'97 International Conference Multimedia Databases on Internet
October 10, 1997, Seoul Korea

인터넷 환경하의 분산 컴퓨팅을 위한 특수미들웨어로서의 객체포장 방법

(Object Wrapping as a special middleware of distributed computing in internet environments)

강대석 (현대정보기술(주))
임남홍(제임스마틴코리아)
조선형(왕컴퓨터코리아(주))

요 약

인터넷 환경하의 어플리케이션 구축이 확산되면서 기존시스템의 정보(Legacy sources)를 새롭게 전개되는 분산객체 환경의 시스템에서 접근이 가능하도록 하는 요구가 증대되게 되었으며, 이를 해결하기 위하여 WWW, 분산객체 미들웨어, 특수목적 미들웨어를 사용할 수 가 있다.

본 논문에서는 특수목적 미들웨어로서의 객체포장(Object Wrapping) 방법에 대해서 논한다. 기존시스템의 객체포장은 기존시스템에 대한 깊은 이해없이 접근을 숨기는 (Information Hiding) 방법이다. 기존시스템의 내용은 데이타와 기능을 포함하며 잘 정의된 인터페이스를 제공하므로써 기존시스템을 이용하려는 어떠한 클라이언트나 타 시스템에서 이용할 수 있도록 해준다. 객체포장은 기존시스템에 대한 깊은 이해 없이도 인터넷상에서 기존시스템의 서비스를 개발위험은 최소로 하면서 서비스의 중복이나 불일치 (Inconsistency)를 회피하며 구현하는 대안이다.

한 번 객체포장이 된 기존시스템은 그 서비스를 필요로 하는 어떠한 종류나 클라이언트의 숫자와 관계없이 클라이언트에 대하여 표준 인터페이스를 제공하는 미들웨어로서 인터페이스 외부의 환경에 대하여 다양한 융통성을 가지게 된다.

1. 개요

근래에 정보기술이 급속하게 발달되면서 분산처리시스템(Distributed Computing)이 Peer-to-Peer 모델에서 Client/Server 모델로 발전되고, WWW(World Wide Web)와 분산객체(Distributed Objects)가 C/S 에 접목됨으로써 새로운 분산처리기능이 제공되게 되었다. 이에 따라 실제적으로 Networking 기반의 성장, 재활용할 수 있는 조립형 미들웨어(off the shelves middleware), 어플리케이션 요소(application component)를 분산처리하여 경제적으로 정보시스템을 구축할 수 있는 환경이 확산되고 있다.

C/S 는 분산처리의 업계표준(De facto standard)으로 자리를 잡으며 네트웍 기반 컴퓨팅(Network centric computing)과 합세하여 ESC(Enterprise service center)의 새로운 장을 열어 나가고 있다.

그러나 이렇게 분산환경으로 전개되어 감에도 불구하고 분산시스템과 기존시스템(Legacy system)이 공존하는 문제는 여전히 그대로 남아 있다. 더욱이 인터넷 환경에서 급속하게 확산되는 분산처리는 기존 자료(Legacy data)를 접근하여 처리하는데 많은 어려움을 겪고 있다.

본 논문에서는 이러한 문제에 대한 대응책으로 Object Wrapping 을 제시한다.

2. WWW, 분산객체(Distributed objects)와 분산처리

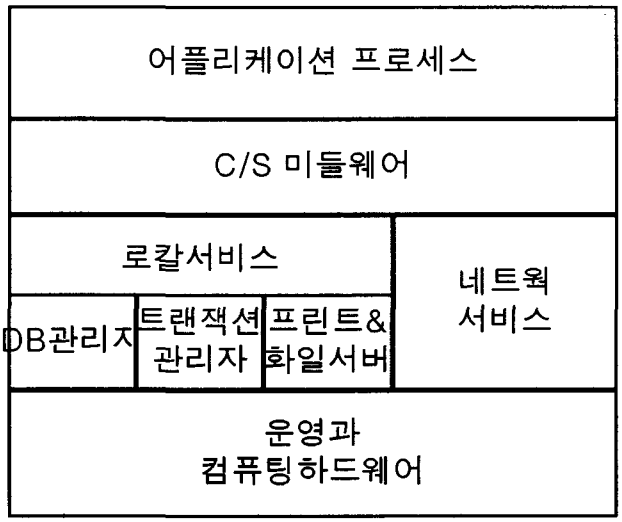
가. C/S 의 모델

C/S 개념은 오랫동안 비즈니스에서 사용되어 오는 개념이다. 예를 들어 식당에서는 고객이 클라이언트가 되고 식당주인이 서버가 된다. 전산분야에서는 C/S 모델이 컴퓨팅 프로세스간의 커뮤니케이션을 기술해 주는 개념이다. 즉, 서비스가 필요해서 요청하는 측이 클라이언트가 되고, 이에 대한 서비스를 제공해 주는 측이 서버가 된다. 클라이언트와 서버는 인터페이스상의 기능적인 모듈이며 이들간의 인터페이스는 메세지를 통해서 교환된다. 개념적으로 클라이언트와 서버는 동일한 기계나 별도의 기계에서 운영될 수 있으나 분산컴퓨팅 환경에서는 별도의 기계에서 운영되며 네트웍을 통하여 연결되어 상호운용된다. 따라서 모든 C/S 서비스 요청들은 네트웍 서비스를 통해서 실시간(real time)메세지가 교환된다.

