The Journal of Digital Policy & Management 2013 Oct; 11(10): 411-417

개발 생산성 있는 엔터프라이즈 프레임워크 구축을 위한 N-Tiers 플랫폼의 설계

이명호
세명대학교, 전자상거래학과

A Design of N-Tiers Platform for Building Enterprise Framework with Development Productivity

Myeong-Ho Lee
Dept. of eCommerce, Semyung University

요 약 정보 기술의 디지털 컨버전스 속에서 기업들은 전략적으로 소프트웨어 개발을 활용하고 있다. 정보 기술의 소프트웨어 개발의 발전 방향도 전사적 데이터 및 비즈니스 프로세스의 통합을 통하여 전 영역에 걸쳐 신기술을 이용하여 표준화와 통합화로 진보되고 있다. 그러나 점차 세계화가 가속화 되고 있는 기업들의 개발 환경은 각종 혁신 활동에 대한 통제 및 관리와 비용 절감 노력을 하고 있음에도 불구하고 프로젝트에 따라 고객의 요구사항에 표준화가 되어 있지 못하다. 또한 기존의 시스템과 새로운 시스템과의 연결이 조화롭지 못함에 따라 과도한 소프트웨어 개발 통합 작업과 관리가 필요하고 있다.
따라서 본 연구에서는 MVC 디자인 패턴을 적용하여 개발 생산성 있는 N-Tiers 플랫폼에서 엔터프라이즈 프레임워크 환경의 프리젠테이션 티어와 미들(비즈니스) 티어 및 데이터(EIS) 티어의 설계를 제안토록 한다.

주제어 : 소프트웨어 개발, N-Tiers 플랫폼, 엔터프라이즈 프레임워크, MVC 디자인 패턴

Abstract Enterprises utilize software development strategically within digital convergence of information technology. Software development direction of IT takes advantage of new technology through integration of across-the-board data and business process standardization and integration. But, software development environment of enterprises which globalization is accelerated gradually is doing cost-cutting effort with controls and administration about various reform activity. Nevertheless, have not normalized in customer's requirement according to project, there is visual point that excessive software development integration work and administration are necessary according as connection is not harmonious with new system. Therefore, in this study, to construct these real-time integration environment, do to propose database implementation that have productivity by deent,ing N-Tiers platform for building enterprise framework with development productivity presentation, middle(business) and data(EIS) tier that take advantage of MVC design pattern

Key Words : Software Development, N-Tiers Platform, Enterprise Framework, MVC Design Pattern

Received 23 August 2013, Revised 25 September 2013
Accepted 20 October 2013
Corresponding Author: Myeong-Ho Lee(Semyung University)
Email: mhlee@semyung.ac.kr

ISSN: 1738-1916

1. 서론

개발 플랫폼 환경의 변화는 1960-1970년대 중앙 집중 방식의 메인 프레임 시대에서 1980년대 클라이언트-서버 시대를 지나, 2000년대 분산 컴퓨팅과 웹 기반의 N-Tiers 시대와 모바일 웹앱을 거쳐, 이제는 클라우드 컴퓨팅 환경의 N-스크린 서비스의 시대로 발전과 진화를 하고 있다[1]. 이와 같은 플랫폼의 변화는 IT 산업의 소프트웨어 분야에서도 배포 문제에 따른 소요 비용의 증가 때문에 웹 애플리케이션 기반을 거쳐 크로스 플랫폼과 모바일 웹앱과 N-스크린으로 전환되고 있는 실정이다[2]. 인터넷 기반의 웹 애플리케이션 시스템은 분산 컴퓨팅 시스템 이지만 서버 측에 상당히 많은 컴퓨팅이 필요하게 되었고, 서비스가 다양화 될수록 복잡도가 크게 증가되고 있다. 또한 점차 다양한 상호 작용과 UI/UX에서도 사용자의 요구가 점점 더 많아지고 있는 실정이다. 이러한 고객의 요구사항에 따라 N-Tiers나 엔터프라이즈 개발 환경에서 다양하고 복잡한 요구사항들을 수용하기 위하여 MVC(Model-View-Controller) 디자인 패턴 기반의 소프트웨어 디자인 패턴을 적용한 생산성 있는 프리젠테이션 티어의 설계와 개발 생산성이 있는 프레임워크를 구축하기 위한 미들(비즈니스) 티어와 데이터(EIS:Enterprise Information System) 티어 설계가 필요한 시점이다[4][8].

