데이터베이스 도식에 비 의존적인 다 계층 어플리케이션 구조

박병석* 한영태* 민덕기*
*건국대학교 컴퓨터.정보통신공학과
{bspark, ythan, dkmin}@cse.konkuk.ac.kr

A Multi-Tier Application Architecture Independented on Database Schema

Byongseok Park*, Youngtae Han*, Dugki Min*
*Dept of Computer Science and Engineering, Konkuk University

요 약

기업의 전사적인 자원 관리(ERP)의 필요성이 급증하면서 이에 따른 광범위한 데이터베이스 관리의 중요성이 점차 부각되고 있다. 이러한 이유로 말미암아 데이터베이스의 확장 즉, Schema의 변화에 따른 데이터 베이스 접근을 위한 응용프로그램으로써 Multi-Tier 구조가 제시된 바 있다. 그러나 또 다른 시스템에 적용하는 경우 데이터 베이스의 접근을 제 변경해야 하는 등, 여전히 많은 문제점이 드러나고 있다. 본 고에서는 이러한 문제점을 해결하기 위한 방법으로써 데이터 베이스의 Schema에 독립적으로 접근할 수 있는 새로운 Multi-Tier 구조를 제시한다.

1. 서론

급변하는 경영환경에서 기업은 경쟁력을 유지하기 위해 기업의 각종 자원을 하나의 체계로 통합 관리하는 ERP(Enterprise Resource Planning) 시스템 즉, 데이터 베이스를 중심으로 응용프로그램 형태의 서비스를 제공하는 통합 패키지 소프트웨어를 도입하고 있다.

ERP와 같은 데이터 베이스 Transaction이 빈번한 프로그램을 개발하다 보면, 데이터 베이스의 Schema가 개발 과정에서 계속적인 변경을 가져온다. 따라서 계속적으로 데이터 베이스를 제어하는 부분을 수정하여야만 한다. 뿐만 아니라 동일한 기능을 하는 다른 시스템에 적용하는 경우 역시 다시 이 부분을 수정해야만 한다. 이를 해결하기 위한 대안으로써 Application Logic과 Data Source를 분리하기 위하여 Multi-Tire 개념이 나왔으나, 이는 단지 Application Logic이 Data Source의 종류나 환경 등의 Platform에 Independent하게 할뿐, Schema의 변화에 대한 대처방법이 없다.

본 논문에서는 이러한 데이터 베이스 Schema의 변화에도 아무런 코딩이나 수정 없이 재사용 가능한 구조를 제시한다. 즉, 데이터 베이스의 Schema에 대하여 독립적으로 실행될 수 있도록 Application Logic Tier를 작성하기 위하여, 실질적인 작업을 수행하는 객체로부터 Storage Tier에 적용시킬 Query를 생성하는 객체를 따로 떼어내어 별도의 객체 Layer를 만든 후, 각각의 객체간의 Data교환 방식을 새롭게 적용하는 방법을 제안한다.

본 고에서는 2절에서 현재 제시되어있는 Multi-Tire Application의 구조와 그에 따른 실질적인 취약점을 파악하고, 3절에서는 2절에서 설명된 Multi-Tire 구조를 개선하여 데이터 베이스 Schema에 독립적으로 수행되면서도 재활용능력을 최대화하는 방법을 제시한다. 그리고 마지막으로 4절에서 결론과 향후 과제에 대해서 알아본다.

2. Multi-Tier 구조

초기 Client/Server 구조가 등장한 이후, 기존의 Mainframe 시스템보다 많은 부분이 개선되었지만 다

음과 같은 문제점이 있었다.

- Server측의 변동사항에 대한 대비책 필요: 서버가 어떤 이유로 인해서 변화가 생길 경우 중간에서의 중재가 필요했다. 데이터 베이스만 바뀌어도 전체 Application을 다시 작성해야 할 정도였다.
- 반복되는 결과에 대한 불필요한 Server 부하 : 반복적인 작업의 연속을 Server에 까지 전달하여 똑같은 결과를 돌려주는 것보다는 중간에서 저장하고 있다가 돌려주는 것이다.
- Server측의 자원의 효과적인 활용 : 많은 Client에 대한 응답을 주기 위해서는 Server측의 부하가 커지는데, 이럴 경우, 중간에서 서버로 전달되는 요청을 통제가 필요했다. 또한 Server측의 자원의 감시 및 통제 역시 필요했다.

