컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인을 위한 코드 생성 시스템☆

Code Generation System for Component-based Real-time Embedded Software Product Lines

최 승 훈*
Seung Hoon Choi

요 약

소프트웨어 프러덕트 라인 개발 방법론이란, 개발 초기에 시스템의 공통적인 부분과 가변적인 부분을 명확히 하여 소프트웨어 자산을 구축한 후 요구 사항에 따라 가변적인 부분을 커스터마이징하여 목표 시스템을 생성하는 소프트웨어 개발 패러다임이다. 일반적인 소프트웨어 프러덕트 라인에 대한 연구는 활발히 진행되고 있지만, 컴포넌트 기반의 실시간 임베디드 소프트웨어 프러덕트 라인에 대한 연구는 상대적으로 미약하다. 본 논문에서는 실시간 임베디드 소프트웨어 프러덕트 라인 구축의 생산성 향상을 위해, 특성 모델을 통해 기능적 가변성을 지원하고 상태 모델을 통해 동기화 지원 코드를 생성하는 코드 생성 시스템을 제안한다.

Abstract

Software product-lines methodology is the software development paradigm to build the target system by customizing the variable part of software assets according to requirements. To attain this, the commonalities and variabilities of the system family should be modeled explicitly at early stage. Although the researches on general software product-lines are active, the researches on component-based real-time embedded software product-lines are rather inactive. In this paper a code generation system to support the functional variabilities via feature model and generate the code for synchronization via state model is proposed to increase the productivity of the development of the real-time embedded software product-lines.

☞ Keyword : Real-time Embedded Software Product Lines, Component, Variability, Automatic Code Generation

1. 서 론

소프트웨어 프러덕트 라인이란, 개발 초기에 시스템의 공통적인 부분과 가변적인 부분을 명확히 하여 소프트웨어 자산을 구축한 후, 요구 사항에 따라 가변적인 부분을 커스터마이징하여 목표 시스템을 빠르게 생성하는데 그 목적이 있는 소프트웨어 개발 방법론이다[1]. 최근에 여러 가지 컴포넌트 기반의 소프트웨어 프러덕트 라인 개발 방법론이 제안되었다[2-5]. 이러한 방법론들의 핵심은, 프러덕트 라인 개발 시 소프트웨어 자산을 프러덕트 라인 소프트웨어 아키텍처와 컴포넌트들로 모델링하고 개발하는데 있다.

그동안 컴포넌트 기술 적용에 따른 성능(performance) 저하에 대한 우려로 인해 실시간 임베디드 소프트웨어 개발에 컴포넌트 기술을 적용하는 시도가 활발하지 못했다. 하지만, 빠른 시간에 다양한 제품을 개발해 내야하는 임베디드 소프트웨어의 특성상 컴포넌트 기술이 제공하는 여러 가지 장점을 얻기 위해 최근에는 컴포넌트 기술을 임베디드 소프트웨어 개발

* 정회원 : 덕성여자대학교 컴퓨터공학부 교수
csh@duksung.ac.kr
[2006/05/15 투고 - 2006/05/29 심사 - 2006/07/05 심사완료]
☆ 이 논문은 한국과학재단의 해외 Post-doc. 연수지원에 의하여 연구되었음

에 적용하려는 노력이 활발해지고 있다[6].

한편, 소프트웨어 프러덕트 라인에서 가변성을 해결하고 코드를 자동으로 생성하는 기술에 대한 연구가 활발히 진행되었다. 특히 본 연구와 관련해서, 다음과 같은 연구들이 있다. GenVoca[7]는 특성들을 점진적으로 추가함으로써 복잡한 시스템을 생성해낼 수 있다는 'step-wise refinement' 개념을 적용하여 확장 가능한 소프트웨어를 개발하는 설계 방법론이다. AHEAD [8]는 이러한 개념을 프로그램 코드 뿐 만 아니라 다른 소프트웨어 도큐먼트까지로 확장하고 조립 연산자도 여러 가지가 가능하도록 한 방법론이다. XVCL[9]은 가변점(variation point)을 지원하는 x-frame이라는 기본 요소를 트리 형태의 계층 구조로 구조화함으로써 구현된다. 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 [10]이라는 도구를 제공하여 코드 자동 생성을 지원한다.

이러한 방법들은 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. 또한, 실시간 개념을 고려한 코드 생성 기법을 제공하지 않는다.

컴포넌트 기반 실시간 임베디드 소프트웨어의 가장 큰 특징 중에 하나는 병렬성이며, 여러 스레드 또는 태스크가 동시에 접근해야 하는 컴포넌트는 내부 상태에 대한 데이터 일관성을 유지하기 위한 동기화를 제공해야 한다. 현재 실시간 임베디드 소프트웨어 개발에 소프트웨어 프러덕트 라인과 컴포넌트 개념을 적용하는 방법에 대한 연구는 초기 단계에 있다. 특히, 코드 자동 생성 기법을 컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인에 적용한 사례가 거의 없다.

