정형 명세를 이용한 제품계열 아키텍처의 인스턴스화 기법

(A Method for Instantiating Product Line Architecture using Formal Specifications)

신 숙 경 † 허 진 선 †† 김 수 동 †††
(Suk Kyung Shin) (Jin Sun Her) (Soo Dong Kim)

요 약 제품계열 공학(Product Line Engineering, PLE)은 최근 각광받고 있는 효율적인 소프트웨어 재사용 접근 방법 중 하나로 핵심자산을 인스턴스화(Instantiate)하여 여러 어플리케이션을 개발한다. 핵심자산(Core Asset)의 구성요소로는 제품계열 아키텍처(Product Line Architecture, PLA), 컴포넌트, 의사결정모델(Decision Model)이 있다. 이런 요소 중, PLA는 핵심자산의 전체적인 구조를 정의하고 있어 가장 핵심적인 요소라 할 수 있다. 현재 많은 PLE 방법론들이 소개되어 있으나, PLA의 구체적인 구성요소와 어플리케이션을 만들기 위해 PLA를 인스턴스화하기 위한 체계적인 기법이 미비하다. PLA의 구성요소를 명확히 정의하고 인스턴스화 프로세스를 상세히 정의하기 위해 정형명세가 효과적으로 사용될 수 있다. 본 논문에서는 먼저 PLA의 메타모델을 제시하고 PLA를 정형명세 언어인 Object-Z로 명세하는 방법을 제시한다. 또한 정형명세를 이용한 인스턴스화 규칙을 제안하며, 이런 규칙은 PLA를 인스턴스화하기 위한 제약사항을 상세히 정의하고 있다. 제안된 정형명세를 적용함으로써, PLA의 인스턴스화는 상세하고 명확하게 수행될 수 있어 고품질의 소프트웨어를 생산할 수 있다.

키워드 : 제품계열 공학, 정형명세, Object-Z, 제품계열 아키텍처, 인스턴스화

Abstract Product line engineering (PLE) is one of the recent and effective reuse approaches that enables developing a number of applications by instantiating a core asset. Elements of a core asset are product line architecture (PLA), component, and decision model. Among these elements, PLA is the key element since it defines the overall structure of the core asset. Although numerous PLE methodologies have been introduced, it is still unclear what should be the elements of a PLA and how to systematically instantiate it for specific applications. Formal specifications can play a key role in defining detailed and precise instantiation process. In this paper, we first present a meta model of PLA and show how to specify PLA in a formal language, Object-Z. Then, we propose instantiation rules using formal specification and those rules precisely define constraints for instantiating PLA. By applying the proposed formal specification, we believe PLA instantiation can be carried out precisely and correctly, yielding high quality software development.

Key words : Product Line Engineering, Formal Specification, Object-Z, Product Line Architecture, Instantiation

1. 서 론

PLE는 최근 각광받고 있는 효율적인 소프트웨어 재사용 접근 방법 중 하나이다. PLE는 핵심자산 공학(Core Asset Engineering)과 어플리케이션 공학(Application Engineering)으로 구성된다[1-3]. 핵심자산 공학에서는 패밀리 멤버간의 공통성과 가변성을 분석하여 이를 핵심자산으로 개발하고, 어플리케이션 공학에서는 의사결정 해소모델(Decision Resolution Model)을 기반으로 이런 핵심자산을 인스턴스화하여 여러 어플리케이션을 개발한다. 핵심자산은 제품계열에 속한 여러 어플리케이션을 개발하기 위한 기초가 되는 모든 자산을 포함하고 있으며 그 대표적인 자산으로는 PLA, 컴포넌트, 의사결

· 본 연구는 한국과학재단 특정기초연구(R01-2005-000-11215-0)지원으로 수행되었음
† 정 회 원 : 한국학술진흥재단 학술정보팀장
skshin@krf.or.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
jsher@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
논문접수 : 2005년 11월 15일
심사완료 : 2006년 11월 8일

정모델 등이 있다[1,4]. PLA는 핵심자산이 가지고 있는 컴포넌트와 컴포넌트 사이의 구조와 컴포넌트가 제공하는 인터페이스 등에 대해 정의하고 있어 핵심자산의 핵심적인 요소라 할 수 있다. 그러나, PLA의 상세한 구성요소에 대한 정의와 PLA가 어플리케이션 공학 과정에서 어떻게 인스턴스화되는지의 체계적인 절차는 미비한 상태이다.

본 논문에서는 정형명세 기법을 적용하여 PLA의 상세한 구성요소를 정의하고 체계화된 인스턴스화 기법을 제안한다. 우선 PLA의 메타모델을 제시하고 PLA를 정형명세 언어인 Object-Z로 명세하는 방법을 제시한다[5,6]. 또한 PLA를 인스턴스화하기 위한 제약사항을 상세히 정의한 인스턴스화 규칙을 정형명세를 이용하여 제안한다.

PLA에 대한 정형명세는 컴포넌트와 컴포넌트간의 관계를 적절하게 표현할 수 있는 언어를 사용해야 한다. 컴포넌트는 클래스 단위로 구성되어 있으므로 클래스를 명세하는데 적합한 Object-Z 언어를 사용하는 것이 적합하다. Object-Z 언어는 Z 언어를 확장하여 클래스를 명세하며, 클래스를 포함하는 클래스를 명세할 수 있다. 그러나 Object-Z는 인터페이스와 컴포넌트의 분리를 명확히 보여줄 수 없으며 컴포넌트 가변성을 표현하기 어려우므로 이를 보완하여 컴포넌트를 정의한 [7]를 활용한다.

정형명세 기법을 이용한 인스턴스화 기법의 정의는 명세 기법의 어려움과 복잡성으로 인해 부담을 가져올 수도 있지만 다음의 여러 장점이 있다. 첫째, 현재의 인스턴스화 프로세스가 덜 체계적이고 아직 미성숙한 단계에 있기 때문에, 정형적인 접근을 통해 인스턴스화 규칙을 규명함으로써 체계적인 인스턴스화 기법을 제안할 수 있다. 둘째, 특정 어플리케이션에 맞게 정의된 의사결정 해소모델의 내용을 PLA에 상세하고 정확하게 반영할 수 있어 최종 어플리케이션 아키텍처의 품질을 높일 수 있다. 셋째, PLA의 정형명세와 PLA 인스턴스화를 위한 변환 규칙을 통해 아키텍처 인스턴스화 절차를 자동화할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 인스턴스화 기법에 대한 관련연구를 살펴보고, 3장에서 Object-Z를 이용한 PLA의 정형명세를 제시한다. 4장에서는 특정 어플리케이션의 요구사항에 맞게 의사결정모델을 특화한 의사결정 해소모델과 인스턴스화 아키텍처에 대한 정형명세를 제시하고 5장에서는 의사결정 해소모델을 기반으로 PLA를 인스턴스화하기 위한 규칙을 정의한다. 6장에서는 본 논문에서 제안한 정형명세 기반 인스턴스화 규칙을 적용한 실예를 보여주고, 7장에서는 본 논문에서 제안한 인스턴스화 기법에 대한 검증을 수행한다.

2. 관련 연구

PuLSE(Product Line Software Engineering)는 IESE에 의해 개발된 제품계열 개발 프로세스이다[8]. PuLSE는 배포 단계(Deployment Phases), 기술적 컴포넌트(Technical Components), 지원 컴포넌트(Support Components)의 세 개의 하부 요소로 구성된다. 배포 단계는 재사용 가능한 자산과 제품을 생산하기 위한 단계들을 정의하고 있으며 관련된 기술적 컴포넌트들을 사용한다. 기술적 컴포넌트로는 PuLSE-BC, PuLSE-Eco, PuLSE-CDA, PuLSE-DSSA, PuLSE-I, 그리고 PuLSE-EM이 있다. 이 중에 PuLSE-I는 어플리케이션을 개발하기 위한 기술적 컴포넌트로 프로덕트라인 모델과 참조 아키텍처를 인스턴스화하기 위한 활동들을 포함하고 있다. 프로덕트라인 모델은 핵심자산을 나타내기 위한 스토리보드와 객체모델을 포함하고 있으며 의사결정 모델과 어플리케이션 종속적인 특징들을 이용하여 가변성이 해결된다. PuLSE-I는 인스턴스화를 위한 활동들을 제시하고 있으나 활동을 수행하기 위한 상세하고 체계화된 지침이 부족하다. 뿐만 아니다, 정확성과 정밀성을 보여주기 위한 정형적인 접근이 부족하다.

KobrA는 UML을 이용한 컴포넌트기반의 제품계열 공학 방법론이다[2]. KobrA의 두 가지 주요한 인스턴스화 활동은 컨텍스트 실현화 인스턴스화(Context Realization Instantiation)와 명세와 실현화 인스턴스화(Specification and Realization Instantiation)이다. 컨텍스트 실현화 인스턴스화 활동에서는 프레임웍에 의해 제공되는 중복된 휘쳐(feature)를 분석한다. 그리고 컨텍스트 실현화는 어플리케이션에 적용되지 않는 모든 휘쳐를 지우고, 의사결정모델을 의사결정 해소 모델로 바꿈으로써 변경함으로써 인스턴스화된다. 명세와 실현화 인스턴스화 활동에서는 의사결정모델을 해결하고 필요없는 휘쳐를 지움으로써 프레임웍 컨테인먼트 트리(Containment Tree)내의 컴포넌트를 반복적으로 인스턴스화한다. KobrA는 비교적 상위 수준에서 인스턴스화 프로세스를 정의하고 있어, 그 인스턴스화 과정의 정확성이나 정밀성을 검증하기 어렵다.

