프로그램 이해 지원과 재사용을 위한 객체 지향 클래스 라이브러리 설계 및 구현

정 계 동† · 권 오 진†† · 최 영 근†††

요 약

본 논문에서는 프로그램의 이해와 재사용에 촛점을 둔 객체 지향 클래스 라이브러리 설계 방법 및 객체를 효율적으로 재사용하여 프로그래밍 할 수 있도록 객체에 대한 정보 추출 방법을 제시한다. 프로그램의 재사용을 위한 부품을 모듈 단위로 생성하여 각 정보를 테이블에 저장하며, 모듈간에 참조할 수 있는 인터페이스 클래스를 추출한다. 프로그램의 이해를 쉽게 하기 위하여 프로그램 코드를 기반으로 하여 클래스 관계성을 그래프로 표현하고 노드 클래스를 아이콘화하여 볼 수 있도록 하였다. 각 모듈 안에서의 참조 관계, 상속 관계, 복합 관계를 추출 및 세부적인 다형성 관계, 프랜드 관계등의 추가적인 정보를 생성할 수 있다. 본 논문에서 제시하는 방법은 프로그램 개발 및 유지보수시에 프로그램의 이해력를 높여 재사용 시스템 구축을 용이하게 한다.

Design and Implementation of Object-Oriented Class Library for Supporting Understanding and Reusing the Programs

Kye-Dong Jung† · Oh-Jin Kwon†† · Young-Gun Choi†††

ABSTRACT

This paper presents the design method of object-oriented class library for understanding and reusing a program. Also this paper presents the information retrieval method for object which can be used for programming reusing objects. The reusable component is created and stored in table and these information enables the retrieval of interface class which can cross-reference each module. For understanding the program, the relationship of class is represented by graph and the node class is iconized for visibility. In the relationships of each module, inheritance relationships, and composite relationships are created and detailed polymorphism and friends relations can be created. Since this paper's method increases the understanding of program at design and maintenance, it makes the construct of reusable system easy.

1. 서 론

소프트웨어의 품질과 생산성을 높이기 위해 새로운 기법이 많이 연구되어왔는데 그 방법중 하나가 재사용 기법이다[5,8,12,13]. 이것은 기존의 소프트웨어를 이용하여 새로운 소프트웨어를 작성하기 위한 방법이다. 일반적으로 재사용에는 소프트웨어 설계와 프로그래밍 언어의 두 가지 측면으로 나뉘어진다. 즉, 문제의 해결책으로 응용 문제의 구조를 중요시하며 해결하려는 설계 방법을 응용 중심(application driven)이라고 하며 프로그래밍 기법을 위주로 설계하는 것을 해결책 중심(solution-driven)이라 한다. 그러나 두 가지 기법은

† 정 회 원 : 광운대학교 전자계산학과 박사과정
†† 정 회 원 : 산업기술정보원(KINITI) 연구원
††† 정 회 원 : 광운대학교 전자계산소 소장
논문접수 : 1997년 1월 24일, 심사완료 : 1998년 3월 19일

아주 밀접한 관계를 가진다[16]. 그리고 이러한 재사용 시스템은 소프트웨어 부품을 효율적으로 분류하고 이들을 컴퓨터에 저장하여 검색하는 시스템이며, 새로운 시스템을 개발하고 유지 보수 시간을 줄이기 위해 기존 시스템의 다양한 정보를 소프트웨어 부품으로 사용한다[3,14]. 그러나 이 부품들은 소프트웨어 코드, 설계, 분석, 요구 사양서이며 서로가 불일치 하는 경우가 많다[4,15].

본 논문에서는 객체 지향 클래스 라이브러리 시스템을 구축하여 분석 단계에서 생성된 클래스를 효율적으로 재사용하여 설계 및 프로그래밍 할 수 있도록 한다. 또한 재사용 컴퍼넌트를 위해 삽입 할 때 상속성과 참조관계를 고려하여 계층 구조를 쉽게 확장 할 수 있다. 유사성 정보 및 분해된 클래스 계층 정보를 기반으로 하여 클래스 삽입이 이루어진다. 이러한 시스템 요소들은 제안된 정보 저장소에 저장되어 재사용이 가능한 소프트웨어 요소들의 목록과 이들에 대한 이해 및 접근 방법을 용이하게 한다.

본 논문에서 제시하는 방법은 다음과 같다.

첫째, 객체 지향 클래스 라이브러리 시스템 설계 방법을 제시하며,

둘째, 설계 단계에서 클래스 계층 구조를 하나의 모듈 단위로 설정하여 클래스들의 관계성을 저장하고,

셋째, 클래스 노드의 삽입시, 상위 클래스로부터 하위 클래스간의 상속 및 참조 관계를 고려하여 클래스를 일관성 있게 확장 할 수 있도록 하며,

넷째, 모듈별로 구현된 각 정보를 윈도우상에서 동시에 볼 수 있도록 한다.

따라서 본 논문에서 제시하는 방법은 개발자에게 프로그램의 이해를 쉽게 하여 소프트웨어 재사용 및 유지보수를 용이하게 하도록 한다.

본 논문에서의 객체 지향 클래스 라이브러리 시스템 구현은 마이크로소프트사에서 제공되는 데이터베이스를 사용하여 정규화 과정을 거쳐 일관성을 유지하도록 하였으며, 구현 언어는 Borland C++ 3.1, Visual Basic 4.0, SQL 언어를 사용해 구현하였다. 논문의 구성을 살펴보면, 2장에서는 관련 연구에 관하여 살펴보고, 3장에서는 효율적인 정보를 저장하고 검색 할 수 있는 객체 지향 클래스 라이브러리 시스템 설계 방법을 제시하고, 4장에서 실 시스템의 예를 적용한 구현과 기존의 라이브러리 시스템과 평가 분석을 한다. 끝으로 5장에서 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

소프트웨어 재사용 시스템은 소프트웨어 부품들을 효율적으로 분류하여 이들을 컴퓨터에 저장하여 검색하는 시스템이며, 새로운 시스템을 개발하고 유지보수하는 노력을 줄이기 위하여 기존 시스템의 다양한 지식들을 소프트웨어 부품들의 정보로서 이용한다. 이와 같은 지식들은 소프트웨어 부품들의 정보로서 이용하며, 이 지식들은 소프트웨어 코드, 설계 정보, 요구 분석 사양서 등이 있다. 그러나 대부분의 시스템에는 소프트웨어 코드와 설계정보의 불일치 또는 설계 정보가 없는 경우가 많다. 따라서 지금까지의 연구는 소프트웨어 코드의 재사용에 중점을 두고 있다.

재사용 가능한 소프트웨어를 위한 분류 방법을 살펴보면 나열식(enumerative) 분류 방법과 패싯(facet) 분류 방법이 있다[6]. 나열식 분류 방법은 각 클래스의 계층 구조로 분류한 후 사용자가 인위적으로 할당하는 분류이지만 상속성 표현이 없고 확장성에 문제점이 있다. Diaz의해 개발된 라이브러리 시스템은 소프트웨어 모듈들이 갖는 속성들을 패싯으로 분류하여 검색하는 시스템으로, 질의어 구성 검색 등위를 결정하는 서브시스템으로 구성되어 있다 이러한 분류 방법은 한번 패싯이 결정되면 고정이 된다. 패싯 분류 방법은 소프트웨어 부품을 공통적인 특성을 모아서 패싯을 구성한 후에 각 패싯에 적합한 부품들의 내용들을 선택하여 조합한다. 이 방법은 확장성은 좋으나 클래스의 계층 구조의 명확성을 나타내기가 어려운 단점이 있다. 따라서 위의 두 가지 방법은 객체 지향 언어의 클래스들의 계층 구조에 새로운 클래스를 변경하기에 적합하지 않다.

