제품 계열 공학에서의 산출물간의 추적성 기법
(A Method of Applying Traceability among Product Line Engineering Artifacts)

라 현 정 † 장 수 호 †† 김 수 동 †††
(Hyun Jung La) (Soo Ho Chang) (Soo Dong Kim)

요 약 제품계열 공학(Product Line Engineering, PLE)은 핵심 자산을 이용하여 어플리케이션을 경제적으로 개발하는 대표적인 재사용 기술이다. PLE는 프레임웍(Framework) 공학과 어플리케이션 공학으로 구성된다. 프레임웍 공학은 한 도메인 내에 있는 여러 패밀리 멤버들이 가지고 있는 공통적인 기능인 핵심 자산을 개발하는 단계이고, 어플리케이션 공학은 핵심 자산을 패밀리 멤버에 맞게 인스턴스화하여 어플리케이션을 생산하는 단계이다. PLE는 핵심 자산을 이용하여 특정 어플리케이션을 개발함으로써 재사용성이 높을 뿐 아니라 어플리케이션을 적은 시간과 노력으로 만들 수 있으므로 개발하는 효율성도 높다. 그러나, PLE 개발 절차에 대한 산출물 정의 및 템플릿 제공이 미비하여 개발자들이 PLE 프로세스를 따라 산출물을 만드는데 어려움이 있고, 산출물간 관계 정의가 충분하지 못하여 산출물간 일관성을 유지하기 힘들어 개발자들은 PLE 프로세스의 실용적 적용에 어려움이 있다. 본 논문에서는 PLE의 핵심단계인 프레임웍 공학 과정의 각 단계마다 도출되는 산출물의 메타모델을 정의하고 각 산출물간의 추적 관계를 추적성 맵(Traceabitliy Map)으로 나타내며 산출물간 추적 관계를 적용할 수 있는 지침을 제시한다. 마지막으로, 추적성 맵에 대한 평가와 적용되는 방법을 제시한다.

키워드 : 제품 계열 공학, 핵심 자산, 메타 모델, 추적성

Abstract Product Line Engineering(PLE) is one of the technologies that develop applications economically reusing core assets. PLE consists of Framework Engineering(FE) and Application Engineering. Framework Engineering is to develop core assets that have common functionality shared by a set of family members. Application Engineering is to develop a specific application by instantiating the core assets. The PLE process increases reusability and efficiency because a specific application is developed by using core assets with less time and effort. Since definition of PLE artifacts and relationship between artifacts are not clear, developers have several troubles to make artifacts based on PLE process, are difficult to maintain consistency between artifacts, and do not use PLE process more practically. In this paper, we define meta-models of artifacts that are produced in FE activities of PLE process and describe the traceability relationship between artifacts by using traceability map and guidelines that can apply traceability relationship. Finally, we define the way how trace links and guidelines of traceability map are applied.

Key words : Product Line Engineering, Core Asset, Meta-model, Traceability

1. 서 론

PLE는 핵심 자산을 이용하여 어플리케이션을 경제적으로 개발하는 대표적인 재사용 기술이다. PLE는 프레임웍 공학과 어플리케이션 공학으로 구성된다[1]. 프레임웍 공학은 한 도메인 내에 있는 여러 패밀리 멤버들이 가지고 있는 공통적인 기능을 핵심 자산으로 개발하는 단계이고, 어플리케이션 공학은 핵심 자산을 패밀리 멤버에 맞게 인스턴스화하여 어플리케이션을 개발하는 단계이다. PLE는 핵심 자산을 이용하여 특정 어플리케이션을 개발함으로써 재사용성이 높을 뿐 아니라 적은 시간과 노력으로 어플리케이션을 만들 수 있으므로 효율성도 높다. 그러나 PLE가 이런 장점을 갖는데 비해

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-005-D00172)
† 학생회원 : 숭실대학교 컴퓨터학과
hjla@otlab.ssu.ac.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
shchang@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@ssu.ac.kr
논문접수 : 2004년 7월 29일
심사완료 : 2005년 2월 15일

현재 PLE의 프로세스나 산출물의 정의는 개념적인 수준에 그쳐, 산업계에서 널리 이용되지 못하고 있다. 그 이유는 PLE는 프로세스의 자세한 절차와 지침이 부족하고 정의되어 있는 프로세스는 프레임웍 공학 과정에 치중되어 있으며, 산출물이 메타모델 수준에서의 구체적인 정의와 템플릿 제공이 미비하여 개발자가 실용적으로 프로세스를 따라 산출물을 생산하기에 어려움이 있고, PLE의 복잡한 산출물의 일관성을 효율적으로 관리, 유지보수 할 수 있는 방법이 제시되지 않기 때문이다 [2]. 이런 문제를 해결하기 위해 보다 구체적으로 정의된 프로세스와 상세한 지침이 필요할 뿐만 아니라 프로세스의 각 단계마다 나와야 하는 산출물이 명확하게 정의되고, 산출물의 템플릿도 구체적으로 제시되어야 한다. 그리고 각 단계의 산출물의 연관 관계를 명확하게 함으로써 개발도중 변경되는 사항에 대한 산출물간의 일관성을 유지하도록 하고, 기존의 프로세스를 실용적으로 만들어 개발자가 쉽게 제품계열 공학 개념을 이용하여 고품질의 핵심 자산과 어플리케이션을 개발할 수 있게 한다.

본 논문에서는 PLE의 한계점을 극복하기 위해 4장에서 PLE의 핵심단계인 프레임웍 공학 과정의 각 단계마다 도출되는 산출물의 메타모델을 정의하고 5장에서는 각 산출물간의 추적 관계를 정의하여 추적성 맵으로 나타내며 산출물간 추적 관계를 적용할 수 있는 지침을 제시하고 6장에서는 추적성 맵에 대한 평가와 적용되는 방법을 제시하며, 7장에서 결론을 맺는다.

2. 관련연구

2.1 Bayer의 연구[3]

Bayer는 제품 계열을 효과적으로 개발하고 유지하기 위해 PuLSE 프로세스의 각 단계를 추적할 수 있는 방법을 제시한다. 효율적 추적을 위해 추적을 잠재적인 추적과 구체적인 추적으로 분류한다. 잠재적인 추적은 프로세스의 메타모델 사이에서 정의되고, 이는 각 프로젝트의 PuLSE-BC(Baselining and Customization) 단계에 구체적인 추적으로 특화된다. 구체적인 추적은 개발하고자 하는 핵심 자산에 맞게 생성된 범용 모델(Generic Model), 의사 결정 모델(Decision Model), 시스템에 특정한 모델(System Specific Model)의 구성 요소 관계 사이에서 이루어진다. 제품 계열을 개발하는 각 단계, 각 단계를 바라보는 뷰, 뷰로 인해 생긴 모델은 제품 계열의 추적을 정의하는데 도움이 된다. 그러나 이 연구는 프로세스의 메타모델과 범용 모델, 의사 결정 모델, 시스템에 특정한 모델 간의 추적 관계를 정의하고, 각 모델의 구성요소 간의 추적 관계 정의는 미비하여 추적 관계가 다소 추상적이고 완전하지 못하다.

