UML 클래스 다이어그램 기반의 효율적인 C++코드 생성기의 설계와 구현

(Design and Implementation of an Efficient C++ Code Generator based on UML Class Diagram)

조 형 주 † 정 진 완 †† 김 형 주 †††
(Hyung-Ju Cho) (Chin-Wan Chung) (Hyoung-Joo Kim)

요 약 UML은 OMG에 의해서 표준 객체지향 모델링 언어로 승인 받았다. 그러나, UML을 지원하는 몇몇의 CASE 도구들이 생성한 C++ 코드는 1대 n 관계나 aggregation 관계의 의미를 정확히 반영하지 못하고 있다. 또한, 상용 CASE 도구들이 하나의 UML 클래스 다이어그램에서 너무 많은 프로그래밍 언어를 지원하기 때문에, 그들은 C++언어가 가지는 특징을 효율적으로 지원하지 못하고 있다. 제안된 C++코드 생성기는 1대 n의 관계, aggregation 관계, 코드 패턴(code pattern), 디자인 패턴(design pattern)을 지원한다.

본 논문에서는 UML 클래스 다이어그램 기반의 효율적인 C++ 코드 생성기의 설계와 구현에 대하여 기술한다.

Abstract The Unified Modeling Language(UML) became the standard object oriented modeling langauge approved by Object Management Group(OMG). However, C++ codes which are generated by some CASE tools supporting UML do not reflect the correct semantics of one-to-many relationship and aggregation relationship. Additionally, since the commercial CASE tools support too many programming languages on one UML class diagram, they do not support efficiently the characteristics of C++ language. Our C++ code generator supports one-to-many relationship, aggregation relationship, code patterns and design patterns.

In this paper, we describe design and implementation of the efficient C++ code generator based on UML class diagram.

1. 서 론

C++이나 자바 같은 객체지향 프로그래밍 언어는 C나 파스칼과 같은 구조적 프로그래밍 언어에 비하여 재사용성, 모듈화, 추상화, 실세계를 잘 모델링 할 수 있다는 장점이 있지만, 구조적 프로그래밍에 비하여 개념이 어렵고, 시스템 설계가 어렵다는 단점이 있다. 객체지향 방법론은 이런 객체지향 프로그래밍 언어를 사용하여 시스템을 만들 때, 보다 효과적으로 분석하고, 설계할 수 있도록 해준다. OOCASE 도구는 객체지향 방법론을 통하여, 사용자의 요구 분석, 시스템의 설계, 시스템의 개발, 시스템의 유지보수에 필요한 문서화 작업을 도와준다. UML이 OMG에서 객체지향 모델링 언어의 표준으로 채택[1]됨으로써, CASE 도구들이 UML을 지원하기 시작했다. UML 클래스 다이어그램은 시스템에 있는 객체들의 타입과 이들 사이에 존재하는 다양한 정적인 관계를 기술한다[2]. 클래스 다이어그램은 UML에서 가장 중요한 다이어그램으로, 소스 코드 생성도 이것을 통해서 이루어진다. 그러나, 상용 CASE 도구들이 UML 클래스 다이어그램에서 C++코드를 생성할 수 있지만, 의미적으로 부적합한 경우가 있다. 예를 들면, 1대 n 관계, aggregation 관계의 경우, 적절하지 않은 C++ 코드가 생성된다. 왜냐하면, C++에서 1대 n 관계나

† 비 회 원 : 한국과학기술원 전자전산학과
hjcho@islab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
chungcw@ngis.kaist.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjkim@oopsla.snu.ac.kr
논문접수 : 1998년 11월 3일
심사완료 : 2000년 5월 22일

aggregation 관계를 직접 지원하지 않기 때문이다.

본 논문에서는 사용자가 UML 클래스 다이어그램 기반에서 설계하고, 이것을 C++코드로 직접 생성할 수 있게 함으로써, C++을 사용하여 시스템을 설계하고 구현하는 것을 돕는다. 그리고 상용 CASE 도구에서 C++코드를 생성할 때 발생하는 의미상의 부적절성을 STL(Standard Template Library) 컨테이너 클래스를 이용하여 해결하고, 나아가서 여러 가지 C++코드 패턴과 디자인 패턴을 제공하여, C++을 사용하여 시스템을 개발하기 위한 객체지향 설계를 도와준다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하고, 3장에서는 본 연구에서 개발한 코드 생성기의 기능 및 설계에 대해서 설명하고, 4장에서는 실제 구현 내용을 담았으며, 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

객체지향 설계에서 중요한 것은 클래스와 클래스들 사이의 관계를 설계하는 것이다. 소프트웨어 개발에서 중요한 문제는 유지 보수를 하기 위한 소프트웨어의 코드에 대한 충분한 이해이다. 이런 요구 사항들을 만족시키기 위해서 다양한 객체지향 방법론이 등장하였다[3]. Booch, Rumbaugh, Coad/Yourdon은 각자의 객체지향 방법론을 제시한 대표적인 사람들이다. 많은 객체지향 방법론은 사용자나 이것을 기반으로 CASE 도구를 만드는 회사에게나 단일화된 표준의 필요성을 느끼게 하였다. 이런 요구가 UML(Unified Modeling Language)이라는 하나의 표준화된 객체지향 모델링 언어를 등장시켰다.

다음의 관련 연구에는 상용 CASE 도구, C++ 역공학 도구, 새로운 C++ 표준[1], 코드 패턴, 디자인 패턴에 대한 간략한 소개를 하였다.