그러나 현재까지 표준 엔터프라이즈 프레임워크 환경들은 매우 복잡하고 기술 습득 시간이 많이 소요됨에 따라 현업에서 적용하는데 어려움을 격고 있는 실정이다.

따라서 본 연구에서는 MVC 디자인 패턴 기반의 소프트웨어 디자인 패턴을 적용하여 개발 생산성 있는 엔터프라이즈 환경의 프리젠테이션 티어(Presentation Tier)와 비즈니스 로직이 동작되는 미들 티어(Middle Tier) 및 비즈니스 영역에서 발생하는 모든 정보를 보관 및 관리하는 데이터 티어(Data Tier)의 설계를 제안토록 한다.

2. 디자인 패턴에 대한 고찰

2.1 MVC 디자인 패턴

MVC 디자인 패턴은 모델, 뷰, 컨트롤러로 구성된 소프트웨어 디자인 패턴이다. 다시 말하면, 애플리케이션을 모델, 뷰, 컨트롤러의 세 가지의 핵심적인 컴포넌트로 분리하여, 각각의 컴포넌트들이 고유의 업무를 처리하도록 하는 것이다[3][7].

뷰는 시각적인 표현을 담당하는 것으로 사용자가 눈으로 볼 수 있고 사용자에 상호작용 하는 인터페이스이다. 컨트롤러에 의해 생성되며, 어떠한 처리과정도 나타나지 않는다. 오직 모델의 데이터를 사용자가 요구하는 형식으로 보여주는 역할을 담당한다.

컨트롤러는 사용자의 요청을 해석하고 요청을 수행하기 위해 필요에 따라 모델과 뷰 부분을 호출하여 모델과 뷰의 관계를 제어 해주는 역할을 담당한다. 이러한 MVC 디자인 패턴의 장점은 보여주는 것으로부터 데이터와 로직을 분리했기 때문에 같은 모델을 사용하는 다중 뷰를 두어 코드의 재사용성을 높일 수 있고, 구현, 테스트 및 유지보수가 쉬워지게 된다.

모델은 표현에 의존하지 않고 애플리케이션 에서 필요로 하는 본질적인 모든 데이터와 비즈니스 로직을 캡슐화한다. 그리고 사용자가 뷰나 컨트롤러를 통해 데이터를 요청할 때, 뷰에 데이터를 주는 역할을 담당한다.

웹 애플리케이션에서는 이러한 방식의 통지가 불가능하거나 구현하기 어렵기 때문에 모델에 변경사항을 뷰로 밀어 넣는 푸시 방식 대신 클라이언트가 모델로부터 끌어오는 풀 방식을 사용한다.

2.2 J2EE 디자인 패턴

J2EE/Java EE는 분산 엔터프라이즈 애플리케이션을 개발하는데 현재까지 가장 성공적인 플랫폼으로 알려져 있다. JJ2EE/Java EE는 공개 표준에 기반하고 있으며, 엔터프라이즈 컴퓨터 환경에서 요구하는 모든 분야의 표준을 제공한다. 또한 이기종의 환경에서 상호 운용성을 보장해 주는 플랫폼이기도 하다. 소프트웨어 패턴은 크게 디자인 패턴, 아키텍처 패턴, 분석 패턴, 생성 패턴, 구조 패턴, 행위 패턴으로 분류할 수 있다[5]. J2EE 패턴은 디자인 패턴과 아키텍처 패턴 사이에 있다고 할 수 있다. 따라서 본 연구의 대상은 논리적 아키텍처 패턴 중에서 비즈니스 티어에서의 디자인 패턴으로 볼 수 있다.

