분산 환경을 위한 코드 생성

박찬모*, 정성옥**, 이준***
*조선대학교 컴퓨터공학과
**광주여자대학교 인터넷정보학과
***조선대학교 컴퓨터공학과

Generating Code for Distributed Environment

Chanmo Park*, Sung Ok Jung**, Joon Lee***
*Dept. of Computer Engineering, Chosun Univ.
**Dept. of Internet Information, Kwangju Woman's Univ.
***Dept. of Computer Engineering, Chosun Univ.

E-mail : uljima@msn.com, jlee@stmail.chosun.ac.kr, sojung@kwu.ac.kr

요 약

본 논문에서는 분산 환경을 위해 OMG에서 발표한 CORBA의 표준에 따라 하는 컴파일러의 전반부를 위해 OMG IDL 컴파일러를 사용하였다.

CORBA에서 클라이언트와 서버간의 상호 동작을 위해 인터페이스를 정의한다. CORBA에서 인터페이스는 IDL(Interface Definition Language)를 통해 정의를 할 수 있다. IDL은 CORBA의 인터페이스를 기술하기 위한 추상 표기법이므로 개발에 사용되는 언어와는 연관성이 없다. 이를 개발에 사용하기 위해서 해당 언어로 맵핑을 하여야 한다.

본 논문은 IDL로 정의된 인터페이스를 C++로 맵핑하도록 하여 분산 객체 환경을 지원하도록 하기 위한 것이다. IDL 컴파일러는 IDL 정의를 입력 받아 어휘 및 구문 분석을 한 후 AST 트리를 생성하며, 생성된 각 노드를 통해 맵핑된 C++ 코드를 생성토록 한다.

1. 서론

어플리케이션 개발에 대한 요구들은 더 이상 중앙 집중식으로서는 빠른 시간내에 개발해서, 유지하는 것이 어렵게 되고 있다. 그러므로 어플리케이션은 유지가 용이하고, 개발하기에 적당한 정도로 분할되어 네트워크상에 분산되어 개발될 필요가 있다. 이것은 현재 계속 증가하는 네트워크의 속도와 함께 중요한 이슈가 되고 있다.

하지만 분산 어플리케이션은 개발에 몇 가지 문

제점을 갖고 있다. 개발자의 능력에 의지하여 개발된 어플리케이션의 컴포넌트들이 서로 다른 언어를 사용하므로 이질적으로 되며 하드웨어 종속성을 갖게 된다. 따라서 이러한 문제점들을 해결하기 위한 방안으로 미들웨어가 필요하다. CORBA(Common Object Request Broker Architecture) 또한 이들 미들웨어의 하나이다.

본 논문은 미들웨어인 CORBA상에서 IDL (Interface Definition Language)을 사용한 인터페이스 정의를 실제 개발자들이 이용하는 C++언어로 변환하는 IDL 컴파일러를 구현한다. 본 논문의 구성은 CORBA와 IDL 컴파일러에 대해 간략히 살펴보고, IDL 컴파일러 구현을 제시한다. 마지막으로 결론 및 향후 연구 과제에 대해 논의한다.

2. CORBA 와 IDL 컴파일러

2.1 CORBA

MG의 CORBA는 분산 환경상에 존재하는 객체간의 상호 운용성을 지원하며 요청과 응답의 투명성을 제공한다. CORBA 시스템에서 클라이언트는 서비스에 대한 요청의 생성자이며 객체 구현(Object Implementation)은 서비스 요청에 대한 처리자이다. 서비스의 처리 과정은 먼저 클라이언트가 클라이언트 스터브나 동적 호출 인터페이스(Dynamic Invocation Interface)를 통하여 요청을 생성하며, 생성된 요청은 ORB(Object Request Broker)를 통하여 객체 구현쪽으로 전달된다. 전달된 요청은 구현 골격이나 동적 골격 인터페이스(Dynamic Skeleton Interface)를 통하여 객체 구현에게 전달되어 서비스가 처리된다. 처리의 결과인 응답은 ORB를 거쳐서 클라이언트에게 전달된다 [1][2].

CORBA는 다음과 같은 요소들로 구성된다.

ORB

ORB는 객체에게 요청을 전달하고 요청을 생성한 클라이언트에게 응답을 전달하는 부분이다. 일반적으로 ORB는 객체 위치, 객체 구현, 객체 실행 상태, 통신 메커니즘에 대한 투명성을 제공한다[1][2].

