도메인 핵심자산의 가변성 분석을 위한 2차원적 접근방법

(A 2-Dimensional Approach for Analyzing Variability of Domain Core Assets)

문 미 경 † 채 흥 석 †† 염 근 혁 †††
(Mikyeong Moon) (Heung Seok Chae) (Keunhyuk Yeom)

요 약 소프트웨어 재사용에 대한 활동들을 미리 계획하고 개발 프로세스의 연속적인 부분으로 이루어지도록 지원하는 방법이 소프트웨어 프로덕트 라인 공학이다. 이 방법에서 가장 중요한 것은 관련된 시스템들, 즉 도메인에서 공통성과 가변성(commonality and Variability: C&V)을 분석하는 일이다. 재사용 가능 항목들은 분석된 C&V를 명시적으로 나타냄으로써 프로덕트 라인의 핵심자산이 된다. 이러한 핵심자산들은 소프트웨어 개발의 각기 다른 단계에서 생산되기 때문에 표현요소들의 추상화 수준이 다르며 이로 인해 각 핵심 자산이 가지고 있는 가변성 또한 각기 다른 수준에서 각기 다른 유형으로 나타나게 된다. 핵심자산의 C&V 분석에 대한 기존 연구들에서는 핵심자산의 구분 없이 일관되게 가변성을 분석하였으며, 공통성과 가변성 식별을 단지 개발자의 직관이나 도메인 전문가의 경험에 의존하고 있었다. 본 논문에서는 소프트웨어 프로덕트 라인에서 핵심자산의 가변성을 분석하기 위하여 수직적 측면과 수평적 측면으로 나누어 접근해가는 2차원적 분석방법을 제안한다. 수평적 접근 방법은 개발 프로세스의 각기 다른 단계에서 산출되는 요구사항, 아키텍처, 컴포넌트의 수준에서 가변성의 유형을 분석하는 것이고, 수직적 접근 방법은 가변성의 상세화 정도에 따라 공통성을 식별하는 수준과 가변점을 상세화하는 수준으로 나누어 분석하는 것이다. 이러한 2차원적 가변성 분석접근 방법은 핵심자산들의 가변성이 서로 연관관계를 가질 수 있도록 해주며, 핵심자산의 재사용 활동이 끊어짐 없이 이루어지도록 한다.

키워드 : 소프트웨어 프로덕트 라인, 공통성과 가변성, 가변성 분석, 핵심 자산

Abstract Software product line engineering is a method that prepares for the future reuse and supports to seamless reuse in application development process. Commonality and variability play central roles in all product line development processes. Reusable assets will become core assets by explicitly representing C&V. Indeed, the variabilities that are identified at each phase of core assets development have different levels of abstraction. In the past, these variabilities have been handled in an implicit manner and without distinguishing the characteristics of each core assets. In addition, previous approaches have depended on the experience and intuition of a domain expert to recognize commonality and variability.

In this paper, we suggest a 2-dimensional analyzing method that analyzes the variabilities of core assets in software product line. In horizontal analysis process, the variation types are analyzed in requirements, architecture, and component that are produced at each phase of development process. In vertical analysis process, variations are analyzed in different abstract levels, in which the region of commonality is identified and the variation points are refined. By this method, the traceability of variations between core assets will be possible and core assets can be reused seamlessly.

Key words : Software Product Line, Commonality and Variability, Variability Analysis, Core Asset

· 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT 기술연구사업단)의 지원에 의하여 연구되었음
† 정 회 원 : 부산대학교 컴퓨터및정보통신연구소 교수
mkmoon@pusan.ac.kr
†† 정 회 원 : 부산대학교 컴퓨터공학과
hschae@pusan.ac.kr
††† 종신회원 : 부산대학교 컴퓨터공학과 교수
yeom@pusan.ac.kr
논문접수 : 2005년 7월 27일
심사완료 : 2006년 3월 27일

1. 서 론

소프트웨어 개발 생산성과 품질 향상을 위하여 소프트웨어 재사용에 대한 이슈가 수십 년 전부터 끊임없는 연구과제로 진행되어왔다. 컴포넌트 기술과 도메인 공학에 대한 연구들은 소프트웨어 재사용에 대한 발전을 가져왔다. 이 두 개념을 모두 포함하는 컴포넌트 기반 소프트웨어 프로덕트 라인은 개발 자산들을 추출하여 이들을 각각 독립적으로 재사용될 수 있는 핵심 자산으로 만듦과 동시에 서로 연관성을 지어 재사용 활동이 프로세스를 따라 이어갈 수 있게끔 만드는 방법이다. 소프트웨어 프로덕트 라인에서 핵심 자산들의 재사용성을 향상시키기 위해서는 다시 재사용 될 수 있는 공통성을 가진 영역과 시스템마다 다소 상이하게 나타날 수 있는 지점인 가변점을 식별하고 이를 명시적으로 표현하는 가변성 분석 방법이 핵심이 된다. 핵심 자산들은 소프트웨어 개발의 각기 다른 단계에서 생산되기 때문에 표현요소들의 추상화 수준이 다르며 이로 인해 각 핵심 자산이 가지고 있는 가변성 또한 각기 다른 수준에서 각기 다른 유형으로 나타나게 된다. 핵심자산의 가변성 분석에 대한 기존 연구들에서는 공통성과 가변성 식별을 자산의 특성을 반영하지 않고 일관되게 분석하고 있었으며 대부분이 암시적으로 컴포넌트만을 그 대상으로 하고 있었다.

본 연구에서는 소프트웨어 프로덕트 라인에서 핵심자산을 개발하기 위하여 2차원적으로 가변성을 분석해나가는 방법을 제안한다. 먼저 가변성은 상세화 정도에 따라 공통성을 식별하는 수준과 가변점을 상세화하는 수준으로 나누어 분석한다. 이를 가변성 메타모델에 명시적으로 표현한다. 이렇게 2가지 형태로 실현되는(realize) 가변성 메타모델을 바탕으로, 각기 다른 산출물 즉, 도메인 요구사항과 도메인 아키텍처, 그리고 도메인 아키텍처를 구성하는 도메인 컴포넌트의 가변성 유형을 분석한다. 이를 또한 메타모델로 표현한다. 이렇게 분석된 각 자산별 가변성 메타모델은 자산들 사이의 연관관계를 바탕으로 서로 연결관계를 맺게 된다. 그 결과 애플리케이션 개발 단계별 재사용 활동이 끊어짐 없이 이루어지도록 하는 기틀을 마련할 수 있으며, 각 자산 별 가변성의 추적(traceability)이 가능해지도록 한다. 즉, 도메인 요구사항의 가변성이 다른 핵심 자산의 공통성과 가변성(Commonality and Variability: C&V) 성질을 결정하는 기준이 되며, 공통적인 요구사항을 대상으로 컴포넌트를 개발하게 되면 컴포넌트가 매우 높은 재사용성을 가지게 된다. 또한 도메인 아키텍처는 컴포넌트가 플러그인 될 수 있는 컴포넌트들의 공통된 구조를 정의하며, 시장의 요구사항에 맞게 필요한 컴포넌트가 선택적으로 조립되어 빠르게 시스템을 생산할 수 있게 한다.

2. 관련연구

2.1 요구사항 가변성 분석 방법

요구사항 단계에서 가변성을 분석하기 위하여 도메인 내의 관련된 시스템들의 공통성과 가변성을 표현하기 위한 기본 요소인 '피처'(feature)가 다방면 사용되었다[1,2]. 피처는 시스템 집합 중에 주도적인 또는 독특한 특성을 나타내는 것으로 정의된다. 추출된 피쳐는 공통성과 선택성을 나타내기 위하여 구별되는 표기법을 사용하고 있으며 이 표현들이 모여 피쳐 그래프(feature graph)를 만들게 된다. 그러나 피쳐 그래프에서는 가변점이나 가변값들이 명시적으로 나타나 있지는 않다. PLA(Product Line Analysis) 방법은 피쳐를 인식하는 것에 기초하여 전통적인 객체지향 모델링 방법을 혼합하여 프로덕트 라인 요구사항을 분석하는 연구이다[3]. 그러나 피쳐는 요구사항을 뜻하는 것이 아니며[4], 피쳐의 속성을 결정짓게 하는 논리적인 근거를 제시하지도 못한다. 피쳐의 분류는 모두 도메인 분석가의 경험이나 직관(heuristic)에 의존하고 있다[5].

Kuusela와 Savolainen은 요구사항을 추출하고 구조화하기 위한 방법으로 정의 계층화 방법(definition hierarchy method)을 제시하였고[6], Thompson과 Heimda는 요구사항을 n-차원적, 계층적인 프로덕트 라인을 위하여 요구사항을 구조화하는 기법을 연구하였다[7]. 이들은 도메인을 구조화 시키는 것에 초점을 맞추고 있다. 가변성 정보를 제시해주기 위한 최소한의 노력은 적절한 가변성 지점의 식별임에도 불구하고 요구사항 단계에서 가변성 유형을 분류하거나 가변성 지점을 명시적으로 추출해주는 연구는 이루어지지 못하였다.

