Journal of the Society of Korea Industrial and Systems Engineering
Vol, 28, No. 2, pp.53－59, June 2005.

X-인터넷 환경에서 생산성 있는 소프트웨어 개발 방법의 설계

이 명 호*

*세명대학교 인터넷정보학부

A Design for Development of Productive Software on X-Internet Environment

Myeong-Ho Lee*

*Division of Internet Information, Semyung University

The environment of IT is, currently, on its developing process to the period of Web Service, Ubiquitous, and Telemetics which not only enable computer and internet to be utilized like the water or the air, but also be a new motivating force for its advance. In the respect of various interactions and User Interface, however, it is requiring more demands from its users, and additional functions which cannot be provided by the Web Browser. It is required a new model for supplementary of the limitation of Web_based according to their customers' requirement, to enrich the good aspects of Client-Server System, and accept such various intricate requirements. It is X-Internet, which has born with these requirements. In this study, therefore, it will be suggested a productive development framework and a method to perform the project on Rich-Thin Client Platform of X-Internet Based.

Keywords : Web Services, User Interface, X-Internet, Rich-Thin Client, Framework

1. 서 론

첨단 정보기술의 급속한 발달로 정보산업 인프라의 환경은 하루가 다르게 변모되고 있으며, 유·무선 디지털 정보 기술과 인터넷 기술의 발달은 기업과 기술 환경의 변화를 유발시켰다. 향후에는 컴퓨터와 인터넷을 물과 공기처럼 이용할 수 있고, IT 산업에 새로운 성장 동력이 될 수 있는 웹 서비스(Web Services), 유비쿼터스(Ubiquitous) 및 텔레메틱스(Telemetics)의 시대가 도래하고 있다. 개발 플랫폼 환경에서도 '60~'70년대 중앙 집중 방식의 메인 프레임 시대에서 2-티어의 클라이언트-서버 시대로, 이제는 분산 환경의 3-티어 및 웹 기반의 N-티어 시대에서 웹 서비스의 시대로 발전과 진화를 하고 있다. 또한 IT 산업의 소프트웨어 분야에서도 배포 문제에 따른 소요 비용의 증가 때문에 웹 기반으로 전환되고 있는 실정이다.

그러나 인터넷 기반의 웹 환경 시스템은 분산 시스템이지만 서버 측에 상당히 많은 컴퓨팅이 필요하게 되었고, 서비스가 다양화 될수록 복잡도가 크게 증가되고 있다. 네트워크 측면에서도 플랫폼의 진화에 따라 자원의 효율성이 낮아지고 있으며, 다양한 상호 작용과 사용자 인터페이스(User Interface : UI)도 사용자의 요구가 점점 더 많아지고 있다. 또한 웹 브라우저가 제공하지 못하는 추가적인 기능의 필요성도 점점 많이 요구되고 있는 실정이다.

현재 HTML(HyperText Markup Language)은 멀티미디어와 사용자 인터페이스의 속도나 구현에 많은 한계를 가지고 있으며, 동적 HTML(Dynamic HTML)에서도 사용자들은 여전히 다양하고 풍부한 사용자 인터페이스를 원하고 있다. 위와 같은 고객의 요구사항에 따라 웹 기

반의 한계를 보완하고, 클라이언트-서버 환경의 장점을 살리고, 다양하고 복잡한 요구사항들을 수용하기 위하여 새로운 모델이 필요하게 되었다. 이러한 요구로 태어난 것이 바로 X-인터넷(X-Internet)이다[9].

따라서 본 연구에서는 이러한 X-인터넷 기반의 리치-씬(Rich-Thin) 클라이언트 플랫폼에서 프로젝트를 수행하기 위한 생산성 있는 개발 프레임웍과 방법을 제안하도록 한다.

2. 기존 연구에 대한 고찰

2.1 X-인터넷

X-인터넷이란 한마디로 말해, 클라이언트-서버에 준하는 사용자 환경을 웹에서 제공해준다. 윈도우의 터미널 서비스(Terminal Service)와 웹을 결합한 것이라고 얘기할 수 있을 것이다.

