□ 研 究 論 文 □

韓國軍事運營分析學會誌

第 23 卷, 第 2 號, 1997. 12 . 31

분산 객체를 이용한 웹기반 클라이언트 /서버 구조의 구현 (An Implementation of Web-based Client/Server Architecture using Distributed Objects)

박희창*, 이태공**

Abstract

Internet users have been rapidly increased due to the convenient GUI environment. Current Web-based HTTP/CGI client/server architecture has several problems such as the CGI bottleneck, no maintaince of state, and no load balancing. However, with Java and CORBA technologies called "Object Web technology", we can solve them because Java is not only a mobile code but also a platform-independent code, and CORBA has ability to build distributed object and language-independent object model. The goal of "Object Web technology" is to create multivendor, multiOS, multilanguage "legoware" using objects

This paper implements "Book Search System" which is Web-based client/server architecture using distributed objects. Environments of this implementation are Hangul Windows NT(included IIS) server, Hangul Windows 95 client, Visigenic's VisiBroker for Java 1.2 which is a product of CORBA 2.0, HTTP protocol on TCP/IP-based, Sybase SQL Anywhere 5.0 database server, and the interface between application server and database is JDBC-ODBC bridge middleware.

* 국방대학원 전자계산학과 석사과정

** 국방대학원 전자계산학과 조교수

1. 서 론

논문은 네트워크 환경에서 탁월한 성능을 발휘하는 Java 언어에 대하여 특징, 보안체계, 모빌 코드 시스템, JDBC(Java Database Connectivity)의 순으로 살펴보고, 네트워크 어느 곳에서도 원하는 지식(Knowledge)을 이용할 수 있는 CORBA의 분산 객체에 대하여 살펴 볼 것이다. 또한 Java와 CORBA 간의 상호 보완적인 효과를 제공하는 기능에 대하여 알아 보고, Java와 CORBA의 두 기술들을 결합한 분산 객체를 이용한 웹기반의 클라이언트/서버 구조를 구현할 것이다. 구현하고자 하는 시스템의 환경은 한글 윈도우즈 NT(IIS 포함)를 서버로 하고, 한글 윈도우즈 95를 클라이언트로 하여, TCP/IP를 통한 네트워크 환경에서 HTTP 프로토콜과 CORBA 2.0의 제품 중에서 Visigenic사의 VisiBroker for Java 1.2를 사용하여 분산 객체를 이용한 웹기반 클라이언트/서버 구조의 "도서 검색 시스템"을 구현하였다.

본 논문은 총 4장으로 구성되는데, 제1장 서론, 제2장 Java와 CORBA, 제3장 분산 객체를 이용한 웹기반 클라이언트/서버 구조의 구현, 제4장 결론의 순이다.

2. Java와 CORBA

2.1 Java

2.1.1 Java의 특징

Java 언어는 단순함, 객체지향, 분산, 견고성, 보안, 시스템 독립성, 이식성, 다중 스레드, 동적 특성, 예외처리 능력, 쓰레기 수집 등의 여러 특징을 가지는데, 이것들에 대해서 간략히 살펴보면, 다음과 같다[2][10].

Java의 문법은 C++에 대한 문법의 정리된 버전이다. 헤더 파일, 포인터 연산(포인터 문법 조차도), 구조체, 유니온, 연산자 오버로딩, 가상 기반 클래스 등이 필요없다. 즉, Java는 경험상 이점보다는 해악을 유발하는 C++의 특징을 제거하고, 드물게 사용되고 이해하기 어려운 것들을 생략하였다. 또한, Java의 목표중 하나는 소형 기계에서 단독으로 실행될 수 있는 소프트웨어를 만드는데 있다.

HTTP(Hypertext Transfer Protocol)와 FTP(File Transfer Protocol)같은 TCP/IP(Transmission Control Protocol/Internet Protocol) 프로토콜을 쉽게 사용하는 확장 라이브러리 루틴이 있다. Java 어플리케이션은 프로그래머가 지역의 파일 시스템을 접근할 때와 같이 URL(Uniform Resource Locator)을 통해서 네트워크를 쉽게 접근할 수 있다. Java의 네트워킹 능력은 강력하며 사용하기 쉽다. 다른 언어로 인터넷 프로그래밍을 하는 것중에 소켓 연결과 같은 것을 Java가 얼마나 쉽게 하는지 알 수 있다. 이는 즉각적으로 여러 컴퓨터에 접속하는 인터넷을 사용할 수 있게 해준다. 네트워크용 게임, 분산 환경의 어플리케이션 프로그램, 또는 넷 서핑이 어느 때보다 쉽다.

Java는 다양한 방법으로 신뢰적인 프로그램을 작성할 수 있다. Java는 가능한 문제에 대한 점검을 하며, 실행시간에 점검, 에러를 유발하기 쉬운 상황의 제거 등을 강조한다. Java는 메모리 덮어쓰기의 가능성과 데이터 오염을 제거하는 포인터 모델을 갖

는다.

Java는 네트워크와 분산 환경에 사용하기 용이하다. Java는 바이러스와 해킹에 안전하다. 이것은 Java의 보안 메커니즘을 속이기가 지극히 어렵다는 것이다.

Java는 시스템 독립성을 보장한다. 더 이상 코드를 하나의 기계에서 다른 기계로 이식할 필요가 없다. 매킨토시로 작성된 프로그램이 인텔 기계나 그 외의 어떤 기계에서도 실행이 가능하다. 즉, Java 컴파일러는 특정 컴퓨터 구조에 관계없는 바이트코드를 생성한다. 바이트코드는 어떠한 기계에서도 실행할 수 있다.

Java는 이식성이 강하다. C/C++와는 달리 Java는 구현-독립적이다. 즉 Java에서 "int"는 항상 32-bit 정수이다. C/C++에서는 16-bit 정수, 32-bit 정수, 또는 컴파일러 업체에 따라서 다른 크기를 의미한다.

Java는 다중 스레딩을 지원한다. 다중 스레딩은 한 프로그램이 한 번에 한 가지 이상의 일을 할 수 있는 능력을 말한다. 다중 스레딩의 잇점은 상호작용적인 반응과 실시간 연산이다.

Java는 동적(Dynamic)이다. 여러 가지 면에서 Java는 C/C++보다 동적인 언어이다. 진화적 환경에 적응하도록 설계되었다. 라이브러리는 새로운 메소드를 추가하고, 클라이언트의 노력없이도 변수를 인스턴스화할 수 있다.

Java는 예외처리 능력을 제공한다. 프로그램을 예기치 못했던 사건으로부터 보호할 수 있고, 진행 도중에 문제를 해결하거나 세련되게 종료 할 수 있도록 설계할 수 있다.

Java는 쓰레기 수집(Garbage Collection)을 제공한다. 상당수의 프로그래머들은 불안정한 메모리 포인터나 메모리 유출에 꽤나 익숙해져 있는데, Java는 이런 재난이 애초에 발생하지 않도록 방지한다.

2.1.2 Java의 보안 체계

Java가 오늘날 존재하는 이유의 대부분은 보안 요구 때문이다. 원래의 생각은 이 환경에 C++를 사용하는 것이었지만, Java 설계자들은 C++에 지나치게 많은 보안상의 허점이 있음을 알게 되었다. Java는 이런 허점을 제거하고 이런 위험을 막기 위해 4단계 보안 방어 체계를 제공한다[1][10].

먼저, Java 컴파일러 단계이다. 일반적으로 제어되지 않는 시스템 접근을 허용하는 어떤 행동도 언어 명세에서 배제되었다. 무엇보다 주목할 만한 것은 Java가 프로그래머에게 실제 메모리 포인터를 손댈 수 없도록 한다. 컴파일러는 언어명세에 제한을 한다. Java는 원래대로 사용되는지 검사한다.

둘째, 바이트코드 검증 단계이다. 대부분의 어셈블리 언어와는 달리 Java 바이트코드는 상당한 양의 타입 정보를 전송한다. 이것이 검증기가 다른 경우에 행하지 않았던 많은 것들을 검사할 수 있도록 해준다.

셋째, 클래스 적재 단계이다. 클래스 적재기는 내장 클래스들과 네트워크를 거쳐서 온 클래스들 사이의 구분을 명확하게 한다. 일반적으로 이것은 내부적으로 실행시간이 코드 조각들을 출신지에 근거하여 분류한다는 의미이다.

보안의 마지막 단계는 시스템 자원 보호로부터 온다. 이 단순한 규칙은 Java 프로그램은 시스템을 변경(예, 하드 디스크를 지우는 것)할 수 없다는 것이다. 외부로부터 들어온 코드가 제어 중인 시스템 자원(디스크)에 접근하려고 어떠한 시도로부터 보호

된다.

