가변성 타입을 이용한 프로덕트 라인 핵심자산 특화 프로세스

(A Core Asset Instantiation Process using Variability Type in Product line Engineering)

강 현 구 † 장 수 호 † 김 수 동 ††
(Hyun Koo Kang) (Soo Ho Chang) (Soo Dong Kim)

요 약 프로덕트 라인 공학(Product Line Engineering, PLE)은 한 도메인의 공통기능을 핵심자산화하고 이를 재사용하여 다양한 어플리케이션을 생성 할 수 있는 재사용 패러다임이다. 핵심자산을 효과적으로 활용하기 위해서는 각 어플리케이션의 요구사항을 기반으로 특화 해야 할 값 즉, 가변치를 도출하고 이를 기반으로 핵심자산을 특화 한다. 이를 위해, 아키텍처 가변성과 컴포넌트 내부의 가변성이 정확히 도출되어야 하며, 이를 반영한 체계적인 특화 프로세스와 지침이 정의되어야 한다. 본 논문에서는 핵심자산의 구성요소와 구체적인 가변점 종류를 제안하고 이를 표현하기 위한 핵심자산 산출물 양식을 정의한다. 그리고, 제안된 핵심자산의 구성요소와 가변점 종류를 기반으로 정의된 핵심자산을 이용하여 어플리케이션을 생성하는 체계적인 프로세스를 제안한다. 또한 제안된 프로세스를 적용하는 사례연구를 통하여 정의된 가변성 표현 및 특화 프로세스의 실용성을 검증한다. 제안된 프로세스를 이용하여 구체적인 핵심자산 및 가변성의 설계가 가능하며 프로덕트 라인에서의 실용적인 어플리케이션의 개발이 가능해 질 수 있다.

키워드 : 제품계열공학, 핵심자산 특화, 프로세스, 가변성

Abstract *Product Line Engineering(PLE) is a software reuse paradigm that core assets are defined using common features in a domain and are instantiated in various applications. To apply the core asset effectively, variants which satisfy application requirements are extracted and the core asset should be also instantiated based on the variants. For the work, variability on architecture and components should be extracted exactly and an instantiation process and guidelines should be defined based on this variability. In this paper, we define variability types depending on core assets elements and describe artifact templates related to the variability. We also propose a systematic process which uses defined core assets including variability and verify practicability of the proposed process and variability expression through doing case study. If utilizing with theproposed process in PLE, it can be feasible to model concrete core asset and variability and to utilize practical application engineering.*

Key words : Product Line Engineering, Instantiation, Process, Variability

1. 서 론

프로덕트 라인 공학(Product Line Engineering, PLE)은 핵심자산을 이용하여 여러 어플리케이션을 효과적으로 만들기 위한 대표적인 재사용 개발 패러다임이다. PLE는 크게 핵심자산 공학(Core Asset Engineering)과 어플리케이션 공학(Application Engineering)으로 이루어진다. 핵심자산 공학에서는 어플리케이션 패밀리에서 재사용 가능한 자산을 준-어플리케이션(Semi-Application)으로 정의하고 개발하며, 어플리케이션 공학에서는 핵심자산 공학에서 개발된 핵심자산을 재사용하여 어플리케이션을 생성한다.

어플리케이션 공학에서 핵심자산을 이용하여 어플리케이션을 생성하는 활동을 특화라 한다[1]. 특화는 핵심자산에 포함된 어플리케이션 패밀리의 공통성(Commonality) 및 가변성(Variability)에서 가변적인 부분을 목표 어플리케이션에 맞게 어플리케이션을 만드는 활동이

· 본 연구는 한국과학재단 특정기초연구(R01-2005-000-11215-0)지원으로 수행되었음
† 학생회원 : 숭실대학교 컴퓨터학과
h9kang@otlab.ssu.ac.kr
shchang@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
논문접수 : 2005년 7월 29일
심사완료 : 2005년 12월 22일

다. 기존의 PLE 연구에서는 효과적인 특화를 위해 가변성을 중요하게 다루면서 가변성에 대한 연구가 활발히 진행되고 있다. 특히 가변성 정의, 가변점과 가변치와의 관계, 가변성간의 의존성에 관한 연구가 진행 되어 왔다. 가변적인 부분을 효율적으로 특화 하기 위해서는 핵심자산의 가변성이 발생하는 부분, 즉 가변점(Variation Point)이 구체적으로 정의되어 있어야 한다. 구체적인 가변점의 정의란 핵심자산의 구성요소에 따른 가변점을 의미하며, 가변점간 또는 가변점들에 대한 가변치(Variant)간의 연관관계에 대한 정의도 고려되어야 한다[2].

본 논문에서는 프로덕트 라인의 실용적인 특화를 위해 핵심자산의 구성요소에 따른 상세한 가변점 종류를 정의하고 이를 표현하기 위한 산출물 양식을 정의한다. 그리고, 정의된 가변점을 이용하여 핵심자산을 정의하는 체계적인 프로세스를 제안한다. 또한 제안된 프로세스 적용하는 사례연구를 통하여 정의된 가변성 표현 및특화 프로세스의 실용성을 검증한다.

2. 관련연구

KobrA는 컴포넌트 기반의 제품 계열 공학 방법론으로서 프레임워크 공학(Framework Engineering, FE)과 어플리케이션 공학(Application Engineering, AE)로 구성된다[1]. FE는 재사용 단위인 프레임워크(Framework)을 정의하는 단계고 AE는 FE에서 정의된 핵심자산을 이용하여 어플리케이션을 생성하는 단계이다. 이 단계에서 가변성을 해결하는 활동을 인스턴시에이션(instantiation)이라 하고, 가변성을 해결하기 위해 의사결정모델(Decision Model)로부터 유도된 의사결정해소모델(Decision Resolution Model, DRM)을 사용한다. AE는 환경실현 인스턴시에이션(Context Realization Instantiation)과 명세와 실현 인스턴시에이션(Specification and Realization Instantiation)으로 구성된다. 첫 단계인 환경실현 인스턴시에이션은 프레임워크에서 의사결정 모델에 의해서 인스턴시에이션 되고, 이때 어플리케이션 개발자로부터 조언을 얻는다. 이것은 효율적인 비용을 결정하고 프레임워크에서 제공하는 기능과 고객이 요구하는 기능의 중복을 결정하는 단계이다. 두 번째 단계인명세와 설정 인스턴시에이션은 프레임워크에서 반복적인 명세와 설정을 통해 특정 컴포넌트 제약 구조(Komponent containment tree)를 생성한다. 생성된 특정 컴포넌트 제약 구조를 일관성 규칙을 적용하여 특정 컴포넌트 모델을 만든다. 본 연구는 프로세스를 정의하였지만 아키텍처 수준에서의 프로세스 명세가 필요하고 산출물에 관한 명세와 산출물간의 관계 설명이 더 필요하다. 특히 어플리케이션 공학에서 상세한 활동이 정의되어야 한다.

Bosch의 어플리케이션 공학 프로세스는 어플리케이션 공학이 효율적으로 사용될 수 있도록 제안한 논문이다[3]. 범용적 제품 유도 프로세스(Generic Product Derivation Process)는 초기(initial) 단계와 반복(Iteration) 단계로 구성된다. 초기 단계는 초기 제품 세부활동으로는 초기 제품 개발(Initial Software Product Configuration)에서 공통자산(Shared Product family assets)의 부분집합을 조립(Assembly) 활동과 기존의 형상(Configuration)과 가장 가까운 형상(Configuration)을 선택하는 개발선택(Configuration Selection)활동으로 구분한다. 반복단계(Iteration Phase)에서는 초기 제품을 반복해서 요구사항을 만족하게 한다. 특히 적용(adaptation)은 제품 유도 프로세스 동안에 코드가 요구되는 단계로 초기, 반복 단계 모두 사용한다. 이 단계에서는 공통 산출물(Shared Artifact) 외에 것, 즉 가변성을 설명한다. 이 연구에서는 적용의 3가지 수준(level)를 제시하고 있다. 초기수준 제품 명세 적용(Product Specific Adaptation)으로 공통 산출물을 이용하여 어플리케이션 공학을 수행한다. 두 번째(중간) 수준은 반응 진화(형성)로 공통 산출물이 다른 제품과의 연관성을 고려 해야 한다. 세 번째(상위) 수준으로는 사전(행동)진화로 현재 요구 기능 분석과 미래에 요구 기능 분석을 모두 고려한다. 그러나 본 연구에서는 가변성과 각 프로세스의 활동간의 연관관계 명세가 부족하다. 실용적인 핵심자산을 특화하기 위해서는 가변성에 따른 프로세스와 지침이 필요하다.

