碩士學位論文

XML - RPC 기반의 분산 환경 문서관리 시스템

濟州大學校 大學院

컴퓨터工學科

高 赫 浚

2003年 12月

XML - RPC 기반의 분산 환경 문서관리 시스템

指導敎授 郭 鎬 榮

高 赫 浚

이 論文을 工學 碩士學位 論文으로 提出함

2003年 12月

高赫浚의 工學碩士學位 論文을 認准함

審査委員長 이 상 준 印

委 員 곽 호 영 印

委 員 변 영 철 印

濟州大學校 大學院

2003年 12月

DEDMS : Distributed Environment Document Management System Model based on the XML-RPC

Hyuck-Jun Ko

(Supervised by professor Ho-Young Kwak)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Engineering

Department of Computer Engineering

GRADUATE SCHOOL

목 차

[그림 목차]

[표 목차]

DEDMS : Distributed Environment Document Management System Model based on the XML-RPC

Hyuck-Jun Ko

Department of Computer Engineering
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY
Supervised by professor kwak Ho Young

Summery

Even the document resources offered from web server can be represented in the form of URL/URI, it can not necessarily be guaranteed that corresponding resources exist due to a dynamic change of server environment. In this paper, integrated document administration system is therefore proposed and modeled using the XML-RPC technology which guarantees the reliance of resources, and handles a dynamic server resource management and request of clients.

The proposed system is composed of middleware and server systems. The former system manages dynamic server resources, and the latter reports the updated information of documentations stored in server by client from the server to middleware system. As a result, effective storing management of dynamic resource in distributed server could be archived and building cost of a new web server could be reduced due to an applicability to current web server. In addition, platform independent and efficient data management was obtained by using the XML-RPC protocol.

I. 서 론

인터넷의 급격한 발전으로 WWW(World Wide Web)은 우리 생활 깊숙이 파고들어 많은 웹 기반 프로그램들을 개발하게 하였으며, 또한 웹 브라우저를 통해 임의의 플랫폼에서도 응용프로그램에 접근하여 정보공유와 경제활동을 하고 있는 비즈니스 부분으로 그 활용분야를 넓혀가고 있다. 그러나 현재의 비즈니스 관련 웹 개발환경은 서버/클라이언트 구조를 가지는 2-Tier환경에서 많은 사용자와 대용량의 데이터를 처리하기 위한 Business Logic과 Application Logic을 담당하는 미들웨어(Thin Server)를 갖춘 3-Tier환경으로 변해왔고, 현재 지속적인 서비스를 위한 가용성과 성능향상 그리고 용량을 초과하는 서비스들을 처리하기 위해서 확장성도 요구되어 n-tier환경으로 변해가고 있다[1][2].

이러한 흐름 속에서 미들웨어 시스템은 분산 환경의 자원들을 표현하고 통합 및 표준화하여 클라이언트들에게 서비스를 제공해 줄 수 있는 다양한 방식[3,4,5,6,7]으로 연구되고 있지만, 현재 분산 환경의 통신수단으로 활용되고 있는 MicroSoft사의 COM/DCOM, OMG사의 CORBA, Sun사의 RMI/JavaBeans 등은 서로 호환성을 가지고 있지 않으며, 또한 객체기반 분산 컴퓨팅 환경에서 클라이언트는 ORB(Object Request Broker) 소프트웨어에 의존하는 형식으로 서버와 데이터를 교환하고 있어서 상호운영성의 결여 등이 문제점으로 지적되고 있다[8,9].

따라서 현재 분산 환경의 서버 상에 존재하는 동적인 문서들을 효율적으로 통합 및 표준화하여 클라이언트들에게 제공하는데 어려움을 가지고 있으며, 이에 따라 분산된 문서들을 효과적으로 저장하고 관리하는 서비스의 필요성이 제기되고 있다.

이 중에서 분산된 웹 환경의 문서들에 대해 플랫폼에 독립적으로 특정한 공간에 저장 및 관리하고 효과적으로 공급하기 위해 특정 자료로부터 추출한 정보를 검색에 사용될 수 있는 형태로 가공한 정보, 즉, 메타정보(Meta-Information)들을 미들웨어(Middleware)를 통해 활용하는 연구가 대두되고 있다[10].

현재 XML[11]이라는 데이터 표준 데이터 포맷을 이용하여 각각의 웹 환경의 자원들을 플랫폼 독립적으로 전달하기 위하여 XML 프로토콜(Protocol)인 XML-RPC[12,13,14], SOAP[15], WDDX등이 활용되고 있으며, 응용프로그램간의 단순한 데이터 교환뿐만 아니라 그에 적합한 처리를 할 수 있는 응용프로그램까지 전달 할 수 있으며 상호운용성도 좋다[16].

본 논문에서는 다양한 XML 프로토콜(Protocol)[17]중에서 구조적 데이터를 표현할 수 있고, 분산 환경에서 통신수단으로 활용할 수 있는 XML-RPC 프로토콜 [18,19]을 사용하여 분산된 웹 서버에 있는 동적인 문서들의 정보를 관리하고 효과적으로 클라이언트들에게 제공하기 위한 문서 관리 시스템 모델을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 연구의 이론적 배경에 대해 알아보았으며, 3장에서는 제안하는 문서관리 시스템 모델에 대한 시스템 분석 및 설계를 설명하고, 4장에서는 시스템 구현 결과 및 고찰을 그리고 5장에서는 결론 및 향후 연구 방향을 제시한다.

Ⅱ. 연구의 이론적 배경

2.1 분산 시스템

분산 시스템 환경은 여러 개의 컴퓨터 환경과 통신망 속에서 시스템과 응용 프로그램들이 분산되어 제반 자원의 공유와 제어를 행하는 시스템 형태로 널리 이용되고 있다. 그러나 이러한 분산 시스템의 환경에서는 자원의 공유나 제어를 통한 시스템의 성능을 향상시키는 것에 반하여 분산된 중복 데이터 처리에 따라 통신 처리의 오버헤드 문제 해결, 효율적인 메커니즘의 활용이 주요한 문제점으로 대두되고 있다. 이러한 문제점을 해결하기 위하여 원격 프로시저 호출(RPC : Remote Procedure Call)이 제시되었는데 RPC는 분산처리 시스템과 네트워크 사이를 프로세스간의 통신 메커니즘 화하여 통신망의 과부하를 줄이고 단순성(Simplicity), 투명성(Transparency)의 특성을 갖는 분산 시스템 개발에 광범위하게 이용되고 있다. RPC 에서는 클라이언트에서 서버 프로세서에 대한 서비스 요구를 마치 클라이언트에서 일반적인 호출(Call)프로세서와 같이 동시적인 개념으로 구성되는 특성을 갖고 있다.

오늘날 대형 기종에서 처리하던 데이터베이스 관리, 트랜잭션 관리, 이미지 처리와 같은 고속의 연산이 요구되는 작업들이 하드웨어 기술의 발전과 네트워크의 고속화로 분산 처리가 가능하게 되었다.

이들 중 클라이언트/서버 모델은 분산 처리의 일반화 모델로서, 서비스를 제공하는 원격 서버와 서비스를 요구하는 클라이언트로 구성된다. 클라이언트/서버 모델은 이미 X Window System, NFS(Network File System), NIS(Net Work File System), NIS(Network Information Service) 그리고 상용화된 데이터베이스 관리 시스템인 MySQL, Postgres, Oracle,

Sybase, Microsoft ODBC 등 다양한 분야에서 사용되고 있다.

이런 경향에 따라 하나의 기기에서 수행되도록 중앙집중식으로 개발되었던 여러 프로그램이 클라이언트/서버 구조의 분산 프로그램으로 다시 설계, 개발되었고, 또한 여러 새로운 프로그램들이 클라이언트/서버 프로그램의 형태로 개발되어지고 있다.

이러한 클라이언트/서버 형태의 프로그램 작성은 아직까지 중앙집중식 프로그램에 비해 어려운 것으로 여겨진다. 지금까지 소켓(Socket)과 같은 트랜스포트 레벨의 통신 API의 사용은 중앙집중식 프로그램 작성자에게 익숙하지 않고, 또한 이러 통신의 세부적인 내용을 프로그램 작성자가 정확히 알고 다루어야 한다는 것은 큰 어려움이 되어 클라이언트/서버 프로그램 개발의 생산성을 떨어뜨리게 된다.

이런 문제를 해결하기 위해 RPC에서는 기존의 프로그래밍 언어를 클라이언트/서버 프로그램의 작성이 중앙집중식 프로그램의 작성과 거의 동일한 느낌을 줄 수 있도록 한다.

하지만, 분산기술은 웹과의 호환성을 직접 가지지 못하므로 웹을 통해 데이터 교환에 사용하려는 애플리케이션에서는 별도의 프로토콜을 지원하도록 요구받는다. 이러한 컴포넌트 기술과 웹과의 호환성 문제를 해결하기 위해서 XML을 이용하고자 하는 많은 노력이 있어 왔으며, XML은 기존의 복잡했던 분산 환경을 상당 부분 단순화시키는데 큰 역할을 수행할 수 있다.

2.2 RPC (Remote Procedure Call)

RPC(Remote Procedure Call)는 분산형 컴퓨팅의 클라이언트 서버 모형을 수행하기 위하여 가장 많이 쓰이는 패러다임이다.

RPC는 한 프로그램이 네트워크상의 다른 컴퓨터에 위치하고 있는 프로그

램에 서비스를 요청하는 경우에 사용되는 프로토콜로서, 이때 서비스를 요청하는 프로그램은 네트워크에 대한 상세 내용을 알 필요가 없다. RPC는 클라이언트/서버 모델을 사용하는데 다른 형태의 자체적인 프로시저의 호출과 마찬가지로, RPC도 요청하는 프로그램의 원격 절차의 처리 결과가 반환될 때까지 일시정지 되어야 하는 동기 운영형태를 가진다.

그러나 가벼운 프로세스의 사용이나, 같은 주소공간을 공유하는 스래드 등은 여러 개의 RPC들이 동시에 수행될 수 있도록 허용한다. RPC를 사용하는 프로그램 문장들이 실행 프로그램으로 컴파일 될 때, 컴파일 된 코드 내에 RPC의 대리인처럼 동작하는 스터브(Stub)가 포함된다. 이러한 스터브 프로시저는 원격 프로시저가 클라이언트에 의해 호출될 때 생성하게 되며 로컬 프로시저를 원격 프로시저 콜로 서버 측에 변환해 주는 것이 주목적으로 다음과 같은 일련의 작업을 수행하게 된다.

