MDA/PIM을 이용한 제품계열 핵심자산의 명세 기법
(A Specification Technique for Product Line Core Assets using MDA/PIM)

민 현 기 † 한 만 집 †† 김 수 동 †††
(Hyun Gi Min) (Man Jib Han) (Soo Dong Kim)

요 약 제품계열(Product Line)은 핵심자산(Core Asset)을 서로 공유하는 제품들의 집합이며, 제품계열 공학(Product Line Engineering, PLE)은 제품계열을 특화(Instantiation) 할 수 있는 원리, 기술, 메커니즘과 프로세스들의 집합이다. PLE는 여러 유사한 제품들간에 공유할 수 있는 핵심자산을 만들고, 그 핵심자산을 특정 제품에 맞게 실체화 시켜서 제품을 만든다. 모델 기반 아키텍처(Model Driven Architecture, MDA)는 자동화 도구를 이용하여 모델로부터 구현으로 구체화하는 접근 방법이다. 그러므로, 핵심자산을 MDA의 플랫폼 독립적인 모델(PIM)로 정의하여 구현까지 자동화 한다면, 두 개발 방법의 장점을 극대화 할 수 있다. 하지만, 핵심자산을 표현하는 UML 프로파일은 현재까지 없으며, PLE의 핵심자산 및 제품 조립의 자동화 생산을 위한 PIM 명세 기법의 연구가 부족하다. 본 논문에서는 PLE기술과 MDA 기술을 접목하기 위해 PIM 수준의 핵심자산 명세 기법을 제안한다. 핵심자산을 명세하기 위해 아키텍처 명세, 컴포넌트 명세, 워크플로우 명세, 알고리즘 명세, 결정 모델 명세 기법을 제안한다. 본 논문의 명세 기법은 PLE, MDA 기술을 사용하여 제품의 생산성, 적용성, 유지보수성 및 품질 향상을 지원한다.

키워드 : MDA, PLE, UML 프로파일, 소프트웨어 아키텍처

Abstract A Product Line (PL) is a set of products (applications) that share common assets in a domain. Product Line Engineering (PLE) is a set of principles, techniques, mechanisms, and processes that enables the instantiation of produce lines. Core assets, the common assets, are created and instantiated to make products in PLE. Model Driven Architecture (MDA) is a new software development paradigm that emphasizes its feasibility with automatically developing product. Therefore, we can get advantages of both of the two paradigms, PLE and MDA, if core assets are represented as PIM in MDA with predefined automatic mechanism. PLE framework in the PIM level has to be interpreted by MDA tools. However, we do not have a standard UML profile for representing core assets. The research about representing PLE framework is not enough to make automatically core assets and products. We represent core asset in PIM level in terms of structural view and semantic view. We also suggest a method for representing architecture, component, workflow, algorithm, and decision model. The method of representing framework with PLE and MDA is used to improve productivity, applicability, maintainability and quality of product.

Key words : MDA, PLE, UML Profile, Software Architecture

1. 서 론

제품계열은 도메인에서 핵심 자산을 서로 공유하는 제품들의 집합이며, PLE는 제품계열을 실체화 할 수 있는 원리, 기술, 메커니즘과 프로세스들의 집합이다. 제품계열 공학은 유사한 시스템 계열에 존재하는 소프트웨어들의 공통적인 특징들을 추출하여 고품질의 재사용 가능한 핵심 자산을 만들고, 이 자산을 도메인 내에 있는 제품을 개발하는데 사용하는 접근 방법이다[1]. 하지만 제품 계열 공학에서 재사용 단위인 핵심자산을 표현하는 방법은 현재까지 표준화되지 않았다.

MDA는 시스템 개발을 자동화 도구를 이용하여 모델로부터 구현까지 자동화하여 구체화하는 접근 방법이다

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 정 회 원 : 숭실대학교 컴퓨터학과 전임연구원
hgmin@otlab.ssu.ac.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
mjhan@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
논문접수 : 2005년 3월 22일
심사완료 : 2005년 7월 25일

[2]. MDA는 상위 수준의 구현 플랫폼 독립적인 모델(PIM)을 하위 수준의 구현 플랫폼 종속적인 상세 설세 모델(PSM)로 변환하여 자동으로 구현을 한다. MDA는 다양한 플랫폼의 소프트웨어 자동화 개발에 사용 가능한 기술이다.

PLE는 여러 패밀리 맴버를 개발할 때 재사용할 수 있는 핵심자산을 만들기 위한 도메인 분석 및 프레임워크 생성의 장점이 있고, MDA는 설계 된 핵심자산 모델을 매핑 규칙 등의 자동화 기능을 이용하여 특정 어플리케이션에 적합한 제품을 만들 수 있다. PLE와 MDA 기술을 접목한다면, 효율적인 소프트웨어 생산이 가능하다. 따라서 제품계열 공학의 핵심자산을 MDA의 자동화 생산을 위한 PIM으로 표현하는 기법이 필요하다. 본 논문에서는 MDA기술을 위한 PIM 수준의 핵심자산 명세 기법을 제안한다. 핵심자산을 명세하기 위해 아키텍처 명세, 컴포넌트 명세, 워크플로우 명세, 알고리즘 명세, 결정 모델 명세 기법을 제안한다.

본 논문의 구성은 2장에서는 관련 연구를 서술하며, 3장에서는 기반 연구를 정리하고, 4장에서는 PIM으로 표현하기 위한 핵심자산의 구성 요소를 제안하며, 5장에서는 PIM의 구성 요소를 설명하고, 6장에서는 핵심자산과 PIM 명세간의 차이를 분석하고, 7장에서는 핵심자산을 표현하기 위한 명세 기법을 제안한다. 8장에서는 본 논문을 평가하기 위한 사례연구를 하고, 본 논문의 결론은 9장에서 내린다.

2. 관련 연구

본 절에서는 제안한 PLE 명세를 위한 UML 프로파일과 관련이 있는 핵심자산의 구성 요소와 다른 모델 언어에 대해 고찰한다.

2.1 Muthig의 핵심 자산 구성 요소

Bayer는 제품계열 공학의 목적은 유사한 소프트웨어 시스템의 공통과 서로 구별되는 특징을 이해하고 통제에 의해 유사한 소프트웨어 시스템의 집합을 체계적으로 개발을 지원하는 것으로 정의한다[3]. 이 목적을 달성하기 위해서는 공통성과 가변성이 모두 중요하다. 아키텍처 수준에서 공통성은 아키텍처의 기초를 정의한다. 가변성은 요구사항의 범위를 정하고, 공통 요소의 가변을 예상한다. 이 모두가 적절하게 문서화되어야 한다.

