# 분산 환경에서의 관계형 데이터베이스를 위한 XML-RPC 인터페이스 설계 및 구현

신 성 욱† · 이 영 욱††


## 요 약

RPC는 분산 시스템에서 가장 많이 활용되고 있는 통신 메커니즘 중의 하나이다. 본 연구는 현재 많이 활용되고 있는 RPC 프로토콜 중, XML-RPC를 이용하여 분산 환경하에서 데이터베이스를 활용하기 위한 단일화된 인터페이스 설계에 목적을 둔다. 기존의 RDB를 이용한 프로그램에 있어서 데이터베이스의 접근은 프로그래밍 언어, 플랫폼, 사용 데이터베이스의 종류 등에 따라서 특정한 API를 사용해야만 하는 프로그래밍 형태를 취했으나, XML-RPC 프로토콜을 이용하여 구현한 미들웨어를 둠으로써 데이터베이스 접근 인터페이스를 단일화하여 특정 언어와 환경에서 여러 상이한 데이터베이스에 대한 접근 및 조작을 용이하고 편리하게 하였다.


# Design and Implementation of XML-RPC Interface for Relational Database on Distributed Environments

Sung-Wook Shin† · Young-Wook Lee††


## ABSTRACT

The RPC is one of the most used communication mechanisms in the distributed system. This study is to design the simple interface for making use of database under the distributed environments by using XML-RPC among the RPC protocols. The access to any database of the current RDB programs is needed for a programming format of the special API according to the programming language, flat form and the class of the used database but simplification of database access shows that various different database accesses and manipulation are easier and more convenient in the special computer languages and environments by the middleware implemented using a XML-RPC protocol.


**키워드 : XML-RPC, 데이터베이스(Database)**

## 1. 서 론

분산 시스템 환경은 여러 개의 컴퓨터 환경과 통 신망 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[1, 2]. 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. RPC에서는 클라이언트에서 서버 프로세서에 대한 서비스 요구를 마치 클라이언트에서 일반적인 호출(Call)프로세서와 같이 동시적인 개념으로 구성되는 특성을 갖고 있다[1, 9].

오늘날 대형 기종에서 처리하던 데이터베이스 관리, 트랜잭션 처리, 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(Network Information Service) 그리고 상용화된 데이터베이스 관리 시스템인 MySQL, Postgres, Oracle, Sybase, Microsoft ODBC 등 다양한 분야에서 사용

† 준 회 원 : SALT IT(주) 응용개발실 연구원
†† 정 회 원 : 세명대학교 컴퓨터과학과 교수
논문접수 : 2002년 11월 25일, 심사완료 : 2003년 4월 3일

되고 있다[1, 7].

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.

하지만 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 이런 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 환경을 내부적으로 지원할 수 있도록 확장하여, 클라이언트/서버 프로그램의 작성이 중앙집중식 프로그램의 작성과 거의 동일(transparent)한 느낌을 줄 수 있도록 한다[2, 5, 8]. 본 논문에서는 이러한 클라이언트/서버 환경하의 프로그램에서 RDB(Relational Database)를 보다 편리하게 사용할 수 있도록 함에 목적을 둔다. RPC로는 HTTP(Hyper Text Transfer Protocol) over XML 구조의 XML-RPC를 사용하여 클라이언트의 구현에 있어서 언어에 독립적인 특성을 가질 수 있으며, 미들웨어 형태의 XML-RPC 서버의 구현으로는 최근 관심이 집중되고 있는 객체지향 언어인 Python을 사용하여 플랫폼 독립적인 특성을 가질 수 있도록 설계하였다.

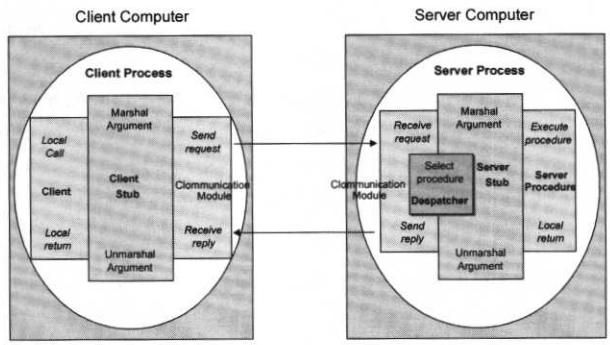
## 2. 관련 연구

RPC(Remote Procedure Call)는 분산형 컴퓨팅의 클라이언트 서버 모형을 수행하기 위하여 가장 많이 쓰이는 패러다임이다. RPC는 한 프로그램이 네트워크 상의 다른 컴퓨터에 위치하고 있는 프로그램에 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 프로그램이 원격 절차의 처리 결과가 반환될 때까지 일시정지 되어야 하는 동기 운영형태를 가진다. 그러나, 가벼운 프로세스의 사용이나, 같은 주소공간을 공유하는 스레드 등은 여러 개의 RPC들이 동시에 수행될 수 있도록 허용한다[3, 7].

