특성 구성과 GenVoca 아키텍처에 기반한 컴포넌트 재구성 자동화 도구☆

Automatic Component Reconfiguration Tool Based on the Feature Configuration and GenVoca Architecture

최 승 훈*
Seung Hoon Choi

요 약

최근 컴포넌트 기반의 소프트웨어 프러덕트 라인에 대한 연구와 소프트웨어 프러덕트 라인에 자동 생성 프로그래밍 기법을 적용하기 위한 연구가 활발히 진행 중이다. 본 논문은, 컴포넌트 기반의 소프트웨어 프러덕트 라인 개발에 활용될 수 있는 컴포넌트 재구성 자동화 도구를 제안한다. 본 도구는 도메인 공학의 주요 산물인 특성 모델로부터 재사용자의 요구 사항을 받아들여 특성 구성(feature configuration)을 만들고 이를 바탕으로 재구성된 컴포넌트 코드를 자동으로 생성한다. 이를 위하여 컴포넌트 패밀리는 자동 생성 프로그래밍의 한 기법인 GenVoca의 아키텍처를 가지며 XSLT 스크립트가 컴포넌트를 구성하는 구현 부품의 코드 템플릿을 제공한다. '은행 계좌' 컴포넌트 패밀리를 사례 연구로 하여 본 논문의 컴포넌트 재구성 자동화 도구가 재사용자의 목적에 맞는 컴포넌트를 자동으로 생성함을 보였다. 본 논문의 연구 결과는 소프트웨어 프러덕트 라인 개발의 생산성을 향상시키는데 응용될 수 있다.

Abstract

Recently a lot of researches on the component-based software product lines and on applying generative programming into software product lines are being performed actively. This paper proposes an automatic component reconfiguration tool that could be applied in constructing the component-based software product lines. Our tool accepts the reuser's requirement via a feature model which is the main result of the domain engineering, and makes the feature configuration from this requirement. Then it generates the source code of the reconfigured component according to this feature configuration. To accomplish this process, the component family in our tool should have the architecture of GenVoca that is one of the most influential generative programming approaches. In addition, XSLT scripts provide the code templates for implementation elements which are the ingredients of the target component. Taking the 'Bank Account' component family as our example, we showed that our component reconfiguration tool produced automatically the component source code that the reuser wants to create. The result of this paper would be applied extensively for creasing the productivity of building the software product lines.

Keyword : Software Product Lines, Generative Programming, Reconfigurability, Feature Configuration, GenVoca, XSLT

1. 서론 및 연구 배경

소프트웨어 프러덕트 라인이란, 공통된 핵심 소프트웨어 자산(core software asset)들로부터 특정 목표나 시장을 위해서 개발된 소프트웨어 집약 시스템들의 집합을 의미한다[1]. 소프트웨어 프러덕트 라인의 목적은, 소프트웨어 개발 단계 초기에 소프트웨어 패밀리에 속하는 멤버들 사이의 차이점과 공통점을 미리 예측하고 분석함으로써 보다 전략적인 재사용이 가능하도록 하여 소프트웨어 개발 생산성을 향상시키고자 하는 것이다.

최근 컴포넌트 기반의 프러덕트 라인 구축 기법에 대한 연구가 활발히 진행되고 있다[2]. 소프트웨어 프러덕트 라인에 존재하는 소프트웨어 부품은 '차이

* 정 회 원 : 덕성여자대학교 컴퓨터학부 교수
csh@duksung.ac.kr(제 1저자)
☆ 본 연구는 2003학년도 덕성여자대학교 자연과학연구소 연구비 지원으로 이루어졌음.

점 또는 가변성 (variability)'을 지원해야 한다. '가변성'이란, 공통점을 공유하는 일련의 소프트웨어 부품들이 각 환경이나 목적에 맞게 변형될 필요가 있는 특성을 의미한다.

소프트웨어 부품들이 가변성을 지원하기 위해서는, 특정 소프트웨어 개발 시 그 소프트웨어가 실행되는 환경과 목적에 맞도록 그 기능과 구조를 쉽게 재구성할 수 있어야 한다. 본 논문에서는 최근 효율적인 프로그래밍 기법으로서 각광받고 있는 자동 생성 프로그래밍 기법을 이용하여 재사용자가 원하는 특성에 따라 컴포넌트의 소스 코드를 자동 생성하는 컴포넌트 재구성 자동화 도구를 제안한다.