인스턴시에이션에 관한 다른 연구[4]에서는 제품계열 공학에서 핵심자산의 인스턴시에이션을 위한 체계적인 방법론을 제안하였다. 핵심자산의 인스턴시에이션을 위한 지침과 프로세스를 3단계로 제안하고 있다. 첫 단계는 DRM(Define Resolution Model)단계로 핵심자산과 어플리케이션 사이의 공통된 특징(feature overlapped)을 구분하고 이중에서 이용 가능한 가변점을 선택하여 가변치의 범위를 Close, Open으로 설정하는 단계이다. 다음 단계는 가변점이 있는 아키텍처와 컴포넌트에 가변치를 어플리케이션 요구사항에 만족하는 단계이고 마지막으로 세 번째 단계는 DRM과 인스턴시에이션 된 핵심자산을 검증하는 단계이다. 그러나 인스턴시에이션의 효과적인 수행을 위해 가변성 형태에 따른 세부 활동이 더 상세히 정의 되어야 하며, 특히 아키텍처 가변성에 대한 인스턴시에이션이 상세히 다루어져야 한다.

3. 핵심자산의 구성요소 및 표현방법

상세한 수준의 특화 프로세스를 정의하기 위해서 핵심자산의 구성요소가 상세한 수준에서 정의되어 있어야 한다. 본 장에서는 핵심자산의 구성요소를 정의하고, 구

성요소에서 발생하는 가변성의 종류 및 표현 방법을 정의한다.

3.1 핵심자산의 구성요소

제품계열공학에서 핵심자산의 구성요소는 제품계열 아키텍처, 컴포넌트, 인터페이스, 가변성을 명세한 의사결정 모델로 이루어진다[5]. 그림 1은 같이 각 요소에 대한 관계를 보여준다.

제품 계열 아키텍처(Product Line Architecture, PLA): 소프트웨어 아키텍처를 표현하기 위해서는 뷰타입을 적용할 수 있고, 적용된 뷰타입에 따른 비기능적인 요구사항을 만족시키기 위해 스타일들을 적용할 수 있다[6][7]. 이러한 아키텍처는 아키텍처의 구성 요소와 요소간의 관계로 표현한다. 그런데 제품계열 아키텍처는 도메인에서의 범용적인 아키텍처를 의미하므로, 요구사항을 표현하는 스타일, 구성요소, 구성요소간의 관계가 재사용 되어야 한다[8].

컴포넌트 모델(Component Model): 컴포넌트 모델은 컴포넌트와 인터페이스로 구성되며, 컴포넌트는 객체와 객체간의 관계로 구성되고, 인터페이스는 제공인터페이스(Provided Interface)와 요구인터페이스(Required Interface)로 구분된다.

의사결정모델(Decision Model): 의사결정모델은 핵심 자산의 구성요소로서 가변성에 대한 정보가 명세된다. 의사결정모델은 가변점, 가변치, 영향(effect), 관련 활동(Attached Task)으로 구성된다[4]. 가변점은 가변성이 발생하는 지점이다[1]. 그림 2에서와 같이 가변점은 핵심자산의 구성요소에 따라 아키텍처에는 아키텍처 스타일, 아키텍처를 구성하는 컴포넌트와 컴포넌트들 간의 관계에서 발생하며[9], 컴포넌트 내부에서는 속성(Attribute Variability), 로직(Logic Variability), 워크플로우(Workflow Variability), 인터페이스(Interface Variability)에서 발생한다[10].

가변치는 가변점에서 해결할 수 있는 값으로 정의되며[1], 한 가변점에 여러 개의 가변치가 적용될 수 있다. 가변점의 가변치의 정의 가능 여부에 따라 'Open'과 'Closed'로 구분되며, 가변치가 정의된 경우, 가변점과 가변치의 연관관계에 따라 '대안'(Alternative)과 '선택'(Optional)으로 구분된다[11]. '선택'은 가변점의 휘쳐가 어플리케이션에 따라 사용되거나 되지 않는 경우이다. 대안은 가변점의 휘쳐가 여러 개 중에 적어도 하나

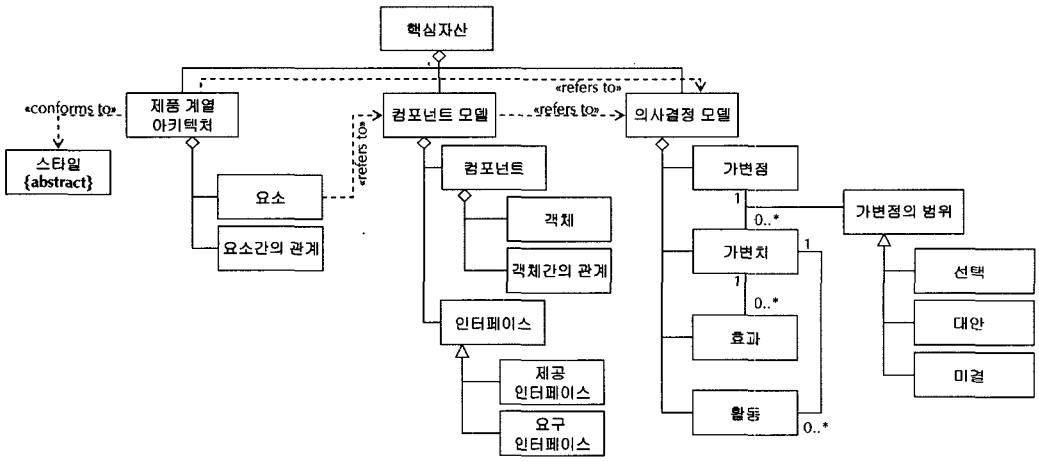

그림 1 핵심자산의 메타모델

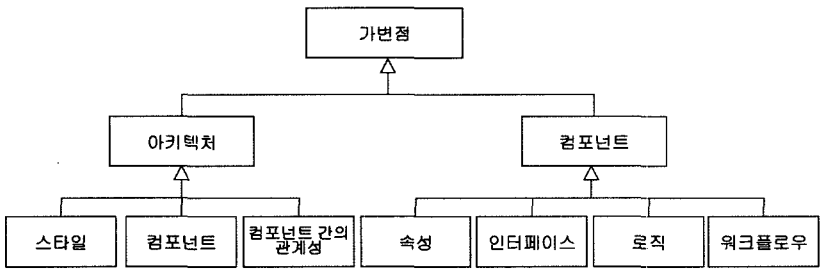

그림 2 가변점의 종류

는 사용되는 경우이다.

영향은 가변점이 다른 가변점과의 연관관계 또는 제약사항을 표현하기 위해 사용된다. 연관관계는 가변성의 범위 결정시 가변성간 의존관계로써 4가지 유형으로 식별하는데, 가변점간의 영향, 가변치간에 영향, 또는 가변점이 가변치에 주는 영향, 가변치가 가변점에 주는 영향이 있다.[12] 관련활동은 설정된 가변치에 대한 가변점을 해결하기 위한 활동들의 집합이다.

3.2 핵심자산 표현

정의된 핵심자산의 구성요소들은 모델에 표현되어야 하며 대표적인 명세는 공통성과 가변성 모델, PLA명세, 컴포넌트 명세, 의사결정모델로 분류할 수 있다.

