

---

# 단말기 독립형 서버 통합 관리 미들웨어

김정희\* · 곽호영\*\*

Integrated Management Middleware for the Servers supporting the terminal-independent

Jeong-Hee Kim\* · Ho-Young Kwak\*\*

---

이 논문은 2005년도 제주대학교 연구비를 지원 받았음

---

## 요 약

인터넷은 우리에게 많은 정보를 제공해 주는 중요한 수단이 되었다. 그리고 정보 제공을 위해 인터넷의 서버 시스템들은 대용량의 정보를 처리하여 다양한 서비스를 제공함과 동시에 이들의 네트워크 구조는 더욱 더 복잡해지고 있다. 따라서 서버 시스템들에 대한 신속하고 지속적인 관리가 중요 시 되고 있다. 그러나 대부분의 서버 시스템들은 방화벽 내부에 위치하여 제한된 접근만을 허용하며 또한 각 서버 시스템들의 서로 다른 운영체제와 서로 다른 관리 방법으로 인해 효율적인 서버 시스템 관리가 이루어지고 있지 않다. 즉 서버 관리자들은 제한된 공간에서 로컬 또는 소켓 기반의 단일 접속 애플리케이션을 이용하여 다양한 형태의 서버들을 직접 관리 해야만 한다.

따라서 본 논문은 CORBA, Java RMI, DCOM 등의 분산 객체 기술을 가지면서 플랫폼이나 개발언어에 제약을 받지 않는 XML(eXtensible Markup Language) 기반의 웹서비스 기술(Web Services Architecture)을 이용하여 다수의 서버 시스템들을 통합적으로 관리할 수 있는 미들웨어 시스템을 설계하고 구현하였다. 시스템은 위치 투명성 및 단말기 독립성을 제공하며 서버 시스템 관리 명령을 호출하여 시스템 관리가 가능하다.

## ABSTRACT

The Internet has become an important instrument for providing many pieces of information to us. And server systems offer the various services after processing a volume of information, it's network topology also has been complex for providing information. Therefore this expansion means the importance of fast and sustained management of server systems. But most of server systems are located in fire-walls to allow only restricted access. Moreover, due to the difference of management based on the different operating system of each server system, management of server system is not to efficiency. That is, sever administrators have to manage by themselves many kinds of servers by using local-based or socket-based single access applications.

Therefore, we design and implement a middleware system which allows to manage many kinds of severns integrally using XML-based Web Services which have distributed object technology such as CORBA, Java RMI and DCOM, but are free from platforms or development languages. System also provides location transparency and terminal independence and calls a server system management command to manage systems.

## 키워드

Middleware, Terminal Independence, Management of server system, Web Services

---

\* 제주대학교 첨단기술연구소  
\*\* 제주대학교 통신컴퓨터공학부

## I. 서 론

인터넷이 발달하면서 우리는 일상생활에서 필요한 정보나 지식, 물품까지도 인터넷을 통해 얻을 수 있게 되었다. 반면 인터넷 사용자는 날로 증가하고 있으며, 또한 시스템은 더욱 복잡해져 감에 따라 웹상에서의 효과적인 시스템 관리가 중요한 문제로 대두되었다. 그러나 인터넷의 특성상 시스템의 장애를 미리 예측하기가 힘들며, 그리고 시스템 관리자가 지속적으로 네트워크를 관리할 수 없으며, 또한 언제 어느 순간에 어떤 시스템 관리가 필요한지 알 수 없기 때문에 장애 발생 시 신속한 발견과 대응이 무엇보다도 중요하다[1]. 이에 서버 시스템을 관리하기 위한 애플리케이션들이 개발되었지만, 대부분의 애플리케이션들은 웹 기반의 클라이언트가 하나의 서버에 접속하는 소켓(socket) 프로그래밍 형태로 개발되어 시스템 관리자가 웹 환경의 클라이언트에서 단지 단일 시스템에 접속하여 서버 시스템을 관리해야 하는 단점이 발생하여 오늘날 수많은 서버 시스템들이 존재하는 환경에서는 매우 비효율적인 관리 형태를 보여주고 있다[2]. 이에 여러 운영체제와 서버의 이용 형태에 종속되지 않으며 다수의 서버 시스템을 통합 관리할 수 있는 시스템 개발에 대한 필요성이 증가하고 있으며, 주로 CORBA(Common Object Request Broker Architecture), Java RMI(Remote Method Invocation), DCOM(Distributed Component Object Model) 등의 분산 객체 기술을 이용하여 많은 연구가 진행되고 있다. 하지만 이러한 연구도 운영체제나 개발언어 등의 조건에 제약이 받아 위의 요구 사항을 만족시키지 못하고 있다.

따라서 본 논문은 CORBA, Java RMI, DCOM 등의 분산 객체 및 XML(eXtensible Markup Language), 그리고 플랫폼이나 개발언어에 제약이 받지 않는 웹서비스 기술(Web Services Architecture)을 이용하여 다수의 서버를 통합하고 인터넷 접근이 가능한 임의의 단말기에서 서버 시스템을 관리하는 명령을 호출하여 서버 시스템들을 관리할 수 있는 미들웨어 시스템을 설계하고 구현하였다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구, 3장은 미들웨어 시스템에 대한 분석과 설계, 4장에서는 시스템 구현 및 결과, 마지막으로 5장에서는 결론과 향후 연구를 기술하였다.

## II. 관련연구

### 2.1 분산 객체 기술

오늘날 컴퓨팅 환경은 네트워크를 중심으로 변화하고 있다. 인터넷 채팅이나 화상회의, 네트워크 게임뿐 아니라 인터넷의 기술을 업무에 접목시킨 인트라넷 등의 클라이언트/서버 환경에서 운영되는 프로그램들은 본질적으로 분산된 환경에서 동작하는 프로그램이라 할 수 있다. 즉 하나의 기계에서 동작하는 경우에는 필요하지 않은 일들이 클라이언트와 서버로 구성된 분산 환경에서는 반드시 고려되어야 하는데, 먼저, 클라이언트가 자기가 원하는 서버의 연산이 어느 곳에 있는지 알아야 하며, 위치가 파악되면 서버의 연산을 호출해야 하는데, 이 때 클라이언트의 연산 호출을 이룩한 서버가 처리 결과를 클라이언트에게 전달하기 위해서는 클라이언트와 서버 사이에 공통된 약속도 필요하다. 이러한 약속을 특정 개발자가 나름대로 정의해 사용하면 본인이 개발한 프로그램에만 적용할 수 있고, 다른 개발자가 이미 개발해 놓은 것 또는 이후 개발자가 개발하는 프로그램들과는 통신할 수 없게 된다. 따라서 이러한 문제점을 해결하기 위해 탄생한 기술이 분산객체의 개념이다. 즉 분산객체 기술의 핵심은 객체지향 개념을 도입해 구현한 객체들 간의 통신을 원활하게 만들어 주는 일종의 소프트웨어 버스라 할 수 있다.

### 2.2 OSF/DCE, RPC

오픈 소프트웨어 재단(Open Software Foundation)은 분산 컴퓨팅에 대한 표준으로 DCE(Distributed Computing Environment)[3]라는 것을 제시하였다. DCE는 보안, 디렉토리 서비스, 타임 서비스, 쓰레드 처리와 RPC와 같은 기능을 지원한다. DCE RPC(Remote Procedure Call)는 클라이언트로부터 서버에 이르기까지 원격지 컴퓨터에서 동작하는 원격 함수를 호출하기 위한 API를 규정한다. 프로그래머는 DCE IDL(Interface Definition Language)로 RPC를 통해 어떤 함수가 호출될 것인지를 정의하고, IDL은 편의상 이러한 함수들을 인터페이스 그룹에 포함한다. 서버에서는 서버 프로그램이 이 함수들을 실행하게 되는데, IDL을 통해 클라이언트와 서버 양쪽 모두 어느 함수가 인터페이스 중에 있고, 함수의 인자가 어느 것인지 알 수 있게 된다. 이러한 DCE의 IDL은 DCOM과 CORBA의 커널이 동작하는 방식의 기본적인 아이디어를 제공한다. 그림 1은 RPC의 동작 방법을 설명하고 있다.

