CompGenX: GenVoca와 XML 기반의 컴포넌트 코드 생성 시스템

CompGenX: Component Code Generation System based on GenVoca and XML

최 승 훈*
Seung-Hoon Choi

요 약

소프트웨어 생산 라인은, 소프트웨어 자산에 존재하는 일반적인 컴포넌트를 구체화하고 미리 정의된 아키텍처를 기반으로 조립함으로써, 고품질의 응용 프로그램을 빠르게 개발할 수 있도록 해 주는 것을 목적으로 한다. 본 논문에서는, 소프트웨어 생산 라인 구축을 지원하기 위하여 GenVoca 아키텍처와 XML/XSLT 기술을 이용한 컴포넌트 코드 자동 생성 기법과, 이를 기반으로 한 컴포넌트 코드 생성 시스템인 CompGenX(Component Generator using XML)를 제안한다. CompGenX은 컴포넌트 코드 생성 시에 컴포넌트의 재구성성을 지원함으로써 재사용자가 컴포넌트 재사용 시 자신의 목적에 맞는 컴포넌트의 소스 코드를 자동으로 생성할 수 있게 해 준다. 이를 위한 컴포넌트 개발 과정은 크게 컴포넌트 패밀리 구축 과정과 컴포넌트 재사용 과정으로 나누어진다. CompGenX는 컴포넌트 패밀리 구축 과정을 위하여, 도메인 분석을 위한 특성 모델 작성 도구, 도메인 설계를 위한 도메인 아키텍처 정의 도구를 제공한다. 또한 코드 자동 생성에 필요한 구성 지식 명세서와 코드 템플릿 작성 도구를 제공한다. 컴포넌트 재사용 과정을 위해서 CompGenX는, 컴포넌트 패밀리 검색 도구, 컴포넌트 커스터마이징 도구 및 컴포넌트 코드 생성기 등을 제공한다. 본 논문의 컴포넌트 코드 자동 생성 기법과 생성 시스템은 컴포넌트 기반 소프트웨어 생산 라인 구축을 위한 기반 기술로서 적용될 수 있다.

Abstract

Software product lines are to attain the rapid development of quality applications by concretizing the general components populated in software assets and assembling them according to the predefined architectures. For supporting the construction of the software product lines, this paper proposes a component code generation techniques based on GenVoca architecture and XML/XSLT technologies. In addition, CompGenX(Component Generator using XML), a component code generation system, is proposed on the basis of this techniques. By providing reconfigurability of component at the time of code generation, CompGenX allows the reusers to create the component source code that is appropriate to their purpose. In this system, the process of the component development is divided into two tasks which are the component family construction task and the component reuse task. For the component family construction, CompGenX provides the feature modeling tool for domain analysis and the domain architecture definition tool. Also, it provides the tool for building the component configuration knowledge specification and the code templates. For the component reuse task, it offers the component family search tool, the component customizing tool and the component code generator. Component code generation techniques and system in this paper should be applicable as basic technology to build the component-based software product lines.

☞ Key words : Software Product Lines, Component, GenVoca, XML/XSLT, Reconfigurability, Automatic Code Generation

1. 서 론

최근 소프트웨어 개발비용의 절감과 신뢰성 있는 소프트웨어 개발을 위하여 컴포넌트 기반의 소프트웨어 생산라인(Software Product Line)에 대한 연구가 활발히 진행되고 있다[1]. 소프트웨어 생산라인의 목적은, 미리 구축된 부품들이 소프트웨어 생산시 정해진 방식에 의해 조립됨으로써

* 정회원 : 덕성여자대학교 컴퓨터과학부 교수
csh@duksung.ac.kr(제1저자)

비슷한 특성을 가지고 있지만 특정 부분이 서로 다른 일련의 다양한 소프트웨어들을 보다 빠르게 생산하는데 있다.

소프트웨어 생산 라인 기반의 개발 방법론 하에서는, 핵심 소프트웨어 자산으로부터 소프트웨어 부품들을 선택하여 인자화나 상속 등의 미리 정의된 변형 메커니즘을 이용하여 목적에 맞도록 수정하는 과정과, 이러한 부품들을 생산 라인 전체를 제어하는 공통된 아키텍처에 따라 조립하는 과정을 통해 구체적인 시스템이 개발된다.

핵심 소프트웨어 자산의 개발을 위한 도메인 공학은 도메인 분석, 도메인 아키텍처 정의, 이를 통한 재사용 가능한 자산 개발을 포함하며, 이때 가장 중요한 점은, '가변성(variabiity)'을 지원할 수 있어야 한다는 것이다. 가변성이란, 공통점을 공유하는 일련의 소프트웨어 자산들이 각 환경이나 목적에 맞게 변형될 필요가 있는 특성을 의미하는 것으로, 실제 구체적인 소프트웨어를 개발하기 위한 컴포넌트 재구성성의 핵심 요소라고 할 수 있다. 따라서 소프트웨어 생산 라인 구축 시에는, 컴포넌트 패밀리들이 공유하는 공통성과 가변성을 개발 초기부터 식별하는 작업이 필요하다. 이를 위한 도메인 분석 방법으로 FODA(Feature-Oriented Domain Analysis), FORM(Feature-Oriented Reuse Method)[2], RSEB(Reuse-Driven Software Engineering Business)[3], DARE(Domain Analysis and Reuse Environment)[4] 등과 같은 여러 가지 기법들이 제안되었다. 이러한 기법들은, 특성 모델(Feature model)을 기반으로 하여 재사용 가능한 소프트웨어 자산들을 분석하고 개발하기 위한 체계적인 방법들을 제시한다.

