특성 다이어그램과 XML/XSLT 기술을 이용한 컴포넌트 코드 자동 생성

Component Code Generation Using Feature Diagram and XML/XSLT

최 승 훈*
Seong-Hoon Choi

요 약

최근 소프트웨어 개발의 생산성 향상을 위해 보다 큰 단위의 재사용을 가능하게 하는 컴포넌트 기반 소프트웨어 생산라인에 대한 연구가 활발히 진행되고 있다. 이는, 소프트웨어 생산라인의 자산에 존재하는 일반적인 컴포넌트들을 구체화하고 조립함으로써 고품질의 특정 응용 프로그램을 빠르게 개발하는데 그 목적이 있다. 컴포넌트 기반 소프트웨어 생산라인 구축의 핵심은 컴포넌트의 품질이며, 컴포넌트의 품질을 결정짓는 가장 중요한 특성 중의 하나가 재사용시 재사용자가 컴포넌트의 기능과 구조를 목적과 환경에 맞게 변경할 수 있도록 해 주는 '재구성성(reconfigurability)'이다. 본 논문에서는, 특성 다이어그램과 XML/XSLT 기술을 이용하여 컴포넌트 코드 생성 시에 재구성 성을 지원하는 컴포넌트 코드 자동 생성 기법을 제안한다. 본 논문의 기법은, 컴포넌트 패밀리의 특성 다이어그램에 표현되어 있는 차이점들에 대하여 특정 값이나 선택을 제공함으로써 컴포넌트 재사용자가 자신의 목적에 맞는 구체적인 컴포넌트의 소스 코드를 자동으로 생성할 수 있게 해 준다. 사례 연구로서 리스트 컨테이너 패밀리에 대한 컴포넌트 코드 시스템을 개발함으로써 특성 모델과 XML 관련 기술의 자동 생성 프로그래밍 영역에서의 적용 가능성을 보여준다. 본 논문의 코드 생성 시스템은 컴포넌트 기반 소프트웨어 생산라인 구축의 기반 기술로서 이용될 수 있으며, 보다 큰 단위의 컴포넌트 생성을 위한 기법으로 확장 가능하다.

Abstract

Recently for software development productivity a lot of researches in the field of software engineering have focuses on the component-based software product lines which allows the reuse of larger-granularity software components. Its purpose is to develop the specific software application of quality more rapidly by instantiating and assembling the components populated in software product line assets. The essential part to build the component-based software product lines is the quality of components, and one of the most important features determining the quality of components is 'reconfigurability'. Component reconfigurability means the extent to which the reusers can change the functions and architecture of the component according to their context and environment. This paper proposes the component code generation technique which provides the reconfigurability at the time of code generation using the feature diagram and XML/XSLT technologies. The approach of this paper allows the component reusers to get automatically their own component source code by providing only the values of variabilities represented in the feature diagram of the component family. The real world example, the code generation system for a list container family, shows the applicability of the feature model and XML related technologies in the area of the generative programming. Our approach should be basis to build the component based software product lines and extensible to support the larger grauiarity components.

1. 서 론

최근 소프트웨어 개발의 생산성 향상을 위해 보다 큰 단위의 재사용을 가능하게 하는 컴포넌트 기반 소프트웨어 생산라인에 대한 연구가 활발히 진행되고 있다[1]. 소프트웨어 생산라인이란, 공통된 핵심 소프트웨어 자산(core software asset)들로부터 특정 목표나 시장을 위해서 개발된 소프트웨어 시스템들의 집합을 의미한다. 컴포넌트 기반 소프트웨어 생산라인은, 소프트웨어 생산라인의 자산에 존재하는 일반적인 컴포넌트들을 구체화하고 조립함으로써 보다 신뢰성 있고 고품질의 특정 응용 프로그램을 빠르게 개발하는데 그 목적이 있다. 컴포넌트 기반 소프트웨어 생산라인

* 정회원 : 덕성여자대학교 전산학과 교수
csh@duksung.ac.kr

의 구축을 위해서는, 소프트웨어 자산에 존재하는 일반적인 컴포넌트들을 재사용자의 목적에 맞는 컴포넌트로 구체화하는 과정을 효율적으로 지원하기 위한 재구성성(reconfigurability)이 중요하다. 소프트웨어의 재구성성이란, 소프트웨어 개발 시 그 소프트웨어가 실행되는 환경과 목적에 맞도록 그 기능과 구조를 쉽게 재구성할 수 있는 능력을 의미한다.

컴포넌트의 재구성성이 지원되는 시점은 다음과 같이 세 가지로 분류될 수 있다.

1) 컴포넌트 코드 생성 시: 재사용을 위한 컴포넌트 코드 생성 시 커스터마이징(customizing) 과정을 통해서 원하는 컴포넌트 코드를 자동 생성한다.
2) 컴포넌트 조립 시: 컴포넌트를 조립할 때 소프트웨어 개발 환경이 제공하는 GUI를 통해서 컴포넌트의 여러 가지 속성의 값을 제공함으로써 원하는 컴포넌트로 커스터마이징 한다. (예: design-time property)
3) 컴포넌트 운용 시: 실행 시간에 사용자가 제공하는 값이나 속성 파일에 저장되어 있는 값에 의해 컴포넌트의 속성 값을 설정한다.

