디자인 패턴에 대한 소스코드 자동 생성 기법

김 운 용[†] · 최 영 근[††]

요 약

객체지향 프로그래밍의 주요 목적은 효율적인 재사용성과 개발시간의 단축 그리고 소프트웨어 품질의 향상에 있다. 이러한 목적을 달성하기 위한 하나의 방법으로 소프트웨어 개발 시 기존에 잘 정의되고 테스트된 설계정보를 이용하는 것이다. 이 잘 정의되고 테스트된 설계정보는 디자인 패턴이라 불리진다. 이 디자인 패턴은 소프트웨어 개발 시 특정상황에서 발생할 수 있는 문제에 대한 해결책을 표현하고 있다. 그러나 이 패턴은 추상적인 해결책을 제시하고 있기 때문에 어플리케이션 개발 시 적용되는 디자인 패턴에 대한 명세와 활용은 주로 개발자의 수작업에 의존하고 다양한 형태로 적용되고 있다. 그 결과 일관된 형태의 분석과 활용이 어렵고 오류발생 빈도를 높일 뿐 아니라 프로그램 개발에 많은 시간이 필요하다. 또한 적용된 패턴은 어플리케이션 내부에서 시각적으로 표현되지 않기 때문에 이 패턴에 대한 분석과 테스팅이 어렵다. 이에 본 논문에서는 어플리케이션 개발 시 디자인 패턴의 요소를 효율적으로 적용할 수 있기 위해 디자인 패턴에 대한 소스코드 자동 생성기법을 제시하고 어플리케이션 내부에 적용된 패턴들간의 분석 및 활용법을 보인다. 이를 통해 어플리케이션 내부의 디자인 패턴들은 일관된 형태의 구조와 효율성을 제공하고 이들간의 분석 및 활용 효과를 증대시킨다.

Automatic Source Code Generating Technique from Design Patterns

Woon-Yong Kim[†] · Young-Keun Choi[††]

ABSTRACT

A purpose of the object-oriented programming is to promote reuse and development time, and to improve software quality. A way for this purpose is using a design information well-defined and tested in previous time when developing software. Such design information is called design patterns. The design patterns are descriptions of abstract solution to recurse software design problems in a systematic and general way. But because the design patterns are descriptions of abstract solution, the specification and application of patterns generally rely on manual implementation and is applied to various forms. As a result, we need to spend a lot of time to develop software program not only because of difficulty in analyzing and applying to patterns consistently, but also because of the frequent programing faults. And because the applied design patterns don't express inside application visually, it is difficult to analyze and test for this design patterns. In this paper, we propose automatic source code generating technique to be able to efficiently apply the element of design patterns when developing application. And we show a way to analyze and use the applied design patterns in application. As a result, the design patterns in application provide the consistent structure and efficiency, and make analysis and using effect increased.

키워드 : 디자인패턴(design pattern), 패턴지향분석설계(pattern-oriented analysis and design), 소스코드자동화(automatic source code generation), 객체지향프로그래밍(object-oriented programming)

1. 서 론

객체지향 분석 및 설계를 통한 프로그래밍의 최대 목적은 효율적인 재사용성과 개발시간의 단축 그리고 소프트웨어 품질의 향상에 있다. 이러한 목적 때문에 소프트웨어 개발에서 설계는 더욱 중요시 되어왔다. 그 결과 효율적인 설계에 대한 연구가 다양한 분야에서 이루어지고 있으며 그 중 하나가 디자인 패턴이다. 이 디자인 패턴은 소프트웨어 개발 시 특정상황에서 발생할 수 있는 문제에 대한 해결책을 표현하는 것이다[5]. 이러한 패턴들에 대해 소개한 책들 중 가장 넓게 활용되는 것은 "*Design Patterns : Reusable Object-Oriented Software*"이다[5]. 이 책에서는 객체지향 시스템에서 유용하게 사용 가능한 23가지의 디자인 패턴을 소개하고 있다. 여기에 정의된 패턴들은 GoF패턴으로 불려진다. 크게 디자인 패턴의 활용에 대한 연구로 첫째 발견된 디자인 패턴을 효율적으로 분류하여 표현하기 위한 방법이다. 이들은 Gamma[5], Tichy[17], Buschman[7]등에 의해 이루어지고 있다. 둘째 어플리케이션에 대한 설계방법을 패턴지향으로 이끌어 내기

† 준 회 원 : 광운대학교 대학원 컴퓨터과학과
†† 정 회 원 : 광운대학교 컴퓨터과학과 교수
논문접수 : 2001년 11월 29일, 심사완료 : 2002년 7월 26일

위한 연구로 Sherif[16], Yacoub[12] 등에 의해 소개되었다. 또한 컴포넌트의 통합[9, 15]과 디자인 패턴정보를 이용한 어플리케이션 개발 지원 분야[1-3, 6, 12, 14] 등으로 연구가 진행되고 있다.

디자인 패턴은 그 내용이 어플리케이션에 적용될 때 구체화되어야 한다. 그러나 이러한 디자인 패턴은 문제 해결에 대한 추상적인 정보를 기술하고 있기 때문에 어플리케이션 내에 구체화될 때 개발자의 의도에 따라 다양한 형태로 변형되어지고, 또한 적용된 디자인 패턴은 적절하게 분석되거나 표현되기가 어렵다. 이러한 문제를 해결하기 위해 본 연구에서는 어플리케이션 개발 시 디자인 패턴의 요소를 효율적으로 구체화시키기 위해 소스코드에 디자인 패턴을 적용하는 기법을 제시하고 어플리케이션 내부에 적용되는 패턴들간의 구체화 정보를 분석하고 활용하기 위한 방법을 보인다. 본 논문은 객체지향 프로그램을 대상으로 디자인 패턴을 적용하는 자동화 기법을 제시하고 있다. 이를 위해 사용되는 디자인 패턴은 객체지향 프로그래밍을 위해 정의된 GoF패턴[5]를 이용한다. 또한 이 논문은 디자인 패턴 요소를 어플리케이션 구현시 자동적으로 적용시키기 위한 기법에 집중한다. 이를 통해 어플리케이션 내부에 구체화된 디자인 패턴정보는 일관된 형태로 유지되고, 이들간의 분석 및 활용의 효율성을 증가시킴으로써, 개발자들에게 보다 효율적인 패턴지향 개발환경을 제공할 수 있다.

본 논문의 구성은 다음과 같다. 제 2장 관련 연구에서는 어플리케이션 개발에 활용되는 디자인 패턴에 대한 기존의 연구를 소개하고, 제 3장에서 패턴지향 개발환경과 각 구성요소를 소개한다. 제 4장에서는 어플리케이션 개발에 필요한 디자인 패턴정보를 분석하고, 분석된 정보를 이용해 디자인 패턴 요소를 소스코드로 자동 생성하는 기법을 제시한다. 제 5장에서는 제시된 기법과 기존의 디자인 패턴 적용 기법에 대한 비교 분석을 보이고, 마지막으로 제 6장에서 결론을 내린다.

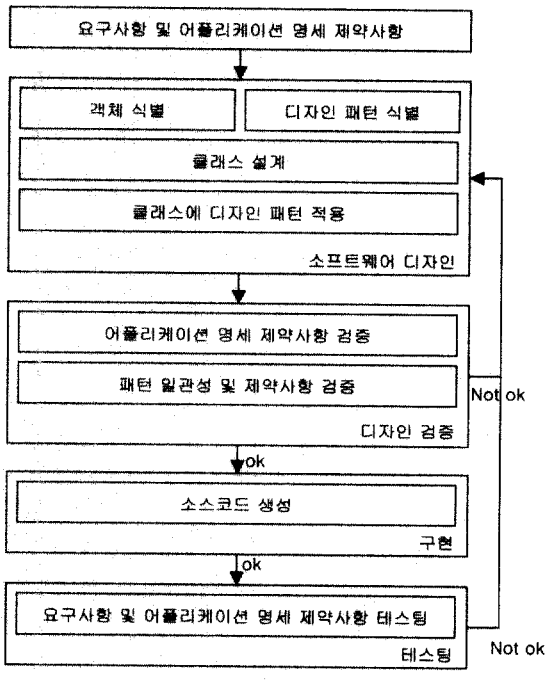
2. 관련 연구

2.1 디자인 패턴의 정의 및 범주

디자인 패턴은 디자인 문제에 대한 추상화 형태의 해결방법으로 소프트웨어 개발에 필요한 전문지식을 활용하기 위한 시도이다[4]. 이러한 것은 소프트웨어 시스템을 구성하는 빌딩블록으로서 크나큰 역할을 담당할 뿐 아니라 소프트웨어 설계 시에 재사용을 증가시키는 수단으로 활용된다[5]. 이러한 패턴들 중 본 연구의 대상이 되는 GoF패턴[5]은 객체지향 소프트웨어 개발에 필요한 패턴들을 정의하고 있다. 이 디자인 패턴의 내용은 패턴의 일반적인 구조, 역할, 상호관계 그리고 제약사항과 관련된 내용을 담고 있다. 또한 디자인 패턴들의 검색 및 관리의 효율성을 증가시키기 위한 패턴 분류법들에 대한 다양한 연구도 이루어지고 있다[5, 7, 17]. 먼저 Gamma[5]는 소프트웨어 디자인 패턴을 생성(creational), 구조(structural) 그리고 행위(behavioral)패턴으로 분류하고 있으며, Buschmann[7]은 패턴을 구조(architectural) 패턴, 디자인(design)패턴, 그리고 코드(idioms)패턴으로 분류하고 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[17]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. 이러한 분류방법들은 존재하는 패턴들을 좀더 효율적으로 구성하기 위한 시도이다.