사용자가 X-인터넷 웹 사이트에 접속하면, 웹 브라우저 화면의 일부가 애플리케이션 영역으로 바뀌며, 사용자는 클라이언트서버 환경과 흡사한 UI를 제공받을 수 있다. 그것은 DHTML를 사용하는 번잡한 방법도 아니며, 개발·배포·보안의 문제가 상존하는 액티브 X 컨트롤도 아니며, 사용상 제약이 많은 자바 애플릿도 아니다. X-인터넷은 진정한 리치-씬 클라이언트이다. 따라서 수행 가능한 인터넷 및 확장된 인터넷이 바로 X-인터넷이다. X-인터넷은 언뜻 보기에 클라이언트-서버 시스템을 웹 페이지에 결합시킨 것 같이 보이지만 많은 차이가 있다. X-인터넷이 클라이언트-서버 시스템의 장점을 포함하고 있기는 하지만 이전의 클라이언트-서버와는 다르다. 다시 말하면, 프로그램은 변함이 없고 클라이언트와 서버 간에 데이터만 주고받는 형태이다.

2.2 소프트웨어 생명주기 모델

가. 폭포수 모델(Waterfall Model)

전통적인 소프트웨어 생명주기 모델로 생명주기를 계획 단계, 분석 단계, 설계 단계, 구현 단계, 시험 단계, 그리고 유지보수 단계로 분류하여 각 단계를 확실하게 매듭을 짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후, 다름 단계로 넘어 가는 모델이다. 개발 시간과 요구 분석 시간이 많이 소요되는 단점을 가진다[6].

나. DoD-2126A 모델

폭포수 모델과 유사하지만 DoD에서는 요구사항 문서, 리뷰 및 베이스라인이 추가되었다. 이 모델의 장점은 구조화 및 정의가 잘 되었고, 몇 가지 베이스라인을 확인하고 작성될 때를 보여준다. 정의되고 문서화된 인터페이스를 요구한다. 몇 가지 리뷰와 검사를 확인하고 시행될 때를 보여준다. "설계 전 정의" 및 "코드 전 설계"의 표시가 강화된다. 단점으로는 폭포수 모델을 기반으로 함에 따라 비슷한 단점이 있다[6].

다. 신속한 프로토타입 모델(Rapid Prototyping Model)

요구 분석의 어려움을 해결하기 위해 실제 개발된 소프트웨어의 일부분을 직접 개발하여 의사소통의 도구로 삼자는 것이다. 개발 타당성을 검증하기 위한 목적으로 개발되기도 한다[5].

라. 나선형 모델(Spiral Model)

이미 개발된 프로토타입을 지속적으로 발전시키자는 진화적 모델의 일종으로 위험 분석(Risk Analysis)을 프로토타입을 발전시킬 때마다 실시하자는 모델이다[2].

마. 복합 모델(Hybrid Model)

복합 모델은 두 가지 이상의 모델의 개념을 결합한 것이다. 많은 복합 모델들은 4GT(Fourth-Generation Techniques)를 기반으로 하며, 이 것은 소프트웨어 공학자들이 아주 높은 수준에서 소프트웨어 특성을 서술할 수 있도록 하는 다양한 도구로 구성된다. 요구 사항을 잘 알면 폭포수 모델로 수행할 수 있고, 요구 사항을 잘 모르면 신속한 프로토타입 모델을 적용할 수 있다. 장점으로는 상황에 가장 적절한 모델을 선택할 수 있으며, 각 모델들의 모든 장점들을 가지고 있다. 단점으로는 특정한 상황에 맞게 설계된 다른 모델들의 유연한 조합 때문에 복합 모델은 광범위한 이해나 인식을 할 수가 없으며, 각 모델들의 모든 단점을 가지고 있다[5].

바. 모델 기반 개발(Model-based Development)

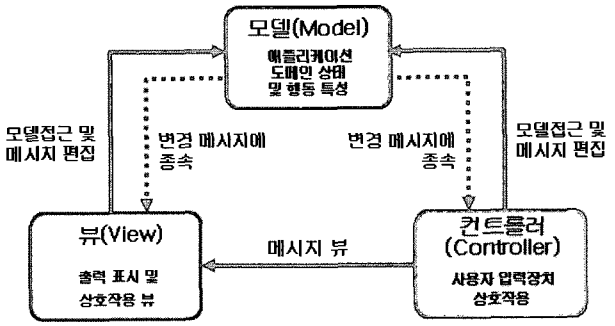
몇몇 모델 기반 개발 접근법이 클라이언트-서버 애플리케이션 및 GUI 기반 애플리케이션을 설계하고 구축하는데 도움을 주도록 최근에 소개되어 왔다. 이 방법은 비즈니스와 정보 시스템(IS) 소프트웨어를 다방면으로 가장 잘 적용시킬 수 있다. 장점으로는 비즈니스 프로세스를 사양화하는데 밀접하게 연결되고, 클라이언트와 서버 애플리케이션을 명확히 묘사하고, 도구는 모델의 사용을 지원하는 것이 가능하고, GUI를 위한 지침 형태를 포함한다. 단점으로는 구조적인 개발 접근을 포함하지 못하며, 특정한 문서, 납기 가능성 혹은 리뷰를 참조하지 못한다[8].