실제로 함수 호출의 원격화는 IDL 컴파일러가 작성한 스텝(Stub) 코드에 의해 수행된다. 스텝 코드는 실행 파일이나 또는 윈도우와 같은 운영체제 안에서 정적으로 실행할 수 있으며, 별도의 DLL로 번역될 수도 있다. 어떤 경우든지 클라이언트 측의 스텝 코드 안에 구축한 함수는 마치 서버에서 구축한 함수처럼 보이게 되며, 클라이언트는 이 코드가 다른 원격지에서 구현되어 있다는 것을 거의 알 수 없게 된다. 반면 클라이언트 측은 함수의 요청과 메서드의 인자를 추출하고 런타임 라이브러리를 통해 서버에 전달한다. 서버에서 RPC 서버는 클라이언트와 연결됨과 동시에 클라이언트에서 전달된 데이터 패킷을 RPC 런타임을 통해 풀고 메서드를 호출하는 스텝 코드를 호출한다. 비슷한 방법으로 서버에서의 수행 결과를 반환할 때에도 RPC 런타임이 반환된 값을 패키징하고 이를 클라이언트에 넘겨준다.

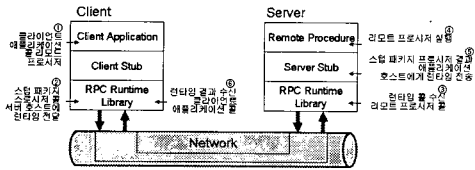


그림 1. DCE RPC 동작  
Fig. 1 Mechanism of DCE RPC

2.3 CORBA

클라이언트 객체가 ORB(Object Request Broker)라는 소프트웨어 버스를 활용해 원격지 서버의 메서드를 호출하고, 이의 수행 결과를 주고받는 것이 CORBA[2,4,5]의 모델이다. 특히 분산 객체 미들웨어는 사용자들과 프로그래머에게 상속이나 다형성처럼 객체지향 기술이 지닌 다양한 장점을 제공하며, 네트워크 프로그램에서 발생하는 문제들을 쉽게 해결할 수 있는 구조를 갖고 있다. 그림 2는 CORBA ORB의 구조를 나타내고 있다.

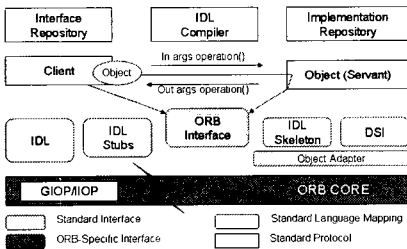


그림 2. CORBA ORB의 구조  
Fig. 2 Structure of CORBA ORB

CORBA는 클라이언트와 서버 객체 간의 통신을 위해 개념화한 소프트웨어 버스라고 생각하면 된다. 외부에 노출된 서버의 인터페이스는 CORBA의 표준 인터페이스 언어인 IDL로 표현되며, IDL로 표현한 인터페이스의 내용을 IDL 컴파일러를 이용해 자바, C++ 등의 언어로 변환해 바인딩한다. 이러한 방법으로 CORBA가 서로 다른 언어의 한계를 뛰어 넘는다. 이를 통해 프로그래머들은 가장 적절한 언어를 선택해 애플리케이션을 작성할 수 있게 되며, 여기서 ORB는 중개자 역할을 하면서 객체 사이의 상호작용을 조절하고, 객체의 위치에 관계없이 접근할 수 있는 위치 투명성을 제공한다. 즉 그림 3과 같이 클라이언트와 서버 객체들은 IDL을 통해 ORB에 필요한 객체를 요청만 하면 된다.

ORB는 OMG의 OMA(Object Management Architecture)의 한 요소일 뿐이다. OMA에는 ORB에 저 수준 객체 서비스와 고 수준 기능이 정의되어 있고, 서비스의 예로는 네이밍 서비스(naming), 이벤트 서비스(event), 그리고 트랜잭션 서비스(transaction) 등이 있다.

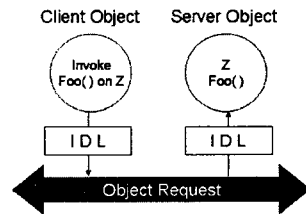


그림 3. CORBA의 호출  
Fig. 3 Call of CORBA

2.4 Java RMI / EJB

자바 RMI[5]는 JRMP(Java Remote Method Protocol)에 의해 동작한다. 자바는 자바 객체의 직렬화에 크게 의존하기 때문에, 객체를 일종의 스트림으로 마샬링(marshaling)한다. 이러한 자바 객체의 직렬화(Java Object Serialization)는 자바의 특징 가운데 하나이기 때문에, 자바 RMI 서버 객체와 클라이언트 객체가 모두 자바일 경우에만 사용할 수 있다. 각각의 자바 RMI 서버 객체는 현재 사용하고 있는 JVM(Java Virtual Machine)의 외부에 있는 서버 객체에 접근할 수 있는 인터페이스를 정의한다. 이렇게 정의한 인터페이스를 통해 서버 객체에 의해 제공되는 서비스 메서드가 결정된다.

클라이언트가 서버 객체에 처음 접근할 때 RMI는 서버 기계에서 동작하는 RMI Registry라는 일종의 명명기

법(naming mechanism)을 이용하는데, 이를 통해 현재 가능한 서버 객체의 정보를 얻을 수 있다. 자바 RMI 클라이언트가 자바 RMI 서버 객체에 대한 객체 레퍼런스를 획득한 뒤에는 마치 같은 클라이언트 어드레스 공간에서 메시지를 호출하는 방법과 같은 방식으로 서버 객체의 메시지를 이용할 수 있다. Java RMI의 시스템 구조는 그림 4와 같이 애플리케이션과 나머지 시스템 사이의 인터페이스인 스텝/스켈레톤(Skeleton) 계층, 클라이언트 스텝과 서버 스텝에 상관없이 다양한 원격 참조자나 호출 프로토콜을 지원할 수 있도록 해주는 원격참조(Remote Reference) 계층, 서로 다른 주소 공간 사이에서 마샬링된 스트림을 흘려주는 하위 수준의 계층인 전송(Transport) 계층으로 이루어져 있고, 각 계층별로 인터페이스 및 프로토콜이 정의되어 있다. 이러한 독립적인 구조 때문에 다른 계층에 전혀 영향을 주지 않고 원하는 계층을 다르게 구현해도 상관이 없게 되어 있다.

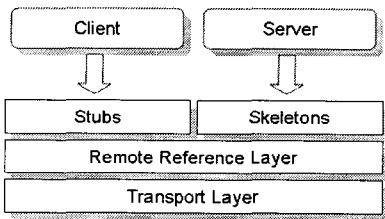


그림 4. Java RMI 구조  
Fig. 4 Structure of Java RMI

EJB(Enterprise JavaBean)는 서버측에서 확장 가능하고, 트랜잭션과 엔터프라이즈 애플리케이션에 적합한 형태로 제작된 사양이다. EJB는 분산된 n-티어 미들웨어를 구성하기 쉽도록 일관된 구조의 컴포넌트 아키텍처 프레임워크를 제공한다. 전형적인 EJB 아키텍처는 EJB 서버와 EJB 서버를 동작시키는 컨테이너, 기본적인 뼈대를 구성하는 EJB 클라이언트, JNDI(Java Naming and Directory Interface)나 JTS(Java Transaction Service)와 같은 부가적인 서비스가 추가될 수 있다.