인터페이스 정의 언어(IDL)

객체의 인터페이스는 객체가 지원하는 오퍼레이션과 형을 명시한다. CORBA에서는 객체의 인터페이스를 언어 독립적인 IDL을 이용하여 기술한다. OMG IDL은 프로그래밍 언어가 아니라 선언적 언어(declarative language)이다. 이것은 객체가 서로 다른 프로그래밍 언어로 구현될 수 있도록 한다.[1][2]

클라이언트 스터브와 구현 골격

클라이언트 스터브는 클라이언트가 정적으로 요청을 생성하고 전달하는 메커니즘이며 구현 골격은 정적 요청을 객체 구현으로 전달하는 메커니즘이다. IDL 컴파일러가 OMG IDL 인터페이스 정의를 이용하여 클라이언트 스터브와 구현 골격을 생성한다.

동적 호출

CORBA 시스템은 실행시에 인터페이스 정보가 저장된 인터페이스 저장소(Interface Repository)를 이용하여 동적 호출 인터페이스(Dynamic Invocation Interface)를 통하여 요청의 생성을 지원한다. [1][2]

2.2 IDL 컴파일러

CORBA 시스템에서 응용프로그램 개발 단계는 다음과 같다.

단계 1) IDL로 객체의 인터페이스를 정의한다. IDL은 클라이언트 객체가 호출하고 객체 구현에서 제공하는 인터페이스를 기술하는데 사용하는 언어이다. IDL은 객체의 인터페이스, 오퍼레이션, 형을 정의하는데 특정 프로그램 언어에 독립적이며, 캡슐화, 인터페이스 다중 상속, 객체지향적 예외 처리 등의 객체지향 개념을 지원하며, 구현을 위한 언어가 아니라 인터페이스를 기술하기 위한 선언적 언어이다. IDL은 분산 환경에서 특정 프로그래밍 언어에 독립적으로 CORBA 응용프로그램을 작성할 수 있게 해준다. 바꾸어 말하면 일단 IDL로 작성된 인터페이스 정의는 CORBA 시스템에서 제공하는 IDL 컴파일러를 통해 원하는 프로그래밍 언어에 적합하게 변환된다. [1][2]

단계 2) 클라이언트 스터브와 구현 골격을 생성한다. IDL 컴파일러는 IDL로 정의한 객체 인터페이스로부터 프로그래밍 언어로 된 클라이언트 스터브와 구현

골격을 생성한다. 클라이언트 스터브와 구현 골격은 서로 다른 언어로 작성될 수 도 있다. 클라이언트 스터브는 IDL로 정의한 오퍼레이션이 클라이언트가 호출 할 수 있는 특정 프로그래밍 언어로 변환된 인터페이스이다. 클라이언트 스터브는 각 인터페이스에 대한 요청을 생성하는데 특히, 인터페이스 호출에 필요한 메서드 선언을 제외한 나머지 부분을 내부적으로 감추며, 특정 ORB에 대하여 최적화된 상태로 만들어진다. 구현 골격은 객체 어댑터(Object Adapter)에 의존적으로 작성된 객체의 메서드에 대한 인터페이스이다. 이 인터페이스를 호출하여 ORB는 요청을 객체 구현에 전달한다. 이 인터페이스를 업-콜(up-call) 인터페이스라 한다. [1][2]

단계 3) 개발자가 적절한 코드를 추가하여 완전한 응용프로그램을 개발한다.

단계 4) IDL 컴파일러를 통하여 생성된 코드에 개발자가 적절한 코드를 추가하여 완전한 CORBA 응용프로그램을 작성하게 된다.

[그림. 1]은 CORBA 응용프로그램의 개발과정에서 IDL 컴파일러의 역할을 나타낸 것이다.

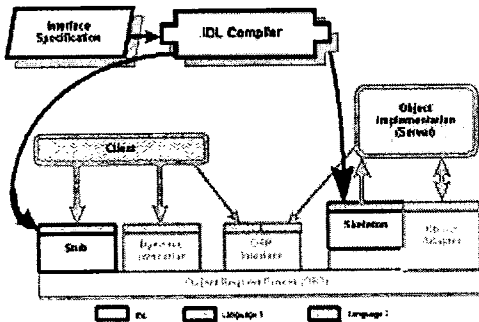

그림 1. IDL 컴파일러의 역할

Fig.1 Role of IDL Compiler

IDL 컴파일러는 특정 CORBA 시스템을 위해 정의된 클라이언트 스터브와 구현 골격 변환 규약을 지원하는데, 분산 응용프로그램을 보다 신속히 개발하기 위하여 개발자의 코드 추가를 최소화하는 방향으로 구현되어야 한다. IDL 컴파일러를 개발하기 위해서는 먼저 IDL을 특정 CORBA 시스템에서 지원하는 프로그래밍 언어로 작성하기 위한 클라이언트 스터브와 구현 골격 변환 규약을 확정하여야 한다. 이 규약은 CORBA 시스템의 ORB 구현에 따라 달라지게 되는데 IDL 컴파일러는 정의한 변환 규약에 따라 IDL로 정의된 인터페이스를 특정 프로그래밍 언어로 작성된 클라이언트 스터브와 구현 골격으로 생성하게 된다.[1][2]

3. IDL 컴파일러 구현

코드 생성은 CFE IDL에서 생성한 AST를 입력으로 받아 트리의 각 노드를 운행하면서 필요한 코드를 생성하고 이 코드로 구성되는 각각의 파일을 생성한다. 우리는 구현 단계를 설명하기 간단한 IDL 정의로 [그림.2]를 사용한다.

```
interface Intf1 {
string op1(in string arg1);
void op2(in string srg2, out string arg3);
};
```

그림 2. example.idl

Fig. 2 example.idl

[그림.2]에 제시된 IDL 정의는 [그림.3]과 같은 AST를 생성한다[3][7].

