XMI를 이용한 디자인패턴 모델링에 관한 연구

A Study on Design Pattern Modeling Using XMI

최한용, 이돈양*, 김귀정**
한북대학교, 경인여자대학*, 건양대학교**

Choi Han-Yong, Lee Don-Yang*, Kim Gui-Jung**
HanBuk Univ., KyungIn Woman's College*, KonYang Univ.**

요약

기존의 디자인패턴도구에서는 디자인패턴을 정형화된 방법으로 표현하고 설계할 수 없었으므로, 설계정보에 디자인패턴이 재사용 될 때 중복적으로 설계정보가 표현되었다. 그리고 정형화된 디자인패턴은 설계자의 의도에 맞도록 확장 가능하여야 하며, 설계자의 설계의도에 따라 객체를 표현할 수 있어야 한다. 따라서 본 논문에서는 디자인패턴을 정형화하기 위한 메타모델로 XMI를 이용하여 정의하는 방법과 메타모델로 표현된 디자인패턴을 확장하기 위한 메타모델을 표현할 수 있도록 하였다.

Abstract

It was expressed repeated information on reuse design pattern in the design information because it was impossible to build a standardized design pattern in the existing design pattern tools. Also, It must possible to extend standardized design pattern just as one intended of designer. And It is able to express objects just as one intended of designer. Therefore, the representative characteristic of this paper was able to define XMI metamodel to express design pattern, and was able to extend metamodel to express design pattern.

I. 서 론

프로그래머들은 자신의 라이브러리를 조합하여 새로운 라이브러리를 생성하거나 기존의 라이브러리를 조합하여 재사용하고 있다[1]. 그러나 설계단계에서는 설계정보를 정형화하여 새로운 설계자들이 재사용할 수 있도록 제공하고 있지 못하며, 또한 제공되는 설계정보를 조합하여 새로운 설계정보를 구성할 수 있는 환경을 지원하지 못하고 있다[2][4]. 그리고 디자인패턴 관련 도구들이 단순히 패턴을 UML로 표현된 정보를 제공할 뿐 설계에 재사용할 수 없다[4][5][6]. 따라서 본 논문은 기존의 설계정보를 재사용하기 위한 방법으로 반복되는 과거의 설계정보를 정형화하고 추상화한 디자인패턴을 독립된 컴포넌트로 설계하고 이를 지원하는 환경구축에 관하여 연구하였다[3]. 디자인패턴의 설계정보는 XMI를 기반으로 메타데이터를 생성하였으며, UML로 설계된 디자인패턴은 XMI 메타데이터로 저장이 가능하게 되었다. 그리고 디자인패턴기반 설계를 위해 정형화된 디자인패턴을 확장가능 하도록 확장된 디자인패턴 모델을 설계하였다. 따라서 확장된 디자인패턴은 패턴을 하나의 컴포넌트단위로 취급하여 설계단계에서 재사용이 가능하며 패턴간의 조립이 가능하다.

II. XMI를 이용한 디자인패턴 설계

2.1 디자인패턴의 메타구조 설계

본 논문에서는 디자인패턴 구조를 컴포넌트화하기 위한 메타 데이터로써 XMI를 이용하여 디자인패턴을 위한 메타 모델을 정의하였다.

```
<XMI xmi.version="1.0">
#HEADER#// Header
#CONTENT#// Content
#EXTENSIONS#// Extensions
</XMI>

// Header
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>UPOD</XMI.exporter>
    <XMI.documentation>
    <XMI.metamodel xmi.name="UML" />
                  </XMI.header>
// Content
  <XMI.content>
    <Model xmi.id="#ID#">
    <xmi.name>#PATTERN_NAME#</xmi.name>
    <ownedElement>
    [repeat CLASS]
      #CLASS#
    [/repeat CLASS]
    [repeat ASSOCIATION]
      #ASSOCIATION#
    [/repeat ASSOCIATION]
    </ownedElement>
   </Model>
  </XMI.content>

      <Class xmi.id="#ID#">
        <xmi.name>#CLASS
        <feature>
          [repeat OPERATION]
            #OPERATION#
          [/repeat OPERATION]
        </feature>
      </Class>

          <Operation xmi.id="#ID#">
            <xmi.name>#OPER_NAME#</xmi.name>
            <visibility xmi.value="#VISIBILITY#"/>
          </Operation>

      <Association xmi.idref="#ID#">// 관계 ID
        <xmi.name>#ASSOC_NAME#</xmi.name>
        <connection>
              :
        </connection>
      </Association>
```