2.3 EJB 디자인 패턴

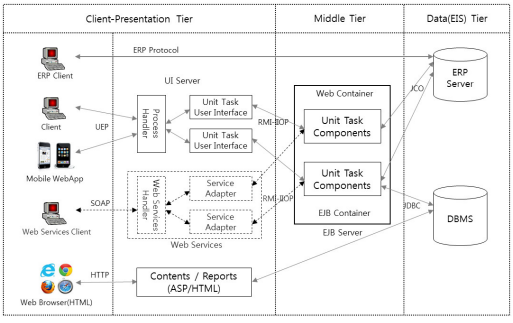
EJB(Enterprise JavaBeans) 기반 시스템의 설계 시

가장 큰 어려움은 성능이나 유지보수 및 이식성과 같은 사항들을 충족시키는 정확한 아키텍처를 선택하거나 로직을 분할하는 일이다. EJB 디자인 패턴을 처음으로 시작한 사람은 Floyd Marinescu이다[6]. 그는 EJB 디자인 패턴을 인터 티어 데이터 전송 패턴, 세션 빈 서비스 패턴, 엔티티 빈 패턴, 그리고 프라이머리 키 생성 패턴으로 크게 분류하였다. 이와 같은 패턴의 분류는 여러 가지 기술에 사용할 수 있는 범용적이고 추성적인 패턴들을 설명하기 보다는 EJB 특정 문제들을 다루면서 EJB에 적용할 수 있는 것들에 초점을 맞추고 있다.

3. N-Tiers 플랫폼의 설계

3.1 N-Tiers 플랫폼의 구조

N-Tiers 플랫폼의 환경 구조는 대용량 분산 컴퓨팅 시스템 구조를 지향하고 있으며, 웹 브라우저가 운용되는 PC 환경의 프리젠테이션 티어, 웹 애플리케이션 및 비즈니스 로직이 동작되는 미들 티어 그리고 비즈니스 영역에서 발생하는 모든 정보를 보관 및 관리하는 데이터(EIS) 티어로 구성되어 있다[9]. [Fig. 1]은 본 연구에서 제안하는 N-Tiers 구조를 도식화한 것이다.



[Fig. 1] Architecture of N-Tiers Platform

프리젠테이션 티어에는 최상위 레벨인 프로세스들을 관리하는 프로세스 핸들러와 프로세스를 구성하는 단위 작업들의 단위 작업 사용자 인터페이스 객체가 위치하게 설계한다. 그리고 각각의 단위 작업 사용자 인터페이스 객체에 상응하는 단위 작업 비즈니스 객체 컴포넌트가 미들 티어를 구성한다. 이 층의 단위 작업 컴포넌트는 EJB 컴포넌트로 구현되며, 기존과 마찬가지로 비즈니스 데이터를 다루고 처리하는 역할을 수행한다. 이 단위 작업 컴포넌트들은 실제 데이터(EIS) 티어에 데이터 객체와 통신하게 된다.

지금까지 제안된 개발의 형태를 살펴보면, 각각의 단위 기능들을 나열하는 형태로 애플리케이션이 개발되었다. 지금도 대다수의 애플리케이션들은 이 같은 형태로 개발되고 있으며, 기반 기술이 OOP 기반이기는 하나 비즈니스와 프로세스의 변화에 유연하게 대처하는 데에는 한계를 가지고 있었다.

따라서 본 연구에서는 이 같은 한계를 극복하기 위해 비즈니스와 프로세스와 코드의 일치화를 목적으로 제안하는 프리젠테이션 티어의 사용자 인터페이스 영역 부분에 대한 설계 방법이다. 비즈니스의 프로세스를 조사하고, 다시 그 프로세스를 구성하는 작업들을 잘 도출하여 구성한다. 그리고 실제 이렇게 조사된 프로세스의 작업 단위별로 개발이 이루어지며, 최종적으로 이 작업들의 조합이 애플리케이션 시스템이 된다. 또한 일반적인 MVC 디자인 패턴을 그대로 사용한 것이 아니라 J2EE(Java EE) 환경에 맞는 구조로 변경하여 설계토록 한다.

3.2 프리젠테이션 티어의 설계

프리젠테이션 티어에서는 UI 객체를 설계하고, N-Tiers 환경에 MVC 디자인 패턴을 도입하여 패키지를 이용하여 최적화하도록 한다. 패키지에는 모델, 뷰, 컨트롤러뿐만 아니라 ListDialog 파일과 SelectListener, Exit Listener 파일이 포함되어 있도록 설계한다. 특정 단위 작업을 개발할 때 기본적으로 패키지를 import하여 단위 작업 고유의 모델, 뷰, 컨트롤러, ListDialog를 생성한다.