공통성과 가변성 모델: 공통성 및 가변성 모델은 분석단계의 공통성과 가변성을 표현하는 모델로 유즈케이스 모델, 휘쳐 모델(feature model)을 사용할 수 있다[13,14]. 분석 모델의 공통성과 가변성은설계단계를 거쳐 PLA와 컴포넌트에서 표현되고, 특히 가변성은 의사결정 모델에서 상세히 명세 된다.

프로덕트 라인 아키텍처: PLA는 클래스 다이어그램, 패키지 다이어그램 등의 UML의 다양한 다이어그램으로 표현하고 아키텍처 가변성은 UML의 스테레오 타입과 같은 표기법으로 표현한다[9]. 프로덕트 라인 아키텍처에 표현되는 가변성은 다음과 같이 표현할 수 있다. 특히 가변성은 아키텍처 스타일 내에서 《type》 NAME {vpID = #, vType='Scope of variants'}와 같이 표기된다. 《type》은 아키텍처 가변성의 타입 즉 optional와 alternative로 나타내며, NAME은 컴포넌트나 아키텍처 스타일 이름, vpID는 가변점의 고유 ID로 숫자 또는 특수문자 표기할 수 있으며, vType은 가변치 범위로 'alt', 'opt', 'open'으로 명세한다.

컴포넌트 모델: 컴포넌트는 객체와 객체 관계로 구성되며, 이는 유즈케이스 모델, 구조적 모델, 동적 모델로 표현할 수 있다. 각 모델은 UML에서 유즈케이스 다이어그램, 클래스 다이어그램, 시퀀스 다이어그램으로 표현한다. 모델에서 가변성 표현은 아키텍처 스테레오 타입을 이용한다[1]. ≪vp-가변점 타입≫ *NAME* {vpID = #, vType='Scope of variants'}을 이용하여 표현한다. ≪vp-가변점 타입≫은 가변점 타입을 약자로 A(속성), L(로직), I(인터페이스), W(워크플로우)로 표현하며 이후 표현은 아키텍처 가변성 표현방법과 같다.

의사결정모델: 핵심자산은 공통성 및 가변성을포함해야 하며, 특화를 위해서는 가변성을 해결하는 설명이 상세히 명세되어 있어야 한다. 의사결정모델은 이러한 가변성 해결을 위한 설명을 명세한 모델로 가변점의 종류와 가변점과 가변치와 연관관계, 가변점간의 연관관계, 가변치간의 연관관계를 명세 해야 한다. 표 1은 의사결정모델로 핵심자산의 가변성을 가변점, 범위, 가변치, 효과, 관련활동으로 명세한다.

'가변점'열에서는 가변점을 표현하기 위해 가변점의 ID를 정의하여 명세한다. 그러나 실용적으로 표현하기 위해서는 ID 외에 가변점 이름, 가변점 종류, 가변점이 포함된 휘처 등의 설명을 위한 정보들이 추가될 수 있다. '범위'열에서는 가변치가 가변점에 적용되는 상황에 따라 'Alternative', 'Optional', 'Open'으로 표현한다. '가변치'열에서는 가변점에 알맞은 값, 즉 가변치를명세하는데, 가변치는 가변점의 종류에 따라 다양한 값으로 나타낼 수 있다. 예를 들어 아키텍처 스타일 내에서 특정 컴포넌트가 어떤 어플리케이션에서는 사용되지만 다른 어플리케이션에서는 사용되지 않을 경우 가변치는 컴포넌트가 된다. 그러나 컴포넌트 내에서 사용하는 객체의 속성이 다를 경우 가변치는 다르게 사용되는 속성이 된다.

'효과'열에서는 가변성의 관계를 명세한다. 가변성의 관계는 크게 가변점과 가변점 간의 연관관계, 한 가변점의 가변치와 다른 가변점의 가변치간의 연관관계로 구분될 수 있다[2]. 이러한 연관관계를 영향을 주는 가변점 또는 가변치와 영향을 받는 가변점 또는 가변치의 관계로 정리하며, 가변점을 VP_i, 가변치를 V_i로 할 때 표에서와 같이 ($VP_i \rightarrow VP_j$), ($V_i \rightarrow V_j$), ($VP_i.V_i \rightarrow V_j$), ($VP_i.V_i \rightarrow VP_j.V_j$,)로 표현한다[12].

관련활동은 가변점에 가변치를 채우는 작업으로 해당 가변점에 가변치를 채우는 활동과, 의존관계에 있는 다른 가변점의 가변치를 채우기 위한 활동으로 구분 될 수 있다. 해당 가변점에 가변치를 채우는 활동에는 사용되지 않는 가변치를 제거하거나, 새로운 가변치를 추가하거나, 또는 기존의 가변치를 수정하는 활동으로 구분

표 1 의사결정모델에서 가변성 스타일

가변점	범위(Scope)	가변치	효과(Effect)	관련활동(Attached Task)
VP_i	Alternative/ Optional/ Open	$variant_1$	• $VariabilityID_{source} \rightarrow VariabilityID_{target}$	• 가변점$_i$에 대한 활동 가변치$_b$ 제거, 가변치 추가, 가변치 수정…. • 연관관계의 가변점으로의 이동
		…	…	…
		$variant_n$	…	…
..	..	..	..	..

한다. 다른 가변점의 가변치를 채우기 위한 활동인 경우는 연관 관계에 있는 가변점으로 이동하여 이동한 가변점의 활동을 따르는 것으로 표현한다.

4. 프로세스 및 지침

본 장에서는 3장에서 정의한 핵심자산의 구성요소 및 표현 양식을 기반으로 핵심자산특화를 위한프로세스와 지침을 제안한다. 이 프로세스는 4단계로 구성되며 각 단계 다시 활동으로 나누어 지고, 각 단계와 활동 내에는 입/출력물과 지침이 정의된다.

4.1 단계 1. 의사결정해소모델 정의(Decision Resolution Model, DRM)

의사결정해소모델(Decision Resolution Model, DRM)이란 의사결정모델에 명세된 가변점에 대한 가능한 가변치 중에서 어플리케이션의 요구사항을 만족시키는 가변치만을 선택하여 명세한 어플리케이션에 종속된 가변치설정을 위한 모델이다.

현실적으로 핵심자산에 정의된 모든 가변점과 가변치는 목표 어플리케이션에 사용되지 않을 수 있다[4]. 이 단계에서는 어플리케이션에 적용 가능한 가변점과 가변치를 추출하여 DRM을 명세하는데 이를 위해 활동을 그림 4와 같이 정의한다.

4.1.1 활동 1-1: 중복영역 식별

활동 1-1은 핵심자산과 어플리케이션의 중복영역을 식별하는 단계이다. 중복영역이란 핵심자산의 휘쳐와 어플리케이션의 휘쳐간의 중복된 휘쳐들의 영역을 의미한다. 중복영역 식별은 목표 어플리케이션을 만들기 위해 핵심자산의 범위를 정의하는 것이다. 이 단계의 입력물은 C&V 모델과 어플리케이션 분석 모델이고 출력물은 중복영역의 휘쳐 목록이다.

