(
   org.omg.CORBA.Object obj) {...}
}
```

(그림 8) IDL 인터페이스에 대한 Helper 클래스
(Fig. 8) A Helper class for IDL interface

(그림 8)은 사용자 정의의 IDL 형이 되는 paper 인터페이스를 다루기 위해 필요로 하는 정적 메소드들을 제공해 주는 paperHelper 클래스이다.

paperHelper 클래스 내에는 자바 언어에서는 존재하지 않는 IDL의 any 형에 대한 삽입 및 추출을 하는 오퍼레이션들인 insert와 extract, repository id를 얻는 오퍼레이션인 id, typecode를 얻는 오퍼레이션인 type, 스트림에서 paper 인터페이스를 읽고 쓰는 오퍼레이션들인 read, write가 있다.

```
public class paperST extends iStub
 implements paper {
  public paperST() {...}
  public paperST(String serverName) {...}
...
  public boolean getstatus(java.lang.String title,
 IntHolder pID) {...}
...
}
```

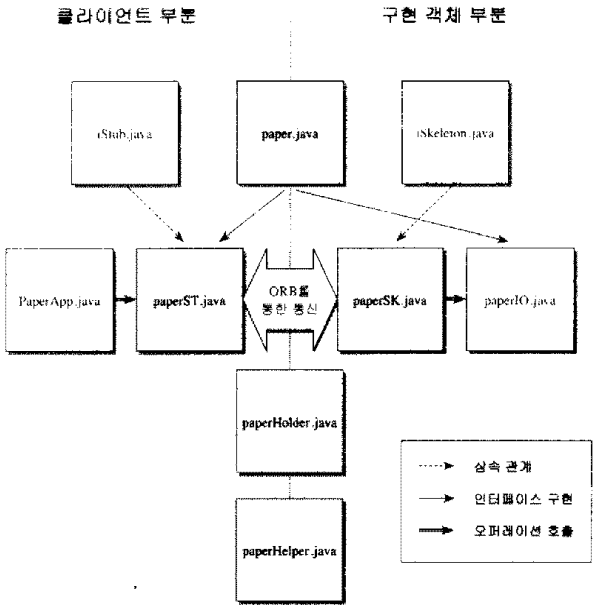
(그림 9) 코드 생성기에 의해 생성된 스텁 코드
(Fig. 9) A stub code produced by code generator

(그림 9)는 IDL 인터페이스인 paper에 대해서 생성된 클라이언트 스텁 코드이다. 클라이언트 스텁 코드는 IDL 인터페이스마다 하나씩 생성되며 IDL 인터페이스 내의 오퍼레이션들이 구현되어 있다. (그림 4)의 예제에서 생성된 스텁 파일내의 paperST클래스는 iStub 클래스로부터 상속을 받으면서 IDL 인터페이스에서 사상된 자바 인터페이스인 paper를 구현하는 기능을 한다. 실질적으로 인터페이스의 구성 요소인 오퍼레이션의 요청을 서버 스켈레톤 클래스에게 전달하게 된다.

```
public class paperSK extends iSkleton
 implements paper {
  public paperSK() {...}
  ...
  public String methodCall(iRequest aRequest)
  {...}
  ...
}
```

(그림 10) 코드 생성기에 의해 생성된 스켈레톤 코드
(Fig. 10) A skeleton code produced by code generator

(그림 10)은 paper 인터페이스에 대한 객체 구현 측의 스켈레톤 코드이다. 생성된 스켈레톤 파일은 iSkeleton 클래스로부터 상속받아 스텁 코드에서 요청되어 수행되고자 하는 해당 객체의 오퍼레이션을 (그림 10)의 methodCall에서 호출하며 오퍼레이션 수행 결과를 리턴 값이나 출력 및 입출력 매개변수를 통하여 다시 스텁 코드에게 전달해 준다.



(그림 11) 코드 생성기에 의해 생성된 파일의 관계
(Fig. 11) A relation among files produced by code gener-ator

생성된 파일들의 상속 및 내부 관계는 (그림 11)과 같다. (그림 4)의 논문 검색 IDL 예제에 대해 컴파일러에서 생성하는 파일들은 (그림 11)의 어둡게 칠해진 부분들인 paper.java, paperHolder.java, paperHelper.java, paperST.java, paperSK.java이다.

iStub.java와 iSkeleton.java는 컴파일러에서 생성되는 모든 스텁 및 스켈레톤 코드들에게 상속되는 클래스를 가지고 있으며 paperApp.java는 클라이언트 측 프로그램이고 paperIO.java는 서버 측 프로그램이다.

서버 측 파일인 paperIO 파일의 내용은 IDL 인터페이스에서 사상된 자바 인터페이스인 paper를 구현하여 실질적으로 논문 검색 서버 객체의 서비스를 수행하는 클래스이며 서버 객체의 서비스를 받는 클라이언트 응용 프로그램인 paperApp.java에서는 대리 객체 역할을 하는 스텁 파일내의 paperST 클래스의 인스턴스를 만들어 서버 객체에게 서비스 받고자 하는 오퍼레이션인 getstatus를 호출하는 부분이 포함된다.

