코드 자동 생성을 위한 XML 기반의 효율적인 디자인패턴 구조

김 운 용[†] · 김 영 철[††] · 주 복 규[†††] · 최 영 근[††††]

요 약

디자인패턴은 어플리케이션 개발 시에 고려된 문제들로부터 독립적이며 확장성과 유지보수성에 대한 문제를 해결하기 위한 디자인 지식이며 현재 광범위하게 이용되고있는 분야이다. 그러나 이러한 광범위한 디자인패턴의 활용에도 불구하고 디자인 패턴에 명세와 활용은 주로 개발자의 수작업에 의존하고 있다. 그 결과 일관된 형태의 분석과 활용이 어렵고 오류 발생 빈도를 높일 뿐 아니라 프로그램 개발에 많은 시간을 필요로 한다. 따라서 이러한 문제를 해결하기 위해 본 논문에서는 디자인 패턴을 구조화하기 위해 XML을 사용한 표현방법, 이를 이용한 디자인 패턴 활용시스템 구조를 제시한다. 또한 이러한 표기법과 활용구조를 통해 소스코드 자동생성 지원 시스템을 제시하고 적용 예를 보이고자 한다. XML을 이용한 구조화된 문서활용은 소스코드 생성시 사용자들에게 더 작은 코드를 작성하게 만들고, 더 안정된 시스템을 구축할 수 있게 한다.

An Efficient Design Pattern Framework for Automatic Code Generation based on XML

Woon-Yong Kim[†] · Young-Chul Kim[††] · Bok-Gyu Joo[†††] · Young-Keun Choi[††]

ABSTRACT

Design Patterns are design knowledge for solving issues related to extensibility and maintainability which are independent from problems concerned by application, but despite vast interest in design pattern, the specification and application of patterns is generally assumed to rely on manual implementation. As a result, we need to spend a lot of time to develop software program not only because of being difficult to analyze and apply to a consistent pattern, but also because of happening the frequent programing faults. In this paper, we propose a notation using XML for describing design pattern and a framework using design pattern. We will also suggest a source code generation support system, and show a example of the application through this notation and the application framework. We may construct more stable system and be generated a compact source code to a user based on the application of structured documentations with XML.

키워드 : 디자인패턴(Design patterns, XML, 소스 코드 자동화(automatic source code generation), 디자인 패턴 활용(application of design patterns), 구조화문서(structed document)

1. 서 론

디자인 패턴은 객체지향 소프트웨어를 개발하는데 필요한 전문지식을 활용하기 위한 시도이다[3]. 이러한 것은 소프트웨어 시스템을 구성하는 빌딩블록으로서 크나큰 역할을 담당할 뿐 아니라 소프트웨어 개발 설계에서 재사용을 증가하기 위한 수단으로 활용된다[4]. 그러나 이러한 광범위한 관심과 활용에도 불구하고, 기존의 패턴명세와 활용은 주로 개발자의 수작업으로 이루어지므로, 오류발생 빈도를 높이고, 프로그램 개발에 많은 어려움과 시간이 필요하다. 효율적인 패턴 적용방법을 위한 여러 연구들이 진행되어왔는데 대부분은 주로 메타언어 및 패턴언어를 이용해 패턴을 구성하고 이를 프로그램에 적용시키는 방법에 관한 것들로서 대부분 추상적이며 실제 프로그램에 적용시키기 위해서는 많은 부가적인 작업이 요구된다. 이러한 부분을 해결하기 위해 본 논문에서는 산업 표준화 문서형태인 XML을 이용해 디자인 패턴의 표현방법을 제시하고 이를 바탕으로 소스코트 생성 지원 시스템과 적용 예를 보여준다.

※ 본 연구는 2001년 광운대학교 연구비 지원에 의해 수행되었음.
† 준 회 원 : 광운대학교 대학원 컴퓨터과학과
†† 정 회 원 : 홍익대학교 전산S/W학과 교수
††† 정 회 원 : 홍익대학교 전자전기컴퓨터공학부 교수
†††† 정 회 원 : 광운대학교 컴퓨터과학과 교수
논문접수 : 2001년 10월 5일, 심사완료 : 2001년 11월 16일

XML을 이용해 구조화된 패턴 문서는 컴퓨터에서 이용 가능한 구조적 문서역할과 프로그램 소스코드 생성에 쉽게 적용시킬 수 있으며 또한 문서의 검증 및 다양한 XML기술에 활용될 수 있을 것이다.

본 논문의 구성은 다음과 같다. 우선 2장에서 관련연구를 소개하고, 3장에서 XML을 이용한 디자인 패턴의 활용구조와 디자인 패턴에 대한 XML 문서 표기법을 제시한다. 4장에서는 이러한 표기법을 통한 XML 기반의 패턴문서 생성을 보이고, 5장에서는 생성된 XML 문서형태의 패턴을 이용한 소스코드 생성 예를 보인다. 마지막 6장에서는 결론을 내린다.