클래스의 삽입시에는 소프트웨어의 재사용 측면을 고려하여 유사성을 고려하여야 한다. 대표적인 유사성 측정 방법은 inner-product measure, cosine measure,

pseudo cosine measure, Dice measure 방법 등이 있다[7]. 이러한 방법은 두 문서(i,j)간에 가중치들을 결정하는데 여기에서 가중치란 두문서들의 용어들에 의해 결정한다. 이 가중치들은 정규화하는 식을 이용하여 두문서의 유사성을 결정한다. 문서의 유사성을 결정하는 Dice 측정방법에서는 다음 식으로 결정한다.

$$S_{ij} = \frac{2 \times C}{A+B}$$

여기에서 C는 i 문서와 j 문서에 공통적으로 존재하는 의미 있는 단어의 수이고, A는 i문서에 존재하는 의미 있는 단어들의 수이고, B는 j문서에 존재하는 의미 있는 단어들의 수이다.

이것을 근거로 클러스터링하는 방법은 single pass, reallocation, complete link, group average link, Ward방법 등이 있다[7]. 그러나 이러한 방법은 객체 지향 언어에서 지원하는 상속 구조를 표시할 수 없다. 객체 지향 언어에서의 유사성은 클래스간의 데이터 멤버와 멤버 함수의 유사성 정도를 나타내며, 유사성 정도가 큰 것을 대상으로 클러스터링하게 된다. 클러스터링 하는 이유는 유사한 기능을 수행하는 클래스들을 묶어 동일한 장소에 두어 클래스 정보 재사용성이 높으며 인접 기억 장소에 두므로 검색 효율을 높일 수 있다[12,13].

이러한 객체 지향 소프트웨어의 재사용을 위해 사용하는 중요한 개념은 클래스 객체 상속 및 합성(composition)이 있다. 합성이란 이미 구현된 클래스들의 객체들을 새로운 클래스의 데이터 멤버들로 조합하여 클래스를 정의하는 것이며, 상속은 새로운 데이터의 추상화가 이미 정의된 자료 구조의 변화형 또는 간략형일 경우 사용된다. 일반적으로 객체 지향 언어로 프로그램을 작용했을 때 프로그램을 이해하는 기본 단위는 객체나 클래스이다. 그러나 객체 지향 언어는 객체 및 클래스 외에 클래스 관계성 정보(상속성, 포함성, 참조성), 다형성 정보(연산자 중복, 함수 중복, 가상 함수), 프랜드 함수 정보 등과 같은 개념의 이해가 반드시 필요하다. 현재 소프트웨어 재사용을 위한 연구 대상은 주로 원시 코드의 재사용을 목표로 하고 있으며 재사용 가능 프로그램 검색, 이해, 수정 그리고 새로운 부품으로의 합성을 위한 기법 개발을 위한 연구가 이루어지고 있다. 그러나 설계 문서의 재사용은 부품의 이해 및 수정에 대한 용이성을 제공하고 있기 때문에 이에 대한 연구도 활발히 진행되고 있다[12]. 특히 객체 지향 기법은 분석, 설계, 구현 결과가 재사용될 수 있는 단위로 만들어지므로 재사용을 용이하게 한다. 따라서 클래스에 대한 설계 결과를 정보 저장소에 등록하고 이를 추출할 수 있도록 지원한다.

일반적으로 재사용을 위한 클래스 탐색을 위한 기법은 다음과 같다[7].

첫째, 클래스 이름을 데이터로 탐색하는 것으로 사용자가 클래스 이름을 알고 있을때 사용할 수 있으며 구현하기가 쉽다.

둘째, 나열식 방법으로 클래스 상속 관계를 트리로 해석해서 찾아낸다.

셋째, 패싯 방법으로 클래스의 특성을 이용하여 클래스를 찾아내는 것으로 각 클래스를 정보 저장소에 등록할 때 그 클래스의 특성을 표현할 수 있는 값들을 한당하고 탐색시에 이 값을 이용한다.

넷째, 기존에 알고 있는 클래스를 기준으로 그 클래스에 인접한 관계를 갖는 클래스를 찾아내는 방법이다. 기존의 부품 탐색 대상은 주로 독립된 소프트웨어를 찾아내는 것이다.

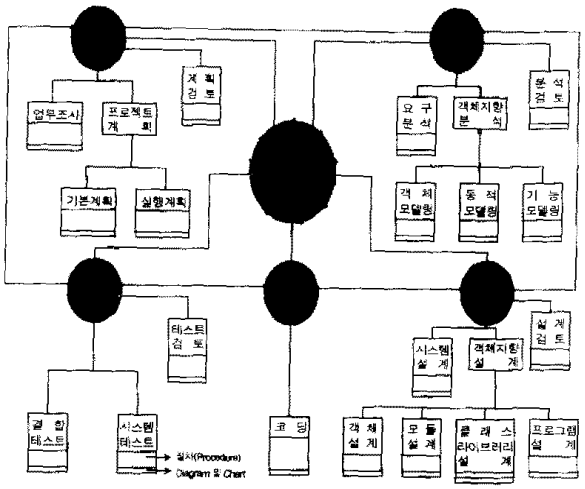
본 논문에서는 키 워드 기법과 트리 기법을 사용한다. 그리고 프로그램의 이해를 돕기 위해 모듈 단위로 분해하여 화면상에서 설계 모듈과 프로그램 모듈을 서로 비교하면서 탐색 할 수 있는 방법을 제시한다.

그리고 이와같은 방법들을 적용하여 구축한 재사용 시스템을 살펴보면, 대표적으로 RSL, Diaz에 의하여 개발된 시스템, 그리고 국내의 CARS 시스템 등이 있다[15]. RSL(Reusable Software Library) 시스템은 정보 검색 방법을 사용한 재사용 시스템으로, 재사용 부품의 속성에 대한 주석과 원시 코드로부터 재사용에 필요한 정보를 추출하여 데이터베이스에 저장한다. 사용자 질의는 메뉴방식으로 처리되며, 질의어의 키워드와 데이터베이스의 키워드를 비교하여 검색한 후, 사용자가 제공하는 선정 기준에 따라 가장 적합한 부품을

평가한다. 따라서 자연어 질의가 가능하다는 장점이 있다. 그러나 저장 및 검색이 사용된 키워드에 따라 시스템의 효율이 결정되며 질의에 사용된 키워드의 의미를 추론하기 곤란한 단점이 있다. Dias에 의해서 개발된 라이브러리 시스템은 소프트웨어 모듈들이 갖는 속성들을 패싯으로 분류하여 검색을 지원하는 시스템으로, 질의어 구성, 검색, 등위를 결정하는 서브 시스템으로 구성되어 있다. 이러한 분류 및 검색 방법은 도메인 모델을 설정하여 질의의 구성과 재구성에 사용하는 방법으로서 계층적 분류 방법에 비해 높은 정확도와 확장성을 갖지만 한번 분류가 설계되면 고정되는 단점이 있다. CARS는 라이브러리 관리 시스템, 검색 시스템, 사용자 인터페이스, 품질 보증 시스템으로 구성되며, 응용 도메인으로 그래픽스, 자료구조, 윈도우즈 라이브러리를 갖추고 있다. 또한 CARS 는 객체 지향 모델을 기반으로 하고 있어, 객체 지향 방법론으로 개발된 부품의 분석을 통하여 부품들간의 관련성을 자동적으로 추출한다.

3. 객체 지향 클래스 라이브러리 설계

본 논문에서는 (그림 1)의 객체 지향 시스템 개발 절차를 따른다[17]. 여기에는 계획, 분석, 설계 구현, 테스팅 단계의 소프트웨어 생명주기를 나타내고 있으며, 각 단계에서 검토는 프로토타입 시스템을 통한 조율(tuning) 과정이라 할 수 있다. 그리고 이 방법은 분석 단계에서 OMT 방법을 기준하여 설계 단계에서는 시스템 설계, 객체 설계, 객체 모듈 설계, 클래스 라이브러리 설계, 프로그램 설계등으로 세분하여 상향식 분석 결과를 하향식 접근 방법에 따라 단계적으로 단위 클래스에 대한 프로그램 단위의 정보를 추출하도록 한다.