제품계열 자산(Product Line Asset)의 구성 요소는 3가지의 패키지로 구성된다[3]. 자산(Asset) 패키지는 단일 시스템의 개발에서 사용되었던 자산이다. 범용 자산(Generic Asset) 패키지는 특정 어플리케이션을 위해 나중에 특화될 수 있도록 단일 자산간의 일반적인 특성들을 모두 포함하여 정의한다. 범용 자산 패키지는 가변점(Variation Point)과 가변치(Variant)를 표현하기 위한 가변 자산 요소(Variant Asset Element)를 포함한다. 가변점은 여러 어플리케이션간에 서로 다른 특성을 갖는 것이고, 가변치는 특정 어플리케이션을 위해 설정되기 위한 특징이다. 의사결정 모델(Decision Model) 패키지는 가변점의 여러 가변치에서 필요한 가변치를 결정하여 범용자산이 특정 어플리케이션을 위한 자산으로 확장되게 한다.

2.2 Fontoura의 UML-F

객체지향 프레임워크를 위한 모델 언어(UML-F: A Modeling Language for Object-Oriented Frameworks)는 객체지향 프레임워크를 제공한다[4]. 그리고 프레임워크 가변점들을 표현한다. UML-F에서의 프레임워크는 큰 단위로 미리 정의된 패턴과 함께 구현된 몇 개의 컴포넌트의 집합(Collection)으로 구성된다. 이 프레임워크는 유사한 특징의 어플리케이션의 집합을 위해 소프트웨어 아키텍처를 구현한다. 이러한 유사한 특성은 어플리케이션에 종속적인 코드를 통해 특화될 수 있다. UML-F에서는 설계를 위해 UML의 제약사항(Constraints)을 확장하여 {appl-class}, {variable}, {extensible}, {static}, {dynamic}, {incomplete}, {for all new methods}, {optional} 8개의 새로운 요소를 제안했다. 그러나 UML-F에서 제공된 프레임워크가 개념적이기 때문에 요소가 모델에서 명확하게 식별되지 않고, 그 요소를 위한 명확한 정의가 제공되지 않는다.

3. 기반 연구

본 절에서는 본 논문에서 제안한 PLE 명세를 위한 UML 프로파일에 기반이 되는 두 가지 기술인 MDA, PLE 기술에 대해 고찰한다.

3.1 모델 기반의 아키텍처(MDA)

MDA는 Object Management Group(OMG)에서 정의한 소프트웨어 개발을 위한 프레임워크이다[2]. MDA는 소프트웨어 개발 프로세스 중 모델을 중요한 요소로 사용한다. 설계된 모델을 변환 규칙을 사용하여 자동으로 소스를 생성할 수 있다. MDA를 이용한 개발 프로세스는 분석된 제품의 요구사항을 플랫폼에 대해 독립적인 모델을 만들고 이를 매핑 규칙에 의하여 자동으로 특정 플랫폼에 종속적인 모델(PSM)을 만든다. 그리고 PSM을 다시 자동으로 변환하여 소스를 만든다[5].

그러나 MDA는 설계 모델을 자동화하는 기법만을 제안하고 있으므로, 모델을 명세하기 위한 기법이 필요하다. OMG는 플랫폼 독립적인 EDOC용 UML 프로파일[6]을 채택했으며, 특정 플랫폼을 위한 EJB용 UML 프로파일[7], CORBA용 UML 프로파일[8], 엔터프라이즈 어플리케이션 통합을 위한 UML 프로파일[9]만을 제공하여, 플랫폼 종속적인 PSM을 표현하기에 적합하지만,

제안된 프로파일들로만 핵심자산의 구성요소 모두를 PIM 수준으로 표현할 수 없다.

3.2 제품계열 공학(PLE)

PLE란 제품계열들의 구현에 사용할 수 있는 원칙, 기법, 메커니즘, 프로세스들의 집합이다. PLE는 핵심자산 공학(Core Asset Engineering)과 어플리케이션 공학 두 가지로 나눠진다[1]. 핵심자산 공학은 하나 이상의 제품에 공통으로 사용되는 핵심 자산을 개발하는 프로세스를 말한다. 그리고 어플리케이션 공학(Application Engineering)은 핵심자산 공학에서 개발된 핵심자산을 사용하여 특정 어플리케이션을 개발하는 프로세스를 의미한다. 핵심자산을 특정 어플리케이션에 맞도록 특화(Instantiation) 시킨 후 어플리케이션 개발에 사용한다[3].

그러나 제품계열은 초기에 핵심자산을 만드는 비용이 많이 들고, 핵심자산을 어플리케이션의 목적에 맞게 특화 시키고, 부가 기능을 추가 구현시켜 어플리케이션을 만들기 때문에 개발 비용이 높다. 그러므로, 최종 제품을 만드는 전체적인 자동화 기법이 필요하다.

4. 핵심자산의 구성요소

본 장에서는 PLE 핵심자산의 구성 요소를 정의한다. PLE 핵심자산을 MDA의 PIM으로 명세하기 위해서는 PLE 핵심자산의 구성 요소가 MDA의 PIM으로 표현되어야 한다. 카네기멜론 대학 SEI에서 제안한 제품 계열 프레임워크 [1]인 핵심자산은 아키텍처, 소프트웨어 컴포넌트, COTS 컴포넌트, 도메인 모델 등으로 구성된다. SEI의 아키텍처는 모듈과, 모듈간의 관계를 여러 뷰타입으로 제안한다[10]. PuLSE 방법론의 컴포넌트 핵심자산은 컴포넌트, 제어 흐름(Control Flow), 한 개 이상의 가변성으로 구성된다[11]. Muthig은 의사 결정 모델을 제안한다[3].

이와 같이 여러 문헌의 PLE의 핵심자산의 구성 요소를 분석하여 정리하면 그림 1과 같다. 구성 계층은 핵심자산의 3가지 구성요소를 보이고, 각 구성요소의 상세한 구성물은 세부 구성 계층에 표현한다. 표현계층은 핵심자산의 구성요소들이 어떠한 명세서에 표현되는지를 나타낸다.

범용 아키텍처는 구성요소와 구성요소간의 관계로 이루어진다. 컴포넌트는 구성요소에 의해 표현되고 컴포넌트간의 관계는 구성요소간의 관계로 표현된다. 컴포넌트 모델은 인터페이스와 그 인터페이스를 준수한 컴포넌트에 의해 표현된다. 컴포넌트는 객체와 객체들간의 관계를 표현한다. 또한 컴포넌트들간의 상호관계에 의해 구성된다. 그리고 이들은 범용 아키텍처에 기반하여 컴포넌트간의 의존관계를 갖는다. 그리고 컴포넌트 모델은 컴포넌트 명세서와 인터페이스 명세서에 의해 표현된다.

의사 결정 모델(Decision Model)은 가변성 명세서를 더 구체적으로 한다. 이것은 가변점들(Variation Points)과 이들과 관련된 가변치들(Variants)과 결정에 따른 영향(Effect)[12] 그리고 작업으로 구성된다. 가변점이란 제품들 사이에 서로 다른 기능적, 비기능적 특징이 있어, 어플리케이션 공학단계에서 특정 가변치에 의해 설정되어야 한다. 가변치는 가변점을 설정하기 위한 값들이다. 영향은 가변성간에 서로 영향을 받는 의존관계에 대한 설명이나 가변치 설정을 위한 사후 조건에 대하여 나타낸 것을 표현한다. 작업은 특정 제품 생성시 그 설정 된 영향을 만족시키기 위한 업무들의 집합을 말한다.