2.2 아키텍처 가변성 분석 방법

도메인 아키텍처에 관련된 연구로는 Feature Oriented Domain Analysis (FODA) 방법[1]을 설계와 구현 단계로 확장한 FORM(Feature-Oriented Reuse Method)[8]이 있다. 이 방법에서는 피쳐 모델을 아키텍처 모델링 단계에서 아키텍처 모델과 적절히 대응시키고 그 결과 참조 아키텍처(reference architecture)와 컴포넌트를 생성한다. 이 방법에서 도메인 아키텍처를 나타내는 참조 모델은 단지 상하의 추상화 레벨을 나누어서 나타내었다. 그러나 복잡한 시스템인 경우에는 추상화의 상하뿐만 아니라 정적, 동적 요소를 나타낼 수 있는 다양한 방면의 뷰(view)를 나타낼 필요가 있다. 또한 이 방법은 컴포넌트를 인식, 선택하고 그들 간의 관계를 고려하여 컴포넌트를 조립해 내는 과정에 대한 정보들-컴포넌트간의 관계, 컴포넌트간의 연결 관계, 커넥터 정보 등을 도메인 아키텍처를 통해서 얻을 수가 없다. Koala는 아키텍처를 형식적(formal)으로 기술할 수 있

게 하는 아키텍처 기술 언어(Architecture Description Language, ADL)이다[9]. 이 방법의 기본 개념들은 인터페이스, 커넥터, 서브컴포넌트, 복합 컴포넌트 등이 있다. Koala에서 적용되는 가변성 메커니즘은 단지 물리적으로 생성된 프로덕트 인스턴스에 포함된 소프트웨어의 행동적 가변성만을 모델하고 있다. 프로덕트 라인 아키텍처에서 가변성을 모델링하는 기법으로 패턴을 사용하는 연구가 있다[10]. 여기서 판별식(discriminant)은 하나의 시스템을 다른 시스템과 구분 짓는 어떤 피처를 의미하게 되고 이를 패턴과 연관지어 가변성을 모델링한다. UML 확장 기법을 사용하여 가변성을 모델링 한 연구도 있다[11]. 여기서는 UML 모델에 스테레오 타입으로 <<variationPoint>>를 기술하고 각 가변치(variant)들을 상속관계로 표현한다[12]. 연구에서는 프로덕트 라인에서 가변성을 모델링하기 위해 매개변수화(parameterization), 정보 은닉(information hiding), 상속(inheritance), 가변성 포인터(Variation Point)를 사용하는 방법들을 제시하고 있다. 이 연구들은 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.

2.3 컴포넌트 가변성 분석 방법

컴포넌트의 의미는 구현에 대한 구체적인 하나의 큰 단위로 정의됨으로 기존 연구들은 미리 구현된 컴포넌트에 적용될 수 있는 가변성 구현 기법들 - 상속, 확장, 사용, 구성, 파라미터, 템플릿 기법 -에 대해서만 언급하고 있다[13,14]. 컴포넌트를 명세하기 위해 고려해야 하는 가변성을 다루는 기존 연구는 부족한 상태이다. KobrA 방법에서는 컴포넌트(Komponent) 명세 개발이 가장 주축이 되는 활동이다[15]. 여기서 컴포넌트 명세는 컴포넌트가 외부적으로 보이는 속성들을 여러 모델들을 통해 기술하고 있다. 모델들에 나타나는 가변성은 단지 텍스트로 기술되는 decision model을 통해서 다루고 있다. 가변성의 명확한 표현을 위하여 사용할 수 있는 개념인 가변점과 가변치, 가변성의 유형에 대해서는 언급되지 않는다. UML Components 책에서는 비록 도메인을 고려한 컴포넌트 명세를 개발하고 있지는 않지만, 프로젝트 생명주기 상에서 달라지는 컴포넌트의 형태를 구분하여 정의하고 있다[16]. 그 중에서 컴포넌트 명세(Component Specification)에 대하여 본 연구와 동일한 정의를 내리고 있으며 컴포넌트 명세의 주요한 부분이 컴포넌트 인터페이스라고 강조하고 있다. 컴포넌트는 그 개념에 재사용을 내포하고 있지만, 재사용을 미리 고려하여 개발하지 않는다면 컴포넌트 자체도 재사용이 어렵게 된다.

3. 가변성 분석을 위한 접근 방법

David M. Weiss과 Chi Lai은 가변성을 프로덕트 패밀리에 속해있는 프로덕트들이 서로로부터 얼마나 달라질 수 있는 가에 대한 가정이라고 정의하였다[9]. 그림 1은 본 논문에서 가변성을 분석하기 위해 두 방향으로 접근해 가는 방법을 표현한 것이다.

3.1 수직적 가변성

그림 1에서 수직축은 가변성의 추상화 정도에 따라 2가지 형태로 구분됨을 나타낸다. 본 논문에서는 이를 수직적 가변성으로 분류한다. 상위수준의 가변성은 CV Property로 나타낸다. CV Property는 기능적 구현 유무에 따라 공통성(common)과 선택성(optional)의 값을 가진다. 이때, 각 단계별 CV Property를 가질 수 있는 자산요소(asset element)가 달라지는데, 요구사항 분석 단계에서는 PR[1](Primitive Requirement), 아키텍처 설계단계에서는 도메인 컴포넌트, 그리고 컴포넌트 개발단계에서는 오퍼레이션이 CV Property 값을 가진다. 자동차에 대하여 예를 들어 보면, 모든 자동차에는 기어, 바퀴를 가지고 있고, 브레이크 밟기, 가속하기 등의 공통된 동작을 지니고 있다. 그러나 네비게이터, 선루프 등은 선택적으로 가지고 있는 기능이다. 이를 표현하는 수준이 CV Property를 통해서이다. 하위수준의 가변성은 가변점으로 나타낸다. 가변점은 변화가 일어날 가능성이 있는 하나 이상의 위치로 정의된다. 자동차 예를 보면, 기어는 공통성을 가진 기능이지만 더 세부적으로는 자동 기어와 수동 기어로 나눌 수 있다. 이렇게 종류

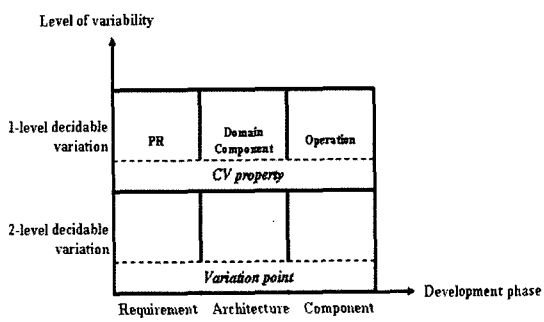

그림 1 가변성 분석을 위한 수직·수평적 분류

1) PR은 도메인 내의 시스템들이 외부 행위자에게 의미 있는 결과를 제공하기 위한 하나의 트랜잭션으로 이루어진 기능적 요구사항을 나타내는 단위이다.

를 선택하는 지점이 가변점이 되며 자동, 수동의 값이 가변치가 된다. 4장에서 가변성에 대한 수직적 분석 방법에 대하여 자세히 설명한다.

3.2 수평적 가변성

그림 1에서 수평축은 가변성이 핵심자산의 종류에 따라 요구사항, 아키텍처, 컴포넌트로 구분됨을 나타낸다. 본 논문에서는 이를 수평적 가변성으로 분류한다. 각 단계별 핵심자산을 나타내는 구성모델은 다르지만 구성모델에 표현되는 가변성은 모두 수직적 가변성에 나타나 있는 CV Property와 가변점으로 표현된다. 그 결과 가변성 사이의 연관관계를 찾을 수 있게 된다. 즉, PR의 CV Property는 도메인 컴포넌트의 CV Property와 오퍼레이션의 CV Property에 영향을 준다. 5장에서 수평적 분석 방법에 대하여 자세히 설명한다.

4. 수직적 가변성 분석 방법

4.1 1-수준 결정가능 가변성

가변성의 상세화 정도에 따른 첫 번째 수준은 기능적 측면에서 이 기능이 존재하는지 아닌지를 판단하는 가변성이다(1-level decidable variable). 이러한 기능적 구현 유무를 공통적 또는 선택적으로 나타나는 가변성을 CV_Property로 표현한다. 여기서는 공통된 영역을 식별해 낼 수 있다. 표 1은 요구사항 단계에서 수집된 요구사항 PR들을 나타낸 것이며 이들은 해당 도메인의 모든 시스템이 가지고 있는 요구사항이기 때문에 PR에 대한 CV_Property는 공통성이 된다. 그러나 PR이 공통적일지라도 이들은 내부적으로 기능의 속성, 절차들이 달라질 수 있는 가변성을 가지고 있다. 이를 2-수준 결정가능 가변성 단계에서 분석하게 된다.

표 1 요구사항에 대한 1-수준 결정가능 가변성의 예

PR1: System presents total with taxes calculated.
PR2: System handles payment.

4.2 2-수준 결정가능 가변성

가변성의 상세화 정도에 따른 두 번째 수준은 기능의 속성과 절차 등에서 가변이 발생하는 지점 즉, 가변점을 표시하고 이에 대응하는 가변치를 제시하는 수준의 가변성이다(2-level decidable variable). 본 논문에서는 가변점을 실현하기 위하여 다음의 세가지 요소들을 구성한다.

