DOI: 10.3745/KIPSTD.2008.15-D.2.187

MDA기반 학사관리 프로세스 유효성 분석

윤 정 모[†] · 김 치 호[††]

요 약

새로운 패러다임의 시스템 개발 접근 방법은 OMG(Object Management Group)에서 MDA(Model Driven Architecture)로 표준화하고 있다. MDA의 핵심 기술은 모델 중심의 시스템 구현을 위한 기술 구조를 정의하는 것으로 시스템의 설계 및 명세에 대한 구현 기술을 플랫폼 독립적 모델(PIM : Platform Independent Model)로 설계한 후, 구현 환경에 적합한 플랫폼 종속적 모델(PSM : Platform Specific Model)로 설계하여 구현환경에 맞는 언어로 변환하는 기술이다. MDA의 핵심인 MOF(Meta-Object Facility), UML, XMI(XML Metadata Interchange), CWM(Common Warehouse Metamodel) 표준 등이 포함된다. MDA 지원 자동화 툴들은 위의 정보들을 갖고 변환 작업을 수행한 후 실행 가능한 시스템으로 생산해 내는 것이다.[1]

본 논문에서는 MDA기반 접근 방법을 적용하여 학사관리 시스템을 설계 및 구현하고, MDA기반 접근 방법에 대한 모델링의 중요성을 강조하였다. 또한 소프트웨어를 효율적으로 개발하는 방법을 제시하여 플랫폼 독립적 모델(PIM)로 작성하고, 플랫폼 종속적 모델(PSM)을 J2EE 플랫폼 기반의 EJB(Enterprise Java Beans)로 변환하는 과정 등을 제시하였다. MDA를 지원하는 자동화 툴 및 편집기(Together Architect 2006 for Eclipse, Edit plus 2)와 데이터베이스 모델링 툴(ER/WIN 4.1)을 사용하여 시스템 설계 및 구현을 하였으며, 전통적 개발 프로세스와 MDA 기반 개발 프로세스에 대한 유효성 분석 결과를 제시하였다.

키워드 : 플랫폼 독립적 모델, 플랫폼 종속적 모델

An Efficiency Analysis of Management System for Academy Affairs Process Based on MDA

Kim, Chiho[†] · Yoon, Jungmo[††]

ABSTRACT

The system development approach method of the new paradigm, as being standardizing MDA(Model Driven Architecture) in OMG(Object Management Group), the core technique of MDA definite technique structure for system materialization focusing on Model, is to build the design and the statement for system in PIM(Platform Independent Model), Materialization technique, and to build PSM(Platform Specific Model) adapt to materialization environment, and then to be the technique transforming into language Platform suitable to materialization environment.

It includes MOF(Meta-Object Facility), UML, XMI(XML Metadata Interchange), CWM(Common Warehouse Metamodel), the core of MDA. Though these operations MDA support automatic tools product the practicable system after carrying out transform operation with the above information.

In this thesis, it will be approached how to design and materialize the Bachelor management system based on MDA and the importance of modeling should be emphasized by applying to the approach method based on MDA. It should be suggested how to develop software efficiently, written it out in PIM, and suggested the process transforming PSM into EJB by J2EE Platform. The system is designed and implemented using automatic tool, edit machine(Together Architect 2006 for Eclipse, Edit plus2) supporting MDA and Database Modeling tool(ER/WIN 4.1).

In conclusion, it should be suggested the efficiency analysis result for development process of traditional and based on MDA.

Key Words : MDA(Model Driven Architecture), EJB(Enterprise Java Beans), PIM(Platform Independent Model), PSM(Platform Specific Model)

† 종신회원 : 서울산업대학교 컴퓨터공학과 교수
†† 정 회 원 : 서울특별시립 상계직업전문학교 웹프로그래밍과 교사
논문접수 : 2006년 8월 23일, 심사완료 : 2007년 1월 18일

1. 서 론

소프트웨어 개발할 때 가장 먼저 고려하게 되는 것은, 바로 다양하게 많은 종류의 플랫폼 중에서 어떤 플랫폼을 선택 하는냐가 중요하다. 어떤 플랫폼이 비용이 적게 들고, 개발 기간이 단축되고, 투자한 것에 비해서 얼마나 많은 유효성을 가질 수 있는가 등의 고려할 사항이 많다.

이와 같이 통합하고 하나의 표준으로 만들어서 하나의 프로그래밍 언어, 하나의 미들웨어, 하나의 네트워크 프로토콜, 하나의 운영체제는 결국 각각 개발한 벤더들이 합의하여 서로 연동을 가능하게 하는 것이다.

MDA 접근방법은 산재한 플랫폼 독립적 모델인 PIM을 UML로 작성한 후, 이를 구현 환경과 관련한 플랫폼 종속적 모델인 PSM을 만들고, 플렛폼 종속적 모델을 통해 소스코드, 설정파일 및 기타 산출물을 자동으로 생산해 냄으로써 시스템을 보다 효율적으로 유지 및 통합 할 수 있다.

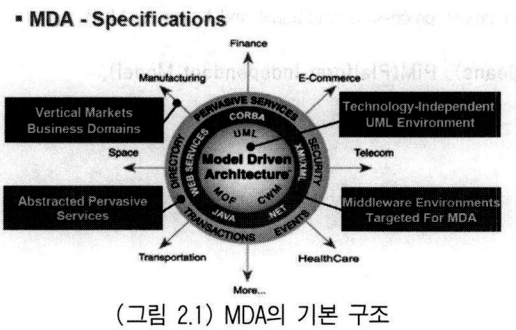
MDA가 성공적으로 이루어지기 위해서는 MDA 지원 도구에서 플렛폼 독립적 모델에 플랫폼 종속적인 정보를 제외한 모든 정보를 모델링 할 수 있어야 하고, 플렛폼 독립적 모델의 완전성을 검증 할 수 있어야 한다. 또한 자동화된 매핑 기준을 제공하여 플렛폼 독립적 모델을 각종 플랫폼에 종속적인 PSM으로 변환할 수 있어야 하고 설계 정보로부터 자동적으로 소스코드를 생성할 수 있어야 한다.

플렛폼 독립적 모델의 효과적인 재사용을 위해서는 도메인 아키텍처 재사용의 기능도 지원해야 한다. 따라서, MDA 기반 접근 방법으로 학사관리 시스템을 개발 사례로 해서 전통적 개발 프로세스와 MDA기반 개발 프로세스에 대한 유효성 분석을 제시하였다.

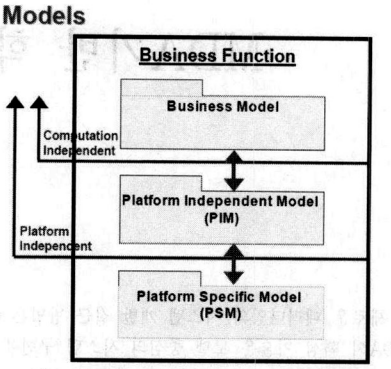
2. 관련 연구

2.1 MDA(Model Driven Architecture)

2001년 9월 OMG는 네트워크 기반 환경의 소프트웨어 개발 시 다양한 종류의 구현 언어, 미들웨어, 네트워크 프로토콜 등 어떠한 것을 선택하여 개발하느냐는 문제점을 해결하고자 그 해 9월에 아키텍처의 표준 모델인 MDA를 제안하였으며, MDA의 핵심은 플랫폼에 독립적인 모델을 구현 환경에 적합한 플랫폼 종속적 모델로 자동 변환 가능한 구조를 정의한 것이다.[1,2,3,4]



(그림 2.1) MDA의 기본 구조



(그림 2.2) MDA의 모델

(그림 2.1)은 MDA의 기본 구조를 나타낸 것이며, (그림 2.2)는 MDA의 모델을 나타낸 것이다.

