소프트웨어 제품라인 아키텍처 모델에서의 가변성 표현 방법 비교 연구†

(Expressing Variability in Software Product Line Architecture Models: A Comparative Study)

이혜선‡ 조성배§ 강교철¶
(Hyesun Lee) (Sungbae Cho) (Kyo Chul Kang)

요 약 소프트웨어 제품라인 공학은 제품라인의 공통적인 부분과 차이점을 핵심 자산으로 관리하여 품질을 높이고, 핵심 자산을 계획된 제품들에 재사용함으로써 소프트웨어의 생산성을 높이는데 효과적인 방법이다. 제품라인 아키텍처에는 제품에 따라서 포함되거나 포함되지 않는 가변적인 부분이 있기 때문에, 기존 단일 소프트웨어를 대상으로 한 아키텍처 모델과는 달리, 제품라인 아키텍처는 제품 간의 차이점인 가변성을 나타낼 수 있어야 한다. 기존에 여러 연구자가 제품라인 아키텍처 모델에 가변성을 표현하는 방법을 제안하였지만, 이들이 제시한 표현 방법들의 강약점을 분석하고 차이점을 비교하는 연구가 부족하였다. 따라서 본 논문에서는 현재까지 제안된 제품라인 아키텍처 모델에서의 가변성 표현 방법을 분석하고 비교하여, 적합한 표현 방법을 선택하는데 가이드가 되고자 한다.

키워드 테스트 소프트웨어 제품라인 공학, 가변성 구현, 소프트웨어 제품라인 아키텍처

Abstract Software product line engineering is a software reuse paradigm that helps organizations improve software productivity and quality by developing software products from reusable core assets. For the satisfaction of common and variable requirements among products in the product line, the core assets must be configurable according to the selection of variable features. Therefore, unlike software architecture model of a single product, product line architecture model must embed and express variabilities among the products. Many researches have proposed methods of embedding and expressing variabilities in the product line architecture models, but there are few comparative studies on the proposed methods. In this paper we discuss strong points and weak points of the proposed methods and compare expressiveness of the methods, which helps select a proper method.

Key words Software product line engineering, variability, software product line architecture

† 본 연구는 정보통신산업진흥원의 SW공학 요소기술 연구개발사업 및 한국연구재단을 통해 교육과학기술부의 세계수준의 연구중심대학육성사업(WCU, R31-10100)으로부터 지원받아 수행되었습니다.
‡ 학생회원 : 포항공과대학교 컴퓨터공학과
compial@postech.ac.kr
§ 비 회 원 : 포항공과대학교 정보전자융합공학부
craxy2010@postech.ac.kr
¶ 종신회원 : 포항공과대학교 정보전자융합공학부 교수
kck@postech.ac.kr
논문접수 : 2011년 09월 03일
심사완료 : 2011년 09월 23일

1. 서 론

소프트웨어 제품라인 공학(Software Product Line Engineering)[1]은 어떤 도메인(Domain)에서 시장 요구를 만족하고 공통된 기능을 공유하며 핵심 자산(Common Assets)으로부터 개발되는 유사한 제품군을 개발하는 새로운 재사용

패러다임이다. 소프트웨어 제품라인 방법론을 통하여 소프트웨어를 개발하면 제품라인의 공통적인 부분과 차이점 부분을 핵심 자산으로 관리하여 품질을 높이고 핵심 자산을 계획된 제품들에 재사용함으로써 재사용성을 보장 받을 수 있다[2].

소프트웨어 제품라인의 핵심 자산(요구사항, 아키텍처, 코드 등을 포함)으로부터 계획된 제품들을 개발하기 위해서는, 제품라인의 제품들 사이에 공통점(Commonality)과 차이점(Difference)이 분석되고, 분석된 차이점이 핵심 자산에 가변성(Variability)으로써 구현이 되어야 한다. 예를 들어, 어떤 휘처(Feature; 제품라인에서 제품군의 공통점과 차이점 분석 단위)가 제품에 따라서 포함이 되거나 포함이 되지 않는 휘처라면, 핵심 자산 중 이 휘처를 구현하는 아키텍처 컴포넌트 및 자산 코드가 휘처의 선택에 따라서 제품에 포함되거나 포함되지 않도록 할 수 있어야 한다. 이러한 가변성을 분석하고 관리하는 것이 성공적인 제품라인 공학을 위해 필수적이다.

제품라인 아키텍처에는 제품에 따라서 포함되거나 포함되지 않는 가변적인 부분이 있기 때문에, 기존 단일 소프트웨어를 대상으로 한 아키텍처 모델과는 달리 아키텍처 모델에 제품라인의 가변성을 나타낼 수 있어야 한다. 이를 위해 여러 연구자가 제품라인 아키텍처 모델에 가변성을 표현하는 방법을 제안하였다 [3-4, 6-12]. 하지만 이들이 제시한 제품라인 아키텍처 모델에서의 가변성 표현 방법들의 차이점 및 강약점을 비교하는 연구가 부족하였다.

따라서 본 연구에서 현재까지 제안된, 제품라인 아키텍처 모델에서의 가변성 표현 방법을 분석하고 강약점을 비교하여, 적합한 표현 방법을 선택하는데 가이드가 되고자 한다. 본 논문에서 엘리베이터 컨트롤 시스템 제품라인이 표현 방법을 적용할 예제로 사용되었다. 본 논문은 아키텍처 모델 상의 가변성 표현 방법 비교에만 초점을 맞추며, 컴포넌트 명세 방법 및 컴포넌트 명세에서의 가변성 표현은 본 논문의 범위에 벗어난다.

본 논문의 구성은 다음과 같다. 2장에서는 예제로 사용될 엘리베이터 컨트롤 시스템 제품라인과 해당 제품라인 아키텍처 모델에 대해 간략히 설명하고자 한다. 3장에서는 본격적으로 예제와 함께 각 아키텍처 모델에서의 가변성 표현 방법을 소개하고 강약점을 분석할 것이다. 4장에서는 분석한 내용을 바탕으로 아키텍처 모델에서의 가변성 표현 방법들을 비교할 것이다. 마지막으로 5장에서는 토론과 함께 논문을 마무리하고 향후 연구를 소개하고자 한다.

