석사 학위 논문

클라이언트/서버 시스템의 2 계층 구조에서 3 계층 구조로의 이행 방법

박 용 진 (朴 龍 鎭)

정보통신학과

포항공과대학교 정보통신대학원

1998

클라이언트/서버 시스템의 2 계층 구조에서 3 계층 구조로의 이행 방법

A Migration Methodology for 2-tier to 3-tier Architectured Client/Server Systems

A Migration Methodology for 2-tier to 3-tier Architectured Client/Server Systems

by

Yong-Jin Park

Department of Computer and Communications Engineering

POSTECH Graduate School of Information Technology

A thesis submitted to the faculty of POSTECH Graduate School of Information Technology in partial fulfillment of the requirements for the degree of Master of Engineering in the Department of Computer and Communications Engineering.

Pohang, Korea

December 19, 1997

Approved by

_____

Major Advisor

클라이언트/서버 시스템의 2 계층 구조에서 3 계층 구조로의 이행 방법

박 용 진

위 논문은 포항공과대학교 정보통신대학원 석사학위논문으로 학위논문 심사위원회를 통과하였음을 인정함.

1997 년 12 월 19 일

학위논문심사 위원회	위원장	홍원기 (인)
	위원	강교철 (인)
	위원	김정현 (인)

MCC
9651M14

박 용진, Yong-Jin Park, A Migration Methodology for 2-tier to 3-tier Architectured Client/Server Systems, **클라이언트/서버 시스템의 2 계층 구조에서 3 계층 구조로의 이행 방법**, **정보통신학과**, 1998, 51P, Advisor: Won-Ki Hong, Text in Korean.

ABSTRACT

Advances in computing and network technologies along with the changes in the way businesses operate influenced the introduction of client/server systems. These early client/server systems had 2-tier architecture where application logic and presentation logic were tightly coupled with specific database servers. These 2-tiered client/server systems were found to possess several major problems. First, these systems were not easily scalable as the number of users increased and thus were faced with severe performance degradations. Second, the systems could not easily cope with changes to business logic or environment. Third, since the applications were tightly coupled with specific databases, the systems could not be easily upgraded or changed without major efforts.

In order to solve the problems in 2-tier client/server systems, 3-tier client/server systems have been proposed. This thesis presents a simple but efficient methodology for migrating 2-tier to 3-tier client/server systems. Both 2-tier and 3-tier architectured client/server systems are thoroughly examined and compared. The client/server systems at POSCO have been analyzed and were determined to be mostly 2-tier architectured. As a proof of concept, the migration methodology has been applied to migrating 2-tier client/server systems at POSCO to 3-tier client/server systems.

목 차

그림 목차

표 목차

1. 서 론

1.1 연구 배경

최근 정보산업의 변화 속도는 통신과 컴퓨터의 결합을 통해 예측할 수 없을 정도로 급변하고 있다. 이러한 변화는 기존의 대형 컴퓨터를 중심으로 한 중앙 집중식 방식에서 중형컴퓨터를 중심으로 한 다운사이징 (Downsizing), 라이트사이징 (Rightsizing), 업사이징 (Upsizing) 등 클라이언트/서버 방식으로의 전환을 가져 왔다. 그리고 비지니스 및 고객 요구 형태가 변화함에 따라 보다 발전된 클라이언트/서버 컴퓨팅의 필요성이 대두되었다.

이러한 필요성에 의해 등장한 클라이언트/서버 시스템 형태는 대부분 2 계층 구조 (2-tier Architecture)로서 사용자 인터페이스 (User interface) 부분과 업무로직 (Business Logic)부분을 클라이언트에 두고 중앙서버에는 데이타베이스 (database)를 두는 형태이다. 이러한 2 계층 형태의 장점은 트랜잭션 (transaction) 부하가 가벼운 부서단위의 소규모 시스템 구축시 쉽고 보편적인 툴을 사용하여 개발할 수 있다는 점이다.

그러나 분산처리 환경이 기업 전반에 확산되고 있는 현 시점에서 이 모델은 여러가지 문제점을 초래했다. 첫째, 클라이언트/서버간에 네트웍 통신량이 많아 시스템의 성능저하와 네트웍 병목현상이 발생했다. 둘째, 애플리케이션의 논리적 구조와 물리적 구조가 완전하게 분리 되지 않아 비지니스 프로세스 (business processes)나 비지니스 환경 (business environment)이 변경될 때 마다 비지니스 로직이 바뀌어야 한다. 따라서 많은 부분의 소프트웨어가 수정되어야 한다. 셋째, 대부분의 애플리케이션이 특정 RDBMS 에 종속되어 데이타나 애플리케이션이 증가되었을 때 통합하기가 어렵다는 확장성의 한계이다. 특히 많은 legacy data sources 를 액세스 하는데 많은 제약조건이 있으며, 비지니스 룰과 프로세스가 분명히 구분되지 않고 개발되기 때문에 애플리케이션 개발과 적용에 있어서 유연성 (flexibility)이 떨어지고, 광범위하게 흩어진 애플리케이션을 확대하거나 유지하기가 어렵다. 넷째, 현재의 애플리케이션 로직이 프리젠테이션 로직에 포함되어 있어 융통성을 그대로 보장하지 못한다는 것이다. 일단 애플리케이션이

개발되면 한 서버에서 다른 서버로 프로그램 기능의 일부를 이동하는 것이 어렵다.

이러한 문제점을 해결하기 위한 효율적인 시스템 아키텍처 수립은 현재 파악된 기술에 제한을 두지 않고 계속 발전하는 기술과 향후 시스템의 통합이나 유지보수에 문제가 없도록 설계되어야 하며, 유연성 있는 이기종 환경을 하나로 연결하여 투명한 사용 환경을 사용자에게 제공하는 것은 클라이언트/서버 시스템의 중요한 과제이다.

1.2 연구 목표 및 글의 구성

현재 구축 운영되고 있는 대부분의 클라이언트/서버 시스템이 2 계층 구조로서 시스템이 확대되면서 이미 많은 문제점들이 드러 나고 있다. 애플리케이션을 분명히 구분하는 3 계층 클라이언트/서버 시스템의 출현은 많은 기업들의 관심사이다.

이에 본 연구에서는 2 계층 형태의 클라이언트/서버 시스템의 문제점에 대한 해결책으로 3 계층 구조의 클라이언트/서버 시스템 구현방안에 대해서 알아 보고, 실제 포스코에서 구축 운영중인 클라이언트/서버 시스템에 대한 현상을 분석하여 문제점을 도출하고, 3 계층 구조의 클라이언트/서버 시스템을 적용하기 위한 방법을 제시해 보겠다.

1 장은 서론으로 연구 배경과 목표를 소개하였고, 2 장은 본 논문 연구에 필요한 클라이언트/서버 시스템 아키텍처에 대한 종류 및 장단점을 비교하고, 클라이언트/서버의 핵심 기술인 미들웨어에 대한 기능 및 원리, 종류별 특징, 대표적인 객체지향 미들웨어인 코바 (CORBA)에 관한 내용과 기업에서 3 계층 구현사례를 살펴보았다. 3 장에서는 3 계층 구조의 클라이언트/서버 시스템 구현 방안에 대해서 살펴보았으며, 4 장에서는 기업에서 실제 구축 적용되고 있는 현재 클라이언트/서버 시스템에 대한 현상을 분석하고 문제점을 정리하여, 5 장에서는 이를 토대로 실제 기업에서의 적용 방안을 제시하였다. 마지막 6 장에서는 본 논문의 결론으로서 향후 추진되어야 할 내용을 중심으로 기술하였다.

2. 관련 연구

이 장에서는 효율적인 3 계층 구조의 클라이언트/서버 시스템 구축을 의해 필요로 하는 주요 관련 연구로서 클라이언트/서버 시스템, 미들웨어, 대표적인 객체지향 미들웨어인 코바 (CORBA), 3 계층 구조의 클라이언트/서버 시스템의 구축 사례에 대해 살펴본다.

2.1 클라이언트/서버 시스템

분산시스템에 대한 정의와 출현배경, 클라이언트/서버 시스템에 있어서 2 계층 구조과 3 계층 구조의 특징 및 장단점을 비교하고, 주요 적용업무에 대해서 서술하겠다.

2.1.1 분산시스템의 정의

분산시스템이란 통합된 컴퓨팅 기능을 제공하도록 고안된 소프트웨어로서 고유한 기능을 가진 컴퓨터들이 네트웍으로 서로 연결된 집합체로서, 시리얼 라인으로 연결된 터미날 (terminal)로 동기/비동기로 통신하던 기존의 메인프레임 (mainframe)과는 대조되는 계념이다 [1]. 예를 들면 Local Area Network (LAN)이나 Wide Area Network (WAN)으로 연결된 네트웍 시스템, 분산파일 공유 시스템 (Distributed File Shared System), 전자메일 (Electronic Mail), 클라이언트/서버 시스템 등이 모두 이에 속한다고 할 수 있다. 이러한 분산시스템의 출현배경은 다음과 같다 [2,3].

? 하드웨어 (Hardware)의 발전

데스크탑 (desktop)의 파워는 386 의 8 밉스 (MIPS)에서 486 의 80 밉스, Pentium 의 200 밉스 이상으로 기하급수적으로 증가하는 반면 호스트는 비용에 비해 더 늦은 비율로 증가해 왔다. 또한 인텔, 모토롤라, RISC 칩 등 다량생산의 칩을 장착한 서버가 기존의 메인프레임이나 중형컴퓨터보다 값싸고 강력한 성능을 발휘하고, 메모리 칩의 용량은 3 년마다 4 배로 증가하는 등

1986 년에 1MB 로 시작하여 1992 년에는 16MB, 1997 년에는 Giga Byte, 2000 년에는 Tera Byte칩이 나오게 될 것이다.

? 소프트웨어 (Software)의 발전

분산형 데이타를 지원하고 클라이언트/서버 애플리케이션을 위한 표준 데이타 접근 언어인 Structured Query Language (SQL)을 지원하는 Data Base Management System (DBMS)제품의 발달, 사용자들이 쉽게 배우고 사용할 수 있는 4th Generation Language (4GL) 개발 툴, Graphical User Interface (GUI) 기반의 사용자 인터페이스의 발달, 32 비트 구조로 체계적인 다중 처리 (Multi-processing) 하드웨어를 지원하는 다양하고 편리한 기능의 윈도우 OS 의 발달이 클라이언트/서버의 출현에 크게 기여하였다. 또한 최근 Java [4] 출현으로 Java 애플리케이션이 많이 개발되고 있는데 다양한 Operating System (OS)와 하드웨어 플랫폼에서 작동이 가능하다는 장점이 있고, Web brower 라는 통일된 사용자 인터페이스를 통해 쉽게 사용할 수 있어 앞으로의 소프트웨어 개발이 Java를 사용하는 것들이 많이 나오리라 예상된다.

? 네트웍 (Network)의 발전

클라이언트/서버 시스템의 구성요소 중 가장 중요한 것이 네트웍이다. 브릿지 (Bridge), 라우터 (Router), 게이트웨이 (Gateway)와 같은 접속장치의 발달로 LAN 이 빠른 속도로 발전하고 있다. 10Mbps Ethernet 에서 부터 100Mbps Fast Ethernet, 100Mbps FDDI, 155Mbps 이상의 Asynchronous Transfer Mode (ATM), 1000Mbps Gigabit Ethernet 등 LAN 기술의 급속한 성장으로 데이타 전송의 신속성 및 정확성으로 내부 조직간의 의사소통이 신속해 질 뿐만 아니라, 타부서, 타조직간의 의사전달 필요성 대두로 WAN 또한 빠른 속도로 발달하게 되었다. 대표적인 것으로 주로 미국을 중심으로 T1 급의 1.54Mbps 에서 T3 의 44.73Mbps, 유럽을 중심으로 E1 급 2.04Mbps 에서 E3 의 34.36Mbps 등 그 외에도 여러 종류의 WAN Backbone이 있다.

? 업무의 변화

기업경영의 국제화, 세계화, 고객지향 중심의 경영으로 다품종 소량생산 등의 경영방식변화로 신속한 의사결정과 대량의 데이타에 대한 공유 필요성

이 대두되고, 사용자의 생산성과 소비자의 서비스 개선을 요구하는 반면에 경비는 삭감하려는 노력이 진행중이다.

이러한 기술적인 발달이 중앙 집중형 컴퓨팅의 제약조건을 극복하고 이를 대체하려는 노력이 한창 진행되었으며 그 결과 수 많은 기술이 출현되었다. 그 중에서도 가장 대표적이고 효율적인 방법이 바로 클라이언트/서버 컴퓨팅 기술(Client/Server Computing)이다.

2.1.2 클라이언트/서버 시스템의 정의

클라이언트/서버 시스템은 하나의 응용 프로그램을 논리적으로 분리된 수개의 부분으로 분산시키는 것을 의미하고, 분리된 각 부분은 각각 고유하고 독립된 역할을 담당하는 것으로 클라이언트는 서버에 서비스를 요청하면 서버는 클라이언트에게 서비스를 제공한다는 것을 기반으로 한다 [그림 1]. 따라서 클라이언트는 기본적으로 1 대 이상의 서버가 제공하는 서비스를 사용하는 소비자에 속한다. 물론 하드웨어적으로 기능과 역할이 명확히 분리 된 것은 아니다 [5].

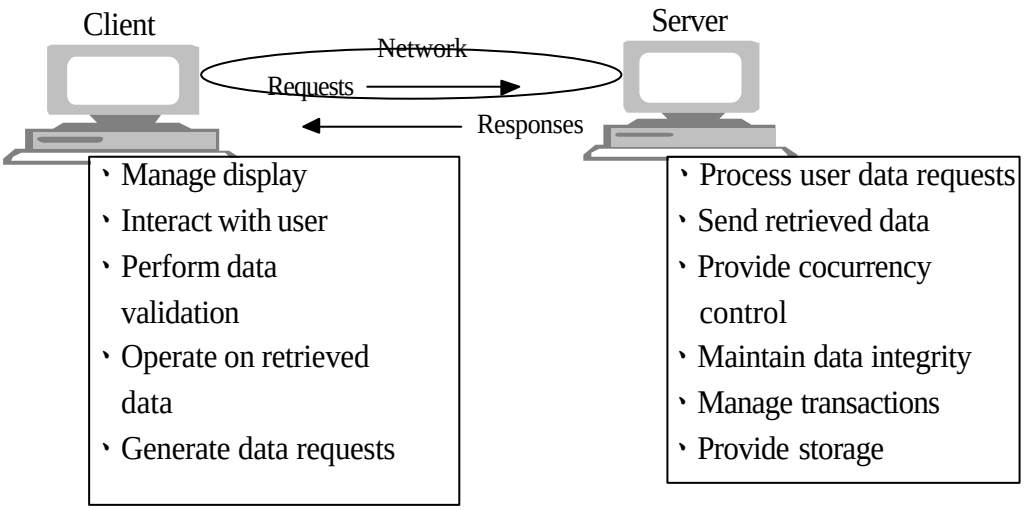


그림 1. 클라이언트/서버 시스템 구조

클라이언트/서버 시스템이 기존의 중앙집중형 메인프레임 시스템과는 달리 다음과 같이 뚜렷한 특징을 가지고 있다 [2].