이러한 기존의 기법들에서는 문제 영역의 분석 도구로서 특성 모델을 연구한 반면, 컴포넌트 재구성이나 코드 생성 과정에서의 커스터마이징 도구로 특성 모델을 이용하는 방법에 대해서는 거의 연구가 이루어지지 않았다. 본 논문에서 제안하는 컴포넌트 코드 자동 생성 기법은 특성 모델을 통하여 컴포넌트 패밀리의 구성 공간을 표현하고 재사용 과정의 도구로 사용함으로써, 특성 모델이 소프트웨어 생산 라인의 재사용 프로세스의 효율적인 지원 도구로 이용될 수 있음을 보인다.

소프트웨어 생산 라인에서 구체적인 시스템을 생산하기 위한 기법으로 여러 가지가 존재한다. 이 중 프로그램 자동 생성에 관한 연구는 최근 활발히 진행되기 시작하였으며, 일반적 프로그래밍(generic programming), 템플릿 기반의 프로그래밍, 자동 생성기(generator) 등과 같은 여러 가지 프로그램 자동 생성을 지원하기 위한 분석 방법과 구현 방법이 제안되었다. 이 중에서 GenVoca 생성 기법은 객체 지향적 계층 조립을 바탕으로 컴포넌트 자동 생성기를 개발하기 위한 접근 방법이다[5][6]. 이 기법은 C++ 템플릿 클래스를 이용하여 컴포넌트 코드 생성기를 구현한다. 이를 위해, 먼저 컴포넌트 패밀리를 위한 특성 모델과 계층 구조를 정의하고, 특정 계층의 템플릿 클래스는 하위 계층을 나타내는 클래스를 인자로 받아들여, 이를 상속받는 것에 의해 계층 조립을 구현한다.

본 논문의 컴포넌트 코드 생성 기법은 GenVoca 생성기를 기반으로 한다. 그러나, 특성 모델을 단순한 분석 모델링 도구로 사용한 GenVoca 생성기와는 달리, 본 연구의 코드 생성 기법에서는 특성 모델을 구체적인 컴포넌트의 커스터마이징 도구로 이용한다. 또한, GenVoca는 컴포넌트 생성에 필요한 구성 지식(configuration knowledge)을 템플릿 클래스에 포함하고 있으므로, 이를 수정하면 전체 코드에 영향을 미치게 된다. 그러나, 본 연구의 기법은 컴포넌트 코드와 분리된 XML 형태의 구성 지식 명세서를 정의함으로써 수정이 용이하고, 보다 다양한 구조와 형태의 컴포넌트 생성이 가능하다. 또한, 컴포넌트의 실제 소스 코드가 생성된다는 점에서 GenVoca 생성기와 다르다.

한편, 네트워크나 웹상에서의 데이터 교환을 목적으로 개발된 XML과 XSLT 기술은 최근 소프트웨어 자동 생성 프로그래밍 분야에서도 적용되기 시작하였다. [7]에서는 프로그램의 추상적 명

세서를 XML을 이용하여 작성하고, 이를 받아들여 프로그램 코드를 자동 생성하는 프로그램 생성기를 제안하였다. [8]에서는, 프러덕트 패밀리를 구성하는 자산들의 가변성을 수용하기 위하여 컴포넌트를 프레임 형태로 구조화하고 XML을 기반으로 한 명세서를 해석하여, 자산의 커스터마이징을 반자동화하는 도구에 대한 연구를 수행하였다. 이는 컴포넌트 자동 생성을 위한 유연한 방법을 제공해주지만, 반드시 XVCL 프로세서가 제공되는 제한적인 환경에서만 실행 가능하며, 컴포넌트 개발자나 재사용자가 실제 이를 적용하기 위해서는 XVCL 언어의 문법을 습득해야 하는 불편함이 있다.

본 논문에서는, 소프트웨어 생산 라인에 존재하는 핵심 소프트웨어 자산의 효율적인 개발을 지원하기 위한 컴포넌트 패밀리 구축 과정 및 컴포넌트 재사용 과정을 기술하고 이를 지원하는 CompGenX 컴포넌트 생성 시스템을 제안한다. 컴포넌트 코드 생성 기법은 특성 모델과 GenVoca 계층 구조를 기반으로 하며, XML 형태의 컴포넌트 명세서와 XSLT 코드 템플릿을 이용하여 구체적인 컴포넌트의 코드를 자동 생성한다.

CompGenX는 컴포넌트 코드 자동 생성을 위한 컴포넌트 패밀리 구축 과정과 재사용자에 의한 컴포넌트 재사용 과정을 지원한다. 이는 특정 컴포넌트 패밀리에 대한 도메인 분석 및 설계 과정을 지원하기 위한 특성 모델 작성 도구와 도메인 아키텍처 정의 도구를 제공한다. 또한 코드 자동 생성에 필요한 구성 지식과 코드 템플릿을 작성하기 위한 도구를 제공함으로써, 컴포넌트 개발자가 보다 쉽게 컴포넌트 패밀리를 구축할 수 있도록 도와준다. 본 논문의 컴포넌트 생성 기법은 컴포넌트 기반의 소프트웨어 생산라인 구축의 기반 기술로서 이용될 수 있으며, 보다 큰 단위의 컴포넌트 생성을 위한 기법으로 확장 가능하다.

본 논문의 구성은 다음과 같다. 제 2장에서는 연결 리스트 컴포넌트를 예제로 컴포넌트 패밀리 구축 과정을 자세히 기술하고, 제 3장에서 CompGenX 컴포넌트 생성 도구의 구조와 구성을 설명한다. 제 4장에서는, 사례 연구로서 CompGenX를 이용한 구체적인 연결 리스트 생성 과정을 보이고, 마지막으로 제 5장에서 결론과 향후 과제를 기술한다.

