제품 라인에서 컴포넌트 구조를 활용한 컴포넌트 스펙 방법

(Approach to Specify a Component using Component Structure in Product Lines)

조 혜 경 †

(Hye-Kyung Cho)

요 약 제품 라인은 재사용을 위한 연구 방법으로 널리 인식되어 왔다. 제품 라인에서 대표적인 중요 자산은 소프트웨어 컴포넌트이다. 그러나, 제품 라인에 대한 많은 관심에 비해 제품 라인에서 컴포넌트 구조 및 스펙에 대한 연구는 아직 미흡하다. 본 논문은 제품 라인에서 가변성(variability)을 반영한 컴포넌트 구조와 컴포넌트 스펙 방법을 제시한다. 본 논문은 FORM(Feature-Oriented Reuse Method)을 기반으로 제품 라인 컴포넌트의 정적 및 동적 구조, 제품 라인 컴포넌트의 행동 및 동시성 정보를 기술한다. 제품 라인 컴포넌트 스펙에 대한 각 정보는 블랙박스(black-box)와 화이트박스(white-box) 형태로 구분되어 기술되며 각 스펙 정보는 BNF로 정형화된다. 그 스펙들이 제품 라인 컴포넌트의 많은 서로 다른 특징의 충분한 고려를 통해 기술되기 때문에 본 논문은 제품 라인에서 컴포넌트의 손쉬운 개발을 돕고 제품 라인 공학 방법론의 적용 방법을 잘 이해하도록 돕는다.

키워드 : 제품 라인, 컴포넌트 스펙, 재사용, 방법론

Abstract Product line is nowadays well known as a representative method for reuse. In the product line, important assets are components. Although enough concerns were given of the product line, it was not accomplished to structure and specify a product-line component with variability. This paper presents an approach to specify components in the product line. The approach describes the static and dynamic structure of a product-line component and explains the behavior and concurrency of the component. The component information is separately described in the black-box and white-box using the Feature-Oriented Reuse Method(FORM). This research also formalizes the data on a component specification in the form of BNF. The specification is described through careful consideration for many different characteristics of the product-line component, so this paper helps to easily develop the components in the product line and to well comprehend how to apply a method for the product line.

Key words : Product Line, Component Specification, Reuse, Methodology

1. 서 론

소프트웨어 위기 극복의 방법으로 컴포넌트와 프레임워크의 재사용 기법이 널리 인식되어 왔다. 현재, 이러한 기법들은 소프트웨어 생명주기에서 재사용을 강조한 제품 라인(product line) 공학 방법론과 결합하여 활발히 연구되고 있다. 제품 라인 공학 방법론은 도메인 공학(domain engineering)과 응용 공학(application engineering) 과정으로 나뉘어 진다[1]. 제품 라인 공학 방법론은 도메인 공학 과정에서 제품들의 공통점과 차이점을 식별하여 자산들(assets)을 개발한 다음에 응용 공학 과정에서 목표로 하는 제품을 만들기 위해 미리 개발된 자산들을 활용한다.

CMU/SEI(Software Engineering Institute at Carnegie Mellon University)는 제품 라인 전문가들을 대상으로 한 설문 조사를 통해 소프트웨어 아키텍처와 코드가 핵심 자산으로 중요함을 발표하였다[2]. 재사용 가능한 컴포넌트는 아키텍처로 부터 유도되고 구현 단계에서 코드로 기술된다. 제품 라인 공학 방법론은 해결방법(solution)이 아니라 문제(problem)를 기반으로 한 재

† 정 회 원 : 포항공과대학교 컴퓨터공학과 연구원
hg4232@empal.com
논문접수 : 2005년 7월 21일
심사완료 : 2006년 1월 5일

사용 기법이기 때문에 제품 라인에서 컴포넌트는 기존 CBD(component-based development) 방법론에서 보다 재사용성이 더욱 강조된다. 본 논문에서 사용되는 컴포넌트 정의는 Szyperski와 Catalysis에 따른다.

풍부한 재사용을 위해 제품 라인 컴포넌트에 관한 정보는 제품 라인의 특성을 반영하여 기술되어야 하나 제품 라인 컴포넌트에 대한 명확한 스펙 방법이 현재 존재하지 않는다. 즉, 기존의 관련 연구들은 제품 라인 컴포넌트의 구조를 정의하지 않았고 제품 라인 컴포넌트의 구조적 특성을 반영하여 가변성(variability)을 정의한 후에 다양한 관점에서 제품 라인에 적합한 컴포넌트 스펙 연구를 수행하지 않았다.

본 논문은 FORM(Feature-Oriented Reuse Method)을 기반으로 제품 라인에서 가변성을 고려한 컴포넌트 구조와 컴포넌트 스펙 방법을 제시하여 제품 라인 컴포넌트의 스펙을 작성한다. 그 스펙은 제품 라인 컴포넌트의 정적 및 동적 구조와 행동(behavior) 및 동시성(concurrency) 정보를 제공한다. 또한, 그 제품 라인 컴포넌트 스펙 정보는 블랙박스(black-box)와 화이트박스(white-box) 형태로 구분되며 각 정보에 정의 및 각 정보들 간의 관계는 BNF로 정형화된다.

본 논문의 구성은 다음과 같다. 2장은 연구배경으로 컴포넌트 스펙의 정의 및 관련 연구들을 살펴본다. 3장은 스펙 방법 및 절차와 FORM에서 제품 라인 컴포넌트 구조가 제시된다. 4장은 스펙의 분류 기법이 제시되며 연구결과로 얻어진 제품 라인 컴포넌트의 스펙 정보와 스펙 정형화가 설명된다. 5장은 사례 연구인 제품 라인 전자상거래 시스템에 연구의 적용과 연구 결과의 평가에 대해 6장은 결론 및 향후 연구에 대해 기술한다.

2. 연구 배경

2.1 컴포넌트 스펙의 정의

CBD 방법론인 Catalysis[3]는 컴포넌트 스펙(specification)을 "외부에 보여 지는 행동을 획득하는 시스템이나 컴포넌트의 경계(boundary)"라 정의하고 있다. 제품 라인 공학 방법론인 KobrA[4]는 컴포넌트 스펙을 "컴포넌트 행동에 대한 정보, 컴포넌트가 제공하는 오퍼레이션에 대한 논리적인 효과, 주변 컴포넌트에 대한 기대 정보를 포함하는 개념"으로 정의하고 있다.

컴포넌트 스펙은 임의의 소프트웨어 개발 방법론을 적용한 CASE 도구에서 컴포넌트 설계로부터 컴포넌트 코드를 자동 생성하기 위해 주로 이용된다. 이 예로 스털링사의 Cool:Spec과 Cool:Gen이 있다. Cool:Spec은 컴포넌트 스펙 작성을 위해 Cool:Gen은 앞서 작성된 스펙에서 컴포넌트 코드의 자동 생성을 위해 사용되는 도구이다[5]. 또한, 컴포넌트 스펙은 정보 저장소에 저장된 컴포넌트들 중에서 대체에 적합한 컴포넌트를 발견하기 위해 컴포넌트의 스펙 매칭시에 활용되기도 한다.

현재, 많은 사람들은 컴포넌트 스펙을 인터페이스 오퍼레이션의 선행조건, 선후조건, 불변조건에 대한 OCL(Object Constraint Language)을 활용한 기술로 단순히 인식하고 있다. 컴포넌트 스펙에는 컴포넌트의 다양한 정보가 포함되어야 하기 때문에 컴포넌트 스펙을 오퍼레이션에 대한 OCL적 표현으로만 이해하는 것은 잘못된 것이다. Dias[6]는 컴포넌트 스펙에 컴포넌트의 정적 구조 및 행동과 컴포넌트 동적 행동이 기술되어야 한다고 주장하였다. 본 논문은 컴포넌트의 스펙을 제품 라인에서 컴포넌트 구현 및 실행에 관한 컴포넌트 내·외부 모든 정보의 기술을 포함하는 개념으로 정의한다.

