2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

설계패턴 추출에 관한 연구

이상식*, 송영재
경희대학교 전자계산공학과
leess@case.kyunghee.ac.kr

A Study on the Extraction Design Pattern

Sang-Sik Lee*, Young-Jae Song
Dept of Computer Engineering, Kyung-Hee University

요 약

소프트웨어 역공학은 기존 원시코드에서 구성요소와 그들의 관계 파악을 통하여 설계요소를 추출함으로써 논리적이고 구현에 독립된 추상화된 설계정보를 제공하고 재사용 하게 함으로써 그 역할이 중요시되어왔다. 최근 패턴구조 컴포넌트의 재사용은 기존의 클래스 단위의 재사용 시스템에 비하여 시스템 설계 및 구현단계에서 개발비용의 절감과 개발기간의 단축에 도움을 줄 수 있으며, 이를 통해 개발되는 시스템의 생산성과 안정성을 보다 향상시킬 수 있다 .본 연구는 기존의 소스코드를 분석하여 재사용 가능한 객체를 추출하고 클래스와 클래스들 간의 관련성을 찾아내어 패턴정보를 추출하고 그 정보를 저장하기 위한 데이터베이스 스키마를 설계하였다.

1. 서론

원시코드로부터 재사용 가능한 부품을 추출하는 연구는 역공학기법의 한 분야로부터 출발하였다. 소프트웨어 역공학은 기존 원시코드에서 구성요소와 그들의 관계 파악을 통하여 설계요소를 추출함으로써 논리적이고 구현에 독립된 추상화된 설계정보를 제공하고 재사용하게 함으로써 그 역할이 더욱 중요시된다. 객체지향 설계는 클래스를 기본단위로 이들 간의 정적관계에는 효율적이지만 동적인 메시지 흐름에 관한 정보제시에는 한계를 가지고 있다[1]. 그러므로 설계문서의 추상화와 특정영역의 일반적인 해결책에 대한 정보표현 및 그 관계를 효과적으로 나타내기 위해서는 패턴형식이 적절하다. 설계패턴은 객체지향설계의 기본문제 해결을 위한 보편적 추상화 표현으로 특정 문제 해결에서 한정적일 수밖에 없는 개인적 경험을 추상화를 통해 다른 사람들과 공유하게 하며 개발자들 간의 의사 교환도구로 사용될 공통어휘를 제공한다[2]. 따라서 본 연구에서는 기존의 객체지향 소스코드를 기반으로 클래스와 멤버함수, 멤버변수들을 추출한 후에 추출된 객체들간의 Dependency, Associations, Generalization, Aggregation을 찾아내고 Compositor, Strategy 등 잘 알려진 추상화된 디자인 패턴들을 추출하는 알고리즘과 그 패턴 정보들을 저장하기 위한 데이터베이스 스키마를 설계한다.

2.관련 연구

2.1 소프트웨어 역공학

소프트웨어 역공학은 기존 코드나 소프트웨어 생명주기의 마지막 단계에서 얻어지는 산출물로부터 초기단계의 설계 사양이나 요구분석서, 데이터흐름, 프로그램 구조, 데이터베이스 설계, 파일구조 등의 정보나 문서를 생성하는 것이다.[2] 이 과정은 코드 레벨의 정보를 보다 상위레벨의 정보로 변환하게 된다.

2.2 설계 패턴

패턴은 객체지향 패러다임에서 객체의 정형화를 위한 방법으로 적용되고 있으며, 프로젝트를 설계하는데 사용되는 기본적인 청사진을 제공해 준다. 패턴의 근본 개념은 프로그램 설계에 대한 개념과 일치하며 구체적인 사례를 일반적인 형태로 추상화시

키는 과정을 부가하는 것이다.[1] 패턴은 특수한 경우에만 적용되는 것이 아니며 개발에 필요한 세부사항까지 구체적인 방법을 일일이 해결안을 제시하는 것이 아니고 상위 설계단계에서 적용될 수 있는 개념이다.

2.3 역공학에 의한 설계 패턴 추출

역공학 도구를 이용하여 원시코드에 포함된 시스템의 분석 설계 정보와 설계당시의 의도를 파악하여 클래스간의 역할 및 협동, 역할 분산 관계를 발견한다. 패턴 추출 알고리즘을 적용하여 가능한 패턴을 추출하며, 추출된 패턴을 구축된 패턴과 비교하여 적절성을 평가한다.