패키지의 모델 클래스는 비즈니스 객체(BO)의 가치 객체(VO)의 값을 가지는 추상화 클래스로 컨트롤러의 요청(처리 명령)에 따라 가치 객체의 값을 받아 뷰에서 적절한 형식으로 값을 보여줄 수 있도록 값을 가지고 있다. 뷰의 변경된 데이터를 가지고 컨트롤러의 단위 액티비티에 따라 미들 티어의 가치 객체에 데이터를 전송하는 역할을 수행한다. 따라서 단위 작업 사용자 인터페이스에서는 모델 클래스를 상속 받는 단위 작업명에 모델 클래스를 첨부하여 생성하며, 미들 티어의 가치 객체를 참조하게 설계한다.

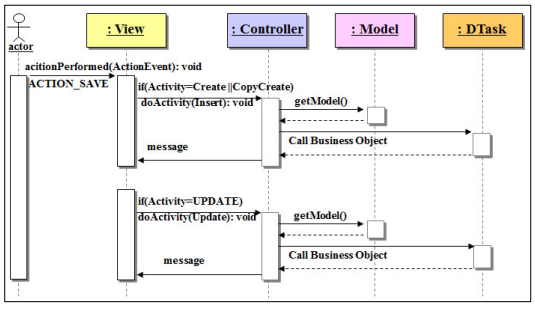
본 연구에서는 단위 작업당 오직 하나의 메인이 되는 뷰를 가지도록 설계하며, 다양한 형식의 검색을 위해서는 하나의 ListDialog를 사용하도록 한다. 또한 사용자의 이벤트를 받기 위해 메인 메뉴 액션을 설계하도록 한다.

뷰 클래스는 눈에 보이는 뷰를 대표하는 클래스이다. 이 클래스는 프레임을 상속 받아 구현된다. 이 외에도 단위 작업이 처리되기 위해 다른 단위 작업을 참조하게 될 경우와 단위 작업에서 특수한 목적으로 어떤 액티비티를 수행해야 할 경우에는 그 액티비티에 따른 하부 액션이 각각의 단위 작업에서 정의될 수 있다. 뷰 클래스에는 액션뿐만 아니라 뷰 생성에 기본이 되는 5가지의 메서드를 포함하게 설계한다.

ListDialog 클래스 역시 넓게는 MVC 디자인 패턴의 뷰에 해당하는 것으로 시각적으로 표현되기 위해 Dialog를 상속받아 구현한다. 이 클래스는 객체의 수정, 조회, 삭제 시에 공통으로 사용되는 리스트 검색 뷰 이다.

컨트롤러 클래스는 모델과 뷰, ListDialog 사이에서 모든 단위 액티비티들을 수행하는 클래스로 모델, 뷰, ListDialog를 생성하는 메서드와 모델의 값을 가져오거나 반환하는 메서드, 단위 액티비티를 처리하는 메서드 등을 구현하고 있다. 단위 액티비티는 단위 작업 내에서 수행되어지는 일련의 활동이다. 단위 액티비티에는 MainActivity와 SubActivity 두 가지가 존재하게 설계한다. MainActivity는 CREATE, UPDATE, DELETE, DIPLAY, SAVE 등과 같이 모든 단위 작업이 공통적으로 가지는 주요 액티비티들로서 주문 생성, 주문 수정, 주문 삭제, 주문 조회, 주문저장 등이다.

[Fig.2]는 ACTION_SAVE 액티비티의 시퀀스 다이어그램을 도식화한 것이다.



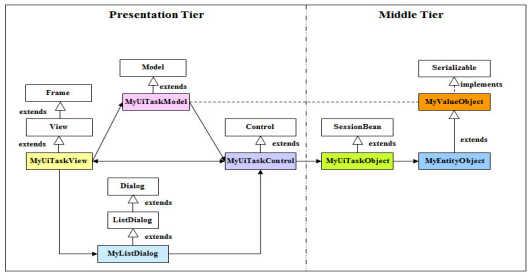
[Fig. 2] Sequence Diagram of ACTION_SAVE

SubActivity는 단위 작업 고유의 액티비티와 단위 작업의 단위 업무를 위해 다른 단위 작업를 호출하는 등의 특정 작업에서만 필요한 액티비티를 말하며, 주문 단위 작업에서 주문 번호를 체번하는 것이나 고객 정보를 입력하기 위해 고객 단위 작업을 호출하는 것 등이다.

