韓國 OA學會 論文誌
第3卷 第3號, 1998. 10.

98-3-3-2-2

프로그램 변환 및 클래스추출기법의 설계

진영배*

Program Transformation and Design of Class Extraction Technique

Young-Bae Jin*

요 약

원시 코드로부터 이 언어들을 충분히 다시 표현할 수 있는 공통의 언어 즉 메타 언어를 설계하여 역공학 시스템의 내부 표현으로 사용하였다. 이것을 입력으로 하여 변수와 함수 사이의 관계에 기반한 유사도 공식을 사용하여 가장 적절한 클래스를 추출하고, 추출된 클래스의 가시성을 자동적으로 분류한다.

Abstract

This paper is to suggest designing and implementing tool by transforming source code to meta language. We use formula, which is based on relationship between variables and functions, in class extraction and restructuring method in other to extract most appropriate class.

* 충청대학 사무자동화과 부교수
** 본 논문은 1998년도 충청대학 산학협동연구진흥비 지원에 의한것임
논문접수:98.8.28. 심사완료:98.9.10.

Ⅰ. 서론

소프트웨어 유지 보수 위기[5,11,8]는 기존 구조적 언어들이 구조적 설계와 단계적 정련으로 대표되는 기능적 설계 방법을 따르고 있으며, 자료 추상화와 정보 은폐가 효과적이지 못할 뿐 아니라 문제 공간의 변화에 적응하기 어렵기 때문에 더욱 심화되고 있다. 이러한 위기 상황을 극복하기 위한 새로운 방법으로 등장한 것이 객체 지향 패러다임이다[6,7]. 이는 각각의 소프트웨어 모듈을 하나의 객체로 만들어 두었다가 새로운 소프트웨어를 개발할 때 재 사용할 수 있도록 하는 것으로 소프트웨어의 생산성을 향상시킬 수 있을 뿐만 아니라 시스템을 유지 보수하기가 용이해지는 등 많은 장점이 있기 때문에 객체 지향 소프트웨어 개발이라는 패러다임이 일반화 되어가고 있다[3,1]. 객체 지향 시스템의 개발은 객체 지향 분석 및 설계를 전제로 하고 개발될 때 효율적이다[10,2]. 따라서, 기존의 소프트웨어를 객체 지향 소프트웨어로 재구성하기 위한 연구가 진행되고 있으며, 특히 유지 보수가 빈번하게 이루어지는 시스템의 경우 객체 지향 시스템으로 재구성이 요구된다. 본 논문에서는 실무에서 많이 사용되는 4세대언어를 모델로하여 이로부터 C언어 형태인 메타언어로 변환하고, 변환된 시스템을 객체 지향 시스템으로 재구성하기 위하여 클래스를 자동으로 추출, 분류한다.

Ⅱ. 프로그램 변환

기존의 작성된 프로그램을 가지고 역공학을 하는 작업은 시스템 설계에 검증 및 효율적 유지 보수를 위해서 필수적이라 할 수 있다. 이러한 4세대 언어들을 충분히 다시 표현할 수 있는 공통의 언어를 설계하여 역공학 시스템의 내부 표현으로 사용함으로써 수많은 4세대 언어와 도구들에 대하여 능동적인 적응성을 갖도록 해야 한다.

2.1 메타언어의 설계

메타언어는 4세대 언어로 작성된 프로그램을 충분히 만족시킬 수 있는 구문을 갖고 있어야 할뿐만 아니라 역공학 작업에 필요한 정보를 갖고 있어야 한다. 따라서 3세대 언어 부분은 C-형태의 구문을 사용하고 데이터베이스 관리 부분은 SQL-형태의 구문을 적용하였다.

스키마 정의를 위한 구문은 함수 선언이나 자료 선언문과 동등하게 취급한다. 그리고 스키마 정의에 필수적인 기초 테이블 정의(base table definition). 뷰 정의(view definition,) 프리빌리지 정의(privilege definition)를 허용하였다. 〈표 1〉은 데이터베이스 정의를 위한 메타언어의 EBNF이다.

