XMI 기반의 디자인패턴 합성

XMI based Design-Pattern Composition

최한용*, 이돈양**
한북대학교 컴퓨터공학과*, 세종대학교 컴퓨터공학부**

Han-Yong Choi(hychoi@hanbuk.ac.kr)*, Don-Yang Lee(dylee11@sejong.ac.kr)**

요약

소프트웨어 생명주기의 각 단계에서 기존의 경험을 재사용하기 위해 다양한 연구가 이루어 져왔다. 그리고 추상화수준이 높은 단계에서 설계상의 문제를 해결할 수 있도록 디자인 패턴에 관한 다양한 연구가 이루어지고 있다. 그러나 생명주기중 설계 단계에서는 잘 정의된 설계 정보를 정형화하여 새로운 설계자들이 재사용할 수 있도록 추상화된 설계 정보를 제공하고 있지 못하고 있다. 또한 기존의 설계 정보를 재사용하여 새로운 설계 정보를 합성할 수 있는 환경을 지원하지 못하고 있다. 그러므로 본 논문에서는 설계상의 문제를 해결할 수 있는 디자인패턴을 합성하여 기존의 설계 정보를 재사용할 수 있으며, 또한 정형화된 설계 정보를 확장해 나갈 수 있도록 하기 위해 XMI를 기반으로 디자인 패턴의 메타모델을 정의하였다. 그리고 XMI로 정형화된 디자인패턴의 메타모델을 이용하여 디자인패턴을 합성할 수 있도록 하였다. 따라서 정형화된 디자인패턴의 메타데이터를 이용하여 설계상의 문제점을 정형화할 수 있고, 디자인 패턴 기반의 설계를 위해서 디자인 패턴으로 설계된 기존의 설계 정보를 합성하여 재사용할 수 있다.

■ 중심어 :□컴포넌트□디자인 패턴□합성□

Abstract

Many researches have achieved to reuse existent experience at each step of software life cycle. It is achieved various study about Design Pattern serving that abstract level solves design problem at high level. But, design step is not supplying design information that is abstracted so that new designers may reuse standardizing defined good design information. Therefore, in this paper, to do so that can reuse existent design information because composes Design Pattern can solve problem at design step, and extend design information that also is standardized XMI to base meta model of Design Pattern. And I did so that can compose Design Pattern utilizing MetaModel of Design Pattern standardized by XMI. Therefore, software designer can reuse composing existent design information that is could standardize problem at design because uses MetaData of standardized Design Pattern, and designs by Design Pattern base.

■ keyword :□Component□Design Pattern□Composition□

접수번호 : #080914-002
접수일자 : 2008년 09월 14일
심사완료일 : 2008년 10월 27일
교신저자 : 최한용, e-mail : hychoi@hanbuk.ac.kr

