XML기반 디자인패턴클래스의 메타데이터 연구

이돈양*, 송영재*
*경희대학교 전자계산공학과
e-mail:dylee62@empal.com

A Study of Design Pattern Class's Metadata based XML

Don-Yang Lee*, Young-Jae Song*
*Dept of Computer Engineering, Kyung-Hee University

요 약

클래스정보에 대한 속성의 추출 및 분류에서 주로 추출된 클래스의 정보가 단지 원시코드의 코멘트에서 추출되었기 때문에 클래스에 대한 정확한 기능 및 용도에 대한 Document가 부족하여 실제로 이용자가 최적의 부분을 추출하기가 어려웠다. 이러한 것들을 향상시키기 위하여 본 연구에서는 객체에 대한 클래스뿐만 아니라 패턴모델의 설계에서도 객체지향모델링 방법을 이용하여 메타모델과 메타데이터를 설계하였다. 그리고 XMI 메타모델로 정의된 디자인패턴의 세부적인 클래스의 메타데이터의 생성에 중점을 두었으며, 마크업언어로 XML-스키마 형식을 이용하여 심플타입(simple type)과 콤플렉스타입(complex type)으로 분류하였다. 그 결과 메타데이터 엘리먼트 단위영역별로 마크업언어를 생성하여 소프트웨어 설계에서 효과적인 재사용을 할 수 있었다.

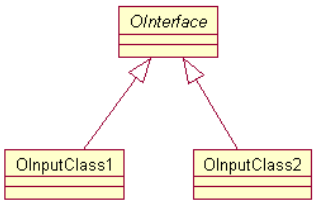
1. 서론

소프트웨어 설계와 관련하여 최근에는 객체지향모델링(Object-Oriented Modelling)[1]을 이용한 방법이 많이 사용되고 있으며, 이는 사용자들의 요구사항에 대한 관점에서 최신의 기술을 적용하고 있다. 특히, OMG의 UML은 객체지향모델링에 대해서 표준화된 언어에 지원이 가능하여 널리 사용되고 있다[2]. 이전에 주로 사용했던 클래스정보에 대한 속성의 추출 및 분류에서는 추출된 클래스의 기능정보가 단지 원시코드의 코멘트에서 추출되었기 때문에 클래스에 대한 정확한 기능 및 용도에 대한 Document가 부족하여 실제로 이용자가 최적의 부분을 추출하기가 어려웠다. 이러한 것들을 향상시키기 위하여 본 논문에서는 객체에 대한 클래스뿐만 아니라 패턴모델의 설계에서도 객체지향모델링 방법을 이용하여 메타모델과 메타데이터를 설계하였다. 그리고 패턴 및 클래스에 대한 부분을 정형화, 표준화하기 위한 방법으로 XML Metadata Interchange Format(XMI)를 이용하였다. 본 연구에서 특히 중점을 두고 있는 부분은 메타모델에서 패턴의 클래스를 메타데이터로 생성하는 부분이다. 일반적으로 XML에서 데이터를 이용한 어플리케이션 개발에서 적어도 두 가지의 기능에 대해서 언급할 수 있다. 하나는 마크업언어의 설계에 관한 것이고, 다른 하나는 적절한 클래스의 생성에 대한 것이다. XML 마크업언어로는 DTD[3]와 XML-스키마[4]를 많이 사용하고 있다. DTD는 현재 널리 사용되고 있으며 광범위한 툴(tools)의 지원을 받고 있다. 그러나 XML-스키마가 트리 구조의 문법을 사용하여 Document와 광범위한 데이터 타입을 표현하는 방법을 가지고 있기 때문에 본 연구에서는 이것을 택하여 설계하였다. 본 연구는 XMI 메타모델로 정의된 디자인패턴의 세부적인 클래스의 메타데이터에 대한 생성방법에 초점을 두었다. 그리고 XML-스키마에 대한 형식을 심플타입(simple type)과 콤플렉스타입(complex type)으로 분류하여 마크업언어를 설계하였으며, 메타데이터 엘리먼트 단위영역별로 마크업언어를 생성 하여 소프트웨어 설계에서 재사용을 최대한 할 수 있도록 하였다.