실시간 임베디드 소프트웨어 프러덕트 라인에 컴포넌트 기술을 적용한 연구 예로 다음과 같은 것들이 있다. [11]에서는 가전 제품에서의 소프트웨어 개발을 위해 Philips 사에서 개발되고 사용되는 컴포넌트 기술인 Koala를 제안하였다. Koala는 프러덕트 라인 아키텍처에 존재하는 컴포넌트를 위한 모델로서 Koala 컴포넌트들은 명확하게 정의된 인터페이스를 통해서 조립된다. 보다 추상적인 모델인 특성 모델로부터 재사용자의 요구사항을 입력받는 본 논문의 제안 시스템과 달리 Koala 컴포넌트 모델에서는 Koala 컴파일러가 Koala 컴포넌트 기술서를 입력받아 코드 생성을 수행한다. 또한, 병렬성 지원에 대한 부분이 컴포넌트 모델에 포함되어 있지 않다.

[12]에서는 임베디드 시스템을 위한 PURE 운영 체제의 구성을 용이하게 하기 위해 특성 모델을 사용하는 기법을 제안하였다. 이 기법은 본 논문의 제안 시스템과 비슷하게 특성 모델을 재사용자 요구 사항의 입력 도구로 사용하였으며, 각 컴포넌트 명세서에는 어떤 특성을 선택하였을 때 어떤 파일의 소스 코드가 선택될 것인지에 대한 로직이 포함되어 있다. 그러나, 본 논문의 기법과 달리 소스 코드 내부의 재구성 자동화는 지원하지 않으며 병렬성에 대한 언급이 없다.

[13]에서는 CORBA Component Model을 따르는 컴포넌트 기반 소프트웨어 구축 시 어플리케이션 조립, 구성, 배치 자동화를 지원하는 모델링 도구를 제안하였다. [14]에서는 field device를 위한 비기능적 요구사항을 표현하고 검사할 수 있는 컴포넌트 모델을 제안하였으며, 각 컴포넌트 마다 메모리 소비량에 대한 정보와 실행 시간에 대한 정보를 두어서 조립 후에 메모리 요구사항이나 스케줄링 가능성 검증에 사용하였다.

본 논문에서는 실시간 임베디드 소프트웨어 개발에 컴포넌트 기반 프러덕트 라인 개념을 적용할 때의 아키텍처 및 컴포넌트 가변성에 대해 연구한다. 그리고, 병렬 소프트웨어 프러덕트 라인 구축을 효율적으로 지원하기 위해, 특성 모델을 통해 기능적 가변성을 지원하고 상태 모델을 통해 동기화를 지원하는 컴포넌트 코드를 자동 생성하는 시스템을 제안한다.

본 논문의 컴포넌트 코드 생성 도구는 보다 추상적인 레벨의 특성 모델을 통해 재사용자의 요구에 맞는 병렬 컴포넌트 소스 코드를 자동 생성함으로써 실시간 임베디드 소프트웨어 프러덕트 라인 개발의 생산성을 향상시킨다. 또한, 코드 생성시에 최종 소프트웨어에 포함될 컴포넌트의 코드가 결정되므로 정적 바인딩(static binding)을 지원하고 이는 run-time 오버헤드를 줄여야하는 임베디드 소프트웨어 개발에 도움이 된다.

본 논문의 구조는 다음과 같다. 제 2 장에서 컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인과 가변성 및 컴포넌트에 대해서 기술한다. 제 3 장에서 가변성과 동기화를 지원하는 컴포넌트 구축 및 코드 자동 생성 기법에 대해서 설명한다. 제 4 장에서는 컴포넌트 코드 생성 시스템의 구현에 대해 기술하며, 제 5 장에서 결론 및 향후 과제를 기술한다.

2. 컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인

2.1 예제 시스템: 마이크로웨이브 오븐 프러덕트 라인

본 논문에서는 연성 실시간(soft real-time) 시스템인 마이크로웨이브 오븐 프러덕트 라인을 사례 연구로 하여 컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인을 개발하기 위한 프로세스와 코드 생성 기법을 기술한다. 마이크로웨이브 오븐 프러덕트 라인에 대한 간단한 요구 사항 명세서는 다음과 같다.

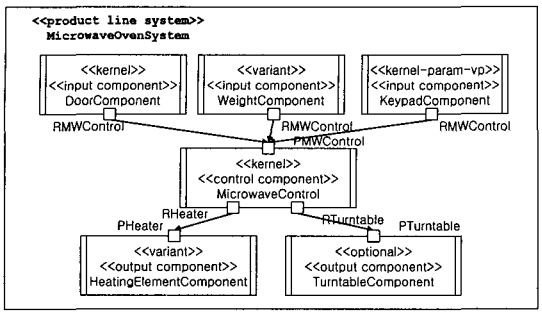
- 모든 제품은 공통적으로 문, 음식 센서, 숫자 키패드, 시작, 취소 버튼을 가지며, 요리 가열 중에는 현재 남은 요리 시간을 보여준다.
- 음식가열 부품(Heating Element)으로는 one-level 또는 multi-level 중 하나가 설치된다.
- 요리 중 요리 시간 1분을 추가하는 "1분 추가" 기능(Minute Plus 기능)은 제품에 따라 포함될 수도 있고 포함되지 않을 수도 있다.
- 음식을 회전시키는 "Turntable" 부품을 제품에 따라 포함할 수도 있고 포함하지 않을 수도 있다.