본 논문의 도구는 도메인 분석의 주요 산물인 특성 모델로부터 재사용자의 요구 사항을 받아들여 특성 구성(feature configuration)을 만들고 이를 바탕으로 재구성된 컴포넌트 코드를 자동으로 생성한다. 이를 위하여 컴포넌트 패밀리는 자동 생성 프로그래밍의 한 기법인 GenVoca의 아키텍처를 가지며 XSLT 스크립트가 컴포넌트를 구성하는 구성 요소의 코드 템플릿을 제공한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 관련 연구들을 살펴보고, 제 3 장에서 특성 모델과 특성 구성의 정의 및 컴포넌트 재구성 자동화 도구의 전체적인 구조를 기술한다. 제 4 장에서는 은행 계좌 컴포넌트 패밀리를 사례연구로 하여 컴포넌트 패밀리 구축 프로세스를 단계별로 설명한다. 제 5 장에서 컴포넌트 소스 코드가 자동으로 생성되는 과정을 살펴보고, 제 6 장에서 본 도구에 대한 장단점을 평가하며, 마지막으로 제 7 장에서 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

2.1 도메인 공학과 특성 모델

소프트웨어 프러덕트 라인 구축을 위하여 특정 도메인을 위한 소프트웨어 아키텍처와 부품들을 개발하고 그 영역 내에서의 재사용을 시도하기 위한 도메인 공학의 중요성이 커지게 되었으며, 지금까지 여러 가지 도메인 공학 기법들이 제안되었다([3,4,5,6,7].

본 논문의 재구성 자동화 도구는 컴포넌트 패밀리 내의 멤버들 사이의 공통점과 차이점을 분석하기 위한 모델로서 FODA[8]에서 제안한 특성 모델(feature model)을 사용한다. FODA (Feature-Oriented Domain Analysis)는 도메인에 존재하는 개체들과 이들 사이의 관계를 식별하고, 고객들이 요구하는 시스템의 기능들을 파악한다. 그리고, 관련된 시스템들 사이의 공통점과 차이점들을 식별하고 이를 특성 모델(Feature Model)로 표현한다. 영역 분석의 주요 산물인 특성 모델은 제품군에 속하는 멤버들 사이의 공통점과 차이점에 대한 고수준의 추상화를 제공하는 요구 사항 모델이다.

2.2 자동 생성 프로그래밍

소프트웨어 프러덕트 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 방법이 제안되었다[9].

최근에는 문서 표현의 표준 언어인 XML(eXtensible Markup Language)을 소프트웨어 자동 생성에 이용하기 위한 연구가 많이 진행되고 있다. [10]에서는 문서 표현의 표준 기술인 XML(eXtensible Markup Language)을 이용하여 프로그램의 추상적 명세서를 작성하고, 이를 받아들여 프로그램 코드를 자동 생성하는 프로그램 생성기를 제안하였다.

본 논문의 내용과 관련하여 계층 구조 기반의 자동 생성 기법인 GenVoca 와 XML 및 프레임 기술을 이용한 프로그램 생성 도구인 XVCL이 제안되었다. GenVoca[11,12]는 객체 지향 패러다임의 계층(layer) 조립을 바탕으로 컴포넌트 자동 생성기를 개발하기

위한 접근 방법이다. GenVoca 생성기는 컴포넌트 명세로부터 재사용 가능한 구현 부품을 선택하고, 이를 미리 정의된 계층 구조에 따라 조립함으로써 구체적인 컴포넌트를 생성한다. 본 논문에서의 컴포넌트 패밀리 구조는 GenVoca에서 제안된 계층 구조를 따른다. 그러나, 특성 모델을 단순한 분석 모델링 도구로 사용한 GenVoca와는 달리 본 논문의 도구는 특성 모델을 재구성을 위한 차이점 입력 도구로 사용한다.

XVCL (XML-based Variant Configuration Language)[13,14]에서는, 프러덕트 라인을 구성하는 자산들이 가변성을 수용할 수 있도록 프레임 기술[15]을 사용하여 구조화하고, XML을 기반으로 한 명세서를 해석하여 자산의 커스터마이징을 자동화하는 도구를 제안하였다. 각각의 소프트웨어 자산은 XVCL에서 제공하는 명령어들을 포함하는 x-frame으로 표현되며, 특정 시스템은 x-frame을 커스터마이즈하고 합성하는 것에 의해 생성된다. 이 기법은 컴포넌트 생성시 유연한 방법을 제공해 준다. 그러나, 반드시 XVCL 프로세서라는 전용 프로세서가 필요하고, 컴포넌트 개발자나 재사용자가 이를 적용하기 위해서는 XVCL 언어의 문법을 추가로 배워야 한다. 또한, 원하는 코드가 가지는 차이점을 명시하기 위한 명세서를 재사용자가 직접 작성하여야 하며, 재사용자는 컴포넌트를 구성하는 코드 템플릿(x-frame)들의 내부 구조와 내용을 이해하여야 한다.

