컴포넌트 코드 생성을 통한 컴포넌트 기반 제품 라인에서의 가변성 지원☆

Variability Support in Component-based Product Lines using Component Code Generation

최 승 훈*
Seung Hoon Choi

요 약

소프트웨어 제품 라인은 소프트웨어 자산에 존재하는 재구성 가능한 컴포넌트를 구체화하고 미리 정의된 아키텍처를 기반으로 조립함으로써 고품질의 응용 프로그램을 효율적으로 개발하기 위한 패러다임이다. 최근 컴포넌트 기반의 여러 가지 제품 라인 개발 방법론들이 제안되었지만, 가변성 지원 컴포넌트의 구체적인 구현 기술은 제공하지 못하고 있다. 한편, 가변성을 지원하기 위한 여러 가지 구현 기술이 개발되었지만, 이러한 구현 기술은 개발 초기부터 가변성을 고려한 체계적인 분석 및 설계 방법을 제공하지 못한다. 본 논문에서는, UML 모델링 기법을 확장한 컴포넌트 기반 제품 라인 개발 방법론인 PLUS와 컴포넌트 자동 생성 기법을 결합하여 특정 제품 생산의 효율성을 높이는 기법을 제안한다. 본 논문에서의 컴포넌트는 가변성을 지원하는 구현 부품들이 계층 구조를 이루며 각 구현 부품들은 XSLT 스크립트로 작성된다. 특성 모델에서 선택된 특성들로부터 개발자가 원하는 컴포넌트의 코드가 자동 생성되며, 마이크로웨이브 오븐 제품 라인을 사례 연구로 해서 가변성 지원 컴포넌트의 개발 프로세스를 살펴본다.

Abstract

Software product-lines is the software development paradigm to attain the rapid development of quality applications by customizing the reconfigurable components and composing them based on predefined software architectures. Recently various methodologies for the component-based product lines are proposed, but these don't provide the specific implementation techniques of the components in terms of variability resolution mechanism. In other hand, the several approaches to implement the component supporting the variabilities resolution are developed, but these don't define the systematic analysis and design method considering the variabilities from the initial phase. This paper proposes the integration of PLUS, the one of product line methodologies extending UML modeling, and component code generation technique in order to increase the efficiency of producing the specific product in the software product lines. In this paper, the component has the hierarchical architecture consisting of the implementation elements, and each implementation elements are implemented as XSLT scripts. The codes of the components are generated from the feature selection. Using the microwave oven product lines as case study, the development process for the reconfigurable components supporting the automatic variability resolution is described.

☞ Keyword : Software Product Lines, Product Line Architecture, Component, Generative Programming

1. 서론 및 연구 배경

소프트웨어 제품 라인은 소프트웨어 자산에 존재하는 일반적인 컴포넌트를 구체화하고 미리 정의된 아키텍처를 기반으로 조립함으로써, 고품질의 응용 프로그램을 빠르게 개발하기 위한 패러다임이다[1]. 이러한 제품 라인 패러다임을 지원하는 여러 가지 소프트웨어 개발 방법론이 개발되었다[2-4]. 이러한 방법론들의 목적은 도메인 공학을 통해서 개발 초기에 소프트웨어 제품 라인에 존재하는 멤버들 사이의 공통점과 차이점을 명확하게 식별하여 응용 프로그램들 사이의 가변성을 지원

* 정 회 원 : 덕성여자대학교 컴퓨터학부 교수
csh@duksung.ac.kr(제 1저자)
[2005/04/14 투고 - 2005/04/20 심사 - 2005/05/06 심사완료]
☆ 본 연구는 2004학년도 덕성여자대학교 연구비지원으로 이루어졌음.

하는데 있다.

최근에는 컴포넌트 기반의 제품 라인 개발 방법론이 다양하게 제안되었다 [5-8]. 이러한 방법론들의 핵심은, 제품 라인 개발 시 소프트웨어 자산을 제품 라인 소프트웨어 아키텍처와 컴포넌트들로 모델링하는데 있다. 특정 어플리케이션 개발 시에는 미리 구축된 컴포넌트들을 목적에 맞게 재구성하고 조립함으로써 비슷한 특성을 가지고 있지만 특정 부분이 다른 일련의 다양한 소프트웨어들을 빠르게 생산하고자 하는 것이다.

이러한 개발 방법론이 효과적으로 동작하기 위해서는 제품 라인 아키텍처와 컴포넌트들이 '가변성(variability)'을 지원해야 한다. 가변성이란, 공통점을 공유하는 일련의 소프트웨어 부품들이 각 환경이나 목적에 맞게 변형될 필요가 있는 특성을 의미한다. 이는 특정 소프트웨어 개발 시 그 소프트웨어가 실행되는 환경과 목적에 맞도록 그 기능과 구조를 쉽게 재구성할 수 있는 능력을 의미한다.

소프트웨어 제품 라인에서 구체적인 시스템을 생산하기 위한 기법으로 여러 가지가 존재한다. 재사용자의 많은 개입이 필요한 조립 방법, 재사용자의 약간의 개입이 필요한 반자동식 조립 방법, 재사용자가 제시한 차이점을 바탕으로 한 자동 생성 방법 등이 그것이다. 특히, 프로그램 자동 생성에 관한 연구가 최근 활발히 진행되어, 여러 가지 프로그램 자동 생성을 위한 분석 방법과 구현 방법들이 제안되었다[9].

특히 본 연구와 관련해서, 다음과 같은 연구들이 있다. GenVoca[10]는 특성들을 점진적으로 추가함으로써 복잡한 시스템을 생성해낼 수 있다는 'step-wise refinement' 개념을 적용하여 새로운 모듈을 추가하거나 제거함으로써 확장 가능한 소프트웨어를 개발하기 위한 설계 방법론이다. AHEAD [11]는 이러한 개념을 프로그램 코드 뿐 만 아니라 다른 소프트웨어 도큐먼트까지로 확장하고 조립 연산자도 여러 가지가 가능하도록 한 방법론이다. XVCL[12]은 가변점(variation point)을 지원하는 x-frame이라는 기본 요소를 트리 형태의 계층 구조로 구조화함으로써 구현되며, 특정 제품을 위한 명세서(SPC)를 작성하여 적용하면 XVCL 프로세서라는 전용 프로세서가 트리 구조 형태로 존재하는 x-frame들을 방문하면서 가변점들을 해결하고 이들을 조립하여 최종 제품을 생산한다. 컴포넌트 기반 제품 라인 개발 방법론의 하나인 FORM 개발 방법론에서도 ASADAL/FORM [13]이라는 도구를 제공하여 코드 자동 생성을 지원한다.