2. 사례 연구 : 엘리베이터 컨트롤 시스템 제품라인

본 장에서는 본 논문의 예제로 사용될 엘리베이터 컨트롤 시스템(Elevator Control System: ECS) 제품라인을 소개하고자 한다. ECS는 실시간 임베디드 시스템(Realtime Embedded System)의 일종으로써, 사용자를 건물의 층 사이에 편리하고 빠르게 이동시키는 것이 목적이다. 본 논문에서 대상으로 하는 ECS 제품라인은 고층 빌딩 엘리베이터, 종합 병원 엘리베이터, 아파트 엘리베이터를 포함하며, 제품라인의 공통점 및 차이점 휘처를 나타내는 휘처 모델은 그림 1과 같다.

ECS 제품라인의 휘처 모델(그림 1)에서 Driving Service, Call Handling, Indication Handling, Door Handling 등은 모든 제품에 공통적으로 들어가는 필수(Mandatory) 휘처이다. Driving Service 중 VIP(특정한 층에만 엘리베이터가 멈추도록 엘리베이터를 운행하는 서비스)와 Emergency (지진, 화재 등의 위급 상황에 안전을 고려하여 엘리베이터를 운행하는 서비스)는 일부 제품에만

들어가는 선택적(Optional) 휘처이다. Motor의 타입을 나타내는 Hydraulic과 Electric은 둘 중 하나만 선택될 수 있는 택일(Alternative) 휘처이다.

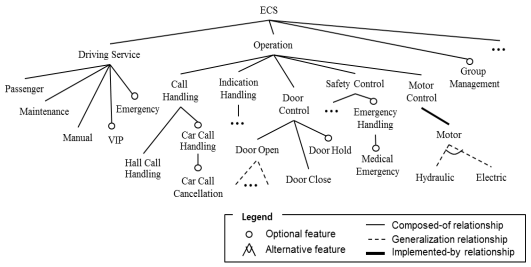

그림 1. ECS 제품라인의 휘처 모델

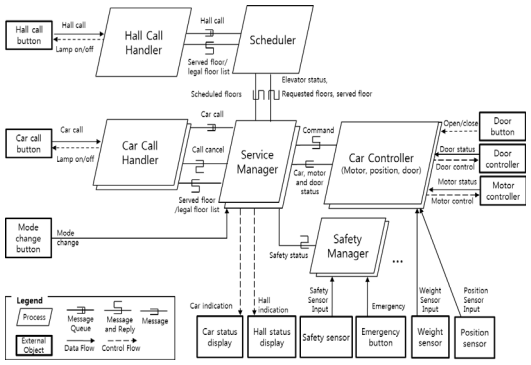

그림 2. ECS 제품라인의 프로세스 아키텍처 모델

ECS 제품라인의 프로세스 아키텍처 모델은 그림 2와 같다. 프로세스 아키텍처 모델에서 프로세스는 동시에 수행될 수 있는 실행 단위를 나타낸다. ECS는 건물의 홀에서부터 오는 Call을 처리하는 Hall Call Handler 프로세스, 엘리베이터의 케이지로부터 오는 Call을 처리하는 Car Call Handler 프로세스, 엘리베이터가 이동할 층을 스케쥴링하는 Scheduler 프로세스, 각 엘리베이터의 상태를 관리하고 Driving Service를 관리하는 Service Manager 프로세스, 엘리베이터 케이지를 관리하는 Car Controller 프로세스, 그리고 안정성을 관리하는 Safety Manager 프로세스로 구성된다.

그림 2에서 프로세스 아키텍처 모델의 가변성은 현재 표현이 되지 않았지만, 프로세스 및 커넥터 중 휘처의 선택에 따라서 제품에 포함되거나 포함되지 않아야 할 부분이 있다. Car Call Handler 프로세스는 Car Call Handling 휘처가 선택될 때에만 제품에 포함되는 프로세스이고, Safety Manager 프로세스는 Emergency Handling 휘처가 선택될 때에만 제품에 포함되는 프로세스이다. Emergency button 외부 객체는 Hospital Emergency 휘처가 선택될 때에만 제품에 포함되는 외부 객체이다.

다음 장에서는 ECS 제품라인의 프로세스 아키텍처 모델을 예제로 하여, 각 가변성 표현 방법을 분석하고 강약점을 파악할 것이다.

3. 소프트웨어 제품라인 아키텍처 모델에서의 가변성 표현 방법

3.1. 독립적 가변성 모델을 이용한 표현 방법

Thiel et al.[3]과 Pohl et al.[4]은 아키텍처 모델과 별개로 독립적인 가변성 모델(Thiel et al.은 독립적인 가변성 모델을 Variability Model이라고 명칭하고, Pohl et al.은 독립적인 가변성 모델을 Orthogonal Variability Model이라고 명칭함)을 관리하여 표현하였다. 가변성 모델에서는 가변점(Variation Point)과 가변점에 결합될 수 있는 가변부(Variant)를 명세하고, 가변점과 가변부 간의 요구(Require) 관계와 배척(Exclude) 관계를 명세 한다. 본 절에서는 그 중 Thiel et al.의 방법을 중점적으로 살펴보고자 한다.

Thiel et al.은 제품라인의 다양한 관점의 아키텍처 모델(개념적 아키텍처, 프로세스 아키텍처, Deployment 아키텍처, Physical 아키텍처)에서의 가변성을 나타내는 방법을 제안하였다(모듈 아키텍처는 대상으로 하지 않았지만 동일한 표현 방법으로 확장이 가능하다). 각 아키텍처 모델의

아키텍처 컴포넌트가 가변부가 될 수 있다고 보았으며, 각 아키텍처 모델마다 가변부와 가변점과의 관계를 독립적인 가변성 모델로 나타내었다. 그리고 각 아키텍처 모델의 독립적인 가변성 모델 사이의 관계 또한 표현하였다.