표 1은 본 논문에서 제안한 도구와 GenVoca 및 XVCL의 차이점을 보여준다.

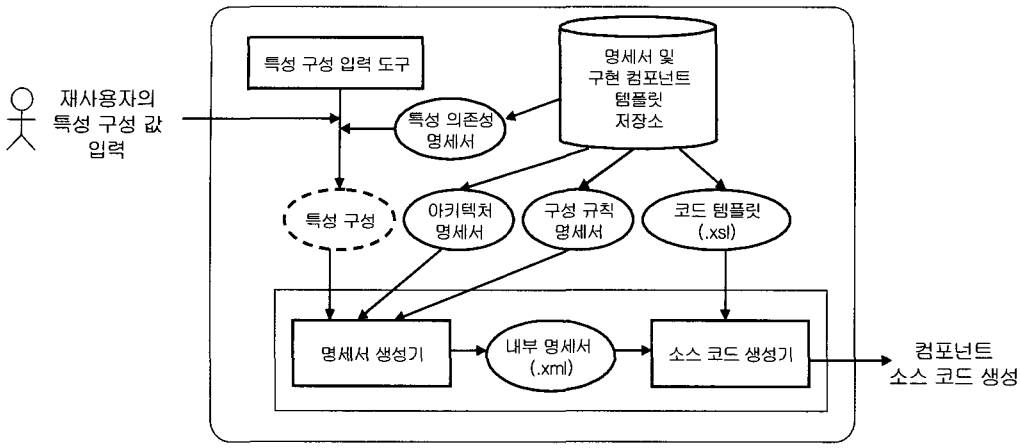
3. 컴포넌트 재구성 자동화 도구

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 의미한다. 특성 모델에서 제품군에 속하는 멤버들이 공유하는 공통점은 필수적 특성(mandatory feature)으로 표현되며, 멤버들 간의 차이

〈표 1〉 GenVoca, XVCL 및 본 논문에서 제안한 도구의 비교

구분	GenVoca	XVCL	제안된 자동화 도구
구현 기술	C++	XML / Java / JAXP	XML / Java / JAXP
아키텍처	Layer 기반의 계층 구조	일반적인 tree 형태 (XML 도큐먼트의 계층 구조)	Layer 기반의 계층 구조
조립 방식	상속 또는 집합	Frame 기술	상속 또는 집합
코드 템플릿 형태	C++ 템플릿 클래스	XML 기반의 X frames (텍스트 + XVCL 명령어들)	XSLT 파일 (자바 코드 + XSLT 원소들)
Variation point 표현 방법	템플릿 클래스의 parameter	XVCL 명령어들	XSLT 원소들
요구 사항 입력 방식	재사용자가 직접 코드 작성	XML 형태의 명세서 작성(SPC)	특성 모델을 통한 특성 구성 입력
코드 생성 소프트웨어 (Resolver of Variation point)	C++ 컴파일러	전용 XVCL 프로세서	XSLT 프로세서
생성된 컴포넌트 형태	C++ 바이너리 코드	자바 소스 코드 (텍스트 형태의 모든 소프트웨어 관련 문서 생성에 적용 가능함)	자바 소스 코드



〈그림 1〉 재구성 자동화 도구의 전체적인 구조

점은 선택적(optional), 택일적(alternative), 비결정 인자(free parameter) 특성 등으로 표현된다.

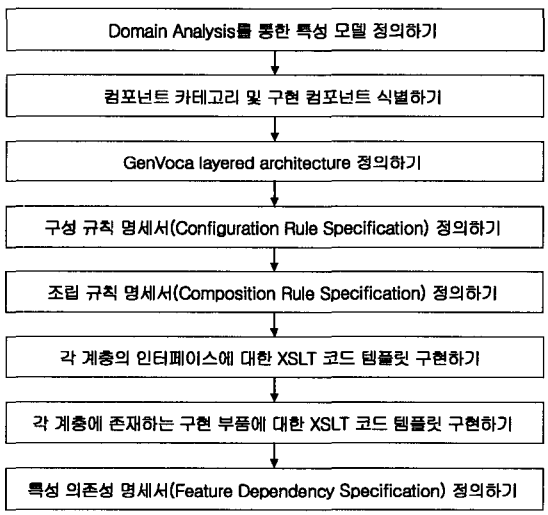
특성 구성이란, 특성 모델로부터 재사용자가 결정한 특성 선택 결과를 의미한다[16]. 즉, 특성 모델에 표현되어 있는 프러덕트 라인 멤버들 사이의 차이점들에 대하여 재사용자가 원하는 값을 선택하거나 입력한 결과로서, 재사용시 생성하고자 하는 컴포넌트에 대한 요구 사항을 의미한다.

