객체지향개발에서의 속성 클러스터링과 클래스 계층구조생성

이 건 호†

요 약

객체지향 소프트웨어 개발 초기단계에서 클래스의 결정은 많은 객체와 관련된 속성들의 클러스터링을 하는 복잡한 문제이다. 클래스의 재사용을 위해 라이브러리에 클래스의 등록은 반복적인 시행착오에 의존하여왔다. 클래스를 등록하는 전통적인 방법과 모델링 혹은 설계단계에서 클래스와 그 계층구조의 정의를 위한 통합적인 방법에 대해 논의한다. 속성 클러스터링 문제를 위해 객체들의 속성 유사도에 근거하여 0-1 정수프로그램 위한 모형을 제시하고 또한 네트워크 기법을 이용한 클러스터링 알고리즘을 제안한다. 클래스 계층구조를 생성하기 위한 규칙을 제시하였으며 계층구조그래프 생성알고리즘을 제안한다. 본 연구결과를 이용하여 실제 현장의 문제를 사례로 제시한다.

Clustering Characteristics and Class Hierarchy Generation in Object-Oriented Development

Gun Ho Lee†

ABSTRACT

The clustering characteristics for a number of classes, and defining the inheritance relations between the classes is a difficult and complex problem in an early stage of object oriented software development. We discuss a traditional iterative approach for the reuse of the existing classes in a library and an integrated approach to creating a number of new classes presented in this study. This paper formulates a characteristic clustering problem for zero-one integer programming and presents a network solution method with illustrative examples and the basic rules to define the inheritance relations between the classes. The network solution method for a characteristic clustering problem is based on a distance parameter between every pair of objects with characteristics. We apply the approach to a real problem taken from industry.

키워드 : 클래스 계층구조(Class Hierarchy), 속성 클러스터링(Characteristics' Clustering), 클래스 분류(Class Classification), 동시개발 공정(Concurrent Development Process)

1. 서 론

최소의 개발비용으로 고품질의 우수한 소프트웨어를 짧은 기간 내에 개발하는 것은 소프트웨어 개발자나 사용자의 주요 관심사이다. 이러한 요구에 부응하기 위한 노력의 일환으로 객체지향개발기법의 소프트웨어 재사용성에 많은 연구가 이루어지고 있다. 다양한 분야에서 사용되고 있는 프로그램들은 40~60%의 서로 유사한 기능을 사용하고 있다고 한다[1]. 정확성, 신뢰성, 등 품질이 검증된 소프트웨어의 부품을 재사용함으로써 개발기간 단축, 생산성 향상 및 품질향상에 기여할 것이다.

전통적으로 객체지향 개발에는 사용자의 요구분석, 설계, 코딩, 테스팅 및 통합 등이 반복 수행되는 점증형 모형이 널리 사용되고 있다[7].

또한 기존의 클래스 결정과 계층구조를 위한 연구는 클래스의 분류가 되어 있는 상태에서 새로운 클래스를 추가하고자 할 때 기존의 계층구조를 분석, 평가하여 클래스를 삽입하거나 기존 클래스를 변형하여 클래스계층을 재구성하는 것이다[2, 12, 13]. 이러한 계층구조변형과 클래스의 수정은 개발 사이클에 따라 반복적으로 수행되어진다.

많은 문헌에서 이러한 반복 사이클을 단축시키기 위해 요구분석에서 테스트에 이르기까지 다수의 기능들의 동시개발을 강조하고 있다[14-17]. 하지만 반복 사이클로 인한 많은 비용, 시간, 노력이 소요되는데도 불구하고 개발 사이클을 반복 하게 되는 것은 대부분 소프트웨어 엔지니어에게 다수의 클래스를 동시에 결정하고 정의하기가 매우 어렵기 때문이다[18].

본 연구에서는 클래스의 라이브러리 등록단계에서 반복적인 구조변경을 최소화하기 위해서 요구분석을 위한 모델링 단계나 설계단계에서 가능한 많은 객체들과 해당되는 모든 속성들을 파악하고 그 관계를 분석하여 클래스를 결정하는 것이 바람직한 것으로 보았다.

하지만 모델링단계나 설계단계에서 요구되는 객체들과 관련된 속성들의 관계를 분석하여 속성들을 클러스터링 하여 클래스와 그 계층구조를 결정하는 방법에 대한 연구는 이루어지지 않았다.

† 종신회원 : 숭실대학교 산업·정보시스템공학과 부교수
논문접수 : 2004년 6월 8일, 심사완료 : 2004년 10월 2일

본 연구에서는 요구분석단계나 설계단계에서 요구되는 모든 객체들과 해당되는 모든 속성들과 관계를 전반적으로 동시에 고려하여 클래스의 멤버를 결정하기 위한 속성 클러스터링과 그 클래스들 간의 계층관계를 결정하기위한 최적화 방법을 제시하였다.

본 연구의 적용은 다음과 같은 개발환경에 적합하다.

① 동시개발과정[14],
② 다수 객체와 다수 클래스를 동시에 일괄처리,
③ 모델링 및 설계단계에서 객체-속성 확인가능.

2. 관련 연구

본 장에서는 소프트웨어 부품에 대한 기존의 분류방법과 이에 대한 장단점들을 살펴보고 객체지향 개발을 위한 클래스 결정방향 대해서 고찰하고자 한다.

소프트웨어 분류방법으로 Enumerative 방법, Faceted 방법, 기존 문서의 유사성 방법이 있으나 이 분류 방법에는 C 언어, Cobol 언어를 대상으로 하고 있어 객체지향 프로그램 코드에 적용은 부적당 하다[2].

2.1 Enumerative 방법

예상되는 모든 부품들을 미리 계층구조로 정의한 후에 각 부품들을 가장 적합하다고 생각되는 계층구조상의 클래스에 사용자가 직접 할당하는 방법이다[10]. 이는 클래스들 간의 계층구조들의 표현은 할 수 있으나 객체지향 재사용 라이브러리 부품의 C++ 클래스들의 일반화/상세화의 다중상속관계와 집단화의 포함관계를 표현하기 어렵고 많은 부품을 분류할 경우 분류계층도가 깊어지고 복잡하게 되면 새로운 부품들의 등록이 어려워 확장성이 좋지 않다.