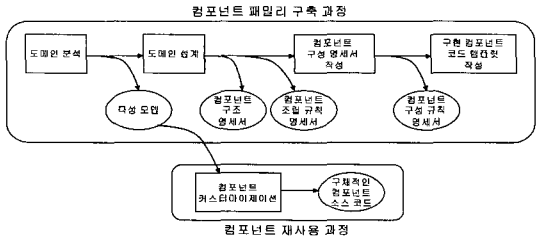
2. 컴포넌트 패밀리 구축

본 논문의 컴포넌트 자동 생성 기법은 크게 컴포넌트 패밀리 구축 과정과 컴포넌트 재사용 과정으로 나누어진다(그림1 참조). 컴포넌트 패밀리 구축 과정은 도메인 분석 및 설계 과정, 조립 규칙 작성 과정 및 코드 템플릿 작성 과정 등 여러 가지 과정을 포함한다. 반면, 컴포넌트 재사용 과정은 특성 모델에 표현된 가변성에 대한 특정 값이나 선택을 제공하는 컴포넌트 커스터마이제이션 과정만을 포함한다.

본 장에서는, [5]에서 parametric polymorphism과 C++ 템플릿 클래스를 이용하여 컴포넌트 자동 생성을 구현했던 연결 리스트를 예제로 하여, 본 논문의 컴포넌트 자동 생성 기법을 위한 컴포넌트 패밀리 구축 과정을 자세히 기술한다.

2.1 도메인 분석

도메인 분석 단계는 도메인 요구 사항을 인식하여 문제 영역의 범위를 정하고 문제 영역에 존재하는 컴포넌트들 사이의 공통성과 가변성 등을 분석하는 과정을 포함한다. 일반적으로 도메인 분석의 결과는 특성 모델(feature model)로 구조화되며, 이 때 컴포넌트 패밀리가 가지는 공통성은 필수적 특성으로, 가변성은 선택적 또는 택일적 특



(그림 1) 컴포넌트 자동 생성을 위한 전체 프로세스

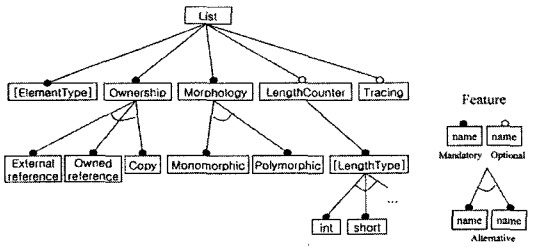
성으로 표현된다. 그림 2는 연결 리스트 패밀리를 위한 특성 모델을 보여준다.

그림 2에서, ElementType은 리스트에 저장될 원소의 자료형을 나타내며, Ownership은 리스트를 구성하는 원소에 대한 소유권을 의미한다. Morphology는 생성된 리스트가 서로 다른 형의 원소를 포함할 수 있는지에 대한 여부를 나타내며, Length Counter는 리스트가 길이에 대한 카운터를 가질 것인지에 대한 여부를 나타낸다. Tracing은 리스트가 각 연산 호출에 대한 로깅 정보를 화면에 출력할 것인지에 대한 여부를 나타낸다. 이와 같이 특성은 구체적인 컴포넌트의 기능과 형태에 직접적인 영향을 미칠 수 있는 중요한 특징을 의미하는 것으로 컴포넌트의 요구 사항을 의미한다.

각 특성의 실제 값은 그것의 유형에 따라 다음과 같은 세 가지 방법으로 재사용자에 의해 정해진다.

(1) 선택적 특성에 대해서는, 재사용자가 이 특성을 선택할지 선택하지 않을지를 결정한다. 그림 2의 예제에서는 LengthCounter, Tracing 특성 등이 이에 해당한다.
(2) 택일적 특성들을 자식으로 가지는 특성에 대해서는, 재사용자가 택일적 자식 특성들 중에 하나를 선택한다. 그림 2에서 Ownership, Morphology 특성 등이 이에 해당한다.
(3) ElementType과 같은 비결정적 인자(free parameter) 특성에 대해서는, 재사용자가 실제 값을 제공한다.

따라서, 특성 모델은 구체적인 컴포넌트 구성을 위한 정확한 결정 영역(decision space)을 나타내며, 특성 모델의 한 인스턴스는 구체적인 기능을 가진 하나의 컴포넌트를 의미한다.

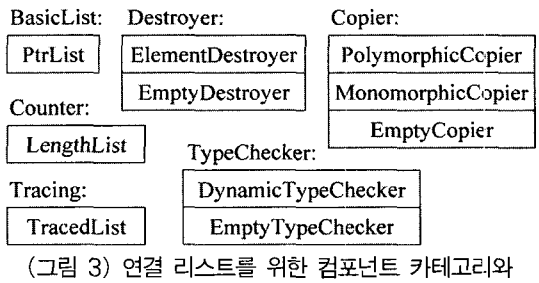


(그림 2) 연결 리스트 패밀리를 위한 특성 모델

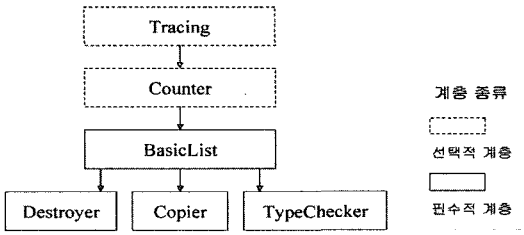
2.2 도메인 설계

도메인 설계의 목적은 도메인 분석 결과를 바탕으로 구현 컴포넌트들을 식별하고, 컴포넌트 패밀리를 위한 아키텍처를 개발하여 구체적인 컴포넌트 생성에 대한 계획을 세우는데 있다. 본 논문에서는 GenVoca 생성기를 바탕으로 컴포넌트 패밀리를 위한 계층적 아키텍처(layered architecture)를 정의하고, 이를 위한 구성, 조립 방법을 정의함으로써 구체적인 컴포넌트 생성을 위한 기반을 마련한다.

