절차지향 소프트웨어로부터 클래스와 상속성 추출
(Extraction of Classes and Hierarchy from Procedural Software)

최 정 란 † 박 성 옥 † 이 문 근 ††
(Jeong-ran Choi) (Sung-og Park) (Moon-kun Lee)

요 약 본 논문은 절차지향 소프트웨어로부터 클래스와 상속성을 추출하기 위한 방법론을 제안한다. 본 논문에서 제안한 방법론은 모든 경우의 객체 후보군으로부터 정의된 클래스 후보군과 그들의 상속성을 생성하여 클래스 후보군과 영역 모델 사이의 관계성과 유사 정도를 가지고 최고 또는 최적의 클래스 후보군을 선택하는데 초점을 둔다. 클래스와 상속성 추출 방법론은 다음과 같은 두드러진 특징을 가지고 있다: 정적(속성)과 동적(메소드)인 클러스터링 방법을 사용하고, 클래스 후보군의 경우는 추상화에 초점을 두며, m개의 클래스 후보군과 n개의 클래스 후보 사이의 상속 관계의 유사도 측정 즉, 2차원적 유사도 측정은 m개의 클래스 후보와 n개의 클래스 후보 사이의 전체 그룹에 대한 유사도를 구하는 수평적 측정과 클래스 후보군들에서 상속성을 가진 클래스의 집합과 영역 모델에서 같은 클래스 상속성을 가진 클래스 집합 사이의 유사도를 위한 수직적 측정방법이 있다. 이러한 방법론은 최고 또는 최적의 클래스 후보군을 선택하기 위해 재공학 전문가에게 광범위하고 통합적인 환경을 제시하고 있다.

Abstract This paper presents a methodology to extract classes and inheritance relations from procedural software. The methodology is based on the idea of generating all groups of class candidates, based on the combinatorial groups of object candidates, and their inheritance with all possible combinations and selecting a group with the best or optimal combination of candidates with respect to the degree of relativity and similarity between class candidates in the group and classes in a domain model. The methodology has innovative features in class and inheritance extraction: a clustering method based on both static (attribute) and dynamic (method) clustering, the combinatorial cases of grouping class candidate cases based on abstraction, a signature similarity measurement for inheritance relations among n class candidates or m classes, two-dimensional similarity measurement, that is, the horizontal measurement for overall group similarity between n class candidates and m classes, and the vertical measurement for specific similarity between a set of classes in a group of class candidates and a set of classes with the same class hierarchy in a domain model, etc. This methodology provides reengineering experts with a comprehensive and integrated environment to select the best or optimal group of class candidates.

1. 서 론

최근 객체지향 SW가 각광을 받으면서[1], 절차적 언어로 작성된 원시 SW를 객체지향 SW로 변형하려는 재공학 연구가 보고되고 있다[2]. 이러한 재공학은 재사용을 통한 SW 생산성 향상, SW의 유지·보수 비용의 절감, 시스템에 대한 새로운 요구의 수용등의 장점이 있다[3].

객체지향 SW로 재공학하려는 시도들을 재공학 방식에 따라 분류하여 보면, 다음과 같다 : 1) 객체 모듈(Module) 형태로 재구성하는 방식[4], 2) 특정 응용 영역에 맞게 SW의 요소들을 싸거나(Wrap)[5], 그 영역의 구조에 사상(Mapping)하는 방식[6, 7], 3) 객체지향 모델을 매개체로 활용하는 방식[8], 4) 재공학을 위해 객체지향 명세를 유도하는 방식[9, 10], 5) 수학적 개념을 사용하는 방식[11], 6) 객체, 클래스 등을 정의하기

· 본 연구는 한국과학재단 특정기초연구(1999-2-303-003-3) 지원으로 수행되었음
† 비 회 원 : 전북대학교 전자정보공학부
jlchai@cs.chonbuk.ac.kr
sopark@cs.chonbuk.ac.kr
†† 종신회원 : 전북대학교 전자정보공학부 교수
mklee@cs.chonbuk.ac.kr
논문접수 : 2001년 2월 19일
심사완료 : 2001년 5월 24일

위하여 역공학과 설계복구 등의 지원을 받아 캡슐화, 추상화등의 나름대로 독특한 변형절차를 사용하는 방식[6, 12, 13, 14, 15, 16, 17]등이 있다[2].

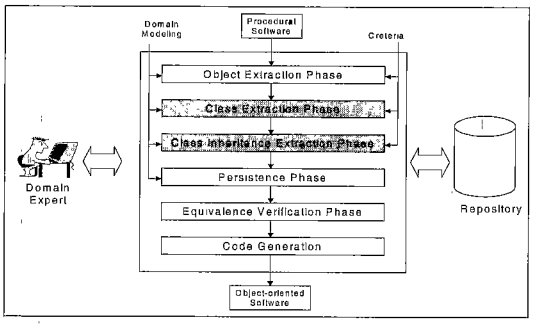


그림 1 절차중심 SW의 객체중심 SW로의 재공학 절차 흐름도

일반적으로 재공학 방법은 그림 1과 같은 절차로 진행되는 것이 바람직하다[18]. 객체 추출 단계에서는 전역변수, 사용자 자료형(UDT, *User-defined Type*), 함수와 파라미터를 기반으로 관련성 정도를 기준으로 클러스터링한다. 클래스 추출 단계에서는 클러스터링된 객체 후보들의 공통적인 특성을 추출하여 클래스를 추출한다. 클래스 추출 단계의 결과는 클래스들간에 대등한 평면 관계를 이루고 있다. 상속관계 추출 단계는 전 단계에서 추출된 평면화된 클래스들의 공통적 특성을 추출하여 *part-of* 관계나 *is-a* 관계를 추출하여 계층 구조를 만든다. 지속성 추출 단계에서는 절차 지향 프로그램에는 존재하지 않는 객체 지향 특성들과 동적인 부분을 추가한다. 이러한 것들은 클래스의 생성자, 소멸자, 동적 메모리 할당/해제 등이 있다. 코드 생성 단계에서는 전단계에서 생성된 뼈대 코드(skeleton code)를 기반으로 실행 가능한 객체 지향 프로그램을 생성한다. 테스트 단계에서는 생성된 객체지향 프로그램이 정상적으로 작동하는지와 원래의 절차지향 프로그램과 변환된 객체지향 프로그램이 의미상으로 동등한지에 대한 검사를 한다.

본 논문에서는 객체지향 프로그램으로 변환 과정 중 두 번째 단계와 세 번째 단계인 클래스 추출과 상속성을 추출한다. 첫 번째 단계인 객체 추출은 [18]의 방법을 기준으로 사용한다. 클래스 추출 단계에서는 클래스와 이들간의 *part-of* 관계를 추출하며 상속성 추출 단계에서는 상속성(*is-a* 관계)을 추출한다. 클래스 추출은 클래스 후보 추출, 속성 추출과 메소드 추출의 순서로 이루어진다. 클래스 후보 추출은 객체 추출결과인 객체 후보군으로부터 클래스를 추출, 속성 추출은 전역 변수와 사용자 자료형으로부터 추출, 메소드 추출은 객체 후보군의 함수로부터 추출한다. 상속성 추출은 전 단계에서 추출된 클래스들을 비교하여 유사도를 측정한 후 일정한도를 초과하는 클래스들간의 공통된 부분을 상위 클래스로 만들고 나머지 부분은 하위 클래스로 존재한다. 프로그램으로부터 상속성이 추출된 후, 프로그램에 대한 서로 다른 결과들이 계층 관계로 존재한다. 따라서 각각의 상속성 추출 결과로부터 적절한 최종적인 결과를 선택하여야 한다. 최종적인 결과의 선택은 영역 전문가에 의하여 수평적 관계(*horizontal relativity*)와 수직적 관계(*vertical relativity*)인 이차원적 결정 방법(two-dimensional decision)을 사용한다.

본 논문의 목적은 영역 전문가에게 영역 모델과 가장 유사도가 높은 클래스 및 상속성을 제시하는 것이 아니라, 비록 최적은 아니라도 다양하게 존재하는 클래스를 제시하여 영역 전문가가 다양한 선택 기회를 제시할 수 있도록 하는 것이다.

본 논문의 가정 사항은 다음과 같다 : 1) 절차 중심 SW에 대한 요구사항(Requirement)이 존재, 2) 요구 사항으로부터 생성된 영역 모델이 존재, 3) 영역 전문가가 존재, 4) 새로운 클래스를 생성하였을 경우 이를 명명하는 방법이 이미 존재하거나 영역 전문가에 이루어진다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 기술하며, 3절에서는 객체 지향 프로그램으로 변환하는 두 번째 단계인 클래스 추출과 세 번째 단계인 상속성 추출 방법론을 기술하며, 4절에서는 결론 및 향후 연구 과제에 대하여 기술한다. 본 논문에서는 모델의 전 과정을 인적관리 연결 리스트 프로그램 예제를 통하여 설명한다.

2. 관련연구

객체지향 재공학을 위하여 제안된 방법중에서 역공학을 통하여 생성한 객체 모델과 영역 지식으로부터 출발하여 생성한 객체 모델을 비교하여 최종 객체 모델을 생성해내는 방법으로 *COREM*[6, 19]가 있다. *COREM*의 접근 방법은 그림 2와 같다.

*COREM*에서는 절차 지향 프로그램으로부터 개체 관계 다이어그램(ERD, *Entity-Relationship Diagram*)을 생성하여 객체간의 상속성을 추출하였다. 생성된 ERD의 개체의 관계를 구별하기 위하여 두 가지 관계의 종류를 정의하였다. 첫 번째 관계인 상세화 관계(special relationship)는 시스템의 모델링을 위해 사용되는 *is-a*

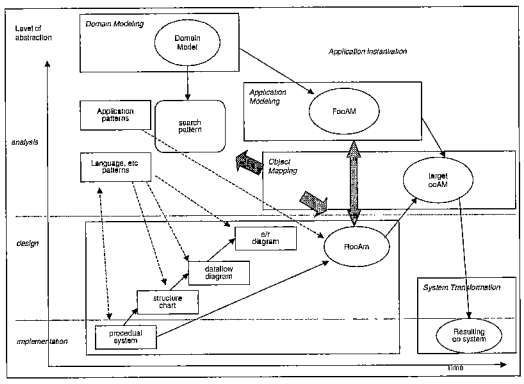


그림 2 COREM의 개관

와 *part-of* 관계를 의미한다. 두 번째 관계인 일반화(general relationship)는 순차적 시스템의 프로시저나 함수로 표현되는 ERD의 개체들의 기능적 관계를 의미한다. *Is-a* 관계는 다양한 레코드로부터 유도되었고, *part-of* 관계는 자료형 정의된 곳의 내부에 존재하는 배열이나 포인터 구조로부터 유도되었다. *is-a* 관계는 하나 또는 그 이상의 개체(객체 후보)사이의 기능적 관계를 나타낸다[6, 19].