### 2.5 DCOM / COM+

COM(Component Object Model)[6]은 OLE와 ActiveX 기술의 기초가 되는 개념으로, 인터페이스라는 미리 정의된 루틴의 세트를 통해 각 객체들 간의 상호운용을 가능하게 해주는 객체기반의 프로그래밍 사양이다. COM은 기본적으로 소스 코드 수준의 표준이 아니라 바이너리 표

준이다. 따라서 여러 가지 다른 언어로 객체를 구현할 수 있으며, 이는 서로 다른 플랫폼과 다른 주소 공간에서 실행과 통신이 가능하다. COM 객체가 동작하는 방식은 클라이언트 애플리케이션이 ClassID를 갖고 COM 객체 생성을 요청하면, OS에서 제공하는 COM 라이브러리가 시스템 레지스트리에서 해당 CLSID를 가진 COM 객체를 검색하게 된다. 이때 이 COM 객체는 클라이언트 애플리케이션의 요구를 서비스하는 COM 서버로 동작한다. 이렇게 검색된 COM 객체의 구현 부분을 찾아 해당 인터페이스 멤버 함수의 포인터를 클라이언트의 애플리케이션 해당 부분에 매핑해 준다. 이때 COM 객체의 위치는 하나의 프로세스 공간에 있을 수도 있고, 다른 작업 공간에 있을 수도 있으며, 심지어는 다른 컴퓨터 안에 있을 수도 있다. COM 객체의 인터페이스 포인터의 위치를 하나의 프로세스 공간에 있는 것처럼 포장하는 과정을 마샬링이라 하며, 이때 COM 서버 측의 포인터를 포장하는 객체를 스텝, 클라이언트 애플리케이션 측의 객체를 프록시(Proxy)라 한다. 이를 정리하면 그림 5와 같다.

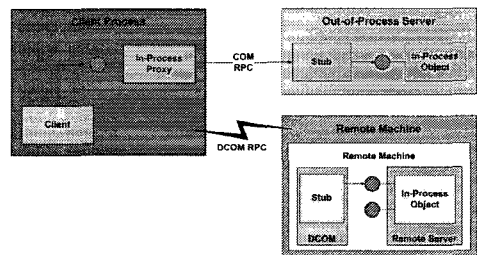


그림 5. DCOM/COM+ 객체  
Fig. 5 Object of DCOM/COM+

### 2.6 Web Services Architecture

앞에서 설명한 여러 가지 분산객체 기술들은 각각의 장점을 가지고 있지만 CORBA와 같이 구현의 복잡함과 어려움, DCOM/COM+와 같이 특정 운영체제에 종속된 기술들이 대부분이다. 따라서 이를 보완하기 위해 개발된 기술이 웹서비스 기술[7-16]이다. 웹서비스 기술은 XML 표준을 기반으로 개발된 표준화된 XML 메시지를 통해 네트워크상에서 접근 가능한 연산들의 집합을 기술하는 인터페이스를 지칭하는 용어로 기존의 웹 환경을 이용한 분산 컴퓨팅을 가능케 함으로써 웹을 통한 시스템 통합을 용이하게 하며, 이러한 특징으로 인하여, CORBA, Java RMI/EJB, DCOM/COM+ 등과 같은 기존의 분산 컴퓨팅 모델을 대체할 수 있는 새로운 기술로써 여러 산업체의

주목을 받고 있다. 웹서비스의 구조는 그림 6과 같고 그 기능은 다음과 같다.

- 서비스 제공자(Service Provider): 웹서비스를 개발하는 이로써 웹서비스 형태로 사용자에게 비즈니스 기능을 제공한다. 이때 사용자가 웹서비스를 이용하도록 하려면 우선 웹서비스를 사용하는 단체에게 널리 알려진 표준 양식으로 서비스의 기능을 기술(describe)해야 한다. 그 다음 공개적으로 접근할 수 있는 중앙레지스트리에 웹서비스의 세부사항을 공개(publish)해야 한다.
- 서비스 소비자(Service Consumer): 웹서비스를 사용하는 애플리케이션 또는 사용자를 말한다. 서비스 제공자가 기술한 웹서비스 상세 내역을 통해 서비스의 기능을 알아낼 수 있다. 먼저 서비스가 공개된 레지스트리를 검색(find)하여 서비스의 세부사항을 알아낸다. 그 다음 원하는 서비스에 바인딩(binding)하여 실제로 해당 서비스의 기능을 이용하게 된다.
- 서비스 레지스트리(Service Registry): 서비스 제공자가 자사의 웹서비스상세 내역을 올려두고 서비스 소비자가 원하는 서비스를 발견할 수 있도록 하는 중앙 저장소를 의미한다. 소비자는 원하는 서비스를 검색하고 바인딩하여 서비스를 이용할 수 있다. 일반적으로 웹서비스를 제공하는 회사의 상세 정보, 서비스 자체 정보 등이 레지스트리에 저장된다.

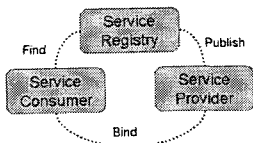


그림 6. Web Services 구조  
Fig. 6 Structure of Web Services

또한 웹서비스는 사실상 새로운 기술은 아니며 웹서비스라는 용어가 출현하기 이전에도 ASP(Application Service Provider)들은 기존의 웹 환경에서 다양한 콘텐츠와 애플리케이션들을 제공해 왔다. 그러나 웹서비스는 기존의 웹 환경에 비해 다음과 같은 여러 이점과 차이점을 보인다[3, 4].

- 단순성: 웹서비스는 XML 기반이므로 기존의 분산 컴퓨팅 모델에 비해 보다 단순하고 확장이 용이한

모델을 제공한다.

- 상호 운용성: 웹서비스는 교환 메시지를 포함한 모든 정보를 표현함에 있어 XML을 이용하며, 기존의 웹 환경 위에 구현할 수 있는 이점을 제공함으로써 이기종 시스템간의 상호 운용성을 극대화한다.
- 표준 기반: 웹서비스는 XML 기술을 기반으로 한 개방형 표준들의 지원을 받는다.
- 빠른 발전 및 업계의 지원: 웹서비스는 Microsoft, IBM, Oracle을 비롯한 주요 IT 업체들의 지원을 받으며 빠른 발전 속도와 함께 다양한 개발 도구의 지원을 받고 있다.

웹서비스 기술이 출현하기 전의 웹 환경은 콘텐츠나 애플리케이션 지향적이며, 사용자만이 서비스를 이용하는 점에 있어 사용자 중심적인 특징을 가지는 반면, 웹서비스가 출현함으로써 서비스 지향적인 특징을 보여주고 있다.

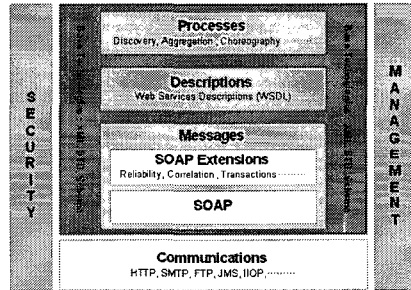


그림 7. Web Services의 스택 구조  
Fig. 7 Stack Structure of Web Services

또한 사용자뿐만 아니라 애플리케이션간의 통신이 가능하다는 특징을 가지며, 서비스 레지스트리를 이용한 서비스의 검색 및 바인딩을 가능하게 한다. 이러한 특징들을 지원하기 위해 웹서비스는 크게 서비스 작성과 기술(description), 등록(registration), 발견(discovery), 호출(invocation)이라는 개발 단계를 가지고 있으며, 각 개발 단계의 지원을 위한 관련 표준들인 SOAP, WSDL, UDDI 등이 존재한다. 그림 7은 웹서비스기술의 스택 구조를 나타낸다.