2.2 KobrA의 메타모델[4]

KobrA 메타모델는 패키지들의 계층구조로 이루어져 있다. 최상위 레벨의 메타모델 패키지는 KobrA 방법론의 기본요소를 나타내는 핵심 패키지(Core package)와 KobrA의 유지활동과 관련된 요소들을 정의하는 유지 패키지(Maintenance package)로 구성되며, 각각은 좀 더 낮은 레벨의 여러 하위 패키지를 포함한다. 핵심 패키지는 KobrA의 논리적인 컴포넌트와 관련된 기본 요소를 정의하는 Komponent 패키지, Komponent의 품질 보증(Quality assurance) 패키지, 시스템 패키지, 산출물을 생산하기 위한 기술적 요소를 정의하는 구현(Implementation suite) 패키지로 구성되어 있다. 핵심자산과 관련된 패키지인 Komponent 패키지는 프레임웍 공학과 어플리케이션 공학 프로세스 과정에 생성된 Komponent의 기본적인 구조를 정의한다. Komponent의 서브 패키지들도 여러 하위 레벨로 나뉘어질 수 있으며, 여기서 나타난 하위 레벨은 Komponent를 개발할 때 중간 산출물로 나올 수 있는 UML 다이어그램과 명세 등을 포함한다. 유지 패키지에는 자산(Asset) 패키지, 변경(Change) 패키지, 전달(Deliverable) 패키지가 포함되어 있다. KobrA 메타모델은 핵심자산의 구성요소 중 컴포넌트에 대해서는 상세히 정의하고 있지만, 아키텍처, 의사 결정의 메타모델과 관련된 자세한 정의는 부족하다.

3. PLE의 대표적 산출물

이 단계에서는 여러 PLE 프로세스의 대표적인 PLE 산출물을 객관적으로 도출하기 위해 여러 프로세스와 각 프로세스의 산출물을 개략적으로 살펴본다.

3.1 SEI의 Software Product Line

SEI Software Product Line은 카네기 멜론 대학 내 소프트웨어 공학 연구실에서 제안된 프로세스이다[1]. 이 프로세스는 핵심 자산 개발(Core Asset Development), 제품 개발(Product Development), 관리(Management) 3가지 활동으로 이루어져 있다. 핵심 자산 개발은 한 도메인에 속하는 여러 제품들이 공통으로 사용하는 핵심 자산을 개발하는 단계이고, 제품 개발 단계에서는 핵심 자산을 이용하여 특정 제품을 개발하는 단계이다. 마지막으로 관리 활동은 제품 계열 공학의 성공 여부를 결정하는 중요한 역할을 하고 기술적인 관리와 조직적인 관리로 나뉘어 진다. 대표적인 산출물로는 제품 계열 범위, 핵심자산, 제품이 있으며 핵심 자산에는 범용 아키텍처, 소프트웨어 컴포넌트, COTS 컴포넌트, 비즈니스 케이스, 가변성 첨부 프로세스(Attached process)가 포함되어 있다.

3.2 Feature-Oriented Reuse Method(FORM)

FORM은 요구사항 공학에서 휘처 모델(Feature model) 개념을 사용한 FODA를 소프트웨어 설계와 구현 단계로 확장한 것이다[5,6]. FORM은 주로 도메인에 속하는 어플리케이션의 공통성과 가변성을 휘처 단위로 식별하여 휘처 모델에 나타내고, 휘처 모델을 사용하여 도메인 아키텍처와 컴포넌트를 개발하는데 중점을 두고 있다. FORM은 크게 도메인 공학 프로세스와 어플리케이션 공학 프로세스로 구성된다. 도메인 공학 프로세스는 도메인 내에 포함되는 어플리케이션을 개발하는데 재사용될 수 있는 도메인 산출물인 참조 아키텍처와 재사용 가능한 컴포넌트를 개발하는 단계이고, 어플리케이션 공학 프로세스는 사용자의 요구사항을 분석하고 도메인 공학 단계에서 나온 참조 아키텍처와 컴포넌트 중 적당한 것을 선택하여 특정 어플리케이션을 만드는 단계이다. 대표적인 산출물로는 휘처 모델, 참조 아키텍처, 재사용 가능한 컴포넌트, 어플리케이션 소프트웨어가 있다.

3.3 PuLSE

PuLSE는 독일 IESE 연구소에서 제안한 제품 계열 개발 방법론이다[4]. 이 프로세스는 배치 단계, 기술 컴포넌트, 지원 컴포넌트 중요한 세 가지 요소로 구성된다. 배치 단계는 제품 계열을 개발하는 논리적인 단계이고, 기술 컴포넌트는 제품 계열을 개발하는데 필요한 기술적 노하우를 제공하며 배치 단계 내내 사용된다. 기술 컴포넌트는 특화된 프로세스를 이용하여 도메인 모델, 의사 결정 모델, 참조 아키텍처를 만들고, 이를 인스턴스화하여 특정 제품을 만드는 단계로 구성된다. 대표적인 산출물로는 특정 제품에 특화된 프로젝트 계획과 프로세스, 제품 지도와 특성 목록, 도메인 모델, 도메인 의사 결정 모델, 참조 아키텍처, 아키텍처 의사 결정 모델, 제품이 있다.

3.4 KobrA

KobrA는 UML을 이용하여 컴포넌트 기반으로 한 제품 계열 공학을 접근하는 프로세스이다[7]. 이 프로세스는 프레임웍 공학, 어플리케이션 공학, 프로젝트 관리, 형상과 변경 관리, 환경 관리의 5가지 서브프로세스로 이루어져 있다. 프레임웍 공학에서는 범용적이고 재사용 가능한 기반 구조를 정의하고, 어플리케이션 공학 단계에서는 프레임웍 모델과 컴포넌트를 이용하여 특정한 패밀리 멤버를 개발한다.

3.5 각 PLE 프로세스의 산출물

위에서 열거한 각 PLE 프로세스는 일반적으로 이름은 다르지만 공통적인 활동들로 구성되어 있고, 각 활동은 이름은 다르지만 공통적인 특성을 가지는 산출물인 제품 계열 범위, 공통성과 가변성(Commonality and Variability, C&V) 명세, 범용 아키텍처, 컴포넌트 모델, 의사결정 모델을 생성한다. 프레임웍 공학에서는 개발하려는 어플리케이션들을 포함하는 도메인을 분석한 후 해당 도메인의 공통성과 가변성을 식별하여 아키텍처, 컴포넌트, 의사 결정 모델을 포함하는 핵심 자산을 개발한다.

