객체지향 어플리케이션의 확장을 위한 클래스 계층 구조의 재구성에 대한 정형기법

황 석 형[†] · 김 대 원[††] · 양 해 술[†††]

요 약

객체 지향 개념에 토대를 둔 어플리케이션 개발에는 여러 가지 이점이 있으나, 그 중 하나로서, 기존 성과물의 재이용을 들 수 있다. 유효한 재이용 수법 중의 하나로서, 본 논문에서는 클래스 계층 구조를 바탕으로 하는 확장 재구성법에 대해 논한다. 우선, 클래스 계층 구조를 보다 형식적인 형태로 정의하고 이론적인 논술이 가능토록 하기 위해서, 클래스 계층 구조를 유한 유방향 그래프 형태인 클래스 계층 그래프로 나타낸다. 또한, 클래스 계층 그래프간의 순서관계로써 객체 확장관계를 정의한다. 객체 확장관계를 만족하는 클래스 계층 그래프의 재구성법으로써, 다섯가지 기본조작을 정의하고, 정의된 기본 조작만을 이용하여 클래스계층 그래프를 확장 재구성할 수 있음을 보이기위해, 기본 조작의 정당성 및 완전성을 증명한다. 본 논문에서 제안한 객체 확장관계 및 기본 재구성기법은, 기존의 객체 지향 어플리케이션을 확장 재구성하고자 할 때 이론적인 토대로서 이용할 수 있다.

A Formal Approach for the Reorganization of Class Hierarchies for the Extension of Object Oriented Applications

Suk-Hyung Hwang[†] · Dae-Won Kim[††] · Hae-Sool Yang[†††]

ABSTRACT

There are some advantages of developing applications based on the object oriented concepts. One of them is that it is possible to reuse the existing designs and products.

This paper provides a formal method for the reorganization of class hierarchies for the object extension in the object oriented design phase. In this paper, we introduce a class hierarchy model called class hierarchy graph to describe class hierarchy structures using vertices to represent classes, and edges to represent the inheritance and aggregation relationship between classes. Based on the graph, we define an order relation(called the object extension) between class hierarchy graphs. And also we present a set of five basic transformations preserving the object extension relation. The set is proven to be correct and complete. The results of this paper help form a theoretical basis for the extension and reorganization of object-oriented application systems.

1. 서 론

객체 지향 설계에 있어서 기존 성과물의 재이용에 대한 장점으로서, 상위공정에서 기존의 하위공정의 성과물을 재이용할 수 있다는 점을 들 수 있다[1,2,14]. 객체 지향 설계 이외의 방법에서는 분석단계 및 설계단계에서의 성과물 재이용률이 상당히 낮은 반면에, 객체 지향 설계에서는 설계단위가 객체이고, 일관성이 있기 때문에 분석 및 설계단계를 포함하는 모든 단계

† 종신회원 : 선문대학교 컴퓨터정보학부 교수
†† 준 회 원 : 선문대학교 대학원 전자계산학과
††† 종신회원 : 한국소프트웨어품질연구소 소장
논문접수 : 1998년 6월 3일, 심사완료 : 1998년 11월 26일

에서 성과물 재이용이 가능하다[3-6,14]. 유효하고 재이용 가능한 설계단계의 성과물로써, 어플리케이션의 정적구조를 나타낸 클래스 계층구조가 있다. 이것을 그래프화한 것이 클래스 계층 그래프이다[7].

클래스 계층 그래프의 재구성에 관한 지금까지의 연구로서, 클래스 계층 그래프의 등가성에 대한 연구[8-11]가 있다. [8]의 연구에서는 어플리케이션의 클래스 계층 구조를 재구성할 때, 계층 구조만을 주목한 연구가 수행되었다. 그리고, [9-11]의 연구에서는 클래스 계층 그래프간의 동치관계(同値關係)로써, 객체부품등가관계(Object Component Equivalence Relation)를 정의하고, 서로 다른 집약구조를 갖는 클래스 계층 그래프간에서 동치류내에 속하는 것이 있으면, 같은 객체가 생성 가능하다는 사실이 밝혀졌다. 더욱이, 클래스 계층 그래프에 기본조작을 적용함으로써, 동치류(同値類)내에 속하는 어떠한 클래스 계층 그래프로도 변환 가능함을 보였다. 위 변환에서는 어플리케이션을 크게 변경하지 않고서도 클래스 계층 구조를 최적화할 수 있는 등의 장점이 있지만, 어플리케이션에 새로운 기능을 추가하는 등의 클래스 계층 구조를 확장할 필요가 있는 경우의 변환에는 대처할 수 없다.

본 논문에서는 클래스 계층 구조의 확장에 대응하는 클래스 계층 그래프의 확장 재구성에 대해 논한다. 우선, 클래스 계층 구조를 나타내기 위한 클래스 계층 그래프를 소개한다. 클래스 계층 그래프는 클래스 계층 구조를 유한 유방향 그래프로 나타낸 것이므로, 이에 대한 재구성에 대해서도 이론적인 논술이 가능하다. 클래스 계층 구조를 클래스 계층 그래프로 표현하여 올바르게 객체가 정의되었음을 보증하기 위해, 클래스 계층 그래프가 만족해야만 되는 몇 가지 조건을 제시한다. 또한, 클래스 계층 그래프의 순서관계로써, 객체 확장관계(Object Extension Relation)를 정의한다. 위 순서관계에 있는 두 개의 클래스 계층 그래프에서는, 한쪽에서 생성 가능한 객체는 다른 쪽에서 생성 가능한 객체에 포함되지만, 그 역은 반드시 성립하지는 않는다. 객체부품등가관계에 있는 임의의 클래스 계층 그래프로부터 생성되는 객체는 같으므로, 임의의 두 개의 클래스 계층 그래프가 객체 확장관계에 있을 때, 각각의 클래스 계층 그래프와 객체부품등가관계에 있는 클래스 계층 그래프에서도 같은 객체 확장관계가 성립한다고 할 수 있다. 또한, 클래스 계층 그래프를 재구성함으로써 같은 순서집합에 속하는 보다 큰 클래스 계층 그래프를 얻는다는 사실을, 객체 확장한다고 정의하고, 객체 확장수법을 제안한다. 객체 확장에 필요한 기본조작으로써 다섯 종류의 기본적인 재구성 조작을 정의한다. 또한, 이들 기본 재구성 조작과 [9-11]의 기본조작만을 적용함으로써, 클래스 계층 그래프를 임의로 객체 확장할 수 있음을 보인다.

1.1 관련 연구와의 비교

참고문헌[13]에서는 클래스 계층 구조를 그래프 형태로 표현한 Class Dictionary Graph[1]를 이용하여, 그래프에서 생성되는 객체에 대한 확장에 관해 논하고 있다. 즉, Class Dictionary Graph의 구조 변경에 있어서, 변경 전 후의 Class Dictionary Graph간의 관계를, 객체등가(Object-equivalent), 약확장(Weakly-extend), 확장(Extend)관계로 구분하였다. 그리고, Class Dictionary Graph의 구조 변경을 위한 그래프간의 기본적인 관계로써, EQU, ADD, GEN, REQ, REN을 정의하였다. EQU 관계에 있는 두 그래프는 서로 다른 구조를 갖지만 같은 객체가 생성된다. ADD 관계에 있는 두 그래프는 기존의 그래프에 새로운 정점과 변이 추가된 관계이고, GEN 관계에 있는 두 그래프는 하나의 정점이 가지는 집약변의 경로가 변환 후의 그래프에서는 다른 경로로 전환된다. 그리고, REQ 관계에 있는 두 그래프는 기존의 그래프상의 임의의 두 정점사이에 집약변이 추가된다. 또한, REN 관계에 있는 두 그래프는 그래프의 구조는 같지만 정점과 라벨(Label)의 이름이 재명명 된다. 이러한 다섯 가지 기본적인 관계들을 사용하여, 앞서 정의한 객체등가, 약확장, 확장관계를 표현하였다.

즉, 참고문헌[13]에서는 객체등가, 약확장, 확장관계를 다음과 같이 표현하고 있다.

- 객체등가 = EQU
- 약확장 = $EQU \cdot GEN \cdot EQU \cdot ADD$
- 확장 = $EQU \cdot GEN \cdot EQU \cdot REQ \cdot ADD$

---

1) Class Dictionary Graph $\phi = (VC, VA, \Lambda; EC, EA, EI)$는 아래 조건을 만족하는 유방향 그래프이다.
1. $V = VC \cup VA$, $VC \cap VA = \emptyset$.
   VC는 구상정점(Construction vertex)의 집합.
   VA는 추상정점(Alternation vertex)의 집합.
2. Λ는 라벨의 집합.
3. $E = EA \cup EC \cup EI$, $EA = VA \times V$, $EC = V \times V \times \Lambda$, $EI = V \times VA$.
   EA는 계승관계, EC는 집약관계, EI는 계승관계의 대한 역관계를 나타낸다.