이러한 방법론은 가변성 해결 방법, 가변성 관리 기법, 확장성, 유지 보수성 등의 관점에서 여러 가지 장점들을 제공하지만, 개발 초기부터 가변성을 명확히 모델링하는 컴포넌트 기반의 체계적인 개발 방법론을 제공하지 못한다. 본 연구와 관련해서, GenVoca/AHEAD 및 XVCL, FORM과의 차이점에 관하여 제 6 장에서 상세히 기술하였다.

본 논문에서는 제품 라인 컴포넌트가 가지고 있는 가변점을 자동으로 커스터마이징함으로써 특정 제품을 자동으로 생성 및 조립 가능하도록 하기 위한 기법으로서, 컴포넌트 기반 제품 라인 개발 방법론인 PLUS[7]와 계층 구조 및 XML 기반의 컴포넌트 자동 생성 기법의 결합을 제안한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 PLUS와 제품 라인 소프트웨어 아키텍처에 대해서 기술한다. 제 3 장에서 특성 모델 및 컴포넌트 기반 프로덕트 라인 아키텍처에서의 가변성을 설명하고, 제 4 장에서 마이크로 오븐 제품 라인을 예제로 하여 자동 생성을 지원하는 컴포넌트 개발 기법을 설명한다. 제 5 장에서 컴포넌트 자동 생성 과정을 살펴본 후, 제 6 장에서 관련 연구와의 비교를 기술하고, 제 7 장에서 결론 및 향후 연구를 제시한다.

2. PLUS(Product Line UML based Software Engineering)

2.1 PLUS 제품 라인 개발 방법론

PLUS[7]는 하나의 시스템 개발에 주로 사용되

는 UML 기반 모델링 기법을 확장하여, 소프트웨어 제품 라인 개발을 지원하기 위한 여러 가지 개념과 기법을 제공한다. PLUS의 목적은, 소프트웨어 제품 라인에 존재하는 제품 멤버들 사이의 공통점(commonality)과 차이점(variability)을 개발 초기 단계에서부터 명확하게 모델링하는데 있다. 본 절에서 PLUS 개발 방법론에 포함된 여러 가지 단계 및 모델링을 간략히 설명한다.

· **소프트웨어 제품 라인 요구 사항 모델링**

- *사용 사례 모델링:* 사용 사례 모델에서의 공통점과 차이점을 모델링하기 위해, PLUS는 사용 사례들을 핵심적 사용 사례(kernel use case), 선택적 사용 사례(optional use case), 택일적 사용 사례(alternative use case) 등으로 분류하였으며, 사용 사례 안에 가변점(variation point)을 정의할 수 있도록 한다.
- *특성 모델링*: 특성 모델은 소프트웨어 제품 라인 개발에 있어서 핵심적인 모델이다. PLUS 는 공통적 특성(common feature), 선택적 특성(optional feature), 택일적 특성(alternative feature)을 UML 표기법을 통해서 모델링하며, 사용 사례 모델로부터 특성 모델을 유도할 수 있는 방법을 제공한다.

· **소프트웨어 제품 라인 분석 모델링**

- *정적 모델링*: 개발하고자 하는 제품 라인의 경계를 위한 컨텍스트 모델(context model)을 정의하고, 핵심적, 선택적 및 택일적 외부 클래스를 결정한다. 또한, 제품 라인 정보 모델을 정의하고, 핵심적, 선택적 및 택일적 엔터티 클래스를 결정한다.
- *동적 상호작용 모델링*: 핵심적, 선택적 및 택일적 사용 사례를 실현하기 위한 상호 작용 다이어그램을 개발한다. 점진적인 개발 방법을 사용하여, 먼저 핵심적 사용 사례를 먼저 개발한 후 차이점을 지원하도록 확장한다.
- *동적 상태 기계 모델링*: 핵심적, 선택적 및 택일적 상태 차트(statechart)를 개발한다. 상속(inheritance)과 인자화(parameterization)를 통해 상태 기계 가변성을 관리한다.
- *특성/ 클래스 의존성 모델링*: 공통적, 선택적 및 택일적 특성들이 어떻게 핵심적, 선택적 및 가변적 클래스에 영향을 미치는지 그 관계를 정의한다.

· **소프트웨어 제품 라인 설계 모델링**

- *소프트웨어 아키텍처 패턴 결정*: 아키텍처 패턴 카탈로그를 참조하여, 개발하고자 하는 소프트웨어 제품 라인에 가장 적합한 소프트웨어 아키텍처 구조 및 통신 패턴을 결정한다.
- *컴포넌트 기반 소프트웨어 설계*: 제품 라인을 위한 컴포넌트 기반 소프트웨어 설계를 개발한다. 이는, 핵심적, 선택적 및 가변적 컴포넌트와 컴포넌트 사이의 관계를 모델링하며 각 컴포넌트의 포트(port)와 제공 인터페이스 및 필요 인터페이스를 정의한다.

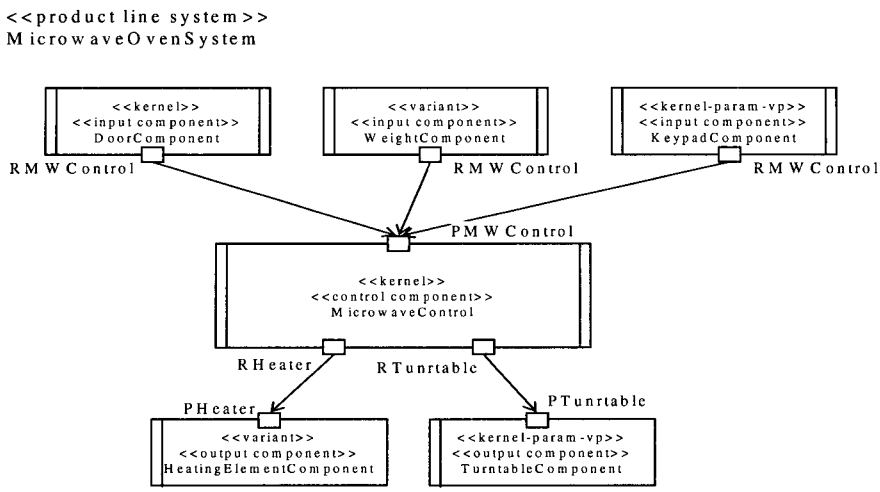
· **소프트웨어 응용 공학(Software Application Engineering)**

특성 모델로부터 목표로 하는 어플리케이션이 포함해야 할 특성을 선택하고 이를 이용하여, 제품 라인 아키텍처와 컴포넌트로부터 특정 소프트웨어 응용을 개발한다.