2.1 상용 CASE 도구

래셔널 로즈(Rational Rose 98i(TM))[4], 패러다임 플러스(Paradigm Plus(TM))[5]는 가장 많이 사용되는 상용 CASE 도구들이다. 이들 상용 CASE 도구들은 최신 버전의 UML을 지원하고 있고, UML 클래스 다이어그램상에서 C++코드를 생성할 수 있다. 그러나 이들 상용 CASE 도구들이 생성한 C++코드는 UML 클래스 다이어그램이 나타내는 의미를 정확히 표현하지 못하는 경우가 있다. 예를 들면, 1대 n 관계, aggregation 관계를 표시하기 위해서 주소형의 멤버 변수나 복수형의 멤버 변수이름을 사용한다. 여기에서의 문제점은 사용자가 생성된 C++코드를 이용하여 시스템을 개발할 경우에, 구현에 많은 부담을 주고, 생성된 C++코드가 전달하는 의미도 명확하지가 않다. 기존의 상용 CASE 도구들이 이러한 부적절한 코드를 생성하는 이유는 C++은 언어 자체가 1대 n 관계나 aggregation 관계를 표시할 수 있는 적절한 방법을 제공하지 않기 때문이다. 또한 상용 CASE 도구들은 하나의 UML 클래스 다이어그램을 통하여 여러가지 언어, 예를 들면, C++, VC++, 자바, 코바/IDL, VB, DDL 코드 등을 생성하기 때문에, C++가 가지는 특징을 충분히 반영하지 못하고 있다.

2.2 C++ 역공학 도구

C나 C++ 원시 코드로부터 역공학을 이용하여 사용자가 클래스에 대한 명세나 클래스 사이의 메시지 전달에 대한 정보를 좀더 그래픽컬하게 보여주는 많은 CASE 도구들이 개발되었다[4, 5, 6, 7, 8]. 이런 도구들은 소프트웨어 재사용성을 높이고, 시스템의 유지보수에 많은 도움을 준다. 그 중에서도 OODesigner[7]는 UML에 근간이 되는 Rumbaugh의 OMT 3개의 모델 중에서 객체 모델을 지원한다. 그러나 역공학 도구에서는 C++ 코드가 클래스 사이의 관계를 명확하게 나타내지 않기 때문에, UML 클래스 다이어그램의 association, aggregation관계를 나타내기가 쉽지 않다. generalization 관계는 클래스의 상속관계를 통해서 명확하게 나타난다.

본 코드 생성기는 UML 클래스 다이어그램에서 C++ 코드를 생성할 때, 클래스 다이어그램에서 나타나는 association, aggregation 관계에 대해서 새로운 C++ 표준을 이용하여 적절한 C++ 코드를 생성한다.

2.3 새로운 C++ 표준

C++은 가장 많이 사용되는 객체지향 프로그래밍 언어이다. 그러나 C++의 표준 라이브러리는 C++의 기능에 비하여 많이 빈약하였다[9]. 최근에 C++ 표준 위원회는 새로운 표준을 정하였는데, 그 중에서 C++ 표준 라이브러리와 템플릿에 많은 변화가 있었다[10]. 본 논문에서 개발한 코드 생성기가 생성한 C++코드는 새로운 C++표준을 지원한다. 예를 들면, 리스트(list), 벡터(vector), 집합(set) 클래스와 같은 STL 컨테이너 클래스을 이용하거나, 불(bool)이나 스트링(string)과 같은 새로운 데이타 타입을 지원한다. 본 논문의 코드 생성기에서 생성한 C++ 코드는 새로운 C++표준을 지원하는 컴파일러에서만 컴파일이 된다. 그러나 이것은 호환성 문제와는 상관이 없다. 왜냐 하면, 앞으로 C++ 컴파일러는 새로운 C++표준을 지원할 것이기 때문이다. 본 코드 생성기에서는 상용 CASE 도구와 달리 1대 n 관계

1) 1997년 11월에 새로운 ANSI/ISO 표준이 발표되었다.

나 aggregation 관계를 표시하기 위해서 STL 컨테이너 클래스를 이용하였다.

2.4 코드 패턴

C++ 코드 패턴은 C++ 전문가들이 프로그래밍을 하면서 얻은 경험의 집합체라고 할 수 있다. 이런 경험들은 다른 C++ 개발자들에게 좋은 객체지향 프로그래밍을 할 수 있는 가르침이 될 수 있다. 그 중 일부는 반복적으로 나타나기 때문에 자동화를 하면, 귀찮은 일을 피할 수도 있다. C++ 코드 패턴의 예로는 메모리를 동적으로 할당받을 때, 그 주소가 유효한가를 검사하는 것이다. 이런 코드들은 실행 중 에러가 발생할 경우에 쉽게 에러가 발생한 위치를 알 수 있다. C++ 클래스에 복사 생성자와 복사 할당 연산자(copy assignment operator)는 클래스에서 많이 사용된다. 또한 소멸자를 사용하기 보다는 가상 소멸자(virtual destructor)를 사용하는 것이 동적으로 생성한 객체가 소멸될 때, 해당 타입에 맞는 소멸자가 불릴 수 있다. 후자의 경우는 클래스를 설계할 때마다 반복적으로 나타나기 때문에 자동화를 하면, 사용자의 반복적인 작업을 줄일 수 있다.

코드 패턴은 템플릿보다 융통성이 있고, 템플릿이 C++컴파일러에 의해 처리되지만, 코드 패턴은 응용 프로그램에서 특정한 변환기(custom translator)가 제공되어야 한다[11]. (표 1)에서 C의 매크로와 C++의 템플릿과 코드 패턴에 대하여 비교하였다.