〈그림 1〉 마이크로웨이브 오븐 프러덕트 라인 요구 사항 명세서

2.2 컴포넌트 기반 소프트웨어 프러덕터 라인 아키텍처와 가변성

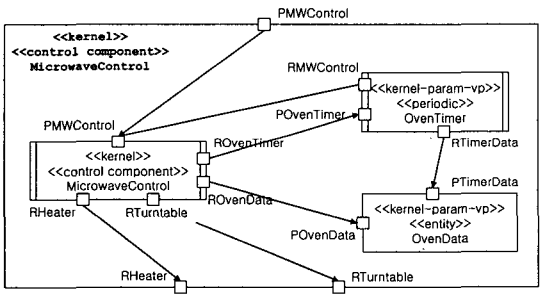
본 논문에서는 제안된 여러 가지 컴포넌트 기반 소프트웨어 프러덕트 라인 개발 방법론 중에서 UML 모델링을 확장하고 특성 모델을 가변성을 표현하기 위한 핵심 모델로 사용하고 있는 PLUS(Product Line UML based Software Engineering) 방법론[4]을 채택하여 컴포넌트 코드 자동 생성 기법을 기술한다.

[그림 2]는 PLUS 방법론을 적용하여 개발된 마이크로웨이브 오븐 프러덕트 라인의 컴포넌트 기반 소프트웨어 아키텍처와 컴포넌트 인터페이스 및 커넥터의 일부분을 보여준다. 그림에서, MicrowaveControl 컴포넌트는 DoorComponent, WeightComponent, KeypadComponent가 필요로 하는 인터페이스를 포함하는 PMW-Control 포트(port)를 가지고 있다. 또한, 출력 컴포넌트인 HeatingElementComponent와 Turntable Component에 서비스를 제공하는 인터페이스를 포함하는 RHeater와 RTurntable 포트를 가지고 있다. MicrowaveControl 내부에 존재하는 서브 컴포넌트들 사이의 병렬 상호작용 다이어그램은 [그림 3]과 같다.



〈그림 2〉 마이크로웨이브 오븐 소프트웨어 프러덕트 라인 아키텍처

컴포넌트 기반 소프트웨어 프러덕트 라인 아키텍처는 가변성을 지원해야 하며 크게 다음과 같은 두 개의 가변성으로 나누어진다. [그림 2]과 [그림 3]의 각 컴포넌트에는 이러한 가변성을 표현하는 스테레오 타입이 명시되어 있다.



〈그림 3〉 MicrowaveControl 복합 컴포넌트의 병렬 상호작용 다이어그램

• 아키텍처 차원의 가변성

목표 시스템에 어떤 컴포넌트가 포함될 지 말지(선택적 컴포넌트, optional component), 또는 여러 종류의 컴포넌트 중 어떤 종류의 컴포넌트가 포함될 지(택일적 컴포넌트, alternative component)를 나타내는 가변성이다. Coarser grained variability[15]라고도 한다. 선택적 컴포넌트는 <<optional>>, 택일적 컴포넌트는 <<variant>>라는 스테레오 타입을 가진다. 선택적 컴포넌트는 미리 프러덕트 라인의 자산으로 개발해 놓은 후 특정 제품 생산 시 포함 여부를 결정하면 되므로 자동 생성 기법을 적용할 필요가 없다. 반면, 택일적 컴포넌트들 사이에는 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.

• 컴포넌트 내부 가변성

내부 구성 인자에 따라 컴포넌트 내부의 코드가 재구성되는 가변성을 의미하며, <<XXX-param-vp>> 형태의 스테레오 타입을 가진다. 이러한 컴포넌트들의 내부 구성 인자의 값은 특정 제품 생산을 위해 선택된 특성 구성 결과에 따라 결정된다. 본 논문에서는 내부 구성을 자동으로 수행하여 특정 제품 생산 시 생산성을 향상시킨다.

일반적으로는 하나의 컴포넌트는 아키텍처 차원의 가변성과 컴포넌트 내부 가변성을 동시에 포함하는 경우가 많다.

2.3 실시간 임베디드 소프트웨어 프러덕트 라인에서의 컴포넌트

소프트웨어 컴포넌트란, 조립을 목표로 독립적으로 개발, 획득, 배포되며 서로 상호작용하여 원하는 기능을 수행하는 시스템을 달성하는 이진 단위(binary unit)이다[16]. 컴포넌트는 다른 부컴포넌트로 분해되는 복합 컴포넌트(composite component)와 더 이상 분해되지 않는 단위 컴포넌트(leaf component)로 분류된다. 본 절에서의 컴포넌트는 구현의 기본 단위인 단위 컴포넌트를 의미한다.

병렬성을 지원하는 실시간 임베디드 소프트웨어는 크게 두 가지 형태의 단위로 구성된다고 볼 수 있는데, 하나는 다른 엔터티에 행동을

행하는 활동(activities)이며 또 하나는 이러한 활동들 사이의 상호 작용을 제어하는 엔터티이다[17]. 다른 엔터티에 행동을 행하는 활동은 '능동적 컴포넌트'에 해당하며 이러한 활동들 사이의 상호 작용을 제어하는 엔터티는 '수동적 컴포넌트'에 해당한다.