Bosch의 연구에서 제시하고 있는 범용적인 어플리케이션 유도(Derivation) 프로세스는 초기 단계(Initial Phase)와 반복 단계(Iteration Phase)의 두 개의 주요 단계들로 구성된다[9]. 초기 단계에서는 프로덕트 패밀리의 자산내의 공유할 수 있는 자산들을 조립하거나 잘 조화를 이루는 기존의 형상을 선택함으로써 첫 번째 형상(Configuration)을 생성한다. 이렇게 생성된 초기 형

상은 요구하는 기능을 어느 정도까지 충족하는지 확인한다. 만약 생성된 형상이 완료되지 못했다고 사료될 때에는 반복 단계를 수행한다. 반복 단계에서는 초기 형상이 요구하는 기능을 만족할 때까지 수정 작업을 반복적으로 수행한다. Bosch의 연구는 개념적인 수준에서 어플리케이션 공학에 대한 전체적인 절차를 제안하고 있다. 따라서, 인스턴스화 단계에 대한 정확성이나 정밀성을 검증하기 위해서 더욱 체계화되고 상세화되어야 한다.

3. 제품계열 아키텍처 명세

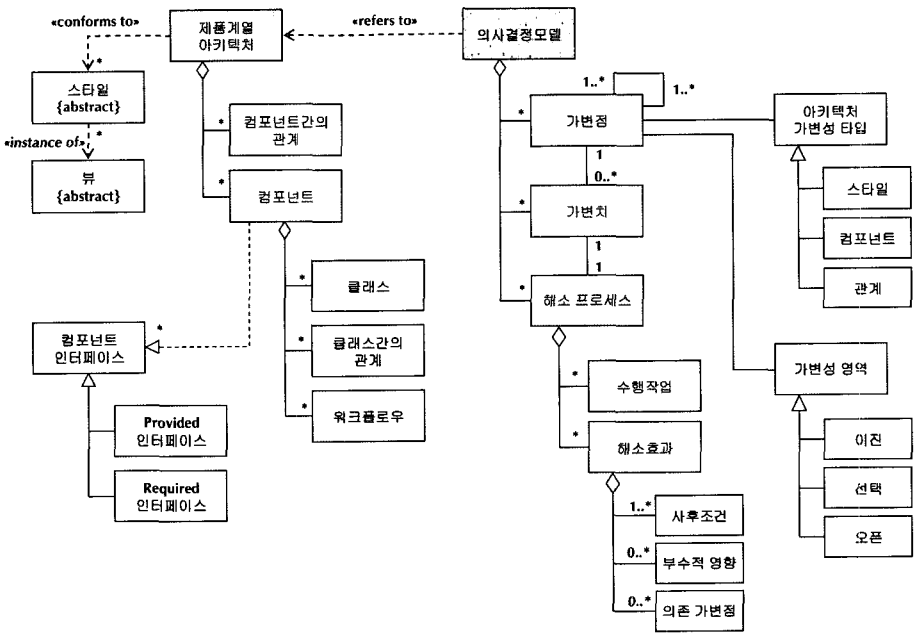
본 장에서는 핵심자산에 대한 대표적인 연구를 기반으로 PLA의 메타모델을 정의하고 메타모델을 기반으로 PLA의 정형명세를 제시한다[10]. 특별히 Object-Z를 이용한 PLA의 명세는 정형적인 변환 규칙을 기반으로 한 인스턴스화 과정을 정의하기 위한 기반을 마련한다. PLA에 대한 메타모델은 그림 1과 같다.

PLA는 일반 소프트웨어 아키텍처와는 다르게 제품계열 내의 여러 어플리케이션들이 공유하는 아키텍처로 어플리케이션들 사이에 공통적이고 가변적인 아키텍처 관련 결정 사항들을 설계한다[4,8]. PLA는 어플리케이션 공학 단계에서 특정 어플리케이션의 요구사항을 만족하도록 인스턴스화된다. 그림 1에서와 같이 PLA는 뷰(View)와 스타일(Style)이라는 추상적인 구성요소와 컴포넌트와 컴포넌트간의 관계라는 구체적인 구성요소로 구성된다. 일반적으로 아키텍처는 모듈뷰(Module View), 컴포넌트와 커넥터뷰(Component and Connector View, C&C View), 그리고 배치뷰(Deployment View)의 관점에서 표현될 수 있다[11-13]. 각 뷰는 아키텍처 설계 시에 적용할 수 있는 제약사항들이 정의되어 있는 아키텍처 스타일들을 통해서 실현된다. 따라서, 아키텍처는 관련 있는 아키텍처 스타일들을 준수하여 설계되며 그림 1에서 이런 관계를 'conforms to'로 표현한다. PLA에 발생하는 가변성에 관한 정보는 의사결정모델에 명세하며[14] PLA의 스타일, 컴포넌트, 컴포넌트간의 관계에 가변점(Variation Point)이 발생할 수 있다. 이런 PLA와 의사결정모델 사이의 관계는 그림 1에서 'refers to'로 표현하고 있다.

PLA의 메타모델을 기반으로 Object-Z를 이용하여 PLA를 명세한다. PLA의 기본요소에 대한 타입 정의는 다음과 같다.

[ARCHSTYLE, VARIATIONPOINT]
RELATION_TYPE::= *association* | *composition* | *generalization* | *dependency* | *aggregation* | *realization*
CLASSIFIER::= *Component* | *Class*

ARCHSTYLE는 아키텍처 스타일을 정의하기 위한 타입이고, VARIATIONPOINT는 가변점을 정의하기 위한 타입이다. RELATION_TYPE은 관계의 종류를 정의하기 위한 상수 변수 타입이다. RELATION_TYPE은 UML의 관계 분류를 준수하여 6 개의 값을 가진다. CLASSIFIER는 관계의 양 끝에 위치한 요소를 나타내며, 컴포넌트간의 관계를 나타내는 *ProductLine-*



Architecture 스키마에서는 *Component*가 된다. 이런 타입 정의를 기반으로 명세된 *ProductLineArchitecture* 스키마는 다음과 같다.

ProductLineArchitecture

archStyles : P ARCHSTYLE
components : *Component* ©
relationships : *Relationship* ©
variationPoints : P VARIATIONPOINT

Component ≙ [*interfaces* : *Interface* ©; *classes* : *Class* ©;
relationships : *Relationship* ©; *workflows* : *Workflow* ©]
Relationship ≙ [*relationType*:RELATION_TYPE;
relationStart, relationEnd:CLASSIFIER]
∀ *Relationships(m)* : P RELATION_TYPE •
∃ *components(i), components(j)* : P CLASSIFIER ∧
components(i) ≠ *components(j)*
⇒ *Relationships(m)* = *components(i)* || *components(j)*
∀ *variationPoints(x)* : P VARIATIONPOINT •
variationPoints(x) ∈ (dom P*archStyles* ∨ dom P*components* ∨
dom P*relationships*)

ProductLineArchitecture 스키마는 네 개의 변수를 정의하고 있다. *archStyles*는 PLA에 적용된 아키텍처 스타일을 명세하고, *components*는 PLA에 포함되어 있는 컴포넌트들을 정의하고, *relationships*는 컴포넌트간의 관계를 명세하고, *variationPoints*는 아키텍처 가변성을 명세한다. Object-Z에서 Containment 관계를 표현하는 ©는 그 타입에 소속되는 것으로, *Component*©는 컴포넌트 타입에 속하는 것을 말한다. Component 스키마는 Object-Z에서 정의된 Class 스키마를 확장하여 정의한 [7]를 참조하고 내부 스키마는 본 논문에서는 제외한다.

스키마의 속성 부분(property part)은 주어진 변수에 적용되는 제약사항과 시맨틱(Semantic)을 정의한다. *relationships* 는 컴포넌트간의 관계를 심볼로 표현하는 병렬 구성 연산자(Parallel Composition Operator) || 를 통해서 표현되며, 두 개의 다른 컴포넌트, *component(i)*와 *component(j)* 사이에 관계가 있음을 명시한다. 모든 *variationPoints*는 아키텍처 스타일, 컴포넌트 또는 컴포넌트간의 관계에 발생할 수 있음을 스키마에서 명세하고 있다.

일반적으로 의사결정모델은 핵심자산이 가지고 있는 가변성 정보를 명세하지만, 본 논문에서의 의사결정모델은 PLA가 포함하고 있는 가변성 정보만 활용한다. 의사결정모델은 그림 1에서와 같이 가변점, 가변점과 관련된 가변치(Variant), 해소 프로세스(Resolution Process)의 세 가지 요소로 구성된다. 가변점은 PLA 상에 가변성이 발생하는 지점을 의미하고, 각 가변점은 가변성 타입(Variability Type)과 가변성 범위(Variability Scope)의 추가적인 정보를 가지고 명세된다. 가변성 범위는 각 가변점당 가질 수 있는 후보 가변치의 개수를 의미하며, 일반적으로 이진(Binary), 선택(Selection), 오픈(Open)의 세 가지 영역이 있다 [15]. 가변치(Variant)는 가변점을 유효하게 채울 수 있는 값이나 인스턴스를 의미하며, 가변점을 설정하기 위한 요소가 된다[16]. 해소 프로세스는 가변점을 설정하기 위한 수행작업(Instruction)과 연관된 해소효과(Effect)를 정의하고 있다. 따라서, 해소 프로세스는 PLA를 인스턴스화하기 위한 일종의 지침이 된다. 해소효과는 가변점에 가변치를 설정한 후의 효과를 사후조건(Postcondition), 의존가변점(Affected Variation Point), 그리고 부수적 영향(Side-effect)의 세부 효과로 나누어 기술한다.