이와 같이 참고문헌[13]에서는 Class Dictionary Graph 간의 기본관계들 만으로써 확장 관계를 나타내었을 뿐, 구체적으로 어떠한 방법으로 클래스 계층 구조를 변환할 수 있는지에 대해서는 정형화시키지 못했다.

본 논문에서는 클래스 계층 구조를 클래스 계층 그래프로 정의하고, 클래스 계층 그래프간의 순서관계로써 객체확장관계를 정의한다. 또한, 객체를 확장을 하기 위한 방법으로 다섯 가지 기본확장 재구성 조작을 새로이 정의하였다. 그리고, 정의된 기본 확장 재구성 조작에 대하여 완전한 알고리즘으로 정형화하고, 본 논문의 객체 확장 관계와 관련된 정당성과 완전성을 증명함으로써, 각종 어플리케이션 분야에 즉각적으로 적용할 수 있도록 하였다.

2. 클래스 계층 구조

객체 지향 설계에서는, 객체의 형(type)을 나타내는 클래스가 정의된다. 각 클래스에는 데이터 구조를 나타내는 데이터 멤버와 동작을 나타내는 멤버 함수가 속성으로써 정의되어, 해당 클래스를 특징 지우는 요소가 된다. 이를 바탕으로 각 클래스간의 관계를 정의함으로써 해당 어플리케이션의 구조를 정의한다. 클래스간의 관계에는 집약관계와 계승관계가 있다. 어떤 클래스 A의 데이터 멤버의 형(type)으로써 다른 클래스 B가 정의되었다면, A와 B 클래스간에는 집약관계가 있다고 한다. 이때, 클래스 B는 클래스 A의 부품클래스라고 하고, 클래스 A는 클래스 B의 집약클래스라고 한다. 집약클래스 중에는 몇 개의 부품클래스를 모아서 정리하기 위한 역할을 하는 클래스가 있고, 이런 클래스를 랩퍼(Wrapper) 클래스라고 한다. 따라서, 랩퍼 클래스로부터는 인스턴스를 생성할 수 없다. 또한, 어떤 클래스 D가 다른 클래스 C를 계승한 경우, 양 클래스간에는 계승관계가 있다고 한다. 이때, 클래스 C는 클래스 D의 부모클래스라고 하고, 클래스 D는 클래스 C의 자녀클래스라고 한다. 이때, 자녀클래스는 계승에 의해서 부모클래스의 속성을 이어받아 자신의 속성으로 갖게 된다. 또한, 부모클래스를 포함하여, 어떤 클래스가 계승하는 모든 클래스를 그 클래스의 선조클래스라고 하고, 반대로 자녀클래스를 포함하여, 어떤 클래스가 계승되는 모든 클래스를 그 클래스의 자손클래스라고 한다. 각 클래스는 객체 생성 가능한(인스턴스를 갖는) 클래스와 객체생성이 불가능한(인스턴스를 갖지 않는) 클래스로 나눌 수 있다. 객체 생성 가능한 클래스를 구상클래스라고 하고, 객체생성이 불가능한 클래스를 추상클래스라고 한다.

이와 같은 전제하에, 특정한 프로그래밍언어에 의존하지 않고, 어플리케이션의 정적구조를 나타내는 방법으로써 객체도(Object Diagram)[2,3,6,14], Class Dictionary Graph[4,7,8,12,13] 등을 이용하고 있다. 이 절에서는 [7,12]에서 정의된 클래스 계층 구조(Class Dictionary Graph)에 대해 살펴보고, 이를 수정·보완한 클래스 계층 그래프(CHG : Class Hierarchy Graph)를 소개한다. 또한, Class Dictionary Graph와 본 논문의 클래스 계층 그래프의 차이점에 대해서도 논의해 본다.

2.1 Class Dictionary Graph의 소개 [12]

본 논문에서 논의하고 있는 클래스 계층 그래프는 Class Dictionary Graph를 토대로 몇 가지 제한사항을 두고 클래스 계층 구조를 표현하고 있다. 이 절에서는 Class Dictionary Graph에 대해서 살펴보고 본 논문의 클래스 계층 그래프와의 차이점을 2.3절에서 설명한다.

참고문헌[7,12]에 정의된 Class Dictionary Graph는 클래스 계층 구조에 대해, 클래스를 정점으로, 그리고, 클래스간의 관계를 변으로 표현한 유방향 그래프(Direct Graph)이다. 정점집합 V는 구상정점의 집합 VC와 추상정점의 집합 VA로 구성되고, 각각 구상 클래스(Concrete Class)와 추상 클래스(Abstract Class)를 나타낸다. 변의 집합 E는 추상변의 집합 EA와 구상변의 집합 EC, 그리고 계승변의 집합 EI로 구성되며, 각각 계승관계, 집약관계, 계승관계에 대한 역관계를 나타내고 있다.

Class Dictionary Graph를 그림으로 나타낼 때는 구상정점(구상클래스)을 사각형(□), 추상정점(추상클래스)을 육각형(⬡)으로 나타낸다. 또한, 정점간의 관계를 나타낼 때, 구상변(EC)은 라벨(l)이 붙은 화살표($\xrightarrow{l}$)로, 추상변(EA)은 이중화살표 ($\Rightarrow$)로, 계승변(EI)은 점선으로 된 이중화살표($\dashrightarrow$)로 나타낸다.

(그림 1)에서는 class dictionary graph의 예를 보여주고 있다.

즉, (그림 1)은 자동차에 대한 클래스 계층 구조를 Class Dictionary Graph 형태로 표현한 것으로, 그래프의 각 구성요소는 다음과 같다.

VC = { 차, 디젤엔진, 점화플로그, 일반엔진, 터보엔진 }

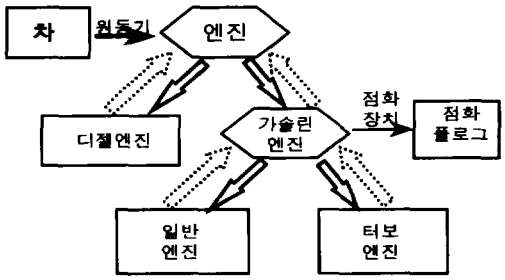
VA = { 엔진, 가솔린엔진 }

EA = { (엔진, 디젤엔진), (엔진, 가솔린엔진), (가솔린엔진, 일반엔진), (가솔린엔진, 터보엔진) }

EC = { (차, 엔진, 원동기), (가솔린엔진, 점화플로그, 점화장치) }

EI = { (디젤엔진, 엔진), (가솔린엔진, 엔진), (일반엔진, 가솔린엔진), (터보엔진, 가솔린엔진) }

Λ = { 원동기, 점화장치 }



(그림 1) Class Dictionary Graph의 예
(Fig. 1) A Example of Class Dictionary Graph

위와 같은 Class Dictionary Graph를 토대로 본 논문에서 도입한 클래스계층 그래프를 정의한다.

2.2 클래스 계층 그래프의 정의

정의 1 (클래스 계층 그래프)

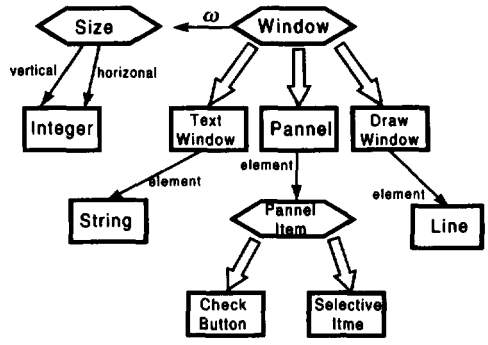
다음의 4가지 조건을 만족하는 (유한)유방향 그래프 G=(V, E, Λ)를 클래스 계층 그래프라고 한다.

1. $V = VA \cup VC,\ VA \cap VC = \varnothing$
2. $E = EA \cup EI,\ EA \cap EI = \varnothing$
3. $EI \subseteq V \times V$
   $EA \subseteq V \times V \times \Lambda$
   $\Lambda = \Phi \cup \Omega,\ \Phi \cap \Omega = \varnothing$
4. $\nexists v \in V[\exists l \in \Lambda[(v, x, l), (v, y, l) \in EA \wedge x \neq y]]$

VA는 추상정점들의 집합(Abstract Vertex Set)이며, 추상정점은 추상클래스를 나타내는 정점이다. VC는 구상정점들의 집합(Concrete Vertex Set)을 나타내며, 구상정점은 구상클래스를 나타내는 정점이다.