2.2 컴포넌트 기반 소프트웨어 제품 라인 아키텍처

본 논문의 컴포넌트 자동 생성 기법은 PLUS 방법론의 설계 산출물을 출발점으로 한다. 본 절에서는 주요 설계 산출물인 컴포넌트 기반 제품 라인 아키텍처를 기술하고 이 절에서 제시된 마이크로웨이브 오븐 제품 라인을 예제로 사용하여 제 4 장에서 자동 생성 기법을 지원하기 위한 컴포넌트 개발 과정을 설명한다.

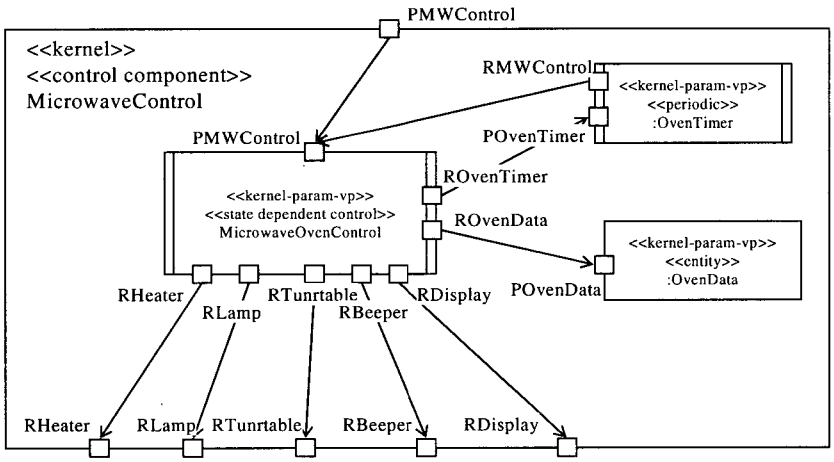
제품 라인을 위한 컴포넌트 기반 소프트웨어 아키텍처는 컴포넌트들을 식별하고 컴포넌트들 사이의 인터페이스를 정의함으로써 개발된다. 컴포넌트는 다시 여러 개의 부컴포넌트(subcomponent)들로 분해될 수 있다. 모든 컴포넌트는 포트(port)를 통해서 다른 컴포넌트들과 통신한다. 각 포트



〈그림 1〉 마이크로웨이브 오븐 제품 라인 소프트웨어 아키텍처

는 제공 인터페이스(provided interface)와 필요 인터페이스(required interface)를 가지고 있다. 하나의 컴포넌트는 하나 이상의 인터페이스를 가질 수 있으며, 컴포넌트들은 두 포트를 연결하는 커넥터(connector)에 의해 통신한다.

그림 1은 마이크로웨이브 오븐 제품 라인의 컴포넌트 기반 소프트웨어 아키텍처와 컴포넌트 인터페이스 및 커넥터의 일부분을 보여주는 그림이다. 예를 들어, MicrowaveControl 컴포넌트는 DoorComponent, WeightComponent, Keypad Component가 필요로 하는 인터페이스를 포함하는 PMW Control 포트를 가지고 있다. 또한, 출력 컴포넌트인 HeatingElementComponent와 Turntable Component에 서비스를 제공하는 인터페이스를 포함하는 RHeater와 RTurntable 포트를 가지고 있다. 그림 2는 복합 컴포넌트(composite component)인 MicrowaveContorl 내부의 소프트웨어 아키텍처를 보여준다.



〈그림 2〉 MicrowaveContorl 복합 컴포넌트의 소프트웨어 아키텍처

3. 컴포넌트 기반 제품 라인에서의 가변성 지원

3.1 특성 모델과 특성 구성

영역 분석의 주요 결과물인 특성 모델(feature model)은 제품군에 속하는 멤버들 사이의 공통점과 차이점에 대한 고수준의 추상화를 제공하는 요구사항 모델이다. 여기에서 '특성'은, 각 소프트웨어 제품들에서 구현되고 테스트되고 배포, 유지되어야 하는 기능적 추상화를 뜻하는 것으로 고객이 중요하게 생각하는 기능이나 요구 사항을 의미한다. 특성 모델에서 제품군에 속하는 멤버들이 공유하는 공통점은 필수적 특성(mandatory feature)으로 표현되며, 멤버들 간의 차이점은 선택적(optional) 및 택일적(alternative) 특성 등으로 표현된다. 이러한 특성 모델은 응용 프로그램 개발에 있어서 '결정 공간(decision space)'을 정의한다[6].

특성 구성이란 특성 모델로부터 재사용자가 결정한 특성 선택 결과를 의미한다[14]. 즉, 특성 모델에 표현되어 있는 제품 라인 멤버들 사이의 차이점들에 대하여 재사용자가 원하는 값을 선택하거나 입력한 결과로서, 재사용시 생성하고자 하는 재품에 대한 요구 사항을 의미한다.

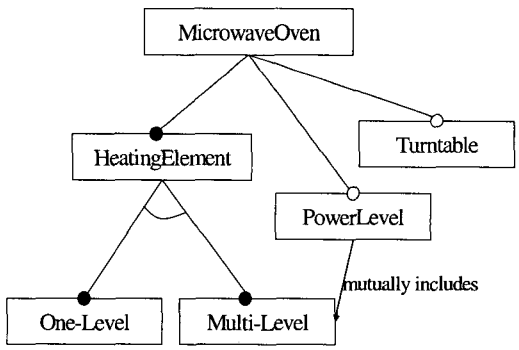
본 논문에서 제안하는 컴포넌트 자동 생성 기법은, 특성 모델을 도메인 공학의 도구로서 뿐 만 아니라 재사용자가 원하는 특정 컴포넌트를 생성하기 위한 요구사항을 입력하기 위한 도구로 사용한다. 본 논문에서 자동 생성 예제로 사용할 두 컴포넌트인 MicrowaveOvenControl 컴포넌트와 HeatingElementComponent 컴포넌트에 영향을 미치는 특성들로만 이루어진 특성 모델을 구성하면 그림 3과 같다. 까만 원은 필수적 특성, 빈 원은 선택적 특성을 의미하며, 원호로 연결된 링크들은 택일적 특성을 의미한다.

본 특성 모델에서 재사용자가 선택한 특성 결과가 특정 제품 생산에 미치는 영향을 분류하면 다음과 같다.

1) 선택적 특성의 선택 여부에 따라, 어떤 컴포넌트가 특정 제품에 포함될 지 말지가 결정된다. 예를 들어, MicrowaveOven.Turntable 특성의 선택여부에 따라, TurntableComponent 컴포넌트가 특정 제품에 포함될 지 말지가 결정된다.
2) 택일적 특성 중 선택된 특성에 따라, 여러 택일적 컴포넌트 중에서 어떤 컴포넌트가 특정 제품에 포함될지가 결정된다. 예를 들어, MicrowaveOven.HeatingElement.Multi-Level 특성이 선택되면, 여러 개의 택일적 컴포넌트 중에서 Multi-LevelHeatingElement 컴포넌트가 특정 제품에 포함된다.
3) 선택된 특성들에 따라, 컴포넌트의 내부 재구성 인자가 결정된다. 예를 들어, MicrowaveOven.HeatingElement.Multi-Level 특성이 선택된다면, 이는 MicrowaveOvenControl 컴포넌트가 여러 단계의 세기를 지원하도록 재구성되어야 한다.