의사결정모델의 메타모델을 기반으로 Object-Z를 이용하여 의사결정모델을 명세한다. 의사결정모델의 기본 요소에 대한 타입 정의는 다음과 같다.

VARIABILITY_TYPE ::= *archstyle* | *component* | *relationship*
VARIABILITY_SCOPE ::= *binary* | *selection* | *open*

VARIABILITY_TYPE은 가변점의 타입을 정의하기 위한 상수 변수 타입이다. VARIABILITY_SCOPE은 각 가변점당 가변치의 범위를 정의하기 위한 상수 변수 타입이다. 정의된 타입을 기반으로 명세된 *DecisionModel* 스키마는 다음과 같다.

DecisionModel

variationPoints : P VARIATIONPOINT
variants : VARIANT
variabilityType : VARIABILITY_TYPE
variabilityScope : VARIABILITY_SCOPE
resolutionProcesses : *ResolutionProcess*©

∀ *variationPoints(x)* : P VARIATIONPOINT •
variationPoints(x) ∈ (dom *ProductLineArchitecture.variationPoints* ∨
dom *Component.variationPoints*)
∀ *variationPoints(x)* : P VARIATIONPOINT • ∃ *variants(m)* , *variants(n)* : P VARIANT
(*variationPoints(x)* ↦ *variants(m)*) = (*variationPoints(x)* ↦ *variants(n)*)
⇒ *variants(m)* = *variants(n)*
∃ *variants(x)* : P VARIANT • ∃ *resolutionProcesses(x)* : *ResolutionProcess* •
variants(x) ↦ *resolutionProcesses(x)*

DecisionModel 스키마는 *variationPoints*, *variants*, *variabilityType*, *variabilityScope*, *resolutionProcesses*의 다섯 개 변수를 정의하고 있다. 속성 부분에서는 *DecisionModel* 스키마의 변수들에 적용되는 제약사항을 기술하고 있다. *variationPoints*는 유일해야 하고 PLA의 가변점을 참조하고 있어야 한다. 각각의 *variationPoints*에 대해 여러 개의 *variants*가 존재할 수 있고, 각 *variants*는 유일해야 한다. 그리고, 각 *variants*에 대해 하나의 관련된 *resolutionProcesses*를 가지고 있다.

다음은 *resolutionProcesses*변수의 타입을 나타내는 *ResolutionProcess*의 스키마에 대한 명세이다.

[INSTRUCTION, EFFECT]

INSTRUCTION은 수행작업(Resolution Instruction)을 정의하기 위한 타입이고, EFFECT는 해소효과(Resolution Effect)을 정의하기 위한 타입이다.

ResolutionProcess
variationPoints : P VARIATIONPOINT
instructions : P INSTRUCTION
effects : P EFFECT

variationPoints ⊂ *DecisionModel.variationPoints*
∀ *variationPoints(x)* : P *variationPoints* • ∃ *variants(i)* : P VARIANT ∧
∀ *variants(i)* : P VARIANT • ∃ *instructions(i)* : P INSTRUCTION •
instructions(i) ↦ *effects(i), effects(i)* : P EFFECT
effects = *PostCondition* ∧ *(AffectedVariationPoint* ∨ *SideEffect)*
// 각 가변점의 instructions과 effects에 대한 제약사항을 추가할 수 있음

ResolutionProcess 스키마는 의사결정모델의 각 가변점을 위한 *instructions*와 *effects*로 구성된다. 하나의 가변점에 대해 여러 개의 *instructions*와 *effects*가 존재한다. 어떠한 가변점에 대한 가변성 해소작업을 실행한 효과는 *PostCondition, SideEffect, AffectedVariationPoint*으로 기술된다. 단, 효과를 명세할 시에, *PostCondition*는 필수적으로 명세해야 하지만, *SideEffect* 또는 *AffectedVariationPoint*에 대한 명세는 선택적이다. 가변점에 가변치를 설정함으로써 수행해야 하는 *instructions*나 *effects*에 대한 상세한 제약조건은 5장에서 각 가변점 타입별로 변환규칙을 제시하면서 기술한다.

4. 의사결정 해소모델 및 인스턴스화 아키텍처 명세

PLA를 인스턴스화하기 이전에 특정 어플리케이션의 요구사항에 맞게 의사결정 해소모델을 정의해야 한다. 우선 의사결정모델에 정의된 가변점 중에 목표 어플리케이션이 사용하는 가변점을 선택하고, 이렇게 선택된 가변점에 대해 가변치를 결정함으로써 의사결정 해소모델이 정의된다. 가변치 결정 사항은 외부의 의사결정 해소모델 정의자로부터 입력 받는다. 가변치 결정 시에, 가변치 범위가 이진이나 선택과 같이 후보 가변치가 알려진 경우에는 의사결정모델에 제시되어 있는 가변치 중에 하나를 선택하고, 오픈 범위처럼 의사결정모델에 제시된 후보 가변치가 없는 경우에는 의사결정 해소모델 정의자가 새로이 정의한다. 의사결정 해소모델을 명세한 *ResolutionModel*은 다음과 같다.

ResolutionModel
DecisionModel
r_variationPoints ? : VARIATIONPOINT
r_variants ? : VARIANT
r_variabilityScope ? : VARIABILITY_SCOPE
r_variabilityType ? : VARIABILITY_TYPE
r_resProc ? : *RM_ResolutionProcess*©

r_variationPoints? ∈ *DecisionModel.variationPoints*
r_variants? ∈ *DecisionModel.variants*
r_variabilityType? ∈ *DecisionModel.variabilityType*
r_resProc? ∈ *DecisionModel.resolutionProcesses*
r_variationPoints ? : P VARIATIONPOINT • ∃ *r_variants*? : P VARIANT ∧
r_variants? : P VARIANT • ∃ *r_resProc* !: *ResolutionProcess* • *r_variants*? ⇝ *r_resProc* !
∃*r_resProc* ? : *ResolutionProcess* • ∃ *instructions(i)* : P INSTRUCTION •
instructions(i) ↦ *effects(i), effects(i)* : P EFFECT
r_variationPoints? : P VARIATIONPOINT •
if *DecisionModel.variabilityScope* = *open* then *DefineVariant* else *SelectVariant*
DefineVariant ≙ [Δ *ResolutionModel; variants* ? : P VARIANT |
r_variants' = *r_variants* ∪ {*variants*?}; *variants* ? ∈ F₁ *NewVariants*
• *NewVariants* ⊄ *r_variants* ∧ # *NewVariants* >= 2]
SelectVariant ≙ [Ξ *ResolutionModel; variants* ? : P VARIANT |
variants ? ∈ *r_variants* ; if *r_variabilityScope* = *binary* then
{ *variants* ?} = 2 else # { *variants* ?} >= 3]
RM_ResolutionProcess≙ [*variationPoints* : P VARIATIONPOINT;
instructions : P INSTRUCTION; *effects* : P EFFECT;
r_variabilityType?:VARIABILITY_TYPE;*instType*? : INSTRUCTION_TYPE]

ResolutionModel 스키마는 *DecisionModel* 스키마를 상속받고, *r_variationPoints, r_variants, r_variabilityScope, r_variabilityType, r_resProc*의 다섯 개 변수로 구성된다.

ResolutionModel 스키마 내의 변수 *r_variationPoints*은 *DecisionModel*에 정의되어 있는 가변점 중에 목표 어플리케이션에서 사용되는 가변점만을 선택한 것이다. *r_variants*는 선택된 가변점에 대한 가변치로 *DecisionModel*에 정의되어 있는 가변치 중에 선택한다. *r_variabilityScope*과 *r_variabilityType*는 *DecisionModel* 스키마에 정의되어 있는 선택된 가변점의 범위와 타입 정보를 가져온 것이다. 그리고 각 가변점에 대한 가변치가 선택되면 연관된 해소 프로세스도 선택된다. *r_variationPoints* 중에 *r_variabilityScope*가 *binary*나 *selection*일 경우에는 *DecisionModel*에 정의되어 있는 *r_variationPoints*의 가변치 중에 하나의 가변치를 선택하고, *r_variabilityScope*가 *open*일 경우에는 새로운 가변치를 정의한다. *DefineVariant*를 통해 입력 받은 가변치로 *open* 가변점을 설정한다. 그리고 *SelectVariant*를 통해 가변치를 선택하는데, *binary* 가변점의 경우 두 개의 가변치 중 하나를 선택하고 *selection* 가변점의 경우 세 개 이상의 가변치 중 하나를 선택한다.

인스턴스화된 아키텍처는 PLA의 변형 형태로 의사결정 해소모델에 명시된 가변점에 대한 가변치와 해소 프로세스를 적용한 것이다. 인스턴스화된 아키텍처의 기본 요소를 위한 타입 정의는 다음과 같다.

RESOLUTION _TYPE ::= *add* | *delete* | *replace*

이 타입을 사용하여 인스턴스화된 아키텍처 스키마 *InstantiatedArchitecture* 를 명세하면 다음과 같다.

InstantiatedArchitecture
ProductLineArchitecture
ResolutionModel
i_components : *Component*©
i_relationships : *Relationship*©

if *ResolutionModel.r_variabilityType*? = *archstyle*
then *ModifyComp*§ *ModifyRel*
else if *ResolutionModel.r_variabilityType*? = *component* then *ModifyComp*
else if *ResolutionModel.r_variabilityType*? = *relationship* then *ModifyRel*
// 속성에 대한 추가 정의는 5장 참조

InstantiatedArchitecture 스키마는 *ProductLineArchitecture* 스키마와 *ResolutionModel* 스키마를 상속받고, 세 개의 변수로 구성된다. 변수 *i_archStyles*는 특정 스타일을 명세하고, *i_components*는 인스턴스화된 컴포넌

트를 명세한다. 그리고, *i_relationships*는 인스턴스화된 관계를 명세한다. 이 스키마의 속성부분은 인스턴스화된 아키텍처 구성요소에 영향을 주는 제약사항과 시맨틱을 정의한다.