실시간 임베디드 소프트웨어에서의 컴포넌트는 하나 이상의 객체로 구현되며, 컴포넌트를 구성하는 객체는 다음과 같이 두 가지로 분류된다.

- 능동적 객체(Active Object)

독자적인 컨트롤(control)을 가지면서 독립적으로 실행되며, 주로 다른 객체들에게 영향을 주는 행동들을 기동시키는(initiate) 일을 담당한다. Concurrent task 또는 스레드라고도 불린다. Java 언어인 경우에는 Thread 클래스가 여기에 해당된다.

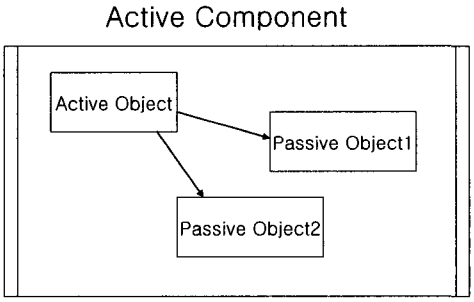
- 수동적 객체(Passive Object)

능동적 객체들에게 서비스를 제공하는 객체로서 주로 정보를 저장하고 제공하는 일을 담당한다. 독자적인 제어(control)를 가지지 않으며 능동적 객체로부터 오퍼레이션이 호출되면 그 능동적 객체의 제어 안에서 그 오퍼레이션이 실행된다. 수동적 객체는 '일반적인 수동적 객체'와 '동기화 지원 수동적 객체'로 구별된다. '일반적인 수동적 객체'란, Vector나 StringTokenizer와 같이 프로그램의 병렬성에 전혀 영향을 미치지 않는 객체를 의미하며 다른 능동적 객체나 수동적 객체에 의해 사용된다. 실시간 임베디드 소프트웨어의 경우 여러 개의 스레드가 동시에 하나의 수동적 객체를 접근하는 경우가 존재하고 이때 여러 스레드들 사이에 동기화를 지원해야 하는데 이러한 스레드를 '동기화 지원 수동적 객체'라고 한다.

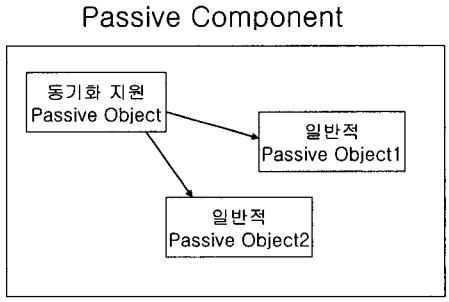
본 논문에서 실시간 임베디드 소프트웨어에서의 컴포넌트는 다음과 같이 크게 두 가지로 분류되며 위에서 정의한 능동적 객체와 수동적 객체의 조합으로 구현된다.

- 능동적 컴포넌트(Active Component)

능동적 컴포넌트는, 하나의 순차적 프로그램이나 순차적 실행을 대표하며 병렬성(concurrency)의 기본 단위가 된다. 구현 시에는 [그림 4]와 같이 능동적 객체(active object) 하나와 0개 이상의 수동적 객체(passive object)로 구현된다. 능동적 객체는 독자적인 컨트롤을 가지면서 0개 이상의 수동적 객체에 대한 접근을 통해서 클라이언트가 원하는 오퍼레이션을 수행한다. 이 컴포넌트 내부에서는 병렬성이 존재하지 않으며, 전체 시스템의 병렬성은 능동적 컴포넌트가 여러 개 존재하며 동시에 실행될 때



〈그림 4〉 능동적 컴포넌트(Active Component) 내부 구조



〈그림 5〉 수동적 컴포넌트(Passive Component)의 내부 구조

이루어진다. 이러한 능동적 컴포넌트는 비동기적으로(asynchronous) 실행되거나 주기적으로(periodically) 어떤 활동을 실행한다.

• 수동적 컴포넌트(Passive Component)

능동적 컴포넌트와 달리 독자적인 제어를 가지지 않으며, 주로 능동적 객체에 서비스를 제공하는 일을 담당한다. 본 논문에서는 동기화 지원이 필요한 객체만을 고려한다. 이러한 수동적 컴포넌트는 멀티 스레드를 지원하는 실시간 임베디드 소프트웨어에서 여러 개의 능동적 컴포넌트가 동시에 접근할 때 동기화를 지원하여 여러 스레드들 사이의 협동을 가능하게 하며 병렬성과 성능을 높여주는 역할을 한다. [그림 5]에서 보는 것과 같이 구현 시에는 하나의 동기화 지원 수동적 객체와 0개 이상의 일반적인 수동적 객체로 구현된다. 본 논문에서는 능동적 컴포넌트와 수동적 컴포넌트를 구성하는 각 구현 객체의 코드를 생성함으로써 실시간 임베디드 소프트웨어 프러덕트 라인의 생산성을 높이려고 한다.