3. 기존 시스템으로부터 설계 패턴 추출

3.1 클래스 정보 추출

추출 알고리즘에는 클래스를 추출할 원시코드 화일에 관한 정보 저장을 위한 임시테이블이 필요하다. Source 테이블과 추출된 클래스 코드 저장을 위한 Class 테이블이다. 클래스간의 호출 관계와 멤버함수와 변수를 저장하기 위한 SupperClass, SubClass, Call, Function, Variable 임시 저장 테이블 등이 운영된다. SupperClass와 SubClass 각각은 상위 클래스와 하위 클래스 관계를 저장한다. 클래스 추출 과정으로서 구성의 기본 요소중 첫 번째의 '};'와 'class' 토큰을 ATC() 함수로 찾는다. 존재하면 그 위치를 각각 기억시켜 둔다. 같은 방법으로 클래스 모두를 추출하여 CLASS 테이블에 저장하는데, 추출 알고리즘은 아래와 같다.

```
do while
   if (found "};")
     endClass=location of "};"
    else
     exit
    endif
   if (found "class")
     startClass=location of "class"
     seek n+1 th "};"
      if (found)
       test=location of "};"
     endif
  if (startClas < test)
    seek "struct"
     structure=location of "struct"
     k=k+1
    if test < structure
      endClass=test
    endif
  endif
  mC&mm  have string from startClass to endClass
   n = n+1
enddo
```

3.2 패턴 정보 추출

객체간에 이루어져 있는 연관관계를 추출한 후에 점차로 일반화 관계, 집단화 관계, 의존성 관계를 추출해나간다. 이는 클래스의 상속관계 외에 어떤 클래스 내에서 상대 클래스의 멤버함수를 정의하는 클래스간의 호출관계 정보를 추출하는 과정이다.

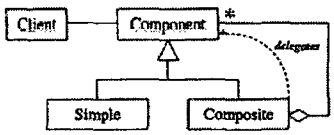
```
select class
go top of file
do while
store 1 to class_count
msource=클래스소스
mclassNo=클래스번호
do while
seek "::" from msource
if (exist "::")
endClass is location of "::"
else
exit
endif
do while
jump character
s
mc=mc-1
enddo
mCN&class_count have string
from mc to endclass-mc
 select file of "call"
 replace 클래스번호 with mclassNo
 replace 호출클래스명 with;
 mCN&class_count
class_count= class_count+1
enddo
select class
skip
enddo
```

3.3 설계패턴의 식별

현재 객체지향 시스템으로부터 가장 기본적인 패턴으로 인정받고 있는 Gamma의 설계 패턴을 바탕으로 위의 패턴정보 추출과정을 통하여 임시 DB에 저장된 설계패턴 정보들의 재사용 가능성을 식별한다.

① Composite 패턴

Composite 패턴은 수퍼클래스와 하위 클래스간의 포함관계 정보를 참조하여 찾아낸다. 이는 상속 관계 속에서 반복적인 구조로 나타난다. 여기에서 어그리게이션과 딜리게이션과 같은 구조적 패턴을 추출한다.

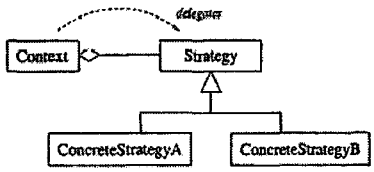


<그림 1>Composite 패턴의 구조

② Strategy 패턴

Strategy 패턴의 특성은 인터페이스나 다른 서브클래스의 선택이 가능한 클래스에 알고리즘을 대표하

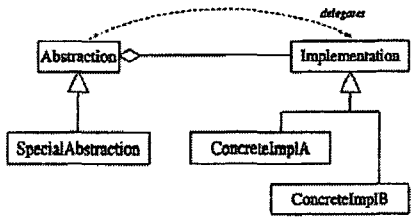
는 것이다. 추상클래스의 엑세스를 통해 서브클래스의 캡슐화된 알고리즘 구현을 위임하는 대표 클라이언트 객체 식별을 통해 패턴을 식별한다.