2.2 패턴지향 분석 설계

패턴지향 분석 설계는 객체지향 분석 및 설계에 디자인 패턴 요소를 추가시켜 구성하는 형태이다. 이러한 형태의 패턴지향 분석 및 설계 방법은 Sherif[16]에서 제시되고 있다. 이 방법에서 디자인 패턴은 요구 분석단계에서 어플리케이션의 요구를 통해 사용 가능한 패턴을 식별하고, 이들을 설계단계에서 객체들과 함께 표현함으로써 설계 시 패턴정보를 유지관리 하기 위한 목적으로 이용된다. 일반적인 패턴지향 소프트웨어 개발과정은 (그림 1)과 같이 표현될 수 있다.



(그림 1) 패턴 지향 어플리케이션 개발 단계

(그림 1)은 패턴 지향 어플리케이션 개발의 개괄적인 단계를 보여준다. 이 과정을 살펴보면, 먼저 요구사항과 어플리케이션 명세 제약사항을 통해 필요한 객체와 사용 가능한 디자인 패턴을 식별한다. 이렇게 식별된 내용은 클래스의 설계에 이용되고 클래스들간의 디자인 요소가 추가된다. 이렇게 구성된 설계는 어플리케이션 명세와 패턴의 일관성을 검증하여 정확하다면 패턴 기능을 포함한 소스코드를 생성한다. 이렇게 생성된 소스코드는 요구사항과 어플리케이션 명세 제약사항을 테스팅 함으로써 정확성을 검증한다. 본 연구는 개발단계에서 설계 단계에 적용된 패턴 정보로부터 소스코드를 자동생성하기 위해 필요한 기법을 다룬다.

3. 디자인 패턴지향 개발환경 및 구성요소

디자인 패턴지향 개발환경은 (그림 2)에서 보여주는 것처럼 크게, 디자인 패턴을 XML 문서로 변환하여 저장하고 관리하는 부분과, 이러한 디자인 패턴정보를 이용해 소스코드 자동 생성과 분석에 이용되는 부분 그리고 저장된 패턴문서를 검색하고 활용하는 부분으로 크게 나눌 수 있다. 첫 번째 디자인 패턴을 XML 문서로 변화하여 저장하고 관리하는 부분은[19]을 통해 제시하였다. 본 연구는 (그림 2)에서 점선으로 둘러싸인 부분으로 디자인 패턴 정보를 이용해 소스코드 자동 생성과 관련된 기법을 제시하고, 어플리케이션 분석 및 유지보수를 위한 기능을 다루는데 있다. 이 과정에 필요한 구성요소는 다음과 같다.

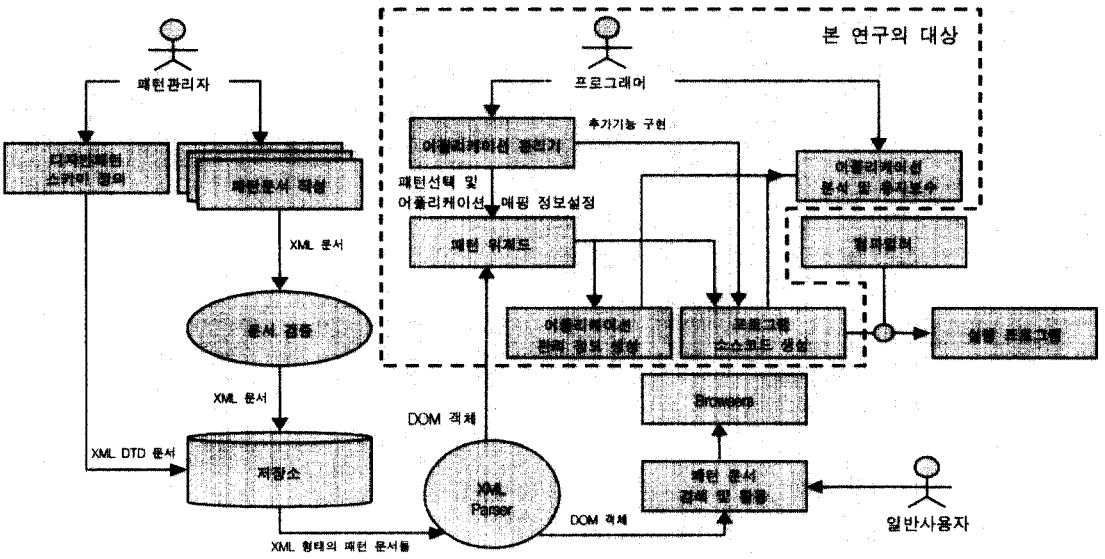
- 어플리케이션 관리기 : 이 부분은 어플리케이션 개발의 주요기능을 담당하는 부분으로 소스코드를 관리하고 조작하는 기능을 담당한다.
- 패턴위저드 : 어플리케이션에 사용될 패턴을 선택하고, 어플리케이션에 필요한 매핑 정보를 통해 해당 패턴을 구체화시키는 작업을 담당한다. 디자인 패턴정보를 어플리케이션에 구체화 시키는 기법은 제 4장에서 자세히 설명한다.
- 어플리케이션 관리정보 생성 : 어플리케이션 개발에 이용되어진 디자인 패턴 정보를 나타낸다. 이 정보는 디자인 패턴의 정보와 이 정보를 이용해 구체화된 어플리케이션 정보의 매핑 구조를 가지고 있다. 이 정보는 어플리케이션 내의 패턴 정보를 검색하고 분석하는데 이용된다.
- 어플리케이션 분석 및 유지보수 : 어플리케이션 관리정보와 프로그램 소스코드를 통해 이들에 적용된 디자인 패턴 정보를 검색하고 분석하는 역할을 담당한다.

4. 디자인 패턴정보로부터 코드 생성 자동화

디자인 패턴 정보로부터 해당 디자인 패턴에 대한 코드 자동생성을 위해서는 먼저 디자인 패턴 정보로부터 코드 자동화에 필요한 구성요소를 식별해야한다. 이장에서는 자동화에 필요한 구성요소를 식별하고, 코드 자동생성 적용단계를 보인다. 이 적용단계에 대한 기술은 목적 / 처리과정 / 산출물의 구성을 가진다. 각 구성단계에 수행 예는 KidList[9]를 사용한다. 이 예는 Clear와 Copy를 위한 두 버튼과 두 개의 리스트박스와 텍스트 필드로 구성되어 있으며, 디자인 패턴들 중 Mediator Pattern[5]과 Command Pattern[5]의 활용성을 보여주기 위해 사용된 예이다.