개념적으로 C/S 모델은 분산컴퓨팅의 모델의 특별한 한경우이다. Peer-to-Peer 모델에서는 다른 사이트에서 프로세스를 기동하는 것을 허용한다. 그리고 클라이언트나 서버 또는 양쪽 어느편에서나 프로세스를 상호작용하게 수 있으나 C/S 모델에서는 한 프로세스는 서비스 제공자의 역할을 하고 다른 프로세스는 서비스 사용자의 역할을 하게 된다.

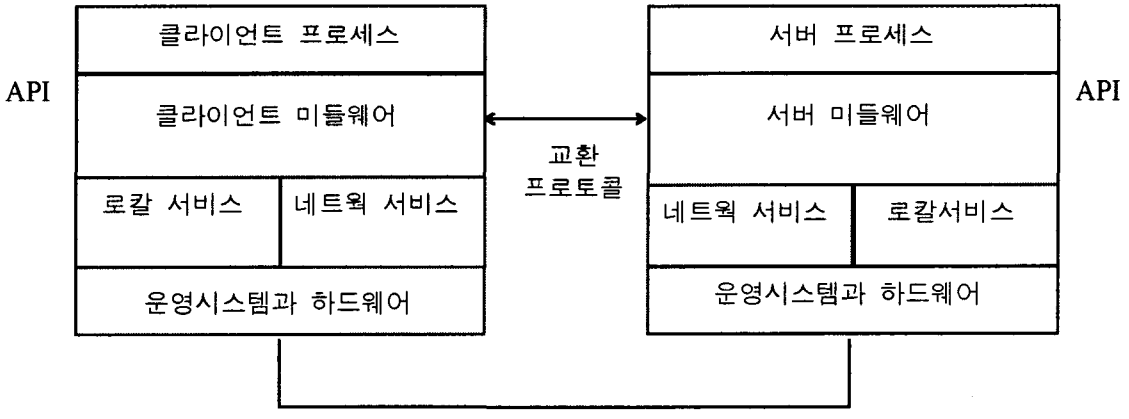
나. C/S 의 구조

C/S 환경에서 클라이언트와 서버간에 요청과 서비스와 관련된 메세지를 전달하기 위하여 통로가 필요한데 네트웍이 이러한 역할을 해주게 된다. 따라서 C/S 구조에서 네트웍의 중요성이 대단히 높으며 C/S 환경의 구성요소는 <그림 1>과 같다.



<그림 1> C/S 환경의 구성요소

C/S 모델은 분산컴퓨팅의 특수한 한 형태로 볼 수 있으며 두 가지 수준에서 구현될 수 있다. 하나는 비즈니스 서비스와 관련한 어플리케이션 서비스 구현이고, 다른 하나는 비즈니스와 관련 없는(컴퓨팅 플렛폼) 서비스 구현이다.



<그림 2> C/S 환경의 개념적 뷰(Conceptual View)

예를 들어 고객 데이타베이스가 서버에 위치하고 있고 클라이언트에서 우량고객에 대한 보고서를 만들고자 할때 클라이언트에서 원하는 조건에 맞는 SQL 문을 작성하여 클라이언트 미들웨어의 API 포멧으로 전달하게 된다. 클라이언트 미들웨어는 SQL 문을 받아 원격지에 있는 데이타베이스 서버 프로세스와 연결을 설정하게 되고, SQL 문을 특정한 형태로 네트웍 서비스에 요청하게 된다. 네트웍 서비스에서는 이를 메시지 형태로 원격지 컴퓨터의 네트웍 서비스에 전달하게 된다. 원격지의 서버에서는 네트웍 서비스에서 SQL 문을 받아 서버 미들웨어에 전달하게 된다.

서버 미들웨어에서는 클라이언트 미들웨어에서 보낸 SQL 문의 포맷(format)을 이해할 수 있어야 한다. 이러한 클라이언트/서버간의 교환 프로토콜(Exchange Protocol)이 C/S 상호운용성(Interoperability)에 주요한 역할을 하게 된다. 서버미들웨어는 이 요청에 대한 처리계획을 하게 되고, 이를 데이터베이스 관리자에 전달하여 이를 처리하게 되고 처리 결과를 상호간에 이해할 수 있는 프로토콜로 클라이언트 미들웨어에 전달해 주게 된다.

이렇게 클라이언트 프로세스는 클라이언트측에서 기능을 수행하게 되고 서버에 요청할 내용을 미들웨어에 전달하게 된다. 교환프로토콜을 위하여 사용되는 서비스로 RPC(remote-procedure call), RDA(remote-data access), MOM (message-oriented middleware)과 같은 기본 C/S 미들웨어가 있으며 WWW 미들웨어를 구성하는 것으로는 HTTP(Hypertext Transfer Protocol), HTML(Hypertext Markup Language), CGI(Common Gatewat Interface)가 있다. 또한 분산객체 미들웨어로 CORBA, OLE/ActiveX 가 있고, 새롭게 출현하는 특수목적(Special-purpose)의 미들웨어로 그룹웨어 미들웨어, 모빌컴퓨팅 미들웨어, 멀티미디어 미들웨어, Legacy-access/integration 미들웨어가 있다.

다. C/S 의 Middleware 가 WWW & 분산객체(Distributed Object)로의 변화

웹(Web)의 사용이 폭발적으로 증가하고 이에 따라 웹을 이용하여 기업의 경영목표를 달성하고자 하는 사례가 증가하고 있다. 이를 기업에 적용하기 위해서는 사용자들이 기업의 정보에 쉽게 접근하여 필요한 정보를 획득하고, 비즈니스 어플리케이션 운영을 지원할 수 있어야 한다.