3.2 재구성 자동화 도구의 구조

본 논문에서 제안하는 재구성 자동화 도구의 전체적인 구조는 그림 1과 같다. 재사용자는 '특성 구성 입력 도구'를 이용하여 자신이 원하는 컴포넌트의 특성 구성 값을 입력한다. 이 때, '특성 구성 입력 도구'는 '특성 의존성 명세서'를 이용하여 재사용자가 입력한 특성 구성 값의 유효성을 검증한다. 생성된 '특성 구성'은 '명세서 생성기'로 입력되며, '아키텍처 명세서', '구성 규칙 명세서', '코드 템플릿' 등을 이용하여 특정 컴포넌트의 소스 코드가 생성된다.

4. 컴포넌트 패밀리 구축 프로세스

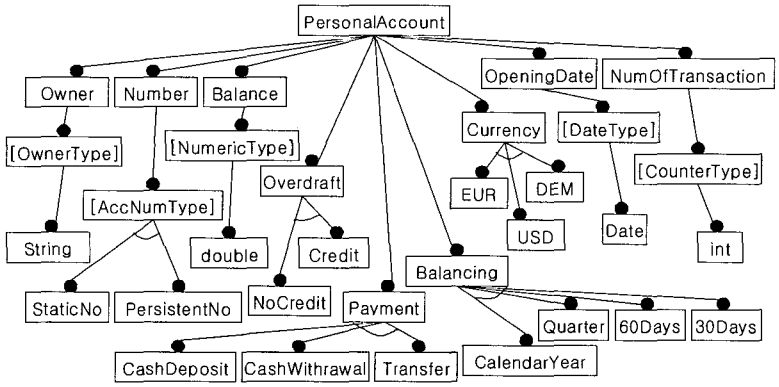
이 장에서는, "은행 계좌" 컴포넌트 패밀리를 사례 연구로 하여 본 논문의 컴포넌트 패밀리 구축 과정을 기술한다. 전체적인 컴포넌트 패밀리 구축 과정은 그림 2과 같다.



〈그림 2〉 컴포넌트 패밀리 구축 프로세스

4.1 도메인 분석을 통한 특성 모델 정의

먼저, 개발하고자 하는 컴포넌트 패밀리에 속하는 멤버들 사이의 공통점과 차이점을 분석하여 특성 모델을 정의한다. 그림 3은 은행 계좌 컴포넌트 패밀리의 특성 모델을 보여준다. 특성 모델에 표현되어 있는 주요 특성의 의미는 다음과 같다.



〈그림 3〉 '은행 계좌' 특성 모델

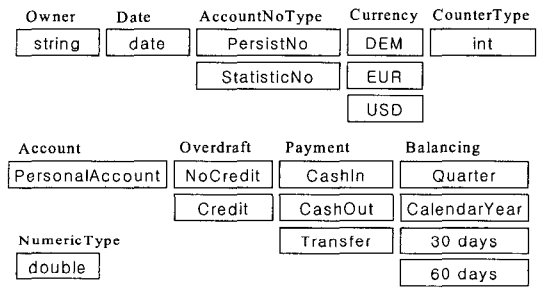
- 'Personal Account': 최종 목표 컴포넌트를 대표하는 특성으로서 concept라고도 한다.
- 'number': 계좌 번호의 형식을 의미한다.
- 'overdraft': 고객이 예금 잔액을 초과하여 현금 인출이 가능한지 그 여부를 나타내는 특성이다.
- 'payment': 은행의 주요 업무인 예금('cash deposit' 특성), 인출('cash withdrawal' 특성), 이체('transfer' 특성) 중 어느 기능을 제공하는지를 결정하는 특성이다.
- 'balancing': 계좌 개설일로부터 얼마 후에 이자를 계산하여 지불할 것인지를 결정하는 특성이다.

4.2 컴포넌트 카테고리 및 구현 부품 식별

도메인 분석 결과를 바탕으로 컴포넌트 카테고리를 식별한다. 컴포넌트 카테고리는 각 특성이 나타내는 주된 책임을 담당하는 구성 요소로서, 각 카테고리 별로 실제 구현 부품들을 포함한다. 컴포넌트 카테고리는 표준화된 인터페이스를 제공하는 하나의 계층이며, 같은 컴포넌트 카테고리에 포함된 구현 컴포넌트들은 모두 같은 인터페이스를 따른다[9]. 그림 4는 은행 계좌를 위한 컴포넌트 카테고리와 카테고리 별 구현 부품을 보여준다.