최초에 절차지향 원시 프로그램으로부터 ERD를 추출한다. 추출된 ERD는 개체와 개체들간의 관계로 정의되며, 이러한 ERD를 이용하여 역공학 모델(*COREM*에서는 *RooAM*)을 생성한다. 정확한 변환을 위하여 영역 모델이 필요하다. 영역 모델(*COREM*에서는 *FooAM*)은 클래스 및 클래스간의 관계로 이루어지며 영역 전문가에 의하여 생성된다. *FooAM*은 절차지향 프로그램으로부터 생성된 *RooAM*과 사상 관계를 통하여 유사도를 측정한다. 두 모델의 비교는 구조적 유사성 추출, 속성의 추출 및 메소드 추출하는 순서로 진행된다. 이러한 세 가지 과정을 거치며 소속이 모호한 부분은 영역 전문가에 의하여 결정된다. 클래스 추출 및 상속성 추출의 기준은 *FooAM*을 사용하였다[6, 19, 20].

Wiggerts[21] 는 기존의 시스템에서 객체를 식별하기 위해 점증적 접근방법을 제안하였다. 이러한 접근방법을 살펴보면 세 가지의 서로 다른 형태의 방법론이 객체 식별을 위한 시나리오로 제안되었다: 함수 중심(function driven), 데이타 중심(data driven) 그리고 객체 대상화(object objectification). 즉, 세 가지의 서로 다른 방법이 객체를 식별하기 위해 적용되었다. 이들 방법 사이에는 서로 상관관계가 없이 독립적인 방법으로 수행된다. Wiggerts의 연구와 비교하여 본 논문은 자료중심(data driven)의 방법으로 진행하였으나, 클래스와 상속성 추출 과정에서 데이타와 자료형의 강한 결합을 기반으로 하고 있다. 또한 모든 가능한 경우의 클래스 후보군을 구성하고, 역비순환 그래프를 생성하여 모든 가능한 경우의 클래스 후보군들이 계층적으로 구성된다. 이러한 방법은 전문가가 원하는 계층(level)에서 클래스들을 선택하도록 한다.

Siff와 Reps[22, 23]는 기존 코드에서 모듈을 식별하기 위한 일반적인 기술을 제안하고 있다. 이 기술은 개념 분석(concept analysis)[24]을 기반으로 한다-객체의 집합에서 유사성을 식별하기 위해 사용 가능한 격자이론(lattice theory)의 분기는 그들의 속성을 기반으로 하고 있다. 논문에서는 두 가지의 중요한 문제점이 거론된다: 1) 개념 격자(concept lattice)의 노드와 분할(partition)의 수가 객체의 수가 증가함에 따라 지수적으로 증가한다, 그리고 2) 유일한 객체 집합의 형태를 이루고 있다.

Linding과 Snelting[25]은 수학적 개념 분석을 기반으로 하여 기존 코드를 모듈화하는 방법론을 제안하였다. 프로시저와 전역변수 사이의 관계를 분석하여 개념적 격자를 구성하였다. 격자는 모듈 후보 사이의 응집도와 결합도를 분석하기 위해 사용된다.

박외진[26]은 재공학 과정 중 발생할 수 있는 소스 코드의 객체모델과 영역 모델 사이의 많은 불일치성에서 추상화 단계와 명명법을 극복하기 위한 방법론을 제안하였다. 이러한 불일치성을 극복하여 두 모델을 비교, 분석하여 효율적인 최종 객체 모델을 정제해내는 객체 모델 정제 기법(ORT, object-oriented model refinement technique)을 제안하였다. ORT를 위한 객체 모델의 생성 정보를 경험적 지식에 기반을 두어 명세 정보 트리(specification information tree)를 구성하고, 트리 구조화된 자료사전(tree-structured data dictionary) 과 클래스 관계 행렬(class dependency matrix)을 이용하여 영역 모델과 비교, 분석이 이루어졌다.

Siff와 Reps, Linding과 Snelting, 박외진의 연구와 그 외 기존의 방법[6, 11, 12, 14, 15, 16, 17, 19, 20]과 비교하여 본 논문에서는 자료형과 데이타가 강하게 결합되어있고, 이들의 모든 가능한 경우가 그래프 내에 각각의 계층의 형태를 이루고 있다. 결합된 자료형과 데이타들은 경우의 수에 따라 계층적으로 구성되어 있다. 또한 여러 실험 자료를 통한 분석 결과 자료형과 데이타에 따른 노드의 수는 $O(n^2)$로 노드의 무한한 증가는 보이지 않았다. 경우의 수에 따른 클래스 후보군이 존재하여 영역 전문가에게 최적 또는 최고의 클래스 후보군을

선택할 수 있는 기회를 제공한다.

3. 클래스 추출 및 상속성 추출

클래스와 상속성을 추출하기 위하여 객체 추출의 결과를 사용하여야 한다. 절차지향 프로그램으로부터 객체의 추출은 다섯 단계로 설정되었다. 사용한 객체 추출 방법은 다음과 같이 *전처리, 기본 분할 및 결합, 정제 결합, 결정* 및 통합의 다섯 단계로 이루어진다[18]. 본 논문의 클래스 추출 및 상속성 추출에서는 객체 추출의 세 번째 단계의 결과인 역 비순환 그래프(G^{RA})[18]를 사용한다.

논문에서 사용한 예제는 UDT(User-Defined Type)가 4개, 전역변수가 5개, 함수가 15개로 이루어진 300라인 정도의 프로그램이다. 프로그램에서는 인력관리를 연결 리스트로 관리한다. [18]의 기준값을 이용하여 객체 추출의 결과 생성된 결과는 표 1과 같다.

표 1에서 각각의 열(Row)은 전체 원시 프로그램에 대한 정보를 나타낸다. 이 곳에서 $G_{3:(1,2,3)}$은 전체 프로그램의 FTV(함수, 사용자 자료형, 전역변수)가 기본적으

표 1 객체 추출 결과

	TYPE	VARIABLE	FUNCTION
$G_{1:(6)}$	person, teacher, student, list	head, tail, current. no_of_teacher, no_of_person	read_person, print_person, add_course, read_teacher, print_teacher, read_student, print_student, insert_list, print_list, find_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element
$G_{2:(1,5)}$			불가
$G_{2:(2,4)}$			불가
$G_{2:(3,3)}$	student, teacher	no_of_teacher	read_student, print_student, add_course, read_teacher, print_teacher
	list, person	head, tail, current, no_of_person	insert_list, print_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element, find_list, read_person, print_person
$G_{3:(1,1,4)}$			불가
$G_{3:(1,2,3)}$	student		read_student, print_student
	teacher	no_of_teacher	add_course, read_teacher, print_teacher
	list, person,	head, tail, current. no_of_person	insert_list, print_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element, find_list, read_person, print_person
$G_{3:(2,2,2)}$			불가
$G_{4:(1,1,1,3)}$			불가
$G_{4:(1,1,2,2)}$	list	head, tail, current	insert_list, print_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element, find_list
	student		read_student, print_student
	teacher	no_of_teacher	add_course, read_teacher, print_teacher
	person	no_of_person	read_person, print_person
$G_{5:(1,1,1,1,2)}$	list	head, tail, current	insert_list, print_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element, find_list
	student		read_student, print_student
	teacher		add_course, read_teacher, print_teacher
		no_of_teacher	
	person	no_of_person	read_person, print_person
$G_{6:(1,1,1,1,1,1)}$	list	head, tail, current	insert_list, print_list, get_no_of_object, get_no_of_person, get_no_of_total_element, print_no_of_element, find_list
	student		read_student, print_student
	teacher		add_course, read_teacher, print_teacher
		no_of_teacher	
	person		read_person, print_person
		no_of_person	

로 6개의 클러스터로 구성되는데, 이들이 객체 추출 결과 후 3개의 클래스터(1개가 모인 1개의 클러스터, 2개가 모인 1개의 클러스터, 3개가 모인 1개의 클러스터)로 구성됨을 의미한다. 표 1에서 $G_{3:(1,1,4)}$, $G_{3:(2,2,2)}$, $G_{4:(1,1,1,3)}$이 생성되지 않은 것을 볼 수 있다. 이 것은 영역 전문가에 의하여 입력된 기준값에 의하여 클러스터링 된 것들이 재분할되지 못했기 때문이다.

일반적으로 객체 지향 프로그램의 클래스들 사이에 두 가지 관계성이 존재한다. 클래스들간의 포함 관계를 나타내는 *part-of* 관계와 클래스들간의 일반화/추상화 관계를 나타내는 *is-a* 관계이다. 본 논문의 3.1절에서는 *part-of* 관계를 추출하며 3.2절에서는 *is-a* 관계를 추출한다.

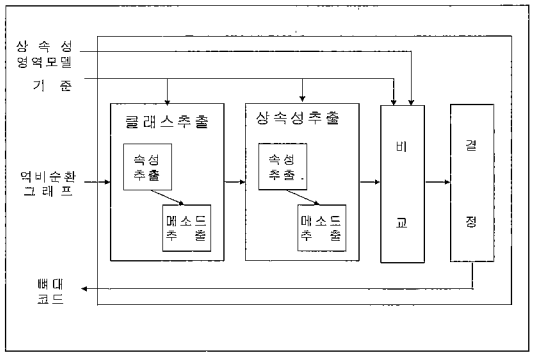


그림 3 클래스와 상속성 추출

이 절에서는 그림 3과 같이 클래스 추출, 상속성 추출, 상속성이 존재하는 클래스들과 상속성이 존재하는 영역 모델링과의 비교 및 결정에 대하여 기술한다.