RPC를 사용하는 프로그램 문장들이 실행 프로그램으로 컴파일될 때, 컴파일된 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 서버측에 변환해 주는 것이 주목적으로 다음과 같은 일련의 작업을 수행하게 된다.

먼저 파라메터들을 마셜링(marshalling)해서 프로시저 식별자로 묶어 전송 메시지에 싣게 되며 마셜링(marshalling)한 전송 메시지를 서버에 보내고 응답을 기다리게 된다. 그 후 (그림 1)과 같이 결과로 받은 메시지를 언마셜링(unmarshalling)하게 되는 역할을 수행한다. 즉, 스터브 프로시저는 파라메터 및 실행 결과를 메시지 형태로 정리하는 역할을 수행한다[2].

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(runtime) 프로그램과 원격절차 그 자신과 인터페이스를 하는 스터브을 포함한다. 처리 결과들은 같은 방식으로 되돌려진다.




(그림 1) 일반적인 RPC 호출 메카니즘

XML-RPC는 HTTP를 기반으로 하는 간단하고 이식성 높은 원격 프로시저 호출 방법이다[4]. 또한 XML-RPC는 분리된 네트워크 시스템간에 밀접한 관계를 맺도록 하는 일반적인 설정이 가능하다는 장점을 가지고 있으며, Perl, Java, ASP, Python, C, C++, PHP, Microsoft.Net과 그 외 다른 많은 언어로 사용할 수 있고, Unix, Windows, 그리고 Macintosh에서 실행이 가능한 프로토콜로써 하부 구조로 HTTP 프로토콜을 사용함으로써 기존의 분산 컴퓨팅 기술이 안고 있던 방화벽(firewall)이나 프록시(proxy) 서버와 관련된 문제점을 해결할 수 있다[8].

XML-RPC는 단방향 통신만을 지원하여 상태유지가 곤란하며, 인증에 관련된 특정한 표준이 존재하지 않는 등의 단점을 가지고 있으나 본 구현 시스템에 있어서는 타 프로토

콜보다 적합한 형태를 취하고 있다.

RPC는 이기종간의 컴퓨터에서 이루어질 수 있기 때문에, 파라미터 전송 방법은 좀 더 일반화된 형태를 취해야 한다. 여러 가지 구성방법이 있겠지만, XML-RPC는 인수나 계산 결과를 주고받기 위해서 XML 문서 형식을 사용한다[11]. XML은 문서 교환 형식의 표준으로 자리를 잡아가고 있다. 이렇게 함으로써 기종이나 플랫폼에 상관없이 표준화된 문서로 정보를 교환할 수 있다. RPC는 네트워크 상의 다른 프로세스와 통신하기 위하여 프로시저 호출의 형태를 사용하므로 RPC가 구현되어지는 프로그램 언어의 성격 및 사용하는 RPC 프로토콜에 따라서 실제적인 구조가 달라질 수 있다.

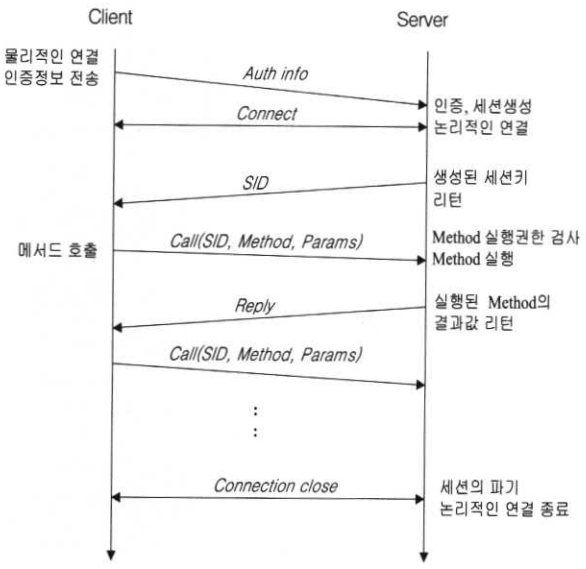
논문에서는 구현 언어로써 객체지향 언어인 Python을 사용하여 언어 및 플랫폼에 독립적인 형태의 RPC 미들웨어를 구현하였다. Python은 국내에서보다는 외국에서 널리 활용되는 언어로써 사용상의 단순함과 그러면서도 객체 지향적인 성격을 띄는 언어이다. 또한 제공되는 프리미티브 데이터 타입이 문자열 및 데이터베이스 구조와 흡사하여 데이터베이스를 다루는 프로그램을 작성할 경우 매우 유용하다[6, 10].