I. 서 론

소프트웨어 개발 환경의 빠른 변화와 중복개발 비용을 줄이고자 기존의 개발 경험을 재사용하고자 한다. 구현 단계에서 프로그래머들은 자신의 구현목적에 맞는 라이브러리를 조합하여 새로운 라이브러리를 생성하거나 기존의 라이브러리를 조합하여 재사용하고 있다[1][2]. 그리고 구현단계에서의 재사용은 개발환경에 의존적이며 복잡도가 높은 코드를 이해해야하는 어려움을 갖고 있으므로 추상화정도가 높은 설계 단계에서 재사용을 요구하고 있다[3]. 그러나 설계 단계에서는 잘 정의된 설계 정보를 정형화하여 새로운 설계자들이 재사용할 수 있도록 제공하고 있지 못하며, 또한 제공되는 설계 정보를 합성하여 설계 정보를 확장할 수 있는 환경을 지원하지 못하고 있다[4][5]. 따라서 설계 단계에서 주어진 문제를 해결할 수 있는 정형화된 설계 정보를 표현하기 위해서 디자인 패턴에 관한 많은 연구가 이루어지고 있다. 이와같이 디자인패턴을 적용하려는 도구는 Omnibuilder[6], ModelMaker[7]와 같은 도구가 있다. 그러나 Omnibuilder와 ModelMaker에서는 단일 패턴의 코드생성이 주 목적인 설계 도구로써 디자인패턴을 지원하고 있으므로, 설계 중 디자인패턴의 코드 템플릿을 삽입하여 설계 정보를 재사용하도록 하고 있다[8]. 따라서 시스템 설계 단계에 과거의 설계 정보를 재사용하기 위해 설계 정보의 추상화 단계가 아닌 구체화된 코드를 읽어야 하므로 개발 환경에 의존적이며 설계 정보의 가독성(readability)이 떨어지는 문제점을 갖고 있다[9][10]. 그리고 단일 패턴의 적용단계에서 기존에 정의된 패턴을 합성하여 새로운 설계 정보로써 재사용이 불가능하다. 즉, 디자인패턴을 재사용하기 위해 설계 단계에서는 개발 환경에 의존적이지 않으며 추상화된 설계 정보의 표현이 필요하다[11-13]. 그러므로 본 논문에서는 디자인패턴을 정형화하여 표현할 수 있으며, 설계 단계에서 디자인 패턴을 합성하기 위해 XMI를 이용하여 디자인 패턴의 메타모델을 표현하였다. 따라서 디자인 패턴은 XMI를 이용하여 정형화된 메타모델로 표현가능하고, 이를 합성하여 디자인 패턴을 설계할 수 있도록 하기위해 IDL을 이용하여 디자인 패턴의 인터페이스를 정의하였다[11]. 디자인 패턴을 합성하기 위해 정의된 IDL 인터페이스는 정형화된 디자인패턴의 메타데이터를 합성하여 확장된 디자인패턴 메타데이터로 표현할 수 있도록 하였다. 본 논문의 구성은 2장에서 디자인패턴의 메타 모델의 정의방법을 고찰하였으며, 3장에서는 메타모델을 이용한 디자인패턴의 합성방법을 제안하였다. 그리고 4장에서는 목적 시스템에 적용한 결과를 비교 설명하였으며, 끝으로 5장에서는 결론에 대하여 서술하였다.

II. 디자인패턴의 메타 모델 정의

1. XMI를 이용한 디자인패턴의 메타 모델 정의

본 논문에서는 환경에 독립적인 디자인패턴 구조를 정형화 하여 표현하기 위해 메타 모델을 이용하였다.

```
<XMI xml.version='1.0'>
#HEADER#// Header
#CONTENT#// Content
#EXTENSIONS#// Extensions
</XMI>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>UFOD</XMI.exporter>// 사용 도구
    <XMI.documentation>
     // 메타모델과 XMI 버전
    <XMI.metamodel xml.name='UML' xml.version='1.0'/>
  </XMI.header>
  <XMI.content>
   <Model xml.id='#ID#'>
    <xml.name>#PATTERN_NAME#</xml.name>//  패턴 명
    <ownedElement>
    [repeat CLASS]// 모든 클래스/인터페이스 메타모델 기술
      #CLASS#
    [/repeat CLASS]
    [repeat ASSOCIATION]// 직접 연결된 관계정보 기술
      #ASSOCIATION#
    [/repeat ASSOCIATION]
    </ownedElement>
   </Model>
  </XMI.content>
// 클래스 정보를 기술하는 세그먼트
      <Class xml.id='#ID#'>// 클래스 ID
        <xml.name>#CLASS_NAME#</xml.name>// 클래스명
        <feature>
          [repeat OPERATION]// 오퍼레이션 메타모델 기술
            #OPERATION#
          [/repeat OPERATION]
        </feature>
     </Class>
// 오퍼레이션 정보를 기술하는 세그먼트
          <Operation xml.id='#ID#'>
            // 오퍼레이션명
            <xml.name>#OPER_NAME#</xml.name>
            <visibility xml.value='#VISIBILITY#'/>// 가시성
          </Operation>
// 관계정보를 기술하는 세그먼트
     <Association xml.idref='#ID#'>// 관계 ID
       <xml.name>#ASSOC_NAME#</xml.name>// 관계명
       <connection>// 관계정보
              :
       </connection>
     </Association>
```

그림 1. 기본 디자인패턴 메타 모델

메타 데이터로써는 XMI(XML Metadata Interchange)를 이용하여 디자인패턴을 위한 메타 모델을 표현하였다.