### 2.6.1 SOAP (Simple Object Access Protocol)

웹서비스가 발전함에 따라 과거의 메시지를 통한 형태에서 나아가 XML 형태의 RPC 프로토콜이 요구되었다.

XML-RPC는 이기종 운영체제에서 인터넷을 통해 다른 환경에서 수행되는 프로시저를 호출할 수 있도록 하기 위해 HTTP를 통해 XML 형태로 전송하는 것이다. 그러나 XML-RPC를 상이한 객체 하부구조상에서 사용하는 경우 쌍방간에 브리지를 제공해야 하는 어려움이 있다. 그리고 각 요청과 응답을 위한 스키마를 생성하는 메커니즘이 필요하기 때문에 각 레이어를 복잡하게 만들고 응답 속도를 떨어뜨리기도 한다. SOAP은 XML-RPC와 매우 유사하다. SOAP 역시 HTTP와 XML 문서를 통한 프로시저 호출 형태이다. 초기 SOAP 1.0은 XML-RPC를 기본으로 하고 HTTP에서만 가능했지만, SOAP 1.1부터는 HTTP, FTP, SMTP, POP3 에서도 사용할 수 있고, HTTP 확장 프레임워크도 지원하게 됐다. SOAP은 메서드와 인자를 넘기거나 객체를 호출하는 RPC 형태와 애플리케이션 통합이나 EDI 데이터 교환을 위한 Self-Describing XML 메시지 형태로도 가능하다.

2.6.2 WSDL(Web Services Description Language)

WSDL은 기업에서 UDDI에 등록된 웹서비스를 어떻게 찾고 알아볼 수 있는지에 관한 표준 XML Vocabulary이다. WSDL은 MS SOAP과 NASSL(Network Accessible Service Specification Language)에 기반을 두고 있다. 웹서비스를 기술하는 스크립트인 WSDL은 XML 포맷으로 구성되고 HTTP를 통해 전달될 수 있으며 인터페이스를 정의하는 IDL에 해당한다. WSDL을 이용해 웹서비스 제공자는 사용자에게 해당 웹서비스의 정확한 인터페이스와 사용되는 타입, 전송 프로토콜에 대한 상세 정보를 전달할 수 있다.

2.6.3 UDDI(Universal Description Discovery and Integration)

2000년 MS, Ariba, IBM이 제안한 UDDI는 B2B 트랜잭

션, 전자상거래, 웹서비스를 자동으로 처리하기 위한 프레임워크로 SOAP 메시지 스키마가 웹서비스에서 동작하도록 하는 API 조합이다. 기업들은 외부에 공개된 UDDI 레지스트리에 자신들의 제품이나 제공하는 서비스를 등록, 소개해 인터넷을 통해 사업을 운영하고 고객이나 파트너를 편리하게 검색할 수 있도록 해준다. UDDI를 사용하면 애플리케이션은 각 디렉토리 서버가 지원하는 항목 내에서 어떤 서비스가 가능한지를 바로 질의해서 얻어낼 수 있다. 데이터들은 표준 분류법을 사용해 분류할 수 있으므로 범주별 정보 검색이 가능하며, 무엇보다도 중요한 점은 UDDI는 회사의 서비스 기술 인터페이스에 대한 정보를 포함한다는 것이다. 일련의 SOAP 기반 XML API 호출을 통해 디자인타임과 런타임 모두 UDDI와 상호 작용해 기술 데이터를 검색함으로써 이러한 서비스를 호출하고 사용할 수 있도록 할 수 있기 때문에 UDDI는 웹서비스 기반으로 하는 소프트웨어 환경의 인프라 역할을 한다. 표 1은 CORBA, DCOM 그리고 웹서비스에 대해 항목별로 비교한 것이다.

III. 제안 모델 분석 및 설계

3.1 기존 연구의 문제점 분석

기존의 서버 관리를 위한 애플리케이션에 대한 연구의 대부분은 단일 서버 시스템 관리에 치중해 있다. 이러한 관리 시스템들은 특정 운영체제에 종속되어 있고 또한 종속된 운영체제에 대해 최적화되어 있으며 단일 서버 시스템에 대한 관리만 가능하기 때문에 다수의 서버를 동시에 관리하여야 하는 환경에서는 부적합하다고 할 수 있다. 이에 다수의 서버를 관리할 수 있도록 하는 연구가 진행되었지만 운영체제에 종속되지 않은 간단한 처리에 대해

표 1. CORBA, DCOM, Web Services 비교  
Table. 1 Comparison between CORBA, DCOM and Web Services

구분	CORBA	DCOM	Web Services
전송계층	ORB Protocol	DCE-RPC	HTTP, SMTP
인터페이스	IDL	IDL	WSDL
데이터 인코딩	CDR	NDR	SOAP, XML
서비스 발견	명명, 레지스트리 서비스	레지스트리	UDDI
시스템 호환성	부분적	윈도우 계열	이기종 호환성 높음
Preferred Language	Java, C++	C++, Visual Basic	C#, Java, C++, Visual Basic 등
접속형태	접속 비지향	접속지향	접속 비지향
방화벽 친화도	낮음	낮음	높음

서만 관리가 가능하도록 되어 있어서 실제 시스템 관리에 적용하기에는 부족한 부분이 많다. 또한 애플리케이션 개발에 특정 언어를 사용함으로써 개발에 사용한 언어가 하드웨어 시스템에 종속되는 단점도 발생한다. 그림 8은 웹을 통해 Unix/Linux 시스템을 관리할 수 있는 애플리케이션(Webmin)의 기능 중 한 부분을 보여주고 있다.

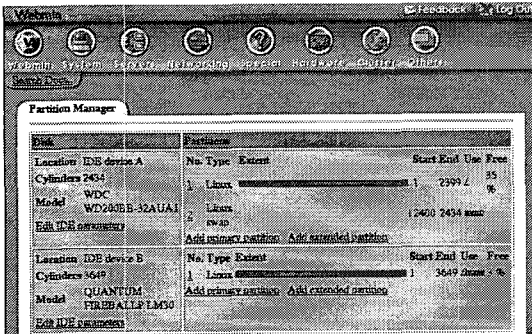


그림 8. Web용 Unix/Linux 서버 관리 예  
Fig. 8 Example about Unix/Linux Server

### 3.2 제안 시스템 구조

본 논문에서는 소켓 프로그래밍을 기반으로 한 관리 프로그램의 단점을 개선하고 다수의 서버 시스템을 효과적으로 관리하기 위하여 단말기의 기종에 관계없이 인터넷에만 접근할 수 있다면 서버 시스템들을 손쉽게 관리할 수 있는 서버 통합관리 시스템(미들웨어 시스템)을 설계하고 구현한다. 따라서 웹서비스 기술을 적용한 서버 통합관리 시스템의 전체적인 참조 모델은 그림 9와 같고, 서비스 제공자는 서버, 서비스 사용자는 관리자들, 서비스 저장소는 웹서비스 표준인 UDDI로 구성이 가능하다.