5. 제품계열 아키텍처 인스턴스화를 위한 규칙

본 장에서는 Object-Z를 이용하여 PLA가 인스턴스화된 아키텍처(Instantiated Architecture)로 인스턴스화되기 위한 기법을 제안한다. 그림 2는 PLA의 구성요소와 인스턴스화된 아키텍처의 구성요소간의 매핑 관계를 보여주고 있다. 의사결정모델은 PLA가 가지고 있는 가변성 정보를 명세하고 있고, 의사결정 해소모델은 각 어플리케이션의 요구사항에 맞게 의사결정모델의 가변성을 선택한 정보를 가지고 있다. PLA는 이런 의사결정 해소모델의 결정 사항을 반영하여 인스턴스화된 아키텍처로 인스턴스화된다. 따라서, PLA의 각 구성요소인 스타일, 컴포넌트, 컴포넌트간의 관계는 의사결정 해소모델을 반영하여 인스턴스화된 아키텍처의 스타일, 컴포넌트, 컴포넌트간의 관계로 변경된다. 다음 섹션에서 각 요소에 대한 해소 패턴(Resolution Pattern) 및 해소 패턴을 기반으로 한 인스턴스화 기법을 제안한다.

5.1 해소 패턴

PLA는 PLA의 구성요소별로 해소되어 인스턴스화된 아키텍처로 변환된다. 각 구성요소별로 해소가 이루어질 때, 표 1의 해소 패턴을 적용한다. 각 패턴에 대해, 적용 가변점, 가변치, 수행작업, 그리고 해소효과가 제시되어 있다. 해소 패턴은 크게 스타일, 컴포넌트, 컴포넌트 간의 관계 해소를 위한 해소 패턴으로 나눌 수 있다. 스타일은 표 1과 같이 모듈뷰, C&C 뷰, 배치뷰의 스타일로 나눌 수 있다. 아키텍처 스타일 가변점에 가변치를 설정함으로써 아키텍처 스타일의 변환이 수행되는데 아키텍처 스타일 가변점에 가변치를 설정할 때에 수행해야 하는 수행작업과 만족해야 하는 해소효과에 대한 패턴은 표 1과 같다. 표 1의 해소패턴을 이용하여 스타일 가변성이 해결되며 이를 기반으로 5.2에서 인스턴스화 규칙을 정의한다. PLA의 물리적인 구성요소로는 컴포넌트와 컴포넌트간의 관계가 있으며 이런 요소들은 스타일을 적용함으로써 설계된다. 즉, 스타일은 컴포넌트와 컴포넌트 간의 관계를 유도하기 위한 중간 역할을 하는 개념적인 요소이다. 따라서, 스타일 가변성을 해결하기 위한 효과 중 의존가변점은 대체적으로 컴포넌트나 컴포넌트간의 관계 가변점이 된다.

컴포넌트 해소 패턴의 경우, 선택된 가변치에 따라 컴포넌트 추가, 컴포넌트 삭제, 컴포넌트 대체의 3 가지 수행작업이 있다. '컴포넌트 삭제' 수행작업에 대해, 가변점 해소 후의 사후조건은 '컴포넌트가 삭제된 상태'이며 의존가변점은 '삭제된 컴포넌트와 관련된 관계를 삭제한다' 이다. 예를 들어, 회원, 재고, 대여, 예약, 회계 컴포넌트를 가지고 있는 도서 대여 핵심자산을 인스턴스화하여 도서 대여 시스템을 개발하려고 할 때, 도서 대여 시스템은 예약 서비스를 제공하지 않는다고 하자. 이런 경우, 인스터스화 시에 '컴포넌트 삭제'라는 해소 작업을 적용하여 예약 컴포넌트를 제외할 수 있다. 사후조건은 '예약 컴포넌트 삭제됨'이 되며 의존가변점으로 예약 컴포넌트와 관련된 관계들이 삭제될 것이다.

5.2 인스턴스화 규칙

그림 2와 표 1을 기반으로 Object-Z를 이용한 인스턴스화 규칙을 정의한다. 특히 스타일 가변성을 해결하기 위한 인스턴스화 규칙을 정의하기 위해, 아키텍처 스타일 해소를 위한 수행작업 및 해소효과에 대한 제약사항이 다음과 같이 해소 프로세스 스키마에 추가되어야

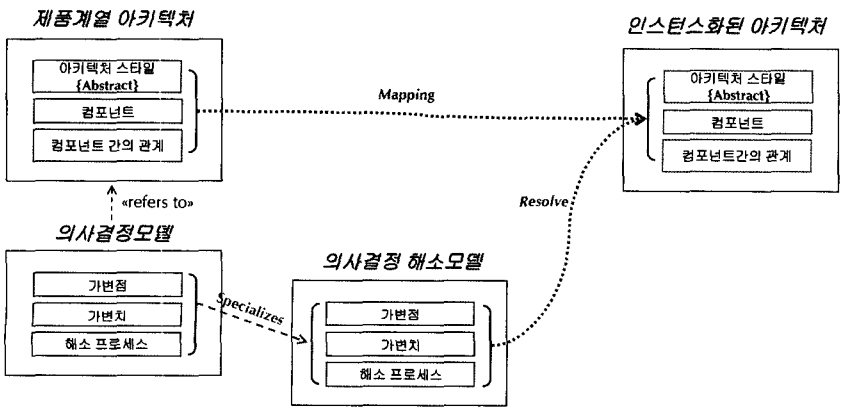


그림 2 제품계열 아키텍처와 인스턴스화된 아키텍처간의 매핑

표 1 PLA를 위한 해소 패턴

패턴명	가변점	가변치	해소 프로세스		
			수행작업	해소효과	
스타일 해소 패턴	모듈뷰의 스타일	선택된 모듈뷰 스타일의 집합	모듈뷰 스타일 변경	사후조건	➢ 인스턴스화된 아키텍처를 구성하는 모듈뷰에 대한 해당 스타일이 변경된 상태
				의존가변점	➢ 모듈뷰를 구성하는 컴포넌트를 추가, 삭제, 또는 대체한다. ➢ 모듈뷰를 구성하는 관계를 추가, 삭제, 또는 대체한다.
	C&C 뷰의 스타일	선택된 C&C뷰 스타일의 집합	C&C 뷰 스타일 변경	사후조건	➢ 인스턴스화된 아키텍처를 구성하는 C&C뷰에 대한 해당 스타일이 변경된 상태
				의존가변점	➢ C&C뷰를 구성하는 컴포넌트를 추가, 삭제, 또는 대체한다. ➢ C&C뷰를 구성하는 관계를 추가 또는 삭제한다.
	배치뷰의 스타일	선택된 배치뷰 스타일의 집합	배치뷰 스타일 변경	사후조건	➢ 인스턴스화된 아키텍처를 구성하는 배치뷰에 대한 해당 스타일이 변경된 상태
				의존가변점	➢ 배치뷰를 구성하는 HW 컴포넌트를 추가, 삭제, 또는 대체한다. ➢ 배치뷰를 구성하는 관계를 추가, 삭제, 또는 대체한다. ➢ 배치뷰를 구성하는 컴포넌트(모듈뷰와 C&C뷰에서 추출된 컴포넌트)를 복사한다(Duplicate).
컴포넌트 해소 패턴	컴포넌트	컴포넌트 가변점에 대한 선택된 가변치	컴포넌트 추가	사후조건	➢ 컴포넌트가 추가된 상태
				의존가변점	➢ 추가된 컴포넌트와 관련된 컴포넌트를 추가한다. ➢ 추가된 컴포넌트와 관련된 관계를 추가한다.
			컴포넌트 삭제	사후조건	➢ 컴포넌트가 삭제된 상태
				의존가변점	➢ 삭제된 컴포넌트와 관련된 컴포넌트를 삭제한다. ➢ 삭제된 컴포넌트와 관련된 관계를 삭제한다.
			컴포넌트 대체	사후조건	➢ 컴포넌트가 삭제되고 추가된 상태
				의존가변점	➢ 대체된 컴포넌트와 관련된 컴포넌트를 삭제하고 추가한다. ➢ 대체된 컴포넌트와 관련된 관계를 삭제하고 추가한다.
관계 해소 패턴	컴포넌트간의 관계	컴포넌트간의 관계 가변점을 위해 선택된 가변치	관계 추가	사후조건	➢ 관계가 추가된 상태
				의존가변점	
			관계 삭제	사후조건	➢ 관계가 삭제된 상태
				의존가변점	

한다.

INSTRUCTION_TYPE ::= *add* | *delete* | *replace*

INSTRUCTION_TYPE은 상수 변수 타입으로써 PLA 상의 가변점에 가변치가 설정되었을 경우에 수행되는 실제 행동을 나타낸다.

```
ResolutionProcess
// 변수(variable) 정의는 3장 참조
instType : INSTRUCTION_TYPE
---
// 기본 속성(Predicate) 정의는 3장 참조
if DecisionModel.variabilityType = archstyle then ModifyComp § ModifyRel
else if DecisionModel.variabilityType = component then ModifyComp
else if DecisionModel.variabilityType = relationship then ModifyRel
ModifyComp= [if instType = add then AddComp § AddRel
      else if instType = delete then DeleteComp § DeleteRel
      else if instType = replace then DeleteComp § DeleteRel § AddComp § AddRel]
ModifyRel= [if instType = add then AddRel
      else if instType = delete then DeleteRel]
```

ResolutionProcess 스키마는 스타일에 대한 해소 수행 작업을 위한 실제 활동을 나타내는 *instType* 변수를 추가적으로 가지고 있다. 속성 부분에서는 가변점의 가변성 타입이 ARCHSTYLE일 경우에는 *ModifyComp*나 *ModifyRel*와 같은 수행작업이 수행됨을 나타낸다. 그리고, 이런 수행작업들은 *instType*에 따라서 *AddComp*, *AddRel*, *DeleteComp*, *DeleteRel*와 같은 실제 활동으로 상세화된다.