4.3 GenVoca 계층 구조 정의

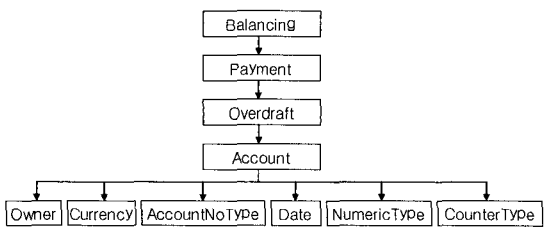
이 단계에서는 전 단계에서 식별된 컴포넌트 카테



〈그림 4〉 '은행 계좌' 컴포넌트의 컴포넌트 카테고리 및 구현 부품

고리와 구현 부품을 바탕으로 GenVoca 계층 구조를 정의한다. GenVoca 아키텍처에서의 계층은 컴포넌트 카테고리를 의미하며, 보다 세분화된 계층은 상위에 위치하도록 계층 구조를 정의한다. GenVoca 아키텍처에 대한 자세한 사항은 [9]를 참조한다.

그림 5는 '은행 계좌' 컴포넌트를 위한 GenVoca 계층 구조를 보여준다. 이 그림에서 화살표는 'depend-on' 관계를 나타낸다. 예를 들어, 'Payment' 계층에 존재하는 구현 부품은 'Overdraft' 계층이 제공하는 구현 부품을 정제하고 세분화하여 만들어진다.



〈그림 5〉 '은행 계좌' 컴포넌트의 계층 구조

```
<ComponentArchitecture>
 <layers>
  ... ...
  <Component_category name="Payment"
   type="mandatory" depend_on="Overdraft, Payment">
     <ImplCom name="CashIn"/>
     <ImplCom name="CashOut"/>
     <ImplCom name="Transfer"/>
  </Component_category>
       ... ...
    </layers>
</ComponentArchitecture>
```

〈그림 6〉 '은행 계좌' 컴포넌트의 아키텍처 명세서 일부

정의된 계층 구조는 그림 6 과 같이 '아키텍처 명세서'로 저장되어 컴포넌트 생성 시 사용된다.

4.4 구성 규칙(Configuration Rule) 명세서 정의

구성 규칙이란, 구체적인 컴포넌트 생성 시 재사용자가 입력한 특성 구성에 따라 각 컴포넌트 카테고리 내의 어떤 구현 부품을 선택할 지를 정의한 규칙이다. 구성 규칙 명세서를 작성하기 위해서, 먼저 각 특성의 값에 따라 어떤 구현 부품이 선택 될 것인지를 나타내는 결정 표(decision table)를 작성한다.

표 2는 '은행 계좌' 컴포넌트에서 특성 구성 값에 따라 컴포넌트 카테고리에 포함된 구현 부품 중 무엇이 선택될 것인가를 나타내는 표의 일부이다. 표에 의하면, 예를 들어 특성 모델 중 'Payment' 특성의 하위 택일적 특성 중 'CashDeposit' 특성이 선택되었다면 'Payment' 카테고리의 구현 부품 중에서 'CashIn' 구현 부품이 선택되어 짐을 알 수 있다. 이러한 결정표에 따라 그림7과 같은 구성규칙 명세서가 만들어진다.

```
<Component_category name="Payment">
 <ImplCom name="CashIn">
 <OR_cond><AND_cond>CashDeposit</AND_cond></OR_cond>
 </ImplCom>
 <ImplCom name="CashOut">
 <OR_cond><AND_cond>CashWithdrawal
  </AND_cond></OR_cond>
 </ImplCom>
 <ImplCom name="Transfer">
  <OR_cond><AND_cond>Transfer</AND_cond></OR_cond>
 </ImplCom>
</Component_category>
```