서버 측의 기본적인 역할은 서버의 관리를 위한 명령에 대한 웹서비스를 인터넷상에 공개하는 것이며, 이를 위해 서버는 웹서비스 공개와 함께 UDDI 저장소에 서버의 정보와 관리를 위한 명령들에 대한 WSDL 문서를 등록하여 관리자가 관리 명령을 호출할 때 어떠한 정보가 필요한지 제공하게 된다. UDDI 저장소는 서버 측에서 등록한 서버 정보와 관리 명령에 대한 WSDL 문서를 저장하고 관리자의 요청에 의해 제공하는 역할을 한다. 제공된 WSDL 문서는 관리자가 서버의 관리 명령을 호출하기 위한 여러 가지 정보를 제공하게 된다. 관리자는 UDDI 저장소로부터 서버의 관리 명령에 대한 WSDL 문서를 이용하여 명령 호출에 필요한 정보와 함께 서버 측에서 공개한

관리 명령 웹서비스를 호출하여 필요한 명령을 수행할 수 있다. 이와 같이 웹서비스 기술을 이용하므로 관리자는 인터넷에 접근 가능한 기기(PC, Notebook Computer, PDA, Mobile Phone 등)를 가지고 기기에서 작동 가능한 웹 브라우저나 관리 웹서비스용 애플리케이션을 이용하여 시간이나 장소에 구애 받지 않고 서버관리 명령을 수행할 수 있게 된다. 또한 어떠한 프로토콜로도 호출이 가능하기 때문에 방화벽 내부에 있는 서버에 대해서도 아무런 제약 없이 접근된다.

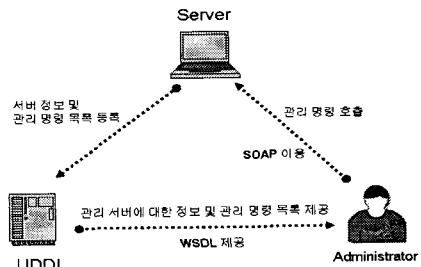


그림 9. 제안 시스템 참조 모델  
Fig. 9 Reference model of proposed system

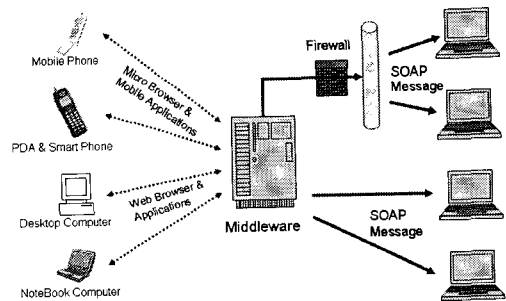


그림 10. 제안 시스템 구조  
Fig. 10 Structure of proposed system

### 3.3 제안 시스템 설계

제안 시스템의 기본 구조는 일반적인 웹서비스 모델을 기본으로 구성하였으나 서버 관리자의 측면에서 살펴보면 개방되어 있는 웹서비스기술을 제안 시스템에 그대로 적용하기에는 적합하지 못한 부분이 있다. 이에 대한 문제점을 다음과 같이 분석할 수 있다. 첫째, 기밀성 보장(Privacy), 인증(Authentication), 부인부채(Non-repudiation)와 같은 보안 문제에 대해 웹서비스기술은 취약하다. 둘째, 안전한 메시지 교환과 QoS(Quality of Service)를 위해서 웹서비스 호출 시 호출 명령이 정확하게 전달되었는지 또는 정

확하게 전달받았는지 확인할 수 있는 능력이 부족하다. 셋째, 서버 관리 명령의 대부분은 트랜잭션 처리 부분이 많다.

따라서 본 논문은 이러한 문제점들을 보완하기 위해 그림 10과 같이 관리자와 관리 서버의 중앙에 서버 관리를 위한 통합 관리 미들웨어 시스템을 배치하여 해결하고자 한다. 제안 시스템은 크게 관리하고자 하는 서버 상에서 관리 명령 요청을 수행하는 서버 애플리케이션, 관리자와 관리 서버들 사이에 위치하면서 여러 단말 장치들의 요청을 처리하거나 관리 서버들에게 명령을 요청하는 미들웨어 시스템, 웹서비스를 적용하여 미들웨어에게 관리 서버에 대한 명령 요청을 하는 클라이언트 애플리케이션 부분으로 구성된다. 그리고 여러 종류의 단말기들은 인터넷 기반 웹 브라우저나 서버 관리 웹서비스용 애플리케이션을 이용하여 관리 대상 서버들에 대한 필요한 관리 명령을 요청할 수 있도록 한다.

### 3.3.1 서버 애플리케이션

서버 애플리케이션(SPA)의 기본 기능은 서버 관리를 위한 웹서비스를 공개하며 서버 정보, 구동 상태, 그리고 관리 명령들을 미들웨어 내의 UDDI 저장소에 등록한다. 그리고 등록된 정보들은 명령 요청 시 인증 받은 사용자에 의해서만 실행되도록 한다. 그림 11처럼 SPA는 서버에 대한 직접적인 명령을 처리하며 실행한 명령이 올바르게 수행되었는지 확인하는 Server Side Command Processor, 서버 관리 명령들을 웹서비스로 공개하는 Web Services Processor, 이 중간에 요청 명령을 서버에서 실행시키기 위해 서버 운영체제의 명령으로 변환하는 Command Processor, 명령 실행에 대한 로그를 처리하는 Log Processor, 그리고 관리자 인증 부분을 처리하는 Authentication Processor로 구성된다.

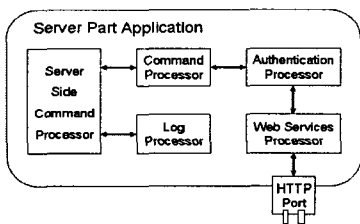


그림 11. 서버 애플리케이션  
Fig. 11 Server Part Application

우선 관리자의 요청은 네트워크를 통해 전달되며, 이때 사용 가능한 포트(Port)는 제한이 없다. 일반적으로

HTTP 포트를 사용하지만 조건에 따라 SNMP나 JMS(Java Message Services)등 다른 포트도 사용할 수 있다. Web Services Processor 부분은 내부적으로 관리 명령에 대한 공개 부분과 미들웨어 내의 UDDI 저장소에 서버 정보와 관리 명령에 대한 정보를 등록하는 부분, 그리고 요청 명령에 대한 검증부분으로 구성된다. Command Processor 부분은 서버의 운영체제에 따라 비록 동일한 관리명령일지라도 내부 처리 절차가 다를 수 있기 때문에 Command Processor는 이러한 관리명령들을 모듈화하여 명령의 추가, 수정, 그리고 삭제가 쉽도록 하고, 필요시 모듈들을 조합할 수 있도록 하여 효율적인 명령 처리가 가능하도록 지원한다. Server Side Command Processor에서는 서버와 실제적으로 명령을 처리하는 부분으로 Command Processor에서 요청한 명령을 실제로 수행하며 또한 수행한 명령이 올바르게 수행되었는지 확인하여 그 결과를 반환한다. Log Processor는 서버 관리에서 서버에 문제가 발생했을 경우, 원인 분석을 위한 정보를 제공한다. 따라서 관리 명령 요청이나 요청 종료 시 정보를 Log Processor에 전달하여 로그를 생성하고 저장한다. 로그에서 저장할 정보는 명령 요청자, 실행 명령, 실행 시간, 실행 결과, 그리고 요청자의 접근 IP 주소 등의 포함된다.

### 3.3.2 미들웨어 시스템

미들웨어 시스템은 그림 9의 문제점을 보완하기 위해 그림 12와 같이 UDDI Registry, Command Processor, Log Processor, Authentication Processor, Web Services Processor, Connection Interface와 관리자 및 서버 정보를 저장하기 위한 관계형 데이터베이스를 포함한다.