3.1 클래스 추출

절차 중심 SW를 객체 중심 SW로의 재공학 절차의 두 번째 단계는 클래스 추출이다. 클래스를 추출하는 경우는 크게 두 분류로 구분할 수 있다. 첫 번째 경우는 영역 모델의 클래스를 이용하여 다음 단계로 계속 진행하는 것이다. 두 번째 경우는 원시 프로그램으로부터 추출된 객체 후보로부터 클래스를 추출하여 다음 단계로 계속 진행하는 것이다. 어떠한 방법을 이용하는가에 따라 각기 장단점이 있다.

영역 모델로부터 추출된 클래스를 이용할 경우 클래스를 간단히 추출할 수 있다는 장점이 있다. 그러나 이에 상응하는 여러 가지 단점이 존재한다. 이러한 단점으로는 만일 영역 전문가가 개략적인 영역 모델링만을 제시하였을 경우 너무 추상화된 클래스가 추출이 되므로 실제 변환된 객체 지향 코드가 부실할 수 있고 실행 가능한 코드로 변환하기에 너무 빈번한 영역 전문가의 개입이 필요하다는 것이다. 또한 변환되지 않는 부분들이 존재하므로 절차지향 코드 내부에 많은 비 절차지향 코드가 존재한다는 단점도 있다.

원시 프로그램으로부터 추출된 객체 후보를 이용하여 클래스를 추출할 경우 여러 가지 장단점이 있다. 원시 프로그램 정보을 기준으로 할 경우 영역 모델의 의도를 벗어날 수 있다는 단점이 존재한다. 그러나, 영역 전문가에게 프로그램의 프로토타입만을 제공하는 것만으로도 객체 지향 프로그램 작성에 비용을 줄일 수 있고, 원시 프로그램이 객체 지향 프로그램과 1:1로 매핑되며 영역 전문가가 미세한 부분만을 수정한다면 프로토타입만을 제공하는 것보다 비용을 줄일 수 있다는 장점이 존재한다. 본 논문에서는 원시 프로그램을 기준으로 할 경우 발생할 수 있는 단점을 극복하기 위하여 영역 전문가에게 1개의 선택이 아닌 다양한 선택 기회를 주어 정확한 프로그램 변환을 유도할 수 있기 때문에 원시 프로그램으로부터 클래스를 추출하였다.

클래스 추출 절차는 객체 추출 단계의 결과 생성되는 G^{RA}로부터 클래스 후보를 추출, 클래스의 속성을 추출하고 클래스의 메소드를 추출하는 순서로 진행된다.

3.1.1 역 비순환 그래프와 클래스 후보 추출

객체 추출 결과 생성된 G^{RA}는 각각의 계층(Layer)을 형성하는 객체 후보군(OCG, *Object Candidate Group*)으로 구성되며, 하나의 OCG는 여러 개의 객체 후보(OC, *Object Candidate*)로 구성된다[18]. 객체 후보는 서로 밀접하게 연관된 전역변수, UDT와 함수로 구성된다. OC는 3가지 구성 요소 모두로 구성될 수도 있고 이들 중 1개 혹은 2개가 포함되지 않고 존재할 수도 있다.

1개의 OC로부터 클래스 후보의 추출하는 기준은 다음과 같다.

[정의 1] 클래스 후보의 추출 기준

1) OC에 UDT가 존재하지 않을 경우 OC로부터 1개의 클래스 후보를 추출한다.

클래스 후보를 추출할 때, 원칙적으로 1개의 OC가 1개의 클래스 후보로 된다. 이 것은 G^{RA}를 생성할 때, 영역 전문가에게 다양한 선택 기회를 제공하기 위하여 다양한 경우로 클러스터링을 하였고 클러스터링 된 1개의 단위(즉, OC)가 1개의 클래스로 될 것이라 가정했기 때문이다. 어떠한 계층에서 1개의 OC가 1개의 클래스 후보(CC, *Class Candidate*)가 될지라도 하위 계층에서는 다수의 클래스 후보가 될 수 있기 때문에 1개의 OC는 1개의 클래스 후보가 된다[18].

2) OC 내부에 UDT가 존재할 경우는 다음의 기준에 따른다.

- UDT와 함수를 제외한 나머지 요소(즉, 전역변수)들은 1개의 클래스 후보로 추출된다.
  OC에서 UDT를 제외한 부분도 클래스의 구성 요소로 되어야 한다. UDT를 제외한 부분은 함수와 전역 변수이다. 이들은 OC 전체를 포함하는 1개의 클래스 후보로 추출된다.
- OC 내부에 존재하는 UDT는 클래스 후보로 추출된다.
  절차 지향 프로그램의 UDT는 객체지향 프로그램에 존재하는 메소드가 없이 속성만을 가진 클래스와 유사하기 때문이다[14].

[정의 1]에 의하여 클래스를 추출할 경우, UDT가 n개이면 클래스 후보는 최대 n+1개가 나오며 최소 n개가 나온다. 클래스 후보가 n개일 경우는 OC가 UDT와 관련된 함수로서 완전히 분할이 가능할 경우이고, n+1개일 경우는 완전히 분할되지 않고 나머지 부분이 존재할 경우이다. 1개의 OC로부터 생성되는 클래스의 모임을 CCs(*Class Candidates*)라 부르기로 한다.

OC 전체와 관련된 클래스 후보를 추출한 후 클래스 이름을 명명하는 문제가 남아있다. UDT로부터 추출된 클래스가 아닌 새롭게 생성된 클래스의 이름은 의미있게 하기 위해서는 영역 전문가의 개입이 필요하다. 따라서 본 논문에서는 영역 전문가가 필요할 경우 클래스 이름을 명명한다고 가정한다.

그림 4는 $G_{3:(1,2,3)}$에 존재하는 1개의 OC인 *struct list*에 대한 것이다.

```
struct person {
  char name[9] ;
} ;
struct list {
  void *content ;  int  type;
  struct list *next ;
} ;
int no_of_person ;
struct list *head, *tail, *current ;
void insert_list( int sel_type ) ;
void find_list( void ) ;
void print_list( void ) ;
int get_no_of_object( int aType ) ;
int get_no_of_person() ;
int get_no_of_total_element( void ) ;
struct person *read_person( void ) ;
void print_person( struct person *per ) ;
void print_no_of_element( void ) ;
```

그림 4 $G_{3:(1,2,3)}$에서의 1개의 OC

그림 4에서 UDT가 2개 있음을 관찰할 수 있다. 클래스 추출 기준에 따라 2개의 UDT인 *struct person*과 *struct list*는 각각의 클래스를 형성한다. 또한 객체 추출 가정(OC는 1개의 클래스를 추출한다)에 따라서 OC 전체에 대한 클래스를 형성한다. OC 전체에 대한 클래스를 *person_list*라고 영역 전문가에 의하여 명명된다고 가정한다. 그림 5는 정의 1의 기준에 의하여 그림 4에 대한 클래스 후보 추출 결과이다.

```
class person { … } ;
class list { … } ;
class person_list { … } ;
```

그림 5 그림 4에 대한 클래스 후보 추출 결과

3.1.2 속성의 추출

3.1.1절에서 추출된 클래스 후보 추출은 UDT로 추출된 클래스 후보와 OC로부터 추출된 클래스 후보의 2종류가 존재한다는 것을 알 수 있다. UDT로부터 추출된 클래스의 속성은 UDT가 가지고 있는 내부 변수를 사용할 수 있기 때문에 해결해야할 문제가 없다. 그러나 OC로부터 추출된 클래스 후보의 속성을 추출하기 위해서는 별도의 작업이 필요하다. OC로부터 추출된 클래스의 속성의 추출은 다음과 같은 기준에 의한다.

1개의 OC로부터 클래스 속성의 추출 기준은 다음과 같다.

[정의 2] 클래스 속성의 추출 기준

1) OC로부터 추출된 클래스 후보의 속성은 전역변수를 기준으로 한다. 만일 전역 변수가 존재하지 않을 경우 속성은 존재하지 않는다.

2) OC 내부에 존재하는 UDT로부터 추출된 클래스의 속성은 UDT 내부의 함목을 속성으로 한다.

클래스의 속성을 추출한 결과를 보면 클래스 사이에 존재하는 *part-of* 관계가 존재한다는 것을 발견할 수 있다.

그림 4를 보면 4개의 전역변수(*struct list *head, *tail, *current, no_of_person*)가 존재한다. 따라서 정의 2를 기준으로 속성을 추출하였을 경우, *class list_manage*는 4개의 속성(*list *head, list *tail, list *current, int no_of_person*)을 가지고 UDT로부터 파생된 클래스인 *class person*은 원래의 내부 항목(*char name[9]*)을 속성으로 가진다. *person_list*와 *list*는 *part-of* 관계를 가지며 1..3의 차수(Cardinality)를 가짐을 볼 수 있다. 그림 6은 정의 2의 기준에 의하여 그림 4에 대한 클래스 속성 추출 결과이다.

```
class person {
  char name[9] ;
} ;
class list {
  void  *content ;  int  type;
  list  *next ;
} ;
class person_list {
  int  no_of_person ;
  list  *head, *tail, *current ;
}
```

그림 6 그림 4에 대한 속성 추출 결과

3.1.3 메소드의 추출

속성을 추출한 후 OC에 남아 있는 부분은 함수이다. 남아있는 함수는 OC로부터 파생된 클래스의 메소드로 존재하든지 UDT로부터 파생된 클래스의 메소드로 존재해야 한다. 메소드의 클래스에 대한 소속의 결정은 OC 내부에 존재하는 관계성의 종류와 방향성을 이용한다.

함수, UDT와 전역변수와 이들간의 관계를 그래프로 표현할 경우, OC 내부에 존재한 에지는 함수들간의 관계, 함수와 변수의 관계, UDT와 변수의 관계와 UDT와 함수의 관계의 4종류가 존재한다. 이러한 4종류의 관계를 방향성을 고려하여 세분화할 경우, OC에는 총 7가지의 에지가 있다. 절차지향 프로그램에서 함수는 객체지향 프로그램의 클래스의 메소드로 변경된다. 따라서 메소드를 추출하기 위해서는 함수와 관련된 에지만을 고려하면 된다. 함수와 관련된 에지는 5가지가 존재한다.

1개의 OC로부터 클래스 메소드의 추출 기준은 다음과 같다. 이러한 추출 기준은 OC가 UDT를 가지고 있을 경우 UDT로부터 추출된 클래스의 소속으로 될 것인지 OC로부터 추출된 클래스의 소속으로 될 것인지에 대한 결정사항이다.