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

그림 2. 디자인패턴의 메타모델 구성 문법

디자인패턴은 크게 두 가지의 메타 모델로 설계하였다. 첫째, 시스템 정의 디자인패턴을 표현하기 위한 메타 모델과 둘째, 합성된 디자인패턴을 표현하기 위한 확장 메타 모델로 정의하였다.

```
<XMI xmi.version="1.0">
#HEADER#// Header 코드 세그먼트
#CONTENT#// Content 코드 세그먼트
#EXTENSIONS#// Extensions 코드 세그먼트
</XMI>
// Header 코드 세그먼트
    <XMI.documentation>
      <XMI.exporter>UPOD</XMI.exporter> // 사용 도구
    <XMI.documentation>
    // 메타모델 과 XMI 버전
    <XMI.metamodel xmi.name="UML" xmi.version="1.0"/>
  </XMI.header>
// Content 코드 세그먼트
  <XMI.content>
    <Model xmi.id="#ID#">
    //혼합 패턴명
    <xmi.name>#HYBRID_PATTERN_NAME#</xmi.name>
    <ownedElement>
    [repeat PATTERN]// 모든 패턴 메타모델 기술
      #PATTERN#
    [/repeat PATTERN]
    // 패턴간의 직접 연결된 관계 정보 기술
    [repeat ASSOCIATION]
      #ASSOCIATION#
    [/repeat ASSOCIATION]
    </ownedElement>
   </Model>
  </XMI.content>
// 패턴 정보를 기술하는 세그먼트
      <PATTERN xmi.id="#ID#">// 패턴 ID
        // 패턴 명
        <xmi.name>#PATTERN_NAME#</xmi.name>
      </PATTERN>
// 관계정보를 기술하는 세그먼트
     <Association xmi.idref="#ID#">// 관계 ID
       <xmi.name>#ASSOC_NAME#</xmi.name>// 관계명
       <connection>// 관계 정보
        <feature>
         #class#
        </feature>
       </connection>
     </Association>
// 클래스 정보를 기술하는 세그먼트
      <Class xmi.id="#ID#">// 클래스 ID
        <xmi.name>#CLASS_NAME#</xmi.name>// 클래스명
      </Class>
```

그림 3. 확장 디자인패턴 메타 모델

시스템 정의 메타모델은 표준 함수와 같이 시스템에서 제공하는 디자인패턴을 의미하며, 이를 합성하여 확장한 디자인패턴은 사용자 정의 패턴이다. 미리 정의된 디자인패턴을 표현하기위해 [그림 1]과 같이 XMI를 이용하여 메타모델을 정의하였다. 그리고 컴포넌트로 조립된 디자인패턴의 메타모델을 구성하기 위한 문법을 정의하면 [그림 2]과 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. 이와 같이 사용자 정의 패턴은 시스템 정의 패턴의 합성으로 새로운 디자인패턴을 구성한다. 그러므로 [그림 3]과 같이 확장되는 디자인패턴을 표현하기 위한 메타모델은 미리 정의된 디자인패턴을 표현한 메타모델을 이용하여 합성되는 디자인패턴 정보와 관계정보를 포함하여 새로운 디자인패턴이 표현될 수 있도록 XMI로 표현된 메타 모델을 정의하였다. XMI 구조를 이용하여 [그림 1]에 표현된 디자인패턴의 메타모델은 <Header>, <Content>, <Extension>의 세 부분으로 구성하였으며, <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 구성하였다.

