재공학 기반의 클래스 합성을 통한 정련화된 정보 생성에 관한 연구

김행곤[†] · 한은주[††]

요 약

소프트웨어 재공학은 기존 시스템의 유지보수 문제에 대한 해결책으로 많은 연구가 이루어지고 있다. 재공학은 역공학과 순공학을 통해 기존 시스템에 대한 소프트웨어 개발을 의미하며, 기존 시스템의 소프트웨어에서 클래스를 추출하여 시스템의 이해를 높일 뿐만 아니라 소프트웨어 유지보수를 향상시키는데 적용된다. 이를 위해 사용되는 중요한 개념으로 "합성"은 서로 다른 컴퍼넌트들로부터 필요한 기능을 가져와 재구성하는 것이다. 또한, 정보 저장소에 저장된 클래스와 클러스터들은 고수준에서 재사용되기 위해 제공되는 시스템의 주요 컴퍼넌트들과 그들간의 구조적인 관계를 가진다. 이들은 그 각각으로 하나의 아키텍처를 구성하여 향후 동적 정보로 참조된다.

따라서, 기존의 객체지향 원시코드를 논리적으로 표현함으로써 추출기와 검색기, 합성기에 의해 클래스가 생성되며 클래스와 클러스터 정보는 각각 최적화(optimization)를 통해 정련화된 정보를 추출해 낸다. 이러한 정보들은 정보 저장소에 저장되며, 클래스간의 관계성에 의한 클러스터를 하나의 새로운 아키텍처로써 생성한다. 또한, 이 정보는 향후 실행 가능한 코드로써 사용되어진다.

본 논문에서 제시한 툴은 재공학을 기반으로 객체지향 정보를 분석하고 합성 방법론을 수행하여 새로운 정보로써 사용자에게 제시된다. 또한, 새로운 코드와 재구축된 고수준의 합성 클래스는 재사용을 높이고, 기존 소프트웨어에 대한 고수준의 이해성과 유지보수성을 제공한다.

A Study on Refined Information Generation through Classes Composition Based on Reengineering

Haeng-Kon Kim[†] and Eun-Ju Han[††]

ABSTRACT

Softaware reengineering is making various research for solutions against problem of maintain existing systems. Reengineering has a meaning of development of softwares on existing systems through the reverse-engineering and the forward-engineering. It extracts classes from existing system's softwares to increase the comprehension of the system and enhance the maintenability of softwares. Most of the important concepts used in reengineering is ***composition*** that is restructuring of the existing objects from other components. The classes and clusters in storage have structural relationship with system's main components to reuse in the higher level. These are referenced as dynamic informations through structuring an architect for each of them.

The classes are created by extractor, searcher and composer through representing existing object-oriented source code. Each of classes and clusters extract refined informations through optimization. New architecture is created from the cluster based on its classes' relationship in storage. This information can be used as an executable code later on.

In this paper, we propose the tools, it presented by this thesis presents a new information to users through analysing, based on reengineering, Object-Oriented informations and practicing composition methodology. These composite classes will increase reusability and produce higher comprehension information to consist maintainability for existing codes.

[†] 대구효성가톨릭대학교 컴퓨터 공학과
[††] 대구효성가톨릭대학교 대학원 전산통계학과

1. 서 론

소프트웨어 재공학은 기존 시스템에 대한 새로운 기법을 도입한 것으로 시스템 기능을 증진하고 소프트트웨어 유지보수 향상시키는데 적용된다. 또한, 기존 소프트웨어에서 클래스를 추출하고, 이를 통해 새로운 소프트웨어 개발에 적용한다.

재공학은 소프트웨어 재사용을 기반으로 하며, 소프트웨어 합성은 이를 위해 사용되는 중요한 개념으로 서로 다른 컴퍼넌트들로부터 필요한 기능을 가져와 재구성하거나 다른 컴퍼넌트안에서 동작하는 컴퍼넌트를 지칭하는 plug-in을 포함하는 것으로, 주로 개발되는 특정 분야의 부품만을 가지고 있어도 소프트웨어를 개발하는데 충분하며 거의 수정없이 재사용 할 수 있다는 장점을 가진다. 재사용되기 위해 제공되는 컴퍼넌트들과 그들간의 구조적인 관계를 가지는 클래스, 클러스터, 그리고 시스템 클래스들은 그 각각 향후 실행 가능한 정보로 필요하게 된다. 이러한 합성방법으로 내부구조는 알려져 있지 않고 그 기능만이 외부에 알려져 있는 블랙 박스(Black box)와 사용자의 요구에 따라 수정 가능한 화이트 박스(White box)로 구분할 수 있다[1]. 또한, 기존 합성 방법론으로 Wrapper는 개별적 컴퍼넌트들을 바로 적용시키는 것이 아니라 전체 서브시스템을 적용시키는 일반적 캡슐화 기술이다.

이러한 객체지향 패러다임과 일련의 합성방법론들은 객체 지향 소프트웨어의 재사용을 위해 사용하는 중요한 개념이다. 현재 소프트웨어 재사용을 위한 연구 대상은 주로 원시코드의 재사용을 목표로 하고 있으며, 재사용 가능한 프로그램 검색, 이해, 수정 그리고 새로운 부품으로의 합성을 위한 기법의 개발을 위하여 연구가 이루어지고 있다. 또한, 설계 문서의 재사용도 부품의 이해 및 수정에 대한 용이성을 제공하고 있기 때문에 이에 대한 연구도 활발히 진행되고 있다.