2.3 디자인 패턴(Design Pattern) 모델

패턴이란 특정 설계 환경에서 반복하여 발생하는 특정한 문제를 기술하고, 그 해결 방안으로 검증된 방법을 제시하는 것이다. 이와 같은 해결 방안은 그 구성 컴포넌트들의 역할(Role)과 관계(Relationship) 그리고 그들 간의 상호작용을 기술로 구성된다. 패턴의 종류를 보면, 애플리케이션 도메인으로부터 형식과 개념으로 묘사되는 형태의 개념적 패턴(Conceptual Pattern), 객체, 클래스, 상속 및 집단화와 같은 소프트웨어 설계 구성으로 묘사되는 형태의 디자인 패턴(Design Pattern), 그리고 프로그래밍 언어 구성으로 묘사되는 형태의 프로그래밍 패턴(Programming Pattern)으로 분류된다[7].

가. MVC 모델

MVC(Model-View-Controller) 모델은 <그림 1>과 같이 모델(Model), 뷰(View), 컨트롤러(Controller)의 축약으로 소프트웨어 디자인 패턴이다. 즉 애플리케이션을 모델, 뷰, 컨트롤러의 세 가지의 핵심적인 컴포넌트로 분리하여, 각각의 컴포넌트들이 고유의 업무를 처리하도록 하는 것이다[4].



<그림 1> 일반적인 MVC 모델 구조

모델(Model)은 프리젠테이션에 의존하지 않고 애플리케이션에서 필요로 하는 본질적인 모든 데이터와 비즈니스 로직을 캡슐화한다. 그리고 사용자가 뷰나 컨트롤러를 통해 데이터를 요청할 때 뷰에 데이터를 주는 역할을 담당한다.

뷰(View)는 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.

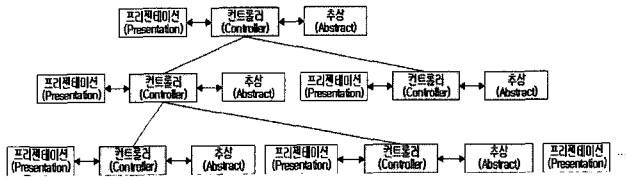
컨트롤러(Controller)는 사용자의 요청을 해석하고 요청을 수행하기 위해 필요에 따라 모델과 뷰 부분을 호출하여 모델과 뷰의 관계를 제어 해주는 역할을 담당한다.

이러한 MVC 모델의 장점은 보여주는 것으로부터 데이터와 로직을 분리했기 때문에 같은 모델을 사용하는 다중 뷰를 두어 코드의 재사용성을 높일 수 있고, 구현, 테스트 및 유지보수가 쉬워지게 된다. 단점으로는 MVC는 상당한 설계를 요구한다는 것으로 설계 시 많은 노력이 필요하며, 높은 단계의 복잡성을 생각해야 한다. 또한 MVC는 규모가 작은 애플리케이션이나 재사용이 적은 것(특수한 목적의 애플리케이션)에는 오히려 자원의 낭비를 초래 할 수 있다.

나. PAC 모델

PAC(Presentation-Abstract-Controller)의 기본 구조는 프리젠테이션-추상-컨트롤러에 의한 아키텍처 패턴이다. PAC의 기본 구조와 계층 구조는 다음과 같다[3].

<그림 2> PAC의 기본 모델 구조



<그림 3> PAC의 계층 모델 구조

프리젠테이션은 외부와 인터페이스를 담당하는 객체이고, 추상은 처리 대상인 모델이며, 컨트롤러는 처리를 제어하는 객체이다. MVC와 유사한 구조이지만 컨트롤러의 역할이 다르다. MVC의 컨트롤러는 마우스 이벤트나 키보드 입력 등의 데이터를 GUI 주변의 입력장치를 추상화하기 위한 것이지만, PAC의 컨트롤러는 애플리케이션의 로직을 구현하기 위한 것이다. 따라서 MVC의 뷰와 컨트롤러는 PAC에서는 프리젠테이션으로 구현된다. 최근에는 뷰와 컨트롤러는 위젯(Widget)이라는 형으로 패키지된 부품으로 제공되는 경우가 많다.

PAC 패턴의 장점은 PAC의 계층을 정의한다는 점이다. 애플리케이션을 구성하는 각 층마다 PAC에 의한 구조가 있기 때문에 이것이 모여 애플리케이션 전체의 구조가 구성된다. 따라서 이 구조는 애플리케이션을 컴포넌트 기반으로 구축하는 경우에 아주 유용한 기능이다.

