□ 研 究 報 告 □

韓國軍事運營分析學會誌
第 22 卷, 第 1 號, 1996. 6 . 30

클라이언트/서버 구조에 기반한 시스템 구축과 시스템 구축시 고려사항

(Client/server system construction and practical guidelines for the configuration of the client/server system)

김 유 정*

Abstract

In this paper, a client/server architecture and design concerns deduced from a real system design are described.

The MIS system carried on IDIS is the one to be described. The experience from the system design is listed up and analized also. It is common to see that a distributed computing environment has much more complex structure than a conventional centralized one. As a consequence, a distributed system design is a complex task. It is worth to note that there should be sufficient considerations which had not been thought ever for a centralized system.

The objectives of this paper is to list up these problems and the solutions from experiences on a real system construction.

* 국방정보체계연구소

1. 서론

급변하는 정보환경 속에서 기업들은 새로운 시장과 새로운 경쟁, 새로운 기술에 적응해야만 살아남을 수 있게 되었다. 이러한 당면과제들에 시급히 요구되는 것 중의 하나가 정보시스템을 환경변화에 신속히 대응할 수 있도록 변화시키는 것이며 기업들은 이런 요구에 대응하기 위해 다운사이징 전략을 추진하였고 이러한 다운사이징을 추진하기 위한 구현 아키텍처로서 클라이언트/서버 컴퓨팅이 사용되고 있다. 클라이언트/서버 컴퓨팅은 분산환경으로 발전되어 가고 있는 컴퓨팅 환경의 하나로 처리를 클라이언트와 서버가 나누어 처리하는 형태를 말한다.

분산컴퓨팅시스템(Distributed Computing System)은 특정 기능을 수행하기 위해 컴퓨터 네트워크를 통해 연결된 독립적인 컴퓨터들의 집합으로 컴퓨터간의 정보교환은 단지 네트워크 상에서 메세지를 통해서만 이루어진다. 일반적으로 분산 시스템이라고도 하며 이러한 분산 컴퓨팅을 하기위한 방법으로 클라이언트/서버와 동배간 처리 방식이 있는데 현재 주로 사용되고있는 처리 방식은 클라이언트/서버 컴퓨팅을 사용한다.

이러한 클라이언트/서버 컴퓨팅 기술을 사용해 구현된 시스템을 클라이언트/서버 시스템이라 하며 본 논문에서는 이 클라이언트/서버 시스템을 구현한 사례와 시스템을 구축한 경험으로 얻어진 문제점 및 고려사항들을 중심으로 기술하고자 한다.

우선 앞으로 사용될 용어 중 그 단어의 의미를 미리 정의해 두고자 한다. 일반적으로 클라이언트/서버 모델과 클라이언트/서버 컴퓨팅, 클라이언트/서버 시스템, 클라이언트/서버 어플리케이션이란 단어를 많이 사용한다. 이 세가지 용어를 혼용하여 사용되는 경우가 많으나 그 의미를 구분해 보면, 우선 클라이언트/서버 구조는 시스템 구성요소를 분산하는 방법으로 각각의 구성요소를 어디에 배치할 것인지를 결정한다. 클라이언트/서버 모델은 서비스 요청자 즉, 클라이언트와 서비스 제공자 즉, 서버 간의 통신을 서술하기 위한 개념이다. 분산컴퓨팅에서 사용되는 모델의 종류는 클라이언트/서버 모델 그리고 동배간(peer_to_peer) 모델 등이 있다. 본 논문에서 다루고자 하는 시스템은 클라이언트/서버 모델에 근거한 클라이언트/서버 시스템이다. 클라이언트/서버 시스템은 클라이언트/서버 모델에 근거한 하드웨어 및 소프트웨어 시스템을 말한다. 클라이언트/서버 어플리케이션은 클라이언트/서버 모델을 사용한 어플리케이션을 말한다.

클라이언트/서버 시스템을 구축하기 위해서 가장 중요한 과제는 구조를 어떻게 설계하는가이다. 클라이언트/서버 구조는 시스템의 구성요소들을 어디에 배치하는가에 따라 그 구조가 결정된다. 이러한 구조를 결정하는 것은 업무구조나 조직에 따라 다르게 구성된다.

2. 클라이언트/서버 시스템

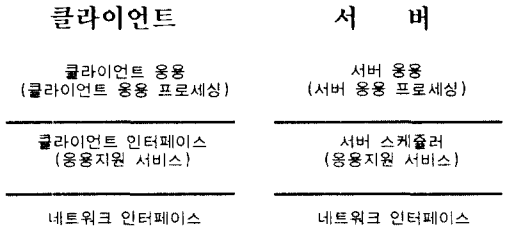
구성요소가 클라이언트에 위치하는지, 서버에 위치하는지에 따라 클라이언트/서버 시스템의 구조는 크게 5가지 종류로 구분한다. 클라이언트/서버 구조는 시스템을 설계할 때 매우 중요한 요소

이며 그 조직의 특성과 업무의 특성에 따라 결정될 수 있다.

2.1 클라이언트와 서버

클라이언트와 서버는 서로 독립적인 프로세스로서 같은 컴퓨터 또는 다른 컴퓨터에서 실행될 수 있다. 또한 일반적으로 서버는 지정된 컴퓨터를 사용하는 것 처럼 특정 기능의 프로세스는 지정된 컴퓨터에서 실행될 수 있다. 클라이언트와 서버로 나누어 처리하므로서 서로 내부적인 정보를 숨길 수 있다. 이는 상대방이 어떻게 처리하는지에 대한 지식 없이도 사용할 수 있는 것을 의미한다. 또한 신뢰성있는 시스템을 구축하기 위해 데이터를 중복시키기도 하며 서버의 고장에 대비하기 위해 서버간의 동기화를 필요로 한다. 이는 서버를 둘 이상 사용할 경우이며 비교적 적은비용의 컴퓨터를 여러개 사용함으로써 얻어질 수 있는 장점이기도 하다.

2.2. 클라이언트/서버 구성요소



<그림 1> 클라이언트/서버의 소프트웨어 구성요소

'클라이언트/서버를 구성하는 요소는 크게 하드웨어와 소프트웨어로 나눌 수 있다. 그 중 소프트웨어 측면에서 살펴보면, 클라이언트측에서는 클라이언트 프로세스와 클라이언트 인터페이스 그리고 네트워크 인터페이스를들 수 있다. 서버측에서는 서버응용과 서버 스케쥴러 그리고 네트워크 인터페이스를 들 수 있다.

클라이언트 프로세스는 클라이언트측에서 응용 기능을 실행하는 프로세스이다. 이 프로세스의 특징은 사용자 인터페이스를 통해 사용자와 대화를 하며, 응용 프로그램을 실행하고 Query나 명령어를 구성하여 클라이언트 인터페이스와 대화한다. 또한 클라이언트 인터페이스로 부터 요청에 대한 응답을 받아 필요시 이를 출력하는 기능을 가진다.

서버 프로세스는 서버측에서 응용기능을 수행하는 프로세스이다. 이 프로세스는 클라이언트에게 서버스를 제공하며, 클라이언트가 세부사항을 알지 못하게 함으로서 내부정보를 숨긴다. 또한 이 프로세스는 서버 스케쥴러에 의해 실행되고 그 결과를 스케쥴러에게 돌려준다.

