소프트웨어 프로덕트 라인에서 가변성 분석을 통한 도메인 아키텍처 개발 방법

(An Approach to Developing Domain Architecture Based on Variability Analysis in Software Product Line)

문 미 경 † 염 근 혁 ††
(Mikyeong Moon) (Keunhyuk Yeom)

요 약 소프트웨어 프로덕트 라인을 구축하고자 할 때, 첫 단계 활동은 도메인 분석을 통해 요구사항들에서 가변성을 식별하는 것이고, 다음 단계는 일련의 관련된 프로덕트들의 전체적인 구조를 나타내는 도메인 아키텍처를 개발하는 것이다. 도메인 아키텍처는 소프트웨어 프로덕트 라인에 포함되어 있는 프로덕트들의 공통성과 가변성을 기술함으로써 프로덕트 라인의 핵심자산이 된다. 핵심자산의 가변성은 개발 프로세스가 진행됨에 따라 식별될 수 있는 가변 요소의 종류와 상세화 수준이 달라지기 때문에 아키텍처 수준에서 식별될 수 있는 가변성을 정의하고 이를 체계적으로 식별하여 아키텍처 모델에 명시적으로 표현하는 것이 중요하다. 아키텍처 수준에서 고려해야 하는 가변성은 아키텍처 구성 요소들에서 발생하는 가변성 뿐만 아니라 이들의 구성(configuration) 관계를 나타내는 모델에서 나타나는 가변성들까지 고려해야 하기 때문에, 이들 사이의 복잡한 관계를 이해하고 표현하는 것은 매우 힘든 일이며 이에 대한 기존 연구가 부족한 실정이다.

본 논문에서는 공통성과 가변성이 명시적으로 고려되는 프로덕트 라인의 핵심 자산으로서 도메인 아키텍처를 개발하는 방법을 제시한다. 이를 위해, 최근 Object Management Group(OMG)에서 채택한 재사용 자산 명세(Reusable Asset Specification; RAS) 모델을 확장하여 공통성과 가변성 개념이 명확히 정의된 도메인 아키텍처 메타모델을 제시한다. 제시되는 메타모델에는 아키텍처의 구성요소들이 정의되어 있으며, 각 구성요소와 모델에서 식별 될 수 있는 가변성이 상세화 수준에 따라 두 가지 형태로 구분되어 제시되어 있다. 또한 본 메타모델을 기반으로 특정 도메인에 대한 아키텍처에 가변성이 명시적으로 표현되는 방법을 보인다.

키워드 : 소프트웨어 프로덕트 라인, 핵심자산, 도메인 아키텍처, 가변성 분석

Abstract When the decision to initiate a software product line has been taken, the first step is the domain analysis describing the variability in the requirements, the second important step is the definition of a domain architecture that captures the overall structure of a series of closely related products. A domain architecture can be a core asset in product line by describing the commonalities and variabilities of the products contained in the software product line. The variabilities, which are identified at each phase of the core assets development, are diverse in the level of abstraction. Therefore, it is important to clearly define, systematically identify, and explicitly represent variability at the architectural level. However, it is difficult to identify and represent the variability which should be considered at the architecture level, because these may be appeared in architecture elements and in architecture configuration.

In this paper, we suggest a method of developing domain architecture as a core asset in product line where commonality and variability are explicitly considered. First of all, we will describe a domain architecture metamodel that can explicitly define commonality and variability concepts by extending the Object Management Group's (OMG™) Reusable Asset Specification (RAS) model. Using the

· 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음
† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
mkmoon@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
yeom@pusan.ac.kr
논문접수 : 2005년 2월 14일
심사완료 : 2007년 2월 12일

domain architecture metamodel, architecture elements are defined and the variations that can be identified at the architecture level are classified into two types in according to abstract level. Additionally, we describe a domain architecture where commonality and variability are explicitly considered on basis of this metamodel.

Key words : software product line, core asset, domain architecture, variability analysis

1. 서 론

소프트웨어 프로덕트 라인 공학은 소프트웨어 재사용에 대한 활동들을 미리 계획하고 이 계획에 따라 변화하는 시장의 요구사항에 맞추어 일련의 관련된 시스템들을 연속적으로 생산할 수 있도록 지원하는 방법이다. 이 방법에서 가장 중요한 것은 일련의 관련된 시스템들, 즉 도메인 내에서 다시 재사용될 공통된 부분들을 식별해내고, 시스템들마다 상이하게 나타나는 가변적 요소들을 분석하는 일이다. 도메인의 주요 산출물들은 분석된 공통성과 가변성(Commonality and Variability : C&V)을 명시적으로 표현함으로써 프로덕트 라인의 핵심자산이 된다.

소프트웨어 프로덕트 라인에서는 특히 아키텍처의 역할은 매우 중요하다[1]. 이는 시장의 요구사항에 맞추어 변화에 신속히 적응하고 필요한 컴포넌트를 선택적으로 조립하여 시스템을 빠르게 개발할 수 있어야 하기 때문에 컴포넌트가 플러그인 될 수 있는 프레임웍 역할을 하는 아키텍처 기술이 필수적이라고 할 수 있다. 아키텍처가 소프트웨어 프로덕트 라인에서 핵심자산이 되기 위해서는 소프트웨어 프로덕트 라인에 포함되어 있는 프로덕트들의 공통성과 가변성을 아키텍처 수준에서 명시적으로 기술하고 있어야 한다. 이와 관련한 기존 연구들은 특정 자산의 특성을 고려한, 즉 아키텍처 자산만의 가변성을 구분하여 정의하지 않았으며, 이로 인해 아키텍처 수준에서 자산의 구성요소 각각에 가변성을 명시적으로 나타내지 못하였다.

본 논문에서는 도메인 공학 프로세스에서 나오는 여러 산출물 중 도메인 아키텍처를 대상으로 이를 소프트웨어 프로덕트 라인에서 재사용 가능한 핵심자산으로 개발하는 방법에 대하여 제안한다. 먼저, 도메인 아키텍처에 C&V의 개념을 결합하기 위해 도메인 아키텍처 메타모델을 제시한다. 이 도메인 아키텍처 모델은 최근 Object Management Group(OMG)에서 채택한 재사용 자산 명세(Reusable Asset Specification: RAS) 모델[2]을 기반으로 한다. 도메인 아키텍처 메타모델을 기반으로 도메인 아키텍처 수준에서 나타날 수 있는 가변성의 유형을 분석한다. 정의한 도메인 아키텍처 개념에 따라 특정 도메인에서 가변성이 도메인 아키텍처에 명시적으로 표현되도록 아키텍처를 개발하는 방법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 본 논문의 전체 구성 틀인 계층적 메타모델링 구조를 설명한다. 이 중, 가장 상위에 위치하고 있는 RAS 모델에 대하여 3장에서 설명하고, 이 계층을 기반으로 확장된 모델 형태인, 두 번째 계층, 도메인 아키텍처 메타모델에 대하여 4장에서 설명한다. 도메인 아키텍처 메타모델을 기반으로 특정 도메인에 적용된(본 논문에서는 전자여행정보시스템(e-Travel Systems: e-TS) 도메인에 적용되었다.) 세 번째 계층, 도메인 아키텍처 모델을 5장에서 설명한다. 6장에서는 본 연구와 관련된 기존 연구들을 살펴봄으로써 본 연구와의 차별성을 설명하고, 마지막으로 7장에서 결론 및 향후 연구 과제를 제시한다.

2. 계층적 메타모델링 구조

그림 1은 본 연구에서 제시하는 4계층 메타모델링 구조이다. 이것은 일반적으로 널리 사용되고 있는 계층적 메타모델링 구조[3]에 기반을 둔다. 이 구조에서는 상위 계층의 요소들은 인접한 하위 계층의 모델을 나타내기 위한 요소들을 정의한다.

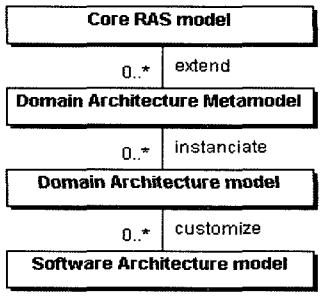


그림 1 4계층 메타모델링 구조

- ***Core RAS 계층***: 본 메타모델링 구조의 기반이 되는 층이다. 이 계층에서는 Core RAS 모델을 통한 자산에 대한 일반적인 표현을 기술한다.
- ***도메인 아키텍처 메타모델(Domain architecture metamodel) 계층***: 이 계층은 Core RAS 모델의 기본 구성체들을 도메인 아키텍처의 특성에 맞게 확장한 것이다. 도메인 아키텍처 메타모델에서는 도메인 아키텍처의 특성에 맞추어 구성 요소별 가변성 수준

및 타입을 명시한다.

