2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

Corba Web서버 시스템의 설계 및 구현

황호찬*, 김광수*, 박규석*
*경남대학교 컴퓨터 공학과
e-mail:kspark@kyungnam.ac.kr

Design and Implementation of A Corba Web Server System

Ho-Chan Hwang*, Kwang-Su Kim*, Kyoo-Seok Park*
*Dept of Computer Engineering, Kyung-Nam University

요약

인터넷의 서비스개선으로 사용자의 수는 기하급수적으로 늘어나고 있으며, 이러한 경향에 발맞추어 홈페이지의 수도 날로 늘어가고 있는 추세이다. 홈페이지를 만드는데 있어서 필수적인 것이 바로 Web Server라고 할 수 있으며, 현재 Web Server에 대한 연구가 계속되고 있는 추세이다. 본 논문에서는 현재 사용되고 있는 Web Server보다 좀더 개선된 Server 시스템을 제안하였으며, 이 Server 시스템은 OS에 독립적일 뿐만 아니라 사용자에 대한 서비스 시간과 사용 자원을 줄일 수 있다.

1. 서론

사용자들은 웹에서 쉽고 편리하게 서비스를 이용할 수 있게 되었지만 참조 무결성, 이동 투명성 그리고 유지·보수의 어려움과 같은 웹의 문제점을 인식하게 되었다[1][2]. 또한 웹서비스의 수요가 급속히 증대해가면서 서비스와 응용 프로그램들도 점점 커져가고 있다. 하지만 향상된 서비스를 제공하기 위해서 기존의 시스템 및 정보 등을 웹에 통합한 응용 프로그램을 개발하는데 어려움을 지니고 있다. 또한 웹 시스템은 클라이언트/서버 모델에 기반을 두고 있으며 통신 때 사용되는 프로토콜인 HTTP(Hyper Text Transfer Protocol)는 상태 정보를 유지하지 않는(stateless)특성을 지니고 있다. 이는 서버의 부하를 줄여주는 장점은 가지고 있으나 올바른 자원의 참조가 이루어지지 않을 위험이 있다.

본 논문에서는 이러한 문제점을 개선하기 위하여 분산 객체 기술을 이용한 웹 지원 시스템인 CWS(Corba Web Server)를 설계 및 구현하였다. 분산 객체 기술을 이용하기 위해서 CORBA(Common Object Request Broker Architecture) 및 Java를 이용한다.

본 시스템은 객체 지향 웹서버를 이용하여 자원과 응용 프로그램의 상태성을 유지하며, 객체 단위로 자원의 관리가 이루어진다. 따라서 대규모화되는 웹 자원의 유지·보수도 객체 지향 기반으로 효율적으로 이루어지며, 이러한 객체 단위의 관리는 더 높은 수준의 유연성과 확장성을 제공할 수 있다.

2. 관련 연구

2.1 인터넷 네트워크 기술

2.1.1 TCP/IP

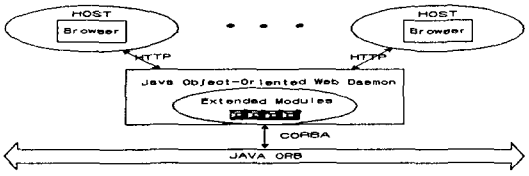
TCP/IP(Transmission Control Protocol / Internet Protocol)는 OSI(Open Systems Interconnection) 모델을 직접 따르지 않으며, 그 계층은 Application Layer, Transport Layer, Internet Layer, Physical Layer로 구성된다.

2.1.2 WWW(World Wide Web)

HTTP(Hyper Text Transfer Protocol)는 분산환경 및 정보서비스 제공을 목적으로 설계된 응용계층의 프로토콜로서 웹(World Wide Web)에서의 하이퍼텍스트 문서의 전송을 위해 쓰이는 프로토콜이다. 또한 요구 명령어의 추가를 통해 네임 서버나 분산 객체 관리 시스템 등과 같은 여러 가지 일에도 사용할 수 있는 객체 지향 프로토콜이며, MIME(Multipurpose Internet Mail Extension)에 의해 정의될 수 있는 모든 문서 형식을 전송할 수 있다[3].