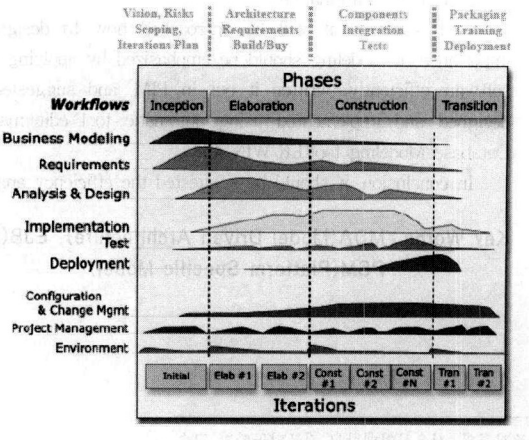
플랫폼에 독립적인 모델을 설계하고, 이를 자동화 도구를 이용하여 플랫폼에 맞는 종속적인 모델 및 시스템으로 개발함으로써 생산성 향상이 구현 단계에서 가능하게 하고, 메타모델에 기반을 둔 시스템들의 일반적인 호환성을 기대할 수 있게 해주는 장점을 가지게 된다.

MDA는 CORBA와 같이 설계 수준의 표준이다.[5]

MDA는 오래 전부터 소프트웨어 공학도들에 의해 만들어지고 정제되어 왔으며, 대부분의 소프트웨어 통합(SI) 프로젝트에서 적용되고 있는 '비즈니스 모델링 → 요구사항 분석 → 설계 → 구현' 방식의 톱 다운(Top Down) 모델의 개념을 갖는다.

(그림 2.3)과 같이 대표적인 객체지향 방법론인 UP (Unified Process)에서는 비즈니스 모델링을 통해 전체 비즈니스를 파악하고, 유스케이스 다이어그램을 통해 요구사항을 모델링하며 이것으로 분석 모델을 만든다. 이때의 분석 모델을 기반으로 적절한 플랫폼과 설계 메커니즘을 적용해서 플랫폼 종속적인 설계 모델을 만들고 이를 기반으로 구현한다.

요구사항 모델은 시스템을 개발자 관점으로 가져오기 전에 고객과 의사소통하기 위한 수단이고, 분석 모델에서는 어떤 환경에서 어떤 제약 조건을 갖고 구현될지는 생각



(그림 2.3) Unified Process 단계

하지 않고 순전히 비즈니스만을 반영한 모델을 모델링 한다. 그리고 설계 모델에서는 비기능적 요구사항을 반영해서 세부 설계를 하고 이 모델을 이용하여 구현 소스 코드를 만든다.

MDA는 UML을 통해서 시스템을 모델의 형태로 설계 및 명세 하는 것과 시스템의 개발 라이프 사이클 전체에서 모델 역할에 대한 것이다. MDA 개발 방법은 우선 플랫폼 독립적 모델을 정의함으로 시작한다.

플랫폼 독립적 모델은 특정한 기술 플랫폼이나 기반 기술에 독립적인 방법으로 시스템을 설계한 모델을 말한다. 플랫폼 독립적 모델은 단일 플랫폼에 매여 있지 않고 프로그래밍 언어에 독립적이기 때문에 다른 시스템으로의 가교 역할을 할 수 있다. MDA에서는 이러한 플랫폼 독립적 모델에 정형화된 변환 법칙을 사용해서 플랫폼 종속적 모델을 생성한다.

플랫폼 종속적 모델은 특정 기술에 종속적인 요구사항 들이 포함된 시스템 모델 정보이다. 여기서 플랫폼 독립적 모델과 플랫폼 종속적 모델 모두 UML로 기술된다. 마지막으로 플랫폼 종속적 모델을 통해서 실제로 동작하는 구현 코드를 만들어 낸다.

MDA 방식으로 개발된 시스템은 다음과 같은 이점을 지니게 된다.[5]

(1) 기술 변화 상황에 효율적으로 대처할 수 있다. MDA 방식으로 개발된 시스템은 플랫폼 독립적 모델을 통해서 변경된 기술 플랫폼으로 이식이 쉽게 이루어질 수 있기 때문이다.
(2) 시스템 인프라 변화에 유연하게 대처할 수 있다.
(3) MDA 방식으로 개발된 시스템은 그 시스템의 유지보수 비용이 적으며 시스템의 수명이 길다. 따라서 투자 비용이 보존된다

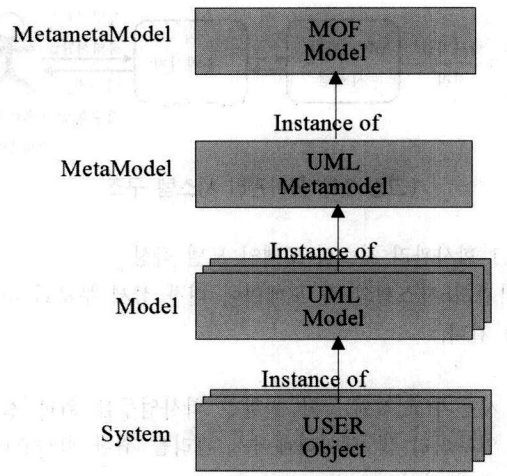
2.2 4계층 메타 모델링 구조

2000년도에 접어들면서 UML을 보다 체계적으로 변화시키지 않고서는 새로운 개발 환경, 새로운 개발 접근 방법, 새로운 도구와의 호환성 등을 UML에 적용할 수 없다는 것이 명백해졌다. 그 결과 보다 근본적인 개선 작업이 UML에 이루어졌으며 2003년도에 UML2.0이 발표되었다.[3]

(그림 2.4)는 MOF(Meta-Object Facility)와 UML 모델들 간의 관계를 나타낸다. 이러한 4계층 메타모델링 구조는 MDA기반 시스템 모델링의 기본 구조이다. MDA에서 UML 2.0은 구현 기술에 플랫폼 독립적인 모델이나 구현 환경에 적합한 플랫폼 종속적인 모델 작성시에 사용되기도 하며, MOF나 CWM(Common Warehouse Metamodel)을 표현하기 위한 기호로서의 기반을 제공하기도 한다.

MDA 기반 시스템 개발의 첫 번째 단계가 UML 2.0을 이용하여 구현하고 독립적 모델을 작성하는 것이다.[8,9,10]

UML 2.0을 모델링 영역의 경계를 확립하는 프로파일(profile)은 누구나 정의할 수 있으나, CORBA나 EDOC



(그림 2.4) 4계층 메타 모델링 구조

(Enterprise Distributed Object Computing) 등 OMG의 표준 프로파일과 같이 다수의 사람들이나 기업에 의해 정의된 프로파일이 가장 유용하게 사용된다. 이렇게 표준화된 프로파일은 구현 독립 모델로부터 구현 종속 모델을 생성하거나, 구현 종속 모델로부터 코드를 생성하기 위한 규칙들을 제공하게 된다.