컴포넌트 코드 생성 시에 재구성성을 지원하면, 보다 일반적인 컴포넌트들이 컴포넌트 저장소에 존재하며 재사용 시에 재사용자가 원하는 구체적인 컴포넌트를 자동 생성할 수 있다. 재구성성이 지원되지 않는 컴포넌트의 경우에는 무수히 많은 구체적인 컴포넌트들이 컴포넌트 저장소에 존재해야 할 것이며, 따라서 원하는 컴포넌트 검색이 어려워진다. 또한, 미리 구축된 컴포넌트들은 특정 환경에 맞게 개발된 것이 아니기 때문에, 불필요한 기능이 포함되어 있는 경우가 많으며(fat component), 이 컴포넌트를 재사용한 시스템이 불필요하게 커져서 성능 상의 문제를 일으킬 수 있다. 그리고, 컴포넌트에 대한 소스 코드를 접근할 수 없어 철저한 테스트를 실행할 수 없으므로, 신뢰성(dependability)이 떨어진다[2].

컴포넌트 코드 생성 시 재구성성을 지원하기 위해서는, 구체적인 소프트웨어를 개발할 때 기존의 소프트웨어 구성 부품을 재사용하여 조합하기 위한 체계적이고 효율적인 방법을 지원해야 한다. 또한, 컴포넌트 개발자가 하나의 컴포넌트 개발에 집중하기 보다는 공통된 특징을 공유하는 컴포넌트 패밀리 개발에 초점을 맞추어야 한다. 이러한 컴포넌트 패밀리 구축 프로세스는 도메인 공학(domain engineering)과 특성 모델에 기반을 둔다.

한편, 네트워크나 웹 상에서의 데이터 교환을 목적으로 개발된 XML(eXtensible Markup Language)은 도메인 지식의 표현에 대한 표준 및 자기 서술적인 기능을 제공함으로써 데이터검색의 효율성과 보다 단순한 응용프로그램 개발을 가능하도록 한다. 이러한 XML 관련 기술의 적용 분야가 점점 확대되고 있다. 특히, 소프트웨어 개발의 생산성을 높이기 위하여 최근 활발히 연구되는 소프트웨어 자동 생성 프로그래밍(Generative Programming) 분야에서 XML 기술을 적용하기 위한 연구가 진행되고 있다. 또한, XSLT(eXtensible Stylesheet Language Transformation)의 문서 변형 기능(document transformation capability)을 코드 자동 생성에 응용할 수 있다.

본 논문에서는, 특성 모델의 주요 구성 요소인 특성 다이어그램과 XML/XSLT 기술을 이용하여 컴포넌트 코드 생성 시에 재구성성을 지원하는 컴포넌트 코드 자동 생성 기법을 제안한다. 본 논문의 기법은, 컴포넌트 패밀리의 특성 다이어그램에 표현되어 있는 차이점들에 대하여 특정 값이나 선택을 제공함으로써 컴포넌트 재사용자가 자신의 목적에 맞는 구체적인 컴포넌트의 소스 코드를 자동으로 생성할 수 있게 해 준다. 사례 연구로서 리스트 컨테이너 패밀리에 대한 컴포넌트 코드 시스템을 개발함으로써 특성 모델과 XML 관련 기술의 자동 생성 프로그래밍 영역에서의 적용 가능성을 보여준다. 본 논문의 코드 생성 시스템은 컴포넌트 기반 소프트웨어 생산라인 구축의 기반 기술로서 이용될 수 있으며, 보다 큰 단위의

컴포넌트 생성을 위한 기법으로 확장 가능하다.

본 논문의 구성은 다음과 같다. 제 2 절에서 관련 연구를 기술하며, 제 3 절에서는 본 논문의 기법을 설명하기 위한 예제인 리스트 컨테이너를 기술한다. 제 4 절에서는 컴포넌트 코드 생성 기법을 자세히 기술하고, 제 5 절에서 결론 및 향후 연구에 대하여 기술한다.

2. 관련 연구

2.1 컴포넌트 기반 소프트웨어 생산라인

소프트웨어의 재구성성을 지원하기 위해서는, 구체적인 소프트웨어 개발 시 기존의 소프트웨어 구성 부품을 재사용하여 조합하기 위한 체계적이고 효율적인 방법을 지원해야 한다. 소프트웨어 공학 분야에서 오래 전부터 재사용을 지원하기 위한 많은 연구가 진행되어 왔다. 1960년대의 서브루틴(subroutine), 1970년대의 모듈(module), 1980년대의 객체(object), 1990년대의 컴포넌트(component) 등으로 이어지는 개념들은 소프트웨어의 복잡성을 관리하고 재사용을 지원하기 위한 연구자들의 계속되는 노력을 보여준다.

'객체' 개념은 소프트웨어 개발 생산성의 획기적인 향상을 가져오기는 하였지만 현대 소프트웨어의 복잡성을 해결하고 효율적인 재사용을 지원하는데 한계가 있음이 드러났다. 재사용에 있어서 객체의 단점을 해결하기 위해 '컴포넌트' 기반의 소프트웨어 개발 방법론이 크게 각광을 받았다. 컴포넌트란, 객체보다 단위가 큰 개념으로서 서로 관련 있는 기능을 미리 정의된 인터페이스를 통해 제공하고 조합을 목적으로 개발된 소프트웨어 부품을 의미한다. 컴포넌트 패러다임이 소프트웨어의 재사용성을 많이 향상시킨 반면, 새로운 아키텍처나 기능적(functional) 또는 비기능적(non-functional) 요구 사항이 발생하였을 때나 다른 운영 환경에 동작시키도록 하려고 할 때 컴포넌트 재사용자의 직접적인 커스터마이징 또는 변형(adaptation)을 필요로 하는 단점이 있다. 이로 인해 컴포넌트의 복잡성 증가, 성능 하락, 비슷한 특성의 중복된 구현 등의 문제점이 유발되어 컴포넌트의 재사용성이 감소하는 단점이 발견되었다. 또한, 재사용자의 목적에 적합한 컴포넌트를 검색하는데 있어서도 많은 어려운 점이 나타났다.