표 1에 제시된 해소 패턴 중의 하나가 아키텍처 스타일 패턴이다. 그림 3은 PLA 중 아키텍처 스타일에 대한 가변성을 해소하기 위한 전반적인 인스턴스화 규칙이다.

ProductLineArchitecture 스키마의 *archStyles*을 *InstantiatedArchitecture* 스키마로 인스턴스화하기 위해서는 우선 *DecisionModel* 스키마를 참조해서 *ResolutionModel* 스키마를 생성해야 한다. 여기에서, 가변점과 가변치는 '*archStyles*' 가변성을 위해 정의된 것이어야 한다. 의사결정 해소모델은 의사결정모델로부터 가변점과 적절한 가변치를 선택하고, 선택한 가변치에 대한 해소 프로세스를 수행함으로써 PLA를 인스턴스화한다.

구체적인 수준의 PLA 집합 *PLArch*는 컴포넌트들와 컴포넌트간의 관계들로 구성된다.

$$PLArch = \{Comp_1, Comp_2, Comp_3, ..., Comp_i, Rel_1, Rel_2, Rel_3, ..., Rel_j\}$$

의사결정모델에서 가변점의 집합을 *SetOfVarPt_Dec-*

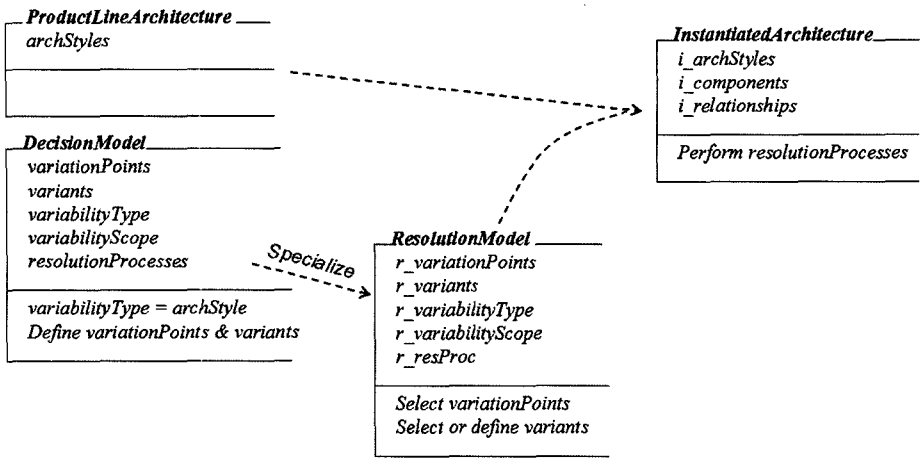


그림 3 PLA 중 아키텍처 스타일 명세의 인스턴스화

*Model*라 하고, 가변치의 집합을 *SetOfVariant_DecModel*, 그리고 해소 프로세스의 집합을 *SetOfResProc_DecModel* 라 하자. 의사결정모델은 가변점과 가변치, 해소 프로세스로 구성되므로 다음과 같이 정의된다.

$DecModel = SetOfVarPt\_DecModel \cup SetOfVariant\_ecModel \cup SetOfResProc\_DecModel$

의사결정모델로부터 가변점의 집합 *SetOfVarPt_ecModel*은 다음과 같이 정의된다.

$SetOfVarPt\_DecModel = \{VP_1, VP_2, VP_3, ..., VP_n\}$

하나의 가변점에 대하여 여러 개의 가변치가 존재하므로 특정 가변점 VP_i에 대한 가변치의 집합 *Variant_ecModel*(VP_i)은 다음과 같이 정의된다.

$Variant\_DecModel(VP_i) = \{(VP_i, V_{i1}), (VP_i, V_{i2}), (VP_i, V_{i3}), ..., (VP_i, V_{ij})\}$

그러므로, 전체 가변치의 집합 *SetOfVariant_DecModel*은 다음과 같다.

$SetOfVariant\_DecModel =$
$\{(VP_1,V_{11}), (VP_1, V_{12}), (VP_1, V_{13}), ..., (VP_1, V_{1j}),$
$(VP_2,V_{21}), (VP_2, V_{22}), (VP_2, V_{23}), ..., (VP_2, V_{2k}),$
$.....,$
$(VP_i,V_{i1}), (VP_i, V_{i2}), (VP_i, V_{i3}), ..., (VP_i, V_{im})\}$

특정 가변치 V_{ij}에 대해 정의된 해소 프로세스를 RP_j라 하면, 해소 프로세스의 집합 *ResProc_DecModel*(V_{ij})은 다음과 같다.

$ResProc\_DecModel\ (V_{ij}) = \{(V_{i1}, RP_1), (V_{i2}, RP_2), (V_{i3}, RP_3), ..., (V_{ij}, RP_j)\}$

그러므로, 전체 해소 프로세스의 집합 *SetOfResProc_DecModel* 은 다음과 같이 정의된다.

$SetOfResProc\_DecModel =$
$\{(V_{11}, RP_1), (V_{12}, RP_2), (V_{13}, RP_3), ..., (V_{1i}, RP_i),$
$(V_{21}, RP_1), (V_{22}, RP_2), (V_{23}, RP_3), ..., (V_{2j}, RP_j),$
$.....,$
$(V_{i1}, RP_1), (V_{i2}, RP_2), (V_{i3}, RP_3), ..., (V_{in}, RP_n)\}$

PLA 를 인스턴스화하기 위해서는 의사결정모델을 커스터마이징하여 의사결정 해소모델을 생성해야 한다. 의사결정 해소모델의 가변점 집합은 의사결정모델의 가변점의 부분집합이므로 다음과 같이 정의된다.

$SetOfVarPt\_ResModel \subset SetOfVarPt\_DecModel$
$SetOfVariant\_ResModel \subset SetOfVariant\_DecModel$
$SetOfResProc\_ResModel \subset SetOfResProc\_DecModel$

하나의 가변점이 있을 때, 어플리케이션에 맞도록 가변치들 중에서 하나의 가변치가 선택된다. 만약 가변점 VP_i에 대해 선택된 가변치의 가변성 범위가 이진이거나 선택 범위라면 어플리케이션에 적절한 가변치가 선택될 것이다. 가변성 범위가 오픈인 경우는 새로운 가변치를 정의해야 한다.

if *DecisionModel.variabilityScope* = *open* then
$SetOfVariant\_ResModel(VP_i) = DefineVariant$
else $SetOfVariant\_ResModel(VP_i) = SelectVariant$
$DefineVariant = \{\exists V_j : \mathbb{P}\ \text{VARIANT} \mid \text{new}\ V_j\ ,\ V_j \notin SetOfVariant\_DecModel\}$
$SelectVariant = \{\exists V_j : \mathbb{P}\ \text{VARIANT} \mid V_j \notin SetOfVariant\_DesModel\}$

인스턴스화된 아키텍처는 의사결정 해소모델 안에 명세된 가변점 VP_i에 맞는 가변치와 해소 프로세스를 적용하여 PLA의 형태를 수정한 것이다.

$InstArch(VP_i)=PLArch(SetOfVariant\_ResModel(VP_i))$

의사결정 해소모델에 명세된 가변치와 해소 프로세스에 따라 아키텍처 구성요소가 수정된다. 모듈뷰 관점에서 다양한 스타일이 아키텍처 스타일의 가변치로 선택

되면 컴포넌트와 컴포넌트간의 관계는 선택된 스타일에 맞게 변경된다. 상세히 표현하면 컴포넌트와 컴포넌트간의 관계에서 추가, 삭제, 또는 대체와 같은 작업이 발생하고, 이는 *AddComp*, *AddRel*, *DeleteComp*, *DeleteRel* 스키마에서 정의된다. *Addcomp* 스키마는 PLA 내의 컴포넌트의 집합에 특정 컴포넌트 $Comp_x$ 를 추가하는 것이다.

$$\exists Comp_x : Component© \cdot SetOfComp\_InstArch = SetOfComp\_PLArch \cup \{Comp_x\}$$

또한 컴포넌트 $Comp_x$가 PLA에 추가되면, 의존가변점으로 컴포넌트 $Comp_x$와 관련된 관계가 추가된다. 이 상황에서 컴포넌트 $Comp_x$와 관계를 갖는 컴포넌트 $Comp_y$가 존재하게 된다. 그러므로, 이 관계는 PLA내에 있는 관계의 집합 *SetOfRel_PLArch*에 추가된다. 스키마 *DeleteComp*와 *DeleteRel*도 같은 방법으로 컴포넌트와 관계가 정의된다.

$$\exists Comp_y: Component©,\ Rel_i: Relationship© \cdot Rel_i = Comp_x \,||\, Comp_y \wedge SetOfRel\_InstArch = SetOfRel\_PLArch \cup \{Rel_i\}$$

위에서 정의한 인스턴스화 규칙을 적용하여 인스턴스화된 아키텍처 스키마는 다음과 같이 정의된다.