위의 정의 1에서, 조건 1은 V가 추상정점들의 집합과 구상정점들의 집합의 합집합이며, 추상정점들의 집합과 구상정점들의 집합은 서로소임을 의미한다. EA는 집약관계를 나타내는 변들의 집합이며, EI는 클래스간의 계승관계를 나타내는 변들의 집합이다. 조건 2는 E가 집약변 집합과 계승변 집합의 합집합이고, 집약변 집합과 계승변 집합은 서로소임을 의미한다. 조건 3은, 계승변은 정점과 정점의 카테시안 곱(Cartesian Product)으로 표현되고, 집약변은 정점과 정점, 그리고 라벨의 카테시안 곱으로 표현됨을 의미한다. 또한, Λ는 집약변의 라벨집합이며, Λ는 데이터 멤버의 이름을 나타내는 집합 Φ와 랩퍼(wrapper)의 라벨집합 Ω의 합집합이고, 이들은 서로소이다. 조건 4는 각 정점으로부터 나오는 집약변의 라벨이 모두 상이함을 의미한다.

클래스 계층 그래프를 그림으로 나타내는 경우의 표시방법은 Class Dictionary Graph와 유사하다. 즉, 추상정점은 육각형, 구상정점은 사각형으로 각각 나타낸다. 계승변은 이중화살표, 집약변은 보통 화살표로 나타내고, 특히, 집약변에는 라벨이 붙어있다. (그림 2)에 클래스 계층 그래프의 예를 보인다.



(그림 2) 클래스 계층그래프의 예(Window System)
(Fig. 2) A Class Hierarchy Graph of *a Window System*

(그림 2)의 예에서는,

VC={TextWindow, Pannel, DrawWindow, Integer, String, Line, CheckButton, SelectiveItem}

VA={Window, Size, PannelItem}

EA={(Window, Size, ω), (Size, Integer, vertical), (Size, Integer, horzontal), (TextWindow, String, element), (Pannel, PannelItem, element), (DrawWindow, Line, element)}

EI={(Window, TextWindow), (Window, Pannel), (Window, DrawWindow), (PannelItem, CheckButton), (PannelItem, SelectiveItem)}

$\Lambda = \Phi \cup \Omega$ = {ω}∪{ horzontal, vertical, element} = {ω, horzontal, vertical, element}

2.3 클래스 계층 그래프와 Class Dictionary Graph의 차이점

본 논문의 클래스 계층 그래프는 Class Dictionary Graph[7,12]를 토대로 몇 가지 제한 사항을 추가하여 재정의 되었다. Class Dictionary Graph에서는 객체를 생성하는 구상 클래스, 객체를 생산하지 못하는 추상 클래스만으로 그래프의 각 정점들을 표현하였다. 한편, 클래스 계층 그래프에서는 여러 개의 부품클래스들을 구성요소로 갖는 집약 클래스에 대하여 부품 요소들 중에 일부를 그룹화 하여 개념상으로 정리하고자 할 때, 랩퍼(Wrapper) 역할을 하는 클래스를 표현하기 위하여 집약변 위에 랩퍼 라벨($\omega \in \Omega$)을 추가하였다. 또한, Class Dictionary Graph에서의 변의 집합은 추상변(Alternation edges), 구상변(Construction edges), 계승변(Inheritance edges)으로 정의하고 있다. 추상변은 클래스간의 계승관계를, 구상변은 클래스간의 집약관계를 나타내고 있으며, 계승변은 계승관계에 대한 역관계를 표현하고 있다. 그러나, 본 논문의 클래스 계층 그래프에서의 변의 집합은 집약관계를 나타내는 집약변과 계승관계를 나타내는 계승변으로 단순화하여 정의하였다.

Class Dictionary Graph와 클래스 계층 그래프에는 몇 가지 제약 조건들이 있다. 우선 Class Dictionary Graph에서는 어떠한 클래스도 자기자신을 계승할 수 없고, 하나의 클래스에서 갖는 부품 및 계승하는 부품명의 이름은 일의적(Unique)이어야 한다는 조건들을 만족하는 정당한 그래프만을 논의의 대상으로 하고 있다[12]. 이에 대하여, 본 논문에서는 주어진 클래스 계층 그래프 중에 그 형태와 의미가 명확하고 올바른 것만을 논의의 대상으로 하기 위해서, 클래스 계층 그래프에 대하여 위의 두 조건과 더불어, 어떠한 클래스도 자기자신을 부품으로 가질 수 없고, 구상정점으로부터 추상정점을 계승하는 경우가 없다는 두 가지 조건을 추가하여 정당한 클래스 계층 그래프를 2.5절에 정의하였다.

위에서 언급한 각각의 제약 조건을 살펴보면, 자기자신을 계승하는 클래스를 정의할 경우에는 객체 생성시 무한하게 계승되어 객체 생성이 불가능할 수 있다. 그리고, 임의의 클래스가 갖는 부품 및 계승하는 부품의 이름이 일의적이여야 한다는 의미는 하나의 클래스가 동일한 부품명을 가질 경우에, 클래스는 어떠한 클래스를 부품으로 가질지 판단할 수 없게 된다. 또한, 자기자신을 부품으로 클래스를 정의할 경우에는 객체 생성시 무한한 부품클래스를 갖게 되어 객체 생성이 불가능하게 될 수 있다. 그리고, 추상클래스는 복수개의 구상클래스가 공통의 데이터 멤버를 갖는 경우에 부모클래스로 설정되어, 그곳에 공통의 데이터 멤버를 일반화시켜서 계승기능을 이용한 재사용 및 다형성 구현에 사용된다. 그러나, 구상클래스부터 추상클래스를 계승하는 경우, 추상클래스의 최하위 클래스가 구상클래스가 아닌 경우 실행시 객체 생성을 보장할 수 없게 된다. 따라서, 구상클래스로부터 추상클래스를 계승하는 형태는 제외시킨다.

본 논문에서는 이와 같은 네 가지 제약조건을 만족하는 클래스 계층 그래프만을 논의의 대상으로 한다. 그러나, 앞서 설명한 네 가지 제약 조건에 맞는 클래스 계층 그래프라 하더라도 불필요한 것을 배제하고 보다 간결하고, 명확한 클래스 계층 그래프를 얻기 위하여 하나의 클래스가 중복계승 되는 것을 금지한다. 또한, 모든 자손 클래스가 공통되는 부품을 갖는다면, 이 공통되는 부품을 선조 클래스의 부품으로 일반화한다. 이러한 경우를 만족하지 않는 클래스 계층 그래프에 대하여 특별한 언급이 없는 경우, 또는 재구성시 경우에 따라 일시적으로 발생할 수밖에 없는 경우를 제외하고는 본 논문에서 일절 취급하지 않는다.

2.4 클래스 계층 그래프의 도달 가능성

클래스 계층 그래프에서는 정점사이의 변이 클래스간의 관계를 나타내므로, 정점간의 도달 가능성을 파악함으로써 클래스간의 관계를 알 수 있다. 본 절에서는 클래스 계층 그래프에 있어서의 두 종류의 도달 가능성을 클래스간의 관계와 관련시켜서 논의한다.

2.4.1 계승 도달 가능

클래스 계층 그래프에 있어서, 정규표현 EI^*로 나타낼 수 있는 경로를 계승경로(Inheritance Path)라고 한다. 정점 u, v간에 계승경로가 존재할 때, $u \overset{*}{\Longrightarrow} v$로 나타내고 u는 v까지 계승 도달 가능하다라고 한다. 또한, 한 개 이상의 정점을 경유해서 u로부터 v에 계승 도달 가능하다는 것을 $u \overset{+}{\Longrightarrow} v$로 표현한다. u가 v에 계승 도달 가능할 때, 계승기능에 의해 v는 u에 정의된 속성을 자기자신의 속성으로 갖게 된다.

정의 2 (계승 도달 가능)
두 정점 u, $v \in V$에 대해서,

$$u \overset{*}{\Longrightarrow} v \Leftrightarrow \exists u_1, u_2, \ldots, u_n \in VA\,[\forall i{:}1 \le i \le n-1[(u_i, u_{i+1}) \in EI]]$$

or $u \Rightarrow v \in EI$

or $u = v$.