따라서, 본 논문에서는 기존의 객체지향 원시코드를 시스템 내용과 구성요소들을 논리적으로 표현함으로써 클래스를 추출하여 정련된 새로운 클래스를 기존의 클래스 라이브러리에 저장한다. 저장된 정보는 사용자 인터페이스를 통해 클래스간의 관계를 통한 클러스터를 새로운 클래스로 정의하므로 향후 실행 가능한 코드로써 사용되어진다. 또한, 추출기와 검색기, 합성기에서 생성된 클래스와 클러스터 정보는 각각 최적화(Optimization)를 통해 정련화된 정보를 추출해 낸다.

본 논문에서 제시한 툴은 재사용을 위해 재공학 기반으로 각각의 클래스를 합성함으로 사용자에게 새로운 정보를 제시된다. 또한, 객체지향 정보를 분석하여 합성 방법론을 수행하고, 재구축을 위해 새로운 코드와 기존의 정보를 합성함으로 재사용성을 높이고 기존 소프트웨어에 대한 고수준의 이해성과 유지보수성을 제공한다.

2. 관련연구

2.1 재공학

소프트웨어의 유지보수 처리를 개선하고 새로운 기법과 툴을 지원하므로써 기존 시스템을 향상시키는 재공학은 다음(그림 1)과 같이 역공학과 순공학을 통해 기존 시스템을 개발하는 것이다. 순공학은 상위 추상화와 논리적 구현에서부터 시스템의 물리적 구현으로 옮겨지는 기존의 과정이며 이들의 요구사항으로부터 설계, 구현을 통해 순차적으로 처리된다. 이와 반대로 역공학은 컴퍼넌트와 그들의 상호관계, 다른 형태의 시스템이나 추상화의 더 높은 수준에서 구체적으로 나타내기 위해 시스템을 분석하는 과정이며 어떤 추상화 수준이나 생명주기의 어느 단계에서도 역공학은 실행 가능하며 재구성 및 재설계가 수반된다.

2.2 합성

기존에 잘 정의 · 설계 그리고 구현된 컴퍼넌트들로부터 필요한 기능을 가져와 재구성하는 합성에서 상속 개념이 재사용을 지원하는 주요 특성임에는 틀림없지만 클래스의 속성과 인터페이스 뿐 아니라 구

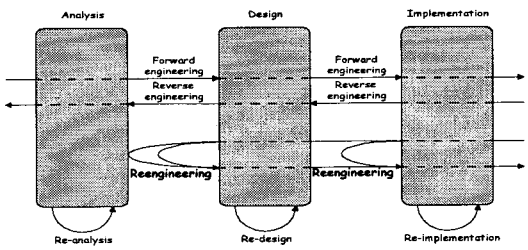


그림 1. 재공학 모델

현부분까지도 완전히 이해해야 한다. 그러나 재사용이 이루어지는 과정에서 비교해 보면, 주로 개발되는 특정한 분야의 부품만 소유하여도 소프트웨어 개발에 충분하며, 수정없이 재사용 할 수 있는 경우가 오히려 많다. 이러한 재사용을 목표로, 객체지향 방법론에서는 다른 객체들로부터 필요한 기능을 가져와 재구성하는 객체의 합성을 지원한다. 다음 (그림 2)는 클래스 합성을 도식화한 것이다[2].

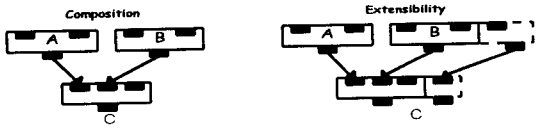


그림 2. 클래스 합성

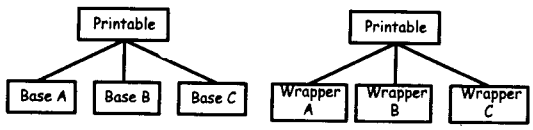
기존의 합성 방법론으로 Wrapper는 개별적 컴퍼넌트들을 바로 적용시키는 것이 아니라 전체 서브시스템을 적용시키는 일반적 캡슐화 기술이다[3].

각각 다른 컴퍼넌트 A, B, C를 가지고 응용프로그램을 작성한다면 컴퍼넌트들이 제공하는 인터페이스는 (그림 3)과 같다.

```
type BaseA {
            method print();
            methods mbasea1, mbasea2, ...
}

type SubA : Base A {
            methods msuba1, msuba2, ...
}

type BaseB {
            method printPart1();
            method printPart2();
            methods mbaseb1, mbaseb2, ...
}

type SubB : BaseB {
                  methods msubb1, msubb2, ...
}
type BaseC{
            method drucke();
            methods mbasec1, mbasec2, ...
}

type SubC : BaseC {
                  methods msubc1, msubc2, ...
}
```

그림 3. 컴퍼넌트 A, B, C의 인터페이스

이 Base A, B, C로부터 유도된 객체들을 통합하고, 어떤 객체를 가져와 화면에 출력하는 메소드를 작성하려면 (그림 4)의 (a)와 같이 공통적인 상위 타입 printable을 생성해야한다, 그러나, 서로 다른 기능을 요구하기 때문에 공통적인 수퍼 타입으로 유도되지 못했다. 따라서, (b)와 같이 합성을 사용하여 포인터가 각각 기본 타입의 객체를 포함하는 세 가지 Wrapper를 정의한다. 세 개의 컴퍼넌트 객체를 서브타입 관계에 배열하는 대신에, Wrapper를 적절한 관계에 배열한다. 예를 들면 (그림 5)처럼 Wrapper A는 Base A객체를 포함할 것이다.