제품 계열 범위는 비즈니스 케이스 분석을 하고 패밀리 멤버와 관련된 휘처를 식별하여 제품 계열 범위를 설정하는 단계의 산출물이고, C&V 명세는 해당 도메인의 공통성과 가변성을 명시한 산출물이며, 범용 아키텍처, 컴포넌트 모델, 의사 결정 모델은 핵심 자산의 구성요소이다. 이 중 C&V 명세는 기존 프로세스에서 핵심 자산에 내포되어 구체적으로 명시되지 않았지만 본 논문에서는 C&V 명세가 핵심 자산을 설계하는데 기반이 되므로 별도로 명시하였다. 이 산출물은 메타모델의 단위이자 추적 대상이 된다.

4. 메타모델

이 장에서는 앞에서 비교한 프로세스의 다섯 개 산출물에 대한 메타모델을 제시한다. 각 메타모델은 논리적인 요소를 나타내는 산출물 계층(Artifact Layer), 요소 계층(Element Layer), 하위 요소 계층(Sub-Element Layer)과 물리적인 요소를 나타내는 표현 계층(Representation Layer)으로 구성된다.

4.1 제품 계열 범위

제품 계열 범위는 비즈니스 케이스를 통해 경제성을 분석하여 해당 도메인에 관련된 패밀리 멤버와 휘처를 식별하는 분석을 통해 얻은 산출물이다. 그림 1은 제품 계열 범위의 메타모델을 나타낸 것이다. 제품 계열 범위는 한 도메인 내에서 제품 계열에 속할 수 있는 제품으로 구성되며, 한 제품은 여러 기관에 의해 사용될 수 있고, 한 기관은 제품 계열에 속하는 여러 제품을 사용할 수 있다. 제품은 기관이 공통으로 사용하는 공통 요구사항과 특정한 패밀리 멤버에 종속된 제품에 특정한 요구사항으로 구성된다. 제품 계열 범위에 속하는 제품에 특

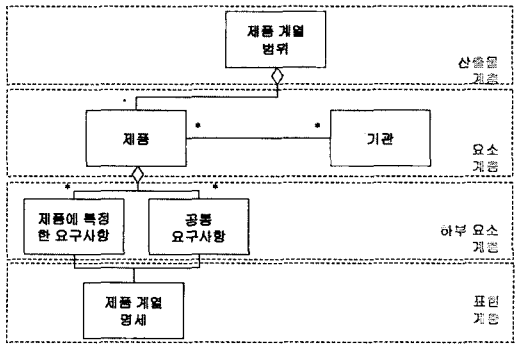

그림 1 제품 계열 범위의 메타모델

정한 요구사항과 공통 요구사항은 제품 계열 명세로 표현된다.

4.2 C&V 명세

C&V 명세는 여러 패밀리 멤버에서 공통으로 사용되는 공통성과 공통성 내에 패밀리 멤버마다 서로 다른 가변성을 식별하는 C&V 분석 단계의 산출물이다. 그림 2는 C&V 명세의 메타모델을 나타낸 것이다. C&V 명세는 공통 요구사항을 명시한 공통성 명세와 공통성 내 제품마다 다른 가변성을 나타낸 가변성 명세로 구성되어 있다. 공통성 명세는 기능적인 휘처와 비기능적인 휘처의 공통성을 나타내고 가변성 명세는 휘처 내의 가변점과 각 가변점에 해당하는 여러 가변치를 나타낸다. 기능적, 비기능적 휘처는 FORM 에서 제시된 휘처 모델을 사용하여 표현되고[5], 특히 기능적인 휘처는 유즈 케이스 모델로 표현될 수 있다. 또한, 가변성 명세는 공통성 명세로부터 추출되므로 휘처 모델, 유즈 케이스 모델과 가변성 명세표(Variability Range Table)로도 표현된다. 휘처 모델이나 유즈 케이스 모델에는 휘처 자체가 가변점이 되는 필수(mandatory), 선택(optional), 대안(alternative) 휘처가 표현되지만[5,6], 가변성 명세표에는 이외에 한 휘처 내에 발생할 수 있는 속성, 로직, 워크플로우, 데이터베이스에 저장될 수 있는 영구 데이터의 가변성도 표현된다[9]. 가변성 명세표에 명시된 가변성은 핵심 자산에 나타나는 모든 가변성의 기초가 되므로 핵심 자산에 표현된 복잡한 가변성을 효율적으로 관리하는데 중요한 역할을 한다.

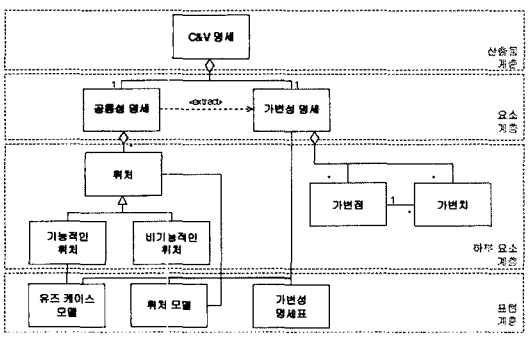

그림 2 C&V 명세의 메타모델

4.3 핵심 자산

핵심 자산은 어플리케이션들의 공통 자산으로, 컴포넌트보다 큰 단위로 표현된다. 그림 3은 핵심 자산의 메타모델을 나타내며, 핵심 자산은 범용 아키텍처, 컴포넌트 모델과 의사 결정 모델로 구성된다.

아키텍처는 일반적으로 모듈 뷰타입(Module Viewtype), C&C 뷰타입(Component and Connector Viewtype, C&C Viewtype), 할당 뷰타입(Allocation Viewtype)의 세 가지 뷰로 정의된다[10]. 범용 아키텍처는 여러 패밀리 멤버가 공통으로 사용하는 아키텍처로서 공통성뿐만 아니라 패밀리 멤버마다 다른 가변성도 포함한다. 모듈 뷰타입은 잘 정의된 인터페이스를 가지며 서로 연관된 기능으로 묶인 컴포넌트와 컴포넌트간의 관계를 나타내고, 기능모델(Functional Model)과 객체 모델로 표현된다. C&C 뷰타입은 실행 시간에서 컴포넌트 간의 상호작용 관계를 나타내며, 행동 모델(Behavioral Model)로 표현된다. 할당 뷰타입은 어플리케이션과 해당 어플리케이션이 실행되는 환경요소간의 관계를 나타낸 것으로, 모듈 뷰타입과 C&C 뷰타입에 비해 여러 패밀리 멤버의 공통성이 상대적으로 적어 할당 뷰타입는 고려하지 않는다.

