재공학 환경에서 적용성 향상을 위한 디자인 패턴의 UML 표현

°최성만* 김송주* 유철중* 장옥배* 이정열**

*전북대학교 컴퓨터과학과

**전북과학대학 인터넷정보계열

{°sm3099, songju}@cs.chonbuk.ac.kr, {cjyoo, okjang}@moak.chonbuk.ac.kr, ljy8383@hanmail.net

Design Pattern to Improve the Applicability In a Reengineering Environment Represented with UML

°Seong-Man Choi* Song-Ju Kim* Cheol-Jung Yoo* Ok-Bae Chang* Jeong-Yeal Lee**

*Dept. of Computer Science, Chonbuk National University

**Dept. of Internet Information, Jeonbuk Science College

요 약

본 논문은 재공학 환경에서 기존의 디자인 패턴을 적용성 향상을 위해 UML로 표현하였으며, 대상으로는 디자인 패턴 중에서 Strategy pattern과 Visitor pattern을 이용해 보았다. Strategy pattern에서는 {variation}과 {incomplete}를 이용하였다. {variation}은 메소드 구현시 패턴을 캡슐화하여 다양하게 변경될 수 있도록 하였다. 또한, {incomplete}는 주어진 관계를 만족하는 새로운 클래스가 패턴 인스턴스화 동안에 추가될 수 있도록 하였다. Visitor pattern에서의 {extensible}은 클래스 인터페이스가 패턴을 캡슐화하고 있는 개념으로 다양하게 변경될 수 있도록 하였다. 즉, 클래스 인터페이스는 패턴 인스턴스화에 의존적이며, 새로운 메소드와 속성을 클래스가 기능적으로 확장할 수 있는 기능을 갖는다.

1. 서 론

소프트웨어는 점점 대형화 및 복잡해짐에 따라 개발과정에서 많은 요구사항을 추가하고 기존의 요구사항을 만족해야만 한다. 따라서, 이러한 상황을 만족시키기 위해서는 소프트웨어를 재사용하고 적용성을 향상시켜 소프트웨어를 효율적이고 유지보수가 쉽도록 해야 한다. 이렇게 함으로써 소프트웨어의 규모가 축소되며 유지보수의 비용 또한 절감되는 장점을 갖는다. 이것을 수용하기 위해서는 기존의 소프트웨어에서 행동에 대한 변화가 없도록 해야 할 것이다[1]. 본 논문은 재공학 환경에서 기존의 디자인 패턴을 적용성 향상을 위해 UML로 표현하였으며, 대상으로는 디자인 패턴 중에서 Strategy pattern과 Visitor pattern을 이용해 보았다. Strategy pattern에서는 {variation}과 {incomplete}를 이용하였다. {variation}은 메소드 구현시 패턴을 캡슐화하여 다양하게 변경될 수 있도록 하였다. 또한, {incomplete}는 주어진 관계를 만족하는 새로운 클래스가 패턴 인스턴스화 동안에 추가될 수 있도록 하였다. Visitor pattern에서의 {extensible}은 클래스 인터페이스가 패턴을 캡슐화하고 있는 개념으로 다양하게 변경될 수 있도록 하였다. 즉, 클래스 인터페이스는 패턴 인스턴스화에 의존적이며, 새로운 메소드와 속성을 클래스가 기능적으로 확장할 수 있는 기능을 갖는다. 본 논문의 구성은 2장에서는 관련연구로서 재공학(Reengineering)의 개념 및 장점에 대해서 알아본다. 3장에서는 재공학 환경을 적용하기 위한 디자인 패턴의 구조에 대해서 기술하였고, 4장에서는 재공학 환경에서 적용성 향상을 위한 디자인 패턴을 UML로 표현하였다. 마지막으로 5장에서는 결론 및 향후 연구과제를 제시하였다.