2.1.3 Java의 모빌 코드 시스템

모빌 코드 시스템은 모빌 에이전트를 위한 기초 기술을 제공하는데, 이것은 클라이언트와 서버를 교차하는 분산 코드 및 데이터를 제공한다. 모빌 객체는 Java에서 애플릿이라고 하는데 전통적인 소프트웨어와는 달리 웹 브라우저, 서버들과 같은 독립적인 프로그램에 의해 동적으로 적재되고 실행된다. 모빌 코드 시스템에서는 모빌 코드를 실행하기 위한 안전한 환경을 제공해야 하며, 플랫폼에 독립적인 서비스를 해야하고, 애플릿의 적재, 해제 및 코드 실행을 위한 실행환경을 제공해야 하며, 네트워크간에 애플릿이 이동할 수 있는 기능을 제공해야 한다[6].

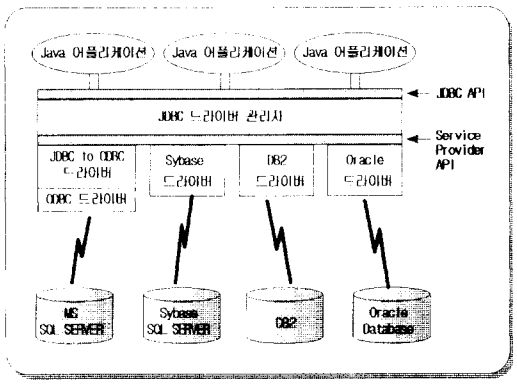
Java는 전체적으로 새로운 웹을 위한 클라이언트/서버 상호작용 모델을 도입했다. 이것은 Java와 호환되는 브라우저에서 다운로드 할 수 있는 애플릿 프로그램을 만들 수 있게 해 준다.

2.1.4 JDBC(Java Database Connectivity)

JDBC API는 SQL 추상화와 개념에 기초한 순수한 Java 인터페이스이다. JDBC API는 처음부터 출발한 것이 아니고 ODBC에 바탕을 두기 때문에, ODBC에 친숙한 프로그래머는 JDBC를 배우기 쉽다. JDBC는 ODBC의 기본적인 설계 원칙을 가진다. 사실 두 인터페이스는 X/Open SQL CLI(Call Level Interface)에 근거한다. 큰 차이점은 JDBC는 Java의 스타일과 장점을 강화하도록 만들어진다는 것이다[6].

썬 마이크로시스템즈는 다음과 같은 목표로 JDBC를 설계하였다: i) Java 프로그램을 위한 데이터베이스 독립적인 데이터 접근이 가능하고, ii) ODBC와 X/Open SQL CLI와의 호환성을 유지하고, iii) Java 언어에 충실을 기한다[12].

JDBC는 <그림 2-1>에서 보는 바와 같이 두 가지의 주요한 인터페이스를 제공한다: i)어플리케이션 인터페이스는 DBMS 독립적으로 SQL 서비스를 접근할 수 있게 하고, ii) 드라이버 인터페이스는 DBMS 업체가 그들의 특정 데이터베이스로 적합하도록 해야 하는데, JDBC는 주어진 데이터베이스와 통신하도록 적절한 JDBC 드라이버가 자동적으로 적재되도록 드라이버 관리자를 사용한다[6].



<그림 2-1> JDBC의 구조

2.2 CORBA

2.2.1 분산 객체

CORBA(Common Object Request Broker Architecture) 객체는 네트워크 어느 곳에서든지 존재할 수 있는 지식(Knowledge)이라고 할 수 있다. 서버 객체를 생성하는데 사용된 언어와 컴파일러는 클라이언트에 대하여 전체적으로 투명하며, 클라이언트는 분산 객체가 존재하는 곳이 어디인지 또는 어떤

운영 체제에서 실행되는지를 알 필요가 없다. 이것은 같은 프로세스나 거대한 네트워크에 존재하는 기계에서 모두 가능하다. 또한 클라이언트는 서버 객체가 어떻게 구현되어 있는지를 알 필요가 없다. 단지 서버의 인터페이스만 알면 된다. 이것은 IDL (Interface Definition Language)로 작성되며, 클라이언트와 서버 사이에 연결을 제공한다[6].

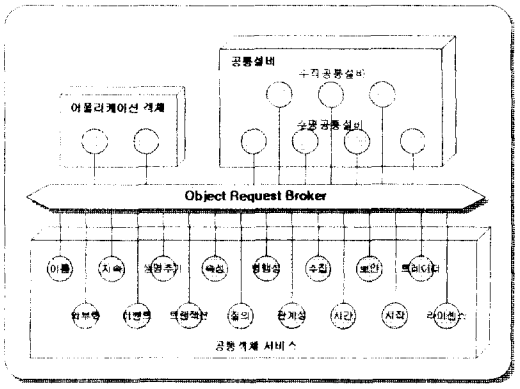
CORBA IDL은 선언적인 것으로 상세히 구현되지 않는다. IDL은 운영체제와 프로그래밍 언어에 모든 서비스와 CORBA 버스에 존재하는 컴포넌트에 대하여 독립적인 인터페이스를 제공하며, 상호운용하기 위하여 다른 언어로 쓰여진 클라이언트와 서버 객체들을 허용한다[6].

CORBA의 목표는 모든 클라이언트/서버 미들웨어와 ORB(Object Request Broker) 위에 존재하는 모든 컴포넌트를 "IDL화"하는 것이다. OMG(Object Management Group)는 모든 객체들을 IDL로 정의하여 네트워크에 존재하는 컴포넌트들을 사용할 수 있도록 하는 것이다. 클라이언트/서버 컴퓨팅 환경에 객체 물결이 오고 있는데, 목표는 객체를 사용하여 다중 업체, 다중 운영체제, 다중 언어, "레고웨어"를 만드는 것이다[6].

2.2.2 OMG의 OMA

OMA(Object Management Architecture)는 어플리케이션 프로그램들 간의 결합뿐만 아니라 객체의 생성, 소멸에서부터 저장, 트랜잭션 기능에 이르기까지 분산 객체 환경에서 필요한 모든 서비스를 총칭하는 것으로 4개의 주요 요소로 구성되어 있다[6].

i) ORB(Object Request Broker): CORBA를 정의한 것으로 서로 다른 분산 객체들 사이의 버스 역



<그림 2-2> OMG OMA

할을 하는 OMA의 가장 중요한 요소로 요청하는 객체의 위치에 상관없이 필요한 객체를 사용할 수 있도록 해준다.

ii) CORBAservices(CORBA 서비스): 분산 객체 버스를 확장한 여러 가지 시스템 수준의 분산 객체 서비스를 정의한다. 여기에는 생명주기 서비스, 이름 서비스, 이벤트 서비스, 병행 통제 서비스, 트랜잭션 서비스, 관계성 서비스, 외부화 서비스, 질의 서비스, 트레이딩 서비스 등이 포함된다.

iii) CORBAfacilities(CORBA 설비): 어플리케이션 프로그램 객체를 지원하기 위한 설비로써 업무 객체에 의해 직접 사용되는 수직 또는 수평적 어플리케이션 프레임워크를 정의한다. 수평적 설비는 사용자 인터페이스, 정보 관리, 시스템 관리, 태스크 관리 등을 지원하며, 수직적 설비는 의료, 소매, 금융 등 구체화된 상호작용 객체의 집합을 말한다.

iv) 어플리케이션 객체: 업무 객체 및 어플리케이션으로서 CORBA 기반구조를 구성하는 4가지 요소의 최상위 수준의 관점을 제공한다. 업무 객체는 고객, 주문, 지불 등과 같은 어플리케이션 독립적인 개념을 정의하기 위한 자연스러운 방법을 제공한다.

2.2.3 CORBA 2.0

OMG 문서 CORBA 명세서는 OMA 참조 구조(Reference Architecture)의 ORB 컴포넌트의 기능성을 정의한다. 이것은 CORBA 클라이언트에서 CORBA 객체 구현(Object Imple -mentation)으로 객체 연산 호출에 대한 요청을 전송하는 메시지 버스이다[6].

가. 클라이언트, 서버, 객체 구현

CORBA는 분산 컴퓨팅에 대한 피어-투-피어(Peer-to-Peer) 모델을 제공한다. 전통적인 클라이언트/서버 어플리케이션에서 GUI와 업무 로직을 구현하는 클라이언트를 볼 수 있다. CORBA에서 클라이언트와 서버의 용어는 분산 어플리케이션에서 컴포넌트에 의해 행해지는 역할에 의해 설명된다[13].