- ***도메인 아키텍처 모델(Domain architecture model) 계층***: 이 계층은 도메인 아키텍처 메타모델의 인스턴스 층이다. 특정 도메인에 대한 도메인 아키텍처 모델들이 제공된다.
- ***소프트웨어 아키텍처 모델(Software architecture model) 계층***: 이 계층은 특정 애플리케이션의 요구사항에 맞게 도메인 아키텍처 모델을 커스터마이즈한 층이다. 특정 애플리케이션에 대한 소프트웨어 아키텍처 모델이 제공된다.

3. Reusable Asset Specification(RAS) Model

RAS는 최근 OMG에서 채택한 소프트웨어 자산 관리를 위한 공개표준(open standard) 명세서이다. 이 명세는 재사용 절차의 일관성을 위하여 재사용가능한 소프트웨어 자산들의 구조와 내용, 설명에 대하여 기술하고 있다. RAS는 크게 Core RAS와 프로파일로 분류되어 기술된다. Core RAS는 자산 명세에 대한 기본 구성요소들을 나타내며, 프로파일은 특정 형태의 자산들에 대한 추가적 의미를 소개하기 위해 Core RAS의 확장을 나타낸다. 그림 2는 RAS를 구성하는 주요 섹션과 요소들을 보여주고 있다. Core RAS는 classification 섹션, solution 섹션, usage 섹션, related assets 섹션으로 구성된다. classification 섹션은 자산을 분류하기 위한 일련의 기술서들을 열거하며, solution 섹션에서는 자산의 산출물들에 대하여 기술한다. usage 섹션에서는 자산을 사용하고 커스터마이즈하기 위한 규칙을 포함하고 있으며, related assets 섹션은 다른 자산들과의 관계를 기술한다. 본 논문에서는 특정 도메인 산출물에 대한 메타모델의 개발을 위하여 RAS의 solution 섹션의 모델을 사용한다.

그림 3은 Core RAS 모델 중, solution 섹션 부분의

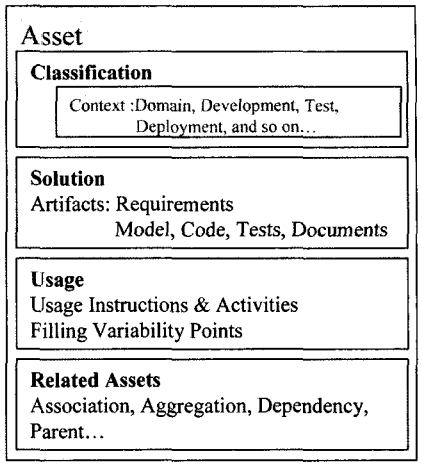


그림 2 Core RAS의 섹션요소

모델을 수정하여 간략히 도식화 한 것이다. 자산은 하나 이상의 산출물들로 구성되며, 하나의 산출물은 다른 산출물을 포함할 수도 있고, 다른 산출물과 관계를 가질 수도 있다. 하나의 산출물은 요구사항 개발, 설계 또는 런타임 문맥과 같은 특정 문맥(artifact context)과 관련된다. 산출물은 산출물 구성요소(artifact constituent)와 모델의 형태로 특수화(specialization) 될 수 있다. 산출물 구성요소는 재사용 될 때, 수정될 수 있는 지점을 나타내는 가변점(variability point)을 가지기도 한다. 모델은 또 다른 모델, 또는 다이어그램, 명세서로 구성되며, 각 형태에 따라 산출물 구성요소가 내포하고 있는 가변점을 표현할 수 있어야 한다.

4. 도메인 아키텍처 메타모델 정의

도메인 아키텍처는 일련의 아키텍처적 결정들, 일련의 재사용 가능한 컴포넌트들 그리고 어떤 특정한 클라이

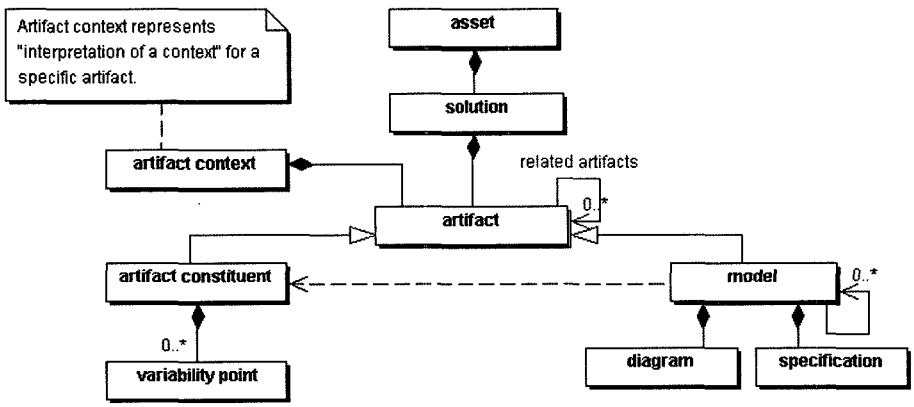


그림 3 RAS의 solution 모델

언트의 요구사항에 대응하는 선택적 컴포넌트들로 구성된다. 도메인 아키텍처는 도메인 내 시스템들의 공통적이며 전체적인 구조를 가변성을 포함하여 제공해야만 한다. 비록 소프트웨어 아키텍처는 하나의 시스템만을 대상으로 하는 반면, 도메인 아키텍처는 관련된 일련의 시스템들을 대상으로 하지만, 소프트웨어 아키텍처가 가지고 있는 뷰, 모델, 이해관계자의 관심의 분리, 컴포넌트, 이들의 상호작용과 같은 개념들이 도메인 아키텍처에도 반영되어 진다.

4.1 도메인 아키텍처 메타모델

그림 4는 도메인 아키텍처 정의를 바탕으로 그림 3을 확장하여 도식화한 도메인 아키텍처 메타모델이다. 그림 3에서 표현된 각 요소들에 해당하는 도메인 아키텍처 요소를 나타내기 위하여 그림 3의 클래스 이름이 스테레오 타입으로 사용되었다. 도메인 아키텍처는 여러 가지 형태의 산출물들로 구성된다. 이러한 산출물들은 자산이 재사용 될 때, 수정될 수 있는 지점을 나타내는 가변점을 가진다. 도메인 아키텍처 산출물은 크게 도메인 아키텍처를 구성하는 구성체들과 이들의 관계 및 세부사항을 나타내는 모델, 다이어그램, 명세서로 구성된다.

그림 4의 왼쪽 편에 위치하고 있는 도메인 아키텍처의 구성체는 다음과 같다.

- ***도메인 컴포넌트***: 이는 가변점을 갖추고 있는 논리적인 측면에서 기능들의 한 묶음으로 된 서비스를 의미하는 모델 구성요소이다. 도메인 컴포넌트는 하나 이상의 인터페이스로 분할되어 정의된다.
- ***도메인 컴포넌트 바인딩***: 도메인 컴포넌트 사이의 상호관계를 정의하기 위한 모델 구성요소이다.
- ***인터페이스***: 도메인 컴포넌트를 외부에서 접근하기 위해 정의해 놓은 오퍼레이션의 집합이다.
- ***오퍼레이션***: 각 오퍼레이션은 컴포넌트 인스턴스가 수행하게 될 어떤 서비스나 기능의 각 행위들을 의미한다.

그림 4의 오른쪽 편에 위치하고 있는 도메인 아키텍처 모델과 명세는 다음과 같다.

- ***도메인 구조 모델(Domain structure model)***: 도메인을 구성하는 도메인 컴포넌트간의 전체 구성을 보여준다. 이 모델을 구성하는 기본 요소로는 도메인 컴포넌트와 인터페이스, 그들 간의 관계가 있다. 이 모델에서 표현하는 관계는 도메인 컴포넌트와 그 컴포넌트가 제공하는 인터페이스 사이의 실체화(realization) 관계와 도메인 컴포넌트와 그 컴포넌트가 요구하는 인터페이스 사이의 의존 관계가 있다.
- ***도메인 행위 모델(Domain behavior model)***: 이 모델은 도메인 컴포넌트들 간의 다양한 상호작용을 정의하며, 도메인 컴포넌트의 인터페이스를 구성하는 오퍼레이션의 처리과정, 순서와 같은 상세한 정보를 표현한다. 이 모델을 구성하는 기본 요소는 도메인 컴포넌트 인스턴스와 특정 문맥에 속한 도메인 컴포넌트 인스턴스들이 주고받는 메시지들이다.
- ***도메인 컴포넌트 명세서(Domain component specification)***: 도메인 컴포넌트 명세에는 도메인 컴포