이 활동은 두 개의 세부 활동으로 구성된다. 공통성과 가변성을 가지는 C&V모델의 휘쳐는 요구사항의 종류 및 적용되어야 할 핵심자산의 구성요소에 따라 그림 5와 같이 분류할 수 있다. 첫째 세부활동에서는 비기능적 휘쳐의 중복영역을 찾는다. 비기능적 요구사항은 품질속성에서 표현되고 각 품질속성의 상세내역은 요구사항에

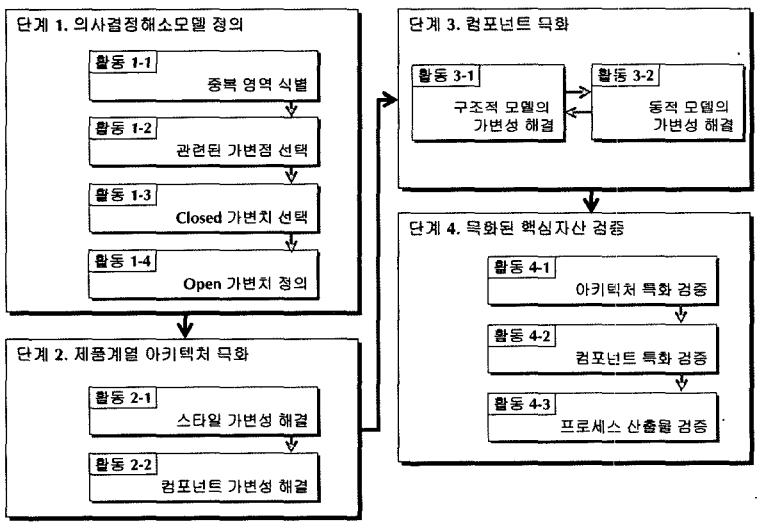

그림 3 특화 핵심 자산의 전체 프로세스

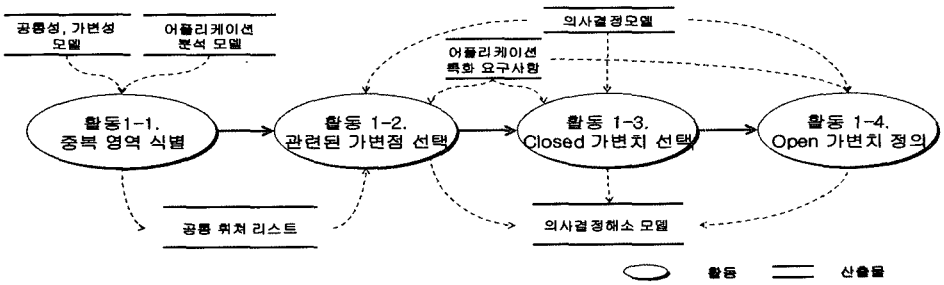

그림 4 단계 1의 활동

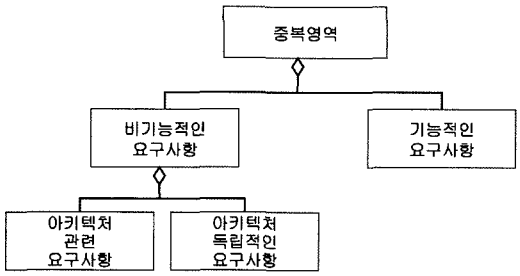

그림 5 중복영역 세부분류

명세된다. 이렇게 명세된 비기능적 요구사항은 아키텍처 설계에 적용되거나 컴포넌트 내부 설계에 적용된다. 두 번째 세부활동에서는 기능적 휘처의 중복영역을 찾는다. 기능성은 하부 시스템 단위와 유즈케이스 단위로 명세된다.

4.1.2 활동 1-2: 관련된 가변점 선택

프로덕트 라인에서 핵심자산은 어플리케이션에서 요구하는 모든 휘처를 제공하고, 어플리케이션은 개발된 핵심자산의 모든 휘처를 사용하는 것이 이상적이다. 그러나 핵심자산 개발단계에서 정의된 일부 휘처들은 어플리케이션에서 사용되지 않을 수 있고, 어플리케이션에서 요구하는 기능을 핵심자산에서 제공하지 않을 수 있다. 이러한 특징은 그림 6의 '어플리케이션에서 사용되지 않는 영역'과 '핵심자산에서 제공하지 않는 영역'으로 표현된다.

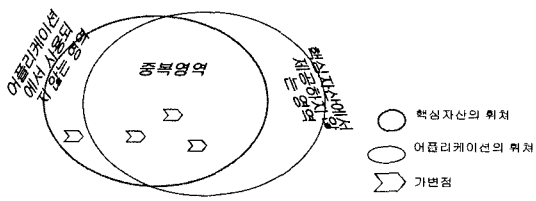

그림 6 핵심자산과 어플리케이션 간의 중복영역

입력물로는 중복영역 휘처 리스트, 의사결정모델, 어플리케이션 요구사항이고 출력물은 가변점의 목록을 가지는 초기 의사결정해소모델이다. 의사결정해소모델은 어플리케이션에 적용해야 하는 가변점, 가변치, 효과, 관련활동을 명세하는 모델이다. 본 활동에서는 어플리케이션에 적용 가능한 가변점을 식별하므로, 초기 의사결정해소모델이란 의사결정해소모델에서 가변점만이 식별되어 명세한 모델로 정의한다.

활동 1-2는 활동 1-1에서 식별한 중복영역에서 가변점들을 식별한다. 그림 6과 같이 어플리케이션에 사용되지 않는 휘처가 있기 때문에 의사결정모델에 명세된 가변점이 모두 어플리케이션에 사용 되지는 않는다. 즉 한 어플리케이션의 가변점은 전체 가변점의 부분 집합이 될 수 있다.

4.1.3 활동 1-3: Closed 가변치 선택

선택된 핵심자산의 가변점에 가변치를 설정하기 위해서는 적합한 가변치의 존재 여부를 확인해야 한다. 활동 1-3는 식별된 가변점에 대해 의사결정 모델에 정의된 가변치중 목표 어플리케이션에 적용 가능한 가변치를 선택하는 활동이다.

활동 1-2의 가변점이 선택된 초기 의사결정해소 모델이 이 단계의 입력물이고 이 활동을 통해 나온 출력물은 정제된 초기 의사결정해소모델이다.

의사결정 모델에 정의된 목표 어플리케이션의 요구사항에 맞는 가변치를 선택하면, 이에 대한 영향, 관련활동을 함께 초기 의사결정해소모델에 작성한다. 3.1절에서와 같이, 가변성은 크게 아키텍처의 가변성과 컴포넌트 내부에서 나타나는 가변성으로 구분 할 수 있다. 따라서 선택될 가변치는 가변점의 종류에 따라, 아키텍처에서의 스타일, 컴포넌트, 컴포넌트내부의 속성, 로직 등이 될 수 있다. 의사결정 해소 모델 작성시에는, 가변점의 종류에 따라 내용을 구분하여 작성하는 것이 단계 2와 단계 3에서 아키텍처 가변성 해결과 컴포넌트 내부 가변성 해결의 구분된 단계에서 참조하기 쉽다.

4.1.4 활동 1-4: Open 가변치 정의

가변점의 가변치가 Open일 경우, 어플리케이션을 위해 가변치를 새롭게 정의해야 한다. 따라서 그림 7과 같이 가변점은 의사결정모델에서 가변치를 선택하거나 새로운 가변치로 정의 된다. 활동 1-3에서 선택된 가변치와 초기 의사결정해소모델은 이 단계의 입력물이고 이 활동을 통해 나온 출력물은 최종 의사결정해소 모델이다.

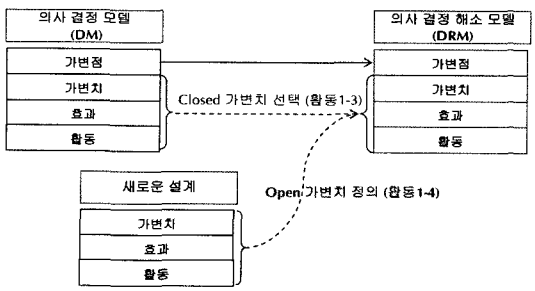

그림 7 가변치 정의

새로운 가변치의 정의는 가변치 자체의 정의와 정의된 가변치에 대한 영향, 관련활동을 새롭게 정의하는 것을 의미한다. 새롭게 정의된 가변치는 핵심자산의 휘처들과의 의존관계가 고려되어야 한다. 즉, 정의된 가변치

의 값이 다른 휘처들과 충돌되는지의 여부를 검증해야 한다. 이러한 검증에 따라 영향 및 관련활동을 정의한다.