그러나 대부분의 기업정보들이 플렛화일(텍스트화일, 그래픽자료), 인덱스 화일(VSAM), 관계형 DB, IMS 데이타베이스 형태로 저장되어 있기 때문에 이들을 활용하기 위한 방안이 마련되어야 한다. 즉, 웹브라우저는 HTML 화일을 디스플레이 하도록 고안되어 있으나 기업의 대부분의 고객정보나, 생산관리정보, 회계정보들은 HTML 화일형태로 저장되어 있지 않다.

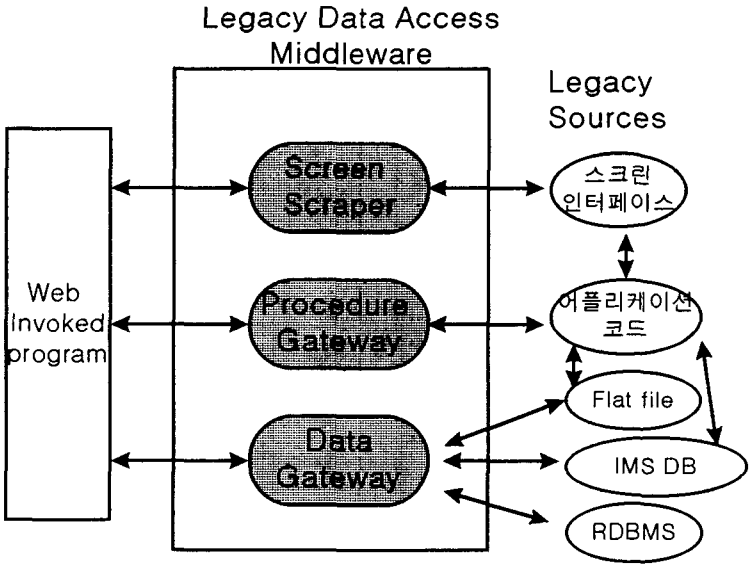
기업에서 웹의 사용이 전개되고 특히, 인트라넷(Intranet)의 적용이 확산되면서 기존시스템(Legacy system)과 인터넷 기반의 시스템을 완벽하게 연결해 주는 것에 대한 요구가 증대되고 있다.

기술적으로 말하면 WWW 이 네트웍(즉, 인터넷)위에서 운영되는 미들웨어들의 집합이다. 그러나 현재 WWW 를 위한 미들웨어로 웹브라우저나 HTTP, CGI 게이트웨이등 초보적인 수준의 요소들 만이 개발되어 있어서 관계형 데이타베이스 자료나 기본시스템 정보를 접근하기 위한 조립식 미들웨어의 개발이 필요한 실정이다.

이러한 문제를 해결하기 위하여 분산객체를 이용하여 기존시스템의 정보를 접근하거나 특수목적의 미들웨어를 이용할 수 있다.

기존데이타 접근을 위한 미들웨어는 <그림 3>과 같이 분류할 수 있다.

- 기존데이타를 직접 접근하는 데이타게이트웨이(SQL 호출 발생)
- 기존어플리케이션의 프로스져(기능)를 기동시키는 프로스져(기능)게이트웨이 (IMS 프로스져 기동)
- 기존 스크린서비스를 직접 접근하는 스크린스크랩퍼(Screen scrapers)



<그림 3> 웹에서 기존데이타 접근

3. 객체 포장 (Object Wrapping)

가. 클라이언트 / 서버 컴퓨팅 관점에서의 인터넷 기반 시스템 인터페이스

인터넷 기반 시스템을 점진적으로 개발해 가면서, 기존시스템과 대단히 많은 기능 중복 또는 이들 간의 자료나 서비스에서 불일치 (Inconsistency)가 발생할 가능성이 있다. 비슷한 종류의 정보를 조작하는 시스템이지만 상호간의 운용성은 거의 없거나, 있더라도 특정 벤더가 제공하는 일대일 방식의 접근 방식이어서 언어 독립적이지 않고 반드시 선행하는 실제 구현 어플리케이션이 있어야 하는 제약이 있다. 이와 같은 인터넷 기반 시스템은 자체 DB 또는 CGI 중심의 기존 시스템의 이용이 사용자 수준에서 준 상호작용 (Semi-interactive) 이라는 점에 주의해야 한다.

Java 언어로 구현된 Applet 은 사용자 수준에서 상호작용이 이벤트 구동 방식 (Event-driven processing)으로 이루어 지지만 서버의 서비스를 받는 것은 별도의 socket 이나 RMI (Remote Method Invocation) 등을 이용해야 한다. 이 때 서버측의 서비스를 제공하는 객체에 대한 관리나 통제와 같은 것이 없기 때문에 다른 기능을 하는 어떤 applet 에서 동일한 서비스를 원할 때 상호 인터페이스가 제공되기 어렵고 동일한 서비스에 대해 수 개의 서비스 객체가 존재할 때 상호운영성 (interoperability), 동시성 (concurrency), 일치성 (consistency) 을 해결하기 어렵다.

CORBA [OMG], [Orfali, et al.] 를 기반으로 할 때 비로소 분산 컴퓨팅에서의 상호운용성 (interoperability)이 보장되고 객체 서비스의 concurrency, consistency 와 같은 서비스 특성을 만족시킨다. 이 CORBA 기반 시스템은 서버측 스켈레톤(skeleton)과 클라이언트측 스텁(stub)이 각각 서비스 요청과 처리에 대한 표준 인터페이스를 제공하므로써 진정한 의미의 인터넷상의 클라이언트 / 서버 컴퓨팅[Dewire] 이 이루어지고 사용자 수준에서의 완전한 상호작용 (fully interactive)이 가능하다.