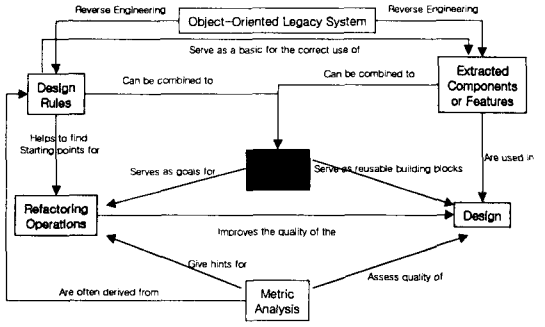
2. 재공학의 개념 및 장점

재공학은 설계 정보를 이끌어내는 것 뿐만 아니라, 이 정보를 이용하여 보다 나은 시스템을 만드는 것을 의미한다. 재공학을 통해 새로운 기능을 추가할 수도 있고 성능을 향상시킬 수도 있다. 변경하기 힘든 시스템을 재구성하여 변경이 용이한 시스템을 만들거나, 보다 나은 기능을 추가할 때도 재공학을 이용한다[2]. 재공학은 시스템 생명주기 전반에 걸쳐 생산성과 품질향상을 가지게 하며 소프트웨어 유지 보수성을 새로운 기법과 유지보수 툴의 적용을 통해 기존 시스템의 성능을 향상시킨다. 또한 기존 시스템의 이해성을 높이고 시스템 전반에 걸쳐 설계구조나 자료구조와 같은 소프트웨어 컴포넌트를 추출하는데도 필요하다. 이러한 컴포넌트들은 시스템 개발 또는 재개발시 재사용된다[3].

3. 재공학 환경을 적용하기 위한 디자인 패턴의 구조

재공학 분야에서 이용 및 관계를 정하기 위해서 기존의 디자인 패턴을 주의깊게 분석하였다. 이러한 결과 재공학 환경을 적용하기 위한 디자인 패턴의 구조를 <그림

1>에서 보여주고 있다. <그림 1>에서는 디자인 패턴이 객체지향 소프트웨어 개발의 몇 가지 다른 개념들과 관련되어져 해결하는 방안을 보여주고 있다[4]. <그림 1>에서 컴포넌트와 소스코드는 개발자가 개발하는 동안에 중요한 정보를 제공해주는 객체지향 레거시 시스템에서부터 후보 패턴에 적용되는 과정을 보여주고 있다.



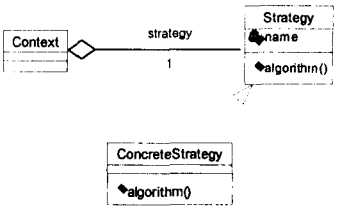
[그림 1] 재공학 환경을 적용하기 위한 디자인 패턴의 구조

4. 적용성 향상을 위한 디자인 패턴의 UML 표현

재공학 환경을 적용하기 위한 디자인 패턴의 구조를 이용하여 디자인 패턴 중에서 Strategy pattern과 Visitor pattern을 대상으로 UML로 표현해 보았다. 이렇게 해본 결과 Strategy pattern에서는 {variation}을 이용함으로써 메소드 구현시 패턴을 캡슐화하여 다양하게 변경될 수 있도록 하였다. 또한, {incomplete}를 이용하여 주어진 관계를 만족하는 새로운 클래스가 패턴 인스턴스화 동안에 추가될 수 있도록 하였다. Visitor patter에서는 {extensible}을 이용함으로써 클래스 인터페이스가 패턴을 캡슐화하고 있는 개념으로 다양하게 변경될 수 있도록 하였다. 즉, 클래스 인터페이스가 패턴 인스턴스화에 의존적이며, 새로운 메소드와 속성을 클래스가 기능적으로 확장할 수 있는 기능을 갖도록 하였다.