(그림 1) 객체 지향 시스템 개발 절차
(Fig. 1) Development Process of Object-Oriented System

구 분	객체설계 단계	객체 모듈 설계 단계			프로그램 설계 단계		구현 단계
		추상화 클래스 단위 분할		클래스 단위 분할	이벤트단위 분할(프로그램단위)		코 딩
시스템	객체모델	추상화클래스A1 · · 추상화클래스An α1	추상화클래스A1.1 · · 추상화클래스A1.n α2	클래스 A1.1.1 클래스 A1.1.2 ..... 클래스 A1.1.n 클래스 A1.n.1 클래스 A1.n.2 ..... 클래스 A1.n.n α3	이벤트 A1.1.1.1 이벤트 A1.1.1.2 ..... 이벤트 A1.1.1.n 이벤트 A1.n.1.1 이벤트 A1.n.1.2 ..... 이벤트 A1.n.n.n	이벤트A1.n.1.1.1 이벤트A1.n.1.1.2 이벤트A1.n.1.1.3	

(그림 2) 객체 지향 설계 과정별 모듈 분해
(Fig. 2) Module Decomposition for Object-Oriented Design Process

본 논문의 연구 범위는 설계 정보 및 원시프로그램을 모듈 단위로 저장하고 탐색할 수 있는 객체 지향 클래스 라이브러리 설계 방법을 제시한다. 따라서 프로그램 재사용을 위한 부품을 모듈단위의 클래스로 생성하여 각 정보를 테이블에 저장하고 모듈간의 참조 관계 클래스를 추출한다. 재사용 및 유지보수시 설계모듈과 프로그램을 참조하여 동시에 볼 수 있다. 따라서 모듈간의 참조 관계, 상속 관계, 부품 관계, 참조 관계, 다형성 관계, 프랜드등의 추가적인 정보를 생성하여 데이터베이스에 테이블화 하여 필요한 정보를 조인하여 사용 할 수 있다. 다음은 객체지향 클래스 모듈 분해 방법을 살펴본 후, 클래스 분류 및 라이브러리 설계 방법을 제시하고자 한다.

3.1 객체 지향 모듈 분해 방법

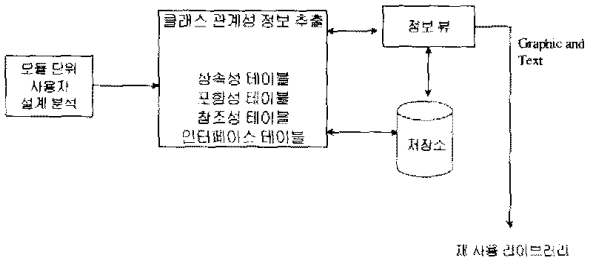
본 연구의 객체 모듈 분해 방법은 객체 설계 모델에서 모듈 설계를 위한 (그림 2)와 같은 하향식 방법을 따른다[17]. 즉, 하나의 사건을 처리하는 프로그램 단위로부터 상위의 추상화 클래스로 일반화(generalization)시켜 최상위 수준의 하나의 클래스를 생성하는 과정을 보여주고 있다. 여기에서 추상화 클래스란 객체 지향 언어에서의 추상 클래스(abstract class) 개념은 아니다. 그러나 최상위 수준에서는 이를 포함한다.

(그림 3)은 일반화된 클래스 계층구조에서 모듈의 개수를 나타내고 있는데, 상위 클래스는 적어도 하나 이상의 서브 클래스를 포함하고 있기 때문이다. 즉, (그림 3)에서 모듈의 개수는 $\alpha 1 \leq \alpha 2 \leq \alpha 3$ 의 조건을 만족하여야 한다.

3.2 클래스 라이브러리 설계 방법

클래스 라이브러리는 구현 또는 설계시 필요한 객체 클래스 집합으로, 프로그램의 재사용을 위한 부품은 모듈 단위이다. 각 정보는 테이블에 저장하여 모듈간에 참조할 수 있는 인터페이스 클래스도 추출한다. 또한 소프트웨어 재사용 및 유지 보수시 설계 모듈과 프로그램를 참조하여 동시에 볼 수 있게 한다.

본 논문에서는 나열식 분류 방법을 개선하여 클래스 상속 관계를 그래프로 해석해서 찾고 그것에 관계되는 데이터 멤버와 멤버 함수을 효율적으로 사용할 수 있도록 설계 하였다. 그리고 삽입시 클래스의 특성을 찾아 그 클래스의 특성에 맞도록 할당하므로서 탐색시에 효율을 기할 수 있도록 하였다. (그림 4)는 객체 지향 모듈 설계의 이해를 돕는 시스템 설계를 나타내고 있는데, 여기에서는 설계 측면의 정보를 저장하고 탐색 가능하도록 한다.



(그림 4) 객체 지향 모듈 설계의 이해를 돕는 시스템 설계
(Fig. 4) The Design of System for Understanding Object-Oriented Module Design

객체설계	객체 모듈 설계		프로그램 설계
	추상화 클래스	클 래 스	이벤트 (최종 모듈 단위)

객체모델 $\leq \left[\begin{bmatrix} A1 \rangle A1.1 \\ \vdots \\ An \rangle An.n \end{bmatrix} \leq \left[\begin{bmatrix} A1.1.1 \rangle A1.1.1.1 \\ \vdots \\ An.n.n \rangle An.n.n.n \end{bmatrix}\right] \leq \begin{bmatrix} A1.1.1.1\ldots \\ \vdots \\ An.n.n.n\ldots \end{bmatrix}\right]$

(그림 3) 모듈 분해 과정별 모듈 개수 관계
(Fig. 3) Relationship of Account for Module Decomposition Process

아래의 알고리즘 #1은 객체 지향 모듈 설계에 따른 클래스 정보 저장소 저장 및 정보 뷰를 표현하기 위한 알고리즘이다.

```
〈 #1 클래스 설계 모듈 저장 알고리즘 〉
ModuleInput(ModuleName,class)
begin
        OpenDataBase();
        for every class ∈ Module begin
              AddToTable(ModuleName,class)
        end;
   InformationView();
end;
```

3.2.1 클래스 테이블 생성

클래스를 생성 모델을 통해 저장소에 설계된 관계 테이블에 정보를 삽입하기 위해 모듈명과 클래스명을 입력한다. 그리고 멤버 입력, 데이터 멤버명, 멤버 함수, 멤버 타입, 멤버의 식별자 키를 모두 삽입하고 클래스 입력을 하면 클래스 내용 테이블에 삽입된다. #2 알고리즘 의해 클래스 관계 테이블을 생성할 수 있다.

```
〈 #2 클래스 입력 알고리즘 〉
ClassInput(className)
begin
        {Member_function,  Data_member}
                ∈ MemberAttribute
        OpenDataBase(0;
        for every member ∈ class begin
                create(className,
                MemberAttribute);
        end;
        AddAttributeCount++;
end;
```