컴포넌트 모델은 여러 클래스로 구성된 컴포넌트와 인터페이스로 구성되며 컴포넌트간의 관계는 범용 아키텍처에 표현된다. 컴포넌트는 컴포넌트 명세로 나타내어

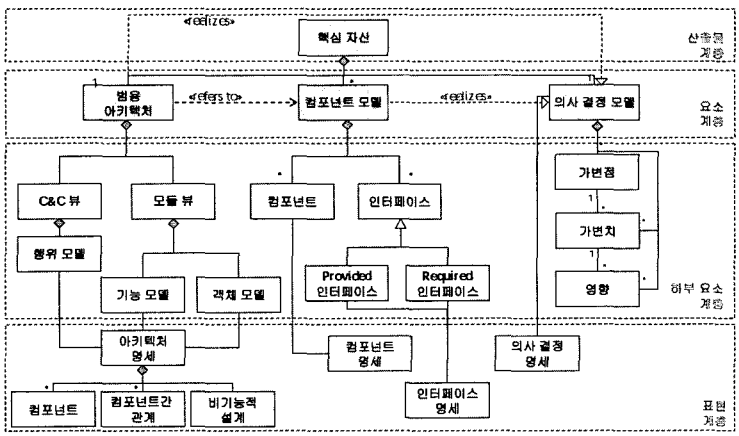

그림 3 핵심 자산의 메타모델

지며, 인터페이스는 인터페이스 명세로 표현된다.

컴포넌트뿐 아니라 범용 아키텍처에도 가변성이 발생한다. 아키텍처의 가변성은 뷰에 따라 여러 가변성이 존재하고, 가변성에 따라서 아키텍처는 다르게 표현된다[11]. 의사 결정 모델(Decision Model)은 C&V 명세 중 가변성 명세를 상세화한 것으로 가변점, 가변치, 의사 결정했을 때 생기는 영향(Effect)으로 구성되고 의사 결정 명세로 표현된다[4]. 의사 결정 명세에 명시된 가변성은 행위모델, 기능모델, 객체모델을 포함하는 아키텍처 명세, 컴포넌트 명세, 인터페이스 명세에 적용되어 설계된다.

5. 추적 링크(Trace Link) 및 매핑 지침

이 장에서는 4장에서 정의한 산출물의 메타 모델을 기반으로 산출물간의 추적링크를 정의하고 이를 추적성 맵으로 표현한다. 추적성 맵은 각 산출물간 관계를 표현하여 한 산출물의 요소가 다른 산출물의 요소로 어떻게 추적되는지를 알아보기 위해 제안된 그림으로 그림 4, 그림 5, 그림 6과 같이 표현된다. 그림 4에서는 각 산출물 간의 추적 링크를 나타내고, 각 추적 링크에 포함된 더 상세한 수준의 여러 추적 항목은 그림 5와 그림 6에 표현된다.

위의 그림과 같이, 추적성 맵에는 2개 추적 링크가 있다. 각 추적 링크에 포함된 추적 항목은 5.1에서 자세하게 다뤄지며, 5.2에서는 각 추적 링크에 해당하는 상세한 지침을 설명한다. 5.1에서 표현된 추적 항목은 다른 산출물간의 추적 관계를 나타낸 것과 한 산출물 내의 추적 관계를 나타낸 것이 있다.

5.1 추적 링크

5.1.1 제품 계열 범위와 C&V 명세간 추적

추적 링크 1은 제품 계열 범위와 C&V 명세의 관계를 나타내고, 그림 5와 같이 15개의 추적 항목을 가진다. 제품 계열 범위와 C&V 명세 간의 추적 관계를 나타낸 추적 항목은 추적 항목 1.1, 1.3, 1.4, 1.5, 1.8, 1.10, 1.11, 1.12, 1.14, 1.15에 나타나고, C&V 명세 내의 추적 관계를 나타낸 추적 항목은 추적 항목 1.2, 1.6, 1.7, 1.9, 1.13에 표현된다.

제품 계열 명세의 공통 요구사항은 기능적인 요구사항과 비기능적인 요구사항으로 구성된다. 기능적인 요구사항은 사용자가 인식할 수 있는 기능 단위로, 설계 시에 하나 이상의 더 작은 기능 단위인 유즈 케이스와 개발자나 사용자의 시각에서 식별 가능한 고유한 특성을 가진 단위인 기능적인 휘처[5]로 분할되거나 상세화되고, 공통 요구사항의 가변성 정보도 같이 표현되며, 이는 추적 항목 1.1과 1.8이 나타난다. 사전 조건(Pre-condition), 사후 조건(Post-condition), 제약 사항 등의 기능적 요구사항에 대한 상세한 정보, 기능을 수행하는 전반적인 흐름(flow), 가변성에 대한 정보가 유즈 케이스 명세에 명시되고, 이는 추적 항목 1.4이 나타낸다. 휘처가 기능적 요구사항을 수행하기 위해 연관된 다른 휘처의 기능을 사용하는 경우는 휘처 모델의 휘처간 관계로 설계되며, 추적 항목 1.13가 이 관계를 나타낸다.

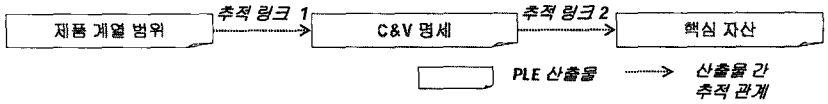

그림 4 추적성 맵

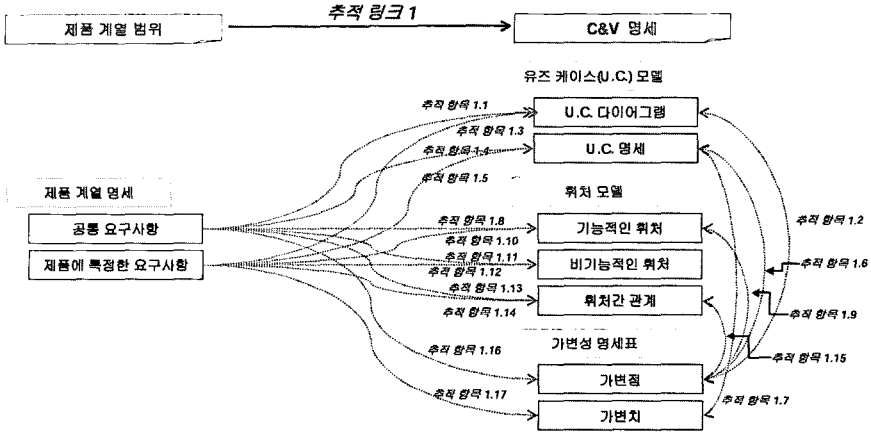

그림 5 추적 링크 1의 추적 항목에 대한 추적성 맵

비기능적인 요구사항은 제품 계열을 개발하는데 영향을 미치는 품질 요소(Quality Attribute)로 휘처 모델의 비기능적인 휘처로 표현되며 이 관계는 추적 항목 1.11이 나타낸다. 핵심 자산을 개발하는데 영향을 미치는 여러 품질 요소가 서로 충돌을 일으킬 수 있는 경우가 있다[12,13]. 이 관계는 공통 요구사항과 휘처 모델의 휘처간 관계를 매핑하는 추적 항목 1.13이 나타낸다.