[정의 3] 함수와 관련된 에지의 종류/의미 및 클래스 메소드의 추출 기준

1) OC가 UDT를 포함하고 있지 않을 경우 OC로부터 파생된 클래스만 존재한다. 따라서 모든 함수는 OC로부터 추출된 클래스의 메소드이다.

2) OC가 UDT를 포함하고 있다면 파생된 여러 개의 클래스가 존재한다. 따라서 함수는 UDT로부터 파생된 클래스의 소속이거나 OC로부터 파생된 클래스의 메소드로 존재한다. 메소드를 추출하기 위하여 5가지 에지는 순차적으로 적용해야 한다. 여러 개의 관계성이 혼합되어 존재할 경우 첫 번째로 적용될 수 있는 관계성의 기준에 따른다. 이 것은 각각의 에지는 함수가 객체 지향 프로그램의 메소드로 변경될 때 특별한 의미를 표현하기 때문이다.

i) 함수가 변수를 참조 : 함수에서 변수의 값을 변경하지 않고 참조(use)하는 것이다. 변수는 전 단계에서 이미 어떠한 클래스의 속성으로 선정이 되었다. 함수는 변수가 소속된 클래스의 메소드로 포함되어야 한다. 이러게 추출된 메소드는 일반적으로 메소드 이름에 get 접두어를 가지며 클래스에 소속된 일부 속성의 값을 반환하는 경우이다.

ii) 함수가 변수를 조작 : 함수에서 변수의 값을 참조하지 않고 기록(manipulate)하는 것이다. 함수는 변수가 소속된 클래스의 메소드로 포함된다. 이렇게 추출된 메소드는 일반적으로 메소드 이름에 set(혹은 put) 접두어를 가지며 클래스에 소속된 일부 속성의 값을 변경하는 경우이다.

iii) 함수가 UDT를 반환 : 함수가 UDT를 반환하는 것이다. 함수는 UDT로부터 추출된 클래스의 메소드로 포함된다. 이러한 메소드는 클래스의 인스턴스를 반환하는 경우이다.

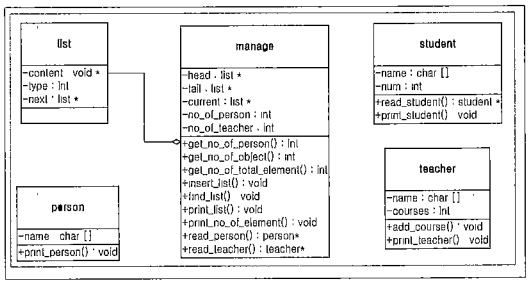
iv) 함수가 함수를 호출 : 호출 함수(caller)가 다른 함수(호출된 함수, callee)로부터 호출하는 것이다. 호출된 함수는 위의 기술된 기준("1)~3)")에 의하여 이미 어떠한 클래스의 메소드로 포함되어 있다. 호출함수는 호출된 함수가 소속된 클래스와 동일한 클래스의 메소드로 포함된다. 호출된 함수는 클래스의 속성을 일차적으로 참조/변경하는 메소드의 경우이고 호출 함수는 호출된 함수를 이용하여 이차적으로 자료를 가공하는 메소드이다.

v) 함수 내부에서 UDT를 사용 : 함수에서 UDT의 변수를 지역 변수로 사용하는 것이다. 함수는 UDT로부터 생성된 클래스가 아닌, UDT와 포함관계(*part-of*)를 가진 컨테이너(container) 클래스의 메소드로 포함되어야 한다.

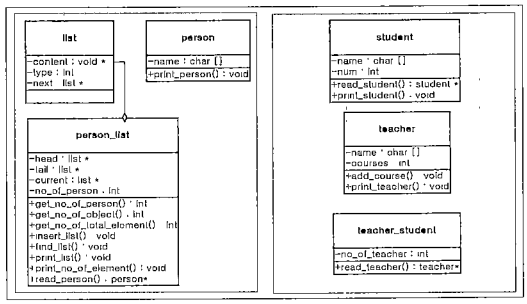
그림 6에 대하여 정의 3을 적용한 메소드 추출한 결과는 그림 7과 같다.

```
class person {
  char name[9] ;
  void print_person(struct person *per) ;
} ;
class list {
  void  *content ;  int  type;
  list  *next ;
} ;
class person_list {
  int  no_of_person ;
  list  *head, *tail, *current ;
  void insert_list( int sel_type ) ;
  void find_list( void ) ;
  void print_list( void ) ;
  int get_no_of_object( int aType ) ;
  int get_no_of_person() ;
  int get_no_of_total_element( void ) ;
  struct person *read_person( void ) ;
  void print_no_of_element( void ) ;
}
```

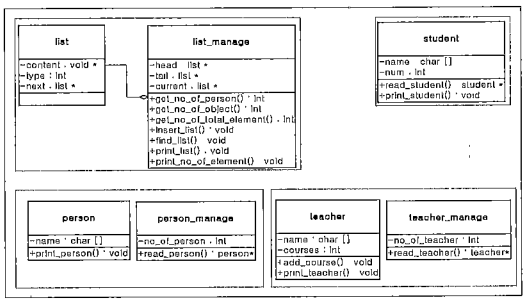
그림 7 그림 4에 대한 클래스의 메소드 추출 결과



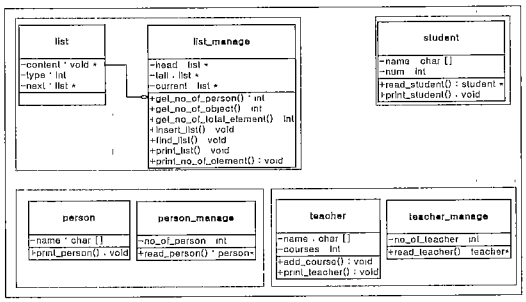
a) G_1에 대한 결과



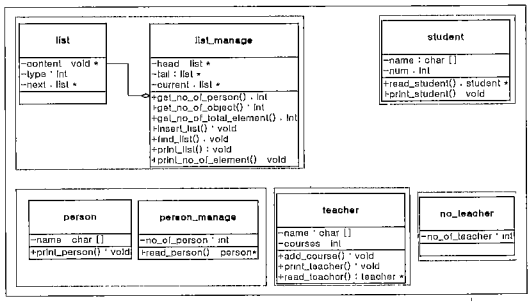
b) G_2에 대한 결과



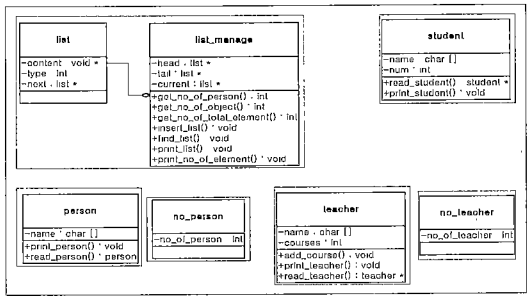
c) G_3에 대한 결과



d) G_4에 대한 결과



e) G_5에 대한 결과



f) G_6에 대한 결과

그림 8 표 1에 대한 클래스 추출결과

그림 8은 G^{RA} 전체에 대한 클래스 추출 결과이다.

그림 8의 각각의 그림("a)~f)")은 OCG를 의미하고, OCG안의 가는 점선은 OC를 의미한다. 클래스 추출 결과를 보면 6개의 계층으로부터 5~7개의 생성함을 볼 수 있다. 또한 클래스의 수가 동일할 지라도 클래스의 세부사항에 미세한 차이가 존재하는 것을 볼 수 있다.

3.2 상속성 추출

절차 중심 SW를 객체 중심 SW로의 재공학 절차의 세 번째 단계는 상속성 추출이다. 클래스를 추출과 마찬가지로 상속성 추출도 크게 두 분류로 구분할 수 있다. 첫 번째 경우는 상속성이 존재하는 영역 모델을 이용하여 다음 단계로 진행하는 것이다. 두 번째 경우는 원시 프로그램으로부터 추출된 클래스로부터 상속성을 추출하여 다음 단계로 계속 진행하는 것이다. 어떠한 모델을 기준으로 사용할 것인지에 대한 논의는 클래스 추출에서와 마찬가지이다. 따라서 본 논문에서는 원시 프로그램으로부터 추출된 클래스를 기준으로 삼는다.

상속성 추출 과정은 클래스 추출 과정과 마찬가지로 상속성을 구할 대상 클래스의 결정, 상속될 속성의 결정과 상속될 메소드의 결정 순서로 이루어진다.

상속성 추출에는 영역 전문가의 개입이 필요하다. 첫째, 클래스들간의 유사도를 비교하기 위한 기준을 영역 전문가가 선정하여야 한다. 둘째, 상속 구조를 가진 영역 모델링과 상속성을 가진 클래스 구조를 비교하여 유사도가 측정이 될 경우 높은 유사도를 가진다고 잘 변환된 프로그램이라 할 수만은 없다. 이 것은 객체추출 단계의 결정 과정에서와 마찬가지로 상속성을 가진 영역 모델링이 원시 프로그램보다 세밀한 부분에 대한 정보를 가지고 있지 않기 때문이다. 영역 모델링은 단지 개략적인 정보만을 가지고 있다. 따라서 상속성을 가진 영역 모델링이 결정하지 못하는 부분은 영역 전문가가 결정하여야 한다.

3.2.1 상속성 추출 대상 클래스의 선정

객체 지향 프로그램에서 상속성을 가진 클래스들은 모듈화가 잘 된 절차지향 프로그램과 마찬가지로 응집도는 높지만 결합도가 낮을 것이다. 논문의 예에서 제시된 *person, teacher*와 *student*의 속성만을 고려한다면 *teacher*와 *student*는 *person*으로부터 상속받는 것을 알 수 있을 것이다. 그러나 *person, teacher*와 *student*과 이들과 연결된 함수들(*read_person, print_person, add_course, read_teacher, print_teacher, read_student, print_student*)만을 고려한다면 3개의 UDT들 간의 연관성은 전혀 존재하지 않는 것을 볼 수 있다. 따라서 그래프에서 객체 후보간에 많은 연관관계(즉, 에지)가 존재하기 때문에 이들간의 클래스 상속 구조를 추출한다는 것은 잘못된 것이다. 반대로 클래스 사이에 연관관계가 많이 존재한다면 두 클래스 사이에는 어떠한 관계가 존재한다는 것을 알 수 있다. 따라서 두 클래스 사이에 상속 관계가 존재하는지를 검사할 필요성도 있다. 이러한 연관 관계의 많고 적음에 따라 상속성을 추출한다면 논란의 여지가 많다. 본 논문에서는 클래스 후보들간의 연관 관계를 고려하지 않고 모든 클래스 후보들간의 상속성을 추출하도록 한다.