표. 1. 데이터베이스 정의를 위한 메타언어 EBNF
Table. 1. Meta Language EBNF for Database definition

```
⟨external-def⟩  ::=⟨fun-def⟩ | ⟨data-def⟩ | ⟨schema-def⟩
⟨schema-def⟩    ::=create schema ⟨schema-element⟩
⟨schema-element⟩::=⟨base-table-def⟩ | ⟨view-def⟩ | ⟨privilege-def⟩
⟨base-table-def⟩::=create table ⟨base-table-name⟩
                   ({base-table-def}[,]*)
⟨view-def⟩      ::=create view ⟨view-name⟩ [({⟨column⟩ [,]*}] as
                   ⟨query-spec⟩ [with check option]
⟨privilege-def⟩ ::=grant ⟨privileges⟩ on ⟨table-name⟩ to
                   {⟨grantee⟩ [,]}*
```

데이터베이스를 조작할 수 있는 구문은 C-형태 구문의 명령문과 동등하게 취급하였다. 데이터베이스의 기본 조작 명령인 open, close, insert, delete, update, select, fetch, commit, rollback을 위한 명령문을 허용하였다. 이것은 메타언어 CSQL이 변환 중의 내부 형태로만 사용되는 것이 아니라 메타언어 자체로도 충분히 우수한 정보를 갖게 하기 위해서이다. 〈표 2〉는 데이터베이스 조작을 위한 EBNF를 나타내고 있다.

표. 2. 데이터베이스 조작을 위한 EBNF
Table. 2. EBNF for Database manipulate

```
⟨statement⟩ ::=
                .
                .
            ⟨db-state⟩;
                .
                .
⟨db-state⟩          ::=⟨close-statement⟩
                       ⟨commit-statement⟩
                       ⟨delete-state-pos⟩
                       ⟨delete-state-sear⟩
                       ⟨fetch-statement⟩
                       ⟨insert-statement⟩
                       ⟨open-statement⟩
                       ⟨rollback-statement⟩
                       ⟨select-statement⟩
                       ⟨update-state-pos⟩
                       ⟨update-state-sear⟩
⟨close-statement⟩   ::=dbclose ⟨cursor⟩
⟨commit-statement⟩  ::=dbcommit work
⟨delete-state-pos⟩  ::=dbdelete from ⟨table-name⟩ where
                       current of ⟨cursor⟩
⟨delete-state-sear⟩ ::=dbdelete from ⟨table-name⟩
                       [where ⟨search-cond⟩]
⟨fetch-statement⟩   ::=dbfetch ⟨cursor⟩ into {⟨parameter⟩[,]}*
⟨insert-statement⟩  ::=dbinsert into ⟨table-name⟩
                       [({⟨column⟩[,]}*)] {values
                       ({⟨insert-atom⟩[,]}*) ⟨query-spec⟩}
⟨open-statement⟩    ::=dbopen ⟨cursor⟩
⟨rollback-statement⟩::=dbrollback work
⟨select-statement⟩  ::= dbselect [all | distinct]⟨selection⟩
                       into{⟨parameter⟩[,]}* ⟨table-exp⟩
⟨update-state-pos⟩  ::=dbupdate ⟨table-name⟩ set {⟨assign⟩[,]}*
                       where current of ⟨cursor⟩
⟨update-state-sear⟩ ::= dbupdate ⟨table-name⟩ set {⟨assign⟩[,]}*
                       [where ⟨search-cond⟩]
```

질의(query)에 대한 표현은 기존의 표현식에 첨가하여 활용하도록 했다. ⟨표 3⟩이 질의에 대한 EBNF이다.

메타언어 CSQL은 기존의 프로그래머나 단말사용자들에게 낯설지 않고 쉽게 접근할 수 있도록 가장 광범위하게 사용되는 언어를 기반으로 정의한 언어다. 그러나 이 결합 형태의 언어인 CSQL은 4세대 언어를 포함한 오늘날 사용되는 모든 시스템들에서 처리되는 원시코드를 충분히 대치할 수 있다.