- 서버는 동시에 많은 클라이언트에게 서비스를 제공하고, 각종 서버의 시스템 자원을 공유할 수 있도록 해 준다.
- 클라이언트/서버는 네트웍을 경유해서 서로 다른 컴퓨터에 위치 할 수 있으며, 한 컴퓨터에 동시에 위치할 수도 있고, 클라이언트 프로그램이 서버 역할을 동시에 할 수 있으며, 서버 프로그램 역시 클라이언트가 될 수 있다.
- 클라이언트와 서버는 서로 다른 곳에 놓일 수도 있고 동일한 컴퓨터안에 놓일 수도 있으며 또한 역할도 동시에 할 수 있다.
- 클라이언트와 서버는 하드웨어 플랫폼에 무관해야 한다. 즉 클라이언트와 서버가 어떤 운영체제나 하드웨어에 위치하건 서버는 클라이언트에게 서비스를 제공해야 한다.
- 클라이언트의 수를 추가하거나 서버의 수를 추가하는 등의 규모의 변화에 대해 전체시스템 운영에 무관해야 한다. 또한 서버 기능을 추가한다거나 업그레이드 (upgrade)해도 클라이언트에게 영향을 미쳐서는 안된다.
- 서버 코드와 데이타가 중앙에 집중관리 되기 때문에 데이타 무결성 (integrity)을 보장하고 유지비용이 적게 든다.
- 변화하는 업무요구에 신속하게 대처 가능하다.

이러한 특징으로 인해 과거의 중앙 집중형 컴퓨팅은 점점 사라져 가고, 클라이언트/서버를 기반으로 하는 분산컴퓨팅이 등장하게 되었다.

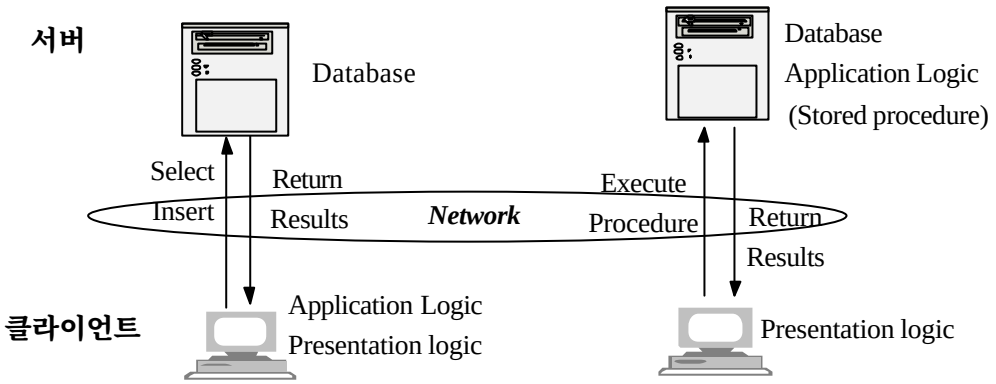
2.1.3 클라이언트/서버 아키텍처

클라이언트/서버 시스템의 아키텍처 구성요소는 사용자 인터페이스 (User Interface)를 담당하는 프리젠테이션 로직 (Presentation logic), 업무 프로세스 (Business Process)를 정의한 애플리케이션 로직 (Application logic), 데이타

서비스 (Data service)를 담당하는 데이타베이스 (Database)로 구성된다. 이들 구성요소의 위치나 역할에 따라 2 계층 구조 (2-tier Architecture)와 3 계층 구조 (3-tier Architecture)로 분류된다 [7, 10, 32].

2.1.3.1 2계층 구조 (2-tier Architecture)

2 계층 클라이언트/서버 모델은 클라이언트에는 사용자 인터페이스의 프리젠테이션 로직이, 서버에는 데이타베이스가 위치하는 것으로 애플리케이션 로직의 위치에 따라 그림 2 와 같이 원격 데이타베이스 접근 모델 (Remote Database Access Model)과 데이타베이스 서버 모델 (Database Server Model)로 나눌 수 있다.



(a) 원격 데이타베이스 접근 모델 (Remote SQL)

(b) 데이타베이스 서버 모델 (Stored Procedure)

그림 2. 2계층 구조의 2가지 모델

- 원격 데이타 베이스 접근 모델

  클라이언트에 애플리케이션 로직을 포함한 프리젠테이션 로직이 놓이는 것으로 Structured Query Language (SQL)문을 작성하여 원격지의 컴퓨터에 위치한 데이타베이스 서버측에 전달하면 서버는 데이타베이스에 대한 Read/Write 만을 처리하여 결과를 다시 클라이언트에 보내는 데이타 전송 형태이다. 일명 팻 클라이언트 (fat client)라고 한다.

 데이타베이스 서버 모델

클라이언트에는 프리젠테이션 로직만 놓이고 서버에는 애플리케이션 로직에 해당하는 Stored Procedure (SP)와 데이타베이스가 위치하고, 클라이언트는 이 프로시저를 호출하고 단지 화면이나 입출력 등의 사용자 인터페이스만을 담당하는 형태의 메시지 패싱 (Message passing)방법이다. 일명 씬 클라이언트 (thin client)라고 한다.

위의 두가지 형태의 2 계층 구조는 클라이언트/서버 시스템의 제 1 세대로서 지난 수년 동안 컴퓨팅 시장의 주류를 이루었다. 그러나 지금 클라이언트/서버 컴퓨팅 기술이 새롭고 더 복잡한 애플리케이션을 포함하는 방향으로 확산됨에 따라 2계층 아키텍처의 한계점이 나타나고 있다.

첫째, 온라인 사용자수 증가에 따라 성능이 떨어진다. 이는 DBMS 서버 프로세스의 접속으로 모든 클라이언트에 쓰레드를 유지하면서 지속적으로 연결상태의 메시지를 교환한다. 그래서 클라이언트 수가 일정 범위를 넘으면 현저하게 성능이 떨어진다.

둘째, 애플리케이션의 논리적 구조와 물리적 구조가 완전하게 분리 되지 안아 비지니스 프로세스 (business processes)나 비지니스 환경 (business environment)이 변경될 때 마다 비지니스 로직이 바뀌어야 한다. 따라서 많은 PC의 소프트웨어가 수정되어야 한다.

셋째, 대부분의 애플리케이션이 특정 RDBMS 의 SQL, stored procedure, APIs, 미들웨어 등의 기술에 종속되어 있기 때문에 데이타 소스(data sources)나 애플리케이션이 증가되었을 때 통합하기가 어렵다. 특히 수 많은 legacy data sources 를 액세스 하는데 많은 제약조건이 있으며, 비지니스 룰과 프로세스가 분명히 구분되지 않고 개발되기 때문에 애플리케이션 개발과 적용에 있어서 유연성 (flexibility)이 떨어지고, 광범위하게 흩어진 애플리케이션을 확대하거나 유지하기가 어렵다.

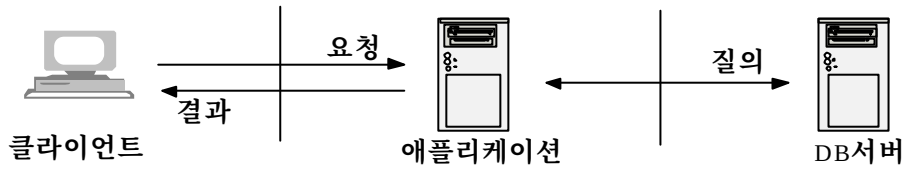
넷째, 애플리케이션 로직이 프리젠테이션 로직에 포함되어 있어 애플리케이

션이 개발되면 한 서버에서 다른 서버로 프로그램 기능의 일부를 이동하거나 재사용하는 것이 쉽지 않다.

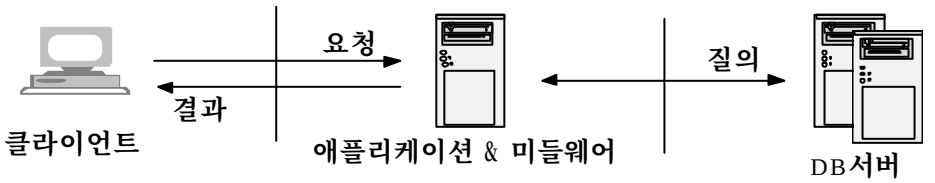
그래서 대부분의 2 계층 구조는 모든 정보가 단일 데이타베이스에서 나오고 LAN 인 경우, 업무량이 적은 부서단위의 Batch 업무 (통계분석, 리포트 발행 등)와 메인프레임에서 클라이언트/서버 컴퓨팅환경으로의 전환초기에 위험부담을 최소화하고 저렴한 비용으로 시스템을 구축하고자 할 때 적당하다.

2.1.3.2 3 계층 구조 (3-tier Architecture)

2 계층 구조의 문제점을 해결하는 방안으로서 현재 많은 관심사로 대두되고 있는 것이 3 계층 구조이다. 이 방법은 데이타베이스가 위치한 서버 부분과 사용자가 주로 사용하는 클라이언트를 완전히 분리하고, 애플리케이션 로직을 모듈화하여 중간계층에 별도로 위치시킴으로서 좀더 유연하고 확장 가능한 시스템 구현이 가능하다. 그래서 중간계층에 미들웨어의 유무에 따라 2 가지 모델로 구분할 수 있다 [그림 3].



(a) 순수형 3계층 모델



(b) 중재형 3계층 모델

그림 3. 3계층 구조의 2가지 모델

 순수형 3계층 모델

클라이언트와 서버로부터 애플리케이션만을 분리해서 작성하는 것으로 실

제 3 개의 층으로 구성되어 있지만 물리적 계층분리에 의미를 둔 것으로 분산환경에서 다양한 서버에 접근하려면 많은 애플리케이션을 작성해야 한다.

 중재형 3계층 모델

미들웨어라는 중재 소프트웨어를 마련하여 복수개의 클라이언트와 서버사이에서 인터페이스를 처리하는 방안으로 애플리케이션 로직을 모듈화하여 중간 계층에 두어 이질적인 대규모 분산 환경에서 일관성 있는 접근을 가능케 하는 것으로 2 계층 클라이언트/서버 구조의 단점으로 지적된 클라이언트 수의 증가에 따른 성능저하 문제를 해결하기 위해 애플리케이션 로직을 분산시키기 위한 구조이다.

진정한 의미의 3 계층 구조의 클라이언트/서버 시스템은 애플리케이션 서버에 미들웨어를 적용하여 데이타베이스의 위치나 공급자에 상관없이 이질적인 DBMS 를 계속 연결시켜 확장이 가능하며, 서버에 많은 부하가 걸릴 때 부하를 균등하게 분배하는 부하분산 (load balancing)이 가능한 중재형 3 계층 모델을 지칭한다. 이러한 3계층 모델의 주요 장점은 다음과 같다 [6].

첫째, 비지니스 프로세싱의 부담이 클라이언트에서 덜어지고 개별 애플리케이션 서버가 비지니스 프로세싱을 맡는다. 즉 사용자 인터페이스, 비지니스 프로세싱, 데이타베이스 관리 등은 별도의 프로세스들에 할당된다. 그래서 각 층은 자체적인 하드웨어 및 소프트웨어 아키텍처를 가질 수 있다. 또한 중간 층은 대기, 애플리케이션 실행, 데이타베이스 스테이징 (staging) 등과 같은 서로 다른 많은 기능을 실행할 수 있어 시스템의 성능과 융통성에 있어 많은 향상을 가져온다. 2 계층 방식의 동기화 프로세스는 3 계층에서는 비동기 큐를 관리하여 트랜젝션 완료 신뢰성을 보장하면서도 클라이언트가 다른 작업을 할 수 있도록 한다.

둘째, 애플리케이션 서버에 비지니스 로직을 배치 하므로서 개발자들이 프로그래밍 언어에 크게 구애 받지 않고 개발이 가능하고, 애플리케이션 로직을 모듈화하는 방법으로 일반관리와 변경관리를 간소화하여 유지관리의 비용을 줄일 수

있다. 또한 잘 정의된 객체 오브젝트는 다른 애플리케이션에서 재사용될 수 있어 개발 노력을 줄일 수 있다.

셋째, 확장성을 크게 향상 시킨다. 데이타베이스 서버는 애플리케이션 서버 기능을 하지 않는 대신 전적으로 데이타 작업만 수행하게 된다. 따라서 데이타베이스 서버가 처리 용량이 증가하여 균형과 확장성 향상을 위해 다중 애플리케이션 서버 상으로 분산될 수 있다.

주요 적용대상 업무는 온라인 트랜젝션 (Online Transaction)이 많이 발생하는 업무에 적합하다. 또한 여러 종류의 데이타베이스을 사용하거나 LAN 과 WAN 이 혼재된 환경에서 동시 사용자수가 많고, 전사적 애플리케이션 통합이 필요한 업무나 계속적으로 확장이 필요한 업무에 적당하다.

그러나 개발환경이 복잡하여 디자인이 어렵고, 하드웨어와 소프트웨어 비용이 증가한다는 점과 시스템이 복잡해지고 사용자의 숙련도에 따라 생산성이 저하되는 단점이 있다. 종합적으로 2 계층과 3 계층을 비교하면 다음과 같다 [표 1].

구분	2계층 구조	3계층 구조
장점	✍개발 환경이 용이하다. ✍구현이 쉽다. ✍변화에 대한 위험부담을 최소화 한다.	✍퍼포먼스,데이타 통합기능이 우수 하다. ✍시스템 확장성이 좋다 ✍애플리케이션 재사용이 용이하다 ✍일반관리나 변경관리등의 유지관리가 용이하다.
단점	✍사용자가 많아지면 성능이 저하 된다. ✍애플리케이션 수정시 유지관리의 어려움이 있다. ✍애플리케이션이 RDBMS에 종속되고, 재사용이 어렵다.	✍개발환경이 복잡하고 프로그래밍이 어렵다. ✍미들웨어 및 서버 도입에 따른 추가 비용이 소요된다.
적용업무	✍통계분석, 리포팅, 배치성업무 ✍단일데이타베이스의 업무 ✍로컬 LAN의 부서단위 업무	✍전사적으로 애플리케이션 통합이 필요한 업무 ✍LAN과 WAN이 혼재된 환경 ✍온라인 트랜젝션이 많은 업무 ✍동시 사용자수가 많은 업무

표 1. 2 계층 구조와 3 계층 구조의 비교

2.2 미들웨어

미들웨어는 네트웍이 기본으로 구축된 분산 컴퓨팅 환경에서 사용자의 컴퓨터와 네트웍상에서 실행되는 응용프로그램들간에 자유롭게 데이타를 이동시킴으로서 응용 애플리케이션 개발을 지원하는 소프트웨어로서 3 계층 클라이언트/서버 시스템 구축에 있어서 필수적인 기술이다 [8].

그래서 본 절에서는 미들웨어 기술의 특징과 동작원리를 알아보고 기능별 미들웨어 종류에 대해서 살펴 본다.

2.2.1 미들웨어 특징

미들웨어는 개발에 따른 비용을 줄일 수 있을 뿐만 아니라 개발기간을 단축하고 환경변화에 신속하고 유연하게 대처 할 수 있으며, 다양한 하드웨어 환경에서도 오픈시스템을 쉽게 구축 가능하다는 이점을 가지고 있다 [2,9].

- 분산처리 환경에서 필연적인 multi-vendor, multi-protocol 에 대해 독립성을 제공하고, 하드웨어 및 네트웍 변화나 확장시에 유연하게 대처할 수 있다.
- 각각의 하드웨어 플랫폼에서 운영되는 다양한 데이타베이스의 모든 기능을 지원 한다. 사용자는 자신이 각각의 하드웨어 플랫폼의 데이타베이스를 사용하는 것과 같은 투명성을 제공한다.
- 사용자 중심의 시스템 개발의 변화로 다양한 사용자 개발툴을 지원한다.
- 모든 기술에 대한 표준을 지원하여 다운사이징을 가속화 한다.
- 현재 컴퓨팅환경에 즉시 적용 가능한 개발 환경을 제공하여 전체적인 시스템 구축 시간을 최소화하고 도입에 따른 비용을 절감한다.

미들웨어는 이기종의 복합서버가 포함된 멀티밴더의 클라이언트/서버 환경에서 클라이언트 애플리케이션으로부터 데이타베이스에 대한 빠르고 투명한 액세스가 요구되는 환경에서 필요하다.

2.2.2 동작원리

미들웨어는 네트웍상의 응용프로그램간에 자유롭게 데이타를 이동시켜 주는 것으로 두가지 동작방식이 있다.