3.2 컴포넌트 기반 제품 라인의 가변성 지원

PLUS로부터 정의된 제품 라인 아키텍처 그림 1, 2에는 <<optional>>, <<variant>> <<kenel-param-vp>> 등의 여러 가지 스테레오 타입을 이용하여 다음과 같은 3가지의 가변성이 표현되어 있다.



〈그림 3〉 마이크로웨이브 오븐 특성 모델

· **선택적 컴포넌트(optional component)**
이것은 하나의 컴포넌트를 특정 제품에 포함시킬 지 말지를 표현하며, <<optional>> 스테레오 타입을 가진다(예: LampComponent 또는 TurntableComponent 컴포넌트). 이러한 컴포넌트들은 제품 라인 구축 시 소프트웨어 자산으로 미리 개발해 놓은 다음, 특정 제품 생산 시 포함 여부를 결정하면 된다. 따라서, 자동 생성 기법을 특별히 적용할 필요가 없다.

· **택일적 컴포넌트(alternative component)**
이는 같은 인터페이스를 제공하는 여러 개의 컴포넌트들 중에서 어떤 하나가 선택되어 특정 제품에 포함될 것인지를 표현하며, <<variant>> 스테레오 타입을 가진다 (예: WeightComponent 또는 HeatingElementComponent 컴포넌트). 택일적 컴포넌트의 경우에는, 여러 개의 컴포넌트들을 각각 독립적으로 구현한 다음, 특정 제품 생산 시 어떤 컴포넌트를 포함시킬 것인지를 결정하면 될 것이다. 그러나, 택일적 컴포넌트의 경우에는 서로 비슷한 기능을 수행하는 경우가 많기 때문에 코드 중복을 최소화하고 각 택일적 컴포넌트들 사이의 차이점을 해결하여 필요한 코드 만을 생성하는 자동 생성 기법의 적용이 필요하다.

· **내부 가변성 지원 컴포넌트**
이는 내부 구성 인자에 따라 컴포넌트 내부의 코드가 재구성되는 컴포넌트를 표현하며, <<XXX-param-vp>> 형태의 스테레오 타입을 가진다. (예: MicrowaveOvenControl 또는 OvenTimer 컴포넌트) 이러한 컴포넌트들의 내부 구성 인자의 값은 특정 제품 생산을 위해 선택된 특성 구성 결과에 따라 결정된다. 이러한 내부 구성을 자동으로 수행한다면 특정 제품 생산 시 생산성이 크게 향상될 것이다.

따라서, 프러덕트 라인 소프트웨어 아키텍처를 구성하는 컴포넌트들 중에서 택일적 컴포넌트와 내부 가변성 지원 컴포넌트에 자동 생성 기법을 적용하는 것이 효과적이다. 본 논문에서는 택일적 컴포넌트인 'HeatingElementComponent'와 내부 가변성 지원 컴포넌트인 'MicrowaveOvenControl'을 예제로 하여 컴포넌트 자동 생성 기법에 대해서 기술한다.

4. 가변성 지원 컴포넌트 개발 프로세스

컴포넌트 기반 제품 라인 아키텍처에서 가변성을 지원하기 위한 컴포넌트 개발 프로세스는 그림 4와 같다.

〈그림 4〉 가변성 지원 컴포넌트 개발 프로세스

4.1 특성/컴포넌트 의존성 식별

먼저, 특성 모델에서의 특성과 컴포넌트들 사이의 의존성 식별을 수행한다. 그림 3의 특성 모델에서 선택된 특성에 따라 영향을 받는 컴포넌트와 어떤 영향을 받는지를 나타내는 표가 표 1이다. 예를 들어, MicrowaveOven.HeatingElement.One-Level 특성이 선택되면, 특정 제품 생산 시 One-Level HeatingElementComponent 컴포넌트가 포함되어야 함을 알 수 있다. 이 표는, PLUS의 '특성/클래스 의존성 모델링'으로부터 유도될 수 있다.

4.2 계층 구조 아키텍처링

본 논문에서의 각 컴포넌트는 GenVoca[10]에서 제안한 step-wise refinement 기법(상속을 이용해서 새로운 특성을 추가)을 활용하여 가변성을

〈표 1〉 특성 / 컴포넌트 의존성

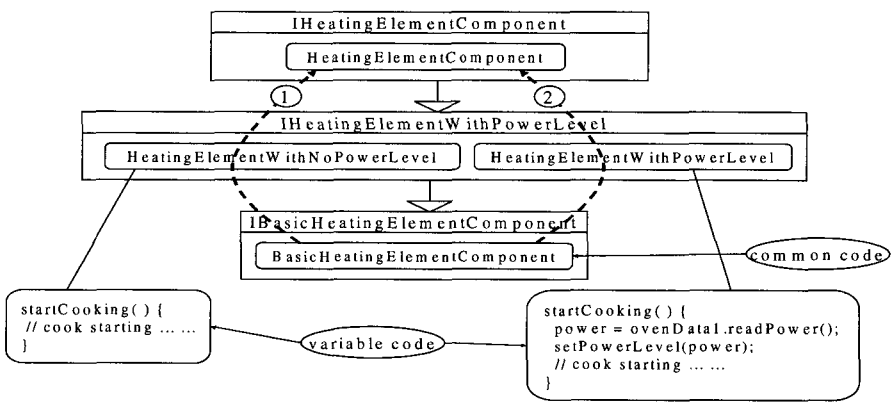
선택된 특성	영향을 받는 컴포넌트	특정 제품 생산 시 미치는 영향
MicrowaveOven.Turntable	TurntableComponent MicrowaveOvenControl	-선택적 TurntableComponent 포함 -컴포넌트 재구성
MicrowaveOven.HeatingElement.One-Level	HeatingElementComponent MicrowaveOvenControl	-택일적 One-LevelHeatingElementComponent 포함 -컴포넌트 재구성
MicrowaveOven.HeatingElement.Multi-Level	HeatingElementComponent MicrowaveOvenControl	-택일적 Multi-LevelHeatingElement Component 포함 - 컴포넌트 재구성
MicrowaveOven.LineUnit.One-Line	KeypadComponent	-컴포넌트 재구성
MicrowaveOven.LineUnit.Multi-Line	KeypadComponent	-컴포넌트 재구성

지원한다. 본 절에서는 택일적 컴포넌트와 내부 재구성 지원 컴포넌트 각각에 대해서 계층 구조 개발 과정을 기술한다.