3. X-인터넷 구조의 구성

3.1 시스템 구조의 진화

시스템 구조는 시스템 구성의 전체 프레임웍을 결정하므로 시스템을 설계하거나 실제 프로그래밍을 하는 작업들보다도 가장 중요하고 우선순위를 두어서 결정해야 하는 문제이다. 구조를 어떻게 구성하는가에 따라 시스템 전체의 성능이 좌우될 뿐만 아니라, 서버의 구성이나 개발 방법의 결정이 뒤따르기 때문이다. 시스템 구조는 최근 몇 년 동안 많은 발전을 거듭하여 왔다. '60~'70년대의 시스템 구조는 메인 프레임을 기반으로 한 호스트 기반의 중앙 집중 구조였다. 이 구조에서는 모든 시스템의 부하가 서버에 작용하므로 시스템의 성능 향상을 위해서는 서버를 계속 증설할 수 밖에 없는 구조적인 단점을 가지고 있었다. '80년대부터 이러한 단점을 해결하기 위해 등장한 시스템 구조가 클라이언트-서버 형태의 2-티어 구조이다. 이 구조는 서버의 부하를 작게 가져가는 대신 클라이언트 프로그램에 시스템의 기능의 일부를 수행하게 하는 구조이다. 그러나 과거 중앙 집중 구조에 비해 서버의 부하는 감소하였으나, 서버와 클라이언트 간의 과도한 병목이 발생하였고, 클라이언트에 프로그램을 각각 설치 및 배포해야 하는 관리문제가 발생하였다. 또한, 모든 비즈니스 로직(Business Logic)이 클라이언트 프로그램에 있다 보니 비즈니스 로직이 중복해서 들어가는 경우가 많았고, 클라이언트 프로그램이 무거워져 PC 성능에 따라 시스템의 성능이 좌우되었으며, 비즈니스 로직이 바뀔 때마다 클라이언트 프로그램을 재 배포해야 하는 문제점을 가지게 되었다. '90년대에서부터 바로 이러한 클라이언트-서버 구조의 단점을 보완하고자 등장한 시스템 아키텍처가 3-티어 구조이다. 3-티어 구조는 표현(Presentation) 기능만 담당하는 클라이언트 영역과 비즈니스 로직 만을 수행하는 애플리케이션 서버(Application Server) 영역, 데이터베이스 기능을 담당하는 데이터베이스 서버(Database Server) 영역으로 이루어져 있다. 이러한 구조를 분산 시스템 구조라고 부르기도 하는데, 이 구조에서는 클라이언트가 가볍기 때문에 클라이언트 PC의 성능에 영향을 덜 받고, 비즈니스 로직이 별도의 애플리케이션 서버에서 작동하므로 시스템 성능 향상이 필요하면 애플리케이션 서버만을 증설해주면 되었다. 그러나 이 또한 클라이언트 프로그램을 배포하고 관리해야 하는 문제는 완전히 해결되지 못한 상태이다.

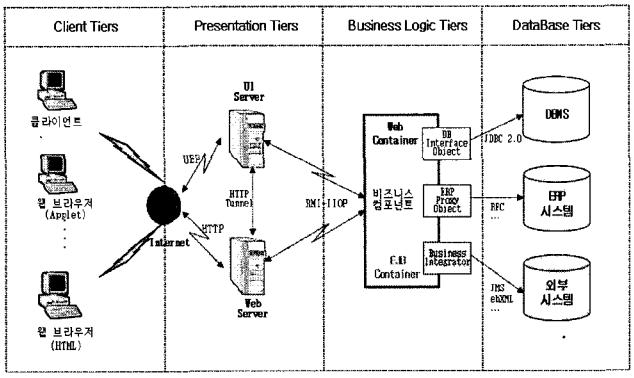
인터넷 환경의 등장은 바로 이러한 3-티어 구조에서 해결하지 못했던 클라이언트 프로그램의 배포 및 관리에 대한 문제까지 해결해주는 기반 환경이 되었다. 이러한 인터넷 환경에서는 웹 서비스를 담당하는 웹 서버가 별도로 필요하게 되었고, 이 구조가 N-티어 구조이다. 이 구조는 클라이언트 영역의 웹 브라우저와 표현 영역에 해당하는 웹 서버와 비즈니스 로직 영역에 해당하는 애플리케이션 서버, 데이터베이스를 담당하는 데이터베이스 서버로 구성되어진다. 따라서 클라이언트는 웹 브라우저만 있으면 되므로 3-티어 구조에서의 프로그램의 배포 및 관리, 클라이언트 PC 성능에 따라 시스템의 성능에 영향을 미치던 문제까지 해결할 수 있게 되었다.