이러한 문제점들을 해결하기 위하여 '소프트웨어 생산라인(Software Product Lines)'이라는 새로운 소프트웨어 개발 패러다임이 출현하게 되었다[3]. 소프트웨어 생산라인의 목적은, 미리 구축된 부품들이 소프트웨어 생산시 정해진 방식에 의해 조합됨으로써 비슷한 특성을 가지고 있지만 특정 부분이 서로 다른 일련의 다양한 소프트웨어들을 보다 빠르고 좋은 품질을 갖도록 생산하는데 있다. 소프트웨어 생산라인 기반의 개발 방법론 하에서는, 핵심 소프트웨어 자산으로부터 소프트웨어 부품들을 선택하여 인자화나 상속 등의 미리 정의된 변형 메커니즘(variation mechanism)을 이용하여 목적에 맞도록 수정하는 과정과, 이러한 부품들을 생산라인 전체를 제어하는 공통된 아키텍처에 따라 조립하는 과정을 통해 구체적인 시스템이 개발된다.

핵심 소프트웨어 자산 개발에 있어서 가장 중요한 점은, 차이점(variability)을 지원할 수 있어야 한다는 것이다. 차이점이란, 공통점을 공유하는 일련의 소프트웨어 부품들이 각 환경이나 목적에 맞게 변형될 필요가 있는 특성을 의미한다. 이는 소프트웨어 생산라인에 속하는 멤버들이 공유하는 공통점(commonality)과 반대되는 개념의 특성으로서 재구성성의 핵심 요소이다. 소프트웨어 생산라인 구축 시에는, 소프트웨어 패밀리들이 공유하는 공통점과 차이점을 개발 초기에서부터 식별하는 작업이 필요하다. 공통점과 차이점을 식별하기 위한 분석 방법으로 FODA(Feature-Oriented Domain Analysis)와 FORM (Feature-Oriented Reuse Method)[4], RESB(Reuse-Driven Software Engineering Business)[5,6], DARE (Domain Analysis and Reuse Environment)[7]와 같은 여러 가지 기법들이 제안되었다. 이러

한 기법들은 특성 모델을 기반으로 하여 재사용 가능한 소프트웨어 자산들을 분석하고 개발하기 위한 체계적인 방법들을 제시하였다.

이러한 방법론들의 핵심적인 구성 요소는 특성 모델(feature model)이며, 이는 재사용 가능한 자산들을 모델링하고 분석하는 도구로 사용되었다. 하지만, 이러한 특성 모델이 컴포넌트 재구성이나 코드 생성 프로세스에서 유용한 커스터마이징 도구로 이용하는 방법에 대해서는 거의 연구가 이루어지지 않았다. 본 논문에서는, 특성 모델의 주요 산물인 특성 다이어그램(feature diagram)을 컴포넌트 패밀리의 구성 공간을 표현하고 재사용 과정의 도구로 사용함으로써, 특성 모델이 소프트웨어 생산라인에서의 재사용 프로세스의 효율적 지원 도구로서 이용될 수 있음을 보였다.

2.2 자동 생성 프로그래밍과 XML 기술

소프트웨어 생산라인에서 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(generation) 등이 그것이다[8]. 프로그램 자동 생성에 관한 연구는 최근에 활발히 진행되기 시작하였으며, 여러 가지 프로그램 자동 생성을 지원하기 위한 분석 방법과 구현 방법이 제안되었다[9]. 일반적 프로그래밍(generic programming), 템플릿 클래스 기반의 프로그래밍, Aspect 지향 프로그래밍, 자동 생성기(generator), 정적 메타 프로그래밍, 의도적 프로그래밍(intentional programming) 등이 그것이다.

한편, 네트워크나 웹상에서의 데이터 교환을 목적으로 개발된 XML과 XSLT 기술은, 최근 활발히 연구되는 소프트웨어 자동 생성 프로그래밍(generative programming) 분야에서도 적용되기 시작하였다. [10]에서는 프로그램의 추상적 명세서를 XML을 이용하여 작성하고, 이를 받아들여 프로그램 코드를 자동 생성하는 프로그램 생성기를 제안하였다. [11]에서는, 컴포넌트 명세서 마법사와 컴포넌트 코드 마법사 도구를 이용하여 XML 기반의 구체적인 컴포넌트의 명세서를 만들고, 이를 받아들여 자바 빈즈 컴포넌트의 소스 코드를 자동 생성하는 기법에 관하여 연구하였다. [12]에서는 프러덕트 패밀리를 구성하는 자산들을 차이점을 수용하기 위한 방법으로 구조화하고 XML을 기반으로 한 명세서를 해석하여 자산의 커스터마이징을 반자동화(semi-automatic support)하는 도구에 대한 연구를 수행하였다. 이 기법은 프레임 기반의 방법과 도구를 이용하여 유연하고 단순한 형태로 관심 부분을 분리(separation of concern)할 수 있는 기능을 제공하지만, 재사용자가 컴포넌트 커스터마이징을 위한 스크립트를 직접 작성해야 하며, XVCL이라는 전용 언어를 컴포넌트 개발자와 재사용자가 알아야 하는 단점을 가지고 있다.