3. 시스템 설계

본 논문에서 제안하는 CWS(Corba Web Server)를 구현하기 위해 Java로 작성된 웹 서버 데몬 및 서버의 확장된 모듈 그리고 CORBA의 기능을 제공하는 Java ORB를 사용한다. Java로 작성된 객체 지향 웹 서버는 기존의 많은 웹 서버 데몬과 같은 역할을 하고 또한, 확장된 모듈은 웹 문제점을 개선하며 CORBA 자원과 서비스의 활용을 가능하게 한다.

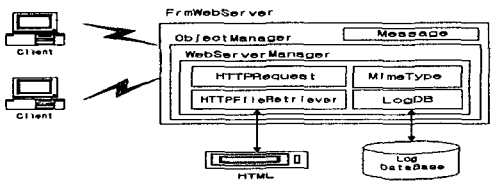


[그림 1] 시스템 구조

Java ORB는 Java로 작성한 응용 프로그램에서 CORBA 자원으로 직접 접근을 가능하게 하여 유연성과 확장성을 향상시킨다. CWS 시스템은 그림 1에서처럼 두 부분으로 구성된다. 첫째는 객체 지향 웹 서버이다. 객체 지향 웹 서버는 객체 단위로 자원을 관리하며 객체 관리자가 그 기능을 담당하게된다. 둘째는 웹의 최하위 구조로서 CORBA의 기능을 완벽하게 제공하는 Java ORB이다. Java ORB는 CORBA의 자원 및 이름 서비스를 제공한다.

3.1 CWS 구성

시스템의 전체 구성도는 다음이 클라이언트의 요구에 응답할 Daemon으로 구성된다.

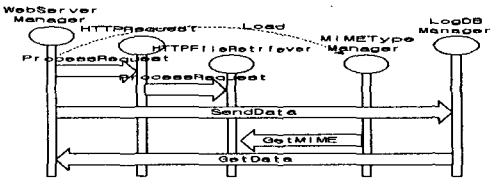


[그림 2] CWS 시스템 구성

CWS 시스템의 구성은 그림2와 같으며 Application인 FrmWebSerber, 각 모듈을 관리하는 ObjectManager, 그리고 각 모듈을 활성화시키는 WebServerManager, 클라이언트 요구에 서비스할 HTTPRequest, HTTPFileRetriever, MIME Type을 관리하게 되는 MIMETypeManager, 클라이언트 정보를 저장할 LogBManager로 구성되어 클라이언트에게 서비스를 제공한다. 그리고 구현객체에서 서비스하는 내용을 보여주기 위한 Message 모듈로 구성된다.

CWS시스템의 전체 시나리오는 그림3과 같다. WebServerManager가 클라이언트의 요구를 받게 되면 응답할 모듈인 HTTPRequest를 활성화시키기 위해 ProcessRequest 이벤트를 생성한다. 그리고 로컬에 저장된 HTML 문서와 MIME Type의 내용을 클라이언트에게 전송하기 위한 HTTPFileRetriever을 활성화할 ProcessRequest를 생성한다.

HTTPFileRetriever이 클라이언트에게 HTML 문서를 전송할 때 MIMEType Manager로 등록된 MIME Type을 전송 받게되고 클라이언트에 대한 정보를 LogDBManager에게 전송하여 결과를 Data Base에 저장한다.



[그림 3] CWS 시스템의 시나리오

3.2 모듈별 기능

CWS의 각 모듈별 기능은 다음과 같다.

3.2.1 ObjectManager

구현객체와 연결하게 되는 모듈이다. 그리고, 향후 객체의 추가나 기타 다른 객체의 사용을 용이하도록 하기 위한 역할을 수행한다.

3.2.2 WebServerManager

WebServerManager는 클라이언트 요구를 감시하는 이종의 Daemon역할을 하는 모듈이다. 이 모듈의 역할은 클라이언트의 요구가 들어왔을 경우 HTTPRequest 모듈을 활성화시키고 MIMEType로부터 설정된 MIME Type의 내용을 전송 받게 된다. HTTPRequest가 활성화된 후 HTTPFileRetriever 모듈은 로컬에 있는 HTML 문서를 클라이언트에게 서비스한다. 또한, 접속된 클라이언트에 대한 정보를 LogDBManager에게 그 정보를 전송 하게 된다. 또한 현재 상태를 확인하기 위해 클라이언트가 요구한 파일의 이름을 보여주기 위해 Message 모듈에게 그 내용을 전송하여 화면상에 보여주는 기능을 가진다.