클라이언트: 분산 어플리케이션에서 다른 컴포넌트에게 요청을 하는 역할을 한다. 순수한 클라이언트는 서버 역할을 하지 않는 프로그램이다.

서버: 클라이언트가 사용하는 컴포넌트의 구현을 제공하는 역할을 한다. 서버도 종종 다른 컴포넌트에게 클라이언트 같이 행한다.

CORBA에서 기능성을 제공하는 컴포넌트는 CORBA 객체이다. 기능성을 구현하는 객체는 인터페이스 정의에 의해 기술된다. CORBA는 그 구현으로부터 객체의 인터페이스는 명백히 분리된다. CORBA의 클라이언트 객체는 인터페이스에만 의존한다. 그래서 구현은 상호변환되게 사용될 수 있다.

CORBA 서버는 하나 또는 그 이상의 CORBA 객체의 구현을 제공하는 프로그램이다[13].

나. 투명성

ORB는 객체의 위치 투명성을 제공한다. 즉, 객체는 클라이언트가 마치 지역의 객체에 있는 메소드인 것처럼 연산을 호출할 수 있도록 객체 참조로 표현된다. 객체 참조는 동일한 프로그램, 동일한 기계의 다른 프로세스, 혹은 원격 기계에 있는 객체를 참조할 수 있다. 각각의 객체는 모두 똑같은 방법으로 사용된다[13].

분산 어플리케이션의 통합을 돕는 또 다른 투명성은 프로그래밍 언어 투명성이다. CORBA 객체는 인터페이스 정의에 의해 구성되고, 객체 참조에 의해 참조되기 때문에, 인터페이스 후면의 구현은 여러 프로그래밍 언어중 하나일 수 있다[13].

이러한 투명성은 OMG의 IDL(Interface Definit -ion Language)에 의해 가능하다. 즉, 객체의 인터페이스는 OMG IDL에서 정의된다. 인터페이스 정의는 객체가 수행할 연산, 요구되는 입출력 파라미터, 생성될 수 있는 예외들을 명세한다[13].

다. ORB 구조

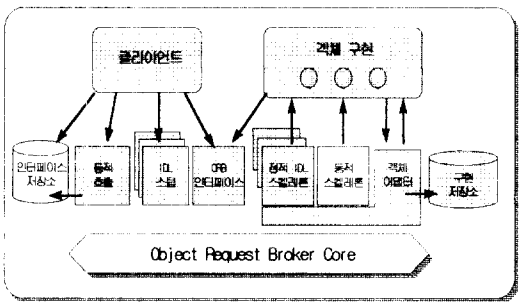
CORBA 2.0 ORB는 단일 프로세스 내에서의 객체 사이의 중개 역할뿐만 아니라, 여러 업체의 ORB 사이의 상호운용 기능을 추가한 것으로 그 구조는 <그림 2-3>과 같이 클라이언트와 서버의 두 측면에서 볼 수 있는데, 먼저 클라이언트 측면은 클라이언트 IDL 스텁(Stub), 동적 호출 인터페이스, 인터페이스 저장소, ORB 인터페이스로 구성되어 있으며, 서버 측면은 정적 스켈레톤(Skeleton), 동적 스켈레톤 인터페이스, 객체 어댑터, 구현 저장소, ORB 인터페이스로 구성되어 있다[6].

IDL 스텁은 객체 서비스로의 정적인 인터페이스를 제공한다. 이 프리컴파일된 스텁은 클라이언트가

서버에 있는 해당 서비스를 어떻게 호출하는지를 정의한다.

동적 호출 인터페이스(DII)는 실행시간에 호출되어지는 메소드를 찾도록 한다. CORBA는 서버 인터페이스를 정의하는 메타데이터의 검색, 파라미터 생성, 원격 호출의 요청, 결과의 접수 등에 대한 표준 API를 정의한다.

인터페이스 저장소는 등록된 모든 컴포넌트 인터페이스, 지원하는 메소드, 요구되는 파라미터들을 획득하고 수정하도록 한다. 인터페이스 저장소는 기계가 읽을 수 있는 IDL로 정의된 인터페이스 버전을 포함하는 실행시간 분산 데이터베이스이다.



<그림 2-3> CORBA 2.0 ORB의 구조

ORB 인터페이스는 어플리케이션과 연관된 지역 서비스로의 몇몇 API로 구성된다. 예를 들면, CORBA는 객체 참조를 문자열로 변환하거나 반대의 기능을 하는 API를 제공한다. 이것은 객체 참조들을 저장하거나 통신할 때 유용하게 호출할 수 있다. 정적호출 혹은 동적호출이 모두 가능한데, 정적호출은 프로그램이 쉽고, 빠르며, 자기-문서적(Self documenting)인 반면, 동적호출은 프로그램하기 어렵지만 실행시간 서비스에 최대의 융통성을 제공한다.

서버측에서는 정적호출 혹은 동적호출이 모두 동일한 메시지 구문(Syntactics)을 갖는다. 두 경우에, ORB는 서버 객체 어댑터를 가리키고, 파라미터를 전송하고, 서버 IDL 스켈레톤을 통하여 구현 객체로 통제 신호를 보낸다[6].

서버 IDL 스켈레톤은 서버에 의해 외부로 출력되는 서비스로의 정적인 인터페이스를 제공한다. 이것은 IDL 컴파일러에 의해 생성된다.

동적 스켈레톤은 서버에 대한 실행시간 바인딩 메커니즘을 제공한다. 동적 스켈레톤은 ORB간의 일반적인 브리지 구현에 매우 유용하다.

객체 어댑터는 ORB의 코어 통신 서비스의 위에 위치하며, 서버의 객체를 대신하여 서비스 요청을 승인한다. 객체 어댑터는 서버 객체의 인스턴스화, 인스턴스로의 요청 전달, 인스턴스의 객체 식별자(객체 참조) 할당에 대한 실행시간 환경을 제공한다. CORBA는 각 ORB가 지원해야만하는 표준 어댑터인 기본 객체 어댑터(BOA, Basic Object Adapter)를 명세한다. 서버는 한 객체 어댑터 이상을 지원한다.

구현 저장소는 서버가 지원하는 클래스에 관한 정보의 실행시간 저장소를 제공한다. 구현 저장소는 ORB의 구현과 연관된 추가적인 정보를 저장하는 공통 장소로도 지원된다.

ORB 인터페이스는 클라이언트측에서 제공하는것과 동일한 지역 서비스로의 몇몇 API로 구성된다.

CORBA ORB를 사용함으로써 메소드의 호출을 컴파일 시간에 정의하거나 실행 시간에 동적으로 정의하는 정적 및 동적 메소드 호출, 객체가 구현된 언어에 관계없이 메소드 호출이 가능한 고수준의 바인딩, 실행 시간에 필요한 서버측 구현 객체의 기능

및 관련 매개 변수를 인터페이스 저장소를 통해 제공하는 자기 기술 시스템(Self describing System), 네트워크 상의 위치나 기반 운영체제에 대한 투명성 제공, 기본적인 보안 및 트랜잭션 기능, 객체의 중요한 특성 중의 하나인 다형성을 이용한 호출, 기존 시스템과의 공존 등이 가능하게 된다[6].

라. ORB간 호환 구조

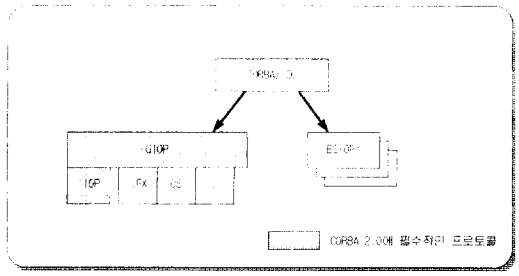
GIOP(General Inter ORB Protocol)는 ORB들 사이의 통신을 위한 일련의 메시지 형태와 공통의 데이터 표현을 명세한다. GIOP로 명세된 클라이언트의 요청 사항은 TCP/IP를 비롯해 일련의 네트워크를 공유해 다른 ORB에게 바로 전달된다. 이때 IDL로 정의된 데이터 타입을 네트워크 메시지 형태로 변환하기 위해 CDF(Common Data Format)라는 방법이 사용된다[6][8].

IIOP(Internet Inter-ORB Protocol)는 GIOP로 정의된 메시지가 TCP/IP(Transmission Control Protocol/Internet Protocol) 상에서 어떻게 전달되는지를 정의하고 있다. IIOP는 인터넷을 ORB의 기본 네트워크로 사용할 수 있다. 결국 CORBA 2.0을 지원한다는 것은 TCP/IP상에서 GIOP를 사용해 서로 다른 ORB들 사이의 메소드 호출을 지원한다는 말이다[6][13].