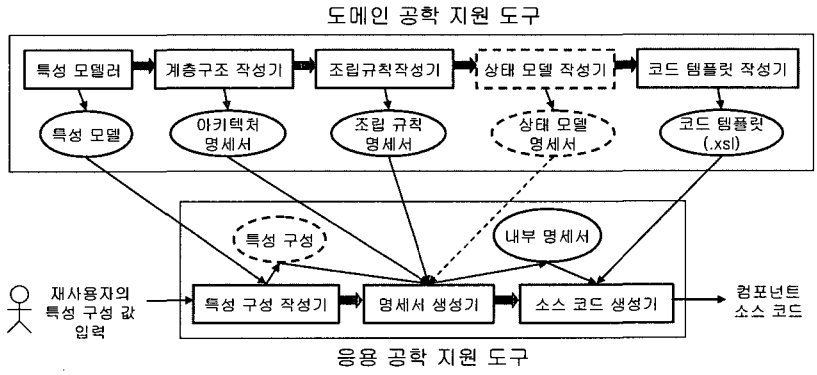
3. 실시간 임베디드 프러덕트 라인을 위한 코드 생성 시스템

본 논문에서 제안하는 실시간 임베디드 프러덕트 라인을 위한 코드 생성 시스템의 전체구조는 [그림 6]과 같다. 본 시스템은 크게 도메인 공학 지원 도구와 응용 공학 지원 도구로 나뉘어진다. 본 도구는 특성 모델[18]로부터 개발자가 원하는 목표 시스템에 대한 명세서를 입력받아 특성 구성(feature configuration)을 만들고 이로부터 가변성을 해결하여 컴포넌트를 구성하는 객체의 코드를 자동으로 생성한다. 특성 구성이란, 특성 모델로부터 재사용자가 목표 시스템에 포함하기로 결정한 특성 선택 결과를 의미한다[19].

도메인 공학 지원 도구를 구성하는 세부 모듈은 가변성 지원 컴포넌트 구축 프로세스의 각 단계를 지원한다. 그림에서 상태 모델 작성기가 점선으로 되어 있는 이유는, 컴포넌트 중에서 동기화를 지원하는 컴포넌트에 대해서만 상태 모델을 작성하기 때문이다.

응용 공학 지원 도구는 특성 모델을 이용하여 재사용자의 요구 사항을 입력받아 특성 구성을 만들고 도메인 공학 단계에서 만들어진 여러 가지 자료를 이용하여 내부 명세서를 만든 후 XSLT로 이루어진 코드 템플릿들을 처리하여 최종 소스 코드를 생성한다. 특성 모델, 계층구조, 조립 규칙, 상태 모델, 내부 명세서 등은 모두 xml 형식의 파일로 저장, 관리된다.

본 절에서는, 내부 가변성과 동기화 지원 특



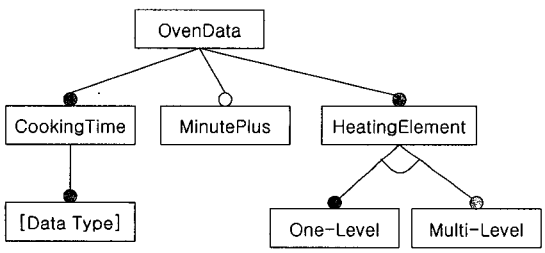
〈그림 6〉 실시간 임베디드 소프트웨어 프러덕트라인의 컴포넌트 코드 생성 시스템

징을 가지는 OvenData 컴포넌트의 코드 생성을 예로 도메인 공학에서의 가변적 컴포넌트 구축 프로세스와 코드 생성 시스템을 기술한다. [그림 3]의 MicrowaveControl 컴포넌트의 병렬 상호작용 다이어그램을 살펴보면 OvenData 컴포넌트는 <<kernel-param-vp>> 스테레오 타입으로 표현되어 있듯이 내부 가변성을 지원한다. 또한, 두 스레드 MicrowaveControl과 OvenTimer는 OvenData가 관리하고 있는 '남은 요리 시간' 데이터에 동시에 접근 가능하므로 동기화를 지원해야 한다.

3.1 특성 구성을 통한 가변성 지원

3.1.1 특성/컴포넌트 의존성 식별

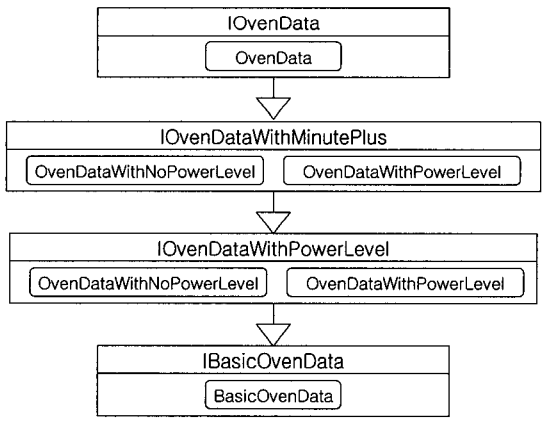
먼저, 특성 모델에서의 특성과 컴포넌트들 사이의 의존성 식별을 수행한다. [그림 7]은 전체 마이크로웨이브 오븐 특정 모델 중에서 Oven Data 컴포넌트 구현에 영향을 주는 특성들로만 이루어진 특성 모델을 보여준다.