- 기본 알고리즘을 이용한 Web Server
  ① 클라이언트가 요구한 Socket을 Open 하여 클라이언트와 연결한다.
  ② 클라이언트가 요구한 파일, 디렉토리 그리고 Method를 확인한다.
  ③ 로컬에 있는 HTML 문서를 파싱 한다.
  ④ Open된 Socket으로 파싱한 HTML문서를 전송한다.
  ⑤ Socket을 Close한다.

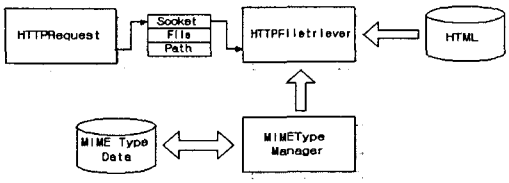
3.2.3 HTTPRequest

HTTPRequest는 WebServerManager로부터 클라이언트의 요구에 활성화되는 모듈로서, 모듈을 초기화한 후 클라이언트가 요구한 디렉토리의 위치, 파일의 이름, Host 이름, Method 방식 등을 지정하거나 확인한다. 그리고 구성된 FORM Method가 POST또는 GET 방식에 따라서 클라이언트에게 서비스 할 수 있도록 확인하는 모듈이다. 클라이언트에게 응답할 Server Socket을 Open하고 이상유무를 체크한 후 전송하게 되는 파일의 이름과 내용을 확인하고 그 내용을 HTTPFileRetriever 모듈에게 전송한다.

Method는 POST와 GET 방식의 가장 큰 차이점은 "?" 의 존재 여부로 구분할 수 있다. GET 방식은 "?" 뒤에 전송할 코드가 함께 넘겨진다.

3.2.4 HTTPFileRetriever

HTTPFileRetriever 모듈은 HTTPRequest 모듈로부터 전송 받은 파일을 다시 확인하고 로컬에 저장되어 있는 HTML문서를 파싱해서 그 내용을 클라이언트에게 전송하게 된다. 그리고 전송 시 MIMEType 모듈로부터 이미 설정되어 있는 MIME Type에 따라 서비스하는 모듈이다.



[그림 4] HTTPFileRetriever 데이터 흐름도

그림 4는 HTTPFileTriever 모듈을 중심으로 데이터 흐름을 나타낸 것이다. HTTPRequest 모듈은 클라이언트와 연결한 Socket 및 전송 요구를 받은 파일 이름, 파일이 있는 경로를 나타내는 Path 등 3개의 데이터를 HTTPFileTriever 모듈에게 전송한다. 이러한 작업이 정상적으로 이루어지면 HTTPFileTriever 모듈은 로컬에 저장된 HTML 문서를 파싱해서 클라이언트와 연결된 Socket으로 전송할 준비를 하게 된다. 이때 MIMEType 모듈은 로컬에 저장된 MIME Ttype 데이터를 읽어들여 HTTPFileTriever 모듈에게 전송하여 클라이언트가 요구하는 대로 처리한다.

3.2.5 MIMEType

MIME Type의 내용을 Network Working Group에서는 MIME Type을 RFC 822에서 정의하고 있다. 이러한 구성요소들은 확장 가능한 메카니즘과 다양한 표현형식으로 전송할 수 있도록 하기 위해서이다.

MIMEType 모듈은 로컬에 저장된 MIME Type 파일을 관리하게 된다. WebServerManager 모듈에서 확인하는 것은 로컬에 저장된 파일이 정상적인지를 확인하는 것이며, 실제적으로 MIME 데이터를 불러들이는 것은 HTTPFileRetriever 모듈에서 처리하게 된다. HTTPFileRetriever 모듈이 로컬에 저장된 HTML 문서에 MIME Type이 있는지 확인하고 있으면 이에 대한 처리를 MIMEType 모듈에게 넘겨주어 처리하도록 처리한다. 또한 HTTPFileRetriever 모듈이 설정된 MIME 요구 시 getMIME 이벤트로 자료를 전송하여 클라이언트 요구에 대한 서비스를 담당한다.