2.3 웹에서 Java와 CORBA의 결합

2.3.1 Java가 CORBA에게 제공하는 기능

Java는 주요 운영체제에서 작동되는 이식성 있는 객체 기반구조를 제공한다. CORBA는 네트워크



<그림 2-4> ORB 상호운용성 프로토콜

투명성을 다루는 반면, Java는 구현 투명성을 다룬다. Java가 CORBA에 제공하는 것들을 살펴보면, 다음과 같다.[6]

Java의 모빌 코드는 그것을 필요로하는 모든 CORBA 기반구조로 CORBA가 동적으로 정보(Intelligence)를 이동할 수 있도록 해준다. 이는 클라이언트와 서버 모두가 동적으로 정보를 획득하도록 한다. 즉, 재 컴파일링이 없이 클라이언트와 서버에서 실행될 수 있도록 실행시간에 어플리케이션을 분리할 수 있다.

Java는 CORBA 생명주기와 외부화 서비스를 보완한다. 즉, CORBA ORB가 객체의 상태(State)와 행위(Behavior)를 이동 가능하게 한다.

Java는 CORBA를 네트워크 어디에나 존재할 수 있도록 해준다. 넷스케이프는 모든 브라우저와 서버에 Visigenic사의 VisiBroker for Java(Java ORB와 호환)를 포함할 것이다. 이는 CORBA IIOP가 인터넷과 Java 모두에 대한 분산 객체 모델이 되도록 하는 것이다.

Java 코드를 서버에서 관리 및 배분하므로 CORBA 시스템을 단순화한다. 코드를 서버에서 한 번 갱신하면, 클라이언트는 그것이 필요로 할 때 수신할 수 있다.

Java는 CORBA 에이전트 기반구조를 보완한다.

CORBA는 분산 객체에 대한 에이전트 프레임워크를 정의한다. 이 프레임워크는 로밍(Roaming) 객체가 정의된 규칙에 따라 노드에서 노드로 이동하게 한다. 로밍 객체는 전형적으로 그것의 상태(State), 순회 내역(Itinerary), 이동간 행위(Behavior)를 운반한다.

Java는 CORBA 컴포넌트 서비스를 보완한다. OpenDoc에 바탕을 두는 CORBA의 복합문서 서비스는 모빌 저장 컨테이너 뿐만 아니라 컴포넌트에 대한 비쥬얼 컨테이너를 정의한다. OpenDoc의 벤토(Bento)에 바탕을 두는 CORBA의 모빌 컨테이너 구조는 Java Beans와 다른 컴포넌트의 이동에 이상적이다.

Java는 CORBA 객체를 작성하는 훌륭한 언어이다. Java는 클라이언트와 서버 객체 작성에 거의 이상적이다. Java의 객체 모델은 CORBA의 객체 기술을 보완한다. 즉, Java와 CORBA는 구현으로부터 객체의 정의를 분리하도록 인터페이스 개념을 사용한다.

결론적으로 CORBA는 언어 독립적인 객체 모델이다. OMG는 Java를 객체 이식을 위한 CORBA 계획의 확장으로 취급할 것이다. Java는 CORBA 바인딩에서 다른 객체 언어보다 좋으며, 또한 이식성 있는 객체 플랫폼이다.

2.3.2 CORBA가 Java에 제공하는 기능

CORBA에 의한 웹 기반구조도 Java 환경에 대하여 몇 가지 이점을 제공한다[6].

먼저, CORBA는 CGI 병목현상을 제거한다. 즉 클라이언트가 서버의 메소드를 직접 호출한다. 클라이언트는 프리컴파일된 스텁을 사용하여 파라미터를 직접 송신한다. 혹은 CORBA의 동적 호출 서비스를 사용하여 실시간에 그것을 생성한다. 양쪽의 경우에 서버는 프리컴파일된 스켈레톤을 경유하여 호출을 직접 수신한다. IDL로 정의된 서버의 어떠한 메소드도 호출할 수 있고, 문자열 대신에 정형화된 파라미터를 전송한다. 이것은 특히 HTTP/CGI와 비교되는 매우 작은 클라이언트/서버 오버헤드를 의미한다. CGI를 사용하면, 서버에 있는 메소드를 호출하는 애플릿이 매번 새로운 인스턴스를 시작해야 한다.

CORBA는 확장성 있는 서버간 기반구조를 제공한다. 서버 업무객체 풀(Pool)은 CORBA ORB를 사용하여 통신 할 수 있다. 이 객체들은 클라이언트의 요청을 적절한 분산(Load Balancing)이 되도록 다중 서버에서 실행 될 수 있다. ORB는 가장 먼저 가용한 객체에 요청을 처리하고, 요청이 증가함에 따라 다른 객체를 추가한다. CORBA는 서버 객체가 트랜잭션 제한 및 관련된 CORBA 서비스를 사용함으로써 조화롭게 작동되도록 한다. 반대로, CGI 어플리케이션은 수천 건의 요청에 응답해야 함으로 병목현상이 있다. 즉, 다중 프로세스나 프로세서로 부하는 분산할 방법이 없다.

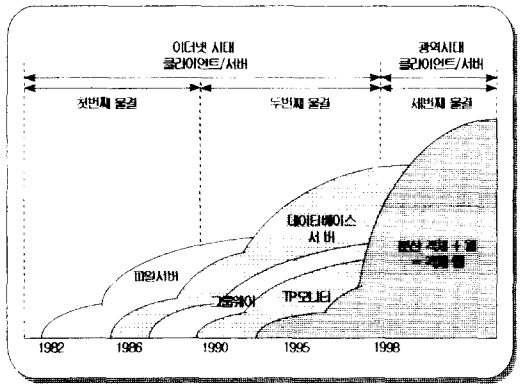
CORBA는 분산 객체 기반구조로 Java를 확장한다. 현재, Java 애플릿은 원격 메소드 호출을 사용하여 주소공간을 교차해서 통신 할 수 없다. 이것은 Java 애플릿이 원격 객체에 있는 메소드를 호출할 방법이 없다는 의미이다. CORBA는 Java 애플릿이 주소공간과 네트워크를 교차하여 다른 언어로 작성된 다른 객체와 통신 가능하도록 한다. CORBA는 Java를 확장하는 풍부한 분산 객체 서비스를 제공한다. CORBA를 사용하면 Java 클라이언트와 애플릿은 서버의 IDL로 정의된 연산을 매우 다양하게 호

출할 수 있다.

2.3.3 클라이언트/서버 객체 웹

Java는 근본적으로 모빌 코드 시스템인 반면에, CORBA는 분산 객체 기반구조이다. Java는 네트워크 내의 어떠한 기계에서도 실행될 수 있는 이식성 있는 어플리케이션을 작성할 수 있다. CORBA는 인터넷상의 분산 객체 기반 구조를 제공한다. CORBA는 Java 객체가 어디에 있는 다른 어떤 객체와도 통신가능하도록 해준다. 그러므로 이 둘은 매우 보완적인 것이다.

비록 현재는 Java의 어플리케이션들이 단독적이지만, Java의 진정한 가치는 큰 트랜잭션 시스템에서 이식성 있는 클라이언트를 만들 때이다. CORBA와 Java의 관계는 객체 웹에 대한 좀더 다양한 서비스를 제공할 때에 성장할 것이다.



<그림 2-5> 클라이언트/서버의 물결

객체 웹 기술은 이미 시작되었고, 이것은 TP모니터, 데이터베이스, 그룹웨어를 비롯한 클라이언트/서버 컴퓨팅의 모든 형태를 포함할 것이다. 객체는 거대한 단순 어플리케이션을 관리하기 쉬운 다중 업체 컴포넌트로 분리할 수 있도록 한다. <그림 2-5>는 클라이언트/서버의 물결을 보여주는데, 물결의 흐름은 파일 중심 어플리케이션(예, 네트웨어)의 물결에서 데이터베이스 중심(예, 오라클)의 물결로 이어지는 이더넷 시대(그룹웨어와 TP 모니터도 작은 파장을 형성), 그리고 객체 웹은 이어지는 큰 물결이다.

