설계 패턴의 자동 추출을 위한 역공학에 관한 연구

황 하 진*, 차 정 은**, 김 행 곤**

* 대구효성가톨릭대학교 경영학과
** 대구효성가톨릭대학교 전자정보공학부

요 약

시스템 성능을 개선하고 변화하는 환경에 적응하기 위해서는 기존 시스템을 실험, 분석함으로써 정확한 이해를 획득하고 나아가 재사용 자원으로 활용할 수 있는 소프트웨어 역공학이 필요하다. 또한 설계 문제의 추상화와 특정 영역의 일반적인 해결책에 대한 정보 표현 및 그 관계는 패턴 형식을 통해 효과적으로 나타낼 수 있다. 즉, 시스템의 설계 구조를 추출하여 시스템 분석과 설계를 향상시키고 표준화된 설계 용어 및 컴퍼넌트 관계 구조를 통해 재사용을 용이하게 하는 설계 패턴 추출을 위한 역공학은 중요하다. 따라서 본 논문에서는 기존 코드에서 설계 패턴 추출을 위한 역공학 적용의 타당성과 설계 패턴 자동 추출을 위한 몇 개의 알고리즘을 살펴보고 간단히 적용시켜 본다.

1. 서론

소프트웨어 기술의 급속한 발전과 호환성 측면의 문제, 그리고 사용자의 다양한 욕구 변화는 기존 시스템의 유지보수에 많은 기술적, 비용적 지원을 요구할 뿐 더러 그 생명을 점차 축소시키고 있다. 따라서 시스템 성능을 개선시키고 변화하는 환경에 적응하기 위해 시스템을 실험하고 분석함으로써 정확한 이해를 획득하고 나아가 재사용 가능한 자원으로 활용할 수 있도록 하는 소프트웨어 역공학이 절실히 요구 된다. 소프트웨어 역공학은 기존 시스템의 구성 요소와 그들의 관계 파악을 통해 설계 요소를 추출함으로써 논리적이고 구현에 독립된 추상화된 설계 정보를 제공하고 재사용케 함으로써 그 역할이 더욱 중요시 된다. 객체지향 설계는 클래스를 기본 단위로 이들 간의 정적 관계의 표현에는 효율적이지만 동적인 메시지 흐름에 관한 정보 제시에는 한계를 가지고 있다[1]. 그러므로 설계 문제의 추상화와 특정 영역의 일반적인 해결책에 대한 정보 표현 및 그 관계를 패턴의 형식을 통해 효과적으로 나타낼 수 있다. 설계 패턴은 객체지향 설계의 기본 문제 해결을 위한 보편적 추상화 표현으로 특정 문제 해결에서 한정적일 수 밖에 없는 개인적 경험을 추상화를 통해 다른 사람들과 공유하기 하며 개발자들 간의 의사 교환 도구로 사용될 공통 어휘를 제공한다.

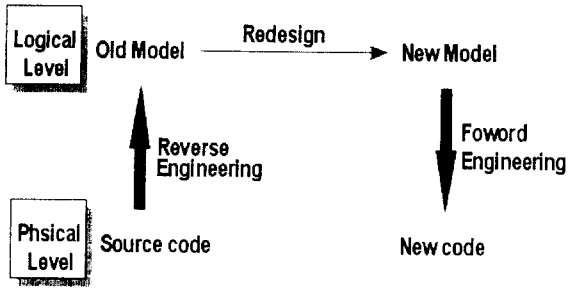
따라서 본 연구는 기존 원시 코드에서 객체지향 설계 패턴 추출을 위한 역공학 기술 적용의 타당성과 설계 패턴 자동 추출을 위한 역공학 도구의 아키텍처를 고려해 보며 패턴 추출을 위한 알고리즘을 살펴보고 그 적용 예를 제시한다.