2.2 Faceted 방법

서재분류 방법으로 소프트웨어 부품들의 공통적인 특성을 모은 각 Facet에서 적합한 부품들의 내용들을 선택하고 이들을 조합하여 분류하는 방법으로 1924년 Colon 분류방법이 제시되었다[4]. 이 후 1985년 Facet 모델[10]이 개발되었으나, 이 방법은 소프트웨어 확장성은 좋지만 클래스들의 계층구조를 명확하게 표현하지 못하기 때문에 상속성을 표현 할 수 없는 단점이 있다.

2.3 문서 클러스터링 방법

문서들을 분류하고 검색하는 방법으로 유사성을 이용하게 된다. 대표적인 유사성 측정방법으로는 inner-product measure 방법, cosine measure 방법, pseudo-cosine measure 방법, dice measure 방법 이 있으며 이 방법은 두 문서 i, j 사이의 가중치 w(i,j)를 결정하여 두 문서의 유사성을 평가하게 된다[10].

문서들을 클러스터링하는 방법은 Single pass 방법, Real-location 방법, Single linkage 방법, Complete linkage 방법, Average linkage 방법, Ward's Method 방법 등이 있다[10].

이들 방법은 문서들을 분류하는 방법으로 문서 파일의 내용은 단순한 단어의 집합이지만 객체지향 프로그램의 클래스 내부에는 일정한 의미를 갖는 구성요소를 갖고 있고 문서 파일 간 관계처럼 서로 독립적이지 못하고 흔히 종속적인 관계를 나타낼 수도 있기 때문에 문서정리에 사용되는 분류방법으로 객체지향 프로그램의 java, C++ 등 의 클래스 분류에는 적합하지 않다.

2.4 C++클래스 계층구조 변형

기존 클래스 라이브러리가 구축된 상태에서 새로운 클래스를 추가로 라이브러리에 등록 하고자 할 때 라이브러리에 등록된 가장 유사한 클래스를 찾아 비교분석하여 필요시 기존 클래스 구조를 변형하여 등록하는 방법이다[2]. 새로운 클래스가 추가 될 때 마다 지속적으로 시행착오를 반복해서 점진적으로 구조를 변형 개선하는 것이다. 추가로 등록되는 클래스의 구조에 따라 기존 클래스 변경이 언제나 이루어져야 하는 것으로 빈번한 클래스 구조변경을 해야 하므로 부수적인 개발업무의 유발과 개발일정 지연과 혼란을 초래 하게 되는 것이다.

3. 객체에 따른 속성 클러스터링

클래스 라이브러리는 속성들로 이루어진 수많은 클래스 그룹들로 구성되어 있으며 속성은 클래스를 특징짓는 멤버 데이터의 집합과 알고리즘, 연산, 또는 서비스라고 불리는 멤버함수들의 집합으로 구성 되어있다. 본 연구에서 두 함수가 같다는 것은 기능상으로 같으며, 함수의 인자 수 및 형, 함수의 리턴값, 참조하는 변수들의 형, 함수 내의 제어흐름, 처리 순서가 같으며, 각 변수 별로 프로그램 slicing을 수행한 결과도 같다는 것을 의미 한다[6].

클래스 정의의 문제는 유사한 객체들을 분류하여 공통적으로 사용되는 속성들을 확인하고 결정하여 속성들의 클러스터링을 통하여 클래스를 형성하는 것을 말한다.

라이브러리 형성 초기단계에서 다수의 계층을 갖는 복잡한 클래스들이 동시에 라이브러리에 추가 될 경우 매우 복잡한 문제가 될 것이다. 라이브러리 규모가 커질수록 추가되는 클래스의 수도 증가하게 되고 구조변형도 또한 복잡해 질 것이다.

대규모의 클러스터링 문제는 NP(nonpoly nomialy) - complete 문제에 해당되어 최적 해를 구하기 위해 모든 가능한 클러스터 대안들을 열거해서 찾는 것은 대형 컴퓨터라도 할지라도 거의 불가능하다[11]. 가용한 모든 클러스터 대안들의 수는 비교적 작은 규모의 문제 크기에 대해서도 천문학적일 수 있다. n개의 속성을 공집합이 아닌 부분집합 p개로 나눌 수 있는 방법의 수는 Stirling's number, $S(n, p) \approx p^n/p!$에 해당된다[5].

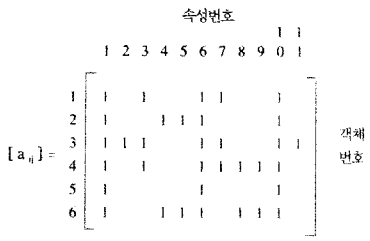
클래스와 클래스 내의 세부구조는 필요한 객체들의 조합과 각 객체의 세부구조에 의해 결정된다. 클래스를 정의하기 위하여 객체들 상호간의 연관관계를 철저히 조사, 분석하여야 한다.

소프트웨어 엔지니어는 요구분석 단계에서 필요로 하는 객체와 속성을 추출하게 된다. 객체지향분석은 요구되는 객체 파악, 객체의 정의, 객체 간 상호연관관계, 객체연산의 정

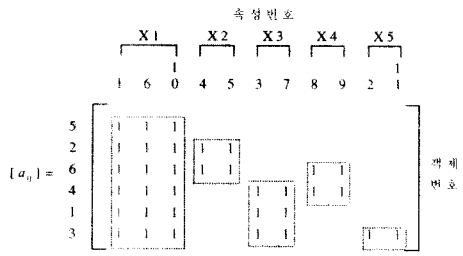
의, 그리고 구체적인 객체를 정의하는 것을 포함한다[7].