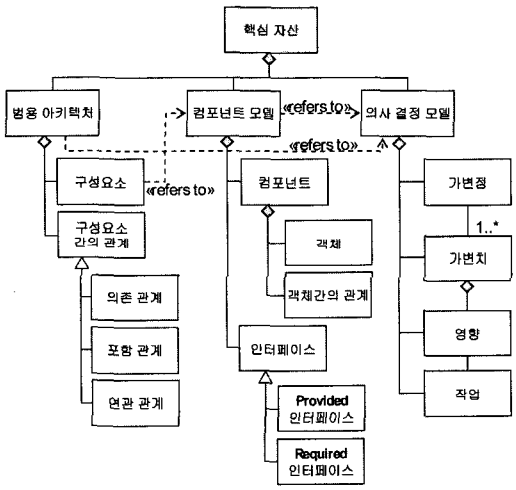

그림 1 PLE 핵심자산의 구성 요소

5. PIM의 구성요소

MDA 명세서[13]와 MDA 가이드[2]에서는 PIM의 개념적 구성요소만 명시하고 UML, MOF, CWM을 이용하여 표현할 수 있다고 제안 한다. 본 연구의 목적은 핵심자산을 PIM 수준으로 설계하여, 이 설계 모델을 파일과 같이 물리적으로 표현한 후, 물리적으로 표현된 PIM을 MDA 기술을 이용하여 PSM 및 프로그램 코드를 생성할 수 있게 하는 것이다. 그러므로, PIM을 표현하기 위한 구성 요소가 정의되어야 한다. 본 장에서는 핵심자산을 PIM으로 표현하기 위한 PIM의 구성요소를 그림 2와 같이 제안한다.

PIM은 구조적 모델(Structure Model)과 행위적 모델(Behavior Model)로 정의한다. 구조적 모델은 시스템에서 객체의 구조를 표현한 것이다. 구조적 모델에는 타입, 클래스들, 이들의 관계, 속성, 오퍼레이션들과 같은 정적인 구조를 나타낸다[14]. 구조는 시스템을 구성하는 논리적 또는 물리적 객체와 객체들 간의 관계로 구성된

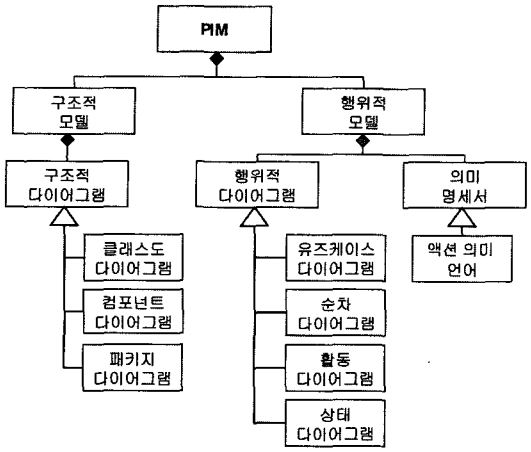

그림 2 PIM의 구성 요소

다. 구조의 특징에 따라 UML의 여러 다이어그램으로 표현된다.

행위적 모델은 시스템의 인스턴스들의 메소드, 협력(Collaboration), 상태 변화와 같은 행위 측면을 초점을 둔 모델이다[14]. 행위는 시스템의 기능적 요구사항과 비기능적 요구사항을 기반으로 시스템 내부 및 외부적으로 관찰 가능한 기능과 이러한 기능을 수행하기 위한 업무 순서와 같은 행위를 표현한다.

PIM의 구조적 모델은 특징에 따라 클래스 다이어그램, 컴포넌트 다이어그램, 패키지 다이어그램과 같은 구조적 다이어그램으로 표현 할 수 있다. 특히, 클래스 다이어그램은 PIM을 표현하기 위한 핵심 다이어그램으로 사용된다. 행위적 모델을 표현하기 위해서 UML의 유즈케이스 다이어그램, 순차 다이어그램, 활동 다이어그램, 상태 다이어그램과 같은 행위적 다이어그램을 사용한다. 그러나, 이러한 UML 다이어그램만으로 표현 못하는 행위는 액션 의미 언어(Action Semantic Language)와 같은 의미 명세서(Action Specification)을 사용하여 행위를 묘사한다. 이러한 UML 다이어그램과 액션 의미 언어는 최종적으로 XMI로 표현되어 도구내의 모듈 및 도구간의 상호간에 호환이 가능하게 한다.

6. 핵심자산과 PIM 사이의 차이(Gap) 분석

핵심자산은 제품계열 개발에서의 중요한 재사용 산출물이다. 이 산출물이 MDA의 PIM 단위로 표현된다면, PIM에서 PSM, PSM에서 코드 생성의 기존 기술 및 도구를 이용하여, 핵심 자산을 S/W개발 자동화에 의해 생산성을 향상 시킬 수 있다. 6장에서는 핵심 자산을 PIM의 구성요소로 표현하기 위한 Gap을 핵심 자원의 구성요소 별로 분석한다. 범용 아키텍처, 컴포넌트 모델, 결정 모델과 PIM을 비교 한다.

6.1 범용 아키텍처와 PIM 사이의 Gap 분석

범용 아키텍처의 표현 가능한 PIM 구성요소, Gap의 종류는 표 1과 같다. 범용 아키텍처를 표현하기 위해 PIM의 구조적 모델내의 모든 다이어그램을 활용할 수 있다. 다양한 특징을 가진 소프트웨어와 하드웨어의 구성 요소를 표현할 UML 2.0 표준이 미흡하다. 객체간의 의존관계, 포함 관계, 상속 관계 등만을 표현하고 있다. 표현 가능여부는 '지원(○)', '부분 지원(△)', '지원 안 함(-)'으로 분류한다.

6.2 컴포넌트 모델과 PIM 사이의 Gap 분석

표 2는 컴포넌트 모델의 표현 가능한 PIM 구성요소, Gap의 종류를 보여주고 있다. 컴포넌트 모델은 기능적, 정적, 그리고 동적인 관점에서 표현될 수 있는데, 각 관점은 표 2에 명시된 다이어그램을 통해 명세 가능하다.

표 1 범용 아키텍처와 PIM간의 Gap 분석

범용 아키텍처 구성요소		Gap 종류	PIM 구성 요소
분류기준	상세 세부요소		
구성 요소	소프트웨어 구성요소	△	컴포넌트 다이어그램
구성요소 간의 관계	의존 관계	○	의존관계
	포함 관계	○	포함관계
	상속 관계	○	상속관계

표 2 컴포넌트 모델과 PIM간의 Gap 분석

컴포넌트 모델의 구성 요소			Gap 종류	PIM 구성 요소
분류 기준	세부요소			
세부 구성 요소	컴포넌트	객체, 내부 컴포넌트	△	클래스, 컴포넌트 다이어그램
		연관관계	○	연관관계
	인터페이스		○	지원 인터페이스, 요청 인터페이스
관점	기능적 관점		△	유즈케이스 다이어그램 활동 다이어그램
	정적 관점		○	클래스 다이어그램 컴포넌트 다이어그램
	동적 관점		○	순차, 활동, 상태 다이어그램

기능적 관점은 유즈케이스와 활동 다이어그램을 일반적으로 사용한다. 구조적인 관점은 클래스 다이어그램을 사용한다. UML의 순차, 활동, 상태 다이어그램을 이용하여 작업의 순서와 메시지 흐름 등과 같은 동적 알고리즘을 표현할 수 있다. 그러나, 기능 및 동적 관계를 UML 다이어그램만으로 표현하여 PSM, 코드를 자동으로 변환하기에는 부족하다면 액션 의미 언어가 부가적으로 필요하다.