본 논문의 기법은, 컴포넌트 패밀리의 특성 다이어그램에 표현되어 있는 차이점들에 대하여 특정 값이나 선택을 제공하기만 하면, 컴포넌트 재사용자가 자신의 목적에 맞는 구체적인 컴포넌트의 소스 코드를 자동 생성하는 자동 생성기 기반의 재사용을 지원한다. 이 기법은 요구 사항의 변화가 적은 안정된 문제 영역에서의 소프트웨어 생산라인 구축에 유용하게 적용될 수 있다.

3. 리스트 컨테이너 컴포넌트 패밀리

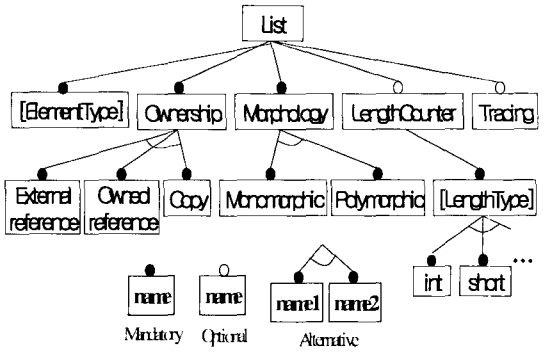
이 절에서는 본 논문의 코드 생성 기법을 보여주기 위해 사용한, 리스트 컨테이너 컴포넌트 패밀리의 도메인 분석 및 설계 결과를 기술한다. 이것은 [9]에서 C++ 템플릿 클래스를 이용한 자동 생성 프로그래밍 기법을 설명하기 위해 사용되었던 예제로서 본 논문의 사례 연구를 위해 도입되었다.

3.1 도메인 분석

도메인 분석은, 문제 영역의 범위를 정하고 문제

영역에 존재하는 컴포넌트들 사이의 공통점과 차이점 등을 분석하는 과정을 포함한다. 일반적으로 도메인 분석의 결과는 특성 모델로 표현되며 그 중에 가장 핵심적인 부분이 특성 다이어그램이다.

그림 1은 리스트 컨테이너에 대한 특성 다이어그램을 보여준다. 이 다이어그램에서의 여러 가지 특성들의 의미는 다음과 같다. ElementType은 리스트에 저장될 원소의 자료형을 나타낸다. 이것은 어떤 형태의 자료형도 이 특성의 값이 될 수 있기 때문에 free parameter라고 하며 각 괄호로 표시한다. Ownership은, 초기(original) 원소에 대한 레퍼런스를 가지고 있으며 원소의 메모리 해제에 대한 책임을 지지 않는지(external reference 특성), 또는 초기 원소에 대한 레퍼런스를 가지고 있으며 원소의 메모리 해제에 대한 책임을 지는지(owned reference 특성), 또는 초기 원소의 복사본을 가지고 있으며 원소의 메모리 할당 및 해제에 대한 책임을 지는지(copy 특성)를 나타내는 특성이다. Morphology는 리스트의 원소들이 같은 자료형인지(Monomorphic 특성) 또는 다른 자료형인지(Poloymorphic 특성)를 나타낸다. LengthCounter 특성은, 리스트가 길이에 대한 카운터를 가질 것인지에 대한 여부를 나타낸다. LengthType은 카운터의 자료형을 의미한다. Tracing 특성은, 리스트가 각 연산 호출에 대한 로깅 정보를 화면에 출력할 것인지에 대한 여부를 나타낸다. 이 특성 다이어그램은, 리스트 컨테이너 컴포넌트 패밀리에 존재하는 구체적인 리스트들을 생성하기 위한 구성 공간(configuration space)을 제공한다.

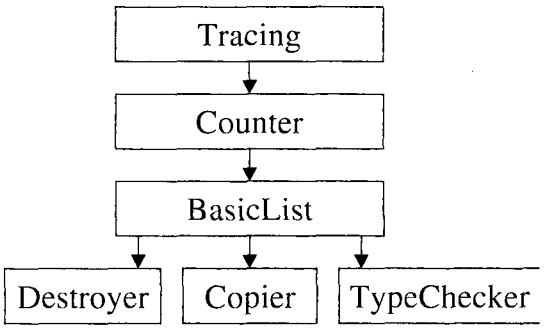


(그림 1) 리스트 컨테이너를 위한 특성 다이어그램

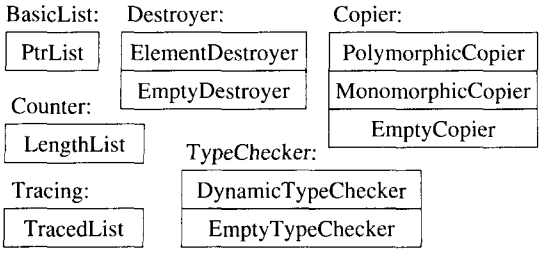
3.2 도메인 설계

도메인 설계는, 도메인 분석 결과를 기반으로 소프트웨어 패밀리를 구현하기 위해 필요한 컴포넌트들을 식별하고 아키텍처를 결정하는 과정을 포함한다. [9]에서는 GenVoca라고 불리는 계층적 아키텍처(layered architecture)를 이용하여 리스트 컨테이너 패밀리의 아키텍처를 정의하였다. GenVoca는 기본적으로 객체지향 아키텍처이며, 하나 또는 하나 이상의 클래스를 가지는 조정 가능한(configurable) 계층들의 집합으로 구성된다. GenVoca 아키텍처를 정의하기 위해서는 먼저 컴포넌트 카테고리(category)와 각 카테고리에 속하는 컴포넌트들을 식별한다. 그 다음, 식별된 컴포넌트 카테고리들 간의 "uses" 의존 관계를 파악한 후 이를 바탕으로 계층적 아키텍처에 맞도록 각 카테고리를 분류한다. 그림 2는 리스트 컨테이너를 위한 컴포넌트 카테고리들 사이의 "uses" 의존 관계를 보여주며, 그림 3은 각 컴포넌트 카테고리에 속하는 컴포넌트들을 보여준다.