표. 3. 질의에 대한 EBNF
Table. 3. EBNF for Query

```
⟨expression⟩     ::=⟨general-exp⟩ | ⟨query-exp⟩
⟨query-exp⟩      ::=⟨query-term⟩
                     ⟨query-exp⟩ union [all] ⟨query-term⟩
⟨query-term⟩     ::=⟨query-spec⟩ | (⟨query-exp⟩)
⟨selection⟩      ::={⟨scalar-exp⟩[,]}* | *
⟨table-exp⟩      ::=⟨from-clause⟩ [⟨where-clause⟩]
                     [⟨group-by-clause⟩] [⟨having-clause⟩]
⟨from-clause⟩    ::=from {⟨table⟩ [⟨range-variable⟩]}[,]}*
⟨where-clause⟩   ::=where ⟨search-condition⟩
⟨group-by-clause⟩::=group by {⟨column-ref⟩ [,]}*
⟨having-clause⟩  ::=having ⟨search-cond⟩
```

2.2 메타 언어의 변환 규칙

기존에 작성된 프로그램을 메타 언어로 변환하는 작업은 언어의 함축적인 의미를 풀어 헤지는 것과 거의 같다. 그래서 각각의 함축적인 의미들이 각각 장문의 메타 언어로 변환되어야 한다. 이들 변환의 모든 사례와 여러 가지의 언어를 다 소개할 수는 없기 때문에 언어가 갖고 있는 문법적 특성에서 앞 절에서 정의된 메타 언어로의 변환만을 설명한다.

첫 번째로 데이터베이스를 정의하는 부분들에 대한 변환이다. 4세대 언어에서는 데이터베이스와 데이터 모델의 일부분을 서술할 수 있어야 하기 때문에 project, select, sort, join과 같은 문장이 가능할 것이고 이러한 문장이 메타 언어로 변환되어야 한다. T1은 1의 문장들이 메타 언어 구문으로 변환됨을 의미한다.

T1
{ Database Definition } ——⟩ {⟨db-state⟩}

⟨db-state⟩는 4세대 언어의 각각의 관계 연산에 해당되는 구문들로 변환 가능하다. 예를 들어, 4세대 언어에서 "select"에 대한 문장이 있었다면 메타 언어 ⟨select-statement⟩로 변환이 되어 다음과 같은 형식의 문장이 될 것이다.

dbselect all * into sum, total from s-table

두 번째로 데이터베이스의 질의 표현이나 갱신에 대한 구문 변환이 필요하다

Tl
{Database Operation} ——〉 {〈query-exp〉}

〈query-exp〉 는 데이터베이스의 간단한 질의 표현식이나 자료의 수정과 같은 연산을 표현해 주는 구문이다. 그러므로 4세대 언어에서 제의된 모든 연산이 대치 가능하다.

세 번째는 스크린 디자인 및 Dialogue 디자인이다. Dialogue 디자인은 스크린 디자인의 확장 형태이므로 사실상 스크린을 디자인하는데 필요한 모든 구문만 변환하면 된다.

Tl
{screen design} ——〉 {〈write&position-state〉}

위치 이동과 형식화된 출력 문장의 조합으로 조합으로 스크린 디자인 문장은 대치 가능하다.

마지막으로 리포트 제너레이션 문장의 변환이다. 스크린 및 다이얼로그 디자인은 화면상에서 출력 위치를 변화해 가면서 자유롭게 출력 가능하지만 리포트 제너레이션은 한 방향 출력만 가능하므로 버퍼에 데이터 집합을 옮긴 후 출력하는 작업이 반복된다.

Tl
{Report Generation} ——〉 {〈buffering&write-state〉}

이외에도 일반적인 절차적 기술에 대한 문장들이 있는데 이러한 문장들은 메타 언어 각각의 구문들로 충분히 표현 가능하다. 왜냐하면 메타 언어의 나머지 문장들은 3GL의 절차적 구문을 그대로 수용하기 때문이다.