2. 관련 연구

2.1 소프트웨어 역공학

많은 비용이 투자된 새로운 소프트웨어 개발이 항상 진보된 품질의 시스템을 확신하지 못하고 기존에 운영되는 소프트웨어와의 원활한 상호 호환을 보장하지 못할 뿐더러 사용자의 많은 적응 노력을 요구한다. 따라서 기존 시스템이 새로운 기술을 활용하고 또한 기

존 시스템 재사용을 통한 시스템 개발이 필요하다. 소프트에어 재공학의 한 기술 체계인 소프트웨어 역공학은 자동화된 도구를 이용하여 물리적이고 구현 의존적인 시스템 정보를 일반적이고 추상화된 시스템 컴퍼넌트와 그들 간의 관계를 식별함으로써 시스템을 개선하고 유지 보수 및 재개발을 위한 명확한 이해를 제공한다. (그림 1)은 소프트웨어 역공학의 개념적 체계를 나타낸 것이다[2].



(그림 1) 역공학의 두 가지 부분

2.2 설계 패턴

객체지향 설계에서 필요한 클래스와 그것들의 역할과 책임 분산을 하나의 패턴으로 추상화시킴으로써 가치있는 이전 설계 경험을 재사용할 수 있다. 설계 패턴은 클래스들과 인스턴스들 상의 추상화들을 식별하고 명명하며 정의함으로써 시스템 복잡도를 줄인다. 뿐만 아니라 재사용 컴퍼넌트 구축을 위해 경험있는 참여자들에 의해 얻어진 설계 지식 재사용을 위한 수단을 정제하고 클래스 계층들의 재조직이나 재요소화의 목표 달성을 용이하게 한다. 설계 패턴에 대한 연구는 소프트웨어 재사용에 대한 다양한 접근 방법 중 하나로 설계 패턴의 생성, 저장, 응용 그리고 효율적인 검색에 촛점을 맞추고 있다[3,4].

2.3 역공학에 의한 설계 패턴 추출

역공학은 물리적이고 한정적인 표현 정보들을 보다 일반적이고 추상화되어진 정보로 변화시키며 향후의 재사용을 위한 공유 저장소에 알맞는 형태로 저장한다. 이것은 기존 시스템의 이해성을 위한 문서 도구이며 새로운 시스템 구축의 바탕이 된다. 따라서 전체 시스템의 일반적인 설계 구조를 추출하여 시스템 분석과 설계의 생산성을 향상시키고 표준화된 설계 용어 및 컴퍼넌트 관계 구조를 통해 재사용을 용이하게 하며 품질을 향상시키는 설계 패턴 추출을 위한 역공학이 필요하다.

역공학적 관점에서 설계 패턴은 기존 시스템의 설계 과정에 대해 공유될 수 있는 공통 용어를 형성하고 특정 문제의 해결책 및 상반 관계를 제시함으로써 기존 설계 이해의 일반성을 획득, 사용할 수 있으며 구현에 한정적인 객체지향 프레임워크에 대한 문서화 도구로 사용될 수 있어 유지보수성과 유사한 구조의 새로운 시스템 구축시에 크게 도움을 줄 수 있다. 즉, 설계 패턴을 위한 역공학으로 시스템 이해성 증진과 소프트웨어 재사용 효과의 극대화를 객체지향 시스템 (재)개발에 최상으로 적용시킬 수 있다.

3. 설계 패턴 추출

3.1 추출 가능한 패턴의 특성

설계 패턴은 구문적 형태를 가지므로 의미 해석이 다양성하다. 그러므로 인식되어 추출될 수 있는 것은 해결책의 구현 결과 즉, 코드 템플리트나 설계 다이어그램이다. 패턴의 해결책이 뚜렷이 구별된다면 추출 가능하므로 하나의 표현이 가능해야하며 다중 표현은 배제되어야 신뢰성을 확보할 수 있다[5,6].

		Characterization		
		Creational	Structural	Behavioral
Jurisdiction	Class	Factory Method	Adapter(class) Bridge(class)	Templete Method
	Object	Abstract Factory Prototype Solitaire	Adapter(object) Bridge(object) Flyweight Glue Proxy	Command Iterator Mediator Memento Observer State Strategy
	Compound	Builder	Composite Wrapper	Interpreter Iterator Walker

(그림 2) Design Pattern의 분류

```
class Composite;
class Component{
public:
        //. . . . . .
  virtual Composite* GetComposite() { return 0; }};
class Composite : public Component{
public:
  void Add(Component*);   //. . . . . .
  virtual Composite* Getcomposite() { return this; }};
class Leaf : public Component { //. . . . . };
```