그러나 인터넷 환경에서 사용되는 HTML은 월드 와이드 웹(WWW)을 통해 볼 수 있는 문서를 만들 때 사용하는 프로그래밍 언어의 한 종류로서, 전자문서의 서식을 정의하기 위해 만들어진 것이다. 이러한 HTML은 문서 페이지 단위로 데이터를 전송하므로 기존에 클라이언트-서버 구조나 3-티어 구조에서 개발하던 클라이언트 프로그램과는 상당히 다른 애플리케이션 구조를 가지고 있고, 그 구조 자체가 애플리케이션으로 개발하기에는 비효율적인 구조를 가지고 있다. 또한, HTML은 HTTP (HyperText Transfer Protocol) 라는 인터넷 공용 프로토콜을 사용하므로 보안에 각별히 유의해야만 하는 문제점도 있다. 위와 같은 기존의 인터넷 환경의 단점을 극복하고, 웹 기반 한계를 보완하고, 클라이언트-서버 환경의 장점을 유지하면서, 다양하고 복잡한 요구사항들을 수용하기 위한 새로운 모델이 X-인터넷 구조(X-Internet Architecture)이다.

3.2 X-인터넷의 구조

본 연구에서는 설계하는 X-인터넷 기반의 환경 구조는 N-티어 시스템 구조를 지향하고 있으며, 웹 브라우저가 운용되는 PC 환경의 클라이언트 티어(Client Tiers), X-인터넷이 담당하는 사용자 인터페이스 서버이면서 웹 서버가 없이도 인터넷 환경을 제공해 주는 표현 서버 티어(Presentation Server Tiers), 비즈니스 로직이 동작되는 미들웨어 부분인 애플리케이션 서버 티어(Application Server Tiers), 비즈니스 영역에서 발생하는 모든 정보를 보관 및 관리하는 데이터베이스 서버 티어(Database Server Tiers)로 구성되어 있다.

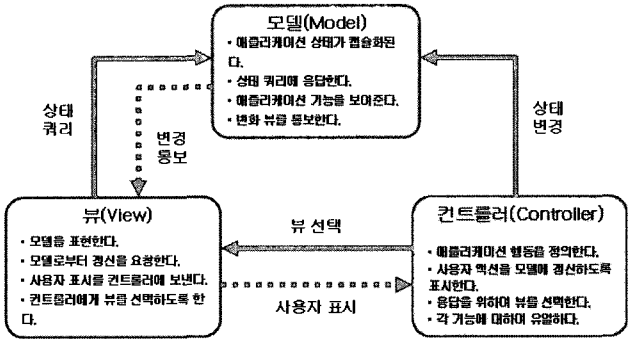
다음 <그림 4>는 본 연구에서 제안하는 X-인터넷 구조를 도식화한 것이다.



<그림 4> X-인터넷의 구조

4. 개발 방법의 설계

본 연구에서의 MBD(MVC-Based Development) 설계 모델의 구조는 <그림 5>와 같이 MVC의 디자인 패턴에 기반을 둔 비즈니스 통합(Business Integration) 설계 개발 방법론이다.