- Remote Procedure Call (RPC) 방식

  RPC 방식 [2]은 썬마이크로시스템즈의 유닉스 환경에서 등장한 것으로서 현재 가장 널리 사용되는 핵심 기술이다. 네트웍상에 분산된 각종 자원을 애플리케이션을 통해 자신이 원하는 데이타를 자유롭게 액세스하는 것이 기본 계념으로, 사용자에게는 마치 원하는 데이타가 자신의 컴퓨터에 있는 것처럼 투명하게 보여 준다.

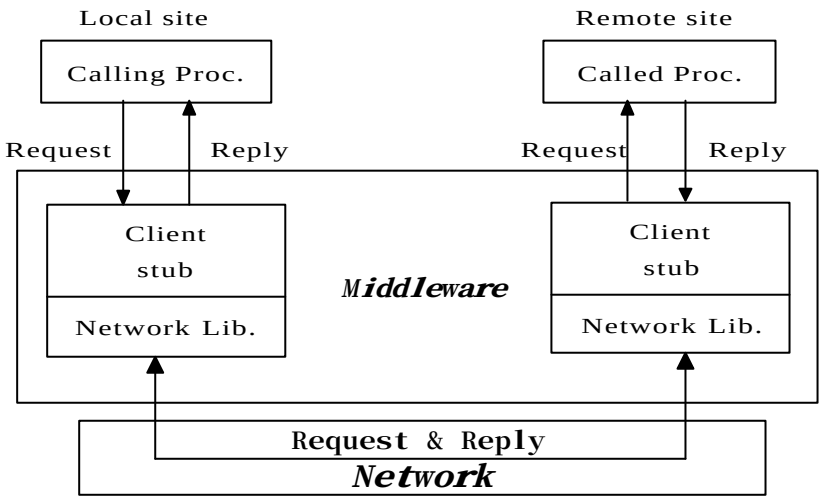


그림 4. RPC 메커니즘

RPC Mechanism 은 그림 4 와 같이 stub 라고 하는 로컬프로시저에서 전달될 아규먼트 (Argument)를 저장하고, 표준형식에 따라 네트웍 메세지로 변환하는 것으로,사용자는 RPC 에서 제공하는 Interface Definition Language (IDL)를 통해 원하는 함수를 작성한다. 이때 IDL 컴파일러는 클라이언트측에서 사용하는 함수와 서버에서 사용하는 함수를 함께 제공한다. 따라서 클라

이언트는 자신에게 주어진 함수를 호출하면 이 호출은 자동으로 RPC 런타임 라이브러리를 통해 서버에 위치한 함수가 호출된다. 해당 함수의 수행이 끝난 후 결과는 다시 클라이언트측에서 호출한 함수의 결과값으로서 전달된다. 주요 특징은 반드시 두 컴퓨터간에 동기가 형성되어야만 실행되도록 설계되어 있어 높은 데이타 무결성을 제공하는 반면, 동기가 형성되지 않을 경우 연결될 때 까지 무작정 기다려야 한다는 비효율성이 있다.

- 메시지패싱 (Message passing) 방식

메시지패싱 방식 [2]은 저장후 전송 (Stored and Forward)방식에 기반을 두고 있는 미들웨어로서, RPC 방식에 비해 반드시 동기형성이 필요한 것은 아니기 때문에 무조건 기다리는 비효율성을 배제한다. 따라서 복수의 요청을 동시에 처리할 수 있는 이점을 제공함으로서 네트웍을 효율적으로 사용할 수 있다. 가장 대표적인 예로서 전자메일 시스템으로 로컬사이트(Local site)의 프로세스는 로컬큐 (Local queue)에 메시지를 저장하고, 큐메니저 (Queue manager)는 메시지를 리모트큐 (Remote queue)에 전송하는 방식을 취하고 있다. 특징은 RPC 방식에 비해 표준화 및 사용자의 기반이 약한 반면, 네트웍상의 자원을 효율적으로 사용할 수 있다는 장점이 있다. 이 방식의 메커니즘은 그림 5 와 같다.

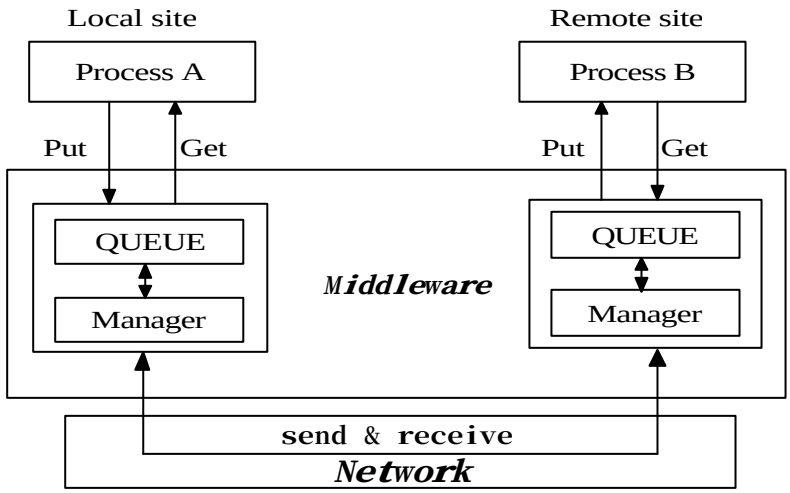


그림 5. 메시지 패싱 메커니즘

RPC 동작방식은 3 단계로서 전단계의 작업이 정상적으로 종료되었을 때 다음

단계로 넘어 간다.

1 단계는 프로세스가 큐에 메시지를 전달하는 과정으로 큐에 정상적으로 들어 가면 작업이 Manager 로 이전된다. 이때 Manager 는 메시지의 목적지가 로컬 인지 리모트인지를 판별하게 되며, 정상적으로 종료되지 않으면 메시지는 로컬 프로시저에 리턴 된다. 2 단계는 리모트 큐로 메시지가 운반되는 과정으로 작업이 비정상적으로 종료되면 메시지는 로컬 큐로 돌려지게 된다. 3 단계는 리모트 프로시저에서 메시지를 입수하는 과정으로 이때의 비정상 종료는 메시지를 리모트 큐로 되돌린다.

그리고 메시지패싱 방식과 RPC 방식의 비교는 표 2 과 같다 [2].

비 교	메시지 패싱 방식	RPC 방식
특징	우편배달 방식	전화통신 방식
클라이언트/서버 Time Relationship	비동기, 클라이언트와 서버는 다른 시간대와 속도로 운영가능	동기, 클라이언트와 서버는 반드시 동시에 수행. 서버는 클라이언트를 기다리고 있어야 한다.
클라이언트/서버 Sequencing	No fixed sequence	서버는 클라이언트가 요청하기 전에 먼저 준비하고 대기
Style	Queued	Call-Return
Load balancing	단일 queue가 FIFO 혹은 Priority -based policy를 구현키 위해 사용되어 진다.	분리된 TP Monitor를 요구
Message filtering	Yes	No

표 2. 메시지패싱 방식과 RPC 방식의 비교

2.2.3 미들웨어 종류

초기 3 계층 시스템 구성을 위한 미들웨어는 복잡한 클라이언트/서버간 통신을 단순화하기 위한 것부터 시작되었다. 이후 실세계의 다양한 요구 사항을 해결하기 위해 보다 다양한 기능이 추가되어 Distributed Computing Environment (DCE) [11]같은 통합 통신 미들웨어와 데이타베이스 전용 미들웨어, 그리고 객체 지향 개념을 적극 수용한 분산객체 미들웨어 등으로 발전을 거듭하게 된다. 대표적인 분산객체 시스템으로 Object Management Group (OMG)의 Common Object request broker Architecture (CORBA) [12]와 MS 의 Component Object Model/Distributed COM (COM/DCOM) [13]을 들 수 있다. 이런 관점에서 각 분야별 미들웨어에 대한 기능과 종류를 알아 본다 [16].

? 통신 기능 중심의 미들웨어

광의의 미들웨어의 의미로서 가장 초보적인 것이 Socket 통신 [34], RPC 에서 출발하여, Peer-to-Peer communication 으로 양쪽이 같은 프로토콜 인터페이스를 사용하여 링크하는 것으로 NetBIOS [35], Named pipes [36] 등이 있고, Open Software Foundation (OSF)의 DCE, 노벨의 네트웨어 [22] 등의 통신 기능 패키지가 있다.

1) Remote Procedure Call (RPC)

RPC 는 통신 미들웨어 중 가장 광범위하게 사용되었고 지금도 기타 다른 미들웨어 시스템의 하부 구조로 이용되는 시스템으로, 사용자에게 원격지에 있는 함수를 호출할 수 있는 방법을 제공함으로서 분산된 환경에서의 프로그래밍을 쉽게 제공하는 것을 목표로 만들어 진 것이다. 이러한 RPC 를 제공하는 것으로는 Sun Open Network Computing (ONC)에서 제공하는 RPC 와 OSF DCE 에서 제공하는 RPC 가 있다.

2) Distributed Computing Environment (DCE)

DCE 는 OSF 에서 제정한 분산환경을 위한 표준안으로서, 분산 애플리케이션의 개발 및 운영을 지원하고 단순화하는 툴과 서비스를 제공한다. 단일 소프트웨어 표준으로서 OSF 가 별도로 만든 것이 아니라, 기존 네트웍 운영체제에

서 우수한 기능만을 모아 분산환경에서 클라이언트/서버 프로그램이 가능하게 만들어진 것이다.

? 데이타베이스 중심의 미들웨어

데이타베이스 미들웨어는 클라이언트의 SQL 요청에 대한 서버의 서비스와 이에 대한 결과를 네트웍에서 주고 받는 일련의 서비스를 제공한다. 데이타베이스 미들웨어의 예로 마이크로소프트의 Open Database Connectivity (ODBC) [14], 볼랜드의 Integreted Database Application Program Interface (IDAPI) [18], SQL Access Group (SAG)의 Call Level Interface (CLI) [16], 오라클의 글루(Glue) [17] 등이 있다.

- Open Data Base Connectivity (ODBC)

마이크로소프트와 몇 개의 회사가 중심이 되어 개발한 것으로 SQL drive 를 사용하지 않고 윈도우 애플리케이션으로부터 어떤 DBMS 도 액세스 가능하게 한 클라이언트/서버 데이타베이스 표준 규격이다.

? On Line Transaction Processing (OLTP) 미들웨어

OLTP 미들웨어는 작업을 균등하게 분배하는 기술을 제공함으로서 데이타베이스 시스템의 성능을 향상시키고, 2-Phase Commit 이란 기능을 통해 데이타 갱신시 발생할 수 있는 데이타의 불일치 문제를 보장해 준다. 따라서 OLTP 미들웨어를 이용하는 클라이언트는 네트웍의 OLTP 서버에 트랜잭션 서비스를 요청하고 이에 대한 결과를 돌려 받는다. 결국 OLTP 미들웨어는 이질적인 분산환경에서 트랜잭션을 처리하고 각종 처리절차를 관리하는 기능을 제공하는데 목적이 있다. 특히 OLTP 미들웨어는 대규모이고 빠른 응답 시간을 요구하는 분야에 유용하다. 관련 제품으로는 노벨의 턱시도(Tuxedo) [22]와 트랜삭의 엔시나 (Encina) [20,23], IBM의 CICS [24] 등이 있다.

? 객체지향 중심의 미들웨어

현재 컴퓨터 사용자들은 서로 다른 운영체제와 네트웍 환경 등 이종의 환경에서 작동하는 클라이언트/서버 소프트웨어를 필요로 한다. 뿐만 아니라 이렇게 개발된 소프트웨어는 분산환경에서 서로 다른 시스템과도 쉽게 통합이

가능해야 한다. 기존의 Remote Procedure Call (RPC)나 MS 의 Object Linking and Embedding (OLE)같은 기법을 사용 해 문제를 해결 해 왔으나, RPC 의 경우 복잡한 프로그램을 작성하는 데는 많은 한계를 갖고 있으며, OLE 의 경우도 현재 윈도우 객체만을 지원하기 때문에 진정한 분산시스템 표준이라고 말할 수 없다. 현재의 컴퓨팅 환경에서 이 같은 분산시스템을 구축할 수 있는 것이 바로 객체지향 미들웨어이다. 대표적인 것으로 CORBA 와 DCOM 이다.

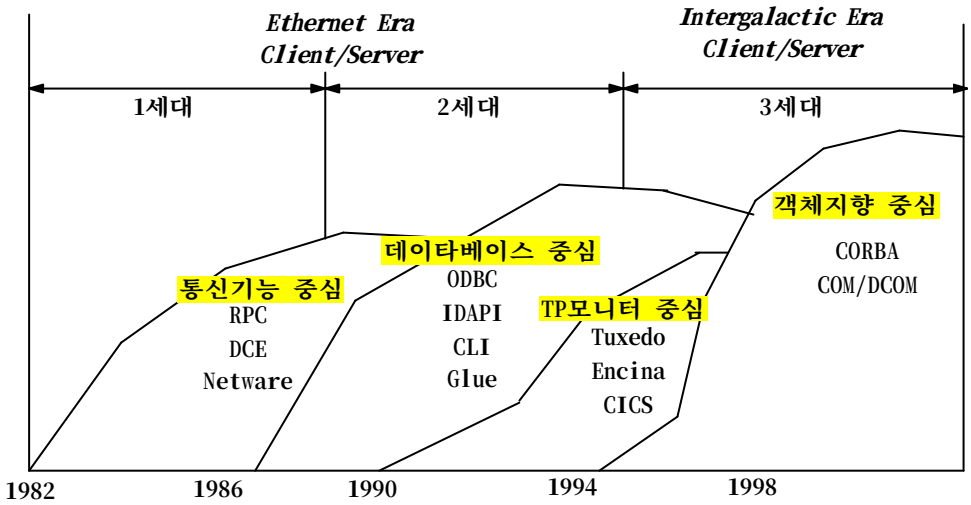


그림 6. 미들웨어 종류 및 발전과정

지금까지 미들웨어의 특징 및 동작원리에 대해서 알아 보았고, 또한 미들웨어를 기능별로 크게 네 가지로 분류하였으며, 여기에 해당되는 미들웨어 종류를 알아 보았다. 분산컴퓨팅이 보다 복잡하고 다양해 지면서 미들웨어의 필요성이 더욱 필요하게 되었다. 종전의 데이타베이스 연결용 미들웨어가 주를 이루고 있었지만, 최근 들어 많은 사람의 관심사인 분산객체 기술이 현재 컴퓨팅 세계에서 가장 중요한 핵심 기술로 대두되고 있다 [그림 6].

2.3 Common Object Request Broker Architecture (CORBA)

1989 년 4 월에 600 개의 컴퓨터 관련 단체들이 참가하여 현재 존재하는 객체 기술을 바탕으로 응용 프로그램들을 결합하기 위한 객체지향 표준을 제정하기 위해 Object Management Group (OMG)라는 비영리 단체가 탄생되었다. 이 단체에서 제정한 객체지향 기술을 기반으로 이종의 분산된 환경하에서 응용 프로그램들을 서로 통합할 수 있는 표준기술이 Object Management Architecture (OMA) [37]이다. OMA 는 응용 프로그램간의 결합뿐만 아니라 객체의 생성, 소멸에서 부터 저장, 트렌젝션 기능에 이르기까지 분산 객체 환경에서 필요한 모든 서비스를 총칭하는 것이다. 이들 기능 중 CORBA [12]는 컴퓨터 내부의 버스처럼 서로 다른 프로그램들 사이의 버스 역할을 하는 모듈로서 OMA 구조에서 가장 중요한 요소가 된다. 여기서 OMG 에서 발표한 OMA 와 CORBA, Object Request Broker (ORB)에 관한 기능과 역할에 대해서 알아 보고자 한다.

2.3.1 Object Management Architecture (OMA)

OMG 의 OMA 는 이종의 분산환경에서 필요한 모든 서비스는 객체개념을 기반으로 하는 것으로, 크게 네가지로 분류될 수 있다 [그림 7].