3.2.2 클래스 관계성 테이블 생성, 인터페이스 테이블 생성

여기에서는 모듈에 대한 클래스의 상속성, 복합, 참조성을 구분하여 테이블을 생성한다. 저장된 클래스명을 추출하여 관계성을 입력하여 관계성 테이블에 저장된다. 또한 입력, 정정, 삭제도 가능하다. 아래의 알고리즘 #3과 #4는 모듈 테이블과 상속, 참조, 복합, 인터페이스 관계 테이블을 생성한다.

```
〈 #3 클래스 관계성 입력 알고리즘 〉
RelationSet : β
  β = { IS_A, PART_OF }
RelationInput(class1, class2)
begin
      if relation(class1,class2) ∈ β then
           AddToTable(class1, class2,
           ClassRelation);
      end;
end;
/* 참조 관계 테이블 구성 알고리즘 */
ReferenceTableInput(class1,
memberFuntion1, class2, memberFunction2 )
begin
      α : method call in member function
      for every memberFunction1 ∈class1
      begin
           for every α ∈ (memberFunction1
                        ∈ class1)
                AddToTeble(memberFunction1,
                class2, memberFunction2,
                Reference);
  end;
end;
```

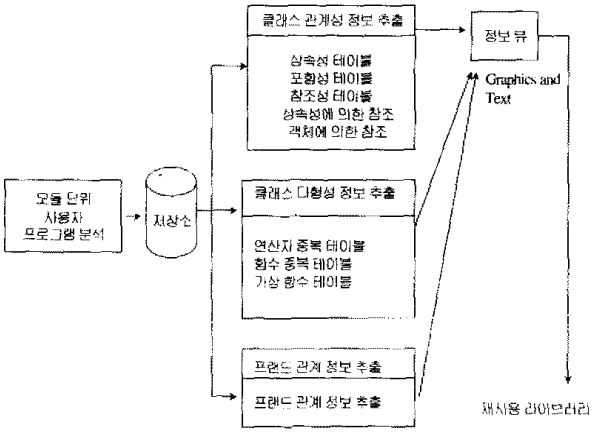
```
〈 #4 인터페이스 클래스 입력 알고리즘 〉
ModuleInterfaceInput(Mddule1,Module2,class )
begin
      for every((class ∈ Module1) ∩ (class
      ∈ Module2))
      begin
           AddToTable(Module1,Module2,class,
           Interface)
  end;
end;
```

다음은 프로그램을 재사용 할 수 있는 방법을 제안한다. 이와 같은 재사용 가능한 프로그램을 이해하기 위하여 일반적으로 세 가지 방법이 사용된다. 즉, 그 프로그램에 연관된 문서를 읽고 이해하는 방법, 프로그램을 직접 읽는 방법, 직접 프로그램을 실행하며 이해하는 방법이 있다. 그러나 기존의 구조적 프로그램을 이해하는데는 프로세스 중심의 모듈 단위 였으나 객체

객체 지향 프로그램에서는 프로그램 이해 단위는 클래스이고 상속, 복합, 참조성을 지원하나 클래스의 복잡도가 커지면 프로그램 다형성 문제, 프랜드 관계등이 프로그램 이해를 어렵게 할 수도 있다. 이러한 문제는 클래스와 클래스간의 구조뿐만 아니라 클래스와 클래스들의 다양한 행위를 포착할 수 있는 행위 모델을 작성하여 수행중에 발생할 수 있는 동적 행위에 따라 클래스 계층 구조를 재구성하는 것이 바람직하며 상속이 부적절할 때 위임(delegation)관계를 이용할 수 있다. 본 논문은 이러한 문제를 정보 저장소에 테이블 형태로 저장하고, 정보를 시각화하여 프로그램 이 해도를 높일 수 있다. 모듈별로 저장된 테이블을 이용하여 다음과 같은 정보를 추출 할 수 있다.

① 일반화 관계인 상속계층 구조
② 집단화 관계인 포함 관계
③ 클래스의 사용관계
④ 클래스에 정의된 멤버 함수 데이터 멤버(실제코드)
⑤ 중복된 함수
⑥ 가상 함수, 재 정의된 멤버 함수
⑦ 프랜드 관계 정보
⑧ 상속을 받는 재정의 되는 함수

다음은 프로그램을 이해를 지원하기 위한 시스템 구성은 (그림 5)와 같다.



(그림 5) 객체지향 프로그램 이해 지원 시스템 설계
(Fig. 5) The Design of System for Understanding Object-Oriented Program

3.3 클래스 삽입을 위한 방법

일반적으로 상속을 남용하게 되면 메시지 전달에 따른 오버헤드가 발생하게 되어 시나지게 복잡도를 증가시킨다. 잘 개발된 클래스 상속 계층 구조는 상속 트리의 깊이가 깊고 너비가 좁아야 한다고 한다. 그러나 폭이 깊을수록 상속되는 함수가 많아지게 되므로 테스트 및 재사용을 어렵게 하여 심리적 복잡도를 더욱 증가시킨다. 이러한 경우 클래스 계층을 재구성하게 되면 지나치게 중첩된 상속 계층 구조의 깊이를 감축시킬 수 있다. 일반적으로 노드 클래스의 삽입으로 인하여 클래스가 계층 구조가 변경될 경우 상속 관계가 변경되어 클래스 내부 구현 프로그램의 의미가 달라질 경우가 있다. 본 논문에서는 일반화(generalization)와 세분화(specialization) 개념을 적용하여 의미의 변화가 거의 일어나지 않도록 할 수 있는 방법을 제안한다. 상위 클래스로부터 하위 클래스로 상속을 받을 때 상속으로인한 참조가 가능하나 객체 참조인 경우에는 명시적으로 표현한다. 본 논문에서는 클래스 테이블과 객체 참조 테이블 그리고 상속 관계 참조 테이블을 조인하여 사용한다. 그리므로 삽입 시에 일관성이 있으며 확장이 용이하다.

3.3.1 클래스 관계성 및 유사성 형식 정의

이와 관련된 클래스 객체들의 관계성을 위한 형식 정의(formal definition)에 관한 연구는 상대적으로 미흡하다. 그 이유는 객체 지향 기법의 발전이 수학적 기반보다는 경험적 원리를 기반으로 하고 있기 때문이다. 형식 정의는 Wand에 의하여 객체 지향 시스템에 적합하도록 정의되었다[11]. 그리고 클래스들의 유사성(similarity)이란 두 개의 클래스 사이에서 동일한 도메인(데이터 멤버 + 멤버 함수)이 많을수록 클래스간의 유사성 값은 증가하고 반면에 적을수록 감소한다. 따라서 이러한 값의 결과로 설계에서 상속 계층 구조 설정과 구현시 코드의 재사용 측면을 결정하게 된다[1,12].

본 논문에서는 기존의 연구 결과를 기반으로 정의한다[12].

[기본 정의]

- d(domain) : 데이터 함수와 데이터 멤버
- dm : 데이터 멤버
- dmf : 데이터 멤버 함수

[정의 1] 상속 계층 관계에는 일반화(generaliza-

tion)와 세분화(specialization) 관계로 나뉘어 진다.

〈조건 1〉 일반화는 상위 클래스 C_i의 도메인(domain)이 하위 클래스 C_j의 부분 집합이라면 C_i는 C_j의 일반화라 한다. 즉, $(C_i \supset C_j)$의 조건을 만족한다.

〈조건 2〉 세분화는 일반화의 역으로 $(C_j \supset C_i)$가 성립하면 하위 클래스 C_j는 클래스 C_i의 세분화라고 한다. 즉, $(C_j \supset C_i)$의 조건을 만족한다.

〔정의 2〕 클래스의 데이터 멤버 유사성

$$SIM_{dm}(A, B) = \frac{dm(A \cap B))^2}{dm(A) \times dm(B)}$$

〔정의 3〕 클래스의 멤버 함수 유사성

$$SIM_{dmf}(A, B) = \frac{dmf(A \cap B))^2}{dmf(A) \times dmf(B)}$$

〔정의 4〕 클래스들의 유사성

$$SIM(A, B) = P \times SIM_{dm}(A, B) + (1-P) \times SIM_{dmf}(A, B)$$

〔정의 2〕에서 $dm(A)$는 클래스 A를 구성하는 데이터 멤버의 개수이고, $dm(B)$는 클래스 B를 구성하는 데이터 멤버의 개수를, 그리고 $dm(A \cap B)$는 두 클래스 내에서 공통적인 기능을 갖는 데이터 멤버의 개수를 나타낸다. 〔정의 3〕 에서도 〔정의 2〕 같은 개념이지만 여기에서는 멤버 함수(dmf)를 나타낸다. 그러나 두 개의 클래스 멤버 함수가 같은 기능을 수행한다는 것을 판명하는 일반적인 알고리즘 개발은 어렵기 때문에 다음의 기준을 따른다.