Tl
{another procedural statements} ——〉 {〈others statements〉}

지금까지의 모든 변환 규칙들은 현재 상용화된 대부분의 4세대 언어를 충분히 변환시킬 수 있다. 그리고 변환된 메타 언어 프로그램은 역공학 도구를 구성하는데 충분한 정보를 유지하고 있으며 내부적 표현뿐만이 아니라 독립된 하나의 언어로서 의미 전달이 가능하다.

Ⅲ. 클래스 추출기법의 설계

메타언어(C 원시 프로그램)를 객체 지향 프로그램으로 재구성하기 위하여 클래스를 추출하기 위한 도구를 설계 및 구현한다.

3.1 설계 개요

원시 코드는 사용자 인터페이스를 통해 파서(parser)로 넘겨진다. 파서는 원시 코드를 분석하여 파스 트리를 생성한다.

다음 과정(Filter)은 프로그램에 존재하는 모순을 제거하는 단계로서 프로그램의 오류로 인한 쓸모 없는 정보들을 걸러내는 작업을 행한다.

클래스 추출 과정(Object Extractor)에서는 함수와 전역 변수들에 관해 생성된 모든 정보들을 분석하여 클래스를 추출하게 된다.

클래스를 추출하기 앞서 각 함수들과 자료들을 클러스터링하고(ADT Generating), 각 함수간의 유사도를 측정하여 서로 밀접하게 연관된 함수들을 클러스터링하는 추상화 자료 생성 작업을 수행하게 된다. 그리고 마지막 과정으로 생성된 각각의 클러스터를 계층 분석하여 베이스 클러스터(Base Cluster)와 유도 클러스터(Derived Cluster)를 형성한다(Class Structuring). 위의 과정을 모두 거치고 난 후 추상화 자료에서 생성된 클러스터내의 함수 및 변수들은 클래스 구성 요소가 된다.

각 단계를 설명하기 전에 알고리즘에서 사용할 용어들을 정리한다.

[정의] 함수 집합 F
F : 프로그램 내의 모든 함수들의 집합.
[정의] 전역 변수 집합 GV
GV : 프로그램 내의 모든 변수들의 집합.

[정의] 함수 fj 의 호출 함수 집합 CF(fj)
CF(fj) = {fi|fj ∈ fi F, fi calls fj directly}

[정의]함수 fj 의 피호출 함수 집합 CedF(fj)
CedF(fj) = {fi|fj ,fi F. fj calls fi directly}

[정의]호출 함수 fj 의 전역 변수의 집합 CF_GV(fj)
CF_GV(fj) = {v|v GV. v는 호출 함수에서 사용}

[정의]피호출함수 fj 의 전역 변수의 집합 CedF_GV(fj)
CedF_GV(fj) = {v|v GV. v는 피호출 함수에서 사용}

[정의]전역 변수 v를 사용하는 함수들의 집합 F_GV(v)
F_GV(v) = {f|f F. f는 v를 사용 혹은 정의}

[정의]함수 f가 사용하는 전역 변수들의 집합 GV_F(f)
GV_F(f) = {v|v GV. v를 f가 사용하거나 정의}

[정의] 함수 수준 Level(f)
Level(f) : 호출 트리에서 나타나는 함수의 수준

3.2 파서 및 모순 제거

원시 코드의 모순을 제거(Filter)하기 위해서는 파스 트리로부터 함수, 전역 변수 및 자료형에 관한 정보를 수집하여 참조관계표에 저장해야 한다. 참조관계표는 함수 호출 참조관계표, 함수 피호출 참조관계표 그리고 함수와 전역 변수에 관한 참조표로 구성된다. 함수 호출 참조관계표에는 하나의 함수에 대해 그 함수를 호출하는 모든 함수들에 대한 정보를 저장하고 함수 피호출 참조관계표에서는 함수들 사이의 관계, 전역함수가 호출하는 모든 피호출 함수에 대한 정보를 저장한다.