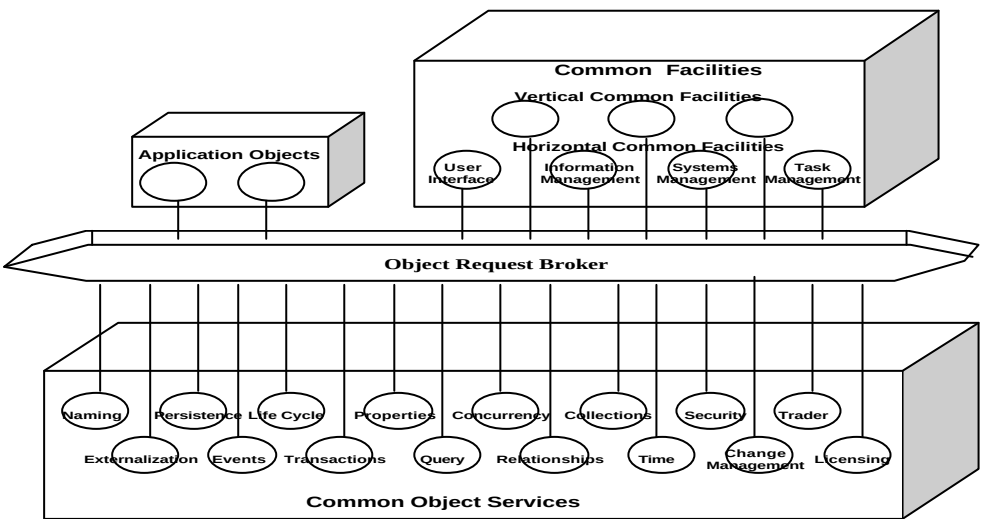


그림 7. OBJECT MANAGEMANT ARCHITECTURE (OMA)

이종의 분산 환경에서 통신을 담당하는 CORBA, 객체를 조작하는 데 필요한 각종 기본 기능들을 정의하고 있는 Common Object Service Specification (COSS), 메일, 프린팅, query 와 같은 추가적으로 제공되는 기능 서비스 (Common Facilities), 마지막으로 이들 기능들을 이용하여 사용자들이 작성한 각종 응용 객체 (Application Objects)들이 존재한다. OMA 구조에서 가장 중요한 것이 CORBA이며 현재 많은 구현 제품들이 출시 되고 있다.

2.3.2 CORBA 구조

CORBA 는 OMA 의 기본 기능으로, 애플리케이션 오브젝트는 일반적인 응용프로그램을 말하며, Object Request Broker (ORB)는 오브젝트의 통신채널을 담당하며, 오브젝트 서비스는 오브젝트를 다루는데 필요한 persistency, concurrency, naming, life-cycle, security 등을 제공하고, printing, mail 과 같은 추가 기능을 제공하는 기본 서비스로 구성되어 있다 [14].

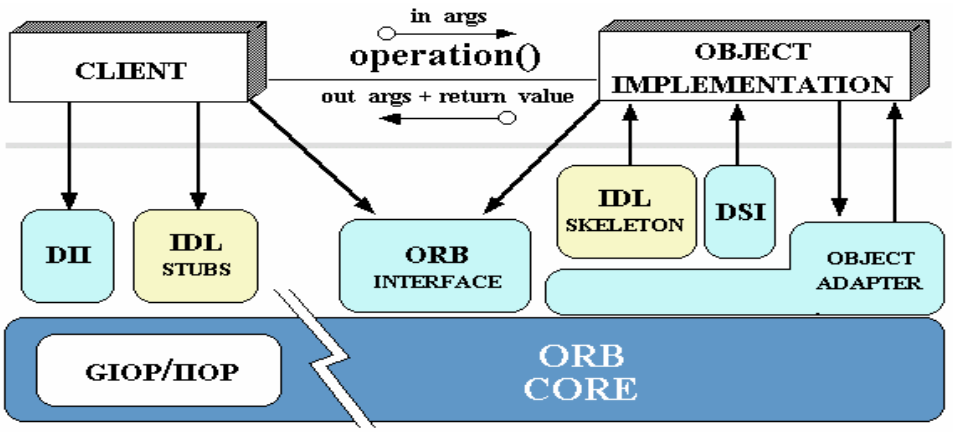


그림 8. CORBA 구조

CORBA 구조에 대한 각각의 기능에 대해서 살펴 보면 그림 8 에서 처럼 크게 ORB Core 부분, 클라이언트 부분, 구현객체 (Object Implementation)부분으로 구성되어 있다 [그림 8].

먼저, ORB는 클라이언트와 서버간의 요청과 응답을 중계하는 역할을 한다.

클라이언트 요청에 의한 서버 오브젝트 위치를 이름서비스 (white page 와 yellow page)를 이용해서 찾아 주며, 구현객체 (Object implementation)가 제공되는 컴퓨터에서 아직 활성화 되지 않았을 경우는 활성화를 시키고 요청을 받을 준비가 되도록 한다.

둘째, 클라이언트 부분으로 구현객체에 대한 서비스 요청을 담당하는 것으로 정적호출 (dynamic Invocation), 클라이언트 스터브 (Client Stub), 인터페이스 저장소 (Interface Repository), ORB 인터페이스로 구성되어 있다. 각각의 기능은 다음과 같다.

- ? 클라이언트 스터브는 오브젝트 서버에 대한 정적인 접속을 제공하고 IDL 정의에 의해서 제공된 코드부터 미리 컴파일 된다.
- ? Dynamic Invocation은 실행 중에 원하는 메스드를 동적으로 호출할 수 있는 기능으로 인터페이스 저장소에서 해당 클래스의 인터페이스를 참조하여 메스드를 수행한다.
- ? 인터페이스 저장소 (Interface Repository)는 등록된 서버에 대한 정보와 IDL의 컴파일된 버전을 가지고 있다.
- ? ORB Interface는 통신기능과 메스드 호출을 담당하는 것으로 클라이언트와 구현 객체 (Object Implimentation)에 모두 해당된다.

셋째, 구현 객체 부분은 skeleton 을 통해 들어온 클라이언트의 서비스요청을 처리하는 것으로 구현 스켈레톤 (Implementation Skeleton), 오브젝트 아답터 (Object Adaptor), ORB 인터페이스로 구성되어 있으며 기능은 다음과 같다.

- ? 구현 스켈레톤은 클라이언트의 서비스 요청에 대한 구현 객체를 직접 호출하는 역할로서, 서비스 요청시 제공되는 정보를 해석하여 구현객체에 전달하고 결과를 클라이언트에 전달한다.
- ? Dynamic Skeleton Interface (DSI)는 IDL-based compiled skeleton을 가지지 않은 컴포넌트를 부를 때 메스드를 처리해야 할 서버를 위해 제공된다.
- ? 오브젝트 아답터는ORB의 핵심 통신서비스와 서버를 접속하는 것으로 서

버 오브젝트에게 실행시 함수의 인자와 호출을 담당하는 것으로 CORBA를 적용하려는 분야에 따라 다양하게 선택되어 사용되어 질 수 있다. 위치적으로 객체 아답타는 구현 스켈레턴과 ORB에 위치한다. 따라서 객체 아답터는 구현 객체를 등록하거나 삭제하며 해당 구현 객체를 유일하게 관리한다. 또한 실제 메소드를 호출시키는 역할을 한다. 이처럼 구현 객체를 호출하기 위해서 모든 구현객체에 대한 정보를 구현 저장소 (Implementation Repository)에 저장하고 이를 참조한다.

CORBA 는 클라이언트에게 2 가지 서비스 호출방식을 제공한다. 하나는 정적인 방법 (Static invocation)이고 다른 하나는 동적인 방법 (Dynamic Invocation)이다. 먼저 정적인 호출방법은 ORB 에서 제공하는 기본 API 기능을 이용하여 서비스를 요청하는 것으로 사용자들이 가장 손쉽게 프로그램을 작성할 수 있는 방법이다. 그러나 마우스 프로그래밍처럼 프로그램이 수행 중에 특정한 서비스를 필요로 할 경우 정적인 방법으로는 불가능하다. 이런 경우에 사용 가능한 방법이 동적 인터페이스 호출로서 동기방식과 비동기 방식 모두를 지원한다. 따라서 사용자는 다양한 응용프로그램을 작성할 수 있다. 단점으로는 프로그램시 정적 인터페이스 호출에 비해 많은 코드와 노력이 필요하다는 것이다.

2.4 3 계층 구조의 클라이언트/서버 시스템 구축 사례

현재 3 계층 구조의 클라이언트/서버 시스템이 구축된 사례가 흔하지 않다. 미들웨어 벤더 (vendor)들이 자사 제품과 연계한 3 계층 아키텍쳐 구축에 대한 솔루션을 제시하고 있으나 아직 까지 검증되지 않았으며, 대부분의 조직들이 현재 2계층 구조에서 3 계층 구조로의 전환을 모색하고 있다.

2.4.1 동원산업

동원산업은 종합식품회사로서 메인프레임급 IBM 4381 모델에서 전사적인 클라이언트/서버 시스템으로 전환한 예다 [38].

처음부터 모든 업무를 클라이언트/서버 환경으로 전환하기에는 무리가 있다고 판단하여 부문별 단계적으로 구축해 나갔다. 서버는 HP 9000/70 시스템을, 데이타베이스는 오라클을, 개발툴은 오라클 폼즈 등을 사용했으며. 클라이언트와 서버를 연결하는 미들웨어는 OLTP 미들웨어 턱시도를 사용했다.

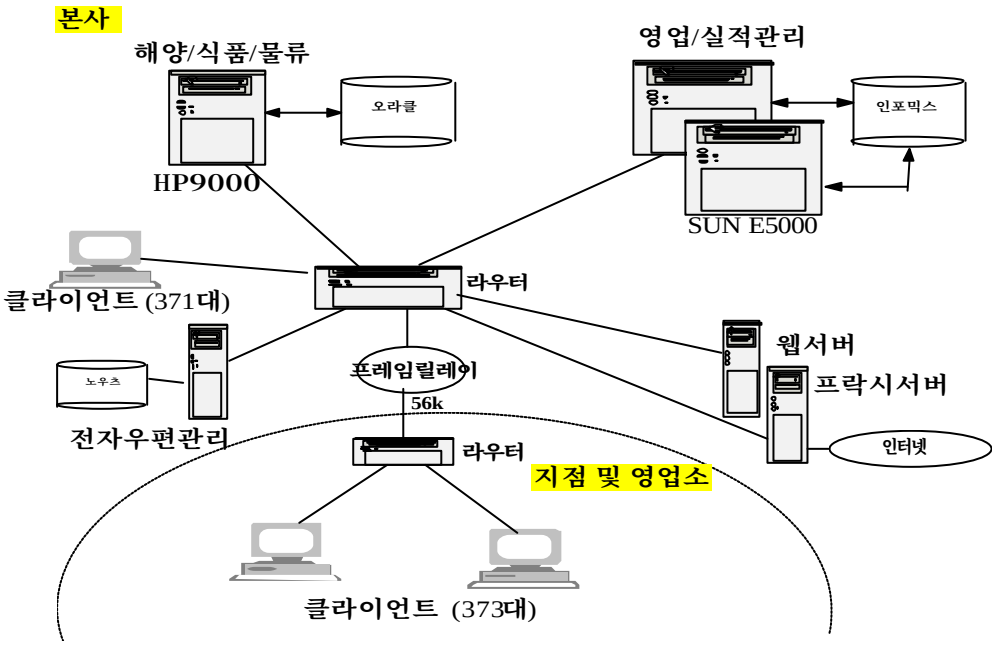


그림 9. 동원산업 시스템 구성도

시스템 구성 방법은 그림 9 처럼 본사에 서버를 집중 배치하여 데이타베이스와 애플리케이션 로직을 두고 클라이언트는 본사를 포함한 지방에 약 700 대의 클라이언트 PC 를 두었다. 영업관리 및 영업실적관리업무는 온라인 트랜젝션이 많은 발생하는 것으로 엔터프라즈 서버급 SUN E5000를 주업무용으로 사용했다.

시스템의 특징은 모든 서버가 본사에 집중되어 있으며, 별도의 애플리케이션 전용서버를 두지 안고 데이타베이스 서버에 애플리케이션 로직을 배치하였다. 클라이언트와 서버 사이의 연결 미들웨어는 OLTP 용 미들웨어를 사용했으나 서버 업무 배치의 잘못, 업무 설계 및 개발방법의 잘못 등으로 업무와 데이타의 분산이 원활하지 않아 특정한 한 두 대의 장비에 부하가 많이 걸렸고 처리가 불가능했던 것이다. 클라이언트/서버 시스템의 주목적이 GUI 에 치중한 나머지 클라이언트/서버의 궁극적인 목적인 자원과 부하의 분산에 실패 했다.

본 사례에서는 기존의 메인프레임을 그대로 사용하면서 부문별로 클라이언트/서버 시스템으로의 전환으로 업무 유형별 시스템 구현방법이 문제시 되었다. 파일구조 형태나 계층적 데이타베이스, 네트웍 데이타베이스 형태의 메인프레임 개발 방법을 클라이언트/서버 시스템 개발방법에 그대로 적용하여 데이타베이스 액세스 효율을 저하시키고 특정 데이타베이스나 서버에 부하가 집중되는 현상이 나타 났다.

2.4.2 대한보증보험

대한보증보험의 기존시스템은 과거 10 년 넘게 구축해 온 정보 인프라로 더미단말기를 이용한 전형적인 중앙집중식 방식이었다 그래서 기존의 후지쯔 M760 2 대의 업무를 3 계층 형태의 신정보시스템 구축에 나섰다 [38].

주 업무를 위해서 SUN ES6000 6 대와 ES3000 4 대 등 10 대의 서버를 도입하여 업무를 배정하고 클라이언트는 본사 지점을 포함해 팬티엄 PC 1700 대를 도입했고, 개발 툴로는 볼랜드 델파이를 이용했다.

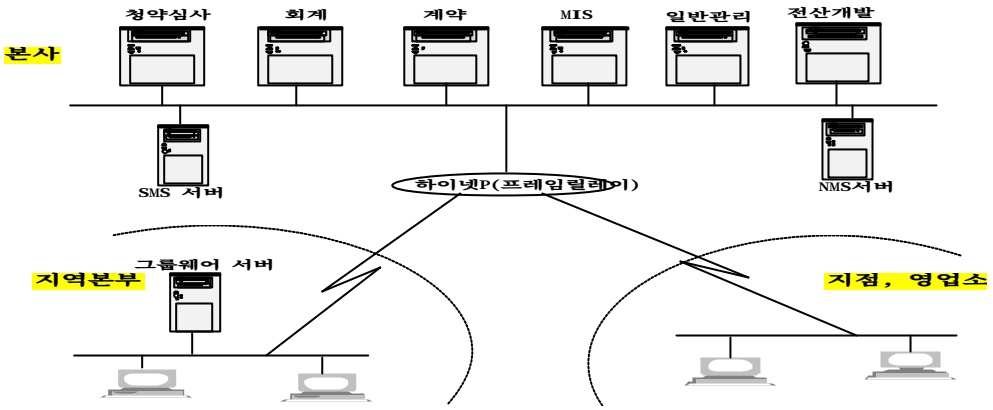


그림 10. 대한보증보험 시스템 구성도

전국 100 여개의 지점을 하이넷-P (프레임 릴레이망)으로 연결하여 모든 기간 업무를 3 계층으로 구현한 것으로 클라이언트와 서버 사이에 TP 모니터를 둠으로서 모든 업무가 TP 모니터를 중심으로 분배된다. 즉 턱시도를 TP 모니터로 이용한 3 계층 클라이언트/서버 환경을 도입함으로서, 서버의 부하를 균등하게 분배했으며, 네트웍 통신량을 감소시켰다. 또한 각종 소프트웨어를 컴포넌트로 구성함으로서 프로그래머의 작업을 간소화 하였고, 철저하게 최종 사용자 위주의 시스템을 구현했다. 그래서 지금까지는 별 문제가 없지만 앞으로 좀더 시스템을 운용해 보아야 할 것 같다 [그림 10].