먼저 파라메타들은 마샬링(Marshalling)해서 프로시저 식별자로 묶어 전송 메시지에 싣게 되며 마샬링(Marshalling)한 전송 메시지를 서버에 보내고 응답을 기다리게 된다. 그 후 그림과 같이 결과로 받은 메시지를 언마셜링(Unmar-shalling)하게 되는 역할을 수행한다. 즉, 스터브 프로시저는 파라메타 및 실행 결과를 메시지 형태로 정리하는 역할을 수행한다. 프로그램이 실행되어, 절차 호출이 이루어질 때, 스터브는 그 요구를 받아서 그것을 로컬 컴퓨터 내에 있는 클라이언트 런타임 프로그램에게 전달한다.

클라이언트 런타임 프로그램은 원격 컴퓨터와 서버 프로그램과 어떻게 접촉해야 하는지를 인지하고 있으므로, 네트워크를 통해 원격처리를 요구하는 메시지를 보낸다. 이와 유사하게 서버는 런타임 프로그램과 원격절차 그 자신과 인터페이스를 하는 스터브를 포함한다. 처리 결과들은 같은 방식으로 되돌려 진다.

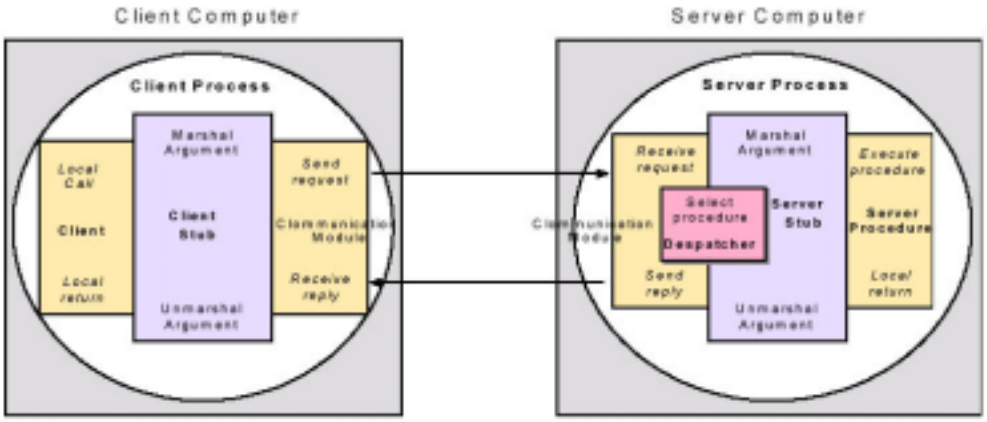

그림 1 일반적인 RPC 호출 메카니즘

2.3 CORBA 와 XML

최근 분산 컴포넌트 기술인 OMG(Object Management Group)의 CORBA(Common Object Request Broker Architecture)와 웹과의 연동을 위해 SOAP(Simple Object Access Protocol)을 이용한 SCOAP(Simple Corba Object Access Protocol)이 제안 되고 있는 상태이다. SOAP이라는 프로토콜은 Microsoft사에 의해 제안되었으며, 플랫폼 독립적으로 인터넷에 분산된 서비스, 오브젝트, 또는 서버에 접근할 수 있도록 HTTP 혹은 SMTP상에서 단순히 서비스 요청/응답에 대하여 표준 XML 문서 구조를 정의하여 서비스 제공업자와 사용자 간에 상호 교환하도록 구성되어 있다.

서비스를 요청하는 문서에는 사용할 수 서비스의 정보와 입력 값을 채워서 보내주면 서비스 제공업체는 이를 분석하여 해당 서비스를 수행한 후 그 결과 값을 응답 문서에 넣어서 돌려주는 방식이다. 현재 SOAP의 표준화가 많이 진행되고 있는 상태이지만 복잡한 구조를 보이고 있다.

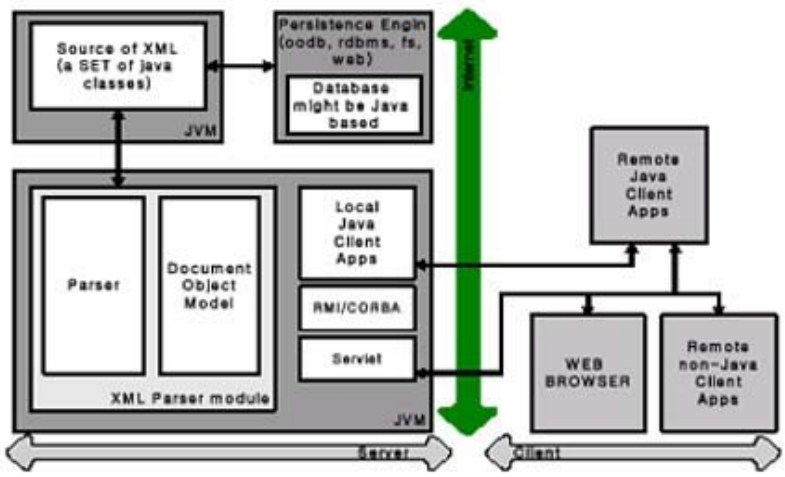

그림 2. Web API를 이용한 CORBA 와 XML의 연동구조

그림 2는 JVM(Java virtual machine)기반의 Web API 를 이용한 경우의 CORBA 와 XML 의 연동구조를 나타내고 있다.

이 경우 XML 과 CORBA 의 연동에서 XML 문서의 처리방법은 CORBA 클라이언트와 CORBA 서버 측에서 XML 문서를 직렬화 형태로 전송하여 CORBA 클라이언트와 서버 측에서 이 XML 문서를 처리하는 방법과, CORBA 클라이언트가 CORBA 서버 측에 요청하여 받은 응답을 XML로 변형하는 방법이 있다.

그러나 이 구조는 기존의 CORBA 환경에서 XML 문서를 처리해야 하기 때문에 CORBA 클라이언트객체와, CORBA 서버객체, IDL 등을 수정해야 만 하는 문제점이 있다.

메타데이터에는 XMI(XML Metadata Interchange), W3C의 RDF(Resource Descript ion Framework), OMG 의 MOF(Meta Object Facility), SMIF (Stream - based Model Interchange Format)등이 있다. UML 기반의 메타모델을 정의하여 분산 환경에서 메타데이터의 교환 및 전송을 위한 XML 형식의 메타데이터를 생성하여 데이터베이스에 저장한다. 이 XML

형식의 메타데이터를 이용하는 것은 인터넷상에서 자료전달 과 교환 시 이진(Binary) 양식이 아닌 텍스트 양식으로 전달이 가능하므로 플랫폼에 따라 발생할 수 있는 호환성 문제를 해결할 수 있기 때문이다. 그러나 이 구조는 서로 다른 메타데이터들 간의 호환성에 문제가 제기 되고 있다. 그림 3은 메타데이터의 관리 구조에 대하여 설명한 그림이다.

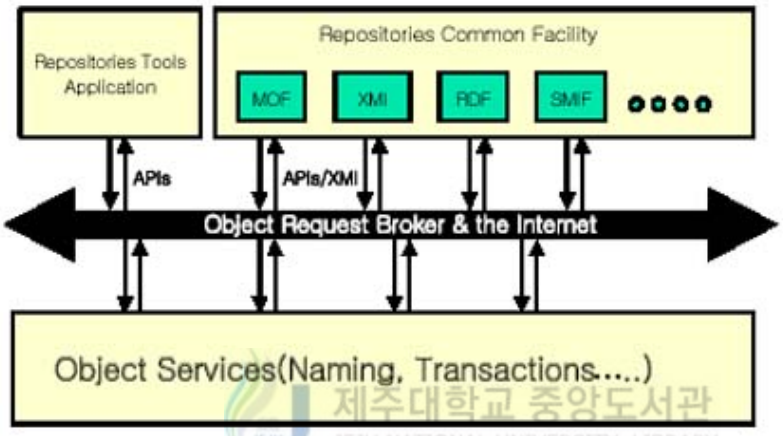

그림 3. MetaData를 이용한 CORBA와 XML의 연동구조

2.4 WDDX (Web Distributed Data eXchange)

WDDX(Web Distributed Data eXchange)는 프로그래밍언어 사이에서의 복잡한 데이터 구조를 교환하기 위하여 고안된 XML 프로토콜의 한 종류로서 XML을 이용하여 데이터 직렬화(Serialization)를 통하여 데이터를 전달한다. 직렬화된 메시지는 데이터에 대한 타입 정보와 변수이름, 변수 값을 포함한다.

그러나 WDDX는 데이터 직렬화에 대한 방법만 정의되어 있고, Remote 메소드 호출의 기능이 없으며, XP(XML Protocol)바인딩이 HTTP로 한정되어 있고, 지원하는 데이터 타입 또한 제한되어 있다.

2.4 XML-RPC

웹 서비스는 현존하는 웹 기반 구조에서 가장 상위에 존재하는 분산 애플리케이션을 개발하도록 여러 가지 도구 모음을 제공한다. 이러한 애플리케이션은 웹을 일종의 '전송 계층'으로 사용하지만, 브라우저를 통한 사람과의 직접적인 인터페이스는 제공하지 않는다. 웹 기반 구조를 재사용하면 이러한 애플리케이션을 설치하는데 소비되는 비용을 매우 많이 줄일 수 있으며, 본래 웹을 위해서 만들어진 모든 종류의 도구를 재사용할 수 있게 된다.

XML-RPC는 가장 단순한(그리고 가장 간단한) 웹 서비스 접근 방법 중 하나이며, 컴퓨터가 다른 컴퓨터에 있는 프로시저를 쉽게 호출할 수 있도록 해준다. XML-RPC는 컴퓨터에 있는 각종 프로그램간의 통신을 지원하는 데 본래 사람들 사이에 사용하는 통신 수단으로 개발되었던 기반 구조를 재사용한다. 확장 마크업 언어(XML, Extensible Markup Language)는 RPC(Remote Procedure Call)를 기술하는 어휘를 제공하며, 이것은 컴퓨터들 사이에서HTTP를 이용하여 전달된다. XML-RPC는 개발과정을 매우 간단하게 해주며, 서로 다른 종류의 컴퓨터들이 쉽게 통신할 수 있도록 해준다. 컴퓨터간의 통신에 초점을 맞추게 되면 개발자는 구시대적 웹 개발 특성이라 말할 수 있는, 사람이 읽을 수 있는 컨텐츠에만 초점을 맞추는 과오에 빠지지 않으면서 웹 기술을 사용할 수 있다.

가장 기본적인 레벨로는XML-RPC를 통해서 네트워크를 오고 가는'함수 호출'을 만들 수 있다. RPC 아키텍처를XML과 HTTP 기술에 혼합한 XML-RPC를 이용하여 컴퓨터들이 네트워크에 있는 자원을 쉽게 공유하도록 할 수 있다. 이는 새로운 것이 진행되는 중에 이미 만들어 놓은 시스템을 단순히 읽고 재사용하는 것이 아니라, 최선의 효과를 낼 수 있도록 프로그램들을 서로혼합하고 재배치하는 것을 의미하는 것으로서, 사용

자들이 원하는 정보에 직접 접근하도록 할 수 있다는 것을 말한다.