참조관계표는 함수가 다른 함수 혹은 변수를 참조할 때는 1로 참조하지 않을 때는 0으로 세팅한다. 함수들 사이의 참조 관계는 DAG[4]를 사용하면 함수 호출 트리 구조로 나타나게 된다. 생성된 참조관계표에서 함수 대 함수 관계를 추출한 후 위상 정렬(topological sorting)을 사용하면 프로그램 단계 표현 구조를 생성할 수가 있다. 프로그램 단계 표현은 함수들의 depth와 fan-out, fan-in 그리고 어느 파일에 존재하는지에 관한 정보를 나타낸다. 이 정보는 함수 클러스터링시에 사용된다.

또한 함수 사이의 호출 관계는 함수 관계를 측정할 수 있는 중요한 요소가 되고 또한 클래스 생성시 메소드를 결정할 수 있는 중요한 요소이기 때문에 생성된 호출 트리에서 호출자와 피호출자에 대한 정보를 수집해야만 한다. 전역 변수도 함수들간의 의존도를 측정할 수 있는 가장 일반적인 정보이기 때문에 그것에 대한 정보를 수집해야만 한다.

다음은 파싱 과정을 통해 나타난 정보들을 사용하여 프로그램 내의 모순을 제거하기 위한 알고리즘이다.

```
for all GV
  if ( gbvar Vi ⊂ any fj ) then
     GV := GV - Vi;
  if ( gbvar Vi ⊆ only one fj ) then
     localize Vi into fj;

for all fi ⊆ CF
  if ( fi is not main ) &&
    ( fi's level == 0 ) then
     F := F - fi;
     for all fj ⊆ CedF
       if CedF(fj) is in CF then
          F := F - fj;
```

위의 알고리즘은 전역 변수의 집합 GV에 속하는 모든 전역 변수들 중 어느 함수에도 사용되지 않은 전역 변수는 제거하고 하나의 함수에서만 사용되는 전역 변수는 지역변수화 시킨다는 것을 보여준다. 또한 모든 함수들을 조사하여 maIn이 아니면서 프로그램 단계 표현표에서 레벨이 0으로 나타나는 함수는 제거하여야 한다는 것을 보여준다. 마지막 루틴은 함수 집합 F를 구성하기 위해 모든 피호출 집합에 대해 호출 집합에 있는 함수를 하나 제거하는 루틴이다.

3.3 자료 및 함수 클러스터링

구조체와 공통 자료 구조, 함수, 각 모듈에서 반복되는 변수 선언 등을 토대로 클래스를 구성할 때 클러스터링된 전역 변수들은 클래스의 속성이 되게 하고, 클러스터링된 함수들은 메소드(method)가 되게 한다. 이러한 과정을 수행하기 위해 다음과 같은 두 가지 작업이 어루어진다.

3.3.1 자료 클러스터링

자료 클러스터링에서는 각각의 함수에서 사용하는 서로 연관된 자료들을 클러스터로 만드는 작업을 수행한다. 자료 클러스터링에서는 전역 변수만을 고려한다. 전역 변수를 참조하는 함수들의 집합을 이용하여 전역 변수 a와 b가 함수 클러스터 f1,f2,f3,f4,..에서 모두 사용되었다면 그 전역 변수는 매우 밀접한 관계를 가지므로 하나의 자료 클러스터로 생성한다.

다음은 자료 클러스터링 과정을 나타내는 알고리즘이다.

```
for all Vi  GV
  if ( Vi's F_GV == Vj's F_GV ) then
   clustering Vi and Vj into one   cluster;
```

위의 알고리즘은 전역 변수1을 사용하는 함수들의 집합이 전역 변수2를 사용하는 함수들의 집합과 같은지를 체크하는 루틴이다. 위의 알고리즘을 통해 어느 함수에서도 사용되지 않았거나 혹은 지역 변수화된 전역 변수는 이 참조 관계 테이블에서 제거된다.