2. 디자인패턴 합성을 위한 인터페이스 정의

서로 독립된 패턴 컴포넌트를 합성하기 위해서는 컴포넌트를 표현하기위한 인터페이스정보를 갖고 있어야 한다. 인터페이스라는 것은 두 개의 개별적인 개체 사이에 내포된 계약관계를 나타내기 위해 사용할 수 있다. 인터페이스는 객체로 하여금 외부세계에 대해 제한적이지만 충분한 상호작용을 표현할 수 있는 방법을 제공한다. 인터페이스는 객체가 동작하는 데 있어 필요한 입력 파라미터를 정의하며 객체로부터 반환되는 값을 또한 정의한다. 또한 객체의 행동양식을 캡슐화 하는 방법은 또한 객체의 내부 작용을 노출된 인터페이스로부터 격리시키는 결과를 낳는다. 디자인패턴을 합성하기위해 IDL(interface definition language)을 이용하여 인터

페이스를 정의하고 정의된 인터페이스를 이용하여 디자인패턴이 합성가능한지 알 수 있다. 이때 합성이 가능하다는 것은 패턴내의 클래스 사이에 관계가 성립할 수 있다는 것을 의미하며, 이것을 IDL에서 바라보면 관계라는 것은 클래스 간의 인터페이스를 위한 메시지 교환을 의미한다.

```
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 인터페이스를 위한 IDL 구조

메시지 교환으로 표기되는 관계정보는 결국 인터페이스에 정의된 함수의 호출 에 의해 메시지가 교환되는 것이므로, 관계가 성립하기 위해서는 함수 호출관계가 성립함을 의미한다. 따라서 인터페이스에 정의된 함수의 입출력 정보에 의해 관계가 성립할 수 있는가를 결정하게 된다. 즉, 관계를 설정하여 합성할 수 있는가를 결정할 수 있다. [그림 4]는 디자인패턴 컴포넌트의 인터페이스 정보를 표현하기위한 IDL 구조를 보여주고 있다. 인터페이스에는 "inheritance" 영역에 인터페이스의 관계 정보를 나타내며, 클래스간의 관계정보 설정을 하기위한 함수를 표현하기위해 함수의 형과 입력과 출력에 사용되는 매개변수 및 입출력 방향을 표현하고 있다. 따라서 디자인패턴으로 구성된 컴포넌트를 합성하기 위해 인터페이스에 정의된 정보에 의해 패턴사이의 관계를 설정하기위한 클래스들 사이의 합성을 위한 매개변수 전달방식을 결정할 수 있다.

3. 메타모델을 이용한 디자인패턴의 표현

본 논문에서는 디자인패턴을 표현하기 위해 XMI를 이용한 메타모델을 정의하였다. 다음 그림은 "Factory Method" 패턴을 XMI 메타 모델을 이용하여 표현한 것이다.

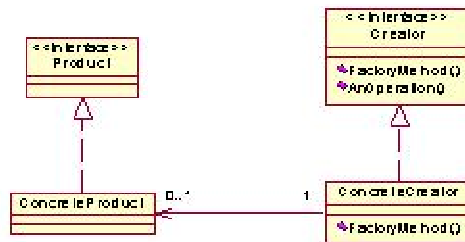

그림 5. UML로 표현한 Factory Method 패턴

"Factory Method" 패턴은 객체 생성에 대한 책임을 인터페이스에 전가함으로써 객체생성에 대한 의존성을 줄임으로써 유지보수 비용을 절감하는 특징을 갖고 있으며 [그림 5]은 "Factory Method" 패턴의 구조를 UML로 표현한 것이다.