2.2 관련 연구

컴포넌트 스펙의 관련 연구들은 CBD의 컴포넌트 스펙 연구와 제품 라인의 컴포넌트 스펙 연구로 분류된다.

2.2.1 CBD에서 컴포넌트 스펙 연구

Beugard[7], Peter[8], Albani[9]와 Bachmann[10]는 CBD에서 컴포넌트 스펙에 관한 연구이다. Peter와 Albani는 Beugard를 확장한 연구이며 Bachmann은 Beugard를 다른 관점에서 기술하고 있다. 각 관련 연구의 상세한 내용은 다음과 같다.

Beugard는 컴포넌트 스펙을 위해 계약(contracts)을 활용하였다. 계약 개념은 Eiffel 언어를 이용하여 Meyer가 제안한 것으로 계약이란 컴포넌트가 제공해야 하는 서비스에 대한 컴포넌트 클라이언트와의 약속이다. Beugard는 컴포넌트에 대한 계약을 구문(syntactic), 행동(behavioral), 동기화(synchronization), 서비스 품질(quality-of-service)로 나누어 기술하였다.

Peter는 Beugard의 연구를 확장하여 인터페이스, 행동, 상호작용, 품질, 용어(terminology), 임무(task), 마케팅으로 나누어 컴포넌트 스펙을 작성하였다. 이 때, 임무는 비즈니스 컴포넌트에 의해 지원되는 비즈니스 태스크를 기술한 것이고 마케팅은 컴포넌트 거래를 위해 필요한 컴포넌트 정보를 기술한 것이다.

CMU/SEI의 Bachmann는 기능적 관점과 비 기능적 관점에서 컴포넌트 스펙을 기술하였다. 기능적인 관점에서 컴포넌트 스펙은 컴포넌트가 제공하는 서비스를 프로그래밍 언어의 API 형태 또는 OMG의 IDL 형태로 기술한 것이다. 비 기능적 관점에서 컴포넌트 스펙은 기능적인 관점에서 스펙을 보완하기 위해 컴포넌트의 행동과 동기화 정보를 기술한 것이다.

위의 연구들에서 Beugard는 여러 관점에서 컴포넌트 정보를 기술한 최초의 시도로서 현재 CBD에서 컴포넌트 스펙을 위한 표준으로 인정받고 있다. 그러나,

Beugard는 구문, 행동, 동기화, 서비스 품질 측면에서 컴포넌트 스펙 내용이 무엇이고 스펙 방법 및 절차는 무엇인지 상세히 정의하지 않았다.

2.2.2 제품 라인에서 컴포넌트 스펙 연구

VTT는 Ana[11]과 Anb[12]에 제품 라인에서 컴포넌트 스펙 방법을 발표하였다. VTT의 연구는 Bosch와 Northrop의 연구를 기초로 한다. VTT는 제품 라인에서 컴포넌트 스펙을 컴포넌트에 대한 기본 정보, 상세 정보, 인증 정보, 지원 정보로 구분하여 작성하였다. 기본 정보는 컴포넌트 인터페이스와 기능에 대해 기술하고 상세정보는 컴포넌트 개발 환경, 컴포넌트들의 상호 의존성, 컴포넌트 구현 정보들을 기술한다.

VTT 연구의 분석 결과 파악된 문제점은 다음과 같다. 1) 제품 라인 컴포넌트 스펙에 휘처와 컴포넌트 사이의 관계가 언급되지 않았다. 2) 제품 라인의 가변성 정보가 다루어지지 않았다. 3) 컴포넌트 스펙 방법이 체계적이지 않다. 4) 컴포넌트 사용법과 테스팅 품질과 같은 컴포넌트의 외부적인 성질에 컴포넌트 스펙이 집중되었기 때문에 컴포넌트의 상세한 내부 정보를 얻을 수 없다.

2.2.3 제품 라인 컴포넌트 스펙의 기타 관련 연구

다른 관련 연구들에는 컴포넌트 모델과 가변성 구현 기술이 있다. 대표적인 컴포넌트 모델은 EJB(Enterprise JavaBeans), CCM(Common object request broker Component Model), COM+, Koala[13]가 있다. Koala는 제품 라인 컴포넌트 모델로서 가전제품 소프트웨어 개발을 위해 필립스가 발표한 컴포넌트 플랫폼이며 Koala는 컴포넌트 구현 언어와 구현 환경도 의미한다. EJB, CCM, COM+, Koala 컴포넌트 모델을 모두 분석하여 위의 모든 컴포넌트 플랫폼과 컴포넌트 구현 기술에 공통적이면서 특정 컴포넌트 모델에 의존하지 않는 제품 라인 컴포넌트 구조가 논문에서 제시된다.

기타 관련 연구로 제품 라인 컴포넌트의 가변성 구현 기술이 있다. 이 연구는 Sharp[14]와 Anastasopoulos[15]가 대표적이다. 가변성 구현 기술로 Sharp는 집합, 상속, 템플릿, 조건적 컴파일, 대체(substitution)를 제시하였으며 Anastasopoulos는 집합/위임, 상속, 매개변수화, 오버로딩, 동적 클래스 적재, 정적 링크 라이브러리, 동적 링크 라이브러리, 조건적 컴파일, 프레임, 반사(reflection), 관점 지향 프로그래밍(aspect-oriented programming), 디자인 패턴을 제시하였다. 위의 여러 가변성 구현 기술을 분석하여 그림 2에 제시된 제품 라인 컴포넌트 구조에 가변성이 발생할 때 각 가변성 구현 기술에 따라 가변성 발생 위치의 예측과 연관된 가변성 정보를 파악하였다.

3. 제품 라인 컴포넌트의 스펙 방법

3.1 제품 라인 컴포넌트의 스펙 절차

제품 라인 컴포넌트에 대한 스펙은 아래의 절차로 수행된다. 아래의 절차는 3장과 4장에서 논문 전개 흐름이 된다. 즉, 3장과 4장에 기술되는 모든 내용은 ①~⑦과정에서 얻어진 것이다. FORM은 계속 발전하고 있으며 현재 완성된 단계가 아니기 때문에 스펙작성을 위해 ①~⑦과 같은 여러 작업이 수행되었다.

① FORM 제품 라인 공학 방법론의 컴포넌트 생명주기를 작성한다.
② FORM 기반 컴포넌트 생명주기에서 컴포넌트 식별 및 컴포넌트 스펙과 관련된 이해관계자(stakeholder)들을 발견한다.
③ 파악된 이해관계자들의 각자 책임(responsibility)과 역할(role)을 CBD 방법론과 다른 제품 라인 공학 방법론을 활용하여 정의한다.
④ 각 이해관계자들의 역할에 대한 결과물(즉, 산출물)을 기초로 제품 라인 컴포넌트의 스펙 정보를 생성한다.
⑤ 찾아진 정보들이 일반 컴포넌트와 제품 라인 컴포넌트의 주요 특성을 모두 반영하고 있는지 관련 연구를 참고하여 검토한다.
⑥ 찾아진 정보들을 제시된 제품 라인 컴포넌트 구조에 배치하여 각 정보 및 각 정보들 간의 관계를 조사한다.
⑦ ①~⑥ 과정을 거쳐 찾아진 제품 라인 컴포넌트 스펙 자료들을 BNF 표기법으로 정형화한다.

FORM의 현재까지 상태를 간략히 기술하면 다음과 같다. FORM은 방법론의 구성(도메인 공학과 응용 공학 단계), 도메인 모델(휘처 모델) 작성, 휘처 바인딩 타임, 마케팅 및 제품 계획 개념, 그리고 몇 가지의 적용 사례(전자게시판, 엘리베이터, 전화교환기) 연구로 요약될 수 있다. 즉, FORM에서는 컴포넌트 생명주기, 각 단계의 명확한 이해관계자 및 산출물, 각 이해관계자들의 역할 및 책임, 제품 라인 컴포넌트 구조, 제품 라인 컴포넌트 스펙에 대한 연구가 아직 수행되지 않았다.

