상속 소프트웨어 시스템을 CORBA 환경에서 재사용하기 위한 객체 포장 기법의 설계 및 구현

황규대, 김현수
금오공과대학교 컴퓨터공학부

Design & Implementation of Object Wrapping Techniques for Reusing Legacy Software System on CORBA Environment

Gyu Dae Hwang, Hyeon Soo Kim
School of Comp. & Soft. Eng., Kumoh Nat'l University of Technology

요 약

상속(Legacy) 소프트웨어 시스템은 오랜 기간 사용되었고 충분히 검증된 안정적인 서비스를 현재까지도 제공하는 유용한 시스템이다. 새로운 분산 객체 환경에서 기존의 시스템에서 제공하는 서비스를 사용하기 위한 방법으로, 기존 시스템을 대체할 새로운 시스템을 개발하는 방법과 기존 시스템의 코드를 수정하는 방법과 기존 시스템을 객체 포장기법으로 포장해서 사용하는 방법이 있다. 본 논문은 이 중에서 기존 시스템을 객체로 포장하여 분산 객체 기술인 CORBA 환경에서 이 시스템을 재사용하는 방법에 대하여 연구한다. 이 과정에서 다양한 형태의 인터페이스를 가진 기존 시스템을 효과적으로 포장할 수 있는 방법으로 LWR(Legacy Wrapping Rule)을 제안하고, 랩퍼(Wrapper)인 구현 객체 클래스를 만드는 랩퍼 생성기를 구현하였다. 이렇게 함으로써 상속 시스템을 보다 쉽고 강력하게 분산 환경으로 이주시킬 수 있다.

1. 서론

최근의 급격한 인터넷의 이용 증가 등으로 인해 사용자의 요구를 만족시키기 위한 개념이 단일 시스템 환경에서 네트워크를 매개로 연결된 분산 시스템 환경으로 빠르게 이동하고 있다. 그런데 새로운 프로젝트를 추진하거나 기존의 단일 시스템을 분산 환경으로 변경시킬 때 기존의 사용되고 있는 시스템을 소홀히 하고 있는 경우가 많다 기존의 시스템을 새로운 환경에서 충분히 재사용할 수 있음에도 불구하고 통합의 어려움을 이유로 기존의 시스템을 대체할 새로운 시스템을 구축하거나 기존의 시스템을 확장함으로써 불필요한 비용과 시간과 인력이 추가된다. 이런 상황에서 CORBA는 기존의 소프트웨어 시스템을 변경하지 않고 그대로 재사용할 수 있게 하는 기술을 제공한다. 이는 CORBA의 IDL을 이용해서 클라이언트 이용자에게 서비스를 제공하기 위해 기존의 상속(Legacy) 시스템을 감싸는 객체를 작성함으로써 구현할 수 있다 이러한 기법을 CORBA 객체 포장 기법이라 한다[3, 4]. 본 논문에서는 이 객체 랩퍼를 자동 생성하는 생성기를 구현한다. 또한 상속 시스템을 포장함에 있어서 기존 시스템이 제공하는 다양한 인터페이스를 클라이언트에게 투명하게 효과적으로 포장하는 방법을 제안한다

2. 관련 연구

2.1 상속(Legacy) 시스템

일반적으로 상속 시스템은 일반 사회에서 중요한 역할을 하고 있고 규모가 크고 객체 개념 없이 구현되었고 현재에도 유용한 시스템으로 정의한다. 이런 오래된 시스템들의 공통적인 특징은 아래와 같다[3].

- 특정 용도로 개발되어 사용되고 있다
- 일반적으로 전통적인 관계 데이터베이스를 이용하거나 파일 시스템을 이용한다.
- 규모가 크고, 복잡하고, 단일 호스트 구조로 되어있다.
- 기업 환경에서 중요한 역할을 담당하고 있다.
- 시스템이 개발되던 시대에 사용되던 기술에 의존한다.

2.2 이주(Migration) 전략

상속 시스템을 분산 환경으로 이주하는 방법으로 아래의 세 가지가 있다[2].

(1) 대체(Replacement)

상속 시스템이 제공하는 서비스를 제공하는 새로운 시스템을 개발하는 것이다. 현재 가장 많이 사용되고 있는 방법이다. 이 방법의 문제점은 상속 시스템의 기능을 완전히 대체하기가 어렵고 개발비용이 많이 들고 안정화에 많은 시간이 요구된다.