6.3 결정 모델과 PIM 사이의 Gap 분석

표 3과 같이 범용 아키텍처와 컴포넌트에서 발생되는 가변성 정보를 담고 있는 의사 결정 모델의 정보를 명세하기 위해 PIM에서 제공하는 구성요소는 없다. 의사 결정 모델은 어플리케이션 공학 단계에서 핵심자산을 특정 제품을 위해 특화(Instantiation)를 지원하기 위한 지침서이다. 그러나, 의사 결정 모델을 표현을 위한 다이어그램은 없다.

7. 핵심자산의 PIM 명세 기법

본 장에서는 PLE의 핵심자산을 MDA의 자동화 기법을 이용하여 제품들을 생산하기 위해, 핵심자산을 PIM으로 명세하기 위한 기법을 제안한다.

7.1 범용 아키텍처 명세 기법

제품계열 아키텍처(Product Line Architecture: PLA) 명세는 그림 1과 같이 컴포넌트, 컴포넌트들간의 관계와 범용 아키텍처로 표현한다. 컴포넌트 단위와 컴포넌트들간의 관계는 성능, 보안등의 비기능적 요구사항[15]을 고려하여 아키텍처를 명세 한다. 아키텍처를 이루는 구성요소와 요소 간의 관계를 PIM으로 표현 가능하며, 본 논문에서는 표 4와 같이 제안한다.

범용 아키텍처의 요소를 표현하기 위해 PIM의 구조적 요소 중 객체를 사용할 수 있다[15]. 그러나, 범용 아키텍처의 요소는 적용되는 뷰타입이나 스타일에 따라 종류가 다양하며 소프트웨어적인 모듈과 하드웨어적인 요소가 함께 포함되어 있다. PIM의 개체로는 다양한 종류의 요소와 특히 하드웨어적인 요소를 표현하기에 부족하다.

SEI의 컴포넌트와 커넥터(Component-and-Connector, C&C) 뷰타입의 스타일의 종류를 Pipe-and-Filter, Shared-Data, Client-Server, Publish-Subscribe Style 등으로 정의한다[10]. 표 4의 비고와 같이 C&C 뷰타입은 짝을 이루어서 작성된다. 상세한 C&C 뷰타입이 아니라, 단지 아키텍처의 구성 단위만을 나타낼 때는 «module»로 표현한다.

7.1.1 아키텍처 구성 요소 명세 기법

소프트웨어 아키텍처는 전체시스템을 작은 단위로 분류하고, 이 부분들간의 연관관계를 명세해야 한다. 각각의 부분들은 다른 부분들에 독립적으로 만들어질 수 있다. 아키텍처 스타일 구현은 컴포넌트와 그들간의 연관관계로 정의 된다. 여기서 컴포넌트란 아키텍처를 구성하는 구성 단위로서 소프트웨어 기능을 위한 논리적인 모듈 또는 서버 등의 하드웨어의 장치가 될 수 있다[14].

아키텍처 스타일을 구현하기 위해 아키텍처의 컴포넌

표 3 의사결정모델과 PIM간의 Gap 분석

의사 결정 모델의 구성 요소		Gap 종류	PIM 구성 요소
분류기준	세부요소		
요소	가변점	-	스테레오 타입을 이용한 다이어그램
	가변치	-	스테레오 타입, 꼬리 값을 이용한 다이어그램
	영향	-	결정 모델을 이용함
	작업	-	결정 모델을 이용함

표 4 범용 아키텍처 명세를 위한 UML 프로파일 요소

요 소	명세 표기법	적용되는 UML 요소	비 고
구성 단위	«module»	클래스, 컴포넌트, 패키지	Abstract
필터	«Filter»	클래스, 컴포넌트	Filter-Pipe
파이프	«Pipe»	연관관계	
저장소	«Repository»	클래스, 컴포넌트	Shared-Data
생산자	«Publish»	연관관계	Publish-Subscribe
구독자	«Subscribe»	연관관계	
소비자	«Client»	클래스, 컴포넌트	Client/Server
제공자	«Server»	클래스, 컴포넌트	
계층 표현	«Layer»	패키지	Layered Style
사용 관계	«use»	연관관계	
통제 관계	«control»	연관관계	
데이타 전달 관계	«data»	연관관계	

트가 추출된다[16]. 추출된 컴포넌트를 스타일에 적용한다. 각각의 아키텍처 스타일에 대해 관련 컴포넌트를 할당하고 연관관계를 도출한다. 스타일을 구현하는 컴포넌트와 그의 연관관계에 따라 의존(Dependency), 포함(Composition), 연관(Association)등의 관계들이 확장되어 표현된다.

범용 아키텍처를 구성하는 단위가 컴포넌트이다. 본 논문에서는 그림 3와 같이 UML 2.0에서 제공하는 컴포넌트 키워드인 «component»를 사용한다. 컴포넌트는 제품 공학 계열에 의해 어플리케이션이 만들어질 때 적당한 컴포넌트를 만들어 채우거나 COTS 컴포넌트를 구매하여 대체한다. 이때 컴포넌트간에는 독립적으로 교체되거나 업그레이드 된다.

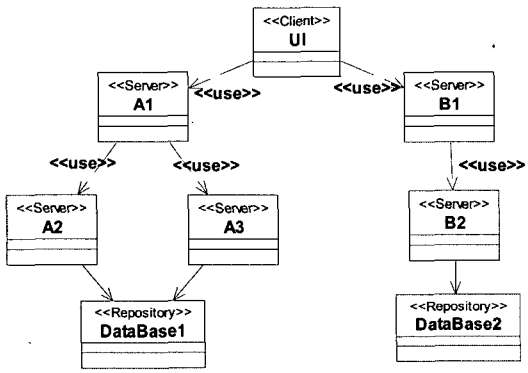

그림 3 아키텍처 명세의 예

7.1.2 컴포넌트간 연관관계 설계 기법

아키텍처에서는 구현 단위인 컴포넌트간의 연관관계를 표현한다. 연관관계는 컴포넌트들 사이의 추상화된 상호 관계이다. 컴포넌트들간의 관계를 표현할 때, 커넥터, 파이프, 생산자, 구독자 역할을 표현 할 수 있어야 한다[10]. 컴포넌트들간의 메시지 호출 관계는 «control», «data», «uses» 의존 관계로 분류한다[17]. «control» 의존관계의 의미는 하나의 컴포넌트가 시스템의 특징을 통제한다. «data» 의존관계의 의미는 하나의 컴포넌트가 입력 데이타를 다른 컴포넌트에게 전달하는 관계를 의미한다. «uses» 의존관계는 일반적으로 메시지 호출 관계를 표현한다.