- ***가변치(variants)*** - 각 가변치는 가변성이 실현될 때, 가변점에 나타날 수 있는 구체적인 값을 의미한다.
- ***가변점 대응치(variation point cardinality)*** - 가변점 대응치는 해당 가변점에 선택되어야 하는 최소한의 가변치 수와 이 가변점에 선택될 수 있는 최대한의 가변치 수를 나타낸다. 가변점 대응치가 의미하는 바는 다음과 같다.
  - [1] v1, v2,...,vn : 가변치중에 반드시 하나 선택되어야 한다. 이는 가변점이 반드시 가변값으로 채워져야 하는 공통성을 의미하며, 나열된 가변값들은 서로 alternative 관계를 가지고 있음을 의미한다.
  - [1..n]v1, v2,...,vn : 가변치중에서 적어도 하나 이상 선택되어야 한다. 이는 가변점이 공통성 속성을 가지면서 다중 어댑터(multiple adapter)가 됨을 의미한다.
  - [0..1]v1, v2,...,vn : 가변치 중에서 하나 선택될 수도 있고 그렇지 않을 수도 있음을 나타낸다. 이는 가변점이 선택성 속성을 가지면서 단독 어댑터(single adapter)가 됨을 의미한다.
  - [0..n]v1, v2,...,vn : 가변치 중에서 선택이 전혀 안될 수도 있고, 하나 이상 선택될 수도 있다. 이는 가변점이 선택성 속성을 가지면서 다중 어댑터가 됨을 의미한다.
- ***가변점 유형(variation point type)*** - 가변점 유형은 가변점에 대응되는 가변치들을 분류하기 위한 수단이 된다. 이것은 개발 과정의 산출물에 따라 다른 형태의 가변성이 나타나기 때문에 각 산출물 별로 가변점 유형을 분류하는 것이 필요하다.

표 2는 표 1의 예의 PR1이 포함하고 있는 가변성에 대하여 분석한 모습을 보여주고 있다. 또한 위에서 제시한 PR들의 가변성에 대하여 분석한 것이 아래에 있다. 여기서 PRelement는 PR을 세분화시키는 단위로 사용된다. 해당 도메인에 있는 대부분의 시스템들이 세금을 처리하는 기능은 공통적으로 가지고 있지만, 세금을 계산하는 방식은 그 기능이 실현될 때마다 달라질 수 있는 가변적 요소이다. 이 가변점이 실현될 수 있는 방식은 ①기본 세금 계산 방식 또는 ②real estate 세금 계산 방식이 있으며, 이는 서로 대안적 방식이며 둘 중 하나의 방식으로 실현된다.

표 2 요구사항 단계의 2-수준 결정가능 가변성의 예

PRelement1: Calculate taxes.
Variation point type: computation
Variation point cardinality: [1]
Variants: ① basic tax law
② real estate tax law
PRelement2: Choose payment way.
Variation point type: control
Variation point cardinality: [1..n]
Variants: ① Paying by cash
② Paying by credit
③ Paying by check

4.3 가변성 메타모델

그림 2에서 상위에 위치한 패키지내의 그림이 수직적

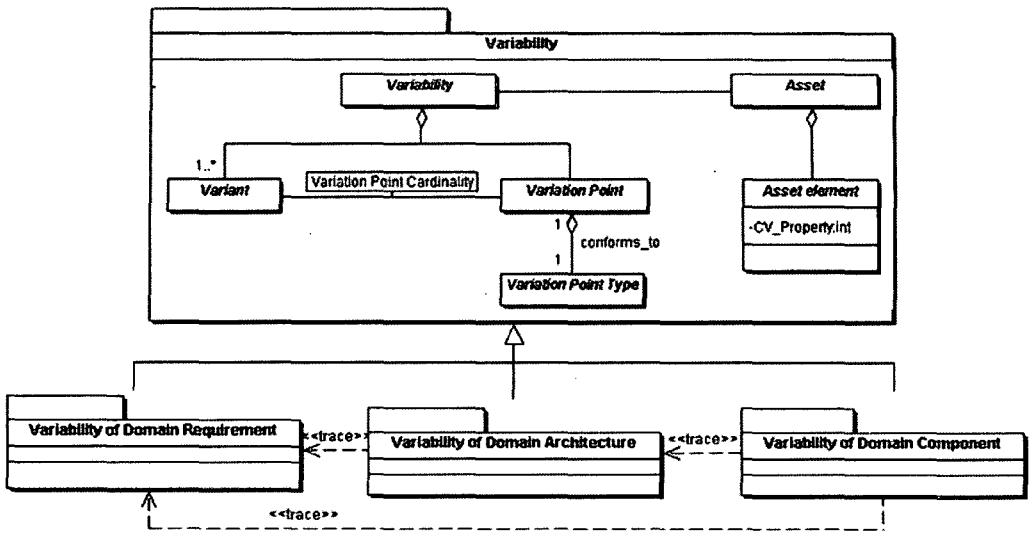

그림 2 가변성에 대한 메타모델

가변성 분석에 대한 개념을 메타모델로 나타낸 것이다. 이 모델에서 보여지는 1-수준 결정가능 가변성인 CV_Property는 수평적 분석 시 고려되는 핵심자산 요소(Asset element) 마다 하나의 속성으로 나타나게 된다. 이 핵심자산 요소가 변함에 따라 이에 의존하고 있는 가변점 유형은 바뀌게 되다. 이 메타모델을 기반으로 각 자산 별로 분석하는 수평적 가변성에 대한 메타모델이 그려질 것이다. 그림 2에서 하위에 위치한 패키지들이 자산 별 가변성을 나타내고 있으며, 이들은 상위의 가변성 메타모델을 상속받아 각 자산 별 모델과 가변성 유형에 대하여 구체적(concrete) 클래스로 그려지게 된다. 최종적으로 이 모델들은 통합되며 각 자산 별 가변성에 대한 추적성을 식별하여 나타낼 수 있게 된다.

5. 수평적 가변성 분석 방법

5.1 도메인 요구사항의 가변성 분석

5.1.1 도메인 요구사항 모델

도메인 요구사항은 기능적 측면에 대해서 계층적으로 분류한다. 도메인 요구사항을 일정한 크기(granularity)로 수집하기 위하여 요구사항 명세 원자(specification atom)인 PR(Primitive Requirement)을 사용한다[18]. PR은 도메인 내의 시스템들이 외부 행위자(external actor)에게 의미 있는 결과를 제공하기 위한 하나의 트랜잭션(transaction)으로 이루어진 기능적 요구사항이다. 여기서 하나의 트랜잭션이란 것은 하나의 논리적인 기능을 수행하기 위한 응집력(cohesive)있는 일련의 오퍼레이션 집합을 나타낸다. 하나의 PR은 두 가지 측면에서 하나 이상의 PRelement들로 정제된다. 행위적(behavior) PRelement는 PR의 기능을 워크플로우 형태로 상세화 시키는 기술 형태를 의미하며, 정적(static) PRelement는 정적 구조 관점에서 PR의 기능 수행 시 필요한 데이타들을 식별하여 기술하는 형태이다. 그림 3은 그림 2의 가변성 메타모델을 상속받아, 도메인 요구사항 가변성에 대하여 메타모델로 나타낸 것이다. 그림에서 점선박스 부분이 도메인 요구사항 모델을 나타낸 것이며, Static PRelement에서 데이타의 가변성이 식별될 수 있으며, Behavior PRelement에서 computation, external computation, control의 가변성이 식별될 수 있음을 나타내고 있다.

5.1.2 도메인 요구사항 가변성

그림 3에서 도메인 요구사항의 1-수준 결정가능 가변성은 Asset element를 상속받은 PR이 도메인 내 시스템에 나타나는 빈도수에 따라 공통성과 선택성의 성질을 가지게 됨을 나타낸다. CV_Property를 가진 PR은 2-수준 결정가능 가변성 수준에서 가변점을 식별해 낼 수 있게 된다. 다음은 도메인 요구사항이 가지는 가변성에 대하여 그 유형을 분류한 것이다.

- PR CV_Property - 도메인 요구사항들 중, 1-수준에서 추출될 수 있는 가변성 유형이다. 이는 단지 도메인 내의 기능적인 요구사항이 확률적으로 반드시 존재해야 하는지 그렇지 않아도 되는지를 나타낼 수 있는 가변성이다. 즉, 기능의 선택성에 관한 가변적 성질을 의미한다.
- Computation 가변성 - 이것은 플로우차트에서 하나의 프로세스에 대한 가변성을 의미하는 것으로 하나의 프로세스가 비즈니스 규칙이나 법규 등을 포함하는 경우이다.

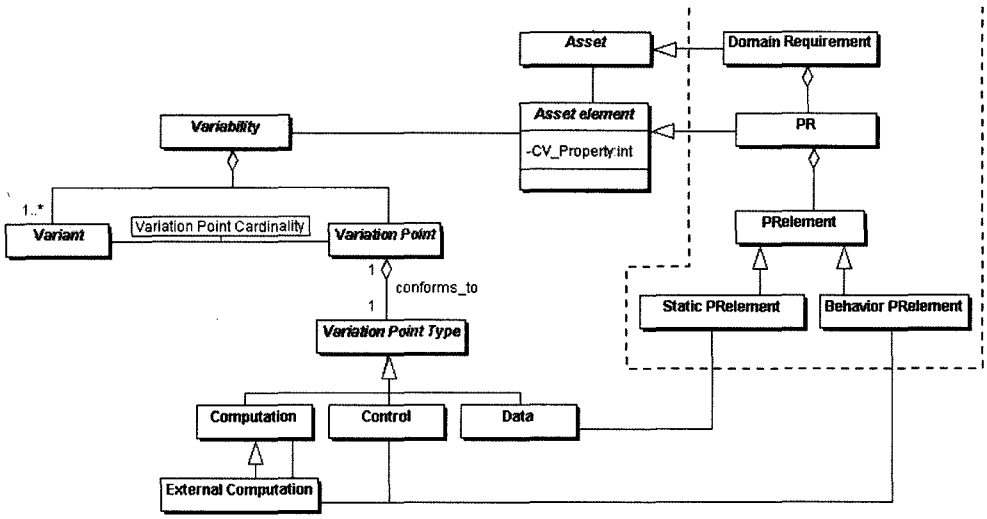

그림 3 도메인 요구사항 가변성에 대한 메타모델

- External Computation 가변성 - computation 가변성 중, 외부 서비스에 의해 처리되는 경우가 이에 해당한다.
- Data의 가변성 - 이것은 시스템에서 다루는 data의 구조에서 나타나는 가변성을 의미한다.
- Control 가변성 - 이것은 플로우차트에서 제어 상태의 가변성을 의미하게 된다. 즉, 선택적 상황에서 판단되는 기준이 가변되는 경우던지, 일정한 흐름의 패턴이 가변되는 경우가 이에 해당한다.