본 논문에서는, 각 컴포넌트 카테고리에 속하는 컴포넌트들을 구현 컴포넌트(implementation component)라고 하며, 이들 구현 컴포넌트들을 이용해 완성되는 최종 컴포넌트인 구체적 리스트 컨테이너를 목표 컴포넌트(target component)라고 한다.



(그림 2) 리스트 컨테이너를 위한 계층적 구조



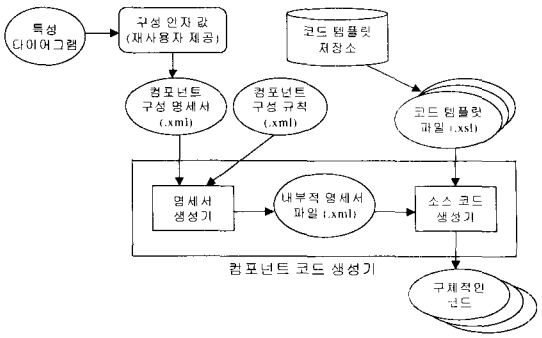
(그림 3) 리스트 컨테이너를 위한 컴포넌트 카테고리

4. 컴포넌트 코드 자동 생성

본 논문의 코드 생성 기법은, 구성 공간을 제공하는 특성 다이어그램을 이용하여 재사용자가 자신의 목적에 적합한 컴포넌트에 대한 구성 인자 값(customization data)을 입력하면 이를 바탕으로 XSLT의 문서 변형 기능을 이용하여 구체적인 컴포넌트에 대한 코드를 자동 생성한다. 본 논문의 컴포넌트 코드 생성 시스템의 전체적인 구조는 그림 4와 같다.

본 논문의 코드 생성 기법을 이용한 컴포넌트 재사용 프로세스는 다음과 같다.

1) 재사용자는 자동 생성하기를 원하는 컴포넌트가 속한 컴포넌트 패밀리를 결정한다(예: 리스트 컨테이너).
2) 컴포넌트 개발자에 의해 미리 정의된 특성 다이어그램으로부터, 재사용자는 구체적인 컴포넌트 생성에 필요한 필수적(mandatory), 선택적(optional), 택일적(alternative) 특성을 선택하고 free parameter 특성을 위한 특정 값을 입력하여, XML 형태의 컴포넌트 구성 명세서를 자동 생성한다.
3) 컴포넌트 코드 생성기는, 특정 컴포넌트 구성 명세서와 구성 규칙 파일을 바탕으로 목표 컴포넌트 생성에 필요한 구현 컴포넌트들에 대한 XSLT 스크립트 파일들을 선택한다.
4) 선택된 구현 컴포넌트 XSLT 스크립트 파일들로부터 구체적인 구현 컴포넌트 코드를 생성한 후, 목표 컴포넌트 XSLT 스크립트 파일로부터 최종 컴포넌트 코드를 자동 생성한다.



(그림 4) 컴포넌트 코드 생성 시스템의 전체적인 구조

이러한 재사용 프로세스를 지원하기 위해서는, 컴포넌트 개발 초기에 많은 시간과 노력이 필요하다. 본 논문에서의 컴포넌트 패밀리 개발 과정은 다음과 같다.

1) 도메인 분석 과정을 통하여 목표 컴포넌트에 대한 특성 다이어그램을 개발한다.
2) 도메인 설계 과정을 통하여 컴포넌트 카테고리(category)와 각 카테고리에 속하는 구현 컴포넌트들을 식별한다.
3) 구체적인 컴포넌트 생성 시 필요한 구현 컴포넌트의 종류를 명시하는 구성 규칙을 정의한다.
4) 각 컴포넌트 카테고리를 위한 XSLT 스크립트 파일을 개발한다.
5) 목표 컴포넌트를 위한 XSLT 스크립트 파일을 개발한다.

다음 절들에서는 컴포넌트 코드 생성 시스템을 구성하는 각 구성 요소에 대하여 자세히 기술한다.

4.1 구성 공간을 위한 특성 다이어그램

컴포넌트 패밀리를 위한 특성 다이어그램은, 컴포넌트 멤버들 사이의 공통점(commonalities)과 차이점(variabilities)들을 표현한다. 그림 1에서, 리스트 컨테이너 패밀리의 멤버들 사이의 공통점은, 필수적 특성들에 의해 표현되며, 차이점들은 선택적 특성과 택일적 특성들에 의해 표현된다. 차이

점들을 표현하는 특성들에 의해 리스트 컨테이너의 구성 공간이 정의되며, 재사용자는 이러한 특성 다이어그램을 통해서 자신의 목적에 맞는 컴포넌트를 생성하기 위한 커스터마이징 데이터를 입력한다. 재사용자의 컴포넌트 커스터마이징 과정은, 특성의 유형에 따라 다음과 같은 세 가지 종류의 작업을 포함한다.