시스템 특징은 업무 유형에 따라 애플리케이션 서버를 별도로 구성하여 성능 향상를 꾀하였다. 온라인 트랜젝션이 많고 사용자수가 많은 업무는 엔터프라이즈 서버를 운용하고 있다. 여러 종류의 서버와 데이타베이스의 이질적인 환경을 연결하는 TP 모니터용 미들웨어를 사용하였다. 또한 시스템 확장성을 위한 방안으로 DBMS 를 계속 연결시켜 확장이 가능하며, 서버에 많은 부하가 걸릴 때 부하를 균등하게 분배하는 부하분산이 가능한 애플리케이션 서버를 구축했다.

본 사례는 기존의 메인프레임 시스템과의 연계 개발이 아니라 처음부터 TP 모니터용 미들웨어를 도입하여 전 업무에 걸쳐 3 계층 클라이언트/서버 시스템을 구축하였다. 그런데 애플리케이션 전용 서버를 사용하지 않고 데이타베이스 서버에 애플리케이션 서비스를 배치시켰다.

3. 클라이언트/서버 시스템 현상분석

이 장에서는 클라이언트/서버 시스템의 예로서 포스코의 전체적인 시스템 아키텍처에 대한 내용을 살펴 보고, 현재 포스코에서 구축 운영중인 약 70 여개의 클라이언트/서버 시스템에 대해서 '97 년 4 월부터 6 월까지 3 개월간 클라이언트/서버 시스템 개발자를 중심으로 설문조사 및 전화에 의해 조사한 내용을 부문별로 분석 정리 하였다.

3.1 포스코 시스템 아키텍처

회사 전반적인 업무를 관리하는 메인프레임 베이스의 HOST 시스템이 지역별로 서울, 포항, 광양으로 분산되어 있고, 최근에 구축하여 운영 중에 있는 클라이언트/서버 시스템, 공장에서의 공정 제어를 맡고 있는 프로세스 컴퓨터 (Process Computer) 및 설비 제어용 시스템으로 구분되어 있다. 시스템의 주요 특징은 대부분의 시스템이 서로 이질적인 환경으로 구성되어 있고, 특히 제철소 조업관리 시스템은 연중 무휴로 운영되며 그래서 시스템 중단이 바로 조업 중단으로 연결되어 막대한 손실을 가져 올 수 있어 시스템 안정성이 무엇보다 중요하다. 또한 리얼타임 프로세스로 트래픽이 많이 발생되는 업무가 대부분을 차지하고 있다.

네트웍 구성은 서울, 포항, 광양을 연결하는 T1 급 WAN 이 설치되어 있고 각 지역별 메인프레임을 중심으로 한 조업용 광 LAN 과 클라이언트/서버 시스템을 중심으로 최근에 설치된 서울의 ATM, 포항, 광양의 FDDI LAN 으로 구성되어 있고 그 하부에 각 공장 및 사무실 단위로 Ethernet 이 구성되어 있다 [그림 11].

하드웨어 구성은 메인프레임은 서울은 IBM 9121 2 대 (DASD 360GB), 광양은 IBM 3090 1 대, IBM 9021 2 대 (DASD 283GB), 포항은 FACOM M780 4 대(DASD 476GB)로 구성되어 있고, 서버 시스템은 UNIX, NT, OS/400, VMS 등 다양한 종류가 있고 공장 및 사무실에는 약 8000 대의 486 및 팬트엄급 PC 가 있다.

먼저 포스코 클라이언트/서버 시스템에 대한 특성을 살펴 보면 다음과 같다.

- 많은 다른 애플리케이션 컴포넌트와 모듈을 가지고 있다.
- 클라이언트수가 약 7500 개 정도이다.
- 지역별로 LAN 과 WAN 이 혼재된 아주 복잡한 환경에서 동작한다.
- 개발주기가 3 년 이상 긴 생명주기를 가진다.
- 온라인 트랜젝션이 많이 발생하는 업무가 대부분이다.
- 복수개의 서로 다른 데이타베이스, 서로 다른 서버 기종을 포함하고 있다.
- 시스템의 우선순위가 안정성으로 시스템 다운은 조업에 치명적이다.
- 업무 및 시스템이 계속적인 증가요소가 있다.

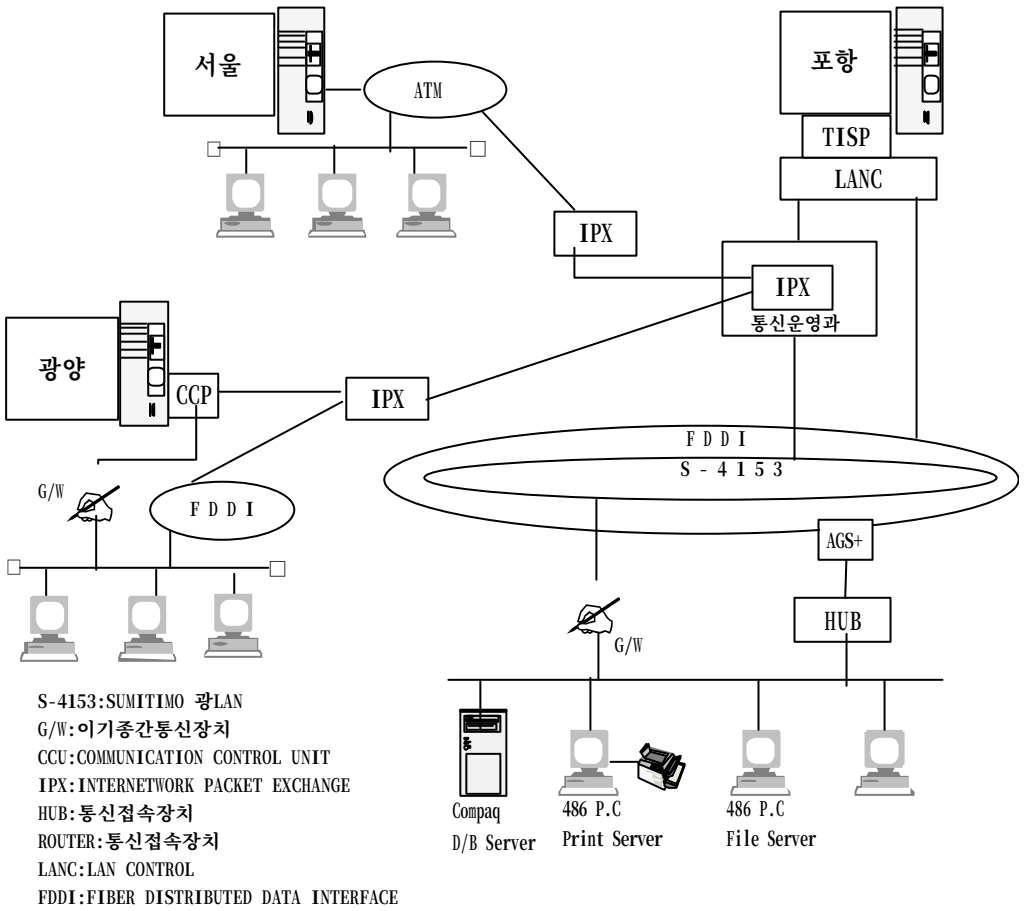


그림 11. 포스코 현 엔터프라이즈 시스템 구성도

클라이언트/서버 시스템 운영은 대부분의 기업에서와 마찬가지로 포스코에서도 기존의 메인프레임을 위주로 하고 부서별로 클라이언트/서버 시스템을 단계적으로 구축해 가고 있는 형태다. 부서단위에서 Graphic User Interface (GUI)기반의 개발툴을 가지고 RDBMS 의 데이타를 활용하는 2 계층 형태의 클라이언트/서버 시스템이 주를 이루고 있으며, PC 는 게이트웨이를 경유하여 메인프레임의 더미 터미날 용도와 서버의 클라이언트 용도로 사용된다.

메인프레임과 서버와 데이타 전송방법은 RDBMS 제품이 제공하는 Data Gateway 를 이용하고, 반복 수행되는 일에 대해 네트웍 트래픽을 줄여주기 위해 SQL 문의 집합체를 procedure 로 만든 Stored Procedure 을 이용하고 있다.

클라이언트/서버 시스템에서 주로 적용되는 업무는 메인프레임의 데이타를 정주기로 다운받아 부서단위로 통계분석이나 리포트 발행하는 등의 배치성 업무와, 부서 자체의 인사노무 업무 등에 활용된다. 이러한 RDBMS 위주의 시스템은 전통적으로 데이타 oriented 한 방식으로 클라이언트 애플리케이션은 서버의 RDBMS 의 데이타만 액세스하고 처리하기 위해 작성되어 있다.

3.2 시스템 아키텍처의 현상

클라이언트/서버 시스템 아키텍처는 1 세대로 불리우는 2 계층 구조가 전체 시스템의 90%을 차지하고 있다. 이는 메인프레임에서 클라이언트/서버 시스템으로 전환하는 대부분의 기업에서 적용한 것으로, 개발환경이 비교적 단순하고 구현이 용이하여 위험부담이 적다는 것이다. 또한 2 계층 구조의 특징인 부서단위 분산처리시스템형태로 업무형태가 Batch 성이 강한 통계분석 및 실적조회가 대부분을 차지하고 있다.

애플리케이션 구현 형태도 Embedded SQL 과 Stored Procedure 가 혼재되어 있어 업무특성이나 개발 편이에 따라 두 가지 모델을 동시에 사용하고 있으며, 실제 애플리케이션 로직의 위치가 성능에 영향을 미쳤다고는 볼 수 없다. 왜냐 하면 데이타베이스 서버모델 (Stored Procedure 를 이용)을 사용한 시스템이 주로

성능이 좋은 것으로 나타난 것은 아니기 때문이다.

전체 시스템 중 약 10%정도가 3 계층 구조로서, 전사를 대상으로 실적 수집을 주로 하는 인사노무 시스템으로서 동시 사용자수가 100 명 이상인 중대형 시스템에 속한다. 이 시스템은 최근에 개발된 시스템으로 순수형 3 계층 구조로서 하드웨어적으로 계층이 분리되어 있을 뿐, 애플리케이션 로직이 계층화 되지 않았다.

3.2.1 2계층 구조에서 발생되는 문제점

가장 큰 문제점으로 성능이 저하하는 것이다. 동시 사용자가 적은 시스템의 약 80% 는 3~5초 이내의 응답시간을 나타내지만 동시 사용자수가 100 인 이상인 경우는 성능이 현저하게 떨어져 10 초 이상의 응답시간대를 나타냈다. 물론 사용자수와 상관없이 응답속도가 늦은 것은 시스템 구조상의 문제이기 보다는 데이타베이스 설계 및 업무설계 잘못으로, 데이타베이스 및 SQL 튜닝으로 성능향상이 가능하다고 생각 된다.

둘째, 시스템 안정성이 부족하다는 것이다. 대부분의 시스템이 부서고유의 통계분석 업무이기에 큰 지장은 없다고는 하지만 서버 시스템과 데이타베이스 시스템이 자주 다운된다. 클라이언트/서버 시스템 개발초기 운영관리 경험부족에서 원인 분석 및 장애복구에도 많은 시간이 소요되고 있다.

셋째, 시스템 활용성 저하다. 대부분의 시스템이 부서 단위 단독시스템, 즉 하나의 서버에 하나의 데이타베이스 서버로서 운영되는 것으로 메인 프레임과의 연계 활용이 어렵다.

마지막으로, 개발생산성 저하다. 2 계층의 가장 큰 장점이 시스템 구현이 쉽다는 것인데, 개발생산성 저하는 개발 경험 및 개발표준, 기술력 부족 등으로 기인한다. 즉 HOST-Based 보다는 기술의존도가 상당히 크며 개발자의 기술에 좌우되어, 정상적인 기술 수준일 경우 HOST 개발생산성보다 4~10배 정도 높다.

3.2.2 성능 저하 요인

성능이 떨어지는 가장 근본적인 원인은 동시 사용자수가 많아지면서 네트웍 트래픽이 다량 발생하고, 데이타베이스에 대한 접근이 집중됨으로 인한 병목현상(bottleneck)이 발생한다.

다음으로 메인프레임 형태의 업무 및 설계로 특정 데이타베이스에 부하가 집중되는 현상이 발생하고, 또한 애플리케이션 로직 설계에 있어서 SQL 의 잘못된 사용으로 네트웍 트래픽을 증가 시켜 성능 저하의 원인이 된다.

3.3 미들웨어의 현상

각 시스템별 인테페이스 형태에 따라 미들웨어 사용 형태를 알아 본다. 대부분이 2 계층 구조의 클라이언트/서버 시스템으로 서버의 데이타베이스 연결용 미들웨어가 많다.

? 메인프레임 데이타를 서버에 연결할 때

정주기 (1 시간, 조별, 일별) 데이타는 대부분 File Transfer Protocol (FTP)에 의한 file 단위 전송으로 메인프레임의 스케줄러에 의해 기동 된다. Socket 통신방식은 메인프레임의 데이타를 편집하여 5 분 또는 그 이상의 주기로 서버에 전송하면 서버는 이를 받아 데이타베이스에 등록하는 형식이다. 이로 인한 문제점으로 메인프레임의 스케줄러 기동 시간에 부하가 집중되고, 개발자가 서버별로 모든 Socket 프로그램을 작성해야 한다.

? 클라이언트가 서버 데이타베이스를 액세스 할 때

2 계층에서 많이 사용되는 데이타베이스 연결용 DB-library 및 ODBC 가 대부분을 차지하고, 그 외 2 계층과 3 계층 구조의 혼합형태에서는 시큐링크(SequeLink)를 사용하고 있다.

? 다른 서버의 데이타베이스를 연결할 때

대부분의 클라이언트/서버 시스템이 부서 단위의 단독업무를 수행하는 것으로 타 서버의 데이타를 액세스해야 할 업무가 거의 없다. 다른 서버 데이타

필요시는 사용자가 파일 편집 프로그램을 작성하여 파일을 전송한다. 이는 원하는 서버의 데이타베이스 구조를 알기가 어렵고 보안을 이유로 데이타베이스 접근에 상당한 제약이 있다. 클라이언트/서버 시스템이 오픈시스템을 추구하는 만큼 모든 서버에 자유롭게 접근 가능하도록 하기 위해서 데이타베이스 및 테이블 설계표준, Naming 표준, Data Dictionary 구축이 반드시 필요하다.

3.4 시스템 운영의 현상

클라이언트/서버 시스템의 가장 큰 장점이 사용자에게 편리한 인터페이스를 제공하는 반면에, 클라이언트의 애플리케이션이 많아지고 복잡해 짐에 따라 시스템 보안, 장애에 대한 복구, 데이타 백업 등의 유지보수에 상당한 비용과 노력이 필요하다.

3.4.1 장애발생 유형

운영성 측면에서도 가장 큰 문제점이 시스템 안정성에 대한 대책이 부족하다는 것이다. 클라이언트/서버 시스템이 초기 기술력이 완전히 정착되지 않은 시점에 데이타베이스 및 서버 시스템에 대한 원인을 알 수 없는 장애가 다량 발생한다. 또한 클라이언트 수가 증가하면서 애플리케이션 부문의 장애가 많고, 클라이언트 환경설정 파일의 사용자 임의 변경으로 인한 장애도 상당부분 발생하고 있다.

클라이언트/서버 시스템 개발경험 및 표준화부족, 장애관리 툴 부재 등으로 장애가 다량 발생하고, 또한 장애 이력관리 미흡으로 유사 장애가 계속 발생하고 있다.