2. 관련 연구

기존의 디자인 패턴 명세의 활용에 대한 연구로 Partha는 각각의 협력자들에 책임을 부여함으로써, 패턴에 의해 나타나는 행위가 어떻게 정규화 형태로 변화될 수 있는지를 증명하였다[8]. Klarlund & Koistinen는 Parse Tree를 이용한 디자인 제약의 인증방법을 제시하고 있다[7]. 그러나 이러한 표현방법들은 구현에 대한 상세한 기술이나 접근이 어렵고, 각각 서로 다른 특수한 디자인 패턴 표현언어를 사용함으로써, 이해하거나 분석하기가 어렵다. 또한 문서로써의 활용능력을 담지 못한다. 또 다른 접근으로 디자이너에 의해 제공되는 정보로부터 자동으로 디자인 패턴에 대한 소스코드를 생성할 있는 도구 개발이다[1-3, 9]. 그러나 이러한 방법은 패턴을 이용한 소스코드의 생성에 활용되고는 있으나 패턴에 대한 정보가 도구에 의존해서 존재함으로 활용성이 떨어진다.

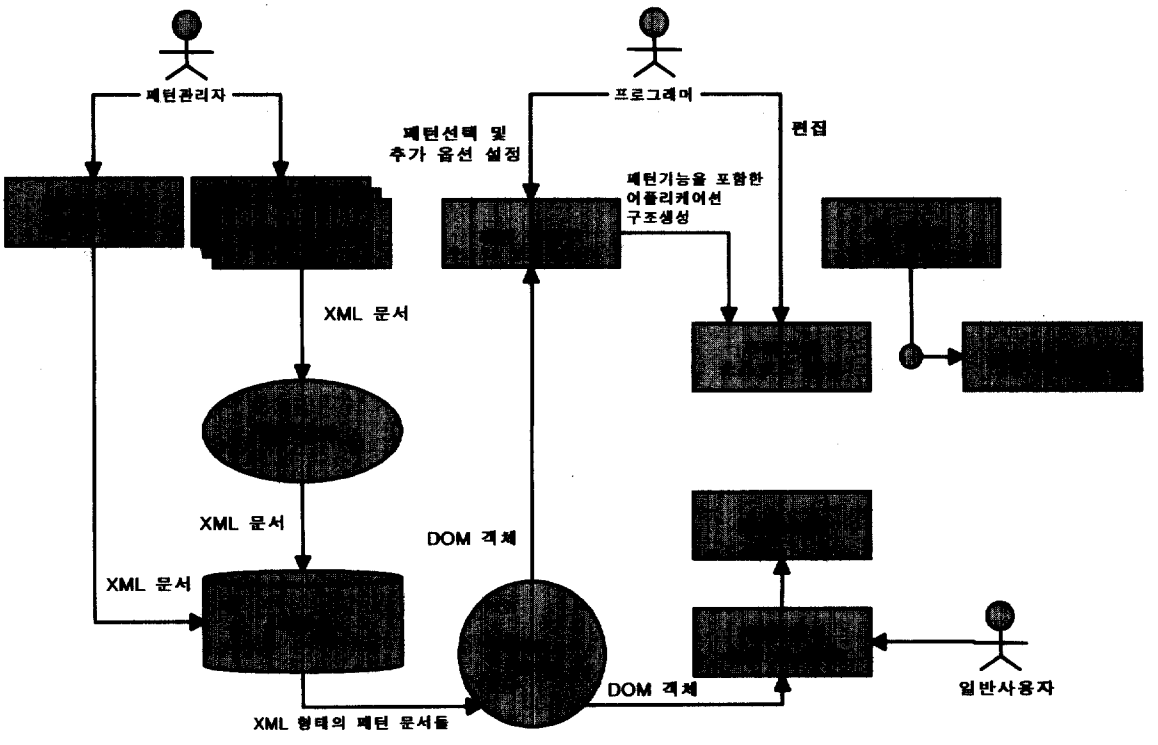
디자인 패턴을 프로그램에 효율적으로 활용하기 위해서는 작성된 문서가 쉽게 분석되고 적용될 수 있어야한다. 이러한 특징을 만족시키기 위해 본 논문에서는 산업표준화 문서인 XML을 기반으로 디자인패턴에 대한 구조화된 표기법을 제시하고 그 활용 예를 보인다.

3. 디자인 패턴에 대한 XML문서 표현

디자인 패턴의 표기방법은 자동화도구의 기본 틀을 형성하는 중요한 수단이며, 패턴 문서 작성 및 활용의 기본 요소가 된다. 본 장에서는 디자인 패턴을 소스코드에 활용하기 위해 디자인패턴의 활용구조를 제시하고, 이 구조에 이용되는 패턴문서의 XML 기반의 표기법을 제시한다.