<표.1> XML-RPC 기본 자료형

자료형	타입 설명
<i4>또는<int>	4바이트 정수
<boolean>	0과 1
string	아스키 문자열
double	배정도 실수
<dateTime.iso8601>	ISO 8601식의 날짜 시간 표기(19980717T14:08:55)
<base64>	base64로 인코딩된 문자열
<struct>	키와 값으로 구성된 자료형
<array>	자유로운 배열, 재귀적일 수 있는 모든 스칼라 변수포함

웹 프로토콜과 기반 구조 재사용하기 위해 XML-RPC는 웹의 또 다른 중요 요소인 전송 프로토콜을 재사용한다. HTTP 프로토콜은 웹 서버에 적합한 환경에서부터 프로그램 내부에서 직접 사용하는 마이크로서버까지 많은 개발 환경에서 만들어졌다. 개발자들은 HTTP 전송을 하기 위해 문서를 조합하는 과정을 수행해왔으며, 네트워크 관리자들은 웹 서버와 웹 환경의 방화벽 등을 수년 동안 지원해왔다.

호출된 메소드의 식별자를 열어서 매개변수를 제공하고, 그 메소드가 어떤 값을 리턴해야 할지 결정하는 것 등 여러 가지 측면을 볼 때 HTTP는 RPC 기반의 프로토콜이다. HTTP는 서로 다른 종류의 컨텐츠를 일치시키고 인코딩하는 표준 규약인MIME(Multipurpose Internet Mail Extensions)에 기초한, 비교적 개방적인 접근 방법이다. 따라서 웹 사이트에 필요한 많은 종류의 컨텐츠를 유연하게 다룰 수 있다. 이러한 유연성은RPC 프로토콜 요청에 따른 다양한 응답 방법들을 수행하기에 충분하다.

XML-RPC의 반쪽이 RPC로부터 파생되었지만, 다른 반쪽은 월드 와이드 웹에서 나왔다. 지난10년간 웹의 성장은 폭발적이었고, 과학적인 호기심에서 흔히 존재하는 소비자의 도구로써 빠르게 변화해왔다. 웹은 개발자들이 개발하기 쉽고, 일반인들이 처리하기에 충분할 정도로 간단한 인터페이스를 제공한다. 최초에는 웹이 인간 사이에서 통신하기 위한 도구였지만, 현재에 이르기까지 인간과 컴퓨터간의 상호 작용을 위한 복잡한 인터페이스로 변화해왔다. 또한 컴퓨터간의 복잡한 통신 수단으로 빠르게 변화하고 있다.

HTML은 분명 대단히 성공적이었지만, 이는 사람들에게 정보를 제공하는 데 유용할 뿐이었다. HTML의 한계가 드러나면서, HTML 규격을 보존하고 있는 월드 와이드 웹 컨소시엄(W3C, World Wide Web Consortium)은 XML을 사용할 것을 주장했다. 이 언어는 HTML과 동일한 환경에 적용되면서 프로그램들이 통신할 때 훨씬 더 유연하도록 해준다. 개발자는XML을 이용해서HTML보다 정교하게 기술되는 컨텐츠를 가진 문서를 생성할 수 있다. 즉, XML을 이용해서 단순히 표현하는 수준이 아닌 컴퓨터가 해석할 수 있는 메시지를 생성할 수 있다는 의미이다.

XML을 통해 책의 카탈로그 정보에 사용될 수 있는 <title>과 <author>와 같은 여러 개의 태그 집합들을 해당 데이터에 적합하게 새로 생성할 수 있다. XML-RPC는 마크업 프로시저 호출에 사용하는 고유의 태그 집합을 사용한다. XML이 HTML을 전송하는 프레임워크에 적합하도록 개발되었기 때문에, XML-RPC를 포함하는 웹을 위한 새로운 가능성을 창조해왔다. 그림 4는 XML-RPC의 기본적인 전송방식을 나타내고 있다. XML-RPC는 기존의 HTTP 프로토콜과 기반 구조의 장점을 유지하면서 앞에서 설명한 RPC 접근 방법을 실제로 구현할 수 있도록 한다. HTTP는 모든 종류의 개발환경과 운영체제에서 동작할 수 있고, XML

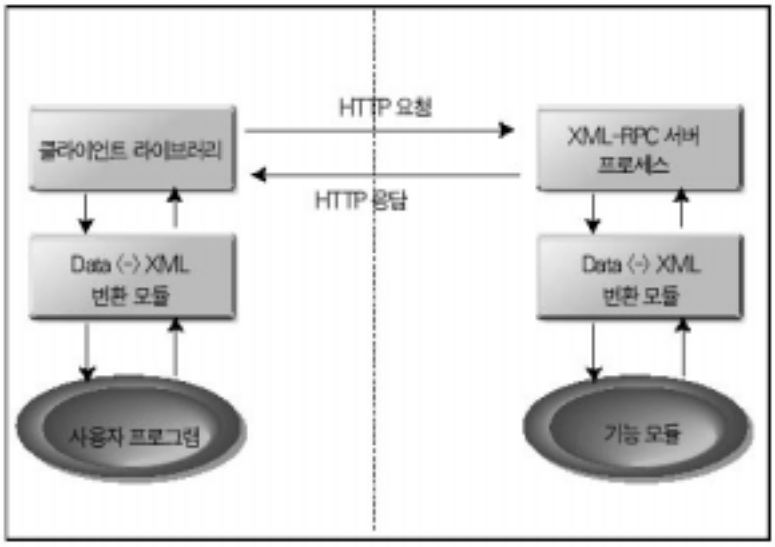

그림 4. XML-RPC 전송방식

파서는 필수품과도 같기 때문에 어떠한 환경에서도XML-RPC 툴킷을 사용해서 시스템을 구성할 수 있다. 대부분의 웹 애플리케이션은 사람들에게 정보를 제공하려는 목적으로 만들어진다. 반면에 웹은 XML-RPC와 웹 서비스를 이용하여 컴퓨터들이 서로 밀접하게 연결된 통로를 통해 정보를 주고받는 프로시저 연결의 모임이다. 하이퍼텍스트 링크를 통해 사람이 직접 찾아다닐 필요 없이, 컴퓨터가 정보를 교환할 때 사전에 잘 정의된 규칙을 따른다. 이러한 정보 교환은 웹으로 구축된 클라이언트-서버 모델을 따를 필요가 없다. XML-RPC는 클라이언트-서버 형식뿐만 아니라 P2P도 지원하기 때문에, 브라우저에서 서버로 정보를 전달하는 데 일반적인 웹 브라우저보다 훨씬 더 많은 HTTP 기반의 장점을 가진다.

Ⅲ. 시스템 분석 및 설계

3.1 제안한 전체 시스템 모델 구조

본 논문에서 제안하는 시스템 모델은 각각의 분산된 서버에 존재하거나 등록할 문서에 대해 효율성과 신뢰성을 보장하며 클라이언트들에게 정보를 제공하기 위하여 문서정보의 등록/검색/삭제 등의 이벤트들을 미들웨어와 XML-RPC 프로토콜을 도입하여 문서 정보에 대하여 웹 애플리케이션을 통해 제공 및 저장하고 메타정보들을 데이터베이스에 저장하기 위한 시스템을 모델링 한다.

클라이언트들은 분산되어 있는 서버에 상관없이 웹 브라우저를 통해 문서에 대한 메타정보를 제공 및 등록하는 웹 애플리케이션 서버에 접근하여 자신이 원하는 문서 정보들에 대하여 등록/검색/삭제 등의 작업을 하게 되고, 그에 대한 결과 값을 데이터베이스에 저장하여 신뢰할 수 있는 메타 정보를 미들웨어를 통해 제공받는다. 제공받은 정보를 가지고 클라이언트들은 직접 문서가 저장된 분산된 서버로 접근하여 문서를 얻어 올 수 있게 된다.

웹 애플리케이션 서버와 미들웨어 그리고 데이터베이스는 각각 분리되어 있으며, 각각의 시스템들은 XML-RPC 프로토콜을 이용해 통신을 하게 된다.

웹 애플리케이션 서버와 분산된 서버에서 이벤트가 발생할 때마다 미들웨어는 이벤트 프로세스를 통해 마샬링된 Documents( )객체를 전송받아 언마샬링하여 관계형 데이터베이스 매핑 과정을 거쳐 문서정보들을 저장하게 된다.

저장된 메타정보는 클라이언트들의 문서에 대한 관리에 사용되게 되며,

제안하는 시스템은 크게 웹 애플리케이션 서버, 미들웨어 시스템, 분산 서버 시스템으로 구분되며, 전체 시스템 모델 구조는 그림5와 같고, 등록/검색/삭제/수정에 대한 이벤트 발생에 대한 처리 흐름에 대해 기술한다.

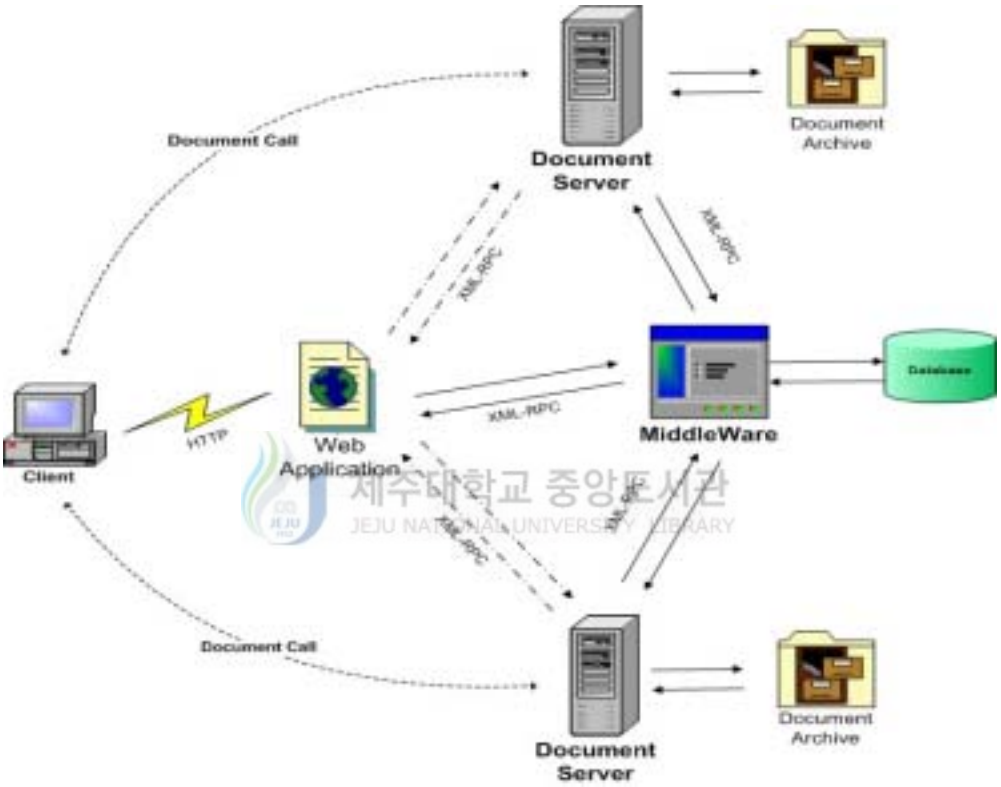

그림 5. 제안한 전체 시스템 모델 구조.