〈그림 7〉 'Payment' 카테고리를 위한 구성 규칙

4.5 컴포넌트 카테고리를 위한 코드 템플릿 작성

이 단계에서는 각 컴포넌트 카테고리가 정의하는 인터페이스를 생성하기 위한 코드 템플릿을 구현한

〈표 2〉 '은행 계좌' 컴포넌트에서 구현 부품 선택을 위한 결정표

컴포넌트 카테고리	구현 부품	구현 컴포넌트가 선택되기 위한 특성 구성 값
Owner	비결정인자	PersonalAccount.Owner.[OwnerType] = v1
	String	PersonalAccount.Owner.[OwnerType] = "String"
AccountNoType	PersistNo	PersonalAccount.accountNumberType = "PersistNo"
	StaticNo	PersonalAccount.accountNumberType = "StaticNo"
Currency	DEM	PersonalAccount.currency = "DEM"
	EUR	PersonalAccount.currency = "EUR"
	USD	PersonalAccount.currency = "USD"
Account	PersonalAccount	PersonalAccount = 선택
Overdraft	NoCredit	PersonalAccount.Overdraft = "NoCredit"
	Credit	PersonalAccount.Overdraft = "Credit"
Payment	CashIn	PersonalAccount.Payment = "CashDeposit"
	CashOut	PersonalAccount.Payment = "CashWithdrawal"
	Transfer	PersonalAccount.Payment = "Transfer"

다. 그림 8은 'Payment' 카테고리의 인터페이스를 위한 XSLT 코드 템플릿을 보여준다. 코드 생성 시에는, 앞 단계에서 정의된 아키텍처 명세서 및 구성 규칙 명세서가 재사용자가 입력한 특성 구성과 결합되어 내부 명세서가 만들어지고 이 내부 명세서가 XSLT 코드 템플릿에 적용되어 구체적인 인터페이스 코드가 생성된다. 예를 들어 그림 8 두 번째 줄의 select = "Layers/Component [@category='Payment']" 부분은 XML 형태의 내부 명세서로부터 값을 얻어온다.

```
<xsl:template match="Specification">
public interface <xsl:apply-templates select=
  "Layers/Component[@category='Payment']"/>
}
</xsl:template>
<xsl:template match="Layers/Component[@category='Payment']">
        <xsl:apply-templates select="@*"/>
</xsl:template>
<xsl:template match="@category">
        <xsl:value-of select="."/>
        <xsl:text> {</xsl:text>
</xsl:template>
```

〈그림 8〉 Payment 인터페이스를 위한 XSLT 코드 템플릿

4.6 구현 부품을 코드 템플릿 작성

이 단계에서는 컴포넌트 카테고리에 속하는 각 구현 부품을 위한 XSLT 코드 템플릿을 구현한다. 컴포넌트 생성기는 컴포넌트를 구성하는 구현 부품들의 XSLT 코드 템플릿과 재사용 시 생성되는 내부 명세서를 이용하여 구현 부품들의 소스 코드를 생성하고 생성된 구현 부품들의 조립을 통하여 최종 컴포넌트의 소스 코드를 생성한다. 그림 9는 'Payment' 카테고리에 속하는 'Transfer' 구현 부품을 위한 XSLT 코드 템플릿을 보여준다.

4.7 특성 의존성 명세서 정의

특성 의존성 명세서는 여러 가지 특성들 사이에 존재하는 관계 - 예를 들어, 특정 특성이 선택되면 또 하나의 다른 특성은 선택되어서는 안 되어야 하는 관계 - 를 정의한 문서로서, 특성 구성 입력 도구는 이 명세서를 이용하여 재사용자가 입력한 특성 구성의 유효성을 검증한다.

그림 10은 본 도구의 특성 의존성 명세서를 보여

```
<FeatureDependency>
 <Feature name = "PersonalAccount.Payment.Transfer">
  <inclusive>
   <Feature name = "PersonalAccount.Balancing.Quarter">
   <Feature name = "PersonalAccount.Overdraft.Credit">
  </inclusive>
  <exclusive>
   <Feature name = "PersonalAccount.Balancing.30days">
  </exclusive>
  ... ...
</FeatureDependency>
```

〈그림 10〉 특성 의존성 명세서의 예

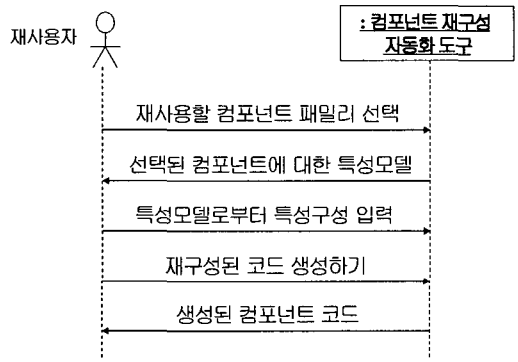
```
<xsl:template match="Specification">
public class <xsl:value-of select="Layers/Component[@category='Payment']/ImplCom"/>
extends <xsl:value-of select="Layers/Component[@category='Overdraft']/ImplCom"/>
implements Payment {
 public <xsl:value-of select="Layers/Component[@category='Payment']/ImplCom"/>() {
 }
 public void transfer(<xsl:value-of select="Configuration/Config_Component[@category=
'NumericType']"/> amount, Account toAccount) {
   if(this.currency() == toAccount.currency()){
    if (amount > 0) { super.debit(amount);  toAccount.credit(amount);  } } } }
</xsl:template>
```