3.2 제품 라인 컴포넌트의 생명주기

그림 1은 FORM 방법론에서 컴포넌트의 생명주기, 컴포넌트 생명주기에서 각 단계, 각 단계와 관련된 이해관계자 및 산출물들을 보여준다. 그림 1에 타원은 컴포넌트 생명 주기에서 컴포넌트 식별 및 스펙 단계를 나타낸다. 이 두 단계에서 FORM 이해관계자들의 역할 정의를 통해 상세한 제품 라인 컴포넌트의 스펙 내용을 파악할 수 있었다. 그 이해관계자들은 휘처 모델러, 제품 라인 요구 명세서 작성자, 제품 라인 설계 객체 모델러, 제품 라인 아키텍트, 가변성 관리자이다. 그 밖에 제품 라인 컴포넌트 스펙과 관련된 이해관계자로 도메인 전문가, 제품 라인 컴포넌트 구현자가 있다.

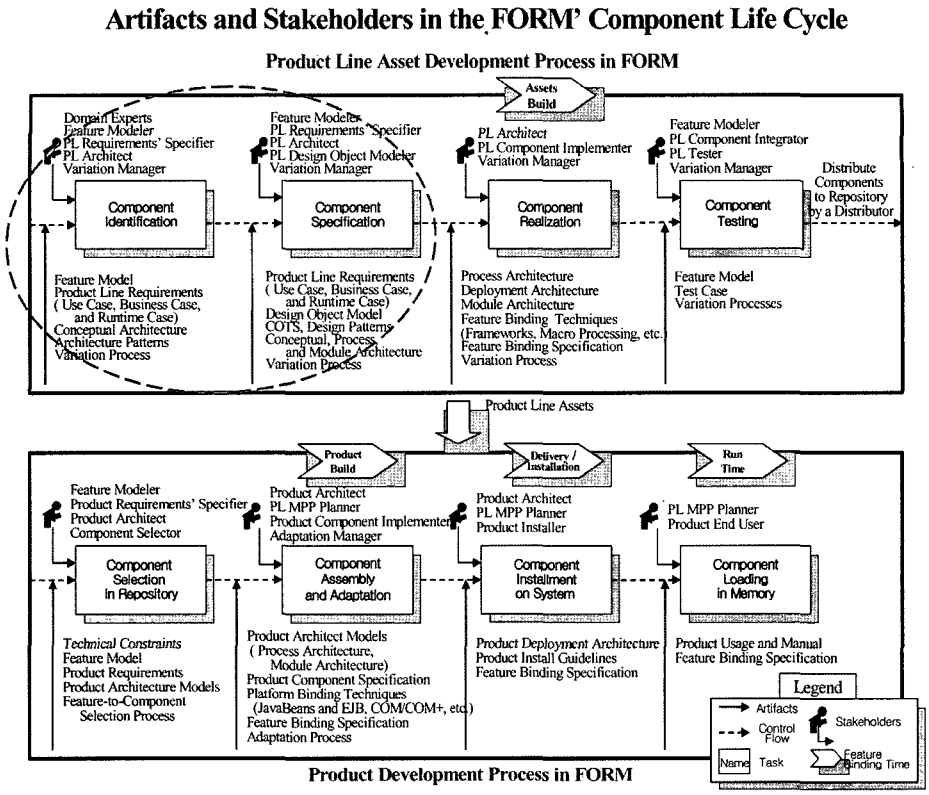

그림 1 FORM 재사용 방법론에서 컴포넌트 생명주기

3.3 제품 라인 컴포넌트 스펙 관련 이해관계자 역할

제품 라인 컴포넌트 스펙에 가장 중요한 역할을 수행하는 이해관계자들은 제품 라인 설계 객체 모델러와 아키텍트이다. 이들에 역할 정의는 각자의 책임에 있는 산출물들을 명확히 해주기 때문에 컴포넌트 스펙 정보 획득의 주요 시작점이 된다.

표 1에서 제품 라인 설계 객체 모델러는 컴포넌트의 내부 구조 설계와 컴포넌트 내부에서 발생하는 가변성 표현에 관한 역할을 담당한다. 즉, 제품 라인 설계 객체 모델러는 각 컴포넌트 내부에 존재하는 객체의 정적 구조(object, data, function), 객체들 간의 호출 관계(relationship), 객체 기능(function)에 대한 실행 조건으

표 1 Product Line Design Object Modeler

No.	Design Object Modeler's Responsibility and Role
1	Find Candidate Objects from Feature Model
2	Find Relationship between Objects from Feature Model
3	Refine the Candidate Objects with Requirements' Model
4	Refine the Objects Relationship with Requirements' Model
5	Find Object Elements (Data and Function) from Feature Model
6	Find Object Elements (Data and Function) Using Requirements' Model
7	Build a Conceptual Design Object Model
8	Confirm Variability in Feature Model to the Conceptual Design Object Model
9	Coordinate the Conceptual Design Object Model with Feature Binding Unit.
10	Find Design Patterns
11	Refine the Conceptual Design Object Model with Design Patterns
12	Define Concrete Design Object Model

로 객체의 상태(state)를 파악하고 이러한 정보에 가변성을 표현한다.

표 2에서 제품 라인 아키텍트는 컴포넌트의 외부 구조 설계와 컴포넌트 외부에서 발생하는 가변성 표현에 관한 역할을 담당한다. 즉, 제품 라인 아키텍트는 각 컴포넌트 외부의 정적 구조(component interface and operation), 컴포넌트들 간의 호출 관계, 컴포넌트 인터페이스 오퍼레이션에 대한 실행 조건으로 컴포넌트의 상태 파악, 컴포넌트들 간의 동시성 정보, 그리고 이러한 정보들에 가변성을 기술한다.

제품 라인 컴포넌트 스펙에 관한 기타 이해관계자들의 역할은 다음과 같다. 휘처 모델러는 휘처들의 관계를 이용하여 휘처 바인딩 유닛(units)을 정의하여 컴포넌트 식별의 초기 작업을 수행한다. 제품 라인 요구 명세서 작성자는 사용 사례(use case) 시나리오를 작성하여 컴포넌트 내부의 스펙 정보를 추가하고 사용 사례 모델을 이용하여 컴포넌트 식별 작업을 돕는다. 가변성 관리자는 생명주기 각 단계의 산출물에 적합한 가변성 정의 절차를 수립한 다음에 휘처 모델의 임의의 가변성이 생명주기의 여러 단계를 거쳐 변형되고 세분화되는 것을 추적한다. 도메인 전문가는 도메인 지식을 이용하여 컴포넌트 식별과 컴포넌트 스펙 결과가 올바른지 그리고 각 단계의 산출물에서 가변성 표현이 올바른지를 검증한다. 제품 라인 컴포넌트 구현자는 컴포넌트 식별 및 스펙 과정을 거친 컴포넌트에 디자인 패턴 또는 매개변수화와 같은 특정 가변성 구현 기술을 적용하여 제품 라인에서 컴포넌트 실체화(realization)를 담당한다.

3.4 제품 라인 컴포넌트의 구조

제품 라인 컴포넌트의 스펙 작성을 위해 제품 라인 컴포넌트의 구조가 먼저 정의되어야 한다. 특정한 어느 컴포넌트 모델에 의존하지 않는 제품 라인 컴포넌트 구조는 그림 2에 제시된다. 완전한 제품 라인 컴포넌트 구조 제시를 위해 EJB, CCM, COM+, Koala의 컴포넌트 모델들을 모두 분석하였다. 위 컴포넌트 모델들의 분석 결과 컴포넌트 개발 플랫폼에 따라 컴포넌트 구현 언어와 구현 기법이 서로 다르더라도 모든 컴포넌트 모델에서 컴포넌트 구조는 거의 유사함이 파악되었다.