3.3.2 함수 클러스터링

함수 클러스터링을 하기 위해서 순환적 호출을 하는 함수를 하나의 클러스터로 생성하는 작업을 수행한다. 순환적 호출을 가지는 두 함수들 서로 밀접한 관계를 지니고 동작하기 때문에 그것을 하나의 클러스터로 생성하는 것이 필요하다. 또한 단지 하나의 호출 함수만을 갖는 함수는 전역 변수와의 연관성을 고려하여 피호출 함수, 즉, 프로그램 단계 표현에서 fan-out이 0 이고 fan-in이 1 인 함수와 클러스터링 한다. 함수와 전역 변수와의 연관성은 함수가 참조하는 전역 변수들의 집합 즉, GV_F를 이용하여 호출 함수가 사용하는 전역 변수의 집합체에 피호출 함수가 사용하는 전역 변수의 집합이 속하는지 속하지 않는지를 검사함으로써 알 수 있는데, 클러스터링의 조건은 전자에 속한다.

다음은 함수 클러스터링 과정을 나타내는 알고리즘이다.

```
for all fi ⊆ F
 if recycling between fi and fj then
/*fj means every function except fi in F*/
   clustering fi and fj into one cluster;

 for all fi ⊆ F
  if ( fi's fan-out == 1) then
   if ( fi's level ≠ 0) &&
     ( CedF_GV(fi) ⊆ CF_GV(fi) ) then
    clustering fi and called function of fi
      into one cluster;
```

3.4 추상화 자료 생성

함수 f가 사용하는 전역 변수에 대한 다른 함수들과의 관계는 참조관계표를 사용하여 분석할 수 있는데 그 정보를 사용하여 전역 변수(혹은 전역 변수 클러스터)와 함수(혹은 함수 클러스터)간의 밀접한 관계를 찾아낸다. 또한 함수(혹은 함수 클러스터)들 사이의 관계성을 분석하여 각 정보를 이용하여 함수 유사성을 구한다. 함수 유사성은 함수 A와 함수 B사이의 공통적인 특성의 집합과 각 함수에서만 사용되는 특성 집합, 그리고 두 함수 사이의 호출 종속성 집합을 이용한다. 일단 함수 유사성이 구해지면 함수 클러스터링을 수행한 후 그 결과를 가지고 함수, 전역 변수, 그리고 자료형을 하나의 구조로 생성하는 추상화된 자료형을 추출한다. 본 논문에서는 함수 유사성을 구하기 위해서 Arch의 유사성 공식[9]을 바탕으로 작성한 새로운 유사성 함수를 사용한다.

$$SIMILARITY(A,B) = \frac{W^{*}n(A\cap B)+k^{*}Linked(A,B)}{1+W^{*}(n(A\cap B)+n(A-B)+n(B-A)}$$

여기서, A B : A와 B에서 공통적으로 사용하는 특성.
A - B : A에서만 사용되는 특성.
B - A : B에서만 사용되는 특성.
Linked(A,B) : 두 함수 사이의 호출 관계로서 Arch의 식과 같다.
k : 1 / { (A와 B 레벨 차이의 절대값)+1 }

이 식에서는, n을 1로 하여 유사성 함수를 0과 1사이의 값으로 정규화 되도록 하였다. 또한 두 함수 사이의 구별되는 특성 즉, A-B 혹은 B-A에 곱해지는 d 값은 1로 주었는데 그 이유는 각각의 함수가 가지고 있는 특성 값은 다른 함수에게 영향을 크게 미치지는 않게 때문이다.

다음은 두 함수 사이의 특성 집합을 정의한 것이다.

[정의] 집합 (A∩B)A∩B:
[(CF(A)CF(B))][(GV_F(A)(GV_F(B))]
[CedF(A)CedF(B)]

즉, (A와 B를 공통적으로 호출하는 집합) + (A와 B의 공통적인 전역 변수의 집합) + (A와 B가 공통적으로 호출하는 집합) 을 나타낸다.