이러한 요구로 등장한 Multi-Tier는 그림1과 같은 구조를 가지고 있다.

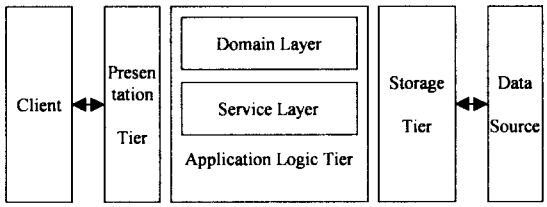

그림 1 일반적인 Multi-Tier Application 구조

Presentation Tier에서는 Client와의 통신을 담당하며 최초의 Client로부터의 Request를 받아들여 이를 Application Logic Layer에 보내고 그에 해당하는 Response를 받아 Client에 전달하는 역할을 한다.

Application Logic Tier는 다시 Domain Layer와 Service Layer로 나뉘는데 Domain Layer는 해당 System에서만 처리하는 기능을 가진 객체들이 있는 곳이며, Service Layer는 해당 System뿐만이 아닌 일반적인 System에 필요한 기능을 가진 객체들이 있는 곳이다.

Storage Tier는 Data Source를 관리하는 부분으로 Application Logic Layer에서 동일한 방법을 통하여 Data Source를 제어하도록 한다.

이 구조에서 가장 눈여겨보아야 할 부분은 Storage Tier를 따로 분리했다는 것이다. 이것은 Data Source의 Platform과 관계없이 전체 Application이 수행되도록 하겠다는 의미이다. 이로 인해서 Data Source의 Platform에 독립적이 되었지만, 변화하는 Data Source의 Schema에 대한 Application Logic Layer의 재사용성은 없었다.

3. 데이터 베이스 Schema Independent 구조

Multi-Tier 구조에서 Storage Tier를 사용함으로써 Data의 Platform에는 독립적이었지만 실제 ERP과 같이 데이터 베이스 Transaction이 많은 Application을 개발할 때, 수시로 바뀌는 Schema의 변화에 대한 대비책은 전혀 없었다. 즉, 데이터 베이스 Schema에 맞추어 프로그램을 작성하다가 그 이후에 어떤 시점에서 데이터 베이스의 Schema가 바뀌는 경우에는 Storage Tier의 내용까지도 모두 바꾸어 주어야만 된다.

하지만 기존 Multi-Tier중 Application Logic Tier를 더욱 세분화하고 그 사이의 Parameter Data Format을 바꾸어줌으로써, 이러한 문제를 해결할 수 있다. 그림 2는 이러한 구조를 설명하고 있다.

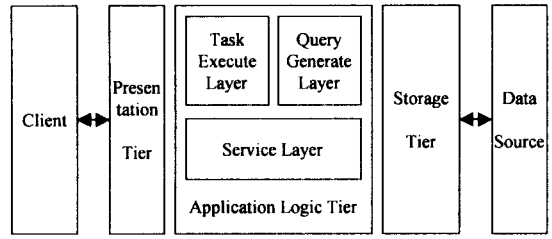

그림 2 데이터 베이스 Schema Independent Multi-Tier 구조

Application Tier는 Task Execute Layer와 Query Generate Layer및 Service Layer의 3개 층으로 구성된다. 이 3가지 Layer 가운데 기존의 Multi-Tier 구조와 다른 2가지 Layer의 기능은 다음과 같다.

Task Execute Layer에는 Unit Task Execute 객체가 존재하며 Query Generate Layer의 객체들을 사용하여 실질적인 작업을 수행하는 역할을 한다. 또한 Presentation Tier로부터 외부에서 들어오는 데이터를 아무런 가공 없는 상태에서 받아서 내부적으로 사용하는 Parameter Data Format으로 Translation하는 역할을 한다. 내부적으로 Parameter Data Format은 여러 가지가 있을 수 있다. 예를 들면, 대표적인 자료구조중의 하나인 Hashtable일 수도 있으며, XML Document가 될 수도 있다.