〈그림 7〉 OvenData를 위한 특성 모델

3.1.2 계층 구조 아키텍처링

본 논문에서의 각 컴포넌트는 GenVoca[7,8]에서 제안한 step-wise refinement 기법(상속을 이용해서 새로운 특성을 추가)과 Basset이 제안한 프레임 기술[20]을 이용하여 구현 코드가 결정된다. 이를 위해서 각 컴포넌트를 위한 계층 구조를 정의해야 하며, OvenData 구현에 영향을 주는 특성들을 고려하여 정의한 계층 구조가 [그림 8]과 같다. 상속 관계에 따른 조립 시 조립 규칙에 따라 각 계층에 존재하는 여러 개의 구현 클래스 중 하나가 선택되어 조립된다(3.1.3절 참조). 또한, 각 계층에 존재하는 구현 클래스는 XSLT로 작성되며 XSLT 엘리먼트가 프레임 명령어로서 동작하여 내부 재구성을 지원한다(3.1.4절 참조).



〈그림 8〉 OvenData를 위한 계층 구조

〈표 1〉 OvenData 컴포넌트를 위한 조립 규칙

컴포넌트 카테고리(인터페이스)	포함될 구현 클래스	특성구성에 포함될 특성
IOvenData	OvenData	OvenData
IOvenDataWithMinutePlus	OvenDataWithMinutePlus	OvenData.MinutePlus
	OvenDataWithNoMinutePlus	OvenData
IOvenDataWithPowerLevel	OvenDataWithPowerLevel	OvenData.HeatingElement.PowerLevel
	OvenDataWithNoPowerLevel	OvenData.HeatingElement
IBasicOvenData	BasicOvenData	OvenData

```
<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 OvenData;

public class OvenDataWithPowerLevel extends BasicOvenData implements
IOvenDataWithPowerLevel {
 IOvenData ovenData1;
 private int level;
 public OvenDataWithPowerLevel() {}

 // IPowerData {feature = Multi-levelHeating}
 public void clearPower() {
   power = ovenData1.readPower();
   setPowerLevel(power);
 }
 public void setPower(int level){
 this.level = level;
 }
 public int readPower() {
 return 1;
 };
}
</xsl:template>
</xsl:stylesheet>
```

〈그림 9〉 BasicOvenData 구현 클래스를 위한 XSLT 파일(일부)

3.1.3 조립 규칙 정의

다음 단계로 조립 규칙을 작성한다. 조립 규칙이란, 특성 구성에 포함된 특성에 따라 코드 생성 시 계층 구조 상의 각 계층에 존재하는 구현 클래스 중에서 어떤 구현 클래스가 조립에 포함될 지를 나타내는 규칙이다.

OvenData 컴포넌트를 위한 조립 규칙은 [표1]과 같다. 예를 들어, 특성 구성에 OvenData. Minute Plus 특성이 포함된다면 [그림 8]의 IOven Data WithMinutePlus 계층의 두 구현 클래스 중에서 OvenDataWithMinutePlus 클래스가 목표 컴포넌트 조립에 포함된다.

3.1.4 가변성 지원 XSLT 코드 템플릿 개발

이 단계에서는 3.1..2절에서 정의한 각 컴포넌트의 계층 구조에 존재하는 각 계층 인터페이스 및 구현 클래스의 코드 템플릿를 개발한다. 각 코드 템플릿은 XSLT 스크립트[21]로 구현되며, XSLT 스크립트에 포함된 XSLT 엘리먼트가 특성 구성과 계층 구조를 이용해 생성된 내부 명세서로부터 필요한 데이터를 읽어와서 내부 재구성을 실현한다. [그림 8]의 Oven Data 컴포넌트의 계층 구조 중 IOvenDataWith MinutePlus 계층에 존재하는 OvenDataWith MinutePlus 구현 부품을 위한 XSLT 파일 일부를 보면 [그림 9]와 같다.

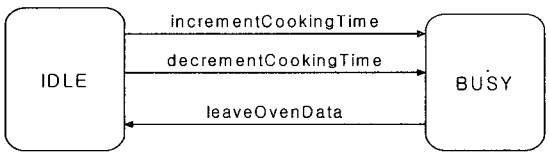
3.2 상태 모델을 통한 동기화 지원

3.2.1 상태 모델 작성

실시간 임베디드 소프트웨어 프러덕트 라인에 존재하는 컴포넌트가 수동 컴포넌트이면서 두 개 이상의 능동 컴포넌트(스레드)가 동시에 접근하는 경우에는 이들 사이의 협력을 위해서 동기화(synchronization)를 지원해야 한다. [17]에서 제안한 모니터 자료 구조를 이용한 상태 전이 다이어그램으로부터의 동기화 지원 프로그래밍 메커니즘을 본 논문의 코드 생성 시스템에 적용한다.

[그림 10]은 OvenData를 위한 상태 모델이다.

OvenTimer 스레드는 decrementCookingTime 메시지를 이용해서 OvenData 컴포넌트에 접근한다. MicrowaveOvenControl은 incrementCookingTime 메시지를 이용하여 OvenData 컴포넌트에 접근한다. 이 두 스레드는 동시에 OvenData의 cookingTime 속성에 접근할 수 있으므로 cookingTime의 데이터 무결성을 위해서 OvenData는 동기화를 지원해야 한다.