3. MDA를 적용한 학사관리 시스템 설계 및 구현

3.1 시스템 개요

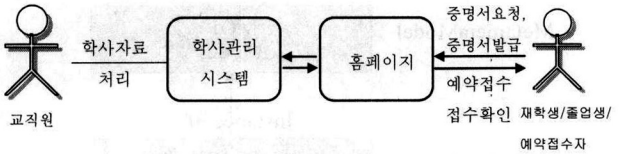
MDA를 적용하여 학사관리 시스템을 구현하는 과정에 대하여 설명하고, 이 시스템 개발 과정을 통하여 요구사항을 분석하고, 플랫폼 독립적 모델을 구축하여 플랫폼 종속적 모델 과정에서 J2EE 기반으로 EJB를 구현하게 된다. MDA 기술을 지원하는 자동화 도구(Together Architect 2006)와 데이터베이스 모델링 도구(ER/WIN 4.1)를 이용하여 소스코드를 생성한 애플리케이션을 구현한다.

3.2 도메인 모델(CIM : Computation Independent Model) 설계

CIM은 시스템 요구사항과 환경에 대한 것을 기술하는 모델로써 도메인 모델, 비즈니스 모델이라고 할 수 있다. 본 논문에서는 MDA를 적용한 학사관리 시스템을 설계 구현한다. 이 시스템은 3 Tier 구조를 갖고 있으며 교직원 누구나 쉽게 접속해서 학사 업무를 할 수 있도록 하고, 클라이언트는 웹 기반의 JSP 클라이언트가 필요하다.

재학생 및 졸업생, 예약 접수자는 홈페이지를 통해서 증명서 발급 및 예약 접수를 할 수 있도록 하고, 학사업무를 관리하기 위한 독립적인 데이터베이스를 사용하며, 미들웨어는 학사업무를 수행하기 위한 비즈니스 로직을 구현한다.

전체적인 시스템 개요는 (그림 3.1)과 같으며, 시스템 사용자는 학사관리 시스템으로부터 보안인증, 기초자료, 재료수불, 예약접수, 입학지원, 학적 관리, 교육계획, 교육실적, 통계자료, 전자결제, 증명서발급 및 홈페이지와 연동하여 처리할 수 있도록 지원한다.



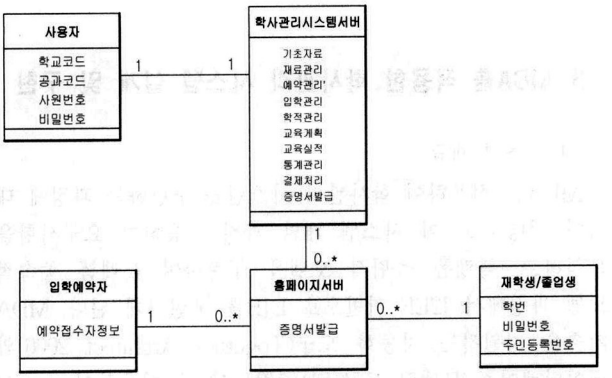
(그림 3.1) 학사관리 시스템 구조

3.2.1 학사관리 시스템 도메인 모델 작성

학사관리 시스템의 각 도메인에 대한 상세 부분의 정의는 다음과 같다.

(1) 사용자(교.직원)들의 원활한 학사업무를 위한 실시간 업무처리 및 데이터베이스 관리를 위한 학사관리 시스템 서버
(2) 재학생 및 졸업생들에 대한 증명서 발급을 위한 홈페이지 서버
(3) 예약 접수자 들에 대해 학사관리 시스템에 직접 접근이 불가능함으로 인해 웹 서버(홈페이지)를 통한 예약접수

(그림 3.2)는 학사관리 시스템의 도메인 모델을 설계 하였다.



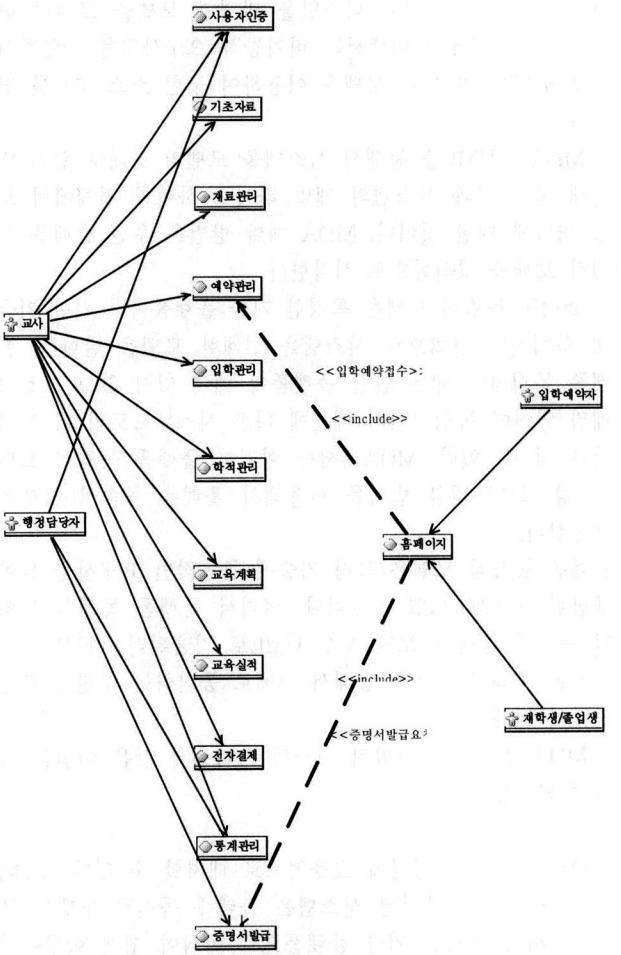
(그림 3.2) 학사관리 시스템 도메인 모델

3.2.2 학사관리 시스템 유스케이스 다이어그램

학사관리 시스템을 설계하기 전에 사용자로부터 요구사항을 명확하게 정의하고 분석하여 향후 개발을 진행하기 위해 유스케이스 다이어그램을 통해서 도식화 한다. (그림 3.3)은 학사관리 시스템에 대해 전체적인 유스케이스 다이어그램을 표현하였다.

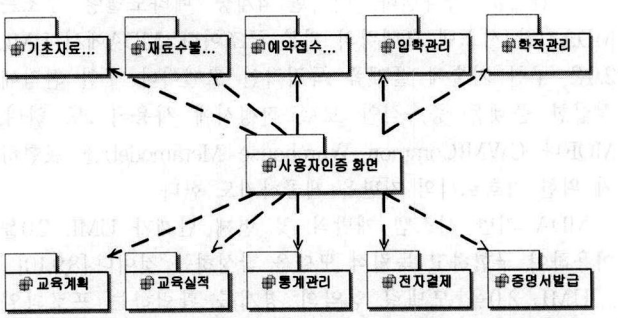
학사관리 시스템의 전체 패키지 구성을 살펴보면 크게 사용자 인터페이스와 비즈니스 부분으로 나누어 관리한다. 첫 번째 사용자인터페이스에는 사용자에게 사용자 인터페이스 제공 및 자료 입력 받는 패키지들로 구성되어 있고, 두 번째 비즈니스에는 여러 개의 패키지에서 공통으로 사용되고 있는 재사용 가능한 비즈니스 로직으로 구성되어 있으며, 데이터베이스 접근은 사용자 인터페이스에 입력된 정보를 비즈니스 로직에서

받아 스토어드프로시저의 파라미터에 값을 넘겨주고, 스토어드프로시저를 실행해 줌으로서 처리한다.