또한, 서로 다른 두 정점 u, v∈V에 대해서,

$u \overset{+}{\Longrightarrow} v \Leftrightarrow \exists u_1, u_2, ..., u_n \in V[u \Rightarrow u_1 \in EI, u_1 \Rightarrow u_2 \in EI, ..., u_n \Rightarrow v \in EI]$

or $[u \Rightarrow v \in EI]$

2.4.2 집약 도달 가능

클래스 u에 정의된 데이터 멤버의 형(type)이 클래스 v인 경우, 클래스 v는 클래스 u의 부품클래스라고 부른다. 이것을 클래스 계층 그래프로 치환하면, 정점 u로부터 집약변을 통해서 정점 v에 도달 가능하게 된다. 이것을 u로부터 v에 집약도달 가능하다라고 부른다. u가 v에 집약도달 가능하다는 것은 아래와 같은 경우에 해당한다.

① 집약변(u, v, l)이 클래스 계층 그래프에 존재하는 경우,
② u에 계승 도달 가능한 정점(즉, u의 선조 클래스) w에, 집약변(w, v, l)이 존재하는 경우,
③ v에 계승 도달 가능한 정점(즉, v의 선조 클래스) x에, 집약변(u, x, l)이 존재하는 경우,
④ u에 계승 도달 가능한 정점 w와 v에 계승 도달 가능한 정점 x에, 집약변(w, x, l)이 존재하는 경우

또한, 위 경우들은 모두 1단계의 집약관계이고, 다단계의 집약관계에 대해서도 마찬가지로 집약도달 가능하다고 부른다.

정의 3 (집약도달 가능)

집약변(u, v, l)이 존재하는 경우, 또는 정점 u가 자신의 선조 클래스 w로부터 집약변(w, v, l)을 계승하는 경우, $u \xrightarrow{l} v$로 나타낸다. 즉,

$u \xrightarrow{l} v \Leftrightarrow \exists w \in V[w \overset{*}{\Longrightarrow} u \wedge (w, v, l) \in EA]$.

그리고, 정점 v에 계승 도달 가능한 정점 w에 있어서 $u \xrightarrow{l} w$인 경우, $u \underset{\sim}{\overset{l}{>}} v$로 나타낸다. 즉,

$u \underset{\sim}{\overset{l}{>}} v \Leftrightarrow \exists w \in V[u \underset{\sim}{\overset{l}{>}} w \wedge w \overset{*}{\Longrightarrow} v]$.

라벨의 계열 $\alpha = \langle l_1\ l_2\ ...\ l_{k-1} \rangle \in \Lambda^*$에 있어서, $u \underset{\sim}{\overset{\alpha}{>}} v$를 아래와 같이 정의하고, 이를 u는 v에 집약도달 가능하다라고 부른다.

$u \underset{\sim}{\overset{\alpha}{>}} v \Leftrightarrow \forall i: 0 \le i \le k-2[\exists x_i, x_{i+1} \in V, l_i \in \Lambda[x_i \underset{\sim}{\overset{l_i}{>}} x_{i+1}]]$

$\wedge\ x_{k-1} \xrightarrow{l_{k-1}} x_k$

$\wedge\ x_0 = u$

$\wedge\ x_k = v$

또한, 라벨에 관심이 없는 경우에는 u는 v에 집약도달 가능하다는 것을 $u \underset{\sim}{\overset{+}{>}} v$로 나타낸다.

정점 u가 정점 v에 집약도달 가능하고, $u \underset{\sim}{\overset{\alpha}{>}} v$로 표현될 때, v는 u의 부품(Part)이라고 부르며, α는 부품의 이름이라고 한다.

(그림 2)의 클래스 계층 그래프에서는, 계승 도달 가능의 한가지 예로서, Window $\Longrightarrow$ TextWindow와 PannelItem $\overset{*}{\Longrightarrow}$ SelectiveItem을 들 수 있다. 집약도달 가능의 예로서는 DrawWindow $\underset{\sim}{\overset{\langle \omega\ \text{vertical} \rangle}{>}}$ Integer와 Pannel $\underset{\sim}{\overset{\langle \text{element} \rangle}{>}}$ CheckButton을 들 수 있다.

2.5 정당한 클래스 계층 그래프

앞절에서는 클래스 계층 그래프를 도입했으나, 클래스 계층 그래프로부터 올바르게 객체를 생성할 수 있다는 것을 보증하기 위해 본 논문에서 취급하는 클래스 계층 그래프의 형태를 한정하기로 한다.

이하, 본 논문에서 다루는 클래스 계층 그래프는 다음의 네 가지 조건을 만족하는 것으로 가정한다. 단, 조건 1과 3은 참고문헌[12]의 legal Class Dictionary Graph에 관련된 공리(Axiom)를 채용하였다.

1. Cycle-free inheritance condition [12]

$\nexists v \in V_G[v \overset{+}{\Longrightarrow} v]$

2. Cycle-free aggregation condition

$\nexists v \in V_G\left[v \underset{\sim}{\overset{+}{>}} v\right]$

3. Unique labels condition [12]

$\forall v, w, w', x, y \in V, \alpha \in \Lambda^*$

$\left[(w \overset{*}{\Longrightarrow} v) \wedge (w' \overset{*}{\Longrightarrow} v) \wedge (w \underset{\sim}{\overset{\alpha}{>}} x) \wedge (w' \underset{\sim}{\overset{\alpha}{>}} y) \Rightarrow (w = w')\right]$

4. Inheritance condition

$\nexists v \in VC, w \in VA[v \overset{+}{\Longrightarrow} w]$

조건 1(Cycle-free inheritance condition)은 자기자

신을 계승하는 정점이 존재하지 않음을 나타내고 있다. 자기자신을 계승하는 정점은 자기자신을 계승하는 클래스를 정의했다는 의미이다. 이와 같은 클래스 정의는 객체 생성시 무한하게 계승되어, 객체생성이 불가능하게 될 가능성이 있다.

조건 2(Cycle-free aggregation condition)는 자기자신과 집약관계에 있는 정점이 존재하지 않음을 의미한다 자기자신과 집약관계에 있는 정점은 자기자신을 데이터 멤버로써 갖는 클래스를 정의한 것이 된다. 이와 같은 클래스 정의도 객체 생성시 무한한 부품클래스를 갖게 되어, 객체생성이 불가능하게 될 가능성이 있다.

조건 3(Unique labels condition)은 한 개의 클래스가 갖는 부품명, 또는 선조 클래스로부터 계승하는 부품명으로부터 그 부품의 형(type)이 일의적(Unique)으로 결정되지 않으면 안된다는 의미를 갖는다.

조건 4(Inheritance condition)는 추상정점을 구상정점으로부터 계승하는 경우는 없음을 나타내고 있다. 추상클래스는 본래, 복수개의 구상클래스가 공통의 데이터 멤버를 갖는 경우에 부모 클래스가 되는 추상클래스를 설정하여, 그곳에 공통되는 데이터 멤버를 모아서 정리해 둠으로써 클래스 계층구조를 명확하게 하기 위한 목적으로 사용되고 있으므로, 추상클래스가 구상클래스로부터 계승하는 것은 의미가 없다.

위의 4가지 조건을 만족하는 경우, 클래스 계층 그래프는 정당하다고 부른다. 본 논문에서는 이 이후로는 정당한 클래스 계층 그래프만을 취급하기로 한다.

3. 객체 확장

3.1 객체 확장에 관한 정의들

여기서는, 객체 확장을 정의하는데 필요한 여러 가지 제반사항에 대해 정의를 한다. CS(v) (*Concrete subclassSet*)는 정점 v로부터 계승 도달 가능한 모든 구상정점의 집합을 나타낸다. 정점 v가 다른 정점 u의 부품클래스로써 정의된 경우, u의 부품의 형(type)은 $CS(v)$에 포함된 정점 중에 어느 하나가 된다.

정의 4 (Concrete Subclass set)

$$CS(v) \equiv \{ w \in VC \mid v \stackrel{*}{\Longrightarrow} w \}$$

정점 v가 가질 수 있는 부품 전체의 집합으로서 WP(v)(*Whole parts*)를 정의한다. WP(v)는 정점 v(v가 추상정점인 경우에는 v의 자손 클래스중의 구상정점)으로부터 인스턴스가 생성될 때에 인스턴스가 반드시 가지는 부품의 이름과, 그 부품이 가질 수 있는 형(type)을 나타내는 클래스 전체의 집합 등의 두 요소를 쌍(Pair)으로 구성시킨 집합이다.