1) Free parameter 특성에 대해서는, 재사용자가 필요한 실제 값을 제공된다.
2) 선택적 특성에 대해서는, 재사용자가 이 특성을 선택할지 선택하지 않을지를 결정된다.
3) 택일적 특성들을 자식으로 가지는 특성에 대해서는, 재사용자가 택일적 자식 특성들 중에 하나를 선택한다.

4.2 컴포넌트 구성 명세서를 위한 XML 파일

특성 다이어그램을 통한 재사용자의 커스터마이징 결과는 XML 형식의 컴포넌트 구성 명세서로 변환된다. 컴포넌트 구성 명세서는, 목표 컴포넌트의 이름, 필수적 특성과 선택된 택일적 특성의 이름, free parameter 특성의 이름과 실제 값 등에 대한 정보를 포함한다. 이 명세서는 재사용자의 목적에 적합한 구체적 컴포넌트를 생성하기 위한 여러 가지 정보를 제공한다. 그림 5는 구성 명세서 파일을 위한 DTD(Document Type Definition)을 보여준다. 특성 다이어그램에서의 노드와 그것의

```
<!ELEMENT Tree (Root, Node*)>
<!ELEMENT Root (#PCDATA)>
<!ELEMENT Node (name, type, parent, value?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT parent (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

(그림 5) 컴포넌트 구성 명세서를 위한 DTD

타입, 트리 구조 등을 표현하기 위한 엘리먼트들이 정의되어 있음을 알 수 있다.

그림 6은 하나의 특정 리스트 컨테이너 컴포넌트의 구성 명세서의 일부분을 보여준다. 이 구성 명세서를 생성하기 위해 재사용자가 실행하였던 커스터마이징 과정의 일부는 다음과 같다.

1) free parameter 특성인 ElementType의 값을 Base로 설정하였다.
2) 필수적 특성인 Morphology의 택일적 자식 특성들 중에서 Monomorphic 특성을 선택하였다.
3) 선택적 특성인 LengthCounter를 선택하였다.

4.3 컴포넌트 구성 규칙을 위한 XML 파일

컴포넌트 구성 규칙은, 특성 다이어그램의 여러 가지 특성들과 구현 컴포넌트들 사이의 관계를 XML 형식으로 정의한다. 즉, 목적 컴포넌트를 생성하기 위해서 재사용자가 특성 다이어그램에서 선택한 특성들에 따라 어떤 구현 컴포넌트들이

```
<Tree>
  <Root>List</Root>
  <Node>  <name>ElementType</name> <type>Mandatory</type>
    <parent>List</parent>  <value>Base</value>  </Node>
... ...
  <Node>  <name>Morphology</name> <type>Mandatory</type> <parent>List</parent>  </Node>
  <Node>  <name>Monomorphic</name> <type>Alternative</type> <parent>Morphology</parent>  </Node>
  <Node>  <name>LengthCounter</name> <type>Optional</type> <parent>List</parent>  </Node>
... ...
</Tree>
```

(그림 6) 리스트 컨테이너 구성 명세서

```
<!ELEMENT Config (category*)>
<!ELEMENT category (ImplCom*)>
<!ELEMENT ImplCom (OR_cond*)>
<!ELEMENT OR_cond (AND_cond*)>
<!ELEMENT AND_cond (#PCDATA)>
<!ATTLIST category name CDATA #REQUIRED>
<!ATTLIST ImplCom name CDATA #REQUIRED>
```

(그림 7) 컴포넌트 구성 규칙을 위한 DTD