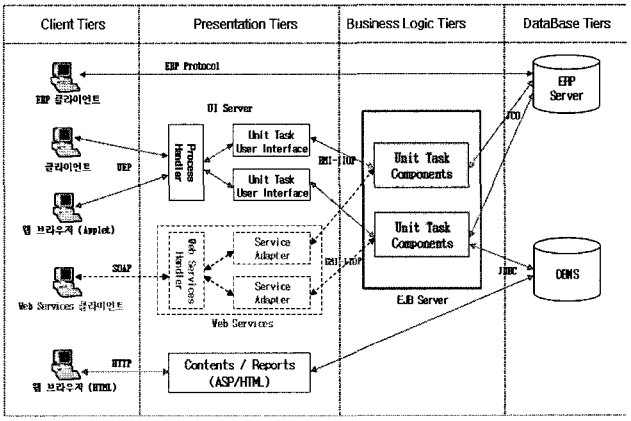
<그림 5> MBD 모델의 구조

지금까지 제안된 개발의 형태를 살펴보면, 각각의 단위 기능들을 나열하는 형태로 애플리케이션을 개발하였다. 지금도 대다수의 애플리케이션들은 이 같은 형태로 개발되고 있으며, 기반 기술이 OOP(Object-Oriented Programming) 기반이기는 하나 비즈니스와 프로세스의 변화에 유연하게 대처하는 데에는 한계를 가지고 있었다. MBD 방법론은 이 같은 한계를 극복하기 위해 비즈니스와 프로세스와 코드의 일치화를 목적으로 제안하는 방법론이다. 따라서 이 방법론에서의 핵심은 비즈니스의 프로세스를 조사하고 다시 그 프로세스를 구성하는 작업(Task)들을 잘 도출하여 구성하는 것이라 하겠다. 그리고 실제 이렇게 조사된 프로세스의 작업 단위별로 개발이 이루어지고 최종적으로 이 작업들의 조합이 애플리케이션 시스템이 되는 것이다.

4.1 MBD 방법론의 프레임웍

프레임웍은 문제해결 환경이라고 부르기도 하며, 공학적인 관점에서 응용 소프트웨어를 생성해 낼 수 있는 소프트웨어 체계라고 말할 수 있다. 따라서 프레임웍은 특정한 목적에 사용할 소프트웨어를 작성하기 위한 기반 구조 환경에서 각각의 해석, 설계, 시뮬레이션 코드들이 명확하게 정의되어 있고, 이들 간의 데이터 변환 및 전송이 가능하고, 각각의 체계들을 연결할 수 있는 인터페이스를 제공하는 것으로 볼 수 있다. 프로그램의 인터페이스를 위하여 원시코드를 이용할 수 있으나, 설계에서 원시코드가 주어지지 않는 경우가 대부분이며, 원시코드가 있다 하더라도 양이 방대하여 직접 작성한 코드가 아니면 이해하는데 많은 노력과 시간이 필요하다. 경우에 따라서는 새로 작성하는 것이 경제적일 수도 있다. 결국 실행코드를 잘 연계한 설계 환경을 조성해야 하며, 이러한 작업을 효율적으로 하고자 하는 노력이 필요한데 이것이 바로 프레임웍 구축이다[1].

다음 <그림 6>은 MBD 방법론의 프레임웍을 나타낸 것이다.



<그림 6> MBD 방법론의 프레임웍

그림에서 볼 수 있듯이 UI Server가 위치하는 표현 층(Presentation Tiers)에는 MBD 방법론의 최상위 레벨인 프로세스들을 관리하는 프로세스 핸들러(Process Handler)와 프로세스를 구성하는 단위 작업들의 단위 작업 사용자 인터페이스 객체(Unit Task UI Object)가 위치하게 된다. 그리고 각각의 단위 작업 사용자 인터페이스 객체에 상응하는 단위 작업 비즈니스 객체 컴포넌트(Unit Task Business Object Component)가 비즈니스 층(Business Tiers)을 구성한다. 이 층의 단위 작업 컴포넌트(Unit Task Component)는 EJB(Enterprise Java Beans) 컴포넌트로 구현되며, 기존과 마찬가지로 비즈니스 데이터(Business

Data)를 다루고 처리하는 역할을 수행한다. 이 단위 작업 컴포넌트들은 실제 데이터베이스 층(Database Tiers)에 데이터 객체(Data Object)와 통신하게 된다.

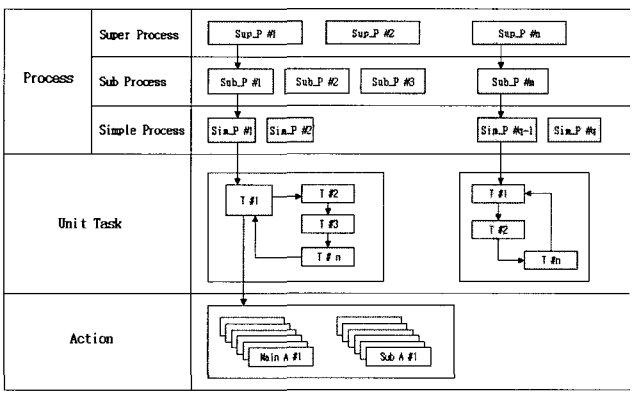
현업에서 기업의 비즈니스를 처리하기 위해 시스템을 개발 할 때 프로세스 중심에서 시스템을 기획하고 프로세스를 기반으로 개발하지만, 실제 관리하는 프로젝트는 프로세스를 구성하는 단위 작업 내에서 이루어져야 한다. 왜냐하면 단위 작업에서 실제 단위 업무에 대한 상세한 일들이 이루어지고 있기 때문이다.