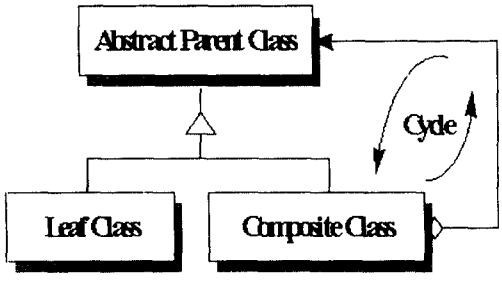
(그림 3) Composite 패턴 예

3.2 설계 패턴 추출을 위한 방법

Gammar는 재사용 규모와 추상화 수준에 따라 패턴을 (그림 2)과 같이 분류했다. 본 논문에서는 Gammar의 패턴 중 객체지향 개발에 널리 사용되는 패턴 몇 가지를 식별하고 추출 알고리즘을 제안한다.

(1) 클래스 간의 관련성에 의한 추출

서브 클래스들은 추상 부모 클래스의 메소드 구현을 위해 독립적으로 재정의 하거나 상위 클래스의 계속적 참조를 위해 상속/포함 관계에 의한 사이클을 형성한다. 후자와 같은 사이클의 발견을 통해 특정 패턴 즉, Composite와 Decorator 패턴을 찾을 수 있다. 이들 패턴들은 concrete 개념의 leaf 클래스와 컨테이너 개념의 composite 클래스 를 포함하여 프로토콜을 구현하는 추상 수퍼 클래스를 생성한다. Composite 클래스는 추상 클래스에서 상속 받은 요소 집합의 concrete 클래스이다. 그러므로 상속/조합 그래프에서 표현되어질 사이클 즉, 상속받은 클래스에서 포함 관련성을 찾음으로써 감지할 수 있다. (그림 3)은 Composite 패턴의 한 예이며 (그림 4)는 Composite 패턴의 구조이다.



(그림 4) Composite 패턴에서의 사이클

```
void View::Display() {
   SetFocus();
   DoDisplay();
   resetFocus();
}
void View::DoDisplay() {  }  //Client에서의 호출

void MyView::DoDisplay() {
      // render the view's contents }
```

(그림 5) Template Method 패턴의 원시 코드

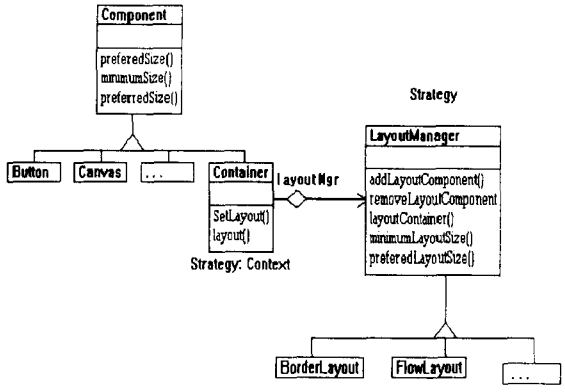
(2) 추상 클래스의 메소드 타입에 의한 추출

추상 클래스에서 선언된 메소드의 타입은 여러 타입으로 구분된다. 이 중 Template 메소드는 객체들 사이에서 관련성이나 추상 행위 혹은 제어의 일반적 흐름을 정의 함으로써 알고리즘을 대략적으로 서술하는 것이며 그 내부에 선언되어진 Hook 메소드들은 서브 클래스에서 오버라이딩되어짐으로써 알고리즘 상에 많은 다양성을 제공한다. 즉, 추상 클래스에서 메소드 타입 각각을 분류함으로써 Template Method 패턴을 추출할 수 있다.

Template Method 패턴은 서브 클래스가 알고리즘 구조의 변경없이 특정 단계를 재정의하게 함으로써 공통된 부분의 변경없이 작성된 기본 알고리즘 상에 많은 다른 다양성을 허용한다. 즉, 추상 클래스의 Template 메소드의 전체 알고리즘은 변화없이 그 내에 선어된 Hook 메소드들은 서브 클래스에서 재정의되어 변화되어지는데 이 두 메소드의 구별을 통해 인식할 수 있다. (그림 5)는 Template Method 패턴의 예로 "Display()"가 Templaye Method임을 나타낸다.