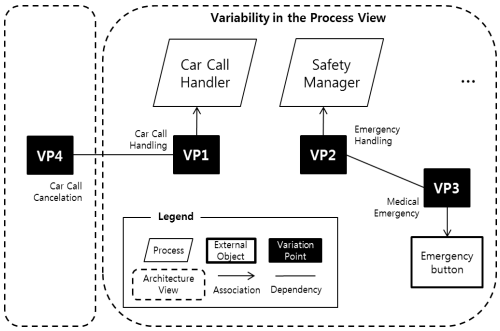

그림 3. Thiel et al.의 표현 방법으로 나타낸 ECS 제품라인의 프로세스 아키텍처 모델의 가변성 모델

그림 3은 ECS 제품라인의 프로세스 아키텍처의 가변성을 Thiel et al.의 방법으로 나타낸 그림이다. 제품라인 프로세스 아키텍처의 가변성 모델에서 "VP"는 가변점을 의미하며, 휘처 모델의 가변 휘처와 맵핑된다. 독립적인 가변성 모델을 통해서 Car Call Handler 프로세스는 Car Call Handling 휘처가 선택될 때 제품에 포함이 되고, Safety Manager 프로세스는 Emergency Handling 휘처가 선택될 때 제품에 포함이 된다는 것을 알 수 있다.

Thiel et al.의 표현 방법의 강점은 가변성을 독립적인 모델로 관리함으로써 제품라인 아키텍처 모델에 가변성을 함께 표시하는 것보다 복잡도가 감소한다는 것이다. 그리고 다양한 관점의 아키텍처 모델을 고려하였으며, 서로 다른 관점의 아키텍처 모델에 대응되는 가변성 모델 사이의 관계를 명시하였기 때문에 서로 다른 관점의 아키텍처 모델에 구현된 가변성 사이의 관계를 파악할 수 있다.

하지만 독립적인 가변성 모델을 관리하기 때문에 아키텍처 모델만으로는 가변성을 파악할 수 없다는 약점이 있다. 또한, 아키텍처의 컴포넌트만을 가변 대상으로 하며 커넥터는 가변 대상으로 하지 않는다. 따라서 커넥터에 가변성이 있는 경우(예를 들면, 프로세스 모델에서 두 프로세스의 연결 방법이 제품마다 다를 수 있는데, 어떤 제품은 메시지 큐(Message Queue) 방법[5]을 이용하지만 어떤 제품은 메시지 리플라이(Message Reply) 방법[5]을 이용할 수 있다.) 이를 표현할 수 없다. 또한, 아키텍처 컴포넌트 내부에 가변성이 있는 경우(Internal Variability; 예를 들어 어떤 휘처의 선택에 따라서 컴포넌트 내부의 구현이 달라질 경우)는 가변성 표현 대상으로 고려하지 않는다.

3.2. Gomaa의 표현 방법

Gomaa[6, 7]는 UML의 스테레오타입(Stereotype)을 확장하여 다양한 UML 모델에 가변성을 표현하였다. 본 절에서는 Gomaa의 표현 방법 중 UML 아키텍처 모델(컴포넌트 모델, 패키지 모델, 클래스 모델)에 가변성을 표현한 방법에 초점을 맞추었다.

Gomaa는 휘처 모델과 유즈케이스(Usecase)를 바탕으로 컴포넌트의 기능 및 컴포넌트가 구현하는 휘처에 따라 컴포넌트 종류를 다양하게 구별하였다. 제품에 항상 포함이 되는 필수적인 컴포넌트는 "kernel"을, 제품에 포함이 되거나 포함이 되지 않을 수 있는 선택적인 컴포넌트는 "optional"을, 컴포넌트 내부가 변할 수 있는 변화를 내포하고 있는 컴포넌트는 "variant"를 컴포넌트의 스테레오타입에 명시하도록 하였다. 가변점이 되는 컴포넌트는 "vp"로 컴포넌트의 스테레오타입에 명세하였는데, 그 중 추상(Abstract) 컴포넌트는 "abstract-vp"로, 매개변수화(Parameterization)를 사용하는 컴포넌트는 "param-vp" 표현하였다. (또한 "control", "coordinator", "data collection", "external output component" 등 컴포넌트의 역할을 구분하여 이들 컴포넌트가 프로세스로 어떻게 동작하는지를 표현할 수 있도록 하였다.)

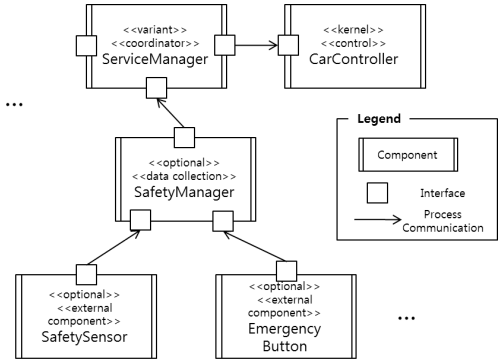

그림 4. Gomaa의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 4는 Gomma의 표현 방법으로 ECS 제품라인의 프로세스 아키텍처 모델의 가변성을 나타낸 그림이다(그림 4의 UML 컴포넌트 모델에서 각 컴포넌트는 프로세스를 의미한다). “<<kernel>>”로 표시된 CarController 컴포넌트는 모든 제품에 포함이 되는 컴포넌트이며, “<<optional>>”로 표시된 SafetyManager 컴포넌트는 일부 제품에 포함이 되는 컴포넌트이고, “<<variant>>”로 표시된 ServiceManager는 제품에 따라서 내부 구현이 바뀌는 컴포넌트임을 알 수 있다.

Gomaa의 표현 방법의 강점은 UML 기본 표현 방법을 확장하여 사용하였기 때문에 모든 UML 모델(컴포넌트 아키텍처, 패키지 아키텍처 등)에 대해 적용 가능하며 기존 UML 모델링 도구를 사용할 수 있다는 점이다. 또한, “<<variant>>”를 사용하여 컴포넌트 내부의 가변성도 표현이 가능하다.