Code Navigator for C++[12]는 GUI환경에서 C++코드를 자동으로 생성하지만, 특정한 객체지향 방법론을 바탕으로 하지 않는다. 여기에서는 C++에 대한 코드 패턴을 이용하여 사용자의 반복적인 작업을 자동화하였다. 그러나 특정한 객체지향 방법론을 사용하지 않기 때문에 범용성이 떨어지고, 클래스와 클래스 사이의 정적인 관계를 나타낼 수 있는 방법도 제공하지 않는다. 또한 시스템의 설계보다는 시스템 구현을 도와주는 역할을 수행한다. 논문에서 제안한 코드 생성기는 Code Navigator for C++에서 사용하는 몇 가지 C++코드 패턴을 지원하여 사용자의 반복적인 작업을 자동화하였다.

표 1 매크로, 템플릿, 코드 패턴의 비교[11]

	C 매크로	C++ 템플릿	코드 패턴
서술	#define문	지너릭 타입 (generic type)	특별한 템플릿
처리기	전처리기	컴파일러	변환기
지능	단순환 텍스트 치환	인자의 타입 치환	임의의 인자 타입 치환
구현	70년대 후반	80년대 후반	90년대 중반

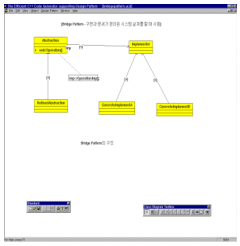
2.5 디자인 패턴

디자인 패턴은 객체지향 설계 경험을 표현하는 새로운 메커니즘으로 객체지향 모델링 연구에 중요한 관건이 되고 있다. 디자인 패턴은 많은 객체지향 설계에서 발견된 유용한 설계에 이름을 붙여서 재사용성을 높였다[13, 14, 15]. 디자인 패턴은 다음과 같은 장점이 있다[14].

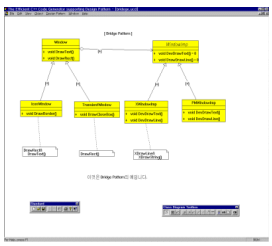
- 시스템 설계자가 의사 소통, 문서화, 새로운 설계 대안을 말할 때, 공용어 역할을 한다.
- 재사용 가능한 소프트웨어를 만들 때, 재사용 가능한 설계의 기본을 제공한다.
- 클래스 라이브러리를 배울 때, 배우는 시간을 줄일 수 있다. 왜냐 하면, 설계의 이유를 파악할 수 있기 때문이다.
- 클래스 계층 구조를 재구성할 때, 하나의 목표가 된다.

디자인 패턴은 객체지향 설계를 할 때 유용할 것이고, 이것을 지원하는 것은 재사용성이 있는 시스템을 설계하고, 좋은 설계를 할 수 있는 안내자의 역할을 수행할 수 있다. (그림 1)은 디자인 패턴의 하나인 Bridge 패턴이다. Bridge 패턴은 구현과 명세를 분리해서 두가지가 서로 독립적으로 작동하게 하였다. 특히 하나의 명세에 여러 가지 구현을 가지게 함으로써, 실행 중에 다른 구현으로 전환하는 시스템을 설계할 때, 이 디자인 패턴을 사용하면 된다[14]. 예를 들어, 윈도우 시스템을 구현할 때, 하나의 명세에 따라 구현, XWindow시스템, PMWindow시스템 등의 여러 가지로 구현하고, 실행 중에 XWindow시스템을 사용하다가 PMWindow시스템으로 쉽게 전환할 수가 있다. 왜냐하면, 두 개의 윈도우 시스템은 구현은 다르지만, 명세는 같기 때문이다.

UML을 지원하는 많은 상용 CASE 도구들은 디자인 패턴을 지원하고 있다[4, 5]. 본 논문의 코드 생성기는



(a) 디자인 패턴의 하나인 Bridge 패턴



(b) Bridge 패턴을 이용한 설계의 예

그림 1 디자인 패턴을 이용한 설계의 예

하나의 디자인 패턴을 하나의 템플릿으로 만들어서, 사용자는 필요한 정보만을 추가하여, 각 디자인 패턴이 가지는 장점을 이용할 수 있게 하였다.

3. 시스템의 기능과 효율성

본 논문의 코드 생성기는 UML 클래스 다이어그램으로부터 C++코드와 문서를 생성하는 일을 수행한다. 기존의 상용 CASE 도구가 UML 클래스 다이어그램을 통해서 C++코드와 문서를 생성하는 것보다 C++ 표준 라이브러리의 STL 컨테이너 클래스를 이용하여 C++만으로 표현하기 어려운 UML 클래스 다이어그램의 표기(notation)들을 더 잘 표현할 수 있게 하고, 코드 패턴과 디자인 패턴을 지원해서, C++을 위한 객체 지향 설계의 틀을 마련하였다.

3.1 시스템 설계 측면에서의 기능

C++과 같은 객체지향 프로그래밍 언어를 사용하여 시스템을 구현할 때 중요한 것은 클래스 설계를 잘 하는 것이다. 이 설계는 각 모듈간의 독립성을 보장하고, 유지 보수에 적은 비용이 들 수 있도록 하는 것이다. 유지 보수를 하는 프로그래머는 전체적인 시스템의 기능을 알기 위해서, 전체 코드를 다 살펴보지 않고도 쉽게 이해할 수 있어야 한다. 시스템을 설계할 때, 객체 지향 설계 기법을 이용하면 전체적인 클래스 구조와 클래스 사이의 관계를 그래픽하게 표현할 수 있기 때문에, 시스템 개발자들에게 큰 도움을 준다. 시스템 설계 단계가 중요한 이유는 설계 에러가 구현 단계로 전파된다면, 이것을 수정하는 비용과 시간이 훨씬 많이 들기 때문이다. C++ 코드를 생성하는 OOCASE 도구를 사용하여 시스템을 설계할 때 중요한 것은 클래스와 클래스들 사이의 관계를 잘 설계하는 것이다. 그리고 그 결과를 C++코드로 잘 표현할 수 있어야 한다.