3.2.6 LogDBManager

웹을 통해 많은 수의 클라이언트 접속이 이루어지며, 이에 따라 서버가 받게되는 부하라든가, 빠른 처리시간의 요구, 다량의 데이터 전송 등을 위한 DBMS에의 접근을 위해 ODBC에 연결하여 질의를 처리하였으며, 이때 구현객체의 ODBC 연결에는 JDBC(Java DataBase Connectivity)를 사용하였다.

LogDBManger 모듈은 WebServerManager 모듈로부터 전송된 클라이언트 정보를 DB에 저장하고 관리하는 모듈이다. 저장되는 내용은 클라이언트의 IP 주소, IP 주소에 대한 Host이름, 클라이언트가 접속한 날짜와 시간, 그리고 HTML 전송 시 사용된 Method와 전송하는 HTML 파일이름을 저장하고 관리한다.

3.3 알고리즘

그림 5의 알고리즘에서 서버가 클라이언트의 요구를 대기하다가 접속이 이루어지면 사용자의 요구에 맞는 서비스를 행하게 됨을 알 수 있다.

또한 웹서버에서 기본적으로 지원해야 될 MIME을 구성하게 되고 사용자가 GET 혹은 POST중 어느 방식을 사용하였는지 확인한 후 그에 맞는 서비스를 수행 하도록 되어 있다.

```
public void openServerSocket()
{
    Socket Open
}
public void run()
{
    // IMPLEMENT: Operation
    Socket requestSock;
    while (!isTerminated()) {
        상용자 서버스요규 대기 및 서비스 시작
    }
    try { closeSocket(serverSock); }
    catch (Exception e) { }
}
void processRequest(Socket requestSocket) throws IOException
{
    try {
        If (getMethod().equals("GET")) {
            GET방식으로 넘어온 데아타 Parsing
        }
        else if (getMethod().equals("POST")) {
            POST방식으로 넘어온 데아타 Parsing
        }
        else formVariables = new String("Unknown Method");
    } catch (Exception e) {
    formVariables = "";
}

public static void load() throws IOException
{
   try {
      ObjectOutputStream out = new ObjectOutputStream(
    FileOutputStream("mimetypes.dat"));
        Default MIME TYPE구성
    } catch(Exception e) {}
    out.writeObject(dict);
    out.close();
}

public void connect(java.lang.String url)
{
    ODBC연결
}

public void insert(Cws.UserInfoDef userInfo)
{
    사용자정보저장
}
```

[그림 5] 알고리즘

4. 구현 및 평가

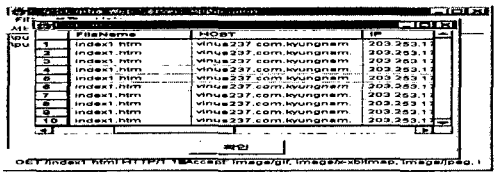
4.1 구현

본 논문에서 제안한 CWS 시스템의 설치 및 운용에 대한 구현 환경은 표 1에서 나타낸 것과 같다.

[표 1] 구현환경

분류	CWS 시스템	클라이언트
사용OS	Windows NT 4.0	Windows 95
프로세서	펜티엄 200	펜티엄 166
RAM 크기	64M	40M
개발 도구/ 브라우저	Visual Cafe 3.0 Inprise의 VisiBroker	Netscape
사용 DB	Access97	

제안 시스템은 환경설정에 등록된 디렉토리에 HTML 문서를 작성해서 CWS 시스템을 동작시킨다.



[그림 6] Log 화면

그림 6은 클라이언트가 접속한 내용에 대한 정보를 보고자 할 때의 처리 화면이며, 클라이언트 주소, Host 이름, 접속 날짜, 시간, Method, 파일 이름 등을 볼 수 있다.

4.2 평가 및 분석

본 논문에서 설계한 CWS 시스템과 Sun 사에서 발표한 Java Web Server와 수행 성능을 비교하였으며, 표 2는 각각에 대한 수행 환경을 나타내고 있다.