4.2 단계 2. 제품계열 아키텍처 특화

단계 1로부터 생성된 의사결정 해소모델에는 아키텍처 가변성과 컴포넌트 내부 가변성을 함께 명세한다. 본 단계에서는 두 가변성 중 아키텍처 가변성을 해결하는 단계이다. 따라서 그림 8과 같이 본 활동의 입력물은 아키텍처에 관한 의사결정 해소 모델과 PLA이고 출력물은 특화된 프로덕트 아키텍처이다.

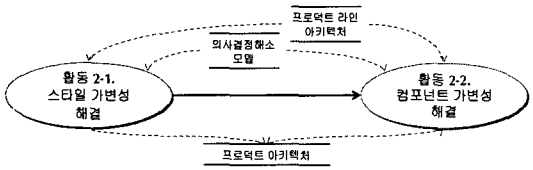

그림 8 단계 2의 활동

4.2.1 활동 2-1. 스타일 가변성 해결

PLA는 공통성과 가변성을 포함하는 아키텍처 스타일 집합으로 표현된다. 활동 2-1에서는 의사결정해소 모델을 참조하여 PLA 스타일 집합에서 어플리케이션에 사용되는 아키텍처 스타일을 적용한다. 입력물은 의사결정해소 모델과 PLA이고 출력물은 스타일 가변성이 해결된 목표 어플리케이션을 위한 프로덕트 아키텍처이다.

선택된 가변적인 아키텍처 스타일을 프로덕트 라인 아키텍처에 반영하기 위해서는 의사결정해소모델의 관련활동을 참조해야 한다. 이는 핵심자산 공학단계에서 프로덕트 라인 아키텍처에 대한 의사결정 모델을 설계할 때 관련된 아키텍처 스타일에 대한 관련활동을 이미 정의했기 때문이다. 이렇게 미리 정의된 관련활동에 의해 아키텍처 스타일 가변성을 해결 할 때 다음과 같은 사항을 고려할 수 있다.

- 아키텍처에 대한 스타일의 적용 범위는 다양하다. 예를 들어 계층 스타일은 아키텍처 전체에 걸쳐 나타날 수 있다. 그러나 파이프와 필터 스타일은 특정한 컴포넌트 간의 관계에서 나타나 아키텍처의 부분에 걸쳐 나타날 수 있다. 따라서 스타일이 적용되는 범위에 따라 다른 스타일과의 연관관계를 고려해야 한다.
- 아키텍처의 컴포넌트에 따라 여러 스타일에 적용되는 컴포넌트가 있고, 특정 스타일에만 적용되는 컴포넌트가 있을 수 있다. 그런데 사용되지 않는 아키텍처 스타일이 제거될 경우, 제거되는 스타일에 속하는 컴포넌트가 함께 제거될 수 있다. 이때 제거되는 컴포넌트가 다른 스타일에서 사용되는지 여부가 체크되어야 한다.

4.2.2 활동 2-2. 컴포넌트 가변성 해결

아키텍처 내에는 스타일 가변성 이외에 컴포넌트 자체가 어플리케이션에 따라 요구되거나 요구되지 않을 수 있다. 활동 2-2에서는 이러한 아키텍처 내의 컴포넌트에 대한 가변성을 해결한다. 이 단계를 통해 프로덕트 라인 아키텍처는 완전히 어플리케이션에 맞게 특화된다. 입력물은 의사결정해소 모델과 PLA이고 출력물은 컴포넌트 가변성이 해결된 최종 프로덕트 아키텍처이다.

아키텍처 수준에서의 컴포넌트 가변성은 새로운 컴포넌트가 추가되거나 또는 사용되지 않는 컴포넌트의 삭제되는 형태가 일반적이다. 이 경우 다음과 같은 사항을 고려할 수 있다.

- 컴포넌트의 추가 및 삭제에 대해 컴포넌트의 인터페이스의 연결 확인을 해야 한다. 새로운 컴포넌트가 추가되는 경우 기존의 컴포넌트에서 추가된 컴포넌트의 기능을 사용하는지의 여부, 추가된 컴포넌트가 기존의 컴포넌트의 기능을 사용하는지의 여부가 확인되어야 하며, 컴포넌트가 삭제되는 경우 삭제될 컴포넌트의 기능을 사용하는 다른 컴포넌트의 존재 여부를 확인해야 한다.
- 추가된 컴포넌트가 이미 존재하는 컴포넌트 내부의 가변성에 주는 영향 등이 체크 되어야 한다. 추가된 컴포넌트의 휘처가 기존의 컴포넌트의 가변치와 비즈니스 로직의 측면에서 또는 품질 속성 측면에서 충돌이 없는지를 고려해야 한다.

4.3 단계 3. 컴포넌트 특화

3.1절에서 명시한 가변점의 두 분류 중 아키텍처 가변성은 단계 2에서 해결하였고, 본 단계에서는 의사결정해소모델에서 명세한 컴포넌트 내부 가변성을 해결한다. 입력물은 핵심자산의 컴포넌트 모델과 의사결정해소 모델이며 출력물은 어플리케이션 요구사항에 맞게 특화된 구조적 모델과 동적 모델이다. 그림 9는 입출력물을 이용한 컴포넌트 내부 가변성 해결에 대한 프로세스이다.

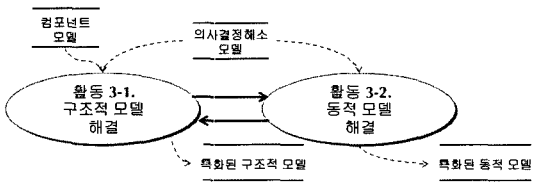

그림 9 단계 3의 활동

소프트웨어 설계 모델은 구조적 모델과 동적 모델이 일반적이다. 이는 핵심자산의 컴포넌트 설계모델도 구조적 모델과 동적 모델로 표현될 수 있다. 컴포넌트 내부 가변성 또한 구조적 모델과 동적 모델로 나타낼 수 있다. 따라서 본 단계에서는 속성, 워크플로우, 인터페이스, 로직의 컴포넌트 내부 가변성을 해결하기 위해서 구

조적 모델에서의 가변성과 동적 모델에서의 가변성을 해결한다.

4.3.1 활동 3-1. 구조적 모델 해결

핵심자산의 구조적 모델에는 어플리케이션에서 사용될 객체들의 구조와 함께 객체에 대한 가변점과 가변치가 명시되어 있다. 예를 들어 클래스 다이어그램에서는 가변적인 클래스, 또는 클래스의 가변적인 속성을 스테레오 타입을 이용하여 표현할 수 있다. 그리고 구조적 모델에 표현된 가변점에 대한 상세한 정보는 의사결정모델에 명시된다. 입력물은 의사결정해소모델과 컴포넌트 모델이고 출력물은 컴포넌트의 특화된 구조적 모델이다.

구조적 모델의 가변성 해결을 위해서는 의사결정해소모델에 명시된 모든 가변점들 중 구조적 모델에 표현된 가변점들을 찾고 의사결정해소모델에 정의된 가변치의 '관련활동'에 따라 구조적 모델에 사용될 가변치를 추가하거나 사용되지 않을 가변치를 삭제한다. 컴포넌트 내부 가변성 중 특히, 속성 가변성은 오퍼레이션이나 워크플로우 등의 다른 가변점에서 사용될 수 있다. 따라서 객체의 속성이 추가/수정/삭제되었을 경우 관련된 오퍼레이션이나 워크플로우를 확인하여, 속성 변화에 따른 일관성 유지해야 한다. 이처럼 가변성간의 영향은 구조적 모델 해결을 위해서도 적용되지만 동적 모델 해결을 위해서도 중요한 영향을 주게 된다.