GenVoca는 기본적으로 객체지향 아키텍처이며, 하나 또는 하나 이상의 클래스를 가지는 조정 가능한 계층들의 집합으로 구성된다. GenVoca 아키텍처를 정의하기 위해서는 먼저 포함될 수 있는 컴포넌트 카테고리와 각 카테고리별 구현 컴포넌트들을 식별한다. 그 다음, 식별된 컴포넌트 카테고리들 간의 "use" 의존 관계를 파악한 후, 보다 일반적인 계층은 하위에 위치시키고, 세분화된 계층은 상위에 위치시켜 일련의 계층적 아키텍처를 형성한다. 그림 3은 연결 리스트에 포함될 수 있는 컴포넌트 카테고리와 카테고리별 구현 컴포넌트를 나타낸다. 그림 4는 이들 간의 의존 관계를 바탕으로 정의된 연결 리스트의 계층적 구조를 나타낸다.



(그림 3) 연결 리스트를 위한 컴포넌트 카테고리와 카테고리별 구현 컴포넌트



(그림 4) 연결 리스트를 위한 계층적 아키텍처

계층 구조에서 각 계층은 필수적 계층과 선택적 계층으로 분류된다. 필수적 계층은 컴포넌트 패밀리가 가지는 공통된 기능을 지원할 수 있는 컴포넌트 카테고리를 의미하는 것으로 구체적인 컴포넌트 생성 시 반드시 포함되어야 하는 계층을 말한다. 선택적 계층은 재사용자의 요구사항에 따라 포함 여부가 결정되는 계층으로, 그림 4의 계층 구조에서 점선으로 표현되어 있다. 각 계층은 표준화된 인터페이스를 제공한다. 따라서 같은 컴포넌트 카테고리에 포함된 구현 컴포넌트들은 모두 같은 인터페이스를 제공하며, 각 구현 컴포넌트의 특성에 따라 다른 구현을 제공한다. 구체적인 컴포넌트 생성 시, 특정 계층은 그것의 바로 하위에 위치한 계층에만 접근 가능하며, 이를 정제, 세분화시킴으로서 계층 조립이 일어난다.

2.3 컴포넌트 구성 명세서 정의

컴포넌트 코드 자동 생성을 위한 내부 지식으로는 컴포넌트 구조에 대한 지식, 컴포넌트 조립 규칙, 컴포넌트 구성 규칙 등이 있다. 본 논문에서 제안하는 코드 생성 기법에서, 이들은 모두 XML 형태의 명세서로 표현된다.

2.3.1 컴포넌트 구조 명세서

컴포넌트 패밀리를 위한 계층 구조를 표현한 XML 형태의 명세서이다. 여기에는 각 계층을 이루는 컴포넌트 카테고리와 그것의 계층 타입, 각 계층에 포함된 구현 컴포넌트들을 명시한다. 그림 5는 컴포넌트 구조 명세서를 위한 DTD와 연결리스트를 위한 구조 명세서를 나타낸다.

```
<!ELEMENT Layers (layer,Config_Repository)>

<!ELEMENT layer (Component_category+)>
<!ELEMENT Component_category (ImplCom)+>
<!ELEMENT ImplCom EMPTY>

<!ELEMENT Config_Repository (Config_Component_category)*>
<!ELEMENT Config_Component_category (Config_ImplCom)+>
<!ELEMENT Config_ImplCom EMPTY>

<!ATTLIST Component_category name CDATA #REQUIRED>
<!ATTLIST Component_category type (mandatory|optional) "mandatory">
<!ATTLIST ImplCom name CDATA #REQUIRED>

<!ATTLIST Config_Component_category name CDATA #REQUIRED>
<!ATTLIST Config_Component_category type (mandatory|optional) "mandatory">
<!ATTLIST Config_ImplCom name CDATA #REQUIRED>
```

```
<!DOCTYPE Layers SYSTEM "file:LayerdArchitecture.dtd">
<Layers>
 <layer>
  <Component_category name="Tracing" type="optional">
    <ImplCom name="TracedList"/>
  </Component_category>
  <Component_category name="Counter" type="optional">
    <ImplCom name="LengthList"/>
  </Component_category>
  <Component_category name="BasicList" type="mandatory">
    <ImplCom name="PtrList"/>
  </Component_category>
</layer>
<Config_Repository>
  <Config_Component_category
              name="Destroyer" type="mandatory">
    <Config_ImplCom name="ElementDestroyer"/>
    <Config_ImplCom name="EmptyDestroyer"/>
  </Config_Component_category>
    .....
  <Config_Component_category
              name="Copier" type="mandatory"> .....
  </Config_Component_category>
 </Config_Repository>
</Layers>
```

(그림 5) 컴포넌트 구조 명세서 DTD와 연결 리스트 구조 명세서

연결 리스트를 위한 계층 구조는 하위에서 상위로, 즉, BasicList, Counter, Tracing 순으로 일련의 순서를 가지고 정의된다. 가장 하위 계층에 위치한 구성 저장소(Configuration Repository)에는 상위의 모든 계층에서 사용되는 Destroyer, Copier 등의 비교적 크기가 작은 부품 컴포넌트나 타입 등이 위치한다. 구조 명세서에 의하면, 가장 상위의 Tracing 계층은 선택적 계층으로서 TracedList 구현 컴포넌트를 포함하고 있음을 알 수 있다.