5.2 도메인 아키텍처의 가변성 분석

5.2.1 도메인 아키텍처 모델

도메인 아키텍처는 도메인 내의 시스템들의 공통적인 논리적 구조를 나타내기 위해 논리뷰를 정의한다. 논리뷰는 도메인 구성(configuration)을 가진다. 도메인 구성은 정적(static) 모델과 행위적(behavior) 모델을 통해 도메인 컴포넌트, 도메인 바인딩으로 이루어진 기능적인 논리적 구조를 나타낸다. 도메인 컴포넌트는 논리적인 측면에서 기능들의 한 묶음으로 된 서비스를 의미하고, 도메인 바인딩은 도메인 컴포넌트들 사이의 관계를 정의한다. 도메인 구성의 정적 모델은 도메인에서 제공해야 하는 기능들을 응집력 있는 한 단위로 묶어 표현함으로써 도메인 내의 기능들의 분할된 모습을 보여주고, 그들 사이의 의존 관계를 나타낸다. 도메인 구성의 행위적 모델은 도메인 컴포넌트의 인스턴스와 그들 사이에 주고받는 메시지를 가지고 도메인 컴포넌트들 간의 다양한 상호작용을 나타낸다. 그림 4는 그림 2의 가변성 메타모델을 상속 받아, 도메인 아키텍처 가변성에 대하여 메타모델로 나타낸 것이다. 그림에서 점선박스 부분

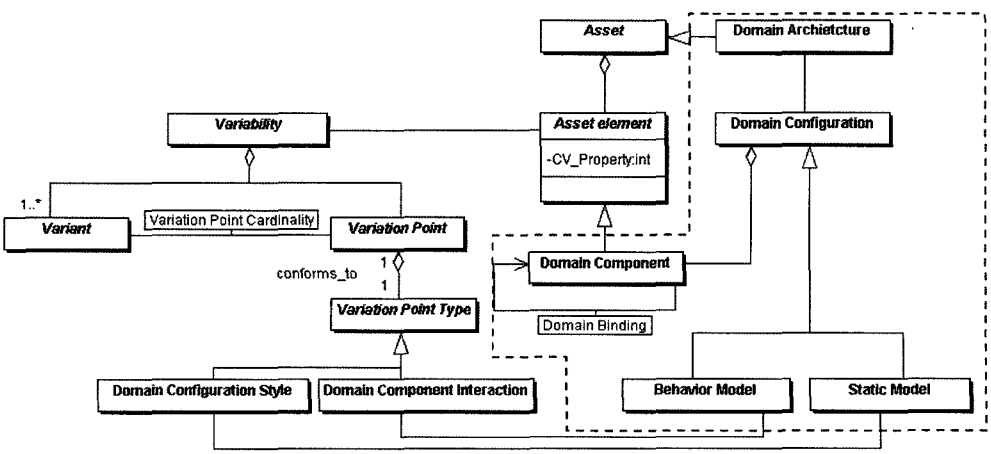

그림 4 도메인 아키텍처 가변성에 대한 메타모델

이 도메인 아키텍처 모델을 나타낸 것이다.

5.2.2 도메인 아키텍처 가변성

도메인 아키텍처는 가변성을 포함한다. 도메인 아키텍처의 1-수준 결정가능 가변성은 도메인 컴포넌트와 도메인 바인딩 그 자체가 아키텍처에 포함되는지 그렇지 않는지에 따라 공통성과 선택성 속성을 나타내는 CV_Property이다. 그림 4에서 Asset element를 상속받은 도메인 컴포넌트와 도메인 바인딩이 이러한 가변성을 속성으로 가지게 된다.

- 도메인 아키텍처의 2-수준 결정가능 가변성은 일련의 도메인 컴포넌트와 도메인 바인딩의 가변점들이 연결되면서 나타나는 도메인 구성의 변화의 가변성을 다룬다. 이는 연속된 가변점의 집합으로 나타나게 된다. 도메인 구성의 정적 모델에서 도메인 구성 스타일의 가변성이 식별될 수 있으며, 도메인 구성의 행위적 모델에서 도메인 컴포넌트 상호작용 가변성이 식별될 수 있다. 다음은 도메인 아키텍처가 가지는 가변성에 대하여 그 유형을 분류한 것이다.
- 도메인 컴포넌트 CV_Property 가변성 - 도메인 컴포넌트는 소프트웨어 개발 시 바로 배치(deploy)될 수 있는 물리적 컴포넌트와는 달리, 플랫폼 독립적인 논리적 수준의 서비스 중심 단위 패키지로 정의된다. 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 - 바인딩은 도메인 컴포넌트 사이의 관계를 나타낸다. 도메인 컴포넌트 바인딩의 가변성은 직접, 또는 간접적으로 발생하게 된다. 직접 발생하는 도메인 컴포넌트 바인딩 가변성은 두 개의 도메인 컴포넌트 사이의 연관관계를 추가하거나 삭제함으로써 생기는 경우이다. 간접 발생하는 도메인 컴포넌트 바인딩 가변성은 도메인 컴포넌트 가변성에 연관되어있는 컴포넌트의 바인딩인 경우, 도메인 컴포넌트의 가변으로 인해 바인딩이 같이 추가, 삭제되는 경우이다. 이와 같이 직간접적으로 도메인 컴포넌트 사이의 연관관계가 가변된다면 이는 도메인 아키텍처 수준에서 도메인 컴포넌트 바인딩 가변점이 된다.
- 도메인 구성 스타일(Domain Configuration Style) 가변성 - 이는 도메인 컴포넌트와 도메인 바인딩의 가변으로 인하여, 도메인 구성의 일부분이 가변되는 경우이다. 도메인 컴포넌트와 바인딩 가변점들은 일련의 연결 관계를 가지고 있기 때문에 이러한 관련된 가변점들은 연쇄적으로 발생하게 된다. 그러므로 이는 하나의 가변점으로 정의할 수가 없으며 관련된 가변점들의 집합으로서 다루어지게 된다.
- 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.

5.3 도메인 컴포넌트의 가변성 분석

5.3.1 도메인 컴포넌트 모델

그림 5는 컴포넌트가 가지고 있는 여러 측면의 모습을 나타내고 있다. 컴포넌트에 관심이 있는 여러 stakeholder가 컴포넌트의 어떤 측면을 보려고 하는지를 화살표로 표현하고 있다. 컴포넌트 명세에는 인터페이스를 비롯한 컴포넌트의 용도, 유형, 컴포넌트 내용에 대한 정보들이 포함되어 있다. 컴포넌트 구현물은 컴포넌트 명세를 실현하는 내부설계와 코드를 의미한다. 컴포넌트 실행물은 고객에게 전달되어 사용이 가능한 모듈들의 집합체로서 실제 실행시간에 바인딩 할 수 있도록 컴파일이 완료된 상태를 의미한다. 본 연구에서 정의하는 도메인 컴포넌트는 컴포넌트를 직접 개발하고자 할 때, 아키텍처 설계 단계에서 고려되는 상세화된 설계 문서로서 컴포넌트 명세를 의미한다.

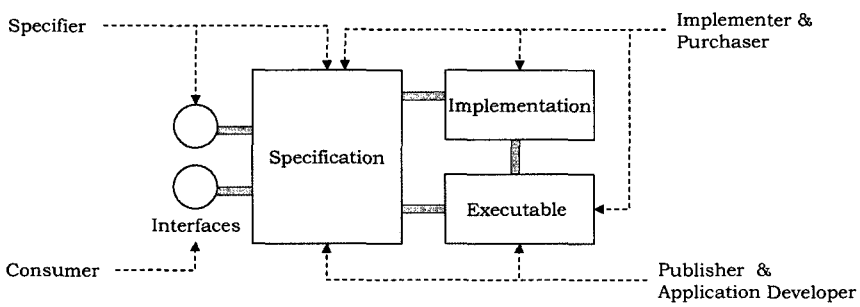

그림 5 컴포넌트의 여러 측면

도메인 컴포넌트 명세는 외부에서 접근할 수 있도록 컴포넌트가 제공하는 서비스를 정의하기 위하여 하나이상의 인터페이스 형태로 명세화 한다. 인터페이스는 하나이상의 관련된 오퍼레이션과 그 오퍼레이션이 수행되기 위해 필요한 정보 또는 상태를 추상화시킨 타입으로 기술된다. 인터페이스 명세의 정적 측면은 도메인 컴포넌트 내부의 정보는 외부에 모두 숨긴 채, 단지 컴포넌트를 어떻게 사용할 것인가에 대한 설명만을 기술한다. 이는 도메인 컴포넌트 명세에 대한 사용 계약(Usage Contract)을 의미하는 것이다. 인터페이스 명세의 행위적 측면은 도메인 컴포넌트가 다른 컴포넌트들과 어떻게 상호작용해야 하는지를 정의하고 오퍼레이션이 구현되는 방식을 기술한다. 이는 도메인 컴포넌트 명세에 대한 실현 계약(Realization Contract)으로 컴포넌트를 구현하는 사람이 따라야 하는 계약을 의미한다. 그림 6은 그림 2의 가변성 메타모델을 상속 받아, 도메인 컴포넌트 가변성에 대하여 메타모델로 나타낸 것이다. 그림에서 점선박스 부분이 도메인 컴포넌트 모델을 나타낸 것이다.