2. 관련연구

2.1 디자인 패턴

디자인 패턴은 객체지향 소프트웨어를 설계하는데 있어서 반복적으로 발생하는 설계상의 문제들을 패턴으로 등록하여, 설계자들이 어떠한 설계 문제에 대해서 다시 해결책을 모색하는 과정을 거치지 않고, 바로 디자인 패턴을 선택하여 설계에 적용할 수 있도록 한 것이다[5]. Simple Abstract Factory 패턴은 concrete로 구체화되는 subclass 인스턴스에 대한 팩토리 메소드를 정의 하는 추상 클래스를 가지고 있다[6].



<그림1> Simple Abstract Factory 패턴

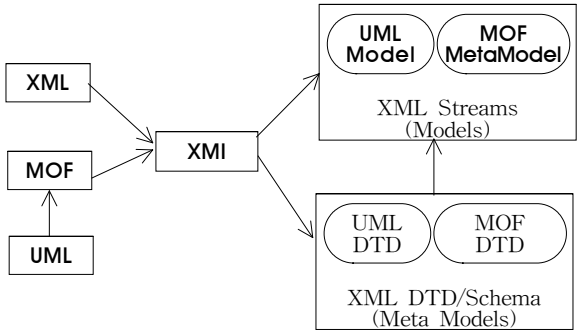
2.2 XML

World Wide Web Consortium(W3C) 는 XML을 SGML의 일부분으로 정의하고 있다. 이는 HTML과 같이 웹에서 정보를 제공하거나, 받거나, 처리가능하도록 하고있다. XML은 SGML과 HTML의 서로 상호간의 작용과 구현의 용이함을 위해 사용되고 있다. SGML[ISO 8879]는 1986년 이후 국제적인표준으로 사용되고 있으며, 언어를 마크업 구조화하는 방법을 명세하는 메타언어로 사용되고 있다. XML은 SGML의 가장 간단한 부분집합이다[7].

2.3 XMI MetaModel

XMI는 프로그래머들과 다른 사용자들이 메타데이터에 관해 정보를 교환하기 위한 표준 방식으로, XML을 사용할 것을 제안한 것이다. 명확히 말하면, XMI는 자신들의 데이터 모델들을 서로 교환하기 위한 갖가지 언어들이나 개발도구들과 함께 UML을 사용하여 프로그래머들을 돕기 위해 의도된 것이다[8]. 그밖에, XMI는 데이터웨어하우스에 관한 정보를 교환하기 위해 사용될 수도 있다. 실제로, XMI 형식은 어떤 메타데이터 셋(set)이 어떻게 묘사되어야 하는지, 그리고 여러 종류의 산업계나 운영환경에 있는 사용자들이 같은 방식으로 데이터를 보기 위해 필요한 것이 무엇인지 등을 표준화한다. XMI는 이러한 산업계 표준이나 권고안을 만들고 확장하는 기구인 OMG로부터 나온 제안이다.

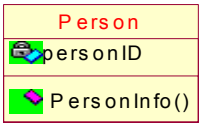
◇ XML : W3C의 표준
◇ UML : OMG로부터의 표준
◇ MOF (Meta Object Facility) : 메타모델링과 메타데이터 저장



<그림2> XMI 와 XMI DTD/Schema

3. XML 기반 클래스 정의

본 연구에서는 UML에서 정의하고 있는 클래스에 대한 표기방법을 적용하기 위해 <그림3>을 예제로 들어 메타데이터를 정의하였다. 여기서 Person은 클래스의 이름, personID은 어트리뷰트 그리고 PersonInfo()은 오퍼레이션으로 정의하였다. 일반적으로 클래스는 Concrete 클래스와 Abstract 클래스로 분류할 수 가 있다. 클래스를 XML 메타데이터로 표시하는 엘리먼트에서는 <isAbstract> true </isAbstract>로 표시하여 Abstract 클래스임의 나타내고, Concrete 클래스인 경우에는 <isAbstract> false </isAbstract>로 표시한다[9].