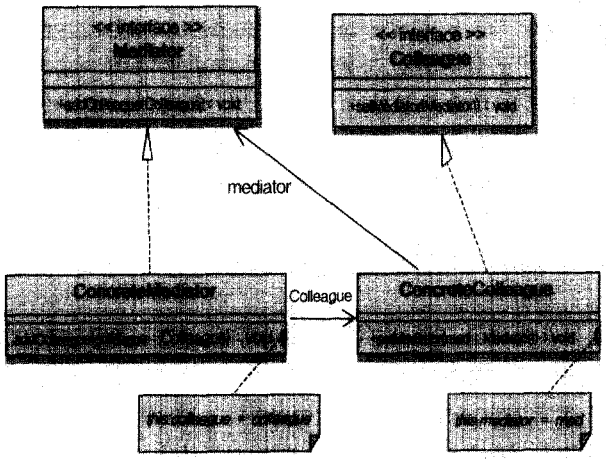
4.1 코드 자동생성을 위한 디자인 패턴 정보

디자인 패턴은 어플리케이션 개발 시 고려되는 문제들에



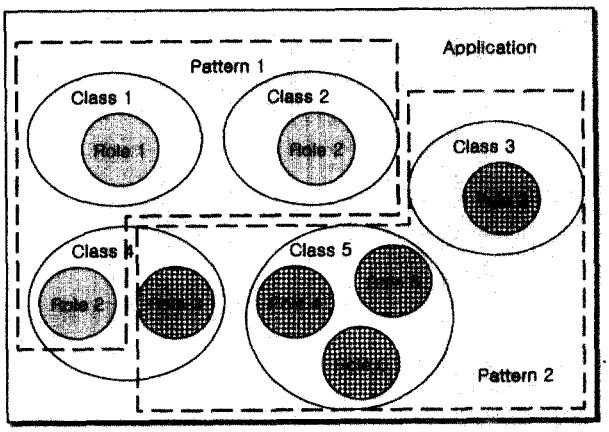
(그림 2) 디자인 패턴 지향 개발 환경 구조

대한 해결책을 기술한 것이다. 이러한 디자인 패턴들 중 객체지향 프로그램을 위해 정의된 패턴으로 GoF패턴[5]이 있다. 그러나 이들 디자인 패턴정보는 문제해결을 위한 추상적인 형태를 가지고 있기 때문에 코드 자동생성과 같은 구체화 정보를 이끌어내기 위해서 이에 필요한 정보를 추출할 필요가 있다. 어떠한 정보가 필요한지를 설명하기 위해 본 논문에서는 GoF패턴들 중 하나인 Mediator Pattern[5]의 예를 이용한다.



(그림 3) Mediator Pattern 구조

이 그림에서 보듯이 Mediator Pattern에 필요한 구성요소는 Mediator, Colleague, ConcreteMediator, ConcreteColleague 요소가 존재한다. 또한 각각의 구성요소들간에는 연관관계를 가지고 있으며 이 디자인 패턴을 위해 필요한 메소드와 프로퍼티를 포함한다. 그러나 이들 구성요소가 어플리케이션 내에 존재 시 어플리케이션 내부의 구성요소의 일부로써 동작한다. 어플리케이션 내의 구성요소와 디자인 패턴 내의 구성요소와의 혼돈을 피하기 위해 패턴에 사용되는 구성요소를 "역할(Role)"이라 정의하고, 어플리케이션 내의 구성요소를 "클래스(Class)"로 정의한다. 어플리케이션과 클래스, 패턴과 역할간의 상호관계는 (그림 4)에서 보여준다.

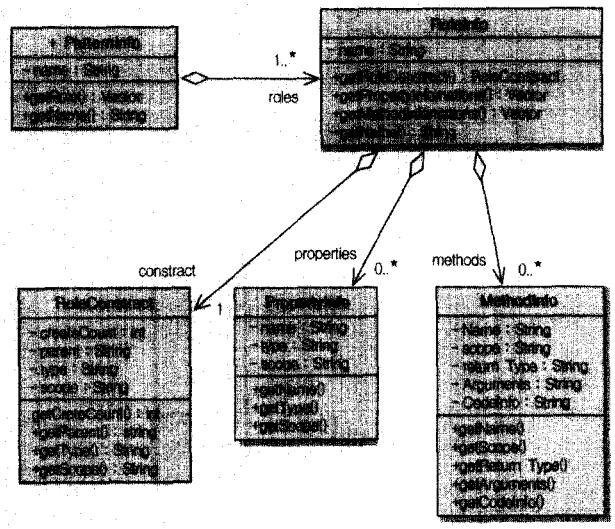


(그림 4) 구성요소간의 상호관계

이 그림에서 보듯이 어플리케이션에 디자인 패턴을 적용 시 어플리케이션의 구성요소와 디자인 패턴의 구성요소 사이에는 다양한 관계가 존재한다. 이들 관계를 분석하면 다음 과 같다.

1) 하나의 어플리케이션은 다수의 패턴을 포함할 수 있다.
   예) Application = { Pattern 1, Pattern 2 }
2) 하나의 어플리케이션은 다수의 클래스의 집합으로 구성된다.
   예) Application = { Class 1, Class 2, Class 3, Class 4, Class 5 }
3) 클래스들은 어느 특정패턴의 역할을 담당한다.
   예) Class1 = { Role 1 }
4) 하나의 클래스는 하나이상의 역할을 포함할 수 있다.
   예) Class 5 = { Role 4, Role 5, Role 6 }
5) 동일한 역할이 다른 클래스에 포함될 수 있다.
   예) Role1 = { Class2, Class 3 }
6) 하나의 클래스는 다른 패턴의 역할들을 포함할 수 있다.
   예) Class 4 = { Pattern 1 : Role 2 , Pattern 2 : Role 3 }

이러한 디자인 패턴의 구성요소와 관계분석을 통해 소스코드 자동생성을 위해 필요한 정보를 구성할 수 있다. 디자인 패턴 정보로부터 이끌어내는 정보는 디자인 패턴에서의 역할들과 각 역할의 이름 그리고 어플리케이션에서 구체화 될 수 있는 역할의 개수 등을 포함한다. 또한 각 역할에서는 해당 패턴을 수행하기 위해 필요한 프로퍼티와 메소드를 제공한다. 이들 역시 어플리케이션의 클래스의 프로퍼티와 메소드의 정보로 매핑되어 처리되어져야한다. 이를 기반으로 디자인 패턴을 통해 소스코드 생성에 필요한 정보는 (그림 5)와 같이 표현될 수 있다.

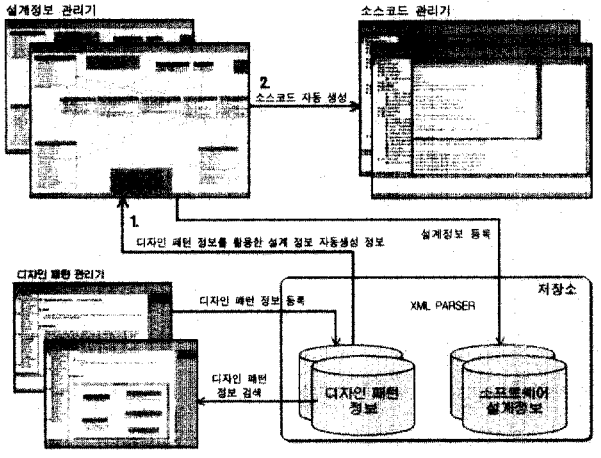


(그림 5) 자동생성을 위해 필요한 디자인 패턴정보

패턴에 대한 정보는 역할들의 집합으로 구성되어지고, 패턴에 포함되는 역할은, 생성에 필요한 제약사항과 프로퍼티, 메소드로 구성된다. 역할에 대한 제약사항은 해당되는 역할이 적용될 수 있는 수와 역할들 간의 관계정보 등을 포함하고 있으며, 프로퍼티 정보와 메소드 정보는 해당 디자인 패턴을 구성하는데 필요한 메소드와 프로퍼티 정보를 담고 있다. 이러한 정보는 디자인 패턴 자동생성을 위한 정보로 활용된다.