핵심 자산에는 공통적인 기능 내에 멤버마다 다른 가변성을 가지고 있다. 공통 요구사항 내의 가변점에 대한 정보는 C&V 명세의 가변성 명세표에 가변점으로 표현된다. 한 공통 요구사항이 가지는 가변점은 가변성 명세표에서 여래 개의 가변점으로 표현된다. 이와 같은 관계는 추적 항목 1.16이 나타낸다.

제품 계열의 제품에 특정한 요구사항은 공통 요구사항의 가변점에 대한 멤버마다 다른 가변치 정보를 포함하여 가변성 명세표의 가변치로 표현되며, 이는 추적 항목 1.17가 나타낸다. PLE의 가변성은 유즈 케이스나 휘처 자체에서 발생할 수 있고, 유즈 케이스나 휘처 내에서 발생할 수 있다[14]. 전자의 경우 가변치는 유즈 케이스나 휘처가 되어 별도의 유즈 케이스나 휘처로 표현되고, 후자의 경우 가변치는 유즈 케이스나 휘처 내의 속성, 로직, 워크 플로우 등이 되어 유즈 케이스 명세에 표현된다. 이 관계는 제품에 특정한 요구사항과 유즈 케이스 다이어그램, 유즈 케이스 명세, 기능적인 휘처, 비기능적인 휘처의 매핑을 나타내는 추적 항목 1.3, 1.5, 1.10, 1.12으로 표시되어 있다. 추적 관계 1.13과 같이 비기능적 요구사항이 서로 충돌을 일으키는 경우 중, 충돌을 일으키는 비기능적 요구사항들이 가변성으로 도출되는 경우 서로간의 연관성은 정의되어야 한다. 이 관계는 제품에 특정한 요구사항과 휘처 모델의 휘처간 관계를 매핑하는 추적항목 1.14이 나타낸다.

이 외에 C&V 명세 내의 요소간의 관계를 나타내는 추적 항목 1.2, 1.6, 1.7, 1.9, 1.15이 있다. 한 산출물 요소 내의 추적 항목은 다른 산출물의 요소간 추적을 하면 묵시적으로 포함되어 추적되는 항목으로, 다른 산출물 요소 내의 추적 항목과는 달리 산출물 요소간의 일관성을 맞추기 위한 것이다.

5.1.2 C&V 명세와 핵심 자산간 추적

추적 링크 2는 C&V 명세와 핵심 자산간 관계를 나타내고, 그림 6과 같이 16개의 추적 항목을 포함한다. 핵심 자산 내의 추적 관계를 나타낸 추적 항목 2.2, 2.5, 2.9를 제외한 나머지는 C&V 명세와 핵심 자산 간의 관계를 나타낸 추적 항목이다.

한 유즈 케이스는 클러스터링 기법을 이용하여 연관있는 유즈 케이스들과 그룹핑되어 하나의 컴포넌트로 표현된다[8]. 여러 컴포넌트로 매핑되는 읽기 전용 오퍼레이션만 지닌 유스 케이스가 있고, 한 컴포넌트에도 매핑되지 않는 유즈 케이스가 생길 수 있다. 이 경우에는

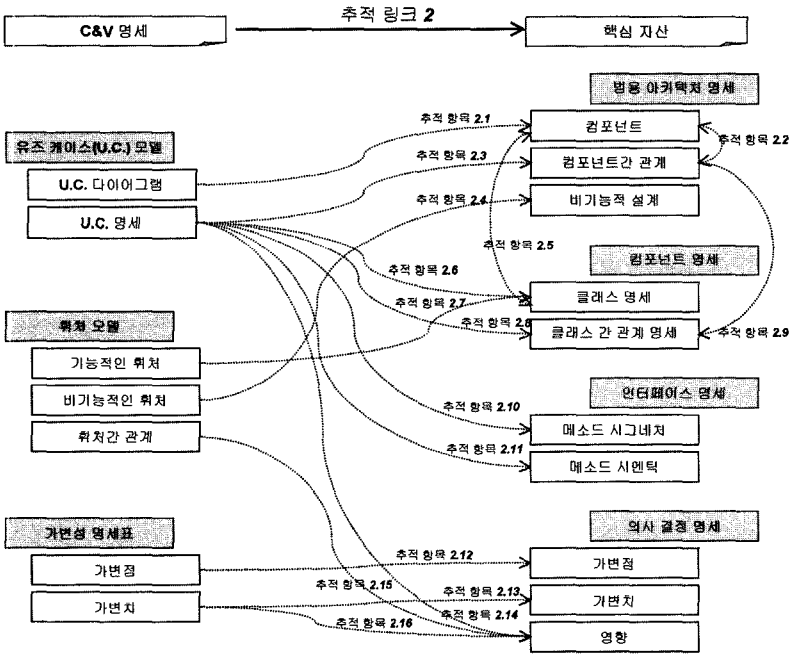

그림 6 추적 링크 2의 추적 항목에 대한 추적성 맵

[8]에서 제시한 지침을 바탕으로 가장 연관관계가 높은 한 컴포넌트로 매핑한다. 이런 관계는 추적 항목 2.1이 나타낸다.