<그림3> 단위 클래스

<그림3>의 클래스 형식에 대해서 본 연구에서는 다음과 같은 XML 데이터로 정의하여 메타데이터를 생성하였다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="classDiagram.xsl"?>
<uml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="classDiagram.xsd" >
  <classDiagram>
    <class classID="person">
      <name>Person</name>
      <isAbstract>false</isAbstract>
      <attribute>
        <name>personID</name>
        <type>CString </type>
        <visibility>private</visibility>
      </attribute>
      <operation>
```

```
        <name>PersonInfo</name>
        <visibility>public</visibility>
      </operation>
    </class>
  </classDiagram>
</uml>
```

<그림4> "<그림3>"의 메타데이터

<그림1>의 Simple Abstract Factory 패턴을 <그림5>와 같이 패턴의 메타모델로 표현하는데 있어서 UML의 namespace는 XMI 메타모델 태그처럼 XMI 엘리먼트에서 이름의 선언으로 사용되고 있다. 그리고 XMI 메타모델로 클래스 패턴을 표기한 주요 목적으로는 패턴에서 각 클래스의 구성에 대해 표준화되고 정형화된 모델을 설계하고자 한 것이며, 아울러 추후 공통의 클래스를 분석하고 추출하기 위해 개괄적인 분석의 요소로 적용하도록 하기 위함이다.

```
File="CommonClass.xml" Namespace="CommonClass":
<XMI version="1.1" XMLNS:uml="org.omg/UML1.3">
  <XMI.header>
    <XMI.model xmi.name="Common" href="CommonClass.xml"/>
    <XMI.metamodel xmi.name="UML" href="UML.xml"/>
  </XMI.header>
  <XMI.content>
    <UML:Class name="OInterface" xmi.id="OInterface" />
    <UML:Class name="OInputClass1" xmi.id="OInputClass1"
generalization="OInterface" />
    <UML:Class name="OInputClass2" xmi.id="OInputClass2"
generalization="OInterface" />
  </XMI.content>
</XMI>
```

<그림5> Simple Abstract Factory 패턴 메타모델

<그림5> Simple Abstract Factory 패턴 메타모델은 XMI 메타모델의 형식에 따라 생성한 것이다. 여기에 적용된 세 개의 클래스 중 SuperClass에 해당되는 OInterface와 SubClass에 해당하는 두 개의 OInputClass1과 OInputClass2를 생성하였다.

4. XML-스키마 기반 클래스의 마크업언어 생성

클래스의 XML 메타데이터에 대한 XML-스키마 언어 정의에서 사용되는 컴포넌트의 형식 중 본 논문에서는 심플타입(simple type)과 콤플렉스타입(complex type)으로 분류하였다.

4.1 XML-스키마 컴포넌트(Components)

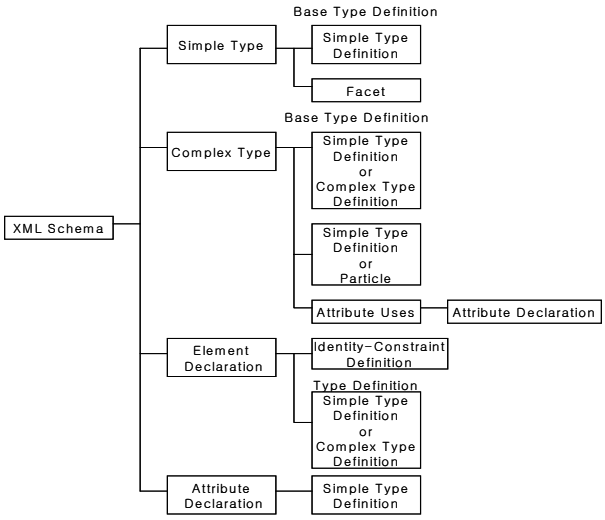
XML-스키마의 컴포넌트 구조는 <그림6>과 같이 정의할 수 있으며 XML-스키마에서 사용되고 있는 중요한 컴포넌트는 다음과 같이 분류할 수 있다.

◇ Element : 타입(type)에 대한 관계와 엘리먼트를 선언

◇ Attribute : 값을 가지고 있는 데이터타입의 집합과 어트리뷰트의 이름을 선언

◇ Simple Type : 텍스트 모드(text mode)와 어트리뷰트 값에 대한 데이터 타입 정의. 패싯(facet)을 사용함.

◇ Complex Type : 어트리뷰트에서 와 같이 연속적인 자식(child) 엘리먼트의 집합을 사용할 수 있도록 명세모델의 사용에 대한 엘리먼트 타입을 정의



<그림6> XML-스키마의 컴포넌트 구조

4.2 XML-스키마 문서의 엘리먼트(Element) 및 데이터 타입선언

XML 문서에서는 엘리먼트의 내용으로 자식을 가지지 않고 데이터 만 가질 수 있는 형식과 자식 엘리먼트를 갖는 엘리먼트 형식, 자식 엘리먼트와 속성을 동시에 갖는 엘리먼트 형식, 속성만 갖는 빈(empty) 엘리먼트 형식, 데이터 및 속성을 동시에 갖는 엘리먼트 형식 등으로 선언할 수 있으나 여기서는 <그림7>과 같이 데이터를 갖는 엘리먼트 선언과 자식 엘리먼트를 갖는 엘리먼트 선언을 설계하였다. 엘리먼트의 name 속성은 XML 문서에 나타나는 엘리먼트 이름을 지정하며 XML 권고안의 작성 규칙을 따라야 한다.