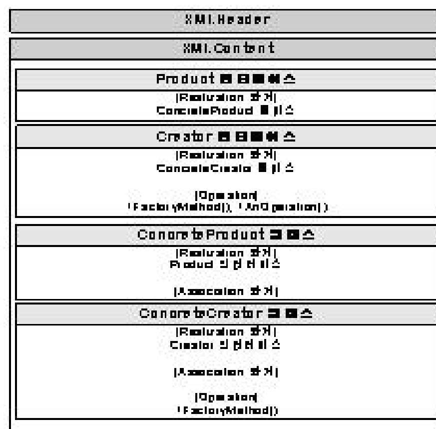

그림 6. Factory Method 패턴 클래스 부분

"Factory Method" 패턴은 UML에 의해 모델링 되고 모델링된 디자인패턴은 메타 데이터베이스에 저장하기 위해 각각의 메타 모델 정보를 XMI 메타 모델로 표

현한다. <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.

[그림 6]에서는 "Factory Method"을 구성하는 인터페이스와 클래스 정보가 표현된 메타 모델이며, [그림 7]는 관계정보가 표현된 메타모델이다. Product 인터페이스와 ConcreteProduct 클래스 사이에 연결된 화살표는 두 객체사이에 Realization 관계가 있음을 나타내며, Creator 인터페이스와 ConcreteCreator 클래스도 마찬가지이다.

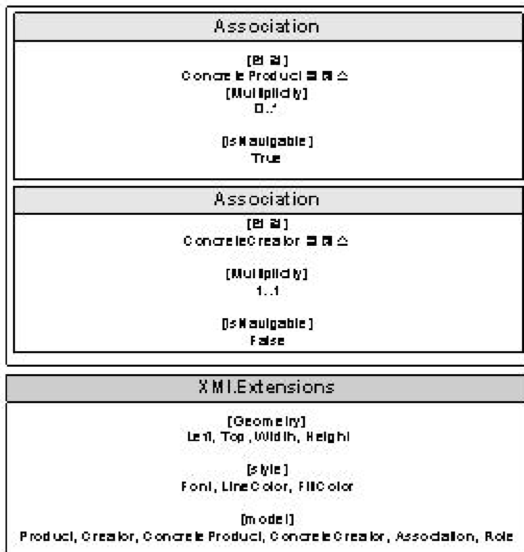

그림 7. Factory Method 패턴의 관계 부분

반면에 ConcreteProduct 클래스와 ConcreteCreator 클래스 사이에는 Associations 요소를 별도로 두어 두 객체간에 Association 관계가 있음을 나타낸다. 이와 같이 디자인패턴을 표현하기 위해 UML 설계 정보를 메타데이터로 표현할 수 있다.그리고 Factory Method 패턴을 컴포넌트화 하고 패턴간의 합성을 위해서는 인터페이스를 정의해야 한다. 따라서 [그림 8]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 8. Factory Method패턴의 인터페이스

인터페이스 정보 내에는 Creator와 ConcreteCreator 클래스에 대한 인터페이스를 갖고 있다. 그리고 Creator 클래스는 하나의 속성(attribute)와 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.

Ⅲ. 메타모델을 이용한 디자인패턴의 합성

디자인패턴의 구조를 정의하기 위해 XMI 메타모델을 이용하였으며, 디자인패턴을 합성하기 위해 패턴의 합성연산을 위한 연산자와 연산방법을 정의하였다. XMI.composite 모델은 [그림 9]에 표기한 바와 같이 메타데이터에 메타데이터를 구성하는 원소(element)를 추가할 수 있도록 하였다. 기본 디자인패턴의 메타데이터 모델에서 패턴이 합성되면서 필요한 데이터를 XMI.composite와 XMI.replace 연산을 적용하여 합성

되는 디자인패턴의 메타데이터를 구성하도록 하였다. [그림 10]은 디자인패턴을 나타내는 메타데이터에서 메타데이터의 합성을 수행하기위한 과정을 설명한 것이다. 합성하려는 패턴을 선택하고, 2번 라인에 표기된 인터페이스(interface)에서 합성하려는 디자인패턴의 IDL 메시지 교환정보를 확인한다.