## 3. 관계형 데이터베이스를 위한 RPC 시스템의 구조

현재의 데이터베이스를 이용한 프로그래밍에 있어서 데이터베이스로의 접근은 프로그래밍 언어와 플랫폼, 사용 데이터베이스에 따라 특정한 API를 사용해야만 했다.

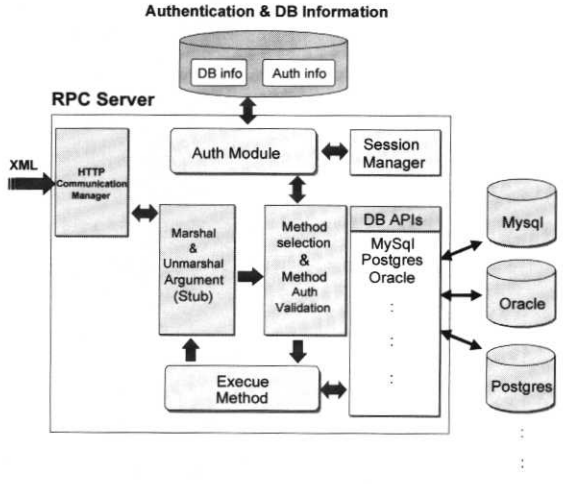
이러한 형태의 프로그래밍에 있어서는 사용 데이터베이스와 언어에 맞는 API 지식 습득, 프로그램 코드의 증가 등 많은 불편이 있었다. 이러한 프로그래밍상의 어려움을 해결하기 위해 RPC를 이용한 미들웨어를 두어 어려움을 해결하고자 하였다.

구현되어진 서버는 클라이언트와 또는 어플리케이션 서버와의 RPC를 수행하게 되는데 그 과정은 최초 클라이언트 측에서의 물리적 연결로 시작이 된다. 클라이언트는 물리적 연결에 성공하게 되면 인증과정을 거치게 된다. 인증과정이 성공적으로 이루어지게 되면 서버측에서는 인증 정보를 이용하여 세션을 생성하게 되고 생성된 세션키를 클라이언트 측으로 넘겨줌으로써 논리적인 연결을 유지하게 된다. 그후 클라이언트 측에서는 Call(method, params)을 통해 서버측에 해당 메소드 호출을 요청하게 되며 서버측에서는 이러한 클라이언트의 요청에 따라 세션키를 이용한 권한 검사를 수행, 권한이 허가된 메소드라면 이를 수행하고 클라이언트 측에 메소드의 수행결과를 넘겨 주게 된다. (그림 2)는 클라이언트와 서버측의 RPC 통신에 있어서 통신절차를 나타낸 그림으로 논리 적인 세션유지를 보여주고 있다.




(그림 2) 논리적인 RPC 통신 절차

제안된 시스템의 전체적인 구조는 다음 (그림 3)과 같다. 네트워크를 통해 전달되어진 XML 형태의 메시지는 통신 관리자를 통해 전달되어지며 전달되어진 메시지는 마셜링(marshalling)과정을 거침으로 서버측에서 인지할 수 있는 형태로 변환되어진다. 변환되어진 파라메터들과 메소드 이름에 의해 서버측의 메소드가 실행되고 이를 다시 언마셜링(unmarshalling)하여 클라이언트로 전송되어진다. 서버 내부 모듈은 기능에 따라 통신 및 연결유지, 데이터의 변환 등을 위한 통신 모듈, 스터브 등의 통신용 내부모듈과 인증 및 기초 데이터 유지 및 관리를 위한 관리 모듈, 데이터 베이스 연결 및 조작을 위한 모듈로 실제적인 서비스 메소드의 집합모듈로 구분할 수 있다. 또한 서비스 모듈에 있어서는 권한에 의한 구분으로 일반 사용자를 위한 메소드와 관리자를 위한 메소드로 구분되어질 수 있다.




(그림 3) XML-RPC 서버의 동작 구조

### 3.1 인 증

구현된 RPC 서버는 미들웨어로써 비즈니스 계층에 존재하게 된다. 사용자측(클라이언트)에서는 최초로 인증과정을 통한 서버로의 접근 시도시 서버측에서는 사용자가 정상적인 권한을 가지고 있는지를 검사하게 되며, 적정한 사용자로 판단되면 세션을 생성하여 논리적인 연결을 유지하게 된다. 사용자는 일반사용자와 슈퍼유저 권한을 가진 사용자로 나뉘어질 수 있으며 슈퍼 유저 권한을 가진 사용자는 일반사용자가 수행할 수 있는 메소드 호출 외에 일반사용자의 등록 및 삭제, 강제적인 세션의 종료, 검색 등의 관리 기능을 수행할 수 있다.