나. CORBA 기반 객체 포장 (CORBA-based Object Wrapping)

객체포장은 기존시스템에 대한 깊은 이해없이 접근을 숨기는 (Information Hiding) 방법이다. 이 기존시스템의 내용은 데이타와 기능을 포함하며 잘 정의된 인터페이스를 제공하므로써 기존시스템을 이용하려는 어떠한 클라이언트 또는 타 시스템이 이를 이용할 수 있도록 한다. 한번 객체포장이 된 해당 기존시스템은 해당 서비스를 이용하는 클라이언트 또는 타시스템을 위해 별도로 인터페이스를 개발할 필요가 없다. 객체포장은 복잡한 기존 시스템에 대한 충분한 이해 없이도 통합이 가능하고 이를 위한 코딩노력 감소 및 구현기간의 단축, 서비스 기능 중복 및 불일치 배제 등 많은 이익을 얻을 수 있다.[Brodie],[Chan,. et al.]

시스템 아키텍쳐가 인터넷 기반이던 전형적인 클라이언트 / 서버 아키텍

쳐와 관계없이 (시스템 아키텍쳐 독립적) 기존 시스템의 데이타나 어플리케이션을 재사용하기 위한 객체들의 집합인 프락시 서버 (Proxy Server)를 구성할 수 있다. 이 프락시 서버의 객체를 통해 인터넷 상에 서비스를 제공함으로써 기존시스템을 이용하는 클라이언트 및 타시스템의 Middleware 로서 기능한다.

인터넷에 기존 시스템의 어플리케이션과 데이타를 통합하는 객체 포장은 인터페이스를 분리하여 정의하고 실제구현은 기존시스템의 RPC (Remote Procedure Call) 또는 API (Application Program Interface)를 이용하여 기존시스템과 매핑한다. 이 때 분산객체에서 사실상의 표준인 CORBA IDL 을 이용한 객체 포장을 수행하면

① 기존시스템이 정하는 플랫폼과 독립적인 클라이언트

② 클라이언트와 서버간의 격리 (isolation)와 투명성 (transparency)

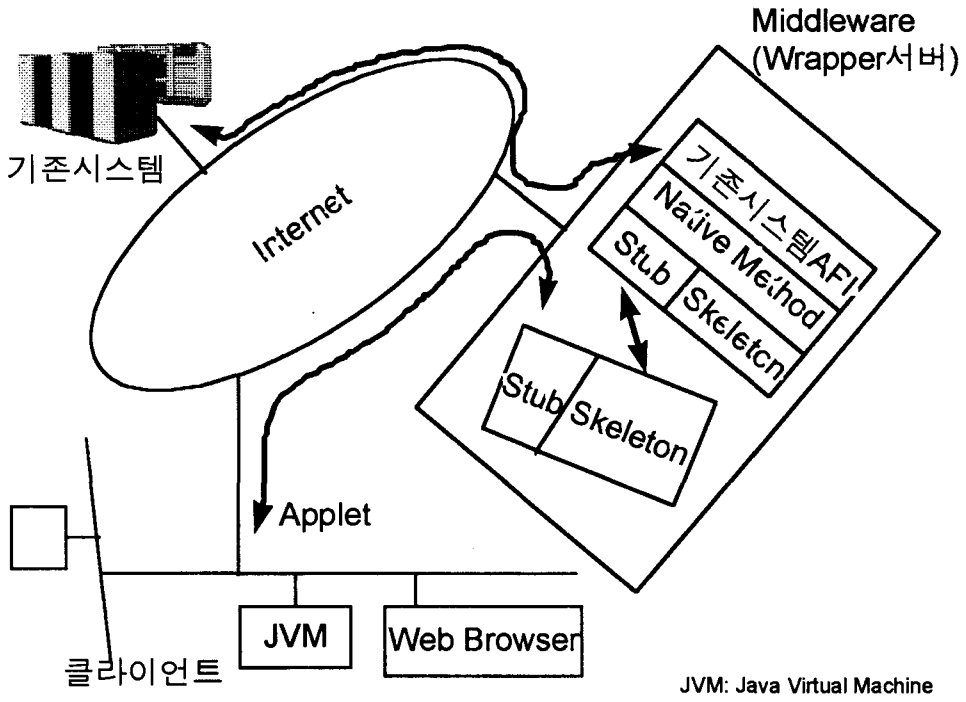
을 제공하는 인터페이스를 개발할 수 있다.

CORBA IDL 을 이용한 객체 포장을 하므로써 상호운용성이 지원되는 시스템 구현이 가능하고, 하나의 IDL 기술로서 서버 어플리케이션과 이를 요청하는 클라이언트 어플리케이션의 인터페이스를 정의하고 서버측의 상세한 실제 구현 프로그램을 요구하지 않는다. [Mowbray].

<그림 4 >는 인터넷상에 기존시스템 과 서버 그리고 클라이언트의 3 계층 클라이언트/서버 시스템 아키텍쳐이다. 서버에는 CORBA 기반의 객체포장을 통한 기존시스템 및 클라이언트간의 인터페이스를 제공하는 객체들이 있다.

기존시스템과 인터페이스하는 객체가 생성되면 이 기능을 요청하는 객체의 수나 종류에 관계없이 해당 IDL 에서 정의한 대로 요청할 수 있다. 객체 포장을 통해 기존시스템의 기능자체의 변화는 없다.