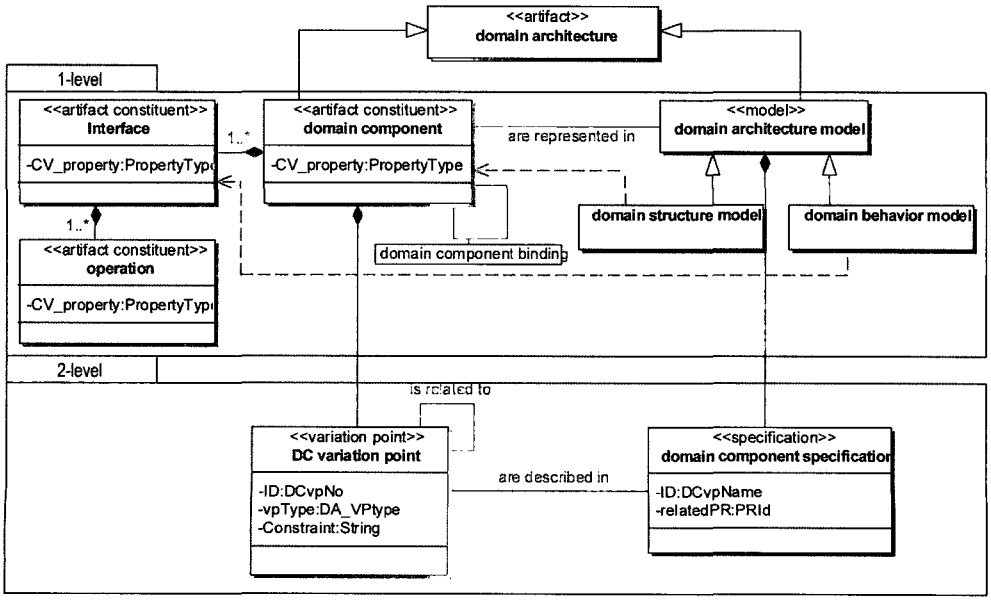


그림 4 도메인 아키텍처 메타모델

넌트에 대한 간략한 기술, 도메인 컴포넌트가 제공하고 요구하는 인터페이스에 대한 기술, 인터페이스에 정의하는 각 오퍼레이션에 관한 명세가 있다. 또한 도메인 컴포넌트에 포함된 도메인 클래스 및 하위 도메인 컴포넌트, 이들 간의 관계를 표현한다. 관련된 도메인 요구사항을 표현함으로써 요구사항과 컴포넌트간의 추적을 가능하게 하며, 도메인 컴포넌트 구현 시 나타나는 가변성에 대해 기술된다.

4.2 도메인 아키텍처 가변성 정의

도메인 아키텍처 가변성은 추상화 정도에 따라 두 가지 형태로 실현된다. 상위 수준의 가변성은 그림 4에서 모델 구성체인 도메인 컴포넌트와 인터페이스, 오퍼레이션의 속성으로 명시되어 있는 CV_property로 나타난다. 이는 그림 4의 1-level 패키지에 표현되어 있다. CV_Property는 기능의 공통성과 선택성을 나타내는 속성으로 도메인 아키텍처를 구성하는 모델 구성체가 도메인 아키텍처 산출물에 반드시 존재해야 하는 여부를 나타낸다. 도메인 아키텍처 하위 수준의 가변성은 가변점으로 나타난다. 가변점은 변화가 일어날 가능성이 있는 하나 이상의 위치로 정의된다. 이는 그림 4의 2-level 패키지에 표현되어 있다.

4.2.1 1-level 도메인 아키텍처 가변성

도메인 아키텍처 산출물에서 식별되는 가변성 중, 구성체에서 발견되는 상위수준의 가변성 유형은 다음과 같다. 이는 애플리케이션 개발 시, 도메인 컴포넌트 그 자체의 실현 가능성과 도메인 컴포넌트의 정의에 대한 가변성을 표현하게 된다.

- ***도메인 컴포넌트 CV_property 가변성***: 요구사항에서부터 선택적 속성으로 분석된 기능들이 하나의 도메인 컴포넌트로 추출된 경우, 이 도메인 컴포넌트는 아키텍처 상에 선택적으로 나타나게 된다. 이와 같이 도메인 컴포넌트가 애플리케이션 개발 요구사항에 따라 선택적으로 나타날 수 있다면 이 도메인 컴포넌트는 도메인 아키텍처 수준에서 가변점이 된다.
- ***도메인 컴포넌트 바인딩 CV_property 가변성***: 도메인 컴포넌트 바인딩의 가변성은 직접, 또는 간접적으로 발생하게 된다. 직접 발생하는 바인딩 가변성은 두 개의 도메인 컴포넌트 사이의 연관 관계를 추가하거나 삭제함으로써 생기는 경우이다. 간접 발생하는 바인딩 가변성은 도메인 컴포넌트 가변성에 연관되어 있는 컴포넌트의 바인딩인 경우, 도메인 컴포넌트의 가변으로 인해 바인딩이 같이 추가, 삭제되는 경우이다. 이와 같이 직간접적으로 도메인 컴포넌트 사이의 연관관계가 가변된다면 이는 도메인 아키텍처 수준에서 바인딩 가변점이 된다.
- ***인터페이스 CV_property 가변성***: 도메인 컴포넌트가 제공해야 하는 기능의 변화로 인해 이 도메인 컴포넌트가 제공하는 인터페이스가 공통적이고 선택적일 수 있는 경우에 이 인터페이스는 가변점을 가지게 된다.
- ***오퍼레이션 CV_property 가변성***: 인터페이스에서 기능의 추가 삭제는 오퍼레이션의 추가 삭제를 의미하는 것이다. 이 때, 인터페이스가 가지고 있는 오퍼레이션 자체가 선택적인 경우에 이 오퍼레이션은 가변점이 된다.

도메인 구성체의 가변성 속성인 CV_property에 따라 도메인 아키텍처의 모델의 변경이 발생하게 된다. 도메인 컴포넌트와 도메인 컴포넌트 바인딩의 CV_property에 따라 도메인 구조 모델이 가변(도메인 구성 스타일 가변성)되며, 인터페이스의 실현(realization) 유형에 따라 도메인 행위모델이 가변(도메인 컴포넌트 상호작용 가변성)된다.

- ***도메인 구성 스타일(Domain Configuration Style) 가변성***: 이는 도메인 컴포넌트와 도메인 바인딩의 가변으로 인하여, 도메인 구성의 일부분이 가변되는 경우이다.
- ***도메인 컴포넌트 상호작용(Domain Component Interaction) 가변성***: 도메인 컴포넌트가 제공하는 각각의 기능(오퍼레이션)들은 다른 도메인 컴포넌트와 상호작용하여 구현된다. 이 때, 도메인 컴포넌트 사이의 흐름의 패턴이 가변되는 경우, 이를 도메인 컴포넌트 상호작용 가변성으로 정의한다.

4.2.2 2-level 도메인 아키텍처 가변성

도메인 아키텍처 산출물에서 식별되는 가변성 중, 구성체에서 발견되는 하위수준의 가변성 유형이 다음과 같은 요소들로 이루어진 가변점으로 표현된다. 이는 도메인 컴포넌트의 구현에 관련된 가변성을 표현하게 된다.

- ***가변점 타입 (vpType)***: 도메인 컴포넌트 내부에 가변점이 설계되어진 유형에 대하여 기술한다. 일반적인 설계 방식으로 Information hiding, Parameterization, 또는 Inheritance 등이 있다[4].
- ***제약사항(constraint)***: 도메인 컴포넌트 내부에 설계되어진 가변점들에 대한 전후 의존관계들을 표현한다. 사전 의존관계는 본 가변점이 의존하고 있는 도메인 요구사항에서의 가변점 또는 다른 도메인 컴포넌트가 가지는 가변점들과 연결된다. 사후 의존관계는 본 가변점으로 인해 의존하게 되는 다른 가변점 또는 다른 도메인 컴포넌트와 연결된다.

5. 도메인 아키텍처 모델

본 장에서는 5장에서 설명한 도메인 아키텍처 메타모델을 바탕으로 전자 여행 정보 시스템(*e-Travel Systems: e-TS*)의 도메인 아키텍처에 대하여 설명한다. e-TS는

전자 여행 예약 서비스 시스템으로 고객이 웹 기반으로 여행관련 상품(비행기, 호텔, 차 등)의 정보를 조회하고 예약할 수 있으며, 자신의 예약 정보를 수정, 삭제할 수 있는 기본 기능을 제공한다. 또한 선택적으로 고객의 의향에 따라 조금씩 변경될 수 있는 맞춤형 여행 상품을 예약할 수 있는 기능이 제공되기도 한다. 이러한 기능들은 e-TS 도메인 분석 시 PR-Context 매트릭스를 통해 분석된다. PR-Context 매트릭스는 도메인 내의 요구사항들을 PR단위로 수집한 결과물이다[5]. 다음 그림은 e-TS 도메인의 PR-Context 매트릭스의 일부분을 보여준다. 이 매트릭스를 통해 PR의 공통성과 선택성을 나타내는 CV_property 가변성을 확인할 수 있다. PR의 CV_property와 각 PR의 명세서에서 식별되는 가변점들은 도메인 아키텍처의 구성체인 도메인 컴포넌트 및 인터페이스, 오퍼레이션의 가변성에 직접적인 영향을 준다.