본 코드 생성기는 기존의 상용 CASE 도구가 하나의 클래스 다이어그램에서 많은 프로그래밍 언어를 생성하는 것과 달리, C++ 코드만을 생성하기 때문에 C++가 가지는 특징을 잘 지원하도록 했다. 예를 들어 UML 클래스 다이어그램의 애트리뷰트와 메쏘드 표기는 타입이 이름보다 먼저 오는 데, 이것을 C++ 개발자에게 익숙한 표현으로 바꾸었다.

원래 UML의 표기법

+ area(width : long, length : long) : long

본 코드 생성기에서의 표기법

+ long area(long width, long length)

UML 클래스 다이어그램의 경우는 클래스 사이의 상속 관계를 표시할 때, 두 개의 클래스를 화살표로만 연결한다. 그러나 본 코드 생성기는 public, protected, private 중 어떤 것으로 상속을 받았는지를 표시를 할 수 있게 하였다. 다중 상속의 경우는 virtual 이라는 키워드를 사용할 수 있게 하여, C++ 다중 상속에서 발생하는 모호성(ambiguity) 문제를 해결할 수 있게 하였다.

(그림 2)에 있는 클래스 다이어그램에서 Employee 클래스와 Student 클래스는 Person 클래스를 public이면서 virtual로 상속을 했다는 것을 표시하고 있고, TA 클래스는 Employee 클래스와 Student 클래스를 public으로 상속하고 있다. 또한 생성된 C++ 코드는 새로운 C++ 표준을 따르도록 했으며, C++을 위한 코드 패턴과 디자인 패턴을 지원하여 사용자가 시스템 설계하는 것을 돕는다. 본 코드 생성기의 특징을 크게 3가지로 나누면 다음과 같다.

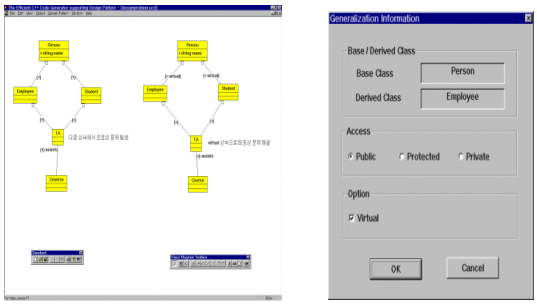


그림 2 다중 상속에서 모호성 문제 해결 방법

(특징 1) 1대 n 관계와 aggregation 관계의 효율적인 C++ 코드 생성

UML 클래스 다이어그램에서 표기(notation)중에서 코드 생성과 관련이 있는 것은 클래스와 클래스들 사이의 관계이다. UML 클래스 다이어그램에서 클래스들 사이의 관계를 나타내는 것들 중에서 코드 생성과 관련이 있는 것은 association, generalization, aggregation 등이다. 이런 UML 표기들을 C++ 코드로 생성하려면 규칙이 필요하다. UML의 클래스는 C++의 클래스에 해당하고, UML의 generalization은 클래스들 사이의 상속으로 표시할 수 있다. 여기서 문제가 되는 것은 association과 aggregation 관계인데, C++에서는 1대 n 관계나 n대 n 관계를 표시할 수 있는 방법이 명시적으로 존재하지 않는다. 래셔널 로즈나, 패러다임 플러스와 같은 상용 CASE 도구에서는 주소형 멤버 변수나 복수 형태의 멤버 변수를 사용해서 표시하였다. 이렇게 생성된 코드는 사용자로 하여금 1대 n 관계나 aggregation 관계를 유지하기 위해서 많은 부담을 준다.

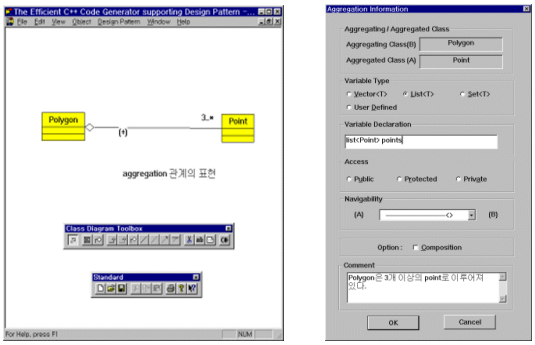


그림 3 aggregation 관계의 표현

예를 들어, (그림 3)의 UML 클래스 다이어그램을 상용 CASE 도구를 사용하여 C++코드로 생성하면 다음과 같다. 하나의 다각형이 여러 개의 점들로 구성되어 있다고 할 때, 다각형에서 새로운 점을 추가하거나 삭제하는 것이 쉽지 않다.

```
#include "Point.h"
class Polygon {
    // 기타 멤버나 멤버 함수.
public:
   Point points; // 복수형태의 변수 이름의 사용
  };
또는
#include "Point.h"
class Polygon {
    // 기타 멤버나 멤버 함수.
public:
    Point* points; // 포인터 변수의 사용
};
```