2.3.2 컴포넌트 조립 규칙 명세서

컴포넌트 조립 규칙 명세서는, 계층 간의 조립 형태와 방법을 명시해 놓은 XML 형태의 규칙 파

```
<!ELEMENT LayerComposition (layer,Config)>

<!ELEMENT layer (Component_category)+>
<!ELEMENT Component_category (Alternative)+>
<!ELEMENT Alternative (ImplCom)>
<!ELEMENT ImplCom (parameter?)>
<!ELEMENT parameter (#PCDATA)>

<!ELEMENT Config (Config_Component_category)*>
<!ELEMENT Config_Component_category (OR)+>
<!ELEMENT OR (Config_ImplCom)>
<!ELEMENT Config_ImplCom EMPTY>

<!ATTLIST Component_category name CDATA #REQUIRED>
<!ATTLIST ImplCom name CDATA #REQUIRED>
<!ATTLIST parameter implement (inheritance|aggregation) "inheritance">

<!ATTLIST Config_Component_category name CDATA #REQUIRED>
<!ATTLIST Config_ImplCom name CDATA #REQUIRED>
```

```
<!DOCTYPE LayerComposition SYSTEM "file:CompositionRule.dtd">
<LayerComposition>
  <layer>
    <Component_category name="Tracing">
      <Alternative><ImplCom name="TracedList">
        <parameter implement="aggregation">Counter</parameter>
      </ImplCom></Alternative>
      <Alternative><ImplCom name="Counter"/></Alternative>
    </Component_category>
    .....
    <Component_category name="BasicList">
      <Alternative><ImplCom name="PtrList"/></Alternative>
    </Component_category>
  </layer>
  .....
</LayerComposition>
```

(그림 6) 컴포넌트 조립 규칙 명세서 DTD와 연결 리스트 조립 규칙 명세서

일이다. 조립 규칙은 생성된 계층 구조를 바탕으로 상위에서 하위 방향으로 작성된다. 선택된 특정 계층의 구현 컴포넌트는 바로 하위의 계층과 상속(inheritance) 또는 집합(aggregation)에 의해 조립될 수 있다. 그림 6은 컴포넌트 조립 규칙 명세서를 위한 DTD와 연결 리스트를 위한 조립 규칙 명세서의 일부를 나타낸다.

정의된 명세서에 의하면, Tracing 계층의 TracedList 구현 컴포넌트는 바로 하위의 Counter 계층과 집합 관계에 의해 조립될 수 있다. 만약 Tracing 계층이 포함되지 않는다면, 바로 하위의 Counter 계층이 올 수 있음을 명시하였다.

2.3.3 컴포넌트 구성 규칙 명세서

컴포넌트 구성 규칙 명세서는, 특성 모델로부터 재사용자가 선택한 특성에 따라 어떤 구현 컴포넌트들이 포함되어야 할 것인가에 대한 규칙을 정의해놓은 명세서이다. 즉, 여기에는 구체적인

```
<!ELEMENT Config (Component_category)+>
<!ELEMENT Component_category (ImplCom)+>
<!ELEMENT ImplCom (OR_cond)+>
<!ELEMENT OR_cond (AND_cond)+>
<!ELEMENT AND_cond (#PCDATA)>

<!ATTLIST Component_category name CDATA #REQUIRED>
<!ATTLIST ImplCom name CDATA #REQUIRED>
```

```
<!DOCTYPE Config SYSTEM "file:Configuration_Rule.dtd">
<Config>
.....
  <category name="Copier">
      <ImplCom name="EmptyCopier">
            <OR_cond>
              <AND_cond>External reference</AND_cond>
            </OR_cond>
            <OR_cond>
              <AND_cond>Owned reference</AND_cond>
            </OR_cond>
      </ImplCom>
.....
      <ImplCom name="MonomorphicCopier">
          <OR_cond><AND_cond>Copy</AND_cond>
              <AND_cond>Monomorphic</AND_cond>
          </OR_cond></ImplCom>
</Config>
```

(그림 7) 컴포넌트 구성 규칙 명세서 DTD와 연결 리스트 구성 규칙 명세서

컴포넌트 생성 시 컴포넌트의 구성을 결정하는 규칙을 포함한다. 특정 구현 컴포넌트는 하나의 특성, 또는 몇 가지 특성들의 조합에 의해 선택되어진다. 그림 7은 컴포넌트 구성 규칙 명세서를 위한 DTD와 연결 리스트를 위한 구성 규칙의 일부분을 보여준다.

각 구현 컴포넌트를 선택하기 위한 조건은 'and' 조건의 결합으로 이루어진 'or' 조건들로 정의하며, 재사용자에 의해 선택된 특성 값과 각 조건 값들을 비교하여 구현 컴포넌트의 포함 여부를 결정한다. 명세서에 의하면, 컴포넌트 재사용자가 External reference 특성 또는 Owned reference 특성 중 하나만 선택하면 EmptyCopier 구현 컴포넌트가 포함되고, Copy와 Monomorphic 특성 모두를 선택해야만 MonomorphicCopier 구현 컴포넌트가 포함된다는 규칙을 표현한다.

2.4 XSLT 컴포넌트 코드 템플릿

본 논문의 기법에서, 구현 컴포넌트와 자동 생성될 구체적인 컴포넌트에 대한 코드 부분은 XSLT

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:template match="Specification">
public interface Counter <xsl:apply-templates
      select="Layers/Component[@category='Counter']"/>
          final int length();
}
</xsl:template>

<xsl:template match="Layers/Component[@category='Counter']">
   <xsl:choose>
   <xsl:when test="count(self::node()[Composition='inheritance']) = 1">
            extends <xsl:value-of select="following-sibling::*/@category"/> {
   </xsl:when>
   <xsl:otherwise>{

