# C++ 코드로부터 클래스 관련 정보 생성 도구의 설계 및 구현

장 덕 철† · 박 장 한††


## 요 약

객체 지향 프로그램의 특성인 클래스명, 멤버함수, 데이터 멤버 등을 추출하여, 프로그램 이해와 유지보수를 위한 프로그램 분석 지원 자동화 도구가 필요하다. 본 논문에서는 객체 지향 소프트웨어의 효과적인 유지보수를 위한 자동화된 도구를 설계 및 구현한다. 이 방법은 역공학의 개념을 바탕으로 C++ 원시 코드에서 클래스 관계성 정보를 추출하여, 객체 지향 표준화 방법론인 UML의 클래스 다이어그램으로 모델을 생성한다. 또한 분석 및 클래스 정보를 저장하여 클래스들의 정의 및 상속 관계, 집합 관계, 단순 참조 관계 등의 관계성 정보를 시각화하여 개발자에게 제시한다. 따라서 본 논문에서는 분석 단계에서 클래스 관계성 정보를 테이블로 구성하고, 테이블 구성 형태를 링크 형태로 표시한 방법을 제시하여 개발자에게 프로그램의 이해와 유지보수를 효율적으로 수행할 수 있도록 하였으며, 또한 객체 지향 모델에서의 클래스 재구성과 재사용이 가능하도록 하였다.


# The Design and Implementation of Class Relation Information Tool from C++ Code

Duk-Chul Jang† · Chang-Han Park††


## ABSTRACT

Automation tools for program analysis are needed in order to program understand and maintain, extract the characteristics of object-oriented program such as class name, member function and data member. In this paper, we carried out design and implementation of the automation tool for effective maintenance of object-oriented software. Being based on Reverse Engineering, this approach extracts class relationship information from C++ source code and generates object-oriented model of class diagram using UML as the standard object-oriented methodology. Therefore, this paper provides developers visualized including class information, definitions of classes, inheritance relationships, set relationships, and simple reference relationships. Finally in this paper, we propose a method that construct class relationship information to table in analysis state and make form of table construction to link form so that developers can perform understanding and maintaining program efficiently. And this method enable to restructure and reuse in object-oriented model.


## 1. 서 론

최근 객체 지향 방법론의 영향으로 객체의 효율적인 관리와 사용 방법에 관한 관심이 높아지고 있으며, Booch, OMT, Jacobson 등의 객체 지향 방법론이 등장하였다. 또한 이들의 장점들을 통합한 객체 지향 분석 및 설계 방법론인 UML(Unified Modeling Language)이 등장함에 따라 객체 지향 소프트웨어를 개발하는데 필요한 많은 객체들이 요구되고 있다[1].

본 논문에서 제시하는 방법은 객체 지향 프로그램의 이해와 유지보수를 도와주기 위해 C++ 코드로부터 객체 지향 모델의 UML 클래스 다이어그램을 생성하고, 나아가 클래스 재구성(restructuring)과 재사용(reusa-

✱ 이 논문은 1999년도 광운대학교 교내학술연구비에 의하여 연구되었음.
† 정 회 원 : 광운대학교 컴퓨터과학과 교수
†† 준 회 원 : 삼육의명대학 전산정보과 겸임교수
논문접수 : 1999년 7월 8일, 심사완료 : 2000년 2월 22일

bility)에도 응용할 수 있는 자동화된 도구(UCDT : UML Class Diagram Tool)를 개발하였다. UCDT는 역공학의 개념에 입각하여 C++ 코드의 클래스 정보 추출을 위해 클래스 계층 구조를 하나의 모듈 단위로 설정하고, 클래스에 관련된 사항만으로 제한하였으며, 클래스들의 관계성 정보를 정보 저장소에 저장한다. 클래스 부품 정보, 상속 관계, 계층도의 정보 등으로 객체의 이해력을 높이고, 객체 생성을 위한 프로토타입을 지원한다. 또한 클래스를 삽입할 경우에 상위 클래스로부터 하위 클래스간의 상속 및 참조 관계를 고려하여 일관성 있게 확장할 수 있도록 하며, 윈도우상에서 각 정보를 볼 수 있도록 함으로써 개발자에게 프로그램의 이해와 소프트웨어 재사용 및 유지보수를 효율적으로 할 수 있도록 하고자 한다.

본 논문의 구성의 다음과 같다. 2장에서는 관련연구인 UML 및 역공학에 대해 살펴보고, 3장에서는 클래스 다이어그램 생성 도구 설계의 클래스 관계성 정보 추출을 위한 알고리즘을 제시하고, 4장에서는 클래스 다이어그램 생성 도구의 적용 사례와 분석을 보이며, 끝으로 5장에서는 결론을 제시한다.

## 2. 관련 연구

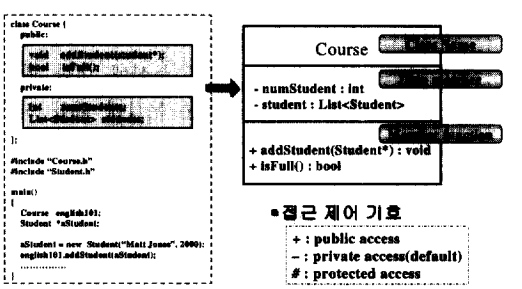
본 장에서는 표준화된 객체 지향 모델링 방법인 UML(Unified Modeling Language)과 역공학(Reverse Engineering)에 대한 관련 연구를 살펴보고, 그 문제점들을 제시한다.

### 2.1 UML의 정의 및 기본 개념

현재 객체 지향 모델링 방법론으로 많이 부각되고 있는 UML 기법은 기존의 객체 지향 모델링 방법론인 Booch, OMT, Jacobson 방법론의 장점들을 통합한 표준 객체 지향 모델링 언어이며, 객체 지향 분석 및 설계 방법이다[1, 2, 4, 9]. 그러나 UML 에서는 UML 다이어그램간의 관계에 대한 표현이 없어 좀 더 유연하고 신뢰도 높은 모델을 구축하지 못하고 있으며, 기존의 UML도구들은 주로 다이어그램의 표현을 위한 편집(editor)의 기능에 비중을 두고 있어서 객체 지향 분석 및 설계를 위한 객체 식별, 추상화, 집단화의 작업은 지원되지 않기 때문에 분석을 지원하는 기능의 필요성이 있다.

본 논문에서 제시하는 클래스 다이어그램은 객체 지향 분석 및 설계의 가장 중요한 요소로서 시스템 내부 클래스들간의 정적인 구조를 나타낸다. 클래스들은 서로간의 관계를 가질 수 있는데, 이러한 클래스들간의 관계는 속성이나 연산과 같은 클래스의 내부구조와 함께 정적 구조인 클래스 다이어그램에 나타나게 된다[11]. 또한 모든 클래스는 하나의 스테레오타입(stereotype)을 가질 수 있으며, Boundary, Entity, Control, Exception, Metaclass, Parameterized, Utility 클래스 등이고, 클래스의 이름 표기 부분에 "≪≫" 사이로 표시한다. 또한 각 객체들은 다른 객체와 서로 상호 작용함으로써 시스템의 행위를 나타내는 관계가 필요하며, 분석할 때 중요한 관계로는 연관화(association)와 집단화(aggregation) 등이 있다. 연관된 클래스는 객체 사이에 연결을 의미하며, 클래스 다이어그램에서 연관된 클래스를 연결하는 선(line)으로 표시한다. 데이터는 연결을 통하여 단방향 또는 양방향으로 표시하며, 연관은 양방향성(bi-directional) 관계이다. 집단화는 하나의 전체가 부분과 관련되는 연관화의 특별한 형태로 포함관계(part-of)이며, 집합체(전체)를 나타내는 클래스 옆에 다이아몬드 모양으로 표시한다[1, 5, 11]. 또한 C++을 클래스 다이어그램으로 표기할 때, "class" 키워드로 클래스를 정의하며, 속성과 관계를 구현한 데이터 멤버(data member), 연산을 구현한 멤버 함수(member function)로 클래스를 구성한다. 즉, 2개의 주요 구역으로 구분하면 클래스의 데이터 멤버와 멤버 함수를 기술하는 클래스 선언부(class declaration), 클래스 멤버 함수(class member function)의 구현 방법을 기술하는 클래스 정의부(class definition)로 구분된다[4, 8, 12]. 클래스 멤버에 대한 접근은 다음과 같으며, (그림 1)과 같이 클래스 다이어그램을 표기한다.