5.1 도메인 아키텍처 구성체의 가변성

- ***e-TS 도메인 컴포넌트 CV_property 가변성***: 도메인 컴포넌트는 도메인에서 제공해야 하는 기능들을 응집력 있는 한 단위로 묶은 것이다. 이는 도메인 요구사항의 PR-Context 매트릭스와 도메인에서 추출된 타입(type)들의 관계를 분석하여 도메인 타입들에 대한 생성, 소멸의 책임이 있는 PR들을 하나로 묶어서 추출된다(그림 6).

도메인 컴포넌트는 그림 6처럼 기능적으로 관련된 PR들로 구성되어 있기 때문에 PR의 CV_property에 따라 가변성의 속성이 결정된다. 도메인 컴포넌트의 CV_property 결정은 다음과 같은 기준에서 이루어진다.

- 도메인 컴포넌트의 공통성 속성 - 도메인 컴포넌트에 포함된 PR들의 속성이 모두 공통성인 경우, 또는 도메인 컴포넌트에 포함된 PR들의 속성이 혼합

LEGEND: C = Create U = Update R = Read □ = function package	Ratio	type1	type2	type3	type4	⋮	typem
Type / PR							
PR1	100%	C	C	R			
PR2	100%	C	U	R			
PR3	100%	U	R	R			
PR4	25%			C	R	R	
:				U	C	R	
PRn	10%			R	C	C	

그림 6 도메인 컴포넌트 추출

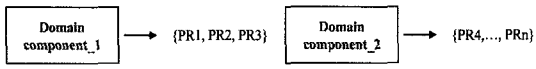


그림 7 도메인 컴포넌트와 PR들 간의 관계

되어 있는 경우 도메인 컴포넌트 속성은 공통성을 가진다. 이 속성은 도메인 컴포넌트의 디폴트 속성이기 때문에 모델에 따로 명시하지 않는다.

- 도메인 컴포넌트의 가변적 속성 - 도메인 컴포넌트에 포함된 PR들의 속성이 모두 선택성인 경우, 도메인 컴포넌트 속성은 선택성을 가진다. 모델에서는 명시적으로 UML의 스테레오타입을 사용하여 <<v.p>>로 표현한다.

그림 8은 e-TS 도메인에서 추출된 도메인 컴포넌트의 일부를 보여준다. 그림 5의 PR12, PR13, PR14를 포함하는 도메인 컴포넌트 *Customer Reservation Mgr*이 <<v.p>> 스테레오타입을 사용하여 명시적으로 가변적 도메인 컴포넌트임을 나타내고 있다.

- ***인터페이스 CV_property 가변성*** - 도메인 컴포넌트간 의존성을 인터페이스 사이의 의존관계로 줄이기 위해 컴포넌트의 기능을 하나 이상의 인터페이스로

PR No	PR	RATIO(%)	eT Portal	Tourmall...	C-Com...	Joy View	Online-...	TourE...
PR1	Login	100	O	O	O	O	O	O
PR2	Logout	100	O	O	O	O	O	O
PR3	Register	100	O	O	O	O	O	O
PR4	Modify the member information	100	O	O	O	O	O	O
PR5	Withdraw	100	O	O	O	O	O	O
PR6	Search for vacation packages	100	O	O	O	O	O	O
PR7	Book vacation packages online	100	O	O	O	O	O	O
PR8	Confirm a trip reservation	100	O	O	O	O	O	O
PR9	Cancel a trip reservation	100	O	O	O	O	O	O
PR10	Review a trip reservation	100	O	O	O	O	O	O
PR11	Modify a trip reservation	100	O	O	O	O	O	O
PR12	Make a DIY travel planning	28	O	O	X	X	X	X
PR13	Customize trip plan	28	O	O	X	X	X	X
PR14	Add featured attraction and services	28	O	O	X	X	X	X
PR15	Search for flights	100	O	O	O	O	O	O
PR16	Make a flight reservation	100	O	O	O	O	O	O
PR17	Confirm a flight reservation	100	O	O	O	O	O	O
PR18	Cancel a flight reservation	100	O	O	O	O	O	O
PR19	Review a flight reservation	100	O	O	O	O	O	O
PR20	Modify a flight reservation	100	O	O	O	O	O	O
PR21	Search for Hotels	100	O	O	O	O	O	O

그림 5 e-TS 도메인의 PR-Context 매트릭스의 일부분

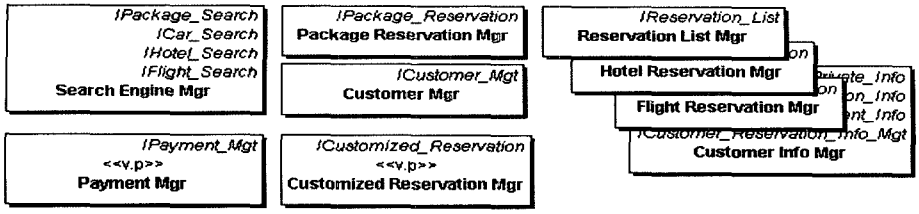


그림 8 e-TS 도메인에서 식별된 도메인 컴포넌트

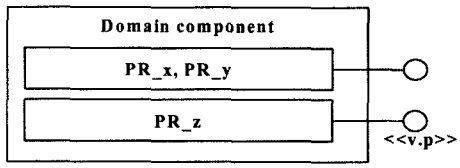


그림 9 가변적 인터페이스의 표현

나누어 정의한다. 이때, 그림 9와 같이 PR들의 속성이 혼합되어 있는 도메인 컴포넌트인 경우, 선택적 속성을 가진 PR들(그림 9에서 PR_z)의 기능을 나타내기 위한 인터페이스는 따로 분리하여 정의한다. 그리고 이러한 인터페이스는 PR의 속성에 의해 선택적으로 구현될 수 있는 기능이기 때문에 이를 명시적으로 UML의 스테레오타입을 사용하여 <<v.p>>로 표현한다.

그림 10은 *Customer Info Mgr* 도메인 컴포넌트가 제공하는 인터페이스를 나타내고 있다. *Customer Info Mgr* 도메인 컴포넌트는 고객 관련 정보를 관리하는 기능을 가진다. 이때, 고객의 e-money, 상품권, 적립금, 쿠폰 등에 대한 추가정보(supplement information)는 이러한 것을 이용한 지불이 허용되는 애플리케이션에서만 유지되어야 하는 정보들이다. 그러므로 *ICustomer Supplement Info* 인터페이스는 애플리케이션에서 선택적으로 실체화될 수 있기 때문에 <<v.p>> 스테레오타입을 사용하여 명시적으로 가변적 인터페이스임을 나타내고 있다.

- ***도메인 오퍼레이션 CV_property 가변성*** - 인터페이스는 서로 연관된 오퍼레이션들을 그룹핑한다. 이때, 오퍼레이션은 각각의 PR 또는 각 PR의 명세서(PR specification)에서 분석된 기능들을 정의한다. 그러므로 선택적 PR 또는 PR 명세서에서 가변점으로 식별된 기능은 오퍼레이션 또한 선택적으로 구현될 수 있기 때문에 이를 명시적으로 UML의 스테레오타입을 사용하여 <<v.p>>로 표현한다.

그림 11은 *Package Reservation Mgr* 도메인 컴포넌트가 *IPackage Reservation* 인터페이스를 통해 여행 패키지 상품에 대한 옵션 선택, 예약, 재검토, 수정, 취소의 기능을 제공하고 있음을 보여준다. 이 인터페이스가 가지고 있는 다양한 오퍼레이션들 중, *select_Package_option*은 패키지 상품 예약 시 옵션들을 선택할 수 있는 기능으로서, 애플리케이션 개발 시 선택적으로 구현될 수 있기 때문에 <<v.p>> 스테레오타입을 사용하여 명시적으로 가변적 오퍼레이션임을 나타내고 있다.