(a) (b)

그림 4. 컴퍼넌트와 Wrapper 계층구조

```
type Wrapper A : Printable {
                        B : Base A;
                        method print();
}
```

그림 5. Wrapper의 예

2.3 재사용을 위한 기존 방법론

재사용에 있어서 가장 중요한 요소는 얼마만큼 재사용 될 수 있는가하는 점이다. 이것은 재사용 가능한 컴퍼넌트가 생성되는데 드는 비용에 비해 얼마만큼 그 가치를 얻을 수 있는가를 평가하는 척도이다. 따라서 재사용성이 높다는 것은 적은 비용으로 재사용을 통해 얻을 수 있는 효과가 크다는 것을 의미한다[4]. 소프트웨어 재사용은 많은 형태를 가질 수 있지만, 본 논문에서는 다음 <표 1>과 같이 재사용 입자에 따라 크게 코드, 객체, 설계 패턴 그리고 서브젝트로 분류한다.

현재 소프트웨어 재사용을 위한 연구 대상은 주로 원시코드의 재사용을 목표로 하고 있으며, 재사용 가능한 프로그램 검색, 이해, 수정 그리고 새로운 부품으로의 합성을 위한 기법의 개발을 위하여 연구가 이루어지고 있다. 또한, 설계 문서의 재사용도 부품의 이해 및 수정에 대한 용이성을 제공하고 있기 때문에 이에 대한 연구도 활발히 진행되고 있다.

따라서, 본 논문에서는 기존의 객체지향 원시코드를 시스템 내용과 구성요소들을 논리적으로 표현함

표 1. 기존 재사용 방법

	코드	객체	설계 패턴	서브젝트
적용 단계	코딩	코딩	설계	설계+코딩
비용	코딩 수정	객체 수정 or 상속	패턴 적용	거의 요구되지 않음
적용 범위	좁음	좁을수록 효과적	넓음	특정 영역에 한정
일치성	해당 코드 찾기 어려움	해당 객체 찾기 어려움	해당 패턴을 찾기 쉬움	해당 부품을 찾기 쉬움

으로써 클래스를 추출하여 정련된 새로운 클래스를 기존의 클래스 라이브러리에 저장한다. 저장된 정보는 사용자 인터페이스를 통해 클래스간의 관계를 통한 클러스터를 새로운 클래스로 정의하므로 향후 실행 가능한 코드로써 사용되어진다. 또한, 추출기와 검색기, 합성기에서 생성된 클래스와 클러스터 정보는 각각 최적화(optimization)를 통해 정련화된 정보를 추출해 낸다.

3. 클래스 합성 방법론

전체 서브시스템을 적용시키는 일반적 캡슐화 기술인 Wrapper는 기능적인 부분에서는 사용할 수 없으며 표준화하는데 비용이 많이 들고 오버헤더가 많은 문제점을 가지고 있다.

본 논문에서는 재공학 기반의 객체지향 분석으로 입력된 소스코드에서 클래스 추출 및 검색, 이해, 합성의 3 단계를 통한 합성 방법론을 제시한다.

1단계 : 클래스 추출 및 검색

입력된 소스코드는 문법적으로 최소의 의미있는 토큰단위로 어휘분석을 하고 구문분석에서 정의하는 규칙을 통해서 요구되는 클래스를 추출할 수 있다. 클래스 추출시에는 국부 클래스 추출, 전역 클래스 추출, 매개변수에서의 클래스 추출, 메소드 추출로 나누어 진다. 또한, 원하는 클래스를 찾기 위해 종류, 대상, 기능, 응용영역, 사용언어의 5개 패싯으로 구성된 용어를 선택, 조합하여 사용자가 원하는 클래스의 후보 클래스를 검색하게 된다. 후보 클래스들이 검색되면 사용자는 (그림 6)과 같이 클래스 이름, 메소드, 속성, 수퍼 클래스, 그리고 현재 클래스가 구성하고 있는 합성 클래스를 표현하는 합성 항목들을 가진 클래스 정보 리스트를 보고 요구에 맞는 클래스를 선택한다.

Class Name	Method	Attribute	SuperClass	Composition

그림 6. 클래스 정보 리스트

2단계 : 추출 · 검색된 정보 이해

클래스 추출 및 검색을 통해 선택된 클래스는 소프트웨어 컴퍼넌트를 재사용 할 때 원하는 정보를 찾고 이해하는 것이 중요하다. 따라서, 다른 컴퍼넌트와 쉽게 구별될 수 있도록 그 특성이 표현되어야하며, 원하는 재사용 컴퍼넌트를 찾지 못할 경우 가장 유사한 컴퍼넌트를 제공할 수 있도록 컴퍼넌트간의 관련성도 정의되어 있어야 한다. 컴퍼넌트의 특성과 관련성의 표현은 컴퍼넌트를 어떻게 분류하느냐에 따라 크게 달라지므로 본 논문에서는 분류하려는 컴퍼넌트들의 공통적인 특성을 모아서 패싯을 구성한 후 각 패싯에 적합한 부품들의 내용을 선택하고 이들을 조합하는 분류방법을 사용한다.