InstantiatedArchitecture

ProductLineArchitecture
ResolutionModel
i_components : *Component©*
i_relationships : *Relationship©*

---

if *ResolutionModel.r_variabilityType?* = *archstyle* then *ModifyComp*; *ModifyRel*
else if *ResolutionModel.r_variabilityType?* = *component* then *ModifyComp*
else if *ResolutionModel.r_variabilityType?* = *relationship* then *ModifyRel*
ModifyComp≙ [if *ResolutionModel.instType ?* = *add* then *AddComp* ; *AddRel*
else if *ResolutionModel.instType ?* = *delete* then *DeleteComp* ; *DeleteRel*
else if *ResolutionModel.instType ?* = *replace* then
DeleteComp ; *DeleteRel* ; *AddComp* ; *AddRel*]
ModifyRel≙ [if *ResolutionModel.instType ?* = *add* then *AddRel*
else if *ResolutionModel.instType ?* = *delete* then *DeleteRel*]
AddComp≙ [∃ *components(x)* : *Component©* ·
i_components' = *i_components* ∪ { *components (x)*}]
AddRel≙ [∃ *relationships(i)* : *Relationship©* ·
i_relationships' = *i_relationships* ∪ { *relationships* (i)}]
DeleteComp≙ [∃ *components(x)* : *Component©* ·
i_components' = *i_components* \ { *components (x)*}]
DeleteRel≙ [∃ *relationships(i)* : *Relationship©* ·
i_relationships' = *i_relationships* \ { *relationships* (i)}]

인스턴스화된 아키텍처 스키마 *InstantiatedArchitecture*는 PLA 스키마 *ProductLineArchitecture*와 의사결정 해소모델 스키마 *ResolutionModel*을 상속받아 구성되고, 네 개의 변수를 정의하고 있다. *i_archStyles*은 선택된 특정 스타일을 명세하고, *i_components*는 인스턴스화된 컴포넌트들을 정의하고, *i_relationships*는 인스턴스화된 관계를 명세한다. 스키마의 속성 부분은 의사결정 해소모델의 해소 프로세스를 적용하여 생성된 아키텍처의 요소에 관한 제약사항과 시맨틱을 정의하고 있다.

6. 사례 연구

본 장에서는 본 논문에서 제안한 정형 명세를 이용한 인스턴스화 기법을 대여 도메인에 적용하여 제안된 기법의 적용성을 보여준다. 대여 제품계열에 대해 간략히 소개한 후, 범용 대여 PLA를 설계하고 이를 도서 대여 시스템의 요구사항에 맞게 인스턴스화한다.

6.1 대여 제품계열에 대한 소개

대여 제품계열 내에는 자동차 대여 시스템, 도서 대여 시스템, 비디오 대여 시스템과 같은 여러 대여 시스템이 존재한다. 일반적으로 대여 시스템은 회원 관리, 재고 관리, 예약 관리, 대여 관리의 서비스를 제공한다. 대여 시스템은 회원 정보를 등록, 갱신, 조회, 삭제, 비활성화할 수 있어야 하고, 대여 대상 품목에 대한 정보 또한 등록, 갱신, 조회, 삭제, 비활성화할 수 있어야 하며 대여 대상 품목의 상태 변화도 추적할 수 있어야 한다. 또한 대여 시스템의 중요한 기능으로는 대여, 반납, 이전의 대여 여부 검사, 연체 검사 및 연체료 수납 등이 있다. 일부 시스템에서는 대여하기 이전에 예약을 할 것을 요구한다.

6.2 제품계열 아키텍처의 모듈뷰 설계

본 절에서는 UML을 이용하여 대여 PLA의 모듈뷰를 설계하고 이를 Object-Z로 명세한다. 그림 4는 [11]에서 제시한 Decomposition 스타일과 Uses 스타일을 적용하여 설계한 대여 PLA에 대한 모듈뷰를 보여준다. 대여 PLA 내에는 C_Member, C_Inventory, C_Rental, C_Reservation의 4개의 컴포넌트가 있으며, C_Reservation 컴포넌트는 일부 시스템에서는 사용하지 않기 때문에 C_Reservation 컴포넌트와 이와 연관된 관계에 가변성이 존재한다.

이제 그림 4의 설계를 기반으로 3장에 정의된 *ProductLineArchitecture* 스키마를 적용하여 *RentalProductLineArchitecture* 스키마를 명세한다.

RentalProductLineArchitecture

archStyles : ARCHSTYLE
c_member, c_inventory, c_rental, c_reservation : *Component©*
rel1, rel2, rel3, rel4 : *Relationship©*
c_reservation, rel2, rel4 : ℙ VARIATIONPOINT

∀ *rel1, rel2, rel3, rel4* : *Relationship©* ·
∃ *c_member, c_inventory, c_rental, c_reservation* : *Component©*
⇒ *rel1* = *c_member* || *c_rental* ∧
rel2 = *c_member* || *c_reservation* ∧
rel3 = *c_inventory* || *c_rental* ∧
rel4 = *c_inventory* || *c_reservation* ∧

RentalProductLineArchitecture 스키마는 대여 제품계열의 회원 관리, 재고 관리, 대여 관리, 예약 관리 서비스를 위해 *c_member*, *c_inventory*, *c_rental*, *c_reservation*의 4 개 컴포넌트를 정의하고 있으며, 이들

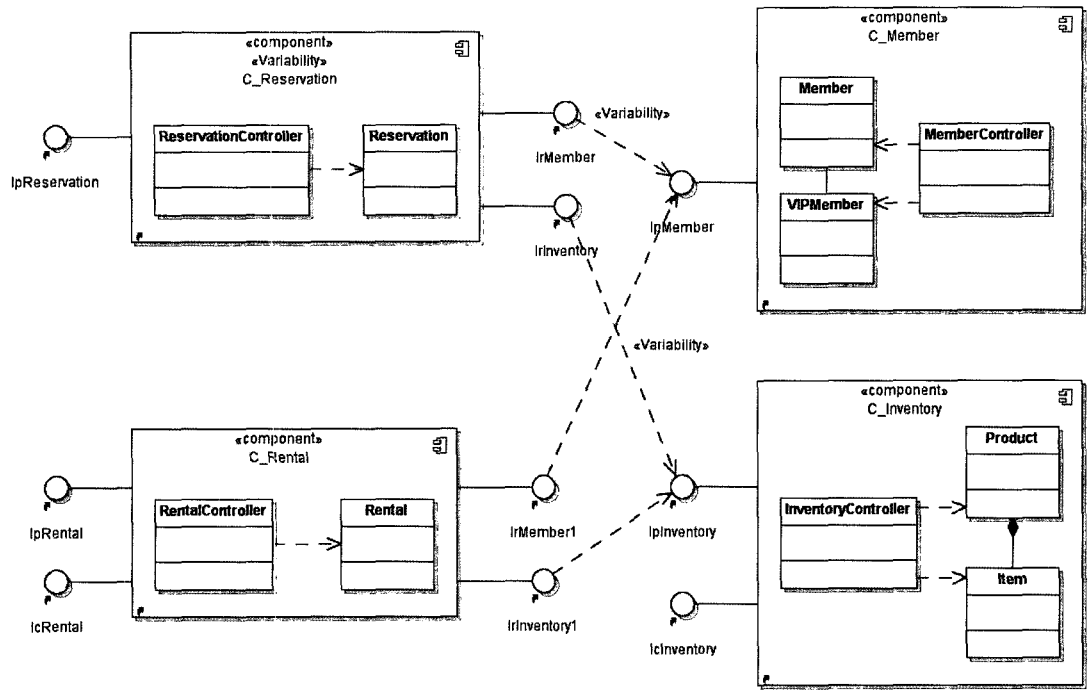


그림 4 대여 제품계열 아키텍처의 모듈뷰

컴포넌트 사이의 관계인 rel_1, rel_2, rel_3, rel_4를 정의하고 있다. 또한, *c_reservation* 컴포넌트와 이와 연관된 관계인 *rel2*, *rel4*에 가변점이 존재한다. 속성 부분에서는 병렬 구성 연산자 || 를 통하여 컴포넌트 사이의 관계를 명세한다. 또한, 3장에서 *ProductLineArchitecture* 스키마 내부에 정의된 컴포넌트 스키마를 적용하여 4 개의 컴포넌트에 대한 내부 스키마를 다음과 같이 명세한다.

C_Member
ipMember : Interface ©
member, vipMember, memberController : Class ©
memberRel : Relationship ©
memberWF : Workflow ©

C_Inventory
ipInventory, icInventory : Interface ©
product, item, inventoryController : Class ©
inventoryRel : Relationship ©
inventoryWF : Workflow ©

C_Rental
ipRental, icRental, irMember, irInventory : Interface ©
rental, rentalController : Class ©
rentalRel : Relationship ©
rentalWF : Workflow ©

C_Reservation
ipReservation, irMember, irInventory : Interface ©
reservation, reservationController : Class ©
reservationRel : Relationship ©
reservationWF : Workflow ©

6.3 의사결정모델 설계

본 절에서는 대여 PLA에 대한 의사결정모델을 설계하고 이를 Object-Z로 명세한다. 표 2는 PLA에서 발생하는 가변성에 대해 기술하고 있다. 인스턴시에이션시에 예약 기능을 제외할 수 있으며 가변점 A01과 A02의 '이진' 가변성 영역은 해당 가변치가 선택적이라는 것을 명시한다.

표 2의 의사결정모델을 기반으로 3장에서 정의된 *DecisionModel* 스키마를 적용하여 명세하면 다음과 같다. VARIANT는 표 2에 명시된 가변치를 나타내는 타입이다.