객체-속성(object-characteristic) 클러스터링 문제를 나타내기 위해 개체-속성 매트릭스[a_{ij}]를 사용하였다.

$$a_{ij} = \begin{cases} 1 : \text{객체 i가 속성 j를 사용하는 경우} \\ 0 : \text{그렇지 않는 경우} \end{cases}$$

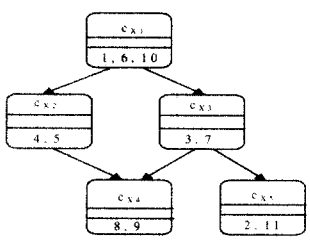


(그림 1) 객체-속성 매트릭스



(그림 2) 객체-속성 분해 매트릭스

(그림 1)의 매트릭스로부터 (그림 2)와 같이 개체-속성 매트릭스로 분해하고 개체-속성 매트릭스는 속성 클러스터를 결정하여 (그림 3)과 같은 클래스 구조를 얻고자 하는 것이다.



(그림 3) 클래스 계층구조

(그림 3)의 클래스 계층구조에서 클러스터 X1에 해당하는 클래스 c_{X1}은 속성 1, 6, 10을 가지며 c_{X2}는 속성 4, 5를 포함하여 속성 1, 6, 10을 사용 할 수 있다. 또한, c_{X3}은 속성 1, 6, 10, 3, 7을, 그리고 c_{X4}는 속성 1, 6, 10, 4, 5, 3, 7, 8, 9를, c_{X5}는 속성 1, 6, 10, 3, 7, 2, 11을 갖도록 속성들을 클러스터링 하여 클래스와 그 계층구조를 결정하는 것이다.

4. 속성 클러스터링을 위한 네트워크 기법

본 장에서는 네트워크 이용하여 속성들을 클러스터링 하는 방법을 제시하고자 한다. 클러스터링 하는 과정은 다음 3단계로 이루어진다.

1단계 : 특성의 각각의 쌍에 대해 이격차(δ_{ij})를 계산하고 매트릭스를 작성한다.

2단계 : 매트릭스를 사용해서 속성 클러스터링에 대한 문제를 0-1 정수계획법 모형의 목적함수 최소화로 문제를 푼다.

3단계 : 해답으로 제시된 클래스로부터 모든 폐쇄루프를 식별한다. 각 폐쇄루프를 속성그룹으로 클래스로 간주한다.

속성들의 클러스터링이 형성되면 5장의 알고리즘을 이용하여 클래스들 간의 상속관계 및 계층관계를 결정하여 방향성의 acyclic 그래프를 생성하게 된다.

본 연구에서 제안된 문제 해결방법은 모든 속성들 간의 비유사성(dissimilarity)의 정도를 계량화하기 위하여 매개변수를 이용하였다.

Z_i와 Z_j는 객체-속성 매트릭스의 이진 속성 벡터를 다음과 같이 정의하였다.

$$Z_i = [a_{1i}, a_{2i}, a_{3i}, \cdots, a_{Mi}]^T$$
$$Z_j = [a_{1j}, a_{2j}, a_{3j}, \cdots, a_{Mj}]^T$$

임의 두 속성간의 이격정도(Hamming distance)를 나타내는 측정기준을 이격함수로 나타내었다.

$$\delta_{ij} = \sum_{k=1}^{M} \rho(a_{ki}, a_{kj})$$

여기서, $\rho(a_{ki}, a_{kj}) = \begin{cases} 1 & \text{if, } a_{ki} \neq a_{kj} \\ 0 & otherwise \end{cases}$

$\delta(Z_i, Z_j) = 0$는 다음 특성을 만족시킨다고 할 때 :

(a) $\delta(Z_i, Z_j) = 0$ if and only if $Z_i = Z_j$

(b) $\delta(Z_i, Z_j) = \delta(Z_j, Z_i)$

(c) $\delta(Z_j, Z_i) \leq \delta(Z_j, Z_k) + \delta(Z_k, Z_j)$ for $k \neq i, j$

위 정의의 예를 위해 $Z_i = [1011100]^T$와 $Z_j = [0001101]^T$인 문제를 고려해보면, δ_{ij} =1+0+1+0+0+0+1 = 3이 된다.

클러스터링 하는 것을 네트워크 G = (V, A)로 표현하기로 한다, 여기서 V는 노드의 집합이고 A는 속성들을 연결하는 아크(arc)의 집합에 해당된다. 이 네트워크에서, 속성은 두 노드로 표현 될 수 있다, 즉 노드 i와 방향성 회로(directed circuit)에 의해서 연결된 노드 j에 의해 나타낼 수 있다. 해답모델의 네트워크는 다음과 같이 구축되어 진다.

- 2N 노드로 시작한다. 여기서 i, j = 1, 2, ⋯, N
- i에서 j로 향하는, 아크(i, j)를 만든다, 단 i ≠ j.
- 네트워크로 네트워크 G = (V,A)를 순환(circulation) 아크(j, i)로 변화시키기 위한 i = j인 복귀(return)아 크(j, i)를 형성한다.
- 각 아크(i, j)에 대해서 δ_{ij} 값을 할당한다.

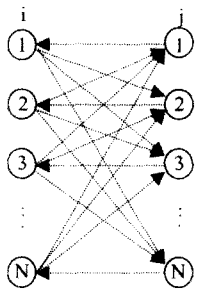
N-속성 클러스터링 문제의 네트워크가 (그림 4)에 제시되어있다. 네트워크 해법은 클러스터링 문제에 해당하는 속성

들 사이의 비유사도값(dissimilarity values)들의 합을 최소로 하고자 하는 것이다.

문제 해결 이전에는 클러스터들의 수가 몇 개가 될지 알 수 없으며 클러스터는 결국 최소한의 데이터와 함수로 구성되는 클래스이기 때문에 적어도 2개의 속성을 가지고 있다고 가정하였다.