3. 분산객체를 이용한 웹기반 클라이언트/서버 구조의 구현

3.1 분산객체를 이용한 웹기반 클라이언트/서버 구조

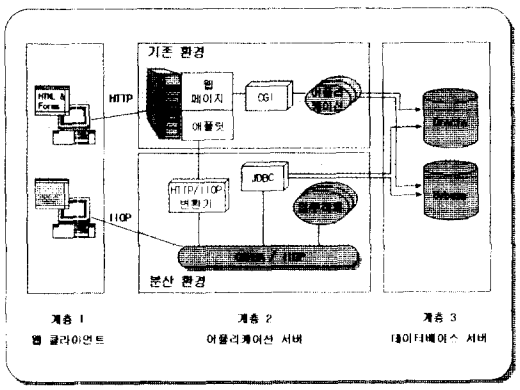
계층 1은 클라이언트에서 필요한 업무를 처리하고자 하는 사용자 관점에서의 표현을 나타내고, 계층 2는 CGI 어플리케이션을 대신하는 지속적인 업무 객체를 표현하는 서버 객체를 가지는 핵심적인 계층으로써, Java의 장점을 최대한 살린 JDBC를 포함한다. 계층 3은 기존의 데이터베이스를 표현한다. 또한 IIOP와 HTTP는 동일한 네트워크에서 운영된다.

Java 애플릿은 IIOP 프로토콜을 사용하여 CORBA 객체를 직접 호출한다. 즉, 다중 호스트와의 통신이 불가능한 애플릿 고유의 제한점을 극복함으로써, CGI, HTTP를 완전히 우회하고, ORB를 사용하여 클라이언트/서버는 직접 연결을 설정하기 때문에 최고의 확장성을 갖는다. 상태정보를 유지하므로 매번 CGI를 호출하는 오버헤드가 없다. 서버 객체가 등록되어 있으므로 빠른 응답시간이 보장된다. 프리젠테이션, 로직, 데이터가 분리된 3 계층의 구조이므

로 데이터와 프로세스의 분산처리가 이루어진다.

3.1.1 웹 클라이언트

HTML 기반의 클라이언트는 서버로부터 매번 HTML 페이지를 연속적으로 다운로드 받기 때문에, 널(Null) 클라이언트로 간주된다. 씬(Thin) 클라이언트는 완전한 GUI를 제공하고, 상태 정보를 유지한다. 그러나, 모든 어플리케이션 로직과 데이터는 서버에서 유지한다. 씬 클라이언트는 상태와 어플리케이션 로직을 포함하기 때문에 널 클라이언트의 한계를 극복한다. 씬 클라이언트는 상대적으로 작은 크기이고, 네트워크를 통해서 신속히 다운로드될 수 있다.



<그림 3-1> 분산객체를 이용한 웹기반 클라이언트/서버 구조

씬 클라이언트는 또한 소프트웨어 갱신과 배포의 문제를 극복한다. 전형적인 분산 시스템은 상대적으로 작은 수의 서버를 가지는 반면, 클라이언트의 전위(Front-ends)는 매우 많다. 새로운 버전의 클라이언트 소프트웨어를 갱신할 경우에 각각의 클라이언트 기계에 모두 설치되어야만 한다. 이것은 대단히 단순하면서 소모적이며, 고비용이 소요된다[13].

애플릿 클라이언트는 이러한 문제를 극복한다. 설치과정은 웹 브라우저에 의해서 자동적으로 이루어진다. 오늘날의 캐싱 메커니즘은 단순하다. 그러나, 애플릿의 버전 숫자에 근거한 캐싱 메커니즘의 구현이 가능하다. 이 전체 환경 프로세스는 클라이언트 애플릿은 서버에 있는 것보다 옛 버전이 캐시 되었을 때에만 다운로드 되기 때문에 보다 효과적이다[13].

3.1.2 어플리케이션 서버

가. 기존 환경

현재의 대부분의 HTTP 서버는 표준 CGI 프로토콜을 경유하여 어플리케이션을 이용한 통신이 이루어진다. 여기에서는 핵심적인 내용인 HTTP, HTML, CGI에 대해서 살펴보고자 한다.

웹은 URL(Unified Resource Locator)에 존재하는 자원을 액세스하기 위해서 HTTP(Hypertext Transfer Protocol)라 불리는 프로토콜을 지원한다. HTTP 프로토콜은 요청/응답 패러다임에 근거한다. 일반적으로 인터넷에서의 통신은 TCP/IP(Transmission Control Protocol/Internet Protocol) 연결 위에서 이루어진다. HTTP는 접속상태를 유지하지 않는데, 통신 절차는: i) 클라이언트와 서버간의 연결을 설정하고, ii) 반환된 파일을 포함하는 파라미터를 전송하고, iii) 클라이언트와 서버간의 연결을 해제한다[6][2].

HTML(Hypertext Markup Language)는 웹 문서의 공통어이다. 웹은 전세계의 거대한 파일 서버로 연결된 HTML 문서들의 거대한 집합이다. 웹 문서는 내장된 HTML 명령어들로 구성된 단순한 ASCII(American Standard Code for Information

Interchange) 문자의 파일이다. 문서의 구조를 기술하고, 폰트와 그래픽 정보를 제공하고, 다른 웹 문서나 인터넷 자원으로의 하이퍼링크(Hyperlink)를 정의하는 태그(Tag) 명령어를 사용한다 HTML 문서는 웹 서버라 불리는 HTTP 서버에 존재한다. HTML의 장점은 단순하고 이식성이 있다는 것이다. 또한 서버 프로그램은 클라이언트의 요청에 응답하는 HTML 문서를 쉽게 작성할 수 있다는 것이다[6].

CGI(Common Gateway Interface)는 웹 클라이언트가 웹 서버에 있는 프로그램을 실행하여 그 결과를 얻도록 하는 메커니즘이다. CGI 어플리케이션은 종종 실행간에 HTML 문서를 생성하며, 또한 HTML 문서로부터의 입력을 처리한다. 게이트웨이는 클라이언트가 사용할 수 있는 형태로 정보를 변환하는 프로그램이며, 웹에서의 게이트웨이는 HTML이 아닌 입력이나 데이터를 가지고 웹 브라우저에서 보일 수 있는 HTML 문서 결과를 생성하는 프로그램이다. CGI는 웹 서버에서 구현된다. CGI 구현은 웹 서버에서 필수적이지는 않지만, CGI의 사용이 일반적이므로 대부분의 웹 서버는 CGI를 포함한다[2]. CGI 프로그램은 실시간으로 실행되며, 정보를 얻은 후 클라이언트의 요청을 만족하는 동적인 웹 페이지를 작성한다. CGI는 웹을 보다 동적으로 만드는데, 야후(Yahoo)나 알타비스타(Altavista)와 같은 인터넷 검색엔진뿐만 아니라 일반 목적의 백-엔드(Back-end) 인터페이스로 사용될 수 있다[6].

나. 분산 환경

Java 애플릿은 CORBA 클라이언트일 수 있다. CORBA에는 CORBA 객체를 호출하는 Java 어플리케이션과 애플릿간의 차이는 없다. 그러나, 애플릿에 대한 웹의 보안 모델은 한계를 보인다[13].

믿을 수 없는 애플릿에 의한 위험을 방지하기 위해서, 프라이버시를 보장하는 특정한 경우에 애플릿은 오직 호스트로부터 그것이 다운로드된 곳까지의 네트워크 연결만이 가능하다. 이 모델은 클라이언트가 그들의 물리적인 위치에 무관하게 객체의 연산을 호출하는 CORBA와 상충되는 것이다.

1) HTTP 터널링

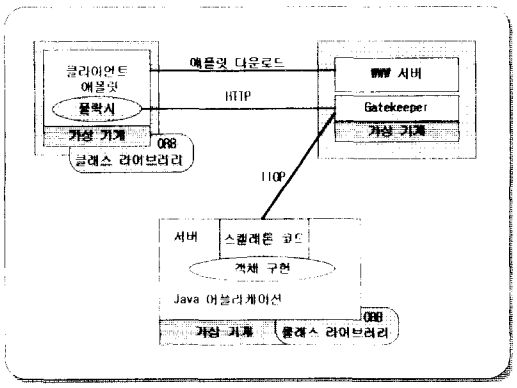
Visigenic사의 VisiBroker for Java는 <그림 3-2>와 같은 HTTP 터널링(Tunneling) 접근을 제공한다[13].

HTTP 터널링은 다음과 같은 과정을 따른다.

i) IIOP 호출을 방화벽을 통과할 수 있도록 HTTP 포장(Envelope)으로 번역한다.

ii) 애플릿이 다운로드 되어지는 호스트에 존재하는 데몬(Daemon)에게 모든 요청을 한다. 이것은 객체 참조에서 지정된 호스트에게 요청을 하는 데몬으로써 CORBA 위치 투명성을 제설정한다.