3단계 : 합성

클래스를 합성할 때, 설계자들은 클래스와 그들간의 관계를 검사해야하고, 어떻게 결합하는지를 결정해야 한다. 따라서, 본 논문에서는 오퍼레이션, 클래스, 인스턴스 변수 일치 그리고 메소드 조합의 4가지 문제들을 고려하여 전개한다.

1) 합성 선언

C++ 클래스에서는 정보 은닉을 위해 클래스 내부의 변수와 메소드를 해당 클래스의 유효범위에서만 엑세스 가능한 private와 프로그램 어디에서나 엑세스 가능한 public으로 선언한다. 합성은 하나의 목적에 밀접하게 관련되어 있는 객체들을 추상화한 것이므로 클래스들의 복잡한 내부 구조는 알 필요 없이, 프로그램의 나머지 부분에서 이용할 수 있는 public으로 선언된 멤버만 사용하며 (그림 7)과 같은 형태로 정의할 수 있다.

다음(그림 8)은 합성 선언을 BNF(Backus Naur Form)로 표현한 것이며, (그림 9)처럼 구문 다이어

```
composition 합성명 {
  class 클래스명 {
                변수타입 변수명;
                        .
                        .
                오퍼레이션 타입 오퍼레이션명(파라메타1,...,파라메타n);
                        .
                        .
                };
                        .
                        .
  mapping {
        오퍼레이션명(파라메타1,...,파라메타n)
        =>대응하는 기존클래스의 오퍼레이션명(파라메타1,...,파라메타n);
                        .
                        .
          };
};
```

그림 7. 선언 형식

```
<composition-specifier> ::= <composition-head>'{'{<composition-member-list>}'}';'
<composition-head> ::= <composition-key><identifier>
<composition-key> ::= 'composition'
<composition-member-list> ::= <class-specifier><mapping-specifier>
<class-specifier> ::= <class-head>'{'{<class-member-list>}'}';' |
                      <class-specifier><class-head>'{'{<class-member-list>}'}';'
<class-head> ::= <class-key><identifier>
<class-key> ::= 'class'
<class-member-list> ::= <class-ivar-list><class-operation-list>
<class-ivar-list> ::= <simple-type-name><identifier>';' |
                      <class-ivar-list><simple-type-name><identifier>';'
<class-operation-list> ::= <function-definition>';' |
```

그림 8. BNF 표현

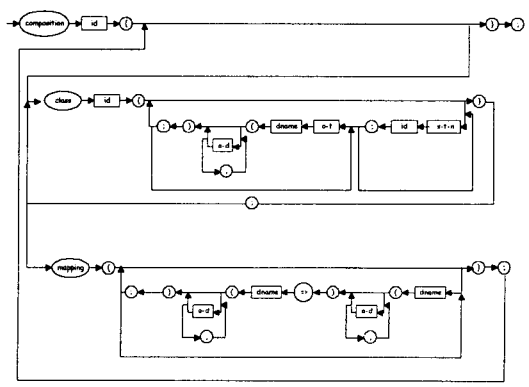


그림 9. 구문 다이어그램

그램(문법 흐름도)으로 도식화함으로써 구문 구조를 쉽게 이해할 수 있도록 했다.

사각형은 nonterminal 심벌의 이름, 타원 안에는 terminal이 표기되었으며 지시선을 따라 움직이는 경로는 문법적으로 올바른 구문 형태를 정의하고 있다. 이 구문 다이어그램이 완전하기 위해서는 사각형안에 기술된 것에 대한 정의가 있어야 하지만 본 논문에서는 최소한의 이해를 돕기위한 부분만 다루었다.

2) 합성 생성

객체지향 파라다임에 기존 클래스들을 합성하여 확장된 기능의 재사용 가능한 클래스, 클러스터 그리고 아키텍처를 생성하도록 하는 합성 생성규칙을 제시한다. 이 규칙은 기존 클래스들을 합성하여 새로운 클래스를 생성하는데도 적용되며 선언절, 생성절, 생성규칙을 통해 유도된다.

● 선언절

오퍼레이션, 클래스, 인스턴스 변수와 같은 엔티티 선언의 집합이다. 이러한 선언은 문맥을 설정하는데 사용되며, 프로그래머를 위한 문법으로는 바람직하지 못하지만 선언의 순서가 고정된 모델에서는 유용하다.

표 2. 선언절의 의미

항 목	선 언 절	의 미
합성	C=composition C : 합성 식별자	C는 합성
클래스	C.c=class c : 클래스 식별자	C.c는 클래스
인스턴스 변수	C.c.v=ivar of type t v : 인스턴스 변수 식별자 t : 타입	C.c.v는 t타입의 인스턴스 변수
오퍼레이션	C.c.o=operation with parameter p o : 오퍼레이션 식별자 p : 파라메타	C.c.o는 파라메타 p를 갖는 오퍼레이션
매핑	C.c.o mapping C'.c'.o'	C.c.o의 원시코드는 C'.o'.c'의 원시코드에 매핑

● 생성절

생성규칙의 의미론적 기초 형태로 클래스 생성을 명세하는데 직접 사용되는 것이 아니라 좀 더 높은 수준의 규칙을 정의하는데 사용된다.

생성절은 오퍼레이션 파라메타나 인스턴스 변수 타입과 같은 세부사항들이 직접 명세되지 않고 합성되고자 하는 클래스나 합성 선언내의 세부사항에 적용되는 생성자로 명세된다. 매핑관계처럼 세부사항

이 명세되지 않는 절은 선언절과 동일하다. 결과적으로 생성절은 다음(그림 10)과 같다.