(3) 캡슐화된 객체의 파라미터를 통한 추출

동등 레벨에서 각 객체의 구체화된 역할 책임을 캡슐화 하고 추상 부모 클래스의 인터페이스를 사용함으로써 클라이언트 객체가 함수의 파라미터화를 통해 객체를 구체화 시키는 패턴들 즉, Strategy, State, Command들은 대표(클라이언트) 객체에 의한 캡슐화된 객체의 참조를 포함한다. 즉, 공통적인 조상 클래스로부터 상속된 형제 클래스의 집합에 속할 수 있는 서버 객체를 참조하는 하나의 대표

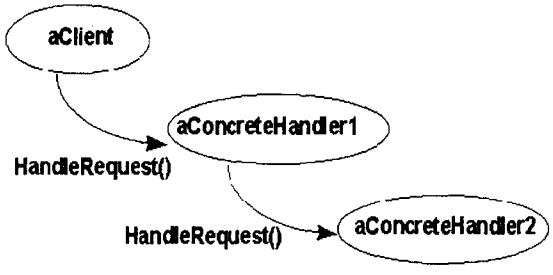


(그림 6) Strategy 패턴의 예

객체를 갖는다. 대표 객체는 형제 클래스 모두에 의해 공유화된 공통 프로토콜을 이용한다. 예로 Strategy 패턴은 알고리즘 군을 정의, 캡슐화하여 상호 교환 가능하게 하며 Strategy 알고리즘으로 하여금 사용하는 클라이언트들로부터 독립적으로 변경 가능케 한다. 이에 의한 해결은 인터페이스 뒤에서 알고리즘을 캡슐화하는 Strategy와 Strategy Object를 유지하고 이에 대한 알고리즘 구현을 위임하는 StrategyContext의 역할 수행으로 이루어지는데 그 예는 (그림 6)과 같다.

(4) 다이어그램을 통한 패턴 추출

복수의 패턴 구조가 혼합될 경우 원시 코드의 구조적 분석에 의해서는 구별이 어렵다. 특히 클래스나 객체의 상호 작용이나 그 역할을 분산시키는 보다 동적인 패턴 구조에서는 더욱 그러하다. 이런 경우 객체 메시지 다이어그램의 파악은 패턴 파악에 매우 유익하다.