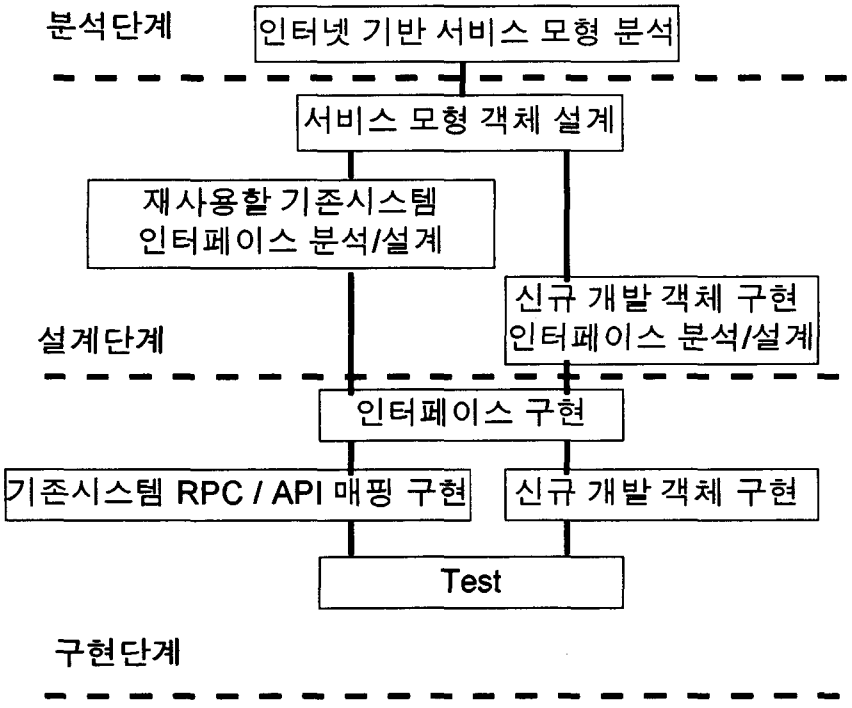
사용자는 웹 브라우저를 통해 서버로부터 해당 서비스의 클라이언트측 사용자 인터페이스 객체를 다운로드 받아서 사용 (Web Browser 와 Java Applet) 할 수도 있고, 사용자의 클라이언트 플랫폼에 이식되어 구동하는 사용자 인터페이스 프로그램 (Java Program, C++ Program, 등)을 이용할 수도 있다. 어느 경우에 있어서든 사용자 인터페이스 객체는 동일한 객체포장된 객체 (Wrapper)와 인터페이스한다.



<그림 4> 인터넷상에 객체포장을 이용한 3 계층 C/S 시스템 아키텍쳐

다. CORBA 기반 객체 포장을 이용한 인터넷 기반 시스템 개발 접근방법

인터넷상에 기존시스템의 서비스 제공을 위한 인터페이스에 대한 분석과 설계가 있어야 성공적인 기존시스템의 객체 포장이 가능하다. 이상적인 프로젝트의 개발단계로서 <그림 5 >과 같이 인터넷 기반 서비스 모형분석, 서비스 모형객체 설계 [Booch], [RSC] 및 기존시스템 분석 / 설계와 객체 구현 및 기존시스템의 인터페이스 구현의 과정을 수행한다.



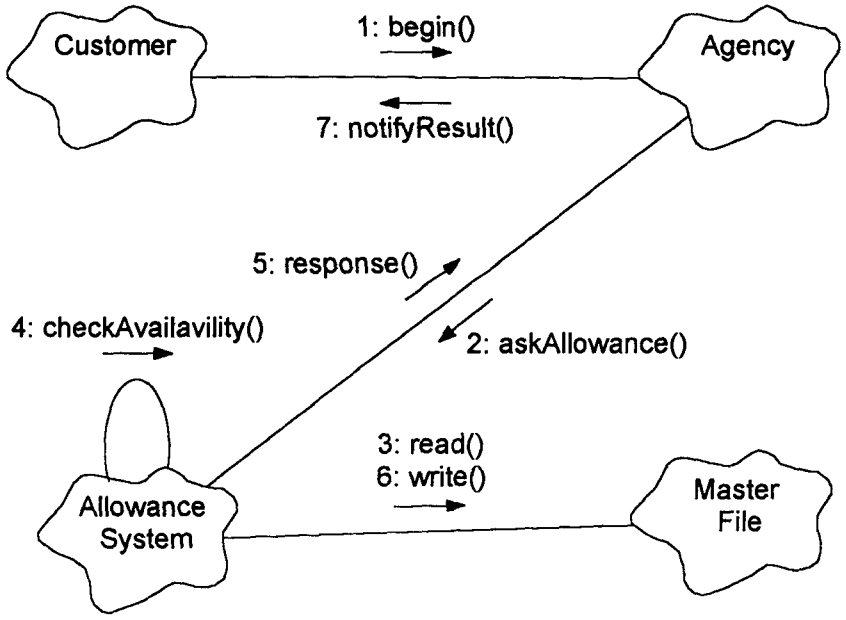
<그림 5 > 기존 시스템을 재사용하여 인터넷 기반 시스템에 통합하는 객체 포장 개발 접근방법

4. 사례연구

기존시스템의 객체포장에 대한 개발 접근방식에 대한 사례로서 S 카드사 거래승인 업무의 인터넷으로 통합을 선정하였다.

S 카드사는 현재의 호스트중심 (Mainframe System)에서 클라이언트/서버 시스템 아키텍쳐로 이주하고자 하고, 현재는 보안 (security) 상의 이유로 인터넷(Internet) 으로 통합은 고려하고 있지 않지만 향후 인터넷상의 상거래 (EC; Electronic Commerce) 가 급증하면 기존 시스템에서 확장하여 별도의 컨텐트 (Content)를 개발해야 할 것으로 보고 있다.