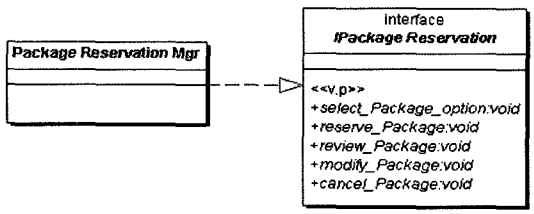


그림 11 *IPackage Reservation* 인터페이스의 오퍼레이션

5.2 도메인 아키텍처 모델의 가변성

5.2.1 도메인 구조 모델 (Domain structure model)

그림 12에서 보여주고 있듯이 도메인 구조 모델은 도메인을 구성하는 컴포넌트간의 전체 구성을 보여준다. 이 모델에서 도메인 컴포넌트가 가변점을 가진다는 의미(도메인 컴포넌트 CV_property 가변성)는 애플리케이션의 요구사항의 가변에 따라 이 도메인 컴포넌트가 이 모델에서 추가 삭제 될 수 있음을 의미하는 것이다. 또한 이 모델에서 도메인 컴포넌트 바인딩 CV_property

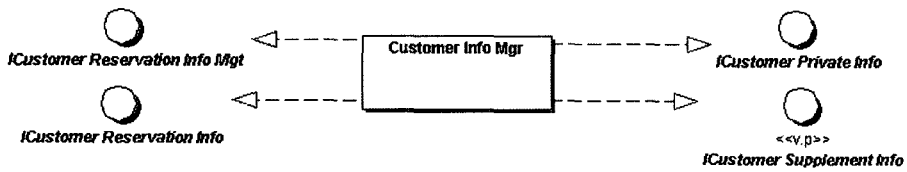


그림 10 *Customer Info Mgr* 도메인 컴포넌트의 인터페이스

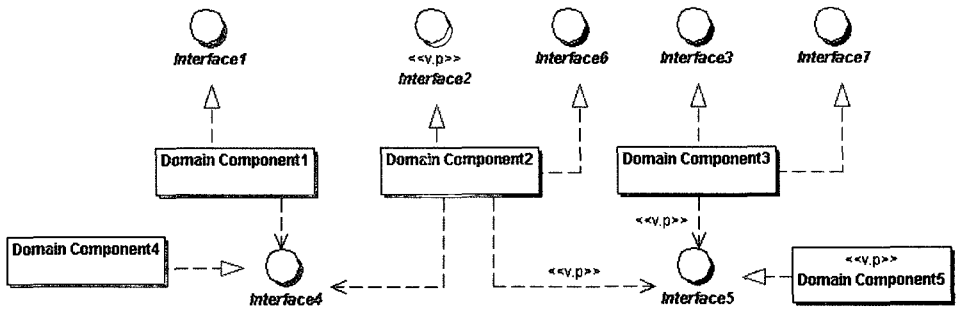


그림 12 도메인 구조 모델

가변점이 도메인 컴포넌트의 추가 삭제 및 가변적 인터페이스와의 의존관계로 인해 나타나게 된다. 그림 12에서 가변점을 가진 도메인 컴포넌트5, 이에 의존하고 있는 도메인 컴포넌트 바인딩, 도메인 컴포넌트2의 인터페이스2가 의존하고 있는 도메인 컴포넌트 바인딩들이 애플리케이션 개발에 따라 이 모델의 구성 스타일을 가변시키게 된다. 즉, 도메인 아키텍처 구성체의 CV_property 가변성에 직간접적으로 영향을 받는 일련의 가변점들로 인해 도메인 아키텍처 구성을 변화시키게 된다. 이는 도메인 아키텍처 구성 스타일의 가변성에 해당된다.

그림 13은 *Customer Mgr* 도메인 컴포넌트 관련한 도메인 구조 모델의 일부분을 보여준다. e-TS 도메인에서 고객은 등록 시 유무료회원을 선택할 수 있는 기능을 선택적으로 가지고 있다. 그러므로 애플리케이션 개발 시 선택적으로 나타날 수 있는 *Payment Mgr* 도메인 컴포넌트와 *IPayment_Mgt* 인터페이스를 사용하는 도메인 컴포넌트 바인딩에 <<v.p>> 스테레오타입을 사용하여 명시적으로 가변성을 표현하고 있다.

5.2.2 도메인 행위 모델(Domain behavior model)

도메인 행위 모델에서 나타나는 가변성은 도메인 컴포넌트 사이의 흐름의 가변성이다. 그림 14의 상위 모델에서는 도메인 컴포넌트1이 도메인 컴포넌트3을 통해서 도메인 컴포넌트4로 흐름을 전달하는 모습이다. 하위 모델은 도메인 컴포넌트4로 가는 흐름을 도메인 컴포넌트1에서 직접 가는 모습이다. 이와 같이 메시지 흐름의 패턴이 바뀌는 경우 발생할 수 있는 상호작용 흐름을 각각 구분하여 그려준다. 또한 컴포넌트로 가는 메시지 중에서 해당 기능의 호출이 가변적으로 발생하는 위치에서 가변성이 나타난다. 그림 14의 예에서 도메인 컴포넌트2가 가지고 있는 operation1은 가변적 속성을 가지고 있기 때문에 메시지 흐름에도 이것을 반영하여 나타내게 된다. 이러한 오퍼레이션의 가변성으로 인한 메시지 흐름의 가변은 메시지의 조건에 [v.p]를 표시하여 명시적으로 표현할 수 있다.

그림 15와 그림 16은 *Package_Reservation* 도메인 컴포넌트가 제공하는 기능들이 어떻게 동작하는가를 보여주는 도메인 행위 모델이다. 예약은 먼저 고객이 *Package item*을 검색한 후 이루어지기 때문에 VP_Search 도메인 컴포넌트로 가게 된다. *Package_Reservation*이 *Package item* 중 옵션을 선택하게 해 주는 기능이 있다면 *Package_Search*는 *select_Package_option*을 부르게 된다. 이 기능은 선택적으로 수행되는 기능이기 때문에 조건에 [v.p]가 명시적으로 나타나게 된다. 그림 16은 그림 15의 *reserve_Package*에 대한 상호작용의 패