4. 결론 및 향후 계획

본 논문에서는 향후 인터넷에 사용될 CORBA 시스템들 중에서 주종을 이루게 될 것이라고 전망되는 자바로 구현한 CORBA 시스템을 위한 IDL-to-Java 컴파

일러를 구현하기 위해 입력 파일인 IDL 파일의 구조를 분석하였고 IDL-to-Java 사상 규약에 따른 클라이언트 관련 파일들과 객체 구현 관련 파일들을 생성해주는 Java ORB 시스템의 IDL-to-Java 컴파일러를 구현하였다.

구현한 컴파일러의 구조는 입력 IDL 파일을 정규 표현에 해당하는 토큰 스트림으로 나누는 어휘 분석기, 토큰 스트림의 올바른 문법 사용 여부를 검사해주며 중간 코드를 생성하는 파서, 중간 코드로부터 해당 자바 언어로 사상된 코드뿐만 아니라 클라이언트에서 사용되는 스텁 코드, 서버 객체에서 사용되는 스켈레톤 코드들을 생성해 주는 목적 코드 생성기로 나누어진다.

향후 본 컴파일러에서 생성되는 데이터형과 인터페이스들에 대해서 인터페이스 저장소에 등록할 수 있는 기능들을 보완할 예정이며 사용자들이 더욱 편리하게 CORBA를 이용할 수 있기 위한 컴파일러의 기능도 연구하고 있다.

참 고 문 헌

[1] A. V. Aho, R. Sethi, and J. D. Ullman, "Compilers Principles, Techniques, and Tools", Addison-Wesley, 1986.

[2] G. Brose, "An Introduction to Programming with JacORB", http://www.in..fu-berlin.de/~brose/jacorb/tutorial.ps, 1997.

[3] G. Brose, "JacORB - A Java Object Request Broker", TR B 97-2, April 1997.

[4] IONA Technology Ltd,, "OrbixWeb Programmer Guide", 1998.

[5] IONA Technology Ltd., "OrbixWeb Reference Guide", 1998.

[6] J. R. Levine, T. Mason, and D. Brown "lex & yacc", O'Reilly & Associates, Inc., 1993.

[7] T. J. Mowbray and R. C. Malveau "CORBA Design Patterns", John Wiley & Sons, Inc., 1997.

[8] T. J. Mowbray and R. Zahavi, "The Essential CORBA : Systems Integration Using Distributed Objects", John Wiley & Sons, Inc., 1995.

[9] A. Nisar and H. G. Dietz, "Optimal code scheduling for multi-pipeline processors". In Proc of 1990 Int'l Conf. on Parallel Processing, Volume II, pages 61-64, St. Charles, IL, August 1990.

[10] OMG, "The Common Object Request Broker: Architecture and Specification, Revision 2.0", July 1995.

[11] OMG TC Document, "IDL-to-Java Language Mapping : Joint Revised Submission", March, 1997.

[12] R. Orfali, D. Harkey, and J. Edwards, "The Essential Distributed Objects Survival Guide", John Wiley & Sons, Inc., 1996.

[13] T. Parr, H. Dietz, and W. Cohen, "PCCTS Reference Manual, Version 1.0", ACM SIGPLAN Notices, 27(2):88-165, February 1992.

[14] J. Siegel, "CORBA Fundamentals and Prog-ramming", John Wiley & Sons, Inc., 1996.

[15] Visigenic, "Visibroker for Java Programmer Guide", 1997.

[16] Visigenic, "Visbroker for Java Reference Guide", 1997.

[17] 박성진, 이동헌, 김영곤, 박양수, 이명준, "ReCA CORBA 시스템을 위한 IDL 컴파일러", 한국정보과학회 97 춘계 학술발표논문집 제24권 1호, 1997.

이 진 호

1997년 성균관 대학교 정보공학과(학사)

1997년~현재 성균관대학교 정보공학과 석사과정

관심분야 : 분산 실시간 시스템, Real-Time CORBA

이 근 영

1986년 대전산업대학교 전자계산학과(학사)

1983년~현재 한국전자통신연구원 선임기술원

관심분야 : 분산 객체 기술, 멀티미디어

정 태 명

1981년 2월 연세대학교 전기공학과(학사)

1984년 5월 일리노이 주립대학 전산학과(학사)

1988년 12월 일리노이 주립대학 컴퓨터공학과(석사)

1995년 8월 퍼듀대학 컴퓨터공학과 박사

1985년~1987년 Waldner & Co., Systems Engineer

1987년~1990년 Bolt Beranek and Newman, Staff Scientist

1995년~현재 성균관대학교 정보공학과 조교수

관심 분야 : 실시간 시스템, 컴파일러, 네트워크 관리