5.3.2 도메인 컴포넌트 가변성

도메인 컴포넌트는 명세 수준에서 가변성을 가진다. 도메인 컴포넌트의 1-수준 결정가능 가변성은 오퍼레이션의 공통성과 선택성 속성을 결정하는 CV_Property이다. 그림 6에서 Asset element를 상속받은 오퍼레이션이 이러한 가변성을 속성으로 가지게 된다. 도메인 컴포넌트의 2-수준 결정가능 가변성은 인터페이스의 명세에서 나타나는 가변성을 다룬다. 인터페이스에 대한 정적 측면에서 도메인 타입 가변성과 오퍼레이션 명세에 대한 가변성이 식별될 수 있으며, 행위적 측면에서 도메인 객체 상호작용에 대한 가변성을 식별할 수 있다. 다음은 도메인 컴포넌트가 가지는 가변성에 대하여 그 유형을 분류한 것이다.

- 오퍼레이션 CV_Property 가변성 - 인터페이스는 오퍼레이션들의 집합이며 각 오퍼레이션에는 컴포넌트 오브젝트가 수행하게 될 서비스나 기능이 정의되어있다. 그러므로 인터페이스에서 기능의 추가 삭제는 오퍼레이션의 추가 삭제를 의미하는 것이다. 이 때, 인터페이스가 가지고 있는 오퍼레이션 자체가 선택적인 경우에 이 오퍼레이션은 인터페이스가 가지는 가변점이 된다.
- 도메인 타입(Domain Type) 가변성 - 타입은 도메인 내에서 유지하고 관리할 필요가 있는 데이타나 상태를 정의해 놓은 것이며, 비즈니스 개념이나 프로세스로 인식하고 있는 것들이다. 타입에는 세부적으로 각기 다른 타입 가변성이 세 가지 있다.
  - 타입 자체가 도메인에서 선택적인 경우 컴포넌트 내에서 이 타입은 가변점이 된다. 타입의 이름에 가변점을 명시적으로 표시한다.
  - 도메인 컴포넌트 내의 필수적인 타입이지만, 타입 내부에 가변점을 가질 수 있다. 이 형태의 타입은 도메인 컴포넌트 분석을 통해서 타입의 애트리뷰터와 오퍼레이션에서 식별할 수 있다. 예를 들어, 인터페이스 명세를 구성하는 오퍼레이션의 가변성은 내부의 타입이 가지는 오퍼레이션의 가변성으로 연결된다. 해당 가변성이 나타나는 지점(타입의 애트리뷰트나 오퍼레이션)에 명시적으로 가변점을 나타낸다.
  - 다른 컴포넌트와 공유하는 타입인 경우에는 이 타입에 대한 컴포넌트들의 의존성에 따라 구현되어야 하는 컴포넌트가 결정된다. 이러한 결정은 실제 컴포넌트 개발자에 의해서 가변 될 수 있는 요소이지만, 도메인 컴포넌트 타입 모델에서는 이 지점에

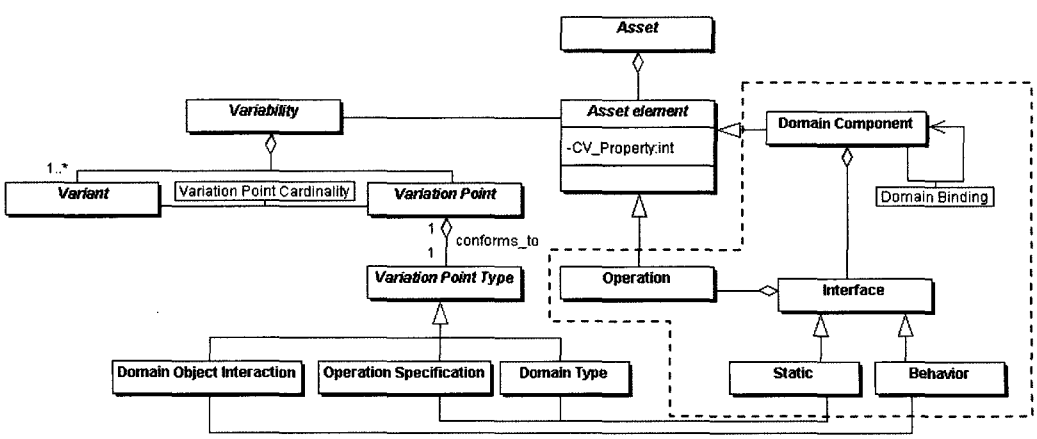

그림 6 도메인 컴포넌트 가변성에 대한 메타모델

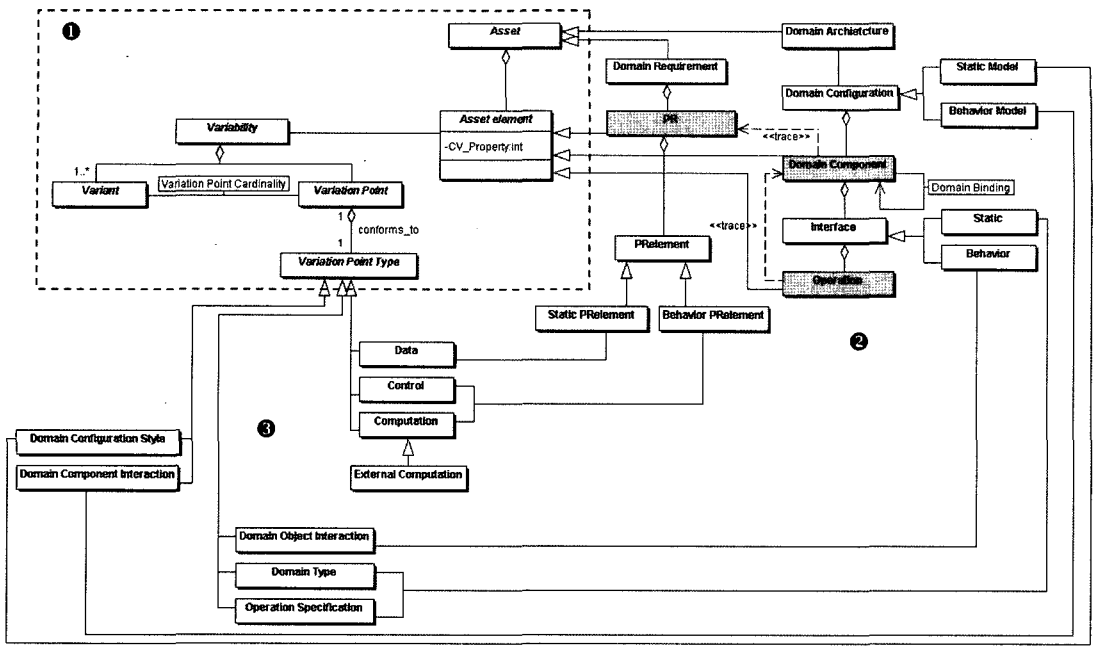

그림 7 핵심 자산 가변성의 연결관계

가변점을 두게 된다.

- 오퍼레이션 명세(Operation Spec) 가변성 - 오퍼레이션 명세에는 입력 파라미터, 리턴 값, 사전, 사후 조건 등이 기술된다. 오퍼레이션은 반드시 존재해야 하는 기능이지만, 이러한 오퍼레이션 내부 요소 중 일부분이 가변 될 수 있다. 이러한 값들이 가변 될 경우, 오퍼레이션 명세를 구분해서 작성하고 이 유형을 오퍼레이션 명세 가변점으로 구분한다.
- 도메인 오브젝트 상호작용(Domain Object Interaction) 가변성 - 이는 컴포넌트 내부의 상호작용 패턴에 대해 발생하는 가변성을 의미한다. 이러한 가변성이 발생하는 경우는 첫째, 컴포넌트 내부에 존재하는 타입의 가변에 따라 상호작용이 바뀌는 경우가 있다. 둘째, 컴포넌트 내부의 타입(클래스)들이 특정 품질 향상을 위해 설계 패턴을 적용하여 직접(direct) 연결을 간접(indirect) 연결로 변환한다던지, 또는 그 반대의 경우로 발생하는 경우가 있다. 이러한 경우 메시지의 흐름이 바뀌게 되고 이를 컴포넌트 오브젝트 상호작용 가변점으로 정의한다.

6. 핵심자산 가변성의 연관 관계

가변성에 대한 수직, 수평적 분석은 서로 연관관계를 지을 수 있다. 그림 7은 지금까지 정의하여온 핵심자산들에 대한 가변성 메타모델을 연결한 모습이다. 그림 7의 점선 부분은 가변성에 대한 메타모델을 표시하고 있다. 여기서 asset을 상속받은 그림의 오른쪽 편(②위치 부근)은 도메인 요구사항과 도메인 아키텍처에 대한 모델들 간의 전체 연관관계를 나타낸다. CV_Property 속성을 가지는 Asset element는 각 모델 별로 PR, 도메인 컴포넌트, 그리고 오퍼레이션이 있다. 도메인 요구사항에서 PR은 도메인 아키텍처에서 하나의 도메인 컴포넌트나 또는 컴포넌트에 속하는 하나의 오퍼레이션으로 실현되기 때문에 PR CV_Property 가변성은 도메인 컴포넌트 CV_Property 가변성과 도메인 컴포넌트 가변성 중 오퍼레이션 CV_Property 가변성에 직접적인 영향을 미친다. 가변성 메타모델에서 variation point type을 상속받은 그림의 왼쪽아래 편(③위치 부근)에는 각 핵심자산 별 나타날 수 있는 가변점 유형이 나열되어있다.