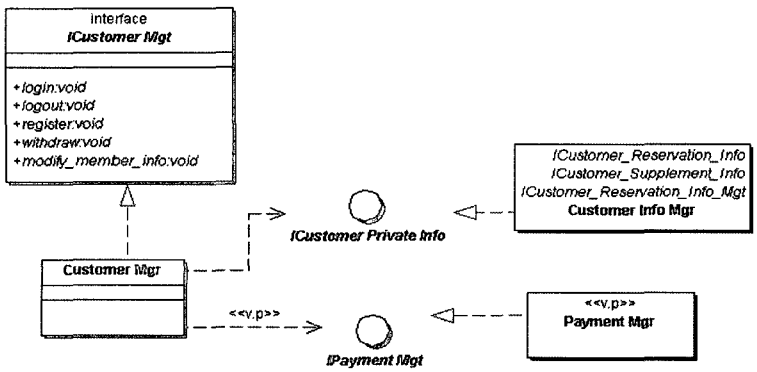


그림 13 *Customer_Mgr* 도메인 컴포넌트 관련 도메인 구조 모델

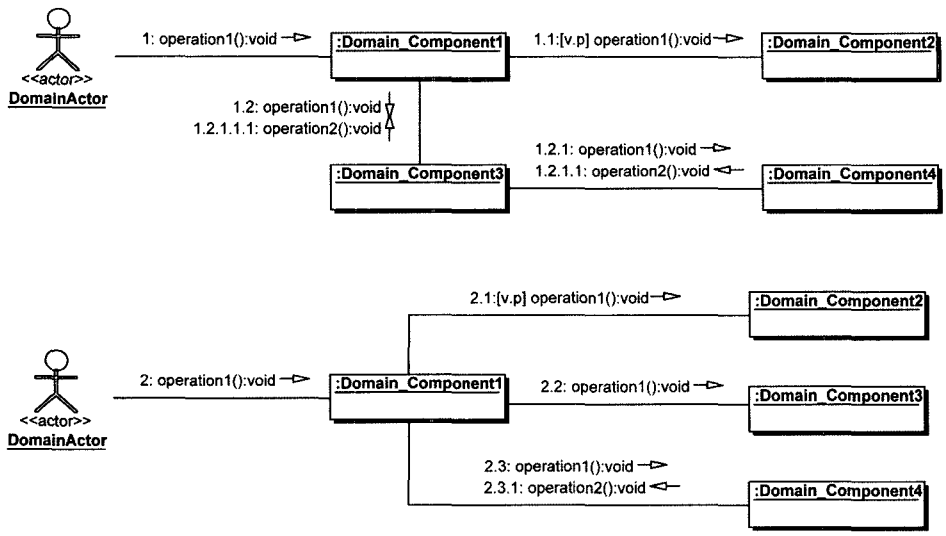


그림 14 도메인 행위 모델

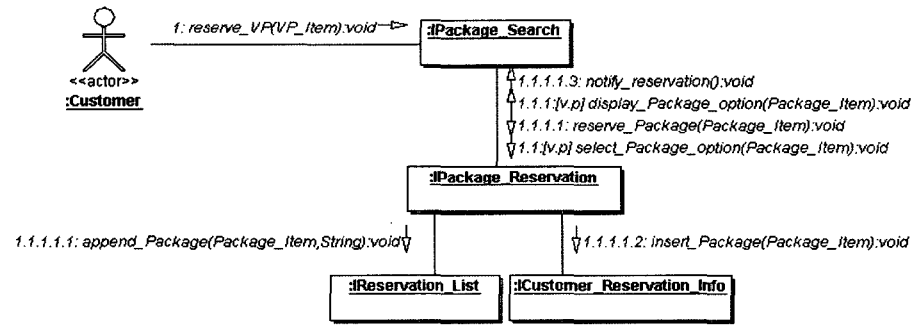


그림 15 *Package Reservation*을 위한 도메인 행위모델 패턴11

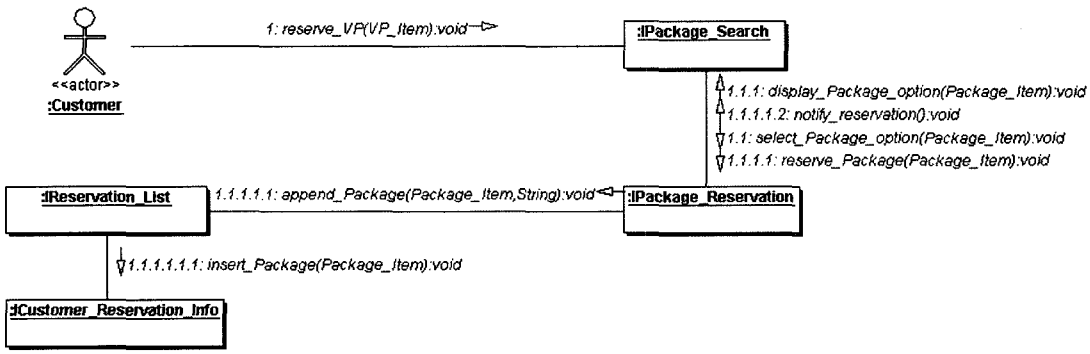


그림 16 *Package Reservation*을 위한 도메인 행위모델 패턴2

턴이 바뀐 예를 보여주고 있다. 예약 요청이 들어온 자료(*Package_item*)는 이 시스템 전체에서 다루는 예약 목록에 삽입 된다. 또한 이 정보는 고객 예약 정보에도 일관성 있게 들어가야 한다. 그러나 *Package_Reservation* 도메인 컴포넌트에서 예약 자료를 두 개의 도메인 컴포넌트에 동시에 넘겨 줄 수도 있지만, 그림 16처럼 전체 예약 목록을 수정한 후, 개인 예약 정보를 삽입하게 할 수도 있다. 이러한 메시지 흐름에 대한 가변성이 이 모델을 통해 각각 표현된다.

5.2.3 도메인 컴포넌트 명세서(Domain component specification)

도메인 컴포넌트의 CV_property가 공통성 또는 선택

성일지라도 내부 구현 시 발생할 수 있는 가변성을 포함할 수 있다. 도메인 컴포넌트 명세서에는 일반적인 컴포넌트 명세서에서 기술되는 사항들 - 인터페이스 기술, 오퍼레이션 기술 - 외에 관련 요구사항(Related PR), 가변점에 대한 기술과 컴포넌트 구현에 참여하는 클래스들의 구조와 상호작용 다이어그램이 포함된다. 그림 17은 *Customer_Mgr*에 대한 도메인 컴포넌트 명세서의 일부분이다. 이 도메인 컴포넌트는 공통성 CV_property를 가지며, 도메인 요구사항 중, 그림 5에서 보이고 있는 PR1, PR2, PR3, PR4, 그리고 PR5의 기능을 구현한다. 이 컴포넌트는 가변점을 가지는데, 이는 요구사항 PR3(*Register*)이 가지는 가변치 v1(*Membership without fee*)와 v2(*Membership with fee*) 중, v2가 선택이 되었을 때, 구현하게 될 부분임을 제약사항(Constraint) 항목에서 기술하고 있다. 이는 도메인 타입 모델에서 <<v.p-2>>로 명시되어 있다. 또한 이 가변점이 구현될 시, 의존해서 가변되어야 할 지점이 도메인 타입 다이어그램의 <<v.p-1>> 지점과 다른 도메인 컴포넌트 *Payment_Mgr* 임을 기술하고 있다. 동일 다이어그램 내의 가변점 사이의 의존관계를 명확히 구분하기 위해 가변점에 번호를 부여한다. 가변점 사이의 의존관계의 순서는 도메인 타입 다이어그램의 행위적 모습을 통해서도 확인할 수 있다.

6. 관련 연구

6.1 아키텍처 수준의 공통성과 가변성 분석

Mari Matinlassi는 [6]에서 COPA[7], FAST[8], FORM[9], KobrA[10], QADA[11]를 대상으로, 소프트웨어 프로덕트 라인 아키텍처에 대한 비교 평가 질문을 만든 후, 이들의 설계 방법들에 대하여 비교하였다. 비교평가 항목 중, 방법론이 가변성의 표현을 잘 지원하는가에 대하여, FORM과 FAST는 요구사항 추출단계에서 가변성에 대한 고려를 명시적으로 하지만, 다른 것들은 그렇지 못하다고 평가하였다. FORM(Feature-Oriented Reuse Method)은 Feature Oriented Domain Analysis (FODA) 방법[12]을 설계와 구현 단계로 확장한 것이다. 이 방법에서는 도메인 내의 공통성과 차이성을 식별하여 피쳐 모델로 나타내고, 이를 아키텍처 모델링 단계에서 아키텍처 모델과 적절히 대응시킨다. 그 결과 참조 아키텍처(reference architecture)와 컴포넌트를 생성한다. 이 방법에서 도메인 아키텍처의 가변성은 명시적으로 나타나있지 않으며 단지 피쳐 그래프와의 매핑을 통해서 암시적으로 가변성을 다루고 있다. FAST(Family-Oriented Abstraction, Specification and Translation)는 조직이 공통된 중요한 속성들 -행위, 인터페이스 또는 코드 등- 을 공유하는 다양한 버전의 제품을 생산할 때 유용하게 적용될 수 있는 프로세스이다. 이 방법은 일반적인 텍스트 형식의 표현법을 사용하여 명시적으로 도메인의 공통성과 가변성을 식별한다. 그러나 이를 매우 상위 수준에서 다루고 있으며 구체적인 구현기술 또는 아키텍처로의 연결에 대해서는 언급하지 않는다. 이로 인해 도메인 분석 절차에 대한 지침이 모호하고 규정적이지 못하다(unprescriptive)는 평가가 있다[10].

Hassa Gomma는 [13]에서 소프트웨어 프로덕트 라인의 가변성을 관리하기 위하여 다양한 뷰들에 대한 메타모델을 제시하였다. 이 논문에서는 소프트웨어 프로덕트 라인의 전 과정(요구사항 모델링, 분석, 설계)에 거쳐 산출되는 모든 모델들을 언급하고 이들 사이의 관계를 분석하여 연관 지었다. 그러나 각 단계별 식별되는 가변성의 유형이 모델별, 자산별 구분되어야 함에도 불구하고 이러한 구분은 전혀 고려하지 않았다. Pierre America는 [14]에서 프로덕트 라인에서 발생하는 다양한 가변성에 대하여 가변성 모델을 제시하고 이를 시나리오 기반으로 평가하는 방법을 제시하였다. 가변성 모델은 CAFCR 뷰 -Customer, Application, Functional, Conceptual, Realization 뷰- 내의 가변성을 다루기 위해 그려지는데, 그 모양은 피쳐 그래프의 형태와 유사하다. 이 방법은 아키텍처 평가 방법을 프로덕트 라인에 적용한 연구로서, 가변성에 따라 변화하는 아키텍처를 아키텍처 평가 방법을 통해 결정할 수 있도록 하였다. 이 방법은 아키텍처 뷰에 가변성 모델을 분리하여 가변성 정보를 유지하고 있는 형태를 보여주었다. COVAMOF (ConIP Variability Modeling Framework)[15]는 프로덕트 라인에서 가변성을 모델링 할 수 있는 프레임워크로서 모든 단계에서 가변점을 독립된 클래스 요소로서 나타내고 가변성에 대하여 계층적 조직을 지원하며, 그들 사이의 관계를 모델링 하도록 한다. 이는 피쳐 모델, 아키텍처, 컴포넌트 구현물들에 대하여 가변점 뷰(variation point view)와 의존 뷰(dependency view)로 구성된 가변성 뷰(Variability view)를 제공한다. 각 산출물들에 대한 가변성을 독립된 뷰를 통해 명시적이고 구체적으로 표현하도록 하였지만, 각 산출물들(예를 들어, 아키텍처)의 특징을 고려하여 가변성의 유형을 구분하지는 않고 있으며, 또한 산출물과 가변성 뷰의 연결에 대한 고려도 부족하였다.