첫째, 두 함수의 이름과 함수가 반환하는 값의 타입(type)이 같아야 한다.
둘째, 함수의 모든 인자들의 타입이 같아야 한다.
셋째, 함수가 참조하는 전역 변수의 타입이 같아야 한다.
넷째, 지역 변수들의 타입이 같아야 한다.
다섯째, 함수내의 제어 흐름도가 같아야 한다.
여섯째, 각 변수(함수 인자, 전역 변수, 지역 변수) 별로 프로그램상에서 프로그램 흐름도가 같아야 한다.

본 논문에서는 같은 이름을 갖는 멤버 함수인지와 타입을 비교하여 측정한다. 〔정의 4〕는 클래스들의 유사성으로 〔정의 2〕와 〔정의 3〕의 식을 사용하여 정의한 것이다. 여기에서 P는 클래스 A, B의 데이터 멤버들의 총 개수와 멤버 함수들의 총 개수를 합한 값에서 데이터 멤버들의 총 개수의 비율이다.

3.3.2 클래스 삽입 방법

객체 지향의 소프트웨어 주기에서 사용자의 요구사항을 만족시키기 위하여 또는 주요 응용 프로그램 업무는 처음부터 완벽하게 개발한다고는 할 수 없다. 그리고 클래스를 일관성 없이 삽입하면 유지보수시 프로그램 코드가 필요이상으로 커지고 이해하기가 어렵게된다. 그러므로 클래스 삽입으로 약간의 재구성이 필요하고 좀더 일관성을 유지할 필요성이 있다. 클래스와 클래스 사이에 클래스를 삽입하여 재구성 할 경우는 상당한 문제를 가지고 있다. 그러므로 본 논문에서는 프로그램 분석을 기반으로한 정보 테이블을 기반으로 행렬 메트릭스 라고 부르는 격자 구조로된 인터페이스의 집합을 사용한다. 또한 유사성은 같은 이름의 멤버 함수인지와 타입을 비교하여 측정하게 되며 유사성을 근거로 하여 삽입 위치를 결정하게 된다. 완전히 구현된 실용예에서의 삽입은 항상 리프에 오버라이딩 하여 사용하는 것이 일반적인 경우이다. 본 논문에서는 프로그램 모듈을 기반으로 하여 클래스에 대한 정보를 추출하고, 삽입할 클래스와 모듈내의 클래스 계층에서 유사성 검사를 통하여 유사성 경로를 찾은 후, 삽입하는 방법을 제시하고자한다.

본 논문에서 제시하는 클래스 삽입 방법은 다음과 같다.

① 기존의 한 모듈 안의 클래스 계층 구조에서 유사성 검사를 하여 삽입할려는 클래스의 유사성 경로를 구한다.
② 검색된 유사성 경로의 모든 노드에 대해 루트노드에서 부터 리프노드까지 탐색을 시작한다.
③ 탐색경로내의 노드(α)와 삽입할려고 하는 노드

(β)에서 공통 정보를 추출한다.

④ 공통정보가 존재할 경우

▷ α 내에 공통 정보를 제외한 나머지 정보가 NULL과 같지 않을 경우

- 임시 테이블(γ)을 만든다.
- 만들어진 임시 테이블에 α와 β의 Attribute (공통정보)를 삽입한다.
- γ에 superclass 정보를 currentClassSet를 참조하여 생성한다.
- α의 superclass정보를 갱신한다.
- Attribute(γ)=Attribute(α)이면 α는 β의 superclass가 된다.
- Attribute(γ)≠Attribute(α)같지 않으면 currentClassSet에 γ를 추가한다

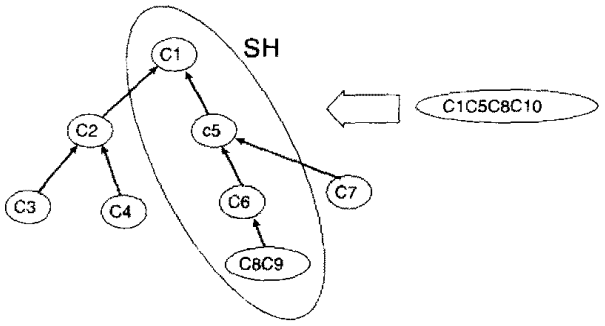
▷ α 내에 나머지 정보가 NULL과 같을 경우

- β 공통정보를 제외한 나머지 정보가 존재할 경우
  α는 β의 superclass가 되고 current-ClassSet에 α를 추가한다.
- 나머지 정보가 존재하지 않을 경우
  α는 β와 동일 정보가 되고, equalClass 플래그를 true로 설정한다.

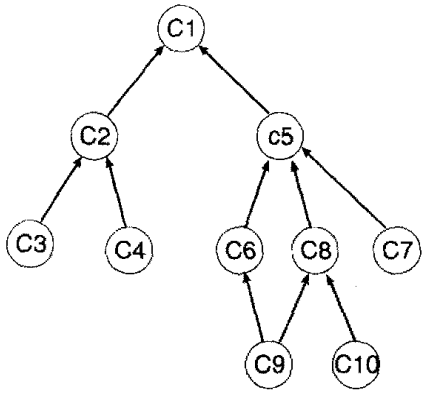
⑤ equalClass가 false이면

▷ 삽입할려는 클래스중에 아직 남아 있는 특성 정보가 있을 때 기생성된 class에 추가한다.

(그림 6)에서 유사성 경로 검사를 통해 SH (Similarity Hierarch)를 구하고, SH의 집합은 {C1, C5, C6,(C8, C9)}가 된다. 삽입 집합은 β는 {C1, C5, C8, C10}이 된다. 검색된 유사성 경로를 루트에서 부터 리프로 탐색한다. 첫 번째 노드 C1과 삽입할려는 클래스 β의 C1이 동일하다. α 내의 공통 정보 C1을 제외한 나머지 정보는 NULL과 같고, β의 공통정보를 제외한 나머지 정보가 존재하기 때문에 current-ClassSet에 α를 추가한다. currentClassSet은 {C1}이 된다. 유사성 경로내의 다음 노드 C5와 삽입할려는 클래스 β의 C5가 동일 하기 때문에 currentClass-Set은 {C1,C5}가 된다. 다음 노드 C6은 β내의 어떤 원소와도 동일하지 않기 때문에 처리하지 않고, 유사성 경로내의 후위 노드를 처리한다. 후의노드 {C8, C9}와 β와의 공통정보는 {C8}이고, α의 나머지 정보는 C9가 된다. 따라서 삽입방법에 의해 임시 클래스 γ를 생성하고, 공통정보를 γ에 삽입, γ의 수퍼 클래스 정보를 currentClassSet를 참조하여 생성하기 때문에 super(γ)은 {C1, C5}가 된다. super(α)는 {C1,C5,C6,C8}이 된다. 또한 Attribute(γ)와Attribute(α)가 같지 않기 때문에 currentClassSet에 γ를 추가한다. currentClassSet는 {C1, C5, C8}이 된다. β의 처리되지 않는 원소 C10은 이미 생성된 클래스 뒤에 삽입한다. 그림(7)은 삽입후의 클래스 계층이다.