7. 사례 연구[2)]

본 연구에서 제시한 2차원적 가변성 분석 방법을 테트리스 게임 도메인에 적용해보았다. 다양한 형태의 테트리스 게임은 동일한 기본 게임 방식을 가지면서 동시에 각 테트리스 게임 유형별 다양한 특징을 가진다. 사례연구에서 해당 도메인의 각 단계별 공통성과 가변성이 분석된 핵심자산을 개발하고, 이를 바탕으로 실제 여러 종류의 테트리스 게임을 개발해 보았다. 본 절에서는 사례연구 내용을 설명하고 이를 통해 제시한 방법이 핵

2) 본 사례연구에 대한 문서는 http://se.ce.pusan.ac.kr에 있습니다.

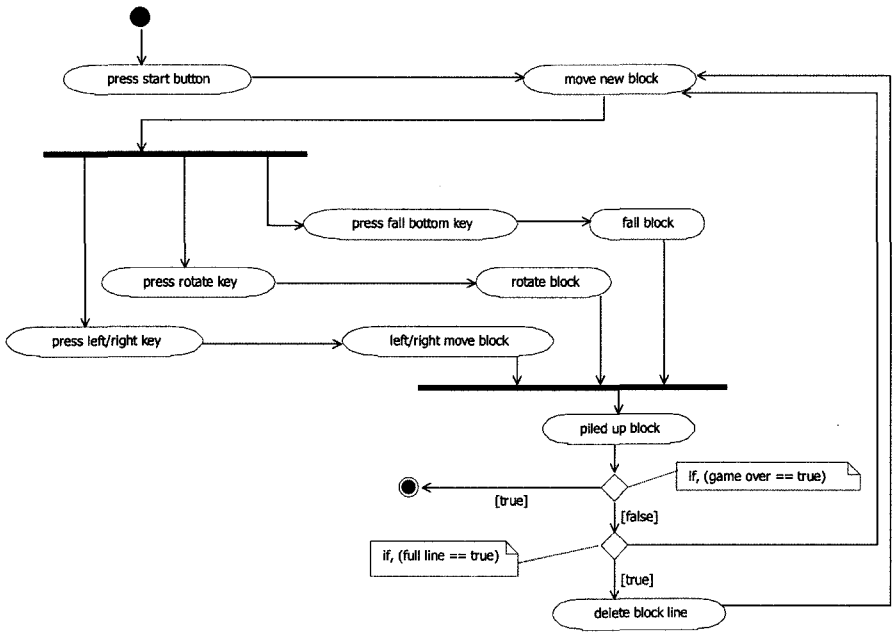

그림 8 테트리스 게임의 기본 동작 흐름

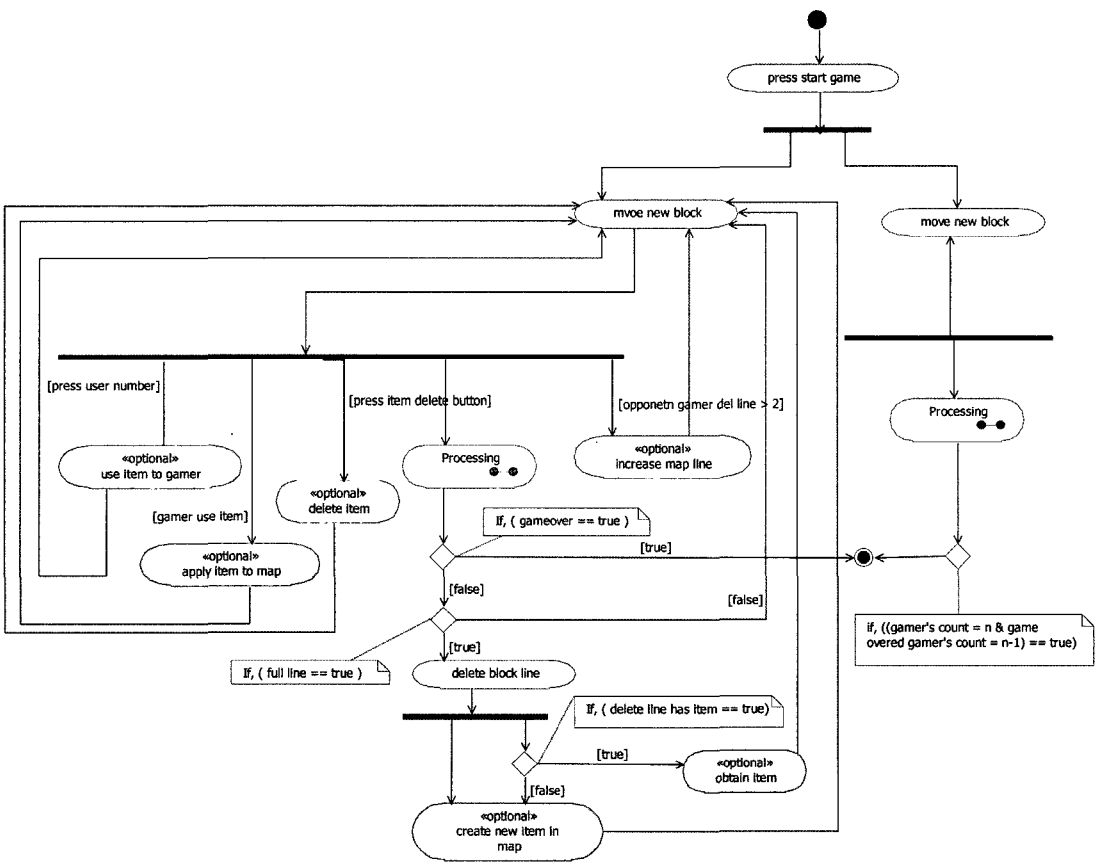

그림 9 테트리스 게임 도메인의 범위를 나타내는 액티비티 다이어그램

심자산들의 재사용성의 용이함을 제공하고 개발에 따른 시간과 노력을 절감해 줄 수 있음을 확인한다.

그림 8은 테트리스 게임의 기본 동작흐름을 나타내는 다이어그램이다. 그림 9는 기본 동작흐름을 바탕으로 본 도메인에서 분석하고자 하는 테트리스 게임의 범위를 나타내는 액티비티 다이어그램이다. 그림 9에서 기본 동작흐름(그림 8)은 *Processing* 액티비티로 표현되어 있으며, 그 외, 아이템을 이용하는 기능과 두 사람 이상의 사용자가 대전할 수 있는 기능들이 다이어그램에 표현되어 있다. 이 범위에 속하는 테트리스 게임 애플리케이션들을 대상으로 본 논문에서 제시한 방법을 따라 가변성을 분석하여 도메인 핵심 자산을 개발하였다.

7.1 도메인 핵심 자산 개발

도메인 요구사항을 구성하는 여러 산출물 중, 도메인 요구사항의 CV_Property를 보여주는 산출물(그림 10)과 가변점을 보여주는 산출물(그림 11와 그림 12)이 다음과 같이 만들어진다. 그림 10에서 수집된 도메인 요구사항 중, 블록랜덤생성, 블록이동, 블록회전, 블록낙하 등의 PR들이 공통성의 속성으로 추출되었음을 보여준다. 그러나 도메인 요구사항 중, 아이템을 생성하고 획득하고 사용하고 버리는 등의 기능들은 선택적 속성으로 추출되었음을 보여준다. 이 중 게임레벨 선택과 블록랜덤생성에 대한 PR-명세서가 그림 11와 그림 12에 나와 있으며 도메인 요구사항 분석 단계에서 가변점이 식별된 모습을 보여주고 있다.

PR NO	기능	Ratio	기본 게임	오락실 게임	스펙실 테트리스	현당 테트리스	넷마블 테트리스 _혼자놀기	넷마블 테트리스 _놀이아템	넷마블 테트리스 _아이템
PR1	게임레벨선택	57.1	X	O	O	O	O	X	X
PR2	게임업선택	28.6	X	X	X	X	X	O	O
PR3	효과음on/off	57.1	X	O	X	X	O	O	O
PR4	콤팩낙하지점표시on/off	42.9	X	X	X	X	O	O	O
PR5	대전플레선택	28.6	X	X	X	X	X	O	O
PR6	블록 랜덤 생성(7가지)	100.0	O	O	O	O	O	O	O
PR7	블록고정	100.0	O	O	O	O	O	O	O
PR8	블록이동	100.0	O	O	O	O	O	O	O
PR9	블록 회전	100.0	O	O	O	O	O	O	O
PR10	속도에 따른 블록 낙하	100.0	O	O	O	O	O	O	O
PR11	블록 바른 낙하	100.0	O	O	O	O	O	O	O
PR12	라인삭제	100.0	O	O	O	O	O	O	O
PR13	게임 시작	100.0	O	O	O	O	O	O	O
PR14	게임오버	100.0	O	O	O	O	O	O	O
PR15	게임속도조절	100.0	O	O	O	O	O	O	O
PR16	블록등장입보기	100.0	O	O	O	O	O	O	O
PR17	옆에 쌓인 블록보기	100.0	O	O	O	O	O	O	O
PR18	효과음플레이	100.0	O	O	O	O	O	O	O
PR19	시간계산	100.0	O	O	O	O	O	O	O
PR20	다음블록 미리보기	100.0	O	O	O	O	O	O	O
PR21	추가 블록 랜덤생성(4가지)	14.3	X	X	O	X	X	X	X
PR22	게임 선택 및 레벨업	71.4	O	O	O	O	O	X	X
PR23	점수계산	42.9	O	O	X	X	O	X	X
PR24	게임 일시 및 재시작	71.4	O	O	O	O	O	X	X
PR25	목표점수달성여부	14.3	X	X	X	O	X	X	X
PR26	아이템 생성	14.3	X	X	X	X	X	X	O
PR27	아이템 획득	14.3	X	X	X	X	X	X	O
PR28	아이템 사용	14.3	X	X	X	X	X	X	O
PR29	아이템버리기	14.3	X	X	X	X	X	X	O
PR30	상대방 라인생성	28.6	X	X	X	X	X	O	O
PR31	상대방 맵 보여주기	42.8	X	X	X	X	O	O	O
PR32	게임레벨 보여주기	71.4	O	O	O	O	O	X	X
PR33	점수 보여주기	57.1	O	O	O	X	O	X	X
PR34	게임 진행 시간 보여주기	14.3	X	O	X	X	X	X	X