- public : 모든 클라이언트가 접근 가능(기호 → + )
- private : 클래스 자신만 접근 가능(기호 → - )
- protected : 모든 서브클래스에 대해서만 접근 가능(기호 → # )




(그림 1) 클래스 다이어그램의 예

### 2.2 역공학의 정의 및 기본 개념

역공학 용어는 하드웨어 분야에서 완성된 제품으로부터 설계 사양(design specification)을 판독하는 하드웨어 분석에 그 기원을 두며, 제품의 질적 개선을 위해 정기적으로 할뿐만 아니라, 경쟁사의 제품을 분석하는데 적용되기도 한다. M. G. Rekoff는 역공학을 "시스템의 표본(specimens)을 정기적으로 검사함으로써 복잡한 하드웨어 시스템에 대한 일련의 명세서를 개발하는 과정"이라고 정의하였다[21]. 이러한 과정은 그 시스템의 개발자가 아닌 다른 개발자에 의해 진행된다. 또한, 이러한 개념을 소프트웨어 시스템에 적용함으로써, 시스템에 대한 기본적인 개념과 그 시스템의 구조를 이해할 수 있는 다양한 접근 방법을 발견할 수 있다. 하드웨어에 있어서 역공학의 목표는 그 시스템을 복제하는데 있으나, 소프트웨어에 대한 역공학은 시스템의 개선, 유지보수 지원, 확장성 증대 또는 대체 지원을 위해 설계수준을 충분히 이해하도록 하는데 그 목표를 둔다[6, 9, 16].

재사용의 용이성은 소프트웨어 재사용성으로써, 중요한 문제는 대 규모의 소프트웨어 부품들을 효율적으로 관리하는 것이다. 역공학은 현존 시스템으로부터 재사용 가능한 소프트웨어 부품을 추출하는데 도움을 제공한다[9, 16].

본 논문에서는 역공학 개념에 입각하여 C++ 원시 코드의 관계성 정보를 추출한다. 객체에 대한 정보를 클래스명, 멤버함수, 데이터멤버 등의 추출된 정보는 정보 저장소에 저장하고, 각 클래스에 대한 상속 관계 정보를 UCDT로 실험하였다. 또한 UCDT 기능은 새로운 클래스를 생성, 삭제, 삽입할 수 있도록 하고, 이때에 클래스의 유사성을 판별하여 클래스 부품에 대한 혼합된 정보를 재구성할 수 있도록 지원하였다.

### 2.3 유지보수와 관련된 도구

본 절에서는 본 연구와 관련이 있는 유지보수 연구 중, 특히 대표적인 구조적 방법론 분석에 대한 연구, 구조적 방법과 객체 지향적 방법론 분석에 대한 연구, 그리고 국외의 리엔지니어링 관련 도구들을 비교 분석하였다.

C Information Abstraction System(CIA)은 전형적인 C 프로그램의 구조를 분석하여 이해를 돕는 시스템이다. 즉, 객체 지향 방법으로 작성된 프로그램이 아니라, 기존의 방법으로 작성된 C 프로그램의 구조를 분석하기 위한 시스템이다[15]. CIA 시스템은 대규모 C 프로그램의 구조를 분석하는데 아주 유용하다. 그러나, 교차 참조에 대한 정보는 미비한 실정이다. 또한, 사용자 편의 제공에 미흡한 면도 있다. 본 논문에서는 C 언어가 아닌 C++ 언어로 작성된 프로그램을 대상으로 하며, 구조적 방법이 아닌 객체 지향 방법에 의한 소프트웨어를 대상으로 함으로 CIA와는 직접적인 비교가 될 수 없다. 하지만, 소프트웨어의 전반적인 구조와 설계 정보를 원시 코드로부터 분석하여 추출한다는 공통점이 존재한다.

Information Viewer는 소프트웨어 개발 중 작성된 모든 문서(요구 분석 명세서, 시스템 설계서, 원시 코드 등)로부터 소프트웨어 유지보수에 필요한 정보를 추출하여 소프트웨어 베이스에 저장하고, 유지보수자가 필요한 정보를 필요한 시기에 용이하게 찾아볼 수 있도록 도와주는 시스템이다. 이 시스템은 유지보수자가 개발된 소프트웨어를 이해하는데 도움을 줄 수 있을 뿐만 아니라 소프트웨어 베이스의 활용을 통한 재사용을 지원한다. 또한, 객체와 객체간의 관계를 분석함으로써 소프트웨어 시스템에서 사용되지 않는 부분(dead code)을 찾아내고, 소프트웨어의 객체 사이의 결합도(coupling)를 추정하는데 활용될 수 있다[18]. 본 논문에서 제시한 도구와는 달리 Information Viewer는 코드 수준의 정보 생성과 상속성 등을 보여주는 것이 주된 목적이다. 따라서 Information Viewer도구에서는 객체지향 분석 및 설계 과정에서 나타나는 클래스 및 상속성, 구성 관계, 속성과 연산, 통신, 메시지 등과 같은 다양한 관계 정보를 쉽게 파악하기 어렵다. 즉, 완전한 객체 지향 모델 정보를 얻어낼 수 없다.

현재 객체 지향에 관한 객체 지향 리엔지니어링 모델링 도구는 Stp/OMT, Rational Rose/C++ 등을 찾아볼 수 있으나 객체 지향 프로그램의 이해에 도움을 주는 상용화된 CASE 도구는 많지 않다. 대부분의 객체 지향 관련 도구들은 객체 지향 소프트웨어의 개발, 즉 순공학(forward engineering) 측면에서는 다이어그램, 분석, 설계 저장소 등 다양한 기능을 갖추고 있다. 그러나 역공학을 위한 기능은 DECODE 시스템[10]과 같이 COBOL Division에서 SQL 자료 정의를 뽑아낸다든지 절차 중심의 재래식 프로그램에서 객체를 찾아내어 자동적으로 모델링 해주는 정도로 아직 미비하고, 코드의 변환, 다이어그램 생성 정도에 불과하며, 프로그램 이해를 지원할 목적으로 CASE 도구에서 기능을

제공하는 클래스 관계성 정보가 부족하다.