```
XMI.Composite Meta Model
<!ELEMENT XMI.composite ANY>
<!ATTLIST XMI.composite
                    %XMI.element.att;
                    %XMI.link.att;
                    xmi.position CDATA "-1">
XMI.replace Meta Model
<!ELEMENT XMI.replace ANY>
<!ATTLIST XMI.replace
                    %XMI.element.att;
                    %XMI.link.att;
                    xmi.position CDATA "-1">
```

그림 9. 합성을 위한 메타데이터 연산자

그리고 합성하려는 패턴의 ID, 요소의 ID 획득 하여, 합성연산으로 패턴ID와 요소ID를 추가하고 content 영역에 composite 연산으로 부여받은 메타데이터를 추가한다. 이때 합성되는 패턴 명, ID의 메타데이터 추가되며, 패턴의 요소에 대한 메타데이터 추가되어 새로운 패턴 명과 ID 부여받는다. XMI.composite 연산에 의해 패턴을 합성하려면 <XMI.composite href="패턴">을 이용하여 합성하려는 패턴을 선택하고, XMI.composite 연산에서 합성하려는 패턴과 관계정보를 얻기 위해 xmi.id 값에 의해 각 패턴을 구분하고, 패턴내의 요소(elements)를 구분 한다.

```
1: select pattern for composition;
2: if (interface) {
3:    get patterns-ID, elements-ID;
4:    composite metadata of pattern-ID, elements-ID;
5:    composite metadata of pattern-name in content area;
6:    composite metadata of pattern-element in content area;
7:    create new metadata of pattern-name, pattern-ID;
8: }
9: else
10:    can't composite this pattern;
```

그림 10. 패턴의 메타데이터 합성 과정

```
XMI.composite

<XMI.content>
  <XMI.difference href="original">
    [repeat pattern]
      <XMI.composite href="original/pattern-ID>
        <XMI.composite xmi.id=
                    "#ID#" xmi.name="#PATTERN_NAME#"/>
      <XMI.composite href="original/association-ID>
        <XMI.composite xmi.idref=
                         "#ID#" xmi.name="#ASSOC_NAME#"/>
      <XMI.composite href="original/Class-ID>
       <XMI.composite xmi.id="#ID#"
                              xmi.name=#CLASS_NAME#"/>
    [/repeat pattern]
    <XMI.replace href="original/Model/>
     <Model xmi.id=
         "#New_ID#" xmi.name="#New_pattern_Name#"/>
  </XMI.difference>
</XMI.content>