식 (1)~식 (5)에서 이 모델을 공식화를 위해서 X_{ij}를 다음과 같이 정의한다.

$$X_{ij} = \begin{cases} 1 : \text{아크}(i, j)\text{가 선택될 경우} \\ 0 : \text{그렇지 않은 경우} \end{cases}$$



(그림 4) N-속성 클러스터링 문제의 네트워크

수학적 모형은 다음과 같다.

$$Min \sum_{i=1}^{N} \sum_{j=1}^{N} \delta_{ij} X_{ij} \quad (1)$$

S.T

$$\sum_{j=1}^{N} X_{ij} = 1 \ \forall \ j \in V, i \neq j \quad (2)$$

$$\sum_{j=1}^{N} X_{ij} = 1 \ \forall \ i \in V, i \neq j \quad (3)$$

$$X_{ij} = 1, 0 \ \forall (i, j) \in E, i \neq j \quad (4)$$

$$X_{ij} = 1 \ \forall (i, j) \in E, i = j \quad (5)$$

목적 함수 식 (1)은 속성 사이의 비유사도를 나타내는 이격함수 값의 총합을 최소화 하고자 하는 것이다. 제약조건 식 (2)와 식 (3)에서는 각 노드마다 하나의 들어오는 아크(inbound are)와 하나의 나가는 아크(outbound arc)가 반드시 존재한다는 것을 보장하고 있다. 즉, 한 노드에서 다른 노드까지의 경로(path)를 허용한다는 것이다. 제약조건 식 (4)는 결정변수 X_{ij}의 선택 여부를 확정하는 것이고, 제약조건 식 (5)는 순환 네트워크를 완성하기 위한 복귀아크(return arc)를 부여하는 것이다.

목적함수 식 (1)과 제약조건 식 (2)~식 (5)는 0-1 정수계획법(integer programming)의 문제해결모델이며 이는 심플렉스 방법(simplex method)의 절차에 의해서 해결될 수 있다. 만약 네트워크의 해가 $X_{ij} = X_{jk} = X_{ki} = 1$이면, 아크(i, j), (j, k), 그리고 (k, i)가 선택된다는 것이며 {i-j-k}는 폐쇄루프를 만든다는 것을 의미한다.

(그림 1)의 데이터로부터 δ_{ij}의 값은 (그림 5)와 같이 얻을 수 있다. (그림 5)의 δ_{ij}의 값을 목적함수 및 제약식 식 (1)~식 (5)에 적용하여 얻은 0-1 정수프로그래밍 코드로 LINDO 소프트웨어[8]를 이용하여 해를 얻을 수 있다. 문제를 풀면, 다음과 같은 해를 얻을 수 있다.

$[\delta_{ij}] =$

객체번호 \ 속성번호	1	2	3	4	5	6	7	8	9	10	11
1	0	5	3	4	4	0	3	4	4	0	5
2	5	0	2	3	3	5	2	3	3	5	0
3	3	2	0	5	5	3	0	3	3	3	2
4	4	3	5	0	0	4	5	2	2	4	3
5	4	3	5	0	0	4	5	2	2	4	3
6	0	5	3	4	4	0	3	4	4	0	5
7	3	2	0	5	5	3	0	3	3	3	2
8	4	3	3	2	2	4	3	0	0	4	3
9	4	3	3	2	2	4	3	0	0	4	3
10	0	5	3	4	4	0	3	4	4	0	5
11	5	0	2	3	3	5	2	3	3	5	0

(그림 5) δ_{ij} 매트릭스

$$X_{16} = X_{610} = X_{101} = X_{37} = X_{73} = X_{45} = X_{54} = X_{89} = X_{98} = X_{211} = X_{112} = 1,$$
$$X_{ij} = 1 \text{ for all } (i, j) \in E, i = j$$

위의 해는 루프{1-6-10}, {3-7}, {4-5}, {8-9}, {2-11}이 폐쇄루프에 해당된다는 것을 의미한다.

4.1 속성들의 클러스터링 알고리즘

속성들의 클러스터링 문제에서 문제의 규모가 큰 경우에는 목적함수 및 제약식 식 (1)~식 (5)에 사용되는 변수의 수도 급격히 늘어나게 되어 LINDO 소프트웨어 사용이 부적합한 경우도 있을 수 있다. 이런 경우에 방문판매원 문제(Traveling salesman problem)의 분배 해에서 서브타워(subtour)를 나타나는 것과 유사하게 다음 알고리즘으로 폐쇄루프를 확인할 수 있다[9]. 알고리즘은 계산된 δ_{ij} 매트릭스와 N속성 클러스터링의 네트워크로 시작한다. loop[p][i][j]의 인덱스 p는 폐쇄루프 번호, i, j는 아크(i, j)를 나타낸다.

```
p ← 0.
WHILE(모든 속성이 loop[p][i][j]에 미포함){
  FOR(δ_i*j* = ∞, i = 0; i<n; i++)
    FOR(j = 0; j < n; j++){
      if( i ≠ j and min > δ_ij ) δ_i*j* ← δ_ij.
    }
    loop[p][i*][j*]=1. //폐쇄루프p, 아크(i*, j*) 생성
  WHILE( δ_k*i* = δ_i*j*인 노드 k* 가 존재){
    loop[p][j*][k*] = 1. //폐쇄루프p, 아크(j*, k*) 생성
      i* ← j*, j* ← k*.
  } // END of WHILE
  p ← p + 1.
}//END of WHILE
```

최대 객체의 수를 m, 속성의 최대 수를 n이라고 하면 알고리즘의 시간 복잡도는 $O(n^2m+m^2)$으로 다항시간 내에 효율적으로 문제를 해결할 수 있다.