C = composition
C.c = class
C.c.v = ivar of type C_t(q)
C.c.o = operation with parameter C_p(q)
C.c.o = mapping C'.c'.o'

그림 10. 생성절

● 생성규칙

생성규칙도 클래스나 합성의 선언부에 적용되며 이러한 선언부는 이름을 선언으로 바꾸는데 사용된다. 그리고, 생성절의 집합은 생성규칙 평가의 결과로써 입력되는 클래스나 합성의 선언부에 적용되어 새로운 클래스를 생성한다. 다음은 앞에서 제시한 4가지 문제를 고려한 생성규칙과 클래스와 관련된 집합 생성규칙을 제시하겠다.

① instance variable ; 입력되는 인스턴스 변수를 합성하여 새로운 클래스의 인스턴스 변수로 생성한다.

$$I_v\,[C_t](n,q)=\{\text{“}n=\textit{ivar of type } C_t(\textit{types}(q))\text{”}\} \quad \text{(규칙 1)}$$

② operation ; 파라메타 생성자를 명세하는 일반 파라메타를 가진다.

$$O\,[C_p](n,q)=\{\text{“}n=\textit{operation with parameter } C_p(\textit{parameter}(q))\text{”}\} \quad \text{(규칙 2)}$$

③ class ; 인스턴스 변수와 오퍼레이션을 생성하는 방법을 명세한다.

$$C\,[S_v, S_o](n,q)=\{\text{“}n=\textit{class}\text{”}\} \cup S_v(n,\textit{instvar}(q)) \cup S_o(n,\textit{operation}(q)) \quad \text{(규칙 3)}$$

④ mapping ; 오퍼레이션의 집합으로 합성 클래스에서 어떤 클래스의 오퍼레이션을 호출할 때 실행되는 메소드의 원시코드가 저장되어 있는 클래스의 메소드를 명세한다.

$$M\,(n,q)=\{\text{“}n\ \textit{mapping}\ q\text{”}\} \quad \text{(규칙 4)}$$

⑤ aggregation ; 선언 집합의 생성을 처리하는데 사용된다. 이것은 생성하고자 하는 집합의 요소와 그들의 생성 방법을 명세한다.

$$S_{name}[R](n,Q)=\bigcup_x R(n.\ x,\langle x_1,x_2,\cdots,x_k\rangle) \quad \text{(규칙 5)}$$

추출기와 검색기, 합성기를 통해서 생성된 클래스들은 시스템에서 요구하는 적절한 정보는 아니다. 따라서 주어진 입력 클래스와 의미적으로 동등하면서 좀더 효율적인 클래스로 바꿈으로써 유지보수 생산성을 증가시키고 정련화된 정보로 정보 저장소에 저장시킬 수 있다.

4. 사례 연구

본 절에서는 앞에서 제안한 합성 생성규칙을 통해 합성 클래스가 어떻게 생성되는지를 예제에 적용시켜 본다. 입력 클래스 CCircle과 CSquare은 그래픽에서 가장 기본적으로 제공하는 도형을 드로잉(drawing)하는 클래스이다. 두 클래스는 유사한 속성을 가지고 있으며(그림 11), 이 두 클래스에 생성규칙을 적용하여 합성 클래스 CCircle_CSquare를 생성해 보자.

```
class CCircle
{
  public:
    double m_dRadius;
    int m_xCoord;
    int m_yCoord;
    CCircle();
    double GetRadius();
    double GetArea();
    void GetCenter( int *x, int *y);
    void SetCenter( int *x, int *y);
    void Draw();
};
```

```
class CSquare
{
  public:
    double m_dLength;
    int m_xCoord;
    int m_yCoord;
    CSquare();
    double GetSideLength();
    double GetArea();
    void GetCenter( int *x, int *y);
    void SetCenter( int *x, int *y);
    void Draw();
};
```

그림 11. 입력 클래스

우선, 클래스 CCircle의 인스턴스 변수(instance variable) m_dRadius를 합성 클래스 CCircle_CSquare의 클래스 CCircle의 m_dRadius를 생성한다.

I_v[identity](CCircle_CSquare.CCircle.m_dRadius)
= {" CCircle_CSquare.CCircle.m_dRadius
= ivar of type identity (types(.CCircle.m_dRadius))" }
= {" CCircle_CSquare.CCircle.m_dRadius= ivar of type double" }

클래스 CCircle의 오퍼레이션(operation) GetCenter를 합성 클래스 CCircle_CSquare의 클래스 CCircle의 오퍼레이션 GetCenter로 생성한다.

O[identity](CCircle_CSquare.CCircle.GetCenter, .CCircle.GetCenter)
={" CCircle_CSquare.CCircle.GetCenter
= operation with parameter identity(parameter(.CCircle.GetCenter))" }
={" CCircle_CSquare.CCircle.GetCenter
= operation with parameter int *x, int *y" }

클래스 CCircle을 합성 클래스 CCircle_CSquare의 클래스(class) CCircle로 생성한다.