그림 10 테트리스 게임 도메인에서 추출된 PR

PR1	게임레벨선택	Ratio	57.1%
Description	gamer가 게임의 레벨을 선택한다.		
Data	1. gamer는 게임을 시작하기 전에 레벨을 선택할 수 있다.	V.P 레벨범위 (var_level)	Variants 1..10
Data	2. gamer가 레벨을 선택하지 않은 경우는 레벨1에서 시작한다.	시작레벨	1

그림 11 게임레벨선택 PR 명세서

PR6	블럭랜덤생성	Ratio	100%
Description	블록을 랜덤으로 생성한다.		
Data	1. 게임이 시작되면 기본 블록 중 하나가 현재와 다음 사용할 블록으로 랜덤 하게 결정된다.	V.P 기본 블록수 범위 (block_no)	Variants 1..7
Computation		랜덤함수 algorithm random_1()	

그림 12 블록랜덤생성 PR 명세서

도메인 아키텍처 단계에서 하나 이상의 PR들이 하나의 기능 패키지로 추출이 되어 도메인 컴포넌트가 된다. 도메인 컴포넌트의 CV_Property는 도메인 요구사항의 PR CV_Property를 근거로 결정된다. 그림 13은 도메인 컴포넌트와 도메인 컴포넌트 바인딩으로 구성된 도메인 아키텍처 구성 모델을 보여준다. 여기서 도메인 컴포넌트 *ItemPlayMgr*는 그림 10의 PR26~PR29들로부터 추출이 된 것이며, PR들의 선택적 속성에 의해 이 도메인 컴포넌트의 CV_Property도 선택성을 가지게 되었다. 이를 <<optional>> 스테레오타입으로 명시되어있음을 볼 수 있다. 또한 선택성 도메인 컴포넌트로의 바인딩에 대한 가변성을 표현하기 위하여 도메인 컴포넌트 바인딩 위에 <<v.p>>로 명시하였다.

도메인 컴포넌트와 도메인 컴포넌트 바인딩의 CV_Property 가변성과 더불어, 이들의 내부 요소들에서 또한 가변점을 식별할 수 있다. 그림 14는 공통적 CV_Property 속성으로 결정된 *GamePlayMgr* 도메인 컴포넌트와 *GameMapMgr* 도메인 컴포넌트의 내부 구현을 위해 필요한 도메인 타입의 일부분을 보여준다. *GameMap* 도메인 타입은 맵에 대한 정보를 저장하고, 컨트롤하는 클래스이다. 이는 환경설정에서 사용자가 설정한 맵을 저장하고 맵의 라인을 삭제하고 맵에 블록을 고정하는 기능 등을 제공한다. 특히 맵에 아이템의 효과를 적용하는 여러 가지 기능들은 요구사항으로부터 선택적 기능으로 분석되었기 때문에 도메인 타입의 오퍼레이션에 명시적으로 <<v.p>> 스테레오타입을 사용하여 이 오퍼레이션이 가변적임을 표현하였다.

위에서 살펴본 사례 연구를 통해 요구사항 분석 단계에서부터 추출된 공통성과 가변성이 아키텍처와 컴포넌트 개발 단계에 연결되어 그 정보가 유지됨을 알 수 있다. 또한 각 단계별 추출되어야 하는 가변성의 정보가 그 단계의 산출물을 구성하는 구성요소들의 유형에 맞게 본 논문에서 제시한 가변성 유형의 형태를 가지고 추출됨을 알 수 있다. 이는 각 단계별 추출된 가변성 간에 추적을 가능하게 해 준다.

7.2 도메인 자산 기반 애플리케이션 개발

본 연구의 사례연구로써 개발한 테트리스 게임 도메인의 핵심자산들이 애플리케이션 개발시재사용의 용이함을 제공하고 개발 시간을 단축시킴을 보이기 위하여

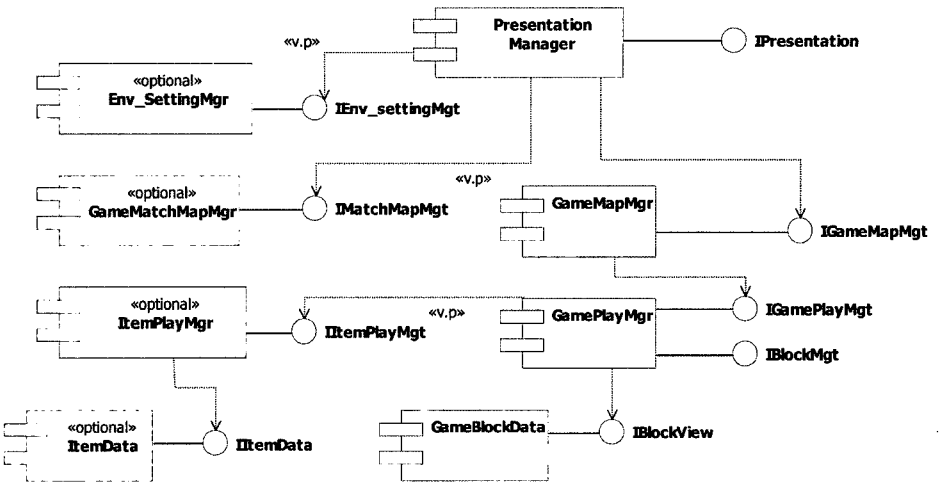

그림 13 테트리스 게임 도메인에서 추출된 도메인 컴포넌트와 도메인 컴포넌트 바인딩 - 도메인 컴포넌트 CV_Property는 스테레오 타입 <<optional>>을 사용하여 표현되어 있으며, 도메인 컴포넌트 바인딩 CV_Property는 스테레오 타입 <<v.p>>를 사용하여 표현되어 있다.

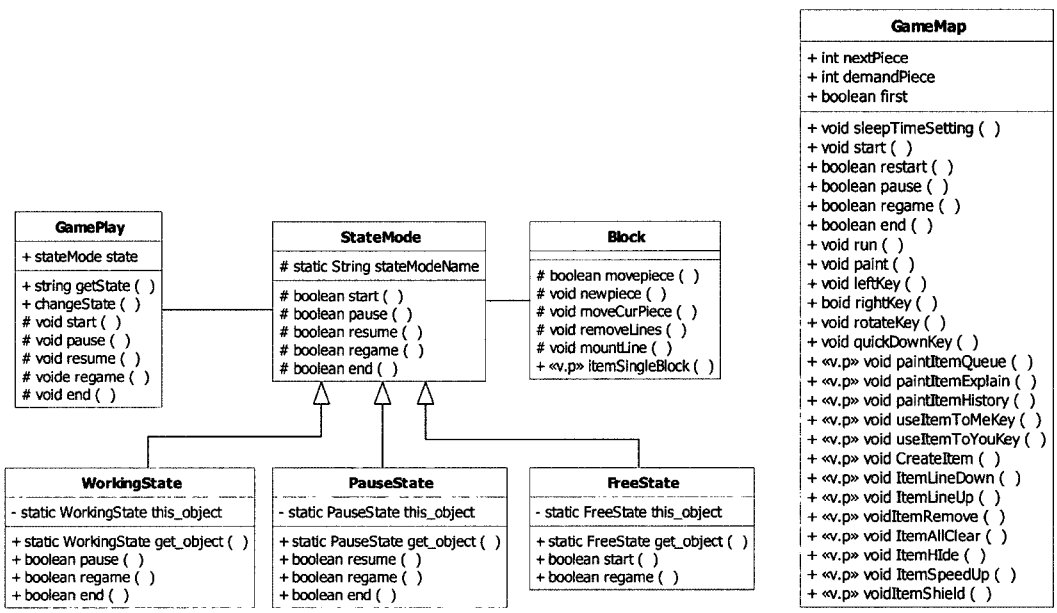

그림 14 *GamePlayMgr* 도메인 컴포넌트와 *GameMapMgr* 도메인 컴포넌트에 속하는 도메인 타입들

다음과 같이 두 가지 형태의 실험을 하였다.

- 도메인 핵심자산을 사용하지 않고 다양한 형태의 테트리스 게임을 개발한다.
- 도메인 핵심자산을 사용하여 다양한 형태의 테트리스 게임을 개발한다.