▶▶ 그림 1. XMI를 이용한 디자인패턴의 메타모델

디자인패턴은 크게 두 가지의 메타 모델로 설계하였다. 첫째, 시스템 정의(PreDefined) 디자인패턴을 표현하기 위한 메타 모델과 둘째, 설계자의 의도에 의해 이를 사용자 정의(UserDefined) 디자인패턴을 표현하기 위한 확장 메타 모델로 정의하였다. 시스템 정의 메타모델은 표준함수와 같이 시스템에서 제공하는 디자인패턴을 의미하며, 이를 조합하여 확장한 디자인패턴은 사용자 정의 패턴이다.

2.2 확장디자인패턴의 메타구조 설계

[그림 1]은 미리 정의된 디자인패턴을 표현하기위해 XMI를 이용하여 메타모델을 정의한 것이다.

```
<pattern> → <pattern> * <association>
<pattern> → <class> * <association>
<class> → <operation>
<association> → <relationship code>
```

▶▶ 그림 2. 디자인패턴의 메타모델 구성 문법

그리고 확장되는 디자인패턴을 표현하기 위한 메타모델은 미리 정의된 디자인패턴을 표현한 메타모델을 이용하여 조합되는 디자인패턴 정보와 관계정보를 포함하여 새로운 디자인패턴이 표현될 수 있도록 XMI로 표현된 메타 모델을 정의하였다.

```
<XMI xmi.version="1.0">
#HEADER#
#CONTENT#
#EXTENSIONS#
</XMI>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>UPOD</XMI.exporter>
    <XMI.documentation>
    <XMI.metamodel xmi.name="UML" />
  </XMI.header>
  <XMI.content>
    <Model xmi.id="#ID#">
    <xmi.name>#HYBRID_PATTERN_NAME#</xmi.name>
    <ownedElement>
    [repeat PATTERN]
      #PATTERN#
    [/repeat PATTERN]
    [repeat ASSOCIATION]
      #ASSOCIATION#
    [/repeat ASSOCIATION]
    </ownedElement>
   </Model>
  </XMI.content>
      <PATTERN xmi.id="#ID#">
        <xmi.name>#PATTERN_NAME#</xmi.name>
      </PATTERN>
      <Association xmi.idref="#ID#">
        <xmi.name>#ASSOC_NAME#</xmi.name>
        <connection>
          <feature>
            #class#
          </feature>
        </connection>
      </Association>
      <Class xmi.id="#ID#">
        <xmi.name>#CLASS_NAME#</xmi.name>
      </Class>
```

▶▶ 그림 3. 확장된 디자인패턴 모델

XMI 구조를 이용하여 [그림 1]에 표현된 디자인패턴의 메타모델은 <Header>, <Content>, <Extension>의 세 부분으로 구성하였다. <Header> 부분은 메타 모델의 종류와 XMI 정보를 기술한다. <Content> 부분에는 실제 디자인패턴의 메타모델 정보를 갖는 XMI 정보를 기술한다. 마지막으로 <Extension> 부분은 메타 모델의 확장된 정보를 표현하기 위한 부분으로 구성되어 있다. 따라서 디자인패턴 메타모델을 구성하기 위한 문법을 정의하면 [그림 2]와 같이 정의된다. 정의된 문법에 다르면 디자인패턴은 다시 여러 개의 패턴과 관계의 집합으로 구성되며, 각 디자인패턴은 다시 클래스와 관계의 집합으로 구성하였으며, 관계는 관계 설정 코드로 구성하였다. 따라서 [그림 3]과 같이 사용자 정의 패턴은 시스템 정의 패턴의 조합으로 확장된 새로운 디자인패턴을 구성하여 중복된 설계정보를 줄일 수 있도록 하였다.