본 연구에서 기존의 방법론과 MBD 방법론을 비교했을 때 시스템 구현과 관련하여 가장 크게 변경된 부분은 X-인터넷 환경인 표현층 부분이다. 이 층에서 프로세스를 정의하고 관리할 수 있는 프로세스 핸들러(Handler)를 추가하였고, 단위 작업 사용자 인터페이스 객체를 설계하였다. 기존의 방법론에서는 사용자 인터페이스(UI) 부분에 대해서는 특별한 객체를 설계하지 않고 UI 화면이 바로 UI 객체가 되었다. 따라서 한 화면 내에서 여러 개의 관련 단위 작업을 수행하게 함으로서 시스템을 복잡하게 만들고, 다른 시스템 또는 다른 프로세스에서 재사용하기 힘든 구조를 가지게 되었다.

MBD 방법론에서는 이 같은 문제를 해결하기 위해 단위 작업으로 UI를 설계하고, 향후의 웹 서비스(Web Services)에 대응하기 위해 표현 층의 단위 작업 UI 객체에 MVC라는 일반적인 디자인 패턴인 MVC 모델 개념을 도입하였다. 그러나 본 연구에서는 일반적인 MVC 모델을 그대로 사용한 것이 아니라 J2EE(Java 2 Enterprise Edition) 환경에 맞는 구조로 변경하여 설계토록 한다.

4.2 MBD 방법론의 논리적 구조

본 연구에서 제안한 MBD 방법론의 논리적인 구조는 다음 <그림 7>과 같다.



<그림 7> MBD 방법론의 논리적 구조

그림에서 볼 수 있듯이 최상위 수준에는 비즈니스에 필요한 여러 가지의 프로세스(Process)가 존재한다. 각각의 프로세스는 그 프로세스를 구성하는 단위 업무인 단위작업(Unit Task)들을 가지고 있으며, 각 단위 작업에서는 정형화된 활동(Action)들이 수행된다.

먼저 기업의 프로세스를 이해할 때에는 상위 프로세스(Super Process), 하위 프로세스(Sub Process), 단순 프로세스(Simple Process)의 단위를 정의할 수 있다. 여기에서 프로세스란 "어떤 목표를 달성하기 위해서 필요한 업무의 논리적인 순서"라고 정의할 수 있다. 사실 이렇게 정의를 내려보아도 프로세스의 개념을 이해하는 것은 어려울 수도 있다. 특히, 그냥 프로세스를 이해하기도 어려운데 프로세스를 수준 단위로 나누어 정의하는 것은 더욱 어렵다. 왜냐하면 프로세스는 보는 관점(View)에 따라 조금씩 다를 수 있고, 각 프로세스 단위별 경계선도 뚜렷하지 않기 때문이다.

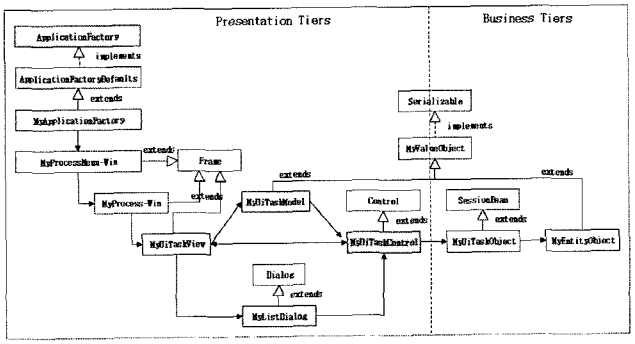
상위 프로세스(Super Process)는 그 기업의 가치를 창출하는 단위의 프로세스 레벨이다. 따라서 메가 프로세스는 전사적인 비즈니스 관점에서 바라보아야 하며, 대부분의 기업이 경영지원이나 연구개발과 같은 메가 프로세스는 서로 유사하며, 그 기업의 형태에 따라 고객가치를 창출하는 부분이 서로 상이할 것이다. 이렇게 정의된 상위 프로세스(Super Process)는 하위 프로세스(Sub Process)로 세분화 되고, 하위 프로세스는 다시 단순 프로세스(Simple Process)로 세분화된다. 여기서 어디까지가 상위이고 하위인지는 그다지 중요하지 않다. 이것은 단지 프로세스를 구성하는 분류의 정도로 인식하면 된다. 그 보다 중요한 것은 프로세스를 구성하는 단위 작업(Unit Task)을 잘 정의하는 것이다.