   // 패턴 정보를 기술하는 세그먼트
     <PATTERN xmi.id="#ID#">// 패턴 ID
       <xmi.name>#PATTERN_NAME#</xmi.name>// 패턴 명
     </PATTERN>
    // 관계정보를 기술하는 세그먼트
     <Association xmi.idref="#ID#">// 관계 ID
       <xmi.name>#ASSOC_NAME#</xmi.name>// 관계명
       <connection>// 관계에 사용된 클래스
         <feature>
          #class#
         </feature>
       </connection>
     </Association>
     // 클래스 정보를 기술하는 세그먼트
     <Class xmi.id="#ID#">// 클래스 ID
       <xmi.name>#CLASS_NAME#</xmi.name>// 클래스명
     </Class>
```

그림 11. 메타데이터의 합성방법

[그림 11]과 같이 XMI.composite 연산을 반복하며 합성하려는 패턴의 요소(elements)를 메타모델을 이용하여 합성한다. 합성하려는 pattern-ID와 association-ID정보를 얻은 후, 패턴의 XMI 메타모델에 합성한 메타모델 정보를 합성하여 확장된 패턴의 메타데이터를 구성하게 된다. 이때 <XMI.repalce> 연산에 의해 새로운 패턴 명을 부가하게 되며 요소(elements)의 이름을 변경한다. 메타모델 내에 추가되는 패턴은 content 영역에 기존에 부여받은 패턴의 이름과 ID를 등록하며, 이때 추가되는 패턴과의 관계정보에 ID를 부여한다.

Ⅳ. 평가

본 논문에서는 그래픽 디자인 도구, 클래스 설계도구, 게임엔진 설계 도구, 모바일 게임, 모바일 스크립팅 도구, 대여점 관리 프로그램을 대상으로 각각 두 가지 설계 정보를 평가하였다. 첫 번째, 클래스 수준의 설계는 과거의 설계 정보 재사용 없이 클래스와 클래스간의 관계를 직접적으로 표현하여 설계하였으며, 두 번째, 메타데이터로 표현된 과거의 설계 정보를 디자인 패턴으로 저장해두고 디자인 패턴을 합성해 가며 설계하였다. 그리고 클래스 수준에서 직접 설계한 경우와 메타데이터를 이용하여 디자인 패턴을 합성하여 설계한 경우 설계 정보의 응집력이나 복잡도가 어떻게 달라지는가를 여섯 개의 시스템을 대상으로 평가하였다.

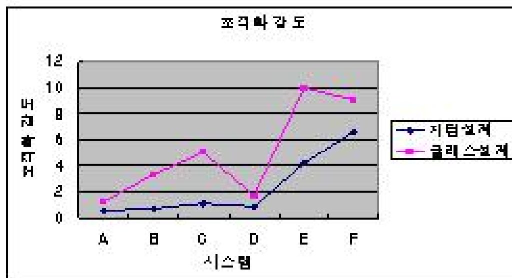

그림 12. 시스템 조직력 비율

설계 정보의 응집도의 경우 역시 설계상의 문제점을 해결하는 디자인패턴을 적용한 설계하였을 때 적용 대상 시스템 모두 디자인 패턴이 적용되어 설계되었을 때 설계 정보의 응집력이 A시스템은 0.56에서 0.35, B시스템은 0.35에서 0.21, C시스템은 0.84에서 0.22, D 시스템은 0.66에서 0.43, E시스템은 0.85에서 0.42, F 시스템은 0.83에서 0.32로 응집도가 더 강하게 나타나는 것을 알 수 있었다. 그리고 디자인패턴을 적용한 설계 방법이 A시스템에서는 4, B 시스템에서는 3, C시스템에서는 4, D시스템에서는 2, E 시스템에서는 6 그리고 F 시스템에서는 4씩 복잡도가 상대적으로 감소됨을 알 수 있었다.

따라서 시스템의 조직력(Organization Ability)은 그 시스템의 복잡한 정도에 비해 시스템이 강한 응집력을 갖는가를 알 수 있으므로 여섯 개의 시스템을 대상으로 측정한 것이다. 복잡도가 높은 시스템을 설계하게 될 때 각 시스템은 복잡도가 증가하더라도 강한 응집력을 갖는 디자인패턴을 적용함으로써 시스템을 설계할 때 조직력이 높아진다는 것을 [그림 12]와 같은 관계를 보면 알 수 있다. 따라서 시스템 설계시 설계 목적에 맞는 최적의 클래스의 수 선택과 관계를 설정하기 위해 과거의 설계 경험을 최적으로 추상화한 디자인패턴을 적용함으로써 설계 정보의 응집도를 강하게 만들 수 있다.

Ⅴ. 결 론

설계 단계에서는 반복적으로 설계되고 있는 설계 정보를 정형화된 설계 정보로 표준화하지 못하고 있으며, 정형화된 표현에 의해 설계 정보를 합성해 가며 시스템을 설계할 수 있는 환경이 구축되어 있지 못하였다. 따라서 본 논문에서는 설계상의 문제해결을 정형화한 디자인패턴을 이용하여 컴포넌트를 XMI 메타모델로 정형화하고, 표준화된 UML을 이용하여 설계한 디자인패턴을 서로 다른 설계자들이 공유하고, XMI 메타데이터에 의해 기존의 설계 정보를 재사용할 수 있는 XMI 기반의 디자인패턴의 합성을 위한 메타데이터와 인터페이스를 설계하여 이를 지원하는 환경을 구축하였다. 패턴지향 설계를 목적으로 디자인패턴은 XMI 메타모델로 정형화하였고, 정형화된 패턴을 중심으로 패턴 정보저장소를 구성할 수 있다. 그리고 메타데이터로 정형화된 디자인패턴을 이용하여 패턴을 합성해 가며 디자인패턴을 기반으로 시스템을 설계하여 확장 할 수 있다. 또한 디자인패턴을 재사용 하여 설계하였을 경우 설계자에 의해 클래스와 관계가 선택되어 설계 되었을 때 보다 얻게 되는 장점을 평가하기 위하여 클래스를 직접 설계했을 때 보다 디자인패턴을 재사용하여 설계했을 때 시스템의 복잡도가 감소하고 응집력이 강해져 시스템 전체의 조직력을 향상 시켰다. 따라서 디자인패턴의 사용한 설계와 분석 과정은 객체 도입에 있어 패턴이라는 개념에 근거한 클래스 역할의 지정과 상호관계 설정은