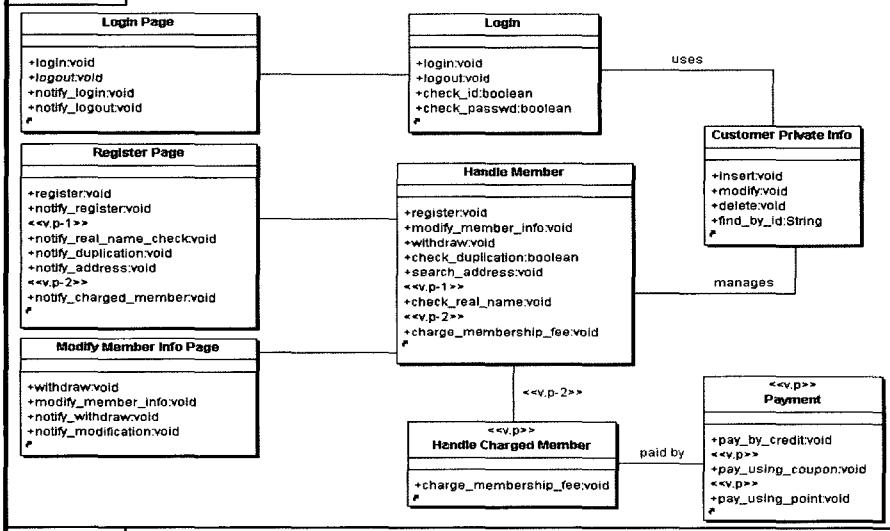
6.2 아키텍처 가변성 표현 방법

프로덕트 라인 아키텍처에서 가변성을 모델링하는 기법으로 패턴을 사용하는 연구가 있다[16]. 여기서 판별식(discriminant)은 하나의 시스템을 다른 시스템과 구분 짓는 어떤 피쳐를 의미하게 되고, 이를 패턴과 연관지어 가변성을 모델링한다. 이때, 가변성 또는 선택성이 발생하는 부분을 베이스 클래스로 표현하고, 가변되어지

ID	[DC:Customer_Mgr]	CV_property	Common
Related PR	PR1, PR2, PR3, PR4, PR5		
Variation Point	ID	DC3_vp1	
	vpType	Information hiding	
	Constraint	Variant v2 in PR3_vp1 [PR3.2] → <<v.p-2>>	
		<<v.p-2>> →<<v.p-1>> <<v.p-2>> →[DC:Payment Mgr]	
Interface Description	Provided Interface	ICusomer_Mgt	
	Required Interface	Icustomer_Private_Info <<v.p>>IPayment_Mgt	

Domain Type Model

Structure



Behavior

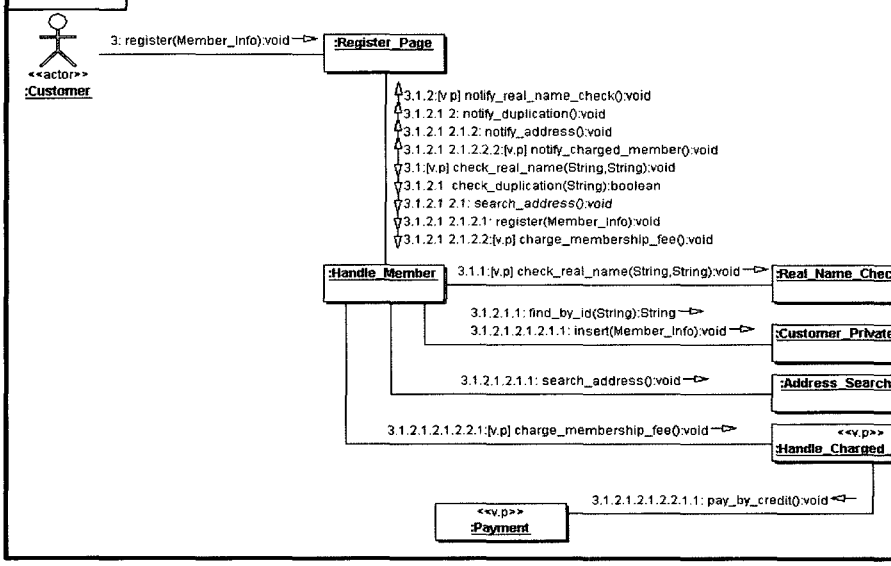


그림 17 *Customer_Mgr* 도메인 컴포넌트 명세서

는 값들을 서브 클래스의 집합인 영역(realm)으로 묶는다. 이때, 영역에 속하는 서브 클래스가 어떻게 결합되는지에 따라 single adapter pattern, multiple adapter pattern, optional pattern으로 구분한다.

UML 확장 기법을 사용하여 가변성을 모델링 한 연구도 있다[17]. 여기서는 UML 모델에 스테레오타입으로 <<variationPoint>>를 기술하고 각 가변치(variant)들을 상속관계로 표현한다. [18] 연구에서는 프로덕트 라인에서 가변성을 모델링하기 위해 매개변수화(parameterization), 정보 은닉(information hiding), 상속(inheritance), 가변성 포인터(Variation Point)를 사용하는 방법들을 제시하고 있다.

이러한 연구들은 아키텍처 수준에서 단지 컴포넌트의 선택과 대체하는 수준에서의 가변성만을 다루고 있다. 아키텍처 수준에서 가변점에 대응하는 가변치들은 하나의 모델링 요소(예를 들어, 클래스 단위)로 나열할 수 있는 유형의 가변성뿐만 아니라 연관된 요소들을 함께 고려해야 하는 가변성들이 많이 있다. 즉, 이 연구들은 가변성 유형에 대한 고려 없이 단지 모델링 기법에 초점을 두고 있으며, 더 상위 수준에서 아키텍처 구성의 가변성이나 더 상세한 수준에서 컴포넌트 내부에 포함하고 있는 가변성을 표현하지는 못한다.

6.3 관련 연구와의 비교 평가

기존의 방법들이 모두 공통적으로 도메인 아키텍처의 가장 핵심 요소로 가변성을 다루고 있다. 가변성을 표현하기 위한 기존 방법들은 그림 18과 같이 크게 두 가지 유형으로 분류할 수 있다. 여기서 기본모델(Base model)이란 단일 애플리케이션 개발 시 산출되는 모델과 같이 가변성이 표현되지 않는 형태를 의미한다.

- ***기본모델과 가변성을 함께 표현*** - 그림 18의 왼쪽에서 표현하고 있는 형태와 같이 기본모델에 가변성을 표현하기 위한 요소를 추가하여 특정 자산을 개발하는 것이다. 이러한 방법은 [9,10,13] 등의 연구에서 적용이 되었다. 이 방법은 특정 자산, 예를 들어 도메인 아키텍처 모델(DA Model)만을 고려할 때, 가변성을 나타내기 위한 확장 기법만 쉽게 이해할 수 있다면, 기존의 기본모델에서 나타내고자 하는 의미를 그대로 이용할 수 있어서 개발자에게 도메인 자산을 쉽게 이해시킬 수 있는 장점이 있다. 그러나 특정 하나의 자산뿐만 아니라 다른 자산들, 예를 들어, 도메인 요구사항 모델(DR Model), 도메인 컴포넌트 모델(DC Model))들을 함께 고려한다면, 이들 사이의 가변성의 관계를 일관성 있게 다루지를 못하게 되는 단점이 있다.
- ***기본 모델과 가변성 모델을 분리하여 표현*** - 이러한 접근 방법의 대표적인 연구는 [15,19] 등이 있다. 이 방법에서는 기본모델과 가변성 모델을 분리하여 표현한다. 그리고 하나의 가변성 모델을 기반으로 여러 자산의 기본모델에 연결 관계를 제공하고 있다. 이러한 방법은 가변성 모델을 하나의 독립된 모델로 분리함으로써 가변성만을 일관성 있게 다룰 수 있으며, 이로 인해, 가변성의 의존관계도 잘 관리할 수 있는 장점이 있다. 그러나 가변성은 개발이 진행됨에 따라 가변성이 정제되어 나타나게 되고 각 자산마다 가변성이 표현되어야 하는 자산의 요소가 달라지는데, 이를 표현할 방법이 부족하다는 단점이 있다.