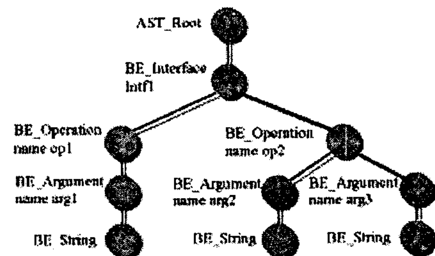

그림 3. AST 트리

Fig. 3 AST Tree

[그림.3]의 각 노드들은 IDL CFE의 전반부에서 사용되는 클래스들을 상속한 클래스의 인스턴스들이다. 예를 들어, 클래스 BE_Interface는 AST_Interface를 상속한 것이다. 컴파일러 후반부를 구성하는 주요 클래스들은 다음과 같다.

오퍼레이션의 인자를 나타내고 처리하는 BE_argument 클래스

상수를 나타내고 처리하는 BE_constant 클래스

나열형을 나타내고 관리하는 BE_enum 클래스

예외 정의를 나타내고 관리하는 BE_e xception 클래스

인터페이스를 나타내고 관리하는 BE_int erface 클래스

모듈을 나타내고 관리하는 BE_module 클래스

오퍼레이션을 나타내고 관리하는 BE_oper ation 클래스

시퀀스를 나타내고 관리하는 BE_sequence 클래스

구조체를 나타내고 관리하는 BE_structure 클래스

타입 정의를 나타내고 관리하는 BE_type def 클래스

몇 개 클래스에 범용적인 유틸리티 함수들

코드 생성은 함수 BE_produce()를 통해 이루어진다. 이 함수는 [그림.4]와 같은 파일을 생성한다. 이 파일은 omniORB를 이용하기 위해 기존의 omnidl 컴파일러의 결과와 동일하게 생성토록 구성한다.[8]

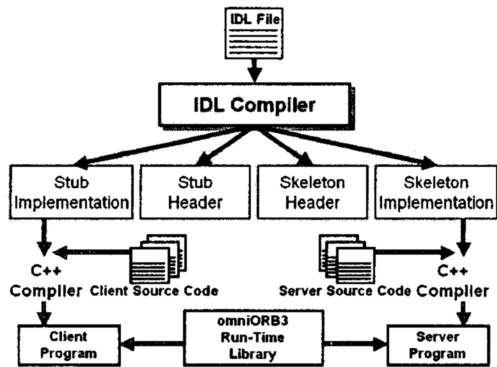

그림 4. 생성 파일

Fig. 4 Generated Files

example.hh 파일은 각 인터페이스에 대한 맵핑을 정의한다. 이 파일은 클라이언트 스터브와 서버 구현 클래스와 관련된 내용을 갖는다. 이 파일은 다음과 같은 과정을 통해 생성한다.

1. IDL 입력의 파일 이름을 사용하여 example.hh 파일을 생성한다.

2. 입력된 파일이름을 사용하여 omniORB를 사용하기 위해 "#include <omniORB3/ CORBA.h>"를 포함하도록 헤더를 생성 한다.[1]

3. idl_global->root()를 통해 최상위 노드인 AST_Root에서 시작하여 각각의 노드를 트래버셜하여 모듈, 시퀀스, 열거형, 예외등등의 노드가 발견되면 이에 대한 선언을 추가하고 각 인터페이스 정의에 대해 매크로 정의, 포워딩, 예외, 그리고 typedef등등에 대한 선언을 생성한다.

4. 인터페이스 노드가 발견되면 해당하는 인터페이스의 Helper 클래스를 생성한다.

5. 인터페이스 객체 참조 변수 및 인터페이스의 out 인자로 사용되는 클래스를 정의한다.

6. 인터페이스에 대한 클래스를 정의한다.

7. 인터페이스 참조 클래스를 정의한다.

8. 인터페이스의 프락시 클래스를 정의한다.

9. 인터페이스의 서버측 구현 클래스를 정의한다.

10. POA_Interface_name 클래스를 정의한다.

11. 인터페이스 정의 클래스에서 사용하는 마샬링 코드를 정의한다.