클라이언트 인터페이스는 API(Application Program Interface)를 통해 클라이언트 프로세스에 의해 실행되는 소프트웨어 모듈의 집합으로 클라이언트 어플리케이션과 서버들 간의 인터페이스를 제공한다. 그리고 서버에 대한 연결을 설정하고 서버로 요구를 보내며 서버로부터 요구에 대한 응답을 받아 이를 번역하여 클라이언트 프로세스로 넘겨 준다.

서버 스케쥴러는 클라이언트의 요구를 감시하고 적절한 서버 프로세스를 실행한다. 이는 네트워크로 인터페이스로 부터 클라이언트의 요구를

받아 이를 번역하고 보안성을 점검하며 서버 프로세스를 관리한다. 그리고 서버 프로세스로부터 응답을 받아 이를 클라이언트로 전송한다.

네트워크 인터페이스는 네트워크상에서의 주소지정과 전송기법을 제공한다. 대표적인 네트워크 인터페이스로는 TCP/IP, SNA, NetBios, SPX/IPX등이 있다.

2.3 클라이언트/서버 모델

클라이언트/서버 모델은 처리를 요청하는 클라이언트와 클라이언트의 요구에 대한 서비스를 제공하는 서버 형태로 분산한다. 이들은 클라이언트의 요청과 서버의 반응간의 상호 작용이다. 이러한 상호 작용을 하기위한 일반적인 응용 구성 요소들을 몇가지로 구분할 수 있는데 이는 표현논리와 업무논리 그리고 데이터베이스 논리, 데이터베이스 처리로 구분한다. 이러한 각각의 기능을 어디에 배치하는가에 따라 클라이언트/서버 시스템의 구조(Architecture)가 결정된다.

표현논리(Presentation Logic)는 최종사용자와 접속하고 동시에 업무논리와 상호작용한다. 이는 화면형식(Formatting)과 화면의 읽기과 쓰기, 윈도우 관리, 키보드와 마우스 다루기와 같은 일을 수행한다. 이러한 일들은 최근 워크스테이션 기술의 발전으로 표현논리 개발자들이 GUI를 이용하여 응용을 좀더 쉽고, 다양하게 개발할 수 있도록 한다. GUI는 최종사용자의 생산성을 높일 수 있을 뿐아니라 사용하기가 쉬우므로 훈련기간이 짧아지고, 그래서 응용개발과 훈련에 대한 투자 가치가 더 높아진다. 표현논리를 제공하는 설비로는 IBM사의 CICS, Microsoft사의 MS-Windows, X 윈도우, OSF의 Motif, SUN사의 Open Look등이 있다. 본 논문에서 소개할 시스템에서는 표현논리로 Microsoft사의 MS-Windows 3.1을 사용하였다.

업무논리(Business Logic)는 사용자나 데이터베이스의 I/O와 직접 관련되지 않은 응용코드의 일부로서 요구사항, 규정 그리고 수행하도록 계획된 특별한 업무작업(Business Task)등에 따라서 화면이나 데이터베이스로부터의 입력 데이터를 사용한다. 일반적으로 이 코드는 3세대언어 또는 4세대 언어로 작성된다. 본 논문에서는 업무논리 코드를 4세대 언어를 사용하여 작성하였다.

데이터베이스 논리(Database Logic)는 응용안에서 데이터베이스를 조작하는 응용코드의 일부분 이다. 데이터는 DBMS가 관리하며 이에 대한 데이터 조작은 SQL을 사용한다. 본 논문에서 소개되는 시스템은 4세대 언어로 작성한 코드에 SQL이 삽입되었다.

데이터베이스 처리(Database Processing)는 DBMS가 수행하는 부분으로 DML로 표현된 요청에 직접관련된 데이터의 실제적인 처리 즉, 물리적 I/O, LOG, LOCK 관리 등이 DBMS에 의해 수행된다.

이와 같이 응용요소들을 구분할 수 있으나 실제 구현상황에서 이러한 범주로 구분하는 것은 항상 간단하지만은 않으며 구성 요소들간의 경계도 명확하게 정의되지 않을 수 있다.

호스트 기반 환경에서 이러한 응용요소들은 같은 시스템상에 존재하며 일반적으로 하나의 실행 프로그램으로 연결되나 분산환경에서는 이러

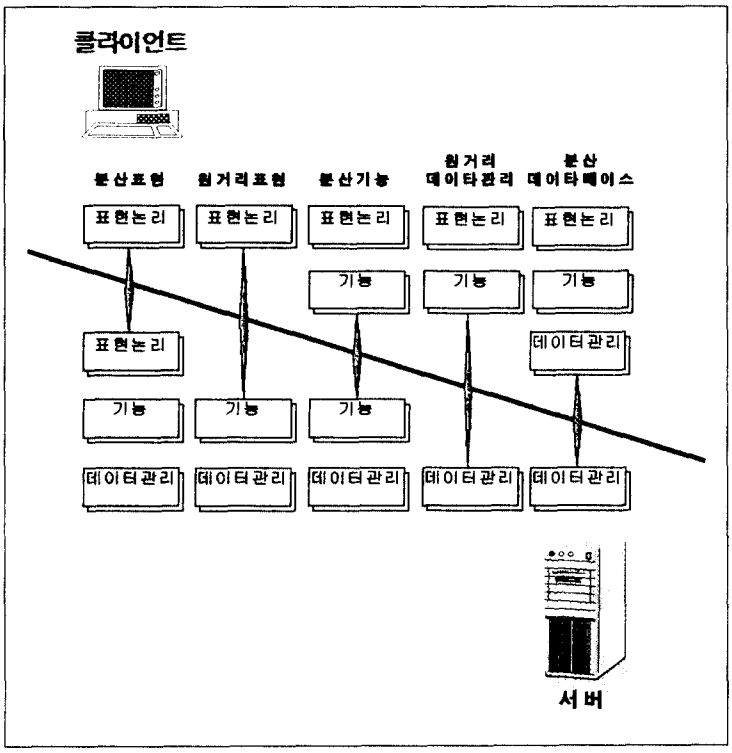
한 구성요소들이 여러 시스템에 분산되어 네트워크로 서로 연결된다.

3. 클라이언트/서버 구조

분산 컴퓨팅은 컴퓨터 네트워크를 통하여 자원을 사용하는 응용들을 개발하고 수행하는 것을 더 쉽게 만든다. 응용들은 그 일에 가장 적합한 컴퓨터에서 수행되도록 분산될 수 있으며 각 작업들은 성능을 높이고 자원을 보다 효과적으로 사용하기 위해 병렬로 수행될 수 있다.

클라이언트/서버 구조는 분산 컴퓨팅으로 구축되며 분산컴퓨팅의 한 특별한 형태이다. 클라이언트/서버의 주요 요구사항은 네트워크로 상호 접속된 매우 다양한 플랫폼에서도 자유롭게 데이터를 저장, 조작하고 응용들을 수행하는 것이다. 클라이언트/서버 구현은 개방되고, 유연하며, 표준화된 다양한 업체의 컴퓨팅 환경에서 가능해야 한다.