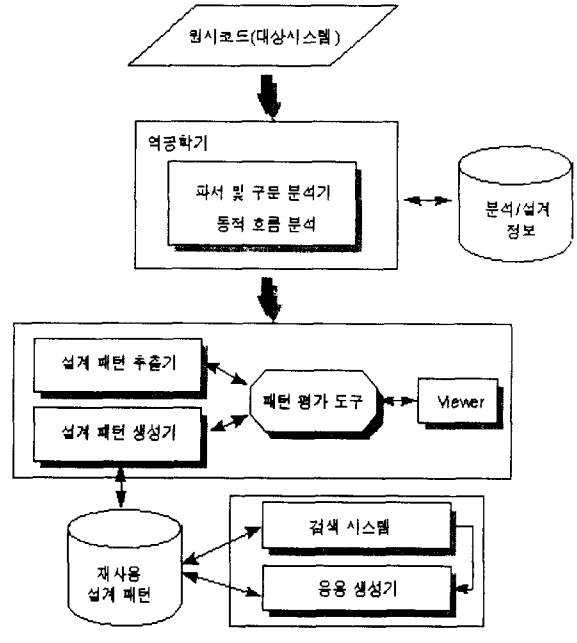


(그림 7) Chain of Resposibity

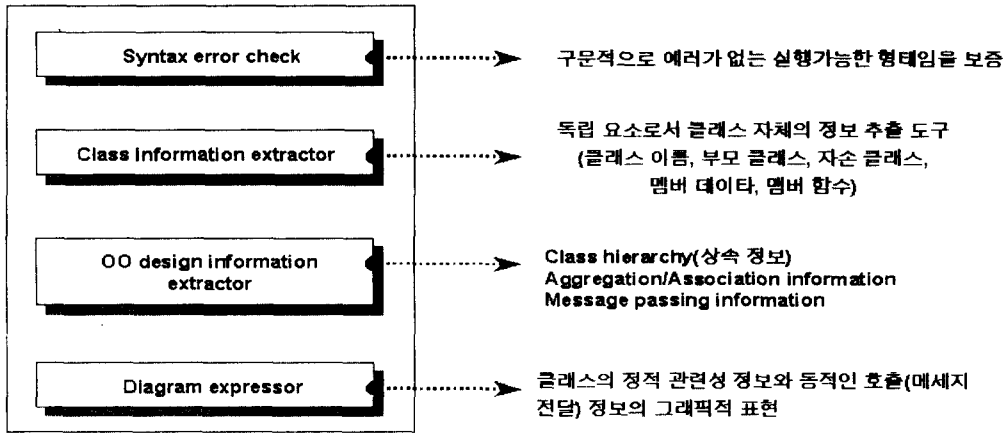
패턴 Chain of Responsibility는 Decorator와 Composite의 객체 구조와 결합하여 구성되어지므로 객체 메시지 다이어그램을 사용하여 감지하는 것이 용이하다. 이것은, request를 제어할 기회를 하나 이상의 객체에 부여함으로써 요구의 송신자가 수신자와 결합하는 것을 피하는 패턴으로 객체가 요구를 제어할 때까지 수신 객체들을 연결하고 그 체인을 따라 전달한다. 따라서 객체 메시지 다이어그램에서 동일한 메시지 선택기를 가지는 메시지 전송의 순차적인 시리즈는 Chain of Responsibility의 사용으로 인식될 수 있다(그림 7).

4. 설계 패턴 추출을 위한 역공학 도구

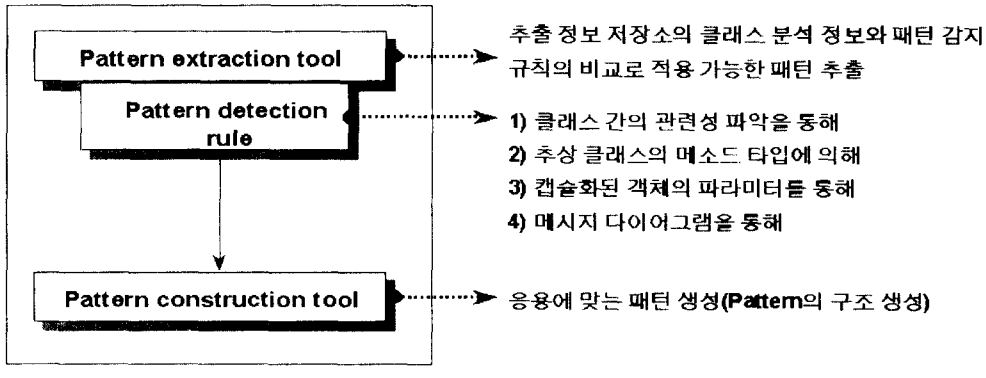
기존 시스템에서 역공학 적용으로 설계 패턴을 추출하기 위한 본 연구 위한 시스템 아키텍쳐는 일반적인 처리 과정에 따라 (그림 8)과 같이 제시한다. 대상 시스템의 원시 코드는 역공학기를 거치면서 개념적이며 논리적인 데이터 구조가 추출되며 이것은 추상화된 일반적인 클래스로 변환된다.



(그림 8) 패턴 추출을 위한 역/재공학 도구



(그림 9) 역공학 도구



(그림 10) 설계 패턴 추출기

이 과정에서 구문과 동적인 의미 분석을 위해 파서 및 동적 흐름 분석기가 필요하다. 이렇게 형성된 클래스들은 저장소에 저장된다. 다음 단계는 추출된 클래스 정보를 파악하여 가능한 설계 패턴들을 추출하고 생성하며 적절한 패턴이 생산되어졌는지를 평가하며 이것은 사용자에게 편리한 뷰어를 통해 보여진다. 다음 단계는 새로운 응용을 위한 순공학으로의 재구축 단계로 형성된 설계 패턴은 가치있는 재사용 대상으로 검색 시스템을 통해 필요시 검색되어 응용 생성기를 통해 새로운 시스템 구축을 위해 활용되어진다.