정의 5 (Whole Parts set)

$$WP(v) \equiv \left\{ (\bar{\alpha} \in \Lambda^{+}, CS(w)) \mid v \underset{\sim}{\overset{\alpha}{>}} w \right\}$$

여기서 $\bar{\alpha}$는 라벨의 계열 α로부터 Ω의 성분을 제외시킨 것이다. 따라서, 랩퍼클래스에 할당된 라벨이 WP로부터 배제된다. 랩퍼(Wrapper)는 몇 개의 부품을 집약하여 정리해 둘 경우에만 사용되는 것이므로 객체생성은 불가능하며, 객체 생성시에는 존재하지 않는 것으로 생각한다.

3.2 객체 확장

본 논문에서 확장된 어플리케이션이란, 기존의 어플리케이션의 기능을 완전히 포함하고, 또한 새로운 기능이 추가된 어플리케이션이라는 정의에 토대를 두고, 이에 관련된 확장 재구성을 생각한다. 따라서 확장된 어플리케이션의 정적구조는 기존의 정적구조를 완전히 포함하지 않으면 안된다. 본 논문에서는 어플리케이션의 정적구조를 나타낸 것으로 클래스 계층 그래프를 다루고 있으므로, 클래스 계층 그래프가 다른 클래스 계층 그래프를 완전히 포함한다라는 사실이 무엇인가를 생각해본다.

우선, 맨 처음 생각할 수 있는 것으로는, 클래스 계층 그래프 G_1이 클래스 계층 그래프 G_2의 부분 그래프인 경우에, G_2가 G_1을 포함한다고 말할 수 있다. 이런 경우, G_1의 구조가 G_2에 보존되어 있으므로, G_2가 G_1을 "완전히 포함한다"라는 관계가 성립하는 것은 명백하다. 그러나, 이와 같은 경우만이 전부라고는 말할 수 없다. 클래스 계층 그래프는 생성 가능한 객체의 형태(즉, 객체의 클래스와 객체간의 관계)에 관한 모든 패턴을 기술한 것이라고 할 수 있다. 이와 같은 클래스 계층 그래프의 기능을 생각하면, "완전히 포함한다"라는 관계가 무엇을 의미하는지 알 수 있다. G_1에서 생성 가능한 객체가 G_2에서도 생성가능하고, 또한 G_1에서 생성 가능한 객체간의 관계가 G_2에서도 보존된다.

이와 같은 조건을 만족할 때, G_2는 G_1을 완전히

포함한다라고 말할 수 있다. 이때, G_1과 G_2는 객체 확장관계가 있다고 부른다. 생성된 객체에 나타나는 관계는 집약관계뿐이다. 계승관계는 객체가 생성된 뒤에는 보이지 않게 된다. 이상에서 논의된 내용을 정리하여 클래스 계층 그래프 G_1, G_2에 대해서 아래와 같은 두 가지 조건을 생각한다.

- G_1의 임의의 구상정점이 나타내는 클래스로부터 생성된 인스턴스를 생각할 때, G_2로부터도 같은 클래스의 인스턴스를 생성할 수 있다.
- G_1으로부터 생성된 인스턴스가 갖는 모든 부품을 G_2로부터 생성된 인스턴스도 반드시 갖는다.

위의 2가지 조건을 만족할 때, G_2는 G_1으로부터 객체 확장되었다고 말하고, $G_1 \leq G_2$로 나타낸다. 첫 번째 조건은 G_1의 모든 구상정점이 G_2에도 존재함을 의미하고, 두 번째 조건은 G_1의 임의의 구상정점 v에 대해, v의 모든 부품에 대해 G_2에도 같은 이름을 갖는 v의 부품이 존재하며, G_1에서 그 부품이 취할 수 있는 형(클래스)이 G_2에서도 존재함을 의미한다. 이상에서 논의한 내용을 토대로 하여 객체 확장 관계를 정의한다.

정의 6 (객체 확장)

$$G_1 \leq G_2 \Leftrightarrow \begin{cases} VC_{G_1} \subseteq VC_{G_2} \\ \forall v \in VC_{G_1}[\forall(\alpha, \beta_1) \in WP_{G_1}(v) \\ \quad [\exists(\alpha, \beta_2) WP_{G_2}(v): \beta_1 \subseteq \beta_2]] \end{cases}$$

$G_1 \leq G_2$인 경우, G_1과 G_2는 객체 확장관계에 있다라고 한다.

4. 기본 확장 재구성 조작

제3절에서 정의한 객체 확장 관계를 만족하는 클래스 계층 그래프의 도출에 대해 생각해 보자. 이때 적용하는 조작으로써, 다섯 종류의 기본 확장 재구성 조작을 정의한다.

4.1 정점의 추가 (ADV)

정의 7 (ADV)

기본 확장 재구성 조작 ADV는 주어진 클래스 계층 그래프에 새로운 정점 v를 추가하는 조작이다.

ADV(v, G_1)
//입력 : CHG G_1, G_1에 추가할 정점 v
//출력 : G_1으로부터 재구성된 CHG G_2
  if v $\in VC$ then
    $VC_{G_2} \leftarrow VC_{G_1} \cup \{v\}$; $VA_{G_2} \leftarrow VA_{G_1}$;
    $EA_{G_2} \leftarrow EA_{G_1}$; $EI_{G_2} \leftarrow EI_{G_1}$;
  end_if
  else
    $VC_{G_2} \leftarrow VC_{G_1}$; $VA_{G_2} \leftarrow VA_{G_1} \cup \{v\}$;
    $EA_{G_2} \leftarrow EA_{G_1}$; $EI_{G_2} \leftarrow EI_{G_1}$;
  end_else
End_of_ADV

4.2 변의 추가(ADE)

주어진 클래스 계층 그래프의 두 정점간에 새로운 변을 추가하는 조작이다. 여기에는 두종류의 조작이 있다. 새로운 변으로서 집약변을 추가하는 조작과 계승변을 추가하는 조작이다. 추가할 변의 양쪽 또는 한쪽에 정점이 존재하지 않는 경우는 ADV를 적용한 후에 ADE를 적용한다.

정의 8 (ADE)

기본 확장 재구성 조작 **ADE**는 주어진 클래스 계층 그래프에 변을 추가하는 조작이며, 아래의 두 종류가 있다.

- 집약변을 추가하는 조작 : **ADE$_1$**
- 계승변을 추가하는 조작 : **ADE$_2$**

ADE$_1$((u, v, l), G_1)
//입력 : CHG G_1, G_1에 추가될 집약변 (u, v, l)
//출력 : G_1으로부터 재구성된 CHG G_2
    $V_{G_2} \leftarrow V_{G_1}$;
    $EA_{G_2} \leftarrow EA_{G_1} \cup \{(u, v, l)\}$;
    $EI_{G_2} \leftarrow EI_{G_1}$;
End_of_**ADE$_1$**

ADE$_2$((u, v), G_1)
//입력 : CHG G_1, G_1에 추가될 계승변 (u, v)

//출력 : G_1으로부터 재구성된 CHG G_2

$V_{G_2} \leftarrow V_{G_1}$;

$EA_{G_2} \leftarrow EA_{G_1}$;

$EI_{G_2} \leftarrow EI_{G_1} \cup \{(u, v)\}$;

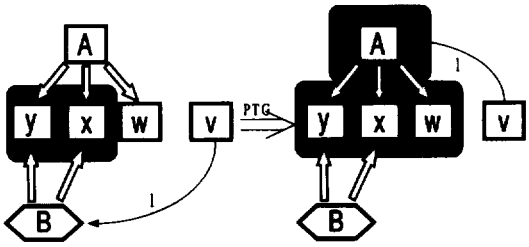
End_of_**ADE2**

4.3 부품일반화(PTG)

정의 9 (PTG)

기본 확장 재구성 조작 PTG는, $CS(B) \subseteq CS(A)$를 만족하는 두 정점A, B에 대해, 집약변(v, B, l)을 (v, A, l)로 변경하는 조작이다.

클래스 B로부터 계승 도달 가능한 모든 구상클래스는 클래스 A로부터도 계승 도달 가능하므로, 클래스 v가 가지고 있는 l이라는 이름의 데이터 멤버의 형이 줄어드는 경우는 없다. 또한, **PTG** 조작은 반드시 $A \stackrel{+}{\Rightarrow} B$임을 요구하지 않는다(그림 3 참조).