VARIANT ::= *v1* | *v2*

VARIABILTY_SCOPE ::= *binary* | *select* | *open*

DecisionModel
C_Reservation_DecModel
c_reservation : ℙ VARIATIONPOINT
variant ? : ℙ VARIANT
---
if variant ? = v1 then deleteC_Reservation ;

표 2 대여 PLA를 위한 의사결정모델

ID	가변점	가변성 타입	가변성 영역	가변치	수행작업	해소효과	
A01	C_Reservation 컴포넌트	컴포넌트	이진	V1	C_Reservation 삭제	사후조건	C_Reservation 삭제된 상태
						의존 가변점	'C_Reservation 컴포넌트와 연관된 관계' 가변점의 'V1' 가변치
A02	C_Reservation 컴포넌트와 연관된 관계	컴포넌트 간의 관계	이진	V1	관계(C_Rental: C_Reservation), 관계(C_Reservation: C_Inventory), 관계(C_Reservation: C_Member) 삭제	사후조건	관계(C_Rental:C_Reservation), 관계(C_Reservation:C_Inventory), 관계(C_Reservation:C_Member) 삭제된 상태
						의존 가변점	-

C_ReservationRel_DecModel

C_ReservationRel_DecModel
rel2, rel4 : P VARIATIONPOINT
variant ? : P VARIANT

if variant ? = v1 then deleteRel2 ⨟ deleteRel4

DecisionModel 스키마는 PLA의 가변성을 명세한다. 스키마 내에 PLA 내의 C_Reservation 컴포넌트와 관계에 발생하는 가변점, 가변치, 해소 프로세스에 대해 정의한다.

6.4 의사결정 해소모델 정의

대여 PLA를 인스턴스화하기 이전에 도서 대여 시스템에 맞게 의사결정모델에 정의된 가변치를 선택하여 의사결정 해소모델을 정의한다. 도서 대여 시스템의 경우, 예약 업무를 지원하지 않는다. 따라서, A01과 A02 가변점에 대해 V1 가변치를 선택하고, 이에 따라 인스턴시에이션시에 C_Reservation 컴포넌트와 이와 연관된 관계가 삭제된다.

표 3 도서 대여 시스템을 위한 의사결정 해소모델

ID	가변점	가변치	수행작업	해소효과
A01	V1 선택. 사후조건, 의존 가변점은 표 2와 동일함			
A02	V1 선택. 사후조건, 의존 가변점은 표 2와 동일함			

4장에서 정의된 *ResolutionModel* 스키마를 적용하여 *DecisionModel* 스키마를 도서 대여 시스템에 맞게 특화한 *ResolutionModel*을 명세한다.

ResolutionModel
C_Reservation_ResModel
c_reservation : P VARIATIONPOINT
variant? : P VARIANT

variant ? = v1
deleteC_Reservation ⨟ C_ReservationRel_ResModel

C_ReservationRel_ResModel
rel2, rel4: P VARIATIONPOINT
variant ? : P VARIANT

variant ? = v1
deleteRel2 ⨟ deleteRel4

ResolutionModel 스키마는 도서 대여 시스템에 맞게 *DecisionModel*를 특화하였다. *DecisionModel* 스키마에 정의된 각 가변점에 대해 v1 가변치를 선택하고 이에 대응하는 해소 프로세스를 선택하였다.

6.5 인스턴스화된 제품계열 아키텍처 도출

마지막으로 6.4 절의 의사결정에 맞게 대여 PLA를 인스턴스화하면 다음과 같이 명세된다.

InstantiatedRentalArchitecture
RentalProductLineArchitecture
i_components : *Component*©
i_relationships : *Relationship*©

i_components = {ic_member, ic_inventory, ic_rental}
i_relationships = {rel1, rel3}
∀ *rel1, rel3* : *Relationship*© •
∃ *ic_member, ic_inventory, ic_rental* : *Component*©
⇒ *rel1 = ic_member || ic_rental* ∧
rel3 = ic_inventory || ic_rental

*InstantiatedRentalArchitecture*는 도서 대여 시스템의 아키텍처로 *RentalProductLineArchitecture*를 인스턴스화하여 생성하였다. 인스턴시에이션 시에 예약관련 컴포넌트를 삭제하여 총 3개의 컴포넌트(*ic_member, ic_inventory, ic_rental*)와 2개의 관계(rel_1, rel_3)로 구성된다. 3개의 컴포넌트에 대한 내부 명세는 다음과 같다.

IC_Member
ipMember : Interface ©
member, vipMember, memberController : Class ©
memberRel : Relationship ©
memberWF : Workflow ©

IC_Inventory
ipInventory, icInventory : Interface ©
product, item, inventoryController : Class ©
inventoryRel : Relationship ©
inventoryWF : Workflow ©

IC_Rental
ipRental, icRental, irMember, irInventory : Interface ©
rental, rentalController : Class ©
rentalRel : Relationship ©
rentalWF : Workflow ©

사례 연구를 통해 대여 PLA를 도서 대여 아키텍처로 인스턴스화하는 과정을 정형적으로 보여주었다. 도서 대여 시스템은 예약 기능을 사용하지 않기 때문에 도서 대여 아키텍처 생성시에 C_Reservation 컴포넌트와 이와 관련된 연관관계를 삭제하였다. 이렇듯, 인스턴스화 과정에 정형명세를 적용함으로써 대여 PLA의 내부 구성요소와 가변점에 대해 명확히 보여주며, 의사결정모델이나 의사결정 해소모델의 정의를 명확하게 해준다. 또한, 의사결정해소모델의 정형명세를 기반으로 정의된 인스턴스화 규칙을 적용함으로써 핵심자산이 올바르게 인스턴스화될 수 있게 해주며, 의사결정해소모델 스키마를 통해 PLA의 설계 내용이 어떻게 인스턴스화된 아키텍처로 변환되는지를 추적하기 용이하게 해준다. 따라서, 인스턴스화된 아키텍처는 인스턴스화의 요구사항을 정확하게 반영할 수 있어 그 품질을 높이며 인스턴스화의 효율성이 높아진다. 또한, 수행작업과 해소효과를 이용한 해소 규칙은 PLE에서 핵심자산을 인스턴스화할 컴포넌트 내부에 대한 해소규칙으로 확장하여 활용될 수 있으며 인스턴시에이션 자동화에 활용될 수 있다.

7. 검 증

본 장에서는 정형명세를 사용하여 제안한 PLA 인스턴스화 규칙이 정확하게 정의되었는지를 검증한다. 검증 방법은 PLA의 정형명세 스키마와 인스턴스화된 아키텍처의 정형명세 스키마를 매핑하고, 인스턴스화된 정형명세가 4장에서 제시한 인스턴스화된 아키텍처 스키마와 동일한 구조를 가지는지 알아본다. 그림 5는 PLA 구성요소 중에서 특별히 컴포넌트를 인스턴스화하는 과정을 검증하는 매핑을 나타낸 것이다.

tFS_IA 의 의사결정 해소모델 스키마는 의사결정모델에 특정 어플리케이션의 요구사항을 적용하여 생성함으로써 명세된다(매핑 1-1). 의사결정모델로부터 가변점과 가변치가 선택되고 가변성 타입과 가변성 범위가 결정된다. tFS_IA 의 인스턴스화된 아키텍처 스키마내의 컴포넌트는 FS_PLA 내의 컴포넌트를 매핑 1-3의 의사결정 해소모델을 적용하여 명세된다(매핑 1-2). 이 컴포넌트는 매핑 2-1을 통해 연결된 FS_IA의 인스턴스화된 아키텍처 스키마내의 컴포넌트와 동일하다.

PLA의 인스턴스화 중에서 특별히 컴포넌트의 인스턴스화는 다음과 같이 검증된다.

FS_PLA에서 PLA 내의 컴포넌트 집합:
$\Sigma Comp\ (FS\_PLA)$
FS_PLA에서 의사결정모델내의 가변점의 집합:
$\Sigma vp_i\ (FS\_PLA)$
FS_PLA에서 의사결정모델내의 가변치의 집합:
$\Sigma v_i\ (FS\_PLA)$
FS_PLA에서 의사결정모델내의 해소 프로세스의 집합:
$\Sigma rp_i\ (FS\_PLA)$
tFS_IA에서 인스턴스화된 아키텍처내의 컴포넌트 집합:

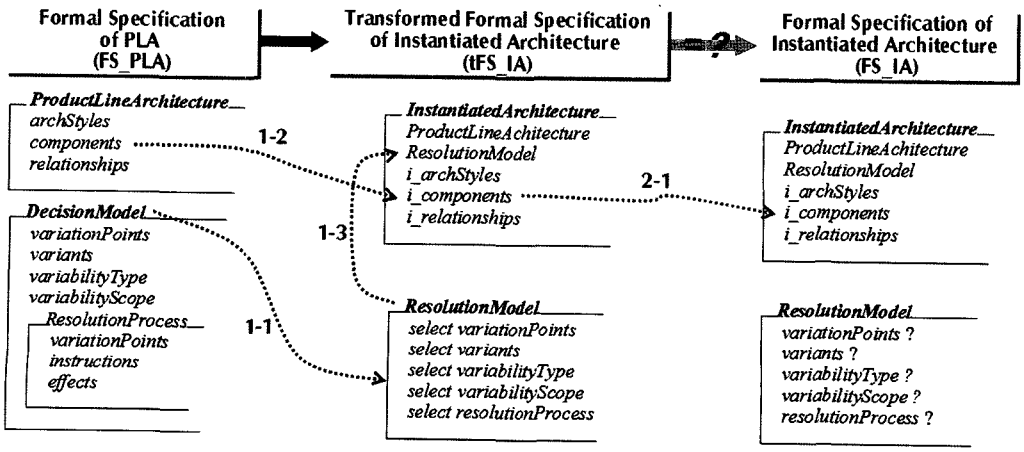


그림 5 컴포넌트 명세의 인스턴스화

$\Sigma iComp\ (tFS\_IA)$

FS_IA에서 인스턴스화된 아키텍처내의 컴포넌트 집합: $\Sigma iComp\ (FS\_IA)$

tFS_IA에서 의사결정 해소모델내의 가변점의 집합: $\Sigma vp_i\ (tFS\_IA)$

tFS_IA에서 의사결정 해소모델내의 가변치의 집합: $\Sigma v_i\ (tFS\_IA)$

tFS_IA에서 의사결정 해소모델내의 해소 프로세스의 집합: $\Sigma rp_i\ (tFS\_IA)$

↾ : 의사결정모델에서 해소 집합을 선택하기 위한 추출 연산자

의사결정 해소모델의 가변점의 집합 $\Sigma vp_i\ (tFS_IA)$은 가변성 타입이 컴포넌트일 경우에 관한 가변점의 집합 $\Sigma vp_i\ (FS_PLA)$과 같다.