본 시스템은 세 가지 단계의 서브 시스템으로 구분할 수 있는데 이것은 다음과 같이 요약된다.

1) 역공학기 : 기존 시스템의 원시 코드를 읽어 들여 포함되어진 설계 정보를 추출하는 시스템으로 구조적이고 논리적인 데이터 구조와 클래스 간의 상호 관련성 그리고 동적인 호출 관계를 파악하여 저장소에 저장한다.

2) 설계 패턴 추출 도구 : 분석되어진 시스템의 분석 정보에서 설계 당시의 의도를 파악하여 클래스와 인스턴스 그리고 그것들의 역할과 협동 관계 및 책임 분산을 식별, 추상화함으로써 사용되어진 설계 패턴을 감지, 추출한다. 이것은 패턴 형식으로 구성하여 평가한 후 사용자에게 제시된다.

3) 재사용 시스템 : 생성된 설계 패턴들은 재사용의 단위로서 저장소에 저장되어지며

이것은 새로운 시스템 구축을 위해 검색되어지고 응용 생성기를 통해 조합되어져 또 다른 시스템으로 완성된다.

(그림 9)와 (그림 10)은 역공학 도구와 패턴 추출 도구에 대한 세부 시스템 구조이다.

5. 결론

본 연구에서는 기존 시스템에서 설계 패턴 추출을 위한 역공학 도구에 관한 연구 중 초기 단계로 기존 코드에서 설계 패턴 추출을 위한 역공학 적용에 관한 타당성과 패턴들의 기본적인 속성을 파악하여 몇몇 패턴들의 자동 추출을 위한 몇 개의 알고리즘을 살펴보고 이를 구현하기 위한 패턴 추출 역공학 도구의 아키텍쳐를 제시했다.

Composite 패턴은 상속받은 클래스에서 포함 관련성을 찾음으로써, Template Method 패턴을 감지하는 것은 추상 클래스에서 메소드 타입 각각의 분류를 통해 그리고 Chain of Responsibity 패턴은 동일한 메시지 선택기를 가지는 메시지 전송의 순차적인 연결을 발견함으로써 추출이 가능하다.

설계 패턴 추출을 위한 시스템은 클래스 자체 정보와 객체지향 설계시 필요한 클래스 간의 관련성(inheritance, association, aggregation, 메시지 패싱) 정보를 추출하는 역공학 도구와 적절한 패턴을 감지, 생성하는 패턴 생성기로 구성된다.

설계 패턴을 위한 역공학으로 시스템 이해성 증진과 소프트웨어 재사용 효과의 극대화를 객체지향 시스템 (재)개발에 최상으로 적용시킬 수 있을 것이라 기대된다.

현재 본 연구 진행에서는 대상 시스템의 선정과 자동화 지원 도구 활용의 최대화 그리고 추출 패턴의 적절한 평가 요소의 설정이 요구된다.

향후 보다 체계적인 패턴 식별 알고리즘의 정의와 이를 통한 자동화된 역/재공학 도구 구축과 재사용 시스템의 프레임워크로의 조합을 목적으로 한다.

【참고문헌】

[1] James Rumbaugh et al, "Object-Oriented Modeling and Design", Prentice Hall, 1991

[2] Carma McClure, The Three Rs Of Software Automation : Re-Engineering, Repository, Reusability, Prentice Hall, 1992

[3] Ted Lewins et al. "Object-Oriented Application Frameworks", Manning Publications Co, 1995

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

[5] E.Gamma, "Applying Design Pattern In Java", Jounal of Java Report, Dec. 1996

[6] Kyle Brown, "Design Reverse-Engineering and Automated Design Pattern Detrction in Smalltalk", http://www2.ncsu.edu/eos/info/tasug/kbrown/thesys2.htm

[7] 김 행곤 외, "자바 프로그램 개발을 위한 설계 패턴에 대한 연구", 한국정보처리학회 춘계 학술발표회, 1997, 4

[8] 김 행곤 외, "설계 패턴 자동 추출을 위한 역공학에 관한 연구", 한국정보과학회 추계 학술발표회, 1997, 10