하지만 UML의 스테레오타입에 컴포넌트의 종류만 나타내고 가변 컴포넌트에 대응되는 훼처는 표현하지 않기 때문에, 아키텍처 모델만 보고 각 가변점/가변부와 관련이 있는 훼처를 파악하기 어려운 점이 있다. 예를 들어, 그림 4에서 SafetyManager 컴포넌트가 제품에 따라서 포함이 될 수 있는 컴포넌트임을 알 수 있지만 어떤 훼처가 선택됐을 때 제품에 포함이 되는지는 알 수 없다. 또한, 아키텍처 컴포넌트만을 가변성의 대상으로 보고, 컴포넌트의 커넥션은 가변성의 대상으로 보지 않는다(클래스 모델에서도 클래스 단위가 아닌, 속성(Attribute)과 함수(Method) 단위에 대해서는 가변성의 언급이 없다). 하지만 컴포넌트의 커넥션, 클래스 모델의 속성과 함수에도 UML 스테레오타입을 확장한 가변성 표현 방법을 사용하면 가변성을 나타낼 수 있을 것이다.

3.3. Moon et al.의 표현 방법

Moon et al.[8]은 제품라인의 아키텍처 모델의 메타 모델(Meta Model)을 그림 5와 같이 정의하였는데, 제품라인 아키텍처 컴포넌트와 인터페이스, 함수(Operation)에 가변성을 나타낼 수 있는 CV_property 타입의 “vp” 속성을 추가하였다. CV_property 타입은 “common” 또는 “optional”로 정의되어 해당 개념의 가변성을 표현할 수 있다.

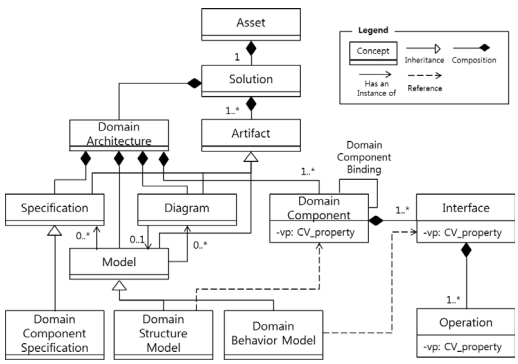

그림 5. Moon et al.의 표현 방법의 제품라인 아키텍처 모델의 메타 모델

Moon et al.은 그림 5에 정의된 제품라인 아키텍처 모델의 메타 모델을 바탕으로, 3.2절에서 소개한 Gomaa[6, 7]의 표현 방법처럼 UML의 스테레오타입을 확장하여 하여 컴포넌트, 인터페이스, 함수의 가변성을 표현하였다.

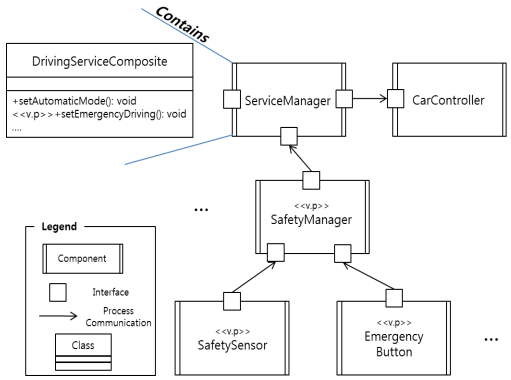

그림 6. Moon et al.의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 6은 Moon et al.의 표현 방법으로 ECS 제품라인의 프로세스 아키텍처 모델의 가변성을 나타낸 것이다. 프로세스를 나타내는 각 컴포넌트에서 "<<v.p>>"는 제품에 따라서 포함되는 아키텍처 컴포넌트를 의미한다. 그림 6에서 SafetyManager 컴포넌트, SafetySensor 컴포넌트, EmergencyButton 컴포넌트가 가변적인 컴포넌트임을 알 수 있다. 또한, ServiceManager 컴포넌트 안에 들어있는 DrivingServiceComposite 클래스에서 setEmergencyDriving() 함수가 가변적인 것을 알 수 있다.

Moon et al.의 표현 방법은 Gomaa처럼 UML의 스테레오타입을 확장한 것은 동일하지만, Gomaa가 다양한 스테레오타입 표기법을 정의한 것과 달리 "<<v.p>>"라는 한 가지 스테레오타입을 사용하여 표기법을 단순화하였다. 따라서 Moon et al.의 표현 방법을 적용한 아키텍처 모델이 Gomaa의 표현 방법을 적용한 아키텍처 모델보다 간결해 보이지만, 그만큼 표현력이 부족하다. Moon et al.의 표현 방법은 Gomaa의 표현 방법과 달리 컴포넌트 내부의 가변성을 표현할 수 없고, Alternative 가변성을 표현할 수 없다(Gomaa의 표현 방법에서 "<<optional>>"은 선택적 가변성을, "<<vp>>"는 택일적 가변성을 표현할 수 있지만; Moon et al.의 표현 방법에서 "<<v.p>>"는 선택적 가변성만 표현할 수 있다). 또한 Gomaa의 표현 방법과 마찬가지로 UML의 스테레오타입에 가변 컴포넌트에 관련이 있는 휘처를 함께 표현하지 않기 때문에, 아키텍처 모델만 보고 각 가변성에 대응되는 휘처를 파악하기 어렵다. Moon et al.의 표현 방법은 다른 표현 방법들과 달리 제품라인 컴포넌트 바인딩(Domain Component Binding)에 CV_property를 정의하여, 가변적인 컴포넌트 사이의 직접적인 또는 간접적인 컴포넌트 바인딩을 고려한 것이 강점이다.

3.4. Czarnecki et al.의 표현 방법

Czarnecki et al.[9]도 Gomaa[6, 7]나 Moon et al.[8]처럼 UML 모델의 스테레오 타입을 확장하여 제품라인 아키텍처의 가변성을 표현하였다. 하지만 Gomaa나 Moon et al.의 표현 방법과는 달리, 스테레오 타입에 가변성의 타입이 아닌 휘처의 이름을 표시하였다.