[정의] 집합 (A - B)

A-B:
[(CF(A)-CF(B))][(GV_F(A)-(GV_F(B))]
[CedF(A)-CedF(B)]

즉, (A를 호출하는 집합 - B를 호출하는 집합) + (A가 사용하는 전역 변수의 집합 - B가 사용하는 전역 변수의 집합) + (A가 호출하는 집합 - B가 호출하는 집합)

[정의]집합 (B - A)
B-A:
[(CF(B)-CF(A))][(GV_F(B)-(GV_F(A))]
[CedF(B)-CedF(A)

즉, (B를 호출하는 집합 - A를 호출하는 집합) + (B가 사용하는 전역 변수의 집합 - A가 사용하는 전역 변수의 집합) + (B가 호출하는 집합 - A가 호출하는 집합)

[정의] n(A+B)
n(A+B):
n(CedF(A) + GV_F(A) + CF(A) + CedF(B) + GV_F(B) + CF(B))

즉, (A가 호출하는 집합 + A가 사용하는 전역 변수 집합 + A를 호출하는 집합) + (B가 호출하는 집합+ B가 사용하는 전역 변수 집합 + B를 호출하는 집합)

본 논문에서는 클래스를 생성하기 위한 기본 단계로 함수들간의 참조 관계를 사용한 함수 호출 트리를 구성하게 된다. 또한 함수 호출 트리를 이용한 프로그램 단계 표현(layering) 구조를 생성하여 각 함수의 레벨(level)과 호출 및 피 호출 상태(fan-in, fan-out)를 분석함으로써 원시 코드 프로그램에 내재해 있던 모순도 제거한다.

다음은 두 함수 사이의 유사성을 가지고 클러스터링을 하는 알고리즘이다.

```
for all fi  F
{
  for all fj ⊆ F  /* fj is fi+1 */
    calculate similarity between fi and fj;
  if  SIMILARITY(fi,fj)>=  threshold  value
  then
      clustering fi and fj into one cluster.
}
for all generated cluster group
{
  if all functions in A cluster  another
cluster B
      remove A cluster.
}
```

3.5 클래스 계층화

앞 절에서 클러스터링한 각각의 클러스터를 비교 분석하여 일정한 임계치 값을 넘으면 중복된 함수 및 변수들을 새로운 하나의 클러스터에 넣은 후 비교된 두 클러스터와의 연관성 있는 이름을 사용하여 클래스 생성시 상위 클래스로 명명한다(Class Structuring). 이 작업은 다음과 같은 방법으로 수행된다.

클러스터 A와 B가 있다고 가정한다. 함수는 A(f), B(f), 변수는 A(v), B(v)로 표시한다.

```
COMF_CLU(A∩B)= A와 B의 공통된 함수
COMV_CLU(A∩B)= A와 B의 공통된 변수
COMMON(A,B) = (n(COMF_CLU(A∩B)
  + COMV_CLU(A∩B)) /(n(A(f)∪B(f))
  +n(A(v)∪B(v))

while (Flag )= 0.25)
 for all cluster group fi, fj  (단, fi≠fj )
   calculate COMMON( fi, fj );
 COMV := search the highest COMMON(
        fi, fj ) value;
 Flag  := COMV;
 while (COMV )= 0.25)
   fi, fj의 공통 부분을 추출하여 상위
   클러스터로 형성;
  COMV:= search next highest COMMON
        value,
          not including both fi and fj;
 fi:= (fi) - (fi∩fj);  fj := fj - (fi∩fj);
```

위의 알고리즘을 사용하여 각각의 추상화된 클러스터

들을 반복적으로 비교하여 연관 있는 것들을 클러스터링한 후 베이스 클러스터를 형성한다.

Ⅳ. 실험 및 고찰

제안한 방법을 사용하여 스크린 에디터 프로그램을 실험한 결과를 다음과 같이 나타내었다.

모순 제거 단계에서는 전역 변수 helpline이 단지 help 함수에서만 사용되었기 때문에 그 변수를 지역 변수로 처리하였다. 자료 클러스터링 단계에서는 클러스터링되는 자료가 없음을 확인하였다.

함수 클러스터링 단계에서 하나의 호출자만을 갖는 함수를 찾은 결과 search, load, replace, home, gotoend, left, right, pagedown, kill_line, save, yank 함수가 edit 함수에 의해 호출되고 그 함수들의 전역 변수가 모두 edit 함수의 전역 변수에 포함된다는 것을 알 수 있었다.