<그림 2>Strategy 패턴의 구조

③ Bridge 패턴

Bridge 패턴의 특징은 두 평행 분류 체계들이다. 하나는 좀더 추상적 모델을 위한 것이고 다른 것은 그것의 이행을 위한 것이다. 모델 분류체계의 각 원소는 이행 분야에 대응하는 링크를 유지한다. 이 링크는 보통 두 루트들 사이의 어소시에이션 이나 어그리게이션에 의해 형성된다. 그것은 분류체계의 하위 클래스에 계승된다.



<그림 3 >Bridge 패턴 구조

Bridge 패턴의 구조적 관점은 오직 분류체계에 있는 하나의 원소에만 집중하고 그것이 Strategy나 State에서도 동일하다.

4. 데이터베이스 설계

저장시스템으로 일반적인 패턴정보가 저장되는 패턴정보 데이터베이스와 UML로 표현된 패턴 구조를 코드화하기 위해 구성 요소인 메타모델들을 저장하는 메타모델 데이터베이스를 구축하기 위해 Visual foxpro6.0을 사용하였다.

4.1 패턴정보 데이터베이스 설계

데이터베이스는 클래스와 상속관계정보를 저장하기 위한 클래스저장테이블과 멤버함수를 저장하기 위한 테이블로 구성한다.

<표 1> 클래스 저장 스키마

field name	Type	width	비 고
ccode	Character	4	클래스 번호
name	Character	20	클래스명
classsource	Memo	4	클래스
super1	Character	30	수퍼클래스명 1
super2	Character	30	수퍼클래스명 2
description	Memo	4	클래스 기능설명
ancestors	Character	40	하위 클래스명

디자인패턴들의 정보는 Gamma의 디자인패턴 표현에 사용되는 형식을 사용한다. strCategory 필드 값으로 사용되는 사용목적에 따라 분류된 패턴은 FDP, LDDP, RDP로 분류되는 strAnalysis 필드의 값과 함께 패싯 검색에서 하나의 패싯으로 사용되어진다. docPartic 필드는 디자인패턴의 구조에 참여하는 클래스나 오브젝트 같은 참여자 명을 기술하게 되며, 코드 생성 시에 추상적인 참여자 명으로 구성된 패턴 구조에서 구체적인 패턴 구조로 바꿀 수 있는 입력 값이 된다. (표 2)는 패턴정보 테이블의 스키마를 나타내며, 각 필드에 대한 세부적인 설명을 기술한다.

<표 2> 패턴정보 스키마

필드명	데이터 형식	설 명
strCategory	varchar	사용목적에 따른 분류 (Fundamental,Creational,Partioning,Structural,Behavioral)
strAnalysis	varchar	검증에 따른 분류(FDP,LDDP,RDP)
strName	varchar	디자인패턴명
docIntent	text	원리와 의도
docMotiv	text	패턴의 추상적 기술의 이해를 돕는 예제
docApplic	text	패턴이 적용될 수 있는 상황
imgStruct	image	패턴구조의 이미지
docPartic	text	디자인패턴에 참여하는 클래스와 오브젝트들
docConseq	text	패턴 사용의 결과
docImple	text	패턴을 구현시 인식해야 하는 함정, 힌트, 기술
docCode	text	C++이나 Java로 패턴을 구현하는 방법을 기술한 코드
docRelated	text	밀접하게 관련된 패턴과 중요한 차이점
Mem_ID	varchar	사용자 ID

4.2 메타모델 데이터베이스 설계

<표 3>에서 <표 7>은 디자인패턴 구조를 구성하는 메타모델 데이터베이스이다. <표 3>은 클래스와 인터페이스 메타모델의 정보를 갖는 테이블 구조이며, 인터페이스는 스테레오타입이 Interface인 클

래스로 기술된다. Visibility 필드는 접근권을 말하며, UML에서는 특정 언어에 독립적인 방법으로 표기한다. 예를 들어, public은 "+", protected는 "#", private는 "-"로 표시한다. Stereotype 필드는 클래스를 한 단계 높은 차원에서 분류하여 붙인 이름을 말하며, 일반적인 스테레오 타입에는 Entity, Boundary, Control, Utility, Exception등이 있다. Constraint 필드는 제약 사항을 말하며, 정형화된 규칙 없이 자유롭게 써주면 되며, 속성뿐만 아니라, 클래스들의 객체간 관계에 존재하는 제약사항도 표시한다[4]. <표 4>와 <표 5>는 <표 3>의 클래스와 인터페이스에 속한 속성과 오퍼레이션 메타모델을 정보를 갖는 테이블의 구조이다.