(그림 3) PTG의 예
(Fig. 3) An Example of PTG

PTG $((u, v, l), (u, v', l), G_1)$

//입력 : CHG G_1, 제거될 집약변 (u, v, l), 추가될 집약변 (u, v', l)

//출력 : G_1으로부터 재구성된 CHG G_2

$V_{G_2} \leftarrow V_{G_1}$;

$E_{G_2} \leftarrow E_{G_1}$;

if $(CS_{G_1}(v) \subseteq CS_{G_2}(v'))$ then

$EA_{G_1} \leftarrow EA_{G_2} - \{(u, v, l)\} \cup \{(u, v', l)\}$;

end_if

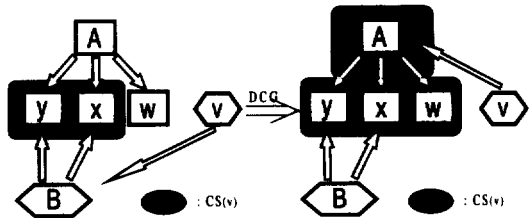
end_of_**PTG**

4.4 자손일반화 (DCG)

정의 10 (DCG)

기본 확장 재구성 조작 DCG는, $CS(B) \subseteq CS(A)$를 만족하는 두 정점 A, B에 대해서, 계승변(v, B)을 (v, A)로 변경하는 조작이다.

클래스 v로부터 클래스 B를 경유하여 계승 도달 가능한 모든 구상클래스는, DCG에의해 변경시킨 클래스 계층 그래프에서도 클래스 v로부터 클래스 A를 경유하여 계승 도달 가능하므로, CS(v)에 포함된 모든 구상클래스가 계승하는 클래스가 줄어드는 경우는 없다. 또한, 이 조작은 반드시 $A \stackrel{+}{\Rightarrow} B$ 임을 요구하지 않는다(그림 4 참조).



(그림 4) DCG의 예
(Fig. 4) An Example of DCG

DCG $((u, v), (u, v'), G_1)$

//입력 : CHG G_1, 제거될 계승변(u,v), 추가될 계승변(u,v')

//출력 : G_1으로부터 재구성된 CHG G_2

$V_{G_2} \leftarrow V_{G_1}$;

$E_{G_2} \leftarrow E_{G_1}$;

if $(CS_{G_1}(v) \subseteq CS_{G_2}(v'))$ then

$EA_{G_1} \leftarrow EA_{G_2} - (u, v) \cup (u, v')$;

end_if

end_of_**DCG**

5. 확장 재구성의 예

(그림 5)의 G_1과 G_2는 객체확장 관계에 있다. G_1에 대해 본 논문에서 정의한 확장 기법들(ADV, ADE_1, ADE_2, PTG, DCG)을 적용하여, G_2를 도출해 내는 과정을 보인다.

(그림 5)에서는 클래스 계층 그래프 G_1에 기본 확장 재구성 조작을 적용하여 객체 확장 관계에 있는 G_2를 얻는 각 과정을 나타내고 있다. 즉, 주어진 G_1

에 대하여 다음과 같은 순서에 의해 각각의 기본 확장 재구성 조작을 적용한다.

먼저 주어진 클래스 계층 그래프 G_1에 대해서 ADV를 다섯 번 적용하여 구상 정점(클래스)인 ToggleButton, Polygon, MouseInterface와 추상 정점인 Button, Figure가 추가된 그래프 G_1^5를 얻는다(Step 1~5). G_1^5에 ADE_1을 적용하여 추상 정점 Window와 구상 정점 MouseInterface사이에 집약변을 추가한 G_1^6를 얻는다(Step 6). 다시 그래프 G_1^6에 ADE_2를 적용하여 Button과 ToggleButton, Button과 CheckButton, Figure와 Line, Figure와 Polygon사이에 계승변을 추가한 G_1^{10}를 얻는다(Step 7~10). 또, G_1^{10}에서 CS(Line)={Line}이고, CS(Figure)= {Line, Polygon}이므로 PTG를 적용하여 집약변(DrawWindow, Line, element)을 (Draw Window, Figure, element)로 변경하여 G_1^{11}를 얻을 수 있다(Step 11). 이때, DrawWindow 클래스가 가지고 있는 데이터 멤버 element의 클래스는 PTG 적용 후에도 줄어들지 않는다. 그리고, G_1^{11}에서 CS(Button)={CheckButton, ToggleButton}이고, CS(PannelItem)={Check Button, SelectiveItem}이므로 DCG를 적용하여 두 정점 CheckButton과 Button에 대해서 계승변(PannelItem, CheckButton)을 (PannelItem, Button)으로 변경하므로써 G_1^{12}를 얻게된다(Step 12). 이때, 변경후의 그래프 G_1^{12}에서 CS(PannelItem)={CheckButton, ToggleButton, SelectiveItem}이므로 DCG를 적용한 후에도 CS(PannelItem)에 포함된 구상 클래스는 줄어들지 않는다. 즉, $CS_{G_1^{11}}$(PannelItem) $\subset$ $CS_{G_1^{12}}$(PannelItem)이다. 이와 같은 방법으로 G_1에 본 논문에서 제안한 기본 확장 재구성 조작만을 적용하여 $G_2(=G_1^{12})$를 얻을 수 있다.(그림 5 참조)

6. 증 명

클래스 계층 그래프 G_A에 어떤 기본조작을 적용하여 확장 재구성하여 얻어진 클래스 계층 그래프가 G_B 일 때, $G_A \leq G_B$의 관계에 있다면, 이때 사용된 기본조작은 정당하다고 한다. 또한, $G_A \leq G_B$ 관계에 있는 임의의 두 클래스 계층 그래프에 대해, G_A에 어떤 정해진 기본 조작만을 적용함으로써 확장 재구성하여 G_B를 얻을 수 있다면, 이때 사용된 기본조작은 완전하다고 한다. 여기에서는, 제4절에서 정의한 기본 확장 재구성 조작이 모두 정당하고, 완전하다는 사실을 증명한다.

6.1 기본 확장 재구성 조작의 정당성

6.1.1 ADV

클래스 계층 그래프 G_1에 ADV를 적용하여 G_2를 얻었다고 하자. 이 때, 정의 7에 의해 G_1의 요소는 줄어들지 않으므로,

$$\begin{cases} V_{G_1} \subset V_{G_2} \\ \forall v \in VC_{G_1}[\forall(\alpha, \beta_1) \in WP_{G_1}(v) \\ \qquad \exists(\alpha, \beta_2) WP_{G_2}(v): \beta_1 \subseteq \beta_2] \end{cases}$$

가 성립하고, 이것은 $G_1 \leq G_2$의 정의에 해당한다.

6.1.2 ADE

클래스 계층 그래프 G_1에 ADE를 적용하여 G_2를 얻었을 때, ADE의 정의8에 의해, G_1의 요소는 줄어들지 않으므로

$$\begin{cases} V_{G_1} = V_{G_2} \\ \forall v \in VC_{G_1}[\forall(\alpha, \beta_1) \in WP_{G_1}(v) \\ \qquad \exists(\alpha, \beta_2) WP_{G_2}(v): \beta_1 \subseteq \beta_2] \end{cases}$$