VisiBroker는 *pomoco.iiop.Gatekeeper* 클래스에서 구현된 특정한 목적의 HTTP 데몬을 제공한다.

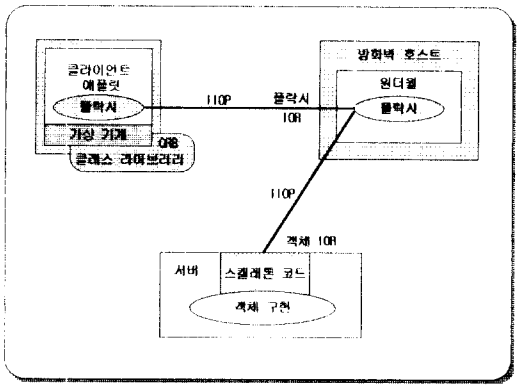


<그림 3-2> HTTP 터널링

Gatekeeper는 방화벽 외부의 웹 서버에서 실행되는데, IIOP 메시지의 통과가 불가능한 방화벽을 통하여 클라이언트와 객체간의 통신을 가능하게 한다. 또한 클라이언트가 애플릿이 다운로드된 서버의 객체와 바인딩이 가능하도록 한다[11].

2) 원더월

Iona사의 OrbixWeb은 원더월(Wonderwall) 접근을 제공한다. 위치 투명성 제공에 관하여 원더월은 HTTP 터널링 접근과 유사하다. 이것은 클라이언트로부터의 IIOP 요청을 수신하는 데몬을 제공하고, 그것을 실제 객체로 보고, <그림 3-3>에서 보는 바와 같이 동일한 방법으로 결과를 반환한다[13].



<그림 3-3> 원더월

원더월에 의해 접근되는 방화벽 문제는 HTTP 터널링과는 다른데, 원더월은 기존의 방화벽을 사용하지 않고 자신의 방화벽을 사용한다. 원더월은 보안 사설 네트워크와 인터넷 사이에 설치된다.

원더월 접근은 많은 결점을 갖는다. 원더월은 복잡한 정책적이고 관리적인 새로운 소프트웨어의 설치를 요구하기 때문에, 그때 그때의 솔루션을 제공하지 못한다. 보다 의문시되는 것은 원더월이 객체 식별을 깨뜨린다는 것이다. 원더월에 의한 방화벽을 통과하는 서버를 호출하기 위해서는 객체 참조는 원더월에 있는 플락시(Proxy) 객체의 주소 정보를 포함하도록 수정될 필요가 있다. 이것은 객체마다 한 쌍의 IOR (Interoperable Object Reference)에 이르고, 원래의 것은 원더월 없이 사용되고 플락시의 IOR은 원더월로 사용된다.

3.1.3 데이터베이스 서버

데이터베이스 서버는 기존의 관계형 데이터베이스뿐만 아니라 객체지향 데이터베이스까지 확장된 데이터의 접근이 가능토록 한다.

업무 객체가 데이터베이스 서버로 SQL 요청 메시지를 보내면, 데이터베이스 서버는 각 SQL 명령어의 결과를 반환한다. 데이터베이스 서버는 모든 레코드를 클라이언트나 어플리케이션 서버로 보내어 연산하지 않고 자신의 처리 능력을 사용한다. 데이터베이스 서버는 즉흥적인(Ad hoc) 질의와 융통성 있는 보고서를 요구하는 의사결정 지원 시스템에 활용될 수 있다[7].

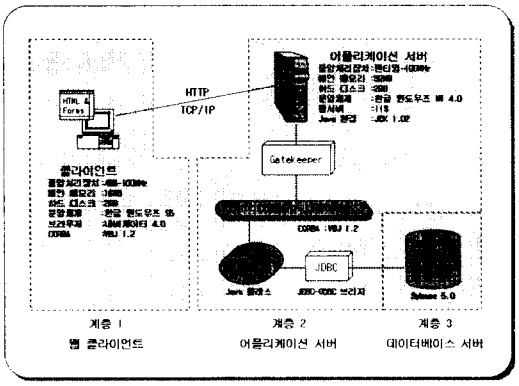
3.2 도서 검색 시스템의 구현

3.2.1 구현 환경

본 논문의 구현을 위한 환경은 <그림 3-4>와 같이 웹 클라이언트, 어플리케이션 서버, 데이터베이스 서버의 세 부분으로 나누어진다.

웹 클라이언트의 구성은 다음과 같다: CPU는 486-100MHz, 메인 메모리는 16MB, 하드디스크는 2GB, 운영체제는 한글 윈도우즈 95, 브라우저는 넷스케이프 내비게이터 4.0, CORBA는 VisiBroker for Java 1.2를 사용하였다.

어플리케이션 서버의 구성은 다음과 같다: CPU는 펜티엄-100MHz, 메인 메모리는 32MB, 하드디스크는 2GB, 운영체제는 한글 윈도우즈 NT 4.0, HTTP 프로토콜을 서비스하기 위한 웹 서버는 마이크로소프트사의 인터넷 인포메이션 서버(IIS), CORBA는 VisiBroker for Java 1.2를 사용하였다. 또한 HTTP 프로토콜과 IIOP 프로토콜간의 변환을 위해서 Gatekeeper가 동작한다.



<그림 3-4> 구현 환경

그리고, 데이터베이스 서버는 사이베이스 SQL Anywhere 5.0을 사용하였으며, 어플리케이션 서버와 데이터베이스간의 인터페이스는 JDBC-ODBC 브리지에 의한 미들웨어를 이용하였다.

3.2.2 도서 검색 HTML과 검색과정

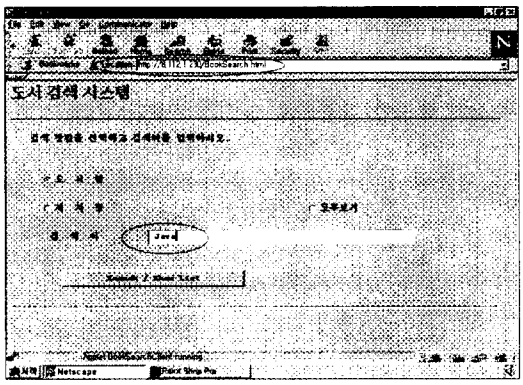
클라이언트 애플릿을 실행하기 위해서 <표 3-1>과 같은 도서 검색 HTML을 사용한다.

<표 3-1> 도서 검색 HTML

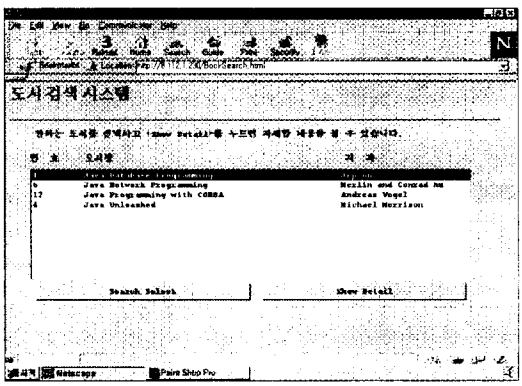
```
<!--BookSearch.html-->

<HTML>
  <HEAD>
    <TITLE>BookSearch.html</TITLE>
  </HEAD>

  <BODY>
    <H2>도서 검색 시스템</H2>
    <HR>
    <CENTER>
      <APPLET
        code=BookSearchClient.class
        codebase=classes width=700 height=280>
      </APPLET>
    </CENTER>
    <HR>
  </BODY>
</HTML>
```