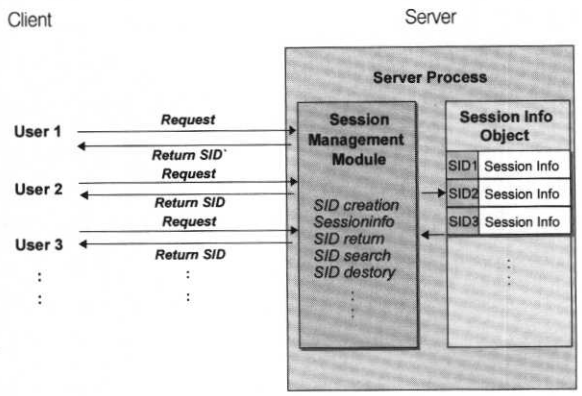
사용자의 관리는 인증과 사용자 정보를 관리하기 위한 데이터베이스에서 따로 저장, 관리가 이루어지게 된다. 이러한 기초정보 데이터베이스에는 사용자 인증정보와 사용자관리 데이터베이스 정보를 기록할 수 있는 데이터베이스 관리 테이블이 존재하게 된다. 사용자의 인증에 있어서 사용자의 패스워드는 128비트의 문자열 압축 기술인 md5과정과 단방향 DES 알고리즘을 적용하여 암호화되어 저장되고 인증 과정은 사용자 정보를 입력받아 이와 똑같은 방법으로 암호화 한 값을 비교하여 이루어지게 된다.

### 3.2 연결과 상태유지

구현된 미들웨어는 HTTP over XML 형태를 사용한다. 따라서 물리적인 연결은 매 호출시마다 이루어지고 서버측에서 이러한 호출에 대한 응답이 이루어진 후 서버측에 의해 강제적으로 연결이 종료된다. 따라서 상태유지를 위해서는 논리적인 연결상태의 유지가 필요한데 이를 위해 세션을 사용하게 된다.

각 사용자의 상태유지를 위한 세션키의 생성은 사용자 아이디, 패스워드 및 현재 시간의 문자열 조합을 md5 방식으로 압축한 문자열로 이루어지며 사용자가 최초 인증과정을 거치게 될 때 키 값이 사용자에게 넘겨지게 되고 논리적인 연결이 이루어진다. 서버측에서는 이러한 세션을 유지하기 위한 특별한 객체를 두어 세션에 관한 정보를 저장관리하게 된다. 세션 정보로는 사용자 정보를 이용하여 생성된 세션키, 세션키가 생성된 시각, 세션 종료 시간으로 이루어지게 되며, 세션의 유지는 사용자 인증시 세션 정보가 있는 객체의 정보를 검사하여 세션의 유지 여부를 결정하게 된다.

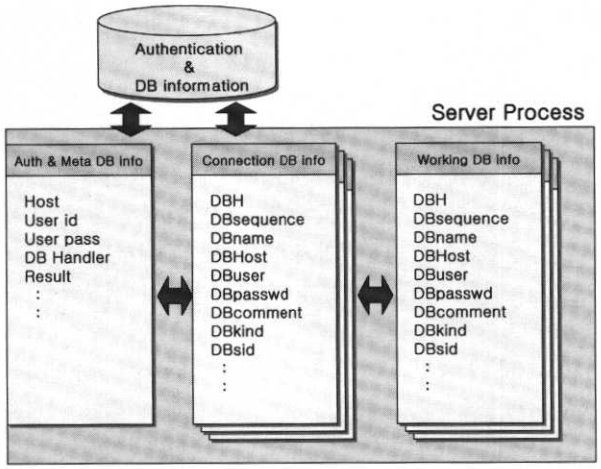
세션의 유지는 사용자의 입력에의한 지속시간을 기본으로 하여 매 메소드의 호출시 세션의 종료 시간과 비교하여 세션의 파기여부를 결정하게 된다. 사용자에 의하여 입력된 세션 만료 시간전에 호출이 일어날 경우 특정한 시간동안 세션을 유지하게 되며, 만료 시간동안 호출이 일어나지 않을 경우 세션이 자동 파기되며 사용자 및 데이터베이스와의 연결이 자동 종료된다.




(그림 4) 상태유지를 위한 세션

### 3.3 클라이언트 정보유지를 위한 자료 구조

호출된 메소드의 실행과 인증에 관한 정보는 서버 기동시 데이터베이스로부터 읽혀져 서버측의 객체로 유지되며 필요시 이러한 정보를 이용하여 인증을 수행하게 된다. 따라서 인증, 메소드 호출시에 사용되어질 정보는 매 호출시마다 데이터베이스로부터 받아오는 것이 아니라 기초 정보 유지를 위한 객체로부터 얻게 되며 데이터베이스 정보 갱신, 사용자 추가 등 사용자에 의한 특정 메소드 호출시마다 데이터베이스와 객체간의 무결성 유지를 위한 정보 교환이 이루어지게 된다. 이러한 정보 관리 객체는 3부분으로 이루어져 있는데 기초 정보객체, 연결 데이터베이스 정보객체, 작업 데이터베이스 정보객체가 있다.