(그림 3.3) 학사관리 시스템 유스케이스 다이어그램

사용자 인터페이스 레이어는 학사관리 시스템이 사용자에게 제공하는 주요 기능들이 포함되었고, 이 시스템의 사용자는 교.직원(교사, 행정담당자)만 자신의 계정으로 로그인하여 접근이 가능하고, 재학생, 졸업생, 입학예약자들은 홈페이지를 통해서 데이터베이스에만 접근이 가능하도록 되어 있고, 비즈니스 로직 레이어는 각각의 패키지에서 공통으로 사용되고 있는 재사용이 가능한 비즈니스 로직과 데이터베이스에 정보를 제공하는 컨트롤러 기능으로 시스템에서 필요한 정보를 저장하는 기능들로 구성 하였다. (그림 3.4)는 사용자 인터페이스 레이어에 포함된 패키지들의 관계를 표현 하였다.



(그림 3.4) 사용자 인터페이스 레이어 패키지의 구성

3.3 플랫폼 독립적 모델 설계

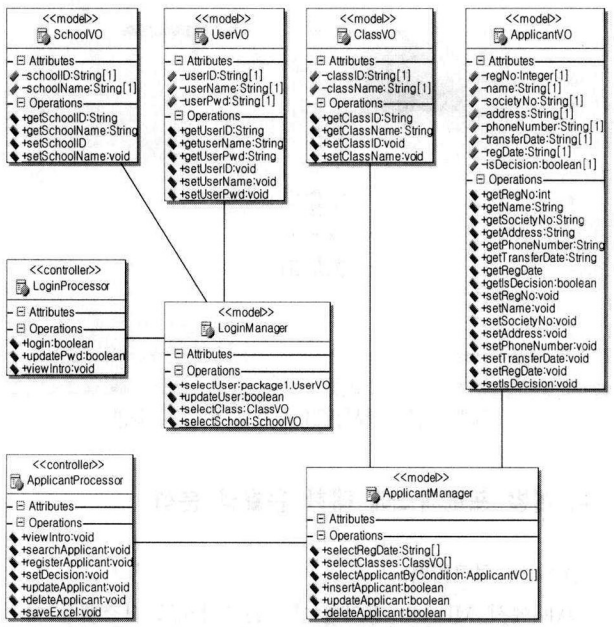
MDA를 적용한 개발 프로젝트의 시작은 플랫폼 독립적 모델을 구축 하면서 부터이다. 요구사항을 분석하고 구현 플랫폼 환경은 고려되지 않은 모델링으로써 순수한 플랫폼 독립적 모델이며, UML 모델링 도구인 Together Architecture 2006을 사용하여 시스템을 모델링한다.

3.3.1 플랫폼 독립적 모델 학사관리 시스템 설계

클래스 다이어그램 표현은 MVC(Model View Control) 기반으로 클래스의 역할을 보다 쉽게 구분하고자 표현하였다. 클래스 위에 확장 메커니즘인 스테레오 타입 등을 사용하여 표현을 상세하게 설계 하였다. (그림 3.5)는 클래스 다이어그램으로 학사관리 시스템 중 사용자 로그인과 입학관리 클래스만 나타내었다.

〈표 3.1〉 각 클래스 설명

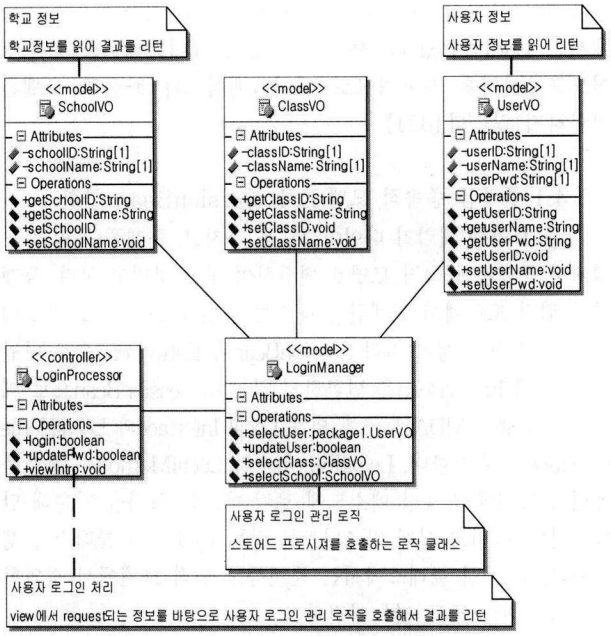
클래스명	클래스 내용
SchoolVO	학교정보 관리(Value Object)
ClassVO	공과정보 관리(Value Object)
LoginProcessor	사용자 로그인 처리
UserVO	사용자(교.직원)정보 관리(Value Object)
LoginManager	사용자 로그인 관리 로직 (스토어드프로시저를 호출하는 로직 클래스)
ApplicantVO	입학지원자 정보 관리(Value Object)
ApplicantProcessor	지원자 정보 처리



(그림 3.5) 학사관리 시스템[사용자 로그인 및 입학관리]의 클래스 다이어그램

3.3.2 플랫폼 독립적 모델 - 사용자인증 패키지 설계

(그림 3.6)은 사용자인증 패키지로서, LoginProcessor는 사용자에게 사용자 인터페이스에서 요구되는 정보를 바탕으로 사용자 로그인 관리 로직을 호출해서 결과를 제공하고, LoginManager는 로그인한 사용자 체크 및 스토어드프로시

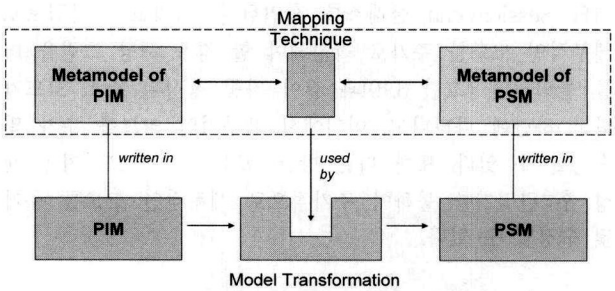


(그림 3.6) 사용자 인증 패키지

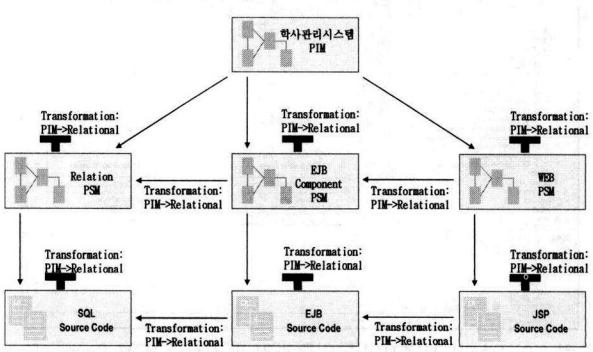
저를 호출하고 프로시저에서 작업한 결과를 받아 SchoolVO, ClassVO, UserVO에 대하여 Value Object를 생성한다.