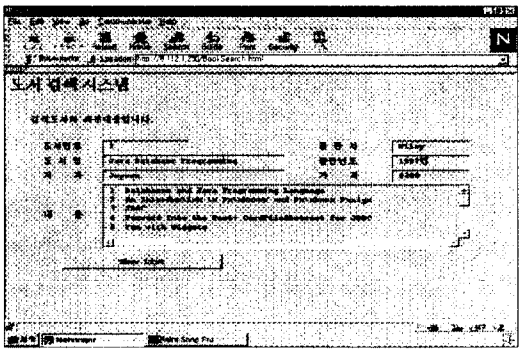


<그림 3-5> 검색 방법 선택 및 검색어 입력



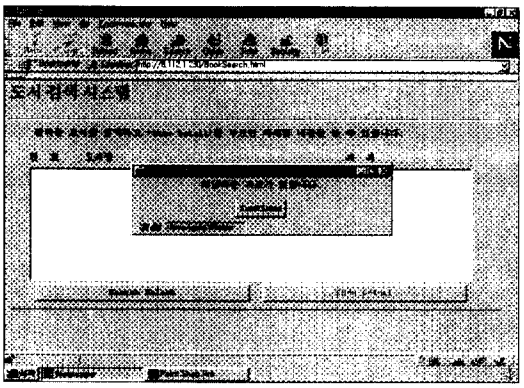
<그림 3-6> 검색 결과 해당 도서의 목록

도서 검색 시스템의 검색과정을 그림으로 살펴보면 다음과 같다. <그림 3-5>의 상단에 표시된 것과 같이 웹 브라우저에서 검색하고자하는 도서 검색 시스템의 HTTP 서버의 페이지를 지정한다(http://8.112.1.230/BookSearch.html). 그러면, 서버의 IIS에서 HTTP 프로토콜에서 해당 페이지와 애플릿 클래스를 보내주어 <그림 3-5>와 같은 페이지가 브라우저에 전시된다. 여기에 검색 방법을 도서명, 저자명, 모두보기 중에서 선택하고, 도서명, 저자명의 경우에는 필요시 검색어를 입력할 수 있다. 본 그림에서는 도서명으로 "Java"를 검색어로 입력하였다. 이에 대한 검색의 결과가 <그림 3-6>에서와 같이 도서명이 "Java"로 시작되는 4권의 도서 목록이 나타난다. 이때, "Java Database Programming"의 내용을 자세히 알아보고 싶은 경우에 "Show Detail" 버튼을 누르면, <그림 3-7>과 같이 나타난다.

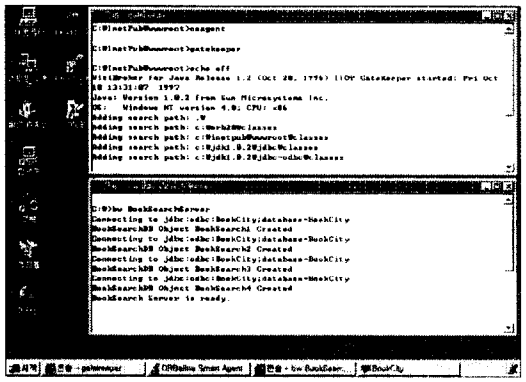


<그림 3-7> 검색 도서의 세부 내용

<그림 3-5>에서 입력한 검색어에 해당하는 도서가 존재하지 않으면, <그림 3-8>과 같이 "해당하는 자료가 없습니다"라는 메시지를 대화창을 통하여 사용자에게 알려준다.



<그림 3-8> 도서 검색 시스템의 메시지 대화창



<그림 3-9> 도서 검색 시스템의 서버측 대기 상태

서버 객체는 자신의 메소드가 호출되도록 클라이언트의 요청을 기다리게 되는데, <그림 3-9>에서 위쪽의 콘솔 윈도우는 HTTP-IIOP 서비스를 제공하기 위한 Gatekeeper가 대기중인 상태이고, 아래쪽의 콘솔 윈도우는 도서 검색 시스템의 서버 객체인 BookSearchServer가 실행되어 클라이언트의 요청을 기다리고 있는 상태를 나타낸다.

3.2.3 도서 검색 시스템의 클라이언트

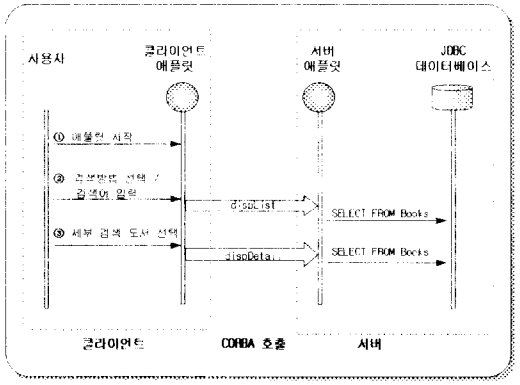
클라이언트/서버 어플리케이션은 본래 클라이언트 위주이다. Java 애플릿은 사용자가 시각적으로 클라이언트를 조작하게 된다. 사용자는 이러한 시각적인 양식에서 CORBA 자원의 메소드를 호출하게 된다. 한편, 서버 객체는 자신 메소드의 실행이 트리거 되도록 클라이언트로부터의 요청을 수동적으로 기다린다.

도서 검색 애플릿의 시나리오를 도식화하면 <그림 3-10>에서 보는 바와 같다. 먼저, 도서 검색 HTTP 서버를 지정한다. 즉, Java가 가능한 브라우저를 작동시킨다(예, 넷스케이프 내비게이터). 그리고, 도서 검색을 위한 서버의 웹 페이지를 지정한다.

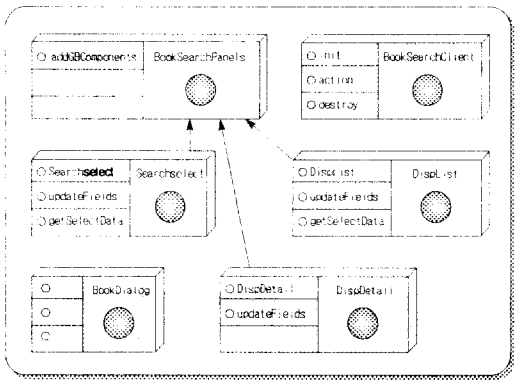
둘째, 검색방법 선택 및 검색어 입력한다. 즉, 도서명, 저자명, 모두보기 중에서 검색방법을 선택하고, 도서명이나 저자명에 의한 검색의 경우에 검색어를 입력한다. 입력된 검색어는 대소문자에 무관하며, SQL의 문자 연산자인 퍼센트(%)를 사용하여 입력된 검색어의 이어지는 뒷 문자는 상관없다. 검색된 목록은 도서명순 또는 저자명순으로 목록이 전시된다.

셋째, 세부 검색도서 선택한다. 즉, 도서의 목차내용 등에 관한 세부적인 내용을 보고자 할 경우에, 원하는 도서를 선택하면, 자세한 내용을 볼 수 있다.

<그림 3-11>은 도서 검색 클라이언트의 애플릿 클래스 구조를 나타낸다. 그림에서 모든 패널(Panel) 클래스는 BookSearchPanels 부모 클래스로부터 상속받는다. BookDialog 클래스는 팝 업 윈도우 다이얼로그로서 각종 메시지 처리를 담당한다. Book-SearchClient 클래스는 전형적으로 애플릿 클래스로부터 상속되었다.



<그림 3-10> 도서 검색 애플릿의 시나리오



<그림 3-11> 애플릿 클래스의 구조

BookSearchClient는 java.applet.Applet를 확장하고, init, action, destroy의 3가지 메소드를 구현한다. 브라우저는 애플릿이 적재된 후에 init를 호출한다. init 메소드는 다음의 기능을 수행한다: i) 새로운 CardLayout 객체를 생성하고, ii) 5개의 새로운 패널 객체를 생성하고, iii) CardLayout에 패널을 추가하고, iv) ORB를 초기화하고, v) 서버에 있는 BookSearchDispenser 객체를 가리킨다. vi) CORBA BookSearch 서버 객체로 참조를 획득하기 위해서 readyBookSearchObject를 호출한다. vii) 최상위

패널에 있는 updateFields를 호출한다.

브라우저는 사용자가 패널의 어느 버튼을 누르면 action 메소드를 호출하고, 애플릿이 종료되면 dest -roy 메소드를 호출한다. 이 메소드는 BookSearch -Dispenser 서버에 있는 releaseBookSearchObject를 호출한다.

BookSearchPannels은 모든 패널 클래스의 상위 클래스(Superclass) 이다. 이것은 다음의 2가지 기능을 제공한다. i) 각각의 유도된 클래스에 의해서 사용되는 CardBagLayout과 CardBagConstraints를 자동적으로 생성한다. ii) 그리드(Grid) 내부에 AWT (Abstract Window Toolkit) 컴포넌트를 위치시키는 addGBComponent 메소드를 제공한다.

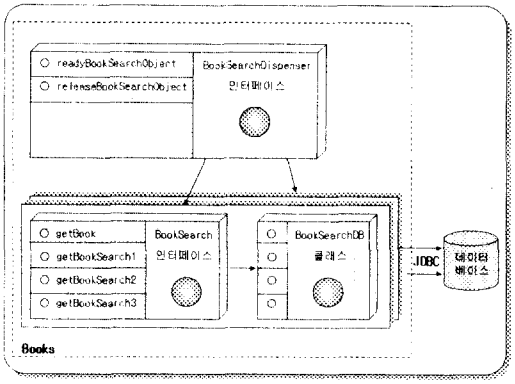
추가적으로, Searchselect, DispList, DispDetail의 3가지 패널 클래스가 있는데, 이 클래스들의 각각은 updateFields 메소드를 통해서 화면의 해당 필드를 직접 갱신하고, getSelectData 메소드를 통해서 패널의 필드에서 데이터를 추출한다.