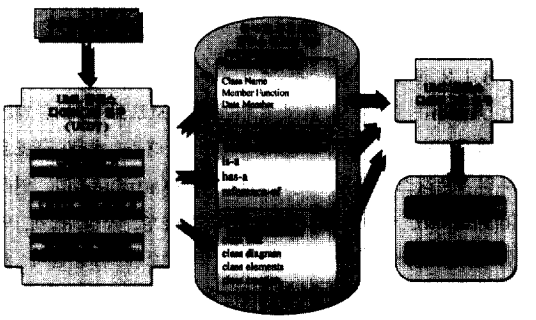
## 3. 클래스 다이어그램 생성 도구의 설계

본 장에서는 C++ 코드의 클래스에 관련된 정보를 추출하기 위한 기능을 갖는 역공학 엔진과 UCDT에 대한 클래스 및 상속성, 구성 관계, 속성과 연산, 통신, 메시지 등의 기능과 전체적인 구조, 각 추출 정보와 알고리즘을 제시하고, 각 추출된 정보의 테이블과 이를 통한 재구성, 프로그램의 이해와 유지보수의 효율적인 관리를 위해 UCDT를 설계 및 구현하고자 한다.

UCDT는 C++ 코드를 입력받아 분석한 후 이 정보들을 일관된 형태로 클래스 정보 저장소에 저장한다. 여기에는 C++ 원시 코드로부터 추출한 클래스 및 상속성, 구성 관계, 속성과 연산, 통신, 메시지 등의 정보가 저장되어 있으며, 이러한 정보 저장소의 특성을 이용하여 보다 효율적인 C++ 프로그램으로 재구성할 수 있다. 또한 클래스의 삽입과 삭제, 수정, 클래스들 간의 관계성 정보를 이용하여 새로운 C++ 코드로 재설정할 수 있도록 하며, UML의 클래스 다이어그램으로 변환함으로써 프로그램의 이해와 유지보수를 지원하는 기능을 제공하고자 한다.

### 3.1 클래스 정보 추출 시스템의 구성

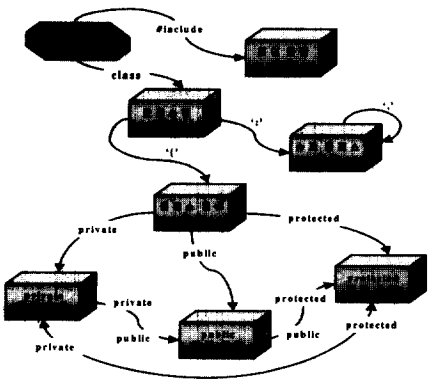
본 UCDT는 C++ 코드를 입력받아 UCDT에서 클래스 관계성 정보와 속성/연산 정보를 추출 후 정보 저장소에 저장하고, UML의 클래스 다이어그램으로 클래스의 관계성을 시각화한다. 또한 변경된 C++ 원시 코드인 경우 이를 분석하여 정보를 정보 저장소에 저장하고 관계성 정보를 재설정함으로써 보다 쉽게 프로그램의 이해와 유지보수에 도움을 주고자 한다. 패키지 다이어그램은 클래스 and/or 다른 패키지의 논리적인 모임이고, 하나의 패키지에 있는 클래스가 다른 패키지의 클래스와 대화한다면, 패키지 단계에서 의존 관계가 존재한다. 각 클래스 관계성의 유사성을 판별하여 패키지 다이어그램으로 만들며, (그림 2)와 같다. 또한, 본 논문에서 제시한 알고리즘을 통해 클래스를 분석 및 설계, 관계성 정보를 (그림 2)와 같이 클래스 계층 구조로부터 분석하여 일반화 관계인 상속 계층 구조, 집단화 관계인 포함 관계, 클래스의 사용 관계, 클래스에 정의된 멤버 함수, 데이터 멤버, 중복된 함수, 가상 함수, 재 정의된 멤버 함수, 프랜드 관계 정보, 상속을 받아 재정의 되는 함수 등을 추출하여 정보를 정보 저장소에 저장하고, UML의 Use Case 다이어그램, Class 다이어그램, 활동(behavior) 다이어그램, 구현 다이어그램중에서 본 논문에서는 클래스를 표현을 위한 클래스 다이어그램으로 시각화한다. 또한 클래스의 유사성을 검사하여 패키지 다이어그램으로 시각화하는 구성도이다. 그리고 클래스의 관계성 정보가 변경될 때 추출된 클래스 정보를 다시 정보 저장소에 저장한다.




(그림 2) UCDT의 클래스 정보 추출 시스템 구성

#### 3.1.1 클래스 정보 추출을 위한 역공학 엔진

역공학 엔진의 입력 자료로 사용되는 C++ 코드에 오류가 없다는 것을 가정함으로써 원시 코드에 대한 컴파일 오류는 검출하지 않도록 했으며, 클래스에 관련된 사항만으로 제한했다. (그림 3)은 C++ 원시 코드를 입력으로 받아 이를 분석하여 클래스 계층 구조적 특성과 기능적 특성을 추상화하기 위해 헤더파일, 선언된 클래스 이름, 수퍼 클래스 및 클래스 내부에 선언된 클래스 멤버(public, protected, private)들을 추출 정보로 정의하도록 하였다. 또한 클래스 멤버 추출에서의 private 멤버 추출, public 멤버 추출과 protected 멤버 추출은 각각 멤버 추출에서 사용될 때 private, public, protected 인지를 판별하게 된다. 즉, private 멤버 추출에서의 public 멤버 추출은 public으로 사용되고, protected 멤버 추출은 protected로 사용됨을 알 수 있으며, public 멤버 추출에서의 private 멤버 추출은 private으로 사용되고, protected 멤버 추출은 protected로 사용됨을 알 수 있다.




(그림 3) 역공학 엔진의 상태 전이도

3.1.2 클래스 정보 추출

UCDT는 (그림 4)에서 C++ 예제 코드로부터 클래스 관계성 정보를 추출하는데, 이는 클래스 계층 구조로부터 클래스 이름, 클래스의 속성(attribute)과 연산(operation), 클래스들 간의 관계(상속성, 속성 관계, 연산 관계 등)들을 추출하여 정보 저장소에 저장한다. 클래스는 공통의 특성과 공통의 행위, 다른 객체에 대한 공통의 관계, 공통의 의미를 갖는 객체들의 집합이다. 또한 클래스의 이름과 명세는 모델사전에 기반을 두며, "무엇"에 관심을 갖고 "어떻게"에 대해서는 무시한다. 모델사전의 예로는 이름 : 학생정보 → 정의 : 대학에서 수업에 참석하기 위해 등록한 학생에 대한 정보, 이름 : 과목 → 정의 : 대학에서 개설된 수업으로 들 수 있으며, 새로운 클래스를 모델사전에 추가할 수 있다.

```
1  class class_Asset {
2  public:
3          class_Asset() {}
4          ~class_Asset() {}
5  protected:
6          void get_next_object_ID();
7          int  object_ID;
8  };
9  void class_Asset::get_next_object_ID()
10 { · · · · · };

13 class InterestBeaning {
14 public:
15         InterestBeaning() {}
16         ~InterestBeaning() {}
17         virtual void GetObject_ID() = 0;
18 private:
19         char* name;
20         char* address;
21 };
22 class InsurableItem:public virtual class_Asset {
23 public:
24         InsurableItem() {}
25         ~InsurableItem() {}
26         void calculateTotal();
27 private:
28         class_Asset* asset;
29         int quantity;
30 };
31 void InsurableItem::calculateTotal()
32 { · · · · · };

35 class RealEstate:public virtual class_Asset {
36 public:
37         RealEstate() {}
38         ~RealEstate() {}
39         virtual float calculateTotal() = 0;
40 private:
41         int date;
42         int time;
43         InterestBeaning* interestBeaning;
44 };
45 class BankAccount:public InsurableItem,
                   public InterestBeaning {
46 public:
47         BankAccount() {}
48         ~BankAccount() {}
49 protected:
50         char* methodOfPayment;
51 };
52 class Savings:public BankAccount {
53 public:
54         Savings() {}
55         ~Savings() {}
56         void print();
57 private:
58         int price;
59         double total_price;
60 };
61 class Security:public InterestBeaning {
62 public:
63         Security() {}
64         ~Security() {}
65         Security(char*, char*);
66 private:
67         char* userID;
68         char* passwd;
69 };
70 class Stock:public Security {
71 public:
72         Stock() {}
73         ~Stock() {}
74         Stock(char& Uid, char& Upsw);
75         virtual float calculateTotal() = 0;
76 private:
77         float total_Stock;
78 };
79 class Bond:public Security {
80 public:
81         Bond() {}
82         ~Bond() {}
83 private:
84         int serving_hours;
85 };
```