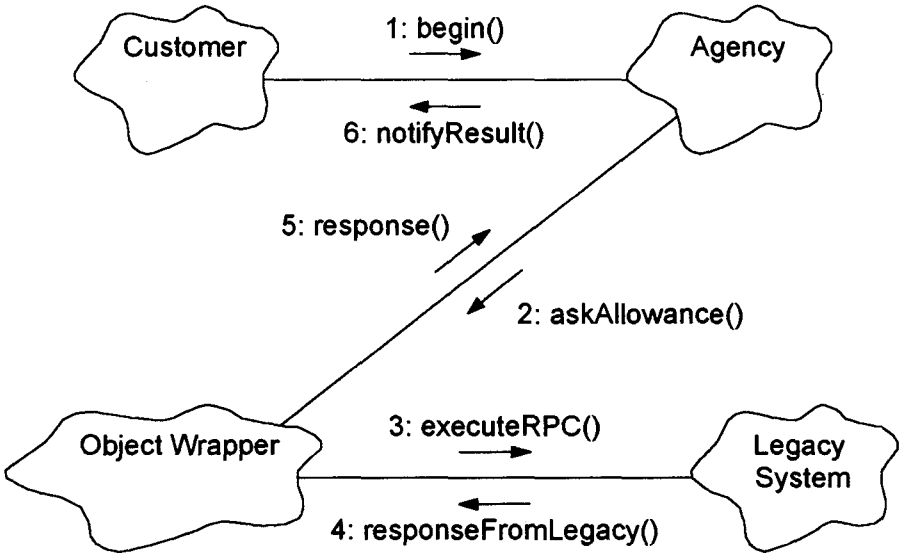
전용 단말기로만 가능하던 것을 유.무선 네트웍으로 연결된 자바 가상 머신 (Java Vertual Machine)이기만 하면 승인 시스템을 이용 영업점 또는 가맹점 영업을 할 수 있도록 기존 시스템의 DataBase 와 기능을 객체 포장을 통해 인터넷에서 이용하는 시스템을 구축한다. 거래승인 업무에 대한 시나리오를 <그림 6 >와 같이 나타낼 수 있다.



<그림 6> 거래승인 업무 시나리오 (Object Message Diagram)

기존 시스템의 승인 원장 (Master File)을 처리하는 승인 시스템을 그대로 공존케 하면서 영업점 (Agency)의 처리를 객체 시스템으로 하도록 한다. 영업점 객체는 기존시스템을 포장한 승인 대리객체(Proxy Object)인 Object Wrapper 와 인터페이스하고 대리 승인 객체는 기존 승인시스템의 기능을 부르는 RPC 를 통해 기능한다. 영업점 객체 입장에서는 이 결과가 기존시스템으

로 부터 오는 것인지 상호작용하는 객체로 부터 오는 것인지 알 필요가 없다. <그림 7>은 이와 같은 내용을 토대로 재구성한 것이다.



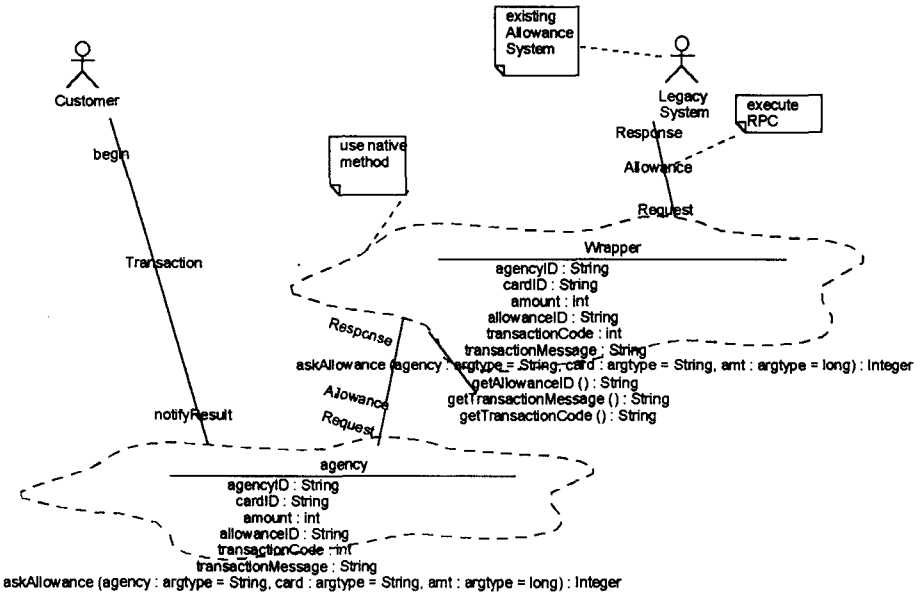
<그림 7> 객체포장으로 재구성한 승인업무 시나리오 (Object Message Diagram)

설계단계에서 시스템 아키텍처와 관련한 다음과 같은 사항이 정의되어야 한다.