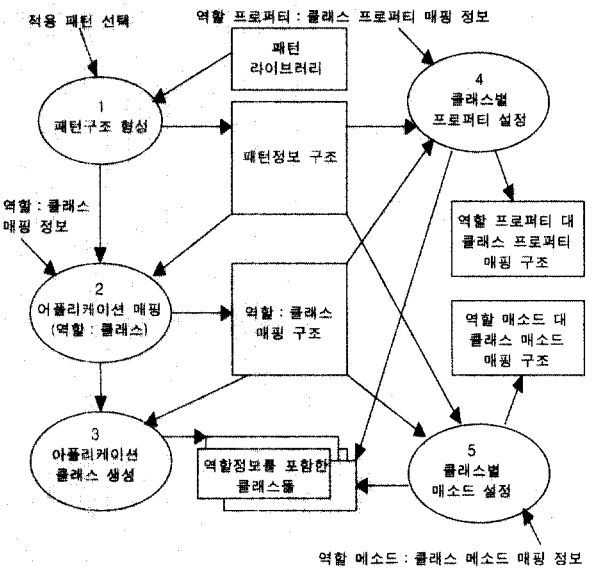
4.2 디자인 패턴정보를 이용한 코드 자동 생성 기법

어플리케이션 개발에 디자인 패턴정보를 적용시키고 이러한 과정을 통해 소스코드 자동 생성과 관련된 구현모델은 (그림 6)에서 보여준다. 실제 구현된 모델의 형태는 디자인 패턴 정보를 활용하기위해 관리되는 디자인 패턴관리기로 GoF 패턴정보를 역할 기반으로 저장 관리되며 어플리케이션 제작시 자동화 정보로 활용된다. 이 형태는 XML형태의 구조를 가진다. 두 번째로 설계정보 관리기로 이것은 클래스 다이어그램 형태를 가진다. 이 설계정보관리기를 통해 객체들에게 디자인 패턴을 적용하기위한 역할을 할당하고 관리된다. 또 하나의 관리기로 소스코드 관리기가 존재한다. 이것은 설계정보 관리기에서 생성된 클래스와 할당된 역할관계 클래스정보를 이용해 자동으로 소스코드를 생성하고 관리된다. 즉 본 논문에서 제시된 디자인 패턴 정보를 통한 소스코드 자동화를 요약하면 설계정보 관리기를 통해 해당 클래스들에 디자인 패턴을 적용시 디자인 패턴관리기를 통해 관리되는 디자인 패턴 정보를 이용하여 클래스에 역할을 할당한다. 이러한 역할의 할당은 해당 패턴의 특성에 따라 자동으로 연관관계가 설정되고 역할에 필요한 메소드와 프로퍼티 정보의 매핑 과정을 통해 소스코드를 자동 생성한다. 이러한 과정은 설계정보관리기의 디자인 패턴 위저드를 통해 쉽게 해결될 수 있다.



(그림 6) 디자인 패턴 활용을 위한 어플리케이션 개발 도구

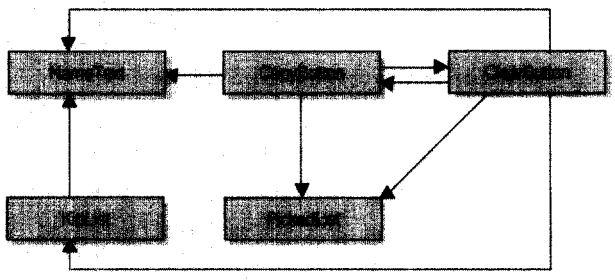
디자인 패턴을 적용하여 소스코드를 생성하는 단계는 다음과 같다. 첫 번째로, 선택된 패턴을 이용하여 역할들을 어플리케이션의 클래스에 매핑하는 과정이고 두 번째는 역할들에 대한 메소드와 프로퍼티 설정단계로 이루어진다. 이러한 단계를 통해 자동으로 생성되는 산출물은 크게 클래스 다이어그램의 설계정보와 소스코드 부분이다. 각 단계에 대한 설명은 "목적"/"처리 과정"/"산출물" 형태로 기술한다. 또한 사용되는 예는 KidList[9]를 이용하여 재시한다. 먼저 디자인 패턴을 어플리케이션에서 적용과정은 (그림 7)과 같다.



(그림 7) 어플리케이션에 디자인패턴 적용 과정

4.2.1 어플리케이션 적용 예

본 논문에서는 디자인 패턴에 대한 소스코드 자동생성과정을 설명하기 위한 KidList[9]예를 이용한다. 이 예제는 각 객체들간의 복잡한 제어구조를 가진 객체들을 보다 효율적으로 구성하기 위해 Mediator Pattern[5]과 Command Pattern[5]을 적용하여 구성하였다. 어플리케이션에서 사용된 클래스와 이들간의 관계는 (그림 8)과 같다. 또한 적용된 소스코드 형태는 객체지향언어인 Java를 활용한다.



(그림 8) 예제 어플리케이션의 클래스요소와 관계

이 어플리케이션의 요구사항은 다음과 같다.

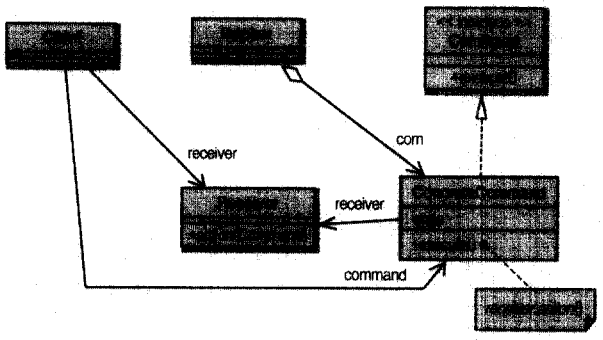
1) 프로그램 시작 시, Copy와 Clear button은 사용불가능 상태로 한다.
2) 왼쪽 리스트 박스에서 하나의 이름을 선택했을 때, 편집을 위해 이 내용을 텍스트필드로 이동하고, Copy 버튼

을 사용가능 상태로 한다.

3) Copy 버튼 선택 시, 선택된 이름은 오른쪽 리스트박스에 추가되고 Clear 버튼이 사용가능상태로 변한다.

4) Clear 버튼 선택 시, 오른쪽 리스트박스와 텍스트 필드의 내용이 지워지고, 왼쪽 리스트박스의 선택된 리스트는 해제된다. 또한 두 개의 버튼은 사용불가능 상태로 변한다.

이 예에서 객체들간의 의존성이 너무 많이 발생하기 때문에 객체들간의 컨트롤이 어렵다는 것을 발견할 수 있다. 이를 해결하기 위한 패턴으로 Mediator Pattern[5]을 이용할 수 있다. 이 Mediator Pattern은 다른 클래스들을 인식하는 클래스를 둠으로써 시스템을 단순화시키는 장점을 가진다. 즉 협력(Colleague) 클래스들의 행동은 중재자(Mediator) 역할을 담당하는 클래스에 의해 이루어짐으로써 모든 협력 클래스들간의 의존관계를 제거하는 효과를 가진다. 또한 객체간의 특정 동작에 대한 요청과 이 동작을 수행하는 대상과의 독립성을 보장하기 위해 Command Pattern[5]이 이용된다. 두패턴에 대한 구조도는 (그림 3)와 (그림 9)에서 보여준다.



(그림 9) Command Pattern 구조

이제 제시된 예제 시스템을 이용해 패턴정보에 대한 소스코드 자동화 과정을 기술한다. 이 과정은 크게 어플리케이션 클래스에 역할기능 추가 단계와 각 역할을 포함하는 클래스에 프로퍼티와 메소드 추가단계를 거쳐 이루어지게 된다.

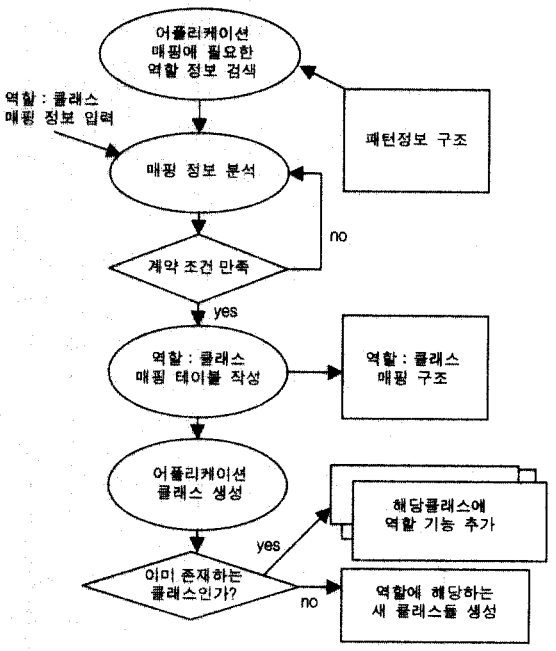
4.2.2 어플리케이션 클래스에 역할 기능 추가

1) 목 적

이 단계에서는 (그림 7)의 적용단계의 2와 3에 해당되는 부분으로, 어플리케이션에 추가하고자 하는 디자인 패턴 정보를 이용해서 각 클래스에 적절한 역할 정보를 추가하는데 그 목적이 있다.

2) 처리과정

개발자로부터 추가하고자하는 디자인 패턴 정보를 이용하여 필요한 역할 정보를 얻어온다. 이 얻어온 역할 정보를 이용해 개발자는 매핑할 클래스들을 선택할 수 있다. 이때 선택된 정보는 패턴정보에 포함된 제약사항을 통해 검증되어지고 소스코드 생성에 필요한 클래스 매핑 테이블을 생성한다. 이렇게 생성된 매핑 테이블은 어플리케이션에 포함된 클래스들에 역할 정보를 추가하는데 사용된다. 이때 사용할 디자인 패턴 구조를 형성하는데 필요한 클래스가 이미 어플리케이션에 존재한다면 해당 클래스에 역할기능만을 추가하고, 그렇지 않을 경우에는 이에 필요한 새로운 클래스를 생성하고 필요한 역할 기능을 추가한다. (그림 10)는 이러한 과정을 묘사하고 있다.