2.3 디자인 패턴의 인터페이스 설계

서로 독립된 디자인 패턴을 조합하기 위해서는 두 개의 개별적인 개체사이에 내포된 계약관계를 나타내기 위해 사용할 수 있다. 인터페이스는 객체로 하여금 외부세계에 대해 제한적이지만 충분한 상호작용을 표현할 수 있는 방법을 제공한다.

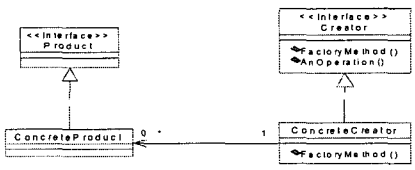
```
module <identifier>
{
  <type declarations>;
  <constant declarations>;

  interface <identifier> [:<inheritance>]
  {
    <type declarations>;
    <constant declarations>;
    <attribute declarations>;

    [<op_type>] <identifier>(<parameters>)
    [raises exception] [context];
                :
                :
    [<op_type>] <identifier>(<parameters>)
    [raises exception] [context];
                :
                :
  }
}
```

▶▶ 그림 4. 패턴의 인터페이스 IDL 구조

디자인패턴을 조합하기위해 IDL을 이용하여 [그림 4]와 같이 인터페이스를 정의하였고 정의된 인터페이스를 이용하여 디자인패턴이 조합가능한지 알 수 있다. 이때 조합이 가능하다는 것은 패턴내의 클래스 사이에 관계가 성립할 수 있다는 것을 의미하며, 이것을 IDL에서 바라보면 관계라는 것은 클래스 간의 인터페이스를 위한 메시지 교환을 의미한다. 인터페이스에는 "inheritance" 영역에 인터페이스의 관계정보를 나타내며, 클래스간의 관계정보 설정을 하기 위한 함수를 표현하기위해 함수의 형(type)과 입력과 출력에 사용되는 매개변수 및 입출력 방향을 표현하고 있다. 따라서 디자인패턴으로 구성된 컴포넌트를 조합하기 위해 인터페이스에 정의된 정보에 의해 패턴사이의 관계를 설정하기위한 클래스들 사이의 조합을 위한 매개변수 전달방식을 결정할 수 있다.

III. 메타모델을 이용한 디자인패턴의 표현

본 논문에서는 디자인패턴을 표현하기 위해 XMI를 이용한 기본 메타모델과 확장 메타모델을 정의하였다. [그림 5]는 "Factory Method" 패턴을 XMI 메타 모델을 이용하여 표현하기 위해 "Factory Method" 패턴의 구조를 UML로 표현한 것이다.