4.3.2 활동 3-2. 동적 모델 해결

핵심자산의 동적 모델에서 가변점과 가변치는 객체와 객체간의 메시지 전달로 통해 표현된다. 동적 모델은 순차 다이어그램과 협동 다이어그램으로 표현 할 수 있다. 이러한 모델은 특히 컴포넌트 내부 가변성 중 워크플로우 가변성을 나타내기에 용이하다. 입력물로는 의사결정해소 모델과 컴포넌트 모델이며 출력물은 어플리케이션의 요구사항을 만족하는 특화된 동적 모델이다.

워크플로우 가변성은 마이크로 워크플로우(micro workflow) 같이 컴포넌트 내부에서 워크플로우 가변성이 발생하고 매크로 워크플로우(macro workflow)는 컴포넌트가 다른 컴포넌트에 영향을 주어 워크플로우 가변성이 발생한다. 매크로 워크플로우는 컴포넌트 간에 관계를 변화시킨다. 컴포넌트 관계가 수정되면 동적 모델과 구조적 모델이 변화 되어 아키텍처에도 수정을 요청하고 아키텍처 변화에도 영향을 준다.

단계 4. 특화된 핵심자산검증

단계 4는 어플리케이션 요구사항을 만족하는 특화된 핵심자산을 검증하는 단계이다. 특화된 핵심자산을 검증하는 활동은 가변성의 해결에 대한 검증을 필요로 하므로 순수하게 한 어플리케이션의 요구사항으로 개발된 단일 시스템의 검증보다 어렵다. 본 논문에서는 특화된 핵심자산의 검증단계를 제안하지만 상세 단계와 지침은 논문 연구 범위에 포함하지 않는다. 그러나 가변성 해결 측면에서의 특화된 핵심자산 검증을 위한 대표적인 체크리스트를 제시한다.

4.3.3 활동 4-1. 특화된 아키텍처 검증

특화된 핵심자산에서 아키텍처 가변성 해결여부를 검증하기 위한 체크 리스트는 다음과 같다.

- 선택된 품질요구사항의 충돌 발생 여부 검증
- 추가되는 아키텍처 스타일과 요구사항 준수성 검증
- 아키텍처 스타일이 제거 되면 제거되는 스타일에 속하는 컴포넌트가 함께 제거되어야 하는지를 검증
- 추가되는 컴포넌트가 아키텍처 스타일 변동에 영향을 주는지 검증
- 추가되는 컴포넌트의 요구사항 준수성 검증
- 사용되지 않는 컴포넌트의 아키텍처 내 존속성 검증 - 사용되지 않은 컴포넌트는 아키텍처 내에 포함되지 말아야 한다.
- 새로 추가된 컴포넌트와의 상호 운용성 검증
- 컴포넌트가 삭제되었을 경우의 아키텍처의 견고성 검증
- 의사결정모델에서 명세되지 못한 가변성이 단계 2을 통해 추가 발견된 경우, 아키텍처 스타일간의 의존성 및 컴포넌트 연관 관계를 검증

4.3.4 활동 4-2. 특화된 컴포넌트 검증

특화 핵심자산에서 컴포넌트 내부 가변성을 해결하는 특화 컴포넌트 검증에 필요한 항목을 제시한다.

- 컴포넌트 가변점과 가변치의 요구사항 준수성 검증 - 특히 Open 가변점의 새로 정의된 가변치가 어플리케이션의 요구사항을 만족하는지 검증
- 정의된 가변치와 가변치, 가변점과 가변점, 가변치와 가변점 간에 충돌이 발생여부 검증
- 아키텍처의 컴포넌트 가변성과 연관된 다른 컴포넌트 내부의 가변성간의 의존성 검증
- 컴포넌트 내부의 한 가변성에 대한 연관된 다른 모든 가변성과의 의존성 검증
- 핵심자산의 가변성이 의사결정모델에 명세되지 못하여 단계 3을 통해 추가 발견된 경우, 컴포넌트 내부 가변성간의 의존성 및 기존 가변치와 연관을 검증

4.3.5 활동 4-3. 프로세스의 추적성 검증 맵

각 활동간의 산출물간의 일관성을 검증하기 위해 추적성 맵을 적용하는 단계이다. 그림 10에서는 산출물간의 상호 연관관계를 표현한다.

산출물은 다시 세부 산출물로 나누어 세부 산출물간의 관계로 표현한다. 산출물의 모든 구성 요소가 영향을 주는 것이 아니기 때문에 세부 산출물로 나누어 세부 산출물간의 관계로 표현한다. 각 산출물간의 추적성을 알아 보기 위해 주요 산출물에 대한 추적관계에 대한 설명을 표 2와 같이 정리한다.

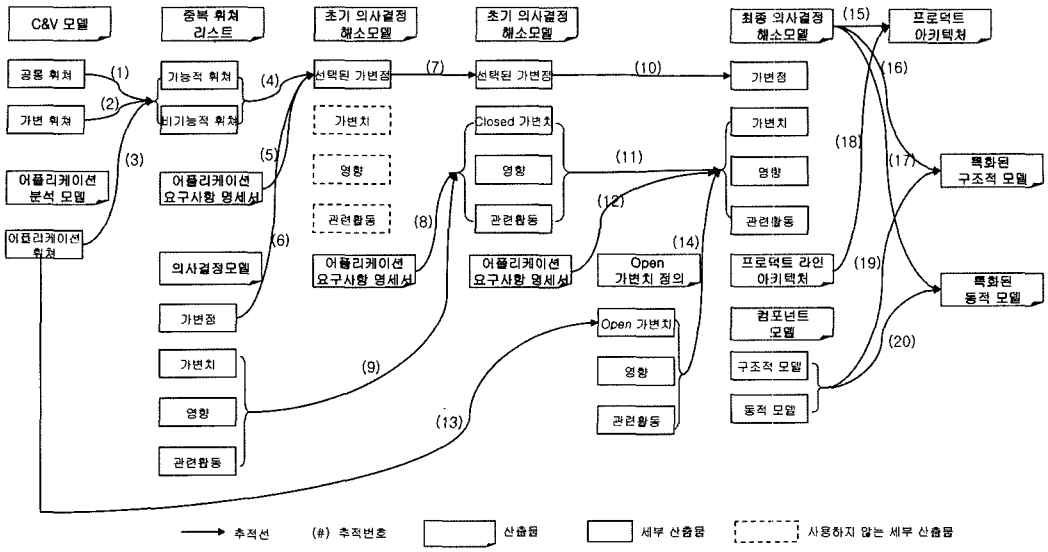

그림 10 프로세스의 추적성 검증 맵

표 2 산출물간 추적성

추적번호	입력물	출력물	추적설명
6	의사결정모델의 가변점	초기 의사결정해소모델의 가변점	의사결정해소모델의 가변점을 정의하기 위해 핵심자산의 의사결정모델의 가변과 중복휘쳐리스트를 적용한다.
9	의사결정모델의 가변치, 영향, 관련활동	초기 의사결정해소모델의 Closed 가변치, 영향, 관련활동	초기 의사결정해소모델의 Closed 가변치를 정의하기 위해 의사결정모델의 가변치, 영향, 관련활동과 어플리케이션 요구사항명세서를 이용한다.
13	어플리케이션 휘쳐	초기 의사결정해소모델의 Open 가변치, 영향, 관련활동	정의 되어 있지 않는 Open 가변치를 정의하기 위해 어플리케이션 휘쳐와 Closed 가변치를 적용한다.
14	초기 의사결정해소 모델의 정의된 Open 가변치, 영향, 관련활동	최종 의사결정해소모델의 가변치, 영향, 관련활동	앞 활동에서 정의된 가변점과 Closed 가변치, Open 가변치, 어플리케이션 요구사항을 적용하여 최종 의사결정해소모델을 정의한다.

5. 사례연구

제안된 프로세스의 적용을 위해 대여 도메인에서 자동차대여 시스템과 도서대여 시스템에 대한 핵심자산 정의 및 도서 대여 시스템 특화 사례연구를 다음과 같이 수행하였다. 미리 정의된 핵심자산을 이용하여 특화 프로세스를 적용한다.