C [S_v, S_o](CCircle_CSquare.CCircle, .CCircle)
= {" CCircle_CSquare.CCircle = Class" }
∪ S_v (CCircle_CSquare.CCircle, instVar(.CCircle))
∪ S_o (CCircle_CSquare.CCircle, operation(.CCircle))
= {" CCircle_CSquare.CCircle = Class" },
{" CCircle_CSquare.CCircle.m_dRadius= ivar of type double",
" CCircle_CSquare.CCircle.m_xCoord = ivar of type int",
" CCircle_CSquare.CCircle.m_yCoord= ivar of type int" },
{" CCircle_CSquare.CCircle = operation",
" CCircle_CSquare.CCircle.GetRadius = operation",
" CCircle_CSquare.CCircle.GetArea = operation",
" CCircle_CSquare.CCircle.GetCenter
= operation with parameter int *x, int *y",
" CCircle_CSquare.CCircle.SetCenter
= operation with parameter int *x, int *y",
" CCircle_CSquare.CCircle.Draw = operation" }

오퍼레이션 CCircle_CSquare.CCircle .GetCenter에서 오퍼레이션 .CCircle.GetCenter로의 매핑(mapping)을 생성한다.

M(CCircle_CSquare.CCircle.GetCenter, .CCircle.GetCenter)
={CCircle_CSquare.CCircle.GetCenter Mapping .CCircle.GetCenter}

클래스 CCircle의 모든 인스턴스 변수들을 합성클래스 CCircle_CSquare의 클래스 CCircle의 인스턴스 변수들로 생성하기 위해 클래스 CCircle에 (규칙 5)를 적용한다.

S_v(CCircle_CSquare.CCircle, instvar(.CCircle))
= S_v (CCircle_CSquare.CCircle,
(".CCircle.m_dRadius = ivar of type double",
".CCircle.m_xCoord = ivar of type int",
".CCircle.m_yCoord = ivar of type int"))
= $\bigcup_x$ (CCircle_CSquare.CCircle.x(" m_dRadius= ivar of type double",
" m_xCoord = ivar of type int",
" m_yCoord = ivar of type int"))
={" CCircle_CSquare.CCircle.m_dRadius= ivar of type double",
" CCircle_CSquare.CCircle.m_xCoord= ivar of type int",
" CCircle_CSquare.CCircle.m_yCoord= ivar of type int" }

위와 같은 방법으로 클래스 CCircle과 CSquare에 생성규칙을 적용하면, (그림 12)와 같이 새로운 합성 클래스 CCircle_CSquare가 생성된다.

```
Composition CCircle_CSquare{
class Ccircle{
   public
     double m_dRadius;
     int m_xCoord;
     int m_yCoord;
     CCircle();
     double GetRadius();
     double GetArea();
     void GetCenter( int *x, int *y);
     void SetCenter( int *x, int *y);
     void Draw();
};
class CSquare{
    public
      double m_dLength;
      int m_xCoord;
      int m_yCoord;
      CSquare();
      double GetSideLength();
      double GetArea();
      void GetCenter( int *x, int *y);
      void SetCenter( int *x, int *y);
      void Draw();
};

mapping{
   CCircle_CSquare.CCircle.CCircle() => .CCircle.CCircle();
   CCircle_CSquare.CCircle.GetRadius() => .CCircle.GetRadius();
   CCircle_CSquare.CCircle.GetArea() => .CCircle.GetArea();
   CCircle_CSquare.CCircle.GetCenter(int *x, int *y) => .CCircle.GetCenter(int *x, int *y);
   CCircle_CSquare.CCircle.SetCenter(int *x, int *y) => .CCircle.SetCenter(int *x, int *y);
   CCircle_CSquare.CCircle.Draw() => .CCircle.Draw();
   CCircle_CSquare.CSqaure.CSqaure() => .CSqaure.CSqaure();
   CCircle_CSquare.CSqaure.GetSideLength() => .CSqaure.GetSideLength();
   CCircle_CSquare.CSqaure.GetArea() => .CSqaure.GetArea();
   CCircle_CSquare.CSqaure.GetCenter(int *x, int *y) => .CSqaure.GetCenter(int *x, int *y);
   CCircle_CSquare.CSqaure.SetCenter(int *x, int *y) => .CSqaure.SetCenter(int *x, int *y);
   CCircle_CSquare.CSqaure.Draw() => .CSqaure.Draw();
```

그림 12. 생성 클래스

5. 시스템 설계 및 구현

본 논문에서 제안된 툴의 구조는 (그림 13)과 같이 추출기, 검색기, 합성기로 구성되어 있으며 간략히 설명하면 다음과 같다.

● 추출기 & 검색기 : 입력된 소스코드는 문법적으로 최소의 의미있는 토큰단위로 어휘분석을 하고 구문분석에서 정의하는 규칙을 통해서 요구되는 클래스를 추출할 수 있다. 클래스 추출시에는 국부 클래스 추출, 전역 클래스 추출, 매개변수에서의 클래스 추출, 메소드 추출로 나누어진다. 또한, 원하는 클래스를 찾기 위해 영역, 대상, 기능, 응용영역, 사용언어의 5개 패싯으로 구성된 용어를 선택, 조합하여 사용자가 원하는 클래스의 후보 클래스를 검색하게 된다. 후보 클래스들이 검색되면 사용자는 클래스 이름, 메소드, 속성, 수퍼 클래스, 그리고 현재 클래스가 구성하고 있는 합성 클래스를 표현하는 합성 항목들을 가진 클래스 정보 리스트를 보고 요구에 맞는 클래스를 선택한다.