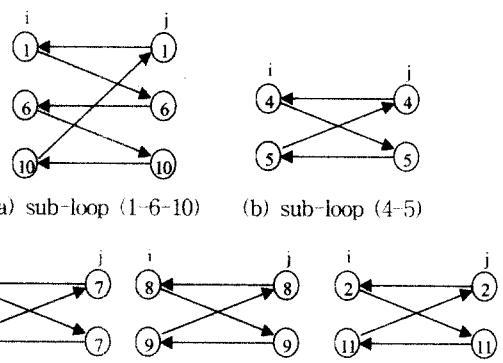
● 속성 클러스터링 알고리즘의 적용 예)

δ_{ij} 값들은 (그림 5)에서 계산되었고 (그림 4)의 네트워크이 이용된다.

```
P = 0
WHILE(모든 속성이 loop[p][i][j]에 미포함){
   δ16 = δ211 = δ37 = δ45 = δ89 = 0이므로
   δ i,j ← δ16
     loop[0][1][6]=1. // 폐쇄루프 0, 아크(1, 6) 생성
   WHILE( δ610 = δ16인 k* 노드 존재){ // k* = 10 존재
     loop[0][6][10]=1. // 폐쇄루프 0, 아크(6, 10) 생성
        i* ← 6, j* ← 10.
   } // END of WHILE
    p ← 0 + 1.
}//END of WHILE. Iteration 0.

WHILE(모든 속성이 loop[p][i][j]에 미포함){
   min = δ211 = δ37 = δ45 = δ89 = 0 에서
   δ i,j ← δ211
   loop[1][2][11]=1. //폐쇄루프 0의 아크(2, 11) 생성
   WHILE( δ11k* = δ211인 k* 노드 존재) 불만족
   } // END of WHILE
    p ← 1 + 1.
}//END of WHILE. Iteration 1.
```

같은 방법으로 WHILE(모든 속성이 loop[p][i][j]에 미포함)의 반복에서 loop[3][3][7]=1, loop[4][4][5]=1, 그리고 loop[5][8][9]=1을 각각 얻게 된다. 즉, (그림 6)과 같이 폐쇄루프 (1-6-10), (4-5), (3-7), (8-9), 그리고 c_4=(2-11) 를 각각 얻게 되는 것이다. 그러므로, 위의 해에서 5개의 하부 루프로부터 5개의 클러스터를 형성하게 된다. 이는 궁극적으로 속성 클러스터 별로 5개의 클래스를 사용하게 되는 것이다.



(a) sub-loop (1-6-10) (b) sub-loop (4-5)

(c) sub-loop (3-7) (d) sub-loop (8-9) (e) sub-loop (2-11)

(그림 6) 네트워크 모형에서 하부루프(Sub-loop)

5. 상속관계를 위한 그래프 생성

5.1 클래스간의 계층관계

두 클래스간의 상속을 위한 상하관계를 나타내기 위해 다음 기호를 정의하였다.

O : 모든 객체의 집합
O_i : 클러스터 i에 포함된 속성을 갖는 객체집합
$n(O_i)$: 객체집합 O_i에 속한 객체의 수
c_i : 클러스터 i의 속성을 멤버로 갖는 클래스
c_i^m : 클래스 c_i에 해당하는 속성들의 집합
$\bar{c}_i^m$: 클래스 c_i에서 가용한 속성들의 집합(상속받은 속성 포함)
D_p : 객체 p에 해당하는 속성들의 집합
L : 클러스터의 집합

본 연구에서는 $\{O_i \cap O_j\} \neq \emptyset$이고 $O_i \neq O_j$이면 $\{c_i^m \cap c_i^m\} = \emptyset$로 가정한다.

[Property 1] 클러스터 i, j에 대하여, $\bar{c}_i^m \subset \bar{c}_j^m$이면 클래스 c_i는 클래스 c_j의 상위 클래스이다.

[증명] 클래스 c_j가 클래스 c_i의 상위 클래스라고 가정하면, $\bar{c}_j^m = c_j^m$와 $\bar{c}_i^m = c_i^m + c_j^m$의 관계가 성립하나 $\bar{c}_i^m \supset \bar{c}_j^m$로 되어 가정 $\bar{c}_i^m \subset \bar{c}_j^m$에 위배된다. 따라서 클래스 c_i는 클래스 c_j의 상위 클래스이다.

[규칙 1] 속성을 갖는 임의의 두 클러스터 i, j에 대해 $O_i \supset O_j$이면 클래스 c_i는 클래스 c_j의 상위클래스(super class)이다.

[증명] 클래스 c_i가 클래스 c_j의 상위클래스가 아니라고 가정하면 c_i가 클래스 c_j의 하위클래스 이거나 상속관계에 있지 않은 경우이다.

[경우 I] 클래스 c_i가 클래스 c_j의 하위클래스인 경우는 c_j에 해당하는 속성으로만 이루어진 객체가 존재하고 클래스 c_i에 해당하는 속성과 클래스 c_j로부터 상속받은 속성으로 이루어진 객체가 존재하게 된다. c_j가 상위 클래스 이므로 c_j에 해당되는 속성들을 갖는 객체는 모든 객체에 포함되어 $O_j \supseteq O$이다. c_i는 하위 클래스이므로 해당되는 속성들은 모든 객체에 다 포함되지는 않으므로, $O_i \subset O$이다. 그러므로 O_i와 O_j의 포함관계는 $O_i \subset O_j$에 있다. 따라서 규칙의 가정 $O_i \supset O_j$에 위배된다.

[경우 II] 본 연구의 가정에서 $\{O_i \cap O_j\} \neq \emptyset$이고 $O_i \neq O_j$이면 $\{c_i^m \cap c_i^m\} = \emptyset$이므로, 규칙의 가정, $O_i \supset O_j$인 두 클러스터 i, j에 해당하는 클래스는 상속관계에 있다. 따라서 클래스 c_i는 클래스 c_j의 상위클래스에 해당된다.

[규칙 2] 임의의 클러스터 i, j, k에 대하여 $O_i \supset O_j$이고 $O_j \supset O_k$이면 클래스 c_i는 클래스 c_j와 클래스 c_k의 기저 클래스(base class)에 해당 된다.

[증명] $O_i \supset O_j$이므로 [규칙 1]에 의해 클래스 c_i클래스 c_j의 상위 클래스이고 마찬가지로 클래스 c_j는 클래스 c_k의 상위 클래스 이므로 클래스 c_i는 최 상위 클래스인 기저클래스에 해당된다.

[규칙 3] $\bigcup_{p \in O_i} D_p \supset \bigcup_{q \in O_i} D_q$이면 클래스 c_i는 클래스 c_i의 상위 클래스이다.

[증명] $\bigcup_{p \in O_i} D_p = \bar{c}_i^m$이고 $\bigcup_{q = O_j} D_q = \bar{c}_j^m$이므로 $\bigcup_{p \in O_i} D_p \supset \bigcup_{q \in O_i} D_q$에서 $\bar{c}_i^m \supset \bar{c}_j^m$.