7.2 컴포넌트 명세 기법

아키텍처 스타일을 구현하기 위해 아키텍처에서 컴포넌트가 추출된다[16]. 컴포넌트 명세는 구조적 관점을 명세하기 위해 컴포넌트의 내부와 인터페이스를 표현한다. 또한 기능적 관점을 표현하기 위해 유즈케이스 다이어그램, 활동 다이어그램 그리고 액션 의미 언어가 사용되며, 동적 관점을 표현하기 위해 순차 다이어그램과 커뮤니케이션 다이어그램을 사용한다.

7.2.1 인터페이스 명세 기법

인터페이스 명세는 컴포넌트의 인터페이스를 기술한 것으로, 인터페이스 명, 인터페이스 시그네처와 각각의 선조건(Pre-Condition), 후조건(Post-Condition)등을 기술한다. UML 2.0 명세서에서 표현하는 인터페이스에는 Provided 인터페이스와 Required 인터페이스로 구성된다[14]. 하지만, 인터페이스는 문맥에 의해 provided 또는 required interface 여부가 결정되므로, 특별히 구별하여 표현하지 않아도 그림 4와 같이 명세가 가능하다. 또한, 컴포넌트 간에 연결된 인터페이스의 호환 검증도 문맥으로 구별하여 분석이 가능하다.

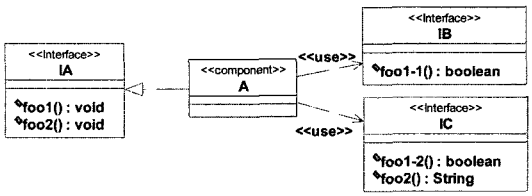

그림 4 인터페이스 명세의 예

표 5 컴포넌트 명세를 위한 UML 프로파일 요소

요 소	명세 표기법	적용되는 UML 요소	비 고
컴포넌트	«component»	컴포넌트	UML 2.0 사용
비영속적인 클래스	«Transient»	클래스	
영속적인 클래스	«Persistence»	클래스	Default
기본 키	«UniqueId»	변수	
동기 메시지	«Sync»	메소드	Default
비 동기 메시지	«Async»	메소드	
클래스간 관계	상속, 복합, 연관, 의존관계 등	연관관계	UML 2.0 사용
인터페이스	«Interface»	인터페이스	UML 2.0 사용
시그네처	함수명(매개변수: 타입,...): 반환타입	오퍼레이션	UML 2.0 사용
제약조건	{ }, pre:, post:, inv:	클래스, 메소드, 연관관계	OCL
알고리즘	문자로 표기	메소드	OCL, ASL

7.2.2 컴포넌트 명세 기법

컴포넌트 명세는 컴포넌트의 구조적인 부분을 묘사하기 위해 컴포넌트들간의 관계, 컴포넌트를 구성하는 클래스 및 클래스들간의 관계, 포함하는 내부 컴포넌트를 명세 가능하다. 그림 5와 같이 UML 2.0의 합성 구조 다이어그램(*Composite Structure Diagram*)을 이용하여 컴포넌트의 내부 설계가 가능하다.

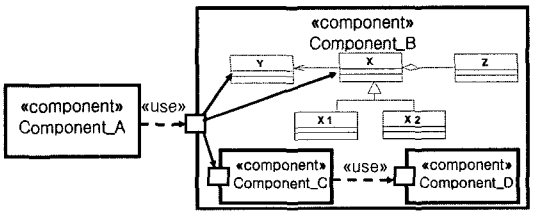

그림 5 UML 2.0을 이용한 컴포넌트 설계 예제

기존의 명세 기법은 구조적 명세에 초점이 이루어져, 객체 이름, 어트리뷰트 이름, 어트리뷰트 타입, 내부 알고리즘이 비워진 상태의 메소드의 시그네처만 생성되는 수준이다. 컴포넌트의 행위는 다음절에 제안한 워크플로우 명세 기법 및 알고리즘 명세 기법을 통해 의미적 내용을 추가하여 명세 한다.

작성된 클래스 및 클래스들간의 PIM 수준의 관계 명세는 MDA 도구를 이용하여 분석되고, 특정 플랫폼에 종속적인 PSM 형식의 클래스로 변환된 후 컴포넌트로 구현된다. 구현된 컴포넌트 소스는 생성된 객체를 컴포넌트 단위로 배포 및 설치 운영된다.

7.2.3 워크플로우 명세 기법

컴포넌트 명세는 해당 컴포넌트가 서비스 할 의미적 관점에서도 명세 되어야 한다. 컴포넌트는 여러 객체들로 이루어져있으므로, 컴포넌트가 특정 서비스를 제공하기 위해서는 객체들간의 메시지 호출 순서가 명세 되어야 차후에 자동화 도구에 의해 워크플로우를 포함하는 구체화된 소스가 생성된다.

메시지 호출 관계는 UML의 순차 다이어그램을 사용한다. 순차 다이어그램을 사용하여, 외부 컴포넌트의 인터페이스 호출, 컴포넌트 내부에 있는 객체의 메소드 호출, 해당 객체 내부의 메소드 호출 등의 순서로 업무의 순서를 표현한다.

순차 다이어그램을 통해 객체와 객체간의 메시지 호출 순서 및 컴포넌트와 컴포넌트간의 메시지 호출 순서를 작성할 수 있다. 순차 다이어그램을 MDA 기술을 이용하여 구현으로 매핑 하면, 해당 메소드는 해당 플랫폼에 맞게 메시지 호출하는 구현을 작성할 수 있다.

7.2.4 알고리즘 명세 기법

UML의 순차, 활동, 상태 다이어그램을 이용하여 작업의 순서와 메시지 흐름 등과 같은 동적 알고리즘을 표현할 수 있다. 또한 액션 의미 언어를 사용하여 표현이 가능하다. 표 6과 같이 알고리즘을 표현하기 위해서 본 논문에서는 UML 2.0과 액션 의미 언어인 Executable UML[18]을 사용한다.

7.3 가변성 및 결정 모델(Decision Model) 명세 기법

핵심자산의 특화(Instantiation)[12]는 컴포넌트의 특화(Customization) 기법[19]과 다르다. 컴포넌트는 이미 다양한 사용자를 위해 가변성을 내포하고 있는 바이너리 상태에서 외부의 사용자의 설정 값에 의해 컴포넌트가 설정되지만, 핵심자산은 설계 시 모든 제품을 위한 요소를 모두 포함하게 된다. 그리고 어플리케이션 공학 단계에서 사용자의 목적에 맞게 설정된 해결 모델(Resolution Model)에 의해 제품에 필요 없는 구조적, 의미적 설계가 선택 및 삭제되고, 적합한 핵심자산만 제품 생산에 적용된다[12].