● 합성기 : 선택된 클래스를 합성규칙에 의해 새로운 정보를 생성하는 시스템으로 객체지향 파라다임에 기존 클래스들을 합성하여 확장된 기능의 재사용 가능한 클래스, 클러스터 그리고 아키텍쳐를 생성하도록 하는 합성 생성규칙을 적용하는 시스템이다. 이 규칙은 기존 클래스들을 합성하여 새로운 클래스를 생성하는데도 적용되며 선언절, 생성절, 생성규칙을 통해 유도된다.

또한, 불필요한 코드를 식별하고 정련화된 정보를 생성하기 위한 시스템으로 추출기와 검색기, 합성기를 통해서 생성된 클래스들은 시스템에서 요구하는 적절한 정보는 아니므로 주어진 입력 클래스와 의미적으로 동등하면서 좀더 효율적인 클래스로 바꿈으로써 유지보수 생산성을 증가시키고 정련화된 정보로 정보 저장소에 저장시킬 수 있다.

● 정보 저장소 : 원시 소스코드와 관련정보, 패싯, 클래스 정보리스트, 합성선언 및 기존의 정보, 추출된 정보, 합성클래스 등을 저장한다. 또한, 재사용 될 부품의 분류상태를 잘 나타내고 검색을 용이하게 할 수 있도록 한다. 이 정보 저장을 위해 부품의 버전을 관리하기 위한 버전관리와, 등록, 검색, 수정, 삭제의 기본적인 클래스 라이브러리 명령을 가지고 있다.

제안된 툴은 MSVC 5.0환경에서 구현한 실행 예

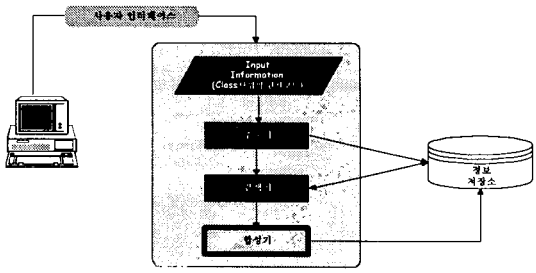


그림 13. 구성도

로써 기본 메뉴는 일반적으로 요구하는 항목들과 같으며, (그림 14)에서 Q (Query)는 원하는 클래스를 찾기 위해 영역, 대상, 기능, 응용영역, 사용언어 5개의 패싯으로 구성된 용어를 선택, 조합하여 사용자가 원하는 클래스의 후보 클래스를 검색하게 된다. 검색된 클래스 리스트들은 S (Selection)를 통해 합성하기 위한 클래스 즉, 앞의 적용 예의 CCircle과 CSquare를 선택하며(그림 15), 사용자에게 최소한의 정보를 명세하는 메시지 윈도우를 포함하고 있다.

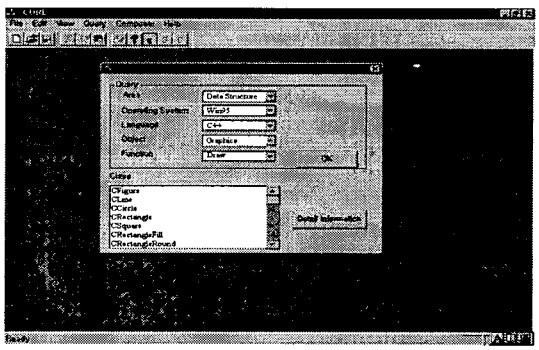


그림 14. 클래스 검색을 위한 패싯 분류

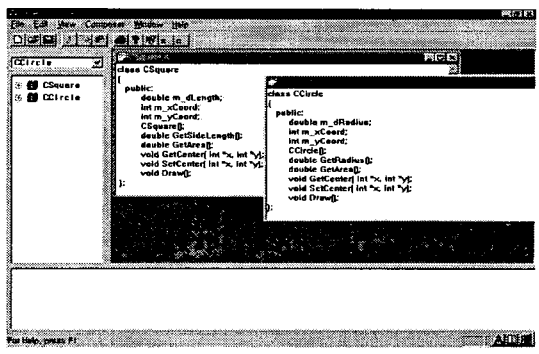


그림 15. 선택된 클래스 소스 코드

또한 (Composer)는 선택된 두 개의 클래스를 합성하는 기능을 하며 (그림 16)에서 보듯이 합성 클래스 CCircle_CSquare가 결과로 나타난다.

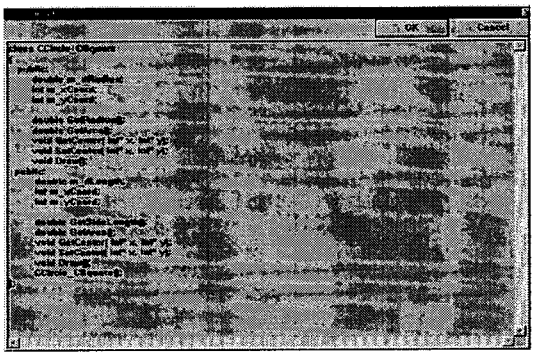


그림 16. 합성된 소스코드

6. 결론 및 향후 연구방향

재공학은 역공학과 순공학을 통해 기존 시스템에 대한 소프트웨어 개발의 의미를 갖는다. 이를 위해 사용되는 중요한 개념으로서의 합성은 서로 다른 컴퍼넌트들로부터 필요한 기능을 가져와 재구성하는 것이다. 또한, 정보 저장소에 저장된 클래스, 클러스터, 그리고 시스템 클래스들은 고수준에서 재사용 되기 위해 제공되는 시스템의 주요 컴퍼넌트들과 그들 간의 구조적인 관계를 가진다. 이들은 그 각각으로 하나의 아키텍쳐를 구성함으로써 향후 실행 가능한 정보로써 요구되어진다.