유즈 케이스 명세는 유즈 케이스 기능, 사전 조건, 사후 조건, 제약 사항 등을 포함한 유즈 케이스에 대한 전반적인 설명과 기능을 수행하기 위한 흐름을 나타낸다. 흐름 중에는 같은 컴포넌트에 속한 다른 유즈 케이스를 호출하거나 다른 컴포넌트에 속한 유즈 케이스를 호출하는 정보도 포함한다. 두 흐름 모두 워크플로우를 발생하고, 이는 클래스 간의 관계와 컴포넌트 사이의 동적인 의존관계로 표현되고, 컴포넌트 간의 의존 관계는 C&C 뷰타입의 적절한 스타일을 선택함으로써 더 상세히 표현된다[10]. 이 관계는 추적 항목 2.3와 2.8가 나타낸다. 유즈 케이스 명세에 명시적 또는 암시적으로 명시된 객체, 속성, 오퍼레이션은 UML 표준을 적용하여 클래스 명세의 클래스, 클래스의 속성, 오퍼레이션으로 상세화되고 설계되고, 이 관계는 추적 항목 2.6이 나타낸다. 유즈 케이스 명세에 표시된 액터와 시스템간의 상호작용은 인터페이스에 포함되는 메소드를 설계하고 메소드 시그네처를 유도하는데 사용되며, 이 관계는 추적 항목 2.10이 나타내고, 흐름과 사전 조건, 사후 조건, 제약 사항은 메소드 시맨틱으로 표현되며, 이는 추적 항목 2.11이 나타낸다. 유즈 케이스 사이의 워크플로우에 가변성이 생기면 한 유즈 케이스의 가변성이 다른 유즈 케이스에 영향을 미칠 수 있고, 이는 추적 항목 2.14가 나타낸다.

휘처 모델의 기능적인 휘처도 유즈 케이스처럼 시스템의 기능을 나타내어 컴포넌트에 속하는 클래스들을 나타낼 수 있고, 이 관계는 추적 항목 2.7이 나타낸다.

휘처 모델의 비기능적인 휘처는 의존성, 이용가능성, 영속성과 같은 제품 계열의 비기능적 요구사항을 표현하며, 이는 아키텍처 설계에 적용되어야 한다. 이 관계는 추적 항목 2.4가 나타내지만, 비기능적인 휘처가 범용 아키텍처 내의 비기능적인 설계에 어떻게 체계적으로 사용되는지에 대한 설명은 이 논문의 범위를 벗어나므로 언급하지 않는다.

휘처 모델의 휘처간 관계는 휘처 간의 연관 관계를 나타내므로 한 휘처의 가변점에 가변치를 설정함으로써 다른 휘처에 미치는 영향을 유추할 수 있고, 이 관계는 추적 항목 2.15가 나타낸다.

C&V 명세의 가변성 명세표에는 개발하고자 하는 핵심 자산의 가변성에 대한 모든 정보를 포함한다. 가변성 명세표의 가변점은 C&V 분석 단계에서 식별된 모든 가변점을 포함하고[9], 가변치는 핵심 자산을 이용하여 어플리케이션을 개발할 때 패밀리 멤버들이 핵심 자산의 가변점에 설정할 수 있는 예상 가능한 모든 가변치들을 명세한 것으로 이 정보는 핵심 자산을 설계할 때 더 상세하게 표현될 수 있다. 이 관계는 가변성 명세표의 가변점과 의사 결정 명세의 가변점 사이의 관계를 나타내는 추적 항목 2.12와 가변성 명세표의 가변치와 의사 결정 명세의 가변치 사이의 관계를 나타내는 추적 항목 2.13이 나타낸다. 한 가변점에 가변치를 설정하면 여러 부수적인 결과가 나타날 수 있고, 이는 의사 결정 모델의 영향에 표현되며, 이 관계는 가변성 명세표의 가변치와 의사 결정 명세의 영향 사이의 관계를 나타내는 추적 항목 2.16이 나타낸다.

이 외에 핵심 자산 내의 요소간의 관계를 나타내는 추적 항목 2.2, 2.5, 2.9가 있다. 한 산출물 요소 내의 추적 항목은 다른 산출물의 요소간 추적을 하면 묵시적으로 포함되어 추적되는 항목으로, 다른 산출물 요소 내의 추적 항목과는 달리 산출물 요소간의 일관성을 맞추기 위한 것이다.

5.2 매핑 지침

이번 단계에서는 5.1에서 정의된 추적 링크 1과 추적 링크 2에 포함되는 각 추적 항목을 실제 PLE 프로세스에 적용하기 위한 지침을 제시한다.

표 1은 제품 계열 범위와 C&V 명세의 산출물 관계를 나타내는 추적 링크 1의 여러 추적 항목들에 관한 매핑 지침을 나타낸다. '추적 항목#'에는 그림 5에 표시된 산출물의 요소간의 관계를 나타내는 추적 항목 번호를 나열한 것이고, '매핑 지침'은 산출물의 여러 요소들의 추적 관계를 매핑하는데 필요한 지침을 나타낸 것이다.

표 2는 C&V 명세와 핵심 자산의 산출물 관계를 나타내는 추적 링크 2의 여러 추적 항목들을 매핑하는데 필요한 지침을 나타낸다.

6. 추적성의 적용

이 장에서는 4장과 5장에서 제시한 PLE의 주요 산출물, 산출물의 메타모델, 그리고 산출물간의 추적 링크를 표현한 추적성 맵의 적용과 핵심 자산 개발에 어떻게 이용될 수 있는지에 대해 살펴본다.

그림 7은 대여 관리 도메인의 결제(Payment) 기능에 대해 추적성 맵을 이용하여 각 산출물간의 추적 관계를 나타낸 것이다.

대여 도메인에서는 결제 기능을 공통적으로 가지고 있지만 어플리케이션에 따라 다양한 결제 방법을 제공하며, 제품에 특정한 요구사항에 도메인마다 제공하는 결제 방법을 명시한다. 결제 기능은 C&V 분석을 통해 유즈 케이스 모델의 유즈 케이스와 휘처 모델의 기능적인 휘처로 표현되고(추적 항목 1.1, 1.8), 여러 결제 방법은 현금 결제, 카드 결제, 계좌 이체의 대안 유즈 케이스와 대안 기능적인 휘처로 표현되며(추적 항목 1.3,