3.1 XML을 이용한 디자인 패턴의 활용 구조

이 절에서는 XML형태의 디자인 패턴문서를 소스코드 자동생성에 활용하는 수행과정에 대한 전체구조를 제시한다. 디자인 패턴문서의 가장 중요한 역할중의 하나는, 사용자가 프로그램 개발 시 어느 정도 쉽게 적용할 수 있는가에 달려 있다. (그림 1)은 XML형태의 패턴에 대한 문서

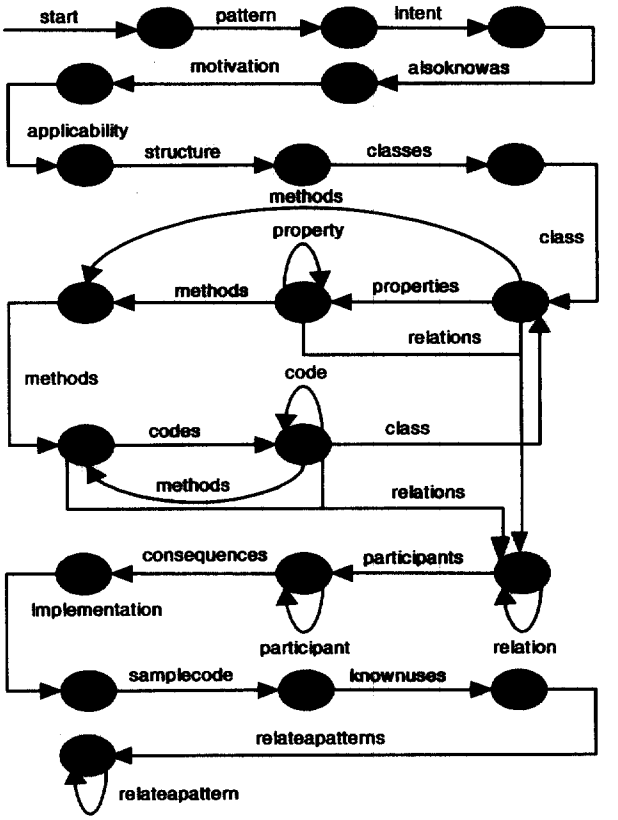


(그림 1) XML을 이용한 디자인 패턴 활용 구조

작성으로부터 그 활용방법까지에 대한 전체적인 구조를 나타낸다. 패턴문서의 작성 및 활용은 패턴관리자, 프로그래머 및 일반사용자로 구성될 수 있다. 그림에서 보듯이 작성된 패턴문서는 다양한 시각에서 활용될 수 있는 구조를 가진다.

3.2 디자인 패턴문서에 대한 XML문서 구조

디자인 패턴은 디자인 문제에 대한 추상화 형태의 해결방법이다. 디자인 패턴은 클래스의 일반적인 구조, 역할, 상호관계 그리고 그들 클래스사이의 책임에 대한 내용을 가진다. 아직까지 디자인패턴에 대한 단일화된 표기법을 제안되지 않고 있다. 현재 몇 가지의 표기방법이 제안되고 있다[4, 5, 10]. 이들 패턴의 표기방법은 패턴의 활용성을 증가시키기 위해 다양한 시각으로 분류하고 있다. 먼저 Gamma[4]의 패턴에 대한 표현은 소프트웨어 디자인 패턴을 생성(creational), 구조(structural) 그리고 행위(behavioral)패턴으로 분류하고있으며 현재 학회 및 패턴활용을 목적으로 가장 많이 활용되고 있는 패턴 표현방법이다. Buschmann[5]의 표현에서는 패턴을 구조(architectural) 패턴, 디자인(design)패턴, 그리고 코드(idioms)패턴으로 구분하고 있다. 구조(architectural) 패턴은 소프트웨어의 뼈대를 형성하는 패턴에 대한 부분을 정의하고 있고, 디자인(design)패턴은 각 컴포넌트를 형성하는 패턴을 정의하고, 코드(idioms)패턴은 프로그램 언어를 이용한 코드 명세에 대한 패턴을 정의하고 있다. 또한 Tichy[10]의 표현에서는 패턴의 범주를 더 다양하게 분류하고 있다. 즉 어떤 목적의 패턴들인가에 대해 Decoupling, Variant Management, State Handling, Control, Virtual Machines, Convenience Patterns, Compound Patterns, Concurrency, Distribution 과 같은 범주로 분류하고 있다. 이들 모든 패턴의 표현 방법은 주로 패턴을 좀더 효율적으로 분류하고 사용 가능하게 만들기 위한 시도이다. 그러나 각각의 패턴을 표현하고있는 내용은 비슷하다. 본 논문에서는 구조화된 문서로 디자인패턴을 표현하기 위해 현재 가장 많이 활용되는 패턴 표현인 Gamma[4]의 표기법을 따른다. 추가적으로 다른 패턴 표현법들을 모두 포함할 수 있는 XML구조의 소프트웨어 디자인 패턴에 대한 연구가 진행되고 있다. Gamma[4]에서 디자인 패턴을 표현하기 위해 13가지의 구성 요소를 가진다. Name, Intent, Also Know As, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses, Related Pattern. 이러한 요소들은 크게 패턴의 사용용도 및 특징을 설명하는 텍스트와 패턴의 구조와 행위를 나타내는 부분으로 나누어질 수 있다. 여기에서 텍스트부분은 XML문서로 표현시 각 요소를 같은 이름의 태그로 구성하여 표현하면 된다. 예로 Intent 요소는 <intent> 태그를 이용해 정보를 저장하여 관리할 수 있다. 또 다른 부분으로 패턴구조와 행위를 나타내는 부분으로 Structure, Collaborations 부분을 들 수 있다. 이들은 실제 소스코드 자동생성에 사용되는 부분이다. 그러므로 이 부분에 대한 정보를 이용해 쉽게 소스코드 생성에 적용시키기 위한 구조가 필요하다. 먼저 이 패턴구조와 행위 및 텍스트 부분을 XML문서 형태로 표현 규칙을 DFA(Deterministic Finite Automation)을 통해 제시한다(그림 2).