· **택일적 컴포넌트 (HeatingElementComponent)** 택일적 컴포넌트는 여러 개의 대치 가능한 컴포넌트 중에서 특성 구성에 포함된 특성에 따라서 하나의 컴포넌트가 선택되는 컴포넌트를 의미한다. 택일적 컴포넌트인 HeatingElement Component에 대한 계층 구조를 정의하면 그림 5와 같다.

그림 5에서, 큰 사각형은 하나의 계층을 나타내며 하나의 특성을 대표한다. 각 계층은 공통된 인터페이스를 제공하며, 그 안에는 최종 목표 컴포넌트(target component) 구현에 참여하는 구현 클래스들이 존재한다. 각 계층의 구현 클래스들은 상속 관계를 통해서 조립되며 부모 클래스를 인자로 받아들여서 가변성을 지원하는 Mixin[15] 기법이 활용되었다.

그림 5에서 HeatingElementComponent가 목표 컴포넌트이며, 각 계층의 둥근 사각형들이 목표 컴포넌트의 구현에 필요한 구현 클래스들이다. Basic HeagintElementComponent 클래스에는 마이크로웨이브 오븐의 가열 장치가 기본적으로 가져야 하는 기능들이 구현되어 있다. IHeatingElementWith PowerLevel 계층에는 두 개의 구현 클래스가 존재하며 선택된 특성에 따라서 이 중 하나의 구현 클래

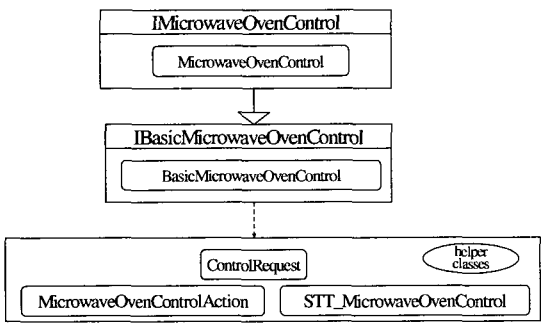


〈그림 5〉 HeatingElementComponent를 위한 계층 구조

〈표 2〉 HeatingElementComponent를 위한 조립 규칙

컴포넌트 카테고리 (인터페이스)	포함될 구현 클래스	특성 구성에 포함된 특성
IBasicHeatingElementComponent	BasicHeatingElementComponent	MicrowaveOven
IHeatingElementWithPowerLevel	HeatingElementWithNoPowerLevel	MicrowaveOven.HeatingElement.One-Level
	HeatingElementWithPowerLevel	MicrowaveOven.HeatingElement.Multi-Level && MicrowaveOven.PowerLevel
IHeatingElementComponent	HeatingElementComponent	MicrowaveOven

스가 선택되어 BasicHeagintElementComponent 클래스를 상속받으면서 조립된다. 예를 들어, 그림 3의 특성 모델에서 One-Level 특성이 선택되었다면, 목표 컴포넌트 생성 시 HeatingElementWithNo PowerLeve 클래스가 조립에 선택된다. 만약 Multi-Level 특성이 선택된다면 HeatingElementWith PowerLevel 클래스가 선택될 것이다. 따라서, 그림 5의 계층 구조에서 개발자가 선택한 특성 구성에 따라 ①번 경로 또는 ②번 경로를 따라 조립이 이루어져 목표 컴포넌트의 코드가 생성된다. 즉, 중복되는 기능은 하위의 BasicHeagintElement Component 클래스에 두고, 특성에 따라 서로 다른 기능을 제공하는 startCooking( ) 메소드는 서로 다른 두 개의 클래스가 구현하도록 한 후 코드 생성 시에 선택하도록 하는 것이다. 본 논문에서 각 구현 클래스는 XSLT 스크립트(4.3.4절 참조)로 작성되며, 어떤 조립 경로를 선택할 지는 조립 규칙 파일에 정의된다(4.3.3절 참조). 맨 위에 있는 IHeating



〈그림 6〉 MicrowaveOvenControl를 위한 계층 구조

ElementComponent 인터페이스와 HeatingElement Component 클래스는 어떤 경로를 통해서 조립이 되어도 동일한 이름의 컴포넌트가 생성되도록 하기 위해 추가로 정의되었다. 이러한 계층 구조는 XML 파일로 저장된다.

· **내부 재구성 지원 컴포넌트 (MicrowaveOven Control)**

내부 재구성 지원 컴포넌트는, 특성 구성에 포함된 특성에 따라 재구성 인자의 값이 결정되고 그 인자의 값에 의해 그 구현이 결정되는 컴포넌트를 의미한다. 내부 재구성 지원 컴포넌트인 MicrowaveOvenControl의 계층 구조를 정의하면 그림 6과 같다. 이 경우에는, 택일적 컴포넌트와 달리 조립 경로는 하나이다. 다만, 각 계층의 구현 클래스 안에 재구성을 지원하는 장치가 포함되어 있으며, 본 논문에서는 Basset이 제안한 프레임 기술[16]을 이용하여 XSLT 엘리먼트가 프레임 명령어로서 동작하여 그 역할을 수행한다(4.3.4절 참조). 계층 구조 맨 아래의 helper classes는 BasicMicrowaveOvenControl 클래스가 동작하는데 필요한 클래스들을 의미하며 이들 또한 재구성이 필요하다.

4.3 조립 규칙 정의

조립 규칙이란, 특성 구성에 포함된 특성에 따라 코드 생성 시 각 계층에 존재하는 구현 클래스 중에서 어떤 구현 클래스가 조립에 포함될 지를 나타내는 규칙이다. 본 절에서는 HeatingElement

Component 컴포넌트와 MicrowaveOvenControl 컴포넌트 각각에 대한 조립 규칙에 대해서 살펴본다.

· **택일적 컴포넌트 (HeatingElementComponent)**

표 2는 HeatingElementComponent 컴포넌트를 위한 조립 규칙을 나타낸다. 예를 들어, 특성 모델에서 Multi-Level 특성과 PowerLevel 특성이 선택되었다면, IHeatingElementWithPowerLevel 계층에 존재하는 구현 클래스 중에서 HeatingElementWithPowerLevel 클래스가 선택되어 조립 시 사용됨을 알 수 있다. 이러한 조립 규칙은 XML 파일로 저장된다.

· **내부 재구성 지원 MicrowaveOvenControl 컴포넌트**

MicrowaveOvenControl 컴포넌트인 경우에는 각 계층마다 하나의 구현 클래스만을 포함하므로 표 3과 같이 항상 동일한 구현 클래스가 특정 컴포넌트 생산 시 사용된다.