패키지의 모델 클래스는 비즈니스 객체의 가치 객체의 값을 가지는 추상화 클래스로 컨트롤러의 요청(처리 명령)에 따라 가치 객체의 값을 받아 뷰에서 적절한 형식으로 값을 보여줄 수 있도록 값을 가지고 있다. 뷰의 변경된 데이터를 가지고 컨트롤러의 단위 액티비티에 따라 중간 티어의 가치 객체에 데이터를 전송하는 역할을 수행한다. 따라서 단위 작업 사용자 인터페이스에서는 모델 클래스를 상속 받는 단위 작업명에 모델 클래스를 첨부하여 생성하며 미들 티어의 가치 객체를 참조하게 설계한다.

3.3 미들(비즈니스) 티어의 설계

표준 엔터프라이즈 비즈니스 티어의 물리적 구조는 엔티티 객체와 단위 작업 객체의 비즈니스 객체로 구성되는데, 본 연구에서는 여기에 MVC 도입과 함께 새로이 가치 객체라는 것을 추가하여 설계한다[3]. 이러한 단위 작업의 비즈니스 객체들은 데이터베이스 티어의 ERP 데이터베이스 서버나 이 기종 데이터 서버와 통신을 하는 구조를 가지고 있다. [Fig. 3]은 특정 프로세스의 특정 단위 작업을 추출하여 시스템을 구성했을 때, 프리젠테이션 티어와 미들 티어 간의 프레임워크를 표현한 것이다.



[Fig. 3] Framework of Special MyUitask

본 연구에서는 비즈니스 로직을 담당하는 비즈니스 객체를 3가지 종류로 설계한다. 첫 번째는 순수하게 데이터만을 가지고 있는 가치 객체이고, 두 번째는 가치 객체

를 상속받아 생성되는 엔티티 객체이다. 이 엔티티 객체는 객체의 상태를 제어하고, 데이터베이스와 상호 작용하여 결과를 다시 가치 객체에 리턴하는 역할을 한다. 세 번째는 어떠한 비즈니스 작업을 처리하는 작업 객체이다.

이상과 같은 가치 객체와 엔티티 객체는 일반적인 자바 객체로 설계하며, 작업 객체는 EJB 스펙에 맞는 무상태 세션 빈으로 설계한다.

일반적으로 비즈니스 티어에서는 비즈니스에 필요한 데이터를 처리하기 위해 데이터를 표현해주는 속성들과 처리하기 위한 메서드가 요구된다. 기존의 방법론에서는 엔티티 객체와 작업 객체 두 가지가 정의되어 엔티티 객체는 비즈니스 프로세스의 수행에 필요한 속성과 메서드를 모두 가지고 실제 데이터 티어와 통신하고, 작업 객체는 비즈니스 데이터의 은폐를 위해 클라이언트의 요청에 따른 비즈니스 프로세스 수행 메서드를 정의하고 있다. 그러나 이와 같은 방법론에서는 메서드 수행 결과 값을 받거나 매개변수를 넘길 때는 클라이언트에서 엔티티 객체에 직접 접근함으로서 논리적인 구조의 모순이 혼재되어 있었다.

본 연구에서 제안하는 방법론은 이러한 모순을 없애고, 향후 웹 서비스에 대응하기 위해 속성과 메서드를 분리하여 속성을 가지는 객체를 가치 객체로 정의하고 엔티티 객체는 이 가치 객체를 상속 받아 비즈니스 룰 처리와 객체 상태처리 및 메서드만을 수행하도록 재정의하게 설계한다. 이것은 가치 객체는 순수한 데이터로서의 기능만을 가지는 객체로 설계하는 것이다.

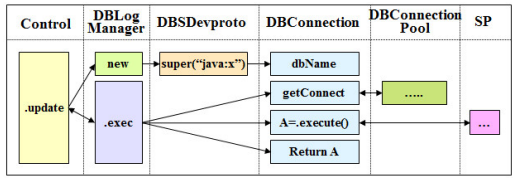
모든 객체는 유일해야 하며 고유의 유일 객체 id를 갖는다. 이 id는 데이터베이스 서버의 필드 속성 중 id 엔티티 이용해야 하고, 객체의 상태가 완벽하더라도 유일 id를 갖기 전에는 인증된 객체라고 볼 수 없다.