○ 클래스 : 속성 (attribute) 및 메쏘드 (method)

○ 클래스간 관계

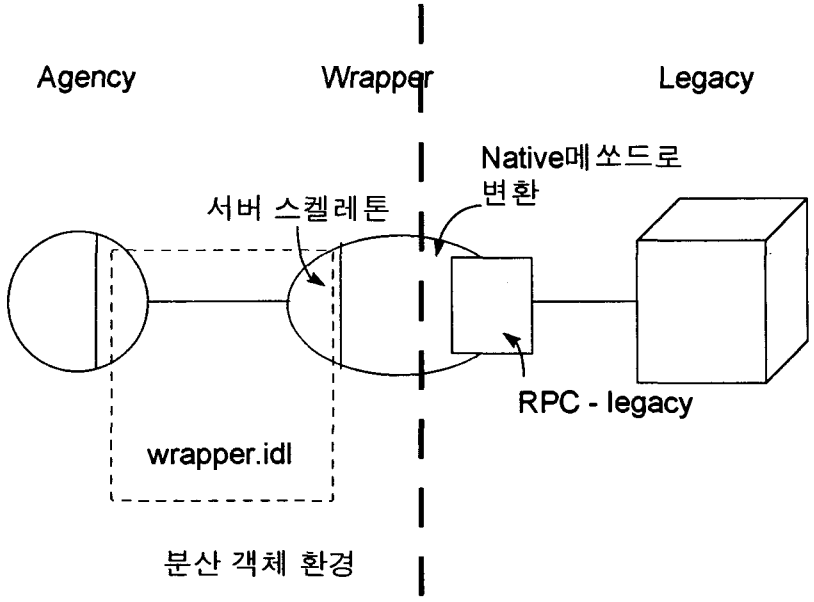
이를 나타내면 <그림 8>와 같다.



<그림 8 > Class Diagram

서버 어플리케이션 및 클라이언트 어플리케이션 인터페이스를 위한 IDL 을 작성하고, 컴파일된 결과로 얻은 구현해야 할 실제 객체를 완성하고 Test 를 하도록 한다.

설계를 통해 얻어진 객체시스템을 구현하기 위한 CORBA IDL 을 이용한 프로그램을 작성할 때, 각 구성요소를 <그림 9>와 같이 나타내어 볼 수 있다.



<그림 9> 구현할 객체포장 구성요소 정의

사례에서 구현되어야 할 객체포장을 위해 기존시스템의 핵심 프로세스를 가지는 프로그램이 설정되고 이를 이용할 수 있는 RPC를 구현해야 한다. 구현된 RPC는 Native 메쏘드 (method)로 이용하기 위해 별도의 클래스를 만들어 놓는다. 객체포장하는 wrapper는 이 별도의 클래스의 메쏘드를 이용하므로써 기존시스템과 인터페이스할 수 있다. 다음은 이러한 프로그래밍의 단계와 코드를 요약한 것이다.

① IDL 정의 (wrapper.idl)

```
Interface Wrapper {
// attribute
. . .
readonly attribute string allowanceID;
. . .
// operation
long askAllowance (in string agency, in string card, in long amt);
};
```

② IDL 컴파일 (_boaimpl_wrapper.java 등과 같은 클래스 생성)

③ 서버 스켈레톤을 구현하는 RPC 프로그램을 작성

```
        if( legacy_bind( "localhost", 0,0,0) ==
                printf("Fail connect to server on host ₩n");
                exit(1);
        }
        legacy_has_this_procedure(makeCString(ag),makeCString
(cd),amt);
        ...
```

④ 단계③에서 작성된 프로그램을 네이티브 메쏘드 (native method)로 변환하는 클래스 (legacyPac.legacyRPC.java) 작성

```
        ...
        public native String execute_legacy(String ag, String cd,
int amt);
        static {
                System.loadLibrary("legacyAPI");
        }
        ...
```

⑤ 단계②에서 생성된 스켈레톤을 구현하는 클래스 (wrapper Server.java) 작성. 이 때 단계④의 클래스를 포함 (import)하고 이 클래스의 메쏘드를 이용한다.

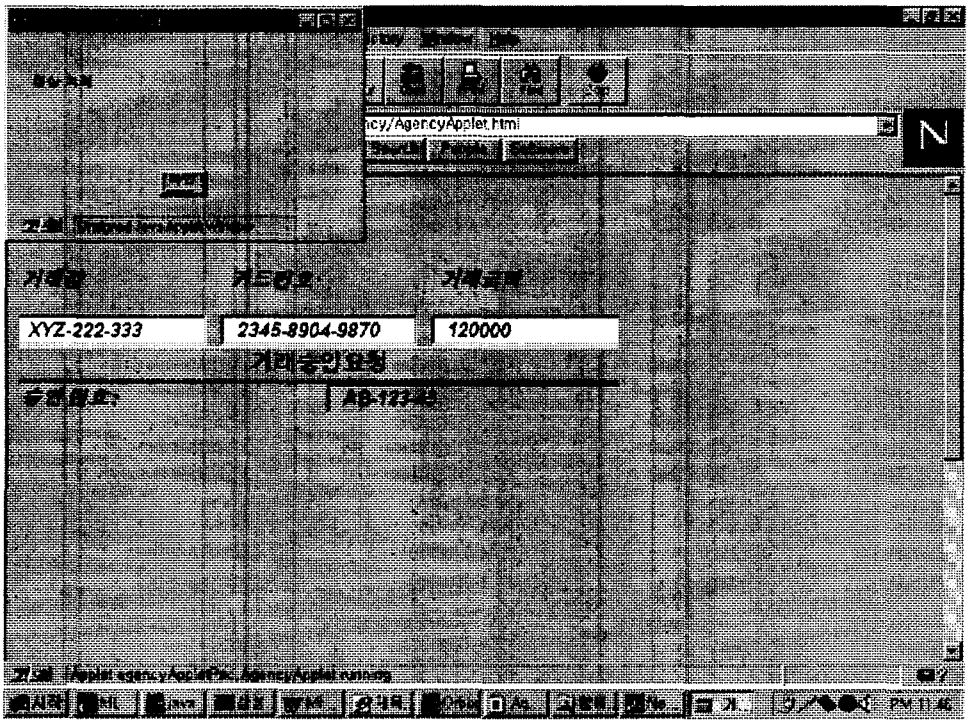
```
import legacyPac.*;
...
try
    _CORBA.Orbix.impl_is_ready("WrapperSrv");
      }
...
class WrapperImplementation extends _boaimpl_wrapper {
...
    legacy.execute_legacy (t_agencyID, t_cardID, t_amount);
...
}
```