7.3.1 가변성 명세 기법

UML은 가변성을 표현하기 위한 장치를 제시하고 있지 않다. 그러므로, 이러한 비 표준 표기법은 일반적인 MDA 도구에서 인식 할 수 없다. 만약 핵심자산의 가변성을 표준 표기법으로 정의한다면, MDA 도구는 가변성과 가변점을 식별할 수 있으며, 가변성이 적용될 가변점의 위치가 식별되고, 그 가변점에 적용될 가변치들이 식별된다면, 도구에 의해 자동으로 구현이 가능하다.

핵심자산의 가변성의 종류를 속성 가변성, 로직 가변성, 워크플로우 가변성, 인터페이스 가변성, 데이타 베이스 가변성 5종류로 분류한다[20]. 그림 6과 같이 핵심자산의 속성 가변점은 «VP-A»로 표현하고, «VP-L»은

표 6 액션 의미 언어의 예

활동	문법
객체 생성	create object instance <object reference> of <class>;
객체 삭제	delete object instance <object reference>;
객체 선택	select [one|any|many] <object reference set> *from instances of <class> where <where clause>*;
속성 쓰기	<object reference>.<attribute name> = <expression>;
속성 읽기	···<object reference>.<attribute name>;
메시지 호출	generate signal action to <class>

표 7 가변성 명세를 위한 UML 프로파일 요소

요 소	명세 표기법	적용되는 UML 요소	비 고
가변점	«VP»	변수, 메소드	
속성 가변점	«VP-A»	변수, 유즈케이스	
로직 가변점	«VP-L»	메소드, 유즈케이스	
워크플로우 가변점	«VP-W»	메소드, 유즈케이스	
인터페이스 가변점	«VP-I»	오퍼레이션	
영속성 가변점	«VP-P»	메소드	
가변성 영역	{vScope = 값}	메소드, 유즈케이스	Close, Open, Unknown
가변점 식별 번호	{vpID = 값}	변수, 메소드	
가변점 형태	{vpType = 값}	클래스 다이어그램, 순차 다이어그램, 활동 다이어그램, 상태 다이어그램	opt, alt
제약조건	{ }, pre:, post:, inv:	클래스, 메소드, 연관관계	OCL
알고리즘	문자로 표기	메소드	OCL, ASL

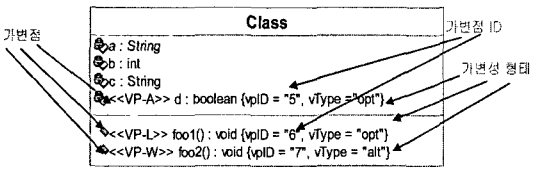

그림 6 가변성 설계 예제

로직 가변치가 적용될 로직 가변점을 묘사한다. 워크플로우 가변치가 적용될 워크플로우 가변점은 «VP-W»로 표현하고, 인터페이스 가변점은 «VP-I»로 표현하고, 데이타베이스 가변점은 «VP-P»로 표현한다. 각 가변점은 가변점을 식별하기 위해 vpID를 꼬리표로 사용하고, 가변성의 형태는 vType을 사용한다. 만약, 가변점에 여러 가변치중에 한 개가 선택되어야 하면 'alt' 값을 사용하고, 선택적으로 가변치가 필요하다면 'opt' 값을 사용한다. 제안된 스테레오 타입 및 꼬리 값은 클래스 다이어그램 외의 모든 UML 다이어그램 명세에 적용 된다.

핵심자산의 가변점은 어플리케이션 개발 때 특화되기 위한 가변치를 명세해야 한다. 그림 6과 같이 핵심자산에 명세 된 가변치는 어플리케이션 생산을 위한 핵심자산의 가변점 설정에 사용된다. 이 가변치에 의해 최종 어플리케이션을 위한 PIM 설계가 완성된다. 각각의 가변치는 가변점의 고유 아이디로 식별된다. 가변치의 서로 다른 로직은 6.2.3절의 워크플로우 명세 기법 또는 6.2.4절의 알고리즘 명세 기법을 사용하여 정의된다.

7.3.2 결정 모델 명세 기법

가변성을 위한 가변점과 가변치를 명세한 후 각 가변점과 가변치에 핵심자산 특화를 위한 영향과 작업을 명세하여야 한다. 그러나 영향과 작업의 경우 다이어그램에 모두 표시할 경우 많은 가변치, 영향과 작업의 명세 내용이 복잡하여 다이어그램의 가독성과 이해성이 떨어진다. 따라서, 표 8과 같이 영향과 작업을 표를 이용하여 명세 한다.

이와 같은 표는 도구의 UI에서도 지원이 가능하다. 제안된 결정 모델의 표는 도구 내에서 또는 도구간의 상호운용을 위해서 XML을 사용하여 표현이 가능하다. 그림 7은 결정 모델을 XMI로 표현하기 위한 스키마이다.

8. 사례 연구

본 장에서는 제안된 핵심자산 명세 기법의 적용을 위해 도서 관리 시스템을 도메인으로 하는 사례연구를 다음과 같이 수행하였다. 도서 관리 시스템의 경우에는 유료, 무료 도서관, 대출 방법, 대출 기간, 도서 회수 방법, 고객에 대한 마일리지 적립 등의 제품별 다양한 휘처를 가지고 있다. 그러므로 제품계열 공학의 접근 방법이 필요하다.

8.1 범용 아키텍처 설계

본 사례 연구의 대여 관리 시스템은 그림 8과 같이 Rental, Customer 컴포넌트가 식별되었다. Rental 컴포

표 8 결정 모델의 예

가변점ID	가변치ID	영향	작업
10	1	고객의 등급에 따라 고객포인트를 적립한다.	*Point = 10 * customer.rank;* generator IncPoint() to Member; ...
	2	...	...

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
        elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="DecisionModel">
    <xs:annotation>
      <xs:documentation>Comment describing your root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="VariantPoint" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Variant" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Effect" minOccurs="0"/>
                    <xs:element name="AttachedTask" minOccurs="0"
                                                    maxOccurs="unbounded"/>
                  </xs:sequence>
                  <xs:attribute name="ID" type="xs:string " use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

그림 7 의사결정 모델의 XMI 스키마

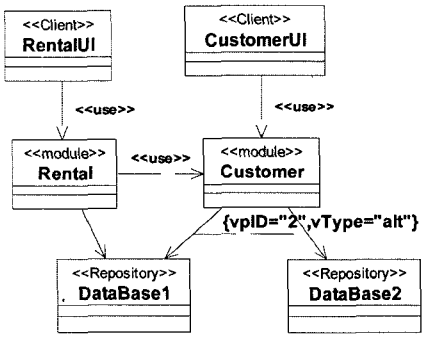

그림 8 대여관리를 위한 아키텍처 명세

넌트는 Customer 컴포넌트를 사용한다. Rental과 Customer 컴포넌트의 데이타베이스는 가변성을 갖는다. 이들은 공유 데이타(Shared Data) 방식인 경우 같은 데이타베이스 서버를 사용하거나 분산 데이타(Distributed Data) 방식인 경우 서로 다른 데이타베이스 서버를 사용할 수 있다. 하지만, Customer 컴포넌트는 2개의 데이타베이스 중에 하나는 사용해야 하는 선택적 속성을 갖는다.