또한, 제품 라인 컴포넌트 구조의 정의 과정에서 아래 항목들이 추가로 얻어져 작성되는 스펙에 반영하였다.

- 컴포넌트 인터페이스(provided interface)내의 오퍼레이션들은 하나의 특정 클래스(a core class)에 의해 구현되고 그 특정 클래스는 그 구현을 돕는 다른 클래스(helper classes)들을 필요로 한다.
- 네트워크에서 운영되는 컴포넌트들은 메모리에 세션(session) 유지 컴포넌트(transient)와 세션 결과를 데이타베이스에 기록하는 컴포넌트(persistent)로 구분된다. 이러한 컴포넌트의 기능 구분은 그림 2에서 특성(properties)의 컴포넌트 상태(status) 값에 기술된다.
- 제품 라인 컴포넌트의 가변성은 컴포넌트 외부와 컴포넌트 내부에서 발생하기 때문에 외부 가변성(external variation)과 내부 가변성(internal variation)의 발생 가능 위치가 그림 2에 표시된다.

표 2 Product Line Architect

No.	Product Line Architect's Responsibility and Role
1	Define Conceptual Architecture Components from Feature Binding Units
2	Define Data Flow of Conceptual Architecture Using Feature Spec. or Feature Binding Unit Spec.
3	Define Conceptual Architecture
4	Find Architectural Patterns
5	Find COTS(Commercial-Off-The-Shelf)
6	Refine Conceptual Architecture with Architectural Patterns and COTS
7	Confirm Variability in Feature Model to the Conceptual Architecture
8	Design Process and Protocol Connector for Networking
9	Define Threads in a Process
10	Find Shared Data and Message-Passing with Multi-Threads
11	Develop Process Architecture with Processes and Threads
12	Confirm Variability in Feature Model to the Process Architecture
13	Define Nodes with Processes
14	Design the Layout of Node Using H/W and S/W
15	Develop Deployment Architecture with Nodes
16	Confirm Variability in Feature Model to the Deployment Architecture
17	Divide Conceptual Architecture Components into Module Components Using Process and Deployment Architecture
18	Define Module' Interfaces with Coherent Data Exchanged between Conceptual Architecture Components in the Conceptual Architecture
19	Develop Module Architecture with Module Components and Interfaces
20	Confirm Variability in Feature Model to the Module Architecture

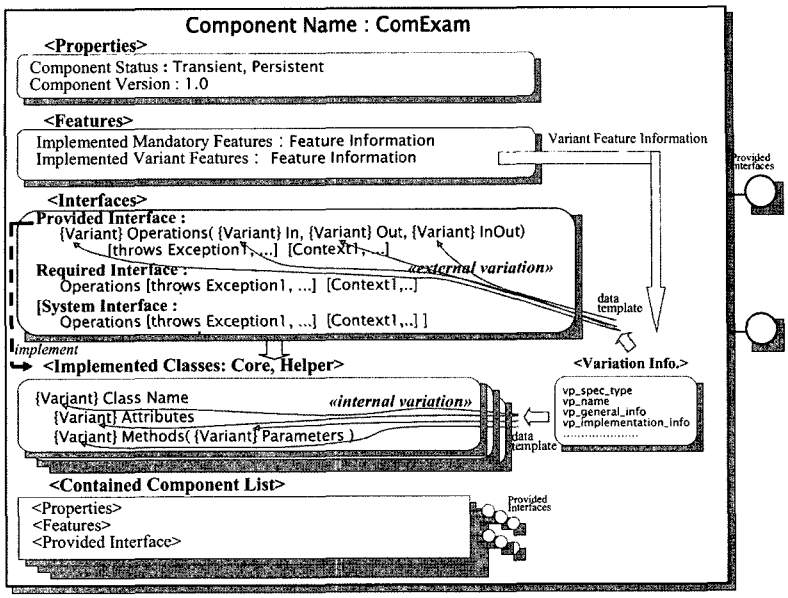

그림 2 제품 라인 컴포넌트의 구조

- 가변성 발생 위치에 상관없이 모든 가변점(variation points)들은 유사한 가변성 정보를 가지고 있기 때문에 각 위치에서 발생하는 가변성에 대해 가변성 정보(Variation Info.)가 일반화되어야 한다.
- 일반적으로 하나의 제품 라인 컴포넌트는 여러 개의 휘처들을 구현하고 있다[16]. 그래서, 현재 컴포넌트가 구현하고 있는 밀접히 관련된 휘처의 정보는 Mandatory와 Variant로 구분되어야 한다.

4. 제품 라인 컴포넌트의 스펙 작성

4.1 제품 라인 컴포넌트의 스펙 구분

그림 2에서 언급된 컴포넌트 외부 가변성과 컴포넌트 내부 가변성을 기초로 모든 제품 라인 컴포넌트 스펙은 표 3과 같이 분류된다. 이러한 컴포넌트 스펙 분류의 이유는 컴포넌트는 객체처럼 다양한 기능과 의미를 내포하고 있기 때문에 단지 하나의 문서로 컴포넌트의 모든 정보를 표현할 수 없기 때문이다. 제품 라인 컴포넌트 스펙은 블랙박스 형태로 EXCS1~EXCS4에 화이트박스 형태로 EXCS1~EXCS4와 INCS1~INCS4에 기술된다. 이때, 기술되는 제품 라인 컴포넌트의 정보는 컴포넌트의 정적 구조, 컴포넌트의 행동, 컴포넌트의 동적 호출관계, 컴포넌트의 동시성 정보이다. 표 3에서 컴포넌트 외부 가변성 정보는 EXCS1, EXCS2, EXCS3, EXCS4에 컴포넌트 내부 가변성 정보는 INCS1, INCS2, INCS3, INCS4에 기술된다. 그리하여, EXCS1~EXCS4로부터 INCS1~INCS4까지의 모든 스펙들은 하나의 제품 라인 컴포넌트에 대해 각각 고유한 정보를 제공하게 된다.

EXCS3와 INCS3를 먼저 설명하면 다음과 같다. EXCS3는 컴포넌트 외부 가변성 스펙으로서 이것은 컴포넌트들 간의 인터페이스 오퍼레이션의 호출 순서에 가변성 정보를 추가하여 시간적 흐름으로 컴포넌트 정보를 기술한다. INCS3는 컴포넌트 내부 가변성 스펙으로서 외부로부터 하나의 결정된 컴포넌트 인터페이스 오퍼레이션이 호출될 때 그 호출된 오퍼레이션을 실행하기 위해 현재 컴포넌트 내부 객체들 간의 메소드 호출 순서를 시간적 흐름으로 기술한다. 이때, 컴포넌트 내부에 객체들 간의 동적 호출시 가변성이 발생한다면 INCS3에 표시된다. 즉, EXCS3는 컴포넌트들 간의 (inter-component) 메시지 순서 챠트 형태로 INCS3는 컴포넌트 내의(intra-component) 객체 메시지 순서 챠트 형태로 기술되게 된다. 또한, INCS3는 객체지향에서 객체 메시지 순서 다이어그램과 유사하나 객체들의 상호작용을 유도하는 외부 자극(다른 컴포넌트가 현재 컴포넌트의 인터페이스 오퍼레이션 호출)이 존재한다는 것이 객체 메시지 순서 다이어그램과의 차이점이 된다.

표 3에서 각 스펙 정보는 이해관계자 역할의 결과인 여러 가지 산출물로부터 더욱 상세히 수집된다. 각 스펙과 관련된 FORM의 산출물은 다음과 같다. EXCS1는 컴포넌트 외부의 정적 구조를 기술하며 스펙 내용은 FORM의 모듈 아키텍쳐에서 얻어진다. EXCS2는 컴포넌트 외부의 행동을 기술하며 스펙 내용은 시스템 전체의 상태를 기술하는 시스템 상태 챠트에서 얻어진다. EXCS4는 컴포넌트 외부의 동시성 정보를 기술하며 스

펙 내용은 FORM의 프로세스 아키텍쳐에서 얻어진다. EXCS4는 한 컴포넌트의 내부에 존재하는 스레드가 다른 컴포넌트의 실행을 요구할 때 그 동시성 정보를 기술하게 된다.