(그림 2) 디자인패턴을 표현하기 위한 DFA

이 DFA의 구성은 순차적으로 [4]에서 표현한 내용을 순서적으로 적용한 결과이다. 여기에서 실제 소스코드에 적용될 부분은 structure 시작되는 부분이다. 디자인패턴의 구조 및 행위에 대한 DFA는 크게 classes 부분과 relations부분으로 나누어진다. classes부분에서는 class들의 집합을 표현하고 있고, 이 클래스 또한 properties와 methods부분으로 구성된다. method부분에는 codes부분이 포함되어있는데 이것은 이 패턴이 동작하기 위해 필요한 기능을 설명하는 소스코드가 기술된다. relations부분에서는 패턴에서 이용되는 클래스의 관계를 설정한다. 이 관계는 크게 생성, 참조, 집합과 상속관계로 나타낼 수 있다. 이러한 DFA와 패턴을 구성하기 위해 필요한 속성들을 가지고 우리는 DTD(Document Type Definition)를 정의할 수 있다. 이러한 정의 형태는 (그림 3)에서 보여준다. 이 DTD에 따른 패턴문서는

패턴 활용에 필요한 구조화된 XML기반 패턴문서를 구성한다. 그로 인해 문서의 검증 및 다양한 형태의 활용이 가능하다.

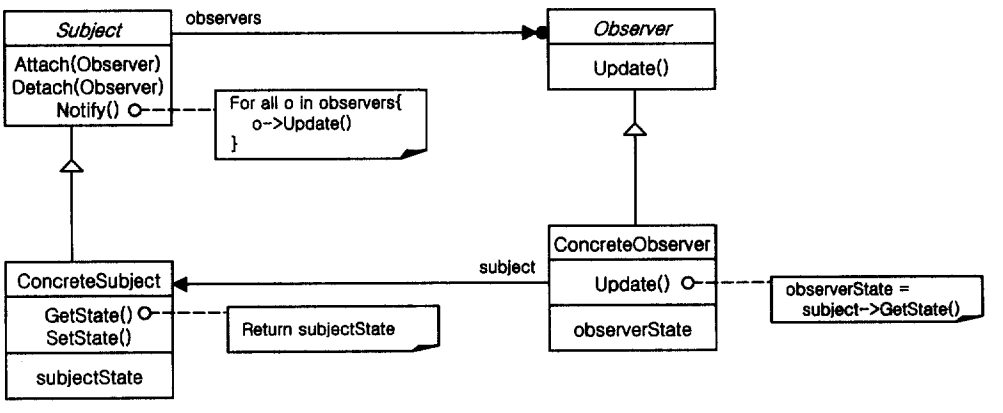
```
<!ELEMENT pattern (intent, alsoknowas, motivation, applicability,
          structure, classes, relations, participant, consequences,
          samplecode, knownuses, relatedpatterns)
<!ATTLIST pattern name CDATA #REQUIRED>
<!ELEMENT intent (#PCDATA)>
<!ELEMENT alsoknowas (#PCDATA)>
...
<!ELEMENT structure (classes,relations)>
<!ELEMENT classes (class+)>
<!ELEMENT class (properties,methods)>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST class scope (public | protected | private) "public">
<!ATTLIST class abstract (true|false) #REQUIRED>
<!ELEMENT properties (property+)>
<!ELEMENT property (#PCDATA)>
<!ATTLIST property scope (public | protected | private) "public">
<!ATTLIST property abstract (true|false) #REQUIRED>
<!ATTLIST property type CDATA #REQUIRED>
<!ELEMENT methods (method+)>
<!ELEMENT method (codes?)>
<!ATTLIST method name CDATA #REQUIRED>
<!ATTLIST method scope (public | protected | private) "public">
<!ATTLIST method abstract (true | false) #REQUIRED>
<!ATTLIST method arguments CDATA >
<!ATTLIST method return_type CDATA >
<!ELEMENT codes (code+)>
<!ELEMENT code (CDATA)>
<!ATTLIST code language CDATA #REQUIRED>
<!ELEMENT relations (relation+)>
<!ELEMENT relation EMPTY>
          <!ATTLIST relation type (create | reference | aggregate
          | inheritance) #REQUIRED>
<!ATTLIST relation from CDATA #REQUIRED>
<!ATTLIST relation to CDATA #REQUIRED>
          <!ATTLIST relation from_cnt (one | more_then_one |
          more_then_ zero) "one">
          <!ATTLIST relation to_cnt (one | more_then_one |
          more_then_zero) "one">
<!ELEMENT participants (participant+)>
<!ELEMENT participant (#PCDATA)>
<!ATTLIST participant name CDATA #REQUIRED>
...
<!ELEMENT relatedpatterns (relatedpattern*)>
<!ELEMENT relatedpattern EMPTY>
<!ATTLIST participant name CDATA #REQUIRED>
```