3.2.4 도서 검색 시스템의 서버

도서 검색 프로그램은 3-계층의 클라이언트/서버 프로그램이다. 여기에서는 3-계층의 요소들을 설명하고자 한다.

3-계층 도서 검색 프로그램에서, CORBA 클라이언트는 CORBA 서버 객체와 통신한다. 서버 객체는 JDBC를 경유하여 하나 이상의 데이터베이스와 통신한다. 도서 검색 클라이언트는 BookSearchClient라는 새로운 애플릿에 의해 제공된다. 중간 계층의 어플리케이션 서버는 3 개의 주요부분으로 구성된다: i) 서버 객체의 풀(Pool)을 관리하는 BookSearch -Dispenser, ii) 클래스 BookSearch의 서버 객체 풀, 그리고 iii) 지속적인 JDBC 연결을 담당하는 클래스 BookSearchDB의 헬퍼 객체의 풀이다. 3-계층 도서 검색 프로그램에서 계층 3은 JDBC 데이터베이스로 구성된다. 이는 지속적으로 상태가 저장된다.

BookSearchDispenser는 도착순서에 의해 클라이언트를 할당함으로써, 서버 객체의 풀(Pool)을 관리하는데, <그림 3-12>에서 보는 바와 같다. 즉, BookSearch 객체의 분배를 담당한다. 각 Book -Search 객체는 자신의 BookSearchDB JDBC 헬퍼 객체를 시작한다. 클라이언트는 검색 작업이 완료되면, 사용한 객체를 해제하여야 한다.



<그림 3-12> BookSearch 객체의 스케쥴러 BookSearchDispenser

BookSearchImpl은 IDL로 정의된 BookSearch 인터페이스를 구현한다. 클래스 생성자는 새로운 BookSearch 객체를 생성하고 연결한다. 이 객체는 데이터베이스로 미리 연결된다.

이러한 메소드는 BookSearchDB 객체에 있는 헬퍼 객체에 상응하는 클라이언트 요청을 서비스한다.

BookSearchDB는 데이터베이스를 캡슐화하는 클래스이다. 이것은 JDBC와의 모든 상호작용을 조정한다. 이 클래스는 connect, closeConnection, get

-BookSearch1, getBookSearch2, getBook Search3, getBook 등의 6개의 메소드를 제공한다. 뒷부분의 4개의 메소드는 BookSearchImpl에 일대일로 상응한다. 이것이 헬퍼인 것이다.

최소한 BookSearchDispenserImpl은 서버 객체의 풀을 유지한다. 각 서버 객체는 BookSearchDB 객체를 경유해서 Sybase 데이터베이스로의 살아있는 JDBC 연결을 유지한다.

3.2.5 BookSearch IDL

CORBA IDL은 훌륭한 명세 언어의 장점을 가지는데, 이것은 서버와 클라이언트간의 통신을 정의한다. <표 3-2>는 도서검색 IDL을 보여준다.

<표 3-2> 도서검색 IDL

```
// BookSearch.idl

module Books
{
  exception BookException
  { string reason;
  };

  typedef long dispNo;

  struct dispDetailStruct
  {
    long   id;
    string title;
    string author;
    string publish;
    long   price;
    string year;
    string content;
  };
  typedef sequence<dispDetailStruct> dispDetailSeq;

  struct dispSumStruct
  {
    dispNo id;
    string title;
    string author;
  };
  typedef sequence<dispSumStruct> dispSumSeq1;
  typedef sequence<dispSumStruct> dispSumSeq2;
  typedef sequence<dispSumStruct> dispSumSeq3;

  interface BookSearch
  {
    dispDetailStruct getBook(in dispNo id);
    dispSumSeq1 getBookSearch1
       (in string search_method, in string title);
    dispSumSeq2 getBookSearch2
       (in string search_method, in string author);
    dispSumSeq3 getBookSearch3
       (in string search_method, in string id);
  };

  interface BookSearchDispenser
  {
    BookSearch readyBookSearchObject()
         raises (BookException);
    void releaseBookSearchObject
         (in BookSearch bookSearchObject)
            raises (BookException);
  };

};
```

앞의 IDL은 BookSearch와 BookSearchDispenser의 두 인터페이스를 정의한다. 또한 파라미터로 보내지는 데이터 구조도 정의한다. IDL은 매우 가독성이 높다.

3.2.6 BookCity 데이터베이스

Bookcity 데이터베이스는 Books 테이블 하나로 간단히 구성되어지며, 그 명세는 <표 3-3>과 같다. 또한 서버 객체인 BookSearchServer가 실행되면 BookCity 데이터베이스 서버가 대기하게 되는데, <그림 3-9>의 아래 작업 표시줄의 우측 하단에 BookCity 버튼이 이를 나타낸다. BookCity 데이터베이스 서버는 클라이언트가 도서 검색 요청을 하면 JDBC에 통해서 결과값을 클라이언트로 보내게 되는 것이다.

<표 3-3> Books 테이블의 명세

필드명	타 입	설 명
Id	Integer	도서 코드
Title	Char(60)	도서명
Author	Char(20)	저자명
Publish	Char(20)	출판사
Price	Integer	가격
Year	Char(4)	출판년도
Content	Char(500)	내용

4. 결 론

현재의 인터넷을 바탕으로 하는 웹기반 HTTP /CGI 클라이언트/서버 구조의 문제점에 대한 해결책이 될 수 있는 기술인 Java와 CORBA에 대해서 알아보았다. Java는 모빌 코드 시스템이며, 이식 가능한 코드를 작성할 수 있기 때문에 구현 투명성을 제공하여 준다. CORBA는 분산 객체의 기반 구조를 제공하기 때문에 네트워크 투명성을 제공한다. Java와 CORBA의 두 기술을 결합하여 이용함으로써 분산 객체 환경에서 강력한 클라이언트/서버 구조를 만들 수 있다.

네트워크 어느 곳에서도 원하는 지식을 이용할 수 있는 CORBA의 분산 객체를 기반으로 하고, 네트워크 환경에서 특히 탁월한 성능을 발휘하는 Java 언어를 사용하고, 또한 Java의 스타일과 장점을 가장 잘 살린 JDBC를 데이터베이스 서버로 접근하는 미들웨어로 사용하였다. 이러한 환경에서 분산 객체를 이용한 웹기반의 클라이언트/서버 구조인 "도서 검색 시스템"을 구현하였다.

현재 객체 웹 기술은 아직은 시작단계이며, 이것은 TP 모니터, 데이터베이스, 그룹웨어와 같은 다른 형태의 클라이언트/서버 컴퓨팅을 포함할 것이다.

기존의 구축되어 있는 종래의 어플리케이션 및 데이터들에 대한 재사용 문제를 해결할 수 있도록 통합의 관점을 고려한 연구도 병행하면서 현재 개발이 한창 진행중인 여러 가지의 CORBAservice들을 이용하여 다양한 서비스들을 제공받을 수 있도록 한다면, "정보 고속도로(Information Highway)"에서 분산 객체를 이용한 클라이언트/서버 구조를 효과적으로 구축할 수 있을 것이다.

참 고 문 헌

[1] George Bond, "JAVA Unleashed", Sams.net Publishing, 1996

[2] John Deep & Peter Holfelder, "Developing CGI Applications with Perl", John Wiley & Sons Inc., 1995

[3] Graham Hamilton & Rick Cattell, "JDBC : A JAVA SQL API", JavaSoft, 1997

[4] Richard Mateosian, "J/SQL User Guide and Reference", 1997

[5] OMG, "The Common Object Request Broker: Architecture and Specification", OMG, 1996

[6] Robert Orfali, Dan Harkey, "Client/Server Programming with Java and CORBA", John Wiley, 1997

[7] Robert Orfali, Dan Harkey, Jeri Edwards, "The Essential Client/Server Survival Guide", John Wiley, 1996

[8] Jon Siegel, "CORBA Fundamentals and Programming", John Wiley & Sons, Inc., 1996

[9] SunSoft, "JDBC Guide: Getting Started",

SunSoft, 1997

[10] Paul M. Tyma, "Java Primer Plus", Waite Group Press, 1996

[11] Visigenic Software, "Distributed Object Computing in The Internet Age", Visigenic Software, Inc., 1997

[12] Visigenic Software, "Enterprise Database Access, Java Style", Visigenic Software, Inc., 1997

[13] Andreas Vogel, Keith Duddy, "Java Programming with CORBA", John Wiley & Sons, Inc., 1996