3.4.2 애플리케이션 배포방법

개발 초기에는 각각의 클라이언트에 일일이 애플리케이션을 등록하는 방법을 사용해 왔으나, 현재는 대부분 서버에서 해당 클라이언트에 일괄 다운로드하는

형태로서 버전을 관리, 비교하여 최신 버전만 다운로드 한다. MIPOS (포스코 자체 개발한 그룹웨어)에 의한 다운로드는 시스템 개발이 완료되고 모든 클라이언트에 일괄적으로 애플리케이션을 등록할 때 사용하는 것으로, 네트웍 및 시스템 부하가 동시에 발생하고, 수정시 마다 전체를 다운로드해야 하는 문제점이 있다. 애플리케이션 버전 및 배포에 대한 표준정립 및 관리통제가 필요하다. 실제로 개발 중 Tool version up 발생으로 이미 개발된 프로그램을 모두 재 conversion 한 사례가 있다.

3.4.3 백업 (Back up) 방법

백업은 데이타베이스 백업을 주로 의미하며 시스템 백업은 특별한 서버에 한에서만 시행되고 있다. 가장 일반적인 백업 방법으로 데이타베이스 명령어를 사용하여 주기적으로 수동 백업하는 것으로 주로 단독개소에 있는 서버들이 이 방법을 사용한다. 대부분의 서버가 전산실에서 일괄 관리되는 서버들로서 백업 전용장치를 이용하여 스케줄링에 따라 온라인 자동 백업을 실시한다. Replication 서버를 이용하여 대상 데이타베이스를 선정하고 해당 데이타베이스가 변경될 때마다 실시간으로 백업되는 형태로서 추가 디스크가 필요하며, 아주 중요하고 실시간성을 요구하는 업무에 적용된다.

CPU 백업은 중요한 시스템에 한에서 듀얼시스템 (Dual System)으로 운영된다. 방법으로는 여러 대의 서버 중 1 대를 백업서버로 지정하는 방식, 하나가 fail 나면 동일한 환경을 가지고 백업서버가 실행되는 비대칭형, 서로 서로 백업하는 대칭형이 업무 유형에 따라 사용되고 있다.

3.4.4 개발시 우선순위

모든 시스템과 마찬가지로 클라이언트/서버 시스템에 있어서도 업무설계, 데이타베이스 설계가 가장 우선시 된다. 이는 클라이언트/서버 시스템 개발 이전에 업무 로직 (Business logic)을 재설계함이 선행되어야 한다.

다음으로, 하드웨어 견적 및 데이타베이스 용량견적에 대한 명확한 기준이

없어 시스템 효율이 저하된 측면이 있다.

지금까지 포스코 클라이언트/서버 시스템 애플리케이션에 대한 현상과 문제점을 부문별로 살펴 보았다. 이러한 실제 운영상의 문제점을 토대로 효율적인 클라이언트/서버 시스템 구현 방안에 대해 살펴 보겠다.

4. 3 계층 구조의 클라이언트/서버 시스템 구현 방안

이 장에서는 기업 전체에 적용 가능한 3 계층 구조의 엔터프라이즈 클라이언트/서버 시스템의 적용방안, 적용시의 고려사항, 요소기술 등에 대해 설명한다.

4.1 아키텍처 구성 방안

기업에서 엔터프라이즈 클라이언트/서버 시스템 프로젝트를 계획할 때 특성을 무시하고 단순히 하드웨어 증설만으로 해결 가능한 것으로 생각한다든지, 현재 나와 있는 상용툴이 이들을 자동으로 지원해 준다고 믿는 경향이 있다. 이는 개발자들이 엔터프라이즈 클라이언트/서버 시스템에 대한 정확한 기술을 가지고 있지 않기 때문이다. 엔터프라이즈 클라이언트/서버에 대한 특성을 충분히 반영하여 효율적인 시스템 구축 [그림 12]을 위해 고려 되어야 할 내용은 다음과 같다 [9,26].

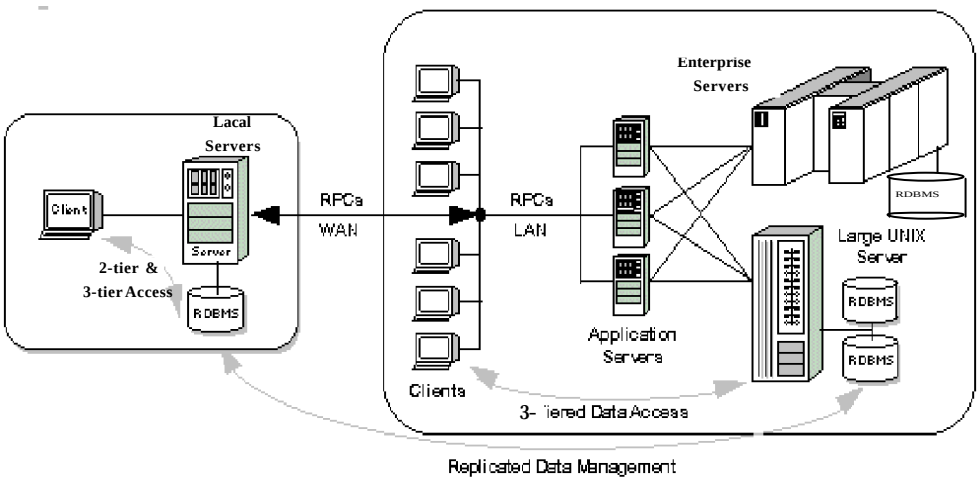


그림 12. 3 계층 클라이언트/서버 시스템 구현 모델

첫째, 업무 유형에 따라 애플리케이션 서버 구축 방법을 달리하여 성능을 향상하는 것이다. 먼저 온라인 트랜젝션이 많고 사용자수가 많은 업무는 DB 서버와 복수개의 애플리케이션 서버를 운용하는 방법이다. 이는 특정 애플리케이션 서버

에서 사용자가 많아지면 다른 서버들은 idle 상태가 된다. 이를 미리 애플리케이션 서버간에 서로 전달하여 전체적으로 부하가 균등하게 분배되도록 하기위해 애플리케이션 서비스를 분산한다. 이때 전용 애플리케이션 서버는 최대 50 대를 넘지 않는 제한된 숫자로 두어야 한다. 다음으로 운영의 효율성을 제고하고 비용 측면에서의 서버추가 도입을 억제하는 방안으로 트랜젝션이 많지 않은 일반적인 업무는 애플리케이션 로직을 별도로 구성하되 네트웍 트래픽을 줄이기 위해 데이타베이스 서버에 함께 두는 방법이다. 이때 물론 기존의 데이타베이스 서버보다는 강력한 기능을 지녀야 한다.

둘째, 시스템 확장성을 위한 방안으로 데이타베이스의 위치나 공급자에 상관없이 이질적인 DBMS 를 계속 연결시켜 확장이 가능하며, 서버에 많은 부하가 걸릴 때 부하를 균등하게 분배하는 부하분산이 가능한 애플리케이션 서버를 구축한다.

셋째, 시스템 안정성을 위한 방안으로 기업의 중요 데이타베이스와 성능을 우선으로 하는 업무는 메인프레임을 엔터프라이즈 서버로 구성하고 master 데이타베이스를 둔다. 또한 부하조절 기능을 이용한 장애대책으로 애플리케이션 서비스를 분산하여 서버가 장애가 발생하면 다른 서버로 전환하는 방법과 특정서버가 장애가 발생하면 서비스 자체를 다른 서버에 이관 (Migration)하는 방법이 업무 유형에 따라 구분하여 적용한다.

넷째, 복수개의 플랫폼과 리소스를 메인프레임 형태의 보안을 제공한다. 외부적 보안사항은 기본적으로 파이어월 (firewall)을 사용하고 시스템측면에서의 보안 기능은 4 가지 수준으로 구성한다. 1 단계는 OS 에서 제공하는 읽고 쓰기 및 실행 권한 (File Permission), 2 단계는 애플리케이션에 패스워드를 적용 (Password), 3 단계는 미들웨어에서 제공하는 인지 (Authentification) 서비스에 의해 사용자의 권한을 검증하고, 4 단계는 특수 업무에 대한 별도의 보안 시스템을 구축한다.

다섯째, 운영성이나 유지관리를 고려한 시스템 구축이 필요하다. 애플리케이션 로직을 모듈화하여 컴포넌트 (Component) 기반의 객체지향 기법 [21]으로 구현 하므로서 변경관리나 배포 등의 유지비용을 줄이고, 컴포넌트를 다른 애플리

케이션 개발시에 재사용할 수 있도록 한다.

여섯째, 전체시스템 아키텍처를 고려하여 시스템을 구축해야 한다. 집중된 컴퓨터 환경으로부터 가능한 많은 프로세싱을 덜기 위해서는 리모트의 서버를 이용한다.

일곱째, 중앙에 집중된 컴퓨팅 환경에서 전체 시스템을 관리 감독할 수 있도록 한다. 대부분의 수정 및 변화가 관리자에 의해 중앙에서 이루어 지므로 버전 관리, 유지 보수, 백업, 시스템 튜닝 등의 관리가 필요하다.

여기서 가능한 기존의 시스템 환경을 최대한 활용하면서 현상 문제점을 해결할 수 있는 방안을 마련하고자 했다.

4.2 서버 시스템별 적용업무

모든 클라이언트/서버 시스템이 일률적으로 3 계층 구조의 클라이언트/서버로 가는 것은 바람직하지 않다. 기존의 부서단위 클라이언트/서버 시스템을 구축하여 비교적 효율적으로 활용하고 있으며, 다소의 응답시간 저하는 클라이언트/서버 시스템 구축 경험부족에서 오는 것으로 퍼포먼스 튜닝작업, 업무 및 데이타베이스 재설계 등으로 충분히 해결 가능한 문제다. 따라서 업무 유형, 사용자수, 시스템 규모에 따라 아키텍처를 구성해야 한다. 그래서 아키텍처 구성 기준은 다음과 같다.

첫째, 로컬 서버 시스템은 기존에 구축되어 있는 2 계층 형태의 시스템으로 서버에는 데이터베이스와 Stored procedure 로 구성되어 있고 클라이언트에서는 애플리케이션 로직을 포함한 사용자 인터페이스가 놓이는 형태로서 다음의 경우에 주로 적용한다.

? 데이타 발생이 대부분 부서내에서 발생되고 수정이 빈번하게 일어나는 업무
? 데이타베이스 규모가 50GB 미만의 소형 시스템에 적용한다.
? 10 초 이내의 response time 을 요하는 비교적 퍼포먼스가 크게 중요시 되지

않은 업무

? 사용자수가 100인 이하로서 부서 단위 소규모 업무에 적용.
? 대부분의 데이타가 단일 데이타베이스로 구성되어 있는 업무로서 타 부문과 인터페이스가 적은 독립된 업무
? 사용자가 보다 쉽고 편리하도록 하는 시스템의 유연성을 필요로 하는 업무
? 주요 적용 업무는 대부분이 Batch성 업무로서 통계처리, 분석업무, 리포트 발행 등의 업무에 적당하다.

둘째, 부문 서버 시스템은 서버에 데이터베이스와 애플리케이션 로직을 두고 클라이언트에서는 사용자 인터페이스만을 두는 형태다. 부서간에 여러 개의 서버 데이타를 결합해서 액세스가 필요할 때 적용되는 업무로서 주요 기준은 다음과 같다.

? 데이타 발생이 여러 부서에서 발생하고 데이타 갱신보다는 검색을 주로 하는 업무
? 데이타베이스 규모가 50~200GB 규모의 중대형 업무
? response time이 3~10초 정도로서 퍼포먼스에 직접적인 영향을 주지 않은 업무
? 사용자수가 100~500명 정도의 시스템에 적당하다.
? 온라인 업무가 대부분이지만 리얼성을 요하지 않은 업무
? 주요 적용 업무는 대부분의 부서에서 공통으로 사용하는 메일 서버, 웹 서버, 그룹웨어 등의 업무에 적당하다.

셋째, 엔터프라이즈 서버 시스템은 엔터프라이즈 서버에는 데이터베이스를 두고 애플리케이션 서버에 애플리케이션 로직, 클라이언트에는 사용자 인터페이스 기능을 두는 형태의 진정한 의미의 3 계층 구조를 말한다. 기존의 메임프레임에서 운영해온 업무 중 연중 무휴로 운영되는 업무, 분할이 쉽지 않은 대형 데이타베이스를 가지 업무가 적당한 것으로 구체적 기준은 다음과 같다.

? 월 평균 100,000 건 이상의 온라인 트랜젝션이 많이 발생하는 업무
? 부하시간이 특정 시간대에 집중되는 경우
? 데이타 생성 및 수정이 여러 부서에서 발생하는 업무
? 데이타베이스 규모가 200GB 이상
? response time이 3초 이내의 빠른 응답을 필요로 하는 업무
? 클라이언트수가 500 명 이상의 대형 업무
? 고도의 보안을 요하는 업무
? 시스템이 가져야 할 우선 순위가 안정성, 정확성, 신속성의 순서
? 주요 적용 업무는 인사, 노무, 자금관리 등의 업무에 적당하다.

이러한 분류기준은 기업에 따라 업무 특성에 따라 다소의 차이는 있을 수 있으나 전체적인 범주는 크게 벗어나지 않는다.

4.3 애플리케이션 구성방안

애플리케이션 구성 방안은 그림 13 에서 처럼 데이타베이스와 애플리케이션 로직을 완전히 분리하여 두는 것으로 사용자는 GUI 로 부터 데이타를 입력받아 메시지를 편집하여 애플리케이션 서버에 전송하면 애플리케이션 서버는 데이타베이스 서버에 데이타를 요청한다.

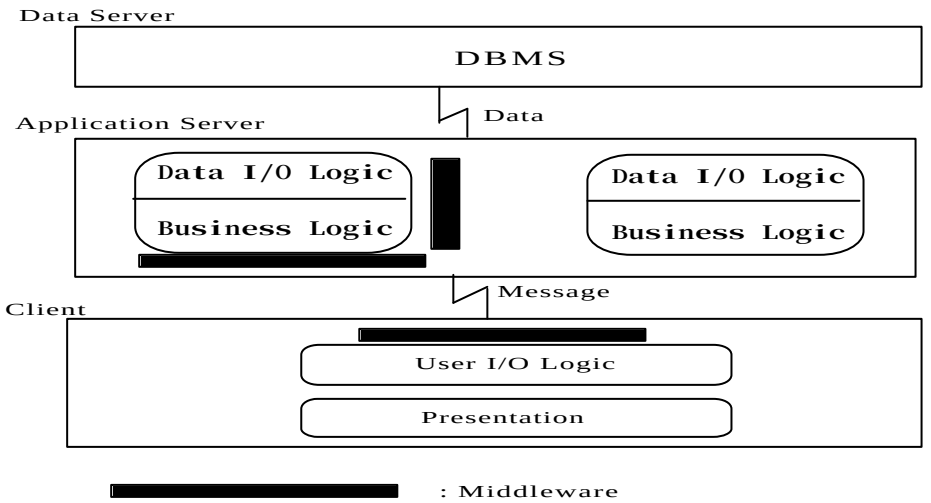


그림 13. 애플리케이션 아키텍처

클라이언트에서는 GUI 화면으로 부터 데이타를 입력받아 애플리케이션 서버로 파라메타 (parameter)를 편집하여 서비스를 call 하면 애플리케이션 서버는 해당 결과를 클라이언트에 보내면 이것을 화면에 출력하는 프리젠테이션 기능이 있다.