따라서 [Property 1]에 의해 클래스 c_j는 클래스 c_i의 상위 클래스이다.

[규칙 4] $O_i=\{o_x, o_y\}$인 클러스터 i가 $o_x \subset O_j$와 $o_y \subset O_k$를 만족하면, 클래스 c_i는 클래스 c_j와 클래스 c_k로부터 다중 상속(multiple inheritance)을 받는 하위 클래스이다.

[증명] 객체 o_x로만 구성된 집합을 가상의 O_x, 객체 o_y로만 구성된 집합을 가상의 O_y로 각각 가정하면 $O_x \subset O_j$와 $O_y \subset O_k$ 이므로, [규칙 1]에 의해 클래스 c_j는 클래스 c_x의 상위 클래스이고 클래스 c_k는 클래스 c_y의 상위 클래스에 해당된다. o_x와 o_y는 O_i의 멤버이므로 $\overline{c}_i^m = \overline{c}_x^m = \overline{c}_y^m$ 이다. 따라서 $O_i=\{O_x \cup O_y\}=\{o_x, o_y\}$에 해당하는 클래스 c_i는 클래스 c_j와 클래스 c_k로부터 다중상속을 받는 하위 클래스이다.

[규칙 5] $O_i \supset O_j, j \in L$이면, 클래스 c_i는 모든 클래스 c_j들의 상위클래스이다.

[증명] 각 클러스터 j에 대하여 $O_i \supset O_j$ 성립하므로 [규칙 1]에 의해 클래스 c_i는 모든 클래스 c_j의 상위 클래스에 해당된다. 따라서 클래스 c_i는 $O_i \supset O_j, j \in L$인 클래스 c_j의 상위클래스이다.

5.2 계층구조 생성 알고리즘

4.1절에서 제시한 알고리즘에 의해 얻은 클러스터의 집합으로부터 클러스터 상호간의 관계에 따라 클래스 계층구조를 5.1절의 [규칙 1]~[규칙 5]를 근거로하여 그래프를 생성하는 절차를 제시하였다. 생성되는 그래프는 방향성 비순환 그래프(a directed acyclic Graph)는 G(V, A)로 표현하며, 여기서 V는 그래프 상의 객체의 집합인 O_i들의 집합이고 A는 그래프 상의 모든 방향성 아크(Arc)의 집합에 해당된다. 이 화살표는 그 방향에 따라 객체집합 O_i에 해당하는 클래스들 간의 상속관계를 나타내게 된다.

n개의 클러스터의 집합, L을 가지고 시작한다.

```
WHILE( L≠∅){
  FOR(max = 0, i = 0 ;  i < n ; i++){
    IF max <= n(Oi) max ← n(Oi)
  } // END of FOR
  Oi* ← max에 해당하는 노드,
  그래프 G의 시작노드 ← Oi*,
  L ← L - i*, V ← V + Oi*,
  WHILE((Oi∩Oj) ≠ ∅, Oi∈ V, j∈ L){
    FOR(max = 0, i = 0 ; i < m ; i++)
      FOR(j = 0 ; j < r ; j++){
        IF max <= n(Oi∩Oj) max ← n(Oi∩Oj)
      } // END of FOR
    max에 해당하는 아크(Oi, Oj)를 생성.
    L ← L - j, V ← V + Oj, r ← r - 1, m ← m + 1
  } // END of WHILE
} // END of WHILE(L≠∅)
```

알고리즘의 WHILE(L≠∅) 루프에서는 [규칙 5]에 대하여 각 클래스구조의 기저클래스는 최대 객체집단을 갖게 되므로 max{n(O_i), i∈ L}에 해당되는 객체집합에 해당하는 클러스터를 그래프의 시작노드로 하였다. WHILE(($O_i \cap O_j$)의 루프에서는 그래프의 노드를 추가시켜가는 순서는 깊이우선원칙 혹은 너비우선원칙 어느 것을 이용하여도 가능하다.

- **계층구조 생성의 예)**

(그림 1)의 객체-속성 매트릭스의 문제에서 4.1절에서 제시한 알고리즘에 의해 얻은 속성 클러스터 1, 2, 3, 4, 5 와 그에 해당하는 객체의 집합, 즉, $O_1=\{5, 2, 6, 4, 1, 3\}$, $O_2=\{2, 6\}$, $O_3=\{3, 1, 4\}$, $O_4=\{6, 4\}$, $O_5=\{3\}$을 가지고 시작한다.

```
WHILE(L≠∅){ // L={1, 2, 3, 4, 5}) 이므로
  n(O1)=5 이므로 그래프 G의 뿌리노드 ← O1.
  V = {O1}, L = {2, 3, 4, 5}.
  WHILE((Oi∩Oj)≠∅){
    V = {O1}이므로 O1∩O3 = {3, 1, 4}로서
    max = n(O1∩O3) = 3이므로 아크(O1, O3) 생성.
    L = {1, 2, 4, 5}
  } // iteration #0
  WHILE((Oi∩Oj)≠∅){
    V = {O1, O3}이므로 O1∩O2 = {2, 6} 로서
    max = n(O1∩O2) = 2이므로 아크(O1, O2) 생성.
    V = {O1, O3, O2}, L = {4, 5}.
  } // iteration #1
  ...
```

같은 방법으로 아크(O_2, O_4), 아크(O_3, O_4), 그리고 아크(O_3, O_5) 각각 생성하고 최종적으로 V = {O_1, O_3, O_2, O_4, O_5}, L = {} 을 갖게 된다.

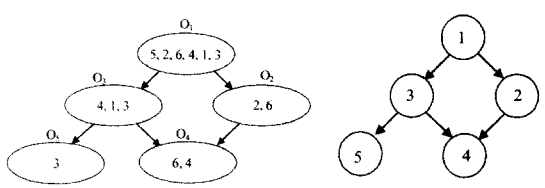
최종적으로 얻어진 객체집합의 관계 그래프의 결과는 (그림 7)(a)와 같다. (그림 7)(a)의 그래프는 다시 해당 클래스로 대체하여 클래스의 계층구조를 나타내는 그래프를 (그림 7)(b)를 얻을 수 있는 것이다.