위와 같은 과정을 통하여 생성되는 example.hh 파일은 [그림.5]와 같다.

```
#include <omniORB3/CORBA.h>
#ifndef __Intf1__
#define __Intf1__
class Intf1;
class _objref_Intf1;
class _impl_Intf1;
typedef _objref_Intf1* Intf1_ptr;
typedef Intf1_ptr Intf1Ref;
class Intf1_Helper {......};
typedef
_CORBA_ObjRef_Var<_objref_Intf1,          Intf1_Helper>
Intf1_var;
typedef
_CORBA_ObjRef_OUT_arg<_objref_Intf1, Intf1_Helper >
Intf1_out;
#endif
```

```
class Intfl {......};
class _objref_Intfl :
  public virtual CORBA::Object, public virtual omniObjRef
{
public:
  char* op1(const char* arg1);
  void op2(const char* srg2, CORBA::String_out arg3);

};
class _pof_Intfl
  : public proxyObjectFactory {... };
class _impl_Intfl :
  public virtual omniServant
{......};
```

그림 5. example.hh

Fig. 5 example.hh

위와 같은 내용의 파일은 파일을 관리하는 함수인 BE_produce_file()함수를 사용하여 열고, 해당 노드에 있는 메서드 dumpIncludeFile()를 사용하여 각 내용을 출력토록 한다.

exampleSK.cc 파일은 각 인터페이스 맵핑에 대해 정의된 구현 코드를 갖는다. 이 파일은 다음과 같은 과정을 통해 생성한다.

1. IDL 컴파일러의 전반부에서 입력된 파일 이름을 사용하여 헤더 파일인 example.hh 를 포함시킨다.

2. Intfl_Helper 클래스의 구현 부분을 정의한다. 이것은 BE_interface 클래스의 dumpHelper()에 의해서 이루어진다.

3. 인터페이스에 대한 객체에 대한 구현을 정의한다. 이것은 dumpInterfaceCode()를 사용하여 코드를 정의한다.

4. 인터페이스 객체 참조 클래스에 대한 구현 코드를 정의한다. DumpInterfaceRefCode()를 사용한다.

5. 클라이언트 프락시 클래스의 구현을 정의한다. 이것은 dumpProxy()를 사용하여 생성한다.

6. 서버 구현에 사용되는 클래스인 구현 클래스에 대한 코드를 정의한다. 이것은 dumpImplCode() 함수를 사용하여 생성한다.

4. 결론

본 논문에서는 IDL 정의를 입력 받아 파싱하는 컴파일러의 전반부를 위해 OMG IDL 컴파일러를 사용하였다. 또한 ORB를 위해 omniORB3를 사용했다. OMG IDL CFE는 IDL 정의를 입력 받아 어휘 및 구문 분석을 한 후 AST 트리를 생성하며, 생성된 각 노드는 우리가 새로 추가한 BE_* 클래스의 인스턴스로 구성된다. IDL 컴파일러의 후반부는 AST의 각 노드를 반복자인 UTL_ScopeActiveIterator 클래스를 사용하여 반복적으로 각 순회하면서 해당하는 출력을 덤프한다. 이때 두개의 출력 파일을 생성토록 했다. 모든 코드 생성은 BE_produce.cc에서 시작되며, idl_global->root() 노드를 시작으로 하여 각 클래스에 해당 코드를 생성하는 dump* 함수를 호출하여 생성했다.

본 논문은 IDL 정의를 C++언어로 맵핑을 실험했으며, 이것은 omniORB3에서 제공하는 IDL 컴파일러와 동일한 결과를 생성했으며, omniORB3 환경에서 동작하는 변환된 C++ 코드임을 실험했다. 향후 IDL 컴파일러를 통한 성능 향상을 위해 마샬링 코드의 최적화를 할 수 있도록 하는 코드 생성이 연구되어야 한다.

[참고문헌]

[1] OMG, "The Common Object Request Broker:Architecture and Specification", OMG, 1999

[2] 박성진, 이동헌, 김영곤, 박양수, 이명준, "ReCA CORBA 시스템을 위한 IDL 컴파일러", 한국정보처리학회 논문지, 제5권,2호, 1998, pp.437-449

[3] Sai-Lai Lo, David Riddoch, Duncan Grisby, "The omniORB Version 3.0 Users Guide", AT&T Laboratories Cambridge, 2000

[4] DIMMA Team, "DIMMA Design and Implementation", APM Ltd, 1997

[5] OMG, "C++ Language Mapping Specification", OMG, 1999

[6] Nigel Edwards, "A Stub Compiler for CGI and HTTP: The Programmer's Guide", APM Ltd, 1995

[7] SunSoft, "WRITING_A_BE (OMG_IDL_CFE /doc)", SunSoft, 1994

[8] Duncan Grisby, "omniidl The omniORB IDL Compiler", AT&T Laboratories Cambridge, 2000