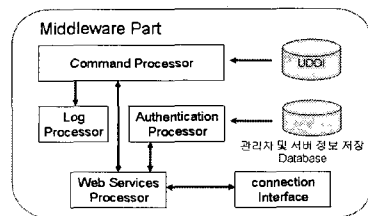


그림 12. 미들웨어 시스템  
Fig. 12 Middleware System

Connection Interface는 단말기에서 웹 브라우저를 통한 접근을 요청할 때 단말기 형태에 부합하는 웹 문서를 생성하여 단말기로 요청 결과를 응답하는 기능을 제공한다. Web Services Processor는 제공하는 웹서비스를 공개하고



클라이언트 요청에 따른 처리를 한다. SPA와는 다른 웹서비스를 제공하며 관리자가 직접적으로 호출하는 웹서비스들이 제공된다. *Authentication Processor*는 관리자 및 서버 정보 저장 데이터베이스를 참조하여 클라이언트에서 요청한 관리자의 인증에 관한 연산을 수행한다. *Command Processor*는 *UDDI Registry*에 등록된 서버 정보 및 서버 관리 명령을 이용하여 클라이언트가 요청한 관리 서버에 대한 명령을 서버 측에 요청하고 수행 결과를 확인하여 명령이 정확하게 수행되었는지 검증하는 역할을 한다. 또한 서버 측의 SPA의 정보와 관리 명령들을 등록해 놓고 이를 관리자들에게 제공해 주는 역할을 한다. *UDDI Registry*의 명령 수행에 대한 검증 방법인 서비스 제어 관련 명령에 대해서는 서비스에 대한 프로세스의 기동 상태를 확인함으로써 간단하게 검증이 가능하다. *Log Processor*는 *Authentication Processor*와 *Command Processor*가 요청을 처리할 때 요청 정보를 로그로 저장하며, 오류나 문제 발생 시 해결하기 위한 정보로 사용할 수 있게 한다.

3.3.3 클라이언트 애플리케이션

클라이언트 애플리케이션은 미들웨어 시스템에서 제공하는 웹서비스를 이용하여 개발된다. 즉 인터넷에 연결 가능한 단말기에 작동하는 개발언어를 이용하여 *User Interface*부분을 생성하고 미들웨어 시스템에서 제공되는 웹서비스를 이용하여 손쉽고 빠르게 단말기마다 적합한 환경을 제공할 수 있는 클라이언트 애플리케이션을 개발할 수 있다.

그림 13은 클라이언트 애플리케이션을 나타낸다. *User*

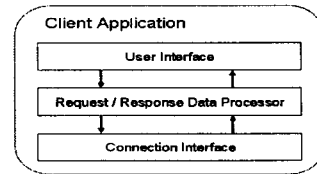


그림 13. 클라이언트 애플리케이션  
Fig. 13 Client Application

*Interface*는 관리자가 직접 조작하게 되는 부분으로 여러 가지 조건에 의해서 많이 달라지는 부분이다. 관리자가 웹 페이지 형태로 제작을 원하면 웹 페이지로 제작이 가능하며, 또한 휴대용 기기에서 작동하는 형태를 원하면 그에 맞도록 제작이 가능한 부분이다. *Request/Response Data Processor*는 미들웨어 시스템에 관리 명령 요청을 보내거나 요청에 대한 응답 메시지를 *User Interface*부에서 이용 가능하도록 변형해 주는 역할을 담당한다. *Connection Interface*는 관리자의 요청을 미들웨어 시스템에 전달하며 반환된 결과를 검증하는 역할을 한다. 또한 미들웨어 시스템과 지속적인 연결을 위해 필요한 정보인 세션과 인증 정보를 소유한다.

IV. 제안 모델 구현 및 결과

4.1 구현 환경

본 논문은 인터넷 환경에서 다수 서버를 단말기 독립적으로 관리하는 통합 관리 미들웨어 시스템을 제안하였다. 이를 위해 클라이언트 시스템은 [표 2]에서와 같이

표 2. 구현 환경  
Table 2. Implement Environment

구분	하드웨어	소프트웨어	언어
Server System I	Intel Pentium III 600Mhz RAM 256MB	RedHat Fedora Core1, JDK 1.4.2_05 Apache Tomcat 5.03, Apache Axis 1.1	JAVA
Server System II	Intel Pentium IV 2.4Gb RAM 1024MB	Windows Server 2003 , IIS 6.0	C#
Middleware System	Intel Pentium IV 2.0Gb RAM 512MB	Microsoft Windows XP Pro, JDK 1.4.2_06 Apache Tomcat 5.03, Apache Axis 1.1 Apache jUDDI 0.93C, MySQL 4.17	JAVA
Client System I	Intel Pentium III 750Mhz RAM 256MB	Microsoft Windows XP Professional Sun Java SDK 1.4.2_06	JAVA
Client System II	Intel Pentium IV 2.8Mhz RAM 512MB	Microsoft Windows XP Professional .NET Framework 1.1	C#
Client System III	HP iPAQ 5550H	Microsoft Pocket PC 2003. .NET Compat Framework 1.1	C#
Client System IV	Intel Xscale 400MHz ROM 64MB / RAM 64MB 3.5인치 TFT Color LCD	Microsoft Windows CE.NET 4.0, .NET Compat Framework 1.1	C#

Java, C#, PDA(CE.NET) 등의 환경을 갖도록 구현하였다.

## 4.2 구현 결과

### 4.2.1 서버 애플리케이션

서버 애플리케이션은 그림 14와 같은 구조로 이루어져 있고, 미들웨어 시스템과의 접근만을 허용한다. 이는 서버 시스템이 외부로부터의 공격에 대해 피해를 최소화하고자 하는 것이며, 또한 서버의 위치를 모른다 하더라도 미들웨어 시스템을 이용하여 접근할 수 있도록 서버의 위치 투명성 보장이 가능케 한다. Command Processor 부분은 웹서비스 호출 부분과 서버 시스템을 서로 연결해 주며, 대부분의 명령 처리가 이루어진다. Web Services Processor 부분은 관리자가 호출할 수 있는 명령들을 나열하는 부분으로써 이곳에 등록되지 않은 명령은 외부에서 호출이 불가능하다. 그림 15는 서버 애플리케이션 부분의 전체적인 처리 흐름도이다.

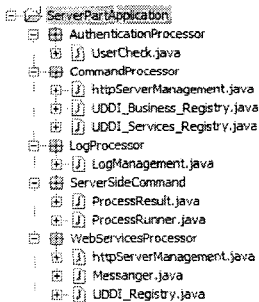


그림 14. 서버 애플리케이션  
Fig. 14 Server Part Application

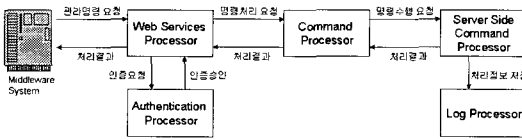


그림 15. 서버 애플리케이션 흐름도  
Fig. 15 Diagram of Server Part Application

### 4.2.2 미들웨어 시스템

그림 16은 미들웨어 시스템의 구조를 나타낸다. 제안 시스템의 구현은 Java를 이용하여 구현하였기 때문에 웹서비스와 관련된 SOAP 엔진, WSDL 해석 등의 작업은 Java Library[17-22]를 이용하였다. 그리고 Java에서 웹서비스의 호출은 스텝을 이용하였다. 그림 17에서 WebServicesProcessor.pllab 부분이 그림 16의 서버 측 웹

서비스의 WSDL을 이용하여 스텝을 생성한 것이다. 그림 18은 미들웨어 시스템의 전체적인 처리 흐름을 나타내고 있다. 미들웨어 시스템에는 UDDI 저장소가 존재한다. 이는 미들웨어 시스템의 하부에 있는 서버에서 제공하고 있는 웹서비스 메서드에 대한 정보를 저장하고, 다시 이를 이용하여 관리자에게 제공하기 위한 웹서비스를 신속하게 생성하기 위함이다. 또한 Connection Interface를 통해 웹 브라우저를 이용한 접근에 대해서도 클라이언트 애플리케이션을 이용한 접근과 같은 기능을 제공할 수 있으며, 추가적으로 웹서비스를 지원하는 JSP, ASP, PHP등의 Server Side Script 언어를 이용하여 구현이 가능하다.