애플리케이션 서버는 클라이언트로부터 받은 서비스 요구에 대한 응답을 주로 하고 주요 기능은 다음과 같다. 2-Phase Commit protocol 를 이용하여 데이타의 일관성을 유지한다. 시스템 및 네트웍 상의 장애나 데이타베이스의 Dead-Lock 등이 발생하면 자동으로 복구작업을 수행하고, 전체 분산시스템에 대한 포괄적인 감시와 관리를 중앙에서 집중처리 한다. 또한 Resource 들의 상태를 항상 알고 있으면서 애플리케이션에 대해 높은 가용성 (High Availability)를 제공하고 전체적으로 모든 애플리케이션 서버에 부하가 균등하게 분배되도록 하여, On-Line System 의 처리량을 최대로 유지하고, 비동기 큐는 트랜젝션 완료 신뢰성을 보장한다.

데이타베이스는 애플리케이션 서버로 부터 데이타 요청을 받고 서비스를 해주는 주된 기능 외에 데이타 무결성 보장, 데이타 접근에 대한 권한, 데이타 장애에 대한 복구 등의 기능을 수행한다.

4.4 3 계층 구조 구축시 고려사항

3 계층 구조는 많은 장점을 가지고 있지만 구현 사례가 많지 않고 또한 메이커 (maker)에서 나온 기술 또한 완전하게 검증된 것이 아니다. 따라서 클라이언트/서버 시스템 구축에 있어서 제품에 대한 단편적인 기술이 모든 것을 해결해 주는 것으로 인식하면 안된다. 따라서 보다 포괄적이고 전체 시스템을 감안한 시스템 구축이 필요하며 이에 대한 고려사항과 예상되는 문제점에 대해 알아 보면 다음과 같다.

시스템 개발에 앞서 표준을 먼저 만들어야 한다. 바로 이것이 Solution

Architecture 이다. Solution Architecture 란 클라이언트/서버 애플리케이션을 성공적으로 구현하기 위한 첫번째 단계로서, 위험을 줄이고 좀 더 빠르고 예측 가능한 개발을 할 수 있도록 도와주는 프레임웍 (Framework)이다 [27].

- 개발 환경의 표준화 작업을 수립하는 것이다.
  불확실성을 줄이고, 각각의 노력을 효과적으로 통합할 수 있다. 또한 표준을 따를 때 개발을 간단하고 편리하게 한다. 그래서 일반적으로 이해하기 쉽고, 잘 정의된, 폭 넓은 지원이 가능한 표준을 채택하고자 한다. 예를 들면 GUI, Naming Convention, Network Protocol, Component Standard & project control standard 등에 대한 표준이다.

- 클라이언트와 서버 툴의 선정이다.
  툴의 기능을 전적으로 사용하는 Packaged Software 를 사용할 것인가, 부분적으로 자체 개발할 것인가를 판단 해야 한다.

- 디자인 접근 방법으로 개발방법론의 선택이다.
  개발툴과의 연계방안, 공정별 프로토타입 구축 방안 등, 설계방법에 있어서 객체계념 [30]의 도입여부와 관계형 데이타베이스로의 설계 등의 고려가 필요하다.

사전에 이와 같은 프레임웍을 정의해 둠으로서 새로운 애플리케이션 개발에 대한 사전교육을 줄이고, 계속해서 추가, 보완해 감으로서 튼튼한 안내자로서 추가적인 개발시간을 줄일 수 있다. 다음으로는 클라이언트/서버 시스템 구축시 일반적으로 고려해야 할 사항이다.

- 2 계층 구조가 대부분 단순 데이타베이스를 액세스하는 반면 3 계층 구조는 애플리케이션 로직을 액세스 하기 때문에 반드시 업무혁신이나 표준화를 통해 업무 로직을 명확하게 정의해야 한다.
- 분산객체에 대한 이해와 분석과 설계, 구현 등의 기술력 확보가 선행되어야 한다.

- 3 계층으로의 이전업무의 우선순위를 결정하여, 신규 개발업무로서 위험 부담이 적은 업무에 우선 시범 적용한다.
- 소프트 웨어 재사용, 장애대책 등의 운영비용의 절감을 고려한 시스템 구축이 필요하다.
- 시스템간의 업무분장 및 데이타 관리 기준을 정립하여 이중관리를 배제한다.
- 정보기술의 발전에 유연하게 대처하기 위한 기술 표준을 따른다.
- 초기 기술력 부족으로 개발 생산성의 저하가 우려 되고 시스템에 문제가 발생되었을 때 해결시간이 지연 될 수 있다.
- 툴가격과 기타 용역비 중복으로 인한 추가적인 비용 소요가 예상된다
- 프로그래밍이 어렵고 새로운 기술을 습득해야 함으로 전문인력이 필요하다.
- 서버측에 프로세스가 항상 상주하므로 년중 무효 운영이 필요한 업무에 잘못 적용시 비효율적일 수 있다.
- 인터넷과의 연계를 고려한 시스템 개발이 필요하다.

4.5 툴 선정 기준

여기서는 클라이언트 개발툴과 미들웨어 선정 기준에 대한 일반사항을 알아보겠다.

4.5.1 클라이언트 개발툴

3 계층 클라이언트/서버 시스템에서 클라이언트는 정보를 온라인으로 신속히 제공함으로서 일반 사용자에 보다 편리한 환경을 제공해야 한다. 그래서 클라이언트 개발툴이 가져야 할 기능은 다음과 같다 [28].

- 16 비트 코드 (윈도우 3.1)에서 작성된 애플리케이션을 윈도우 95 에서 별 다른 코딩없이 다시 컴파일하려면 32 비트 애플리케이션을 생성할 수 있는 이식성이 있어야 한다.
- 배우기 쉽고 코딩이 용이한 스크립트 언어이어야 한다.

- 클래스 (class)를 생성할 때 부모 클래스의 속성 (attribute)을 상속 받고 부모 클래스와 다른 부분만 생성하고, 동일한 하나의 메시지에 대해 서로 다른 일을 수행할 수 있게 한다.
- 클래스 또는 함수 라이브러리를 제공해야 한다.
- 데이타베이스 벤더나 서버 기종에 상관없이 데이타베이스에 접근이 용이해야 한다.
- CASE 툴과의 연결이 가능하여 개발 툴 화면으로 자동 생성이 가능해야 한다.

개발 툴 선정시에 어디에 중점을 두느냐에 따라 달라 지지만 공통적으로 차세대 개발 툴은 객체지향 원칙에 상당히 기반을 두고 있다.

4.5.2 미들웨어 선정 기준

다음은 미들웨어 선정에 관련된 내용으로 시스템 적용을 위해서 고려되어야 할 사항이다.

- 기술추세의 변화에 대응하는 제품으로 다양한 기능을 포함한 미들웨어를 선택해야 한다. 애플리케이션 서비스는 트랜잭션을 모니터하고, SQL을 최적화, 라우팅 (Routing), 데이타베이스 리플리케이션 (Replication) 등의 다양한 기능을 지원해야 하고 이러한 기능들을 통합하는 기술 호환성이 무엇보다 중요하다.
- 현재의 시스템 환경뿐만 아니라 미래지향적 기술 접근이 필요하다. 현재는 데이타베이스 상에서 다양한 DB호환성을 가지고 사용해 왔지만, 향후 기술 추세는 객체지향의 개념을 가지고 RDB에만 국한되어 사용되지 않고 Object Oriented Database (OODB) 등 다양한 데이타베이스를 지원할 수 있는 미들웨어 선택이 필요하다.
- 다양한 종류의 애플리케이션 타입을 지원해야 한다. 시스템간에 호환성은 메시지 핸들링 (Message handling), 정보공유, 다양한 API의 지원 등 많은 요소가 있다.

- 표준화에 준하는 미들웨어를 선택해야 한다. 클라이언트/서버 시스템을 개발할 때 기본프로토콜이 RPC였다. OSF의 DCE는 일찍부터 이기종간 데이타 전송을 위하여 RPC에 의한 커뮤니케이션, 디렉토리 및 Naming에 관한 표준을 정하여 네트웍 인프라쪽을 광범위하게 커버하면서 애플리케이션 개발시 H/W에 관계없이 소프트웨어를 개발할 수 있는 환경을 제공한다. 현재의 애플리케이션 프로그램 개발은 객체지향의 개념에 의해서 만들어진 라이브러리를 사용하여 개발기간을 단축하고 효과적으로 애플리이션을 개발한다. 또한 인터넷의 급속한 보급은 미들웨어 시장에 변화를 가져왔는데 인터넷 지원이 능한 CORBA의 위치 강화로 나타나고 있으며 현재 CORBA 중심의 표준화로 재편되고 있다.
- 데이타베이스, 통신, 네트웍 등의 모든 것을 동시에 고려해야 한다. 향후 미들웨어 시장에 직접적인 영향을 줄 수 있는 기술은 인터넷 기술이다. 미들웨어의 목적이며 이점인, 개발자가 서로 다른 환경에서 운영체제나 데이타베이스에 관계없이 애플리케이션 개발할 수 있는 요건은 미들웨어 뿐만 아니라 인터넷상에서도 가능하다.

결론은 미들웨어에 있어서 모든 것을 해결할 수 있는 최상의 솔루션은 없다. 단지 얼마나 많은 기능을 지원하느냐이며 이 문제는 비용과 기능의 Trade-Off 관계를 가진다. 미들웨어 환경을 고려할 때 향후 정보시스템의 방향을 예측하여 비용상의 문제와 함께 표준화의 문제도 검토하여야 한다. 잘못된 제품의 선택은 미들웨어의 기능 자체를 상실하기 때문이다.

지금까지 엔터프라이즈 클라이언트/서버 시스템 구성방안에 대해서 여러 측면에서 알아 보았다. 현 기업의 클라이언트/서버 환경에서 발생되는 문제점을 해결하는 방안으로 3 계층 구조를 구축하는데 있어서의 아키텍처 및 애플리케이션 구성방안, 구축시 고려사항, 클라이언트 개발툴과 미들웨어 선정기준에 대해서 기술하였다.

5. 3 계층 구조의 클라이언트/서버 시스템 적용

대부분의 중요 시스템이 IBM, FACOM 과 같은 대형 기종의 메인프레임으로 구성되어 있고, 근래에 클라이언트/서버 시스템 도입으로 부서 단위 통계분석을 목적으로 부분적으로 구축 되어 지고 있다. 이 장에서는 4 장에서 제시한 3 계층 구조의 클라이언트/서버 시스템의 구현방안을 포스코 시스템에 적용해 보고자 한다.

5.1 포스코 시스템 아키텍처 구성

앞에서 설명한 두가지 형태의 네트웍 구성을 표준 네트웍 프로토콜인 TCP/IP 로 일원화 하고 기존의 메인프레임을 개방형 시스템으로 전환하고, 이를 기반으로 포스코 3 계층 클라이언트/서버 시스템을 구성하게 되는 주요 기준과 아키텍처 구성도 [그림 14]는 다음과 같다.

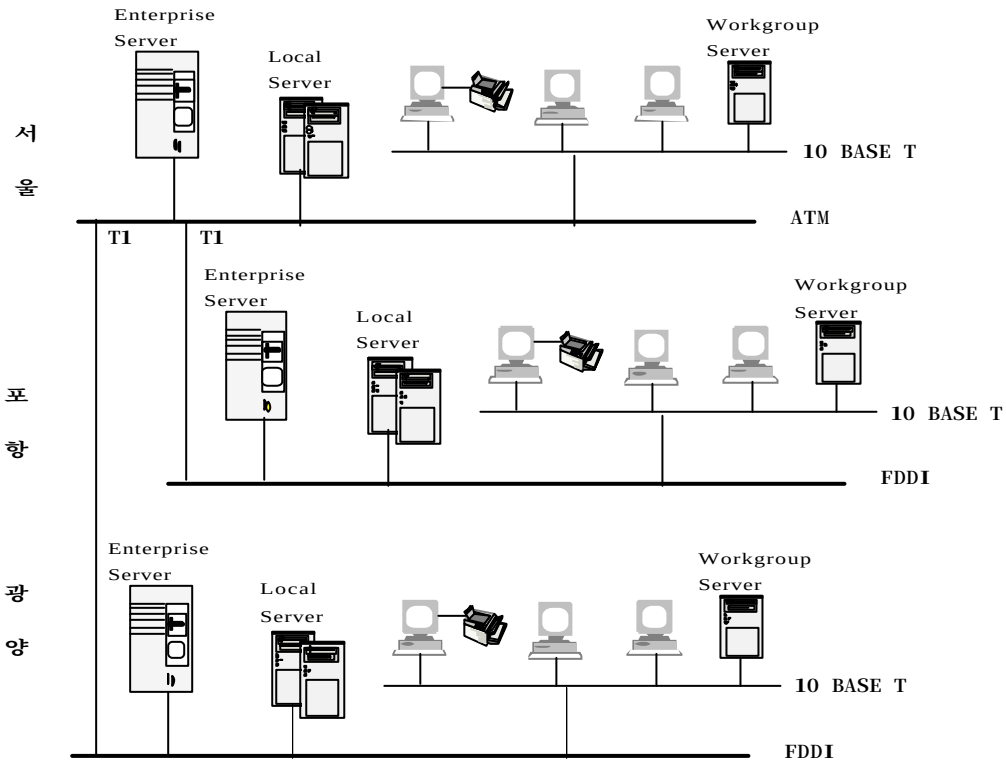


그림 14. 시스템 아키텍처 구성도

- 시스템 구성요건이 최우선 순위가 안정성이다. 현재의 중소형 서버급 시스템으로는 대규모의 기업의 업무를 소화하기 어렵다는 판단에서 중대형 엔터프라이즈 서버를 구축한다.
- 지역별 업무가 대체로 구분되는 업무는 네트웍 트래픽을 최소화하기 위해 지역별로 엔터프라이즈 서버를 두는 것으로 한다. 그래서 서울에는 일반관리 및 판매관리 업무를 두고 포항에는 공정관리 업무를 둔다.
- Wan 간의 트래픽이 많은 업무는 지역별 별도의 서버를 둔다. 전자메일이나 그룹웨어 업무는 지역별로 별도로 설치한다.
- 운영관리의 효율성을 위해 서버 기종과 오퍼레이팅 시스템도 2~3종으로 제한한다 [표 3].
- 효용성 측면에서 모든 시스템에 애플리케이션 전용서버 도입을 억제하고 애플리케이션과 데이타베이스 서버를 통합 구축한다. 많은 애플리케이션 전용 서버를 두는 것은 오히려 네트웍 트래픽을 유발시켜 성능 저하를 가져 온다.

구분	하드웨어	OS	DB 규모	비고
Enterprise Server	IBM	MVS	250GB이상	기간 업무
Local Server	SUN, HP	UNIX계열	100~250GB	지역 공통업무
Workgroup Server	Compaq	Windows NT	100GB 이하	부서단위 업무
Client	PC 486 이상	Windows 95	2GB	개인 업무

표 3. 시스템 사양

- 시스템간의 인터페이스 방법을 개선한다. 첫째, 기존의 서버와 호스트간의 인터페이스는 정주기 단위로 FTP 에 의한 파일전송으로 호스트에서 정주기 시간대 파일편집 및 전송으로 부하가 증가하는 단점을 보완하기 위해 데이타/메시지 전송방식으로 전환하여 클라이언트가 요구시 마다 전송 가능하도록 한다. 둘째로 클라이언트에서 ODBC 나 DB-Library 를 이용하여 직접 SQL 를 발행하거나 스토어드 프로시저를 호출하는 형태에서 애플리케이션을 완전히 분리한다. 이렇게 함으로서 좀 더 유연하고 확장 가

능한 시스템을 구현 할 수 있게 해 준다.