본 코드 생성기에서는 (그림 3)의 UML 클래스 다이어그램으로부터 다음과 같은 C++코드를 생성한다. 하나의 다각형이 여러 개의 점들로 구성되어 있다고 할 때, 이들에 대한 관리는 벡터나 리스트와 같은 STL 컨테이너 클래스의 멤버 함수를 이용하면 되기 때문에, 사용자가 생성된 C++코드를 이용하여 시스템을 구현할 때, 부담을 줄일 수 있다.

```
#include "Point.h"
#include <vector>
class Polygon {
    // 기타 멤버나 멤버 함수.
public:
   vector<Point> points; // STL의 벡터 클래스를 이용
};
또는
#include "Point.h"
#include <list>
class Polygon {
    // 기타 멤버나 멤버 함수.
public:
   list<Point> points; // STL의 리스트 클래스를 이용
};
```

aggregation 관계나 1대 n 관계를 C++코드로 나타내기 위해서 STL 컨테이너 클래스를 사용하여 얻는 이점은 다음과 같다.

- 생성된 C++코드가 클래스 다이어그램 표기를 좀 더 명확하게 표현할 수 있다.
- STL 컨테이너 클래스가 표준 C++라이브러리이기 때문에, 호환성이 뛰어나다.
- 사용자는 STL 컨테이너 클래스의 멤버함수를 사용하여 원하는 일을 쉽게 할 수 있다.

(특징 2) 코드 패턴을 이용한 사용자의 설계 편의성 향상

코드 패턴은 특정한 프로그래밍 언어를 이용하여 프로그래밍을 지속적으로 수행함으로써 얻게 되는 유용한 경험들의 집합이라고 할 수 있다. C++ 코드 패턴은 C++ 프로그래밍에서 반복적으로 나타나는 유용한 것들을 모아 놓은 것이고, 이를 자동화하여 사용자에게 제공하면 여러 가지 유용한 면이 있다. 물론 코드 패턴이 사용자에게 원하지 않는 결과를 가져올 수도 있지만, 그럴 경우 사용자는 그것을 수정할 수 있기 때문에, 큰 문제는 발생하지 않을 것이다.

본 코드 생성기에서 사용하고 있는 코드 패턴의 대해서 알아보도록 하자.

- **표준 클래스(canonical class)**

C++의 클래스에는 디폴트 생성자와 소멸자가 있어야 한다. 사용자가 만든 클래스에 표준 클래스를 적용시키면, 즉 해당 클래스를 표준 클래스로 만들면, 복사 생성자와 복사 할당 연산자(copy assignment operator)가 추가된다. 즉 보통의 클래스에는 복사 생성자와 복사 할당 연산자가 있기 때문에 사용자가 쉽게 자동으로 추가할 수 있도록 한다는 것이다.

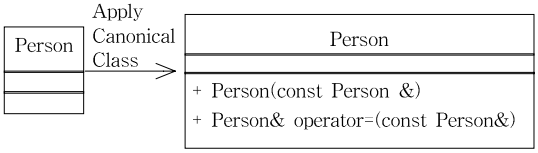


그림 4 표준 클래스 만들기

● **값 읽기와 값 쓰기 메쏘드(get/set methods)**

클래스에서 멤버는 객체의 상태를 나타내고, 멤버 함수는 객체의 상태를 변화시키는 역할을 한다. 멤버들은 private영역에 두어서 다른 클래스에 속하는 객체나 자신의 하위 클래스가 멤버의 값을 함부로 변경하지 못하도록 하는 것은 좋은 프로그래밍 습관이다. 그렇다고 해서 모든 멤버를 private영역에 두고, 이 멤버에 접근하는 멤버함수를 public영역에 따로 만드는 것은 귀찮은 일이다. 여기에 있는 값 읽기와 값 쓰기 메쏘드는 private영역에 있는 멤버에 대해서만 public영역에 이 멤버에 접근할 수 있는 get_과 set_함수를 자동으로 추가해 준다. 본 코드 생성기에서는 이것을 통하여 캡슐화(encapsulation)와 정보 은폐(information hiding)를 제공한다. 그리고 사용자의 필요에 따라 특정 멤버에 대한 get_과 set_함수를 제거하는 것도 가능하다.

● **관련 있는 연산자(related operators)**

C++는 연산자 중복(operator overloading)을 통하여 사용자가 정의한 클래스에 대해서 여러가지 연산자를 사용할 수 있게 함으로써, 코드의 가독성을 높일 수 있다. 또한 서로 관련 있는 연산자는 함께 정의되는 경우가 많기 때문에, 사용자가 하나의 연산자를 정의하면, 이와 대응이 되는 연산자를 자동으로 제공하여 사용자가 클래스를 설계하는 것을 돕는다. 본 코드 생성기에서 제공하는 관련 있는 연산자들은 다음과 같다. 즉, 연산자가 받는 인자의 개수와 타입이 같고 리턴값이 같은 것들을 관련 있는 연산자라고 정의하였다. 그래서, 하나만 정의하면 나머지는 자동으로 추가할 수 있도록 하였다. 다음은 본 코드 생성기에서 관련있는 연산자로 규정하고 있는 것들이다. 이것은 연산자 중복을 사용하는 경우에 많은 도움을 줄 수 있다.

operator== 연산자와 operator!= 연산자
operator++ 연산자와 operator-- 연산자
operator> 연산자와 operator< 연산자
operator>= 연산자와 operator<= 연산자
operator 연산자와 operator/ 연산자*
operator= 연산자와 operator/= 연산자*
operator+ 연산자와 operator- 연산자
operator+= 연산자와 operator-= 연산자
operator& 연산자와 operator| 연산자
operator&& 연산자와 operator|| 연산자