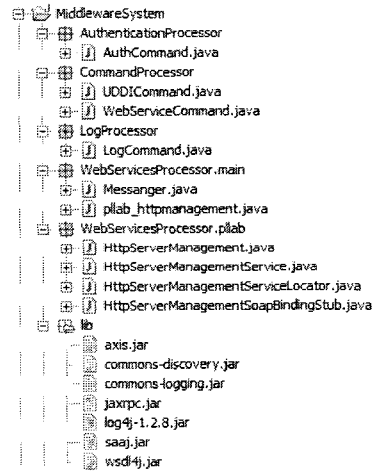


그림 16. 미들웨어 시스템  
Fig. 16 Middleware System

```
<code><pre></pre></code>
```

그림 17. 서버 애플리케이션 관리 명령(WSDL)  
Fig. 17 Management Command of Server Part Application(WSDL)

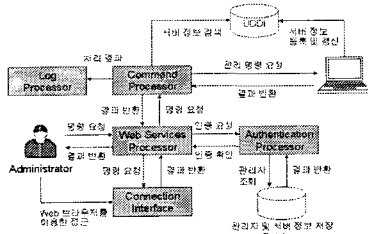


그림 18. 미들웨어 처리 흐름도  
Fig. 18 Diagram of Middleware System

4.2.3 클라이언트 애플리케이션

클라이언트 애플리케이션의 개발은 웹서비스를 지원하는 어떠한 언어로도 개발이 가능하다. 즉 SOA(Services-Oriented Architecture) 개념[23]을 이용하여 개발자는 서버 관리 명령보다는 사용자 인터페이스 부분만 신경을 쓰고 그 후 필요한 서버 관리 명령 웹서비스를 호출하여 사용자 인터페이스 부분에 연결함으로써 개발이 완료된다. 그림 19는 클라이언트 애플리케이션의 전체적인 처리 흐름을 나타내고 있으며, 그림 20은 Java를 이용하여 개발한 클라이언트 애플리케이션의 구조이다. 미들웨어 시스템에서와 마찬가지로 Connection Interface부는 그림 21에서 보여주고 있는 미들웨어 시스템에서 제공하고 있는 웹서비스의 WSDL 문서를 이용하여 웹서비스 호출을 위한 스텝을 생성하였다.

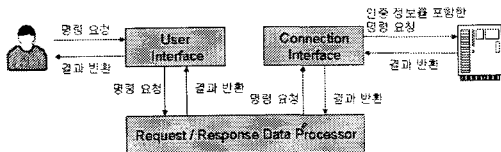


그림 19. 클라이언트 처리 흐름도  
Fig 19. Diagram of Client Application

```

ClientApplication
├── DataProcessor
├── Messenger.java
├── RequestDataProcessor.java
├── ResponseDataProcessor.java
├── UserInterface
├── UserInterface.java
├── ConnectionInterface
├── Ptlab_httpmanagementk.java
├── Ptlab_httpmanagementService.java
├── Ptlab_httpmanagementServiceLocator.java
├── Ptlab_httpmanagementSoapBindingStub.java
├── WSDL 시스템 리미트러리 [tree] <4.2.0_06>
├── Web Services Lib
├── AbsoluteLayout.jar - C:\Wdve\Whetbeans\Wid4\Whodues\Wfest
├── axis.jar
├── commons-discovery.jar
├── commons-logging.jar
├── jaxrpc.jar
├── log4j-1.2.8.jar
├── soap.jar
├── wsdl4j.jar
    
```

그림 20. 클라이언트 애플리케이션  
Fig. 20 Client Application

```

<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
  targetNamespace="http://cyber.chjeju.ac.kr:8080/axis/services/ptlab_httpmanagement"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/soap-envelope/"
  xmlns:tns="http://cyber.chjeju.ac.kr:8080/axis/services/ptlab_httpmanagement"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:axis:="http://cyber.chjeju.ac.kr:8080/axis/Americas/Messenger"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl:="http://www.w3.org/2001/XMLSchema"
  >
  <wsdl:types>
    <xsd:import base="http://www.w3.org/2001/XMLSchema"
      namespace="http://www.w3.org/2001/XMLSchema" />
  </wsdl:types>
  <wsdl:message name="MailInfoRequest">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="mailInfo" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:message name="MailInfoResponse">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="mailInfo" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:message name="StartRequest">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="startRequest" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:message name="StartResponse">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="startResponse" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:message name="StopRequest">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="stopRequest" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:message name="StopResponse">
    <xsd:sequence base="http://cyber.chjeju.ac.kr:8080/axis/services/Messenger"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="stopResponse" type="xsd:string" />
    </xsd:sequence>
  </wsdl:message>
  <wsdl:binding name="ptlab_httpmanagement" type="soap:binding" />
  <wsdl:service name="ptlab_httpmanagementService"
    >
    <wsdl:port name="ptlab_httpmanagementService"
      type="soap:binding" />
  </wsdl:service>
  </wsdl:definitions>
    
```

그림 21. 미들웨어 시스템의 웹 서비스 (WSDL)  
Fig. 21 Web Services(WSDL) of Middleware System

4.2.4 클라이언트 애플리케이션의 응용

웹서비스 기술은 XML을 기반으로 한 통신 프로토콜인 SOAP를 이용하기 때문에 이 기종간의 호환성이 높다. 이는 관리 서버의 플랫폼에 상관없이 클라이언트를 개발하는 언어가 웹서비스를 지원하면 어떤 형태로도 개발이 가능하다. 일반적으로 개발자는 웹서비스의 메서드에서 필요한 인수와 반환 형태 등의 정보를 기술하고 있는 WSDL을 획득하고 이를 이용하여 개발언어에 따른 스텝을 생성하며 필요한 웹서비스 메서드를 호출하게 된다.

4.3 구현결과 고찰

본 논문의 구현에서는 웹서비스의 특징을 이용하고, 또한 웹서비스 기술을 지원하는 언어인 C#, JAVA를 사용하여 4.1절의 다양한 클라이언트 환경에서 웹 서버 중 하나인 Apache Web Server를 제어할 수 있는 클라이언트 애플리케이션을 개발하였다. 그림 22와 23은 각각 C#과 Java를 이용하여 개발된 데스크탑용 클라이언트 애플리케이션이며, 여기서는 하나의 단일 서버의 웹 서버를 시동하고 종료시켰다. 그림 24와 25는 각각 C#을 이용한 CE.NET, Pocket PC용 클라이언트 애플리케이션이며, 특히 서로 다른 다수의 서버가 관리됨은 그림 25의 (a), (b)를 통해 보여주고 있다.