$$\Sigma vp_i(tFS\_IA) = \Sigma vp_i(FS\_PLA) \upharpoonright (variabiltyType = component) \quad (1)$$

의사결정 해소모델의 가변치의 집합 $\Sigma v_i\ (tFS_IA)$은 가변점이 집합 $\Sigma vp_i\ (tFS\_IA)$ 안에 있는 가변치의 집합 $\Sigma v_i\ (FS_PLA)$과 같다.

$$\Sigma v_i\ (tFS\_IA) = \Sigma v_i\ (FS\_PLA) \upharpoonright (variationPoints = \Sigma vp_i\ (tFS\_IA)) \quad (2)$$

의사결정 해소모델의 해소 프로세스의 집합 $\Sigma rp_i\ (tFS_IA)$은 가변치가 집합 $\Sigma v_i\ (tFS\_IA)$ 안에 있는 해소 프로세스의 집합 $\Sigma rp_i\ (FS_PLA)$과 같다.

$$\Sigma rp_i\ (tFS\_IA) = \Sigma rp_i\ (FS\_PLA) \upharpoonright (variants = \Sigma v_i\ (tFS\_IA)) \quad (3)$$

만약 해소 프로세스의 집합 $\Sigma rp_i\ (tFS\_IA)$ 내에 n번째 해소 프로세스가 존재한다면, 인스턴스화된 아키텍처내의 컴포넌트 집합 $\Sigma iComp\ (tFS_IA)$은 n번째 해소 프로세스가 수행되었을 때 적용된 컴포넌트의 집합 $\Sigma Comp_{rpn}\ (FS_PLA)$이 된다.

$$\text{Let } \exists\ rp_n \in \Sigma rp_i\ (tFS\_IA) \cdot \Sigma iComp\ (tFS\_IA) = \Sigma Comp_{rpn}\ (FS\_PLA) \quad (4)$$

그리고, 인스턴스화된 아키텍처 명세로부터 정의된 가변점의 집합 $\Sigma vp_i\ (FS_IA)$은 가변성 타입이 컴포넌트인 가변점의 집합 $\Sigma vp_i\ (FS_PLA)$과 같다.

$$\Sigma vp_i\ (FS\_IA) = \Sigma vp_i\ (FS\_PLA) \upharpoonright (variabiltyType = component) \quad (5)$$

그러므로, (1)과 (5)에 의해 변환에 의해 생성된 가변점의 집합 $\Sigma vp_i(tFS_IA)$과 정의된 명세에 의해 표현된 가변점의 집합 $\Sigma vp_i(FS_IA)$은 같다. 또한, 관련된 가변치의 집합과 해소 프로세스의 집합도 같다.

$$\text{By (1), (5) } \Sigma vp_i(tFS\_IA) = \Sigma vp_i(FS\_IA) \wedge \Sigma v_i(tFS\_IA) = \Sigma v_i(FS\_IA) \wedge \Sigma rp_i(tFS\_IA) = \Sigma rp_i(FS\_IA)$$

해소 프로세스의 집합 $\Sigma rp_i\ (FS\_IA)$ 내에 있는 k번째 해소 프로세스에 대하여, 인스턴스화된 컴포넌트의 집합 $\Sigma iComp\ (FS_IA)$은 k번째 해소 프로세스가 수행되었을 경우의 컴포넌트 집합 $\Sigma Comp_{rpk}\ (FS_PLA)$과 같다.

$$\text{Let } \exists\ rp_k \in \Sigma rp_i\ (FS\_IA) \cdot \Sigma iComp\ (FS\_IA) = \Sigma Comp_{rpk}\ (FS\_PLA)$$

그러므로 (4)에 의해, ΣiComp (FS_IA) = ΣiComp (tFS_IA)은 참(True)이다.

8. 결 론

PLE는 최근 각광받고 있는 효율적인 소프트웨어 재사용 접근 방법 중 하나로 핵심자산을 이용한 높은 생산성을 보장한다. PLA가 핵심자산의 중요한 요소이나 이에 대한 상세한 구성 요소와 체계적인 인스턴스화 프로세스가 명확히 제안되어 있지 않았다. 본 논문에서는, PLA의 구성요소와 인스턴스화 프로세스를 상세히 정의하기 위해 정형명세를 적용하였다. PLA의 메타모델을 제시한 후, 정형명세 언어인 Object-Z를 이용하여 명세하는 방법을 제시하였으며 PLA를 인스턴스화된 아키텍처로 변환하기 위한 인스턴스화 규칙을 정형명세를 이용하여 정의하였다. PLA의 구성요소를 정형명세로 정의함으로써 PLA의 구성요소를 상세하고 정확하게 정의할 수 있고 설계간의 일관성을 보장할 수 있어 PLA를 기반으로 하여 생성된 최종 어플리케이션의 품질을 향상시킬 수 있다. PLA 인스턴스화를 위한 규칙을 정형적으로 정의함으로써 PLA와 인스턴스화된 아키텍처 사이의 추적성을 보장할 수 있으며 어플리케이션 요구사항이 인스턴스화된 아키텍처에 정확히 반영되었는지를 검증할 수 있어 인스턴스화된 아키텍처의 신뢰성을 높일 수 있다. 또한, 본 연구는 모델 주도형 구조(Model Driven Architecture, MDA)와 PLE를 통합한 어플리케이션 자동 생성에 활용될 수 있다. 핵심자산 공학의 최종 산출물을 플랫폼 독립 모델(Platform Independent Model, PIM)로 표현하여 이를 인스턴스화한 후 플랫폼 종속 모델(Platform Specific Model, PSM), 코드 등으로 자동변환하여 어플리케이션을 자동생성할 때, PLA를 도구에서 인식 가능한 PIM 형태로 표현하기 위해 본 연구의 PLA 정형명세가 활용될 수 있다. 또한, 정형명세로 정의된 인스턴스화 규칙은 인스턴스화 자동화도구를 개발할 때에 활용될 수 있다. PLA에 대한 정의와 인스턴스화 프로세스를 정형적으로 정의함으로써 PLE의 다양한 활동들을 더욱 상세하고 정확히 수행할 수 있다.

참 고 문 헌

[1] 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, pp.143-168, 1998.
[2] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.
[3] Clements, P., et al., *Software Product Lines*, Addison-Wesley, 2002.
[4] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, 2000.
[5] Smith, G., *The Object-Z Specification Language*, Kluwer Academic Publishers, 2000.
[6] Smith, G., "An Object-Oriented Approach to Formal Specification," Doctoral Thesis, University of Queensland, 1992.
[7] Shin, S. K., Lee, J. K., and Kim, S. D, "Techniques to Transform Object-Oriented Design into Component-Based Design Formal Specifications Using Formal Specifications," *Journal of Korea Information Science Society(KISS): Software and Applications*, 31(7):883-900, 2004.
[8] Bayer, J., et al., "PuLSE: A Methodology to Develop Software Product Lines," *Proceedings of the Symposium on Software Reusability (SSR'99)*, Los Angeles, 1999.
[9] Deelstra, S., Sinnema, M., and Bosch, J., "Product derivation in software product families: a case study," *The Journal of Systems and Software*, Vol.74, No.2, p.174-194, Jan. 2005.
[10] 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, Nov. 2004.
[11] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, Addison Wesley, 2003.
[12] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method," VTT publication 456, VTT Technical Research Center of Finland, ESPOO2002, 2002.
[13] Bass, L., Clements, P., and Kazman, R., *Software Architecture in Practice*, Addison-Wesley, 1998.
[14] Thiel, S., and Hein, A., "Systematic Integration of Variability into Product Line Architecture Design," *Lecture Notes in Computer Science 2379, Proceedings of SPLC2*, Springer-Verlag Berlin Heidelberg, 2002.
[15] Kim, S., Her, J., and Chang, S., "A theoretical foundation of variability in component-based development," *Journal of Information and Software Technology*, 2005, To Appear.
[16] Geyer, L., and Becker, M., "On the Influence of Variabilities on the Application-Engineering Process of a Product Family," *Lecture Notes in Computer Science 2379, Proceedings of the 2nd Software Product Line Conference*, 2002.

신 숙 경

1986년 이화여대 수학교육과 이학사 1999년 숭실대 정보산업학과 공학석사 2005년 숭실대 컴퓨터학과 공학박사. 현재 한국학술진흥재단 학술정보팀장 및 숭실대 겸임교수. 관심분야는 정형명세, 객체지향소프트웨어공학(OOSE), 컴포넌트기반개발(CBD), 제품계열공학(PLE), 모델기반아키텍처(MDA)

허 진 선

정보과학회논문지: 소프트웨어 및 응용 제 33 권 제 3 호 참조

김 수 동

정보과학회논문지: 소프트웨어 및 응용 제 33 권 제 1 호 참조