표 3에서 INCS1는 컴포넌트 내부의 정적 구조를 나타내며 스펙 정보는 FORM의 모듈 아키텍쳐와 설계 객체 모델을 이용하여 기술된다. 즉, INCS1는 한 컴포넌트 내부에 존재하는 객체들과 합성관계에 있는 다른 컴포넌트들에 대한 정적 구조 정보를 기술한다. INCS2는 컴포넌트 내부 객체들의 행동을 기술하며 스펙 정보는 FORM의 객체 상태 챠트를 이용한다. 즉, INCS2는 컴포넌트 내부 객체들이 합성관계의 다른 컴포넌트와 임의의 객체의 어떤 상태에 서로 연관되는지 알 수 있게 해준다. INCS4는 컴포넌트 내부의 동시성 정보를 나타내며 FORM의 객체 상태 챠트를 이용해 기술된다. 즉, INCS4는 현재 컴포넌트 내의 스레드가 여러 객체들의 실행과 연관될 때 기술된다.

4.2 제품 라인 컴포넌트의 스펙 내용

4.2.1 제품 라인 컴포넌트 스펙의 기본 정보

표 3에 각 스펙은 휘처 정보와 가변성 정보의 기술을 기본적으로 필요로 한다. 표 4는 휘처 정보를 표 5는 가변성 정보를 보여주고 있다. 표 4는 EXCS1에만 기술되고 다른 스펙들은 EXCS1에 기술된 휘처 정보를 서로 같이 공유한다. 표 5의 가변성 정보는 각 스펙에서 가변성이 발생할 때마다 모든 가변성 발생 위치에 존재하게 된다. 표 4는 휘처 모델러의 역할 정의와 휘처 모델을 통해서 얻어지고 표 5는 가변성 관리자와 제품 라인 컴포넌트 구현자의 역할 정의를 통해 얻어진 정보들이다.

표 5에서 가변성 정보는 가변점 기본정보와 가변점 구현정보로 나뉜다. 표 5의 가변점 기본정보인 '현재 가변점에 영향을 받는 휘처 리스트'를 그림 3과 함께 설명하면 다음과 같다. 그림 3 휘처 모델에서 고객 상품 대금 지불 방식(신용 카드, 무통장 입금, 사이버 머니)은 휘처 Payment에서 결정되므로 휘처 Payment에 가변점이 존재하게 된다. 이 가변점에서 무통장입금(On Line Payment)만 지불 방식으로 선택한다면 그 가변점에서 결정에 영향을 받는 휘처는 Bank Information이 된다. 또한, 표 5의 가변점 구현정보인 '가변성 영향 유형'은 임의의 가변점에서 가변성 결정으로 제품 라인 컴포넌트에 미치는 영향을 의미하며 이 항목에는 표 5에서 정의된 여러 가지 영향 유형이 조합되어 기술될 수 있다.

4.2.2 제품 라인 컴포넌트 스펙의 정형화

제품 라인 컴포넌트 스펙은 Z, VDM, PROMELA와 같은 정형 언어로 기술될 수 있다. 정형언어들은 시간 기반, 상태 기반, 이행 기반, 기능, 동작 스펙 언어로 분류되었다[17]. 정형언어는 비정형적으로 기술할 때 찾을 수 없었던 문제의 발견과 스펙에 대한 자동화 지원이 용이한 장점이 있다. 그리하여, 서로 다른 이해관계자가 서로 다른 관점의 여러 정형 언어들을 혼합하여 시스템을 기술하기도 하나 이런 경우에 일관성 유지의 어려움이 발생한다. BNF는 다른 정형언어 보다 상대적으로 사용이 편리하다. 이 논문의 핵심이 특정한 정형 언어의

표 3 제품 라인 컴포넌트 스펙의 분류

컴포넌트 스펙분류	컴포넌트 가변성 위치	컴포넌트 가변성 구현 형태
정적(static) : EXCS1	Interface	interface 변경
	Interface Operation	interface operation 변경
	In, Out, InOut Ports	in, out, inout 매개변수 값 변경과 매개변수 타입 변경
행동(behavior) : EXCS2	Operation Precondition	interface operation 선행조건 변경
동적(dynamic) : EXCS3	Interface Operation Invocation Sequence (workflow)	데이타 전달 흐름 변경 (data-oriented workflow change)
		이벤트 전달 흐름 변경 (event-oriented workflow change)
동시성(concurrency) : EXCS4	Component Thread Control Condition, Shared Info.	컴포넌트 내부 스레드가 다른 컴포넌트를 실행할 경우 그 스레드 실행과 멈춤을 위한 제어조건 및 공유 정보 변경
정적(static) : INCS1	Class	class(objects) 추가 삭제 변경
	Attribute	attribute 값과 타입 변경
	Method	method 추가 삭제 변경
	Parameter	parameter 값과 타입 변경
행동(behavior) : INCS2	Object Method Precondition	method operation 선행조건 변경
동적(dynamic) : INCS3	Method Invocation Sequence (workflow)	데이타 전달 흐름 변경 (data-oriented workflow change)
		이벤트 전달 흐름 변경 (event-oriented workflow change)
동시성(concurrency) : INCS4	Object Thread Control Condition, Shared Info.	컴포넌트 내부 스레드가 여러 객체를 실행할 경우 그 스레드 실행과 멈춤을 위한 제어 조건 및 공유 정보 변경

표 4 제품 라인 컴포넌트의 휘처 정보

Feature Layer	Capability	service	
		operation	
		NFR(Non-Functional Requirement)	
	Operating Environment	hardware operating environment	
		software operating environment	
	Domain Technology	supplementary domain technology	
		core domain technology	
	Implementation technique	GUI presentation markup languages (XML, cHTML, WML, HTML, etc.)	
		data communication protocols (TCP, UDP, HTTP, WAP, X.25, etc.)	
		database connection technology(ODBC, JDBC, etc.)	
		data(ADT, jpg, wav, etc.)	
Feature Type	Mandatory	Alternative	Optional
Feature Relations	composited-of	Feature1, Feature2, Feature3, ...,	
	generalization	Feature1, Feature2, Feature3, ...,	
Feature Configuration Rules	Required	Feature1, Feature2, Feature3, ...,	
	Excluded	Feature1, Feature2, Feature3, ...,	

표 5 제품 라인 컴포넌트의 가변성 정보

가변점소속	가변점이 소속된 컴포넌트 스펙 유형 번호
가변점이름	Variation Points Name
가 변 점 기본정보	가변점 식별 번호(ID)
	가변점 위치(location)
	가변점 설명(description)
	현재 가변점으로 영향을 받는 가변점 식별 번호 리스트
	현재 가변점에 영향을 받는 휘처 리스트
	현재 가변점의 제약사항(constraints)
	가변점의 이론적 근거(rationale)
가 변 점 구현정보	가변점에서 고정된 휘처 바인딩 타임
	가변점에서 모든 가능한 휘처 바인딩 타임
	가변성 구현 기술(상속, 위임, 매개변수화, 정적바인딩, 동적바인딩, 집합 등)
	가변점에서 가변성 결정 식의 위치(할당문, 조건문, 분기문 등)
	가변성 결정 식의 내용
	가변성 결정에 대한 입력
	결정된 가변(resolved variants)
	가변성 영향 유형(interface, interface operation, operation parameter, class, attribute, method, method parameter, method body, database field, database table, work flow)

사용이 아니라 제품 라인에서 컴포넌트 스펙 분류 및 스펙 정보 추출이기 때문에 표 3에 존재하는 스펙들을 모두 BNF로 기술한다면 각 스펙들 간에 일관성 유지와 스펙 내용의 이해가 더욱 용이해질 것이다.