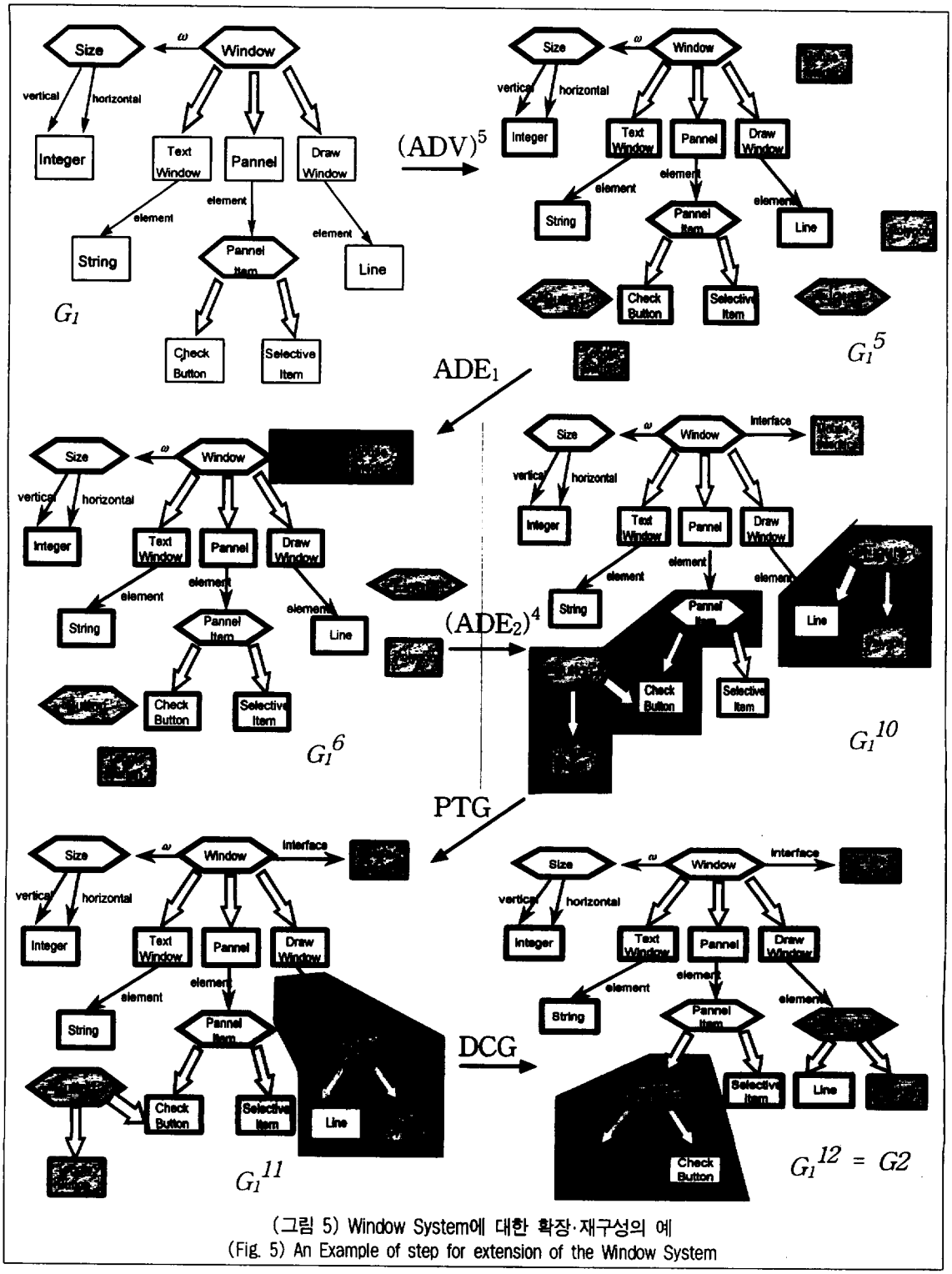
가 성립하고, 이것은 $G_1 \leq G_2$의 정의에 해당한다.

6.1.3 PTG

클래스 계층 그래프 G_1의 집약변(u, v, l)에 대해서 PTG를 적용하여 G_2를 얻었다고 하자. 이 때, 집약변 (u,v,l)이 (u,v',l)로 변경되었다고 하면, 정의 9에 의해, $CS_{G_1}(v) \subseteq CS_{G_2}(v')$이므로 정점 u의 l이라는 부품의 형을 나타내는 클래스가 줄어들지 않는다. 따라서,

$$\begin{cases} V_{G_1} = V_{G_2} \\ \forall v \in VC_{G_1}[\forall(\alpha, \beta_1) \in WP_{G_1}(v) \\ \qquad \exists(\alpha, \beta_2) WP_{G_2}(v): \beta_1 = \beta_2] \end{cases}$$

가 성립하고, 이것은 $G_1 \leq G_2$의 정의에 해당한다.



(그림 5) Window System에 대한 확장·재구성의 예
(Fig. 5) An Example of step for extension of the Window System

6.1.4 DCG

클래스 계층 그래프 G_1의 계승변(u, v)에 대해 DCG를 적용하여 G_2를 얻었다고 하자. 이 때, 계승변(u, v)이 (u, v')로 변경되었다고 하면, 정의 10에 의해 $(v) \subseteq CS_{G_2}(v')$이므로 정점 v의 자손클래스는 줄어들지 않는다. 또한, 정점 집합 $CS_{G_1}(v)$에 포함된 정점은 모두 u로부터 계승한다라는 성질을 보존하고 있다. 따라서,

$$\begin{cases} V_{G_1} = V_{G_2} \\ \forall v \in V C_{G_1}[\forall(\alpha, \beta_1) \in WP_{G_1}(v) \\ \qquad \exists(\alpha, \beta_2) WP_{G_2}(v): \beta_1 \subseteq \beta_2] \end{cases}$$

가 성립하고, 이것은 $G_1 \leq G_2$의 정의에 해당한다.

6.2 기본 확장 재구성 조작의 완전성

6.2.1 복잡한 수식의 약식표현

집합 A에 요소를 추가하거나 삭제하는 조작을 반복 실행해서 집합 B가 얻어졌을 때, A에 추가한 요소의 수를 $\Delta^+(A,B)$로, A로부터 제거한 요소의 수를 $\Delta^-(A,B)$ 로 각각 나타낸다.

$$\begin{cases} \Delta^+(A,B) \equiv |B-(A \cap B)| \\ \Delta^-(A,B) \equiv |A-(A \cap B)| \end{cases}$$

이하, 본 논문에서는 식을 간단히 표현하기 위해서 아래와 같은 약식기호를 사용하기로 한다.

$\Delta^+(V_{G_i}, V_{G_{i+1}}) \rightarrow \Delta^+(V)$
$\Delta^-(V_{G_i}, V_{G_{i+1}}) \rightarrow \Delta^-(V)$
$\Delta^+(EA_{G_i}, EA_{G_{i+1}}) \rightarrow \Delta^+(EA)$
$\Delta^-(EA_{G_i}, EA_{G_{i+1}}) \rightarrow \Delta^-(EA)$
$\Delta^+(EI_{G_i}, EI_{G_{i+1}}) \rightarrow \Delta^+(EI)$
$\Delta^-(EI_{G_i}, EI_{G_{i+1}}') \rightarrow \Delta^-(EI)$
$\Delta^+(V_{G_A}, V_{G_B}) + \Delta^+(EA_{G_A}, EA_{G_B}) + \Delta^+(EI_{G_A}, EI_{G_B}) \rightarrow \Delta^+(G_A, G_B)$

6.2.2 완전성

지금까지 $G_A \leq G_B$를 만족하는 어떠한 두 개의 클래스 계층 그래프에 대해서도 기본 조작수 k가 존재하고, 임의의 $i(1 \leq i \leq k-1)$에 대해 $\Delta^+(G_i, G'_{i+1})=1$인 G'_{i+1} 가 G_{i+1}의 부품등가변환에 의해 얻어진다. 이와 같은 관계에 있는 G_i, G'_{i+1}에 대해, 다음 식이 성립한다.

$$G'_{i+1} = G_i \circ (ADV \mid ADE_1 \mid ADE_2 \mid PTG \mid DCG)$$

<증명>

다음과 같이 세 가지 경우로 나누어 생각한다.

- $\Delta^+(V)=1$인 경우
- $\Delta^+(EA)=1$인 경우
- $\Delta^+(EI)=1$인 경우

1. $\Delta^+(V)=1$인 경우

이때, $\Delta^+(G_i, G'_{i+1})=1$로부터 $\Delta^+(EA) = \Delta^+(EI) = 0$이다. 따라서 보제 1로부터 $\Delta^-(EA) = \Delta^-(EI) = 0$이다. 또한 $G_i \leq G'_{i+1}$로부터 $V_{G_i} \subseteq V_{G'_{i+1}}$이므로 $\Delta^-(v) = 0$이다. 즉, G_i와 G'_{i+1}는 변의 집합이 같고, G_i의 정점집합에 정점을 한 개 추가하면 G'_{i+1}의 정점집합을 얻을 수 있다. 따라서, $G'_{i+1} = G_i \circ ADV$이다.

2. $\Delta^+(EA)=1$인 경우

이때, $\Delta^+(G_i, G'_{i+1})=1$로부터 $\Delta^+(v) = \Delta^+(EI) = 0$이다. 따라서,

$\Delta^-(v) = \Delta^-(EI) = 0$이다.

(a) $\Delta^-(EA)=0$인 경우

이때, G_i의 정점집합 및 계승변의 집합과 G'_{i+1}의 정점집합 및 계승변의 집합은 각각 같으며, G_i의 집약변의 집합에 집약변을 한 개 추가하여 G'_{i+1}의 집약변의 집합을 얻을 수 있다.
따라서, $G'_{i+1} = G_i \circ ADE_1$이다.

(b) $\Delta^-(EA)=1$인 경우

이때, G'_{i+1}에 포함되어 있으나 G_i에 포함되어 있지 않은 집약변이 (u, v, l)이라고 하면, $CS_{G_i}(v) \subseteq CS_{G'_{i+1}}(v')$인 정점 v'에 대해 집약변(u, v', l)이 G'_{i+1}에 포함된다. 따라서, $G'_{i+1} = G_i \circ PTG$이다.

