XSLT 스크립트를 이용한 계층 구조 조립 자동화

정주미[O] 장정아 최승훈
덕성여자대학교 전산및정보통신대학원
{jumi[O], jjung, csh}@duksung.ac.kr

Automatic Composition of Layered Architecture using XSLT Scripts

Ju-Mi Jeong[0] Jeong-Ah Jang Seung-Hoon Choi
Dept. of Computer Science, Duksung Women's University

요 약

소프트웨어 프로덕트 라인은 핵심 소프트웨어 자산의 개발을 위한 도메인 공학과 실제 소프트웨어 부품을 조립하여 구체적인 소프트웨어 시스템을 개발하는 응용 공학을 포함한다. 소프트웨어 프로덕트 라인 구축 시 가장 중요한 점은 특정 도메인에 존재하는 가변성(variability)을 지원할 수 있어야 한다는 것으로, 재사용자의 목적에 따라 효율적으로 소프트웨어를 맞춤 생산할 수 있는 컴포넌트 재구성성(reconfigurability)이 핵심 요소라 할 수 있다. 본 논문에서는 재사용자가 선택한 특성 구성을 바탕으로 계층 구조 조립 자동화를 통해 컴포넌트 코드를 자동 생성하는 도구를 구현하였다. 이를 위하여, 컴포넌트 패밀리의 특성 모델에서 표현되는 차이점에 따라 계층 구조의 각 컴포넌트들이 조립되도록 XSLT 스크립트를 사용하였다. 특성 모델과 XML/XSLT 기술을 이용하여 컴포넌트 코드 생성 시에 재구성성을 지원하고 재사용자의 요구에 맞는 컴포넌트 소스 코드를 자동 생성함으로써 소프트웨어 프로덕트 라인 개발 생산성을 향상시킨다.

1. 서론 및 연구 배경

최근 소프트웨어 개발의 생산성 향상을 위해 보다 큰 단위의 재사용을 가능하게 하는 컴포넌트 기반 소프트웨어 프로덕트 라인에 대한 연구가 활발히 진행되고 있다[1]. 소프트웨어 자산에 존재하는 일반적인 컴포넌트들을 재사용자의 목적에 맞는 컴포넌트로 재구성하기 위해서는 재구성성(reconfigurability)을 지원하는 핵심 자산 또는 컴포넌트 구축이 중요하다. 재사용자의 목적과 환경에 맞는 구체적인 컴포넌트를 자동으로 조립한다면 컴포넌트 재사용을 효율적으로 지원할 수 있다.

본 논문에서는 도메인 공학의 주요 산물인 특성 모델(Feature Model)로부터 특정한 컴포넌트의 요구사항을 입력받아 특성 구성(Feature Configuration)을 만들고, 컴포넌트 명세서와 XSLT 코드 템플릿을 이용하여 재사용 가능한 구체적인 컴포넌트의 코드를 자동 생성하는 도구를 구현하였다. 이를 위하여, GenVoca 계층 구조를 가지는 각 구현 부품들이 특성 구성에 따라 자동으로 재조립되기 위한 기법을 제안한다. 각 구현 부품들은 XSLT 스크립트로 작성되며 계층 구조 조립기는 XSLT 프로세서를 이용하여 구현 부품들을 조립한다

본 논문의 구성은 다음과 같다. 제 2장에서 특성 구성 및 계층 구조 조립을 소개하고, 제 3장에서 XSLT 코드 템플릿을 이용한 계층 구조 조립을 설명한다. 제 4장에서 구현한 예를 보여주며, 제 5장에서 결론 및 향후 연구 과제를 기술한다.

2. 특성 구성 및 계층 구조 조립

본 논문의 전체적인 계층 구조 조립 과정은 그림 1과 같다.

2.1 특성 모델 및 특성 구성

본 연구는 한국과학재단 목적기초연구(과제번호 R06-2002-003-01006-0)지원으로 수행되었음.

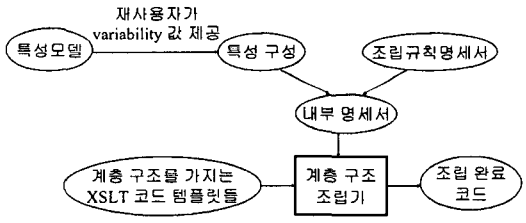

그림 1 XSLT를 이용한 계층 구조 조립 과정

본 논문에서는 특성 중심의 도메인 분석 기법(FODA)[2]을 기반으로 특정 도메인 내에 존재하는 컴포넌트 패밀리의 공통점과 차이점을 특성 모델을 통하여 식별, 조작한다. 그림 2는 본 논문의 예제로 사용한 은행 계좌 컴포넌트의 특성 모델이다.

