MDA 기반 모델 변환 기법을 이용한 컴포넌트 생산성 향상에 대한 사례연구

김학인*, 최오훈**
*고려대학교 컴퓨터과학기술 대학원
**고려대학교 컴퓨터학과
{hakinee*, pens**}@korea.ac.kr

A Case Study for Improving Component Productivity using MDA based Model Transformation Technique

Hak-In Kim*, O-Hoon Choi**
*Graduate of Computer Science and Technology, Korea University
**Dept. of Computer Science & Engineering, Korea University

요 약

소프트웨어 산업 사회에서는 현재까지 생산성 향상이라는 문제에 대한 여러 가지 방안들을 제시하고 있으며, 컴포넌트 기반 개발 프로세스 역시 생산성 향상에 대한 많은 가능성을 제시하고 있다. 하지만 현실적으로 컴포넌트 간의 호환 및 상호 운용에 대한 많은 문제점들을 내포하고 있으며, 본 논문에서는 이러한 문제점들을 해결하기 위한 노력들 중의 하나로 OMG의 MDA(Model Driven Architecture) 기술을 이용하여 특정 도메인의 모델 변환(Model Transformation) 구조를 구현하고, 최적화된 개발 방법론을 적용한 실제 사례를 통하여 컴포넌트의 생산성 향상에 대한 결과를 측정하여 그 효과에 대해서 검증한다.

1. 서론

생산성(Productivity)은 산업사회의 발전 및 소프트웨어 산업에서도 간과할 수 없는 중요한 요소다. 이에 대해 소프트웨어 컴포넌트 기술의 등장은 소프트웨어 산업의 생산성 향상에 새로운 가능성을 제시하고 있다. 우리는 실행 가능한 컴포넌트들을 개발하고, 특정한 목적에 따라 컴포넌트들의 조립을 통하여 새로운 소프트웨어 시스템을 손쉽게 구축할 수 있게 되었다. 이로 인해 소프트웨어의 재사용성 증대, 품질 향상, 빠른 응용 개발을 통한 시장 경쟁력 강화 등과 같은 효과를 기대할 수 있지만, 컴포넌트 기술의 낮은 성숙도로 인해서 많은 문제점들도 발생하고 있다.

컴포넌트 기술에 대한 이해 부족은 그 중 가장 근본적이고도 중요한 문제점이라 할 수 있다. 많은 경우 단순한 UML을 이용한 모델링, CBD 방법론의 적용, EJB나 COM+과 같은 컴포넌트 기술을 이용한 바이너리 컴포넌트의 구현 자체를 컴포넌트 기술의 최종 목표로 오해하고 있다. 이로 인해 지금까지 컴포넌트 간의 상호 운용은 동일 플랫폼 상에서 바이너리 컴포넌트 수준으로 운용되고 있다. 이 경우 이기종 플랫폼 상의 컴포넌트를 상호 운용하기 위해서는 개발 당시 사용했던 참조 모델을 포기하고, 해당 플랫폼에 적합한 참조 모델로 다시 개발 작업을 수행해야만 하는 노력을 필요로 한다. 즉, 이기종 플랫폼이나 기반 기술과 같은 다양한 환경에 대한 표준화된 접근을 간과함으로써 실제 컴포넌트 기술에 기대했던 효과를 얻지 못하고 있는 것이다.

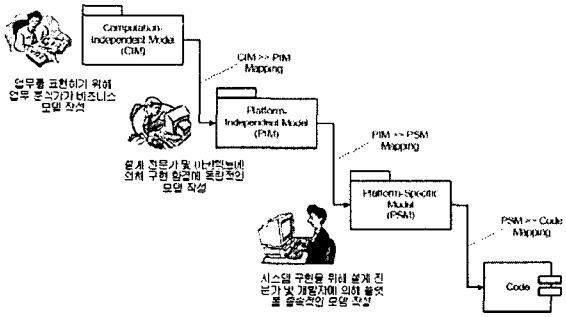
본 논문에서는 컴포넌트 개발 생산성을 향상시킬 수 있는 해결 방안을 OMG의 MDA 기술의 응용을 통해서 제시하도록 한다. MDA 기반의 모델 중심의 접근법을 이용하여 상호 운용성 문제를 해결할 수 있는 컴포넌트 시스템을 설계하고, 특정 도메인의 소프트웨어 아키텍쳐에서 요구하는 기반 환경 요소 및 품질 요소들을 효율적으로 컴포넌트 시스템에 반영할 수 있는 모델 변환(Model Transformation) 구조를 구현한다. 그리고 모델 변환 구조를 실제 적용한 프로젝트 사례를 통해서 그 효과에 대한 검증을 수행한다.