(그림 10) 클래스에 역할기능 추가 과정

3) 산출물

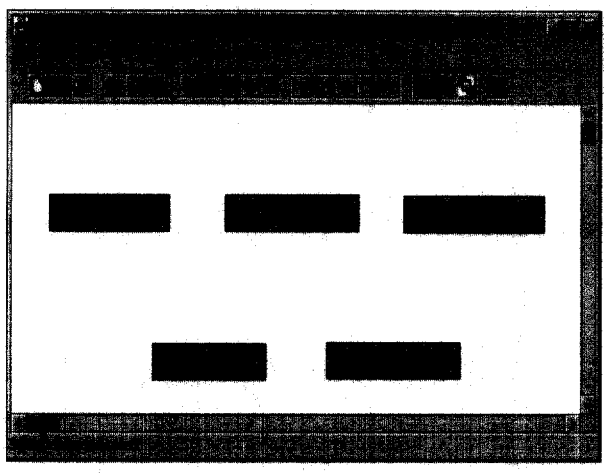
이 과정을 통해 얻어진 산출물은 디자인 패턴내부의 역할과 어플리케이션 내의 클래스들 사이의 매핑 관계를 표현하는 역할 대 클래스 매핑 구조 테이블(그림 11)과 이 구조 테이블을 통해 얻어진 역할 정보를 포함한 클래스들에 대해 자동으로 생성된 클래스 다이어그램과 소스코드이다.

Roles	type	scope	parent	Classes in Application
Mediator	interface	public	none	XMediator
ConcreteMediator	class	public	Mediator	XConMediator
Colleague	interface	public	none	XColleague
ConcreteColleague	class	public	Colleague	NameText
ConcreteColleague	class	public	Colleague	CopyButton
ConcreteColleague	class	public	Colleague	ClearButton
ConcreteColleague	class	public	Colleague	PickedList
ConcreteColleague	class	public	Colleague	KidList

(그림 11) 역할 대 클래스 매핑 테이블

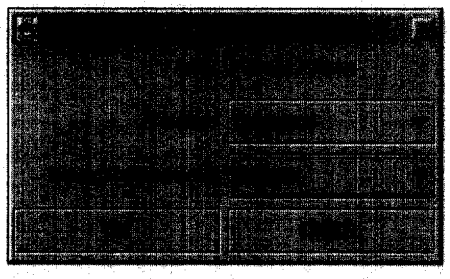
(그림 11)의 정보는 각 클래스들 사이에 많은 의존관계를 줄이기 위해 Mediator패턴을 적용하기위해 필요한 매핑 테이블을 보여준다. 이 매핑 테이블 정보를 이용해 어플리케이션의 적용과정은 다음과 같다.

먼저 어플리케이션에 필요한 클래스를 설계정보관리기를 통해 생성할 수 있다. 이렇게 생성된 형태는 (그림 12)과 같다. 이러한 클래스들에 패턴 위저드를 통해 디자인 패턴을 적용시킨다. 디자인 패턴정보는 역할단위의 메소드와 프로퍼티 그리고 연관관계를 가지고 있으며 (그림 11)의 매핑 테이블을 이용하게 된다.

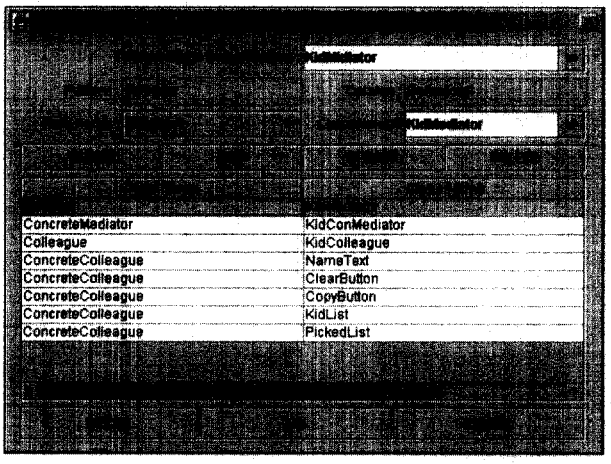


(그림 12) 어플리케이션에 요구되는 클래스들

디자인 패턴 선택과 클래스에 역할 할당과정으로 (그림 13)은 디자인 패턴 선택 박스를 나타내고 (그림 14)는 선택된 디자인 패턴의 역할관계 매핑 다이얼로그박스를 보여준다.

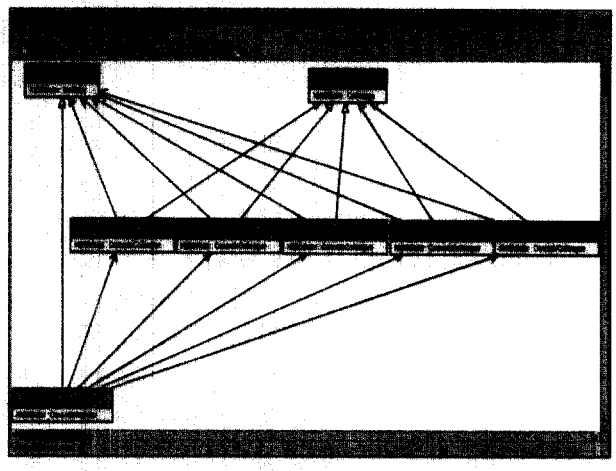


(그림 13) 디자인 패턴 선택



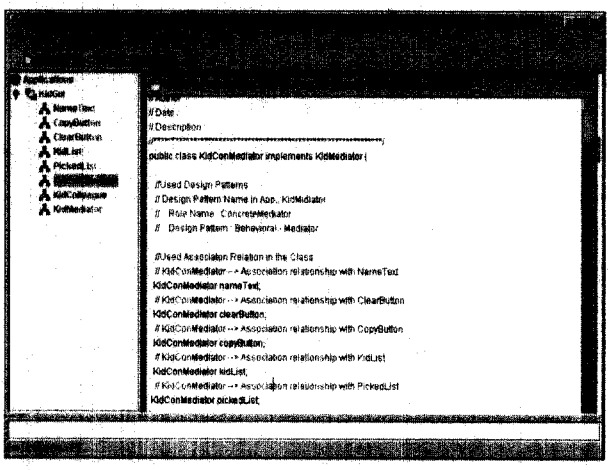
(그림 14) 클래스 대 디자인 패턴 역할관계 매핑

해당 패턴의 역할들은 디자인 패턴 정보로부터 읽어 들여 어플리케이션에 쉽게 할당가능하게 한다. 이렇게 설정된 정보를 이용해 디자인 패턴정보를 포함하는 설계정보와 소스코드가 자동으로 생성된다. (그림 15)는 이 위저드의 단계를 거쳐 자동 생성된 설계정보를 보여준다.



(그림 15) 위저드 단계를 거쳐 자동 생성된 클래스 다이어그램

(그림 15)에서 보듯이 각 클래스들에 해당 패턴의 역할이 추가되어지고 이들 역할들 간의 연관관계가 디자인 패턴정보를 이용하여 자동으로 생성된다. 이러한 클래스 다이어그램과 동시에 이에 적합한 소스코드가 자동 생성된다. (그림 16)은 자동으로 생성된 소스코드이다.



(그림 16) 위저드를 통해 얻어진 소스코드

(그림 16)의 소스코드는 KidConMediator로 Mediator 패턴의 ConcreteMediator 역할을 할당후에 생성된 소스코드를 보여준다. 이코드는 적용된 패턴정보와 역할정보를 담고 있고 이 역할에 의해 다른 클래스들과의 연관관계가 형성된다. 이때 기존의 클래스에 역할을 할당할 경우 해당 클래스에 정보의 추가가 이루어지고 패턴에 의해 새롭게 요구되는 클래스는 자동으로 생성된다. 즉 위저드를 통해 디자인 패턴의 적용

시 두 가지의 자동화가 이루어진다. 하나는 설계정보에 대한 클래스다이어그램이고 또 하나는 이러한 정보를 포함하는 소스코드가 자동으로 생성된다.

4.2.3 어플리케이션 클래스들에 메소드 정보 추가

1) 목 적

이 단계의 목적은 역할을 포함하는 클래스 내부에 해당 디자인 패턴의 기능을 수행하기 위해 필요한 메소드들을 추가한다.