표 1 추적 링크 1의 매핑 지침

추적 항목#	매핑 지침
1.1	공통 요구사항에 나온 기능적 요구사항은 유즈 케이스 다이어그램의 하나 이상의 유즈 케이스로 설계된다.
1.2	가변성 있는 유즈 케이스는 가변점 정보를 포함하고, 이는 가변성 명세표의 하나 이상의 가변점으로 모두 명시되어야 한다.
1.3	유즈 케이스 다이어그램의 선택 또는 대안 유즈 케이스는 제품에 특정한 요구사항의 기능적 요구사항에서 유도된다.
1.4	공통 요구사항의 기능적 요구사항은 유즈 케이스 명세의 유즈 케이스에 대한 설명(이름, 개략 설명, 흐름)으로 명시되고, 비기능적 요구사항은 사전 조건, 사후 조건, 제약 사항으로 표현된다.
1.5	선택 또는 대안 기능적 요구사항에 대한 설명은 유즈 케이스 명세에 명시된다.
1.6	유즈 케이스 명세에 식별된 가변점은 모두 가변성 명세표의 가변점으로 표현된다..
1.7	유즈 케이스에 명시된 모든 가변치 정보는 가변성 명세표의 가변치에 명시되어야 한다.
1.8	공통 요구사항의 기능적 요구사항은 휘처 모델의 하나 이상의 휘처로 설계된다.
1.9	휘처 모델에 표현된 기능적인 휘처의 가변점 정보는 모두 가변성 명세표의 가변점에 명시된다.
1.10	휘처 모델의 선택 또는 대안 휘처는 제품에 특정한 요구사항에서 유도된다.
1.11	공통 요구사항의 비기능적 요구사항에 명시된 항목은 휘처 모델의 비기능적인 휘처로 설계된다.
1.12	제품에 특정한 요구사항 중 비기능적 요구사항은 휘처 모델의 비기능적 휘처로 설계된다.
1.13	공통 요구사항의 명세된 한 시스템의 기능이 여러 휘처로 설계되었다면, 공통 휘처들 간에는 관계가 존재한다.
1.14	제품에 특정한 요구사항의 명시된 한 기능이 여러 휘처로 설계된다면, 제품에 특정한 휘처들 간에도 관계가 존재한다.
1.15	유즈 케이스에 명시된 워크플로우 가변성이 휘처 모델의 관계로 표현되며, 이는 가변성 명세표의 가변점으로 표현된다.
1.16	공통 요구사항은 가변성을 일으키는 가변점을 포함하고 있고, 이는 가변성 명세표의 가변점으로 모두 표현된다. 한 공통 요구사항이 가지는 가변점은 가변성 명세표의 하나 이상의 가변점으로 매핑될 수 있다.
1.17	가변성을 가진 요구사항에 대한 멤버마다 다른 가변치는 모두 제품에 특정한 요구사항에서 유도된다.

표 2 추적 링크 2의 매핑 지침

추적 항목#	매핑 지침
2.1	메시지 호출관계가 잦고 서로 연관 관계가 높은 유즈 케이스들은 하나의 컴포넌트로 표현된다.
2.2	모듈 뷰타입의 적절한 스타일을 선택함으로써 결정된 제품 계열 아키텍처의 정적인 구조는 컴포넌트 간의 관계로 매핑될 수 있다 [13].
2.3	유즈 케이스 명세에 명시된 다른 컴포넌트에 속하는 유즈 케이스를 호출할 수 있는 오퍼레이션는 컴포넌트간 관계로 표현된다.
2.4	휘처 모델의 비기능적 요구사항은 범용 아키텍처의 비기능적 설계에 반영된다.
2.5	한 컴포넌트에 속하는 연관관계가 있는 클래스들은 클래스 명세에 포함된다.
2.6	유즈 케이스 명세에 명시적 또는 암시적으로 명시된 객체, 속성, 오퍼레이션은 클래스 명세의 클래스, 클래스의 속성, 오퍼레이션으로 상세화되고 설계된다. 유즈 케이스의 명시된 객체, 속성, 오퍼레이션이 모두 클래스 명세로 추적이 가능한 반면, 클래스 명세에 표현된 모든 클래스, 속성, 오퍼레이션이 유즈 케이스 명세로 추적되는 것은 아니다.
2.7	기능적인 휘처에 명시된 객체와 오퍼레이션들은 클래스로 정제되고 설계된다.
2.8	유즈 케이스에 명시된 워크플로우는 클래스간 관계 명세에 명시된 클래스간 메시지 호출 관계로 표현된다.
2.9	한 컴포넌트 내의 어떤 기능은 다른 컴포넌트의 메소드를 호출하여 고유한 기능을 수행할 수 있다. 이러한 기능은 클래스 간의 의존관계로 표현될 수 있고, 이는 각 클래스를 포함하는 컴포넌트 간의 의존관계로 확장될 수 있다.
2.10	유즈 케이스 명세에 명시된 액터와 시스템의 상호작용은 메소드 시그네처를 유도한다.
2.11	유즈 케이스 명세에 정의된 사전 조건, 사후 조건, 제약 사항은 메소드 시맨틱 내에 표현된다.
2.12	C&V 명세안에 식별된 가변점은 의사 결정 모델의 하나 또는 하나 이상의 가변점으로 분류되어 상세하게 표현된다.
2.13	C&V 명세에서 식별된 가변치는 의사 결정 모델의 가변치로 매핑되고, 가변치 사이의 매핑관계는 가변치와 관련된 가변점이 의사 결정 명세에 어떻게 매핑되었는지에 크게 좌우된다.
2.14	한 유즈 케이스의 가변성이 이와 관련이 있는 유즈 케이스의 다른 가변성을 유도할 수 있고, 한 유즈 케이스의 가변성에 가변치를 설정함으로써 다른 유즈 케이스에 영향을 미치는 경우, 의사 결정 명세의 영향에 명시되어야 한다.
2.15	한 휘처의 가변성이 이와 관련이 있는 휘처의 다른 가변성을 유도할 수 있고, 한 휘처의 가변성에 가변치를 설정함으로써 다른 휘처에 영향을 미치는 경우, 의사 결정 명세의 영향에 명시되어야 한다.
2.16	의사 결정 명세의 영향은 가변점에 가변치를 지정함으로써 생길 수 있는 부수적인 결과를 표현한 것이다. C&V 명세의 각 가변치는 핵심 자산에 영향을 미치므로 의사 결정 명세의 몇몇 영향으로 매핑된다.