(그림 6) 삽입하기전 클래스 계층
(Fig. 6) Before Class Hierarchy Insertion



(그림 7) 삽입후의 클래스 계층
(Fig. 7) After Class Hierarchy Insertion

본 논문에서 제시하는 클래스 삽입 방법은 다음과 같다.

```
procedure classInsert(CH, α)
begin
//CH : Class Hierarchy
// α : 유사성 경로내의 노드 클래스
//β  : 삽입할려고 하는 클래스
      currentClassSet = NULL
      equalClass = false
      // 기존의 계층 클래스 구조에서 삽입할려는 클래스에대한 유사성
      경로 탐색
      searchSimilarityNode();
      for every node ∈ class hierarchy // 유사성 경로노드의 루트
      노드로부터 출발
          // commonAttribute(α,β)는 계층 구조의 노드와 삽입 노
          드와의 공통 정보를 추출
```

```
if commonAttribute(α,β) ≠ NULL then begin
        // IS_any_element_Attribute(α) 노드에서 삽
        입노드와의 공통
        // 정보를 뺀 나머지 정보
        if IS_any_element_Attribute(α) ≠ NULL
        then begin
        temporaryClassCreate(γ) // 임시 클래스 γ 생
        성
        // γ에 공통 정보의 특성(멤버 데이터와 멤버함
        수) 삽입
        Attribute(γ) = commonAttribute(α,β)
        // γ의 super class 정보 삽입
        Attribute(γ) = commonAttribute(α,β)
        // γ의 super class 정보 삽입
        superClass(γ)
        = superClass(currentClassSet)
        superClass(α) = superClass(α) ∪ γ
        if Attribute(γ) =Attribute(α) then begin
            // β 는 α의 super class가 된다.
            equalClass = true
        end
        else begin
            currentClassSet = currentClassSet ∪ {
            α }
        end
    else begin
    // 기 만들어진 노드와 삽입할려고 하는 클래스의 특성이
    같으면
    // commonAttribute(α,β) == NULL
        if IS_any_element_Attribute(β) ≠ NULL then
        begin
            // α는 β의 superclass가 되고
            // currentClassSet집합에 node를 추가한다.
            currentClassSet = currentClassSet ∪ { α }
        end
        else begin // α와 β는 동일 클래스
                    equalClass = true
                end
            end
        end
    end for every

        //삽입할려는 클래스 중에 아직 남아 있는 특성 정보가 있을
    때 이미 생성된 class에 추가한다.
    if not equalClass then begin
        temporaryClassCreate(γ)
        Attribute(γ) = commonAttribute(α,β)
        superClass(γ) = superClass(currentClassSet)
        superClass(α) = superClass(α) ∪ γ
    end
end // end of procedure classInsert
```

4. 실 시스템 적용 및 평가

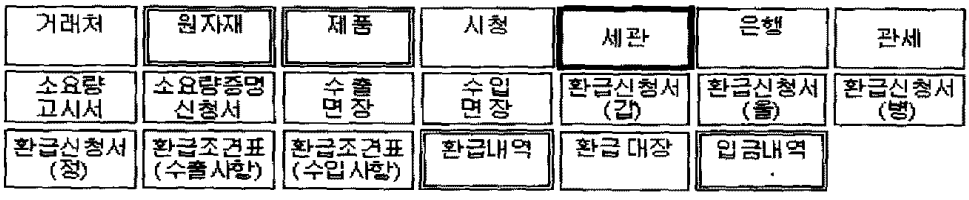
4.1 실 시스템 적용

본 논문의 적용 시스템은 기업의 무역관리 업무중에서 관세환급 시스템이다. 관세 환급 처리시스템을 위한 식별된 클래스들은 (그림 8)과 같다. 여기에서 시청 세관 은행은 외부 클래스로 관계성 설정시 제외되었으며 원자재, 제품, 환급 내역은 인터페이스 클래스로 다른 모듈에서 필요한 클래스로 추출되었다.

이 시스템에서는 다음과 같이 면장 관리 ,환급 신청서 관리, 소요량 증명서 관리, 조건표 관리, 환급처리 내용 관리의 모듈로 구분되었다. 〈표 1〉은 객체지향 프

〈표 1〉 모듈 관계 테이블
〈Table 1〉 Module Relation Table

[illegible]	License
License_Management	Customer
License_Management	Import_License
License_Management	Export_Licence
License_Management	Matrial
License_Management	Product
RAP_Management	Return_amount
RAP_Management	Process_amount



(그림 8) 식별된 클래스
(Fig. 8) Discriminated class

〈표 2〉 클래스상속 관계 테이블
〈Table 2〉 Class Inheritance Relation Table

	Import_License	IS-A	PUBLIC
License	Export_License	IS-A	PUBLIC
Return_Amount_Bird'schart	Import_Return_Amount_Bird'schart	IS-A	PUBLIC
Return_Amount_Bird'schart	Export_Return_Amount_Bird'schart	IS-A	PUBLIC

〈표 3〉 클래스의 복합 관계 테이블
〈Table 3〉 Class Composite Relation Table

License	Transaction	PART-OF	PUBLIC
Import_License	Customs	PART-OF	PUBLIC
Export_License	Product	PART-OF	PUBLIC
Return_Amount	Product	PART-OF	PUBLIC
Return_Amount	Material	PART-OF	PUBLIC

〈표 4〉 클래스 정의 테이블
〈Table 4〉 Table of Class Definition

	chFirmName;	data	private	char *		
Customer	chAddress	data	private	char *		
Customer	chTel	data	private	char *		
Customer	setFirmName	function	public		void	char *
Customer	setAddress	function	public		void	char *
Customer	setTel	function	public		void	char *
Customer	getFirmName	function	public		void	char *
Customer	getAddress	function	public		void	
Customer	getTel	function	public		void	char *
License	nLicenseNo	data	private	long int		
License	nDeclarationNO	data	private	long int		
License	chDeclarationDate	data	private	char[8]		
License	chItem	data	private	char *		
License	chStandard	data	private	char *		
License	nApprovalMethod	data	private	long int		
License	nQuantity	data	private	int		
License	fReportedPrice	data	private	float		
License	License	function	public			
License	~License	function	public			
License	setLicenseNo	function	public		void	long int
License	setStandard	function	public		void	char *
License	getDeclarerNo	function	public		long int	void
License	getDealingClassification	function	public		int	void
License	getReportPrice	function			float	void
exportLicense	nReturnClassification	data	private	int		
exportLicense	chBuyer	data	private	char *		

〈표 5〉 클래스 참조 테이블
〈Table 5〉Table of Class Reference

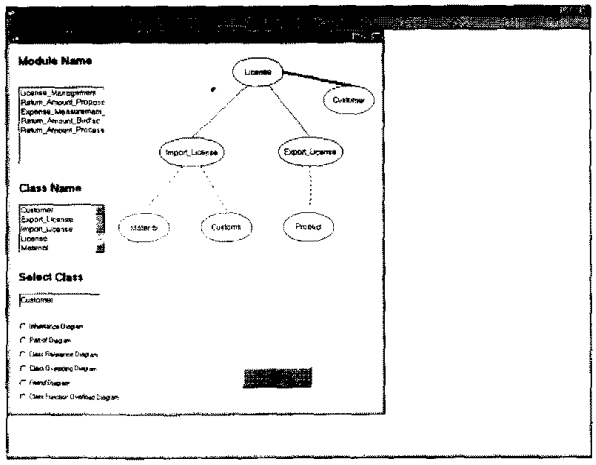
	License()	Customer	Customer()	Instance
License	License(long int, long int)	Customer	Customer(char *, char *, char *)	instance
License	~License			
License	setLicenseNo(long int)			
License	setCustomer(char *,char *,char *)	Customer	Customer(char *, char *, char *)	instance
License	setDeclarationNo(long int)			
License	setDeclarationDate(char [])			
License	setDealingClassification(int)			
License	setApprovalMethod(int)			
License	print()	Customer	print()	
exportLicense	exportLicense()	License	License()	
exportLicense	exportLicense(long int,char [*],char *)	License	License(long int, long int , char *)	instance
exportLicense	~exportLicense			
exportLicense	setReturnClassification(int)			
exportLicense	setBuyer(char *)			
exportLicense	getReturnClassification(void)			
exportLicense	getBuyer(void)			
exportLicense	print()	License	print()	
ImportLicense	ImportLicense()	License	print()	
ImportLicense	ImportLicense(long int,long int,char *,char *)	License	License()	
ImportLicense	~ImportLicense	License	License(long int, long int , char ...)	
ImportLicense	setProvider(char *)			
ImportLicense	setTaxableAmount(float)			
ImportLicense	setTaxRates(float)			
ImportLicense	setAmountofTax(float)			