(2) 확장(Extension)

상속 시스템에 새로운 기능을 추가하거나 기능을 수정함으로써 시스템을 새로 개발하는 것보다 개발비용을 줄일 수 있다. 그러나 기존 아키텍처를 유지함으로써 발생하는 근본적인 한계는 벗어날 수 없다.

(3) 통합(Integration)

시스템의 구조를 그대로 유지한 채 새로운 환경으로 상속 시스템을 통합한다 상속 시스템을 포장하고 클라이언트에게는 추상 인터페이스를 제공하여 이를 통하여 접근하게 한다 이 방법은 효과적인 개발을 지원하고 기존 시스템이 제공하는 여러 장점을 그대로 이용할 수 있다.

2.3 객체 포장(Wrapping) 기법[1,5]

· 계층화(Layering)
포장의 가장 기본적인 형태로서 상속 시스템의 인터페이스를 새로운 시스템의 인터페이스로 일대일 대응한다.
데이터 이주(Data Migration)
기존 데이터 모델을 새로운 데이터 모델로 전환시키는 데 사용되는 포장 기술이다.
· 응용프로그램 재공학(Reengineering Applications)
상속 시스템을 분석하고 재 구현해서 새로운 시스템을 개발한다
· 미들웨어(Middleware)
미들웨어는 다양한 데이터베이스 시스템과 사용자 인터페이스에 대해 공통 접근 메커니즘을 제공한다
· 캡슐화(Encapsulation)
가장 일반적인 형태로서 구현으로부터 인터페이스를 분리함으로써 추상화를 제공한다.

3. 시스템의 설계

본 논문에서 구현하는 시스템은 그림 1과 같은 형태로 구성된다

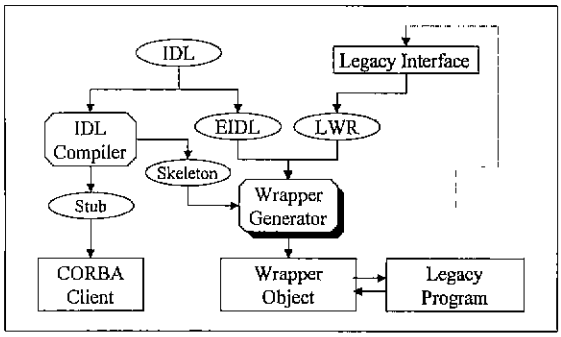

그림 1. 시스템 전체 구성도

3.1 EIDL(Extended Interface Definition Language)

상속 시스템이 랩퍼 객체를 통해서 클라이언트에게 제공할 기능에 대한 인터페이스를 기술한 것이 IDL(Interface Definition Language)이고 그 기능에 접근하기 위해 상속 시스템의 실제 인터페이스를 기술한 부분이 추가된 것이 EIDL(Extended IDL) 이다

(1) IDL 부분
CORBA의 IDL을 제한적으로 사용한다. 클라이언트는 랩퍼 객체가 제공하는 인터페이스에만 접근할 수 있고 실제로 상속 시스템에 대한 접근은 랩퍼 객체를 통해서 이루어진다 이렇게 함으로써 어떤 형태로 작성된 상속 시스템일지라도 클라이언트가 쉽게 접근할 수 있는 방법을 제공한다. IDL 오퍼레이션에서 in 형식의 매개변수는 랩퍼 객체를 통하여 상속 시스템으로 전달되며 inout 형식의 매개변수를 통하여 클라이언트의 요구를 상속 시스템이 처리한 결과(파일이나 표준 출력 등)를 클라이언트로 돌려준다.

```
module Wrapper {
    interface C_lang {
        string compile (in string inFile, inout string outFile);
            // <- "CMDLN · cc $0 -o $1" ->

    },
    interface itp {
        string open(in string host, in string port);
            // <- "SOCKET · open $0 $1" ->
        ...
    },
},
```

그림 2. EIDL 정의 예

(2) 상속 시스템 인터페이스 부분
추가된 상속 시스템에 대한 인터페이스에서 첫 번째 부분은 상속 시스템의 인터페이스의 형식을 기술하고 두 번째는 그 인터페이스를 세분화한 오퍼레이션을 기술한다. IDL 부분의 오퍼레이션의 매개변수는 상속 시스템에 대한 인터페이스의 변수로 대치된다. 랩퍼 객체 자신이 클라이언트에게 오퍼레이션 수행 성공 여부를 해당 오퍼레이션의 string 형식으로 반환한다.

3.2 LWR(Legacy Wrapping Rule)

EIDL의 상속 시스템에 대한 인터페이스의 실제 구현 규칙은 LWR로 정의되어 있다.