(그림 4) C++ 원시 코드 예제 프로그램

3.2.1 헤더 파일 정보 추출

C++ 코드에서 헤더 파일의 정보를 추출하여, 클래스가 헤더 파일에서 포함된지를 체크한다. 또한 헤더 파일의 선언은 "#include"라는 선언자에 의해 선언되며 '<' 와 '>', '"'와 '"' 사이 또는 단순하게 헤더 파일 이름만으로 정의된다. 헤더 파일 추출을 위한 알고리즘은 (그림 5)과 같다.

```
void HeaderExtractor(pBuf)
{
    class_Buffer* pBuf; // 임시 저장소
    i ← 0;
    if ( pBuf == '<' || '"' ) // 기호를 판별
        getchar;
    while( pBuf != '>' && '"' && ' ') {
        nameBuff[i] ← getchar; // 읽어 들인 기호를 저장
        i ← i++; }
    nameBuff[pBuf] ← '\n'; // 문장의 끝을 찾았을 때 저장
    nameBuff[pBuf] ← ' '; // 공백을 찾았을 때 저장
    while ( nameBuff != NULL ) {
        // 헤더 파일에서 추출된 정보를 테이블에 저장
        Add_to_table(pBuf, nameBuff[pBuf]); }
}
```

(그림 5) 헤더 파일 정보 추출 알고리즘

3.1.3 클래스명 정보 추출

클래스명 정보를 추출하기 위해서는 리터럴 스트링과 입력버퍼내의 현재 토큰간의 비교기능에 의해 "class"가 인식되는 경우에만 클래스를 추출하며, 문법 구조에 의하면 클래스 선언은 클래스의 헤드 부분과 클래스 멤버로 분리될 수 있다. 클래스 헤드 부분은 class key, 클래스 이름, 부모 클래스(parent class)선언으로 구성되며, 클래스 멤버는 "{"와 "}" 사이에서 블록 구조로 선언되고 클래스의 끝은 ";" 문자가 반드시 표시되어야 한다. C++ 언어는 다중상속성을 제공하며 부모 클래스는 ','에 의해 분리되며, 클래스 정의시 ':' 문자와 '{' 사이에 위치하므로 부모 클래스의 추출은 입력버퍼로부터 ':' 문자를 인식하면서 시작되어 '{'를 인식할 때까지 계속 진행된다.

클래스명 정보 추출 테이블은 클래스 계층 구조에 대한 상속 정보 구성으로, 부모 클래스, 가상 클래스, 자식 클래스 등의 추출 정보 테이블이며, <표 1>과 같이 정의되어 있고, ☒ 표시는 클래스의 정보가 없음을 의미한다.

[정의 1]. ClassInformation(Class) = <Parent class, Virtual class, Child class>

〈표 1〉 클래스명 정보 추출 테이블

| Class Name | Parent Class | Virtual Class | Child Class |
|---|---|---|---|
| class_Asset | ☒ | ☒ | RealEstate, InsurableItem |
| InterestBeaning | ☒ | ☒ | BankAccount, Security |
| InsurableItem | class_Asset | class_Asset | BankAccount |
| RealEstate | class_Asset | class_Asset | ☒ |
| BankAccount | InsurableItem, InterestBeaing | ☒ | Savings |
| Savings | BankAccount | ☒ | ☒ |
| Security | InterestBeaning | ☒ | Stock, Bond |
| Stock | Security | ☒ | ☒ |
| Bond | Security | ☒ | ☒ |

또한, C++ 프로그램에서 클래스는 예약어 "class"라는 선언자에 의해 선언되므로 클래스의 상위 클래스명, 클래스의 하위 클래스명, 클래스들 간의 접근 지정자(public, protected, private)를 추출한다. (그림 6)은 클래스명 정보 추출 알고리즘이다.

```
void ClassNameExtractor(pBuf)
{
    class_access_mode* parent, virtual, child;
    while( pBuf != NULL ) {
        // 클래스명을 임시 기억장소에 저장
        p ← search_ClassName->pBuf;
        while( p != NULL ) {
            p ← search(p, key);
            // 라인의 끝인 경우 함수 호출
            if(p->lastchar == '\n') getNextLine;
            // 수퍼 클래스인 경우 함수 호출
            if(p->lastchar == ':') call SuperClass;
            // 멤버 함수인 경우 함수 호출
            if(p->lastchar == '{') call MemberList; }
    }
    // 클래스 정보 테이블
    Add_to_table(pBuf, class_table);
    // 부모 클래스를 테이블에 저장
    Add_to_table(pBuf, p, parent(p), class_table);
    . . . . .
}
```

(그림 6) 클래스명 정보 추출 알고리즘

3.3.2 속성/연산 정보 추출

속성의 정보 추출은 클래스 정의의 멤버 선언 중 변수 선언에서 추출하고, 연산의 정보 추출은 클래스 정의의 멤버 선언 중 생성자(constructor), 소멸자(destructor) 및 함수 선언에서 추출한다. 또한, 속성/연산 정보 추출 테이블은 클래스명, 접근지정자, 멤버 함수와 데이터 멤버, 자료형, 가상 함수, 생성자, 인자값, 링크 등의 추출 정보 테이블이며, <표 2>와 같이 정의되어 있고, ☑ 표시는 가상 함수와 생성자의 정보를 의미한다.

〈표 2〉 속성/연산 정보 추출 테이블

| No | ClassName | A_m | M_f&D_m | type | v_f | con | Argu | link |
|---|---|---|---|---|---|---|---|---|
| 1 | class_Asset | + | class_Asset() | | | ☑ | void | |
| 2 | | # | get_next_object_ID() | void | | | void | |
| 3 | | # | object_ID | int | | | | |
| 4 | InterestBearing | + | InterestBearing() | | | ☑ | void | |
| 5 | | + | GetObject_ID() | void | ☑ | | | |
| 6 | | - | name | char* | | | | |
| 7 | | - | address | char* | | | | |
| 8 | InsurableItem | + | InsurableItem() | | | ☑ | void | |
| 9 | | + | calculateTotal() | void | | | void | |
| 10 | | - | asset | class_Asset | | | | |
| 11 | | - | quantity | int | | | | |
| 12 | RealEstate | + | RealEstate() | | | ☑ | void | |
| 13 | | + | calculateTotal() | float | ☑ | | | 9 |
| 14 | | - | date | int | | | | |
| 15 | | - | time | int | | | | |
| 16 | | - | interestBearing | InterestBearing | | | | 4 |
| 17 | BankAccount | + | BankAccount() | | | ☑ | void | |
| 18 | | # | metodOfPayment | char* | | | | |
| 19 | Savings | + | Savings() | | | ☑ | void | |
| 20 | | + | print() | void | | | void | |
| 21 | | - | price | int | | | | |
| 22 | | - | total_price | double | | | | |
| 23 | Security | + | Security() | | | ☑ | void | |
| 24 | | + | Security() | | | | char*, char* | 24 |
| 25 | | - | userID | char* | | | | |
| 26 | | - | passwd | char* | | | | |
| 27 | Stock | + | Stock() | | | ☑ | void | |
| 28 | | + | Stock() | | | | char&, char& | 27 |
| 29 | | + | calculateTotal() | float | ☑ | | | 9,13 |
| 30 | | - | tot_Stock | float | | | | |
| 31 | Bond | + | Bond() | | | ☑ | void | |
| 32 | | - | serving_hours | int | | | | |