표 3의 모든 스펙들은 BNF로 정형화되었고 표 6-9는 각 스펙에서 중요 부분을 나타낸다. 각 표에 대해 설명하면 표 6에서 implementation_tech는 휘처 모델의 구현 기술 계층을 의미한다. implementation_tech는 표 4처럼 GUI_ML, COM_PROTOCOL, DB_CONNECT, DATA로 분류된다. 임의의 제품 라인 컴포넌트가 구현 기술 계층의 휘처를 구현하고 있다면 implementation_tech을 통해 그 컴포넌트가 어떤 종류의 기술을 구현하고 있는지 알 수 있게 된다. 위에서 GUI_ML는 웹 애플리케이션을 위한 마크업 언어를 COM_ PROTOCOL은 데이타 통신 프로토콜을 DB_CONNECT는 데이타 베이스 연결 기술을 의미하며 DATA는 데이타 형식(jpg, gif, wav, mpg, avi, dxf)과 데이타 구조(ADT, E-R diagram)를 의미한다. 단, 이러한 기술에 대한 분류는 신기술의 출현에 따라 새롭게 분류가 추가될 수 있다.

각 스펙에는 발생하는 가변성의 위치가 표시되게 된다. 표 6은 컴포넌트 외부 정적 구조를 나타내며 가변성은 provided interface에서 operation 추가·삭제를 통해 발생한다. 표 7은 컴포넌트 외부 동시성 정보를 나타내며 가변성은 스레드 공유 정보(데이타, 이벤트), 스레드

표 6 EXCS1: 컴포넌트 외부 정적 정보 스펙

```
component external static:
      component_properties features interfaces;
component_properties: component_status
component_version;
component_status: TRANSIENT | PERSISTENT;
component_version: literal;
features: feature | feature features;
feature: feature_name feature_information;
feature_name: literal;
feature information: feature_type feature_layers
feature_config_rules;
feature_type: mandatory | variant;
mandatory: MANDATORY;
variant: OPTIONAL | ALTERNATIVE;
feature_layers:
   capability | operating_env | domain_tech |
implementation_tech;
capability: SERVICE | OPERATION | NFR;
operating env: HARDWARE | SOFTWARE;
domain_tech: SUPPLEMENTARY | CORE;
implementation_tech:
   GUI ML | COM_PROTOCOL | DB_CONNECT |
   DATA;
feature_config_rules: feature_config_rule |
          feature config rule feature config rules;
feature_config_rule: config_rules feature | ;
config_rules: CR_REQUIRED | CR_EXCLUDED;
interfaces: provided interfaces
       | provided_interfaces required_interfaces
   | provided_interfaces required_interfaces
     system interfaces;
provided_interfaces: ≪variation_points≫ provided_interface
   | ≪variation_points≫ provided_interface
     provided interfaces;
provided_interface:
    PROVIDED INTERFACE interface_name operations;
                ..... 생략 .....
```

표 7 EXCS4: 컴포넌트 외부 동시성 정보 스펙

```
component external concurrency: thread_owner_component
multi_threads_synchronization_type;
thread_owner_component: current_component;
current component: identifier;
multi_threads_synchronization_type: critical_section |
mutex | semaphore;
critical section: shared element threads;
mutex: shared_element threads;
/*세마포어는 동기화된 스레드 수만큼 스레드들이 존재해야 함.*/
semaphore: shared element
the_number_of_synchronized_threads threads;
shared_element: ≪variation_points≫shared_data |
                    ≪variation_points≫shared_event;
shared_data: identifier;
shared_event: identifier;
the number of synchronized threads: NUMERIC;
threads: thread | thread threads;
thread: thread_information locking_block;
thread_information: thread_name thread_number
thread_priority
  thread_functionality thread_break_condition
thread description
  thread_constraints invoked_components;
invoked_components: invoked_component |
                invoked component invoked_components;
invoked_component: component_name;
component_name: component_external_static;
thread break condition: ≪variation_points≫statement;
locking_block: statements;
statements : statement | statement statements;
statement : literal;
.....생략.....
```

표 8 INCS1: 컴포넌트 내부 정적 정보 스펙

```
component_internal_static: implemented_classes
contained components;
implemented_classes: core_class | core_class
helper_classes;
core_class: CORE class;
helper_classes: HELPER classes;
classes: ≪variation_points≫ class | ≪variation_points≫
class classes;
class: assess_modifier CLASS class_name fields methods;
class_name: identifier;
fields: field | field fields;
field: ≪variation_points≫ field | ;
.....생략.....
methods: method | method methods;
method: ≪variation_points≫ method | ;
.....생략.....
parameters: parameter | parameter parameters;
parameter: ≪variation_points≫ parameter | ;
.....생략.....
/* 컴포넌트 내부의 합성(composition) 관계에 있는
컴포넌트들 */
contained_components: COMPOSITION components;
components: component | component components;
component: component_external_static | ;
```

표 9 EXCS2: 컴포넌트 외부 행동 정보 스펙

```
component_external_behavior: provided_interface;
provided interface: interface_behavior interface_structure;
interface_behavior: context invariants;
context: interface_name;
invariants: conditions;
interface_structure:
          PROVIDED INTERFACE interface_name
operations;
operations: operation | operation operations;
operation: operation_behavior operation_structure;
operation behavior: preconditions postconditions
exceptions;
preconditions: ≪variation_points≫ conditions;
postconditions: conditions;
operation_structure: access_modifier return_value
                     operation_name '(' parameters ')';
.....생략.....
conditions: condition | condition conditions;
condition: operand operator operand;
operand: identifier;
operator: '>' | '< ' | '>=' | '<=' | '==' | '!=';
exceptions: exception | exception exceptions;
exception: exception_name | ;
exception_name: identifier;
interface_name: identifier;
identifier: STRING;
```

제어 조건에서 발생한다. 표 8은 컴포넌트 내부 정적 구조를 나타내며 가변성은 핵심 클래스와 조력 클래스의 속성, 함수, 매개변수에서 발생한다. 표 9는 컴포넌트 외부 행동 정보를 나타내며 가변성은 전체 시스템의 어떤 상태 진입을 결정하는 provided interface의 operation 선행조건에서 발생한다. 기타의 다른 스펙에서의 가변성 발생 위치는 표 3에 기술되어 있다.

5. 연구 적용 및 평가

5.1 연구의 적용

본 논문은 제품 라인 전자상거래 시스템에 적용되었다. 전자상거래 시스템은 개설 사이트 수가 날로 증가하고 있고 여러 물품 판매가 가능하여 사용자의 다양한 요구가 발생한다. 이로 인해, 전자상거래 시스템은 더욱 복잡해져 제품 라인의 적용에 적합한 도메인이 된다.

전자상거래 시스템은 고급과 저급 기능으로 구분된다. 고급 기능은 경매, 마일리지 등의 기능을 제공하고 여행 및 보험과 같은 상품도 취급하며 고급 인증기능(디지털 서명, 생체 정보)을 제공한다. 반면에, 저급 전자상거래 시스템은 판매 물품이 간단하며(예, 온라인 서점) 고객 ID 및 비밀번호로 간단한 사용자 인증을 처리한다.

그림 3은 제품 라인 전자상거래 시스템에서 휘처들 간의 공통점과 차이점을 식별하여 휘처 모델을 작성한 것이다. 그림 3에서 능력계층의 'Payment', 'On Line Payment', 'Credit Card Payment', 'Cyber Money Payment'는 하나의 휘처 바인딩 유닛으로 선정되고 최종적으로 Payments 컴포넌트로 식별되게 된다.

표 10과 표 11은 Payments 컴포넌트에 대한 스펙 내용이다. 그림 1의 생명주기에 따라 제품 라인 전자상거래 시스템의 핵심 자산인 컴포넌트의 개발이 이루어졌다. 그 과정에서 여러 가지 산출물들과 기타 컴포넌트 스펙이 작성되었으나 기술 공간의 부족으로 여기서 상세한 설명은 생략한다. 이 논문의 좀 더 자세한 내용은 과학재단의 관련 연구보고서에 기술되어 있다.