8.2 컴포넌트 명세

그림 9는 대출 관리 시스템의 Rental 인터페이스를 위한 구조적, 의미적 명세의 예이다. Rental 컴포넌트는 IRental 제공 컴포넌트를 특화하고, Rental 컴포넌트는 ICustomer 요청 인터페이스와 IReservation 요청 인터페이스가 필요하다. 또한 한번 대출시 5개를 초과하여 대여할 수 없기 때문에, OCL이나 액션 언어를 사용하여, IRental 인터페이스의 checkOutItem( ) 메소드의 선행조건을 5개 미만으로 설정한다.

대출물 반환 알고리즘의 예제는 그림 10과 같다. 조건문에 의해서 반납이 반납일자 내에 반납되었는지, 반납일이 지났는지를 분석한다. 반납일이 지난 경우는 연체료 100원을 회계 정보에 남긴다. 반납을 받으면 회원의 마일리지를 수정한다. 만약 반납일 내에 반납하면, 마일리지를 추가하고, 반납일이 지나면 마일리지를 삭감한다. 반납일보다 일찍 반납하면 하루당 100원이 적립되며, 반납일보다 늦게 반납하면 하루당 10원이 삭감된다.

8.3 결정 모델

Rental 핵심자산은 대출을 할 때 패밀리 멤버마다 여러 종류의 마일리지를 적립하는 방법이 있다. 그림 11의 가변점ID 10의 returnItem() 오퍼레이션은 가변치 1의 경우에는 고객의 등급에 따라 차별적으로 마일리지를 적립하는 가변치이고, 가변치 2의 경우에는 모두 100점을 적립해주는 알고리즘을 명세 한다. Rental 클래스는 fee, penalty를 속성 가변성으로 갖는다. 무료 대여 시스템이면, 필요 없는 속성들이다. 따라서 가변성 형태는

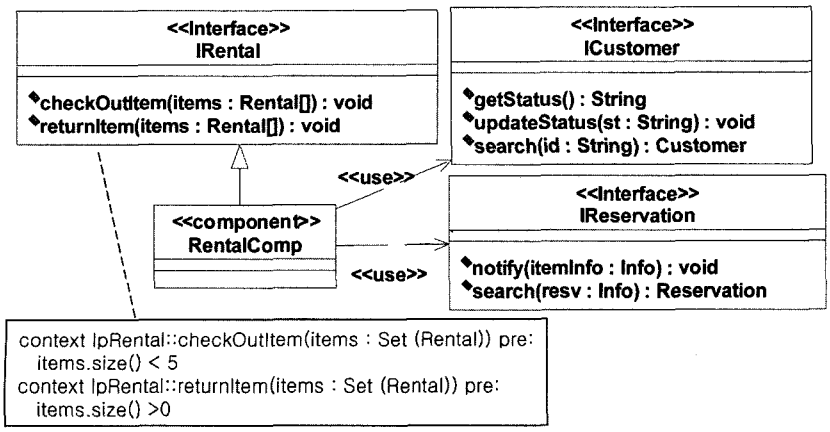

그림 9 대여 컴포넌트의 인터페이스 설계

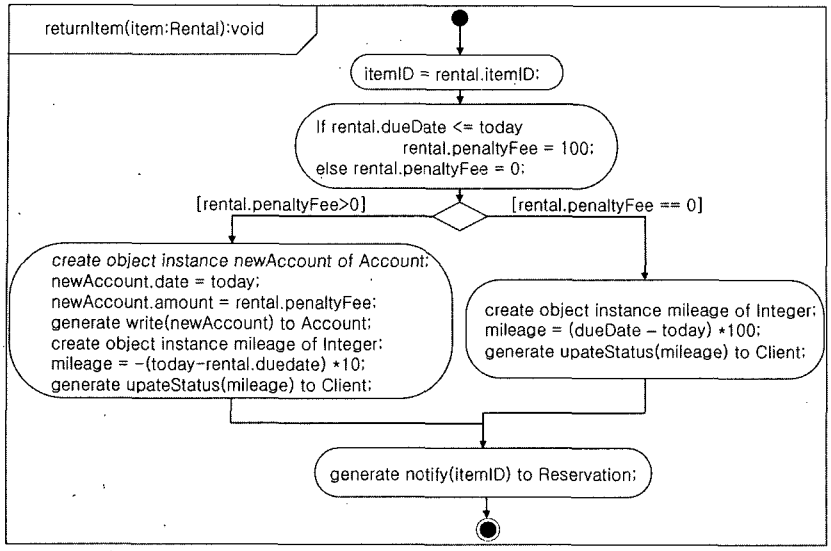

그림 10 도서 반환 알고리즘 설계

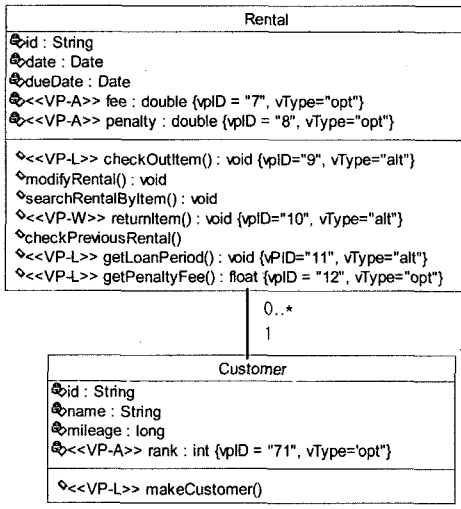

그림 11 가변성이 식별된 대여 컴포넌트 설계 예제

'opt'으로 설계한다. 물품을 반납하는 checkOutItem은 로직 가변성을 가지고 있으며, 그 식별 아이디는 9이다. 또한 여러 반납 알고리즘 중에 한 개가 선택되어야 하므로 'alt' 가변성 속성을 갖는다. 반납이 지연될 경우, 벌금을 부여하는 기능인 getPenaltyFee( )는 무료 시스템의 경우에는 필요 없을 수 있으므로 'opt' 가변성 속성을 갖는다.

그림 11의 returnItem()의 워크플로우 가변성을 영향과 작업을 나타낸 결정 모델은 표 9와 같다. 표 9는 대출품의 반납시 적립금을 고객에게 입금하는 서로 다른 방식을 명세 한다. 그림 11은 returnItem()의 워크플로우 가변성을 영향과 작업을 XMI 형태로 출력한 예이다. 표 8와 같이 작성된 결정 모델은 도구에 의해 모듈간의 연동이 어려우므로, 표현된 서식을 그림 7의 의사결정 모델의 XMI 스키마를 이용하여 그림 12와 같이 XML 형태로 도구를 이용하여 저장이 가능하다.