```
<element  name = "element_name"
          minOccurs="최소횟수"      maxOccurs="최대횟수"
type="data type"/>
  <complexType>
    <sequence>
      자식 엘리먼트 선언
    </sequence>
  </complexType>
</element>
```

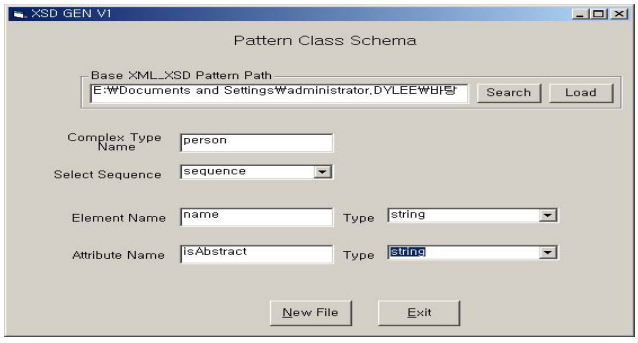
<그림7> 자식 엘리먼트를 갖는 엘리먼트 선언

XML-스키마에서는 다양한 데이터타입 및 속성을

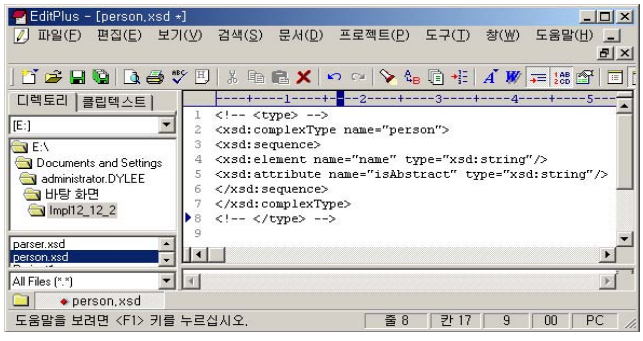
정의할 수 있다. XML-스키마에서 데이터타입은 이미 정의가 되어 사용되고 있는 내장형 심플타입(built-in simple type)과 사용자가 정의하여 사용하는 사용자 심플타입(user define simple type) 그리고 사용자 정의 콤플렉스 타입(user define complex type)으로 나눌 수 있다. 본 연구에 적용한 콤플렉스 타입은 속성을 가지거나 자식 엘리먼트를 가지는 엘리먼트의 선언에 필요한 타입으로써 이 역시 글로벌 콤플렉스 타입과 로컬 콤플렉스 타입이 있다. 그리고 <sequence>를 사용한 자식 엘리먼트의 사용에서는 "순차적 자식 엘리먼트 콤플렉스 타입"과 "선택적 자식 엘리먼트 콤플렉스 타입" 정의를 사용할 수 있다.