   </xsl:otherwise>
   </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

(그림 8) Counter 인터페이스를 위한 XSLT 코드 템플릿

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:template match="Specification">
public class <xsl:value-of select="Layers/Component[@category='Tracing']/ImplCom"/>
   implement Tracing  <xsl:apply-templates select="Layers/Component[@category='Tracing']"/>
   public <xsl:value-of select="Layers/Component[@category='Tracing']/ImplCom"/>
            (<xsl:apply-templates select="Features"/> h){
     super(h);}
  .....
</xsl:template>  .....
<xsl:template match="Features">
   <xsl:value-of select="./Feature[name='ElementType']//value"/>
</xsl:template>
<xsl:template match="Layers/Component[@category='Tracing']">
 <xsl:choose>
   <xsl:when test="count(self::node()[Composition='inheritance']) = 1">
          extends <xsl:value-of select="following-sibling::*/ImplCom"/> {
   </xsl:when>
   <xsl:when test="count(self::node()[Composition='aggregation']) = 1">{
   <xsl:value-of select="following-sibling::*/ImplCom"/>
            low_layer = new <xsl:value-of select="following-sibling::*/ImplCom"/>();
   </xsl:when>
   <xsl:otherwise>{
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

(그림 9) TracedList를 위한 XSLT 코드 템플릿

스크립트 파일이 제공한다. XSLT 스크립트 파일은 컴포넌트 코드 생성 시 필요한 각 컴포넌트의 코드 골격이 되는 코드 템플릿 기능을 제공하며, 또한 재사용자가 제공한 커스터마이징 데이터를 바탕으로 각 컴포넌트 카테고리에 속하는 구현 컴포넌트들 중 특정 구현 컴포넌트와 관련된 코드 부분을 선택하는 코드 선택 기능을 제공한다.

본 논문의 컴포넌트 생성 기법에서 필요한 XSLT 코드 템플릿의 종류는 다음과 같다.

1) 컴포넌트 카테고리별 구현 컴포넌트를 위한 코드 템플릿
2) 목표 컴포넌트를 위한 코드 템플릿
3) 카테고리 또는 계층을 인터페이스 코드 템플릿

그림 8은 Counter 카테고리의 인터페이스를 위한 XSLT 코드 템플릿을 나타낸다. Counter 인터페이스는 Counter 카테고리의 구현 컴포넌트들이 공통으로 가지는 length() 메소드를 선언하는 코드를 포함하고 있으며, <xsl:choose> 태그를 이용하여 하위 계층에 따라 다르게 나타나는 인터페이스 상속 관계를 적절히 표현한다.

그림 9는 TracedList를 위한 XSLT 코드 템플릿의 일부분을 보여준다. 여기에는 Tracing 컴포넌트 카테고리에 속하는 구현 컴포넌트들이 공통으로 가지는 코드 부분이 포함되어 있다. <xsl:value-of select=""> 태그는 컴포넌트의 요구 사항을 담고 있는 XML 명세서에서 특정 특성의 이름 또는 값 등의 각 컴포넌트의 가변성 값을 알아내는 역할을 한다. 또한, <xsl:choose> 태그는 하위 계층과의 조립 방법에 따라 달라지는 구현 코드를 적절히 선택하는 일을 한다.

3. CompGenX 컴포넌트 생성 시스템

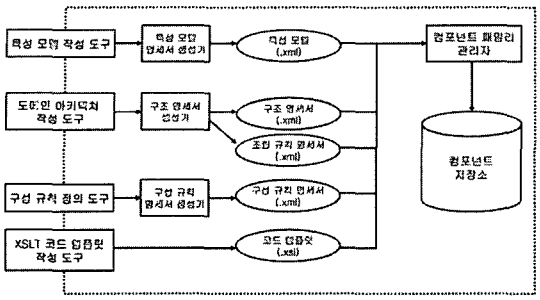
CompGenX(Component Generator using XML)는 본 논문에서 제안하는 컴포넌트의 코드 자동 생성 기법을 지원하기 위한 컴포넌트 생성 시스템으로서, 컴포넌트 패밀리 구축 과정 지원 도구와 컴포넌트 재사용 과정 지원 도구 및 컴포넌트 코드 생성기로 구성된다. 본 장에서는 CompGenX 시스템의 구조와 각 구성 요소가 제공하는 기능을 설명한다.

3.1 컴포넌트 패밀리 구축 지원 도구

그림 10은 컴포넌트 개발자의 컴포넌트 패밀리 구축 과정을 지원하기 위한 CompGenX 도구의 전체 구조를 보여준다.

각각의 도구에 대한 설명은 다음과 같다.

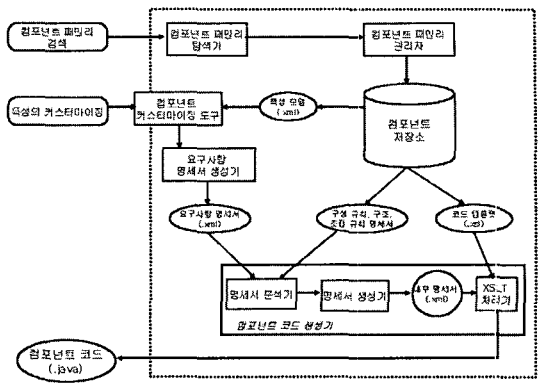
1) 특성 모델 작성 도구: 특정 컴포넌트 패밀리를



(그림 10) 컴포넌트 패밀리 구축을 위한 CompGenX의 전체 구조

위한 특성 모델을 정의하기 위한 개발자 인터페이스를 제공한다.