- **그래프 생성의 예 2)**

B소프트웨어 개발회사로부터 획득한 15×36 객체-속성 문제를 (그림 8)와 같이 편의 상 객체의 이름과 속성을 숫자로 나타내었다.

4.1절의 네트워크를 이용한 속성 클러스터링 알고리즘에 의해 17개의 폐쇄루프를 얻었으며 각 루프의 속성 클러스터는 다음과 같다.

$c_1=\{3-22-36\}$, $c_2=\{4-25\}$, $c_3=\{6-14\}$, $c_4=\{9-18\}$, $c_5=\{11-20\}$, $c_6=\{12-33\}$, $c_7=\{7-15\}$, $c_8=\{8-19\}$, $c_9=\{17-26\}$, $c_{10}=\{10-21\}$, $c_{11}=\{2-23\}$, $c_{12}=\{5-24\}$, $c_{13}=\{16-27\}$, $c_{14}=\{28-31\}$, $c_{15}\{29-30\}$, $c_{16}=\{13-34\}$, $c_{17}=\{1-35\}$.

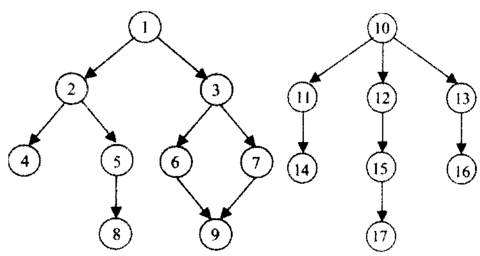


(a) Oi의 관계 그래프 (b) 클래스 계층그래프

(그림 7) 객체집합의 관계 그래프

```
                  속성번호
                  111111111122222222223333333
         123456489012345678901234567890123456
     1   000010000100000000001001000000000000
     2   100010000100000000001001000011100010
     3   001100000010000000001010010000000001
     4   010000000100000000000101000000000000
     5   000000000100000100001000001000000000
     6   001001000000001000000001000000000001
[a_ij]= 7   010000000100000000001010000100100000
     8   001001100000011010000100010000000001
     9   001100010010000000011010010000000001   개체
    10   001001100000011000000010000000000001   번호
    11   001100000000000000000100100000000001
    12   000010000100000000001001000011010000
    13   001001000001010010000100010000001001
    14   000000000100100100001000001000000110
    15   001100001000000000100010010000000001
```

(그림 8) 개체-속성 (15×36) 매트릭스



(a) (b)

(그림 9) 클래스 계층구조 그래프

5.2절의 클래스계층 그래프 생성 알고리즘을 이용하여 (그림 9)에서와 같이 클래스들 사이의 상속관계를 나타내는 두 개의 클래스구조 그래프를 얻었다.

6. 분석과 평가

<표 1>에서 기존의 소프트웨어 분류방법과 본 연구에서 제시한 방법의 적용환경 및 장단점을 나타내고 있다. 기존의 방법의 적용은 코딩단계 혹은 그 이후에 적합한 반면 본 연구에서 제시한 방법은 모델링과 설계단계에서 적용하는데 적합하다.

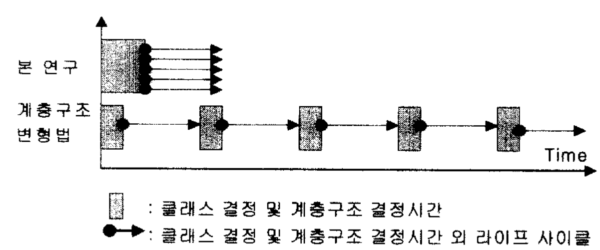
또한 개발전략 면에서 기존의 계층구조 변형방식은 각 클래스의 생성 시에 기존 계층구조를 점진적으로 변형하여 개선시키므로 변형과정을 반복적으로 수행 하여야 하지만 본 연구의 기법은 다수의 객체들과 클래스들을 일괄적으로 동시에 처리함으로써 반복되는 사이클을 효과적으로 제거 할 수 있다.

<표 2>에서는 코딩단계에서 주어진 6가지 사례들을 본 연구의 알고리즘 1, 2를 각각 적용하여 얻은 결과와 구조변형 방법을 이용한 결과를 비교하고 있다. 알고리즘 1과 2 그리고 [2]의 알고리즘을 C++ 언어로 작성하여 펜티엄급 1.99 GHZ 512MB 램의 개인용 컴퓨터에서 마이크로소프트 윈도우 운영체제에서 실행하였다. 두 방법에서 얻은 클래스와 계층구조의 결과는 상이하지 않음을 나타내고 있다.

구조변형방법은 개발 사이클이 반복 되는 상황에서 필요한 새로운 클래스가 단속적으로 생성되면 이를 기존 라이브러리의 클래스와 지속적으로 비교하여 가장 유사한 계층구조에 합병, 삽입 등 변형을 하여 라이브러리에 등록하게 된다. 따라서 (그림 10)과 같이 실제 문제에서는 반복되는 각 클래스의 검색, 수정, 보완, 등록 등의 시간 외에 각 클래스의 추가가 있을 때 마다 비연속으로 반복되는 선행시간과 각 라이프 사이클 간격이 존재한다는 것에 유의 하여야 한다.