그림5 에서 보듯이 클라이언트들은 각각의 분산된 서버에 대해서 알 필요가 없게 됨으로서 위치 투명성이 보장된다. 기존의 웹 애플리케이션 시스템에서 자료를 처리하던 비즈니스 로직을 미들웨어로 분리하고 오직 보여주는 프리젠테이션 부분에 집중할 수 있게 되어 프리젠테이션 부분과 비즈니스로직 부분에 대한 처리에 유연성이 생겨 추가적으로 웹 애플리케이션 서버와 미들웨어의 확장 가능성도 유연하다.

그림6은 문서 등록과정에 대한 데이터 흐름을 나타내며 클라이언트의 요청에 대한 반응을 설명한다.

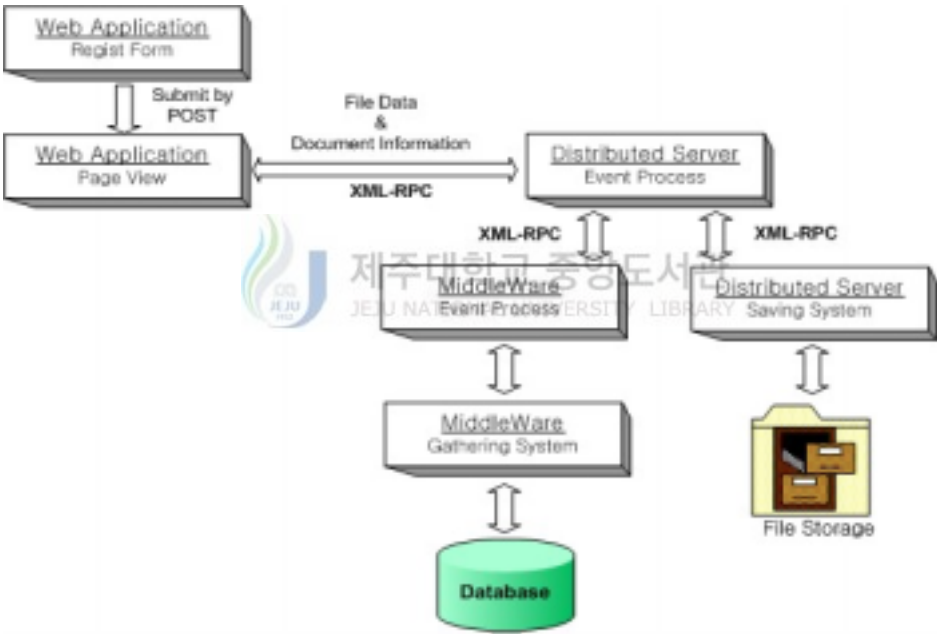

그림 6. 문서 등록 흐름

클라이언트는 웹 애플리케이션 서버로 접근하여 등록 폼을 통해 선택한 서버와 문서 그리고 저자에 대한 값을 입력하고 등록하게 되면, XML-RPC를 통해 해당 분산 서버로 문서 정보들과 파일이 전송되고, 파일은 분산 서버 특정 디렉토리에 저장 되면, 등록된 문서 정보들은 다시 미들웨어로 XML-RPC를 통해 전송되어 이벤트 프로세스에 의해

Gathering 시스템으로 넘겨주고 데이터베이스 매핑 과정을 통해 등록 정보가 저장되게 된다.

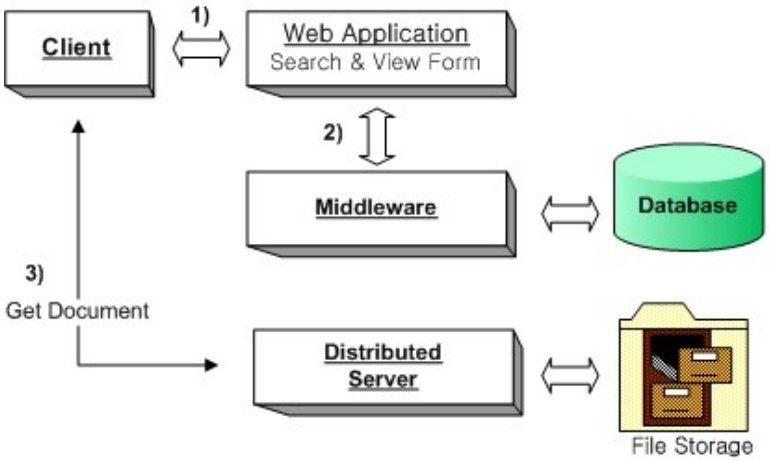

그림 7. 문서 검색 흐름 [순서 1) - 2) - 3)]

그림7 에서 보듯이 웹 애플리케이션 서버는 클라이언트가 검색하는 정보를 미들웨어에 요청하고 결과 값만 보여주기 위해 가공하여 클라이언트에게 되돌려 주고, 데이터베이스에 접근하여 문서에 대한 메타정보와 위치 정보를 얻어내는 일은 미들웨어가 맡는다. 이 때, 각 로직 사이에 통신은 최대한 간결하게 이루어지고 각 로직이 맡은 일에만 집중할 수 있도록 도와주기 때문에 가용성을 높일 수 있다.

이러한 과정을 거쳐 최종적으로 클라이언트가 어느 서버에 접근을 해야 원하는 문서를 얻을 수 있는지를 알 수 있게 되는데, 웹 애플리케이션의 구현 정도에 따라 서버에 대한 정보도 클라이언트가 알 필요 없도록 가려 온전한 투명성을 확보할 수도 있다.

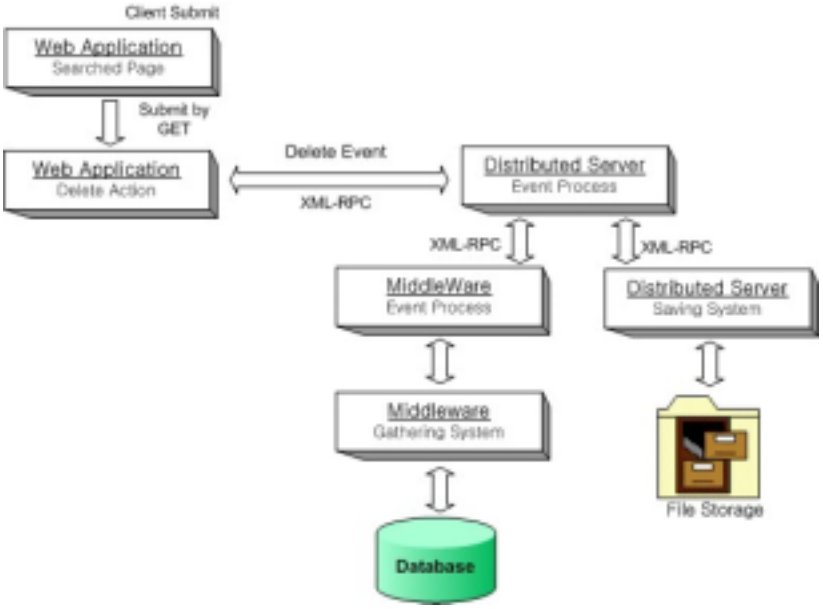

그림 8. 문서 삭제 흐름

그림 8에서 보면 문서를 삭제하기 위해서 클라이언트는 삭제할 문서를 검색된 페이지를 통해 얻게 되고 삭제를 웹 애플리케이션을 통해 요청하게 되면 미들웨어는 삭제에 대한 이벤트를 받아 분산된 서버와 Gathering 시스템에게 알리게 된다.

실질적인 문서파일은 분산된 서버에 저장된 File Storage에서 삭제되고, 동시에 데이터베이스에 저장된 메타 정보가 삭제되어, 신뢰 할 수 있는 정보를 유지하게 된다.

그림 9는 문서 수정에 대한 흐름을 보여주고 있다. 그림6의 문서 등록 과정을 수행하는 것을 Registry Module로, 그림 8의 문서 삭제 과정을 수행하는 것을 Delete 모듈로 놓는다면 문서 갱신과정은 이 두 과정을 이어서 진행함으로서, 문서 갱신과정에 추가되는 여러 과정을 생략할 수 있다.

그림 9 문서 수정 흐름

3.2 미들웨어와 분산된 서버와의 연동구조

웹 애플리케이션 서버는 등록/검색/삭제/수정 등의 작업에 대한 이벤트 발생시 XML-RPC를 통해 분산되어 있는 서버로 문서정보 및 파일을 보내게 되면 파일은 각각의 지정된 서버에 저장이 되고, 문서에 대한 정보는 미들웨어와 분산된 서버와의 XML-RPC 통신을 통해 Document객체로 날려주면, 미들웨어는 그 정보를 데이터베이스 매핑 시스템을 통해 데이터베이스에 저장하고 클라이언트의 문서정보 요청 시 웹 애플리케이션 서버에 결과를 보내준다.

미들웨어를 통해 모든 문서 정보가 수합되고 데이터베이스에 저장이 되며, 각각의 분산 서버들은 이벤트 처리에 대한 결과 값을 실시간 미들웨어로 전달하게 된다. 이러한 과정을 통해 클라이언트는 항상 신뢰할 수 있는 실시간 문서변환 정보 및 파일을 핸들링 할 수 있게 된다.
그림10은 미들웨어와 분산된 서버와의 연동구조를 나타낸다.