3.4 플랫폼 종속적 모델 구현

플랫폼 독립적 모델로 설계한 학사관리 시스템을 플랫폼 종속적 모델로 매핑 룰을 통하여 3 Tier로 구성하여 변환한다. 각 Tier는 서로 다른 플랫폼을 갖게 된다. (그림 3.7)은 MDA의 모델 변환 개념도를 나타내었으며, (그림 3.8)은 플랫폼 독립적 모델을 가지고 플랫폼 종속적 모델로 자동 변



(그림 3.7) MDA의 모델 변환



(그림 3.8) 학사관리 시스템 변환 모델

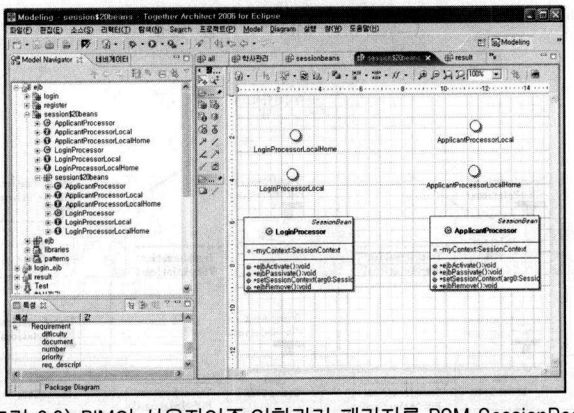
환을 통해서 3 Tier 구조로 나타내고 있다. 학사관리 시스템에서는 Entity Bean을 사용하지 않고 데이터베이스의 스토어드프로시저를 사용하므로 데이터베이스와 관련한 모델은 변환하지 않는다.[8,11]

3.4.1 플랫폼 종속적 모델 EJB SessionBean 구현

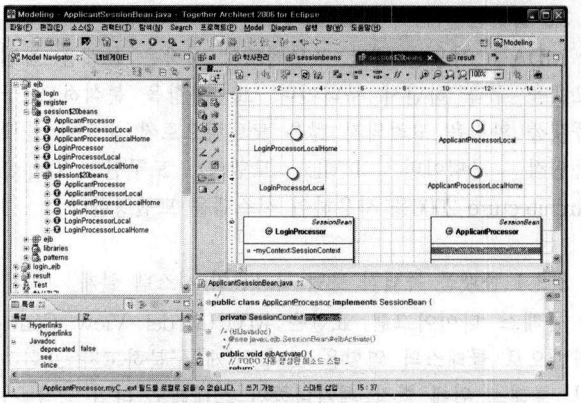
사용자 인증 패키지 다이어그램을 가지고, 플렛폼 독립적 모델에서 플렛폼 종속적 모델로 변환하여 구현 하려고 한다. 플렛폼 독립적 모델에서 설계한 클래스는 플렛폼 종속적 모델로 매핑 되면서 그 속성에 따라 SessionBean과 EntityBean으로 나뉜다. 여기서 EntityBean은 변환하지 않는다. SessionBean으로 매핑되는 모델은 MDA 도구에 의해 Local Interface와 LocalHome Interface를 추출한다. LocalInterface의 LocalMethod는 플렛폼 독립적 모델에서 특정 메소드에 로컬 메소드 성격을 기술해 만들어진다. 그리고 실제 비즈니스를 갖는 EJB 구현 클래스가 생성된다. 이 구현 클래스에서는 플렛폼 독립적 모델에서 정의한 비즈니스 메소드도 만들어진다.

EJB 플랫폼에 대한 플렛폼 종속적 모델은 UML 프로파일(UML Profile for EJB)에 정의된 표준 EJB 표현 형식으로 기술되고, 이후 소스코드로 변화되기 위한 중간 단계인 플렛폼 종속적 모델을 완성하게 된다. 또한 추가적으로 추가해야 하는 소스코드는 편집기를 활용하여 소스코드를 추가하면 된다.

본 논문은 플렛폼 독립적 모델에서 미들웨어를 EJB로 선정해서 자동화 툴인 Together Architecture 2006을 이용하여 플렛폼 종속적 모델을 추출했다. 사용자 인증 및 입학관리 패키지의 각각 한 개의 클래스 LoginProcessor 클래스와 ApplicantProcessor 클래스를 플렛폼 종속적 모델로 매핑 하면서 SessionBean으로 변환된 것을 볼 수 있다. (그림 3.9)는 플렛폼 독립적 모델을 플렛폼 종속적 모델로 매핑된 두 개의 SessionBean 클래스를 확인한다. 그리고 추가적으로 세부적인 코드를 추가로 작성해야 할 경우 해당 그림을 더블 클릭하면 (그림 3.10)과 같이 자동 생성된 소스 코드가 화면 하단에 나타나고, 여기에서 세부적인 코드를 추가 및 수정할 수 있다. 또한 다른 방안으로는 소스 코드 자동 생성 후 편집기를 통하여 추가적으로 세부적인 코드를 추가 및 수정할 수 있다.



(그림 3.9) PIM의 사용자인증,입학관리 패키지를 PSM SessionBean으로 변환

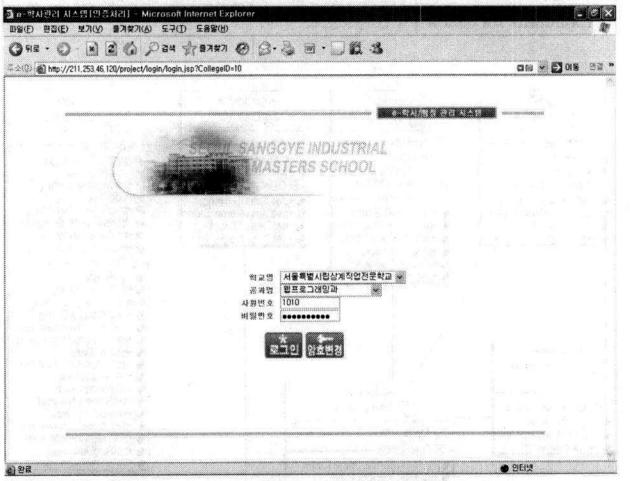


(그림 3.10) PSM SessionBean으로 변환 후 상세코드 추가작성 및 수정

3.5 학사관리 시스템 결과 화면

J2EE 플랫폼의 EJB로 구현한 학사관리 시스템 결과 화면을 간단하게 보여준다.

(그림 3.11)은 학사관리 시스템에 대한 사용자 인증처리 화면이다. 교직원(교사와 행정직원)일 경우 로그인이 가능하고, 그 외에는 접근을 불허한다. 학교, 학과, 직원번호와 비밀번호를 입력하고 로그인 버튼을 클릭함으로 인증처리를 한다.