표 10 Payments 컴포넌트의 외부 정적 구조 스펙(EXCS1)

```
/*EXCS1: black box component specification
          for component external static information */
component external static:
 component properties: component status = TRANSIENT;
                       component_version = "1.0";
 features:
   feature name = "Payment".
        feature type = MANDATORY, feature_layers = SERVICE;
   feature name = "Credit Card Payment".
        feature type = OPTIONAL, feature_layers = SERVICE;
   feature name = "On Line Payment".
        feature type = OPTIONAL, feature layers = SERVICE;
   feature name = "Cyber Money Payment".
        feature_type = OPTIONAL, feature_layers = SERVICE;
 interfaces:
   provided interface:
      PROVIDED INTERFACE interface name = "payment";
        operation: access modifier = PUBLIC, return value = BOOLEAN,
          operation_name = "requestPayment( IN OBJECT "orders" )";
   required interface:
      REQUIRED INTERFACE interface name = "authorize"
                                component name = "Authorization";
        operation: access modifier = PUBLIC, return value = BOOLEAN,
          operation name = "authorize(OUT OBJECT "creditCard",
                                OUT INTEGER "runningTotal",
                                OUT OBJECT "pCustomer" )";
```

5.2 연구의 평가

논문의 목적은 FORM의 확장이므로 이 절에는 FORM과 다른 제품 라인 공학 방법론들의 평가를 기술한다. 대표적인 제품 라인 공학 방법론인 QADA[18], KobrA, COPA[19]가 FORM과 함께 평가에 사용된다.

모든 방법론의 표준적인 평가 프레임워크로 잘 알려져 있는 NIMSAD(Normative Information Model-

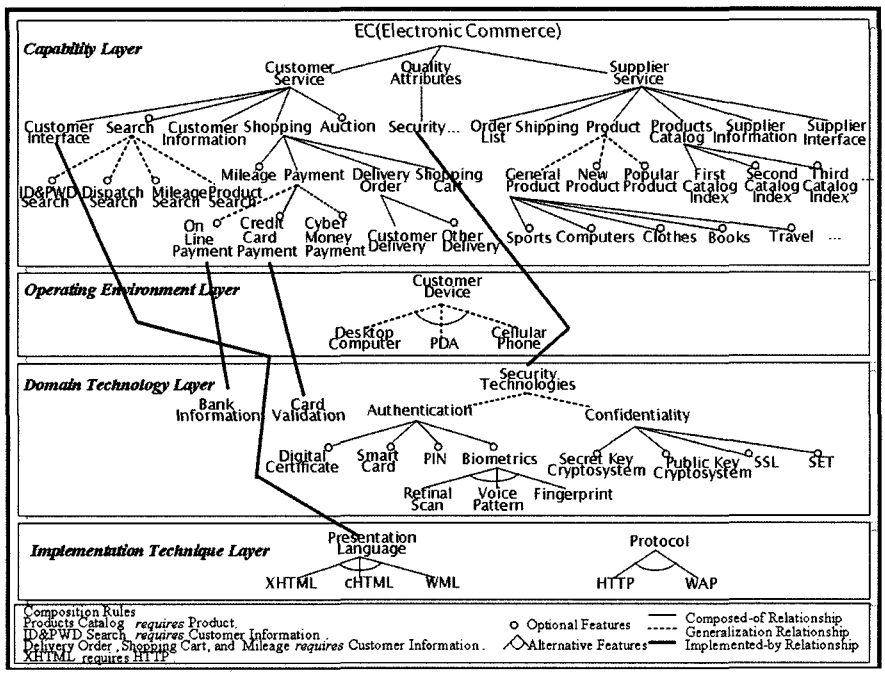

그림 3 제품 라인 전자상거래 시스템의 휘처 모델

표 11 Payments 컴포넌트의 가변성 정보

```
/* Variation_Points Specification for Component */
variation_point:
 vp_spec_type = INCS1;
 vp name = "Product Payment Type";
 vp_general_info:
  vp_identifier = "INCS1_01";
  vp location = "Pavments.Pavments.vdRequestPavment()";
  vp_description = "상품 결제 대금 지불 방식을 선택하는 가변점";
  vp_constraints = "적어도 하나가 선택되어야 됨";
  vp rationale = "고객 요구대로 여러 형태의 상품 대금 지불 방식 처리 필요";
  affected_variation_points = "INCS2_01";
  affected_features = "CreditCardPavment", "OnLinePayment",
                      "CyberMoneyPayment";
 vp_implementation_info:
  vp_binding_time:
   available binding time=ASSETS BUILD, PRODUCT_BUILD, RUNTIME;
  vp_implementation_tech = DYNAMIC_BINDING;
  vp_resolving_starting_point = IF;
  vp resolving statements =
    " if (orders.pavmentTvpe = = CREDIT_CARD)
            Pavment p = new CreditCardPavment(...);
       else if (orders.pavmentTvpe = = ON LINE)
            Pavment p = new OnLinePavment(...);
       else Pavment p = new CyberMoneyPavment( ); ";
  vp resolving inputs = " if (orders.pavmentTvpe = = CREDIT CARD)
                            Pavment p = new CreditCardPayment(...);
                          else Pavment p = new OnLinePavment(...);";
   vp resolved variants = "CreditCardPavment", "OnLinePayment";
   vp_resolution_effect = WORK_FLOW;
```

based System Analysis and Design)[20]와 제품 라인 공학 방법론에 대한 평가 논문인 Matinlassi[21]을 기초로 본 논문의 평가가 수행된다. 표 12의 질문들은 NIMSAD와 Matinlassi로부터 선정되고 표 13의 평가 결과는 Matinlassi에 의거하여 기술된다. 단, 표 12-13에서 Q6은 저자가 새롭게 추가한 항목이다.

표 13의 방법론들은 서로 다른 특징들을 제공한다. 예를 들어, QADA와 KobrA는 UML과 같은 표준에 순응하여 방법론의 사용이 편리하나 Koala나 휘처 모델과 같은 제품 라인에 고유한 모델이 존재하지 않는 단점이 있다. 또한, QADA, KobrA, COPA는 가변성 표현을 아키텍처 설계시에 주로 고려하나 FORM은 요구사항 도출로 휘처 모델을 작성할 때 고려한다. FORM과 같이 방법론의 초기 단계부터 가변성 식별 및 표현은 제품 라인에서 재사용을 증대시킨다고 널리 알려져 있다.

제품 라인 컴포넌트에 관해 각 방법론을 비교하면 다음과 같다. COPA는 object model로 KobrA는 structural model로 FORM은 design object model로 각각 컴포넌트 내부 객체들의 정적 구조를 나타내고 있으며 QADA는 package diagram으로 COPA는 Koala model로 컴포넌트들 간의 정적 구조를 나타내고 있다. 또한, QADA, KobrA, FORM는 UML의 상태와 순서 다이어그램과 유사한 형태로 컴포넌트 내부 객체들의 동적, 행동, 동시성 정보를 나타내고 있으나 COPA는 Interface와 Component Data Sheet로 컴포넌트 기능, 개념, 제약사항, 가변성 정보만을 기술하고 있어 컴포넌트의 동적 및 동시성 정보를 제공하지 않고 있다.