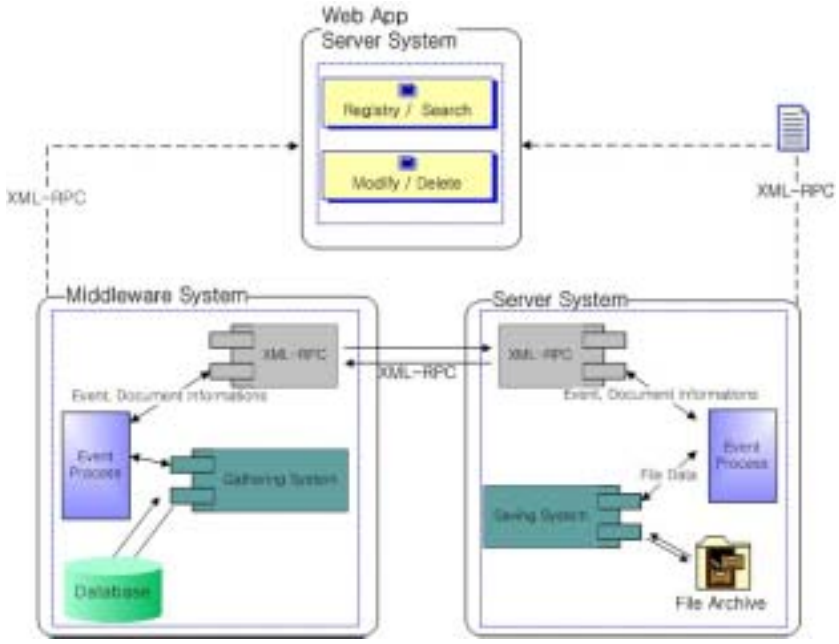

그림 10. 미들웨어와 분산된 서버와의 연동구조

3.3 웹 애플리케이션 서버시스템

웹 애플리케이션 서버는 클라이언트가 익숙한 웹 인터페이스를 제공한다. 클라이언트들은 제공되는 서버들의 목록을 지정하고 저자/문서에 대한 간단한 값을 입력하고 문서파일과 문서정보를 저장할 수 있는 폼과 문서를 검색하는 폼을 제공한다.

웹 애플리케이션 서버시스템은 모든 비즈니스 로직 부분은 미들웨어에 위임하고, 프리젠테이션 부분만 관여하기 때문에 시스템이 유연하고, 미들웨어에 대한 정보 값만 알고 있으면 되므로 미들웨어 확장이나 웹 애플리케이션 확장의 편이성도 제공 받는다. 따라서 다양한 웹 기반의 서비스와 맞물려 웹 기반의 문서관리 시스템을 클라이언트에게 제공하기 쉽고, 운영 중인 서버에도 바로 연동 할 수 있어 보다 효율적이다.

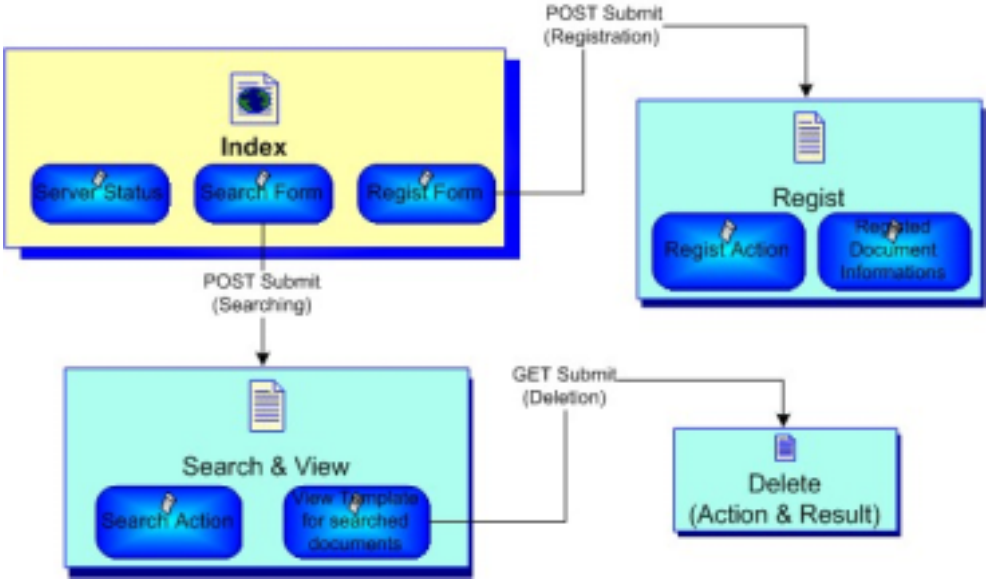

그림 11. 웹 애플리케이션 이벤트 흐름

3.4 Middleware 시스템

미들웨어는 웹 애플리케이션 서버에 의해 실시간으로 발생하는 문서 이벤트들에 대한 정보들을 각각의 분산된 서버로 XML-RPC 프로토콜을 통해 이벤트 프로세스로 전달하게 되고, 이벤트 처리 결과에 대해 미들웨어 시스템으로 실시간으로 통보받게 된다. 처리된 이벤트들을 Gathering System으로 넘겨주고 관계형 데이터베이스 맞게 매핑하여 저장하게 된다.

미들웨어 시스템은 분산되어 있는 서버와 웹 애플리케이션 서버에 대한 정보들을 Config파일로 가지고 있으면서 실시간으로 변하는 이벤트들을 처리하게 된다.

미들웨어 시스템의 가장 큰 역할은 문서 정보들을 수집하고 저장하여 클라이언트에게 신뢰 할 수 있는 메타정보를 제공하기 위한 비즈니스 로직이 구현되는 부분이다.

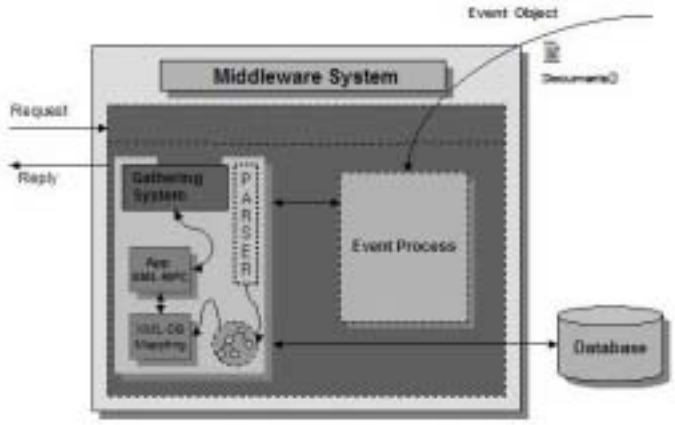

그림 12. 미들웨어 시스템 전체 구조

3.4.1 이벤트 처리 시스템

동적으로 변화하는 문서 정보를 유지하기 위한 수단으로 분산된 서버 시스템에서 응답한 이벤트 처리 결과를 받아서 Gathering 시스템으로 넘겨주는 처리를 하게 된다. Document객체를 전달받아 Parsing 과정을 거치고, 언마샬링하여 객체 정보에서 문서정보를 수집한다.

클라이언트들이 웹 애플리케이션을 통해 문서정보인 메타정보들을 검색할 때, 데이터베이스에서 정보들을 불러와 제공하는 역할도 하게 된다.

3.4.2 Gathering 시스템

이벤트 프로세스에 의해 넘겨받은 문서정보들을 수집하여 데이터베이스에 매핑 시키는 역할을 하는 시스템이다. 또한, 웹 애플리케이션에서 검색에 대한 이벤트들을 처리하여 결과 값을 이벤트 프로세스에게 리턴해 주는 역할을 한다. 즉, 데이터베이스에 문서 정보를 Insert, Delete, Find 하는 역할을 처리하게 된다.

데이터베이스의 주된 역할은 문서정보에 대한 Select정보이기 때문에 본

논문에서는 관계형 데이터베이스를 사용하였다.

표2는 데이터베이스의 테이블 구조를 나타내며, Field 각각의 내용을 포함한다.

<표.2>데이터베이스 Table구조

Filed	Type	Null	Key	Default	Extra
id	int / unsigned		Primary	not null	auto_increment
name	varchar			not null	
server	varchar			not null	
author	varchar	yes			
mimetype	varchar			not null	
regdate	double	yes			

-id : DEDMS table의 문서정보에 대한 index를 지원하며, Primary key.

-name : 문서 이름에 대한 값.

-server : 등록된 서버에 대한 이름값.

-author : 문서 저자에 대한 값.

-mimetype : 문서형식에 대한 값.

-regdate : 문서 등록 / 수정 / 삭제 등에 대한 시간 값.

그림13은 이벤트 발생 시 XML-RPC를 통해 이벤트 프로세스가 넘겨받는 객체의 XML 파일의 구조이다.

```
<params>
  <param>
    <value><array><data><value><struct>
              <member>
                <name>mimetype</name>
                <value><string>image/jpeg</string></value>
              </member>
              <member>
                <name>author</name><value><string>jjoole</string></value>
              </member>
              <member>
                <name>name</name><value><string>CIMG0553.JPG</string>
                </value>
              </member>
              <member>
                <name>regdate</name><value>
                  <double>1067792581.84</double>
                </value>
              </member>
              <member>
                <name>server</name><value><string>jerimo</string></value>
              </member>
            </struct></value></data></array></value>
  </param>
</params>
```


그림 13. Documents( )객체 XML파일의 구조

3.5 분산된 서버 시스템

각각의 분산된 서버들은 미들웨어에 대한 정보를 가지고 XML-RPC 프로토콜을 통해 통신을 하게 된다. 웹 애플리케이션 서버로부터 문서정보와 파일을 받아 저장/검색/삭제/수정 등의 이벤트 처리를 하고 미들웨어로 결과에 대한 정보를 보내주는 시스템으로 XML-RPC Server Module, XML-RPC Client Module, 이벤트 프로세스, Saving 시스템으로 구성된다.

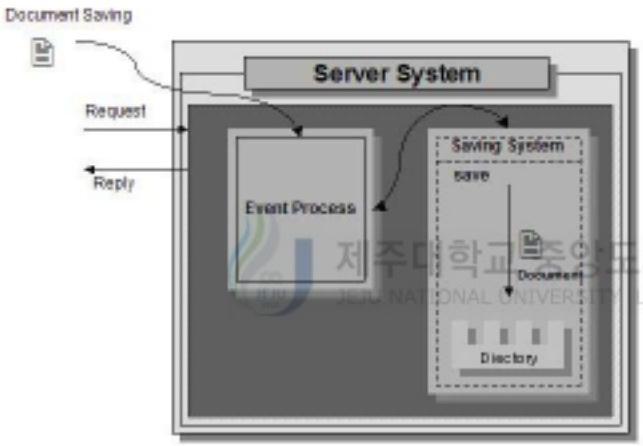

그림 14. 서버 시스템 전체 구조

3.5.1 이벤트 처리 시스템

웹 애플리케이션에서 발생한 저장/삭제/수정 등의 이벤트들을 처리하는 시스템으로 미들웨어에서 전달받은 Documents( ) 객체 정보들과 서버 상태, 문서에 대한 처리 상태 등을 Saving 시스템과 미들웨어로 보내고 받는 처리를 하게 된다.