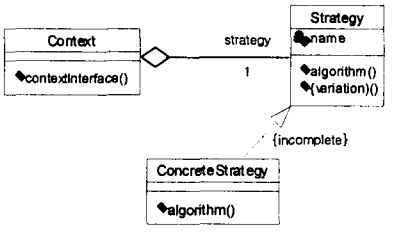
4. 1 Strategy pattern

Strategy pattern은 다양한 알고리즘이 존재하면 이들 각각을 하나의 클래스로 캡슐화하여 알고리즘의 대체가 가능하도록 한다. Strategy pattern을 이용하면 클라이언트와 독립적인 다양한 알고리즘으로 변형할 수 있다. 알고리즘을 바꾸더라도 클라이언트는 아무런 변경을 할 필요가 없다[5]. <그림 2>는 이러한 Strategy pattern의 클래스 다이어그램으로 보여준다.



[그림 2] Strategy pattern의 클래스 다이어그램

<그림 2>에서의 Context는 ConcreteStrategy 객체가 무엇인지를 구체화해준다. 즉, Strategy 객체에 대한 참조자를 관리하고, 실제로는 Strategy 서브클래스의 인스턴스를 갖고 있음으로써 구체화한다. 또한 Strategy가 자료에 접근해가는데 필요한 인터페이스를 정의한다. Strategy에서는 모든 알고리즘에 대한 공통의 오퍼레이션들을 인터페이스로 정의한다. ConcreteStrategy는 Strategy 인터페이스를 실제 알고리즘으로 구현한다. 재공학 환경에서 적용성 향상을 위한 Strategy pattern을 UML로 표현하면 <그림 3>과 같다.



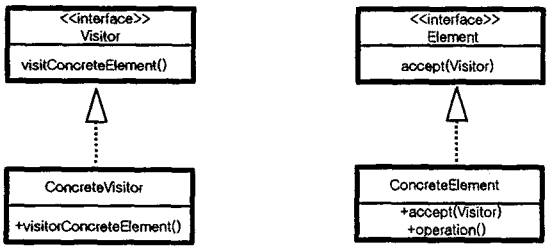
[그림 3] 적용성 향상을 위한 Strategy pattern의 클래스 다이어그램

<그림 3>과 <그림 2>와의 차이점으로는 {variation}과 {incomplete}를 이용하였다는 점이다. {variation}은 메소드 구현시 패턴을 캡슐화하여 다양하게 변경될 수 있도록 하였다. 즉, 메소드 구현을 패턴 인스턴스화에 의존적으로 하였다. 또 다른 차이점은 Strategy의 서브클래스에서 {incomlpete}를 포함하고 있다는 점이다. {incomplete}는 UML의 표기법에서 제공하고 있으며, 종속성 관계에서 적용된다. {incomplete}는 주어진 관계를 만족하는 새로운 클래스가 패턴 인스턴스화 동안에 추가될 수 있도록 하였다. 따라서, 패턴 인스턴스 생성에 관한 하나의 기준보다 더 많은 표현을 가능하게 한다. 이러한 결과 Strategy 클래스 계층은 알고리즘 자체의 재사용을 가능하게 하여 상속을 통해서 알고리즘 공통의 기능성들을 추출하고 이를 재사용 할 수 있게 한다. 또한, Strategy 클래스로 독립시키면 Context와 무관하게 알고리즘을 변

형시킬 수 있고, 알고리즘을 바꿀 때 이해하기가 쉬워진다.

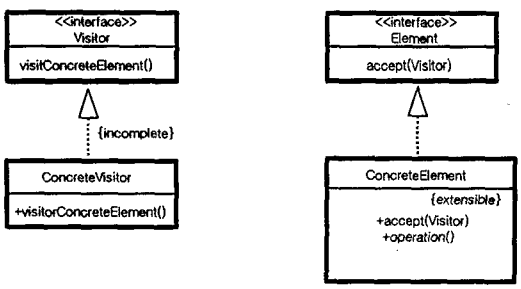
4. 2 Visitor pattern

Visitor pattern은 객체구조에 속한 요소에 수행될 오퍼레이션을 정의하는 객체이다. Visitor pattern은 처리되어야 하는 요소에 대한 클래스를 변경하지 않고 새로운 오퍼레이션을 정의할 수 있게 한다[5]. <그림 4>는 이러한 Visitor pattern의 클래스 다이어그램으로 보여준다.