2. 관련 연구

2.1 MDA (Model Driven Architecture)

MDA(Model Driven Architecture)는 OMG(Object Management Group)에서 2001년 9월에 기술 표준으로 제정한 소프트웨어 아키텍쳐 표준 모델로서, 각 종 플랫폼 및 기술 요소에 대한 상호 운용성(Interoperability) 문제를 효과적으로 해결하기 위한 새로운 시스템 통합화 방안이다. MDA는 OMG가 현재까지 추진한 여러 표준들(UML, MOF, CWM, XMI, CORBA)을 기반으로 모든 컴포넌트 기술 요소의 표준 메타 모델(Meta Model)을 정의하고, 각 구성요소를 정의함으로써 모델 중심의 시스템을 개발하여 모든 컴포넌트 기술 요소들에 대한 호환성과 상호 운용성을 보장한다. 그리고 그 핵심은 [그림 1]과 같이 메타 모델을 기반으로 구현 플랫폼 환경에 독립적인 모델(Platform Independent Model : PIM)을 각 플랫폼

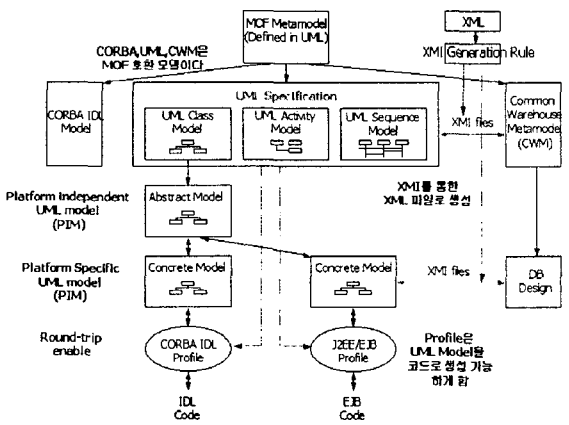
환경에 종속적인 모델(Platform Specification Model : PSM)로 자동으로 변환할 수 있는 구조를 정의하는 것이다.



[그림 1] MDA 모델 관계성 및 개발 프로세스

2.2 UML Profile 및 Meta Modeling

UML 모델을 이용하여 특정 도메인에 존재하는 특징들을 효과적으로 표현하기 위해 확장 메커니즘을 정의할 수 있으며, 이것이 UML Profile 확장이다. 그리고 UML Profile 확장에서는 모델 요소들의 스테레오 타입(Stereotype) 및 Tagged Value등의 요소들을 이용하여 다른 메타 모델의 모델들이나 도메인들을 서로 시각적으로 차별화 시킬 수 있다. [그림 2]와 같이 UML 모델에서 특정 구현 언어 기반의 소스코드를 자동 생성하거나, 생성된 어플리케이션들에 요구되는 각 종 패턴에 대한 품질 요소들을 보장하기 위해 UML Profile 확장을 통해 실현할 수 있다. 즉, UML 명세를 기반으로 확장을 원하는 형태로 추상적(PIM), 구체적 모델(PSM)을 작성하여 UML Profile을 확장한다. 이 과정에서 각 모델 및 모델 간의 변환 구조는 메타 모델링을 통해서 이루어진다. 따라서 UML Profile의 확장 및 메타 모델링을 통해서 특정 구현 언어에 대해 요구되는 형태의 소소코드를 자동 생성하고, 기반 프레임워크 등과 같은 구현 환경 요소에 대한 정보들을 모델 및 소스코드에 자동 반영할 수 있으며, 이를 통해서 어플리케이션에 대한 일관된 품질을 만족시킬 수 있다.



[그림 2] Meta Modeling과 UML Profile을 통한 어플리케이션 구현

3. 적용 사례

3.1 A사 차세대 정보 시스템 분석

A사에서 지금까지 진행되어 오던 구현 종속적 개별 플랫폼 지향 개발의 경우 모델을 통해 자동 생성된 소스의 구현 이외에 데이터베이스, 어플리케이션 서버, 환경 파일, 각 종 기반 프레임워크, 개발 IDE 환경 등에 대한 모든 기반 환경 요소들에 대한 비용이 소요되어야 했으며, 이는 곧, 자동 생성된 소스와 기반 프레임워크 간의 불일치, 수작업량 증가, 프로젝트 기간 연장 및 S/W 품질 저하 등의 여러 가지 문제점들을 유발하는 원인이 되었다.