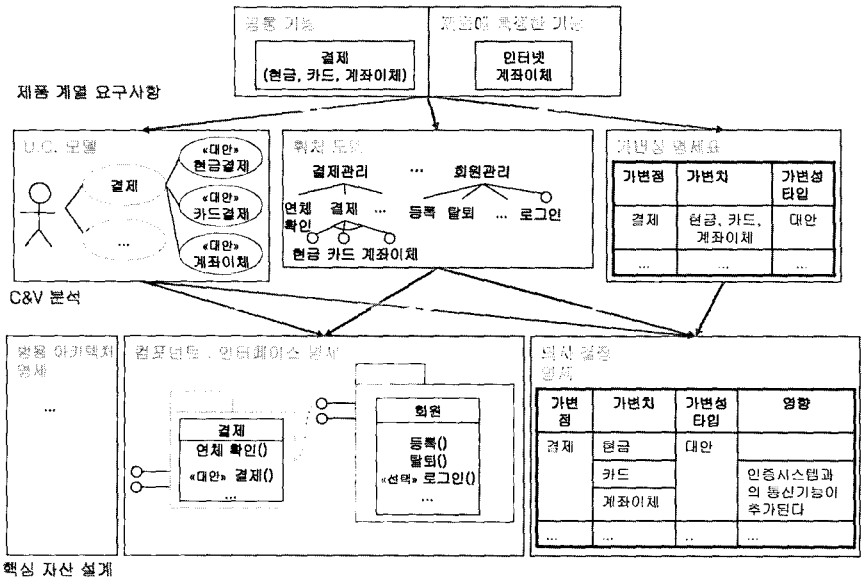

그림 7 대여 관리 시스템의 결제 기능에 대한 예시

1.10), 가변성을 가진 기능이므로 가변성 명세표에 가변점과 가변치에도 명시된다(추적항목 1.14, 1.15). C&V 분석 결과를 통해 핵심 자산을 설계하는데 결제 기능은 공통성과 가변성 정보를 모두 표시하여 클래스 명세를 포함한 컴포넌트 명세와 인터페이스 명세에 나타나고(추적 항목 2.1, 2.3, 2.6, 2.7, 2.10, 2.11), 가변성 명세표에 표기된 결제 기능과 관련된 가변성 정보는 의사 결정 명세에 더 자세하게 명시된다(추적 항목 2.12, 2.13, 2.14, 2.15, 2.16).

본 논문에서 제시한 추적성 맵과 지침은 다음과 같은 방법으로 적용될 수 있다. 핵심 자산 개발 초기에 식별된 가변성은 개발 단계를 거치면서 여러 가변성으로 세분화되어 여러 영향을 미치게 되므로 효과적으로 가변성을 관리할 수 있는 체계적인 방법이 필요하다[16]. 이 논문에서 제시된 메타모델과 추적성 맵은 가변성을 효율적으로 관리하는데 사용될 수 있다. 추적성 맵에서 정의한 추적 링크, 추적 항목, 매핑 지침을 이용하여 핵심 자산의 오류를 발견하는데 사용되어 고품질의 산출물을 생산할 수 있으며, 어플리케이션의 유지보수를 효율적으로 할 수 있다. 메타모델과 추적성 맵을 이용하여 명확하게 표현된 산출물간의 관계로 개발자가 실용적으로 어플리케이션을 개발할 수 있다. 산출물과 산출물들의 메타 모델을 고려하여 실용적인 핵심 자산 개발 프로세스를 정의하는데 기초로서 사용될 수 있고, 산출물의 메타모델과 추적성 맵을 보강하여 다양한 PLE 프로세스의 프레임웍 공학 프로세스를 평가하는 기준으로서 사용될 수 있다.

7. 결 론

소프트웨어 개발에서 다양한 산출물들의 품질은 최종 소프트웨어의 품질과 유지보수성을 결정하며, 중간 산출물들의 품질은 관련 산출물간의 일관성과 추적성에 의해 크게 결정된다. 본 논문에서는 대표적인 PLE 기법들에게 범용적으로 적용될 수 있는 산출물 종류를 열거하고, 각 산출물에 대한 구체적인 구성 요소를 메타 모델로 정의, 제시하였다. 산출물의 메타 모델을 정의함으로써 산출물간의 일관성과 추적성의 정의가 가능해진다. 본 연구에서는 PLE 산출물간의 매핑 관계를 추적성 맵으로 정의하였고, 추적 가능한 추적 링크에 대한 적용 지침과 방법을 제시하였다.

본 연구 결과는 PLE 프로젝트를 수행 시 적용되어 PLE 프로세스의 각 활동들에서 도출될 산출물을 명확히 정의하고, 추적 링크를 적용하여 산출물의 도출 및 검증에 직접 활용될 수 있고, 나아가 PLE 프로세스를 실용적으로 정의하는데 도움이 될 수 있다. 또한, 메타 모델을 기반으로 PLE CASE 도구의 개발과 모델 변환 등 개발 자동화의 연구에 기초로 활용될 수 있다.

참 고 문 헌

[1] Clements, P. and Northrop, L., Software Product Lines: Practices and Patterns, Addison Wesley, pp. 563, Aug. 2001.

[2] Muthig, D. and Atkinson, C.: "Model-Driven Product Line Architectures," Lecture Notes in

Computer Science 2379, Proceedings of the 2ND Software Product Line Conference, 2002.
[3] Bayer, J., and Widen, T., "Introducing Traceability to Product Lines," Lecture Note in Computer Science 2290, Proceedings of the PFE-4 2001, 2002.
[4] Atkinson, C., et al., Component-based Product Line Engineering with UML, Addison Wesley, 2001.
[5] Kang, K., Kim, S., Lee, J., Kim, K., Shin, E. and Huh, M., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," Annals of Software Engineering, vol.5, p.143- p.168, 1998.
[6] Kang, K., Lee, K., Lee, J., and Kim, S., "Feature-Oriented Product Line Software Engineering: Principles and Guidelines," Domain Oriented Systems Development: Practices and Perspectives, Taylor & Francis, page 29-46, 2003.
[7] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J., "PuLSE: A Methodology to Develop Software Product Lines," Proceeding of symposium for Software Reusability '99, ACM, 1999.
[8] Choi, S., et al., "A Systematic Methodology for Developing Component Frameworks," Lecture Notes in Computer Science 2984, Proceedings of the 7th Fundamental Approaches to Software Engineering Conference, 2004.
[9] 김수동, 소동섭, 신석규, "컴포넌트 가변성 유형 및 Scope에 대한 정형적 모델", 한국정보과학회논문지 소프트웨어 및 응용, Vol. 30, No. 05, pp. 414-429, 2003년 6월.
[10] Clements, P., et al., Documenting Software Architectures Views and Beyond, Addison Wesley, 2003.
[11] Thiel, S., and Hein, A., "Systematic Integration of Variability into Product Line Architecture Design," proceeding of SPLC2, LNCS 2379, Springer-Verlag Berlin Heidelberg, 2002.
[12] Bass, L., et al., Software Architecture in Practice, Addison Wesley, 2003.
[13] Clements, P., et al., Evaluating Software Architecture, Addison Wesley, 2002.
[14] Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004.
[15] Bass, L., Clements, P, and Kazman, R., Software Architecture in Practice, Addison Wesley, 2003.
[16] M. Becker, "Towards a General Model of Variability in Product Families," Proceedings of the Software Variability Management Workshop, 2003.

라 현 정
정보과학회 논문지 : 소프트웨어 및 응용
제 32 권 제 3 호 참조

장 수 호
정보과학회 논문지 : 소프트웨어 및 응용
제 32 권 제 3 호 참조

김 수 동
정보과학회 논문지 : 소프트웨어 및 응용
제 32 권 제 3 호 참조