⑥ 단계⑤에서 작성된 클래스를 컴파일

⑦ 컴파일된 클래스를 인터페이스 저장소에 등록

기존시스템은 이제 작성된 객체를 통해서 접근할 수 있으며, 이를 이용하는 객체는 이것이 어떻게 구현되어 있는가를 고려할 필요없이 다른 분산된 객체처럼 이용할 수 있다.

이렇게 구현된 객체를 이용하는 HTML 페이지와 사용자 인터페이스를 갖는 객체 (applet)를 만들어 사례를 완성한다. "거래처" (agency) 객체는 구현된 wrapper 를 이용한다. 이때 이 객체의 구현도 분산객체시스템환경에 통합되도록 IDL 을 작성하고 그 기능을 구현할 수 있다. <그림 10 >은 기존시스템을 이용하는 거래승인 업무를 인터넷 브라우저 Netscape 을 통해 접근하여 실행한 예로서, 카드 번호, 거래액을 입력하면 객체 포장된 프락시 객체가 이 거래 승인 관련 일대일 매핑된 기존시스템의 RPC 를 통해 그 결과로서 승인번호를 받아 온다. 이로써 거래 승인 업무가 종료된다.



<그림 10> 객체포장 객체를 통한 승인 처리 화면

5. 결론 및 향후 연구 과제

객체포장은 기존시스템에 대한 깊은 이해 없이도 인터넷상에 기존시스템의 서비스를 대상으로 해서 개발위험은 최소로 하면서 서비스의 중복이나 불일치 (Inconsistency) 를 회피하며 구현할 수 있는 대안이다. 한 번 객체포장이 된 기존시스템은 그 서비스를 필요로 하는 어떠한 종류의 클라이언트나 숫자에 관계없이 클라이언트에 대해 표준의 인터페이스를 제공하는 미들웨어로서 인터페이스 외부의 환경에 다양한 융통성을 가지게 된다.

객체 포장의 한계로는 기본적으로 기존 시스템에 고착된 기능상의 한계를 그대로 상속한다는 데 있으며 기존시스템이 제공하는 서비스가 장기적으로 고정되게 필요하다는 가정을 하고 있다. 기존 시스템의 기능을 '어떻게 (How) 매핑할 것인가?' 보다 '무엇(What service)에 매핑할 것인가?'를 더욱 고려해야 하는 것이 필요하다. '무엇에 매핑할 것인가?' 라는 범위를 결정한 후에도 API 코드를 한 개 서비스에 한 개씩 만드는 것 자체가 어려운 작업이다.

현재로서 사용자가 느끼는 어느 정도의 처리 속도 저하는 불가피하지만 인프라 스트럭쳐 (Infra Structure) 성능 향상에 따라 그 차이가 유의하지 않은 수준으로 갈 것이다.

연구의 한계로는 직접 S 카드사의 시스템을 인터페이스 시키지는 못하고 그 서비스 기능 일부를 실험실에서 모의하였다.

향후에는 기존시스템이 갖는 기능의 객체와의 매핑 (mapping) 과 기존시스템의 API 를 Native Method 로 변환하는 것의 자동화가 보다 연구되어야 할 것이다.

참고문헌

Alder, R., "Distributed Coordination Models for Client/Server Computing," IEEE Computer Magazine, April 1995

Amjad Umar "Object-Oriented Client/Server Internet Environments" Prentice Hall 1997

Berson, A., "Client/Server Architectures", McGraw-Hill, 1933

Dawna Travis Dewire, Client/Server Computing, McGraw-Hill, 1993

Grady Booch,, Object-oriented Analysis and Design with Applications, 2nd Ed., Addison-Wesley, 1994

Kador, J.,"The Ultimate Middleware," Byte Magazine, April 1996

Lewis, T., "Where Is Client/Server Software Headed" IEEE Computer Magazine, April 1995

Michael L. Brodie and Michael Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information Systems", DOM Technical Report, TR-0222-10-92-165, GTE Laboratories, 1993, ftp://ftp.gte.com/pub/dom/reports/BROD93b.ps

OMG, The Common Object Request Broker: Architecture and Specification, Revision 2.0, 1995, ftp://ftp.omg.org/pub/docs/

Perter Fingar , "Next Generation Computing: Distributed Objects for Business" GIGS Reference Library, 1997

Rational Software Corporation, Succeeding with the Booch and OMT Methods: a Practical Approach, Addison-Wesley, 1996

Robert Orfali, Dan Harkey, and Jeri Edwards, The Essential Distributed Object Survival Guide, John Wiley & Sons, 1996

Taizan Chan, Ciu Leung Chung, and Teck Hua Ho, "An Economic Model to Estimate Software Rewriting and Replacement Times", IEEE Transactions on Software Engineering, Vol. 22, No. 8, 1996

Thomas J. Mowbray, Raphael C. Malveau, CORBA Design Patterns, John Wiely & Sons, 1997