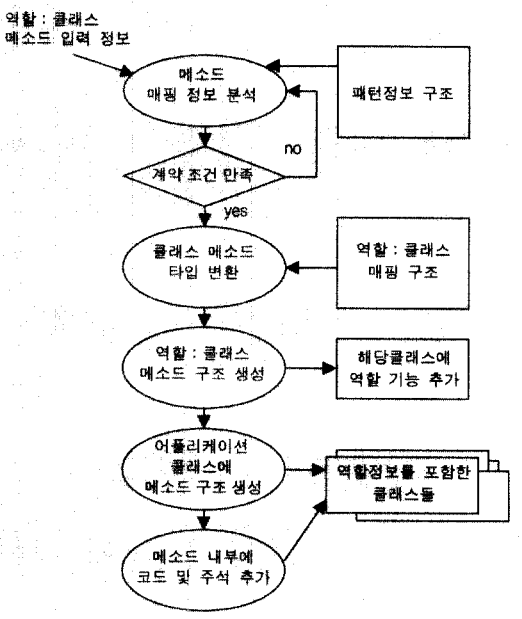
2) 처리과정

이 과정에서는 해당 역할을 수행하기 위해 필요한 메소드를 클래스 내부에 추가한다. 이러한 과정은 해당 디자인 패턴에 대한 정보와 어플리케이션에 필요한 메소드들의 정보를 이용해 이루어진다. 클래스 내부에 메소드가 추가될 때 디자인 패턴 내에 정의된 메소드들의 타입변환이 필요하다. 즉 메소드의 리턴타입과 메소드들에 전달되는 인자들의 타입은 구체화된 클래스 타입으로 변환되어 포함되어져야 한다. 또한 메소드 수행에 필요한 코드가 추가된다. 이 코드형태는 두 가지 형태가 있다. 첫 번째로 디자인 패턴의 역할에 포함된 메소드들의 코드 내용이 어플리케이션에 구체화될 때 변화 없이 적용되는 코드이다. 이러한 형태는 구체화된 클래스 내부에 직접적으로 포함되어진다. 또 다른 형태는 구체화 시 어플리케이션에 의해 변화가 발생되는 코드 부분으로 슈도코드 형태로 제공되어진다. 이러한 과정은 (그림 17)에서 보여준다.

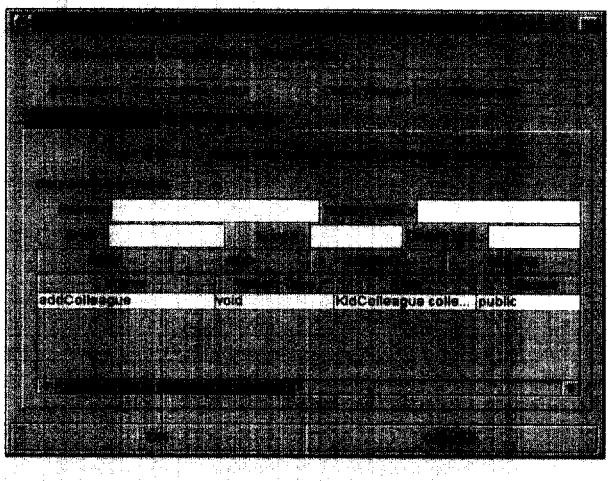
3) 산출물

이 과정을 통해 얻어진 산출물은 클래스 메소드와 역할 메소드들간의 매핑 구조(그림 18)와 이 매핑구조를 통해 자동으로 생성된 클래스 다이어그램과 소스코드이다.

클래스 내에 디자인 패턴에 필요한 메소드를 추가시키는 과정은 다음과 같다. 먼저 위저드를 이용해 해당 역할에 필요한 메소드와 클래스내의 메소드의 매핑과정을 통해 이루어진다. (그림 19)는 ConcreteMediator 역할이 할당된 KidConMediator의 역할 매핑 관계를 보여주고 있다. 해당되는 역할의 필요한 정보는 디자인 패턴정보를 통해 얻어오고 이에 적합한 메소드를 설정하는 과정으로 이루어진다. 또한 이 역할 메소드에서 처리에 필요한 코드가 존재한다면 소스코드에 추가된다. 역할 메소드 할당을 통해 얻어진 결과는 (그림 20)과 (그림 21)과 같다.



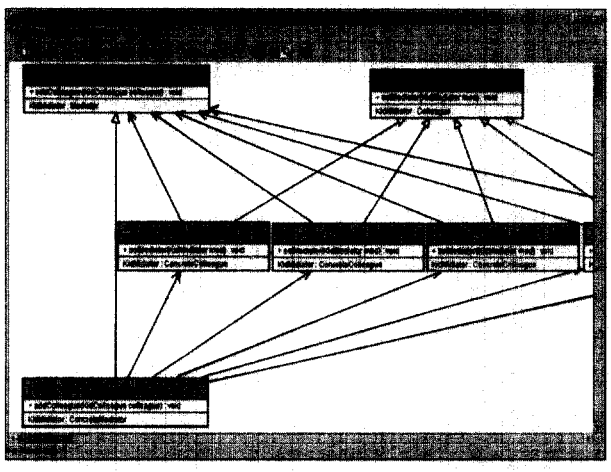
(그림 17) 클래스내 메소드 추가 과정



(그림 19) 클래스에 역할 메소드 할당

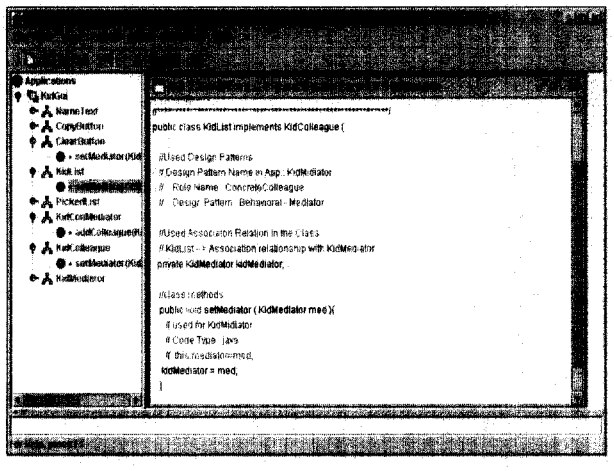
Role in Pattern	Class in App.	Method in Role	scope	ReturnType in role	Args in Role	Method in App	Args in App
Mediator	XMediator	addColleague	public	void	Colleague	addXColleague	XColleague
Colleague	XColleague	setMediator	public	void	Mediator	setXMediator	XMediator
ConcreteMediator	XConMediator	addColleague	public	void	Colleague	addXColleague	XColleague
ConcreteColleague	NameText	setMediator	public	void	Mediator	setXMediator	XMediator
ConcreteColleague	CopyButton	setMediator	public	void	Mediator	setXMediator	XMediator
ConcreteColleague	ClearButton	setMediator	public	void	Mediator	setXMediator	XMediator
ConcreteColleague	PickedList	setMediator	public	void	Mediator	setXMediator	XMediator
ConcreteColleague	KidList	setMediator	public	void	Mediator	setXMediator	XMediator

(그림 18) 클래스 내부의 메소드 매핑 테이블



(그림 20) 클래스에 역할 메소드 할당된 클래스 다이어그램

이와 동시에 할당된 메소드 정보는 소스코드에 추가되어진다. 이렇게 추가된 소스코드는 (그림 21)과 같다. 이 그림에서는 Colleague 역할이 할당된 KidList를 보여준다. 이 Colleague 역할을 수행하기위해 필요한 메소드는 Mediator를 설정하는 메소드인 setMediator 메소드이다. 이 메소드가 추가되어지고 디자인 패턴 정보의 역할 메소드안의 로직이 존재할 경우 클래스에 추가되어진다.



(그림 21) 클래스내 역할 메소드 할당된 소스코드

클래스에 프로퍼티를 설정하는 과정은 이 메소드 설정하는 과정과 비슷한 과정을 거친다.

4.2.4 클래스들 간의 역할 관계유형

클래스들의 관계는 여러 개의 역할들과 패턴정보를 포함하는 관계를 가지고 있다. 이러한 관계가 어떻게 활용되고 있는지를 분석하기위해 이 어플리케이션에 Command Pattern을 적용한다. (그림 22)는 Mediator 패턴을 포함하는 어플리케이션에 Command 패턴을 포함하는 클래스 다이어그램을 포함하고 있다. 예로 CopyButton은 Mediator 패턴의 Concrete-Collague 역할과 Command 패턴의 ConcreteCommand 패턴을 포함한다. 즉 이 클래스는 두가지 패턴의 포함하면서 각 패턴들의 하나의 역할을 포함하는 관계이다. 이 어플리케이션의 구성요소간의 관계를 정리하면 다음과 같다.