3.2.1 LWR 구조

LWR은 크게 아래의 세 부분으로 구성되어 있다(그림 3 참조).
(1) 상속 시스템 인터페이스 타입
상속 시스템에 접근하기 위해 사용하는 인터페이스 타입을 선언하는 부분이다 인터페이스 타입 종류에 따라 랩퍼 객체를 구현하는 방법이 결정된다
(2) 오퍼레이션
해당 인터페이스를 구성하는 오퍼레이션을 선언하는 부분이다. 클라이언트에게 제공하는 서비스의 기본 단위이다. 인터페이스를 이루는 오퍼레이션으로 분할하고 이를 적절하게 조합해서 사용함으로써 인터페이스를 통째로 사용하는 것보다 효율적으로 상속 시스템의 서비스를 활용할 수 있다.
(3) 구현 코드
각각의 오퍼레이션에 실제로 들어갈 자바 언어로 된 랩퍼 코드 부분을 정의하는 부분으로 되어 있다.

```
type legacy_interface_name {  // -- (1) --
    declare {
        ... // import 선언 부분
        ... // 클래스 변수 선언 부분          }
    operation operation_name1 (arguments) { // -- (2) --
        ... // IDL operation에 구현될 자바 코드 // - (3) -
    }
    operation operation_name2 (arguments) {
        ..                                   }
    ...
}
...
```

그림 3. Legacy Wrapping Rule

3.2.2 LWR 규칙 정의

랩퍼 생성기는 EIDL 파일에 선언되어 있는 상속 시스템에 대한 인터페이스를 가지고 LWR을 참조하여 랩퍼 객체 코드를 생성한다. 여기서 LWR을 랩퍼 생성기에 내장하지 않고 파일로부터 읽어 오게 하는 이유는 상속 시스템에 접근하기 위해 제공되는 인터페이스가 시스템에 따라 아주 다양하기 때문에 랩퍼 생성자에게 랩퍼 개발 시점에서 융통성 있는 개발 환경을 제공하기 위해서이다. 개발자는 필요한 경우 직접 LWR 파일에 사용자 정의의 규칙을 추가하거나 이미 정의되어 있는 규칙을 변경함으로써 좀더 유연하게 랩퍼 객체를 생성할 수 있다 그림

4처럼 다양한 상속 시스템에 대한 인터페이스를 랩퍼 객체로 포장해서 캡슐화 함으로써 클라이언트에게 투명한 서비스를 제공할 수 있게 된다

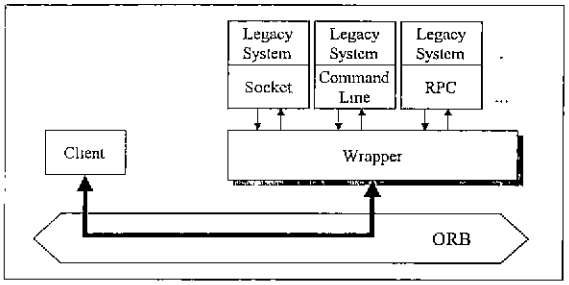

그림 4. 랩퍼를 이용한 상속 시스템의 사용

3.3 랩퍼(Wrapper) 생성기

랩퍼 생성기는 아래의 세 단계 과정으로 랩퍼 객체를 만든다.

(1) EIDL 구문 분석 단계

랩퍼 생성기는 EIDL 파일 내에서 클라이언트에게 제공할 인터페이스(IDL부분)와 그 인터페이스에 대응하는 상속 시스템에 대한 인터페이스를 추출한다.

(2) 변환 단계

위에서 추출한 상속 시스템 인터페이스를 LWR 파일에서 정의된 변환 규칙을 이용하여 자바 언어 코드를 생성한다.

(3) 랩퍼 객체 코드 생성 단계

단계 (1)에서 추출한 IDL 오퍼레이션에 단계 (2)에서 생성한 자바 코드를 삽입하여 랩퍼 객체를 생성한다.