(그림 5) 정보저장 객체 구조

기초 정보객체는 사용자 인증 정보와 사용자 정보를 데이터베이스로부터 전달받아 저장하고 있는 객체로 사용자에 의해서 등록된 데이터베이스에 관한 정보 및 ID, 암호화된

패스워드 등이 저장되어 있으며 시스템에서 사용되는 모든 메소드에서 접근이 가능하며 이 정보를 기본으로 모든 인증과 메소드의 실행이 이루어진다.

연결 데이터베이스 정보객체는 사용자 SID(Session ID)와 연결 데이터베이스 별로 생성이 되며 자료구조의 논리적인 모습이 마치 데이터베이스의 테이블과 같은 형태로 이루어져 있다. SID를 키로 하고 연결 데이터베이스 핸들러(handler), 이름, 데이터베이스가 위치한 호스트, 데이터베이스 연결 ID, 패스워드 등의 정보가 저장되어 있다. 자료의 접근은 SID와 저장 정보이름의 쌍으로 이루어지며 이름으로 직접 검색이 가능하다. 이 정보는 기초정보 데이터베이스의 정보를 바탕으로 생성되어지며 사용자에 의해 새로운 데이터베이스가 연결될 경우 SID와 연결할 데이터베이스를 조합하여 중복되는 자료가 없는 경우에 생성하게 된다.

작업 데이터베이스 정보객체는 연결 데이터베이스 정보를 바탕으로 생성되어지며 SID 별로 각각 1개씩이 생성되어지고 또한 테이블과 같은 형태로 저장된다. 이 객체는 현재 연결이 유지된 사용자가 실제로 작업을 수행하는 데이터베이스의 정보를 저장하게 된다.

### 3.4 통신 하부구조 및 스터브

앞서도 언급했듯이 본 논문에 사용되는 통신 하부구조는 HTTP를 사용하여 구현되었다. 또한 메시지의 전달형태를 XML 형태를 취하여 전송된다. 이러한 형태로 전송되는 메시지를 서버측에서 사용하기 위해 변환해 주는 과정이 필요한데 이러한 부분을 담당하는 개체가 스터브(stub)이다.

스터브는 클라이언트측에서 보내오는 메시지의 언마셜링(unmarshalling) 뿐만 아니라 서버측에서의 결과 값을 클라이언트에게 전송시에도 마셜링(marshalling)을 수행하여 정보를 XML 문서의 형태로 변환해 주는 역할을 수행하게 된다.

서버측에 도착한 요청문은 해석되고 메소드 수행에 대한 인증을 거친 후 수행된 결과가 클라이언트 측으로 전달되어질 형태의 XML 문서로 마셜링(marshalling)된다. 이렇게 클라이언트로 전달된 XML 문서는 서버측에서와 마찬가지로 언마셜링(unmarshalling) 과정을 수행하게 됨으로써 클라이언트 측에서 필요로 하는 형태의 자료로 변환되게 된다. 다음은 질의된 'test(1)'이라는 가상의 메소드를 호출했을 경우 마셜링된 형태이다.

```
<?xml version = "1.0"?>
<methodCall>
<methodName> test </methodName>
<params>
<param>
<value> <int> 1 </int></value>
</param>
</params>
</methodCall>
```

다음은 서버측에 도착한 요청문이 해석되고 메소드 수행에 대한 인증을 거친 후 수행된 결과가 클라이언트 측으로 전달되어질 형태의 XML 문서로 마셜링(marshalling)된 모습이다. 이렇게 클라이언트로 전달된 XML 문서는 서버측에서와 마찬가지로 언마셜링(unmarshalling)과정을 수행하게 됨으로써 클라이언트 측에서 필요로 하는 형태의 자료로 변환되게 된다.

```
<?xml version = '1.0'?>
<methodResponse>
<params> <param>
<value> <array> <data>
<value> <struct>
<member>
<name> dbinfo.id </name>
<value> <int> 1 </int> </value>
</member>
        :
      생략
        :
<member>
<name> dbinfo.host </name>
<value> <string> apollo.semyung.ac.kr </string> </value>
</member>
</struct> </value>
<value> <struct>
```

### 3.5 Method 실행 및 DB APIs

Method의 호출은 앞서 설명한 정보 유지 객체의 정보를 기초로 이루어지게 된다. 메소드의 실행은 서버측의 호출 메소드를 호출함으로써 이루어지게 되는데 호출 메소드는 method와 params를 인수로 받는다. method는 클라이언트에서 요구한 메소드 이름의 문자열이고, params는 클라이언트가 전달한 인수로 튜플(tuple)로 묶여있다. 중요한 것은 메소드 문자열 이름으로부터 실제의 메소드 레퍼런스를 얻는 것이다. 이 레퍼런스를 이용하여 메소드를 직접 호출할 수 있다.