1) 하나의 어플리케이션은 다수의 패턴을 포함한다.

KidList(Application) = { Mediator, Command }

2) 하나의 어플리케이션은 다수의 클래스의 집합으로 구성된다.

KidList = { KidMediator, KidCommand, NameText, CopyButton, ClearButton, … }

3) 클래스는 어느 특정 패턴의 역할을 담당한다.

NameText = { ConcreteMediator } …

4) 하나의 클래스는 하나이상의 역할을 포함할 수 있다.

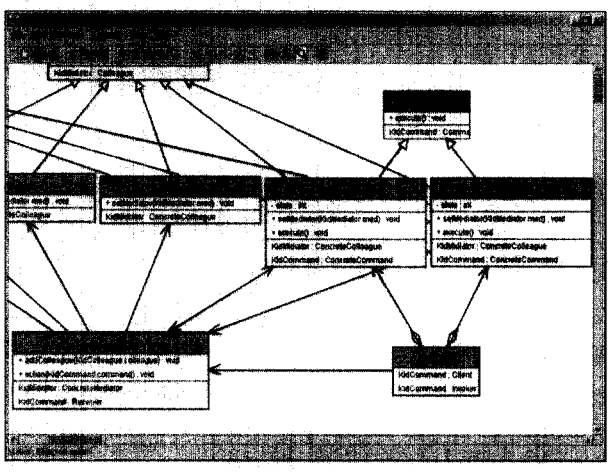
MainGui = { Client, Invoker }

5) 동일한 역할이 다른 클래스에 포함될 수 있다.

어플리케이션은 동일한 패턴은 여러 개 포함될 수 있다. 이러한 형태를 지원하기위해 어플리케이션에 할당된 디자인 패턴에 이름을 부여한다. 예로 한 어플리케이션에 두개의 State 패턴[5]이 적용되고 있을 경우에 ActionState, LoginState와 같이 이름을 부여함으로써 구별될 수 있다. 이러한 상황은 State 패턴에 포함된 하나의 역할이 다른 클래스들에 포함될 수 있다는 의미이다.

6) 하나의 클래스는 다른 패턴의 역할들을 포함할 수 있다.

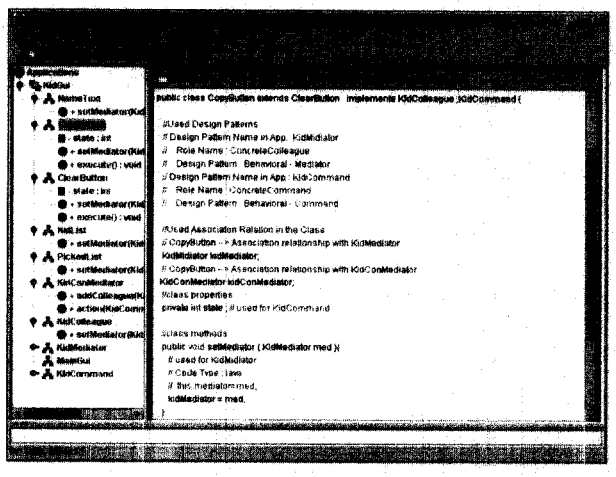
CopyButton = { KidMediator : ConcreteMediator, KidCommand : ConcreteCommand }



(그림 22) Mediator 패턴을 포함하는 어플리케이션에 Command 패턴 추가

(그림 23)은 두 패턴이 적용 후에 자동으로 생성된 소스코

드를 보여주고 있다. 디자인 패턴을 적용시 해당 패턴에 대해 독립적인 클래스의 생성은 아주 많은 클래스의 생성이 요구된다. 그러므로 클래스와 역할관계의 과정을 통해 다수의 역할을 포함시킴으로써 클래스 구성의 효율성을 증가시킬 수 있다. 또한 위저드를 통한 디자인 패턴의 적용은 클래스 다이어그램과 소스코드의 자동생성을 얻을 수 있고 이러한 것은 설계와 구현코드간의 일관된 구조를 만들어냄으로써 오류의 발생 빈도를 줄인다. 또한 클래스 다이어그램과 소스코드 안에 디자인 패턴정보를 포함하고 있기 때문에 수정이나 분석이 용이해 질수 있다.



(그림 23) 두 패턴의 역할이 포함되어 자동 생성된 CopyButton 클래스

디자인 패턴 정보에 대한 소스코드 자동생성은 설계정보인 클래스 다이어그램의 자동 생성과 이에 상응하는 소스코드의 생성과정으로 이루어지므로 설계와 구현 코드 정보의 일관성과 효율성을 증가시킬 수 있을 것이다.

5. 디자인 패턴에 대한 소스코드 자동화 생성 기법에 대한 비교 분석

5.1 디자인 패턴에 대한 소스코드 자동화 생성 기법 분석

해당 객체들 간의 기능을 수행하기 위해 패턴을 적용시키는 일은 상당히 많은 작업이 요구된다. 이러한 작업들이 수작업으로 이루어지게 되면 많은 작업 오류와 시간이 요구된다. 또한 패턴은 어플리케이션 적용 후 시각적으로 가시화되지 않기 때문에 객체들 간의 분석 및 유지 보수가 어려워진다. 그에 반해 본 자동화 과정을 통해 얻어진 어플리케이션 구조는 몇 가지의 이점을 가질 수 있다.

첫째로, 디자인 패턴의 역할을 구체화된 클래스에 자동적으로 매핑함으로서 일관된 형태의 소스코드 생성과 코드내의 오류 및 작업시간을 줄일 수 있다.

둘째로, 적용된 어플리케이션은 자신에 포함된 디자인 패턴정보를 유지하기 위해 클래스, 프로퍼티, 메소드들 간의 매핑 테이블들을 유지 관리함으로써 어플리케이션 내에 적용된 디자인 패턴을 가시화 할 수 있다. 이것은 어플리케이션의 유지보수 및 추적을 용이하게 한다. 이러한 정보는 어플리케이션 관점과 디자인 패턴 관점으로 분석될 수 있다. 어플리케이션 관점에서는 어플리케이션 내의 적용된 디자인 패턴과 포함된 클래스와 디자인 패턴내의 역할 관계를 식별할 수 있다. 또한 디자인 패턴 관점에서는 적용된 역할 관점으로 클래스들의 관계와 구조를 분석함으로써 어플리케이션에 적용된 디자인 패턴이 적절하게 동작하는지를 분석할 수 있는 자료로 활용된다. 마지막으로 디자인 패턴을 어플리케이션에 적용하기 위해 새로운 클래스를 생성하는 대신에 어플리케이션 내에 사용되고 있는 클래스를 이용해 디자인 패턴의 역할을 추가함으로써 디자인 패턴 적용 시 발생되는 클래스 생성 수를 줄일 수 있다.

5.2 기존의 디자인 패턴 자동화 기법

디자인 패턴 자동화 기법을 위한 몇 가지 연구들이 존재한다. 먼저 Sefika[15]는 디자인 패턴을 포함한 설계정보가 구현내용과 일치하는지를 검사하는 방법을 연구하였다. 이러한 방법은 각각의 디자인 패턴을 위해 하나의 규칙을 정의하고 그 규칙을 통해 구현 내용을 검증하는 과정을 통해 이루어진다. 이 방법은 패턴으로부터 개발된 시스템을 검증하기 위한 목적으로 사용되지만 패턴정보를 이용하여 어플리케이션 개발에 적용하는 방법에 대해서는 지적하지 않고 있다. Stephen[10]는 디자인 패턴을 이용해 소프트웨어 컴포넌트를 통합시키기 위한 방법을 연구하였다. 이 방법은 디자인 패턴 정보를 이용하여 어플리케이션에 사용되는 컴포넌트들을 효율적으로 통합하고자 하는 목적으로 연구되었다. 그러나 디자인패턴을 이용해 컴포넌트 래퍼 클래스를 생성하는데 목적을 가지고 있기 때문에 생성된 디자인 패턴들간의 관계나 관리측면에서 미흡하다. Budinsky[6]는 디자인 패턴으로부터 코드생성 기법을 제시하고 있다. 이 방법은 선택된 패턴에 대해 명시적인 제공된 어플리케이션 명세를 이용해 하나의 패턴을 자동적으로 생성하는 기능을 가진다. 그러나 이 방법 역시 코드 생성에 중점을 두고 있기 때문에 디자인 패턴들간의 관계 및 디자인패턴 정보 관리문제를 지적하지 않고 있다. Eden[2]는 메터언어를 이용해 어플리케이션 소스로 디자인 패턴 기능을 포함시키는 방법을 제시하고 있다. 그러나 이 방법은 디자인패턴을 적용하기 위해 메터언어에 대한 추가적인 지식이 필요하고 메터언어를 통해 만들어진 정보는 다시 프로그램에 적용시키는 단계를 거쳐야한다.