물론 클래스들이 무엇을 담당하고, 어떤 순서로 상호작용이 발생하는지를 명백하게 밝혀준다.

참 고 문 헌

[1] J. Cheng and j. Xu, *XML and DB2, In Sixteenth International Conference on Data Engineering (ICDE'00)*, IEEE Computer Society Press, pp.569-576, 2000.

[2] 송영재, 김귀정, 변정우, 서영준, 최한용, 한정수, *소프트웨어공학*, 이한출판사, 2004.

[3] L. C. Briand, Y. Labiche, M. D. Penta, and H. Y. Bondoc, "An experimental investigation of formality in UML-based development," IEEE Transactions on Software Engineering, pp.833-849, 2005.

[4] M. Klettke and H. Meyer, "XML and Object-Relational Data-base Systems- Enhancing Structural Mappings Based on Statistics. In Suciu," Proceedings of the Third International Workshop on the Web and Databases, pp.63-68, 2005.

[5] R. K. Kellera and R. Schauer, "Design Components: Towards Software Composition at the Design Level," ICSE, pp.282-291, 2003

[6] http://www.omnibuilder.com

[7] http://www.modelmakertools.com

[8] http://www.code farms.com

[9] T. D. Han, S. Purao, and V. C. Storey, "A Methodology for Building a Repository of Object-Oriented Design Fragments," ER pp.203-217, 2003.

[10] *Software Engineering : A practitioner's approach*, McGraw-Hill International Edition, 2005.

[11] http://www.xmlmodeling.com

[12] http://www.rational.com/news/inthenews/software-dev_012402.jsp

[13] J. Andrade, J. Ares, R. Garcia, J. Pazos, S. Rodriguez, and A. Silva, "A methodological framework for viewpoint-oriented conceptual modeling," IEEE Transactions on Software Engineering, pp.282-294, 2004.

저 자 소 개

최 한 용(Han-Yong Choi) 정회원

- 1994년 2월 : 경희대학교 전자계산공학과(공학사)
- 1998년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2002년 8월 : 경희대학교 전자계산공학과(공학박사)
- 2002년 9월~2004년 2월 : 경희대학교 전자정보학부 강의교수
- 2004년 3월~현재 : 한북대학교 컴퓨터공학과 교수

<관심분야> : Component Design, XMI, MDA

이 돈 양(Don-Yang Lee) 정회원

- 1987년 2월 : 대구대학교 통계학과(이학사)
- 1993년 2월 : 경희대학교 전자계산공학과(공학석사)
- 2004년 9월 : 경희대학교 전자계산공학과(공학박사)
- 2003년 3월~2006년 2월 : 경인여대 겸임교수
- 2006년 3월~현재 : 세종대학교 컴퓨터공학과 초빙교수

<관심분야> : Design Pattern, AOP, XMI, MDA