[정의 2]. Attribute/Operation<Class, M_f&D_m> = <A_m, type, v_f, cons, Argu, link>

ClassName : 클래스 이름

A_m : Access mode(- : private, + : public, # : protected)

M_f&D_m : Member function & Data member

type : Data type(value = type, method = return type )

v_f : virtual function

cons : constructor

Argu : Argument

link : 클래스 멤버 함수에서 사용되는 멤버 함수 및 데이터 관계

선언된 클래스의 멤버는 정보의 은폐 수준에 따라 public, private, protected 등으로 구분하며, 데이터 멤버는 형(type), 데이터 멤버 이름으로 구성되며 다음 데이터 멤버와는 ','로 분리된다. 멤버 함수는 형(type), 함수 이름, 함수의 매개변수를 정의하기 위한 '()', 멤버 함수의 상세 기능을 위한 '{}' 블록으로 구성되며, (그림 7)은 속성/연산 정보 추출 알고리즘이다.

```
void ClassAttributeOperationExtractor(pBuf)
{
    class_Buffer* pBuf;
    access_mode* pub, pro, pri; // 클래스 접근 지정자
    while( pBuf != NULL ) {
        // 클래스의 접근 지정자와 속성을 검색
        p ← search(pub, pro, pri, ClassAttributes->pBuf);
        // 클래스 속성 정보 테이블
        Add_to_table(p, Attribute_table); }
    access mode* void, generic, virtual, inline, friend;
    while( pBuf != NULL ) {
        // 임시 저장소에서 멤버 함수의 형을 검색
        p ← search(void, generic, virtual, inline, friend,
                ClassOperations->pBuf);
        // 클래스 연산 정보 테이블
        Add_to_table(p, Operation_table); }
    // 클래스의 속성과 연산 테이블에 추가
    Add_to_table(p, ClassAttributeOperation_Table);
}
```

(그림 7) 속성/연산 정보 추출 알고리즘

3.1.5 클래스 관계성 정보 추출

클래스 관계성 정보 추출은 상속 관계, 구성 관계, 연결 관계 추출과 부 시스템 추출을 중심으로 하며, 상속 관계는 클래스 정의의 부모 클래스(parent class) 선언에서 추출, 구성 관계는 클래스 정의에서의 클래스 변수 선언, 멤버 함수 정의에서의 클래스 변수 선언, 멤버 함수 정의에서 new 문장을 통한 동적 할당(dynamic allocation)의 해당 경우에 추출, 연결 관계는 외부에서 선언된 클래스를 함수 파라미터로 전달받아 사용하는 경우나 프랜드(friend) 클래스 선언에서 추출, 부 시스템 추출은 상속 및 집합 관계에 의한 클래스 그룹화를 통하여 추출할 수 있다.

클래스 관계성 정보 추출 테이블은 클래스명, 상속성 링크, 포함성 링크, 참조성 링크 등의 정보 저장 테이블이다. <표 3>과 같이 정의되어 있고, ☒ 표시는 해당 클래스와 관계가 없음을 의미한다.

[정의 3]. ClassRelationship(Class) = <Class Name, is-a_link, has-a_link, reference-of_link>

〈표 3〉 클래스 관계성 정보 추출 테이블

| NO | Class Name | is-a_link | has-a_link | reference-of_link |
|---|---|---|---|---|
| 1 | class_Asset | ☒ | ☒ | ☒ |
| 2 | InterestBeaning | ☒ | ☒ | ☒ |
| 3 | InsurableItem | 1 | 1 | 1 |
| 4 | RealEstate | 1 | 1 | 1 |
| 5 | BankAccount | 2 | ☒ | ☒ |
| 6 | Savings | 3 | ☒ | ☒ |
| 7 | Security | 2 | ☒ | ☒ |
| 8 | Stock | 7 | ☒ | 7 |
| 9 | Bond | 7 | ☒ | ☒ |

또한, 클래스 계층 구조로부터 관계성 정보 추출을 위해 헤더파일 정보 추출, 클래스명 정보 추출, 속성/연산 정보 추출, 모듈 인터페이스 분석 모듈을 통해 상속성, 속성 관계, 연산 관계, 복합, 참조성을 구분하여 테이블을 생성한다. (그림 8)는 클래스 관계성 정보 추출 알고리즘이다. 검색된 모듈 인터페이스 분석 모듈은 C++ 원시 코드의 모든 클래스에 대한 클래스 정보 추출이 완료되면 클래스를 선언할 때 상속된 클래스가 프로그램 내부에서 선언된 클래스인가 아니면 include된 헤더파일로부터 상속을 받은 것인가를 검사하기 위해 클래스 정보 추출 테이블로부터 링크 정보를 검색하여 테이블에 저장한다.

```
void ClassRelationshipInformation(pBuf)
{
    class_Buffer* pBuf;
    class_Relationship* is_a, has_a, referenct_of;
       // 클래스 관계성 정보
    while( pBuf != NULL ) {
       Add_to_table(pBuf, Super(pBuf), class_table);
       // 가상 클래스 정보 테이블, 자식 클래스 정보 테이블
       Add_to_table(pBuf(is_a, has_a, referenct_of),
                    virtual(pBuf), class_table);
       Add_to_table(pBuf(is_a, has_a, referenct_of),
                    child(pBuf), class_table); }
    // 각 정보 테이블에서 관계성 정보를 추가
    Add_to_table(pBuf, class_table(is_a, has_a, reference_of));
}
void ClassReferenceInformation(pBuf, m)
{
    class_Buffer* pBuf;
    class_operation* m;
    while( pBuf != NULL ) {
          class_var v, t;
          class_operation n;
          v ← Declared_variable();
          t ← type_of_variable();
          // 클래스의 관계성 정보를 테이블에 추가
          Add_to_buffer(pBuf, m, t, v, declared_buffer);
          r ← Calling_member();
          . . . . . }
}
```

(그림 8) 클래스 관계성 정보 추출 알고리즘

### 3.4 클래스의 유사성과 삽입

클래스를 삽입할 때 소프트웨어의 재사용 측면과 유사성을 고려하여야 하며, 대표적인 유사성 측정 방법은 inner-product measure, cosine measure, pseudo-cosine measure, Dice measure 방법 등이 있다[14]. 이것을 근거로 클러스터링하는 방법은 single pass, real-location, complete link, group average link, Ward 방법 등이 있지만[14], 이러한 방법은 객체 지향 언어에서 지원하는 클래스 상속 구조를 표시할 수 없다. 객체 지향 언어에서 유사성은 클래스간의 데이터 멤버와 멤버 함수의 유사성 정도를 나타내며, 유사성 정도가 큰 것을 대상으로 클러스터링하게 된다. 클러스터링하는 이유는 유사한 기능을 수행하는 클래스들을 묶고, 동일한 저장소에 두어 클래스 정보의 재사용성을 높이며 인접 기억 장소에 저장함으로써 검색 효율을 높일 수 있다[17, 19].