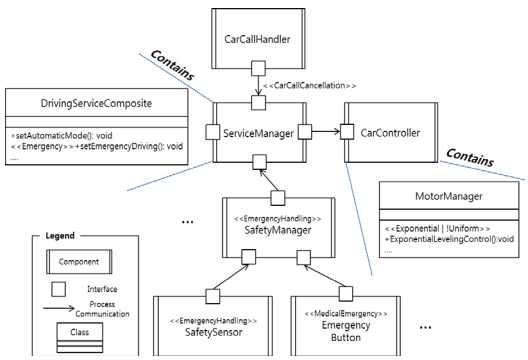

그림 7. Czamecki et al..의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 7은 Czarnecki et al.의 표현 방법으로 ECS 제품라인의 프로세스 아키텍처 모델의 가변성을 나타낸 것이다. 프로세스를 나타내는 각 컴포넌트/커넥터 중 가변 컴포넌트/커넥터는 맵핑되는 가변 휘처 이름을 스테레오타입에 정의하고 있다. 그림 7에서 SafetyManager 컴포넌트는 Emergency Handling 휘처가 선택되었을 때

제품에 포함이 되고, EmergencyButton 컴포넌트는 Medical Emergency 휘처가 선택되었을 때 제품에 포함이 되는 것을 알 수 있다. 또한 UML 모델의 모든 요소(Element)와 관계(Relationship)에 스테레오타입으로 관련된 휘처의 이름을 표시할 수 있도록 하였기 때문에 컴포넌트 뿐 아니라 클래스와 속성, 함수 단위까지 가변성을 표현할 수 있다. ServiceManager 컴포넌트 안에 들어있는 DrivingServiceComposite 클래스의 setEmergencyDriving() 함수는 Emergency 휘처가 선택되었을 때 제품에 포함이 된다. Czarnecki et al.의 표현 방법은 다른 표현 방법과 다르게 스테레오타입에 휘처의 이름으로 논리 연산(Logical Operation)을 표현할 수 있다는 특징이 있다. 그림 7에서 CarController 컴포넌트 안에 들어있는 MotorManager 클래스의 Exponential LevelingControl() 함수는 "<<Exponential ¦ !Uniform>>"을 통하여 Exponential 휘처가 선택되거나 또는 Uniform 휘처가 선택되지 않을 때 제품에 포함이 된다는 것을 알 수 있다.

Czarnecki et al.의 표현 방법의 강점은 가변적인아키텍처 컴포넌트/커넥터에 가변 휘처를 표현하기 때문에 제품라인 아키텍처 모델만 보고서도 아키텍처의 어느 가변 부분이 어느 가변 휘처와 연관이 되어 있는지를 알 수 있다는 점이다. 또한 휘처 사이의 논리 연산을 이용하여, 하나 이상의 가변 휘처의 선택 관계에 의해 제품에 포함 여부가 결정되는 아키텍처 컴포넌트/커넥터를 표현할 수 있다는 것이 강점이다. 그리고 아키텍처 모델에 표현된 논리 연산을 바탕으로 휘처 설정(Configuration)에 따라 아키텍처 모델의 생성을 검증하는 규칙을 함께 제안하고 있다.

하지만 아키텍처 컴포넌트/커넥터가 택일적 휘처와 맵핑되는 경우, 택일 휘처 사이의 관계가 아키텍처 모델에 나타나지 않거나 복잡하게 나타난다(택일 휘처 사이의 관계를 휘처 사이의 논리 연산으로 나타낼 경우, 택일 휘처의 수가 늘어날수록 가능한 조합 경우가 늘어나서 논리 연산이 복잡해진다). 또한, Gomaa의 표현 방법과 달리 컴포넌트 내부의 가변성을 나타낼 수 없다.

3.5. Yoon의 표현 방법

Yoon[10]은 UML을 확장하지 않고 그림 8과 같이 독자적인 아키텍처 모델을 정의 하였다. 가변성의 대상은 다양한 아키텍처 모델의 컴포넌트와 커넥터였는데, 그림 8에서처럼 "선택 또는 택일"과 "컴포넌트 또는 커넥터"의 조합에 따라서 도형의 색과 모양을 다르게 나타내어 이를 구분하였다.

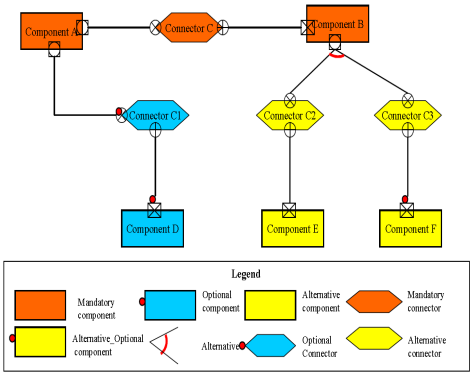

그림 8. Yoon의 표현 방법

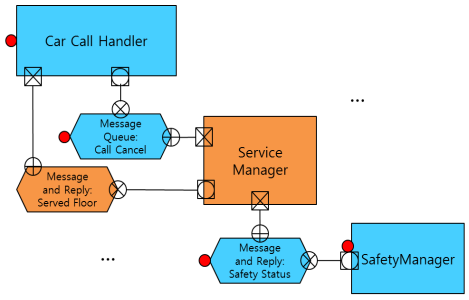

그림 9. Yoon의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 9는 Yoon의 표현 방법을 사용하여 ECS 제품라인의 프로세스 아키텍처의 가변성을 나타낸 것이다. 그림에서 Service Manager 컴포넌트와 Message and Reply: Served Floor 커넥터는

제품라인의 모든 제품에 포함되는 필수 컴포넌트/커넥터이고, Car Call Handler 컴포넌트와 Message Queue: Call Cancel 커넥터는 제품에 따라서 선택적으로 포함되는 컴포넌트/커넥터임을 알 수 있다.

Yoon의 방법의 강점은 커넥터의 가변성을 명시적으로 표현한다는 점과, 컴포넌트/커넥터가 필수적인지, 선택적인 가변성인지, 필수적인 가변성인지의 여부를 한 눈에 알아보기 쉽게 색으로써 명확하게 구분한다는 점이다. 하지만 이 방법은 색이 단색으로 표현될 수밖에 없는 경우에는 사용할 수 없고, 보편적이지 않은 아키텍처 모델 표현 방법이며, 이를 지원하는 도구가 없다는 약점이 있다. 또한, 컴포넌트 내부의 가변성을 표현할 수 없고, 가변 컴포넌트/커넥터에 대응되는 휘처가 무엇인지를 아키텍처 모델만 보고서 알 수 없다. 그리고 하나 이상의 커넥터가 택일적으로 선택되는 관계임을 나타낼 수 있으나, 하나 이상의 컴포넌트가 택일적으로 선택되는 관계는 나타내기 어렵다(즉, 어느 컴포넌트가 택일적으로 선택되는 컴포넌트인지는 표현할 수 있지만, 어떤 컴포넌트와 택일적인 관계인지는 표현할 수 없다).