(특징 3) 디자인 패턴의 지원

디자인 패턴은 객체지향 설계에서 제공하는 유용한 클래스와 클래스들 사이의 관계로 이루어져 있다[14]. 디자인 패턴을 지원하는 것은 유용성이 보장되는 디자인을 재사용할 수 있다는 것이다[15]. 디자인 패턴은 그 자체가 UML 클래스 다이어그램으로 표현이 가능하기 때문에, C++의 템플릿처럼 제공하여 사용자가 자신에게 필요한 정보를 추가하여 사용하면 유용한 디자인이 될 수 있다. 각각의 디자인 패턴을 C++의 템플릿처럼 제공하여 사용자가 필요한 항목을 추가하여 사용할 수 있도록 하였다. 예를 들어, 본 코드 생성기에서는 (그림 1-a)처럼 Bridge 패턴을 사용할 수 있는 템플릿을 제공한다. 본 코드 생성기에서는 각각의 디자인 패턴을 파일로 저장해서, 사용자가 불러서 사용할 수 있고, 또한 편집도 가능하도록 하였다. 사용자는 (그림 1-b)처럼 자신에게 필요한 정보를 추가하면, 이 디자인 패턴이 제공하는 장점을 이용할 수 있다.

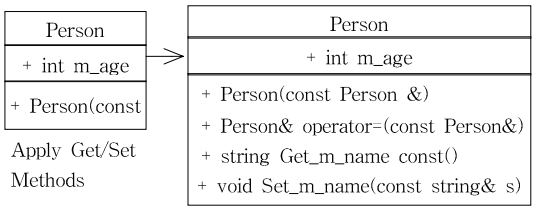


그림 5 get/set 멤버함수 추가하기

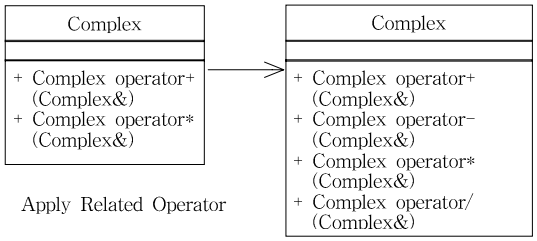


그림 6 관련있는 연산자 추가하기

3.2 유지 보수 측면에서의 기능

소프트웨어의 경우 계속적인 유지 보수가 필요하다. 보통 유지 보수에 드는 비용은 전체 소프트웨어 개발비용의 70%이 이상이 드는 것으로 알려져 있다[16]. 유지 보수에 드는 비용을 줄이기 위해서는 무엇보다도 개발된 시스템에 대한 문서화와 시스템의 이해를 도울 수 있는 자료가 많을수록 유지 보수를 하는 시간과 비용이 절감된다. 또한 하나의 소프트웨어 프로젝트를 한 사람이 하는 경우는 드물다. 여러 사람이 함께 작업을 할 경우에 그들 간에 문서 교환이 필요하며, 이 때 통일된 양식의 문서를 필요로 한다. 본 코드 생성기를 이용하면, 시스템을 설계할 때, 사용자가 입력한 내용을 모아서

html과 latex 양식의 문서를 자동으로 생성한다.

3.3 다른 CASE 도구와의 비교 분석

본 장에서는 본 코드 생성기의 효율성을 설명하기 위해 (표 2)에 다른 상용 CASE 도구와 비교 설명을 하였다. 패러다임 플러스나 래셔널 로즈와 같은 상용 CASE 도구들이 다양한 언어를 지원하기 때문에 C++ 같은 특정 언어를 위한 코드 패턴을 명시적으로 지원하기 힘들고, C++가 가지는 특징을 잘 반영하기도 어렵다. 본 코드 생성기는 C++만을 대상으로 하기 때문에 C++ 코드 패턴을 효율적으로 지원할 수 있다. 또한 1대 n 관계나 aggregation 관계를 C++코드로 생성하기 위해서 STL 컨테이너 클래스를 사용하였다. 그러나 상용 CASE 도구에서는 범용성과 호환성을 위해서 라이브러리 수준의 코드 생성을 하지 않았다. 디자인 패턴은 재사용 가능한 설계의 틀로 많은 상용 CASE 도구들이 지원하고 있다.

그러나 본 코드 생성기는 UML에서 사용하는 여러 다이어그램 중에서 클래스 다이어그램만을 지원하는 단점이 있다.

표 2 다른 CASE 도구와의 비교

	래셔널 로즈	패러다임 플러스	본 코드 생성기
다수의 언어 생성 지원	O	O	X
STL 컨테이너 클래스 사용	X	X	O
C++ 코드 패턴의 지원	X	X	O
디자인 패턴의 지원	O	O	O
다양한 UML 다이어그램 지원	O	O	X

4. 시스템 구현

본 코드 생성기는 PC 윈도우 환경에서 마이크로소프트사의 MFC(Microsoft Foundation Class Library)를 이용하여 구현하였다. 전체 시스템 구성도는 (그림 7)과 같다.

(그림 7)에서 보는 것처럼 크게 4개의 부분으로 구성되어 있다. UML 클래스 다이어그램 도구는 사용자가 마우스와 키보드를 이용하여 구현할 시스템에 대한 UML 클래스 다이어그램을 작성한다. UML 클래스 다이어그램 그리기 도구에서 생성되는 모든 정보는 OODB인 ObjectStore[2] 에 저장이 된다. C++ 코드 생성과 문서 생성은 OODB에 저장된 정보와 정해진 규칙을 이용한다. 다음에 각각의 모듈에 대해서 설명을 하였다.