(그림 3) 디자인패턴을 표현하기 위한 DTD

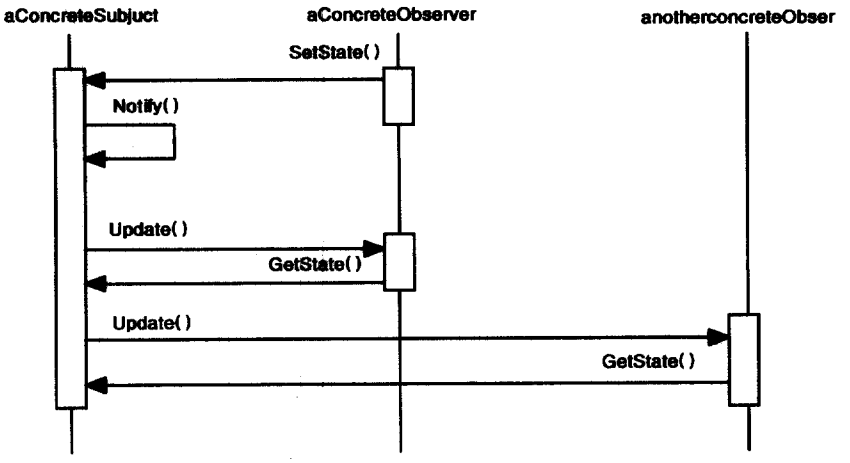
이 DTD는 (그림 2)의 DFA와 문서를 표현하기 위한 추가적 정보를 이용해 구성되고 그 요소는 DFA의 흐름에 따라 설계된다. 포함된 속성으로, 먼저 pattern요소는 패턴의 이름을 담고있는 name속성을 가진다. 또한 class 요소는 properties와 methods로 구성되고 abstract, name, scope와 같은 속성을 가진다. properties는 more then zero의 property로 구성되고, type과 scope의 속성을 가진다. method는 more then zero의 구성요소를 가지고 name, scope, abstract, argument, return_type의 속성을 가진다. 또한 method는 codes 구성요소를 포함하고있는데 이것은 패턴의 행위를 기술하기 위해 사용된다. 이 codes에 기술된 내용은 소스코드에 적용하기 위해 언어별 적용을 위해 code요소에 language속성을 가진다. relation요소는 create, reference, aggregate, inheritance의 관계를 표시할 수 있고 두 클래스간의 관계를 표현한다. 또한 from_cnt, to_cnt 속성들을 이용해 aggregate관계를 자세히 묘사할 수 있다. 4장에서는 이러한 DTD를 이용해 XML문서를 표현하는 예를 보인다.

4. XML기반 디자인패턴 문서

디자인 패턴 활용 적용 과정을 다루기 위해, 본 논문에서는 옵저버(Observer) 패턴[4]을 이용하여 제시된 DTD을 활용한 적용 예를 보인다. 옵저버 디자인 패턴[4]는 객체들 사이의 1대다의 의존성을 정의한다. 그 결과 하나의 객체가 상태를 변경할 때, 모든 의존관계를 가진 객체들에 알려지



(그림 4) 옵저버 패턴 클래스 다이어그램



(그림 5) 옵저버 패턴의 협동다이어그램

고 자동적으로 그 객체들이 업데이트된다. 이러한 디자인 패턴을 XML형태로 표현하기 위해 먼저 텍스트로 기록된 문서들에 대해서는 그 내용을 활용하여 구성할 수 있다. 예로 intent관련된 내용은 다음과 같이 기술된다.

```
<intent>
   Define a one-to many dependency …
</intent>
```

또한 소스코드생성과 관련된 Structure 와 Collaborations 부분은 제시된 패턴문서[4]의 클래스 다이어그램과 상호 협동다이어그램을 통해 얻을 수 있다. 옵저버 패턴과 관련된 클래스다이어그램과 협동다이어그램은 (그림 4)(그림5)에서 보여준다.