이러한 클라이언트/서버 구조는 <그림 2>에서와 같이 그 기능의 위치에 따라 분류하는데 그 종류로는 분산표현과 원거리 표현, 분산 기능, 원거리 데이터관리, 분산데이타베이스등으로 나눌 수 있다.



<그림 2> 클라이언트/서버 구조

3.1 분산표현

분산표현은 클라이언트가 화면표현만을 담당

하고 그 외의 모든 기능은 서버가 담당하는 구조이다. 현재 사용되고 있는 대부분의 그래픽 사용자 인터페이스는 많은 계산력을 필요로 하기 때문에 그러한 요구사항을 취급하도록 특별히 설계된 플랫폼으로 표현논리를 내려 보내는 것이 경제적으로 유리하다. 이러한 요구사항을 만족하는 플랫폼은 사용자 측에 위치하며 이는 인간의 상호작용에 맞도록 설계된다. 또한 분산된 시스템이 사용자에게 분산된 것을 인식하지 못하도록 단일 시스템의 이미지처럼 보이도록 하기 위해서는 응용논리의 표현 부분과 그 나머지 부분 사이의 상호작용은 사용자에게 투명한 방식으로 수행되어야 한다.

이러한 표현기능의 분산구조로 가장 잘 알려진 기술은 X-Windows이다. 이 X-Windows는 클라이언트/서버 모델에 기초를 둔 것이다.

3.2 원격표현

원격표현에서는 클라이언트는 표현 기능을 담당하고 그 외의 기능은 서버가 담당하는 구조이다. 이 경우 클라이언트 측에서 처리하는 기능이 분산표현방식에 비해 많기 때문에 비교적 성능이 좋은 워크스테이션을 필요로 한다. 이러한 처리형태는 RPC(Remote Procedure Call)나 기타 프로세스간 통신(APPC등)을 사용하여 구현될 수 있다.

3.3 분산기능

분산기능에서는 클라이언트와 서버가 모두 업무논리 기능을 처리할 수 있도록 각각 컴파일된 모듈을 소유하고 메세지를 사용하여 서로 대화한다. 이러한 처리형태는 분산처리의 가장 대표적인 처리형태이다. 업무논리 기능은 일반적으로 클라이언트 측의 업무논리 기능과 서버측의 업무논리 기능으로 분리된다. 이들 업무논리기능 간의 협동은 클라이언트에 위치한 업무논리 기능이 서버로 요청을 보내는 것으로 상호작용을 개시한다. 서버 측은 이 요청에 대해 서버 구성요소들이 반응함으로서 이루어 진다.

설계시 표현과 관련된 업무논리 부분은 클라이언트에 배치하고 데이터베이스와 관련된 업무논리 기능은 이상적으로 데이터베이스 관리 시스템이 있는 데이터베이스 서버에 배치한다. 결과적으로 분할된 응용부분들이 협동 트랜잭션으로 있는 동안에 교환해야하는 메세지 수가 크게 줄어들고, 그러므로서 응답시간이 개선되며 이용가능한 계산자원을 훨씬 더 잘 활용하게 된다.

분산업무논리 기능은 설계와 개발이 가장 어려운 협동 처리 응용이다. 가장 간단한 형태 조차 분산업무논리 유형은 별도로 컴파일된 두 개 이상의 프로그램으로 구성되며, 간단함에도 불구하고 협동방식으로 상호간에 동작하도록 설계되어야 한다. 이러한 협동 프로그램과 그것이 존재하는 노드 수가 많아 질수록 응용의 설계와 관리의 복잡성 역시 증가한다. 그러므로 자원의 활용이나 효율성 면에서는 이 구조가 매우 이상적인 구조이나 설계의 복잡함이나 구현의 어려움이 동반한다는 문제점을 내포하고 있다.

결론적으로, 분산 업무논리 기능은 특히 복잡하고 대단히 상호작용적이며, 데이터베이스 I/O집약적인 클라이언트/서버 응용에 매우 적합하다.

3.4 원격 데이터 관리

원격 데이터 관리에서 서버는 데이터관리 기능만을 담당하고 클라이언트는 데이터베이스 질의어를 사용하여 서버에게 자료를 요청하는 형태이다.

일반적인 데이터 관리 논리는 두가지 구성요소를 가진다. 하나는 데이터 처리 논리이다. 이는 응용처리의 일부로서 응용내에서 데이터를 조작하는 부분이다. 다른 하나는 데이터베이스 처리로서 데이터베이스 처리 논리로 기술된 요청에 대해 직접적으로 데이터에 관련된 처리를 수행한다. 데이터베이스 처리 기능은 데이터베이스 그 자체와 동일한 시스템 상에 존재해야 하는 반면에 데이터 처리 논리는 DBMS나 응용업무 논리에 가깝게 위치되기도 한다.

관계데이타베이스 시스템(RDBMS)의 경우 원격 데이터 관리는 SQL을 사용하여 클라이언트/서버 상호작용을 구현한다. 클라이언트는 처리를 위해 SQL구문을 포맷한 후 서버에 전송하는 API를 통해 서버와 통신한다.

원격 데이터 관리 방식은 비교적 구현하기 쉬우며 현재 제품으로 나와있는 대부분의 제품들이 이 방식을 제공한다. 이 방식은 구현하기 쉽다는 장점이 있는 반면 단점도 가지고 있다. 첫째, 단일 시스템에 데이터베이스를 배치함으로서 병목현상이 생길 수 있다. 둘째, 모든 데이터의 요청과 응답을 서버와 응용간의 네트워크를 통해서 전송하며, 잠재적으로 상당한 통신 부담이 생긴다. 셋째, 데이터베이스 서버는 그 기능상 클라이언트로부터 보내온 데이타 조작어에 대한 처리에 국한된다. 넷째, 데이터베이스 서버 장애시 파급효과가 크다. 다섯째, 클라이언트/서버 구조는 상당한 계산자원이 유용함에도 불구하고 데이터베이스 서버의 특이성으로 모든 이용 가능한 자원의 장점을 취하지 못한다.

그러나 이러한 단점에도 불구하고 구현의 용이성등의 이유로 많은 시스템들이 이 방식을 사용해 구현되고 있으며, 대부분의 개발도구들이 이 방식을 지원하고 있다.

본 논문에서 소개하고 있는 시스템 역시 이 원격 데이터 관리방식으로 구현되었다.

3.5 분산 데이터 관리

분산 데이터 관리는 데이터 관리를 클라이언트와 서버가 분담하는 형태이다. 데이터가 다수의 노드에 분산되어 있어 그 데이터를 직접 사용하는 위치에 가까이 배치할 수 있으며, 다른 위치에 중요한 데이터의 복사본을 여러개 배치하므로써 데이터에 대한 높은 유용성을 제공하고, 장애에 대해 효과적인 대처를 할 수 있다.

데이터가 여러개 노드에 분산될 때는 데이터 관리 논리 또한 데이터와 함께 분산되어야 한다. 그러므로 데이터베이스 처리 논리의 일부 분은 데이터베이스가 존재하는 동일한 시스템에 존재해야 한다.