일반적으로 객체 지향 언어로 프로그램을 적용했을 때 프로그램 이해의 기본 단위는 객체나 클래스이지만, 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.

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 상속으로 인한 참조가 가능하나 객체 참조인 경우에는 명시적으로 표현한다. 따라서 헤더파일 정보 추출, 클래스명 정보 추출, 속성/연산 정보 추출, 모듈 인터페이스 분석 모듈 테이블과 클래스 관계성 정보 추출 테이블, 상속 관계 참조 테이블을 조인하여 사용함으로써 삽입 시에 일관성 있으며 확장이 용이하다.

이와 관련된 클래스 객체들의 관계성을 위한 형식 정의(formal definition)에 관한 연구는 상대적으로 미흡하며, 그 이유는 객체 지향 기법의 발전이 수학적 기반보다는 경험적 원리를 기반으로 하고 있기 때문이다. 형식 정의는 Wand에 의하여 객체 지향 시스템에 적합하도록 정의되었다[7]. 그리고 클래스들의 유사성(similarity)이란 두 개의 클래스 사이에서 동일한 도메인(데이터 멤버 + 멤버 함수)이 많을수록 클래스간의 유사성 값은 결과로 설계에서 상속 계층 구조 설정과 구현할 때 코드의 재사용 측면을 결정하게 된다[13, 17].

본 논문에서는 기존의 연구 결과를 기반으로 정의한다[17].

[기호 정의]

- *d*(domain) : 멤버 함수와 데이터 멤버
- *mf* : 멤버 함수
- *dm* : 데이터 멤버

[정의4] 상속 계층 관계에는 일반화(generalization)와 세분화(specialization) 관계로 나누어진다.

<조건1> 일반화는 상위 클래스 $Ci$의 도메인(domain)이 하위 클래스 $Cj$의 부분 집합이라면 $Ci$는 $Cj$의 일반화라 한다. 즉, $(Ci \supset Cj)$의 조건을 만족한다.

<조건2> 세분화는 일반화의 역으로 $(Ci \subset Cj)$가 성립하면 하위 클래스 $Cj$는 클래스 $Ci$의 세분화라고 한다. 즉, $(Ci \subset Cj)$의 조건을 만족한다.

[정의5] 클래스의 데이터 멤버 유사성

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

[정의6] 클래스의 멤버 함수 유사성

$$SIM_{mf}(A, B) = mf(A \cap B)^2 / (mf(A) \times mf(B))$$

[정의7] 클래스들의 유사성

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

[정의 5]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 6] 에서도 [정의 5]와 같은 개념이지만, 여기에서는 멤버 함수($mf$)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 함수와 타입을 비교하여 측정한다. [정의 7]은 클래스들의 유사성으로 [정의 5]와 [정의 6]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. (그림 9)은 클래스 삽입과 유사성 정보 추출 알고리즘으로써 클래스 관계성 재설정은 C++ 원시 코드로부터 프로그램 이해와 유지보수를 돕는 클래스 관계성 정보 추출 테이블을 통해 변경된 클래스명, 상속성, 포함성 등의 정보를 이용하여 객체 지향 방법론인 UML의 관계성 정보 및 클래스 다이어그램 생성을 통한 클래스들의 정의 및 상속 관계, 구성 관계, 메시지 전달과 같은 클래스들 간의 관계성을 재설정 한다.

```
void ClassInsertion::ClassRelationshipInformation(CString pBuf)
{
    CClassDbSet* pcSet;
    ClassName_Buffer* pBuf; // 클래스명의 임시 기억 장소
    while( pBuf != NULL ) {
        class_ao d_m, m_f;
        Add_to_table(pcSet->pBuf, class_table);
                    // 유사성 클래스 테이블 구성
        Add_to_table(pBuf(d_m), class_table);
                    // 유사성 데이터 멤버 테이블 구성
        Add_to_table(pBuf(m_f), class_table);
                    // 유사성 멤버 함수 테이블 구성
    }
    add_Insert++;
    while( pBuf != NULL ) {
        class_hierarchy Ci;
        class_similarity a;
        class_insert b;
        currentClassSet ← NULL;
        equalClass ← false; // 유사 클래스를 초기화
        serch_Similarity_Class(Ci, a, b) // 유사 클래스를 검색
        {
            if( commonAttribute(Ci, a, b) != NULL ) {
                temporaryClassCreat();
                 // 유사성이 있는 클래스를 생성
                // 수퍼 클래스를 검색하여 테이블 구성
                Add_to_table(SuperClass->
                            temporaryClassCreat());
                // 유사성 있는 클래스를 검색 테이블 구성
                Add_to_table(EqualClass->
                            temporaryClassCreat());
                // 유사성 있는 데이터 멤버 검색 테이블 구성
                Add_to_table(DataMember->
                            temporaryClassCreat());
                // 유사성 있는 멤버 함수 검색 테이블 구성
```

```
                Add_to_table(MemberFunction->
                        temporaryClassCreat());
            }
            // 유사서을 검색한 결과를 바탕으로 검색하여 패키지
            // 다이어그램을 만들기 위한 테이블 구성
            Append_to_table(commonAttribute(a, b),
                        packageClass_table);
        }
    }
    info ← search_ClassRelationshipInformation(pcSet->pBuf);
    if(info.attribute == CLASS_Info) {
        Add_to_table(info.class_table);
                // 클래스 정보 테이블 구성
    }
    // 클래스의 속성과 연산정보 테이블에 추가
    Append_to_table(info, attribute_operation_table);
}
```

(그림 9) 클래스 삽입과 유사성 정보 추출 알고리즘

(그림 10)는 클래스 다이어그램의 시각화 알고리즘으로, 본 논문에서 제시하는 UML의 클래스 다이어그램의 유사성을 비교 분석한 후 정보 저장소로부터 정보를 검색하여 패키지 다이어그램으로 구성한다.

```
void ClassViewer::ClassRelationshpInformation(pBuf) {
    class_Buffer* pBuf;
    // 화면에 정보를 보이기 위한 작업을 생성
    infoViewer ← pBuf->Class_Display();
    infoViewer.MakeSuper(); // 클래스 틀을 생성
    // 생성된 클래스에 그래프를 그림
    infoViewer.MakeGraph(class_table);
    while( pBuf != NULL ) {
        // 테이블을 검색하여 화면에 클래스명을 표시
        infoViewer.ClassName(class_table);
        // 테이블을 검색하여 화면에 속성 정보를 표시
        infoViewer.Attributes(class_table);
        // 테이블을 검색하여 화면에 연산 정보를 표시
        infoViewer.Operations(class_table); }
    // 모든 결과를 검색하여 클래스 다이어그램 생성
    infoViewer.ClassDiagram_Display();
    // 모들 결과를 검색하여 패키지 다이어그램 생성
    infoViewer.PackagDiagram_Display();
}
```

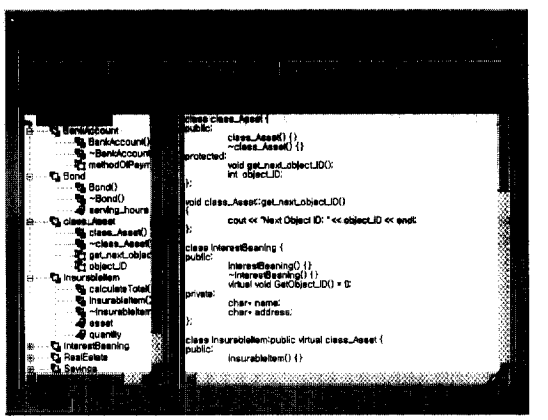
(그림 10) 클래스 다이어그램의 시각화 알고리즘

## 4. 도구의 적용 사례 및 비교 분석

이 장에서는 UCDT 구현과 C++ 원시 코드의 클래스 계층 구조로부터 클래스 정보를 분석하고, 정보 저장소에 저장하여 클래스 다이어그램으로 시각화 과정을 실행한 화면을 설명한다.