[그림 4] Visitor pattern의 클래스 다이어그램

<그림 4>에서 Visitor는 각각의 ConcreteElement 클래스에 대한 Visit 오퍼레이션을 선언한다. 오퍼레이션의 이름과 인터페이스 형태는 Visit() 요청을 보내는 방문자에게 보내는 클래스를 명시한다. ConcreteVisitor()는 Visitor 클래스에 정의된 오퍼레이션을 구현한다. 각 오퍼레이션은 객체에 해당하는 클래스에 정의된 알고리즘을 구현한다. ConcreteVisitor 클래스는 내부상태를 저장하고 있으며, 알고리즘이 운영될 수 있는 상황정보를 제공한다. Element() 아규먼트로 Visitor 클래스를 받아들이는 Accept() 오퍼레이션을 정의한다. ConcreteElement()는 아규먼트로 visitor 객체를 받아들이는 Accept() 오퍼레이션을 구현한다. 재공학 환경에서 적용성 향상을 위한 Visitor pattern을 UML로 표현하면 <그림 5>와 같다.



[그림 5] 적용성 향상을 위한 Visitor pattern의 클래스 다이어그램

<그림 5>에서는 ConcreteElement 클래스의 포함된 {extensible}에서 상속된 태그 값에 의해 표현된다 {extensible}은 클래스 인터페이스가 패턴을 캡슐화하고 있는 개념으로 다양하게 변경될 수 있도록 하였다. 즉, 클래스 인터페이스는 패턴 인스턴스화에 의존적이며, 새로운 메소드와 속성을 클래스가 기능적으로 확장할 수 있는 기능을 갖도록 하였다.

5. 결론 및 향후 연구과제

본 연구는 점점 대형화 및 복잡해짐에 따라 개발과정에서 많은 요구사항을 추가하고 기존의 요구사항을 만족하도록 재공학 환경에서 적용성 향상을 위한 디자인 패턴을 UML로 표현해 보았다. 디자인 패턴 중에서 Strategy pattern과 Visitor pattern을 대상으로 하였으며 Strategy pattern을 재공학 환경에서 적용한 효과로서는 알고리즘 자체의 재사용을 가능하게 하여 상속을 통해서 알고리즘 공통의 기능성들을 추출하고 이를 재사용할 수 있게 하였다. 또한, Visitor pattern을 재공학 환경에서 적용한 효과로서는 복잡한 객체를 구성하는 요소에 속한 오퍼레이션을 쉽게 추가할 수 있으며, 객체의 구조에 적용될 알고리즘의 변화가 자주 발생하거나 객체 클래스에 대한 변화가 자주 발생될 때 적용을 쉽게 할 수 있었다.

향후 연구과제로는 다른 디자인 패턴도 재공학 환경에서 적용성 향상을 위해 UML로 표현하는 것이 필요하다.

6. 참고문헌

[1] 유명환, 정인정, "디자인 패턴을 이용한 리팩토링 사례 연구", 정보처리학회 추계학술발표대회 논문집, 제9권 제2호, pp. 2031-2034, 2002.

[2] 윤청, "패러다임 전환을 통한 소프트웨어 공학", 생능출판사, 1999.

[3] 김행곤, "객체의 개념적 인식과 논리적 분석에 의한 재공학 툴에 관한연구", 한국정보처리학회 논문지, 제3권 1호, pp. 200-210, 1996.

[4] Ladan Tahvildari, Kostas Kontogiannis, "On the Role of Design Patterns in Quality-Driven Re-engineering", in Proceedings of Conference on Software Maintenance and Reengineering, pp. 230-241, 2002.

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns : Elements of Reusabl Object-Oriented Software", Addison Wesley Longman, Inc., 1995.