따라서 함수 클러스터링 단계에서는 함수들이 함수 클러스터(혹은 함수)로 재구성 된다.

유사도를 구한 후 클러스터링 과정을 반복하면 최종적으로 산출된 클러스터의 함수 및 전역 변수를 추출할 수 있다.

클러스터의 이름은 상위 클래스 생성시 기존 클러스터의 이름으로 사용되거나 사용자와의 상호작용에 의해 새로운 이름으로 바뀔 수 있다. 클러스터의 함수들은 클래스 생성시 메소드로 선언하고 또한 계층화 알고리즘을 통해 생성된 상위 클러스터는 수퍼 클래스로 선언한다. 위의 결과를 클래스 생성 규칙을 사용하여 다음과 같은 클래스를 생성하였다.

```
class scrollup1+display_scrn2 {
public :
   int scrnx;
   void printline( char *p );
protected :
 void display_scrn( int x, int y, char *p );
 void help( void );
 void clrscr( void );
}
class scrollup1 :
 public scrollup1+display_scrn2 {
private :
   void scrollup(int topx, int topy, int
   endx, int endy);
public :
   void scrolldn( int x, int y );
}
class display_scrn2 :
public scrollup1+display_scrn2 {
    public :
    void goxy( int x, int y );
}
```

Ⅴ. 결 론

본 논문에서 제안한 방법은 원시 코드로부터 클래스로 구성한 다음 계층 구조를 표현할 수 있고, 또한 역공학 기법을 이용한 상호작용적 질의에 의한 정보 추출도 가능하다. 그러한 정보 추출 기능은 프로그램을 이해하는데 많은 도움을 주고 또한 문서화의 효용성을 높일 수 있다. 그리고 상호작용을 통해 추출된 클래스를 보다 효율적인 클래스를 재구성할 수 있게 한다

참 고 문 헌

[1] A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Language," Conference Proceedings of Object Oriented Programming Systems,

Languages, and Applications, OOPSLA' 86, 1986.

[2] C. L. Ong, W. T. Tsai, "Class and Object extraction from Imperative Cod e," Joirmal of Object Oriented Programming, Mar.-Apr., 1993.

[3] D. R. Reed, M. Cagan, T. Goldstein, and B. Moo, "Issues in Moving from C to C++," Conference Proceedings of Object Oriented Programming Systems, Languages, and Applications, OOPSLA' 91, 1991.

[4] E. R. Gansner, S. C. North, and K. P. Vo, "DAG-A program that draws directed graphs," Software-Practice and Experience, Vol. 18, No. 11, Nov., 1988.

[5] Georgia Tech' s Reverse Engineering and Software Understanding Group, http://www.cc.gatech.edu/reverse/.

[6]I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1992.

[7]K. Lieberherr, I. Holland, and A. Riel, "Object-Oriented Programming: an Objective Sense of Style," Conference Proceedings of Object Oriented Programming Systems, Languages, and Applications, OOPSLA' 88, pp.335-353, Sep., 1988.

[8]R. Koschke, A Bibliography on Reengineering, Rev. 1.9, University of Stuttgart, Germany, E-Mail: koschke @informatik.uni-stuttgart.de

[9]R. W. Schwanke, "An Intelligent Tool or Re-engineering Software Modularity," Proceedings of International Conference on Software Engineering, pp. 83-92, 1991.

[10]S. B. Lee, J. S. Lee, D. H. Chi, and K. W. Rim, "A Reengineering Technique for Transformation of Function Oriented Program to Objectbased Program," The International Conference in Commemoration of 20th KISS Anniversary, Proceedings of InfoScience' 93, pp. 167-172, 1993.

[11]Tennessee Technological University' s Program Comprehension and Reengineering Tools and Technology, http://www.csc.tntech.edu/~linos/.

저자소개

진영배

85.9 - 93.8 인천기능대학 전자과 조교수

93.9 - 현재 충청전문대학 사무자동화과 조교수

95.4 - 현재 충청전문대학 전자계산소 소장