EJB 환경 하에서는 클라이언트가 직접 엔티티 비즈니스 객체의 메서드를 호출할 수 없고, 객체의 상태에 따라 작업 비즈니스 객체를 통해서 일을 처리하게 된다. 앞에서 설명한 객체는 클래스라는 것을 통해서 표현할 수 있다. 따라서 객체를 구현하기 위해서는 클래스를 반드시 정의해야 한다. 엔티티 객체의 클래스는 속성과 메서드로 구성되는데, 그 중에 속성이라는 것은 객체의 특징을 클래스 내에 정의한 것이며, 메서드는 이러한 클래스의 속성을 활용하여 할 수 있는 일에 해당된다.

3.4 데이터(EIS) 티어의 설계

자바 프로그램에서 데이터베이스에 접근하기 위해 JDBC를 이용한다. 이때, 데이터베이스로 커넥션을 맺는 일은 매우 느리며 자원을 많이 소모하는 작업이다. 그러므로 불특정 다수의 사용자들이 동시에 데이터베이스의 커넥션을 요구한다면 최악의 경우 서버가 다운되기도 한다. 이것을 해결하기 위해 이용하는 것이 커넥션 풀이다. 커넥션 풀은 사전에 일정량의 커넥션 객체를 만들어 공유된 장소에 모아두어 커넥션 객체 생성에 소요되는 시간을 줄여 속도향상을 기대할 수 있다. 또한 사용이 끝난 커넥션 객체를 다시 공유된 장소에 넣어두어 데이터베이스와의 연결을 효율적으로 관리하는 역할을 한다. 과거에는 이 같은 커넥션 풀을 직접 개발하곤 했으나 최근에는 JDBC 드라이버 내에 자체적으로 내장되어 있는 경우가 많고, 컨테이너에 포함된 커넥션 풀을 사용하기도 한다. 본 연구에서는 오픈 소스 웹 애플리케이션 서버인 JBoss에서 제공하는 커넥션 풀을 사용하도록 설계한다.

비즈니스 데이터를 처리하기 위해 [Fig. 4]와 같은 내부 데이터 처리 절차를 가지도록 설계 한다.



[Fig. 4] Sequence of Inner Data Process

사용자 인터페이스의 컨트롤에서 사용자가 어떤 작업을 요청하게 되면 작업 객체(세션 빈)는 엔티티 객체의 메서드 내에서는 UI에서 넘겨받은 인자 값들을 벡터에 담아서 SP(Service Pool) name과 함께 DBWrapper 쪽에 보내게 된다. 그리고 결과값을 넘겨줘야 하는 경우는 ArrayList 형태로 UI쪽으로 보내고 있다. 이때, DBWrapper라는 클래스는 비즈니스 객체 데이터들이 항상 데이터베이스 티어와 접속하여 일을 처리하므로 매번 커넥션하는 수고를 덜고, 개발 시 표준적인 방법을 적용하기 위해 사용되어지는 유틸리티이다.

JDBC는 자바에서 데이터베이스를 조작하는 표준 방식이며 데이터베이스 종류에 상관없이 똑같은 코드로 데

이터베이스를 조작할 수 있다. DBWrapper에서는 이러한 JDBC를 이용하여 해당 DB와의 커넥션을 설정하고, 'lookup' 메서드를 통해 JBoss 커넥션 풀로부터 커넥션 객체를 얻어 Entity Object에서 호출하는 SP를 수행하여 결과값을 Array List로 넘겨준다. 이 때 연결 할 데이터베이스 서버와 데이터베이스의 정보들은 배포 시 데이터소스 부분에서 JNDI 이름과 JNDI URL에 지정하게 된다. 따라서 데이터베이스가 바뀌더라도 프로그램 소스를 수정하지 않고 외부에서 JNDI URL의 조작으로 해결할 수가 있다. JBoss의 XML 파일에는 JNDI 이름과 URL 정보를 가지고 있다.

비즈니스 룰은 BusinessObject에서 적용되어야 하는 룰을 체크하기 위해 BusinessRule 이라는 클래스를 만들어 사용한다. 그리고 BusinessRule 클래스는 직렬화 대상에서 제외하기 위해 선언 시 transient라는 키워드 사용하도록 설계한다. 트랜잭션은 어떠한 결과를 실현시키기 위하여 여러 가지의 세부적인 작업이 다 같이 실행되어야 하는 한 단위의 과정을 말한다. EJB가 기업 환경을 기반으로 하고 있기 때문에 이러한 기업 환경에서 데이터의 신뢰성은 아주 중요한 이슈로 취급되고 있다.