4.4 XSLT 코드 템플릿 개발

이 단계에서는 4.2절에서 정의한 각 컴포넌트의 계층 구조에 존재하는 각 계층 인터페이스 및 구현 클래스의 코드 템플릿를 개발한다. 각 코드 템플릿은 XSLT 스크립트[17]로 구현된다.

· **택일적 HeatingElementComponent 컴포넌트**

먼저, HeatingElementComponent 컴포넌트의 계층 구조에 존재하는 각 인터페이스를 위한 XSLT 스크립트를 작성한다 (IBasicHeatingElementComponent.xsl, IHeatingElementWithPowerLevel.xsl, IHeatingElementComponent.xsl). 예를 들어, IHeatingElementWithPowerLevel.xsl 파일의 내용은 그림 7과 같다. 가변성을 나타내는 부분이 없으므로 특별한 XSLT 엘리먼트가 없으며, 특성 구성에 따라 인터페이스가 바뀌는 경우에는 <xsl:if> 등의 XSLT 엘리먼트를 포함할 것이다.

다음으로 각 계층에 존재하는 구현 클래스에 대한 XSLT 스크립트를 작성한다(BasicHeatingElementComponent.xsl, HeatingElementWithNoPower Level.xsl, HeatingElementWithPowerLevel.xsl, HeatingElementComponent.xsl). 예를 들어, HeatingElementWithPowerLevel.xsl의 파일 내용은 그림 8과 같다.

· **내부 재구성 지원 MicrowaveOvenControl 컴포넌트**

먼저, MicrowaveOvenControl 컴포넌트의 계층 구조에 존재하는 각 인터페이스를 위한 XSLT 스크립트 작성한다(IBasicMicrowaveOvenControl.xsl, IMicrowaveOvenControl.xsl). 그리고 나서, 각 구현 클래스에 대한 XSLT 스크립트를 작성한다 (BasicMicrowaveOvenControl.xsl, MicrowaveOvenControl.xsl, ControlRequest.xsl, MicrowaveOvenControlAction.xsl, STT_MicrowaveOvenControl .xsl). 예를 들어, 그림 9은 BasicMicrowaveOvenControl.xsl의 파일의 일부분을 보여준다. 그림에서 <xsl:if> 엘리먼트 부분(코드 13-15줄, 27-29줄)이 코드 생성 시 재구성을 담당하는 부분이다. 특성 구성(5.2.2절 참조)에 MicrowaveOven.Turntable이 포함되었다면, Turntable과 관련된 작업이 코드에 추가됨을 알 수 있다. <xsl:choose> 등의 여러 가지 XSLT 엘리먼트를 이용해서 보다 다양한 종류의 재구성을 지

〈표 3〉 HeatingElementComponent를 위한 조립 규칙

컴포넌트 카테고리 (인터페이스)	포함될 구현 클래스	특성 구성에 포함된 특성
IBasicMicrowaveOvenControl	BasicMicrowaveOvenControl	MicrowaveOven
IMicrowaveOvenControl	MicrowaveOvenControl	MicrowaveOven.HeatingElement.One-Level

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:template match="Specification">
package HeatingElementComponent;
public class BasicHeatingElement implements IBasicHeatingElement {
 public BasicHeatingElement() {
 }
 public void stopCooking(){};
}
</xsl:template>
</xsl:stylesheet>
```

〈그림 7〉 IHeatingElementWithPowerLevel.xsl 파일

원할 수 있다.

4.5 복합적 가변성 지원

보다 복잡한 가변성을 지원하는 컴포넌트인 경우(예를 들어, PLUS의 제품 라인 소프트웨어 아키텍처의 컴포넌트가 <<variant-param-vp>>인 경우)에는 위의 두 가지 구현 기법(택일적 컴포넌트 구현 및 내부 재구성 지원 컴포넌트 구현)을 조합하여 사용하면 된다.

5. 컴포넌트 코드 자동 생성

5.1 컴포넌트 자동 생성 프로세스

각 컴포넌트의 코드 생성 과정은, 컴포넌트의 특성 모델로부터 개발자가 원하는 컴포넌트가 포함할 특성들을 선택하여 특성 구성을 작성하는 것에서 출발한다. 선택된 특성 구성과 조립 규칙을 바탕으로 내부 명세서가 생성되고, XSLT 프로세서가 이 XML 형태의 내부 명세서를 4.4절에서 개

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:template match="Specification">
package HeatingElementComponent;
import OvenData.*;
public class HeatingElementWithPowerLevel extends BasicHeatingElement
                          implements IHeatingElementWithPowerLevel {
  IOvenData ovenData1 = new OvenData();
  public HeatingElementWithPowerLevel() {
  }
  public void startCooking() {
    int power = ovenData1.readPower();
    // statements for starting cooking
  }
}
</xsl:template>
</xsl:stylesheet>
```