자동차대여 및 도서대여의 대여 도메인에서는 공통적인 기능 모듈을 회원관리, 대여물품(자동차, 도서)관리, 대여관리, 회계관리로 구분한다. 본 대여 도메인에서는 데이타가 중앙 관리되거나 데이타의 분산관리 될 수 있다. 또한 각 모듈에 대해서도 각 적용되는 시스템에 따라 사용되는 정보 및 기능의 흐름 등에 가변성을 가진다.

5.1 핵심자산 특화

5.1.1 단계 1. DRM 정의

DRM을 표현하기 위해 먼저 컴포넌트 내부 가변성이 표현된 핵심자산 구성요소인 의사결정모델을 표 3에서 표현하였다. 이는 대여 업무 가변성 중 가장 대표적인 가변점 3개를 선택하여 의사결정모델을 표현했다. 이에 따라 가변성은 4가지 타입중 속성, 워크플로우 가변성으로 표현되며 가변성의 범위는 '대안'과 '선택'이 사용되었다. 또한 컴포넌트 내부 가변성은 아키텍처 가변성 의사결정모델에 없는 가변성 타입을 추가하여 의사결정모델에 표현한다.

도서대여 어플리케이션 요구사항과 표 3의 의사결정모델을 이용하여 의사결정해소모델을 정의한다. 표 4에서는 핵심자산의 의사결정모델중 대표적인 컴포넌트 내부 가변성의 의사결정해소모델을 표현하였다. DRM은 DM과 달리 목표 어플리케이션에필요한 가변치와 효과, 관련활동이 명세되었다.

5.1.2 단계 2. 제품계열 아키텍처 특화

프로덕트 라인 아키텍처는 자동차 대여와 도서 대여 도메인을 통해 작성된다. 두 도메인의 공통성과 가변성을 계층 스타일, 파이프와 필터 스타일, 공유데이타 스

표 3 컴포넌트 내부 가변성의 의사결정모델

ID	가변점	타입(Type)	범위(Scope)	가변치	효과(Effect)	관련활동(Attached Task)
11	rentalPeriod	속성	대안 (alternative)	rentalPeriod = 7일	–	rentalPeriod 속성에 7일 설정
				rentalPeriod = 고객 요구기간	–	rentalPeriod 속성에 고객 요구 기간으로 설정
12	dayPenalty	속성	대안 (alternative)	dayPenalty = 500	vpID=12 → vpID= 15	dayPenalty 속성에 500으로 설정
				dayPenalty = 대여료/10	vpID=12 → vpID=1 5	dayPenalty 속성에 (대여료/10)
...	...	...	...	...	...	...
17	대여	워크플로우	선택 (Optional)	고객 정보 조회 → 대여 물품 검색 → 연체 확인 → 보험 가입 → 대여승인	vpID=17 → vpID=14	1. '연체 확인' 업무 종료후 '보험 가입' 업무 실행(호출) 2. insuraceID 생성
				고객 정보 조회 → 대여 물품 검색 → 연체 확인 → 대여승인	vpID=17 → vpID=14	1. '연체 확인' 업무 종료후 '대여 승인' 업무 실행 2. insuraceID 속성 제거

표 4 컴포넌트 내부 가변성의 의사결정해소모델

ID	가변점	타입(Type)	범위(Scope)	가변치	효과(Effect)	관련활동(Attached Task)
11	rentalPeriod	속성	대안 (alternative)	rentalPeriod= 7일	–	rentalPeriod 속성에 7일 설정
12	dayPenalty	속성	대안 (alternative)	dayPenalty= 500	vpID=12 → vpID= 15	dayPenalty 속성에 500으로 설정
...	...	...	...	...	...	...
17	대여	워크플로우	선택 (Optional)	고객 정보 조회→ 대여 물품 검색 → 연체 확인→ 대여승인	vpID=17 → vpID=14	1. '연체 확인' 업무 종료후 '대여 승인' 업무 실행 2. insuraceID 속성 제거

타일에 적용하여 표현하며, 특히 가변성은 아키텍처 스타일 내에서 《optional》을 이용하여 표현한다. 그림 11에서는 프로덕트 라인 아키텍처와 특화 프로덕트 아키텍처를 같이 표현하였다. 투명으로 표현 된 그림은 프로덕트 아키텍처에서 특화 되어 사용하지 않는 아키텍처 구성요소를 표현한 것이다.

핵심 자산의 대여 프로덕트 라인 아키텍처에서 DRM을 참조하여 도서관 어플리케이션에 요구사항을 만족하는 특화된 프로덕트 아키텍처를 표현한다. 그림 11에서는 DRM을 통하여서 스타일 가변성이 해결되어 특화된

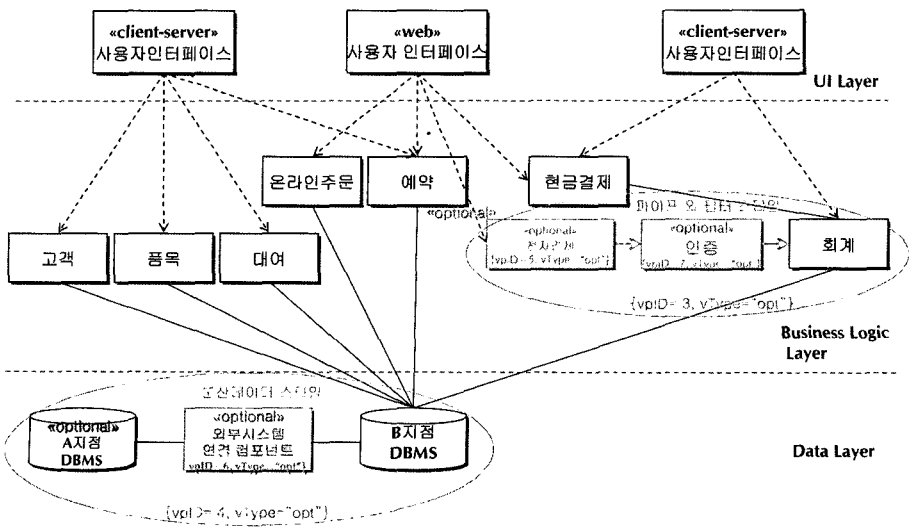

그림 11 특화된 아키텍처

아키텍처이다. 특화된 아키텍처에서는 전자 결제를 위한 파이프와 필터 스타일을 제거 하였고 외부 시스템 연결 컴포넌트를 사용하지 않는다.

5.1.3 단계 3. 컴포넌트 특화

정적 모델은 클래스 다이어그램을 이용하여 UML에서 제공하는 스테레오타입(Streotype)을 사용하여 가변성 타입과 가변점ID, 가변치 범위를 설정한다. 컴포넌트 내부 가변성을 표현한 구조적 모델에서 DRM을 이용하여 컴포넌트 내부 가변성을 해결한다. 컴포넌트 내부 가변성의 특화를 위해 도서 대여 어플리케이션의요구사항을 만족하는 특화된 컴포넌트를 구조적 모델로 표현한다. 그림 12에서는 굵게 표현한 가변성은 의사결정해소 모델을 통하여 해결된 가변성이다.