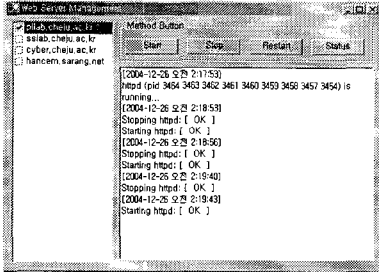


그림 22. C#을 이용한 클라이언트 애플리케이션  
Fig. 22 Client Application using C#

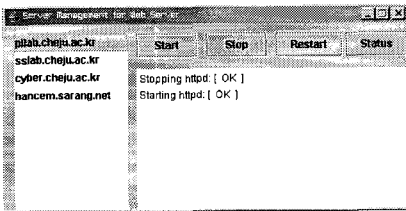


그림 23. 자바를 이용한 클라이언트 애플리케이션  
Fig. 23 Client Application using JAVA

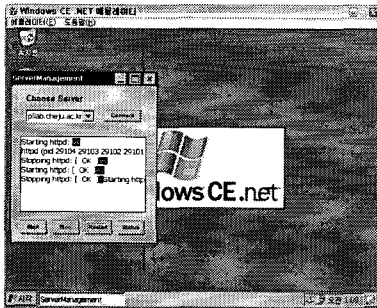
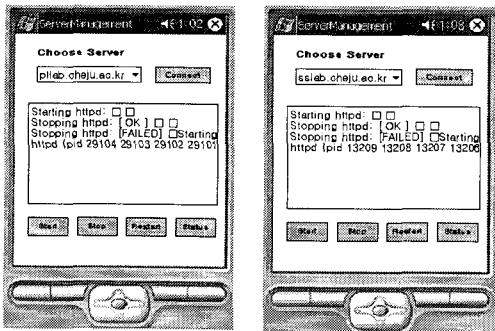


그림 24. 윈도우 CE.NET에서의 클라이언트 애플리케이션  
Fig. 24 Client Application in Windows CE.NET



(a) (b)  
그림 25. 포켓 PC용 클라이언트 애플리케이션  
Fig. 25 Client Application in Pocket PC

## V. 결론

본 논문은 인터넷 상에서 서버 시스템들의 장애 발생 시 신속한 발견과 대응을 가능하게 하는 미들웨어 시스템을 설계하고 구현하였다. 즉 기존의 소켓 방식의 애플리케이션은 특정 시스템 또는 특정 운영체제에 종속적인 경우가 대부분이었지만 제안된 시스템은 웹서비스 기술을 이용함으로써 애플리케이션 개발자들은 관리자가 사용할 시스템(컴퓨터, PDA, Mobile Phone 등)에 맞추어 UI(User Interface)부분만을 개발할 수 있고, 그 외 부분은 웹서비스 기술에 의해 인터넷 상에 공개된 원격 메서드를 호출하여 처리가 가능함으로써 개발 시간 및 애플리케이션의 완성도를 높인다. 또한 웹서비스 기술은 HTTP를 기본 프로토콜로 사용하기 때문에 방화벽 내부에 존재하는 서버에 대해서도 별도의 처리 없이 접근이 가능하며, 필요에 따라서 다른 프로토콜(SMTP, JMS, FTP 등)로 변경도 용이하다.

향후 연구 과제로는 손쉽게 관리 명령의 추가, 수정, 그리고 삭제 가능하도록 서버 애플리케이션, 미들웨어 시스템, 클라이언트 애플리케이션을 상호 연동하는 자동화된 시스템을 구현하도록 하여야 할 것이다. 또한 XML을 기반으로 한 SOAP나 WSDL 문서의 암호화 및 압축, 그리고 웹서비스를 확장한 시멘틱 웹 서비스(Semantic Web Services) 기술을 적용한 미들웨어 시스템으로의 접근이 필요하리라 본다.

### 감사의 글

본 연구는 2005년도 제주대학교 학술연구비의 지원에 의해 연구되었습니다.

### 참고문헌

- [1] Aileen Frisch, "Essential System Administration", 1995, O'Reilly & Associates, Inc.
- [2] 박남섭, 김태윤 "CORBA 기반 원격 서버 관리 시스템의 설계 및 구현", 한국정보처리학회 춘계학술발표논문집 Vol.9, No.1, [2002].
- [3] Microsoft Strategy for Distributed Computing and DCE

Services, <http://www.microsoft.com/TechNet/Analpln/dce.asp>

[4] CORBA Specification, Ver 2.4.2, OMG, <http://www.omg.org>

[5] Java, RMI and CORBA, OGM, <http://www.omg.org/news/compare.html>

[6] Richard Grimes, Professional DCOM Programming, Wrox Press.

[7] 이경하, 이규철, "웹서비스의 표준화 동향과 발전 방향", 데이터베이스연구회지, VOL.19, No.01, pp.0080-0087, 2003. 03.

[8] "Web Services Architecture", "<http://www.w3.org/TR/ws-arch>", 2004. 2. 11

[9] SOAP Version 1.2 Part 0 Primer, <http://www.w3.org/TR/soap12-part0/>, 2004. 6. 25

[10] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, <http://www.w3.org/TR/wsdl20/>, 2004. 8. 3.

[11] UDDI Technical White Paper, UDDI.ORG, [http://www.uddi.org/pubs/Iru\\_UDDI\\_Technical\\_White\\_Paper.pdf](http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf)

[12] David Orchard, "Web Services Pitfalls", <http://www.xml.org/pub/a/2002/02/06/webservices.html>, XML.com, Feb 2002.

[13] Fabio Casati, Ming-Chien Shan, "Models and Languages for Describing and Discovering E-Services", Tutorial, Semantic Web Working Symposium, Stanford, USA, July 2001.

[14] Heather Kreger, "Web Services Conceptual Architecture 1.0". <http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, 2001, 4

[15] Donald F. Ferguson, Tony Storey, Brad Lovering, John Shewchuk, "안전하고 신뢰성 있는 웹서비스", IBM DeveloperWorks, <http://www-903.ibm.com/developerworks/kr/webservices/library/ws-securtrans.html>, 2003, 10.

[16] Eric Newcomer, "Understanding Web Services", Pearson Education, 2002, 12

[17] Henry Bequet, Meeraj Moidoo Kunnumpurath, Sean Rhody, Andre Tost, "Beginning Java Web Services", Wrox Press, 2003, 5.

[18] S. Jeelani Basha, Scott Cable, Ben Galbraith, Mack Hendricks, Romin Irani, James Milbery, Tarak Modi, Andre Tost, Alex Toussaint, "Professional Java Web Services", Wrox Press, 2002, 11.

[19] Ray Lai, "J2EE Platform Web Services", Prentice Hall, 2003, 12, 24

[20] Harvey M. Deitel, Paul J. Deitel, Jonathan P. Gadzik, Kyle Lomeli, Sean E. Santry, Su Zhang, "Java Web Services for Experienced Programmers", Pearson Education, 2003, 9, 17.

[21] Steve Graham, Simeon Sumeonov, Toufic Boubez, Doug Dais, Glen Daniels, Yuichi Nakamura, Roy Neyama, "Building Web Services with Java (Making Sense of XML, SOAP, WSDL, and UDDI)", SAMS Press, 2002.

[22] David Chappell, Tyler Jewell, "Java Web Services", O'Reilly, 2002, 5

[23] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, Tony Newling, "Patterns: Service-Oriented Architecture and Web Services", IBM Red Books, 2004, 4

### 저자소개

#### 김 정 희(Jeong-Hee Kim)



1994년 제주대학교 정보공학과 졸업(학사)  
1997년 제주대학교 대학원 정보공학과 졸업(석사)

2005년 제주대학교 대학원 정보공학과 졸업(박사)  
2000~현재 제주대학교 시간강사  
2005~현재 제주대학교 첨단기술연구소 연구원  
※ 관심분야 : XML, Internet Application, Ubiquitous Sensor Network, Semantic Web 등

#### 곽 호 영(Ho-Young Kwak)



1983년 홍익대학교 전자계산학과 졸업(학사)  
1985년 홍익대학교 대학원 전자계산학과 졸업(석사)

1991년 홍익대학교 대학원 전자계산학과 졸업(박사)  
1990~현재 제주대학교 통신컴퓨터공학부 교수  
※ 관심분야: 객체지향 프로그래밍, 프로그래밍 언어론, Internet Application 등