분산 DBMS는 원격 프로시져 호출(RPC), 프로그램간 통신, 분산 트랜잭션 처리와 같은 협동처리 기술과 Stored Procedure, Trigger등과 같은 기법을 이용한다. 분산 DBMS를 적절하게 사용하면 최종 사용자에게 데이터 위치 투명성, 데이터

무결성, 일관성과 신뢰성등을 제공할 수 있다. 그러나 분산 DBMS는 이와 같은 장점과 더불어 분산과 관련된 많은 문제점도 내포하고 있다. 즉, 데이터가 분산되는 방법, 데이터의 동기화 문제, 일관성, 로킹(Locking), 무결성, 투명성 등 많은 문제를 내포하고 있다.

4. 개발도구

4.1 4세대 언어

클라이언트/서버 구조를 지원하는 개발도구는 일반적으로 앞에서 언급한 원격 데이터 관리 구조를 지원한다. 즉, 클라이언트 측에서 실행될 어플리케이션을 개발할 수 있는 화면 설계 및 프로그래밍 도구를 지원하고 서버에 대해서는 서버에 위치한 데이터를 접근 할 수 있도록 하는 데이터베이스 서버와의 인터페이스를 제공한다. 일반적으로 데이터베이스와의 접속은 SQL을 사용하여 수행되며, 클라이언트 측의 어플리케이션에 SQL을 포함 할 수 있도록 지원하고 있다.

최근에 사용되고 있는 개발도구를 프로그래밍 언어 측면에서 4세대 언어라고 일컫는다. 이는 기존의 3세대 언어에서 한단계 발전된 형태의 새로운 언어로 구분하는데, 이는 기존 3세대 언어의 문제점을 보완한 새로운 언어임을 말해준다. 기존의 3세대 언어는 프로그램의 전문가들이 사용하기 위해 개발된 것이어서 배우기가 어렵고 프로그래밍 하기도 어려웠다. 또한 문법체계가 인간의 언어구조와 매우 다르기 때문에 익히기가 어렵고, 복잡한 구조를 가져 오류를 발견하기가 매우 어려웠다. 이러한 문제점들은 프로그램 개발 요구가 증가하면서 소프트웨어 생산성의 문제는 더욱 두드러 졌고, 사용자가 증가하고 사용자의 요구가 다양해 지면서 보다 나은 해결책을 필요로 하게 되었다. 이러한 이유로 기존의 3세대 언어와는 구조가 다른 인간의 인지구조와 유사한 형태의 4세대 언어가 탄생하게 되었다.

4세대언어는 비 절차적인 언어로서 사용자 중심의 개발방식을 사용하며, 효과적인 사용자 인터페이스를 제공한다. 또한 단순한 구조를 가짐으로서 생산성이 향상되고 양질의 소프트웨어를 개발할 수 있게 해준다. 또한 데이터베이스와 밀접하게 설계되어 데이터베이스의 사용이 용이하다.

그러나 이러한 장점들과 더불어 몇가지의 단점도 나타나고 있다. 우선 특정분야의 개발이 용이하게 하기 위해 설계됨으로서 다양한 요구충족이 어렵다. 3세대 언어는 거의 모든 분야를 충족할 수 있는데 반해 대부분의 4세대 언어는 특정 업무 분야에 적합하게 설계되어 다양한 분야의 개발에는 부족한 면이 있다. 또한 기존의 3세대 언어에 비해 많은 처리능력을 필요로 한다. 4세대 언어로 발전하면서 기존에 사용하던 문자방식의 사용자 인터페이스를 사용자에게 친숙한 그래픽 형태의 사용자 인터페이스로 전환하면서 그래픽 처리를 위해 추가의 처리능력이 필요하게 되었으며, 이와 더불어 자동 프로그래밍을 지원하기 위한 처리 능력이 부가되었다. 또한 클라이언트/서버 구조로 전환하면서 서버와의 통신에 대한 부담도 늘어 기존의 3세대언어로 개발된 어플리케이션 보다 많은 처리능력을 필요로 한다. 즉 성능이 좋은 컴퓨터를 사용해야 한다는 것이다. 그러나 이러한 단점은 컴퓨터 하드웨어의 성능이 급

진적으로 향상되고 있고 가격은 상대적으로 급격히 하락하고 있으므로 사용자는 시간이 지날수록 높은 성능의 컴퓨터를 비교적 저렴한 가격으로 구입할 수 있기 때문에 큰 문제가 되지는 않을 것으로 보인다.

본 논문에서도 시스템의 개발을 위해 4세대 언어를 사용하였다.

4.2 객체지향 프로그래밍

4세대 언어는 개발의 편리함과 함께 최근 관심을 끌고 있는 객체지향 프로그래밍을 지원한다. 객체지향 프로그래밍이란 프로그램을 객체들의 조합으로 구현하는 방법을 말한다. 객체지향 프로그래밍을 하는 이유는 첫째, 실세계에 대한 자연스러운 모델링을 할 수 있도록 한다. 기존의 프로그래밍 언어는 절차적 언어로서 모든 작업은 코딩 순서에 따라 실행되었으나 이는 실세계와는 매우 다른 처리 방법이다. 객체지향 프로그래밍 언어는 실세계에 존재하는 객체 그 자체를 모델링 할 수 있게 함으로써 자연스러운 모델링이 가능하게 한다. 둘째, 모듈성이 높아진다. 모든 대상을 객체로 캡슐화 하여 모듈성을 향상시킨다. 셋째, 재사용성이 높아진다. 프로그래밍 언어에서 클래스라는 형태를 지원하여 이 클래스를 상속할 수 있게 함으로써 한번 정의된 클래스를 상속하여 계속 사용될 수 있도록 함으로써 소프트웨어의 재 사용성을 증가시킨다. 넷째, 병렬성을 증가시킨다. 각각의 독립적인 모듈은 독립적으로 병렬 실행이 가능할 수 있기 때문에 컴퓨터의 능력을 최대한 효과적으로 활용할 수 있도록한다.

이러한 이유로 객체지향 프로그래밍을 사용하며 실제 본 논문에서 구현된 시스템에서도 객체지향 프로그래밍 기법을 적용하여 프로그래밍을 함으로써 소프트웨어의 생산성이 향상되고 오류발생의 가능성이 줄어들어 소프트웨어의 품질이 향상된 것을 볼 수 있었다.