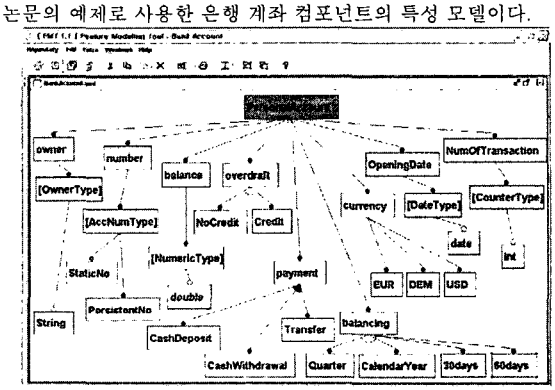

그림 2 은행 계좌 컴포넌트의 특성 모델 예제

그림 2에서 ***overdraft*** 특성은 고객이 예금 잔액을 초과 하여 현금 인출이 가능한지 그 여부를 나타내는 특성이다. ***payment*** 특성은 은행의 주요 업무인 예금, 인출, 이체 중 어느 기능을 제공하는 지를 결정하는 특성이며, ***balancing*** 특성은 계좌 개설일로부터 얼마 후에 이자를 지불할 것인지를 결정하는 특성을 의미한다.

도메인 분석의 결과물인 특성 모델에서 멤버들이 공유하는 공통점은 의무적 특성(mandatory feature)으로 표현되며, 멤버들 간의 차이점은 택일적, 선택적, 인자화 특성으로 표현된다. 특성 모델로부터 재사용자가 결정한 특성 선택 결과를 특정 제품에 대한 " 특성 구성" 이라고 한다[3]. 특성 구성은 재사용자의 요구 사항으로 이용된다.

2.2 조립 규칙 명세서

조립 규칙 명세서 중 하나인 구성 규칙은 재사용자가 제시한 요구사항에 따라 각 계층별로 적절한 구현 컴포넌트를 선택할 수 있게 도와준다. 그림3은 재사용자 선택에 따라 결정되는 ***Payment*** 카테고리의 구현 부품들을 보여준다.

```
<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 3 Payment 카테고리를 생성하기 위한 구성 규칙

2.3 내부 명세서

특성 구성과 구성 규칙 명세서, 계층 구조 명세서로부터 생성되는 내부 명세서는 카테고리별 구현 부품과 재사용자가 선택한 특성 및 입력한 특성값을 보여준다. 그림 4는 은행 계좌 컴포넌트 예에서 작성된 내부 명세서의 일부분이다.