〈그림 9〉 'Transfer'구현 부품을 위한 XSLT 코드 템플릿

준다. 이 명세서에는 반드시 동시 선택되어야 하는 관계(inclusive)와 동시 선택되어서는 안 되는 관계(exclusive)가 정의되어 있다. 이러한 특성 의존성 명세서는 컴포넌트가 적용되는 컨텍스트(context)에 따라 다르게 정의될 것이다.

5. 컴포넌트 재구성 자동화

재사용자가 본 논문의 재구성 자동화 도구를 이용하여 목적에 맞는 컴포넌트의 소스 코드를 생성하는 과정을 UML의 시스템 시퀀스 다이어그램으로 표현하면 그림 11과 같다.



〈그림 11〉 컴포넌트 재구성 자동화 과정

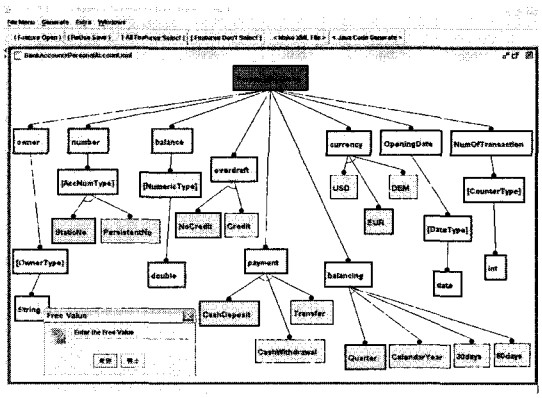
5.1 특성 구성 입력

재사용자는 그림 12와 같이, 컴포넌트 재구성 자동화 도구가 제공하는 특성 모델로부터 요구사

항을 입력하여 특성 구성을 만들고 이를 바탕으로 특정 컴포넌트를 생성한다. 이 도구에서 재사용자가 특정 컴포넌트에 대한 특성 구성을 만들기 위해서 행할 수 있는 동작은 다음과 같다.

(i) 비결정인자 특성에 대해 실제 값 입력
(ii) 선택적 특성에 대해 선택/해제 결정
(iii) 택일적 특성들을 하위 특성으로 가진 특성은, 하위의 택일적 특성들 중 하나 선택
(iv) 논리합 특성들을 하위 특성으로 가진 특성은, 하위의 논리합 특성들 중 하나 이상 선택

그림 12에서 굵은 선으로 둘러싸인 특성은 재사용자가 선택하여 특성 구성에 포함될 특성을 나타내며, 현재 비결정인자 특성인 'OwnerType'의 값을 입력하는 과정을 보여준다. 그림 12의 툴바 메뉴 중에서 <Make XML File> 버튼을 누르면 특성 구성 파일이 생성된다.



〈그림 12〉 재구성 자동화 도구에서의 특성 구성 입력

5.2 컴포넌트 소스 코드 자동 생성

특성 구성을 입력하고 재사용자가 그림 12의 툴바 메뉴 중에서 <Java Code Generate> 버튼을 누르면 컴포넌트 재구성 자동화 도구는 내부 명세서를 생성한 후, 내부 명세서에 따라 최종 컴포넌트에 포함될 구현 부품들에 대한 XSLT 코드 템플릿을 선택하고 조립하여 최종 컴포넌트 소스 코드를 생성한다.

6. 제안 시스템의 평가

6.1 재구성 자동화 도구의 이점

본 논문의 도구는 다음과 같은 이점을 가진다.

• 특성 모델을 통한 컴포넌트 도메인 지식 표현
컴포넌트 패밀리에 속하는 멤버들 사이의 공통점

과 차이점을 특성 모델을 통하여 표현함으로써, 재사용자는 재사용 하고자 하는 컴포넌트에 대한 구현 지식 없이도 컴포넌트 전체의 기능과 특징을 쉽게 이해할 수 있다.

• 컴포넌트 확장성

새로운 기능을 추가하고자 할 때 그 계층의 인터페이스를 구현한 새로운 구현 부품을 그 계층에 포함시키기만 하면 된다.

• 구성 규칙 및 특성 의존성의 변경 용이성

특정 컴포넌트 생성 시 어떤 구현 부품이 선택될 것인지에 대한 규칙과 특성들간의 관계가 독립된 파일로 유지되므로 수정이 용이하다.