〈그림 8〉 HeatingElementWithPowerLevel.xsl 파일

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:template match="Specification">
package MicrowaveOvenControl;
import OvenData.*;
import DisplayInterface.*;
... ...
public class BasicMicrowaveOvenControl {
   ControlRequest currentControlRequest;
   IOvenData ovenData1 = new OvenData();
   ... ...
   // {feature = Turntable}
   <xsl:if test = "FeatureConfiguration/Feature/name = 'MicrowaveOven.Turntable'">
   ITurntableComponent turntable1 = new TurntableComponent();
   </xsl:if>
  public void sendControlRequest(ControlRequest newCr) {
     currentControlRequest = newCr;
     MicrowaveOvenControlAction action;
     STT_MicrowaveOvenControl stt = new STT_MicrowaveOvenControl();
// write what to do by Basic Microwave Oven Control component
     action = stt.processEvent(currentControlRequest);
     switch(action.getCurrentActionValue()) {
          case STT_MicrowaveOvenControl.ActionId_PromptForTime:
            displayInterface1.displayPrompt(IDisplayPrompts.PromptId_EnterCookingTime);
          ... ...
    // {feature = Turntable}
   <xsl:if test = "FeatureConfiguration/Feature/name = 'MicrowaveOven.Turntable'">
          turntable1.startTurning();
   </xsl:if>
          break;
        case ... ...
         }      };   };
</xsl:template>
</xsl:stylesheet>
```

〈그림 9〉 BasicMicrowaveOvenControl.xsl 파일 (일부분)

발된 XSLT 코드 템플릿에 적용하여 목표 컴포넌트의 코드를 최종 생성한다. 그림 10는 컴포넌트 자동 생성의 전체적인 프로세스를 보여준다.

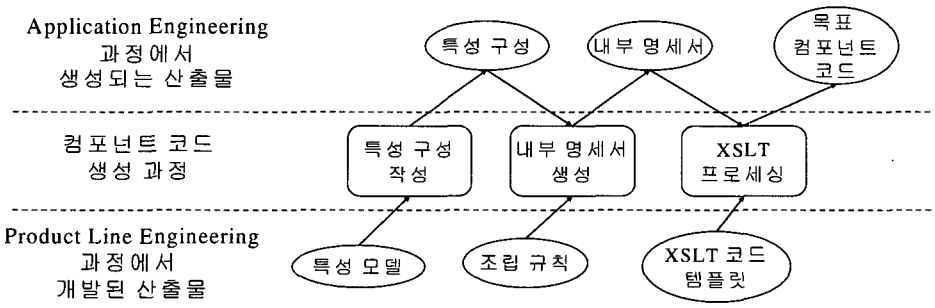
5.2 컴포넌트 자동 생성 예

5.2.1 특성 구성 입력

그림 11은 HeatingElementComponent의 특성 구성을 작성하는 화면을 보여준다. 현재 특성 구성에는 Multi-Level과 Turntable 특성이 포함되었다(굵은 사각형으로 표시). 재사용자가 특성 구성을 작성한 후 '< Java Code Generate >' 버튼을 누르면, 조립 규칙과 특성 구성을 바탕으로 내부 명세서를 생성한 후 XSLT 프로세서가 실행된다.

5.2.2 컴포넌트 코드 생성

XSLT 프로세서는 내부 명세서와 XSLT 코드 템플릿을 바탕으로 목표 컴포넌트의 코드를 생성한다. 그림 12는 5.2.1의 특성 구성으로부터 생성된 HeatingElementComponent.java의 코드 예를 보



〈그림 10〉 컴포넌트 코드 생성 과정

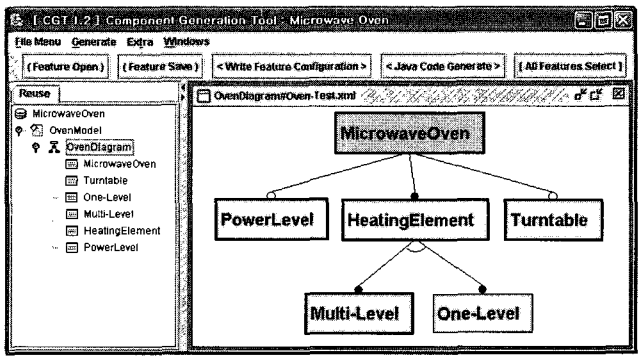
여준다. 목표 컴포넌트인 HeatingElementComponent 클래스가 HeatingElementWithPowerLevel 클래스를 상속받았음을 알 수 있다 (그림 4에서 조립 경로 ②번에 해당됨). 만약 개발자가 One-Level 특성을 선택하였다면, HeatingElementWithNoPower Level 클래스를 상속받았을 것이다(그림 4에서 조립 경로 ①번에 해당됨)

6. 관련 연구

본 논문의 기법은 기존의 여러 자동 생성 프로그래밍 기법으로부터 아이디어를 도입했으며 동시에 많은 다른 점도 가지고 있다. 본 절에서는 본 연구와 가장 관련이 깊은 자동 생성 기법인 GenVoca/AHEAD, XVCL, FORM과의 차이점에 대해서 기술한다.

· GenVoca와 AHEAD

GenVoca[10]는 새로운 모듈을 추가하거나 제거함으로써 확장 가능한 소프트웨어를 개발하기 위한 설계 방법론이다. 이 기법은 특성들을 점진적으로 추가함으로써 복잡한 시스템을 생성해 낼 수 있다는 'step-wise refinement' 개념을 적용한 방법론이다. 특정 프로그램에 대한 명세는 대수식(algebraic equation)으로 표현되며, 이러한 개념을 프로그램 코드 뿐 만 아니라 다른 소프트웨어 도큐먼트까지 확장하고 조립 연산자도 여러 가지가 가능하도록 한 방법론이 AHEAD[11,18]이다. GenVoca 또는 AHEAD는 컴포넌트 기반의 체계적인 개발 방법론과 결합되어 있지 않으며, 하나의 레이어에 존재하는 클래스들은 그 계층이 대표하는 특성을 구현하는데 모두 참여하는 클래스들이다. 본 논문에서



〈그림 11〉 특성 구성 입력 도구 및 코드 생성 도구

```
package HeatingElementComponent;