```
<Specification> ……
 <Component category="Payment" depend_on="Overdraft, Payment">
  <ImplCom>CashIn</ImplCom><ImplCom>CashOut</ImplCom>
  <Composition>inheritance</Composition></Component> ……
 <Features>
  <Feature><name>balancing</name><value>30days</value></Feature>
  <Feature><name>30days</name><value>null</value></Feature>
……
```

그림 4 은행 계좌 컴포넌트의 내부 명세서

2.4 계층 구조 및 XSLT 코드 템플릿

본 논문에서는 자동 생성 프로그래밍의 한 기법인 GenVoca에서 정의한 계층 구조를 가정한다. GenVoca는 객체 지향 계층 조립을 바탕으로 컴포넌트 자동 생성기를 개발하기 위한 접근 방법이다[4][5].

컴포넌트 카테고리는 각 특성이 나타내는 주된 책임을 지원하는 구성 부품으로, 각 카테고리 별로 실제 구현 컴포넌트들을 포함한다. 따라서 각 계층별로 표준화된 인터페이스를 제공해 줌으로써 이를 통한 조립, 확장을 가능케 한다. 은행 계좌를 예로 들면 그림 5와 같은 계층 구조를 가진다. 각 구현 부품과 인터페이스는 각각 하나의 XSLT 스크립트 파일을 가진다.

2.5 계층 구조 조립기

특성 모델로부터 재사용자가 필요한 특성을 선택하고, 특성의 커스터마이징 값을 입력하면 이에 따라 특성 구성이 생성된다. 재구성된 특성 구성에 조립 규칙이 적용되면 내부 명세서가 자동으로 작성된다. 이때, XSLT 프로세서를 이용하여 내부 명세서로부터 값을 얻어 구현 부품의 XSLT 엘리먼트를 처리하여 최종 코드를 생성하는 것이 계층 구조 조립기의 역할이다.

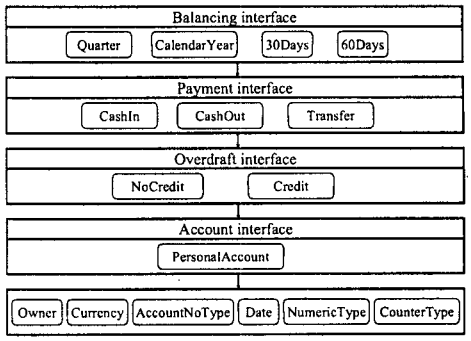

그림 5 은행 계좌 컴포넌트의 계층 구조

3. XSLT 코드 템플릿을 이용한 계층 구조 조립

특성 구성으로부터 계층 구조 조립을 자동화하기 위해서는, 특성 모델에 표현되어 있는 여러 가지 가변성을 계층 구조 조립 시에 지원해야 한다. 각 계층에 속하는 구현 부품들이 조립되어 완성된 코드가 만들어지고, 각 구현 부품은 XSLT 코드 템플릿으로 이루어져 있으므로, XSLT 코드 템플릿 작성 시 가변성(variability)을 지원하기 위한 기법이 필요하다.

표 1은 특성 모델에 표현되는 가변성과 이를 지원하기 위해 계층 구조 조립 시에 발생하는 동작을 나타낸다.

표 1 특성모델에 표현되는 가변성과 계층구조 조립 시 지원방법

특성 모델에 표현되는 가변성	계층구조 조립 시 지원 방법
택일적 특성 (Alternative features)	구현 부품 선택 (특성 구성과 조립 규칙에 따라 적절한 구현 부품이 선택된다)
선택적 특성 (Optional feature)	계층 포함 여부 결정 (선택적 특성의 포함 여부에 따라, 계층 전체가 포함될지 포함되지 않을 지가 결정된다)
논리합 (OR) 관계의 특성들	같은 계층에 존재하는 구현 부품 사이의 상속 관계 형성
인자화 특성 (Parameterized Feature)	인자값 삽입 (인자화 특성의 특정 값을 XSLT 스크립트 코드 템플릿의 한 부분에 삽입한다)

특성 모델에 표현되는 가변성을 지원하기 위한 XSLT 스크립트 작성 기법을 본 절에서 기술한다.

3.1 택일적 특성들 지원

택일적 특성(Alternative features)이란, 여러 개의 택일적 특성들 중 오직 한 개의 특성만이 특성 구성에 포함되는 특성을 의미한다. 조립 규칙 명세서에 따라 계층 구조를 구성하는 각 계층으로부터 적당한 구현 부품을 선택한다. 그림 6은 택일적 특성인 StaticNo 와 PesistentNo 특성을 하위 특성으로 갖는 ***AccNumType*** 의 XSLT 템플릿 중 일부를 보여준다.

```
class <xsl:value-of select="Configuration/Config_Component[@category=
'AccNumType']"/> { private long number_;  static long counter_ = 1;
  public <xsl:value-of select="Configuration/Config_Component
[@category='AccNumType']"/>() {
    <xsl:if test = "Configuration/Config_Component[@category=
'AccNumType']/. = 'PersistentNo'"> …… </xsl:if>
    number_ = counter_++;  } ……
```

그림 6 택일적 특성에 적용되는 XSLT 코드 템플릿

3.2 선택적 특성 지원

선택적 특성(Optional features)의 경우, 특정 구현 부품 안에서 자신의 위 계층이 선택된 경우와 선택되지 않은 경우를 식별하기 위한 XSLT 엘리먼트가 필요하다. 그림 7은 선택적 계층의 선택을 위해 사용된 <xsl:when~ > 의 예를 보여준다.

```
<xsl:template match="Layers/Component[@category=' 선택적계층']">
  <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>
```

그림 7 선택적 계층에 적용되는 XSLT 코드 템플릿

3.3 논리합 관계의 특성들 지원

논리합(OR) 관계의 특성은 이 특성들 중에서 한 개 이상의 특성들이 다수 포함될 수 있으며 같은 계층의 구현 컴포넌트들끼리 조립되어야 한다. 이를 위해 같은 계층의 구현 컴포넌트들끼리 상속관계를 가지고, 같은 인터페이스를 구현하도록 XSLT를 작성한다. 그림 8은 ***Payment*** 특성의 하위 특성들로 OR 관계인 CashDeposit, CashWithdrawal, Transfer 중 ***Transfer*** 특성의 코드 템플릿 중 일부분이다.