〈표 6〉 인터페이스 테이블
〈Table 6〉 Interface Relation Table

	RAP_Management	Matrial
License_management	RAP_Management	Product
RAP_Management	Expanse_Managem	Weight_measure

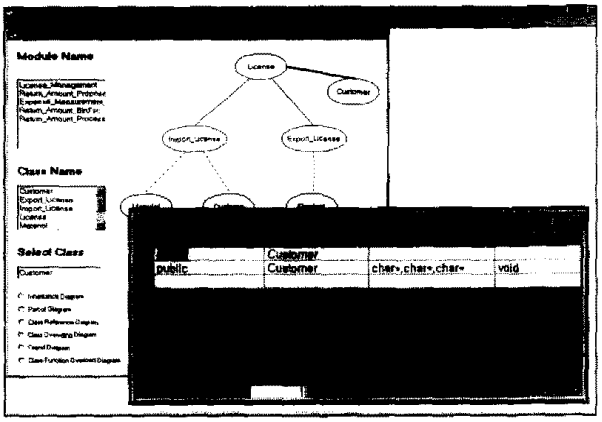
로그램 단계에서 클래스를 모듈별로 관리하기 위한 테이블이고 〈표 2〉는 클래스들 간의 상속 관계로 Class_Name1에서 Class_Name2 관계이다. 〈표 3〉은 클래스의 복합 관계로 Class_Name1에서 Class_Name2와 〈표 4〉은 정의된 클래스의 데이터 멤버 와 멤버 함수를 테이블이고, 〈표 5〉는 클래스의 멤버 데이타의 참조 관계를 나타내고 있다. 위의 테이블을 조인하여 클래스 삽입시 부울 격자를 생성 할 수 있다. 〈표 6〉은 인터페이스 테이블로 다른 모듈에서 공동으로 사용되는 클래스이다.

이와 같은 테이블들은 프로그램의 이해를 위한 정보를 정규화 하여 테이블에 저장하였다. (그림 9)에서 클래스 라이브러리 윈도우에서 모듈명에서 면장관리를 선택하면 면장 관리에 연관된 클래스가 추출된다. 그리고 추출된 클래스명인 면장을 클릭하고 부품 관계 다이어그램(Part_of)를 선택했을때 부품 관계 그래프가 추출되어 관계성 다이어그램에 따라 그래프가 생성된다. 그래프의 실선은 부품 관계를 나타낸다. 그리고 (그림 10) 거래처 클래스를 클릭하고 함수 중복을 선택하면 중복된 함수가 새로운 윈도우에 나타나게된다.

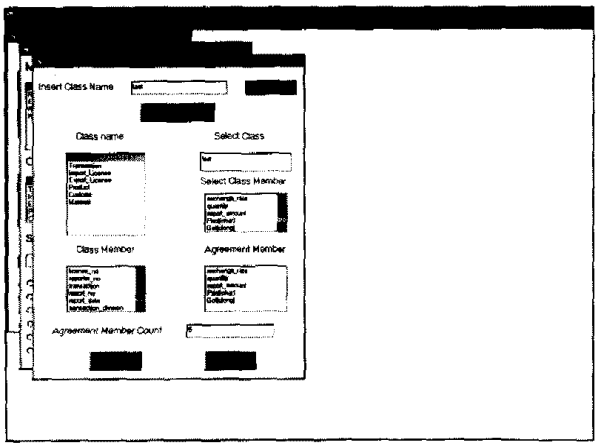


(그림 9) 클래스 라이브러리 정보 - 1
(Fig. 9) Class Library Information - 1

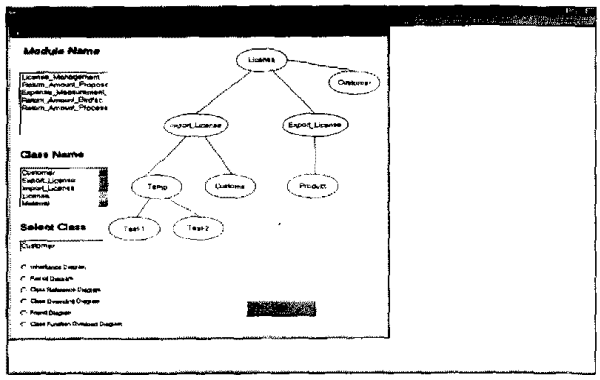
(그림 11)은 Class Search 버튼을 누르면 삽입 클래스를 찾을 수 있는 폼이 뜨며 그 폼에서 클래스를 선택하고, 각 클래스를 클릭하면 삽입 클래스와 비교하여 유사성이 있는 멤버를 Agreement Member에 나타내며, Agreement Member Count에 유사성이 같은 멤버에 대한 개수를 표시하게 된다. Class Name에 나오는 클래스들은 선택된 모듈의 클래스이며, Class Member는 선택된 클래스의 멤버들이다. 삽입 후 모든 관계를 가지고 있는 테이블은 수정이 된다.



(그림 10) 클래스 라이브러리 정보 - 2
(Fig. 10) Class Library Information - 2



(그림 11) 클래스 삽입
(Fig. 11) Class Insert



(그림 12) 삽입후 클래스 라이브러리
(Fig. 12) Class Library After Insert

〈표 7〉기존의 시스템 비교
〈Table 7〉 Comparsion of existing system

비교 항목	RSL	Diaz	CARS	본 논문의 제시 방법
부품의 성격	함수	함수	클래스	클래스
라이브러리 구성요소	함수형 부품, 정 해진 레이블에 따라 기술된 문서	함수형 부품, 동의어 사전	클래스	설계 문서 및 프로그램의 클래스
재사용 방법	함수 호출	함수 호출	객체 생성 상속,	객체 생성 상속,
색인 방법	정보검색방법, 이진색인	정보검색방법, Facet의 term	지식기반구조, 클래스 계층구조	모듈단위 계층구조, 클래스 및 기능 관계정보
분류 방법	Facte	Faceted Conceptual ordering	Inheritance	Inheritance
질의어 성격	자연어	제한된 Facet 자연어	자연어	자연어
검색 방법	Keyword matching	keyword matching	계층 구조도 이용	색인 및 유사도 이용
검색 환경	Menu-driven	Menu-driven	UI, Iconic Interface	UI, Iconic Interface

삽입 클래스 Test, License관리의 관계성에서 삽입 후 변화된 것을 볼 수 있다. 여기서는 Matrial class에서 Test와의 공통 정보를 갖는 클래스 Temp가 Matrial class의 위치에 치환되고 공통 정보를 제외한 Test-1과 Test-2는 임시 클래스가 생성된다. (그림 12)는 클래스 삽입후 변화된 클래스 구성에 관한 것이다.