〈그림 10〉 OvenData를 위한 상태 모델

3.2.2 동기화 지원 XSLT 코드 템플릿 개발

상태 모델로부터 동기화 지원 코드로의 변환 원리를 자바 언어를 기준으로 간략하게 기술하면 다음과 같다.

상태를 전이시키는 메시지(예를 들어, [그림 10]의 decrementCookingTime)는 그 컴포넌트의 메소드로 구현된다.

상태 전이 메시지의 이전 상태(예를 들어, [그림 10]의 IDLE 상태)는, 메소드 주요 본체 실행 전 사전 조건(pre-condition) 검사에 사용된다. 즉, 전제 조건 검사 부분에서 이 컴포넌트의 상태를 검사해서 이전 상태가 일치하지 않는 경우 호출한 스레드가 wait() 명령을 수행하도록 함으로써 이 컴포넌트를 충돌로부터 보

```
<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 OvenData;

public class BasicOvenData implements IBasicOvenData {
 private static final int IDLE = 0;
 private static final int BUSY = 1;
 private int state;
 private <xsl:value-of select="FeatureConfiguration/Feature[name='DataType']/value"/>
cookingTime =0;

 synchronized public int decrementCookingTime() {
 // pre-condition processing {guard}
 <xsl:if
test="StateTransitionDiagram/Transition/Event[@method='decrementCookingTime']">
 while(true) {
 if(state == <xsl:value-of
select="StateTransitionDiagram/Transition/Event[@method='decrementCookingTime']/fromS
tate"/>)
        break;
        wait();
 }
 </xsl:if>

 // main part
 cookingTime--;

 // post-condition processing
 state = <xsl:value-of
select="StateTransitionDiagram/Transition/Event[@method='decrementCookingTime']/toSta
te"/>;
 notifyAll();

 return cookingTime;
 }
 ... ...
}
</xsl:template>
</xsl:stylesheet>
```

〈그림 11〉 BasicOvenData 구현 클래스의 동기화 지원을 위한 XSLT(일부)

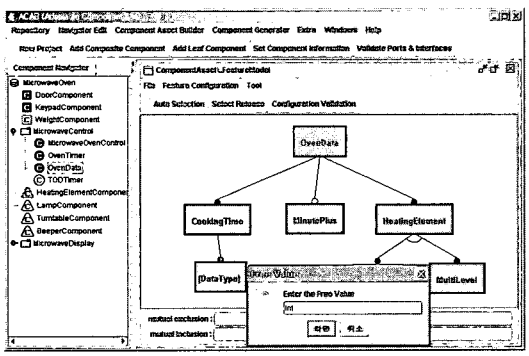
호한다.

상태 전이 메시지의 이후 상태(예를 들어, [그림 10]의 BUSY 상태)는, 메소드 주요 본체 실행 후의 사후 조건(post-condition) 확립에 사용된다. 즉, 컴포넌트의 현재 상태를 바꾸고 notifyAll() 메소드 호출을 통해서 상태가 바뀌었음을 기다리고 있는 다른 스레드들에게 알려준다.

[그림 11]은 수동 컴포넌트인 OvenData 컴포넌트 코드 생성을 위한 XSLT 코드 템플릿의 일부를 보여준다. 전제 조건 확인에 필요한 상태 정보는 내부 명세서에서 얻어오며, 전제조건을 만족하지 않으면 decrementCookingTime()를 호출한 쓰레드를 대기하도록 한다.

4. 시스템 구현

본 논문에서 제안하는 코드 생성 시스템의 전체 GUI는 [그림 12]와 같다. 현재 화면은 응용 공학 단계에서 목표 컴포넌트를 생성하기 위한 특성 구성 값을 입력하는 과정을 보여준다. 생성하고자 하는 OvenData 컴포넌트는 의무적 특성인 CookingTime와 선택적 특성인 MinutePlus을 포함하며, 택일적 특성 중에서는 MultiLevel 특성을 포함한다(굵은 사각형으로 표시함). 그리고, CookingTime을 나타내는 변수의 데이터 타입을 int 형으로 설정한다.



〈그림 12〉 코드 생성 시스템의 전체 GUI 화면

특성 구성을 입력한 후 코드 생성 명령을 내리면, 입력된 특성 구성에 따라 계층 구조 상에 존재하는 구현 부품 중에서 조립 규칙에 따라 적합한 구현 부품이 각 계층 마다 선택되어 조립에 참여하고 그 결과 목표로 하는 클래스의 코드가 생성된다. [그림13]은 [그림 8]의 IBasicOvenData 계층에 존재하는 BasicOvenData 부품이 입력된 특성 구성과 상태 모델에 따라 재구성이 완료된 결과 코드를 보여준다.

```
package OvenData;

public class BasicOvenData implements
IBasicOvenData {
 private static final int IDLE = 0;
 private static final int BUSY = 1;
 private int state;
 private int cookingTime = 0;

 synchronized public int
decrementCookingTime() {
 try {
 // pre-condition processing {guard}
 while(true) {
 if(state == IDLE)
        break;
 wait();
 }
 // main part
 cookingTime--;

 // post-condition processing
 state = BUSY;
 notifyAll();
 } catch (InterruptedException e){
 }
 return cookingTime;
 };
... ...
}
```