그러므로 본 방법에서는 이러한 두 가지 접근 방법의 장점을 혼합하고자 다음 그림 19와 같은 접근 방법을 채택하였다. 먼저, 도메인 아키텍처는 두 번째 기존방법처럼 가변성을 일관성 있게 관리하기 위해 도메인 아키텍처 메타모델을 제시하였다. 이 메타모델은 상위의 자산 메타모델(RAS model)에 기반하고 있으며, 각 자산별 특징에 맞게 가변성을 표현할 수 있도록 확장하고 있다. 특히, 메타모델을 통해 도메인 아키텍처가 가지는 기본 개념들과 도메인 아키텍처 수준에서 나타날 수 있는 가변성의 유형을 명확히 구분하였다. 또한 도메인 아키텍처는 제시된 메타모델을 기반으로 첫 번째 기존 방법처럼 기본모델과 함께 가변성을 표현할 수 있도록 하였다. 이러한 방법이 다른 자산들에 일관성 있게 적용이

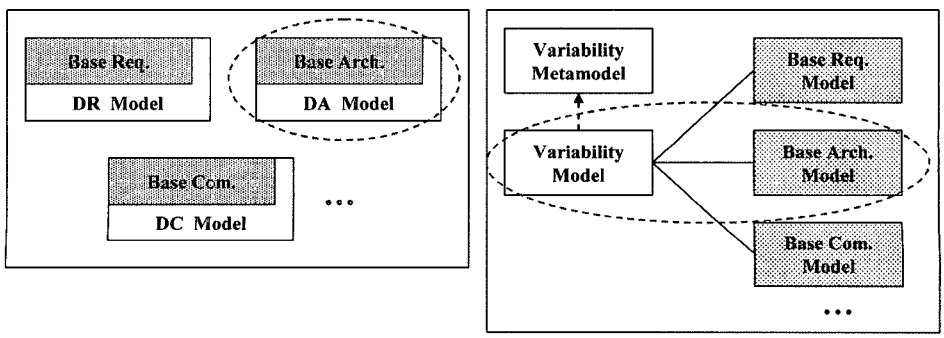


그림 18 가변성을 표현하기 위한 기존의 두 가지 접근 방법

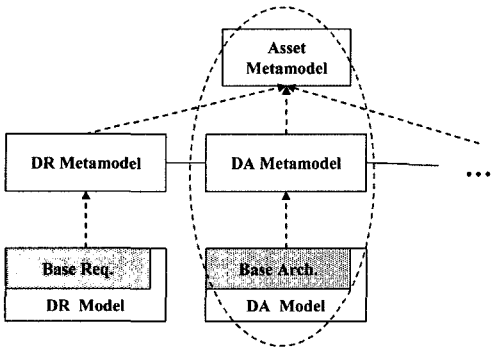


그림 19 본 연구의 가변성 표현 접근 방법

되면, 각 자산 모델들 사이의 연관관계가 메타모델 사이의 연관관계에 의해 추적가능하게 연결될 수 있다.

7. 결론 및 향후 연구

도메인 아키텍처는 관련된 소프트웨어 프로덕트들을 연속적으로 생산해내기 위한 프레임웍 역할을 하는 핵심자산이다. 아키텍처가 핵심자산이 되기 위해서는 이 수준에서 발생할 수 있는 가변성을 식별하고 이를 명시적으로 모델에 반영하여야 한다. 본 논문에서는 먼저, 최근 OMG에서 채택한 재사용 자산 명세(RAS) 모델을 기반으로 한 계층적 메타모델링 구조를 제시하였다. 이 모델링 구조를 바탕으로 도메인 아키텍처 수준에서 C&V의 개념을 결합하기 위해 도메인 아키텍처 메타모델을 제시하였다. 이 모델을 통해 도메인 아키텍처의 특징에 따라 이 수준에서 가질 수 있는 가변성의 유형들을 제시하였다. 또한 도메인 아키텍처를 구성하는 산출물을 정의하고 각 모델에 식별된 가변성이 명확히 반영됨을 보였다. 이 메타모델은 e-Travel System 도메인에 적용되어 도메인 아키텍처 모델이 개발되는 것을 보였다. 그 결과 제시한 방법이 가변성 유형에 맞게 개발과정에서 적절히 식별되고 명확히 모델링 될 수 있음을 확인할 수 있었고, 이 방법이 실제 프로젝트에 적용될 수 있음을 보여주었다.

향후 연구 과제는 소프트웨어 프로덕트 라인에서 핵심자산들의 개발과 관리를 효율적으로 할 수 있도록 지원 도구를 개발하는 것이다. 핵심자산으로서 도메인 산출물을 잘 관리함으로써 체계적인 재사용을 보장해 주게 되고, 이는 결과적으로 시스템 개발 시간과 비용을 줄여주는 효과를 가져다준다.

참 고 문 헌

[1] D. Muthig, and C. Atkinson, "Model-Driven Product Line Architecture," In Proceedings of the Second Software Product Line Conference (SPLC2), San Diego, U.S.A., LNCS Vol.2379, pp.110-129, 2002.

[2] The Object Management Group (OMG), Reusable Asset Specification (RAS) Version2.2, http://www.omg.org/technology/documents/formal/ras.htm, Nov. 2005.

[3] Rational, *UML Semantics version1.1*, http://www.rational.com/uml, 1997.

[4] H. Gomaa, D. Webber, "Modeling Adaptive and Evolvable Software Product Lines Using the Variation Point Model," Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), 2004.

[5] M. Moon, K. Yeom, and H.S. Chae, "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability in a Product Line," IEEE Transactions on Software Engineering, vol. 31, no. 7, pp.551-569, Jul. 2005.

[6] M. Matinlassi, "Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA," Proceedings of the 26th International Conference on Software Engineering(ICSE'04), 2004.

[7] P. America, H. Obbink, J. Muller, and R. van Ommering, "COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products," Denver, Colorado: the First Conference on software Product Line Engineering, 2000.

[8] D. Weiss, C. Lai, and R. Tau, *Software product-line engineering: a family based software development process*, Addison-Wesley, Reading, MA, 1999.

[9] K. Kang, S. Kim, J. Lee, and K. Kim, "FORM: A Feature-Oriented Reuse Method with Domain Specific Reference Architectures," Pohang University of Science and Technology(POSTECH), 1998.

[10] C. Atkinson et al., *Component-based product line engineering with UML*. Addison-Wesley, London, New York, 2002.

[11] M. Matinlassi, E. Niemela and L. Dobrica, "Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture," VTT Technical Research Centre of Finland, Espoo, 2002.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report CMU/SEI-90-TR-21, *Software Engineering Institute*, Carnegie Mellon University, November 1990.

[13] H. Gomma and M. E.Shin, "A Multi-View Meta-modeling Approach for Variability Management in Software Product Lines," In Proceeding's of 8th International Conference (ICSR8), Madrid, Spain, LNCS Vol.3107, pp.274-285, 2004.

[14] P. America, D. Hammer, M.T. Ionita, H. Obbink, and E. Rommes, "Scenario-Based Decision Making for Architectural Variability in Product Families," In Proceedings of the Second Software Product Line Conference (SPLC4), LNCS Vol.3154, pp.284-303, 2004.
[15] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A Framework for Modeling Variability in Software Product Families," In Proceedings of the Second Software Product Line Conference(SPLC4), LNCS Vol.3154, pp.197-213, 2004.
[16] Keepence, B., and Mannion, M., "Using patterns to model variability in product families," IEEE Software, Vol.16, Issue: 4, pp.102-108, 1999.
[17] Clauß, M., "Generic Modeling using UML extensions for variability," OOPSLA 2001, Workshop on Domain Specific Visual Languages, 2001.
[18] Webber, D., and Gomma, H., "Modeling Variability with the Variation Point Model," In Proceedings of the Seventh International Conference on Software Reuse, pp.109-122, 2002.
[19] K. Pohl., G. Bockle, and F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005.

문 미 경

1990년 2월 이화여자대학교 전자계산학과(학사). 1992년 2월 이화여자대학교 전자계산학과(석사). 2005년 2월 부산대학교 컴퓨터공학과(박사). 2005년 3월~2005년 8월 부산대학교 차세대물류IT기술사업단 박사후 연구원. 2005년 9월~2006년 8월 부산대학교 컴퓨터 및 정보통신 연구소 기금교수. 2006년 9월~현재 부산대학교 정보컴퓨터공학부 연구교수. 관심분야는 소프트웨어 프로덕트 라인 공학, 적응형 소프트웨어 개발, RFID기반 미들웨어 및 RFID 비즈니스 이벤트 프레임워크 개발 등임

염 근 혁

1985년 2월 서울대학교 계산통계학과(학사). 1992년 8월 Univ. of Florida 컴퓨터공학과(석사). 1995년 8월 Univ. of Florida 컴퓨터공학과(박사). 1985년 1월~1988년 2월 금성반도체 컴퓨터연구실 연구원. 1988년 3월~1990년 6월 금성사 정보기기연구소 주임연구원. 1995년 9월~1996년 8월 삼성SDS 정보기술연구소 책임연구원. 1996년 8월~현재 부산대학교 컴퓨터공학과 부교수. 관심분야는 소프트웨어 재사용, 프로덕트라인 공학, 소프트웨어 아키텍처, 컴포넌트 기반 소프트웨어 개발, 적응형 소프트웨어 개발, RFID기반 미들웨어 등임