상속성을 추출할 때 첫 번째 과정은 상속성의 추출을 하기 위한 비교 대상의 선정에 관한 것이다. CCs 내부에서 상속성을 추출하고 CCs 외부간의 상속성을 추출할 수도 있고, 반대로 외부간의 상속성을 추출하고 내부의 상속성을 추출할 수도 있다. 그러나 CCs와 CCs 사이의 상속성을 추출하는 것만으로도 G^{RA}의 전체적인 측면에서 본다면 CCs 내부에서 상속성을 추출하는 것과 동일한 결과를 발생한다. 따라서 CCs와 CCs 사이에서 추출하는 방법을 사용하도록 한다.

상속성 추출은 두 개의 클래스를 비교하여 공통된 부분을 찾아 공통된 부분이 상위 클래스로 나머지부분이 각각의 하위 클래스로 된다. 두 개의 클래스를 비교하는 방법은 객체 추출과정의 결정 단계에서 사용한 공식[18]을 사용할 수 있다. 결정 단계는 영역 전문가가 생성한 클래스와 클러스터링된 객체 후보간의 비교이다. 이러한 비교는 객체후보와 클래스가 유사한 구조를 가진다는 가정에서 출발하였다. 따라서 클래스 추출 단계에서 생성된 클래스들간의 유사도를 구하는 방법은 동일하기 때문에 객체 추출단계의 결정 과정의 공식을 적용할 수 있다.

유사도의 수치를 이용하여 상속성을 추출할 클래스들이 정해지면 속성과 메소드가 채워지지 않은 임시 클래스를 생성한다. 임시 클래스의 이름은 의미를 부여해야 하기 때문에 자동적으로 명명하는 것은 문제가 있다. 따라서 영역 전문가가 명명함으로써 클래스에 의미를 부여할 수 있다.

예제 프로그램에 적용할 경우, G_1은 1개의 CCs만이 존재하기 때문에 상속성을 추출할 대상이 존재하지 않는다. G_2, G_3, G_4, G_5, G_6은 여러 개의 CCs가 존재하기 때문에 상속성을 추출할 대상이 존재한다. 이들에 대한 유사성을 비교할 경우 *person, student*와 *teacher* 클래스가 유사도가 높은 것으로 판명되었다. 따라서 이들을 대상으로 상속될 속성 및 메소드를 추출한다.

3.2.2 상속될 속성의 추출

속성의 추출에서 기준이 되는 것은 두 클래스 사이의 유사도의 정도이다. 두 클래스 사이의 유사도가 높은 것에서 상속성을 추출할 수도 있고 반대로 유사도가 낮은 것에서 추출할 수도 있다. 단지 차이가 있다면 높은 유사도를 가진다는 것은 상위 클래스로 존재할 부분이 많아진다는 것이고, 반대로 낮은 유사도를 가진다면 상위 클래스로 존재할 부분이 적어진다는 것이다. 낮은 유사도를 가져도 상속성이 존재할 수 있다. 이러한 경우는 일반적으로 여러 단계의 상속 과정을 거친 경우일 것이다. 따라서 직접적인 상속 관계를 가질 경우는 유사도가 높을 것이다. 따라서 두 클래스 사이의 유사도가 영역 전문가가 결정하는 임계값을 초과하는 것을 대상으로 한다.

객체 추출의 결정과정의 공식과 기준을 적용하여 예제 프로그램에 적용할 경우, *person, student*와 *teacher* 클래스의 *name* 속성이 동일한 것으로 판명되었다. 따라서 *name*이 상위 클래스의 속성으로 결정이 된다.

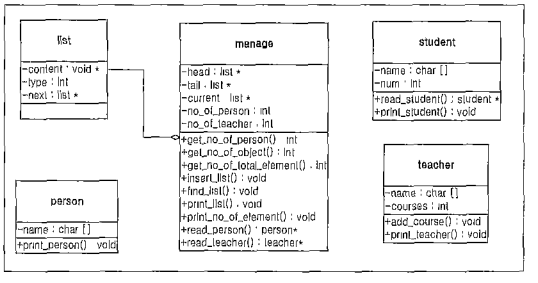
3.2.3 상속될 메소드의 추출

속성들의 공통된 부분에 대하여 상위 클래스의 속성으로 추출한 후 메소드에 대한 상속성을 추출한다. 메소드의 상속성은 상위 클래스에 소속될 속성들 만을 사용하는 메소드로 한정한다. 만일 상위 클래스로 소속될 속

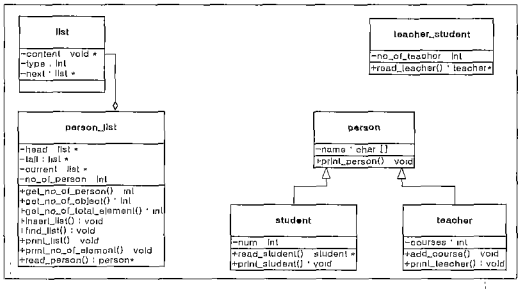
성들만이 아닌 하위 클래스의 속성을 사용한다면 하위 클래스의 메소드가 되어야 한다.

예제 프로그램에 적용할 경우, 속성 *name*을 접근하는 메소드는 *person* 클래스의 *read_person()*과 *print_person()*, *student* 클래스의 *read_student()*와 *print_student()*, *teacher* 클래스의 *read_teacher()*와 *print_teacher()*가 있다. 이들 메소드중 다른 속성에 접근하지 않고 오직 속성 *name*에 접근하는 것은 *read_person()*과 *print_person()*이다. 따라서 2개의 메소드가 상위 클래스의 메소드로 존재한다.

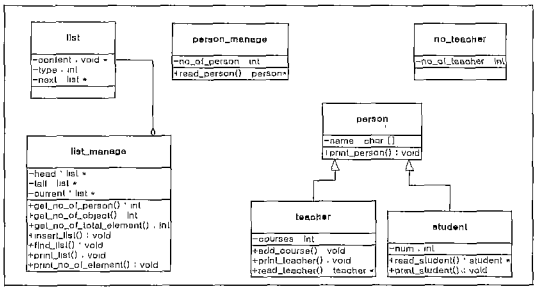
역 비순환 그래프 전체에 대한 상속성 추출 결과는 그림 9와 같다.



a) G_1의 상속성 추출결과



b) G_2의 상속성 추출결과



c) G_6의 상속성 추출결과

그림 9 상속성 추출결과

G^{RA}로부터 계층구조를 계속적으로 유지하여 상속성을 추출한 클래스 그래프($G^{RA:CCG}$)는 상위 계층(G_1)로 갈수록 위로 갈수록 상속의 정도가 낮아지며 하위 계층으로 갈수록 상속의 정도가 높아짐을 볼 수 있다. 예제는 간단한 프로그램이기 때문에 G_1, G_2를 제외한 제외한 4개의 그래프(G_3, G_4, G_5, G_6)가 거의 유사한 형태를 가진다. G_3, G_4, G_5, G_6은 클래스 이름과 몇 개의 메소드의 소속의 차이가 있다. 만일, 커다란 프로그램을 적용한다면 상이한 결과가 많이 발생할 수 있다.

3.3 영역 모델링과의 비교

이 절의 목적은 전과정에서 생성된 상속 관계를 가진 클래스들과 *is-a* 관계와 *part-of* 관계(즉, 상속관계)가 존재하는 영역모델링(G^{HM})과의 유사도를 측정하여 적합한 클래스를 선택할 수 있도록 영역 전문가에게 제시하는 것이다.

G^{HM}와의 유사도 비교는 클래스들 사이의 유사도뿐만 아니라 상속성에서 존재하는 성질들인 *is-a/part-of* 관계에 따른 유사도도 함께 비교해야 한다. 객체 지향 프로그램에서 상속관계에 대한 정보는 객체지향적 특성을 반영하는 대표적 성질이므로 이들을 비교 기준으로 추가한다. 따라서, 유사도 추출은 클래스 사이의 유사도와 상속 관계를 고려한 유사도가 모두 첨가되어한다. 이러한 유사도 측정 공식은 다음과 같다 :

(1) $S^{CCG_i} = \sum_{m=1}^{k} \sum_{n=1}^{l} \left(S_{C_{(m,n)}}^{CCG_i} + S_{I_{(m,n)}}^{CCG_i} \right)$

(2) $S_{CIG}^{CCG_i} = \sum\sum \left(S_{C_{(m,n)}}^{CCG_i} + S_{I_{(m,n)}}^{CCG_i} \right)$

이 곳에서 (1)의 경우는 $S_{C_{(m,n)}}^{CCG_i}$은 클래스 후보와 영역 모델 클래스의 유사도, $S_{I_{(m,n)}}^{CCG_i}$은 클래스 후보와 영역 모델 클래스의 상속 관계를 고려한 유사도, k는 CCG의 클래스 수, l은 영역 모델의 클래스 수를 말한다. (2)의 경우, $S_{C_{(m,n)}}^{CCG}$ 와 $S_{I_{(m,n)}}^{CCG_i}$의 유사도는 상속성이 존재하는 클래스들의 유사도 값이다.

(1)의 경우(S^{CCG_i})는 CCG 전체, 즉 하나의 계층 단위로 유사도를 비교하여 선택하는 방법이고, (2)($S_{CCG_i}^{CIG}$)는 $G^{RA:CCG}$의 각 계층에서 상속성이 존재하는 클래스들의 모임을 개별적으로 선택하는 방법이다.