<표 3> 클래스/인터페이스 메타모델 스키마

필드명	데이터 형식	설 명
ID	varchar	클래스, 인터페이스의 ID
Link_Pattern	varchar	소속된 디자인패턴명
Name	varchar	클래스, 인터페이스명
Visibility	varchar	가시성
Stereotype	varchar	스테레오타입
Constraint	varchar	제약 조건

<표 4> 속성 메타모델 스키마

필드명	데이터 형식	설 명
ID	varchar	속성 ID
Link_Class	varchar	소속된 클래스/인터페이스명
Name	varchar	속성명
Type	varchar	데이터 타입
Value	varchar	속성값
Visibility	varchar	가시성
Stereotype	varchar	스테레오타입
Constraint	varchar	제약 조건

<표 5> 오퍼레이션 메타모델 스키마

필드명	데이터 형식	설 명
ID	varchar	오퍼레이션 ID
Link_Class	varchar	소속된 클래스/인터페이스명
Name	varchar	오퍼레이션명
Type	varchar	데이터 타입
Visibility	varchar	가시성
Stereotype	varchar	스테레오타입
Constraint	varchar	제약 조건

<표 6>은 Dependency, Associations, Generalization, Aggregation 같은 관계 메타모델들의 테이블 구조이며, 관계가 시작되고 끝나는 메타모델들의 ID를 기술한다. <표 7>은 패턴을 구성하는 모든 메타모델들의 표현정보를 저장하는 테이블 구조를 나타내며, 모델이 그려지는 좌측상단의 X, Y좌표, 폭, 높이 같은 좌표 정보와 폰트, 선 색깔과 같은 스타일 정보를 기술한다. <표 8>의 정보를 이용해서 패턴 구조의 시각적 표현이 이루어진다.

<표 7> 관계 메타모델 스키마

필드명	데이터 형식	설 명
ID	varchar	관계 ID
Name	varchar	관계명
Start_Meta	varchar	관계가 시작되는 메타모델
End_Meta	varchar	관계가 끝나는 메타모델

<표 8> 표현정보 스키마

필드명	데이터형식	설 명
ID	varchar	표현하고자 하는 메타모델 ID
Geometry	Integer	좌표
Style	Integer	스타일(Font, LineColor, FillColor정보)

5. 결 론

본 논문에서는 기존의 시스템으로부터 재사용 가능한 객체를 추출하고 추출된 객체들간의 Dependency와 Associations, Generalization, Aggregation을 찾아내고 Compositor, Strategy 등 잘 알려진 추상화된 디자인 패턴들을 추출하는 알고리즘을 제시하였다. 또한 시스템의 성능과 품질 향상에 기여하는 디자인패턴을 공유, 관리할 수 있도록 관련 정보를 저장하는 DB 스키마를 설계하였다. 구현환경은 하부 저장시스템으로 Visual Foxpro6.0을 사용하여 패턴 구조를 구성하는 각 메타모델들을 메타모델 데이터베이스에 저장하고 클래스/인터페이스, 속성, 오퍼레이션, 관계와 같은 메타모델들을 메타모델 데이터베이스의 각 테이블에 매핑시켜 저장할 수 있도록 DB스키마를 설계하였다.

참고문헌

[1] E.Gamma, R.Helm, RJohnson, and J.Vlissides, Design Pattern : Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[2] 김행곤 외 "기존 시스템에서 설계 패턴으로의 재공학 툴에 관한 연구", 한국정보처리학회 논문지 제5권 제9호, pp2334-2343, 1998, 9

[3] Ed Yourdon, "RE-3", American Programmer, Vol.2, No.4, pp.3-10, 1989

[4] J. Suzuki, Y. Yamamoto, "Managing the Software Design Documents with XML", http://www.yy.ics.keio.ac.kr/~suzuki, 1999.