5. 클라이언트/서버 시스템 개발

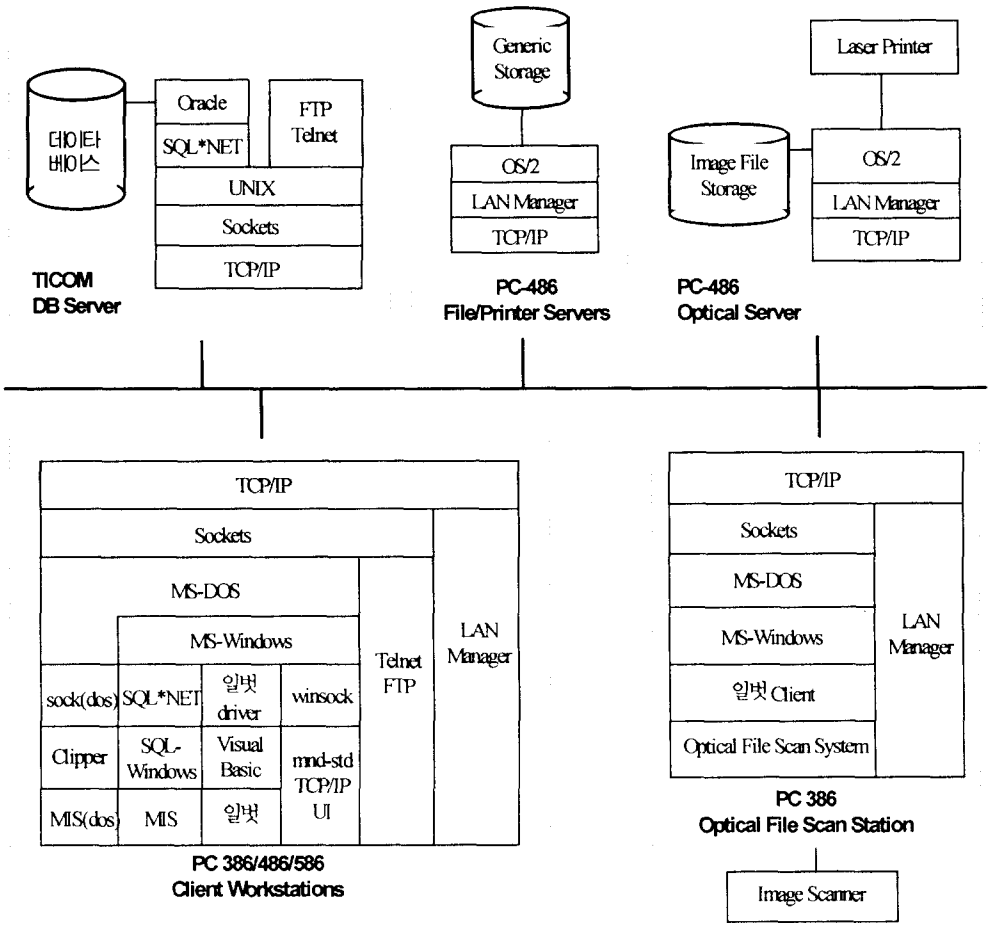
이 시스템은 국방정보체계연구소의 MIS업무를 대상으로 구축된 시스템이다. 국방정보체계연구소에서 개발한 MIS업무는 연구소 내에서 운영되고 있는 업무 중의 일부를 개발한 것이며 지속적으로 개발이 추진되고 있다.

5.1 시스템 구성

시스템 구성은 <그림 3>과 같다. 그림의 내용은 소프트웨어 시스템에 관한 내용이다. 이 시스템에서 사용한 클라이언트/서버 구조는 원거리 데이터관리 방식이다. 클라이언트 측에서는 표현과 처리기능을 수행하고 서버측에서는 데이터베이스 서버의 기능을 수행한다.

클라이언트

▷ 하드웨어 : PC 386/486/586
▷ 운영체계 : MS-DOS, MS-Windows 3.1
▷ 네트워크 운영체계 : MS LAN Manager
▷ 네트워크 프로토콜 : TCP/IP
▷ 데이터베이스 인터페이스 : SQL*NET
▷ 개발도구 : SQLWindows
Clipper,
Visual Basic



<그림 3> 국방정보체계연구소(IDIS) 자동화 시스템 구성도

서버

▷ 하드웨어 : TICOM, PC 386/486

▷ 운영체계 : UNIX, OS/2

▷ 네트워크 운영체계 : MS LAN Manager

▷ 네트워크 프로토콜 : TCP/IP

▷ 데이터베이스 인터페이스 : SQL*NET

▷ DBMS : ORACLE V.7

보안유지

▷ 사용자별 접근권한 제한

▷ DB table, View, Column단위 제한

시스템간 인터페이스

▷ ORACLE의 SQL*NET을 사용하여 데이터베이스 서버와 클라이언트 간 연결

▷ SQLWindows의 ORACLE Router를 사용하여 개발도구인 SQLWindows와 오라클을 연결

5.2 시스템 설계

시스템 설계는 하드웨어 설계와 소프트웨어 설계로 나눌 수 있다. 하드웨어는 그 조직의 요구 수준에 따라 다르게 선정된다. 본 연구소의 하드웨어 환경은 기존에 각 연구원들이 PC를 사용하고 있으며, 몇개의 연구과제는 위크스테이션을 사용하고 있었다. 또한 국방전산환경에 대해 연구하는 기관이므로 국산 주 전산기를 군에 적용하기 전에 연구소에서 미리 설치하여 시험, 운영하고 있었다. 그래서 TICOM을 서버로서의 주 전산기 기능 테스트와 클라이언트/서버 시스템 구축을 위한 서버용으로 사용하게 되었다. 다른 서버로는 연구업무 수행시 자료와 정보의 공유를 위해 PC급의 파일 서버를 사용하였으며, 문자데이타 외에도 이미지를 저장할 수 있도록 광화일 서버를 설치하여 문자 형태로 작성되지 않은 외부 공문등을 저장할 수 있게 하였다. 클라이언트는 각 개인당 PC(386/486/586)가 1대씩 운영되고 있었으므로 이를 그대로 사용하였다.

시스템을 새로이 구축할 때 모든 시스템을 폐기하고 전부 새로운 시스템으로 교체해야 하는 것은 아니다. 그렇게 할 경우 추가의 장비구입에 대한 부담이 크고, 기존 자원에 대한 재사용 측면에서 비 효율적인 운영을 초래하게 된다. 새로운 시스템을 구축하더라도 기존의 시스템을 사용하면서 점진적으로 시스템을 추가하고 교환하는 것이 바람직 하다.

소프트웨어의 설계는 운영환경 조성을 위한 시스템 소프트웨어와 어플리케이션 소프트웨어로 나눌 수 있다. 시스템 소프트웨어는 하드웨어의 운영을 위한 운영체계, 네트워크를 운영하기 위한 네트워크 운영체계, 자료저장을 위한 DBMS(DataBase Management System)와 클라이언트/서버 간의 통신을 위한 통신 소프트웨어, 어플리케이션 개발을 위한 개발도구등에 대한 설계를 의미한다.

운영체계로는 파일서버에는 OS/2를 사용했고 TICOM에는 오픈 개념을 가진 UNIX를 사용하였다. 네트워크 운영체계로는 마이크로소프트사의 LAN Manager를 사용하였고 네트워크 프로토콜은 TCP/IP를 사용하였다. 각 클라이언트는 운영체계로 DOS를 사용하며 그래픽 사용자인터페이스를 위해 MS-Windows를 사용한다.

자료저장을 위한 DBMS로는 ORACLE을 사용하였고, 클라이언트와 ORACLE과의 연결을 위해 SQL*NET을 사용하였다. 개발도구로는 4세대 언어인 GUPTA사의 SQLWindows를 사용하여 개발하였다.