표 3은 개발한 테트리스 게임 애플리케이션의 유형과 각각의 개발 시간을 정리한 표이다. 개발된 테트리스 게임은 도메인 범위 내에 속하는 공통적 기능 PR들과 여러 형태의 선택적 기능 PR들로 이루어진 4가지 유형(Application 1~4)과 도메인 범위 내에서 분석되지 않은 특정 요구사항들을 가지는 3가지(Application 5~7) 유형으로 개발되었다.

각 애플리케이션 유형별 사용된 도메인 컴포넌트들은 다음과 같다(그림 13 참조).

- Application 1: *Presentation Manager, GameMapMgr, GamePlayMgr, GameBlockData* (=common Domain components)
- Application 2: common Domain components +

표 3 개발된 테트리스 게임 애플리케이션의 유형 및 개발 시간

Domain Assets development time	19days	Application type		Develop without Tetris domain assets
Develop based on Tetris domain assets	Used PRs type			
2	common PRs	Application 1 싱글테트리스	한 게이머가 한 맵을 사용하여 진행하는 테트리스 게임	5
3.5	common PRs optional PRs (1)	Application 2 싱글대전 테트리스	두 게이머가 각각 한 맵으로 동시에 진행하는 대전 형식의 테트리스 게임	6.5
4	common PRs optional PRs (2)	Application 3 아이템 싱글테트리스	한 게이머가 한 맵과 아이템 조각을 사용하여 진행하는 테트리스 게임	7.5
5	common PRs optional PRs (3)	Application 4 아이템 싱글대전 테트리스	두 게이머가 각각 한 맵으로 아이템 조각을 사용하여 동시에 진행하는 대전 형식의 테트리스 게임	9
4	common PRs optional PRs Specific req.(1)	Application 5 듀얼협동테트리스	두 게이머가 동시에 한 맵을 사용하여 진행하는 테트리스 게임	8
5	common PRs optional PRs Specific req.(2)	Application 6 네트워크 싱글대전테트리스	한 게이머가 한 맵을 사용하여 진행하여 네트워크 대전하는 형식의 테트리스 게임	8
5.5	common PRs optional PRs Specific req.(3)	Application 7 네트워크 듀얼협동 테트리스	두 게이머가 한 맵을 사용하여 진행하여 네트워크 대전하는 형식의 테트리스 게임	10
29.0	**48**			**54**

GameMatchMapMgr, Env_SettingMgr (optional Domain components)

- Application 3: common Domain components + *ItemPlayMgr, ItemData, Env_SettingMgr* (optional Domain components)
- Application 4: common Domain components + *GameMatchMapMgr, ItemPlayMgr, ItemData, Env_SettingMgr* (optional Domain components)
- Application 5: common Domain components + *Env_SettingMgr, DualMapMgr* component 추가
- Application 6: common Domain components + *Env_SettingMgr, NetMapMgr* component 추가
- Application 7: common Domain components + *Env_SettingMgr, Net_DualMapMgr* component 추가

도메인 자산을 개발하기 위한 시간이 3명의 man power를 가지고 19일 소요되었으며, 각 애플리케이션을 개발하는 시간이 총 29일과 54일 소요되었다. 전체적으로 각 실험마다 48일과 54일이 소요되었다. 그림 15는 두 실험의 개발 소요 시간을 나타내는 그래프이다. 이 그래프를 통해 앞으로 도메인 자산을 기반으로 애플리케이션의 개발이 증가하면 할수록 두 실험의 비교값은 더 커짐을 예측할 수 있다.

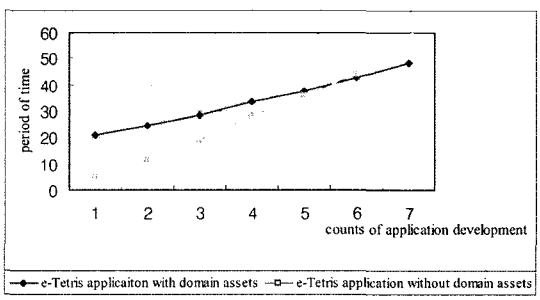

그림 15 도메인 핵심자산 사용 유무에 따른 애플리케이션 개발 시간 비교

8. 결론 및 향후 연구

본 논문에서는 소프트웨어 프로덕트 라인에서 요구사항, 아키텍처, 컴포넌트를 핵심 자산으로 개발하기 위해 이들에 대한 가변성을 2차원적 분석을 통하여 이루어지도록 하였다. 먼저, 수직적 가변성 분석은 가변성의 추상화 정도에 따라 구분하여 2 수준으로 나타내었다. 수평적 가변성 분석은 도메인 요구사항, 도메인 아키텍처, 도메인 컴포넌트에 따라 구분하여 수행되도록 하였다. 이들의 각기 다른 수준에서 C&V의 개념을 결합하고 핵심자산의 특징에 따라 가질 수 있는 가변성의 유형들을 분류하기 위해 각각 메타모델을 제시하였다. 마지막으로 이들에 대한 메타모델을 연결하여 핵심자산에 따른 가변성의 전체적인 연관관계를 성립하였다. 핵심 자산들의 연관 관계를 바탕으로 가변성을 연속해서 분석

해 감으로써 소프트웨어 재사용이 개발 프로세스 전 단계에 끊어짐 없이(seamless) 연속적인 부분으로 수행될 수 있게 할 수 있다. 이는 테트리스 게임 도메인을 대상으로 수행한, 본 연구에 대한 사례연구를 통해 확인할 수 있었다.

향후 연구에서는 본 연구에서 수행한 가변성의 연관관계를 바탕으로 CV_Property 가변성뿐만 아니라 도메인 요구사항의 가변점 유형이 도메인 아키텍처와 도메인 컴포넌트의 가변점 유형에 영향을 주는 여러가지 형태의 가변점 추적 메케니즘을 정의하고자 한다. 또한 소프트웨어 프로덕트 라인에서 재사용 가능한 핵심 자산들이 효율적으로 재사용되도록 재사용 환경 -핵심 자산 저장소, 공통성 분석 도구, 가변성 관리 도구 등- 을 구축하는 것이다.

참 고 문 헌

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

[2] Griss, M., Favaro, J., d'Alessandro., M., "Integrating Feature Modeling with the RSEB," In Proceedings of the Fifth International Conference on Software Reuse, Canada, pp.76-85, 1998.

[3] Chastek, G., Donohoe, P., Kang, K., and Thiel, S., "Product Line Analysis: A Practical Introdution," (CMU/SEI-2001-TR-001), Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 2001.

[4] Mannion, M., "Using First-Order Logic for Product Line Model Validation," In Proceedings of Second Product Line Conference (SPLC2), pp.176-187, 2002.

[5] Trigaux, J.C., and Heymans, P., "Modelling variability requirements in Software Product Lines: A Comparative Survey, Technical Report PLENTY project, Institute Information FUNDP, Namur, Belgium, Nov. 2003.

[6] Kuusela, J. and Savolainen, J., "Requirements Engineering for Product Families," In Proceedings of the 22nd International Conference on Software Engineering (ICSE'00), Limeric, Ireland, pp.60-68, 2000.

[7] Thompson, J.M. and Heimdahl, M.P.E., "Structuring Product Family Requirements for n-Dimensional and Hierarchical Product Lines," Requirements Engineering, vol. 8 pp.42-54, 2003.

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

[9] Rob van Ommering, "Building Product Populations with Software Components," In Proceedings of the 24th International conference on Software Engineering (ICSE'02), 2002.

[10] Keepence, B. and Mannion, M., "Using Patterns to Model Variability in Product Families," IEEE Software, vol. 16, no. 4, pp.102-108, 1999.

[11] Clauß, M., "Generic Modeling using UML Extensions for Variability," OOPSLA 2001, Workshop on Domain Specific Visual Languages, 2001.

[12] Webber, D. and Gomaa, H., "Modeling Variability with the Variation Point Model," In Proceedings of the 7th International Conference on Software Reuse (ICSR7), pp.109-122, 2002.

[13] Bosch, J., *Design and Use of Software Architectures*, Addison Wesley, 2000.

[14] Svahnberg, M. and Bosch, J., "Evolution in Software Product Lines: Two Cases," Journal of Software Maintenance: Research and Practice, vol. 1 no. 6 pp.391-422, 1999.

[15] Atkinson, C., Bayer, J., Bunse, C., etc, *Component-Based Product Line Engineering with UML*, Addison Wesley, 2001.

[16] Cheesman, J. and Daniels, J., *UML Components A Simple Process for Specifying Component-Based Software*, Addison Wesley, 2001.

[17] david M. Weiss and Chi Tau Robert Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, 1999.

[18] Moon, M.K., Yeom, K.H, and Chae, H.S., "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.

문 미 경

1990년 2월 이화여자대학교 전자계산학과(학사). 1992년 2월 이화여자대학교 전자계산학과(석사). 2005년 2월 부산대학교 컴퓨터공학과(박사). 2005년 3월~2005년 8월 부산대학교 차세대물류IT기술개발 연구소 Post-doc. 2005년 9월~현재 부산대학교 컴퓨터 및 정보통신 연구소 교수. 관심분야는 소프트웨어 프로덕트 라인 공학, 적응형 소프트웨어 개발, RFID기반 미들웨어 및 RFID 비즈니스 이벤트 프레임워크 개발 등임

채 홍 석

정보과학회논문지 : 소프트웨어 및 응용
제 33 권 제 2 호 참조

염 근 혁

정보과학회논문지 : 소프트웨어 및 응용
제 33 권 제 2 호 참조