2) 도메인 아키텍처 정의 도구: 특정 컴포넌트 패밀리의 구조를 정의하기 위한 개발자 인터페이스를 제공한다. 개발자는 이를 통하여 컴포넌트 패밀리를 위한 계층 구조를 정의하고 각 계층 간의 조립 방법과 형태를 결정한다.
3) 구성 규칙 작성 도구: 구체적인 컴포넌트의 구성 규칙을 작성하기 위한 개발자 인터페이스이다. 개발자는 이를 통하여 하나의 특성 또는 특성의 집합과 구현 컴포넌트 간의 대응 관계를 정의한다.
4) 명세서 생성기 : (1),(2),(3) 도구의 작업 결과를 가지고 각각 XML 형태의 명세서 파일을 생성한다. CompGenX에 포함된 명세서 생성기는 다음과 같다.
   * 특성 모델 명세서 생성기 : 정의된 특성 모델을 표현한 명세서를 자동 생성한다.
   * 구조 명세서 생성기 : 정의된 계층 구조와 조립 방법을 바탕으로 구조 명세서와 조립 규칙 명세서를 자동 생성한다.
   * 구성 규칙 명세서 생성기 : 구성 규칙 명세서를 자동 생성한다.
5) XSLT 코드 템플릿 작성 도구 : 구체적인 컴포넌트 생성과 관련된 XSLT 코드 템플릿을 작성하기 위한 개발자 인터페이스이다. 작성된 코드 템플릿은 .xsl 파일 형태로 저장된다.
6) 컴포넌트 패밀리 관리자 : 컴포넌트 패밀리별



(그림 11) 컴포넌트 재사용을 위한 CompGenX의 전체 구조

특성 모델, 구조 명세서, 조립 규칙 명세서, 구성 지식 명세서, 코드 템플릿 등을 컴포넌트 저장소에 저장하고 이를 관리한다.

7) 컴포넌트 저장소 : 여러 가지 컴포넌트 패밀리에 대한 특성 모델과 컴포넌트 구조, 조립 규칙, 구성 규칙에 대한 명세서, 그리고 XSLT 코드 템플릿을 저장한다.

3.2 컴포넌트 재사용 지원 도구

그림 11은 컴포넌트 재사용 과정을 지원하기 위한 CompGenX의 재사용 지원 도구의 전체 구조를 나타낸다.

재사용 도구의 각 모듈은 다음과 같은 기능을 지원한다.

1) 컴포넌트 패밀리 탐색기 : 재사용자가 자신의 목적에 맞는 컴포넌트 패밀리를 검색하기 위한 사용자 인터페이스를 제공한다. 컴포넌트 패밀리 관리자는 컴포넌트 재사용 시에 컴포넌트 패밀리 탐색기로부터 질의문을 받아서, 컴포넌트 저장소로부터 해당 컴포넌트 패밀리의 특성 모델과 관련 명세서들을 추출한다.
2) 컴포넌트 커스터마이징 도구 : 해당 컴포넌트 패밀리를 위한 특성 모델을 시각적으로 보여준다. 재사용자는 이를 통하여 자신의 목적에

맞게 특성을 선택하거나, 특성 값을 입력한다.

3) 요구사항 명세서 생성기: 재사용자가 제공한 특성 값을 바탕으로 구성 명세서 생성기는 재사용자가 원하는 컴포넌트에 대한 요구사항 명세서를 생성한다.

3.3 컴포넌트 코드 생성기

그림 11의 컴포넌트 코드 생성기는 실제 컴포넌트 소스 코드를 생성하는 핵심 도구로서, XML 형태의 컴포넌트 구성 명세서와 내부 구성 지식 명세서들을 바탕으로 내부 명세 파일을 생성하고, 각 컴포넌트를 위한 XSLT 코드 템플릿을 이용하여 구체적인 컴포넌트의 소스 코드를 생성한다.

컴포넌트 코드 생성기는 각 모듈별로 다음과 같은 기능을 수행한다.

1) 명세서 분석기
명세서 분석기는, XML 형태의 각 컴포넌트 명세서를 분석한다. 명세서 분석기는 다음과 같은 4가지 분석기로 구성된다.
* 요구사항 분석기 : 재사용자의 커스터마이제이션 결과에 특성 모델컴포넌트 구성 명세서를 분석한다.
* 구조 분석기 : 컴포넌트 구조 명세서를 분석한다.
* 구성 규칙 분석기 : 컴포넌트 구성 규칙 명세서를 분석한다.
* 조립 규칙 분석기 : 컴포넌트 조립 규칙 명세서를 분석한다.

2) 명세서 생성기
명세서 생성기는, 명세서 분석기의 분석 결과를 바탕으로 XML 형태의 내부 명세 파일을 생성한다. 이는 컴포넌트 코드 생성기 내부에서만 사용되는 구체적인 컴포넌트를 위한 명세 파일로서, 구현 컴포넌트로 구성된 계층 구조, 각 계층 간의 조립 방법, 특성 선택 결과, 구현 컴포넌트 종류 등에 대한 정보를 포함한다.

3) XSLT 처리기
XSLT 처리기는 실제 구체적인 컴포넌트의 구현 코드를 생성하는 모듈이다. 이는 내부 명세서와 여러 가지 XSLT 코드 템플릿들을 이용하여 구체적인 컴포넌트 소스 코드를 생성한다.

4. CompGenX를 이용한 컴포넌트 재사용

CompGenX 시스템을 이용한 컴포넌트 재사용 과정은 다음과 같다.

1) 컴포넌트 패밀리 결정
재사용자는 컴포넌트 패밀리 탐색기를 이용하여, 자동 생성하기를 원하는 컴포넌트가 속한 컴포넌트 패밀리를 선택한다(예: 연결 리스트)

2) 특성 모델을 이용한 커스터마이징
재사용자는 해당 컴포넌트 패밀리의 특성 모델로부터 필요한 필수적, 선택적, 택일적 특성을 선택하고 커스터마이징 값을 입력한다. 이를 바탕으로 CompGenX는 컴포넌트 구성 명세서를 생성한다.