(그림 3.11) 학사관리 시스템 로그인 화면

4. 개발 프로세스에 대한 유효성 분석

4.1 MDA 접근방법

OMG에서 MDA를 정의한 내용은 "대규모 시스템을 아키텍처 기반으로 개발하는데, 모델링 기법을 사용하고 각각의 모델의 변환을 통해 시스템을 구축하며 이러한 모든 활동은 자동화된 도구의 도움을 받는다는 것이다." 라고 정의되어 있다.[3]

첫 번째, 모델링을 이용한 소프트웨어 개발은 고객의 요구사항을 초기 모델을 만들고, 이로부터 각종 정보를 첨가하면서 모델을 변경시켜 최종적으로 우리가 필요한 소스코

드를 생성함으로써 실제적으로 실행 가능한 시스템을 구축하는 것이다. 즉 개발의 모든 활동은 모델을 만들고 이 모델에 각종 정보를 첨가하고 보충하여 더욱 세밀한 모델로 변환시켜서 작동 가능한 소스코드를 생성하는 것이다.

두 번째, 각 변환에서 사용되는 도메인 정보나 각종 경험적인 지식 및 플랫폼에 관한 정보들이 축적되는 과정에서 이러한 지식들이 정형화됨으로써 표준화된 지식 패턴이나 아키텍처의 유도를 촉진하게 된다. 초기에 GoF 패턴 밖에 없었던 패턴들이 현재는 많은 수의 패턴들이 발표되고 있으며, 재사용의 중요성이 소프트웨어 개발 생산성에 부각 되었다.

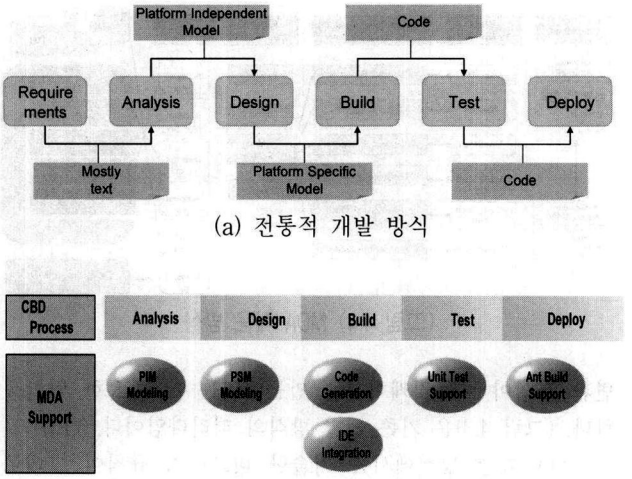
세 번째, 잘 통합된 개발 도구의 도움을 받을 수 있게 된다. 완전히 통합된 도구에 대한 논의를 학술적으로 시작한 것은 십 년 이상 진행되어 왔지만, 실제적인 개발에 적용하지 못한 이유는 통합에 대한 가치를 사용자들이 충분히 수용하지 못했기 때문이다. 그러나 MDA가 지원되면 요구사항으로부터 테스트까지 통합 개발 환경에 대한 요구가 강하게 요청될 것이며, 이러한 요건을 만족 시키는 도구들이 MDA 도구의 주요 부분을 차지하게 될 것이다.

마지막으로, 충분히 자동화된 도구가 지원된다면, 비즈니스 모델링만으로도 실제적으로 작동하는 시스템을 구축할 수 있다는 것이다. 즉 IT 전문가가 아닌 도메인 전문가가 비즈니스에 대한 정책을 모델링 하면 바로 작동하는 제품을 구축할 수 있게 된다. 빠른 시간 안에 비즈니스의 요구사항을 정확히 운용 환경으로 적용시키는 체제가 완성되는 것이다. 금융 상품을 빨리 시장에 출시하기 위해서는 이를 지원하는 IT 서비스도 역시 빨리 지원돼야 가능하기 때문이다 [8,12].

4.2 MDA 적용 방식과 전통적 개발 방식 비교분석

MDA 개발 방식과 기존의 개발 방식은 차이가 있다. 전통적인 방법은 최종적으로 개발자가 효과적으로 하나의 거대한 시스템을 직접 손으로 구현해야 한다는 것에 초점이 맞춰져 설계되어 있다. MDA 개발 프로세스 모델은 요구사항을 정확히 분석하고, 비즈니스를 세밀하게 모델링 하여 플랫폼 독립적 모델을 구축하도록 하고 있다. 이 플랫폼 독립적 모델은 플랫폼 종속적 모델로 변환하기 위한 표준 지침인 QVT(Quality View Transformation)를 기술하고, 이 정보들을 중심으로 플랫폼 종속적 모델을 추출하고 소스코드를 추출하면서 플랫폼 아키텍처 모델, 테스트 코드, 구현 소스코드 등이 작성 된다. (그림 4.1)은 전통적 개발 방식(a)과 MDA 적용 방식(b)을 확인할 수 있다.

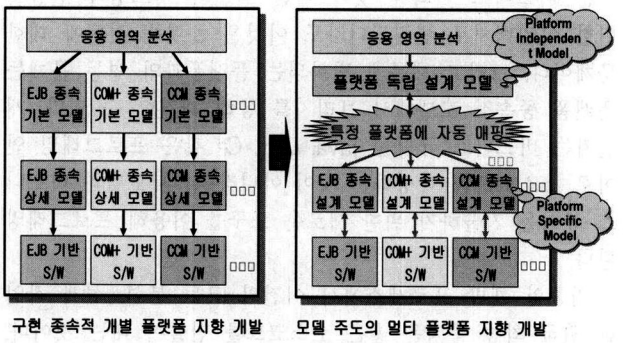
기존에는 대략 전체적인 개념만을 작성하고 필요에 따라 세부 설계를 하여 개발하였으며, 한번 작성한 모델이 구현 단계에서 변화하는 경우가 빈번했다. 그러나 MDA 개발 프로세스에서는 플랫폼 독립적 모델 구축 시 모든 비즈니스에 대한 정적 구조 및 동적인 구조를 완벽하게 기술해야 완전한 애플리케이션을 구축할 수 있다. 결국 MDA 개발 프로세스로 인해 구축돼야 할 핵심이 변화되고, 이에 따라 개발



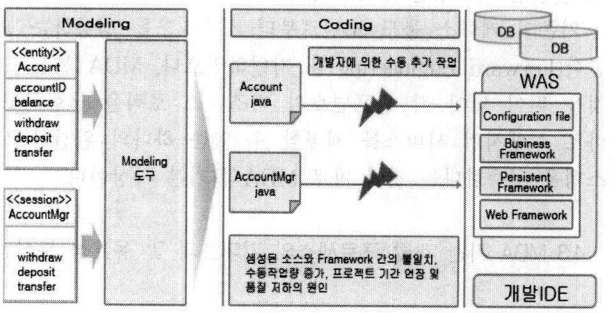
(a) 전통적 개발 방식

(b) MDA 적용 방식

(그림 4.1) 전통적 개발 방식(a)과 MDA 적용 방식(b)



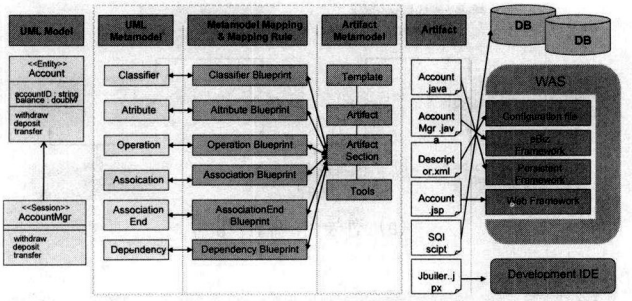
(그림 4.2) 단일 플랫폼 지향 개발 절차와 다중 플랫폼 지향 개발 방식