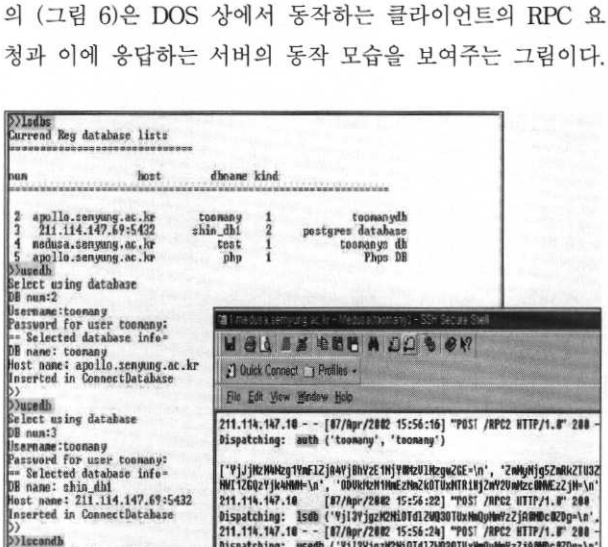
제안된 미들웨어를 이용한 데이터베이스 질의는 최초 클라이언트 측에서 RPC 서버 측으로 연결이 이루어지게 되며, 연결이 완료되면 사용자 인증과정을 통해 SID를 넘겨받게 된다. 사용자 정보는 SID를 이용하여 검색하게 되며 객체형태로 유지되어진다. 내부적으로 이러한 정보를 이용하여 사용하고자 하는 데이터베이스를 선택하여 질의 연산을 수행하게 된다. 다음은 제안된 시스템을 이용하는 파이선 클라이언트의 간단한 질의 과정이다.

```
import xmlrpclib
querystr = "select * from test"
con = connectserver (host,port)
SID = con.login (username, password)
con.selectDB (SID, database_name)
result = con.query (SID, querystr)
```

구현되어진 각각의 데이터베이스 연산 명령은 모듈 형태의 독립적인 파일로 구성되었다. 이러한 구성은 차후 새로운 데이터베이스의 추가 및 기존 데이터베이스 연산 함수의 추가 등에 유연성을 향상시켜 줄 수 있다.

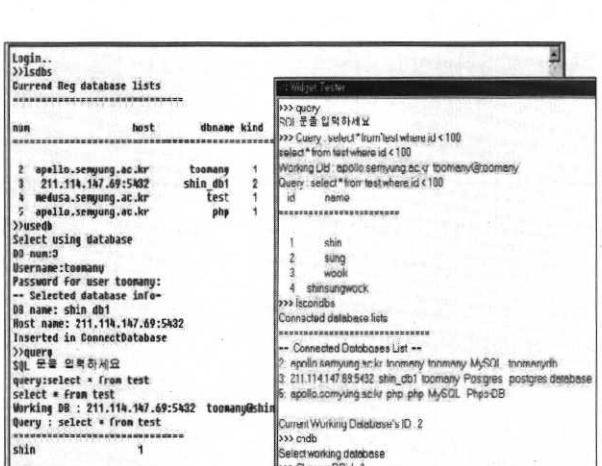
## 4. RPC 시스템 고찰 및 활용

서버측에서는 클라이언트에 의해 처리되는 일련의 과정을 실시간으로 확인하여 로그형태로 저장할 수 있다. 다음의 (그림 6)은 DOS 상에서 동작하는 클라이언트의 RPC 요청과 이에 응답하는 서버의 동작 모습을 보여주는 그림이다.



(그림 6) 클라이언트의 요청에 대한 RPC 서버의 동작

클라이언트의 요청에 의해 서버는 접근한 클라이언트의 정보를 기록하며 관리자는 이러한 기록을 이용해 클라이언트의 접근현황과 정보를 파악할 수 있다. 기록되는 정보로는 접근 클라이언트의 IP구조, 접근 시각, 사용한 메소드 및 메소드 호출을 위한 파라메터 정보 등을 알 수 있다. 구현되어진 클라이언트는 데이터베이스에 대한 기본적인 연산을 취급하도록 설계되었으며 동시에 여러 개의 데이터베이스로의 접근이 가능하며 사용자에 의해 현재 접속되어 있는 데이터베이스를 변경해가며 작업을 수행할 수 있도록 설계되었다. (그림 7)은 위와 같은 기능을 수행하는 Shell로 wxPython 패키지를 이용하여 Win32용으로 제작한 클라이언트의 모습과 Unix 상에서 동작하는 Shell을 보여준다.