8.4 핵심 자산 특화

MDA의 변환 기술을 이용하여 범용 핵심자산과 해결 모델(Resolution Model) [12]을 이용하여 특정 제품을 위한 특화된 핵심자산으로 변환한다. 핵심자산 공학단계에서 정의된 그림 12와 같은 결정 모델에는 가변점과

표 9 물품 반납 휘처의 결정 모델 예

가변점ID	가변치ID	영향	작업

10	1	물품을 반납시 고객의 등급에 따라 고객 포인트를 적립한다.	{if vpID==71 then varID == 1}
			generator notifyReservation() to Reservation;
			Point = 10 * customer.rank;
			generator IncPoint() to Member;

	2	물품을 반납시 고객포인트에 일정 포인트를 적립한다.	Point = 100;
			generator IncPoint() to Member;
			generator notifyReservation() to Reservation;

```
<?xml version="1.0" encoding="UTF-8"?>
<DecisionModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=". \DecisionModel.xsd">
  <VariationPoint ID="10">
        <Variant ID="1">
              <Effect>물품을 반납시 고객의 등급에 따라 고객포인트를 적립한다.</Effect>
              <AttachedTask>{if vpID==71 then varID == 1}</AttachedTask>
              <AttachedTask>generator notifyReservation() to Reservation;</AttachedTask>
              <AttachedTask>Point = 10 * customer.rank;</AttachedTask>
              <AttachedTask>generator IncPoint() to Member;</AttachedTask>
        </Variant>
        <Variant ID="2">
              <Effect>물품을 반납시 고객포인트에 일정 포인트를 적립한다.</Effect>
              <AttachedTask>Point = 100;</AttachedTask>
              <AttachedTask>generator IncPoint() to Member;</AttachedTask>
              <AttachedTask>generator notifyReservation() to Reservation;</AttachedTask>
        </Variant>
  </VariationPoint >
</DecisionModel>
```

그림 12 XMI로 표현한 의사결정 모델

가변점에 적용 될 여러 가변치가 정의 되어 있으며, 어플리케이션 공학단계의 해결 모델에서는 특정 제품 개발을 위해 가변치를 선택한다. 해결 모델에서는 그림 12의 가변점 ID가 10인 물품 반납 방법을 '1' 또는 '2'를 선택한 정보를 갖는다. 그리고 해결 모델의 가변치 정보를 이용하여 특정 제품으로 MDA의 변환 규칙에 의해 특화된 핵심자산을 만든 후에 어플리케이션의 코드로 생성 가능하다[3].

9. 결 론

제품계열은 도메인에서 공통 자산을 서로 공유하는 제품들의 집합이다. PLE는 유사한 시스템 계열에 있는 소프트웨어들의 공통적인 특징들을 추출하여 고품질의 재사용 가능한 핵심 자산을 만들고, 이 자산을 도메인 내에 있는 어플리케이션을 개발하는데 사용하는 접근 방법이다. 하지만, PLE에서 재사용 단위인 핵심자산을 표현하는 방법은 현재까지 표준화되지 않았다. 따라서, PLE의 핵심자산을 MDA의 자동화 생산을 위한 PIM 모델로 표현하는 기법을 제공하였다.

본 논문에서는 핵심자산을 표현하기 위한 핵심자산의 구성요소와 MDA 기반의 PIM을 표현하기 위한 구성요소를 제안하였다. 이들 구성요소를 비교하여 핵심자산과 표현하기 위한 PIM간의 Gap을 분석하였으며, 이를 바탕으로 핵심 자산을 PIM으로 명세하기 위해 범용 아키텍처 명세, 컴포넌트 명세, 가변성 및 결정 모델 명세 기법으로 나누어서, 핵심자산을 PIM으로 명세하는 기법을 제안하였다.

또한 PLE를 위한 UML 프로파일의 실효성과 이해를 돕기 위해 도서 관리 시스템을 기반으로 사례 연구를 하였다. 본 논문에서 제안한 핵심 자산의 명세기법은 MDD 기술과 결합하여 제품의 생산성, 적용성, 유지보수성 및 품질 향상을 지원한다.

참 고 문 헌

[1] Clements P, Northrop L, *Software Product Lines*, Addison Wesley, 2002.

[2] OMG, *MDA Guide Version 1.0.1*, omg/2003-06-01, June 2003.

[3] Muthig, D. and Atkinson, C., "Model-Driven Product Line Architectures," *SPLC2 2002, Lecture Notes in Computer Science 2379*, pp.110-129, 2002.

[4] Fontoura, M., Pree, W., and Rumpe, B., "UML-F: A Modeling Language for Object-Oriented Frameworks," *14th European Conference on Object Oriented Programming (ECOOP 2000), Lecture Notes in Computer Science 1850*, pp. 63-82, 2000.

[5] Kleppe, A., Warmer, J. and Bast, W., *MDA Explained*, Addison-Wesley, 2003.

[6] OMG, *UML Profile for EDOC V1.0*, http://www.omg.org/technology/documents/formal/edoc.htm, 2004.

[7] JCP, *UML™ Profile For EJB_Draft*, Java Community Process. 2001.

[8] OMG, *UML™ Profile for CORBA Specification V1.0*, Nov. 2000.

[9] OMG, *UML™ Profile and Interchange Models for Enterprise Application Integration (EAI) Specification*, 2002.

[10] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, 2003.

[11] Joachim, B., et.al., "PuLSE: A Methodology to Develop Software Product Lines," *The Symposium on Software Reusability'99*, Los Angeles, May 1999.

[12] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E.,

Laitenberger, O., Laqua, R., Uthing, D., Paech, B., Wuest, J. and Zettel, J., *Component-based Product Line Engineering with UML*, Addison-Wesley, 2001.

[13] Object Management Group, *Model Driven Architecture (MDA)*, pp.1-31, July 2001.

[14] Rumbaugh, J., Jacobson, I, and Booch, G., *The Unified Modeling Language Reference Manual Second Edition*, Addison-Wesley, 2004.

[15] Gomma, H., *Designing Software Product Lines with UML from Use Cases to Pattern-based Software Architectures*, Addison-Wesley, 2004.

[16] Kim, S., Chang, S., "A Systematic Method to Identify Component," *Proceedings of APSEC 2004*, Nov., 2004.

[17] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method : A revolutionalry initiation approach to a product line architecture," *VTT Technical Research Enctre of Finland, ESPOO2002*, 2002.

[18] Mellor, S. and Balcer, M., *Executable UML: A Foundation for Model-Driven Architecture*, Addison Wesley, 2002.

[19] Kim, S., Min, H., and Rhew, S., "Variability Design and Customization Mechanisms for COTS Components," *Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2005)*, To Appear, May 2005.

[20] Kim, S., Her, J., and Chang, S., "A Formal View of Variability in Component-Based Development," *Journal of Information and Software Technology*, To Appear, 2005.

민 현 기

1999년 건양대학교 전자계산학과 공학사 2001년 숭실대학교 컴퓨터학과 공학석사 2005년 숭실대학교 컴퓨터학과 공학박사 2005년 3월~현재 숭실대학교 전임연구원. 관심분야는 컴포넌트 기반 개발(CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA)

한 만 집

2004년 부산외국어대학 전자계산학과 공학사. 2004년 3월~현재 숭실대학교 석사과정. 관심분야는 컴포넌트 기반 개발(CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA)

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 32 권 제 8 호 참조