(그림 4.3) 기존 개발 방식

자의 역할도 각 핵심을 실현하는 형태로 변화해야 한다. (그림 4.2)는 단일 플랫폼 지향 개발 방식과 다중 플랫폼 지향 개발 방식의 흐름을 보여 준다.[6]

MDA는 기존 개발 프로세스와 확실히 다른 소프트웨어 개발 패러다임을 갖고 있다. 기존 방법은 분석, 설계 작업을 통해 나온 모델을 개발자가 직접 구현하는데 반해, MDA는 개발 작업의 중심이 플랫폼 독립적인 모델(PIM)을 구축하는 것에 맞춰져 있고, 이후에는 MDA 지원 툴을 사용해 변환 작업을 하여 최종 소스코드, 애플리케이션을 만들어 내는 것이다. MDA 중심에는 플랫폼 독립적 모델이 있고, 툴과



(그림 4.4) MDA 적용 방식

변환 작업이 애플리케이션을 만들어내는 데 중요한 역할을 한다. (그림 4.3)은 기존 개발 방식의 패러다임이다.[6,10]

MDA 개발 방식에서는 기술한 비즈니스 규칙이 그대로 마지막 소스코드로 구현되기 때문에 상당히 주의해야 한다. 플랫폼 독립적 모델은 시간이 아무리 지나도 비즈니스 규칙이 바뀌지 않는 이상 변경되지 않는다. 이후에 시스템이 새로운 비즈니스 기회를 맞아 다른 외부의 시스템과 연동을 위한 아키텍처로 변경되더라도 이것은 플랫폼 선택에 대한 문제이다. 그리고 새롭게 변화되는 플랫폼과의 연동 문제는 플랫폼 종속적 모델에서 브리지를 통해 해결할 수 있다. 개발자는 비즈니스 규칙을 Java, C++, C# 같은 프로그래밍 언어로 소스코드를 작성하는 것이 아니라 설계 모델과 비즈니스 규칙을 기술하기 위한 새로운 도구를 이용해 프로그래밍 된다.

기존의 개발 프로세스에서 이전의 모든 분석, 설계 작업은 결국 최종 목적은 좋은 소스코드를 만들어 내는 것이다. MDA 툴은 컴파일처럼 플랫폼 독립적 모델로부터 플랫폼 종속적인 모델을 추출하고, 소스코드를 생성해서 최종적으로 완전한 시스템을 구축한다.

기존의 케이스 툴도 모델로부터 소스코드를 생성하는 순공학(Forward Engineering)을 지원하였으나, MDA 툴은 그것을 뛰어 넘어 각 컴포넌트의 비즈니스 로직을 구현하고, 바로 실행시켜 서비스를 제공할 수 있는 하나의 완전한 시스템을 구축한다는 것이 바로 MDA의 기본 사상이다.

4.3 MDA 기반 개발 프로세스의 기대효과 및 유효성 분석

4.3.1 MDA 기반 개발 프로세스의 기대효과

MDA를 적용하여 프로젝트를 개발함으로써 기대되는 효과는 여러 가지가 있지만 대략 다섯 가지를 정의하여 보았다.

첫 번째, 생산성 향상이다. 이는 소스코드의 재사용과 소스코드의 자동 생성을 통해 프로젝트로부터 중복적인 코딩 요소를 제거함으로써 효율성을 높일 수 있다.

두 번째, 비즈니스의 민첩성 향상이다. 이것은 필요한 수준에 맞추어 애플리케이션 모델을 변화시킴으로써 변화하는 비즈니스 요구에 대응하는 능력을 가지고 있다.

세 번째, 시스템 개발 및 유지보수 비용 절감과 개발기간 단축이다. 소프트웨어를 개발함에 있어 새로운 플랫폼으로 자동 포팅해 줌으로써, 장기간에 걸친 생산 주기 동안 소프트웨어의 유지보수 업무를 단순화 할 수 있다.

네 번째, 소프트웨어의 품질 향상이다. MDA는 소스코드 개선의 필요성을 인식(코드 작성이 적을수록 예상되는 부담이 준다)하고 있으며, 소프트웨어 설계를 바탕으로 한 테스트로 업계 표준 패턴이 자동으로 적용되어, 아키텍처와 설계 규정에 따른 일관성 유지 및 확장성과 성능, 그리고 가용성을 확보하게 된다.

다섯 번째, 비즈니스 요구에 맞는 소프트웨어 생산이다. 이것은 변화하는 비즈니스 상황에 맞게 즉시 대처할 수 있는 소프트웨어 개발에 대한 신속한 접근방법으로써, 변경되고 테스트된 애플리케이션의 청사진이 업계 표준으로서 소프트웨어를 제공하는데 많은 도움과 기대가 된다.

4.3.2 MDA 기반 개발 프로세스 유효성 분석

MDA를 지원하는 자동화 툴(Together Architecture 2006)을 활용하여 학사관리 시스템을 플랫폼 독립적 모델(PIM)로 작성하고, 플랫폼 종속적 모델(PSM)인 J2EE/EJB 미들웨어로 매핑 룰을 적용하고 변환하여 구축된 학사관리 시스템 사례를 가지고, <표 4.1>와 같이 MDA기반 개발 프로세스에 대한 유효성 분석을 다음과 같이 유추해 볼 수 있다.

첫 번째, 소스코드의 자동 생성과 재사용, 프로젝트 내 중복적인 요소 제거, 템플릿 및 변환 규칙을 적용한 설계 자동화 및 시스템 분석, 관리, 검사 시간 절약 등에 대한 생산성 향상이 약 40% 정도의 향상을 가질 수 있다.

두 번째, 소프트웨어의 유지보수 업무 단순화 및 문서화 작업 자동화, 개발 인력, 개발 기간 절감 효과 등에 대한 CBD로 개발 시 약 40%, 타 방법으로의 개발 시 약 30% 정도의 향상을 가질 수 있었다.

세 번째, 일관성 유지, 문서화 가능, 확장성의 용이, 가용성 확보, 시스템 유지보수에 대한 리스크 관리, 시스템 개발 시 반복적인 작업의 단순화, 창의성 극대화를 통한 소프트웨어에 대한 품질이 약 30% 의 향상을 가질 수 있다.

네 번째, 비즈니스 변화와 요구 대응 능력 및 변화하는 상황에 대한 즉시 대처능력과 소프트웨어 개발에 대한 신속한 접근에 대해서는 대략 15% 정도의 향상을 볼 수 있었다.

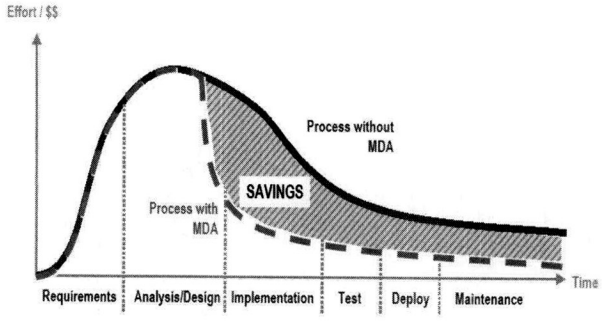
MDA기반 개발 프로세스 유효성 분석에 대한 결과를 정리해 보면 <표 4.1>와 같이 요약 할 수 있다.