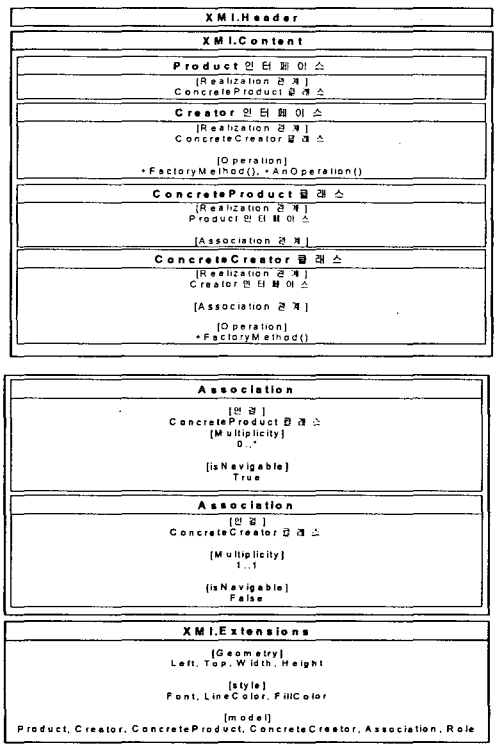


▶▶ 그림 5. UML로 표현한 Factory Method

<XMI.Content> 항목에서는 Factory Method 패턴의 메타모델들 즉, Product, Creator 인터페이스, ConcreteProduct, ConcreteCreator 클래스, Realization, Association 관계에 대한 메타데이터를

기술하게 된다. 그리고 Relationship 관계에서는 상호 연결된 메타모델에 그 관계를 덧붙여 기술하여 방향성을 갖게 한다. <XMI.Extensions>에서는 <XMI.Content>에서 기술한 각각의 메타모델들에 대한 UML 도구에서의 Presentation 정보 즉, 위치와 스타일을 기술한다.

"Factory Method"을 구성하는 인터페이스와 클래스 정보가 표현된 메타 모델, 관계정보가 표현된 메타모델 정보를 이용하여 [그림 6]과 같이 디자인패턴을 정형화 할 수 있다.



▶▶ 그림 6. XMI로 정형화한 디자인 패턴

그리고 Factory Method 패턴을 조합하기위해 [그림 7]과 같이 Factory Method 패턴에 대해 IDL을 이용하여 인터페이스를 정의하였다.

```
module FactoryMethod
{
  interface Creator
  {
    attribute Product product;
    product FactoryMethod();
    void AnOperation();
  }

  interface ConcreteCreator:Creator
  {
    ConcreteProduct* FactoryMethod();
  }
}
```

▶▶ 그림 7. Factory Method패턴의 인터페이스

인터페이스 정보 내에는 Creator와 ConcreteCreator 클래스에 대한 인터페이스를 갖고 있다. 그리고 Creator 클래스는 하나의 속성(attribute)과 두개의 함수를 갖고 있다. 이때 Creator 클래스와 관계를 갖기 위해서는 Creator 클래스에 대한 interface 정보를 만족해야만 관계를 설정할 수 있다. Creator 클래스는 FactoryMethod()의 결과를 product 속성에 저장하고, ConcreteCreator 클래스는 ConcreteProduct 객체를 생성하여 전달하도록 IDL로 정의하였다.

IV. 결 론

본 논문에서 표현한 XMI 메타데이터를 이용하여 디자인패턴을 표현하고 디자인패턴을 컴포넌트로 정의하기 위해 컴포넌트 조립을 위한 IDL 인터페이스를 통한 객체지향 시스템을 설계시 과거의 설계정보를 표현하고 있는 디자인패턴을 효율적으로 재사용할 수 있을 것이다. 향후에는 디자인패턴이 적용되지 않은 상태에서 이미 설계된 UML 설계정보를 재사용하기 위해 UML 다이어그램에서 디자인패턴을 추출하고, 추출된 디자인패턴정보를 XMI 메타정보로 표현할 수 있는 방법이 요구된다.

▌참 고 문 헌▐

[1] Theo Harder, Wolfgang Mahnke, Norbert Ritter, Hans-Peter Steiert, "Generating Versioning Facilities for a Design-Data Repository Supporting Cooperative Applications," IJCIS 2000, pp.117-146.

[2] V.B.Misic, S.Moser, "Measuring Class Coupling and Cohesion :A Formal Metamodel Approach", ASPEC'97, pp.31-40, Dec.,1997.

[3] ModelMaker version 5.0, http://www.modelmaker.demon.nl/, 2000.

[4] Rudolf K. Keller, Richard Schauer, "Design Components:Towards Software Composition at the Design Level", ICSE'98, pp.282-291.

[5] Mark Grand, "Patterns in Java," WILEY, 1998

[6] HTTP://WWW.OMG.ORG