따라서 데이터의 신뢰성 확보를 위해서는 트랜잭션 블럭 내부의 모든 과정이 정상적으로 처리되었을 경우에만 커밋으로 데이터를 갱신하고, 그 외의 경우는 롤백을 통해 트랜잭션 이전 상태로 돌려놓은 방법이 사용되고 있다. 트랜잭션의 영역은 트랜잭션의 시작 부분과 커밋 사이를 의미하며, EJB에서도 이러한 트랜잭션의 경계를 정하는 것은 아주 중요한 일이고, 트랜잭션의 범위는 세부적인 일들이 하나의 트랜잭션에 포함되어야 하는지 그 여부를 결정하므로, 정확한 비즈니스 로직의 구현에 있어 대단히 중요하다.

본 연구에서는 트랜잭션을 데이터베이스 내의 SQL문, JDBC, EJB의 세션 빈에서 각각 수행할 수 있게 설계할 수 있지만 OOP 개념 하에 객체 단위로 트랜잭션을 처리할 수 있도록 설계한다.

4. 결론

본 연구에서 프레젠테이션 티어의 설계는 개발 단위가 단위 작업이라는 단위로 세분화되기 때문에 개발 일정이나 역할 분담이 기존 개발 방법보다 수월하며 단순한 객체의 재사용성을 넘어 단위 작업의 재사용을 할 수 있기 때문에, 대규모 프로젝트 시에 이미 개발된 단위 작업들을 이용하면 소프트웨어 생산성을 크게 향상시켜 줄 수 있는 설계 방법으로 제안하였다.

MVC 디자인 패턴을 활용한 미들(비즈니스) 티어의 설계는 확장성, 결함 허용 능력이 있는 시스템 및 가용성이 높은 구조이기 때문에 개발 일정이나 역할 분담이 기존 개발 방법보다 수월해 질뿐만 아니라, 표준화된 비즈니스 로직의 단위 작업 개발은 실제 개발 기간의 단축과 시스템 소프트웨어 품질의 향상을 가져오게 설계하였다.

데이터(EIS) 티어의 설계는 N-Tiers 환경에서 디자인 패턴을 활용한 데이터베이스 티어의 생산성 있는 설계를 제안하였다.

향후 데이터베이스 스키마의 설계를 통한 모바일 웹 앱 서비스 기반의 실무 사례 연구가 지속되어야 할 것이며, EJB 디자인 패턴을 활용한 확장한 연구가 필요하다.

REFERENCES

[1] M.H. Lee, A Design of User Interface by using Design Patterns on X-Internet Environment for Plant Information, Journal of the Korean Institute of Plant Engineering, Vol. 10, No. 1, pp. 121-131, 2005.

[2] M.H. Lee, A Design of Database Tiers by using Design Patterns on X-Internet Environment for Plant Information, Journal of the Korean Institute of Plant Engineering, Vol. 10, No. 2, pp. 103-112, 2005.

[3] M.H. Lee, A Design of Development of Productivity Software on X-Internet Environment, Journal of the Society of Korea Industrial and Systems Engineering, Vol. 28, No. 2, pp. 53-59, 2005.

[4] Chuck Cavaness, Programming Jakarta Struts, O'Reilly, 2002.

[5] Deepak Alur, Dan Malks, John Crupi, Core J2EE Patterns: Best Practices and Design Strategies, Sun Microsystems Press, 2003.

[6] Floyd Marinescu, EJB Design Patterns, John Wiley& Sons, 2002.

[7] Krasner, G. E., Pope, S. T., A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, 1(3), Aug/Sept, 1988.
[8] Riehle, D., Zulighoven, H., Understanding and Using Patterns in Software Development, Theory and Practice of Object-Oriented Systems, 2(1), 1996
[9] http://java-success.blogspot.kr/2011/09/enterprise-java-interview-questions-and.html

이 명 호(Lee, Myeong Ho)

· 1984년 2월 : 아주대학교 산업공학과(공학사)
· 1986년 2월 : 아주대학교 대학원 산업공학과(공학석사)
· 2001년 2월 : 아주대학교 대학원 산업공학과(공학박사)
· 2002년 3월~현재 세명대학교 전자상거래학과 부교수
· 관심분야 : 물류정보시스템, WAS 프로그래밍, 모니터링 시스템
· E-Mail : mhlee@semyung.ac.kr