〈표 1〉 기존방법과 본 연구의 비교

방 법	본 연구	계층구조변형 방식	Enumera-tive	Faceted	Clustering
적용단계	모델링 및 설계	코 딩	코 딩	코 딩	코 딩
개발 전략	다수의 객체와 다수 클래스의 일괄 동시처리	개발 사이클이 다수인 반복 수정개선	반복수행	반복수행	반복수행
독립된 클래스 동시처리	가 능	가 능	불가능	불가능	불가능
상속관계	가 능	가 능	깊은 계층 표현 어려움	부적합	부적합
클래스 확장성	양 호	양 호	불양	양 호	불 양
클래스의 분류	적 합	적 합	사용자 직접할당	부적합	부적합
기 타	개발 사이클을 소수로 단축	라이브러리 등록 방식	도서분류법	문서 분류법	



(그림 10) 라이프 사이클의 비교

〈표 2〉 계층구조 변형방법과 본 연구의 사례적용

사 례 No.	사 례 클래스 수	본 연구 알고리즘 1, 2 적용수	본 연구 CPU Time 합	계층구조 변형방법 알고리즘 적용 수	계층구조 변형방법 CPU Time 합	클래스와 계층구조 결과
1	5	각 1회	0.25	4회	1.80	동일
2	16	각 1회	1.00	15회	5.50	동일
3	10	각 1회	0.60	9회	3.30	동일
4	12	각 1회	0.71	11회	3.70	동일
5	7	각 1회	0.43	6회	2.20	동일
6	8	각 1회	0.54	7회	2.90	동일

구조변형방법이 지속적, 반복적인 개발환경에서 클래스의 분류를 수정보완 방식으로 하는 반면에 본 연구의 방식은 개발전략상 많은 객체와 클래스의 개발이 동시에 이루어지는 개발환경에 적합하다고 할 수 있다. 따라서 구조변형 방식이 소수의 개발인력으로 긴 일정으로 개발이 가능하다면

본 연구는 다수의 인력이 단 기간에 집중 투입되어야 할 것이다.

적용환경이 다른 기존의 연구와 우열의 논의는 무의미 하지만 본 연구의 이점은 무엇보다도 수정보완의 사이클을 단축함으로서 개발기간을 감소시킬 수 있는 이점을 가질 수 있다. 따라서 적용환경에 따라 기존연구의 기법과 본연구의 기법을 적절히 보완하여 사용하는 것이 바람직 할 것이다.

7. 결 론

객체지향 소프트웨어 개발을 위해 점증형 개발모형은 새로운 클래스 혹은 모듈을 지속적으로 부가적으로 개발해야 했다. 클래스 라이브러리에 현존하는 클래스의 수정과 재 정의를 위해 많은 노력과 많은 반복을 요구한다. 이러한 반복 사이클을 짧게 하는 한 가지 방법은 모델링 단계나 설계단계에서 가능한 한 많은 객체들과 속성들의 관계를 분석하여 적절한 속성들의 클러스터링으로 클래스를 정의하는 것이다. 이는 또한 클래스의 라이브러리 등록 단계에서 클래스 수정과 계층구조의 변경의 노력을 최소화 시킬 수 있는 것이다.

본 연구의 성공적인 적용은 모델링 및 설계단계에서 필요한 객체들과 속성들을 얼마나 많이 확인하고 추출하느냐에 달려있다. 개발 초기단계에서 가능한 많은 클래스를 적절하게 결정함으로서 불필요한 수정보완의 사이클을 줄이는 것이다.

참 고 문 헌

[1] Roger S. Pressman, Software Engineering a ractioner's approach, 3rd Ed., pp.18 MacGraw-Hill, Inc., 1992.

[2] 김갑수, 신영길, "소프트웨어 재사용을 위한 C++ 클래스 계층구조 변형방법", 한국정보과학회지, 제22권 제1호, pp.88-96, 1995.

[3] R. Prieto-Diaz, "Implementing Faceted Classification for software reuse," Communications of the ACM, Vol.34, No.5, pp.89-97, May, 1991.

[4] S. R., Ranghanathan, "Prolegomena to Library Classification," Asia Publish House, Bombay, India, 1967.

[5] R. O. Duta and P. E. Hart, "Pattern Classification and Scene Analysis," Chicheste : John Wiley & Sons, 1973.

[6] G. K. Brian and R. L, James, "Using Program Slicing in software maintainence," IEEE, Trans. on Software Engineering, Vol.17. No.8, pp.751-761, Aug., 1991.

[7] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, "Object-Oriented software engineering : A use Case Driven Approach," Addison-Wesley, 1996.

[8] L. Schrage, "Linear, Integer and Quadratic Programming with LINDO," Scientific Press, Palo, Alto, CA, 1984.

[9] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem : a survey," Operation Research, Vol.16, pp.538-558, 1968.

[10] B. F. William and B. -Y. Ricard, "Information Retrieval," Prentice Hall, pp.419-442, 1992.

[11] E. L. Lawler, J. K. Lenstra and A. H. G. Rinnoony and D. B. Shimoys, (Editors), "The traveling Salesman Problem : A guided Tour of Combinatorial Optimization," John Wiley, New York, 1985.

[12] B. A. Burton, "The Reusable software Library," IEEE Software, Vol.4, No.4, pp.25-33, July, 1987.

[13] E. Damiani, M. G. Fugini, C. Bellettini, "A hierarchy Aware Approach to Faceted Classification of Object-Oriented Components," ACM Transactions on Software Engineering and Methodology, Vol.8, No.4, pp.425-472, 1999.

[14] M. Aoyama, "Concurrent Development Process Model," IEEE Software, pp.46-55, July, 1993.

[15] M. Aoyama, "Web-based agile software development," IEEE Software, Vol.15, pp.56-65, November/December, 1998.

[16] A. Cockburn, "Surviving Object-Oriented Projects," Addison Wesley, Reading, MA, 1998.

[17] J. Highsmith, "Adaptive Software Development," Dorset House, New York, NY, 2000.

[18] R. S. Pressman, "Software engineering : A practioner's approach," Macgrow-Hill, 4th ED., 1997.

이 건 호

e-mail : ghlee@ssu.ac.kr

1986년 대구대학교 산업공학과(학사)

1991년 인하대학교 산업공학과(석사)

1996년 U. of Iowa, Dept. of Industrial Eng.(ph.D)

1997년~1999년 숭실대학교 산업공학과 전임강사

1999년~2004년 숭실대학교 산업·정보시스템공학과 조교수

2004년~현재 숭실대학교 산업·정보시스템공학과 부교수

관심분야 : OOD, E-business, AI, Agile Manufacturing, Product Design.