현재 서버의 상태 및 갱신 정보들을 실시간으로 알려주는 역할을 하게 되며, 미들웨어의 이벤트 프로세스와 마찬가지로 XML-RPC를 통해 통신하

고 미들웨어에 대한 정보를 가지고 있기 때문에 플랫폼 독립적여서 확장이 용이하다.

3.5.2 Saving 시스템

이벤트 프로세스를 통해 전달 받은 문서(파일)을 지정된 공간에 저장하는 시스템이며, 처리 결과에 대하여 이벤트 프로세스를 통해 미들웨어로 통보하며, 문서 삭제 /수정의 요청들을 처리한다.

Ⅳ. 시스템 구현 결과

4.1 시스템 개발 환경

본 논문에서 제한 시스템 구현 환경은 Linux기반의 팬티엄Ⅳ 2.0GHz 256RAM의 시스템과 Linux기반의 팬티엄Ⅱ 300MHz 96RAM을 이용하였고, 웹 서버로는 오픈 소스인 아파치를 사용하였으며, 구현에 필요한 패키지는 PHP 4.1, Python2.3, MySQL Server, MySQL-Client(mysql-python), XML-RPC Library와 인터넷 브라우저인 익스플로어 환경에서 개발되었다.

4.2 시스템 구현 결과

4.2.1 웹 애플리케이션 시스템

웹 애플리케이션 시스템은 HTTPD Server 기반의 CGI와 PHP를 가지고 구현되었으며, 사용자를 위한 UI(User Interface) 부분을 담당한다.
UI는 크게 Documents에 대한 검색(Search) 부분과 등록(Registration)부분 그리고 삭제(Delete)부분으로 나누어지며, 등록과 검색 시 분산된 서버를 선택하는 형식으로 구현하였다. 문서 등록 과정에서는 선택된 서버로 문서(파일)와 문서 정보가 이벤트 객체로 XML-RPC를 통해 통신하게 된다. 웹 애플리케이션 서버는 분산된 서버의 현재 상태(Status)를 나타내주게 되므로, 클라이언트는 서버의 상태를 눈으로 확인 할 수 있게 된다.
그림 15는 웹 애플리케이션 서버의 등록/검색/삭제 등의 메인화면을 나타낸다.

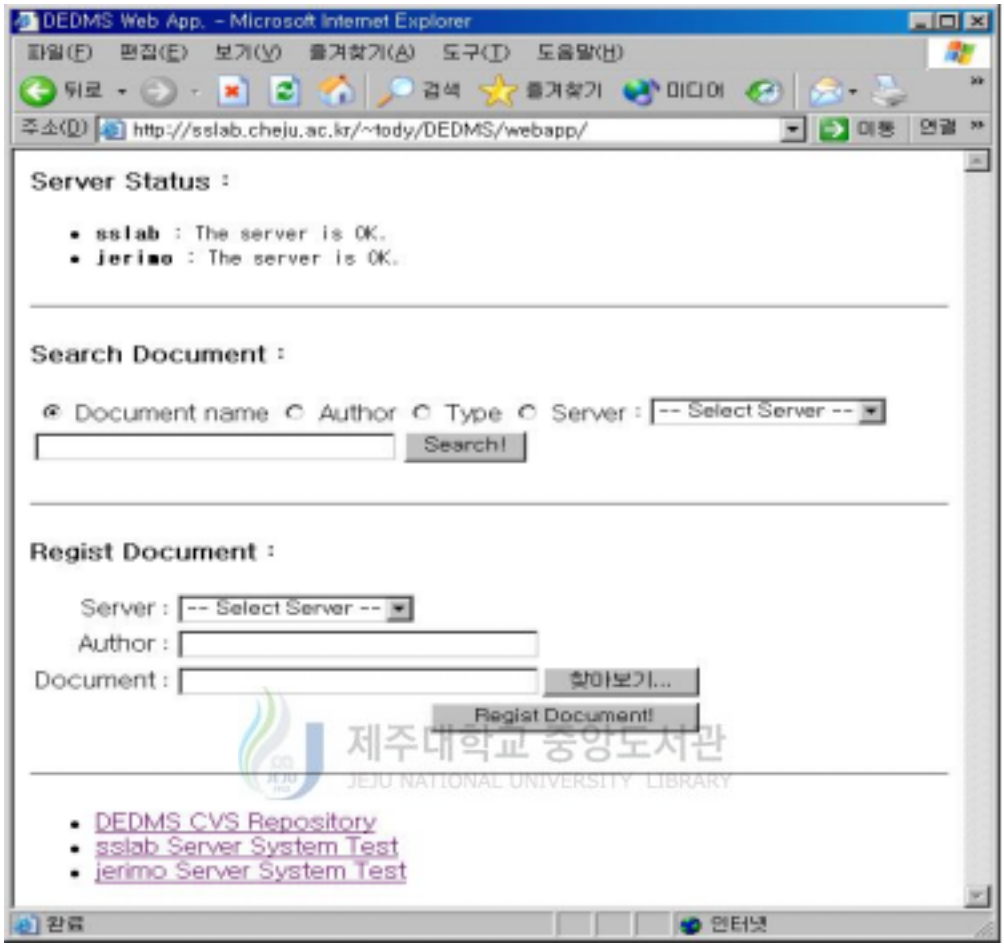

그림 15. 웹 애플리케이션 시스템 메인화면

웹 애플리케이션 서버는 미들웨어를 통해 실시간으로 변화하는 문서정보를 얻어오기 때문에 미들웨어에 대한 Config 정보가 필요하고, 그림 16에서 보듯이 미들웨어 호스트 / 포트 / 패스 정보들을 설정하였다.

```php
<?php
/*
 * Configurations of Web App. System
 */

$middleware_host = 'sslab.cheju.ac.kr';
$middleware_port = 80;
$middleware_path = '/~tody/DEDMS/middleware/middleware.cgi';
?>
```

그림 16. 미들웨어에 대한 Config 정보.

```
<params>
<param>
<value><struct>
<member>
<name>filedata</name>
<value><base64>
VGtsTFJTQzl4Ym5mSU1XcHNlTEhwUTBLZkh3Z0tHMXRLU0I4ZkNBeU1
6QWdmSHdnTWpRd0lIeDhJREkxTUNCOGZDQXlOakFnZkh3Zw0KTWpjd
0lIeDhJREk0TUNCOGZDQXlPVEFnZkh3Z016QXdJSHg4SURNeE1DQjhmQ
0F6TWpBZ2ZId05Dbng4SUUxbGJuTWdmSHdnZkh3Zw0KZkh3Z1ZWTWd
OeUI4ZkNC
                                        .
                                        .(생략)
                                        .
Z055QjhmQ0JWVXlBNElIeDhJRlZUSURrZw0KZkh3Z1ZWTWdNVEFnZkh3
Z1ZWTWdNVEVnZkh3Z2ZId2dmSHdnZkh3Z2ZId05DZz09DQo=
</base64></value>
</member>
<member>
<name>name</name>
<value><string>nike_footware_size.txt</string></value>
</member>
<member>
<name>author</name>
<value><string>jjoole</string></value>
</member>
</struct></value>
</param>
</params>
```

그림 17. 문서 등록 시 객체의 XML 구조

그림17은 등록 과정에서 Documents( )객체로 전송되는 XML 구조를 Dump한 화면이다. 문서를 등록하게 되면 처리된 결과를 그림 18과 같이

나타내주며, 그림17에서 보듯이 클라이언트가 등록하는 문서(파일)는 64바이트로 인코딩되어 분산된 서버로 전송되면, 분산된 서버에서는 64바이트로 디코딩하여 지정된 File Storage에 저장하게 되며, 이벤트 처리 결과는 미들웨어로 통보하여 실시간으로 저장하게 된다.

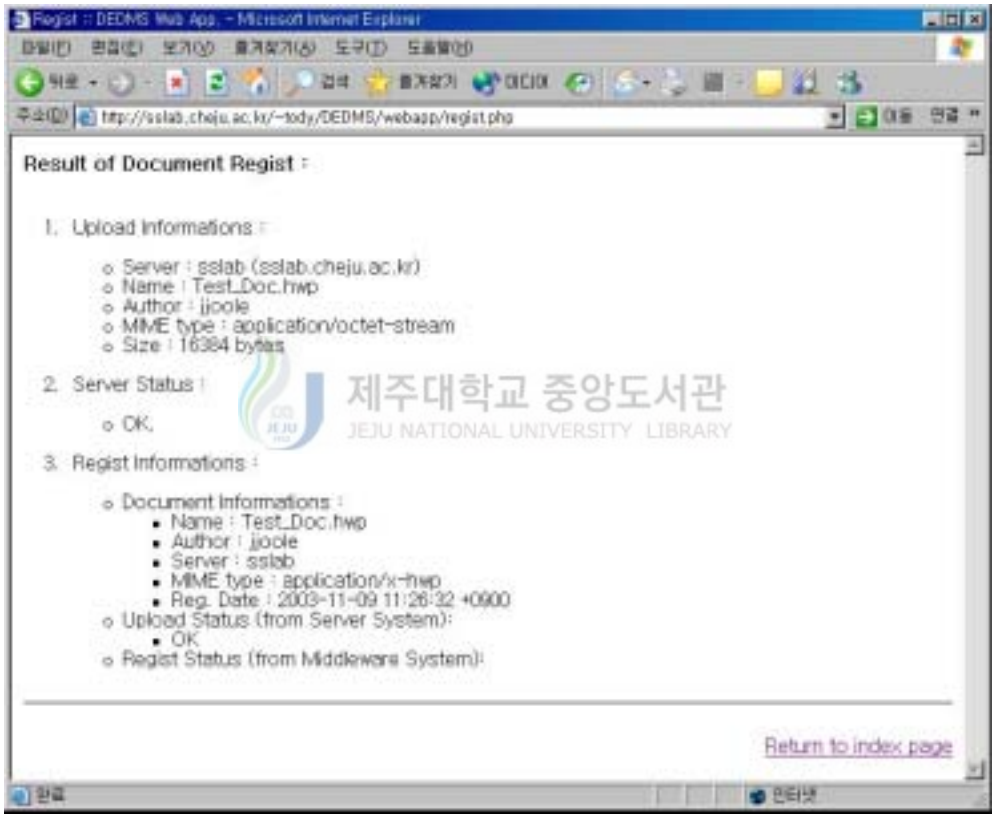

그림 18. 문서 등록 결과 화면

클라이언트는 검색(Search)과정을 통해 문서를 얻어 올 수 있고, 삭제(Delete)도 가능하다. 모든 이벤트에 대한 처리 결과는 미들웨어에서 제공받기 때문에 실시간으로 변화하는 정보에 대한 신뢰성을 갖게 된다.
문서 검색은 문서이름, 저자, 문서타입, 서버 의 형태로 이루어지며, 검색

된 결과는 그림 19과 같고, 하나의 서버에 저장된 파일을 모두 검색한 화면이다. 클라이언트는 문서 정보를 확인 후 문서(파일)를 요청 가능하며, 문서(파일)는 하이퍼링크를 따라 지정된 서버로 직접 요청하고 다운로드 할 수 있다.