3.6. Jang의 표현 방법

Jang[11]은 UML의 모델 중 컴포넌트 모델만을 확장하여 제품라인 아키텍처 모델을 표현하였다. Jang은 UML의 컴포넌트 모델 중 컴포넌트, 인터페이스, 그리고 컴포넌트 내부 구현을 가변성의 대상으로 하였다.

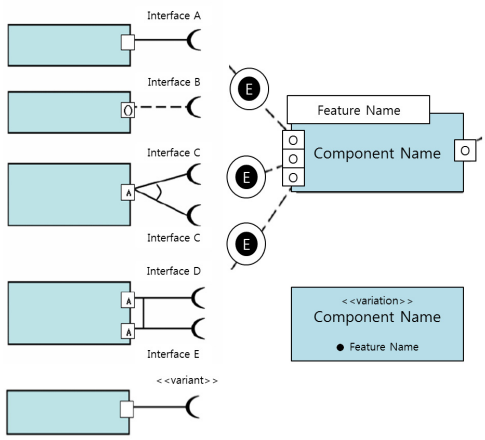

그림 10. Jang의 표현 방법

그림 10은 Jang의 표현 방법을 나타낸 그림이다. 컴포넌트 자체가 제품에 포함되거나 포함되지 않는 가변성은 컴포넌트 위에 해당 컴포넌트와 연관된 가변 휘처의 이름을 적어서 표현한다. 인터페이스가 바뀌는 가변성은 “<<variant>>”로 표현하고, 인터페이스가 선택적 또는 택일적으로 제품에 포함되는 여부를 “O” 또는 “A”로 나타낸다. 제품에 따라서 컴포넌트 내부가 변하는 경우는 컴포넌트에 “<<variation>>” 스테레오타입을 추가하고, “● 대응되는 가변 휘처 이름”을 컴포넌트 하단에 명시한다.

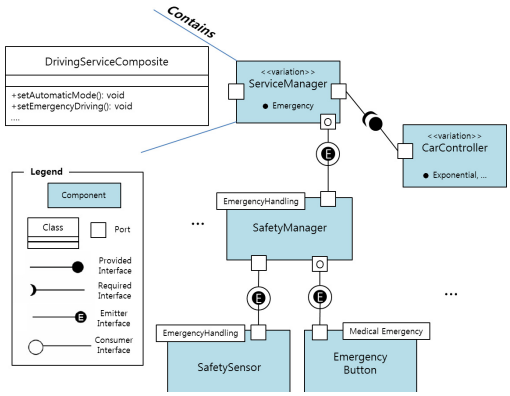

그림 11. Jang의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 11은 Jang의 표현 방법을 사용하여 ECS 제품라인의 프로세스 아키텍처의 가변성을 나타낸 것이다. 그림에서 SafetyManager 컴포넌트는 Emergency Handling 휘처에 대응되고, Emergency Button 컴포넌트는 Medical Emergency 휘처에 대응되는 것을 알 수 있다. 그리고 ServiceManager 컴포넌트는 내부의 구현이 Emergency 휘처의 선택에 따라 변하는 것을 알 수 있다(ServiceManager 컴포넌트 내부에 있는 DrivingServiceComposite 클래스의 setEmergencyDriving() 함수는 Emergency 휘처가 선택된 경우에만 제품에 포함된다).

Jang의 표현 방법은 Yoon의 방법과 마찬가지로 커넥터의 선택적/택일적 가변성을 명시적으로 표현한다는 것이 강점이다. 또한 컴포넌트 또는 컴포넌트 내부의 가변성에 대응되는 가변 휘처가 무엇인지 나타낼 수 있으며, 컴포넌트 내부의 가변성도 표현이 가능하다.

하지만 인터페이스의 가변성에 대응되는 가변 휘처를 나타내지 못한다. 그리고 컴포넌트 또는 컴포넌트 내부의 가변성에 대응되는 가변 휘처가 선택 휘처인지 택일 휘처인지 알기가 어렵다. 또한, UML의 표현 방법을 확장하였지만, UML에서 제공하지 않는 표현 방법(예를 들어 컴포넌트 위에 네모 박스로 휘처 이름을 나타내는 것)을 사용하였기 때문에 일반적인 UML 도구를 사용할 수 없다.

3.7. Lee et al.의 표현 방법

Lee et al.[12]은 Gomaa[6, 7], Moon et al. [8], Czarnecki et al.[9]의 표현 방법처럼 UML 스테레오타입을 확장하여 제품라인 아키텍처 모델의 컴포넌트/커넥터의 가변성을 표현하였다. Lee et al.의 표현 방법의 특징은 가변 휘처에 대응되는 컴포넌트/커넥터의 스테레오타입에 휘처의 타입(선택 휘처 또는 택일 휘처), 대응 방식(컴포넌트/커넥터의 전체에 대응 또는 내부 구현에 대응)과 이름을 함께 나타낸다는 점이다.