이러한 두 다이어그램을 이용해서 앞절에서 제시한 DTD에 정의된 규칙에 따라 <structure>에 해당되는 XML형태의 문서를 만들 수 있다. 이러한 옵저버 패턴에 대한 XML 문서는 (그림 6)과 같다.

```
<?xmlversion = "1.0"?>
<pattern name = "Observer">
 <intent>  Define a one-to-many dependency between objects
        … </intent>
 …
 <structure>
  <classes>
   <class name= "Subject" scope = "public"abstract = "true">
    <properties>
     <property scope = "public" type = "Vector">
      observers
     </property>
    </properties>
    <methods>
     <method name = "Attach" scope = "public" abstract = "true"
                  arguments = "Observers_obs">  </method>
     …
    <codes>
     <code language = "java">
        for (Enumeration e = observers.elements() ;
            e.hasMoreElements() ;){
            ((Observer)e.nextElement()).Update();
        }
     </code>
    …
  </classes>
  <relations>
   <relation type = "aggregate" from = "Observer" from_cnt =
                  "more_then_one"
                   to = "Subject" to_cnt = "one"> </relation>
   …
 </structure>
 <participants>
  <participant name = "Subject"> knows its observers.
     … </participant>
 …
 </participants>
 …
</pattern>
```

(그림 6) 옵저버 패턴에 대한 XML문서

5. XML문서형태의 디자인 패턴 활용 예

이 장에서는 XML문서형태의 디자인 패턴 문서를 활용하여 자동화된 소스코드 생성방법과 활용 구조에 대해 기술한다. (그림 1)의 XML을 이용한 디자인 패턴활용구조에서 보듯이 디자인 패턴을 작성하고 활용하는데 필요한 사용자 유형은 패턴관리자, 프로그래머, 일반사용자 층으로 구성된다. 그들 각 사용자에 대해 살펴보면 다음과 같다.

1) 패턴관리자

디자인패턴에 대한 스키마를 정의하여 이를 관리하는 역할을 한다. 또한 이 스키마 구조에 따라 패턴에 대한 XML 문서를 작성하고, 검증하며 문서에 대한 관리 기능을 담당한다.

2) 프로그래머

실제 응용 어플리케이션을 개발하는 층으로, 프로그램 개발 도구를 이용해 존재하는 패턴을 검색 또는 활용한다. 실제 프로그램 개발에 필요한 XML 문서를 프로그램의 일부로써 활용하는 계층이다. XML 문서는 개발도구의 위저드를 통해 사용자에게 필요한 소스코드를 자동 생성시켜 활용할 수 있게 한다.

3) 일반사용자

프로그램에 대한 기술 및 디자인패턴에 대한 활용 방법 등을 습득하는 사용자로서, XML 구조의 디자인패턴을 문서적으로 검색 및 활용하는 계층이다. 이러한 사용자는 웹브라우저와 같은 도구를 이용해 문서를 검색 및 활용할 수 있다.

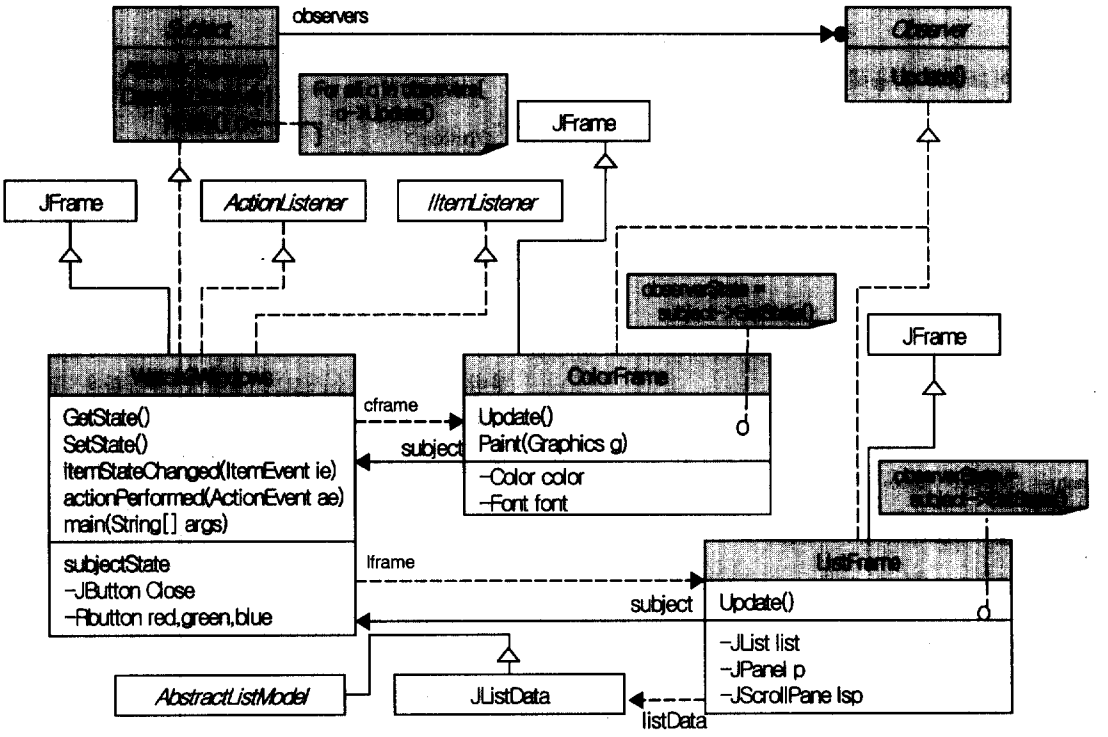
패턴관리자에 의해 구성된 패턴문서는 문서검증을 통해 저장소에 저장되어지고, 저장된 문서는 소스코드 자동생성에 활용되어진다. XML형태의 패턴 문서를 소스코드 자동생성에 활용되는 과정은 다음과 같다.