그림 12는 컴포넌트 구성 명세서를 위한 DTD와 연결 리스트 구성 명세서의 한 예이다. 구성 명세서에 의하면, 재사용자는 ElementType의 값을 Base로 설정하였으며, 선택적 특성인 LengthCounter 특성을 선택하였다. 또한 필수적 특성인 Morphology의 값으로 하위 택일적 특성들 중 Monomorphic을 선택했음을 알 수 있다.

```
<!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 12) 컴포넌트 구성 명세서를 위한 DTD와 연결 리스트의 구성 명세서

```
public interface Counter extends BasicList {
    final int length();
}
```

(그림 13) 자동 생성된 Counter.java

```
public class TracedList implement Tracing extends LengthList {

        public TracedList(Base h){
                super(h);
        }
        .....

        public void setHead( Object h){
                System.out.println("setHead(" + h + ")");
                super.setHead((Base)h);
        }
        .....

        public void setTail(Object t) {
                System.out.println("setTail(" + t + ")");
                super.setTail((TracedList)t);
        }
}
```

(그림 14) 자동 생성된 TracedList.java

3) 구체적인 컴포넌트 코드 자동 생성
특성 모델에서 재사용자가 입력한 가변성의 값들을 이용하여, CompGenX는 재사용자가 원하는 최종 컴포넌트의 코드를 자동 생성한다. 그림 13와 그림 14은 각각 자동 생성된 Counter 인터페이스와 TracedList의 소스 코드를 보여준다.

그림 14의 TracedList 클래스는 Tracing 인터페이스를 구현(implements)하며, LengthList를 상속(extends)받는다. 또한, 생성자 TracedList()의 인자로 넘어올 수 있는 ElementType은 Base형을 갖는다. TracedList는 리스트의 각 연산 호출에 대한 로깅 정보를 화면에 출력해주는 기능을 제공하기 위하여, 상위 클래스로부터 상속받은 setHead(). setTail() 등의 메소드에 로깅 정보 출력문이 추가되었다. 즉, TracedList는 하위의 LengthList를 조립하고 이를 세분화시켰음을 알 수 있다.

CompGenX를 이용한 컴포넌트 코드 생성 기법의 장점은 다음과 같다.

1) 보다 추상적인 모델인 특성 모델을 통해서 컴포넌트를 바라볼 수 있으므로, 재사용자는 컴포넌트 구현에 대한 지식 없이 컴포넌트의 전체적인 의미를 쉽게 파악할 수 있다.
2) 특성 모델과 구성 명세서를 XML로 표현함으로써 컴포넌트 정보에 대한 표준화를 제공하여 효율적인 컴포넌트 정보 저장과 검색을 가능하게 한다.
3) 목표 컴포넌트를 생성함에 있어 구현 컴포넌트의 종류가 바뀌는 경우 구성 규칙 명세서를 수정함으로써 쉽게 대처할 수 있다.
4) 컴포넌트를 구성하는 각 계층은 표준화된 인터페이스를 제공하기 때문에, 새로운 기능을 추가하고자 할 때 그 계층의 인터페이스를 구현한 새로운 컴포넌트를 그 계층에 포함시키기만 하면 된다.
5) XSLT 코드 템플릿 파일에 직접 코드를 추가함으로써, 컴포넌트의 기능을 쉽게 확장할 수 있다.
6) 자동 생성되는 결과물이 소스 코드 형태이므로, 상속이나 그 밖의 방식을 이용하여 재사용자는 생성된 코드를 쉽게 확장할 수 있다.

5. 결론 및 향후 과제

본 논문에서는, GenVoca 계층 구조와 XML/XSLT 기술을 이용한 컴포넌트 코드 생성 기법과 CompGenX 컴포넌트 생성 시스템을 제안하였다. CompGenX은 컴포넌트 코드 생성 시에 컴포넌트의 재구성성을 지원함으로써 재사용자가 컴포넌트 재사용 시 자신의 목적에 맞는 컴포넌트의 소스 코드를 자동으로 생성할 수 있게 해 준다. CompGenX를 이용한 컴포넌트 개발은 크게 두 가지로 나누어진다. 컴포넌트 패밀리 구축 과정에서는, 도메인 분석 및 설계, 컴포넌트 구성 규칙 명세서 작성, 구현 컴포넌트 코드 템플릿 작성 등의 과정을 거쳐 재사용 가능한 소프트웨어 자산을 구축한다. 컴포넌트 재사용 과정에서는, 특성 모델을 통하여 재사용자가 원하는 가변성 값들을 입력함으로써

자신의 목적과 환경에 맞는 컴포넌트 코드를 생성한다.

CompGenX는 컴포넌트 재사용을 위한 체계적이고 효율적인 여러 가지 도구들을 제공함으로써, 컴포넌트 개발의 생산성을 향상시켜준다. 또한 컴포넌트 명세서가 인터넷 문서의 표준이며 자기 서술적인(self-descriptive)인 XML로 표현되기 때문에 컴포넌트의 검색과 개발의 효율성을 높여준다.

향후 과제로는, 컴포넌트 테스팅을 위한 코드 자동 생성, 보다 큰 단위의 컴포넌트 자동 생성 및 EJB(Enterprise Java Beans) 컴포넌트 자동 생성에 대한 연구가 필요하다. 또한, 컴포넌트 저장소 구축과 컴포넌트 검색을 효율적으로 지원할 수 있는 방법 등에 대한 연구가 진행되어야 할 것이다.

Acknowledgement

본 연구는 2002학년도 덕성여자대학교 연구비 지원으로 이루어졌음.

참 고 문 헌

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

[2] 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.

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

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

[5] K. Czarnecki and U. W. Eisenesker, Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000.

[6] D. Batory et al., "The GenVoca Model of Software-System Generators", IEEE Software, pp. 89~94, 9, 1994

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

[8] H. Zhang, S. Jarzabax 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)
2000년~현재 : 덕성여자대학교 컴퓨터학부 교수
관심분야 : 컴포넌트 기반 소프트웨어 공학, 소프트웨어 생산 라인, 자동 생성 프로그래밍, etc.
E-mail : csh@duksung.ac.kr