클라이언트/서버 구조를 구현하는 방법으로는 데이터베이스 서버를 사용하는 원거리 데이터관리 방식을 사용하여 TICOM에 ORACLE을 데이터베이스시스템으로 탑재하여 데이터베이스 서버로 사용한다.

어플리케이션 소프트웨어는 4세대 프로그래밍 언어를 사용해 개발한다. 프로그램의 설계는 기존의 3세대 언어로된 문자방식의 프로그램과는 설계개념이 다르다. 기존의 3세대 언어는 순차적인 프로그래밍 방식으로 코딩되는 순서대로 프로그램이 실행되었으나 윈도우즈 프로그래밍은 이벤트(event)에 대해 메세지(message)가 발생하고 이

메세지에 대한 액션을 실행함으로서 프로그램이 실행되는 Event_Driven 방식이다. 그러므로 기존의 순차적인 프로그램의 설계방식에서 탈피하여 새로운 설계개념을 익혀야 한다. 이러한 사고의 전환이 쉽지 않기 때문에 추가적으로 개발자에 대한 교육이 필요하다.

6. 시스템 구축시 고려사항

클라이언트/서버 구조는 기존의 단일 시스템 구조에 비해 그 구조 자체가 매우 복잡하다. 이 구조가 기존의 단일 시스템에 비해 많은 장점이 있기는 하나 단일 시스템에서 발생하지 않았던 문제들이 곳곳에서 발생하게 된다. 이러한 문제들을 미리 방지하기 위해서는 사전에 미리 대비를 하는 것이 최선이다. 이 절에서는 이러한 문제의 발생을 최소화하기 위한 몇가지 고려사항을 기술하였다.

6.1 설계시 고려사항

클라이언트/서버 구조는 그 구조 자체가 복잡하므로 설계 작업 또한 복잡하게 된다. 구성요소들이 분산되어 있고 각각의 구성요소들을 서로 상호연결성을 갖도록 설계해야 하며 시스템 사용시에 모든 구성요소들을 투명하게 접속할 수 있도록 하는 투명성을 제공해야 한다.

투명성은 시스템을 사용하는 대상자에 따라 크게 4가지로 분류한다. 사용자 투명성과 개발자 투명성, 설계자 투명성 그리고 관리자 투명성으로 구분한다. 첫째로 사용자 투명성은 네트워크, 운영체계, 컴퓨터시스템, 데이터베이스, 트랜잭션 관리로 부터의 투명성을 의미한다. 둘째, 개발자 투명성은 어플리케이션 개발자가 네트워크상의 다른 컴퓨터에 대한 세부적인 사항을 알지 못해도 개발할 수 있도록하는 것을 말한다. 셋째로, 설계자 투명성은 어플리케이션 시스템 구성요소의 위치에 관계없이 설계할 수 있도록 하는 것을 말한다. 넷째, 관리자 투명성은 분산환경의 시스템 관리가 중앙집중 시스템의 관리와 같도록 하는 것을 의미한다. 시스템을 설계할 때는 이러한 요소들을 고려하여 최대한의 상호연결성과 상호운용성, 이식성 그리고 투명성이 보장되도록 설계해야 한다.

이와 더불어 어플리케이션을 설계할 때 고려해야할 사항은 다음과 같다.

첫째, 최초의 응용업무로 핵심적인 것을 선택한다. 클라이언트/서버 시스템을 구축할 때 모든 업무를 한번에 구축하기는 매우 어려운 일이다. 더구나 경영자가 투자에 대한 가능성을 확신할 수 있도록 하기 위해서는 우선 핵심적인 업무를 선택하는 것이 바람직하다. 많은 비용을 투자하는 프로젝트를 확신도 없이 막연하게 추진할 수는 없는 일이다. 또한 이를 구축한 경험을 바탕으로 앞으로 추진될 업무에 적용할 수 있는 기회도 될 것이다.

둘째, 대상업무와 자료를 잘 알아야 한다. 즉, 클라이언트/서버 구조는 기존에 경험하지 않았던 부분을 기획해야 한다. 예를 들어 사용자인터페이스나 프리젠테이션 흐름 및 데이터관리를 클라이언트와 서버 사이에 어떻게 분할 할 것인가를 결정해야 한다. 이는 대상 응용업무에 관한 정확한

지식을 바탕으로 선택할 것이 요구된다. 이것은 어플리케이션 로직이 어디 있느냐 하는 것과 관계된다.

셋째, 임무에 적합한 서버를 고른다. 클라이언트/서버 환경의 핵심적인 측면은 서버에 있다. 서버는 물리적 서버컴퓨터, 컴퓨터 위에서 운용되는 DBMS, 서버와 클라이언트 사이에 지원되는 통신 소프트웨어 등을 포함하여 서비스 작업이 수행되는 총체적 환경을 일컫는다.

앞에서 언급한 물리적 서버 컴퓨터, DBMS, 통신 소프트웨어등을 선택하는 것은 결코 쉬운 일이 아니다. 무조건 용량이 크고 성능이 좋은 것만을 선택하는 것이 최선은 아니다. 자신의 환경에 적합한 것을 선택하는 것이 중요하다. 만일 필요이상의 매우 강력한 기능을 가진 것을 선택했을 경우 오히려 필요 없는 기능을 유지하는데 많은 비용을 낭비하는 경우가 발생할 수 있다.

넷째, 클라이언트/서버 구조가 반드시 GUI기능을 필요로 하는 것은 아니다. GUI가 클라이언트/서버와 꼭 병행하는 것은 아니다. 사용자가 그래픽 환경을 반드시 원하는지, 마우스의 사용을 편리하게 느끼는지, 기존의 텍스트 환경에 만족하는지에 따라 GUI를 사용할 것인지를 선택해야 한다. 모든 사용자 인터페이스를 GUI로 설계하는 것이 반드시 좋은 것만은 아니다. 오히려 사용자가 GUI를 불편해 할 수도 있고 실제 키보드를 사용하는 것이 사용자의 친숙도나 처리속도 측면에서 더 낫다고 말하는 사용자도 있다.

다섯째, 트리거는 신중히 사용한다. 데이터베이스 관리 시스템에서 참조 무결성을 실행하는 것은 트리거이다. 이것은 레코드의 삽입, 삭제, 혹은 수정등 이벤트를 기동할 때 자동으로 호출되는 서버에 직접 내장된 코드를 지칭한다.

트리거는 참조 무결성에 한정되는 것이 아니고 서버에 대한 모든 요청에 적용되는 서버 수준에서 구현되는 룰(Rule)이라는 관점에서도 볼 수 있다. 이것은 매우 강력한 툴로 잘못 사용할 경우 곤란을 자초할 수도 있다. 트리거는 하나의 어플리케이션에 한정된 것이 아니기 때문에 전체적인 고려 없이 설계할 경우 자료를 제대로 유지하기 힘들어 질 수도 있는 것이다.

또한 조건이 너무 일반적이거나 너무 많은 것을 트리거로 설계하는 것은 피해야 한다. 적절하지 못하게 설계된 트리거는 어플리케이션을 망가뜨릴 수 있다.