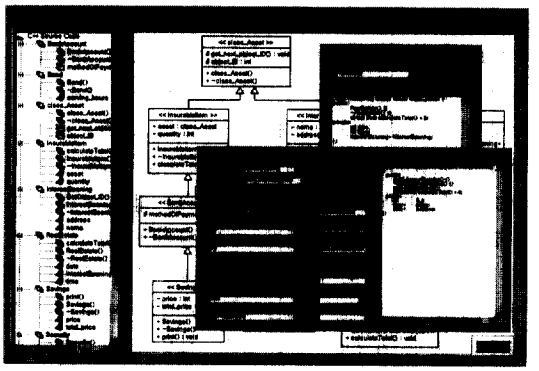
### 4.1 적용 사례

(그림 11)은 UCDT의 초기화면으로 (그림 4)의 C++ 원시 코드를 입력받아 분석하여 클래스명 정보 추출, 속성/연산 정보 추출, 클래스 관계성 정보 추출을 하고 각각 정보를 테이블로 구성한다. (그림 11)에서 원시 코드 분석 버튼을 클릭하면, 클래스명 정보 추출 테이블로 클래스명, 부모 클래스, 가상 클래스, 자식 클래스를 추출한 후 정보 테이블 생성한다. 또한 속성/연산 정보 추출 테이블로 클래스명, 접근제어, 멤버 함수와 데이터 멤버, 자료형, 가상 함수, 생성자, 인자값 등을 추출함으로써, 클래스의 속성과 연산에 해당하는 정보를 클래스 다이어그램으로 표현할 자료를 생성한다. 그리고 클래스 관계성 정보 추출 테이블은 클래스 계층 구조로부터 클래스명, 상속성, 포함성, 참조성을 추출한다.



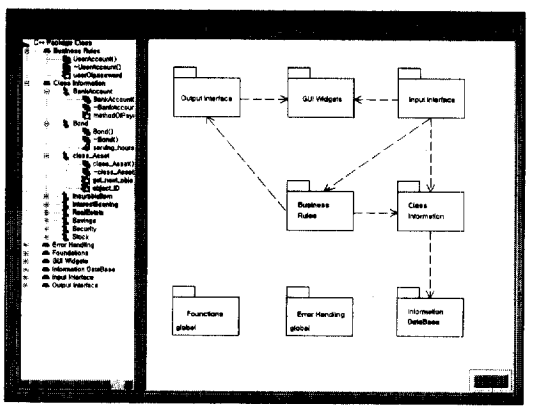
(그림 11) C++ 원시 코드 입력 화면

(그림 12)는 클래스 다이어그램 생성 도구로 C++ 원시 코드의 추출된 정보로부터 클래스 다이어그램의 관계성을 보여주고 있으며, class_Asset 클래스는 자식 클래스인 RealEstate와 InsurableItem 클래스로 상속해 주고, BankAccount는 InsurableItem과 InterestBeaning으로부터 다중 상속을 받는 것을 볼 수 있다. 또한 해당 클래스를 더블 클릭하면 클래스명과 원시 코드화면이 나타난다. 클래스 생성시 부모클래스에서 상속을 받을 것인가, 또는 새로운 클래스를 생성할 것인가를 판별하여 클래스의 프로토타입을 생성하고, 부분적으로 필요한 코드는 오른쪽화면에서 텍스트를 입력받아 생성하면 실제 코드로 변환된다.



(그림 12) C++ 원시 코드의 관계성 클래스 다이어그램

(그림 13)는 클래스의 패키지 다이어그램으로 각각의 Business Rules, Class Information, Error Handling, Foundations, GUI Widgets, Information DataBase, Input Interface, Output Interface 등으로 구분할 수 있다. 따라서 정보 관리 객체를 효율적으로 재사용하여 프로그래밍할 수 있도록 객체에 대한 정보 추출, 시각화, 클래스간의 관계가 변경될 경우 프로그램의 이해와 유지보수를 쉽게 할 수 있다.




(그림 13) 클래스의 패키지 다이어그램

4.2 비교 분석

본 UCDT는 객체 지향 분석 및 설계 과정에서 나타나는 클래스 및 상속성, 구성 관계, 속성과 연산, 통신, 메시지 등과 같은 다양한 관계성 정보를 쉽게 파악하고자 기존의 연구를 바탕으로 한 객체 지향 방법에서 프로그램 이해와 여러 가지 특성에 대해 비교하면 <표 4>와 같다. <표 4>에서 비교한 것처럼 기존 방법A·B로 나누어 각각 Information Viewer과 Stp/OMT, Rational Rose/C++를 비교한 것이다.

〈표 4〉 기존 방법과 비교

| 항목 \ 방법 | 내 용 | 기존 방법A | 기존 방법B | 제안 방법 |
|---|---|---|---|---|
| 기본 구성 요소 | 클래스 | ○ | ○ | ○ |
| 프로그램 이해 요소 (클래스 관계) | 모 듈 | ○ | ○ | ○ |
| | 상속성 | ○ | ○ | ○ |
| | 포함성 | × | × | ○ |
| | 참조성 | × | × | ○ |
| | 재구성 | × | × | ○ |
| | 재사용 | × | ○ | ○ |
| | 다형성 | × | ○ | ○ |
| | 속 성 | × | × | ○ |
| | 연 산 | × | × | ○ |
| 분석 방법 | 구조도 | ○ | × | ○ |
| | 자료 흐름도 | ○ | × | ○ |
| | 그래프 | ○ | ○ | ○ |
| 산 출 물 | 시각화 | × | × | ○ |
| | UML | × | × | ○ |

<표 5>의 기존 도구와 비교 분석하면, 클래스 관계성 정보를 역공학 개념에 입각하여 접근하고, 클래스 정보 테이블 이용한 체계적인 분석을 할 수 있다. 또한 클래스 단위 재사용과 클래스 정보 재사용으로 인한 객체가 가지고 있는 캡슐화나 정보 은폐의 특성에 따른 클래스나 클래스 상속 구조는 재사용의 단위가 되며, 여기에서 원시 코드로부터 추출한 다양한 정보를 정보 저장소에 저장하여 보다 효율적인 C++ 프로그램으로 재구성할 수 있다.

객체 지향 소프트웨어 설계 및 유지보수의 도움을 주기 위해 클래스 정보 저장소에서 클래스 관계성 정보를 효율적으로 탐색할 수 있도록 하였으며, 클래스 변경시 클래스 노드 삽입 위치를 설정하고 멤버 함수와 데이터 멤버를 일반화함으로써 프로그램 이해를 쉽게 할 수 있게 하였다.

따라서 본 논문에서 제시하는 클래스 다이어그램 생성 도구는 프로그램의 이해를 높여 유지보수에 도움을 주며, 테이블 정보를 이용하여 빠른 수정과 시스템 성능을 향상을 얻을 수 있다. 그러나 복잡한 계층 구조일 때 모든 경우를 개발자가 분석하는데 많은 시간을 필요로 한다.