3. $\Delta^+(EI)=1$인 경우

이때, $\Delta^+(G_i, G'_{i+1})=1$로부터 $\Delta^+(v) = \Delta^+(EA)$

$=0$이다. 따라서, $\triangle^{-}(v)=\triangle^{-}(EA)=0$이다.

(a) $\triangle^{-}(EI)=0$**인 경우**

이때, G_i의 정점집합 및 계승변의 집합과 G'_{i+1}의 정점집합 및 집약변의 집합은 각각 같으며, G_i의 계승변의 집합에 계승변을 한 개 추가하여 G'_{i+1}의 계승변의 집합을 얻을 수 있다. 따라서, $G'_{i+1}=G_i \circ ADE_2$

(b) $\triangle^{-}(EI)=1$**인 경우**

이때, G'_{i+1}에 포함되어 있으나 G_i에는 포함되어 있지 않은 계승변이 (u, v)이라고 하면, $CS_{G_i}(v) \subseteq CS_{G'_{i+1}}(v')$인 정점 v'에 대해 **계승변**(u, v')이 G'_{i+1}에 포함된다. 따라서, $G'_{i+1}=G_i \circ DCG$이다.

이상으로부터 k가 기본조작수인 경우, 아래 식이 성립한다.

$$\forall i: 1 \le i \le k-1 [\, G'_{i+1} = G_i \circ (ADV \mid ADE_1 \mid ADE_2 \mid PTG \mid DCG)]$$

또한, 각 $i(1 \le i \le k-1)$에 대해, G_{i+1}와 G'_{i+1}은 객체 부품등가관계가 있으므로 G'_{i+1}에 기본 등가 재구성 조작을 적용하여 G_{i+1}를 얻을 수 있다. [9-11]의 기본 등가 재구성 조작은 쓸모 없는 추상정점의 제거(DUA) 및 추가(AUA), 공통부분의 추상화(ACP) 및 분배(DCP), 부분적인 부품교환(PRP), 랩퍼의 추가(AOW) 및 제거(DOW)로 구성되어 있다. 이러한 조작들을 이용하여 나타내면,

$$\forall i: 1 \le i \le k-1 [\, G_{i+1} = G_i \circ (ADV \mid ADE_1 \mid ADE_2 \mid PTG \mid DCG \mid DUA \mid AUA \mid ACP \mid DCP \mid PRP \mid AOW \mid DOW)]$$

이다. 윗 식과 더불어 객체 확장관계의 추이성에 의해, 객체 확장관계에 있는 임의의 두 개 클래스 계층 그래프 $G_A \le G_B$에 대해, 기본 확장 재구성 조작과 기본 등가 재구성 조작을 적용함으로써 G_B를 얻을 수 있다. 즉,

$$G_B = G_A \circ (ADV \mid ADE_1 \mid ADE_2 \mid PTG \mid DCG \mid DUA \mid AUA \mid ACP \mid DCP \mid PRP \mid AOW \mid DOW)^{*}]$$

이다.

7. 결 론

본 논문에서는, 객체 지향 어플리케이션의 정적구조에 주목하여 클래스 계층 그래프간의 순서관계로서 객체 확장관계를 정의하였다. 또한 객체 확장관계를 보존하는 재구성 조작으로서 [9-11]의 기본 등가 재구성 조작을 도입하고, 이에 덧붙여 기본 확장 재구성 조작을 정의하였고, 이에 대한 정당성과 완전성에 대해 증명하였다.

객체 확장관계에 있는 두 개의 클래스 계층 그래프에서는, 한쪽 그래프에 정의된 객체를 다른 쪽 그래프가 완전히 포함한다는 사실을 보증할 수 있다. 따라서 기존에 정의해 둔 클래스와 클래스간의 관계를 보전하면서, 새로운 클래스와 클래스간의 관계를 추가하여 클래스 계층구조를 확장·재구성할 때 본 논문의 결과를 이론적인 토대로써 이용할 수 있다. 앞으로의 연구과제로는, 기본 조작 수를 최소화하기 위한 클래스 계층 구조 재구성의 최적화 문제 등이 있다.

현재, 본 논문의 결과를 이용하여 기존의 객체지향 어플리케이션의 클래스 계층 구조에 대한 등가변환 및 부품등가변환, 그리고 확장 재구성을 위한 자동화 툴을 개발하고 있으며, SGML 문서의 확장 및 재이용에 관한 연구에 응용할 예정이다.

참 고 문 헌

[1] Casais, E., "Managing Evolution in Object-Oriented Environments : An Algorithmic Approach," Ph.D. thesis, University of Geneva, Geneva, Switzerland, 1991.

[2] 本位田, 山城, "オブジェクト指向システム開發", 日經BP社, 1993.

[3] P. Coad and E.Youraon, Object-Oriented Design, Yourdon Press, 1991.

[4] Karl J. Lieberherr, Adaptive Object-Oriented Software, PWS Publishing Co, 1996.

[5] B. Liskov, "Data Abstraction and Hierarchy," OOPSLA'87 Addendum to the Proceeding, Oct. 1987.

[6] G. Booch, Object-Oriented Design with Applications, The Benjamin/Cummings Publishing, 1991.

[7] Lieberherr K, Xiao. C, "Formal Foundations for Object-Oriented Data Modeling," IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.3, pp.462-478, 1993.

[8] Bergstein, P.L., "Object Preserving Class Transformations," SIGPLAN Notices, Vol.26, No.11, pp.299-313, 1991.

[9] S.H. Hwang, "A Study of the Redesign Framework for Object-Oriented Design Models," Ph.D. thesis, Osaka University, Osaka, Japan, 1997.

[10] S.H. Hwang, Y. Tsujino, N. Tokura, "A Reorganization Framework for Object Oriented Class Hierarchies," Journal of Computer Software, Japan Society for Software Science and Technology, Vol.15, No.4, May 1998.(in Japanese)

[11] S. Hwang, Y. Tsujino and N. Tokura, "A Reorganization Framework of the Class Hierarchy Based on the Object Semi-Equivalence," IPSJ Report SE104-21, Mar. 1995.

[12] Lieberherr K, Xiao. C, "Object-Oriented Software Evolution," IEEE Transactions on Software Engineering, Vol.19, No.4, 1993.

[13] Lieberherr K, Hürsch W, Xiao C, "Object-extending Class Transformations," Formal Aspects of computing, the International Journal of Formal Methods,(6) : 391-416, 1993.

[14] James Rumbaugh, Object-Oriented Modeling and Design, Prentice Hall, 1991.

황 석 형

e-mai : shwang@omega.sunmoon.ac.kr

1991년 강원대학교 전자계산학과(이학사)

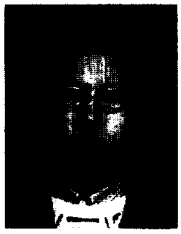
1991년~1992년 日本 오사카대학 기초공학부 연구생

1994년 日本 오사카대학 대학원 정보공학과(공학석사)

1997년 日本 오사카대학 대학원 정보공학과(공학박사)

1997년~현재 선문대학교 컴퓨터정보학부(전임강사)

관심분야 : 객체지향 시스템의 재사용기법, Adaptive Object-Oriented Software 개발 기법



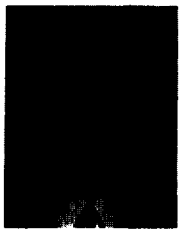
김 대 원

e-mai : terrius@omega.sunmoon.ac.kr

1993년~1997년 선문대학교 전자계산학과(학사)

1997년~현재 선문대학교 대학원 전자계산학과(석사과정)

관심분야 : 소프트웨어공학(재사용, 재공학), 객체지향 시스템, 멀티미디어 시스템



양 해 술

e-mai : insq@unitel.co.kr

1975년 홍익대학교 공과대학 전기공학과 졸업(학사)

1978년 성균관대학교 정보처리학과 정보처리 전공(석사)

1991년 日本 오사카대학교 기초공학 정보공학과 소프트웨어공학 전공(공학박사)

1975년~1979년 육군중앙경리단 전자계산실 시스템 분석 장교

1986년~1987년 日本 오사카대학교 객원연구원

1980년~1995년 강원대학교 전자계산학과 교수

1993년~1994년 한국정보과학회 학회지 편집부위원장

1994년~1995년 한국정보처리학회 논문지편집위원장

1994년~현재 한국산업표준원(KISI)이사

1995년~현재 한국소프트웨어품질 연구소(INSQ)소장

관심분야 : 소프트웨어공학(S/W 품질보증과 품질평가, 품질감리, 품질 컨설팅, OOA/OOD/OOP, CASE, SI), 소프트웨어 프로젝트 관리