여섯째, 어플리케이션 개발도구의 사용을 충분히 고려하는 것이 좋다. PC 클라이언트 플랫폼은 어플리케이션 개발을 가속화하는 다양한 개발도구의 지원이 특징이다. 이러한 도구를 사용하면 개발속도를 가속화 할 수 있고 차후 유지보수 시에 투입되는 노력을 감소시킬 수 있는 장점이 있다. 그러나 모든 업무에 대해 개발도구를 일률적으로 적용하는 것은 잘못된 적용이다. 그 적용업무에 따라 처리속도나 개발도구의 지원 능력이 적합한지를 충분히 고려해야 한다.

일곱째, 데이터베이스 공간을 너무 크게 만드는 것은 피한다. 데이터베이스의 저장장치는 처음에 너무 크게 잡아놓으면 그 공간은 죽은 공간이 되어 버린다. 다른 곳에서 공간이 필요하더라도 줄이기가 어렵게 된다. 데이터베이스 공간은 차후

에도 충분히 물리적, 논리적으로 확장이 가능하다. 성장을 예견하되 공간규모를 과대 평가하는 것은 금물이다. 필요하면 나중에 늘일 수 있다.

여덟째, 기존 기술요원에 대한 기술개편을 예상한다. 기존의 직원이 습득했던 기술만으로는 클라이언트/서버를 구현하기에 부족하다. 일례로 클라이언트/서버를 구축할 때 사용하는 관계형 DBMS는 SQL을 인터페이스로 사용하나 비 SQL DBMS를 사용했거나 DBMS를 사용하지 않았던 기술요원들은 이에 대한 기술을 가지고 있지 않을 것이다. 즉, 기술요원은 새로운 기술습득에 대한 노력이 필요하고 경영자는 이들 기술요원에 대한 교육투자를 예상하여야 할 것이다.

아홉째, 어플리케이션의 특성을 파악해야 한다. 클라이언트/서버 구조는 일반적으로 어플리케이션을 클라이언트 컴퓨터와 서버 컴퓨터에 분산하게 된다. 어플리케이션의 기능이 주로 화면표현과 많이 관련된 부분은 클라이언트 측에 배치하고 화면처리 보다는 데이터베이스와의 상호작용이 많은 부분은 데이터베이스에 밀접하게 배치하는 것이 효과적이다. 이러한 배치를 잘못할 경우 오히려 성능이 떨어지는 시스템을 구축하기 쉽다.

이상의 고려사항을 모두 확인했다고 해서 문제없는 완벽한 시스템을 구축할 수 있는 것은 아니다. 그러나 문제를 최소화 할 수 있다는 면에서 이와 같은 사항들을 기술한 것이다.

6.2 개발도구 선정시 고려사항

클라이언트/서버 도구를 이용하여 개발된 어플리케이션들은 일반적으로 수명이 길다. 따라서 제품을 선정한 뒤 제공업체와의 관계 역시 길어지게 되며 이와 같은 이유로 업체와의 관계가 과거 어느때보다도 중요시된다.

현재는 우후죽순처럼 클라이언트/서버 제품을 제공하는 회사들이 많이 생겨나고 있지만 고객들이 신뢰할 수 있는 제품과 서비스를 제공하는 업체만이 경쟁속에서 계속 살아남을 수 있을 것이며, 자본금이 적은 신생업체보다는 탄탄한 자금력을 바탕으로 한 규모있는 업체가 생존할 가능성이 크다고 추측된다.

이는 끊임없이 기술변화가 요구되는 클라이언트/서버 시장에서 소규모의 신생업체는 고객들이 만족할 만한 교육지원이나 새로운 제품에 대한 지속적인 연구와 투자를 하는데 역부족이기 때문이다.

따라서 제품을 선정하기에 앞서 제공업체의 매출액과 직원 수 등 그 회사의 재정상태 및 규모를 확인 하는 것이 중요하다. 이는 제공업체가 제품을 구매 전과 구매 후에 충분한 기술지원을 할 수 있는지의 여부를 확인하는데 도움이 된다. 이와함께 제품의 시장점유율과 구현 성공사례 등도 살펴보는 것도 중요하다.

클라이언트/서버 시스템을 성공적으로 구현하기 위해서는 다양한 구성요소를 매끄럽게 병합하는 작업이 요구되므로 통합적인 해결책을 제공할 수 있는 업체의 선정도 중요하다. 또한 충분한 기술전수가 이루어질 수 있기위해서는 업체의 교육방법과 교육 프로그램을 살펴보는 것 역시 중요하다.

제품을 선정할 때의 고려사항으로 일반적으로

클라이언트와 서버 양쪽의 개발을 지원하는 도구를 권장한다. 그러나 일부 제품을 제외한 대부분의 제품들이 클라이언트상의 개발만을 적극적으로 지원하고 있다.

클라이언트/서버 시스템을 구축하면서 하나의 업체에 의존하는 것은 오픈 시스템으로 이전해 가는 현재의 추세에 역행되는 것이며 또한 끊임없이 개발되는 최첨단의 신기술을 활용하는데도 적합하지 못하다. 특히 클라이언트/서버 환경으로 이전하는데는 최대의 목적이 다양한 제품의 특성을 최대한으로 살려 사용하는데 있는 만큼 복수 환경을 지원하는 클라이언트/서버의 도구선택은 필수적이다. 따라서 제품을 선택할 때는 제품의 이전성, 투명성(Transparency), 번역성(Translation), 보안성(Security)등을 살피는 것이 무엇보다 중요하다. 이전성을 살펴볼 때는 업체가 얼마나 많은 종류의 플랫폼을 지원하고 있는지와 앞으로 어떤 개발정책을 구상하고 있는지를 파악하는 것이 중요하다. 이는 제품이 여러종류의 플랫폼에서 지원되면 될수록 그만큼 프로그램들이 수정없이 재활용될 수 있기 때문이다. 따라서 제품을 선정할 때는 앞으로 새로운 하드웨어나 데이타베이스를 구입할 때를 대비, 용이하게 새로운 환경으로 이전할 수 있는지의 여부를 살펴보는 것이 중요하다.

또한 이전성과 동일하게 중요한 것이 제품의 투명성이다. 클라이언트/서버 시스템은 처리와 데이타가 분산되어 있는것이 특징이므로 산재된 프로그램과 데이타를 사용자가 세부사항에 신경쓸 필요없이 자동적으로 도구가 관리하는 것이 중요하다.

4세대언어로 출시되고있는 대부분의 시스템은 편리한 환경과 다양한 도구들을 지원하나 이에 따른 부담이 있다. 즉, 클라이언트 측의 높은 처리능력을 요구한다는 것이다. 편리한 기능을 제공하는데 반해 이러한 기능을 제공하기위한 처리능력의 요구 때문에 사양이 낮은 컴퓨터는 사용하기가 힘들다는 것이다. 현재 사용되고 있는 컴퓨터 시스템이 이미 사용되고 있던 낮은 사양의 컴퓨터를 사용해야 한다면 이 부분을 집중적으로 검토하는 것이 중요하다. 시중에는 이러한 문제점을 개선한 도구도 출시되고 있다.

4세대 언어 중 대부분은 클라이언트 프레임웍내의 시스템을 구축하는 데 중점을 두고 있다. PowerBuilder나 SQLWindows, ObjectView등은 클라이언트 중심의 개발도구로 윈도우즈 환경을 중점적으로 지원한다. 그러나 이들 도구들은 현재 PC에 한정된다는 약점이 있다.