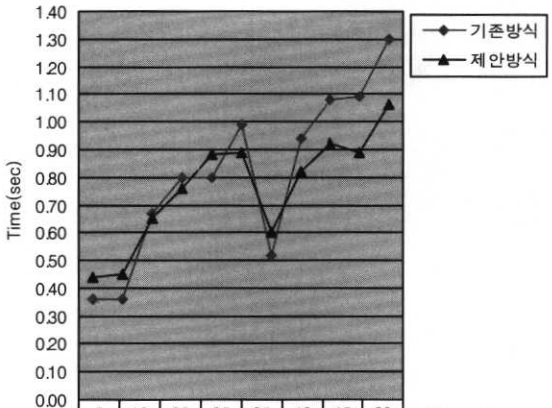
(그림 7) wxPython을 이용한 Shell과 Unix 상에서 동작하는 Shell

이 밖에도 테스트를 위해 C언어로 작성한 데이터베이스 관리 쉘(shell)과 php를 이용한 웹 상의 쇼핑몰에서 활용 여부를 판단해 보았다. 결과 모든 플랫폼과 언어에서 완벽하게 동작했으며 속도 면에서도 독립적인 전용 Shell이나 연결을 사용할 때와 차이를 거의 느끼지 못하였다.

(그림 8)은 기존의 API를 이용하여 직접 연결을 시도한 결과와 구현된 서버를 이용하여 데이터베이스를 조작한 결과를 나타낸 그림이다.

사용환경은 128MB 메모리를 갖는 펜티엄Ⅲ 600Mhz 컴퓨터와 MySQL을 사용하였다. 데이터베이스는 10,000개의 튜플(tuple)을 갖는 하나의 테이블을 대상으로 모든 튜플에 대한 Select 연산으로 실험하였으며, 다수 사용자가 접근하여 작업을 수행할시의 성능평가를 위하여 동시에 다수개의 스레드(thread)를 발생시켜 결과 값이 도착하는 시간의 평균값을 계산하여 그래프를 작성하였다.

(그림 8)에서 보는바와 같이 둘 사이의 성능 차는 스레드의 수가 적을 경우 직접 API를 호출하여 연산을 수행하는 쪽이 약간 빠르게 나타났고, 스레드의 수가 증가하면서 제안된 방식이 기존의 방식보다 빠른 응답 시간을 가짐을 알 수 있다. 기존 방식의 질의에 있어서 데이터베이스로의 연결과 종료가 매회 이루어지는 반면에 제안된 방식은 데이터베이스와 연결이 이루어진 후 사용자에 의한 명시적인 종료나 타임 아웃이 발생하지 않는한 연결이 유지되며, 동시에 여러 데이터베이스와의 연결을 유지하기 때문에 일정한 수 이상의 질의가 발생하였을 경우 기존의 방식에 비해 빠른 응답을 보여주고 있다.




| | 5 | 10 | 20 | 25 | 35 | 40 | 45 | 60 |
|---|---|---|---|---|---|---|---|---|
| 기존방식 | 0.36 | 0.36 | 0.80 | 0.80 | 0.52 | 0.94 | 1.08 | 1.30 |
| 제안방식 | 0.44 | 0.45 | 0.76 | 0.88 | 0.60 | 0.82 | 0.92 | 1.06 |

(그림 8) 구현된 서버와의 성능비교

## 5. 결론 및 향후 연구방안

본 연구의 목적은 지역적으로 분산되어진 환경에 존재하는 이기종 데이터베이스를 이용한 프로그래밍에 있어서 또는 이러한 분산 환경의 데이터베이스를 관리하기 위한 클라이언트를 프로그래밍 함에 있어서 보다 쉽고 빠른 프로그래밍이 가능하도록 단일화된 인터페이스를 제공함으로써 프로그래밍의 효율성을 제공함에 있다.

본 논문에서는 첫째, 미들웨어라는 형태를 띄는 XML-RPC 프로토콜을 이용한 RPC를 이용함으로써 상이한 프로그래밍 언어에 있어서 접근에 투명성을 주며, 객체지향 언어인 Python을 이용함으로써 플랫폼에 독립적인 RPC 서버를 구현하였다. 둘째, 객체 지향 언어의 특징으로는 재사용성과 캡슐화를 들 수 있는데, 본 논문에서 구현된 시스템은 이러한 객체지향 언어의 특징을 살려 각각의 데이터베이스와 관련된 연산을 수행하는 부분들을 각각의 모듈로 구성하여 차후 지원 데이터베이스 추가에 대한 확장성을 높였다.