〈그림 13〉 자동 생성된 OvenDataWithPowerLevel 구현 부품의 코드

5. 결 론

본 논문에서는 컴포넌트 기반 실시간 임베디드 소프트웨어 프러덕트 라인의 핵심 자산인 컴포넌트의 가변성을 해결하고 재구성하여 코드를 자동으로 생성하는 시스템을 제안하였다. 이를 위해, 컴포넌트 기반 실시간 임베디드 소프트웨

어 프러덕트 라인에서의 가변성을 아키텍처 차원의 가변성과 컴포넌트 내부 가변성으로 분류하고 구현 관점에서 컴포넌트의 종류와 컴포넌트를 구성하는 객체의 종류를 정의하였다.

본 논문의 도구는 가변성 지원 컴포넌트를 구축하는 단계를 지원하는 도메인 공학 단계 지원 도구와 특성 구성을 입력받아 목적에 맞는 코드를 자동으로 생성하는 응용 공학 단계 지원 도구로 나누어진다. 컴포넌트는 종류에 따라 독립적인 제어를 가지는 능동적 컴포넌트와 다른 능동적 컴포넌트들에게 서비스를 제공하는 수동적 컴포넌트로 나누어진다. 응용 공학을 수행하는 자가 특성 모델로부터 원하는 특성들을 선택하면 능동적 컴포넌트는 기능적 가변성만을 해결한 코드를 생성하며, 수동적 컴포넌트는 기능적 가변성 이외에 동기화 지원하는 코드를 포함하는 코드를 생성한다.

본 논문에서 제안한 기법은 임베디드 소프트웨어 코드 생성 시에 동기화를 지원함과 동시에 재구성성을 지원하여 재사용자의 요구에 맞는 컴포넌트 소스 코드를 자동 생성한다. 컴포넌트 재구성 지원을 통하여 임베디드 소프트웨어 개발 생산성을 향상시키고, 고수준의 추상화 모델인 특성 모델을 통해서 재사용을 지원하며, XML 형태의 명세서를 쉽게 변경할 수 있는 등의 여러 가지 이점을 제공한다. 본 연구 결과는 병렬성이 중요시되는 실시간 임베디드 소프트웨어 프러덕트 라인 구축에 활용될 수 있을 것이다.

앞으로 보다 다양한 병렬 컴포넌트 및 임베디드 소프트웨어 프러덕트 라인에 본 논문의 기법과 도구를 적용해보아야 할 것이다. 또한, 가변성 지원 컴포넌트 구축 시 실시간 개념과 관련된 제약 조건, 예를 들어 최악 메모리 사용량, CPU 비용, 주기(period) 등에 대한 정보를 제공하고, 조립 후 생성된 컴포넌트가 목표 플랫폼의 제약조건에 맞는지 검사하는 과정에 대한 연구가 필요하다.

참 고 문 헌

[1] P.Clements and L.Northrop,"Software Product Lines: Practices and Patterns", Addison-Wesley, 2002.

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

[3] 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.

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

[5] 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.

[6] Mikael Akerholm, Johan Fredriksson, "A Sample of Component Technologies for Embedded Systems", Malardalen Real-Time Research Centre Technical Report , November 2004.(www.mrtc.mdh.se)

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

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

[9] 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.

[10] 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.

[11] R. van Ommering, F. van der Linden and J. Kramer, "The Koala component model for consumer electronics software", IEEE Computer, 33(3): 78-85, March 2000.

[12] Danilo Beuche, Olaf Spinczyk, and Wolfgang Schroder-Preikschat, "Finegrain Application Specific Customization for Embedded Software", In Proceedings of the International IFIP TC10 Workshop on Distributed and Parallel Embedded Systems (DIPES 2002), Montreal, Canada, August 2002. Kluwer Academic Publishers.

[13] Andrey Nechypurenko and et. al., "Developing Product-lines for Distributed Real-time and Embedded Systems with Modeling Tools and Component Middleware: A Case Study". 2005.

[14] Robert L. Nord, "Meeting the Product Line Goals for an Embedded Real-Time System", IW-SAPF-3, LNCS 1951, pp.19-29, 2000.

[15] Ian McRitchie, T. John Brown, and Ivor T.A. Spence, "Managing Component Variability within Embedded Software Product Lines via Transformation Code Generation", PFE 2003, LNCS 3014, pp.98-110, 2004.

[16] Clemens Szyperski,"Component Software: Beyond Object-Oriented Programming" 2nd Ed., Addison-Wesley, 2002.

[17] Charles W. Kann, "Creating Components: Object Oriented, Concurrent, and Distributed Computing in Java", 2004, Auerbach Publications.

[18] 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.

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

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

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

◐ 저 자 소 개 ◑

최 승 훈 (Seung Hoon Choi)

1990년 서울대학교 계산통계학과 졸업(학사)
1994년 서울대학교 대학원 계산통계학과 졸업(석사)
1999년 서울대학교 대학원 계산통계학과 졸업(박사)
2000년 ~ 현재 덕성여자대학교 컴퓨터공학부 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