$S_C^{CCG_i}$는 객체 추출 단계의 결정단계에서 사용했던 DSS(*Degree of Signature Similarity*) 공식[18]을 적용하였다. $S_I^{CCG_i}$는 상속성이 존재하는 클래스들의 관계를 고려한 유사도로써, 영역 모델링과 CCG의 클래스들 간의 클래스들의 상속정도를 수치화 하여 값으로 표현한 것으로, *is-a* 관계와 *part-of* 관계를 고려한 유사도

이다. 상속 관계를 고려한 클래스의 유사도 공식은 다음과 같다 :

$$S_I^{CCG_i} = S_C^{CCG_i} * \left(\frac{Min(S_{CI}^{CCG_i}, S_{CI}^{HM})}{Max(S_{CI}^{CCG_i}, S_{CI}^{HM})} \right)$$

이 곳에서 $S_C^{CCG_i}$은 클래스 후보와 영역 모델 클래스의 유사도, $S_{CI}^{CCG_i}$은 클래스 후보의 상속 비율, S_{CI}^{HM}은 영역 모델 클래스의 상속 비율을 말한다.

$S_C^{CCG_i}$는 상속관계가 있는 클래스와 G^{HM} 클래스간의 유사도의 값을 사용한다. $S_{CI}^{CCG_i}$는 클래스에서 상속성이 존재하는 클래스들 간의 공통적인 부분을 산출한 값이며, S_{CI}^{HM}는 G^{HM}의 값이다. 이들 클래스들의 공통된 부분을 구하는 공식은 다음과 같다 :

$$S_{CI}^{CCG_i} = \frac{CCG_{CI}}{CCG_C + CCG_{CI}} \qquad S_{CI}^{HM} = \frac{DM_{CI}}{DM_C + DM_{CI}}$$

이 곳에서 CCG_{CI}은 상속하는 상위 클래스 후보 상속 비율, DM_{CI}은 상속하는 상위 영역 모델 클래스 상속 비율, CCG_C은 상속받는 하위 클래스 후보 상속 비율, DM_C은 상속받는 하위 영역 모델 클래스 상속 비율을 말한다. CCG_{CI}와 DM_{CI}는 상속하는 상위 클래스, CCG_C와 DM_C는 상속을 받는 하위 클래스, 이들간의 값은 CCG_{CI}, CCG_C, DM_{CI}, DM_C, 모두 개개의 클래스의 속성과 메소드의 개수를 더한 값이다. $Min(S_{CI}^{CCG_i}, S_{CI}^{HM})$과 $Max(S_{CI}^{CCG_i}, S_{CI}^{HM})$는 각각 이 들간의 최소와 최대값을 나타낸다.

이러한 방법으로 개개의 클래스 후보군과 영역 모델 클래스간의 상속 관계를 고려한 유사도를 계산한 값을 적용하여, 클래스 후보군과 영역모델간의 상속 관계를 고려한 유사도를 구할 수 있다.

그림 10은 비교를 하기위한 영역 모델링을 UML 표기법을 사용하여 표현하였다.

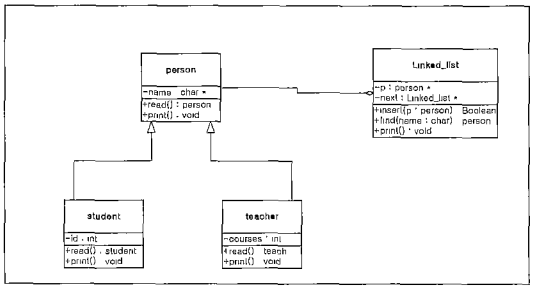


그림 10 영역 모델링

$S_C^{CCG_i}$의 공식을 적용하여, 상속성을 추출한 클래스와 영역 모델링과의 유사도 비교한 결과 값은 표 2와 같다.

표 2는 DSS 공식을 적용한 상속성이 존재하는 클래스와 영역모델의 클래스들간의 유사도(S_C) 비교이다. S_C는 0에서 1사이의 값을 가지며 수치가 높을수록 클래스들 간의 유사도가 높음을 알 수 있다. 예를 들어, $G_{1.4}$의 *person_list*와 영역모델의 *Linked_List*의 유사도 수치가 0.5인 경우, 이 클래스들간의 유사도가 높음을 알 수 있다.

표 2 S_C

CCG \ 영역모델		person	teacher	student	Linked_List
G_1	person	0	0	0	0
	teacher	0	0	0	0
	student	0	0	0	0
	list	0	0	0	0
	manage	0.06	0.08	0.07	0.38
G_2	person	0	0	0	0
	teacher	0.14	0.44	0.12	0.08
	student	0.13	0.19	0.31	0.06
	list	0	0	0	0
	person_list	0.07	0.16	0.08	0.39
	teacher_studen	0	0	0	0
G_3	person	0	0	0	0
	teacher	0.14	0.44	0.12	0.08
	student	0.13	0.14	0.31	0.06
	list	0	0	0	0
	person_list	0.07	0.08	0.08	0.5
	teacher_student	0	0	0	0
G_4	person	0	0	0	0
	teacher	0.14	0.44	0.12	0.08
	student	0.13	0.14	0.31	0.06
	list	0	0	0	0
	person_list	0.07	0.08	0.08	0.5
	teacher_student	0	0	0	0
G_5	person	0	0	0	0
	teacher	0.14	0.44	0.12	0.08
	student	0.13	0.14	0.31	0.06
	list	0	0	0	0
	person_list	0.07	0.08	0.08	0.5
	teacher_student	0	0	0	0
G_6	person	0	0	0	0
	teacher	0.14	0.44	0.12	0.08
	student	0.13	0.14	0.31	0.06
	list	0	0	0	0
	list_mange	0.07	0.08	0.08	0.5
	person_manage	0	0	0	0
	no_teacher	0	0	0	0

표 3-1과 표 3-2은 상속성이 존재하는 클래스들간의 상속정도를 계산한 값이다.

표 3-1 상속성이 존재하는 영역모델의 공통 비율(S_{CI})

영역 모델	person	teacher	student	Linked_List
person	0	0.5	0.5	0
teacher	0.5	0	0	0
student	0.5	0	0	0
Linked_List	1	0	0	0

표 3-2 상속성이 존재하는 CCG의 공통 비율

CCG \ 영역모델		person	teacher	student	list	manage		
G_1	person	0	0	0	0	0		
	teacher	0	0	0	0	0		
	student	0	0	0	0	0		
	list	0	0	0	0	0		
	manage	0	0	0	1	0		
		person	teacher	student	list	person_list	teacher_student	
G_2	person	0	0.4	0.4	0	0	0	
G_3	teacher	0.4	0	0	0	0	0	
G_4	student	0.4	0	0	0	0	0	
G_5	list	0	0	0	0	0	0	
	person_list	0	0	0	1	0	0	
	teacher_student	0	0	0	0	0	0	
		person	teacher	student	list	list_manage	person_manage	no_teacher
G_6	person		0.33	0.4	0	0	0	0
	teacher	0.33	0	0	0	0	0	0
	student	0.4	0	0	0	0	0	0
	list	0	0	0	0	0	0	0
	list_mange	0	0	0	1	0	0	0
	person_manage	0	0	0	0	0	0	0
	no_teacher	0	0	0	0	0	0	0

표 4는 클래스와 영역 모델의 클래스간의 상속 관계를 고려한 유사도이다. 예를 들어, G_4의 *teacher*와 영역 모델의 *teacher* 클래스의 유사도가 0.35의 수치가 나오므로 이들이 서로 상당히 유사한 상속 관계를 가짐을 알 수 있다.

3.4 결정

결정 단계의 목적은 상속성 추출 단계에서 추출한 상속성을 가진 클래스와 *SW*의 요구사항으로부터 전문가에 의하여 구축된 상속성을 가진 영역모델과 유사도를 비교하여 적절한 상속 구조를 가진 클래스를 선택하도록 하는 것이다.

영역 전문가에게 제시된 유사도가 높다고 항시 좋은 객체지향 구조일 수는 없다. 이 것은 영역 모델링을 어느 정도로 상세히 하는가에 따라 달라 질 수 있다. 따라서 영역 전문가에게 다양한 선택기회와 이에 대한 유사

표 4 S_I

CCG \ 영역모델		person	teacher	student	Linked_List
G_1	person	0	0	0	0
	teacher	0	0	0	0
	student	0	0	0	0
	list	0	0	0	0
	manage	0	0	0	0.37
G_2	person	0	0	0	0
	teacher	0	0.35	0.09	0
	student	0	0.11	0.25	0
	list	0	0	0	0
	person_list	0	0	0	0.36
	teacher_student	0	0	0	0
G_3	person	0	0	0	0
	teacher	0	0.35	0.09	0
	student	0	0.11	0.25	0
	list	0	0	0	0
	person_list	0	0	0	0.5
	teacher_student	0	0	0	0
G_4	person	0	0	0	0
	teacher	0	0.35	0.09	0
	student	0	0.11	0.25	0
	list	0	0	0	0
	person_list	0	0	0	0.5
	teacher_student	0	0	0	0
G_5	person	0	0	0	0
	teacher	0	0.35	0.09	0
	student	0	0.11	0.25	0
	list	0	0	0	0
	person_list	0	0	0	0.5
	teacher_student	0	0	0	0
G_6	person	0	0	0	0
	teacher	0	0.29	0.09	0
	student	0	0.11	0.25	0
	list	0	0	0	0
	list_mange	0	0	0	0.5
	person_manage	0	0	0	0
	no_teacher	0	0	0	0

도를 제시하여 영역 전문가가 적절한 클래스를 선택하도록 해야한다.

상속성을 가진 클래스를 선택할 수 있는 방법은 두 가지가 있다. 첫 번째 방법은 수평적 관계를 이용하는 것이다. 두 번째 방법은 수직적 관계를 이용하는 것이다. 이러한 두 가지 방법을 제시함으로써 영역 전문가는 이차원적인 결정방법을 사용할 수 있다.

3.4.1 수평적 결정방법

수평적 결정 방법은 G^{RA}에 존재하는 1개의 계층(CCG)을 선택하는 것이다. 하나의 계층은 프로그램 전체에 대한 뼈대 코드 정보를 가지고 있다. 따라서 수평적 결정 방법을 사용할 경우 간단히 프로그램 전체 뼈

대 코드가 나오고 전체적으로 적절한 뼈대 코드를 생성할 수 있다는 장점이 있다. 그러나 미시적 측면에서 관찰해볼 경우 상속성을 가진 클래스들의 일부분은 영역 전문가가 원하지 않는 형태일 수 있다.

표 5는 상속성 추출 결과와 영역 모델(G^M)의 비교 결과이다.