```
<complexType name="complex type name">
   <sequence>
     Element ...
   </sequence>
</complexType>
```

<그림8> 순차적 자식 엘리먼트 콤플렉스 타입



<그림9> 순차적 자식 엘리먼트 콤플렉스 타입 마크업언어 생성 인터페이스



<그림10> 순차적 자식 엘리먼트 콤플렉스 타입 마크업언어

<그림9>는 <그림4>의 Class에 대한 마크업언어 생성 인터페이스이다. <그림10>은 <그림9>에 의해 생성된 XML-스키마 마크업언어이다. 그리고 같은 방법으로 어트리뷰트와 오퍼레이션에 대한 마크업언어도 생성할 수 있다.

5. 결론

최근에는 메타데이터를 이용한 문서의 표현이 많이 이용되고 있다. 본 논문에서는 XMI 메타모델로 정의된 디자인패턴의 세부적인 클래스의 메타데이터 생성 중점을 두고 연구하였다. 그리고 XML 메타데이터의 마크업언어를 생성하기위해 다양한 데이터의 형식을 지원가능 한 XML-스키마 기반의 인터페이스를 설계하고 구현하였다. 그 결과로 엘리먼트 단위의 마크업언어의 생성이 가능하여 소프트웨어의 재사용에 수월성을 제고할 수 있었으며, Simple Abstract Factory 패턴의 사례연구를 통하여 단위 클래스의 재사용으로 파일의 크기는 47%, 코드라인 수는 53%의 감소를 가질 수 있었다. 향후 연구로는 XML-스키마에서 마크업언어를 생성하는 방법이 데이터타입에 따라 매우 다양하고 복잡하여 장기적인 연구가 요구된다.

참고문헌

[1] Christian Heide Damm, Klaus Marius Hansen, Michael Thomsen, Michael Tyrsted, "Creative Object Oriented Modelling", www.ideogramic.com /download/resources/ecoop2000.pdf, 2000.

[2] "OMG Unified Modeling Language Specification (draft)" Version 1.3. beta R7, June 1999.

[3] Lee, D., Chu, W, "Constraint-Preserving Inlining algorithm for mapping XML DTD to relational schema", Data and Knowledge Engineering 39, pp. 3-25, 2001.

[4] David C. Fallside, editor. XML schema Part 0: Primer. World Wide Web Consortium, 2000.

[6]Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[5] Dirk Riehle, "Composite Design patterns", OOPSLA '97, 11.218-228, 1997.

[7] XML in 10 points. W3C, 2001. http://www.w3.org/XML/1999/XML-in-10-points.

[8] "OMG-XML Metadata Interchange(XMI)" v1.2 specification. OMG, 2002.

[9] Justin elsberry and Nicholas Elsberry, "Using XML and SVG to Generate Dynamic UML Diagrams", http://www.cwu.edu/~gellenbe/docs/xmltechnicalreport.html, March 2003