```
<!DOCTYPE Config SYSTEM
                "file:Configuration_Rule.dtd">

<Config>
  <category name="TypeChecker">
    <ImplCom name="DynamicTypeChecker">
      <OR_cond>
        <AND_cond>Monomorphic</AND_cond>
      </OR_cond>
    </ImplCom>
      <ImplCom name="EmptyTypeChecker">
        <OR_cond>
          <AND_cond>Polymorphic</AND_cond>
        </OR_cond>
      </ImplCom>
  </category>
... ...
</Config>
```

(그림 8) 리스트 컨테이너를 위한 컴포넌트 구성 규칙

포함될 것인가에 대한 규칙을 가진다. 컴포넌트 구성 규칙에 대한 DTD는 그림 7과 같다. 컴포넌트 카테고리와, 어떤 특성들이 선택되었을 때 어떤 구현 컴포넌트가 포함될 것인가에 대한 조건을 표현하기 위한 여러 가지 엘리먼트들이 정의되어 있다.

그림 8은 리스트 컨테이너를 위한 컴포넌트 구성 규칙의 일부분을 보여준다. 이 파일은, 컴포넌트 재사용자가 Monomorphic 특성을 선택하면 DyanmicTypeChecker 구현 컴포넌트가, Polymorphic 특성을 선택하면 EmptyTypeChecker 구현 컴포넌트가 포함되어야 한다는 규칙을 포함하고 있음을 알 수 있다.

4.4 구현 및 목표 컴포넌트를 위한 XSLT 스크립트

본 논문의 기법에서, 구현 컴포넌트와 목표 컴포넌트에 대한 코드 부분을 XSLT 스크립트 파일이 제공하며, XSLT 스크립트 파일은 다음과 같이 크게 두 가지 중요한 기능을 제공한다.

1) 코드 템플릿 기능: 컴포넌트 코드 생성 시 필요한 구현 컴포넌트 및 목표 컴포넌트의 코드 골격을 제공한다.
2) 코드 선택 기능: 재사용자가 제공한 커스터마이징 데이터를 바탕으로, 각 컴포넌트 카테고리에 속하는 구현 컴포넌트들 중 특정 구현 컴포넌트와 관련된 코드 부분을 선택하는 기능을 제공한다.

XSLT 스크립트 파일은 크게 다음과 같이 두 가지 종류로 나누어진다.

1) 각 컴포넌트 카테고리를 위한 XSLT 스크립트파일
2) 목표 컴포넌트를 위한 XSLT 스크립트 파일

그림 9는 TypeChecker 컴포넌트 카테고리를 위한 XSLT 스크립트 파일의 일부분을 보여준다. 이 파일에는 TypeChecker 컴포넌트 카테고리에 속하는 구현 컴포넌트들이 공통으로 가지는 코드 부분이 포함되어 있으며, <xsl:choose> 엘리먼트에 의해 특정 구현 컴포넌트와 관련된 코드가 선택될 수 있도록 하는 기능이 포함되어 있음을 알 수 있다. 그림 10은, 목표 컴포넌트인 리스트 컨테이너를 위한 XSLT 스크립트 파일을 보여준다.

4.5 컴포넌트 코드 생성기

컴포넌트 코드 생성기는, 구체적 컴포넌트에 대한 구성 명세서와 구성 규칙을 읽어 들여 필요한 정보를 추출하여 내부적인 명세서 파일을 만든다.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="Specification">
class <xsl:value-of select="Include_Com/Component[@category='TypeChecker']"/> {
  public <xsl:value-of select="Include_Com/Component[@category='TypeChecker']"/>() {}
  static void check(<xsl:value-of select="Features/Feature[name='ElementType']//value"/> e) {
   <xsl:choose>
    <xsl:when test="Include_Com/Component[@category='TypeChecker'] = 'DynamicTypeChecker'">
    String s = e.getClass().getName();
    if(s.equals("<xsl:value-of select="Features/Feature[name='ElementType']//value"/>")) {
      System.out.println("In DynamicTypeChecker : ok!!");
    } else {
      System.out.println("e : " + e );
      System.out.println("Type Mismatching!! It's Mono!!! ");
      System.exit(0);
    }
    </xsl:when>
    <xsl:when test="Include_Com/Component[@category='TypeChecker'] = 'EmptyTypeChecker'">
    </xsl:when>
  </xsl:choose>
  }
}
</xsl:template>
</xsl:stylesheet>
```

(그림 9) TypeChecker 컴포넌트 카테고리를 위한 XSLT 스크립트 파일

그 다음, XSLT 프로세서의 도움을 받아 내부적인 명세서 파일과 필요한 구현 컴포넌트 및 목표 컴포넌트를 위한 XSLT 스크립트 파일들로부터 특정 컴포넌트에 대한 소스 코드를 자동으로 생성한다.

본 논문에서 제안한 컴포넌트 코드 생성 기법은, 소프트웨어 생산라인의 핵심 자산인 재사용 가능한 컴포넌트를 개발하는데 있어서 다음과 같은 여러 가지 장점을 제공한다.

◎ 목표 컴포넌트의 개념과 기능을 보다 추상적인 특성 다이어그램을 통해서 볼 수 있기 때문에, 컴포넌트 재사용자는 컴포넌트의 내부적인 구현에 대한 지식 없이 컴포넌트의 의미(semantics)를 쉽게 이해할 수 있다.

◎ 목표 컴포넌트에 대한 특성 다이어그램과 구성 명세서를 XML로 표현함으로써 컴포넌트 정보에 대한 표준화를 가능하게 해 준다. 따라서, 컴포넌트 정보의 효율적인 저장과 검색이 가능해 진다.

◎ 목표 컴포넌트를 생성함에 있어 컴포넌트 카테고리나 구현 컴포넌트의 종류가 바뀌는 경우, 컴포넌트의 구성 규칙 파일만을 수정함으로써 쉽게 대처할 수 있다.

◎ XSLT 스크립트 파일에 추가의 코드를 삽입함으로서, 목표 컴포넌트나 구현 컴포넌트의 기능을 쉽게 확장할 수 있다.

◎ 자동 생성되는 결과물이 소스 코드 형태이므로, 상속(inheritance)이나 그 밖의 방식을 이용하여 재사용자는 생성된 코드를 쉽게 확장할 수 있다.

◎ 코드 생성기의 기능을 확장함으로써, 코드 생성 과정에 컴포넌트 테스트 코드 생성 등과 같은 추가의 기능을 덧붙일 수 있다.

```
<xsl:template match="Specification">
class <xsl:apply-templates select="Features/Feature"/> {
<xsl:value-of select="Include_Com/Component[@category='TypeChecker']"/> TypeChecker = new
<xsl:value-of select="Include_Com/Component[@category='TypeChecker']"/>();
 ... ...
<xsl:apply-templates select="Features"/> head_ = null;
<xsl:apply-templates select="Class_Name"/> tail_;
public <xsl:apply-templates select="Features/Feature"/>(<xsl:apply-templates select="Features"/> h) {
      setHead(h);    tail_ = null;  }
... ...  }
<xsl:if test="count(Features/Feature[name='LengthCounter']) = 1">
class <xsl:choose>
<xsl:when test="count(Features/Feature[name='Tracing']) = 1">LenList</xsl:when>
  <xsl:otherwise><xsl:apply-templates select="Class_Name"/></xsl:otherwise>
  </xsl:choose> extends BaseList {
<xsl:value-of select="Features/Feature[name='LengthType']//value"/> length_;
public <xsl:choose>
          <xsl:when test="count(Features/Feature[name='Tracing']) = 1">LenList</xsl:when>
          <xsl:otherwise><xsl:apply-templates select="Class_Name"/></xsl:otherwise>
          </xsl:choose>(<xsl:apply-templates select="Features"/> h) {
             super(h);
             length_ = computedLength();
          }  ... ...   }
</xsl:if>
<xsl:if test="count(Features/Feature[name='Tracing']) = 1">
 ... ...
</xsl:if>
</xsl:template>
<xsl:template match="Features/Feature">
   <xsl:choose>
    <xsl:when test="(count(self::node()[name='Tracing']) = 1 and count(self::node()[name='LengthCounter']) = 1)
         or (count(self::node()[name='Tracing']) = 0 and count(self::node()[name='LengthCounter']) = 1)">BaseList</xsl:when>
    <xsl:otherwise><xsl:apply-templates select="Class_Name"/>
    </xsl:otherwise>  </xsl:choose>
</xsl:template>
<xsl:template match="Class_name"> <xsl:value-of select="."/> </xsl:template>
<xsl:template match="Features">  <xsl:value-of select="./Feature[name='ElementType']//value"/>   </xsl:template>
</xsl:stylesheet>
```