표 5 $G^{RA\ CCG}$와 G^M의 유사도(S^{CCG})

CCG	S^{CCG}
G_1	0.9555
G_2	3.2479
G_3	3.2479
G_4	3.2479
G_5	3.2479
G_6	3.4043

표 5를 보면, 6개의 클래스 후보군들 중에 G_6가 영역 모델링과의 유사도가 높은 것으로 나타났다. 영역 전문가는 6개의 계층중 1개(이곳에서는 G_6)를 선택하여 원시 코드의 모든 결과물인 뼈대 코드의 정보를 구할 수 있다.

3.4.2 수직적 결정 방법

수평적 결정 방법은 부분적으로 영역 전문가가 원하는 상속성을 가진 클래스를 제공할 수 없다는 단점이 없다. 이러한 단점을 보완하기 위해서는 여러 계층에서 상속성을 가진 클래스를 선택할 수 있는 수직적 결정 방법을 이용하여야 한다. 수직적 관계의 이용은 G^{RA}에 존재하는 1개의 계층(CCG)을 선택하는 것이 아닌 여러 개의 계층(CCG)에서 부분적으로 선택하여 전체 뼈대 코드를 생성하는 방법이다. 수직적 결정 방법은 영역 모델과 상속성이 존재하는 클래스들간의 비교후 선택, 비교 결과로부터 중복된 정보 문제의 해결, 원시 프로그램의 정보에는 존재하지만 비교 결과에는 존재하지 않는 생략된 정보 문제의 해결순으로 이루어 진다.

수직적 결정 방법은 개별적인 계층(CCG)에서 부분적 클래스를 선택하는 것은 적절한 클래스의 형태와 상속성을 구할 수 있다는 장점이 있다. 클래스 선택은 1개의 클래스를 선택하는 것이 아닌 상속성이 존재하여 클래스들의 집합, 즉 상속 관계로 연결된 여러 개의 클래스를 선택한다. 이렇게 클래스들의 집합을 선택하는 것은 객체지향의 중요한 정보인 관계성을 보존할 수 있기 때문이다.

수직적 결정 방법을 사용하기 위해서는 표 6과 같이 역 비순환 그래프의 상속성이 존재하는 클래스와 상속성이 존재하는 영역 모델(G^M)의 비교 결과 중 관계(*is-a, part-of*)가 존재하는 클래스들의 집합을 비교한 결과를 구하여야한다.

표 6 $G^{RA\cdot CCG}$와 G^M의 유사도(S^{CCG}_{CIG})

CCG \ 영역모델		student	teacher	person	Linked_List
G_1	person	0	0	0	0
	teacher	0	0	0	0
	student	0	0	0	0
	list	0	0	0.8047	
	manage	0	0		
G_2	person	2.0896			0
	teacher				0
	student				0
	list	0	0	0.7831	
	person_list	0	0		
	teacher_student	0	0	0	0
G_3	person	2.0896			0
	teacher				0
	student				0
	list	0	0	0.7831	
	person_list	0	0		
	teacher_student	0	0	0	0
G_4	person	2.0896			0
	teacher				0
	student				0
	list	0	0	0.7831	
	person_list	0	0		
	teacher_student	0	0	0	0
G_5	person	2.0896			0
	teacher				0
	student				0
	list	0	0	0.7831	
	person_list	0	0		
	teacher_student	0	0	0	0
G_6	person	2.0308			0
	teacher				0
	student				0
	list	0	0	1.0658	
	list_mange	0	0		
	person_manage	0	0	0	0
	no_teacher	0	0	0	0

위 표 6을 보면 영역 모델과 가장 유사도가 높은 상속성이 존재하는 클래스로는 G_2, G_3, G_4, G_5의 *person, teacher, student*와 G_6에서 *list*와 *list_manage*가 있는 것을 관찰할 수 있다. 따라서 이들 클래스들의 집합이

일차적으로 변환될 코드에 대한 코드 정보로 선택된다.

그림 11의 굵은 선 부분은 수직적 결정 방법에 의하여 선택된 상속성이 존재하는 클래스들의 집합을 도시한 것이다.

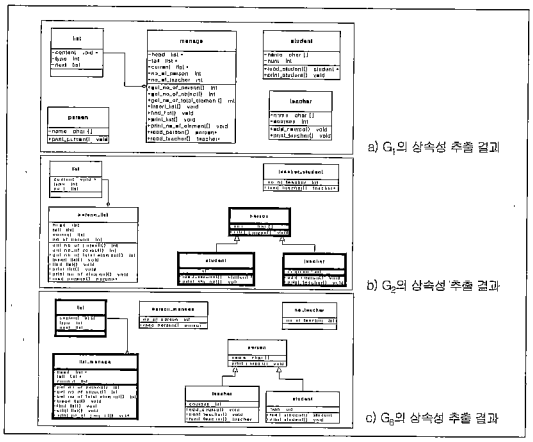


그림 11 상속성 관계를 이용한 수직적 결정 방법의 예

수직적 결정 방법에서 원시 코드와 선택된 클래스 사이에 발생하는 두 가지 문제점이 있다. 첫 번째는 클래스 선택을 여러 계층에서 선택하기 때문에 발생하는 중복 문제이다. 두 번째는 원시 코드는 존재하지만 선택된 클래스에는 존재하지 않는 부분인 생략된 정보 문제이다.

첫 번째 문제인 중복문제에 대한 해결방안은 두 가지가 존재한다. 첫 번째로는 상위 클래스 1개가 여러 하위 클래스를 가지고 있고 하위 여러 클래스와의 관계가 있는 클래스와 중복이 된다면 하위 클래스를 선택한다. 이는 낮은 계층에 있는 클래스가 좀더 정제된 클래스이기 때문이다. 두 번째로 상위 1개의 클래스가 하위 1개의 클래스와 중복이 된다면, 즉 하위 클래스의 서브클래스들이 존재하지 않는다면, 각 계층간의 클래스들의 관계가 높은 클래스를 선택한다. 관계가 높다는 것은 클래스들이 서로 밀접한 연관성이 있다는 것이기 때문이다. 사용된 예제의 경우는 소규모의 프로그램이기 때문에 중복 문제가 발생하지 않았다.

두 번째 문제인 생략된 정보 문제는 원시코드에는 존재하지만 역 비순환 그래프로부터 선택된 클래스들에는 존재하지 않을 경우이다. 따라서 이러한 부분은 추가되어야 한다. 생략된 정보 문제는 역 비순환 그래프의 최하위 계층으로부터 상위 계층으로 관계성을 이용하여 항해함으로써 해결할 수 있다. 생략된 정보 문제 해결을 위한 알고리즘은 다음과 같다.

알고리즘 1 생략된 정보 문제 해결 알고리즘

입력 : 원시 프로그램 정보, 비교 결과 선택된 클래스들
출력 : 원시 프로그램의 모든 정보가 포함된 클래스

1) 원시 프로그램 정보와 선택된 비교결과를 비교
2) 1)의 결과로 생략된 정보가 존재할 경우 다음을 실행
   i) 역 비순환 그래프의 최하위 계층에서 생략된 정보를 발견
   ii) 생략된 모든 정보들이 클래스를 형성하면 클래스를 선택하고 3)으로 이동
   iii) 생략된 정보가 클래스의 부분일 경우 역 비순환 그래프의 상위 계층으로 이동
   iv) ii)로 이동
3) 생략된 정보 문제를 해결함

그림 12의 빗금친 부분(*no_of_person, no_of_teacher, read_person(), read_teacher()*)은 원시 프로그램과 일차적으로 선택된 클래스들간을 비교할 경우 생략된 부분이다. 생략된 정보중 *no_of_person*과 *read_person()*은 최하위 계층에서 하나의 클래스를 형성하였으므로 G_6에서 *person_manage* 클래스를 선택한다. 생략된 정보중 *no_of_teacher*과 *read_teacher()*은 최하위 계층(G_6)에서 다른 클래스의 부분(혹은 독립적 부분)으로 존재하기 때문에 이들은 상위 단계로 항해하면서 클래스를 생성하여야 한다. *no_of_teacher*과 *read_teacher()*은 G_2에서 1개의 클래스를 형성하므로 G_2에서 *teacher_student* 클래스를 선택한다.

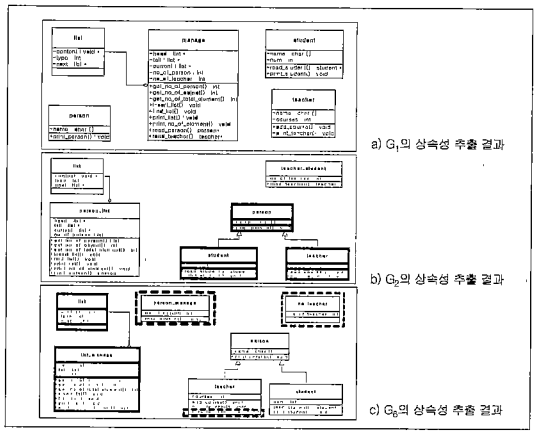


그림 12 생략된 정보 문제의 예

수직적 결정 방법을 이용하여 상속성이 존재하는 클래스의 결정 결과는 그림 13과 같다. 선택된 부분으로부터 재공학될 프로그램의 전체 뼈대 코드를 형성할 수 있다.

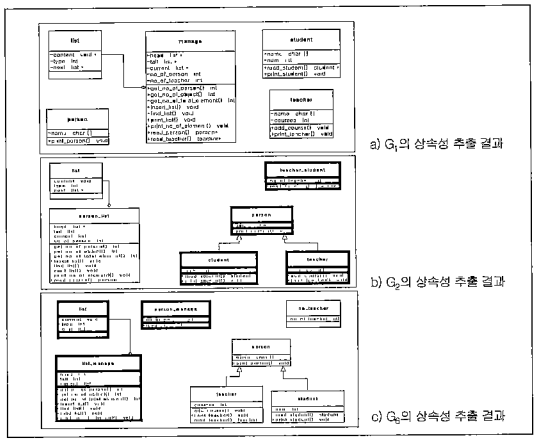


그림 13 수직적 관계를 이용한 클래스의 결정 결과

4. 실험 및 분석

본 논문의 접근 방법에 대한 가능성과 유연성을 증명하기 위해 몇 가지 실험이 수행되었다. 표 5는 무작위로 선택된 입력 *PSW(절차지향 소프트웨어, Procedural Software)*를 기술한 것이다. 표에서 전처리는 기본적인 프로그램의 정보와 객체 추출 단계의 결과를 나타낸다. 아래의 자료는 다음의 네가지 결과로 나타난다: 1) *PSW*의 크기에 따라 전처리와 클래스 추출 단계에서 많은 클러스터링의 변화가 존재하고, 2) 표에서 메소드 클러스터링 부분의 값을 살펴보면, 상대적으로 *PSW*의 크기는 클래스 후보군의 경우의 수를 결정하기 위한 주요 요인이 아니며, 3) 주요한 결정요인은 *TVCC*[1]의 클러스터링의 수이며, 4) 전역 자료형의 수와 새로운 수퍼 클래스의 수간에 강한 의존성이 존재한다. 자료는 클러스터링과 클래스의 수가 상대적으로 전체적인 *PSW*의 노드 수에 비해 작고 다루기 쉽다는 것을 나타내고 있다.