Query Generate Layer에는 데이터 베이스 Schema Independent 객체가 존재하며 실질적인 Query를 생성하게 된다. 즉, Task Execute Layer로부터 받은 Parameter Data를 Storage Tier가 인식할

수 있는 Query를 생성하며 수행하게 된다.

Task Execute Layer와 Query Generate Layer를 별도로 작성하여 각 Layer의 Objecr들의 재사용 성을 높였다.

이들간의 Data 전달의 구현 예를 Web상에서의 일반적인 입력 형태인 HttpServletRequest와 대표적인 자료 구조의 하나인 Hashtable로 들어본다면 그림 3과 같다.

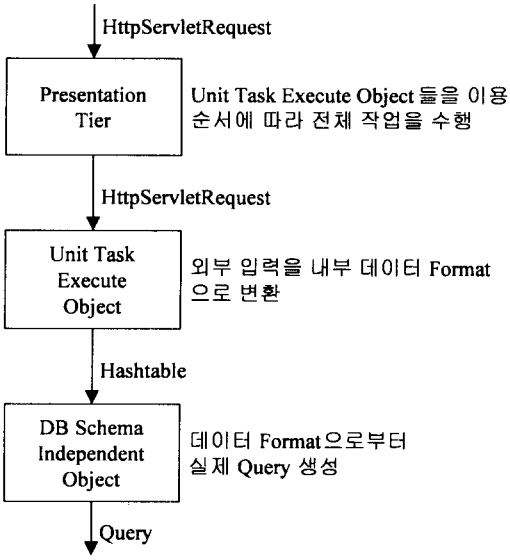

그림 3 HttpServletRequest와 Hashtable 기반의 Data Translation 과정

Presentation Tier로 들어오는 최초의 Client의 HttpServletRequest를 받게 되고 자신의 내부적인 Logic에 따라 처음 수행 되어야하는 Unit Task Execute 객체를 찾게 되고 Unit Task Execute 객체를 하나 생성한 후, HttpServletRequest를 아무런 가공 없이 Unit Task Execute 객체로 넘기게 된다. HttpServletRequest를 넘겨받은 Unit Task Execute 객체는 데이터 베이스의 정보와 Query를 생성하는 데에 필요한 모든 정보를 Hashtable에 저장하여 Query Generate Layer의 객체로 넘겨주게 된다. Hashtable을 넘겨받은 Query Generate Layer의 객체는 데이터 베이스에 알맞은 Query를 생성하여 수행하게 된다.

4. 결론 및 향후 과제

기존에 제시되어있던 Multi-Tier의 구조에서는 데이터 베이스 Schema의 변화에 대한 Dynamic한 대처가 힘들었던 반면 Multi-Tier중 Application Tier를 다시 2부분으로 나누어 한 부분은 외부로부터의 입력된 Data를 내부의 Data 형태로 변환하며 다른 한 부분은 실질 적인 Query를 생성하는 작업을 함으로써 Schema의 변화에 대한 Run-Time의 Dynamic한 Query의 생성으로 대처할 수 있게 되었다.

향후 과제로는 Unit Task Execute 객체에서의 Translation cost의 측정 및 최소화 방법, 또한 내부 Parameter Data Format 할 수 있는 여러 가지 Format 중 최상의 Format을 찾아내는 일 등이다.

5. 참고 문헌

[1] Craig Larman, "APPLYING UML AND PATTERNS" Prentice Hall Published, 1998

[2] SilverStream Software Incorporation, SilverStream Application Server 2.0 Technical White Paper, September 1998

[3] Sun Microsystems, Enterprise JavaBeans Specification version 1.0, 1998

[4] Oracle Corporation, Overview of Oracle Application Server, 1998

[5] Sun Microsystems, Introduction to NetDynamics for NetDynamics 5.0, 1000

[6] ComputerWire PLC, WebLogic Tengah Application Server, July 1998

[7] Mark Grand, "Patterns in Java" Wiley Computer Publishing, 1998

[8] R. Orfali, D. HarKey, J. Edwards, "The Essential Client/Server Survival Guide" Wiley Computer Publishing, 1996

[9] 한국전자 통신연구원, CITIS/CALS 통합 DB기술 개발, 1998. 6.

[10] Bruce A. Burton et al., "The Reusable Software Library," IEEE Software, Vol. 4, No. 4, pp. 24-33, 1987