```
<xsl:template match="Layers/Component[@category='Payment']">
  <xsl:if test="count(ImplCom) &gt; 1">
    <xsl:if test="ImplCom[. = 'Transfer' and position() = 1]">
      <xsl:value-of select="/Specification/Layers/Component[@category=
'Overdraft']/ImplCom"/> implements <xsl:apply-templates select="@*"/>
    </xsl:if>
    <xsl:if test="ImplCom[. = 'Transfer' and position() != 1]">
      <xsl:value-of    select="ImplCom/self::node()[.='Transfer']/preceding-
sibling::*[position()=1]"/> implements <xsl:apply-templates select="@*"/>
    </xsl:if>
  </xsl:if> ……
</xsl:template>
```

그림 8 논리합 관계의 특성에 적용되는 XSLT 코드 템플릿

3.4 인자화 특성 지원

인자화 특성(parameterized features)은 재사용자가 값을 직접 입력하는 것에 의해 특성 값이 결정되는 특성을 의미하며, 특성 구성에 포함되어 있는 인자 값을 그대로 얻어온다. 그림 9는 ***CounterTyper*** 특성에서 인자의 값이 XSLT 스크립트 코드 템플릿의 한 부분에 삽입되는 부분을 보여준다.

```
public class <xsl:apply-templates select="Configuration/Config_Component
[@category='CounterType']"/> {
  <xsl:value-of select="Configuration/Config_Component[@category=
'CounterType']"/> counterType;
    public <xsl:apply-templates select="Configuration/Config_Component
[@category='CounterType']"/> (<xsl:value-of select="Configuration/
Config_Component[@category='CounterType']"/> user_type)
  {    counterType = user_type;    } ……
} ……
```

그림 9 인자화 특성에 적용되는 XSLT 코드 템플릿

4. 구현 예

4.1 특성 구성 생성

그림 10은 특성 모델로부터 재사용자가 요구사항을 입력하여 특성 구성을 만들고 이를 바탕으로 특정 컴포넌트를 생성할 수 있는 컴포넌트 재구성 자동화 도구를 보여준다. 굵은 선의 특성은 재사용자가 특성 구성에 포함시키길 원하는 특성을 나타낸다.

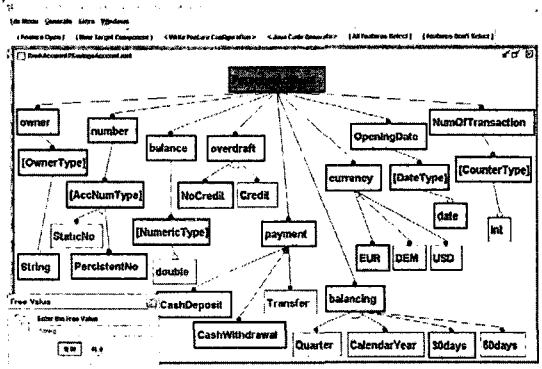

그림 10 은행 계좌 컴포넌트의 재구성 도구 사용 예

4.2 내부 명세서 생성 및 코드 생성

그림 11은 그림10 에서 보여지는 특성 구성에 따라 자동 생성된 코드 중 일부를 보여준다.

```
public class NoCredit extends PersonalAccount implements Overdraft{
  public NoCredit() {  super();  }
  public void debit(int amount){
    if (amount <=0)   return;
    if (super.balance() - amount >= 0)   super.debit(amount);
  }
}
```

그림 11 Overdraft 특성을 구현한 소스 코드의 예

5. 결론 및 향후 연구 과제

본 논문에서는, GenVoca 계층 구조를 가지는 각 구현 부품들이 특성 구성에 따라 자동으로 재조립되기 위한 기법을 제안하였다. 특성 모델과 XML/XSLT 기술을 이용하여 재구성성을 지원하고 재사용자의 요구에 맞는 컴포넌트 소스 코드를 자동 생성하였다.

향후 과제로는 본 논문에서는 상속(inheritance)을 통해서 이뤄진 계층구조의 조립(compotision)을 집합관계(aggregation)로 확장시켜 적용해 보는 것이다. 또한, 특성 모델에서 재사용자에 의해 동시 선택이 가능한 특성들(inclusive)과 동시에 선택되어서는 안되는 특성들(exclusive)의 관계를 정의한 특성 의존성(Feature Dependency)에 대한 연구를 수행하는 것이다.

참고문헌

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

[2] 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

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

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

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