따라서, 본 논문에서는 기존의 객체지향 원시코드를 논리적으로 표현함으로써 추출기와 검색기, 합성기에 의해 클래스가 생성되며 클래스와 클러스터 정보는 각각 최적화를 통해 정련화된 정보를 추출해 낸다. 이러한 정보들은 기존의 클래스 라이브러리에 저장되며, 클래스간의 관계성에 의한 클러스터를 하나의 새로운 아키텍쳐로써 생성한다. 또한, 이 정보가 향후 실행 가능한 코드로써 사용될 수 있도록 했다.

제안한 툴은 재공학을 기반으로 객체지향 정보를 분석하고 합성 방법론을 수행하여 새로운 정보를 사용자에게 제시한다. 또한, 새로운 코드와 시스템 상호구조의 재구축된 고수준의 합성 클래스는 재사용을 높이고, 기존 소프트웨어에 대한 고수준의 이해성과 유지보수성을 제공한다.

향후 연구방향으로는 본 논문에서 제시한 합성 방법론에서 고려하지 않은 생성규칙을 유도하고, 자동화된 CASE 툴로의 접근과 설계부분에서 고려되어야될 설계패턴 생성을 고려한다. 또한, 모든 객체지향 원시코드에서 클래스 뿐 아니라 그에 상응하는 컴포넌트를(모듈, 프로세스, 프로시쥬어) 추출할 수 있는 자동화된 시스템의 구성 및 툴의 개발과 클라이언트/서버 환경을 제공할 수 있도록 연구되어야 한다.

참 고 문 헌

[1] Jan Bosch, "Adapting Object-Oriented Component", *Proceedings of the Second International WCOP '97*, pp. 13-21, September, 1997.

[2] Oscar Nierstrasz & Demis Tsichritzis, *Object-Oriented Software Composition*, Prentice Hall, 1995.

[3] John Lakos, *Large-Scale C++ Software Design*, Addison Wesley, 1996.

[4] 남혜영, "객체지향 시스템에서 서브젝트 생성에 관한 연구", 대구효성가톨릭대학교 대학원 석사학위 논문, 1997.

[5] 김치수, "추상 자료형을 기반으로한 컴퍼넌트 합성과 릴레이션쉽에 관한 연구", 중앙대학교 대학원 박사학위논문. 1996.

[6] Keng-Jyi Chen, P.J. Lee, "On the Study of Software Reuse Using Reusable C++ Component", *J. System Software*, pp. 19-36, 1993.

[7] Sun-Myung Hwang, Eun-Ju Han, Hye-Young Nam, "Construction and Composition of Reusable Subject", *HNTTI' 96*, pp. 170-177, 1996.

[8] 최하정, "설계 정보 복구와 객체지향 구조의 논리적 분석을 통한 재구성 툴 설계 및 구현", 효성여자대학교 대학원 석사학위 논문, 1996.

[9] 현창문, "개념인식과 정제를 통한 소프트웨어 재공학 툴 구축에 대한 연구", 효성여자대학교 대학원 석사학위 논문. 1995.

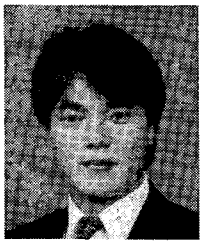
[10] Martin Griss, Ted Biggerstaff, Salie Henry, Ivar Jacobson, Doug Lea, Will Tracz, "Systematic Software Reuse", *OOPSLA' 95*, pp. 281-282, 1995.

[11] 김갑수, 신영길, "소프트웨어 재사용을 위한 C++ 클래스 계층구조 변형 방법", 한국정보과학회논문지, 22권 1호, pp. 88-96, 1995

[12] Harold Ossher, William Harrison, "Combination of Inheritance Hierarchies", *OOPSLA'92*, pp. 25-40, 1992.

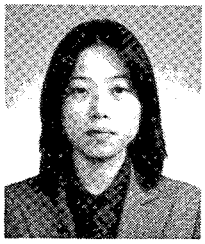
[13] Urs Holzle, "Integrating Independently Developed Components in Object-Oriented Language", *JOOP*, pp. 36-55.

[14] Magraet A. Ellis, Bjarne Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, 1991.



김 행 곤

1985년 중앙대학교 전자계산학과 졸업(이학사)
1987년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1978~1979년 미 항공우주국 객원 연구원
1987~1990년 한국전기통신공사 전임 연구원
1988~현재 대구효성가톨릭대학교 컴퓨터 공학과 부교수
관심분야 : 객체지향시스템 설계, 사용자 인터페이스, 소프트웨어 제공학, 유지보수 자동화 툴, CASE



한 은 주

1994년 경북산업대학교 전자계산학과 졸업(공학사)
1996년 대구효성가톨릭대학교 대학원 전산통계학과 전자계산학(이학석사)
현재 대구효성가톨릭대학교 대학원 전산통계학과 전자계산학 전공 박사과정 수료
관심분야 : 객체지향 프로그래밍, 프레임워크, CBSE, 멀티미디어 시스템, 전자상거래