(그림 10) 리스트 컨테이너 목표 컴포넌트를 위한 XSLT 스크립트 파일

5. 결론 및 향후 연구

본 논문에서는, 특성 다이어그램과 XML/XSLT 기술을 이용하여 컴포넌트 코드 생성 시에 재구성성을 지원하는 컴포넌트 코드 자동 생성 기법을 제안하였다. 본 논문의 기법은, 컴포넌트 패밀리의 특성 다이어그램에 표현되어 있는 차이점들에 대하여 특정 값이나 선택을 제공함으로써 컴포넌트 재사용자가 자신의 목적에 맞는 구체적인 컴포넌트의 소스 코드를 자동으로 생성할 수 있게 해 준다. 이를 위하여 컴포넌트 개발자는 컴포넌트 요구 사항이나 기능들을 바탕으로 하나의

컴포넌트 패밀리 안에 존재하는 컴포넌트들이 가지는 공통점과 차이점들을 분석하여 특성 다이어그램으로 표현한다. 그리고, 컴포넌트 카테고리와 구현 컴포넌트, 목표 컴포넌트들의 종류를 분석하고 이들을 위한 XSLT 스크립트 파일을 개발한다.

본 논문에서는 사례 연구로서 리스트 컨테이너 패밀리에 대한 컴포넌트 코드 시스템을 개발함으로써 특성 모델과 XML 관련 기술의 자동 생성 프로그래밍 영역에서의 적용 가능성을 보였다.

본 논문의 컴포넌트 코드 생성 기법은, 컴포넌트 자동 생성을 통한 구성 용이성을 지원함으로서 컴포넌트 재사용 시 생산성을 높여주며, XML이 가지는 다양한 장점들(도메인 지식의 표준 형식, 자기 기술적인 특징 등)을 제공한다. 향후 과제로서, 테스트 코드 자동 생성, 보다 큰 단위의 컴포넌트 자동 생성, 컴포넌트 패밀리 개발 지원 도구 등에 대한 연구가 필요하다.

Acknowledgement

본 연구는 2001학년도 덕성여자대학교 자연과학연구소 연구비 지원으로 이루어졌음.

참 고 문 헌

[1] C. Atkinson et al., Component-based Product Line Engineering with UML, Addison-Wesley, 2002.

[2] Elaine J. Weyuker, "The Trouble with Testing Components", pp499-512, in "Component-Based Software Engineering: Putting the Pieces Together" (Ed. George T. Heineman, William T. Councill), Addison-Wesley, 2001.

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

[4] Kyo C. Kang et al., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", Annals of Software Engineering, 5, pp. 143~168(1998).

[5] I. Jacobson, M. Griss and P. Jonsson, "Software Reuse: Architecture, Process and Organization for Business Success", Addison-Wesley, Reading, MA, May 1997.

[6] M. L. Griss, J. Favaro and M. d'Alessandro, "Integrating Feature Modeling with the RSEB", Proceedings of the Fifth International Conference on Software Reuse, IEEE Computer Society Press, 1998.

[7] W. Frakes, R. Prieto-Diaz, and C. Fox, "DARE: Domain Analysis and Reuse Environment", April 7, 1996.

[8] T. J. Biggerstaff, "A Characterization of Generator and Component Reuse Technologies", Proceedings of Third International Conference, GCSE(Generative and Component-Based Software Engineering), 2001.

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

[10] J. C. Cleaveland, Program Generators with XML and Java, Prentice Hall, 2001.

[11] Miok kwon and Seung-Hoon Choi, "Component Generating System based on XML", Proceedings of Korean Society For Internet Information Conference, 2001.

[12] H. Zhang, S. Jarzabek and S. M. Swe, "XVCL Approach to Separating Concerns in Product Family Assets", Proceedings of Third International Conference, GCSE(Generative and Component-Based Software Engineering), 2001.

◐ 저 자 소 개 ◑

최 승 훈

1990년 서울대학교 계산통계학과 졸업(학사)
1994년 서울대학교 대학원 계산통계학과 졸업(석사)
1999년 서울대학교 대학원 계산통계학과 졸업(박사)
1999년 8월~2000년 2월 삼성전자 선임연구원
2000년~현재 : 덕성여자대학교 전산학과 교수
관심분야 : 컴포넌트기반 소프트웨어공학, 소프트웨어 생산라인, 자동 생성 프로그래밍, etc.
E-mail : csh@duksung.ac.kr