동적 모델은 대표적인 UML에서 순차 다이어그램을 이용하여 표현한다. 순차 다이어그램을 이용하여 대표 기능인 대여 업무를 표현하며 업무 흐름에서 표현 될 수 있는 가변성을 표현한다. 핵심자산의 동적 모델은 DRM을 참조하여 특화된 동적 모델을 만든다. 그림 13에서는 도서 대여 어플리케이션의 요구사항을 만족하여 특화된 순차 다이어그램이다. 투명으로 표현된 가변성은

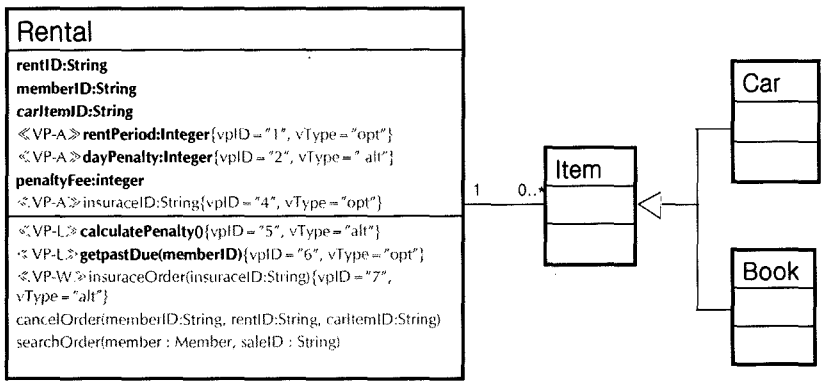

그림 12 특화된 구조적 모델

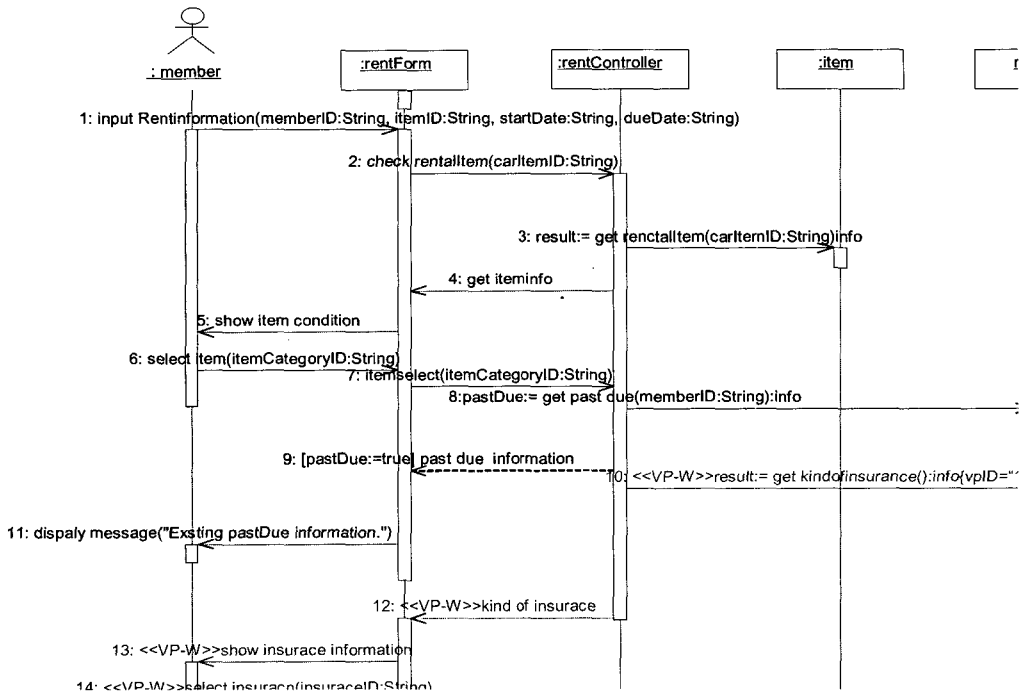

그림 13 특화된 동적 모델의 부분

의사결정해소 모델을 통하여 해결된 가변성이다.

핵심자산 동적 모델에서 도서 대여 요구사항을 만족하기 위해 대여 워크플로우 가변성과 연체료 로직 가변성은 해결되어 그림 13와 같이 표현한다. 대여 워크플로우 가변성은 보험 가입 업무를 삭제 하였고 연체료 로직 가변성은 도서 대여 연체료에 로직을 적용하여 가변성을 해결한다.

6. 향후 연구 및 결론

프로덕트 라인 공학은 핵심자산을 이용하여 여러 어플리케이션을 생성하여 재사용성과 개발 비용을 절감할 수 있는 개발 방법론이다. 핵심자산을 이용하여 어플리케이션을 만들기 위해서는 가변성을 정의하고 목표 어플리케이션을 만들기 위해 가변점에 가변치를 설정한다. 본 논문에서는 이러한 방법론을 이용하여 어플리케이션 만들기 위해 구체적 수준의 핵심자산 가변성 표현과 이를 이용하여 가변성 해결을 위한 체계적인 프로세스를 제안하였다.

핵심자산은 모델, 코드, 바이너리코드로 나타날 수 있다. 즉, 특화는 핵심자산의 추상화 수준에 따라 프로세스의 활동이 달라질 수 있다. 본 논문에서는 핵심자산을 모델 수준으로 정의하였다. 모델 수준인 핵심자산은 Model Driven Architecture(MDA)의 모델 변환 기법에 적용하여 프로덕트 라인에서의 애플리케이션 자동 생성의 기반을 마련할 수 있다. 즉, 모델 수준의 핵심자산은 특정 언어로 표현하여 코드화하고 만든 코드를 바이너리 코드로 바꾸어 제품을 만들게 된다. 여기에 툴을 이용하면 이러한 과정이 자동으로 해결할 수 있다. 제시한 프로세스를 자동화 툴에 적용하기 위해서 관련활동들 간의 산출물은 XMI를 사용하여 표현한다.

본 논문에서 제시한 특화 프로세스와 지침을 통해 프로덕트 라인 방법론에 적용하여 어플리케이션을 이상적으로 만들 수 있으며, 제시된 프로세스를 자동화 도구에 적용할 수 있는 기반을 마련할 수 있다.

참 고 문 헌

[1] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.

[2] Sinnema, M., Deelstra, S., Nijhuis, J., and Bosch, J., "COVAMOF: A Framework for Modeling Variability in Software Product Families," *Proceedings of SPLC 2004, Vol. 3154, LNCS 3154*, 2004.

[3] Deelstra S., Sinnema M., and Bosch J., "A Product Derivation Framework for Software Product Families," *Proceedings of PFE, LNCS 3014*, 2004.

[4] Kim, S., Chang S., and Chang C., "A Systematic Method to Instantiate Core Assets in Product Line Engineering," *Proceedings of APSEC 2004*, pp.92-98, 2004.

[5] 라현정, 장수호, 김수동, "제품 계열 공학에서의 산출물간 추적성 기법," 한국 정보과학회 논문지 소프트웨어 및 응용, Vol.32, No.4, pp.237-246, 2005.

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

[7] Woods, E., "Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report," *proceeding of EWSA 2004, LNCS 3047*, Springer-Verlag Berlin Heidelberg, 2004.

[8] Ceron, R., Arciniegas, J., Ruiz, J., Cuenas, J., Bermejo, J., and Capilla, R., "architectural medeling in Product Family Context," *proceeding of EWSA, LNCS3047*, Springer-Verlag Berlin Heidelberg, 2004.

[9] 장수호, 라현정, 김수동, "제품계열 아키텍처의 실용적 설계기법", 한국정보과학회 논문지 소프트웨어 및 응용, Vol. 32, No. 3, pp.163-172, 2005.

[10] 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.

[11] Jacobson, I., Griss, M., and Jonsson, P., *Software Reuse*, Addison Wesley, 1997.

[12] Jaring, M., and Bosch, J., "Variability Dependencies in Product Family Engineering," *PFE2003, LNCS 3014*, pp.81-97, 2004.

[13] Kang, K., Kim, S., Lee, J., Kim, K., Shin, E. and Huh, M., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, vol.5, p.143- p.168, 1998.

[14] Hassan Gomaa, *Designing Software Product Lines with UML*, Addison Wesley, 2004.

강 현 구

2004년 서경대학교 컴퓨터공학과 공학사
2004년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 제품계열 공학(PLE), 소프트웨어 테스팅

장 수 호

2003년 숭실대학교 컴퓨터학부 공학사 2005년 숭실대학교 컴퓨터학과 공학석사 2005년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 제품계열 공학(PLE), 소프트웨어 아키텍쳐, 모델 기반 아키텍처(MDA)

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA)