클래스계층구조의 품질평가척도를 기반으로 하는 재구성기법

황 석 형† · 양 해 술†† · 황 영 섭†

요 약

클래스계층구조는 객체지향 소프트웨어의 중심적인 구성요소가 되며, 따라서 이에대한 품질은 매우 중요하다. 좋은 품질을 갖춘 클래스계층구조를 구축하는 것은 객체지향 소프트웨어 개발에 있어서 매우 중요한 작업이지만, 좋은 품질의 클래스계층구조를 구축하는 작업은 수월하지 않다. 더욱이, 반복 및 점증적인 소프트웨어 개발에 있어서, 요구사양에 적합하도록 개발중인 클래스계층구조를 재구성하거나 세련시키는 작업이 빈번히 발생한다. 따라서, 객체지향 개발자들이 이와같은 클래스계층구조의 재구성 작업을 수행할 경우에 도움이 될 수 있는 기법 및 도구들에 대한 관심이 증가하고 있다. 본 논문에서는, 클래스계층구조의 복잡도를 일정수준 측정가능한 몇가지 평가척도를 제안하고, 이러한 평가척도들을 바탕으로 클래스계층구조를 재구성하는 알고리즘들을 정의한다. 또한, 각 알고리즘들에 대하여, 알고리즘을 적용하기 전과 후의 각 클래스계층구조들로부터 생성할 수 있는 객체들의 집합이 변화하지 않고 보존됨을 증명하였다. 본 논문에서 제안하는 재구성기법은 클래스계층구조의 구축 및 재구성 등의 지침으로서 사용할 수 있으며, 이러한 평가척도들을 기반으로 하는 클래스계층구조의 재구성 알고리즘들은 객체지향 소프트웨어 개발시에 유용한 도구로서 개발자들에게 도움이 될 수 있다.

A Metrics-Based Approach to the Reorganization of Class Hierarchy Structures

Sun Hyung Hwang† · Hea Sool Yang†† · Young Sub Hwang†

ABSTRACT

Class hierarchies often constitute the backbone of object-oriented software. Their quality is therefore quite crucial. Building class hierarchies with good quality is a very important and common tasks on the object oriented software development, but such hierarchies are not so easy to build. Moreover, the class hierarchy structure under construction is frequently restructured and refined until it becomes suitable for the requirement on the iterative and incremental development lifecycle. Therefore, there has been renewal of interest in all methodologies and tools to assist the object oriented developers in this task. In this paper, we define a set of quantitative metrics which provide a way of capturing features of a rough estimation of complexity of class hierarchy structure. In addition to, we suggest a set of algorithms that transform a original class hierarchy structure into reorganized one based on the proposed metrics for class hierarchy structure. Furthermore, we also prove that each algorithm is "object-preserving". That is, we prove that the set of objects are never changed before and after applying the algorithm on a class hierarchy. The technique presented in this paper can be used as a guidelines of the construction, restructuring and refinement of class hierarchies. Moreover, the proposed set of algorithms based on metrics can be helpful for developers as an useful instrument for the object-oriented software development.

키워드 : 객체지향(Object-Oriented), 클래스계층구조(Class Hierarchy Structure), 객체등가관계(Object-Equivalence Relation), 평가척도(Metics), 재구성알고리즘(Reorganization)

1. 서 론

객체지향 분석 및 설계에 있어서, 요구사양변경 등에 의해 기존에 설계된 클래스계층구조를 재구성하거나 세련시키는 경우가 빈번히 발생한다. 이와 관련된 기존의 클래스계층구조의 재구성에 관한 연구가 활발히 진행되어 왔다.

문헌[1]에서는 클래스계층 모델을 토대로 하여, 클래스계층간의 객체등가관계를 유지하는 재구성기법을 정의하고 있다. 주어진 2개의 클래스계층구조에서 각각 생성될 수 있는 객체의 집합이 같은 경우, 양쪽 클래스계층구조는 객체등가관계에 있다고 한다. 기존의 클래스계층구조에 대하여 [1]에서 정의된 재구성조작을 적용하므로써, 생성가능한 객체의 집합이 동일한 별개의 새로운 클래스계층구조를 구축할 수 있다. 이와같은 재구성기법은, 기존의 어플리케이션을 변경시키지 않고 클래스 라이브러리를 재구성하여 재이용할 수 있다는 잇점이 있다. 그러나, [1]의 연구에서는, 상속계층과 단순한 집약계층구조에 대한 재구성을 논의의 대상으로 하고 있으며, 메소드에 관한 사항은 언급되고 있지 않다.

또한, 문헌[2, 3]에서는, 클래스계층구조의 클래스간의 상속계층 및 집약계층의 데이터멤버뿐만 아니라, 멤버함수, 즉, 메소드에 대한 개념을 도입하고 있다. 즉, 객체등가관계를

† 종신회원 : 선문대학교 컴퓨터정보학부 교수
†† 종신회원 : 호서대학교 벤처전문대학원 교수
논문접수 : 2003년 5월 20일, 심사완료 : 2003년 7월 14일

메소드를 포함한 정의로 확장하였고, 객체등가관계를 유지하는 기본등가재구성조작을 메소드를 고려하여 확장, 재정의하였다. 기존의 클래스계층구조에 메소드를 포함시킨 기존의 등가재구성조작을 적용시키므로써, 생성가능한 객체의 집합이 같은 별개의 새로운 클래스계층구조를 구축할 수 있다.

기존의 연구[1-3]에서 논의된 재구성기법에서는, 정의된 재구성조작들이 객체등가관계를 만족시키고 있음은 알 수 있지만, 클래스계층그래프를 평가하기 위한 척도에 대해서는 언급되어 있지 않으므로, 어떠한 변환을 적용하면 유용한 결과가 얻어질 수 있는지 논의되지 않았다. 또한, [4]의 연구에서는 디자인패턴을 토대로하는 클래스계층에 대한 Refactoring 기법을 제안하고 있으나, 주로 구현된 프로그램 코드에 대한 변환기법을 중심적으로 논의하고 있다.

한편, 객체지향설계에 있어서 소프트웨어의 생산성 및 품질을 개선하기 위하여, 정량적으로 제반품질특성들을 측정하기 위한 평가척도가 다수 제안되었다[5-12]. 특히, [5]의 연구에서는, 객체지향 프로그래밍에 있어서 클래스간의 불필요한 상호 의존관계를 줄이기 위한 프로그래밍 스타일에 관한 규칙으로써 Demeter의 법칙(The Law of Demeter)이 제안되었으며, [6]의 연구에서는 종래의 비형식적인 Demeter의 법칙을 실제의 프로그램에 적용, 평가하기 위해 클래스간의 관계로써 상속과 집약, 그리고 관련관계를 정형화하여 정의하고, Demeter의 법칙을 위반한 프로그램에 대하여 정당한 형태로 변환하기 위한 알고리즘을 제안하였다. 또한, [7]에서는 계측이론에 토대를 두고 6가지 메트릭스가 제안되었으며, 일정한 평가를 얻고 있다[10]. 또한, [11]와 [12]에서는, 각각 객체지향소프트웨어를 구현언어에 종속되지 않는 그래프형태의 메타모델 및 UML 기반의 메타모델로 표현하고, 각 모델을 토대로 각각 다양한 평가척도들을 기술하고 있다. 그러나, 이상의 관련연구들에서 제안된 메트릭스는 각각의 클래스에 대한 지표이며, 클래스계층 전체를 평가하기 위한 지표는 아니다.

본 논문에서는, 클래스계층구조를 평가하기 위한 척도를 설정하므로써, 클래스계층구조의 재구성방침으로서 제시하고, 객체등가관계를 유지하며 클래스계층구조을 재구성하기 위한 알고리즘을 제안한다. 본 연구의 기초가 되는 클래스계층구조의 모델은, 메소드를 고려한 클래스계층그래프를 사용하였다. 이와같은 모델에 의해, 클래스계층구조는 클래스간의 상속 및 집약관계, 그리고 메소드의 집약관계를 표현하고 있으므로, 클래스계층구조에 대한 유용성 평가기준을, 이러한 상속, 집약관계에 의존하는 것으로 설정한다. 그리고, 제안된 평가척도에 따라서 클래스계층구조를 객체등가관계를 유지하면서 재구성하는 알고리즘을 제안한다. 또한, 어떤 평가척도에 따라서 클래스계층구조를 재구성하면, 다른 척도에 영향을 끼칠 수도 있으므로, 부수적으로 발생가능한 파급효과를 최소화하도록 각 알고리즘을 설계하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 클래스계층구조를 정형적으로 모델화한 클래스계층그래프를 정의하고, 관련용어 및 몇가지 제반 정의들에 대하여 설명한다. 제 3장에서는, 클래스계층그래프간의 객체등가관계의 정의와, 객체등가관계를 유지하면서 클래스계층그래프를 재구성하는 기본적인 조작에 대하여 설명한다. 제 4장에서는 클래스계층그래프의 특징을 토대로 5개의 품질평가척도들을 정의하고, 제 5장에서는 제안된 평가척도들을 토대로 클래스계층그래프를 객체등가관계를 유지하면서 변환하는 알고리즘들을 제안한다. 마지막으로 제 6장에서는 본 논문의 연구 결과 및 향후의 연구과제를 설명한다.

2. 클래스계층그래프

클래스계층구조를 정형적으로 모델화하기 위하여 [2, 3]에서는 프로그래밍언어에 의존하지 않고 클래스계층구조를 형식적으로 표현하기 위한 모델로서, 클래스계층그래프(Class Hierarchy Graph : CHG)가 정의되었다. 여기서는, 우선, 클래스계층그래프에 관한 용어 및 클래스계층그래프의 정의, 그리고 이와 관련된 몇 가지 제반 정의들에 대하여 설명한다.

2.1 클래스계층그래프에 관한 용어

여기서는, 본 논문에서 사용하는 클래스계층그래프에 관한 용어에 대하여 설명한다.

2.1.1 메소드

메소드명과 인덱스로 구성된다. 인덱스는 임의의 자연수이며, 메소드는 메소드의 이름 뒤에 [와]로 둘러쌓인 인덱스를 붙여서 나타낸다. 메소드명은 메소드의 이름과 매개변수에 대응하고, 메소드의 인덱스는 메소드의 구현부분(body)에 대응한다(<표 1>참조).

〈표 1〉 메소드명, 인덱스와 실제 메소드의 대응관계

메소드명	A	B	A
인 덱 스	1	0	0
메 소 드	A[1]	B[0]	A[0]

2.1.2 동일메소드

2개의 메소드 $M_1[i]$, $M_2[j]$에 있어서, $M_1 = M_2$이고 $i = j$가 성립할 경우, 즉, 메소드명 및 인덱스가 같은 경우, $M_1[i]$와 $M_2[j]$는 同一메소드라고 부른다.

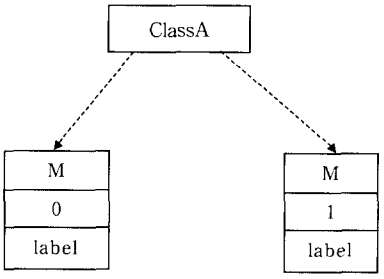
2.1.3 동명메소드

2개의 메소드 $M_1[i]$, $M_2[j]$에 있어서, $M_1 = M_2$가 성립하는 경우, $M_1[i]$와 $M_2[j]$는 同名메소드라고 부른다. 동명메소드는 메소드의 이름 및 매개변수 등, 인터페이스는 같지만 메소드의 구현부분이 상이한 경우에 해당한다.

2.1.4 메소드의 중첩

어떤 클래스가 2개 이상의 상이한 인덱스를 갖는 동명메

소드를 집약하고 있는 경우(즉, 메소드를 overloading하는 경우), 중첩된 메소드를 집약하고 있다고 부르며, 이와같이 중첩된 메소드를 집약하고 있는 클래스를 "중첩클래스"라고 부른다. (그림 1)의 구상클래스정점 ClassA는 중첩정의된 메소드를 집약하고 있는 중첩클래스의 예를 보여주고 있다.



(그림 1) 중첩클래스의 예

2.2 클래스계층그래프의 정의

클래스계층그래프는 아래와 같은 3종류의 정점과 3종류의 변으로 구성된 유향그래프이다.

2.2.1 추상정점

추상클래스를 나타내는 정점으로서, 육각형으로 그리며, 추상정점의 집합을 VA로 나타낸다.

2.2.2 구상정점

구상클래스를 나타내는 정점으로서, 사각형으로 그리며, 구상정점의 집합을 VC로 나타낸다.

2.2.3 메소드정점

메소드를 나타내는 정점으로서, 메소드정점은 메소드명과 인덱스, 그리고 메소드가 참조하고 있는 레이블명에 관한 정보를 사각형내에 3개 영역으로 구분하여 표현한다. 일반적으로 메소드는 메소드의 이름, 매개변수, 구현부분으로 구성되므로, 메소드정점이 갖는 정보와의 대응관계는 <표 2>와 같다.

〈표 2〉 메소드 정점과 실제 메소드와의 대응관계

CHG의 메소드정점	메소드
메소드명	메소드의 이름 및 매개변수
메소드의 인덱스	메소드의 구현부분

또한, 1개의 클래스가 복수개의 메소드를 집약하고 있는 경우, 각 메소드를 나타내는 사각형을 연접시켜서 1개의 사각형 표형태로 표현한다. 메소드 정점의 집합을 VM으로 나타낸다.

2.2.4 상속변

상속관계를 나타내는 변으로서, 이중화살표(⇒)로 그리며, 상속변의 집합을 EI로 나타낸다.

2.2.5 부품집약변

클래스간의 "전체-부품" 관계를 나타내는 관련관계를 집약관계라고 부른다. 이러한 집약관계을 나타내는 변으로서, 부품을 식별하기 위한 레이블(l)을 붙인 화살표($\rightarrow^{l}$)로 나타내며, 집약변의 집합을 EA로 나타낸다.

2.2.6 메소드 집약변

임의의 클래스 C에 메소드 m이 정의되어 있을 경우, 본 논문에서는 클래스 C가 메소드 m을 집약하고 있다고 표현한다. 이와같은 클래스와 메소드간의 집약관계를 나타내는 변으로서, 점선화살표(⇢)로 나타내며, 메소드를 집약하는 집약변의 집합을 EM으로 나타낸다.

2.2.7 레이블

부품을 집약하는 집약변에 부가되는 식별자로서, 레이블의 집합을 Λ로 나타낸다.

클래스계층그래프는 다음과 같이 정의한다.

[정의 1] 클래스계층그래프(Class Hierarchy Graph)

다음 조건을 만족하는 유향그래프 Γ = (VA ∪ VC ∪ VM, EI ∪ EA ∪ EM, Λ)를 클래스계층그래프(CHG)라고 부른다.

① VA ∪ VC ∪ VM

VA는 추상정점의 집합, VC는 구상정점의 집합, VM은 메소드정점의 집합을 나타낸다.

② EI ∪ EA ∪ EM

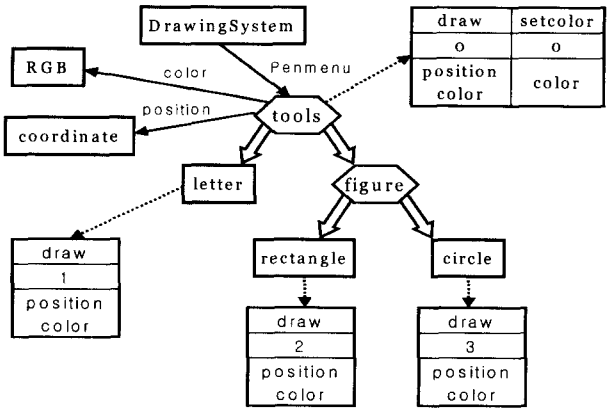
EI는 상속변의 집합, EA는 부품을 집약하는 집약변의 집합, EM은 메소드를 집약하는 집약변의 집합을 나타내며, 각각 다음과 같다.

$$EI \subseteq VA \times (VA \cup VC)$$
$$EA \subseteq (VA \cup VC) \times (VA \cup VC) \times \Lambda$$
$$EM \subseteq (VA \cup VC) \times VM$$

③ Λ

부품을 집약하는 집약변에 부가되는 레이블의 집합을 나타낸다. ■

(그림 2)는 클래스계층그래프의 예를 보여주고 있다.



(그림 2) 클래스계층그래프의 예

2.3 클래스계층그래프에 관련된 제반정의

여기서는, 클래스계층그래프에 관한 몇가지 특징을 파악하기 위한 정의들을 살펴본다.

2.3.1 상속도달 가능

임의의 클래스계층그래프에 있어서, 추상정점 u로부터 정점v에 상속변이 존재하는 경우, (u, v)또는 u⇒v로 나타낸다. 이때, u는 v의 부모클래스, v는 u의 자식클래스라고 부른다. 또한, 1개 이상의 상속변을 경유하여 정점 u로부터 정점v에 도달할 수 있는 경우, u ⇒ +v로 나타내며, 이때, u는 v의 선조클래스, v는 u의 자손클래스라고 부른다. 그리고, u = v 또는 u⇒+v인 경우, u⇒*v로 나타내고, u로부터 v에 상속도달 가능하다라고 부른다. 또한, u로부터 v에 상속도달 가능한 경우, 도중에 경유하는 정점들의 열 $<u = w_1, w_2, \cdots, w_n = v>$을 상속경로하고 부르며, 상속경로 $<u = w_1, w_2, \cdots, w_n = v>$ 내의 변의 개수를 상속경로의 길이라고 부르고, LIR(u, v)로 나타낸다.

2.3.2 집약의 표현

임의의 클래스계층그래프에 있어서, 임의의 정점 x가 레이블 l로 정점 y를 집약하는 경우, [x, y, l] 또는 $x \rightarrow^{l} y$로 나타낸다. 또한, 정점 x가 메소드 m을 집약하고 있는 경우에는 [x, m] 또는 x ↦ m로 나타낸다.

2.3.3 메소드의 유일성

클래스계층그래프의 구상정점으로부터 생성되는 인스턴스가 메소드에 엑세스할 경우에는 메소드의 이름, 매개변수에 대하여 메소드의 구현부분이 일의적으로 결정될 필요가 있다. 이를 메소드의 유일성이라고 부르며, 다음과 같이 정의된다.

[정의 2] 메소드의 유일성

추상클래스 v에서부터 클래스 u까지 상속경로가 존재하는 경우, 이러한 상속경로상에서 정의되는 메소드명 M을 갖는 메소드는, 상속경로상의 가장 하위에 있는 클래스에 집약되어 있는 메소드명 M을 갖는 메소드의 인덱스를 i로 할 경우, M[i]가 된다. 클래스 v로 향하는 모든 상속경로상에 정의되는 메소드명 M을 갖는 메소드들에 있어서, 이러한 메소드들이 동일메소드인 경우, 클래스 v의 메소드 M은 일의적으로 결정된다고 한다. ■

2.3.4 상속도달 가능한 구상클래스의 집합

클래스 v로부터 상속도달 가능한 구상클래스의 집합 CS(v)는,

$$CS(v) = \{u \mid u \in VC \wedge v \Rightarrow *u\}$$

로 정의된다.

2.3.5 선조클래스의 집합

클래스 v까지 상속도달 가능한 모든 추상클래스의 집합 SCS(v)는,

$$SCS(v) = \{u \mid u \in VA \wedge u \Rightarrow *v\}$$

로 정의된다.

2.3.6 집약부품클래스의 집합

임의의 클래스 v가 직접집약 또는 상속집약하고 있는 부품의 집합 DIC(v)는,

$$DIC(v) = \{(l, CS(w)) \mid \forall u \in SCS(v) \cup \{v\} : l \in \Lambda \wedge [u, w, l]\}$$

로 정의된다.

2.3.7 집약메소드의 집합

임의의 클래스 v가 직접집약 또는 상속집약하고 있는 메소드의 집합 AMS(v)는,

$$AMS(v) = \{m \mid \forall u \in SCS(v) \cup \{v\} : u \mapsto m\}$$

로 정의된다.

2.4 정당한 클래스계층그래프

다음의 조건을 만족하는 클래스계층그래프를 정당한 클래스계층그래프라고 부른다.

① 상속변만으로 구성되는 싸이클이 존재하지 않는다.
② 집약변만으로 구성되는 싸이클이 존재하지 않는다.
③ 상속변과 집약변이 혼재된 싸이클을 갖지 않는다.
④ 1개의 정점이 2개 이상의 동명메소드정점을 갖지 않는다.
⑤ 모든 구상정점에 있어서, 구상정점이 상속집약하는 메소드에 대한 유일성이 성립한다.

정당하지 않은 클래스계층그래프로부터는 의미있는 객체가 생성될 수 없으므로, 올바른 형태의 클래스계층구조로서 부적절하다. 따라서, 본 논문에서는, 재구성 도중에 출현하는 경우를 제외하고 정당한 클래스계층그래프만 취급하도록 한다.

3. 클래스계층그래프간의 객체등가관계

여기서는, [2, 3]에서 정의된 클래스계층그래프간의 객체등가관계의 정의와, 객체등가관계를 유지하면서 클래스계층그래프를 재구성하는 기본적인 조작에 대하여 설명한다.

3.1 정 의

클래스계층간의 객체등가관계는 다음과 같이 정의된다.

[정의 3] 객체등가관계

임의의 두 CHG Γ_1, Γ_2가,

① $VC_{\Gamma 1} = VC_{\Gamma 2}$,

② $\forall v \in VC_{\Gamma 1}[DIC_{\Gamma 1}(v) = DIC_{\Gamma 2}(v) \wedge AMS_{\Gamma 1}(v)$

$= AMS_{\Gamma_2}(v)]$

을 만족하는 경우, Γ_1와 Γ_2는 객체등가라고 부르며, $\Gamma_1 \equiv \Gamma_2$로 표시한다. ■

2개의 클래스계층그래프에 있어서, 구상정점의 집합이 같고, 각 구상정점이 직접집약 및 상속집약하고 있는 부품집합과, 메소드정점집합이 양쪽 클래스계층그래프에서 같은 경우, 양쪽 클래스계층그래프는 객체등가관계에 있다고 부른다.

3.2 기본 등가재구성 조작

여기서는, 객체등가관계를 유지하면서 클래스계층그래프를 재구성하는 기본적인 조작에 대해서 설명한다. 이하에서 설명할 9개의 조작을 기본 등가재구성조작이라고 부른다.

3.2.1 쓸모없는 추상정점의 삭제(DUA)

CHG 상의 어떤 추상정점 v가

- 부품 및 메소드정점을 집약하고 있지 않음,
- v를 부품으로 갖는 다른 클래스가 존재하지 않음,
- v의 부모클래스가 존재하지 않음

과 같은 경우, 추상정점 v는 쓸모없는 클래스가 된다. DUA는 이와같은 쓸모없는 추상정점 v를 삭제한다. 추상정점 v와 v의 자식클래스에 해당하는 정점들을 $c_0, \cdots, c_m$으로 가정하여 정점 v를 제거할 경우, DUA(v, Γ)는, v와 클래스계층그래프 Γ가 입력으로 주어지며,

$$VA \leftarrow VA - \{v\}$$
$$EI \leftarrow EI - \{(v, c_0), \cdots, (v, c_m)\}$$

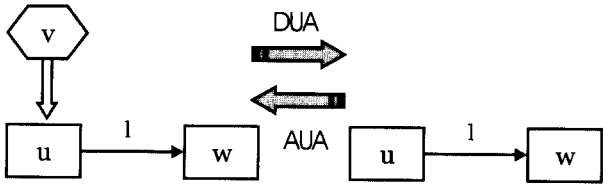
에 의해 변경된다(그림 3).

3.2.2 쓸모없는 추상정점의 추가(AUA)

쓸모없는 추상정점의 삭제조작에 대한 역변환조작이다. 부품, 메소드 정점, 부모클래스정점을 갖지 않는 추상정점을 클래스계층그래프에 추가한다. 추상정점 v와, v의 자녀클래스가 될 정점들을 $v_0, \cdots, v_m$ 으로 가정하여 v를 추가하는 경우, AUA(v, $\{v_0, \cdots, v_m\}$, Γ)는,

$$VA = VA \cup \{v\}$$
$$EI = EI \cup \{(v, v_0), \cdots, (v, v_m)\}$$

에 의해 변경된 클래스계층그래프를 출력한다(그림 3).



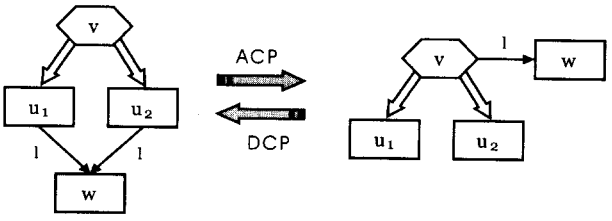
(그림 3) DUA 및 AUA의 예

3.2.3 공통부분의 추상화(ACP)

자녀클래스정점이 공통적인 레이블을 갖는 집약변으로 집약하고 있는 부품을, 부모클래스정점쪽으로 끌어올린다. 정점 $v_0, \cdots, v_m$이 공통적인 부품 p를 레이블 l로 집약하고 있고, 공통적인 부모클래스정점 v가 존재하는 경우, ACP(v, Γ)는,

$$EA = EA - \{[v_0, p, l], \cdots [v_m, p, l]\}$$
$$EA = EA \cup \{[v, p, l]\}$$

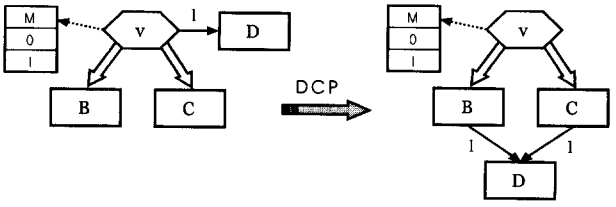
에 의해 변경된 클래스계층그래프를 출력한다(그림 4).



(그림 4) ACP 및 DCP의 예

3.2.4 공통부분의 분배(DCP)

클래스계층그래프상의 어떤 추상정점 v가, 레이블 l로 부품w를 집약하고 있는 경우, 모든 자녀클래스정점들에게 w를 분배한다. 단, 정점 v가 레이블 l을 참조하는 메소드정점 m을 집약하고 있는 경우에는, 자녀클래스정점에 부품을 분배해 버리면 메소드정점 m으로부터 레이블 l을 참조할 수 없게 되므로, 부품을 분배할 수 없다(그림 5).



(그림 5) DCP를 적용하여 부품을 분배할 수 없는 경우의 예

정점 v가 부품 p를 레이블 l로 집약하고 있고, 자녀클래스정점 $v_0, \cdots, v_m$을 갖는 경우에, DCP(l, v, Γ)은 l, v와 클래스계층그래프 Γ을 입력으로 하여,

$$EA = EA - \{[v, p, l]\}$$
$$EA = EA \cup \{[v_0, p, l], \cdots [v_m, p, l]\}$$

에 의해 변경된 클래스계층그래프를 출력한다(그림 4).

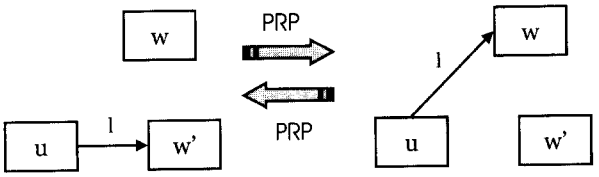
3.2.5 부분적인 부품교환(PRP)

집역변은 반드시 구상정점으로부터 생성되는 인스턴스를 집약하기 때문에, 집약변이 가리키는 정점으로부터 상속도달가능한 구상정점의 집합에 변화가 없는 두개 정점을 자유로이 교환할 수 있다. PRP(a, b, Γ)은, 정점 a, b와 클래스계층그래프 Γ를 입력으로하여, 정점 v가 부품 a를 레이블

l로 집약하고 있으며, CS(a) = CS(b)인 정점 b가 존재하는 경우,

$$EA = EA - \{[v, a, l]\}$$
$$EA = EA \cup \{[v, b, l]\}$$

에 의해 변경된 클래스계층그래프를 출력한다(그림 6).



(그림 6) PRP의 예

3.2.6 메소드의 추상화(ACM)

자녀클래스정점들이 공통적으로 집약하고 있는 메소드정점을, 부모클래스정점으로 끌어올린다. 단, 다음과 같은 조건이 만족되는 경우에 한하여 메소드의 추상화가 가능하다.

① 끌어올릴 부모클래스정점의 모든 자녀클래스정점들이 동일한 메소드를 집약하고 있다.

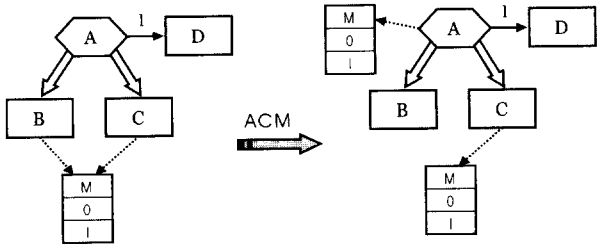
② 메소드정점을 집약하고 있는 정점에 메소드정점이 참조하고 있는 레이블을 갖는 집약변이 존재하지 않는다.

③ 메소드정점을 부모클래스정점으로 끌어올릴 경우에 메소드의 유일성이 성립한다.

정점 $v_0, \cdots, v_n$이 공통적인 메소드정점 a을 집약하고 있고, 공통의 부모클래스정점 v가 존재하는 경우, 메소드정점을 끌어올릴 정점을 $v_{j0}, \cdots, v_{jm} : 0 \leqq ji \leqq n$으로 하면, ACM(v, $\{v_{j0}, \cdots, v_{jm}\}$, Γ)은, 부모클래스정점과 메소드정점을 끌어올릴 정점들을 $v_{j0}, \cdots, v_{jm}$, 클래스계층그래프 Γ를 입력으로하여,

$$EM = EM - \{[v_{j0}, a], \cdots, [v_{jm}, a]\}$$
$$EA = EA \cup \{[v, a]\}$$

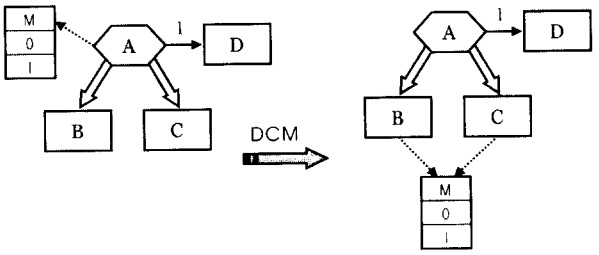
에 의해 변경된 클래스계층그래프를 출력한다(그림 7).



(그림 7) ACM의 예

3.2.7 메소드의 분배(DCM)

부모클래스정점이 집약하고 있는 메소드정점을 모든 자녀클래스정점에 분배한다. 자녀클래스정점이 동일한 메소드를 집약하고 있는 경우에는 메소드정점을 분배하지 않는다. 정점 v가 메소드정점 a를 집약하고 있고, 자녀클래스정점 $v_0, \cdots, v_m$을 갖는 경우,



(그림 8) DCM의 예

$$EM = EM - \{[v, a]\}$$
$$EM = EM \cup \{[v_0, a], \cdots, [v_m, a]\}$$

에 의해 변경된 클래스계층그래프를 출력한다(그림 8).

3.2.8 쓸모없는 메소드의 추가(AUM)

어떤 정점 v가 메소드정점 a를 상속집약하고 있는 경우, 정점 v가 메소드정점 a를 직접 집약하도록 추가한다. 정점 v가 메소드정점 a를 집약하도록 추가하는 경우에는

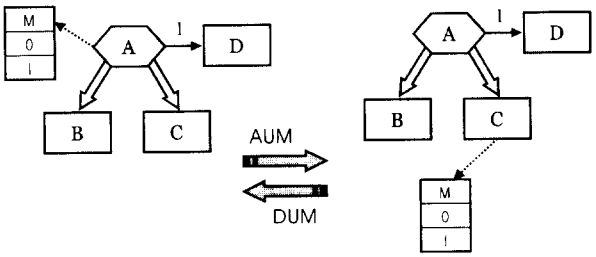
$$EM = EM \cup \{[v, a]\}$$

에 의해 변경된 클래스계층그래프를 출력한다(그림 9).

3.2.9 쓸모없는 메소드의 삭제(DUM)

어떤 정점 v가 메소드정점 a를 직접집약하고 있고, a를 상속집약하고 있는 경우, 정점 v가 직접집약하고 있는 메소드 정점a를 삭제한다. 정점 v가 직접집약하고 있는 메소드정점 m을 삭제하는 경우에는,

EM = EM - {[v, a]}에 의해 변경된 클래스계층그래프를 출력한다(그림 9).



(그림 9) AUM 및 DUM의 예

4. 클래스계층그래프의 평가척도

각각의 클래스에 대한 평가척도 [7-12]들을 토대로하여 클래스계층그래프의 평가척도들을 정의한다.

4.1 상속의 깊이(DIT)

어떤 클래스 v의 Root 클래스[1]로부터의 상속경로의 길이 중에서, 가장 긴 것을 상속의 깊이라고 부르며, DIT(v)로 나타낸다. 상속의 깊이는 다음과 같은 특징에 대한 지표가 된다.

- 상속의 깊이가 깊은 클래스가 많이 존재하는 클래스계층그래프는, 속성 및 메소드의 상속집약이 많을 가능성이 높으며, 전체적인 형태를 파악하기 어려울 가능성이 높다.
- 상속의 깊이가 얕은 클래스상속구조에서는, 많은 수의 클래스 및 메소드와 관계하므로, 클래스계층구조가 복잡하게 된다.

본 논문에서는, 상속의 깊이에 관한 클래스계층그래프에 대한 평가척도로서, 각 클래스마다 상속의 깊이에 대한 산술평균, 표준편차, 최대값을 설정한다. 이러한 척도를 토대로 클래스계층그래프에 대한 상속의 깊이에 관한 대체적인 정보를 얻을 수 있다.

4.2 각 클래스가 갖는 자녀클래스의 개수(NOC)

어떤 클래스 v가 갖는 자녀클래스의 개수를 NOC(v)로 나타낸다. 자녀클래스의 개수는 다음과 같은 특징을 나타낸다.

- 다수의 자녀클래스를 갖는 클래스가 많이 존재하는 클래스계층구조는, 재이용성이 높은 클래스를 많이 포함하고 있을 가능성이 높게 된다.
- 다수의 자녀클래스를 갖는 클래스가 많이 존재하는 클래스계층구조는, 추상화가 부적절하게 적용되었을 가능성이 있다.

본 논문에서는 클래스계층그래프내의 각 클래스가 갖는 자녀클래스의 개수에 대한 평가척도로서, 각 클래스가 갖는 자녀클래스의 개수에 대한 산술평균, 표준편차, 최대값을 설정한다. 이러한 척도를 토대로, 클래스계층그래프의 각 클래스가 갖는 자녀클래스의 개수에 대한 대체적인 정보를 얻을 수 있다.

4.3 다중상속의 개수(NMI)

2개 이상의 클래스를 다중상속하고 있는 클래스의 개수(NMI)는, 다음과 같은 특징을 파악하기 위한 척도가 된다.

- 다중상속을 이용하면, 어떤 클래스를 조사하려는 경우에 참조해야할 정보량이 증가하게 되므로, 조사가 난해해 질 수도 있다.
- 다중상속을 사용하므로써, 클래스가 갖게 되는 부품들을 효율적으로 집약시킬 수 있다.

1) 클래스계층그래프 Γ에서, 상속변의 입력차수가 0인 추상클래스를 클래스계층그래프 Γ의 Root 클래스라고 부른다. Root 클래스는 클래스계층그래프 Γ에 1개이상 존재하며, Root 클래스의 집합을 RC로 나타낸다.

4.4 각 클래스의 집약부품의 개수(NAC)

집약하고 있는 부품이 많을 수록, 다른 클래스와 의존관계가 증가한다. 어떤 클래스 v가 집약하고 있는 부품의 개수를 NAC(v)로 나타낸다. 집약부품은 다음과 같은 특징을 파악하기 위한 지표가 된다.

- 부품의 집약수가 극단적으로 많은 클래스를 갖고 있는 클래스계층구조는, 재이용성이 낮은 클래스가 많이 존재하므로, 클래스계층구조 전체의 재이용성도 낮아지게 될 가능성이 높다.
- 클래스들 사이의 집약관계가 많은 클래스계층구조는, 어떤 부분을 변경할 때마다 다른 부분에 끼치는 영향이 커지게 된다.

본 논문에서는 클래스계층그래프 내의 각 클래스가 직접 집약하고 있는 부품의 개수에 대하여, 각 클래스의 집약부품수에 대한 산술평균, 표준편차, 최대값을 설정한다. 이러한 지표를 토대로, 클래스계층구조 전체의 부품집약에 관한 정보를 얻을 수 있다.

4.5 각 클래스의 집약메소드의 개수(NOM)

어떤 클래스 v가 집약하고 있는 메소드의 개수를 NOM(v)로 나타낸다. 메소드의 개수는 다음과 같은 특징을 파악하기 위한 척도가 된다.

- 집약하고 있는 메소드의 개수가 많은 클래스는, 클래스에 정의된 메소드가 많으므로, 클래스의 규모가 증가하게 될 가능성이 높다. 이와같은 클래스가 많이 포함된 클래스계층구조는, 대규모화되어 유지보수가 곤란하게 될 가능성이 높다.

클래스계층그래프 내의 각 클래스가 직접집약하고 있는 메소드의 개수에 대하여, 각 클래스가 직접집약하고 있는 메소드의 개수에 대한 산술평균, 표준편차, 최대값을 지표로하여, 클래스계층구조 전체의 메소드집약에 관한 정보를 파악할 수 있다.

5. 클래스평가기준에 따른 클래스계층구조 변환 알고리즘

여기서는, 제 4장에서 논의한 평가척도들을 토대로하여, 클래스계층그래프를 객체등가관계를 유지하면서 변환하는 알고리즘에 대하여 설명한다. 우선, 앞에서 논의한 5개의 평가척도들을 토대로 어떤 한 정점에 주목하여 변환을 수행하는 기본 알고리즘을 제안한다. 그리고, 기본알고리즘을 클래스계층그래프에 지역적으로 적용했을 경우에 다른 평가척도들이 어떻게 변화하는가를 살펴보고, 이를 토대로 클래스계층그래프 전체에 적용하기 위한 광역적인 알고리즘을 제안한다.

5.1 기본 알고리즘

여기서는, 클래스계층그래프상에 있는 어떤 한 정점에 대하여 적용하고, 앞에서 논의한 평가척도들을 토대로 지역적인 재구성을 수행하는 기본 알고리즘들을 제안한다. 또한, 이러한 알고리즘들이 객체등가관계를 만족하는 재구성 조작임을 증명한다. 각 기본 알고리즘은,

- 다중상속하고 있는 클래스를 삭제
- 어떤 추상클래스정점을 삭제
- 추상클래스정점을 추가

하는 재구성조작을 수행한다.

5.1.1 어떤 클래스의 다중상속을 삭제하는 알고리즘(CMI : Cancel Multiple Inheritance)

임의의 클래스계층그래프 Γ와, Γ상의 추상클래스정점 v가 주어지면, 추상클래스정점 v가 다중상속을 하고 있는지 여부를 판단하고, 다중상속인 경우, 해당되는 다중상속을 삭제하는 알고리즘이다.

```
Algorithm CMI(v, Γ)
  입력 : 클래스계층그래프 Γ, Γ내의 임의의 추상클래스정점 v
  출력 : 추상클래스정점 v가 다중상속하고 있는 경우, v의 다중상속
         을 소멸시킨 클래스계층그래프와 true 값을 출력한다. 다중
         상속이 아닌 경우에는 입력된 클래스계층그래프 Γ와 false
         값을 되돌려준다.
    Γ_init := Γ ;
    ParentSet ← {v의 부모클래스정점} ;
    loop while |ParentSet| > 1 do
      p := (ParentSet의 요소들 중에서, 상속의 깊이가 최소인
           클래스정점) ;
      Target ← {p까지 상속도달 가능한 정점들의 집합}
               ∩ {p 이외의 v의 부모클래스정점까지 상속도달
                  가능하지 않은 정점들의 집합}
       if ∃c ∈ Target[∃u ∈ VA ∪ VC,
            l ∈ Λ : [u, c, l] ∈ EA] then
          Γ := PRP(c, c', Γ) ;
       else
          ParentSet ← ParentSet - {p} ;
          continue ;  // while루프의 처음으로 돌아감.
       end_if ;
       if ∃x ∈ VA ∪ VC, l ∈ Λ[(v, x, l) ∈ EA] then
          for each l : (v, x, l) ∈ EA do Γ := DCP(l, v, Γ) ;
       end_if ;
       if ∃m ∈ VM[(v, m) ∈ EM] then
          for each m : (v, m) ∈ EM do Γ := DCM(m, v, Γ) ;
       end_if ;
       EI ← EI - {(p, v)} ;
       ParentSet ← ParentSet - {p} ;
    end_loop ;
    if |{v의 부모클래스정점}| > 1 then
       return (Γ_init, false) ;
    else
       return (Γ, true) ;
    end_if ;
end_of_CMI
```

5.1.2 어떤 추상클래스정점을 삭제하는 알고리즘(DAC : Decrease Abstract Class)

클래스계층그래프내의 어떤 한 추상클래스정점을 객체등가관계를 유지하면서 삭제가능한가를 판단해 보고, 삭제가능하다면, 해당 추상클래스정점을 삭제하는 알고리즘이다.

```
Algorithm DAC(v, Γ)
  입력 : 클래스계층그래프 Γ와 Γ내의 임의의 추상클래스정점 v
  출력 : 추상클래스정점 v가 삭제 가능하다면 v를 제거한 클래스계
         층그래프와 true 값을 되돌려 준다. 삭제불 가능한 경우에
         는 입력된 Γ와 false 값을 출력한다.
  Γ_init := Γ ;
  if |{p | p ∈ VA, (p, v) ∈ EI}| > 1 then
     (Γ, result) := CMI(v, Γ) ;
  if ∃w ∈ VA ∪ VC, l ∈ Λ[(w, v, l) ∈ EA] then
     if ∃v' ∈ VA{CS(v) = CS(v')] then Γ := PRP(v, v', Γ) ;
     else return (Γ_init , false) ;
     end_if ;
  end_if ;
  if ∃m ∈ VM[(v, m) ∈ EM] then
  for each m : (v, m) ∈ EM do Γ := DCM(m, v, Γ) ;
  end_if ;
  if ∃x ∈ VA ∪ VC, l ∈ Λ[(v, x, l) ∈ EA] then
       for each l : (v, x, l) ∈ EA do Γ := DCP(l, v, Γ) ;
  end_if ;
  if ∃p ∈ VA[(p, v) ∈ EI] then
       for each p : (p, v) ∈ EI do
            for each c ∈ VA ∪ VC : (v, c) ∈ EI do
                EI ← EI ∪ {(p, v)} ;
            end_for ;
            EI ← EI - {(p, v)} ;
       end_for ;
  end_if ;
  Γ := DUA(v, Γ) ;
  return(Γ, true) ;
end_of_DAC
```

5.1.3 추상클래스정점을 추가하는 알고리즘(AAC : Add Abstract Class)

클래스계층그래프 Γ와, Γ내에 있는 어떤 클래스를 나타내는 정점들의 집합, 그리고 추상클래스정점 v를 입력하면, 입력된 Γ에 추상클래스정점 v를 추가하고, v와 Γ내에 있는 클래스정점들을 상속변으로 연결시킨 새로운 클래스계층그래프를 출력하는 알고리즘이다.

```
Algorithm AAC(v, {c1, ··· cn}, Γ)
  입력 : 추상클래스정점 v, v와 상속변으로 연결될 정점들의 집합
         {c1, ··· cn}, 클래스계층그래프 Γ
  출력 : 입력된 Γ에 추상클래스정점 v를 추가한 클래스계층그래프
  Γ_init := Γ ;
  Γ := AUA(v, {c1, ··· cn}, Γ) ;
  if ∃p ∈ VA[∀c ∈ {c1, ··· cn} : (p, c) ∈ EI] then
       for each c ∈ {c1, ··· cn} do
            EI ← EI - {(p, c)} ;
       end_for ;
            EI ← EI ∪ {(p, c)} ;
  end_if ;
  Γ := ACP(v, Γ) ;
  Γ := ACM(v, {c1, ··· cn}, Γ) ;
end_of_AAC
```

5.2 기본알고리즘 적용시 다른 평가기준들에게 미치는 영향

여기서는, 앞에서 제안한 3가지 기본알고리즘을 클래스계

층그래프에 적용했을때, 제 4장에서 설명한 각 평가척도에 어떠한 영향을 끼치는가를 살펴본다.

5.2.1 알고리즘 CMI

알고리즘 CMI를 클래스계층그래프에 1회 적용시켰을 때, 각 평가척도들은 다음과 같이 변화한다.

(1) DIT

상속변을 삭제할 경우, 다중상속을 하고 있는 정점v의 부모클래스정점들의 집합속에 있는 원소들중에서 상속의 깊이가 가장 얕은 원소(정점)를 순서적으로 선택하므로, 마지막에는 상속의 깊이가 가장 깊은 부모클래스정점이 남는다. 상속의 깊이는 경로가 복수개 존재할 경우 최대값을 갖는 것을 선택하므로, CMI를 적용했을 경우에 상속의 깊이는 변화하지 않는다.

(2) NOC

v에 입력되는 상속변을 삭제할 경우, (삭제할 수 있는 상속변의 개수) ≤ (v의 부모클래스정점들의 개수 - 1)이 성립한다. 클래스계층그래프 내에 존재하는 모든 클래스들의 자녀클래스 개수에 대한 총합은 |EI|과 같으므로, 감소되는 자녀클래스의 개수는 최대(v의 부모클래스의 개수 - 1)개가 된다. 따라서, NOC에 미치는 영향을 최소화하기 위해서는, 다중상속하고 있는 정점들 중에서 부모클래스정점의 개수가 적은 정점을 선택하면 된다.

(3) NMI

CMI는 어떤 한개의 정점에 대하여 다중상속을 삭제하는 알고리즘이므로, 다중상속이 삭제가능한 경우, 다중상속의 수(NMI)는 1 감소하며, 그 외에는 변화하지 않는다.

(4) NAC와 NOM

v에 입력되는 상속변을 삭제하는 경우, 삭제되는 상속변에 의해 상속되는 부품 및 메소드를 모두 v에 집약하는 조작을 수행하므로, v가 갖는 부품 및 메소드의 개수는 증가하거나 변화하지 않는다. 따라서, NAC와 NOM에 미치는 영향을 최소화하기 위해서는 Root 클래스로부터 v까지의 경로들 중에서, 각각 부품과 메소드를 집약하는 정점이 최소인 경로를 통해 상속하는 정점을 선택하면 된다.

5.2.2 알고리즘 DAC

알고리즘 DAC를 클래스계층그래프에 존재하는 추상클래스정점 v에 적용했을 경우, 각 평가척도들은 다음과 같이 변화한다.

(1) DIT

① v로부터 상속도달가능한 정점이 다중상속하지 않는 경우

v의 자손클래스들에 대한 상속의 깊이는 v를 삭제하므로써 1씩 감소한다. 즉, DIT의 변화량은 DIT(v) + |{v로부터 상속도달가능한 정점}|이 된다. 따라서, DAC를 1회 적용했을 경우에 DIT 척도에 미치는 영향을 최소화하기 위해서는, DIT(v) + |{v로부터 상속도달가능한 정점}|의 값이 최소인 정점을 선택하여 DAC를 적용한다.

② v로부터 상속도달 가능한 정점이 다중상속하는 경우

(그림 10)과 같이 v에서는 p 이하의 정점에 대한 상속의 깊이에 관여하지 않는다. 따라서, DAC를 1회 적용했을 때 DIT 척도에 미치는 영향을 최소화하기 위해서는, Root 클래스로부터 다중상속하고 있는 정점 p에 이르는 경로들 중에서, 경로의 길이가 최대인 것 이외의 경로상에 존재하는 추상클래스정점을 선택하고, 또한, v로부터 p까지의 상속경로에 있어서, DIT(v) + |{v로부터 상속도달가능한 정점} - {p로부터 상속도달가능한 정점}|의 값이 최소인 정점을 선택한다.

(2) NOC

① v가 다중상속을 하고 있지 않는 경우

v를 삭제하므로써 v의 부모클래스정점의 NOC 척도가 |{v의 자녀클래스정점의 개수}| - 1이 된다. 따라서, DAC의 적용에 의해서 NOC에 미치는 영향을 최소화하기 위해서는 자녀클래스정점이 최소인 정점을 선택하면 된다.

② v가 다중상속을 하고 있는 경우

다중상속을 삭제가능한 경우에는 v가 다중상속을 하지 않는 경우와 마찬가지로, |{v의 부모클래스정점}|만큼 NOC가 감소한다. 그러나, 다중상속을 삭제할 수 없는 경우에는, DAC는 v의 모든 부모클래스정점으로부터 모든 자녀클래스정점으로 향하는 상속변을 추가하므로, v의 부모클래스정점 각각의 NOC가 |{v의 자녀클래스정점의 개수}| - 1이 된다. 따라서, DAC의 적용에 의한 NOC 척도에 미치는 영향을 최소화하기 위해서는, 다중상속하고 있는 정점은 가능한한 선택하지 않고, 어쩔 수 없이 선택한다면 다중상속이 삭제가능한 정점 또는 자녀클래스정점이 최소인 정점을 선택하면 된다.

(3) NMI

v가 다중상속을 하고 있는 경우, 1 감소한다. DAC의 적용에 의한 NMI척도에 미치는 영향을 최소화하기 위해서는 다중상속하지 않는 정점v를 선택한다.

(4) NAC와 NOM

DAC에서는, v가 집약하고 있는 정점들을 v의 모든 자녀클래스정점들에게 분배한다. 따라서, DAC를 1회 적용했을 경우의 NAC 및 NOM의 변화량은 각각, |{v의 집약부품}| ×|{v의 자녀클래스정점}|과 |{v가 집약하고 있는 메소드}| ×|{v의 자녀클래스정점}|가 된다. DAC를 적용할 경우 각

평가척도(NAC, NOM)에 미치는 영향을 최소화하기 위해서는 위의 NAC 및 NOM의 변화량을 최소화할 수 있는 정점 v를 선택한다.

5.2.3 알고리즘 AAC

알고리즘 AAC를 클래스계층그래프에 적용하여 추상클래스정점 v를 삽입했을 경우, 각 평가척도는 다음과 같이 변화한다. 단, 본 논문에서는 아래에서 논의할 알고리즘으로 AAC를 이용할 때, v로부터 상속변으로 연결되는 정점들의 집합 $\{c_1, \cdots c_n\}$을 AAC에 입력하지만, 이와같은 집합의 요소가 되는 정점들은 모두 공통적인 부모클래스정점을 갖고 있는 것만을 입력하므로, 여기서는 이와같은 특징을 고려하여 각 평가척도들을 고찰해 본다.

(1) DIT

AAC를 적용하면, 집합$\{c_1, \cdots c_n\}$의 각 요소들로부터 상속도달가능한 정점에 대한 상속의 깊이가 1 증가한다. 따라서, AAC의 적용시, DIT에 미치는 영향을 최소화하기 위해서는 $\{c_1, \cdots c_n\}$의 각 요소들로부터 상속도달 가능한 정점의 수가 최소가 되는 정점을 선택한다.

(2) NOC

AAC를 적용하면, v의 부모클래스정점의 NOC는 $|\{c_1, \cdots c_n\}| - 1$만큼 감소하며, v의 NOC는 $|\{c_1, \cdots c_n\}|$만큼 증가한다.

(3) NMI

추상클래스정점을 추가할 경우, v로부터 상속변으로 연결되는 정점들의 집합 $\{c_1, \cdots c_n\}$은 반드시 공통의 부모클래스정점을 갖게 되므로, 반드시 부모클래스정점으로부터 1개의 상속변으로 연결된다. 또한, 집합$\{c_1, \cdots c_n\}$의 요소들 중에서 다중상속하고 있는 정점이 존재하더라도, v에 관계하는 변 이외는 감지하지 않는다. 따라서, NMI 척도값에는 변화없다.

(4) NAC 및 NOM

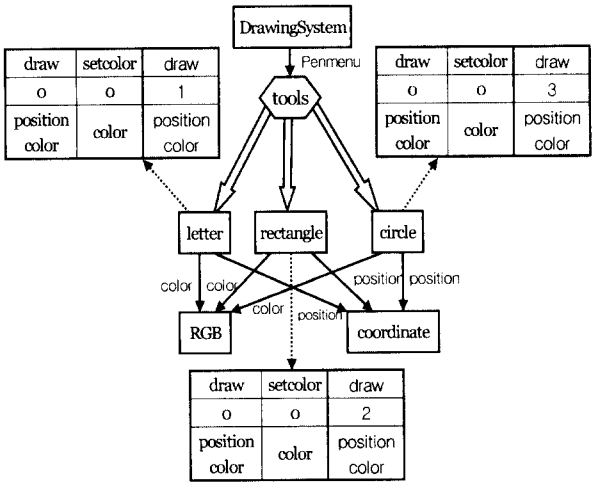
AAC에서는 $\{c_1, \cdots c_n\}$의 모든 정점들이 같은 부품 및 메소드를 집약하고 있는 경우, v에 끌어올려 추상화한다. 따라서, NAC 및 NOM의 변화량은 각각, $|\{$v에 끌어올려진 부품$\}| \times |\{c_1, \cdots c_n\}|$과 $|\{$v에 끌어올려진 메소드$\}| \times |\{c_1, \cdots c_n\}|$ 만큼 감소하며, v에서는 1씩 증가한다. 따라서, AAC를 적용할 경우, $\{c_1, \cdots c_n\}$의 공통부품이 최소화되도록 선택하면 된다.

5.3 적용 사례

여기서는 본 논문에서 정의한 기본 등가재구성조작 및 변환알고리즘들을 적용하여 클래스계층구조를 재구성했을 경우에 제반 평가척도들이 어떻게 변화하는가를 설명한다. 단, 제안된 알고리즘들은 유사한 사례를 이용하여 재구성에 따른 평가척도의 변화추이를 쉽게 유추할 수 있으므로, 여기서는 지면관계상, 제안된 알고리즘들 중에서 어떤 추상클래스정점을 삭제하는 알고리즘(DAC : Decrease Abstract Class)을 적용한 사례만을 살펴보기로 한다.

(그림 2)에 주어진 클래스계층그래프에 대하여, 추상클래스정점 figure를 삭제하는 알고리즘(DAC)를 적용하는 경우, 모든 메소드정점을 구상클래스가 직접 집약하도록 기본 등가재구성조작중에서 메소드의 분배(DCM)조작을 적용하여 재구성한 후에, 클래스계층그래프에 포함되어 있는 추상클래스정점이 집약하고 있는 부품에 대하여 공통부분을 분배하는 조작(DCP)을 적용하여 구상클래스정점까지 끌어내린다. 모든 메소드정점들은 구상클래스정점에 집약되어 있으므로, 공통부분의 분배조작을 자유롭게 적용할 수 있다.

또한, 이와같은 공통부분의 분배조작을 적용한 이후에 추상클래스정점 figure는 삭제가능한 추상클래스정점이 되므로 (그림 10)과 같이 figure 클래스가 삭제된 클래스계층구조로 재구성된다.



(그림 10) 재구성알고리즘 적용사례

이와같은 재구성알고리즘의 적용에 의해 각 평가척도들은 다음과 같이 변화하였다. 단, 주어진 클래스계층그래프의 정점들 중에서 재구성에 따른 척도값 변화영향을 많이 받는 5개 클래스정점(tools, figure, letter, rectangle, circle)에 대한 평가척도값의 변화량을 살펴보기로 하며, 다중상속이 없는 클래스계층구조이므로 NMI 척도에 대한 비교는 수행하지 않는다.

위의 평가값의 비교로부터, 클래스 figure의 자손클래스들에 대한 상속의 깊이는 figure를 삭제함으로써 1씩 감소하였고, figure를 삭제하므로써 figure의 부모클래스정점의 NOC값은 1감소하였다. 한편, (그림 2)의 figure 클래스가 집약하고 있는 부품들은 모두 figure의 모든 자녀클래스들

에게 분배되므로, NAC 및 NOM 척도값은 2씩 증가되지만, 이와같이 클래스계층구조를 재구성함으로써 구상클래스정점으로부터 생성되는 객체들이 집약하고 있는 부품 및 메소드들을 파악하기 수월해진다. 또한, 위의 재구성에서는, 메소드를 구상클래스정점까지 끌어내리기 위하여 DCM을 적용하고, 공통부분의 분배를 위하여 DCP을 적용하였으므로, 재구성 전후의 클래스계층구조들 사이에는 객체등가관계가 보존된다.

〈표 2〉 재구성 전후의 평가척도값의 변화

평가척도	평가대상 클래스	재구성전의 클래스계층 그래프 (그림 2)	재구성후의 클래스계층 그래프 (그림 10)	변화량
상속의 깊이 (DIT)	tools	1	1	0
	figure	2	0(삭제됨)	-2
	letter	2	2	0
	rectangle	3	2	-1
	circle	3	2	-1
각 클래스가 갖는 자녀클래스의 개수(NOC)	tools	4	3	-1
	figure	2	0(삭제됨)	-2
	letter	0	0	0
	rectangle	0	0	0
	circle	0	0	0
각 클래스의 집약부품의 개수 (NAC)	tools	2	0	-2
	figure	0	0(삭제됨)	0
	letter	0	2	+2
	rectangle	0	2	+2
	circle	0	2	+2
각 클래스의 집약메소드의 개수(NOM)	tools	2	0	-2
	figure	0	0(삭제됨)	0
	letter	1	3	+2
	rectangle	1	3	+2
	circle	1	3	+2

5.4 광역적인 변환 알고리즘

여기서는, 위에서 제안한 기본 알고리즘을 사용하여 클래스계층구조를 한개의 평가척도에 따라서, 다른 평가척도값에 미치는 영향을 최소화하여 변환하는 알고리즘들을 제안한다[2].

5.4.1 주어진 정점에 대한 상속의 깊이를 줄이는 알고리즘 (DDI : Decrease Depth of Inheritance)

DDI는 클래스계층그래프 Γ와, Γ내의 추상클래스정점, 또는 구상클래스정점 v와 허용변화율 x가 주어지면, 클래스계층그래프에 대한 각 평가기준의 변화율이 x를 초과하지 않는 범위내에서 최대한도로 DAC를 적용하여, v의 상속의 깊이를 줄이는 알고리즘이다. 이때, DAC를 적용하는 대상이 되는 정점은, 앞에서 논의한 바와 같이, 다중상속하지 않고, 자녀클래스정점의 개수가 최소, 부품 및 메소드의 집약변의 출력차수가 최소가 되는 것을 선택한다.

Algorithm DDI(v, x, Γ)
입력 : 클래스계층그래프 Γ, Γ내의 추상클래스정점(또는 구상클래스정점) v, 허용변화율 x
출력 : DDI를 적용하여 재구성된 클래스계층그래프
initMetrics := CALC(Γ) ;
Metrics := *initMetrics* ;
Current ← {Root 클래스로부터 v까지의 상속경로들 중에서 최장경로} ;
for each w ∈ Current do
color[w] := WHITE ;
loop while ∃w ∈ Current[color[w] = WHITE] ∧
$\left|\frac{initMetrics - Metrics}{initMetrics} \times 100\right| < x$ do
c := (c는 Current의 요소 ∧ 다중상속하지 않음 ∧ 자녀클래스정점의 개수가 최소 ∧ 부품 및 메소드의 집약변의 출력차수가 최소 ∧ color[c] = WHITE) ;
(Γ, result) := DAC(c, Γ) ;
if result = true then
Current ← Current - {c} ;
else
color[c] := BLACK ;
end_if
for each w ∈ Current do color[w] := WHITE ;
Metrics := CALC(Γ) ;
end_of_loop ;
end_of_DDI

5.4.2 주어진 정점에 대한 상속의 깊이를 늘리는 알고리즘(IDI : Increase Depth of Inheritance)

IDI는 클래스계층그래프 Γ와, Γ내의 추상클래스정점(또는 구상클래스정점) v와 허용변화율 x가 주어지면, 클래스계층그래프에 대한 각 평가척도의 변화율이 x를 초과하지 않는 범위내에서 AAC를 적용하여, v의 상속의 깊이를 늘리는 알고리즘이다. 이때, AAC는 Root 클래스로부터 v까지의 상속경로 속에 있는 정점 w를 집합{c_1, … c_n}의 한 요소로서 적용하지만, 정점 w는 앞에서 언급한 바와 같이, w의 부모클래스정점 p가 갖는 자녀클래스정점의 부분집합중에서, w를 포함하고 요소수가 NOC(p) - 1이 되는 집합의 각 요소들이 공통적으로 집약하고 있는 부품 및 메소드의 개수가 최소가 되는 정점을 선택한다.

Algorithm IDI(v, x, Γ)
입력 : 클래스계층그래프 Γ, Γ내의 추상정점 또는 구상정점 v, 허용변화율 x
출력 : IDI를 적용하여 재구성한 클래스계층그래프
initMetrics := CALC(Γ) ;
Metrics := *initMetrics* ;

2) 아래 알고리즘의 정의에서는, 평가척도의 값에 대한 계산방식이 비교적 상세하게 정의되어 있으나, 기술하기에 매우 복잡하게 되므로, "값을 계산한다"라는 함수 CALC를 사용한다. CALC(Γ)는, 클래스계층그래프 Γ에 대하여, 4장에서 제안한 모든 평가기준(평가척도)에 대한 값을 계산하여, 벡터(Vector) 형태로 반환한다.

```
    Current ← {Root 클래스로부터 v까지의 상속경로들 중에서
              가장 긴 경로} ;
    loop while |(initMetrics − Metrics)/initMetrics × 100| < x do
      w := (c는 Current의 요소로서, w의 부모클래스정점 p가 갖는
           자녀클래스정점들 중에서 w를 포함하는 정점들의 집합
           {c1, ··· ck}(k = NOC(p) − 1)이 공통적으로 집약하고 있
           는 부품 및 메소드의 개수가 최소) ;
      Γ := AAC(v, {c1, ··· ck}, Γ) ;
      Metrics := CALC(Γ) ;
    end_of_loop ;
end_of_IDI
```

5.4.3 자녀클래스정점의 수가 큰 정점을 가능한 줄이는 알고리즘(DNC : Decrease Number of Children)

알고리즘 DNC는 클래스계층그래프 Γ와 허용변화율 x가 주어지면, 클래스계층그래프에 대한 각 평가척도의 변화율이 x를 초과하지 않는 범위내에서 가능한 한 AAC를 적용하여 자녀클래스를 많이 갖는 정점의 자녀클래스를 줄이는 알고리즘이다. 클래스계층그래프 Γ내의 추상클래스들 중에서 NOC가 최대인 정점 w에 관하여, w의 자녀클래스정점에 AAC를 적용하므로써, NOC를 줄인다. 이때, AAC는 입력으로 주어지는 집합{c1, ··· cn}의 한 요소로서 정점 w를 적용하며, 정점 w는 앞에서 설명한 바와 같이 w의 부모클래스정점이 갖는 자녀클래스정점들 중에서 w를 포함하는 몇 개의 정점들로 구성된 집합을 C로하였을때, C로부터 상속도달가능한 정점들의 수가 최소이면서, C의 요소가 공통적으로 집약되어 있는 부품 및 메소드의 수가 최소가 되는 정점을 선택한다. 알고리즘 DNC는 다음과 같다.

```
Algorithm DNC(x, Γ)
 입력 : 클래스계층그래프 Γ, 허용변화율 x
 출력 : DNC를 적용하여 재구성한 클래스계층그래프
   initMetrics := CALC(Γ) ;
   Metrics := initMetrics ;
   loop while |(initMetrics − Metrics)/initMetrics × 100| < x do
     maxNOC ← {v | ∀u ∈ VA : NOC(u) ≤ NOC(v)} ;
     C(v) := {v ∈ maxNOC의 자녀클래스정점들 중에서 2개 정점} ;
     C(w) := (C(v)의 각 정점들로부터 상속도달가능한 정점의 수
             가 최소이면서, C(v)의 요소가 공통적으로 집약하고
             있는 부품 및 메소드의 수가 최소) ;
     Γ := AAC(v, C(w), Γ) ;
     Metrics := CALC(Γ) ;
   end_of_loop ;
end_of_DNC
```

5.4.4 자녀클래스정점의 개수가 큰 정점을 가능한 늘리는 알고리즘(INC : Increase Number of Children)

INC는 클래스계층그래프 Γ와 허용변화율 x가 주어지면, 클래스계층그래프에 대한 각 평가척도의 변화율이 x를 초과하지 않는 범위에서 가능한 한 DAC를 적용하여, 자녀클래스의 수가 적은 정점의 자녀클래스를 증가시키는 알고리즘이다. 이때, DAC를 적용하는 대상이 되는 정점v는, 앞에서 논의한 바와같이, v로부터 상속도달가능한 정점이 다중상속하고 있지 않을 경우에는, DIT(v) + |{v로부터 상속도달가능한 정점}|의 값이 적은 것을, 또는 v로부터 상속도달가능한 정점이 다중상속하고 있을 경우에는, Root 클래스로부터 다중상속하고 있는 정점 p로 향하는 경로들 중에서 경로의 길이가 최대인 것 이외의 경로상에서 추상정점 v를 선택하여, v로부터 p까지의 상속경로에 있어서, DIT(v) + |{v로부터 상속도달 가능한 정점} − {p로부터 상속도달 가능한 정점}|의 값이 적은 것을 선택한다.

```
Algorithm INC(x, Γ)
 입력 : 클래스계층그래프 Γ, 허용변화율 x
 출력 : INC를 적용하여 재구성한 클래스계층그래프
   initMetrics := CALC(Γ) ;
   Metrics := initMetrics ;
   for each v ∈ VA do color[v] = WHITE ;
   loop while ∃w ∈ Current[color[w] = WHITE]
         ∧ |(initMetrics − Metrics)/initMetrics × 100| < x do
     for each v ∈ VA do
       if ∃w ∈ ACS(v)[NMI(w) > 1] then
          Point(v) := DIT(v) + |{v로부터 상속도달가능한 정점}
                      − {w로부터 상속도달가능한 정점}| ;
       else
          Point(v) := DIT(v) + |{v로부터 상속도달가능한 정점}| ;
       end_if ;
     end_for ;
     u := (u ∈ VA ∧ Point(u)가 최소 ∧ color[u] = WHITE) ;
     (Γ, result) := DAC(u, Γ) ;
     if result = true then
        for each v ∈ VA do color[v] = WHITE ;
     else
        color[u] = BLACK ;
     end_if ;
     Metrics := CALC(Γ) ;
     end_of_loop ;
end_of_INC
```

5.4.5 다중상속하고 있는 클래스의 수를 가능한 줄이는 알고리즘(DMI : Decrease Multiple Inheritance)

DMI는 클래스계층그래프 Γ와 허용변화율 x가 주어지면, 클래스계층그래프에 대한 각 평가척도의 변화율이 x를 초과하지 않는 범위내에서, CMI를 적용하여 다중상속하고 있는 정점을 가능한 한 줄이는 알고리즘이다. 이때, CMI를 적용하는 대상이 되는 정점 v는, 앞에서 설명한 바와 같이 v로 향하는 상속변의 입력차수가 적으면서 상속집약하고 있는 부품 및 메소드가 적은 정점을 선택한다.

```
Algorithm DMI(x, Γ)
 입력 : 클래스계층그래프 Γ, 허용변화율 x
 출력 : DMI를 적용하여 재구성한 클래스계층그래프
   initMetrics := CALC(Γ) ;
   Metrics := initMetrics ;
   MISet ← {c|NMI(c) > 1} ;
```

```
loop while (MISet ≠ ∅) ∧ (| (initMetrics − Metrics) / initMetrics × 100 |
        < x) do
    v := (상속변의 입력차수가 최소이면서 상속집약하고 있는 부
        품 및 메소드의 수가 최소) ;
    (Γ, result) := CMI(v, Γ) ;
    MISet ← MISet − {v} ;
    MISet ← MISet − {v} ;
    Metrics := CALC(Γ) ;
  end_of_loop ;
end_of_DMI
```

6. 결 론

본 논문에서는, 객체지향분석 및 설계 초기단계에서 설계자가 클래스계층구조를 어떻게 구축하고 재구성할 것인가를 논의 대상으로 하고 있으며, 이러한 클래스계층의 구조적 특징을 추출하여 형식적으로 표현한 클래스계층그래프를 토대로 클래스계층구조를 표현하였다. 또한, 개발과정에서 설계자가 필요에 의해 클래스계층구조를 재구성(또는 재구조화, 구조조정)하는 경우가 빈번하게 발생되므로, 이에 대한 정형화된 재구성기법을 제안하였다.

또한, 본 논문에서는, 클래스계층그래프에 대한 평가척도로서, 상속의 깊이, 각 클래스가 갖는 자녀클래스의 수, 다중상속의 수, 각 클래스의 집약부품수, 각 클래스의 집약메소드의 수와 같은 5가지 척도를 제안했다. 지금까지의 관련 연구에서는, 대부분 각 클래스에 대한 평가척도에 중점을 맞추었으나, 이러한 클래스계층그래프에 대한 평가척도를 이용함으로써 클래스계층구조에 대한 전체적인 품질을 파악할 수 있게 되며, 클래스계층구조를 재구성할 경우에 어떻게 재구성하면 필요하지 않은 품질평가척도에 대한 영향을 최소화하여 재구성할 수 있는지에 대한 구체화된 방법을 제안하였다.

특히, 본 논문에서는, 제안한 클래스계층구조에 대한 평가척도들을 토대로 하여, 클래스계층구조의 재구성방침 5가지를 제안하고, 이러한 방침에 따라서 클래스계층구조를 객체등가관계를 보존하면서 재구성하는 알고리즘을 제안하였다. 클래스계층구조를 재구성할 경우에 사용자가 재구성방침들 중에서 어느 하나를 선택하여, 알고리즘을 클래스계층그래프에 적용함으로써, 객체등가관계를 보존하면서 사용자가 선택한 방침대로 클래스계층구조를 재구성할 수 있으므로, 클래스계층구조의 재구성에 대한 자동화가 용이하게 되며, 객체지향분석 및 설계시에 클래스계층구조의 재구성에 대한 작업자의 부담을 경감시킬 수 있다.

본 논문에서는, 분석 및 설계의 초기단계에서 설계자에게 유용한 기법을 제공하고자, 객체가 갖는 구조적 및 정적특징에 초점을 맞추어 클래스계층구조의 구조적인 관점에 국한시킨 정형화된 재구성기법을 제안하였다. 분석 및 설계의 초기단계에서는 문제영역내에 등장하는 객체들이 갖는 특성(속성 및 동작)을 추출하여 대략적인 클래스계층구조가 만들어져서, 반복/점증적으로 구체화되므로, 이와같은 구축단계에서 객체들사이의 "의미적 등가관계"를 논의하기 위해서는, 앞으로 객체의 정적구조 뿐만아니라 동적특징까지도 고려한 재구성기법에 관한 연구와 더불어서 실제적으로 사용할 수 있는 도구제작 등에 관한 연구가 앞으로의 과제이다.

참 고 문 헌

[1] Paul L. Bergstein, "Object Preserving Class Transformations," SIGPLAN Notices, ACM Press, Vol.26, No.11, pp. 299-313, 1991.

[2] 황석형, 양해술, 박정호, "객체지향 소프트웨어의 재구성을 위한 클래스계층구조의 평탄화", 정보처리학회논문지D, 제8-D권 제6호, 2001.

[3] 황석형, 양해술, "클래스계층그래프의 문법표현에 관한 연구", 소프트웨어공학논문지, 제4권 제4호, 2001.

[4] Lance Tokuda and Don Batory, "Evolving object-oriented designs with Refactorings," Journal of Automated Software Engineering, Vol.8, No.1, pp.89-120, 2001.

[5] Karl Lieberherr and Ian Holland, Assuring Good Style for Object-Oriented Programs, IEEE Software, pp.38-48, September, 1989.

[6] 황석형, 이용근, 양해술, "객체지향 프로그래밍에서의 Demeter 법칙의 정식화", 정보처리학회논문지, 제1권 제1호, pp. 63-72, 1994.

[7] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions of Software Engineering, Vol.20, No.6, pp.476-493, 1994.

[8] Mark Lorenz and Jeff Kidd, "Object-Oriented Software Metrics," Prentice Hall, 1994.

[9] E. Weyuker, "Evaluating software complexity measures," IEEE Transactions of Software Engineering, Vol.14, No.9, pp.1357-1365, 1988.

[10] Victor R. Basil, Lionel C. Briand and WalCelio L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Transactions of Software Engineering, Vol.22, No.10, pp.751-761, 1996.

[11] Tom Mens and Michele Lanza, "A Graph-Based Metamodel for Object-Oriented Metrics," Electronic Notes in Theoretical Computer Science, Elsevier Science, Vol.72, No.2, 2002.

[12] Ralf Reißing, "Towards a Model for Object-Oriented Design Measurement," Proc. of 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering(QAOOSE 2001), 2001.

황 석 형

e-mail : shwang@sunmoon.ac.kr

1991년 강원대학교 전자계산학과(이학사)

1994년 일본 오사카대학교 대학원 정보공학과(공학석사)

1997년 일본 오사카대학교 대학원 정보공학과(공학박사)

1997년~현재 선문대학교 컴퓨터정보학부 부교수

2001년~현재 국방대학교 국방정보화사업관리과정 외래강사

2001년~현재 일본 OGIS-RI Co. LTD. Certified UML Engineer

관심분야 : 객체지향 소프트웨어 시스템의 재구성 및 재이용, UML, Design Pattern, Adaptive Programming 기법, Formal Method 등

양 해 술

e-mail : hsyang@office.hoseo.ac.kr

1975년 홍익대학교 전기공학과(학사)

1978년 성균관대학교 정보처리학과 정보처리전공(석사)

1991년 일본 오사카대학교 정보공학과 소프트웨어공학전공(공학박사)

1975년~1979년 육군중앙경리단 전자계산실 시스템분석장교

1986년~1987년 일본 오사카대학교 객원연구원

1980년~1995년 강원대학교 전자계산학과 교수

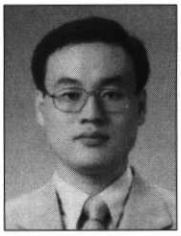
1995년~현재 한국소프트웨어 품질연구소(INSQ) 소장

1999년~현재 호서대학교 벤처전문대학원 교수

2000년~현재 한국정보처리학회 부회장

2003년~현재 미국 ACIS학회 Vice President

관심분야 : 소프트웨어공학(특히, S/W품질보증과 품질평가, 품질감리, 품질컨설팅, OOA/OOD/OOP, CASE, SI), 컴퍼넌트 기반 개발방법론, 전자상거래 기반기술



황 영 섭

e-mail : young@sunmoon.ac.kr

1989년 서울대학교 공과대학 컴퓨터공학과(학사)

1991년 포항공과대학교 컴퓨터공학과(공학석사)

1997년 포항공과대학교 컴퓨터공학과(공학박사)

1997년~2002년 한국전자통신연구원 선임연구원

2002년~현재 선문대학교 컴퓨터정보학부 전임강사

관심분야 : 바이오인포매틱스, 문서인식, 영상처리, 패턴인식 등