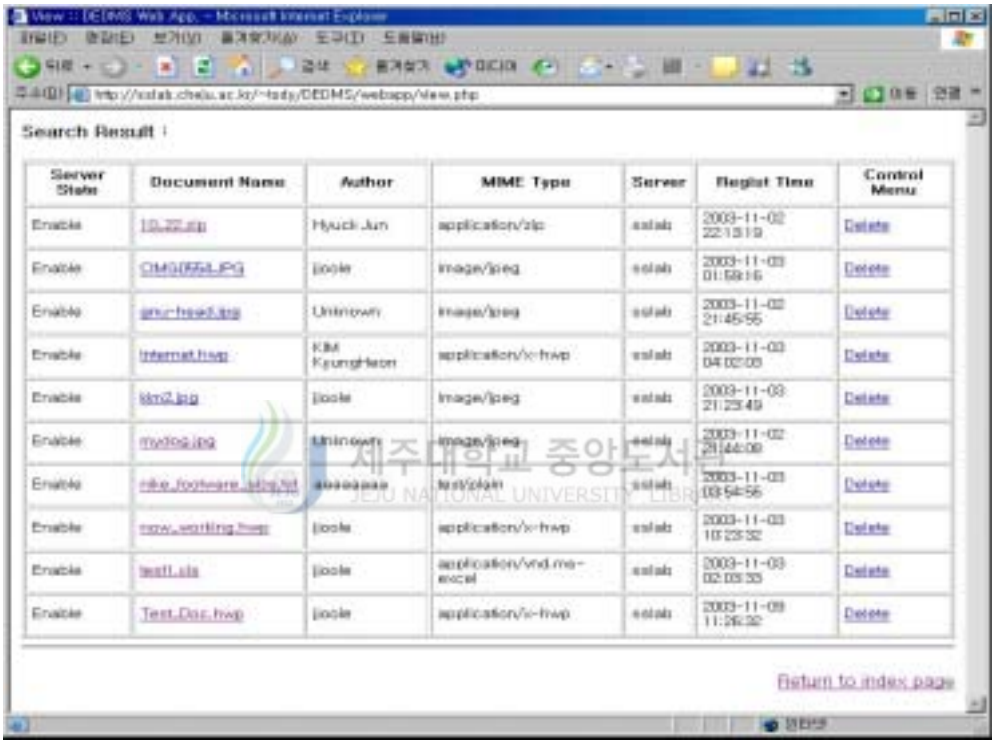

그림 19. 문서 검색 결과 화면

```
<params><param><value><array>
        <data><value><struct>
              <member><name>mimetype</name>
                <value><string>image/jpeg</string></value>
              </member>
          <member><name>author</name><value><string>jjoole</string></value>
              </member>
           <member><name>name</name><value><string>CIMG0553.JPG</string>
                </value>
              </member>
          <member><name>regdate</name><value><double>1067792581.84</double>
                </value>
              </member>
          <member><name>server</name><value><string>jerimo</string></value>
              </member></struct></value><value>
                                    ..... (생략)
           </value>
        </data>
      </array></value></param></params>
```


그림 20. 문서 검색 시 객체의 XML 구조

클라이언트는 검색된 화면에서 삭제과정도 바로 수행할 수 있으며 Delete 링크를 선 클릭하게 되면, 분산된 서버의 File Storage에 저장된 문서가 바로 삭제되고, 미들웨어에 그 정보를 실시간으로 알려주게 된다. 그림 21은 삭제(Delete) 한 결과 화면이고, 그림22는 문서 삭제 시 객체의 XML 구조를 나타낸다.

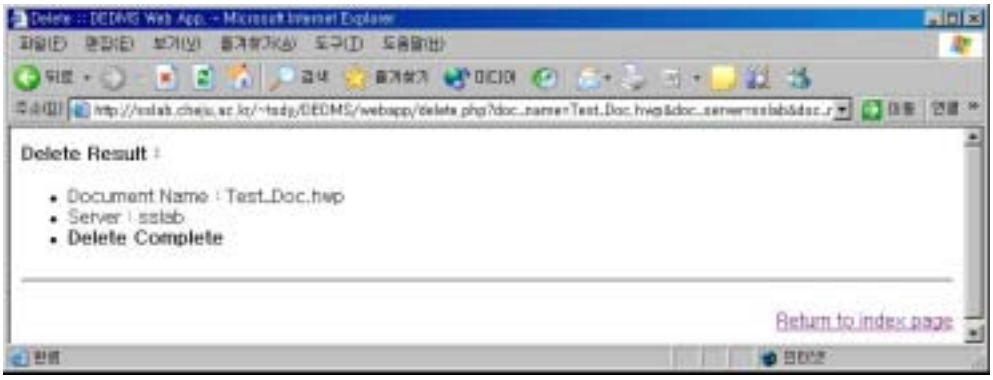

그림 21. 문서 삭제 결과 화면

```
<params>
<param>
<value><string>nike_footware_size.txt</string></value>
</param>
<param>
<value><string>sslab</string></value>
</param>
</params>
```


그림 22. 문서 삭제 시 객체의 XML 구조

4.2.2 미들웨어 시스템

미들웨어 시스템은 클라이언트의 문서관리에 대한 비즈니스 로직 부분을 담당하게 되고, 문서의 등록/삭제 등의 이벤트 발생 시 분산된 서버로부터 처리된 결과를 받아 데이터베이스에 저장하고 클라이언트의 문서 검색에 대하여 신뢰할 수 있는 정보를 제공해준다. 연결되어있는 데이터베이스와 분산된 서버의 정보를 가지고 실시간으로 발생되는 이벤트에 반응하게 된다.

```
db_info = {
        'host' : 'sslab.cheju.ac.kr',
        'user' : 'DEDMS',
        'passwd' : 'jjoole',
        'db' : 'DEDMS'
}
db_table_name =  'Documents'
servers = {
        'jerimo' : {
                'host' : 'jerimo.org',
                'path' : '/~tody/server/server.cgi',
                'port' : 80
        },
        'sslab' : {
                'host' : 'sslab.cheju.ac.kr',
                'path' : '/~tody/DEDMS/server/server.cgi',
                'port' : 80
        }
}
```


그림 23. Configuration of Middleware System

그림 23에서 보듯이 분산된 서버 정보를 입력해줌으로서 서버를 손쉽게 등록 가능하다.

4.2.2.1 이벤트 처리 시스템

분산된 서버에서 처리되는 결과 값을 XML-RPC 통신을 통해 Document( )객체로 전달받고, ServerList를 불러와 그 값을 Gathering 시스템을 통해 데이터베이스에 매핑하고 저장하게 된다. 또한, 클라이언트의 문서 검색 시 결과 값을 XML-RPC 통신을 통해 웹 애플리케이션 서버로 전달해 준다.

그림24는 미들웨어에 등록/검색/삭제 이벤트에 대한 프로세스의 동작 코드를 나타낸다.

```
from common import *
from middleware_config import *
from SimpleXMLRPCServer import *

class MiddlewareEP:
        def __init__(self):
                self.document = Document()
                self.server_list = servers
                self.db = GatheringSystem()
        def get_server_list(self, servername = ''):
                if servername == '':
                        return self.server_list
                else:
                        return self.server_list[servername]
        def status(self, echo_message = 'OK'):
                return echo_message
        def regist(self, reg_document):
                """     return value:
                        0 = Document Error
                        1 = Success
                        2 = DB Error
                """
                if type(reg_document) is type(Document()):
                        self.document = reg_document
                elif type(reg_document) is dict:
```

```
                    self.document = Document(reg_document)
                if not self.document.check():
                    return 0
                if self.db.insert(self.document) == 0:
                    return 1
                else:
                    return 2

        def find(self, search_type, search_text):
                return self.db.find(search_type, search_text)

        def delete(self, delete_document):
                self.document = Document(delete_document)
                return self.db.delete(self.document)
```

그림 24. 미들웨어 이벤트 프로세스 동작 코드

4.2.2.2 Gathering 시스템

이벤트 프로세스에서 전달받은 정보를 수집하고 Insert / Delete / Find 등의 작업을 데이터베이스에 하게 된다.

그림 25/ 26/ 27/ 28에서 Gathering 시스템의 동작 코드를 나타낸다.

```
class GatheringSystem:
        def __init__(self):
                import MySQLdb
                self.db_info = db_info
                self.table_name = db_table_name
                self.db = MySQLdb.connect( **self.db_info )
                self.cursor = self.db.cursor(MySQLdb.cursors.DictCursor)
```

그림 25. Gathering System Class

```
def insert(self, document):
        """
        return value:
                0 = Success
                1 = DB Error
                2 = Already Exists
        """
        try:
        select_query = "SELECT name, server FROM %s WHERE name = '%s'
        AND server = '%s' LIMIT 1" \
        % (self.table_name, document.name, document.server)
        if int(self.cursor.execute(select_query)):
                return 2
        except:
                return 1
        try:
        insert_query = "INSERT INTO %s VALUES('', '%s', '%s', '%s', '%s',
        '%s')" \
        % (self.table_name, document.name, document.server, \
                document.author, document.mimetype, document.regdate)
                if int(self.cursor.execute(insert_query)):
                        return 0
        except:
                return 1
```

그림 26. Insert 이벤트 처리

```
def delete(self, document):
        """
        return value:
                0 = Success
                1 = DB Error
                2 = Not Found
        """
```

```
        delete_query = "DELETE FROM %s WHERE name='%s' AND
server='%s'" \
                % (self.table_name, document.name, document.server)
        try:
                if int(self.cursor.execute(delete_query)):
                        return 0
                else:
                        return 2
        except:
                return 1
```

그림 27. Delete 이벤트 처리

```
def find(self, search_type, search_text):
                """
                return value:
                        dict = Success and Search Result
                        0 = Not Found
                        1 = DB Error
                        2 = Not Found
                """
        select_query = \
        "SELECT name, mimetype, server, author, regdate FROM %s WHERE " \
                % (self.table_name)
        if search_type == 'name':
        select_query += "name LIKE '%%%s%%' ORDER BY name, server" %
                (search_text)
        else:
        select_query += "%s LIKE '%%%s%%' ORDER BY %s, name, server" \
                                % (search_type, search_text, search_type)
        try:
                if int(self.cursor.execute(select_query)) > 0:
                        return self.cursor.fetchall()
                else:
                        return 0
        except:
                return 1
```

그림 28. Find 이벤트 처리

4.2.3 분산 서버 시스템

웹 서버 환경으로 보다 더 확장 가능한 구조이다. 웹 서버 기반 구조로 응용되는 부분은 XML-RPC 자체 모듈은 보안이나, 기타 인증 부분을 웹 서버 환경에 위임함으로서 보다 더 효과적인 서비스가 가능하기 때문이다.