4.2 평가

여기에서는 기존의 구축된 재사용 시스템과, 본 논문에서 제시한 클래스 라이브러리 시스템의 비교 항목을 설정하여 비교하고자 한다. 〈표 7〉의 평가 항목으로는 부품의 성격, 라이브러리 구성요소, 재사용 방법, 색인 방법, 분류 방법, 질의어 성격, 검색 방법 및 검색 환경 등이다.

본 논문에서 제시하는 클래스 라이브러리 시스템은 기존의 방법과는 달리 정형화된 모듈 분해 이론을 적용하여, 정보를 모듈별로 클래스간의 관계성을 분류하여 저장한다. 즉, 이 방법은 프로그램을 모듈 단위로 저장하고 탐색할 수 있으며, 프로그램의 재사용을 위한 부품을 모듈 단위로 생성하여 각 정보를 테이블에 저장하여 모듈간에 참조할 수 있는 인터페이스 클래스도 추출한다. 또한 객체 지향 소프트웨어 설계 및 유지 보수의 도움을 주기 위해 클래스 분류 작업을 수행한 후 이를 클래스 라이브러리 저장소에 저장하여 효율적으로 탐색할 수 있도록 테이블화 하였다. 그리고 클래스 변경시 클래스 노드 삽입은 위치를 설정하고 멤버 함수와 데이터 멤버를 일반화함으로써 프로그램 이해를 쉽게 할 수 있게 하였다. 따라서 본 논문에서 제시하는 방법은 프로그램 개발 및 유지보수시에 프로그램의 이해력를 높여 재사용 시스템 구축을 용이하게 한다.

5. 결 론

본 논문에서는 프로그램의 이해와 재사용을 위한 객체 지향 클래스 라이브러리 설계 방법을 제시하였다. 이 방법은 객체지향 프로그램을 모듈 단위로 저장하고 탐색할 수 있으며, 프로그램의 재사용을 위한 부품을 모듈 단위로 생성하여 각 정보를 테이블에 저장하여 모듈간에 참조할수 있는 인터페이스 클래스도 추출한다. 즉, 프로그램의 이해를 쉽게 하여 프로그램 코드를 기반으로 클래스 관계성을 그래프로 표현하고, 또한 클래스 정보를 시각화함으로써 프로그램 수정 및 삽입을 용이하게 하였다. 또한 객체 지향 소프트웨어 설계 및 유지 보수의 도움을 주기 위해 클래스 분류 작업을 수행한 후 이를 클래스 라이브러리 저장소에 저장하여 효율적으로 탐색할 수 있도록 테이블화 하였다. 기존의 클래스 삽입 위치 결정 및 삽입 방법의 어려움이 있다. 따라서 본 논문에서는 클래스 노드 삽입위치를 설정하기 위하여 메트릭스를 생성하여 유사성을 체크한 후 멤버 함수와 데이터 멤버를 일반화함으로써 프로그램 클래스 계층 구조를 쉽게 확장할 수 있으며, 설계 관점 측면과 프로그램 재사용 측면을 동시에 고려할 수 있도록 하였다. 따라서 본 논문에서 제시하는 방법은 프로그램 개발 및 유지보수시에 프로그램의 이해력를 높여 재사용 시스템 구축을 용이하게 한다. 그러나 제시하는 삽입 알고리즘은 복잡한 클래스 계층 구조일때에 모든 경우의 수를 시스템 개발자가 분석하는데 많은 시간을 필요로하는 단점이 있다. 앞으로의 연구 방향은 객체 지향 분산 환경 하에서의 객체의 시각화 및 응용 문제를 위하여 저장소 문제를 일관성 있게 할 수 있는

Intelligent Object Management System의 구성에 대한 연구가 필요하다.

참 고 문 헌

[1] Amos Tversky, "Features of Similarity", Psychological Review, Vol.84, No.4, pp.327-352, Jul. 1977.

[2] Elliot J. Chikofsky, "Reverse Engineering and Design Recovery : Taxonomy", IEEE Software, Vol.7, No.1, pp.13-17, Jan. 1990.

[3] K.C. K "A Reuse-based Software Development Methodology," proceeding of the workshop on software reusability, Oct. 1987.

[4] Gianluigi Caldiera and Victor R. Basili, "Identifying and Qualifying Reusable Software Compoent," IEEE Software, Vol.8, No.2, pp.61-72, Feb. 1991.

[5] Lehman M.M,"Programs, life-cycles and the laws of software evolation", proc IEEE 15(3), pp.225-252, 1980.

[6] R.Prieto-Diaz, "Implementing Faceted Classification For Software Reuse",Com. ACM, No. 4, pp.89-97, May, 1991.

[7] R.Prieto-Diaz, P. Freeman, "Classifying Software for Reusability", IEEE Software, pp. 6-16, Jan. 1987.

[8] T. C. Jones, "Reusability in Programming : A Survey of the State of the Art", IEEE Trans. on Software Engineering, Vol. SE-10, No.5, pp.488-494, Sep. 1984.

[9] Tim Korson and John D. McGregor," Understanding Object-oriented: A Unifying Paradigm," communications of the ACM Vol.33, No.9, pp.40-60, Sep. 1990.

[10] William B.Frakes and Ricardo Baeza-Yates, "Information Retrieval", Prentic-Hall, pp.419-422, 1992.

[11] Y. Wand, R. Weber, "An Ontological Model of an Information System", IEEE Trans. on SE., Vol.16, No.11, pp.1282-1292, Nov. 1990.

[12] 김갑수, 신영길, "소프트웨어 재사용을 위한 C++ 클래스 계층 구조 변형 방법", 한국정보과학회 논문지 제22권 제1호, pp.88-93, 1995.

[13] 김상욱, 신영길,이정태, "공유 객체를 지원하는 객체 중심 시각 프로그래밍 환경의 개발", 한국정보과학회 논문지 제2권 2호, pp.130-141, 1996.

[14] 김치수, "재사용 소프트웨어 컴퍼넌트의 합성과 릴레이션 쉽에 관한 연구", 한국정보처리학회 논문지, Vol.3, No.5, pp.1112-1119, 1996.

[15] 성백균, "소프트웨어 재사용을 위한 사례기반의 추론 모델", 중앙대학교 대학원 박사논문, pp.30-31, 1994.

[16] 최은만, 김진석, "객체 지향 재사용과 CASE ",한국정보과학회지 제14권 10호, pp.47-54, 1996.

[17] 허계범,이종섭,정계동, 최영근, "객체 지향 설계를 모듈 결합 방법", 한국정보처리학회 논문지 제3권 제4호, pp.817-833, 1996.

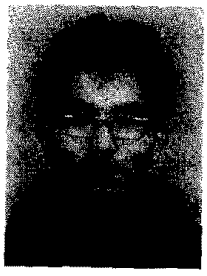
정 계 동

1985년 광운대학교 전자계산학과 졸업(이학사)

1992년 광운대학교 산업대학원 전자계산학과(이학석사)

1992년~현재 광운대학교 전자계산학과 박사과정

관심분야 : 객체 지향 프로그래밍언어 및 데이터베이스, 병렬처리 프로그래밍언어, 객체지향 분산 컴퓨팅



권 오 진

1990년 광운대학교 전자계산학과 졸업(이학사)

1990년~1992년 수도군단 유선소대장

1994년 광운대학교 대학원 전자계산학과(이학석사)

1994년~현재 산업기술정보원(KINITI) 연구원

관심분야 : 병렬처리, 객체지향 소프트웨어, 검색시스템, 시각언어

최 영 근

1980년 서울대학교 수학교육과 (학사)

1982년 서울대학교 계산통계학과 (이학석사)

1989년 서울대학교 계산통계학과 (이학박사)

1992~현재 광운대학교 전자계산학과 교수

1992~현재 광운대학교 전자계산소 소장

관심분야 : 프로그래밍 언어, 병렬 프로그래밍언어, 객체지향 프로그래밍언어 및 설계, 객체지향 분산 컴퓨팅