예를 들어 그림 2의 EIDL 파일에서 첫 번째 IDL 인터페이스는 C언어 컴파일러를 포장하기 위한 예이다. 랩퍼 생성기는 IDL의 오퍼레이션 즉 랩퍼 객체의 서비스 부분의 구현 코드를 생성하기 위해 EIDL에서 추가된 상속 시스템에 대한 인터페이스인 "cc $0 -o $1"를 추출하고 이에 대한 규칙을 LWR 파일을 통해 획득해서 그림 5와 같은 랩퍼를 생성한다.

```
public java.lang.String compiler(
    java.lang.String inFile,
    org.omg.CORBA.StringHolder outFile
) {
        .
        String cmd = "cc " + inFile + "  -o " + _outFile;
        FileWriter out = new FileWriter("./run.bat");
        out.write(cmd, 0, cmd.length());
        out.close();
        Process p = rt.exec("  /run.bat");
        p.waitFor();
```

그림 5 랩퍼 객체 생성 예

또한 랩퍼 객체에 클라이언트 사용자의 예상하지 못한 행동을 막기 위한 보안 체크 루틴도 포함하여 클라이언트가 매개변수를 통해 보안에 위배되는 행동을 못하게 함으로써 시스템의 보안을 유지한다.

3.4 랩퍼 객체 생성 전체 과정

(1) EIDL 작성

개발자는 상속 시스템을 포장하는 랩퍼 객체가 제공하는 서비스에 대한 인터페이스를 IDL 파일로 정의한다. 이 IDL을 IDL 컴파일러로 컴파일하여 스터브, 스켈러턴과 기타 자바 파일들을 생성한다. 이 IDL 파일내에 각각의 인터페이스에 대응하는 상속 시스템에 대한 인터페이스를 추가하여 EIDL 파일을 생성한다.

(2) 랩퍼 객체 생성

그후 개발자는 생성된 EIDL 파일을 기반으로 랩퍼 생성기를 사용하여 랩퍼 객체를 만든다. 이때 랩퍼 생성기는 EIDL 파일에서 상속 시스템에 대한 인터페이스를 LWR 파일을 이용하여 구현 코드를 생성하여 IDL 컴파일을 통해 생성된 자바 인터페이스의 해당 메소드를 구현함으로써 랩퍼 객체 코드가 생성된다. 이렇게 생성된 랩퍼 객체를 해당 ORB용 컴파일러로 컴파일하여 랩퍼 구현 객체를 만든다.

(3) 시스템 통합

개발자는 이렇게 생성된 랩퍼 구현 객체를 새로운 CORBA 환경으로 통합하여 재사용한다.

4. 구현

본 시스템은 Solaris 2.6, Windows 98 환경에서 구현되고 테스트되었으며 사용된 ORB는 Visibroker for Java 3.4와 OrbixWeb 3.0을 사용하였고 사용한 자바 버전은 JDK 1.1.7과 JDK 1.2이다.

상속 시스템이 동작하는 운영체제마다 랩핑 방법이 조금씩 틀리므로 랩퍼를 생성하기 전에 해당 운영체제에 대한 환경 설정 기능이 제공된다

5. 결론 및 향후 연구 과제

본 논문은 상속 시스템을 새로운 분산 환경으로 진화하거나 통합할 때 상속 시스템을 변경하지 않고 재사용할 수 있게 해주는 CORBA 기반의 객체 랩핑 방법에 관하여 논의하였다

본 논문에서 제안하는 방법은 상속 시스템을 접근할 때 이용하는 다양한 인터페이스를 IDL 인터페이스로 대응시키는 효과적인 랩핑 메카니즘을 제공함으로써 상속 시스템을 보다 쉽고 강력하게 분산 환경으로 이주할 수 있도록 하였다. 또한 랩핑 규칙을 확장성있게 설계함으로써 랩퍼 객체 생성 시에 유연성을 제공한다.

현재 시스템은 상속 시스템을 포장하는 랩퍼를 생성하는 부분을 담당한다. 앞으로 구현해야 할 내용은 이 시스템을 확장하여 이러한 랩퍼 객체를 이용하는 전체 분산 환경 시스템을 시각적으로 조립해서 생성할 수 있는 개발 환경을 제공하는 것이다.

[참 고 문 헌]

[1] 서봉원, 최은만, "객체 랩퍼 클래스를 이용한 RPC 프로그램의 재공학 기법", 정보과학회 논문지, 1999.3

[2] Harry M. Sneed, Encapsulating Legacy Software for Use in Client/Server Systems, WCRE'96, 1996

[3] Marco Battaglia, Giancarlo Savoia, John Favaro, "RENAISSANCE: A Method to Migrate from Legacy to Immortal Software Systems", CSMR'98, 1998

[4] Narsim Ganti, William Brayman, "The Transition of Legacy Systems to a Distributed Architecture", John Wiley & Sons, 1995.3

[5] Thomas J., Phd Mowbray, Ron Zahavi, "The Essential CORBA: System Integration Using Distributed Objects", John Wiley & Sons, 1995.8