(그림 4.5)와 같이 David Bertrand[7]는 개발 각 단계별 MDA적용시와 MDA비적용시의 비용 절감을 나타낸 것으로 분석 및 설계단계부터 각 단계별 비용절감을 나타내고 있다.

특히, 구현단계에서는 많은 비용 절감이 있는 것으로 나타내고 있다.

〈표 4.1〉 MDA기반 개발 프로세스 유효성 분석

구 분		효 율	비 고
생산성 및 효율 성 향상		40%↑	- 소스코드의 재사용 증가 - 소스코드의 자동 생성 - 프로젝트내 중복적인 요소 제거 - 템플릿 및 변환 규칙을 적용한 설계자동화 - 시스템 분석, 관리, 검사시간 절약
시스템 개발 및 유지보수 비용 절감과 개발기간 단축	CBD	40%↑	- 소프트웨어의 유지보수 업무를 단순화 - 문서화 작업 자동화 - 개발인력과 기간절감의 효과 등
	객체, 일반	30%↑	
소프트웨어 품질향상		30%↑	- 일관성 유지 - 문서화 가능 - 확장성 용이 - 가용성 확보 - 시스템 유지 보수의 리스크 관리 - 개발의 반복적인 작업 단순화 - 창의성 극대화
비즈니스의 민첩성 향상		15%↑	- 비즈니스 변화 요구 대응능력
비즈니스 요구에 맞는 소프트웨어 생산		15%↑	- 변화하는 비즈니스 상황에 맞게 즉시 대처 - 소프트웨어 개발에 대한 신속한 접근방법



(그림 4.5) 각 단계별 MDA적용시와 MDA비적용시 비용 절감

5. 결론 및 향후 연구과제

본 논문에서는 OMG에서 지금까지 추진한 여러 가지 표준화 작업을 바탕으로 모델 중심의 시스템 명세와 개발 방법인 MDA에 대한 개괄적인 소개와 더불어 관련 표준 및 지원기술에 대해서 알아보고, MDA방법론을 적용한 학사관리 시스템에 대해 설계와 구현을 하였고, MDA기반 개발 프로세스에 대한 유효성을 분석하였다.

MDA방법론을 적용한 개발은 시스템의 구현 결과뿐만 아니라, 분석 설계 관리 등의 프로젝트 진행 전체 결과를 재사용 가능하게 한다. 특히 특정 기술에 종속 되어 있었던 개념적 설계와 물리적 설계를 분리함으로써, 기술 여건의 변화에 따라 동일한 개념적 설계를 반복하는 불필요한 작업에 투입되는 노력을 줄여 줄 수 있었다. 그리고 설계 수준의 시스템 통합을 기술함으로 지속적인 기술 여건 및 기반 기술의 발전에도 불구하고 적은 노력으로 시스템의 통합을 이루고, 상호 운용성을 유지할 수 있다.

또한 변환 규칙을 적용해서 모델의 자동 변환을 통해 다중 플랫폼을 쉽게 지원하고, 개발자의 입장에서는 시간을 많이 필요로 하는 코드 작성 부분을 현격히 줄일 수 있으며, 개발 프로세서의 측면에서도 품질관리(Quality Assurance)를 수월하게 할 수 있고, 기술 플랫폼과 기능 변화에 신속한 대응, 시스템의 생명 주기 증가, 유지보수 용이, 개발 생산성 증진, 용이한 문서 작성, 품질 관리 비용의 감소, 양질의 시스템 구축, 개발 기간 단축, 다중 플랫폼에 대한 생산성 향상 등의 효과를 기대할 수 있다.

MDA기반 개발 프로세스가 전통적 개발 방법론에 비해 여러 가지 유효성 측면에서 효율성이 뛰어난 것을 확인할 수 있었다.

향후 OMG의 지속적인 표준화 작업이 이루어 짐으로 인해 현재 운용 되고 있는 전반적인 레거시 시스템에 대해서도 표준화 작업이 되어야 할 것이며, 그리고 MDA 지원 툴 역시 순 공학과 역 공학에 대해 완전하게 지원이 이루어져야 할 것이다. 또한 플랫폼 독립적 모델 설계 시 도메인에 대해 전반적이고 체계적인 지식을 갖고, MDA 방식을 적용하여 성공적인 개발 사례가 늘어난다면, UML로 SI업체 및 개발자 사회에서 보편적인 개발 패러다임으로 자리 잡아 갈 것으로 사료된다.

참 고 문 헌

[1] OMG, "Draft text for an MDA Guide.pdf," 2003.06.

[2] OMG Model Driven Architecture Home Page : www.omg.org./mda/index.htm.

[3] OMG, "UML 2.0 Specification," 2005.07.

[4] David Bertrand, "Model-Driven Architecture Case Study, "pp.5~6, June, 12, 2003.

[5] Jon Siegel and the OMG Staff Strategy Group, "Developing in OMG's Model Driven Architecture, "ftp://ftp.omg.org/docs/omg/00-11-05.pdf 2001.

[6] OMG, "Meta Object Facility"(MOF), 2006. 1.

[7] David Bertrand, "Model-Driven Architecture Case Study," pp.24, June 12, 2003.

[8] 권소정, "MDA를 적용한 웹 기반 영화 예매 시스템 설계 및 구현," 2004. 12

[9] 이현주, "MDA기반 EJB 컴포넌트의 PIM변환 규칙에 관한 연구," 2004. 7

[10] 천은영, "MDA구현을 위한 EJB 환경에서의 UML Profile 모델 설계 및 구현," 2004. 2.

[11] 김동규 이현정 정기원, "PIM에서 EJB기반의 PSM으로 변환에 대한 일관성 검증 규칙," 2004년도 한국정보과학회 가을 학술발표논문집 Vol.31, No.2, 2004.

[12] 박경민 "조직에서의 MDA 추진 전략과 전술," 마이크로소프트, 2004. 8.

윤 정 모

e-mail : jmyoon@snut..ac.kr

1968년 광운대학교 응용전자공학과 (공학사)

1971년 성균관대학교 경영행정대학원 (경영학석사)

1993년 일본오사카부립대학교 대학원 경영공학과(공학박사)

1966년~1982년 한국전력공사 전자계산소 과장대리

1983년~1988년 서울산업대학교 전자계산소장 겸임

2002년~2003년 일본오사카부립대학 객원교수

1982년~현재 서울산업대학교 컴퓨터공학과 교수

관심분야 : 객체지향 분석설계, 소프트웨어 공학, UML, ERP 등

김 치 호

e-mail : chk1363@naver.com

2001년 서울산업대학교 전자계산학과 (학사)

2006년 서울산업대학교 산업대학원 컴퓨터공학과 (공학석사)

1979년~1981년 삼익악기제조(주) 전산실 근무

1981년~1984년 동국무역(주) 전산실 근무

1984년~2001년 주식회사 E1[구 LG Caltex Gas(주)] 정보기술팀 근무

2001년~현재 서울특별시립 상계직업전문학교 웹프로그래밍과 교사, 서울산업대학교 공과대학 컴퓨터공학과 출강

관심분야 : 시스템개발방법론, 데이터베이스, 정보보안, 시스템 감리 등