6.2 재구성 자동화 도구의 제한점

본 논문의 도구를 이용한 컴포넌트 재사용은 다음과 같은 제한점을 가진다.

• 컴포넌트 패밀리 구축 프로세스의 복잡성

컴포넌트 개발자는 컴포넌트 패밀리를 구축하기 위하여, 일반 컴포넌트 개발보다 복잡한 분석, 설계 및 구현 과정을 거쳐야 한다. 이러한 어려움은 여러 가지 도구 개발로 극복 가능하다.

• 제한된 컴포넌트 구조

모든 컴포넌트는 계층 구조를 가져야 한다는 제한점이 있다.

• 변화가 많은 문제 영역에서의 어려움

컴포넌트 패밀리에 새로운 특성을 추가하거나 삭제할 경우, 특성 모델, 관련 명세서 및 코드 템플릿 등 많은 부분에 있어서 수정이 필요하다. 따라서 본 논문의 컴포넌트 재구성 자동화 기법은 요구 사항의 변화가 적은 안정된 문제 영역에서 더 유용하다.

7. 결론 및 향후 연구

본 논문에서는, 컴포넌트 기반의 소프트웨어 프러덕트 라인 개발에 활용될 수 있는 컴포넌트 재구성 자동화 도구를 제안하였다. 본 도구는 도메인 공학의 주요 산물인 특성 모델로부터 재사용자의 요구 사항을 받아들여 특성 구성(feature configuration)을 만들고 이를 바탕으로 재구성된 컴포넌트 코드를 자동 생성한다. 이를 위하여 컴포넌트 패밀리는 자동 생성 프로그래밍의 한 기법인 GenVoca의 아키텍처를 가지며 XSLT 스크립트가 컴포넌트를 구성하는 구현 부품의 코드 템플릿을 제공한다.

본 논문에서 제안된 도구는, 컴포넌트 재구성성 지원을 통하여 소프트웨어 개발 생산성을 향상시키고, 고수준의 추상화 모델인 특성 모델을 통해서 재사용을 지원하며, 구성 규칙 명세서 및 특성 의존성 명세서를 쉽게 변경할 수 있는 등의 여러 가지 이점을 제공한다.

향후 연구 과제로는, 본 논문의 재구성 자동화 기법을 보다 큰 단위의 컴포넌트 자동 생성에 적용해 보는 것과 컴포넌트 패밀리 구축 프로세스를 지원하는 다양한 도구의 개발 등이 있다.

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, 2002.

[3] W. Tracz, "Domain Specific Software Architecture Pedagogical Example", In ACM SIGSOFT Software Engineering Notes, vol. 20, no. 4, July 1995, pp.46-62.

[4] J. Neighbors, "The Draco Approach to Construction Software form Reusable Comonents", In IEEE Transactions on Software Engineering, vol. SE 10, no. 5, September 1984, pp.564-573, IEEE 1984.

[5] M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allemang, "Organization Domain Modeling(ODM) Guidebook, Version 2.0",

Informal Technical Report for STARS, STAR-SVC A025/0001/00, June 14, 1996.

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

[7] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating Feature Modeling with the RSEB", Proc. of 5th International Conference of Software Reuse, 1998.

[8] K. Kang, S. Cohen, J. Hess, W. Nowak and S. Peterson, "Feature Oriented Domain Analysis (FODA) Feasibility Study", Technical Report, CMU/SEI 90 TR 21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[9] Krzysztof Czarnecki and Ulrich W.Eisenecker, "Generative Programming. Methods, Tools, and Applications", Addison Wesley, 2000.

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

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

[12] Don Batory, Gang Chen, Eric Robertson, and Tao Wang, "Design Wizards and Visual Programming Environments for GenVoca Generators", IEEE Transactions on Software engineering, May 2000, pp.441-452.

[13] Soe, M.S., Zhang, H. and Jarzabek, S. "XVCL: A Tutorial", Proc. 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), ACM Press, Naples, Italy, July 2002, pp.341-349.

[14] Jarzabek, S. and Li, S. "Eliminating Redundancies with a "Composition with Adaptation" Meta programming Technique," Proc. ESECF-SE'03, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, September 2003, Helsinki, pp. 237-246.

[15] P. Bassett, "Framing Software Reuse - Lessons Form Real World", Yourdon Press, Prentice Hall, 1997.

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

◐ 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 프러덕트 라인, 자동 생성 프로그래밍
E-mail : csh@duksung.ac.kr