유니페이스나 매직 등은 비 윈도우즈 플랫폼에서도 작동되는 이 기종 개발환경이다. 이 도구들은 대부분 메인프레임이나 PC DOS환경에서 사용하던 것을 윈도우즈로 변환한 것이 많으며, 다양한 환경에서 작동이 가능하도록 지원하다 보니 어느 한 환경에서도 최대한 활용되지 못하는 단점이 있을 수도 있다.

또한 4세대 언어는 대부분 SQL을 지원한다. 지금까지 언급한 대부분의 개발도구는 데이터베이스 시스템과는 무관하게 다양한 데이터베이스 시스템을 사용할 수 있도록 지원하는 것이었다. 그러나 이러한 도구들은 기존 데이터베이스 벤더들이 제공하는 개발도구에 비해 데이터베이스의

기능을 충분히 활용하지 못하는 단점이 있다. 앞에서도 언급한 바 있지만 다양한 환경을 지원하려다 보면 어느 한 환경을 충분히 지원하지 못한다는 단점이 있는 것이다. 오라클이나 인포믹스, 인그레스등 대부분의 데이터베이스 벤더들이 기존에 사용하던 개발도구들을 윈도우즈 환경과 클라이언트/서버 환경에서 동작할 수 있도록 전환하고 있으나 클라이언트 전용으로 윈도우즈부터 시작한 4세대 언어들에 비해 미숙한 것이 사실이다. 이런 데이터베이스 전용 개발도구들은 다양한 환경을 지원하지는 못하지만 다른 4세대 언어에 비해 SQL처리가 강력하고 데이터베이스의 특성을 충분히 활용할 수 있다는 장점이 있다.

6.3 개발도구 사용시 고려사항

첫째, 개발도구를 선택할 때는 자신의 환경을 충분히 파악하고 자신의 환경에 맞는 도구를 선택한다. 앞에서도 기술했지만 매우 다양한 개발도구들이 있고 이를 선택하는 것 또한 매우 어렵다. 이들 도구들은 절대적으로 어느 도구가 좋고 어느 도구는 나쁘다고 단정지어 말하기가 매우 힘들다. 그것은 각자 나름대로 중점적으로 지원하는 부분이 다르기 때문이다.

특히 4세대 언어와 데이터베이스 벤더가 제공하는 개발도구 중 어느 것을 선택 할 것인지도 결정하기 어렵다. GUI사용의 중요성이나 프로그래밍의 편리성, 구현기능의 다양성, 데이터베이스의 활용성, 지원도구의 다양성, 타 언어와의 연계성등 각자의 환경에 적합한 특성을 고려하여 선택하는 것이 중요하다.

둘째로, 프로그래밍 규격을 수립, 이를 준수한다. 새로운 언어를 사용하므로 기존에 3세대 언어에서 사용하던 규격은 이제 4세대언어를 기반으로 새로이 구성되어야 한다. 윈도우즈 프로그래밍의 특성, 그리고 객체지향언어의 특성 등을 충분히 고려한 프로그래밍 규격을 수립하고 이를 준수하도록 하는 것이 중요하다. 4세대 언어를 사용하면서 기존의 3세대 언어 스타일로 프로그래밍을 하는 것은 자칫 불안정한 시스템을 생산할 가능성이 있다.

셋째, 개발자들은 데이터베이스에 대한 숙련이 필요하다. 클라이언트/서버의 특징은 서버에 독립적이라는 것이다. 그러나 개발자들은 여전히 데이터베이스의 특성을 잘 알아야 한다. 일례로 SQL문은 표준이라고는 하나 DBMS를 제공하는 업체마다 조금씩 다른문법을 제공한다. 또한 SQL은 사용하기에 따라 시스템 성능에 상당히 많은 영향을 미치며, 인덱스를 시스템 특성에 따라 개발자가 적절하게 사용하는 것이 시스템의 성능을 높일 수 있는 방법이다. 또한 기존의 파일 시스템과는 달리 데이터베이스 테이블 간의 JOIN이 성능저하의 요인이 되기도 한다. 그러므로 JOIN의 사용을 최소화 하는 것이 좋다.

넷째, 개발도구는 개발자가 좀 더 편리하게 개발할 수 있는 환경을 제공한다. 개발도구는 일반적으로 통합 개발환경을 제공하여 기존에 직접 명령어를 입력하여 실행하던 방식과는 달리 하나의 통합된 메뉴시스템 하에서 간단하게 사용할 수 있다. 출력양식을 생성하는 것도 지원도구를 사용하여 간단히 만들 수 있도록 한다. 그러나 현재 시중에서 사용되고 있는 대부분의 개발도구들이 외국에서 제작된 것을 수입한 것이기 때문에

우리의 사용 양식과 맞지 않는 부분이 있을 수도 있다. 농산물에서 일고있는 신토불이 붐이 결코 농산물에만 국한 된 것은 아닌 것 같다. 이제 우리의 개발경험과 우리 나라 사용자의 요구사항을 충분히 수용할 수 있는 국산 개발도구의 출현이 기대된다.

다섯째, GUI를 사용할 경우 화면 설계에 많은 관심을 두어야 한다. 기존에 텍스트 기반의 인터페이스로 개발하던 개발자들은 사용자 인터페이스를 설계 할 때 기존의 관습 그대로 텍스트 환경의 화면과 별 차이 없이 그래픽 화면을 설계하기가 쉽다. 그러나 윈도우즈상의 그래픽 화면은 그 사용환경 자체가 다르므로 텍스트 화면과는 다른 차원의 설계 감각을 요구한다. 화면을 설계하기 전에 우선 GUI로 설계된 다른 시스템들을 보고 분석한 후 설계에 들어가는 것이 보다 세련된 화면을 만드는데 도움이 될 것이다.

7. 결론

이상과 같이 클라이언트/서버 환경과 시스템 구축 사례 및 시스템 구축시 고려사항을 살펴보았다. 클라이언트/서버 시스템의 구축을 위해서는 앞에서도 언급한 바와 같이 단일 시스템 환경 구축시 보다 훨씬 많은 고려사항이 있는 것은 분명하다. 그렇기 때문에 실패할 확률도 높을 수 있다. 성공적인 클라이언트/서버 시스템 구축을 위해서는 앞에서 언급된 고려사항들에 대한 충분한 검토가 필요하다.

지금까지 언급된 내용 외에도 클라이언트/서버 구조가 가능한 다양한 환경에 대한 고려가 있을 수 있다. 클라이언트/서버 구조로의 전환은 많은 업체의 홍보 자료에서와 같이 환상적인 것만은 아니다. 자신의 환경과 전환 전략, 구축 기술에 따라 적용여부와 적용정도를 충분히 고려한 후 구축하는 것이 성공의 열쇠라고 본다.

이 글이 군이 앞으로 추구하고자 하는 시스템의 구조를 선택하고 클라이언트/서버 시스템이 선택되어 시스템 구축을 진행할 때 조금이나마 도움이 되기를 기대하며 이 글을 마친다.