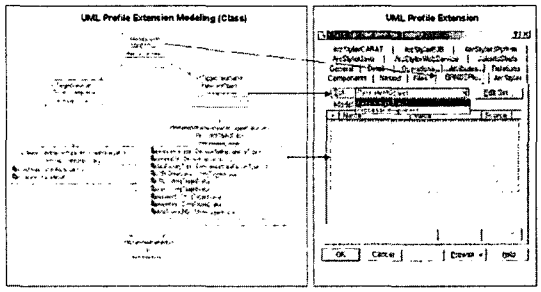
따라서 이와 같은 문제점들에 대한 해결 방안으로 A사는 CBD 기반 프레임워크 적용 기술 아키텍쳐를 정의하고 다음과 같은 작업들을 수행하였다.

- 기반 컴포넌트 프레임워크의 모델링 스타일 적용을 위한 MDA 기반의 UML Profile 확장 모델링 수행
- 생성해야 할 Artifacts에 대한 메타 모델링 수행 및 확장된 UML Profile 메타 모델과의 변환 구조 설계
- 표준화된 모델 변환 구조를 자동화 도구에 플러그인하여 자동화 실현
- 표준 모델 변환 구조를 근간으로 하는 개발 프로세스를 통해서 컴포넌트의 요구 품질 및 생산성 증대

3.2 UML Profile 확장 및 메타 모델링

전사적으로 사용하게 될 컴포넌트 프레임워크 및 컴포넌트 모델링 패턴의 효율적인 적용 및 표준화를 위해서 UML Profile의 확장 모델링 및 메타 모델링 작업을 통해서 모델 변환 구조를 설계하고 구현하였다.

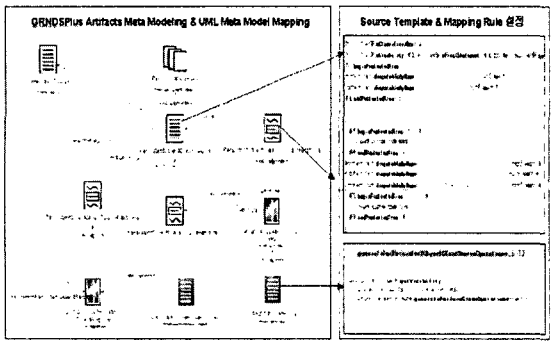
[그림 3]과 같이 기반 프레임워크 요소들을 앞으로 개발될 각 어플리케이션 컴포넌트들에 자동 반영 시키기 위해 스테레오타입 및 Tagged Value를 이용 하여 클래스, 속성, 관계, 기능 등의 메타 모델 요소들에 대해서 UML Profile을 확장하여 모델링 하였다.



[그림 3] UML Profile Extension Modeling

그리고 UML Profile 확장을 통해서 UML 모델로 확장 반영된 기존 컴포넌트 프레임워크 요소들을 실제 구현 소스 및 각 종 산출물로 반영하기 위한 메타 모델링 과정을 [그림 4]와 같이 수행하였다. 실제 생성되기를 원하는 소스 및 각 환경 파일들에 대한 포맷 및 내용들에 대한 메타 모델을 작성한다. 그리고 앞서 확장 정의된 UML Profile 모델과 소스 및 환경 파일들에 대한 메타 모델을 매핑시켜

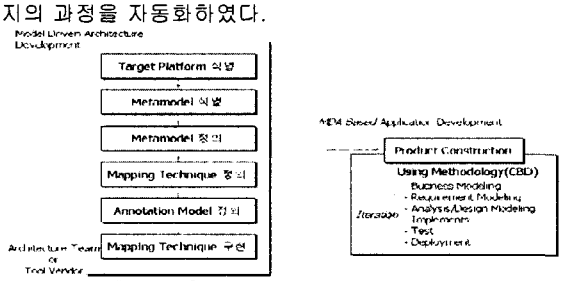
표준 모델 변환 구조를 정의한다. 두 메타 모델 사이의 변환 구조를 통해서 생성될 소스 및 환경 파일에 대한 템플릿을 작성한 후 템플릿 파일의 각 요소들을 해당 메타 모델의 각 요소들과 매핑시킨다. 이때 템플릿 파일 작성 및 메타 모델 요소에 대한 매핑을 위해 JPhython 언어를 이용하였다. 그리고 동일한 방식으로 UML Profile에 정의된 모델링 스타일 준수 여부를 검증하는 검증 규칙을 메타 모델링 하여 모델 및 생성 소스 및 환경 파일의 검증 구조를 정의하였다.