〈표 5〉 기존 도구와의 비교

| | CIA | Information Viewer | 객체 지향 역공학 도구 | 본 연구에서 제안한 도구 |
|---|---|---|---|---|
| 대상언어 | C | C++ | C++ | C++ |
| 부품의 성격 | 함수 | 클래스 | 클래스 | 클래스 |
| 접근 기법 | 역공학 | 역공학 | 순·역공학 | 역공학 |
| 프로그램 분석 방법 | 특정변수 기준 프로그램 분석 | 클래스 구성요소 프로그램 분석 | 클래스 구성요소 프로그램 분석 | 클래스 정보 테이블을 이용한 체계적인 분석 |
| 재사용 방법 | 함수단위 | 클래스 단위 재사용 | 클래스 단위 재사용 | 클래스 단위 재사용, 클래스 정보 재사용 |
| 재구성 방법 | 지원 | 지원 | 지원 | 지원 |
| 검색 방법 | Keyword matching | Keyword matching | Keyword matching | 클래스 정보 테이블을 이용한 색인 및 유사성 이용 |
| Viewer | X | 정보 생성 | 다이어그램 생성 | 소스코드, 프로토타입 생성 |

## 5. 결 론

본 논문에서는 역공학 개념을 적용하여 C++ 원시 코드로부터 클래스 계층 구조를 분석한 후 객체 지향 모델을 생성하는 클래스 다이어그램 생성 도구 및 응용 분야에 대하여 논의하였다. 소프트웨어 유지 보수 과정에서 가장 중요한 단계인 프로그램 이해 과정을 돕기 위하여 클래스 다이어그램으로 생성하는 것이다. 또한 이를 바탕으로 하는 재구성 및 재사용 방법도 지원하고자 하였다.

본 논문에서 제시한 도구는 역공학에 의한 C++ 원시 코드의 클래스 관계성 정보를 이용하여 클래스 다이어그램으로 시각화할 때 얻을 수 있는 주요 장점들은 다음과 같다.

1) 클래스의 계층 구조 및 클래스간의 속성/연산 정보를 파악하여 프로그램 개발 및 유지보수의 용이성을 제공하며,
2) 클래스의 프로토타입을 생성하고, 클래스의 재구성과 재사용이 가능하며,
3) 클래스 정보 테이블을 이용한 색인 및 유사성을 이용해 프로그램의 이해에 도움을 줄 수 있다.

따라서 본 논문에서 제시한 UML의 클래스 다이어그램 생성 도구는 클래스의 관계성 및 클래스 다이어그램 생성을 통해 클래스들의 정의, 상속 관계, 구성 관계, 참조 관계, 메시지 전달과 같은 클래스들 간의 관계를 클래스 다이어그램으로 시각화하여 제시함으로써 단위 다이어그램의 요소들에 대한 일관성을 높이고, 분석의 정확성을 높일 수 있다. 또한 사용자는 방대한 양의 코드를 클래스의 관계성 정보를 이용하여 쉽게 프로그램을 이해하고 유지보수를 할 수 있게 된다. 즉, 개발자가 소프트웨어 설계자의 의도를 파악하는데 도움을 줄 수 있으며, 이를 통해 효율적인 유지보수를 할 수 있다.

앞으로의 연구 방향은 자동화된 코드 생성과 다양한 정보 분석 및 검출을 통한 재사용 가능한 컴포넌트 데이터베이스를 구축하고, 정보를 통합 CASE에 적용할 수 있도록 구현하는 연구가 필요하다.

## 참 고 문 헌

[1] Booch, G., Object-Oriented Analysis and Design with Applications 2nd Edition, Addison Wesley Longman, Inc, 1994.

[2] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison Wesley Longman, Inc, 1997.

[3] Brooks, F.P., "No Silver Bullet: Essence and Accidents of Software Engineering," IEEE Computer, Feb, 1991.

[4] Cantor, M.R., Objected-Oriented Project Management with UML, John Wiley & Sons, Inc., 1998.

[5] Coleman, D., Artim, J., Ohnjec, V., Rivas, E., Rumbaugh, J., Wirfs-Brock, R., "UML: the language of blueprints for software?," In Proceeding of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Language & Applications (OOPSLA '97), Vol.32, No.10, pp.201-205, October 1997.

[6] Murphy, G.C., Notkin, D. and Sullivan, K., "Software reflexion models: Bridging the gap between source and high-level models," In Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.18-28, Washington, D.C., October 1995.

[7] Needham, D., Demurrjian, S., Guemhioui, K.El., Peters, T., Zemani, P., McMachon, M., and Ellis, H., ADAM: A Language-Independent, Object-Oriented, Design Environment for Modeling Inheritance and Relationship Variants in Ada 95, C++, and Effel, Proceedings of 1996 TriAda Conference, Philadelphia, PA, December 1996.

[8] Pohl, I., Objected-Oriented Programming using C++ 2nd Edition, Addison Wesley Longman, Inc. 1997.

[9] Pressman, R.S., Software Engineering: A Practitioner's Approach 4th Edition, The McGraw- Hill Companies, Inc. 1997.

[10] Quilici, A., Chin, D., "DECODE: A Cooperative Environment for Reverse-Engineering Legacy Software," 2nd Working Conference on Reverse Engineering, IEEE Press, pp.217-226, July, 1995.

[11] Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual, Addison Wesley Longman, Inc, 1999.

[12] Stroustrup, B., The C++ Programming Language 3th Edition, Addison Wesley Publishing Company, Inc. 1997.

[13] Tversky, A., "Features of Similarity, Psychological Review," Vol.84, No.4, pp.327-352, Jul. 1997.

[14] Yas, A.S., Harris, D.R., and Chase, M.P., "Manipulating recovered software architecture views," In Proceedings of the 19th International Conference on Software Engineering, pp.184-194, Boston MA, May 1997.

[15] Yie-Farn, C., Nishimoto, M.Y., Ramamoorthy, C.V., "The C Information Abstraction System," IEEE Trans. on Software Engineering, Vol.16, No.3, pp.325-334, 1990.

[16] Yu, S., and Zhuang, Q.,"Software reuse via algorithmic abstraction," In Conference Series Technology of Object-Oriented Languages and Systems (Tools USA '95), pp.277-292, 1995.

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

[18] 김문회, 한재수, "객체지향 언어 C++를 위한 Information Viewer", 한국정보과학지, 제11권, 제2호, pp.84-93, 1993.

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

[20] 박진호, 김수동, 류성열, "UML 다이어그램들 간의 일관성 검증 방법", 한국정보과학회 봄 학술발 논문집, Vol.25, No.1, pp.524-526, 1998.

[21] Chikofsky, E.J., and J.H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, pp.13-17, January 1990.

**장 덕 철**

e-mail : dcchang@cs.kwangwoon.ac.kr

1982년 고려대학교 대학원 경영정보학박사

1979년 광운대학교 전자계산소 소장

1989년 광운대학교 정보 과학 연구소장

1991년 광운대학교 전산사회교육원장

1993년 광운대학교 전산대학원장

1995년 광운대학교 이과대학 학장

1977년~현재 광운대학교 컴퓨터과학과 교수

관심분야 : 소프트웨어공학(소프트웨어 유지 보수, 재공학, 역공학 및 재사용), 객체 지향 설계 방법론

**박 장 한**

e-mail : parkch@explore.kwangwoon.ac.kr

1997년 방송통신대학교 전자계산학과 독학학위취득(이학사)

1999년 광운대학교 전산대학원 소프트웨어공학과(이학석사)

2000년~현재 삼육의명대학 전산정보과 겸임교수

관심분야 : 소프트웨어공학(유지보수, 재사용, 재공학, 역공학), 분산 객체 지향 시스템, 컴포넌트 기반 소프트웨어 개발방법론, CASE