소스코드의 생성과 관련된 기존의 구현도구는 주로 설계도구를 통해 소스코드를 자동생성하거나 디자인 패턴 활용방

〈표 1〉 디자인 패턴 활용을 위한 기존의 연구와의 비교

관련 연구	접근 방법	문 제 점	제안된 구현 도구
Sefika [15]	디자인 패턴에 규칙정의 하여 설계 정보가 그 규칙을 만족하는 지를 검증하기 위한 목적	패턴에 대한 규칙 관리	XML기반의 패턴 정보 관리
		설계 정보에 디자인 패턴 자동화나 적용 방법에 대한 제시 부족	자동화를 통한 디자인 패턴 문서 적용
		사용된 패턴에 대한 추적성 부족	설계문서에 패턴 정보를 추가하고 각 적용 패턴별 검색지원
Stephen [10]	컴포넌트 간의 효율적인 통합을 위한 디자인 패턴 활용 방안 제시	웹퍼클래스의 생성을 통한 컴포넌트 간의 통합을 제공하지만 이들간의 정보제공관리 미흡 컴포넌트에 대한 웹퍼 클래스에 대한 소스코드 제공	객체들간의 책임할당을 통한 소프트웨어 설계 지원하고 이들 객체간의 패턴정보 유지 어플리케이션에 대한 클래스다이어그램과 소스코드 제공
Budinsky [6]	디자인 패턴으로부터 코드 자동생성 기법 제안	적용할 패턴 정보를 통한 소스코드 자동 생성은 제공하지만 설계정보에는 반영되지 않는다. 생성된 소스코드에 대한 패턴 정보 식별이 어려움	적용할 패턴 정보는 디자인 설계정보에 자동적으로 적용되고 이를 통해 소스코드 자동생성 함으로써 설계와 구현 정보에 반영 설계정보와 소스코드 내에 적용된 패턴정보 유지관리를 통한 추적 및 활용성 증대
Eden [2]	메터언어를 이용한 어플리케이션 소스코드 생성기법 제안	메터언어에 대한 추가 지식이 필요하고 이 방법 역시 설계 정보의 활용방법은 제시되지 않는다.	문서기반의 자동적인 패턴 적용과 설계정보 구성
기타 설계 도구	설계문서를 작성하기 위한 효율적인 기능제공 설계 문서에 대한 소스코드 제공	설계에 필요한 다이어그램 작성 및 문서 관리에 목적을 두고 있으며 디자인 패턴 활용 방법 제시 못함.	설계에 필요한 다이어그램 생성시 적용시킬 패턴이 존재시 자동적으로 패턴 요소 추가 및 관리

법에서 소스코드를 생성하는 과정에서 나타난다. 본 논문에서의 디자인 패턴에 대한 소스코드 자동생성은 XML문서로 관리되는 디자인 패턴 문서로부터 어떠한 방법을 통해 효율적으로 소스코드를 생성하고 관리하는가에 목적이 있다. 이에 기존에 제시되는 디자인 패턴 활용방법들과의 비교를 통해 얻어진 결과는 <표 1>과 같다.

6. 결 론

디자인패턴은 문제해결을 위한 추상화된 해결책이다. 이러한 정보는 어플리케이션에서 이용되기 위해서는 디자인 패턴에 대한 구체화 작업에 필요하다. 이를 위해 본 연구에서는 객체지향 프로그램 개발에서 효율적으로 디자인 패턴을 적용하기 위해 필요한 디자인 패턴 자동화 기법과, 어플리케이션 내부에 적용된 패턴들간의 분석 및 활용법을 보였다. 이를 위해 먼저 패턴지향 개발환경을 소개하고, 디자인패턴이 어플리케이션에 적용되기 위해 필요한 디자인 패턴 변환 과정을 기술하였다. 또한 어플리케이션 내에 적용된 디자인 패턴의 정보를 유지 관리함으로써 어플리케이션 내의 디자인 패턴의 관계와 각 구성요소간의 분석 및 활용성을 증가시켰다. 이러한 방법을 통해 보다 일관되고 정형화된 구조의 어플리케이션을 구축할 수 있고 어플리케이션 생성 시에 발생되는 코드의 오류 및 작업시간을 줄일 수 있다. 또한 어플리케이션 내에 사용된 디자인 패턴들의 정보를 이용해 유지보수의 효율성을 증가시킬 수 있다. 이러한 개발환경은 개발자들에게 보다 효율적인 패턴지향환경을 제공할 수 있을 것이다.

참 고 문 헌

[1] A. Cornils and G. Hedin. Statically checked documentation with design patterns, Technology of Object-Oriented Languages, 2000. TOOLS 33. Proceedings. 33rd International Conference on, 2000.

[2] A. Eden, A. Yehudai and J. Gil. Precise specification and automatic application of design patterns, Automated Software Engineering, 1997, Proceedings., 12th IEEE International Conference, 1997.

[3] B. Schulz, T. Genssler, B. Mohr and W. Zimmer, On the computer aided introduction of design patterns into object-oriented systems, Technology of Object-Oriented Languages, 1998, TOOLS 27, Proceedings, 1998.

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

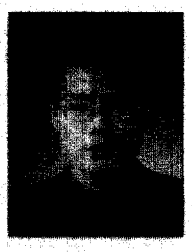
[5] E. Gamma, R. Helm, R.Johnson, and J.Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[6] F. Budinsky, M. Finnie, J. Vlissides, and P. Yu. Automatic Code Generation from Design Patterns. IBM Systems Journal, 35(2), 1996.

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

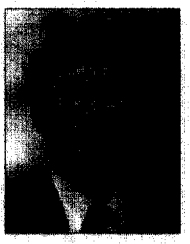
[8] G. Florin, M. Meijers, P. V. Winsen, Tool Support for Object-Oriented Patterns, Proc. of ECOOP'97, 1997.

[9] J. Cooper. Java Design Patterns, Addison-Wesley, 2000.
[10] M. Ohtsuki, N. Yoshida, A. Makinouchi, A source code generation support system using design pattern documents based on SGML, Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific, 1999.
[11] S. Stephen, S. Yau and Ning Dong, Integration in component-based software development using design patterns, Computer Software and Applications Conference, 2000, COMPSAC 2000, The 24th Annual International, 2000.
[12] S. Yacoub, Xue, H. and Ammar, H. H., Automating the development of pattern-oriented designs for application specific software systems, pp.163-170, Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on, 2000.
[13] M. Cinneide, P. Nixon, A methodology for the automated introduction of design patterns, oftware Maintenance, 1999, (ICSM '99) Proceedings. IEEE International Conference on, 1999.
[14] M. Claudia, C. Marcelo, P. Alain. Reifying Design Patterns as Metalevel Constructs. Electronic Journal of SADIO Vol.2, No.1, pp.17-29, 1999.
[15] M. Sefika, A. Sane, and R. Campbell. Monitoring Compliance of a Software System with its high-Level Design Models. Proceedings of the 18th International Conference of Software Engineering, ICSE '96, Berlin, Germany, March, 1996.
[16] M. Sherif, Yacoub and Heany H. Ammar Pattern-oriented analysis and design (POAD) : a structural composition approach to glue design patterns, Technology of Object-Oriented Languages and Systems, 2000, TOOLS 34, Proceedings. 34th International Conference on, 2000.
[17] W. Tichy, Essential Software Design Patterns. University of Karsruhe, http://wwwipd.ira.uka.de/~tichy/patterns/overview.html, 1997.
[18] W. Vanderperren, B. Wydaeghe, Towards a new component composition process, Engineering of Computer Based Systems, 2001, ECBS 2001, Proceedings. Eighth Annual IEEE International Conference and Workshop on the, 2001.
[19] 김운용, 김영철, 최영근, "코드 자동 생성을 위한 XML기반의 효율적인 디자인 패턴구조", 정보처리학회논문지 D, 제8-D권 제6호, 2001.



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



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
1992년~2000년 광운대학교 전산정보원 원장
2001년~2002년 광운대학교 정보통신연구원장
2002년~현재 광운대학교 교무연구처장
관심분야 : 프로그래밍 언어, 병렬 프로그래밍언어, 객체지향 설계 및 분석, 분산 컴퓨팅기술, Mobile agent