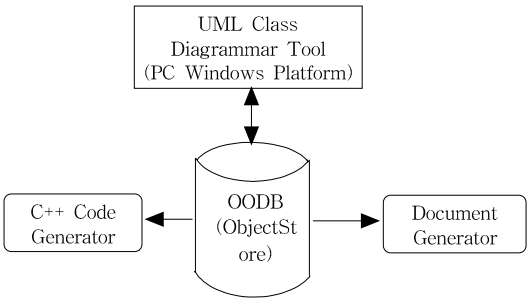


그림 7 전체 시스템 구성도

4.1 UML 클래스 다이어그램 그리기 도구

사용자는 UML 클래스 다이어그램 그리기 도구를 이용하여 구현하려는 시스템에 대한 UML 클래스 다이어그램을 작성한다. 사용자가 클래스 객체를 생성하면, 이것은 모두 OODB에 저장이 된다. 클래스에 들어가는 정보는 클래스 이름, 멤버, 멤버 함수들이 들어간다. 클래스 객체가 생성될 때마다, OODB에 저장되고, 삭제되거나 변경하는 것도 가능하다. 사용자가 만든 UML 클래스 다이어그램의 설명을 위해서, 노트(note)나 텍스트박스 객체를 이용할 수도 있다. 이것들도 OODB에 저장은 되지만, C++코드 생성에는 아무런 영향을 주지 않는다.

왼쪽부터 선택, 클래스, 인터페이스, generalization, realize or refinement, aggregation, association, anchor note to item, dependency, 링크 애트리뷰트, 삭제, 텍스트 박스, 노트, C++ 코드 생성을 나타낸다.

그림 8 클래스 다이어그램 그리기 도구 상자

클래스, 노트, 텍스트 박스 객체는 독립적인 객체들로서 다른 객체와 무관하게 존재할 수 있지만, 관계 객체들은 반드시 독립적인 객체들과 연결되어 사용된다. 만약 독립적인 객체와 연결된 관계 객체들은 연결된 독립적인 객체가 삭제될 경우, 함께 삭제된다.

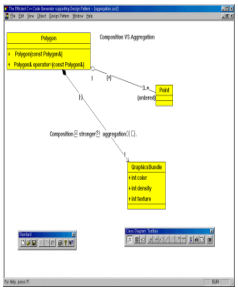
관계 객체로는 association, aggregation, generalization, realize or refinement, dependency, anchor note to item, 노트, association 클래스 등이 있다. 이들 중에서 C++ 코드 생성과 관련이 있는 것은 association, aggregation, generalization이다.

2) Object Design사의 PC용 지속적 저장 엔진(persistent storage engine)이다.

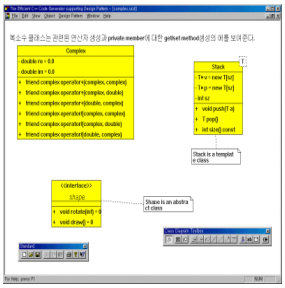
dependency 관계는 클래스 객체간에 의미적으로 의존 관계가 있다는 것을 표시한다. anchor note to item은 노트 객체와 클래스 객체를 연결할 때 사용한다. association class는 association 관계 객체와 클래스 객체를 연결한다. (그림 9)는 본 코드 생성기를 이용하여 작성한 4개의 클래스 다이어그램의 예이다.

여기에서 사용되는 표기는 UML 버전 1.1[17, 18]을 기준으로 하고 있다. UML 버전 1.1에 있는 인터페이스 표기를 제외한 모든 표기를 지원한다. UML 클래스 다이어그램의 인터페이스는 C++에서 사용하지 않기 때문에, 본 코드 생성기에서는 지원하지 않는다.

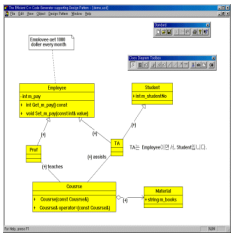
(표 3)은 UML 클래스 다이어그램에서 C++ 코드 생성의 규칙을 설명한 것이다.



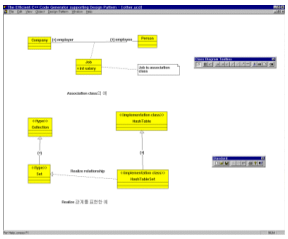
(a) Composition 과 Aggregation 관계의 예



(b) 클래스, 템플릿 클래스, 추상 클래스의 예



(c) 다중 상속을 이용한 디자인의 한 예



(d) Association 클래스와 Realize 관계의 예

그림 9 4개의 예제 클래스 다이어그램

4.2 정보 저장소

UML 클래스 다이어그램 도구에서 객체가 생성될 때마다, 이것을 OODB에 저장한다. 각 객체는 OID를 가지는데, 객체에 부여되는 OID는 그 이전에 생성된 객체보다 1이 증가된 값을 가진다. 이 OID를 이용하여 객체에 대한 정보를 얻어낸다. 그리고 사용자가 객체를 삭제하면, OODB에서 객체가 삭제된다.