셋째, 다수의 데이터베이스를 연결하여 동시에 작업이 가능하도록 데이터베이스에 대한 핸들러 관리객체를 둠으로써 상이한 데이터베이스의 작업에 있어서 또는 같은 종류의 다른 데이터베이스로의 작업에 있어서 매 작업시 연결과 종료를 수행하는 것이 아니라, 연결된 핸들러를 이용하여 간단한 작업 데이터베이스 전환 메소드 호출만을 통해 작업 수행이 가능하도록 구성하여 이종의 데이터베이스 작업시 연결과정에서 발생하는 오버헤드를 줄였다.

넷째, HTTP over XML 기반의 통신 환경의 단점을 보안하기 위하여 세션모듈을 구성, 상태유지기능을 부여하였다. 또한 클라이언트와 서버사이의 상태유지에 있어서 세션을 사용했으나 일반적인 세션이 파일을 이용한 정보 저장 방법을 사용하는 반면, 본 논문에서 구현한 세션은 서버측에 세션 관리를 위한 모듈과 세션정보 저장 및 관리 객체를 두어 세션정보의 확인시 드는 비용을 줄일 수 있었다.

다섯째, 사용자의 편의를 위해 데이터베이스 정보를 저장할 수 있는 메타 데이터베이스를 구성하여 데이터베이스 관리 쉘과 같은 프로그램 작성시 사용자가 매번 같은 데이터베이스에 대한 정보를 입력하는 불편을 덜어 데이터베이스 인덱스 번호와 데이터베이스 접근 아이디와 패스워드만을 가지고 접근이 가능하며, 사용자에 의해 삽입, 삭제, 편집이 가능하도록 설계하였다. 또한 관리자 모드를 두어 관리자는 현재 상태의 세션 열람, 세션 파기 등의 작업을 가능하게 하여 부당한 사용자나, 트래픽 관리(traffic management) 기능을 부가하였다. 또한 관리자에 의해 사용자의 추가 및 삭제가 가능하도록 설계하였다.

본 시스템에서는 적용 데이터베이스를 MySQL, Postgres, Oracle과 같은 관계형 데이터베이스만 고려하였으며, 구현된 호출 메소드들에 의해 데이터베이스의 모든 기능을 제어하기보다는 프로그래밍에 있어서 활용가치가 높은 질의 연산관련 기능 위주로 구성하였다. 차후 이러한 관계형 데이터베이스만이 아닌 OODB, ORDB에도 적용이 가능한 형태로의 연구 및 데이터베이스를 제어하기 위한보다 많은 연구가 필요하다.

## 참 고 문 헌

[1] 이종근, 이창석, 이광휘, "시스템 지원을 위한 그룹 RPC 프로토콜의 설계와 검증", 창원대학교, 1996.

[2] John Bloomer, "Power Programming with RPC," O'Reilly & Associates, 1991.

[3] Sean McGrath, "XML Processing with Python," Prentice Hall PTR, 2000.

[4] XMLRPC How To <URL : http://xmlrpc-c.sourceforge.net>.

[5] M. D. Abram, "Transparent Remote Procedure Calls," Master's Thesis, Computer and Information Science, University of California Santa Cruz, Dec., 1992.

[6] Python <URL : http://python.or.kr>.

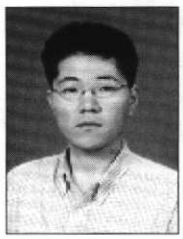
[7] George Coulouris 외 2인, "Distributed Systems Concepts and Design," 2nd Ed., Addison Wesley, 1998.

[8] 한만수 역, "XML-RPC 프로그래밍", 한빛 미디어, 2001.

[9] A. D. Brirell and B. J. Nelson "Implementing Remote Procedure Calls," ACM Trans. Compyut. Syst., Vol.2, No.1, pp. 39-59, Feb., 1984.

[10] Python C reference, <URL : http://python.org/doc/current/api/api.html>.

[11] XML-RPC, <URL : http://www.xmlrpc.com/spec>.




## 신 성 욱

e-mail : toomany@raonbiz.com

2000년 세명대학교 컴퓨터과 학과(학사)

2003년 세명대학교 대학원 전산정보학과 (석사)

2003년~현재 SALT IT(주) 응용개발실 연구원

관심분야 : WAP, Database, 성능평가 등



## 이 영 욱

e-mail : ywl-7562@semyung.ac.kr

1980년 서울대학교 공과대학원 전자공학 (석사)

1993년 텍사스 에이앤엠 대학원(미) 전자공학(박사)

1975년~1977년 동양정밀(주) 기술연구부

1977년~1987년 국방과학연구소 연구개발실

1990년~1993년 텍사스 에이앤엠 대학교(미) 연구조교

1994년~현재 세명대학교 컴퓨터과학과 부교수

관심분야 : 데이터베이스, 인터페이스 설계 등