1) 사용자의 요구사항과 제약사항을 분석하여 필요한 디자인 패턴의 선택
2) 패턴 위저드를 이용해 사용하고자하는 패턴들을 선택하고, 프로그램 생성에 필요한 추가 요구사항 및 옵션 설정
3) 패턴 위저드는 요구된 패턴 정보를 이용해 XML Parser에게 XML형태의 패턴 문서 요청.
4) XML형태의 패턴 문서는 XML Parser에 의해 구조화된 형태의 DOM(Document Object Model) 객체를 제공.
5) 제공된 DOM객체와 2번에서 제공한 프로그램 생성에 필요한 정보를 이용해 사용하고자하는 패턴이 포함된 프로그램 소스코드 자동생성.
6) 프로그래머는 어플리케이션에 필요한 나머지 소스코드를 추가함으로써 어플리케이션 완성.
7) 완성된 소스코드 테스트 및 실행

5.1절에서는 이러한 과정을 통해 XML형태의 디자인패턴에 대한 문서를 어플리케이션 개발에 활용하기 위한 예를 보인다.

5.1 XML로 표현된 디자인 패턴을 이용한 어플리케이션 개발

어플리케이션 개발시 사용자가 디자인 패턴을 적용시키고자한다면, 그때 사용자는 패턴위저드를 통해 어플리케이션 구성에 필요한 정보입력을 통해 해당 어플리케이션에 원하는 패턴을 추가시킬 수 있다. 패턴을 어플리케이션에 적용시키기 위해 필요한 것은 어플리케이션에 필요한 클래스를 옵저버 패턴에 정의된 클래스와 연관시킴으로써 어플리케이션 클래스에 원하는 패턴기능을 추가시킬 수 있다. 이렇게 구성된 어플리케이션은 해당 디자인패턴의 구조와



(그림 7) 어플리케이션 적용 예

기능을 포함하여 구성되어진다.

(그림 7)은 옵저버 디자인패턴이 적용된 어플리케이션을 보여준다. XML문서화된 패턴문서의 정보를 이용해 소스코드 자동생성부분은 그림에서 진하게 표시된 클래스 클래스들이다. 나머지들은 사용자들에 의해 어플리케이션에 필요한 나머지 부분들을 나타낸다. 여기에서 XML형태의 패턴문서에 정의된 Subject는 Subject class로 Observer는 Observer class, ConcreateSubject는 Watch2Windows calss로 그리고 ConcreateObserver는 ColorFrame class과 ListFrame class에 적용되어 있는 것을 볼 수 있다. 즉 패턴문서에 표현된 내용은 어플리케이션에 원하는 클래스에 그 패턴 기능을 추가함으로써 사용자는 코드작성시간을 줄이고 어플리케이션의 구현에 더 많은 시간을 할당할 수 있다.

6. 결 론

본 논문에서는 디자인 패턴의 재활용 성을 높이기 위한 방법으로써 디자인패턴문서에 대한 XML 표현 방법과 이 XML 로 구성된 패턴 문서을 이용한 자동화 소스코드 생성 구조를 제시했다. 또한 이를 이용한 활용 예를 보였다. 우리 방법의 장점은 먼저 디자인 패턴을 구조화하고 일관된 형태의 구성을 제공함으로써 패턴에 대한 분석 및 유지 보수의 효율성을 증대시킬 수 있고 사용자들이 보다 쉽게 디자인 패턴을 어플리케이션 개발에 재활용 할 수 있도록 해주는 것이다. 또한 프로그램 개발의 효율성을 증가시킴으로서 비용 절감효과를 가져올 수 있다. 또한 이러한 XML의 다양한 기술에 접목함으로써 디자인 패턴 재활용을 극대화할 수 있다. 향후연구과제로 다양한 패턴 문서 표현법[4, 5, 10]을 포함할 수 있는 XML문서 표현방법에 대한 연구가 진행되고 있으며, 3.1절에서 제시한 디자인 패턴 활용방안에 대한 구체화 작업과 XML 기술의 응용을 통한 디자인 패턴활용에 대한 연구가 필요하다.