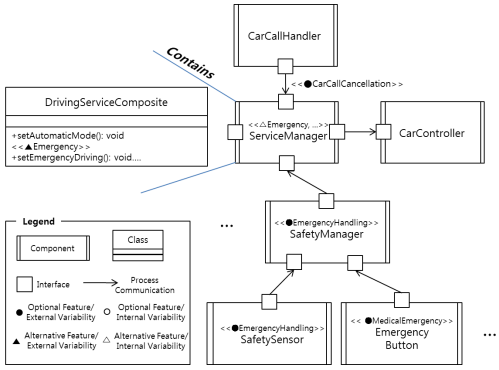

그림 12. Lee et al.의 표현 방법으로 가변성을 나타낸 ECS 제품라인의 프로세스 아키텍처 모델

그림 12는 Lee et al.의 표현 방법으로 ECS 제품라인의 프로세스 아키텍처 모델의 가변성을 나타낸 것이다. Lee et al.의 표현 방법은 휘처가 선택 휘처일 경우는 원을, 가변 휘처일 경우는 삼각형을 사용하는 휘처의 이름 앞에 나타낸다. 그리고 휘처가 컴포넌트/커넥터 전체에 대응되는 경우에는 검은색으로 채워진 도형을, 컴포넌트/커넥터 일부에 대응되는 경우에는 흰색으로 채워진 도형을 사용한다. 그림 12에서 SafetyManager 컴포넌트는 Emergency Handling 휘처에 대응되며, “<<●EmergencyHandling>>”을 통하여 Emergency Handling 휘처가 선택 휘처(도형이 원이기 때문)이고, SafetyManager 컴포넌트 전체에 대응된다는 것(도형이 검은색으로 채워졌기 때문)을 알 수 있다. ServiceManager 컴포넌트는 Emergency 휘처에 대응되며, “<<△Emergency>>”를 통하여 Emergency 휘처가 택일 휘처이고(도형이 삼각형이기 때문), ServiceManager 컴포넌트의 내부 구현에 대응된다(도형이 흰색으로 채워졌기 때문)는 것을 알 수 있다. ServiceManager 컴포넌트 안에 들어있는 DrivingServiceComposite 클래스의 setEmergencyDriving() 함수는 Emergency 휘처가 선택되었을 때 제품에 포함이 된다.

Lee et al.의 표현 방법의 강점은 제품라인 아키텍처 모델만 보고도 각 가변 컴포넌트/커넥터에

대응되는 휘처의 이름과 종류, 대응 방식을 파악할 수 있다는 것이다. 또한 휘처의 종류와 대응 방식을 나타내는 도형의 모양과 색을 사용하여 나타내어 표현 방법이 간결하면서도 쉽게 파악하기가 쉽다.

하지만 Lee et al.의 표현 방법은 표현할 수 있는 정보가 많은 만큼 가변 휘처의 수가 많아지면 아키텍처 모델이 복잡해진다는 약점이 있다. 컴포넌트 내부에 대응되는 휘처를 모두 표시해야 하기 때문에, 추상화가 높은 아키텍처 컴포넌트인 경우 해당 컴포넌트에 대응되는 모든 정제된 (Refined) 컴포넌트의 가변성을 내부 가변성으로 표시하면 아키텍처 모델이 복잡해질 수 있다.

4. 소프트웨어 제품라인 아키텍처 모델에서의 가변성 표현 방법 비교

지금까지 살펴본 제품라인 아키텍처 모델에서의 가변성 표현 방법을 요약하여 표로 나타내면 표 1과 같다. 여덟 가지 항목(아키텍처 모델 자체에 가변성 표현, UML 모델 사용/확장, 휘처 이름 표현, 휘처 타입 표현, 내부가변성 표현, 도구 지원, 휘처 선택 관계 표현, 표현할 수 있는 가변성 범위)에 대하여 각 표현 방법을 비교하였다.

독립적인 가변성 모델을 관리하는 Thiel et al.의 표현 방법을 제외하고는 모두 아키텍처 모델 자체에 가변성을 표현할 수 있었다. Thiel et al.과 Yoon의 표현 방법은 독자적인 아키텍처 모델을 사용하였다. UML의 스테레오타입을 확장한 표현 방법들(Gomaa, Moon et al., Czarnecki et al., Lee et al.의 표현 방법)은 기존 UML 도구를 사용하여 제품라인 아키텍처에 가변성을 표현할 수 있었다. Czarnecki et al., Jang, Lee et al.의 표현 방법은 가변 컴포넌트/커넥터에 대응되는 휘처의 이름을 표현할 수 있었고, Gomaa, Yoon, Jang, Lee et al.의 표현 방법은 가변 컴포넌트/커넥터에 대응되는 휘처의 타입을 표현할 수 있었다. Gomaa, Jang, Lee et al.의 표현 방법은 컴포넌트/커넥터의 내부 가변성을 표현할 수 있었고, Czarnecki et al.의 표현 방법은 휘처 선택 관계를 논리 연산으로 표현하여 이를 가변 컴포넌트/커넥터와 대응시킬 수 있었다. UML의 스테레오 타입을 확장한 Gomaa, Moon et al., Czarnecki et al., Lee et al.의 표현 방법은 UML 모든 단위에 가변성을 표현할 수 있었고, Thiel et al.은 각 아키텍처의 컴포넌트에, Yoon은 각 아키텍처의 컴포넌트/커넥터에, Jang은 컴포넌트 아키텍처의 컴포넌트/인터페이스에 가변성을 표현할 수 있었다.

5. 결론 및 향후 연구

본 논문에서는 현재까지 제안된 제품라인 아키텍처 모델에서의 가변성 표현 방법을 분석하고 엘리베이터 컨트롤 시스템 제품라인 예제에 적용하여 강약점을 파악하였다. 또한 분석 결과를 정리하여 각 표현 방법의 특징을 비교하였다. 비교 결과 각 표현 방법마다 표현할 수 있는 정보가 달랐다. 표현할 수 있는 정보의 범위가 작으면 아키텍처 모델이 간결했지만 아키텍처 모델을 통해 파악할 수 있는 정보가 적었고, 범위가 넓으면 아키텍처 모델이 복잡해졌다. 이를 해결하기 위해 어떤 표현 방법은 색깔이나 도형을 사용하여 많은 정보를 표현하면서 복잡성을 줄이려고 노력하였다. 많은 표현 방법이 UML 모델을 확장하였으며, UML 모델을 확장한 경우 독자적인 아키텍처 모델을 제시한 경우보다 활용도가 높고 도구 지원이 가능하다는 것이 강점이었다.