분산 서버 시스템은 크게 이벤트 처리 시스템과 Saving 시스템으로 구분되며, 웹 애플리케이션을 통해 저장되는 문서(파일)는 File Storage에 저장되고, 클라이언트의 요청 시 파일을 직접 전해주게 된다.
분사된 서버 시스템에는 데이터베이스가 구축되지 않을 수도 있기 때문에, 구현 시 File Storage를 사용하여 특정 디랙토리에 문서(파일)를 저장하게 된다.

```
# 서버 이름. uniq.
server_name = 'sslab'

# base_dir은 웹에서 접근이 가능해야 함.
base_dir = '/home/tody/public_html'
base_uri = '/~tody'
document_dir = base_dir + '/documents'

# Middleware 주소
# cgi면 http에 알맞는 url로, 따로 돌아가는 서버라면 '주소:포트'로.
  middleware_url =
 'http://sslab.cheju.ac.kr/~tody/DEDMS/middleware/middleware.cgi'
```


그림 29. 분산 서버 설정 환경

4.2.3.1 이벤트 처리 시스템

미들웨어의 이벤트 처리 시스템과 비슷한 역할을 하게 된다. 웹 애플리케이션에서 발생한 이벤트(저장/삭제/수정)들을 XML-RPC 통신을 통해

Documents( )객체로 전달받고, Saving 시스템에 전달하고 처리한 이벤트 결과들을 전달하는 역할을 한다.

```
from server_config import *
from common import *
from SimpleXMLRPCServer import *
class ServerEP:
        def __init__(self):
                self.target = middleware_url
                self.server_name = server_name
                self.storage = SavingSystem()
                self.document = Document()
```

그림 30. 이벤트 처리 클래스

4.2.3.2 Saving 시스템

이벤트 처리 시스템에서 넘겨받은 정보에서 Base_dir, Base_uri, Document_dir 들을 체크하고 저장/삭제 등의 작업을 하게 된다. 웹 애플리케이션 서버 시스템에서 64바이트로 인코딩 된 파일을 디코딩하여 지정된 디랙토리에 저장하게 된다.

```
class SavingSystem:
        def __init__(self):
                self.base_dir = base_dir
                self.base_uri = base_uri
                self.document_dir = document_dir
        def save(self, filename, file_data):
                import mimetools
                from StringIO import StringIO
                if not self.check(self.document_dir, 'W', False):
                raise 'SaveError', 'Storage not found or Permission denied'
                if self.check(filename):
                raise 'SaveError', 'The filename \\"%s\\" is alredy exists' %
```

```
(filename)
                    in_string = StringIO(file_data)
                    out_string = StringIO()
                    mimetools.decode(in_string, out_string, 'base64')
                    try:
                              out_file = open(self.document_dir + '/' + filename, 'w')
                              out_file.write(out_string.getvalue())
                              out_file.close()
                    except IOError, ErrorMsg:
                              raise 'SaveError', ErrorMsg
                    in_string.close()
                    out_string.close()
```

그림 31. Saving 시스템

Ⅴ. 결 론

본 논문에서는 분산 환경에 있는 문서자원들의 정보를 저장/관리하고 클라이언트에게 신뢰할 수 있는 정보들을 효과적으로 제공하기 위하여 XML-RPC 기반의 분산 환경 문서관리 시스템을 모델링하고 구현하였다.

기존의 분산 환경 프로토콜보다 XML-RPC를 사용함으로서 웹 환경에서 특정 플랫폼 환경에 따른 프로토콜을 개발하지 않아도 되었고, 웹 서버와 같이 운용되므로 네트워크의 세세한 부분과 보안을 자신을 둘러싼 웹 서버에 위임 할 수 있어 간단하며, 실제 서비스되고 있는 웹 서버에 바로 적용시킬 수 있어 새로운 서버 구축비용을 절감 할 수 있었다.

XML-RPC는 전송은 HTTP를 이용하고 인코딩은 XML을 사용하여 분산 환경에서 메소드의 매개변수와 인코딩 가능한 것이라면 XML-RPC로 그 메소드를 호출할 수 있으므로 분산된 시스템에 있는 클라이언트가 네트워크의 또 다른 부분에 있는 서버에서 실행될 수 있는 작업을 요청할 수 있었다.

동적으로 변화하는 분산 환경의 웹 서버 문서들을 실시간으로 Gathering하기 위해 이벤트 처리 방식을 사용하고, 문서교환은 웹 애플리케이션 서버와 클라이언트가 직접통신하게 함으로서 미들웨어의 부담을 줄일 수 있었고, Gathering된 문서 정보는 데이터베이스에 저장하여 관리되므로 신뢰할 수 있는 메타정보를 검색하여 요청한 클라이언트에게 제공하고 할 수 있었다. 또한, XML-RPC를 이용함으로서 현재 서비스 되고 있는 웹 서버뿐만 아니라 자체적으로 내장한 서버/클라이언트 모듈을 사용해 서비스가 가능하고, java, perl, php, python, asp등 많은 프로그래밍언어에서 API를 제공하고 있기 때문에, 플랫폼 독립적이고 원격 프로시저 호출에서 다른 원격 지 서버에 안정적으로 접근할 수 있었다.

향후 연구 과제로는 메시지 암호화를 통한 안전한 메시지 교환이 이루어져야 하고, 인증과 권한 부분에 대한 보안 연구가 필요하다. 또한 미들웨어 서버의 다운 시 이벤트 처리를 유지하기 위한 다중 서버의 클러스터링 지원을 위해 Server-side cashing이 필요하며, Locking 기술을 사용하여 다중사용자가 하나의 파일을 동시에 접근했을 때 발생할 수 있는 문제점에 대한 연구가 필요하다.

[참고문헌]

[1] A Middleware Architecture to Improve the Efficiency of Web-Based Telemedicine Application/(BIOMEDICAL ENGINEERING APPLICATIONS
BASIS COMMUNICATIONS, vol. 12 No.3, [2002]

[2] 이달상, 김태화, 반상우, “분산 환경에서 자바 빈즈를 이용한 전자상거래 프레임워크의 설계”(産業技術硏究誌, Vol.15 No.1, [2001])

[3] CORBA Specification, Ver 2.4.2, OMG, http://www.omg.org"

[4] 진병률, 정지문, 최성, 우성구 , “이질 환경을 위한 XML 미들웨어 시스템 연구” 반도체장비학술심포지움, [2001]

[5] 황준, 김영신 “실시간 분산 컴퓨팅을 위한 미들웨어 설계 및 구현” 자연과학논문집, Vol.13 No.1, [2001]

[6] 조성연, 장주만, 전병태 “분산 컴퓨팅 환경과 미들웨어” JOURNAL OF COMPUTER SCIENCE & ENGINEERING TECHNOLOGY, Vol.1 No.1, [1999]

[7] 이재완, 전병인 “CORBA를 기반으로 한 XML 정보검색 시스템 통합 구현”, 情報通信技術硏究論文集, Vol.4 No.1, [2000]

[8] Distributed Object Oriented Software System에 관한 연구 (A Study on a Distributed Object Oriented Software System) / 오길호 (産業技術開發硏究, Vol.16 No.1, [2000])

[9] A CORBA extension for intelligent software environments /Filman, R. E. (Advances in engineering software, Vol.31 No.8-9, [2000])

[10] Hypermedia Document Management: A Metadata and Meta-Information System / Suh, W. (Journal of database management, Vol.12 No.2, [2001])

[11] XML(eXtensible Markup Language), W3C.
http://www.w3.org/XML"
[12] Simon St. Laurent, Joe Johnston, EddDumbill, "Programming Web Services with XML-RPC, O'reilly, 2001. 6
[13] Dave Winer, UserLand SoftWare Inc, "XML-RPC Specification" http://www.xml-rpc.com/spec"
[14]XML-RPC HOWTO. http://classic.helma.at/hannes/xmlrpc
[15] W3C, SOAP version 1.2 Spec. "http://www.w3.org/TR/2002/CR-soap12-part1-2002"
[16] Batch control meets the Internet Improvements in the XML protocol for data exchange, and remote Web-based and wireless monitoring, are enhancing batch control for the process plant / (Chemical engineering, Vol.108 No.5, [2001])
[17] W3C, XML Protocol Abstract Model. "http://www.w3.org/TR/2003/WD-xml-am-20030220"
[18] W3C, Document Object Model(DOM). "http://www.w3c.org/DOM"
[19] 김노환, 정충교 "XML DOM을 이용한 웹 문서 검색 알고리즘" VOL. 02 NO. 06 pp. 0775 ~ 0782. 2001. 06.

감사의 글

컴퓨터가 좋아 시작한 대학원 생활이 엊그제 같은데 2년이란 세월이 흘러 어느덧 마무리 하는 시점에 부족함과 아쉬움을 느낍니다.

2년이란 시간이 저에게는 정말 많은 기쁨과 부족함으로 인한 아픔도 있었지만, 꿋꿋하게 나아갈 수 있게 도와주신 곽호영 지도교수님께 깊이 감사 드립니다 . 아울러 많은 지도를 아끼지 않으신 김장형 교수님, 변상용 교수님, 이상준 교수님, 송왕철 교수님, 안기중 교수님, 변영철 교수님 들께 깊이 감사드립니다.

언제나 같이 했던 연구실 선배인 김정희 선생님, 이성철 선생님, 한경복 선생님과 석건, 인석, 훈, 봉남, 태은이 에게도 고마움을 전합니다.

밤새워 논문작업을 도와 준 경헌이에게도 고마움을 전하며, 2년이란 생활동안 도와주신 많은 선배님들과 동기들에게도 고마움을 전합니다.

언제나 철없는 아들이었기에 더 많은 가르침과 힘을 주셨던 부모님과 동생 민정, 인숙에게도 정말 감사드립니다.

끝까지 학업을 마칠 수 있도록 도와주신 많은 분들에게 정말 감사의 말을 전하며, 항상 발전하고 책임감 있는 사람이 되도록 노력하겠습니다.

이 글을 통해 미쳐 감사드리지 못한, 저와 인연이 되었던 모든 분들에게도 감사 드립니다. 언제나 행복하시길 기원합니다.

2003년 12월