참 고 문 헌

[1] A. Eden, J. Gil and A. Yehudai. Automating the Application of design patterns, Report on Object Analysis and Design ROAD. pp.44-46, May, 1997.

[2] A. Eden, J. Gil and A. Yehudai. Precise specification and automatic application of design patterns. In Automated Software Engineering, 1997.

[3] C. Marcos, M. Compos, and A. Pirotte, Reifying Design Patterns as Metalevel Constructs, Electronic Journal of Sadio, 2(1) pp.17-19, 1999.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[5] F. Buschman, R. Meunier, H. Rohnert, P. Sommerlad, and Stal Michael. Pattern-Objected Software Architecture-A System of Patterns. John Wiley&Sons, 1996.

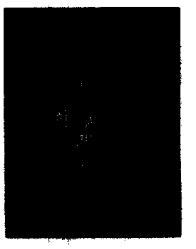
[6] J. Coplien and D. Schmidt, editors. Pattern Languages of Program Design Addison-Wesley, 1995.

[7] N. Klarlund and J. Koistinen, "Formal Design Constraints," Proceedings of OOPSLA. 1996.

[8] P. Pal, "Law-Governed Support for Realizing Design Patterns," Proceedings of TOOLS USA 17. 1995.

[9] S. Yacoub, Xue, H. and Ammar, H. H., Automating the development of pattern-oriented designs for application specific software systems, Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on, pp.163-170, 2000.

[10] W. Tichy, Essential Software Design Patterns. University of Karsruhe. http : //wwwipd.ira.uka.de/~tichy/patterns/overview.html, 1997.



김 운 용

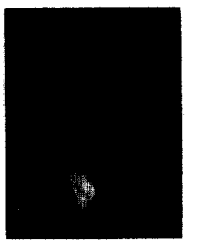
e-mail : wykim@cs.kwangwoon.ac.kr

1996년 독학사 전자계산학과(이학사)

1999년 광운대학교 정보과학기술대학원 (이학석사)

1999년~현재 광운대학교 컴퓨터과학과 박사과정

관심분야 : 객체지향프로그래밍 언어, 객체 모델링, 디자인패턴, 재사용기법, 컴포넌트 개발방법, 분산컴퓨팅기술



김 영 철

e-mail : wow@cic.hongik.ac.kr

1985년 홍익대학교(이학사)

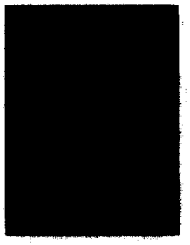
1987년 광운대학교(이학석사)

2000년 Illinois Institute of Technology (전산학박사)

2000년~2001년 LG 산전 중앙연구소 책임연구원

2001년~현재 홍익대학교 과학기술대 교수

관심분야 : Use Case 방법론 및 툴 개발, Design driven Testing, Maturity models, 데이터 모델링, Mobile agent



주 복 규

e-mail : bkjoo@wow.hongik.ac.kr

1977년 서울대학교 계산통계학과 졸업

1980년 한국과학원 전산학과 졸업 (이학석사)

1990년 University of Maryland 졸업 (공학박사)

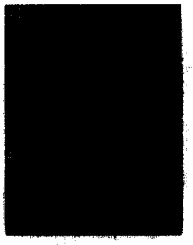
1990년~1998년 삼성종합기술원, 삼성전자 중앙연구소 수석 연구원

1998년~2000년 동양시스템즈 연구소장

2000년~2001년 한국과학기술원 전산학과 초빙교수

2001년~현재 홍익대학교 전자전기컴퓨터공학부 교수

관심분야 : Software Reuse, Software Testing, Network Security, Intelligent Systems



최 영 근

e-mail : ygchoi@cs.kwangwoon.ac.kr

1980년 서울대학교 수학교육과(이학사)

1982년 서울대학교 계산통계학과 (이학석사)

1989년 서울대학교 계산통계학과 (이학박사)

1992년~현재 광운대학교 컴퓨터과학과 교수

1997년~2000년 광운대학교 전산정보원 원장

2001년~현재 광운대학교 정보통신연구원장

관심분야 : 프로그래밍 언어, 병렬 프로그래밍언어, 객체지향 설계 및 분석, 분산 컴퓨팅기술, Mobile agent