각 제품라인 아키텍처 모델에서의 가변성 표현 방법마다 강약점과 표현 범위가 다르기 때문에

표 1 제품라인 아키텍처 모델에서의 가변성 표현 방법 비교표

	독립적 가변성 모델 이용 [3, 4]	Gomaa [6, 7]	Moon et al. [8]	Czarnecki et al.[9]	Yoon[10]	Jang[11]	Lee et al.[12]
아키텍처 모델 자체에 가변성 표현	X	O	O	O	O	O	O
UML 모델 사용/확장	X	O	O	O	X	O	O
휘처 이름 표현	X	X	X	O	X	O	O
휘처 타입 (선택/택일) 표현	X	O	X	X	O	O	O
내부 가변성 표현	X	O	X	X	X	O	O
도구 지원	X	O	O	O	X	X	O
휘처 선택 관계 표현	X	X	X	O	X	X	X
표현할 수 있는 가변성 단위	각 아키텍처의 컴포넌트	UML 모든 단위	UML 모든 단위	UML 모든 단위	각 아키텍처의 컴포넌트/ 커넥터	컴포넌트 아키텍처의 컴포넌트/ 인터페이스	UML 모든 단위

사용하려는 목적에 적합한 표현 방법을 선택하는 것이 중요할 것이다. 하지만 휘처 기반의 제품라인에서는 휘처와 제품라인 아키텍처 자산 사이의 쉬운 추적성을 위해 휘처 이름 표현과 휘처 타입 표현이 필요할 것으로 생각된다. 또한 가변성이 있을 수 있는 컴포넌트/커넥터 전체/내부의 가변성을 모두 표현할 수 있는 표현 방법이 적합할 것이다. 또한 표현 방법을 실제로 적용하고 사용하기 위해 도구(UML 도구 또는 그 외 독자적인 도구) 지원이 가능한 표현 방법을 택하는 것이 필요하다.

본 논문에서는 제품라인 아키텍처 모델에서의 가변성 표현 방법의 표현력에 초점을 맞추어 표현 방법을 비교하였지만, 제품라인 아키텍처 모델에 가변성을 언제 바인딩(Binding)할 것인가에 대한 결정 또한 중요하다. 가변성을 제품라인 아키텍처 모델에 일찍 바인딩하여 가변성을 아키텍처 모델 안에서 관리할 경우, 가변성 휘처가 바뀔 때마다 아키텍처 모델이 계속 바뀌어야 하고 표현된 가변성의 일관성을 검증하여야 한다. 제품라인 아키텍처 모델과 가변성을 따로 관리하고 가변성을 아키텍처 모델에 늦게 바인딩 할수록, 제품라인 아키텍처는 가변성의 변화에 유연해진다. 향후에는 제품라인 아키텍처 모델에 가변성을 늦게 바인딩하여 유연한 제품라인 아키텍처를 갖는 방법을 연구할 것이다.

참 고 문 헌

[1] V. Sugumaran, S. Park, and K.C. Kang, “SPECIAL ISSUE: Software Product Line Engineering,” Communications of the ACM(CACM), Vol. 49, Issue. 12, pp.28-32, December 2006.

[2] F. Van der Linden, K. Schmid, and E. Rommes, “Software product lines in action: the best industrial practice in product line Engineering,” Springer-Verlag New York Inc., 2007.

[3] S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design," 2nd International Software Product Line Conference (SPLC 2002), LNCS 2379, pp. 130-153, 2002.

[4] K. Pohl, G. Böckle, and F. van der Linden, "Software Product Line Engineering: Foundations, Principles, and Techniques," Springer-Verlag New York Inc., 2005.

[5] H. Gomaa, "A Software Design Method for Real-time Systems," Comunications of the ACM, Volume 27, Issue 9, Sept. 1984.

[6] H. Gomma, "Designing Software Product Lines with UML, From Use Cases to Pattern Based Software Architectures," Addison-Wesley, 2004.

[7] H. Gomma, "Designing Software Product Lines with UML," Addison Wesley, 2005.

[8] M. Moon, H.S. Chae, and K. Yeom, "A Metamodel Approach to Architecture Variability in a Product Line," International Conference on Software Reuse (ICSR 2006), LNCS 4039, pp. 115-126, 2006.

[9] K. Czarnecki and K. Pietroszek, "Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints," 5th International Conference on Generative Programming and Component Engineering (GPCE 2006), October 22-26, 2006, Portland, Oregon, USA.

[10] C. Yoon, "Verification of Syntactic Consistency between Feature Model and Product line Architecture Model (휘처 모델과 프로덕트 라인 아키텍처 모델 간의 문법적 일관성 검증)," Master Thesis, 2003.

[11] Y. Jang, "Component architecture specification method for software product line engineering (소프트웨어 프로덕트 라인 공학에서의 컴포넌트 아키텍처 명세 방법)," Master Thesis, 2007.

[12] H. Lee, H. Choi, K.C. Kang, D. Kim, and Z. Lee, "Experience Report on Using a Domain Model-Based Extractive Approach to Software Product Line Asset Development," In: Proceedings of the 11th International Conference on Software Reuse (ICSR 2009), 2009, pp. 149-161.

저 자 소 개

이 혜 선

2009년 포항공과대학교 컴퓨터공학과 졸업(학사).
2009년~현재 포항공과대학교 컴퓨터공학과 대학원 통합과정.

<관심분야> 소프트웨어 재사용, 소프트웨어 제품 라인 공학, 추출식 제품라인 접근, 휘처 기반 소프트웨어 개발 방법

조 성 배

2010년 동국대학교 컴퓨터공학과 졸업(학사)

2010년~현재 포항공과대학교 정보전자융합공학부 석사과정.

〈관심분야〉 소프트웨어 재사용, 소프트웨어 제품라인 공학, 추출식 제품라인 접근

강 교 철

1973년 고려대학교 통계학과 졸업(학사).
1974년 콜로라도대학교 산업공학 졸업(석사).
1982년 미시간대학교 소프트웨어공학 졸업(박사).
1982년~1984년 미시간대학교 Visiting Professor.
1984년~1985년 (미국)벨 통신 연구소 Technical Staff.
1985년~1987년 AT&T 벨 연구소 Technical Staff.
1987년~1992년 카네기멜론대학교 Project Leader.
1992년~현재 포항공과대학교 교수.
2000년~2001년 카네기멜론대학교 Visiting Scientist.

〈관심분야〉 소프트웨어공학, 실시간 내장형 시스템, 소프트웨어 재사용, 소프트웨어 제품라인 공학.