단위 작업을 정의해 보면 '어떤 프로세스 내에서 한 사람이 단위시간 내에 처리하는 가치를 가진 단위 업무'라고 정의할 수 있다. 일반적으로 제조 및 유통업에는 영업, 구매, 생산 등의 프로세스가 존재하게 된다. 그리고 영업이라는 프로세스에서 영업사원이 '주문처리'이라는 단일 업무를 완료하게 되면, A라는 기업에 고객 주문이라는 가치가 만들어 지게 된다. 그러므로 '주문처리'는 하나의 단위 작업이라고 할 수 있다. 그리고 단위 업무인 단위 작업을 처리하기 위해 행해지는 일련의 활동들 고객추가, 품목추가, 주문저장, 주문수정, 주문삭제 등을 실행(Action)라고 한다.

그러나 이론적인 정의만을 가지고 실제 프로세스를 정의하고 작업을 추출하는 것은 그리 쉽지 않은 작업이다. 따라서 현업의 실무에 정통한 관리자와 시스템 개발 및 기획을 하는 팀장 및 프로젝트 매니저는 프로세스에 대해 잘 이해하고 있어야 하며 단위 작업(Unit Task)을 잘 도출해 주는 작업을 해줘야 한다.

4.3 MBD 방법론의 물리적 구조

다음 <그림 8>은 특정 프로세스의 특정 단위 작업을 추출하여 시스템을 구성했을 때, MBD의 물리적인 애플리케이션 구조를 표현한 것이다.



<그림 8> MBD 방법론의 물리적 구조

애플리케이션 구조의 가장 상위에는 프로세스를 핸들링하는 Process Menu와 Process Window(MyProcess Menu-Win)가 존재한다. Process Window에 구성된 단위 작업이 표현 층에 MVC 구조로 위치하게 된다. 비즈니스 층에는 기존의 방법론에서 마찬가지로 엔티티 객체(MyEntity-Object)와 단위 작업 객체(MyUiTask Object)의 비즈니스 객체로 구성되는데, 여기에 MVC 도입과 함께 새로이 가치 객체(MyValueObject)라는 것을 추가하였다. 이러한 단위 작업의 비즈니스 객체들은 데이터베이스 티어(Database Tiers)의 ERP 데이터베이스 서버나 이 기종 데이터 서버와 통신을 하는 구조를 가지고 있다.

5. 결 론

MBD 개발 방법론은 개발 단위가 단위 작업(Unit Task)이라는 단위로 세분화되기 때문에 개발 일정이나 역할 분담이 기존 개발 방법보다 수월해 질뿐만 아니라, 표준화된 단위 작업의 개발은 실제 개발 기간의 단축과 시스템 소프트웨어 품질의 향상을 가져온다. 또한 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 있다. 그리고 이 개발 방법론은 비즈니스 프로세스와 시스템을 동기화시킨 형태이기 때문에 사용자들의 교육을 최소화 할 수 있고, 프로세스가 변경될 시에는 변경되는 부분의 단위 작업만 변경하면 되므로 빈번한 프로세스의 변화에 유연하게 대처할 수 있는 장점이 있다. 이러한 개발과 비즈니스에 요구 이외에도 웹 서비스에 대비하여 단위 작업별로 가치 컴포넌트(Value Component)를 두어 자연스럽게 확장할 수도 있다. 향후 MBD 방법론에 대한 상세 설계를 통한 실무 사례 연구와 웹 서비스 기반의 e-Commerce 및 p-Commerce 구축에 대한 연구가 지속되어야 할 것이다.

참고문헌

[1] 이명호, "N-티어 분산 환경에서 e-Commerce 설계 및 구현," 한국산업경영시스템학회 춘계학술대회, 2003.

[2] Boehm, B., "A Spiral Model for Software Development and Enhancement," IEEE Computers, Vol. 21. pp. 61~72. 1988.

[3] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., Stal, M., "Pattern-Oriented Software Architecture, Volume 1 : A System of Patterns," John Wiley&Sons, 1996.

[4] 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.

[5] Pressman, R., "Software Engineering : A Prac- titioner's Approach," 5th Ed., New York : McGraw-Hill, Inc., New York, pp. 27. 2000.

[6] Rakitin, S., "Software Verification and Validation : A Practitioner's Guide," Artech House, pp. 13-27. 1997.

[7] Riehle, D., Zulighoven, H., "Understanding and Using Patterns in Software Development," Theory and Practice of Object-Oriented Systems, 2(1), 1996.

[8] Withey, J., "Implementing Model Based Software Engineering in Your Organization : An Approach to Domain Engineering" CMU/ SEI-94-TR-01, Carnegie-Mellon University, Pittsburgh:SEI, 1994.

[9] http://www.forrester.com/ER/Marketing/1,1503,214, FF .html