public class HeatingElementComponent
    extends HeatingElementWithPowerLevel
    implements IHeatingElementComponent {
}
```

〈그림 12〉 자동 생성된 HeatingElementComponent .java의 코드

는 한 레이어에 존재하는 클래스들은 조립 시 그 중에서 하나만 선택되어 조립에 참여한다는 점에서 다르다. 또한, 보다 추상적이고 재사용자가 이해하기 쉬운 특성 모델을 통해서 명세서가 만들어 진다는 점에서 GenVoca나 AHEAD와는 다르다.

· XVCL (Frame technology의 한 기술)

XVCL[12, 19]은 가변점(variation point)을 지원하는 x-frame이라는 기본 요소를 이용하여 제품 라인 자산을 구축한다. 제품 라인 아키텍처는 x-frame들을 트리 형태의 계층 구조로 구조화함으로써 구현되며, 이러한 x-framework에 특정 제품을 위한 명세서(SPC)를 적용하면 XVCL 프로세서라는 전용 프로세서가 트리 구조에 존재하는 x-frame들을 방문하면서 가변점들을 해결하고 이들을 조립하여 최종 제품을 생산한다.

이 기법은 x-frame과 x-framework을 개발하기 위한 명확한 분석 및 설계 기술을 포함하지 않는다. 즉, 컴포넌트 및 소프트웨어 아키텍처 모델과 x-frame들 사이의 관계가 명확하지 않다. 그리고, 새로운 컴포넌트 개발자가 새로운 기능을 추가하거나 수정할 때에도 이해하기가 어려워 오류 발생 가능성이 높다[20].

본 논문의 기법은 분석 및 설계 단계에서부터 가변성을 고려한 PLUS 방법론을 기반으로 한다. 설계 단계의 최종 산출물이자 제안된 기법의 출발점인 컴포넌트 기반 소프트웨어 아키텍처에 가변성이 명확하게 표현되어 있으며, 제품 라인 아키텍처에 포함되어 있는 아키텍처 레벨 가변성과 컴포넌트 레벨 가변성을 명확하게 지원한다.

· FORM(Feature Oriented Reuse Method)

FORM[6, 13, 21]은, 개발 초기에 제품들 사이의 공통점과 차이점을 특성 모델을 통해서 명확하게 표현한 후 재구성 가능한 도메인 아키텍처 및 컴포넌트를 구축한다. FORM에서의 제품 라인 아키텍처는 서브 시스템, 프로세스, 모듈 등의 세 단계로 구성되며 컴포넌트의 재구성은 FORM 매크로 언어를 이용한 코드 자동 생성, 캡슐화, 인자화(parameterization), 프레임워크 등의 기술을 이용한다.

이 개발 방법론은 개발 초기에 특성 모델을 통해서 공통점과 차이점을 명확히 하고, 특정 제품 생산 시 특성 선택에 따라 아키텍처와 컴포넌트 재구성을 통해서 특정 제품을 생산한다는 점에서 본 논문의 기법과 유사하다. 그러나, FORM에서의 제품 라인 아키텍처는 PLUS 방법론과 달리 선택적(<<optional>>) 또는 택일적(<<alternative>>) 컴포넌트를 표현하지 않아 가변성이 명확히 표현되어 있지 않으며 특성 선택에 따라 어떤 컴포넌트가 선택되는지에 대한 관계가 명확하지 않다.

코드 생성 기법 관점에서 보면, 본 논문의 컴포넌트는 FORM의 매크로 언어와 유사한 기능을 하는 XSLT 엘리먼트를 이용할 뿐만 아니라, GenVoca에서 제안하는 step wise refinement 개념을 이용한 계층 구조 아키텍처를 가지므로 특성별로 모듈을 구현하고 유연한 특성 조립을 가능하게 한다. 또한, 컴포넌트 내부의 아키텍처가 명확하여 이해도 및 유지보수성이 높아진다.

7. 결론 및 향후 과제

본 논문에서는, 컴포넌트 기반 제품 라인 개발 방법론 중에서 UML 모델링을 확장한 PLUS 개발 방법론의 주요 설계 산물인 제품 라인 아키텍처에 존재하는 가변성을 지원하기 위한 컴포넌트 자동 생성 기법을 제안하였다. 그리고, 마이크로웨이브 오븐을 사례 연구로 해서 가변성 지원 컴포넌트를 구현하는 방법을 살펴보았다. 본 논문의

기법은 GenVoca의 step-wise refinement, Mix-in 상속 기법, Basset의 프레임 기술 등을 활용하여 가변성을 지원하며, 구현 부품들은 계층 구조를 이루며 XSLT 스크립트로 작성된다. 재사용자가 특성 모델이 표현하는 결정 공간으로부터 원하는 특성들을 선택하면 목적에 맞는 컴포넌트의 코드가 자동 생성되며, 이러한 컴포넌트들이 상호 작용하여 컴포넌트 기반의 특정 응용 프로그램이 구축된다.

앞으로 보다 큰 규모의 제품 라인에 대한 사례 연구가 필요하며, 특성들 간의 의존성, 다른 종류의 특성 바인딩 시간을 지원하는 코드의 생성, 가변성 지원 컴포넌트 개발 시 보다 세부적인 가변성을 지원하기 위한 XSLT 엘리먼트 활용 기법 등에 대한 연구가 필요하다.

참 고 문 헌

[1] P. Clements and Linda Northrop, "Software Product Lines-Practice and Patterns", Addison Wesley, 2002.

[2] K. Kang, S. Cohen, W. Novak and A. Peterson, "Feature-oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute(SEI), November 1990.

[3] D. Weiss, C. Lai and R. Tau, "Software Product-Line Engineering: a family-based software development process", Addison-Wesley, Reading, MA, 1999.

[4] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J. Deb, "PuLSE. A Methodology to Develop Software Product Lines", The Proceedings of the Symposium on Software Reuse (SSR '99), May 1999.

[5] C. Atkinson et al., "Component-based Product Line Engineering with UML", Addison-Wesley, London, New York, 2002.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", Annales of Software Engineering, vol.5, 1998, pp.143-168.

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

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

[9] K. Czarnecki, U. W. Eisenecker, "Generative Programming: Methods, Tools, and Applications", Addison-Wesley, 2000.

[10] D. Batory and S. O'Malley, "The Design and Implementation of Hierarchical Software Systems with Reusable Components", ACM Trans. Software Eng. Methodology, Oct. 1992.

[11] D. Batory, J.N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement", IEEE Transactions on Software Engineering (IEEE TSE), June 2004.

[12] Zhang, H. and Jarzabek, S., " An XVCL-based Approach to Software Product Line Development ", Proc. 15 th International Conference on Software Engineering and Knowledge Engineering (SEKE'03), San Francisco, USA, 1 - 3 July, 2003.

[13] Kyo C. Kang, Jaejoon Lee, and Patrick

Donohoe, "Feature-Oriented Product Line Engineering", IEEE Software, Vol. 9, No. 4, Jul./Aug. 2002, pp. 58-65.

[14] S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design", SPLC2 2002, LNCS 2379, pp.130-153, 2002, Springer-Verlag.

[15] G. Bracha and W. Cook, "Mixin-Based Inheritance", In Proceedings of the 8th OOPSLA/ECOOP conference, ACM SIGP LAN Notices, vol.25, no.10, 1990, pp. 303-311.

[16] P. Bassett, "Framing Software Reuse: Lessons from the Real World", Prentice-Hall, 1996.

[17] http://www.w3.org/TR/xslt

[18] D. Batory, "Feature-Oriented Programming and the AHEAD Tool Suite", Internat ional Conference on Software Engineering (ICSE), Edinburgh, Scotland, 2004.

[19] Zhang, H. and Jarzabek, S. "An XVCL approach to hand ling variants: A KWIC product line example", Proc. 10 th Asia-Pacific Software Engineering Conference, APSEC'03, IEEE Comp. Soc., 10-12 December 2003, Chiangmai, Thailand.

[20] J. Blair and D. Batory, "A Comparison of Generative Approaches: XVCL and Gen-Voca", Technical Report December 2004.

[21] J. Lee and K. Kang, "Feature Binding Analysis for Product Line Component Development", van der Linden, F. (eds.), Software Product Family Engineering, Lecture Notes in Computer Science, Vol. 3014, Speinger-Verlag, Berlin Heidelberg (2004) pp.266-276.

◐ 저 자 소 개 ◑

최 승 훈

1990년 서울대학교 계산통계학과 졸업(학사)
1994년 서울대학교 대학원 계산통계학과 졸업(석사)
1999년 서울대학교 대학원 계산통계학과 졸업(박사)
2000년 ~ 현재 덕성여자대학교 컴퓨터학부 교수
2004년 ~ 2005년 George Mason University 방문 연구
관심분야 : 컴포넌트 기반 소프트웨어 공학, 소프트웨어 프러덕트 라인, 자동 생성 프로그래밍
E-mail : csh@duksung.ac.kr