표 12 방법론 평가를 위한 질문 프레임 워크

번호	질 문 내 용
Q_1	각 방법론에서 모델링의 주요 관점은 무엇인가
Q_2	각 방법론이 제시한 제품 라인에 고유한 요구공학 방법은 무엇인가
Q_3	각 방법론을 사용할 때 사용자의 이익은 무엇인가
Q_4	각 방법론의 적용을 사용자에게 어떻게 안내하는 가
Q_5	각 방법론의 설계 단계(design steps)는 무엇인가
Q_6	각 방법론에서 생성·관리되는 상세한 컴포넌트 관련 산출물은 무엇인가
Q_7	각 방법론이 적용하는 아키텍처 관점(viewpoints)은 무엇인가
Q_8	각 방법론은 가변성 표현을 어떻게 지원하는 가
Q_9	각 방법론의 세부 단계에서 가변성의 표현 시작 시점은 언제인가

표 13 제품 라인 공학 방법론의 평가 결과

	QADA	KobrA	COPA	FORM
Q_1	아키텍처 중심	컴포넌트 기반	아키텍처 중심, 컴포넌트 기반	아키텍처 중심, 컴포넌트 기반
Q_2	인터페이스	N/A	N/A	휘처 지향
Q_3	MDA, IEEE-Std-1471-2000 표준 순응	UML, MDA 표준 순응	제품라인공학 이해관계자들 간 의사소통개선	휘처 개념으로 고객과 공학자들 간 공통언어제공
Q_4	보고서	책	N/A	논문
Q_5	요구공학, 개념적 아키텍처 설계, 구체적 아키텍처 설계, 아키텍처 품질분석	프레임워크공학, 응용공학	제품패밀리공학, 제품공학, 플랫폼공학	도메인공학, 응용공학
Q_6	package, state, sequence diagram	structural, behavioral, functional, decision model	object model, interface component data sheet	design object model, message sequence object state chart
Q_7	structural, behavior, deployment, development	N/A	customer, application, functional, conceptual, realization	conceptual, process, module, deployment
Q_8	개조된 UML	개조된 UML	Koala model	Feature model
Q_9	아키텍처 설계	아키텍처 설계	아키텍처 설계	요구사항 도출

즉, 위 방법론들은 현재 제품 라인 컴포넌트 정보에 대한 분류 체계가 애매모호하고 각 분류에 따라 어떠한 정보들이 수집·관리되어야 하는지 정의하지 않았다. 특히, 제품 라인에 가장 중요한 가변성을 기술할 때 요구되는 상세한 정보는 무엇인지 또는 제품 라인에서 가변성이 각 모델의 어느 위치에서 발생하는지를 파악하지 않았다. 이에 관해 본 논문은 제품 라인 컴포넌트 구조를 기반으로 컴포넌트 정보가 몇 가지 관점에서 어떻게 분류·수집·기술되어야 하는가를 보여주고 있다.

6. 결론 및 향후 연구

본 논문은 FORM 기반으로 제품 라인 컴포넌트의 스펙 방법을 제시하였다. FORM의 컴포넌트 생명주기 작성으로 각 단계(phase)에서 이해관계자의 역할과 산출물들을 정의하여 스펙 정보를 수집하였다. 그리고, 제품 라인 컴포넌트 구조의 작성으로 그 구조에 기반하여 제품 라인 컴포넌트의 정적, 행동, 동적, 동시성에 대해 제품 라인 컴포넌트 스펙을 컴포넌트 내부와 외부 정보로 구분하였다. 또한, 모든 스펙에서 수집된 자료와 각 자료들 간의 상관관계는 BNF로 정형화하였고 사례 연구로서 제품 라인 전자상거래 시스템을 기술하였다.

스펙 작성 결과로서 제품 라인 컴포넌트의 동적과 동시성 정보는 BNF로 완벽히 표현할 수 없는 한계를 발견하였다. 즉, 컴포넌트 외부 동적 정보에서 컴포넌트들 간의 시간적 흐름에 따른 호출 관계를 나타낼 때 자기 자신 컴포넌트에 대한 동적 정보를 BNF로 표현하는 데 어려움이 발생했다. 이 문제 해결을 포함한 향후 연구는 다음과 같다. 1) 제품 라인 컴포넌트 구조의 그래픽적 표기로 그 구조를 더욱 개선시킨다. 2) 각 스펙에서 사용되는 표기법을 정의하고 찾아진 자료들을 이용해 각 스펙에서 스펙 절차를 정의한다. 3) 제품 라인 컴포넌트의 동적과 동시성 정보를 적합한 정형언어로 기술한다. 4) 제품 라인 지원 CASE 도구에서 제품 라인 컴포넌트 소스 코드의 자동 생성에 본 논문을 활용한다.

참 고 문 헌

[1] Kang, K., Kim, S., Lee, J., Kim, K. et al., "FORM: A Feature-Oriented Reuse Method with domain-specific reference architectures," Annals of Software Engineering, vol. 5, pp. 143-168, 1998.

[2] Chastek, G., Donohoe, P. and McGregor, J. D., A Study of Product Production in Software Product Lines, Technical Note, CMU/SEI-2004-TN-012, p. 5, 2004.

[3] D'Souza, D.F. and Wills, A.C., Objects, Components and Frameworks with UML, Addison-Wesley, 1998.

[4] Atkinson, C. et al., Component-Based Product Line Engineering with UML, Addison-Wesley, 2001.

[5] Brown, A.W. and Barn, B., "Enterprise-scale CBD: building complex computer systems from components," Sterling Software, STEP '99, 1999.

[6] Dias, M.S. and Vieira, M.E.R., "Software architecture analysis based on statechart semantics," IWSSD'00, IEEE, 2000.

[7] Beugnard, A., Jezequel, J.M., Plouzeau, N. et al., "Making components contract aware," Computer 32 (7), IEEE, pp. 38-45, 1999.

[8] Fettke, P. and Loos, P., "Specification of business components," LNCS 2591, Springer-Verlag, pp. 62-75, 2003.

[9] Albani, A., Keiblinger, A., Turowski, K. et al., "Domain based identification and modelling of business component applications," LNCS 2798, Springer-Verlag, pp. 30-45, 2003.

[10] Bachmann, F., Bass, L., Buhman, C. et al., Volume II: Technical Concepts of Component-Based Software Engineering, Technical Report, CMU/SEI-2000-TR-008, 2000.

[11] Taulavuori, A., Component Documentation in The Context of Software Product Lines, VTT Publications 484, VTT Electronics, Espoo, 2002.

[12] Taulavuori, A., Niemel, E. et al., "Component document - a key issue in software product lines," Information and Software Technology 46, VTT Electronics, pp. 535-546, 2004.

[13] Ommering, R.V., "Building product populations with software components," ICSE'02, pp. 255-265, 2002.

[14] Sharp, D., "Exploiting object technology to support product variability," Proceedings of 18th Digital Avionics Systems Conference, IEEE, 1999.

[15] Anastasopoulos, M. and Gacek, C., "Implementing product line variabilities," ECOOP, ACM, pp. 109-117, 2001.

[16] Myllymäki, T., Variability Management in Software Product Lines, Technical Report of Software Systems Laboratory, Tampere University of Technology, pp. 23-26, 2001.

[17] Lamsweerde, A.V., "Formal specification: a roadmap," ICSE'00, pp. 147-159, 2000.

[18] Matinlassi, M., Niemelä, E. and Dobrica, L., Quality-driven Architecture Design and Quality Analysis Method, VTT Electronics, http://www.inf.vtt.fi/pdf/publications/2002/P456.pdf, 2002.

[19] America, P., Obbink, H., Muller, J. and Ommering, R.V., "COPA: A Component-Oriented Platform Architecting method for families of software intensive electronic products," SPLC1, http://www.extra.research.philips.com/sae/copa/copa_tutorial.pdf, 2000.

[20] Jayaratna, N., Understanding and Evaluating Methodologies : NIMSAD : A Systematic Framework. McGrawHill, London, 1994.

[21] Matinlassi, M., Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA, VTT Electronics, ICSE, 2004.

조 혜 경

1996년 전북대학교 전산통계학과 석사 2001년 전북대학교 전산통계학과 박사 1996년~1997년 전북대학교 컴퓨터과학과 조교. 1998년~1999년 한동대학교 GIS연구소 연구원. 2000년~2003년 한동대학교 전산전자공학부 시간강사. 2004년~2005년 포항공과대학교 컴퓨터공학과 연구원. 관심분야는 소프트웨어공학, 지리정보시스템, 전자상거래