[그림 4] Artifact Meta Modeling

3.3 MDA 기반 개발 프로세스 적용

UML Profile 확장 및 메타 모델링을 통해서 표준화된 프레임워크 및 개발 환경을 기반으로 효율적인 시스템 구축을 위해서 [그림 5]와 같이 MDA 기반의 개발 프로세스를 적용하였다. UML Profile 확장을 원하는 대상 플랫폼을 식별하고, 프레임워크 등의 요소들이 반영되어야 할 컴포넌트 모델의 메타 모델을 식별하여 실제 구조화 된 소스 코드 형태로 자동 생성될 수 있도록 모델 변환 구조를 기술하는 과정을 [그림 5]의 6단계의 절차를 통해서 수행하고, 이를 자동화 도구(ArchStyler3.0)에 플러그인 형태로 적용하였다. 그리고 해당 도메인에 대해서 비즈니스 모델을 설계하고 시스템 모델과 구현에 이르는 개발 프로세스를 수립하고 모델 변환 구조가 플러그인된 자동화 도구를 이용하여 비즈니스 모델에서 컴포넌트 소스 생성까지의 과정을 자동화하였다.



[그림 5] MDA S/W 개발 프로세스

3.4 평가

평가 방법은 모델 변환 구조의 구현을 통해서 정의한 컴포넌트 프레임워크 및 컴포넌트 설계 패턴을 MDA 개발 지원 도구를 이용하여 적용 시켰을 경우와 순수 UML 모델링을 통해 컴포넌트 프레임워크 및 컴포넌트 설계 패턴을 적용 시켰을 경우를 소요 기간 및 자동화 비율의 비교를 통해서 검증하였다. 그리고 효과적인 비교 평가를 위해서 비전문가들을 대상으로 그 결과를 측정하였다.

특정 컴포넌트 및 Web Application의 모델링에서 소스 생성 및 테스트 단계까지의 소스 및 환경 파일들에 대한 자동 생성률 및 오류 확률은 다음 [표 1, 2]와 같다.

[표 1] 컴포넌트 자동 생성률 및 오류 확률

	MDA 기반	표준
Entity 관련 소스 생성률	100%	70%
Session 관련 소스 생성률	80%이상	50%
build, deploy, tester 생성률	100%	70%
오류 확률(%)	0%	10%이상

[표 2] Web Application 자동 생성률 및 오류 확률

	MDA 기반	표준
ActionFormBean 생성률	100%	30%
ActionServlet 생성률	70%이상	10%
Configuration 파일 생성률	100%	0%
오류 확률(%)	0%	20%이상

자동 생성율에 대한 측정 기준은 자동으로 생성된 소스의 라인 수를 통해서 산출하였으며, 라인 수의 측정은 소스 파일의 라인 수 측정 프로그램을 이용하였다. 그리고 해당 소요 기간을 중심으로 생산성을 비교한 결과 표준 개발 공정을 기준으로 MDA 기반 개발 과정을 비교하였을 경우 30%의 개발 기간이 단축되었음을 검증하였으며, 이는 동일 비율의 생산성 향상의 결과로 간주하였다.

4. 결론 및 향후 과제

본 연구를 통해 모델 변환 구조 구현을 통한 컴포넌트의 품질 및 생산성의 향상에 대해서 실제 프로젝트를 통하여 검증하였다. 그리고 향후 계속해서 특정 도메인에 대한 비즈니스 모델을 중심으로 이기종 플랫폼에 대한 시스템 모델 생성에 대한 실제 프로젝트 진행을 통해서 상호 운용성 향상에 대한 효율성을 측정하고, 연구할 계획이다.

참고 문헌

[1] Annerke Kleppe, Jos Warmer, Wim Bast : MDA Explained-The Model Driven Architecture : Practice and Pramise, Addsion-Wesley, 2003
[2] David S.Frankel, Model Driven Architecture : Applying MDA to Enterprise Computing, WILEY, 2003
[3] Rechard Hubert, Convergent Architecture : Building Model-Driven J2EE Systems with UML, WILEY, 2002
[4] OMG, ormsc/02-04-02 (latest ormsc guide)
[5] OMG, ormsc/02-01-04 (Interactive Objects Position Paper)
[6] OMG, MDA Guide (Version1.0.1), 2003
[7] Other papers from http://www.omg.org/mda