표 7 실험 자료

PSW			Dfs.c	PM.c	Chory.c	Os.c
전처리		크기(Line of Code)	195	300	1065	1374
		전체 노드 수	147	254	817	1279
		자료형의 수	2	4	4	1
		전역 변수의 수	5	4	13	17
		함수의 수	9	15	40	14
		클러스터의 수	14	20	51	17
		TVCC 클러스터의 수	5	5	11	3
		함수 클러스터의 수	9	15	40	14
클래스 추출	속성 클러스터	클러스터의 수	10~14	16~20	41~51	15~17
상속성 추출	메소드 클러스터	클러스터의 수	1~5	1~5	1~11	1~3
	상속성	새로운 수퍼 클래스의 수	0	1	1	0

5. 결론 및 향후연구과제

SW는 제작된 후 지속적인 유지·보수가 필요하다. 개발된 시스템들은 사용자의 새로운 요구, 새로운 기술의 개발, 새로운 환경에 맞도록 수정 보완되어야 하며 이는 막대한 비용을 필요로 한다.

이러한 흐름에 맞추어 시스템을 재공학할 때 유지·보수 및 재사용 측면에서 기존의 방식들보다 유리한 객체지향 파라다임을 적용하여 기존의 시스템으로 재개발한다면, 소프트웨어 생산성을 향상시킬 수 있고, 소프트웨어의 유지·보수 비용을 절감할 수 있으며, 시스템에 새로운 요구를 수용할 수 있게 되는 등 많은 장점을 가지게 된다.

절차지향 프로그램을 객체지향 프로그램으로 재공학하는 세 번째 단계인 클래스 추출 단계에서는 클러스터링된 객체 후보들의 공통적인 특성을 추출하여 클래스를 추출한다. 클래스 추출 단계의 결과는 클래스들간에 대등한 평면 관계를 이루고 있다. 네 번째 단계인 상속관계 추출 단계는 전 단계에서 추출된 평면화된 클래스들의 공통적 특성을 추출하여 *part-of* 관계나 *is-a* 관계를 추출하여 계층 구조를 만든다. 클래스와 상속성 추출 단계를 거침으로 뼈대 코드를 형성할 수 있다.

본 논문의 목적은 영역 전문가에게 영역 모델과 가장 유사도가 높은 클래스 추출 및 상속성 추출을 제시하는 것이 아니라, 비록 최적은 아니라도 영역 전문가에게 다양한 선택 기회를 제시할 수 있도록 하는 것이다.

향후 연구로는 상속성 이후의 과정인 지속성 결정, 객체 지향 코드 생성 및 절차지향 소프트웨어와 추출된 객체지향 소프트웨어간의 동일성 검증이다.

참 고 문 헌

[1] G. Booch, Object-Oriented Development, IEEE Transaction on Software Engineering, Vol. SE-12, No. 2, pp. 211-221, Feb., 1986.

[2] 권오천, 신규상, "역공학 및 재공학의 기술 동향", 한국 정보 과학회 소프트웨어 공학 회지, 제12권, 제1호, pp. 6-21, 1999.

1) TVCC(type-based variable connected component)는 동일한 형태로 이루어진 자료형과 동일한 자료형과 연관된 변수를 1개의 노드로 클러스터링한 것을 의미한다.

[3] 김행곤, "기존 시스템에서 객체지향 시스템으로의 재공학 방법론", 박사논문
[4] J. A. Zimmer, "Restructuring for Style," Software Practice and Experience, Vol. 20, No. 4, pp. 365-389, April, 1990.
[5] W. C. Dietrich, Jr., L. R. Nackman and F. Gracer, "Saving a Legacy with Object," Proc. OOPSLA, Association for computing Machinery, N. Y. pp. 77-83, 1989.
[6] Harald C. Gall, Rene R. Klosch and Roland T. Mittermier, "Architecture Transformation of Legacy System," Technical Report Number CS95-418, Seattle, April, 1995.
[7] I. Jacobson, "Re-Engineering of Old System to an Object-oriented Architecture," Proc OOPSLA, Association for Computer Machinery, N. Y. pp. 340-350, 1991.
[8] W. J. Premerlani and M. R. Blaha, "An Approach for Reverse Engineering Relational Database," Communication of the ACM, Vol. 37, No. 5, pp. 42-49, May, 1994.
[9] P. T. Breuer and K. Lano, "Creating Specifications from Code : Reverse Engineering Techniques," J. Software Maintenance : Research and Practice, Vol. 3, pp. 145-162, 1991.
[10] H. M. Sneed and E. Nayary, "Extracting Object-oriented Specification from Procedurally Oriented Programs," 2nd Working Conference on Reverence Engineering, pp. 217-226, Jul., 1995.
[11] P. A. Hausler, M. G. Pleszkoch and L. C. Linger, "Using Function Abstraction to Understanding Program Behaviour," IEEE Software, pp. 55-63, Jan. 1990.
[12] J. Ayre, D. McFall, J. G. Higher and C. Delobel, "A Method for Re-engineering Existing Relational Database Applications for the Satisfaction of Multi-media Based Requirement," Database Reengineering and Interoperability, edited by To-yat Cheung etal., Plenum Press, New York, pp. 1-13, 1996.
[13] A. Quilici, "A Memory-based Approach to Recognizing Programming Plans," Communication of the ACM, Vol. 37, No. 5, May, 1994.
[14] A. Cimitile, A. De Lucia, G. A. Di Lucca and A. R. Fasolino, "Identifying Objects in Legacy Systems Using Design Metrics," The Journal of Systems and Software, vol. 44, No. 3, pp. 199-211. Jan., 1999.
[15] Panos E. Livadas and Theodore Johnson, "A New Approach to Finding Objects in Programs," Journal of Software Maintenance : Research and Practice, Vol. 6, pp. 249-260, 1994
[16] Doris L. Carver, "Reverse Engineering Procedural Code for Object Recovery," Conf. of Software Engineering & Knowledge Engineering, pp. 442-449, 1996
[17] G. Canfora, A. Cimitile and M. Munro, "An Improved Algorithm for Identifying Object in Code," Software-Practive and Experience, Vol. 26(1), pp. 25-48, January, 1996
[18] 박성욱, 노경주, 이문근, "최적합 객체 선정을 위한 다중 객체군 추출", 한국 정보 과학회 논문집(B), 제26권, 제12호, pp. 1468-1481, 1999.
[19] Harald Gall, Rene Klosch and Roland Mittermeir, "Object Oriented Re-Architecturing," Proc. European Software Engineering Conference, Sep., 1995.
[20] Harald Gall and Johannes Weidl, "Binding Object Models to source Code : An Approach to Object-Oriented Re-Architecturing," TUV-1841-87-14, 1998.
[21] T. Wiggerts, H. Bosma, and E. Fielt. "Scenarios for the Identification of Objects in Legacy Systems," In Preceeding of *the Fourth Working Conference on Reverse Engineering* 1997, IEEE Computer Society, 1997, pp.24-32
[22] M.Siff and T. Reps. "Identifying Modules Via Concept Analysis," In *International Conference on Software Maintenance*, ICSM97. IEEE Computer Society, 1997.
[23] M.Siff and T. Reps. "Identifying Modules Via Concept Analysis," IEEE Transaction on Software Engineering, Vol. 25, No.6, Nov/Dec 1999, pp. 749-768.
[24] R. Wille. "Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts," *Ordered Sets*, I. Rival, ed., pp.445-470, NATO Advanced Study Inst., Sept 1981.
[25] C. Lindig, G. Snelting. "Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis," In Proceeding of the 1997 *International Conference on Software Engineering*, pp.349-359.
[26] 박외진, 민상윤, 배두환, 마평수. "객체지향 재공학을 위한 객체 모델 정제기법", 한국 정보과학회 논문집(B), 제25권 10호, 1998.10, pp. 1506-1517.

최 정 란

1999년 전북대학교 컴퓨터과학과 졸업(이학사). 1999년 ~ 2001년 8월 전북대학교 전산통계학과 석사 졸업. 2001년 8월 ~ 현재 전북대학교 컴퓨터통계정보학과 박사과정중. 관심분야는 소프트웨어 재·역공학, 실시간 시스템, 운영체제, 컴파일러 등

박 성 옥

1994년 전북대학교 컴퓨터과학과 학사. 1994년 ~ 1996년 전북대학교 전산통계학과 석사. 1998년 ~ 현재 전북대학교 전산통계학과 박사과정중. 관심분야는 소프트웨어 재·역공학, 실시간 시스템, 운영체제, 컴파일러 등

이 문 근

1989년 The Pennsylvania State University, Computer Science 학과 졸업(이학사). 1992년 The University of Pennsylvania, Computer and Information Science 학과 졸업(이공학석사). 1995년 The University of Pennsylvania, Computer and Information Science 학과 졸업(이공학박사). 1992년 5월 ~ 1996년 1월, 미국, Computer Command and Control Company, Computer Scientist로 근무. 1996년 4월 ~ 1998년 3월 전북대학교 컴퓨터과학과 전임강사. 1998년 4월 ~ 1999년 2월 전북대학교 컴퓨터과학과 조교수. 1999년 3월 ~ 현재 전북대학교 전자정보 공학부 조교수. 관심분야는 소프트웨어 재·역공학, 실시간 시스템, 운영체제, 형식언어, 병렬함수언어, 컴파일러 등