[표 2] 수행 환경

분류	CWS 시스템	Java Web Server	클라이언트
사용OS	Windows NT 4.0	Windows NT 4.0	Windows NT 4.0
프로세서	펜티엄 200	펜티엄 200	펜티엄 200
RAM	64M	64M	64M
DB			Access97

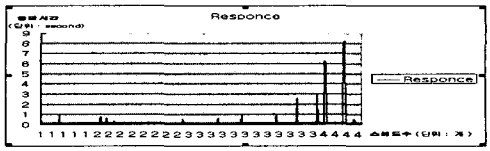
수행을 위한 구성 요소는 다음과 같다.

① 클라이언트 : 네트웍을 통해 연결된 다수의 클라이언트들은 임의의 시간에 Server에게 데이터를 요구한다.

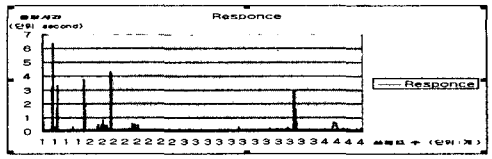
② 서버 : 클라이언트 요구에 대해 로컬에 저장된 HTML문서를 서버에게 전송한다.

다수의 클라이언트환경을 만들어서 테스트하기는 어려운 특성이 있기 때문에 본 성능평가를 위한 수행 환경에서는 이러한 상황을 가정하고, 시스템이 설정할 수 있는 클라이언트 수를 증가하면서 서버에서 데이터를 전송하는 시간을 데이터 베이스에 저장하였다.

그림 7과 그림 8은 클라이언트 수에 대한 응답시간을 나타내고 있다. 클라이언트 수가 1인 경우에는 Java Web Server의 응답지연시간이 짧은 것을 알 수가 있다. 클라이언트 수가 늘어날수록 응답지연시간이 그 만큼 길어지고 있지만, CWS 시스템의 경우 사용 클라이언트의 수가 늘어남에도 불구하고 응답시간이 짧아지는 것을 알 수 있다.



[그림 7] Java Web Server 응답시간



[그림 8] CWS 시스템 응답시간

다수의 사용자 접속에 대한 응답시간은 시스템 자원을 얼마나 많이 사용하고 있는 가를 나타낸다고 할 수 있으며, 따라서 CWS 시스템은 적은 자원을 사용하여 클라이언트에 대한 서비스를 하고 있음을 알 수 있다.

5. 결론

기존의 웹은 참조 무결성, 이동 투명성 그리고 유지·보수의 어려움과 같은 문제점들을 지니고 있으며, 응용 프로그램 개발을 위한 유연성과 확장성을 제공하지 못한다.

본 논문에서는 웹의 상기 문제점들을 보완하고 유연성과 확장성을 제공할 수 있는 CWS 시스템을 설계하고 구현하였다. 본 시스템을 이용할 경우, 객체지향을 기반으로 한 자원의 관리가 가능하게 되며, 웹 서버의 확장된 내부 모듈로 기존의 웹 문제점들을 개선하였다. 또한 인터페이스 환경을 향상시키고 응용 프로그램에 대한 광범위한 자원 참조가 가능하도록 하는 강력한 분산 객체 운영 처리가 가능하다.

향후 기존의 웹에 Proxy 서버 기능을 추가하여 좀 더 빠른 결과와 보안문제 해결을 위한 계속적인 연구가 필요하다.

[참고문헌]

[1] D. B. Ingham, M. C. Little. S. J. Caughey and S. K. Shrivastava, W3Objects : Bring Object-Oriented Technology to the Web, World Wide Web Joural, Issue 1, pp.89-105 : Proceeding of the Fourth International World Wide Web Conference, Boston, Mass., USA, 1995.

[2] D. B. Ingham, S. J. Caughey, and M. C. Little, Fixing the Broken-Link Problem : W3Objects Approach, Computing Network & IDSN System, Vol.28 No. 7-11, pp. 1255-1268 : Proceeding of the Fifth International World Wide Web Conference, Paris, France, 6-10 May 1996.

[3] Roy Fielding, "Hypertext Transfer Protocol HT TP/1.0", RFC 1945, IETE HTTP WG, May 1996.

[4] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, UC Irine, June 1995.