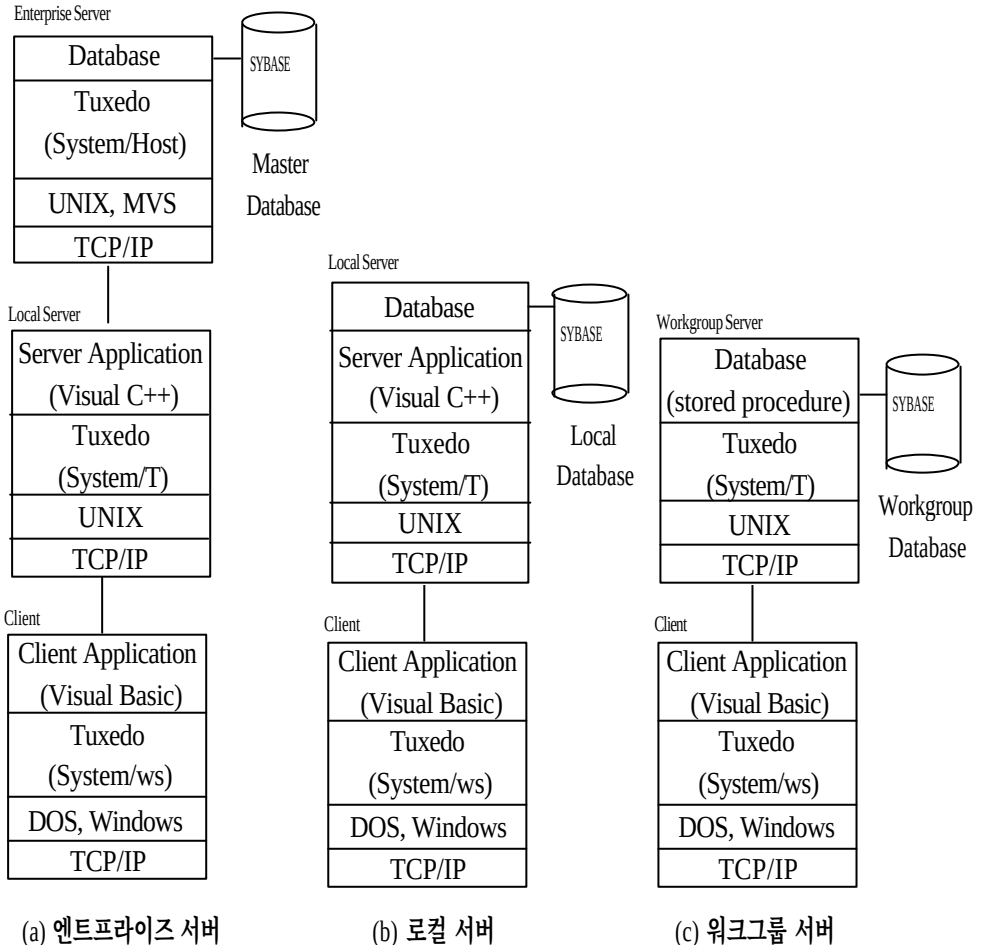
- 네트웍 보안으로 자체 통신망에 방화벽(firewall)을 설치하고 시스템측면에서의 보안 기능은 4가지 수준으로 되어 있는데 OS에서 제공하는 읽고 쓰기 권한, 애플리케이션 패스워드, 미들웨어에서 제공하는 인증서버에 의한 사용자 권한 부여, PC는 부팅시 패스워드를 입력하도록 한다.
- 애플리케이션 서비스 구현은 애플리케이션을 단순히 분리한 것 만이 아니라 재사용, 유지보수 등을 위하여 Component 기반의 객체지향 기법으로 구현한다. 엔터프라이즈 애플리케이션 서비스 서버와 로컬 서버의 서비스는 비주얼 C++로 구현하고, 워크그룹 애플리케이션 서비스는 비주얼 베이직으로 구현하고자 한다.
- 클라이언트 개발툴은 기존의 비주얼베이직 3.0에서 개발경험을 토대로 Component 기반의 분산 애플리케이션 구축 툴인 비주얼베이직 4.0을 도입 적용함으로서 개발생산성 향상을 향상시킨다.

앞서 4 장에서 제시한 시스템별 적용 업무 기준에 따라 포스코 시스템에 적용하면 기능과 역할은 다음과 같다.

- 엔터프라이즈 서버는 전사적 공통업무 및 동일 지역내 부서간의 인터페이스가 많은 극히 안정성을 요하고 퍼포먼스에 영향을 받는 업무로서 마스터 데이타 베이스 서버의 역할을 하는 것으로 전사 단위의 인사노무, 공정관리 등의 시스템이 여기에 속한다.
- 로컬 서버는 동일 지역내에서 타부서간의 인터페이스가 대부분이고 부분적으로 타 지역간의 인터페이스도 발생하는 업무에 적용되는 것으로 크게 두가지 중요한 기능을 한다. 먼저 로컬 데이타베이스 서버로서 전자메일, 그룹웨어, 웹서버로 활용되고, 동시에 엔터프라이즈 서버에 대한 애플리케이션 서버로 활용된다.
- 워크그룹용 서버는 부서내의 단독업무가 주를 이루는 통계분석 업무로서 현재 대부분의 클라이언트/서버 시스템이 이 형태로 구성되어 있다. 향후에도 개발의 편이성으로 단독업무는 이 형태도 바람직하다고 생각된다.

5.2 애플리케이션 아키텍처 구성

지금까지 시스템 아키텍처 구성 기준 및 역할에 대해서 알아 보았으며 이 절에서는 애플리케이션 아키텍처 구성기준에 대해 알아 보기로 한다. 애플리케이션 구성방안은 업무 및 시스템 유형에 따라 세 가지로 구축하고자 한다 [그림 15].



(a) 엔트프라이즈 서버 (b) 로컬 서버 (c) 워크그룹 서버

- System/ws : 서버와 클라이언트과의 연결을 담당
- System/T : 서버에 위치하는 것으로 통신기능, 트랜젝션 제어기능, 시스템 관리 기능, 네임서버(Name Server) 기능을 담당
- System/Host : 서버와 메인프레임과의 연결담당

그림 15. 애플리케이션 아키텍처 구성도

첫째, 엔터프라이즈 서버 중심으로 구성되는 아키텍처가 진정한 의미의 3 계층 구조다. 즉 물리적 구조와 논리적 구조가 동일한 형태로서 클라이언트-애플리케이션 서버-데이타베이스 서버로 구성되는 것으로 안정성이 요구되는 업무, 퍼포먼스가 중요시 되는 업무, 절대적인 보안을 요하는 업무 즉 인사노무, 공정관리 등이 이에 속한다.

둘째, 로컬 서버는 두가지 기능을 한다. 먼저 애플리케이션 전용서버 로서 역할과 로컬 데이타베이스 서버로 활용된다. 로컬 데이타베이스 서버에는 지역안에서 (여기서 지역이라 함은 서울, 포항, 광양을 칭함) 전부서에서 활용되는 업무로서 즉, 전자메일, 그룹웨어 등이 적용된다. 애플리케이션 아키텍처 형태는 로컬 서버에 데이타베이스와 애플프리케이션 서비스가 들어가고 클라이언트에서 접근하는 형태의 물리적으로 2계층 형태로 구성하는 것이다.

셋째, 워크그룹 서버는 기존의 2 계층 구조로 운영한다. 개발 환경이 간단하고 구현이 용이할 뿐만 아니라 이미 기술 축적도 많이 되어 있어 추가적인 학습 기간이 필요 없다.

이런 세 가지 형태의 애플리케이션 아키텍처를 구성한 이유는 업무 유형에 따라 안정성, 효율성, 개발 용이성 등의 우선순위를 고려하여 구성하였고, 전체적인 시스템 아키텍처에 대한 구축 방법과 애플리케이션 아키텍처 구성 방법에 대해서 알아 보았다.

결론적으로 3 계층 클라이언트/서버 모델은 클라이언트와 데이타베이스 서버 사이에 애플리케이션 서버를 둠으로서 좀 더 유연하고 확장 가능한 시스템을 구현 할 수 있게 해준다. 단순히 애플리케이션 로직의 분리로 3 계층 애플리케이션을 구성하는 것이 아니라 객체지향 및 컴포넌트 기반의 개발방법을 최대한 3 계층 애플리케이션 개발에 적용하고자 한다. 그래서 모든 것을 개발자가 개발하는 형태가 아니라 원하는 기능이 담겨져 있는 객체를 구입해 조립하는 형태의 개발이 만연하고 있다는 사실을 인식할 필요가 있다.

6. 결론 및 향후 계획

본 논문은 2 계층 구조의 클라이언트/서버 시스템에 대한 문제점을 해결하기 위한 방안 수립을 위해 관련 연구부문에서 2 계층구조와 3 계층 구조의 클라이언트/서버 시스템에 대한 중요한 특징과 장단점을 비교 분석하고 주요적용업무에 대해서 알아 보았다. 그리고 3 계층 클라이언트/서버 시스템 구축에 있어서 필수 기술인 미들웨어에 동작원리와 각 기능별 종류에 대해서 알아 보았으며, 객체 지향 미들웨어의 가장 대표적인 코바에 대해서 알아 보았다. 또한 현재 기업에서 3 계층 구조의 클라이언트/서버 시스템 구현 사례를 살펴 봄으로서 본 연구의 주제인 3계층 구조로의 이행 방안 수립에 참고로 활용 했다.

실제 구축하여 운영중인 포스코 클라이언트/서버 시스템에 대해서 살펴 봄으로서 현상 문제점을 보다 구체적으로 알 수 있었으며 이러한 분석 결과 현재 클라이언트/서버 시스템의 대부분이 2 계층 구조의 아키텍처로 구성되어 있으며 점차 확대되어 가는 과정에서 많은 문제점들이 도출되었다. 먼저, 사용자수와 애플리케이션이 증가하면서 성능이 급격하게 저하되는 것이다. 둘째, 시스템이 불안정하다는 것이다. 서버 시스템과 데이타베이스 시스템의 장애가 자주 발생하고, 장애복구에도 많은 시간이 소요되고 있다. 셋째, 시스템의 활용성 저하다. 대부분의 시스템이 부서단위의 단독 시스템로 구성되어 있어 타 부서 시스템과의 연계 활용이 어렵다. 그 외에도 구축 경험 부족에서 오는 개발생산성 저하, 운영비용의 증가 등의 여러 가지 문제점이 발생했다.

이에 대한 해결 방안으로서 3 계층 구조의 클라이언트/서버 시스템으로 이행 방안에 대한 가이드라인을 제시하고, 애플리케이션 아키텍처 구성방안, 3 계층 아키텍처 구성시 고려 사항 등을 제시 하였다. 가능한 기존의 시스템을 활용하면서 3 계층으로 이행방안으로서 다음을 우선적으로 고려 하였다. 첫째, 성능 저하에 대한 방안으로는 애플리케이션 서비스를 여러대의 서버에 배치하여 부하를 분산하고자 했고, 네트웍 트래픽을 줄이기 위해 지역별 서버를 별도로 구축하였다. 둘째, 시스템 안정성을 위해서는 기존의 메인 프레임에 개방형 시스템을 도입하

여 엔터프라이즈 서버를 구축하여 중요 업무의 메인 데이타베이스로 활용하였다. 또한 중앙에서 집중 관리 감독이 가능한 시스템을 도입 적용하였다. 마지막으로, 전사 시스템이 개방형으로 가면서 데이타 공유 범위가 넓어지고 사용자는 데이타 위치에 무관하게 접근 가능하게 함으로서 활용성을 증대하였다. 이러한 이행 방안을 포스코 시스템에 적용하여 3 계층 구조의 클라이언트/서버 시스템 모델을 제시 하였다.

향후에도 발전해 가는 정보기술에 능동적으로 대처하고, 기술 경쟁력 우위를 위해서는 차세대 기술이라 할 수 있는 분산객체를 활용한 코드 재사용 기술과 미들웨어의 적용을 통한 3 계층 방식의 애플리케이션 개발로 개발 생산성의 향상 및 운영 기술력을 확보하여야 한다. 뿐만 아니라 기술과 시스템 상호간에 호환성을 염두해 두고 개발해야 하며 Web 과 CORBA 를 이용한 3 계층 시스템과의 효율적인 연동 방안에 대해서도 지속적인 연구 노력이 필요하다.

참고문헌

[1] J. Won-Ki Hong, *Distributed System*, Fall, 1996.

[2] Robert Drfail, Dan Harkey, Jeri Edwards, *The Essential Client/Server Survival Guide*, WILEY, 1996.

[3] Dawna Travis Dewire, *Client/Server Computing*, 1996.

[4] David Flanagaan, *JAVA in a Nutshell*, 1997.

[5] "What is Client/Server?", http://www.wenet.net/~jtmalone/what_cs.html.

[6] John Gallaugher, "The Critical Choice of Client Server Architecture", 1996, http://web.syr.edu/~jmgallau/cccsa.html.

[7] Neil Jenkins, Tim Evans, Ellen Gottsdiener, Lee Huang, Paul Hipsley, Vinay Nadig, Micbael Ricbards, Cbares Wood, *Client/Server UNLEASED*, SAMS, 1996.

[8] Microsoft, "Understanding Middleware", http://203.30.174.83/home/jirubak_paper/middleware/middleware.html.

[9] David E.Y. Sarna and George J. Febish, "Implementing Client/Server Solutions", http://www.objectsoftcorp.com/.

[10] POSDATA , " 2-tier and 3-tier Client/Server Computing", 1996.

[11] ORG, "DCE Overview", http://www.osf.org/comm/lit/TOG-DCE-PD-1296.htm

[12] Jon Siegel, *CORBA fundamentals and Programming*, 1996.

[13] Microsoft, "COM Overview", http://www.microsoft.com/oledev/olecom/Com_modl.htm

[14] H. Onozawa, *Distributed object Oriented Technology CORBA*, 1996.

[15] OSF , "Often Software Foundation homepage", http://www.opengroup.org/index.htm.

[16] Microsoft, "The kind of middleware", http://www.infoage.co.kr/maso/9612/spec.html.

[17] "Moving PC database applications to Client/Server", http://www.dbpro.co.kr/advantage/.

[18] IDAPI/BDE Year 2000, http://www.prestwood.com/discuss_paradox/_misc/0000001d.htm.

[19] CLIENT DEPOSITOR SERVICES, http://www.cryolab.com/client.htm.

[20] TRANSARC, homepage, http://www.transarc.com/afs/transarc.com/public/www/.

[21] Oracle, homepage, http://www.oracle.com/.

[22] Novell , Homepage, http://www.novell.com/.

[23] Peter J. Houston, "Introduction to DCE and Encina",

http://www.transarc.com:80/afs/transarc.com/public/www/Public/ProdServ/Product/Whitepapers/.

[24] Encina and DCE for CICS, http://www.hursley.ibm.com/transaction/bookshelf/manuals/aix/erzgag/erzgag21.htm.

[25] Robert Drfail, Dan Harkey, Jeri Edwards, *The Essential Distributed objects Survival Guide*, 1996.

[26] Peter J. Houston, "Encina Monitor Architectural Case Studies", Jan. 1995.

[27] Transarc , "Building Large Scale Client-Server Systems Using the Encina Monitor", http://www.transarc.com:80/afs/transarc.com/public/www/Public/ProdServ/Product/Whitepapers/EncTux/enctuxedo.html.

[28] David E. Y. Sarna and George J. Febish , "Developing Enterprise-Wide Applications with Visual Basic 4.0", http://www.objectsoftcorp.com/ .

[29] John Bloomer, "Power programming with RPC", 1992.

[30] Andrew T.F, *Object Analysys & design*, 1995.

[31] "The Evolution of Client/Server Architectures", http://www.hansen.com/about/cs.htm.

[32] John Gallaugher Syracuse University , "The Critical Choice of Client Server Architecture", http://web.syr.edu/~jmgallau/cccsa.html.

[33] "Why is Client/Server", http://www.pts.co.uk/pts/whtpaper/cli_serv.frame.html.

[34] Stevens, *UNIX Network programming*, 1991.

[35] This page defines 'NetBIOS' (Network Basic Input/Output System), http://whatis.com/netbios.htm.

[36] Named Pipes, http://www.ece.sc.edu/class/eece890n/shared/projects/dan/namedpipes.htm.

[37] OMA Overview, http://www.omg.org/about/omaov.htm.

[38] Open Computing, Jan, 1997.