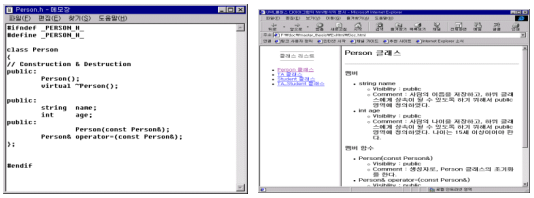
4.3 C++ 코드 생성기

C++코드 생성기는 OODB에 저장된 정보와 미리 정해진 규칙을 가지고 C++코드를 생성한다. 생성된 C++ 코드는 새로운 C++표준을 따르고, UNIX상에서 g++로

표 3 C++ 코드 생성 규칙

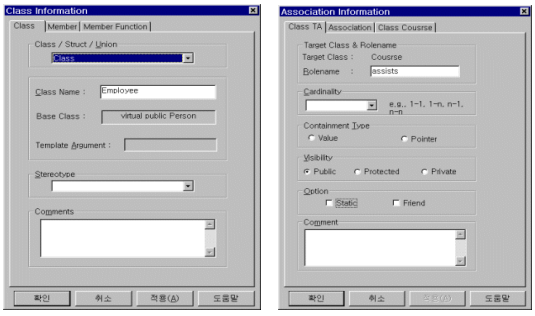
UML 클래스 다이어그램 표기	C++ 코드 생성 규칙	
클래스	종류	사용자의 선택에 따라 Class/Struct /Union으로 생성가능, 클래스는 다시 템플릿 클래스와 추상 클래스, 일반 클래스로 구분된다.
	클래스 이름	클래스 이름
	상위 클래스	클래스 사이에 generalization relationship이 존재할 경우에 나타난다.
	생성자와 소멸자	디폴트 생성자와 가상 소멸자를 자동으로 생성한다.
애트리뷰트	멤버 변수 타입	멤버 변수 타입
	멤버 변수 이름	멤버 변수 이름
	멤버 변수의 초기값	생성자에서 해당 멤버를 초기화한다.
	멤버 변수의 접근성	public / protected / private
메쏘드	멤버 함수 타입	멤버 함수의 리턴값
	멤버 함수 선언	멤버함수의 이름과 인자들을 기록한다.
	멤버 함수의 접근성	public / protected / private
	옵션	static / virtual / pure virtual / inline / friend 의 옵션을 지정한다.
generalization 관계	상속의 접근성	public / protected / private
	다중 상속의 모호성 문제	virtual / no virtual
association 관계	방향성	양방향성은 서로 상대방에 대한 정보를 멤버로 유지하고, 단방향성은 한쪽에서만 정보를 유지한다
	카디널리티(cardinality)	1대 n이나 n대 n인 경우에는 STL 컨테이너 클래스를 이용하여 C++코드를 생성한다.
	접근성	public / protected / private
	옵션	static / friend 멤버로 지정한다.
aggregation 관계	방향성	양방향성은 서로 상대방에 대한 정보를 멤버로 유지하고, 단방향성은 한쪽에서만 유지한다
	접근성	public / protected / private
	코드 생성	STL 컨테이너 클래스를 이용하여 C++ 코드를 생성한다.

컴파일이 된다. 각각의 클래스는 .h파일과 .cpp파일이 생성된다.

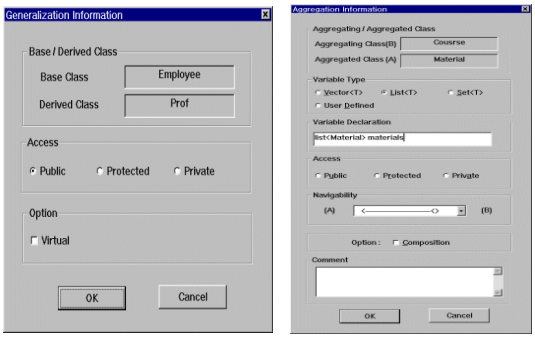


(a) C++ 코드 생성의 예 (b) 문서 생성의 예

그림 10 C++ 코드와 문서 생성



(a) 클래스, 애트리뷰트, 메쏘드 (b) Association 관계



(c) Generalization 관계 (d) Aggregation 관계

그림 11 C++ 코드 생성과 관련이 있는 대화 상자

4.4 문서 생성기

시스템을 개발할 때 중요한 것은 시스템에 관한 문서를 생성하는 것이다[19]. 이것은 시스템을 개발하고, 이를 유지 보수하는 과정에 필요하다. 본 코드 생성기를 이용하여 시스템을 설계할 때, 클래스에 대한 정보나 멤버, 멤버 함수, 클래스들 사이의 관계에 대한 설명을 모아서, 이것을 하나의 문서로 만든다. 본 코드 생성기는 html과 latex형식의 문서를 생성한다.

5. 결 론

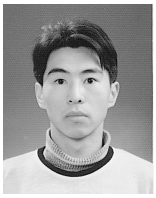
본 코드 생성기의 가장 큰 의의는 C++를 이용하여 시스템을 개발하는 데 있어서, 시스템 설계 단계와 구현 단계를 분리시키지 않도록 했다는 것이다. 그리고 객체지향 프로그래밍을 하는데 가장 중요한 설계 단계를 도와준다. 기존의 CASE 도구들이 C++에서 1대 n 관계나 aggregation 관계를 잘 표현하지 못했던 것을, C++의 STL 컨테이너 클래스를 이용함으로써, 생성된 코드가 좀 더 효율성을 가지게 하였다. 또한 C++코드 패턴을 이용하여 사용자의 설계 편의성을 더했고, 디자인 패턴의 지원을 통하여 객체지향 설계의 질을 높을 수 있는 방법을 제공하였다. 그러나, 본 코드 생성기가 가지는 문제점도 있다. 설계단계에서 발생하는 에러를 검출하는 부분이 미약하고, OODB에 저장된 데이터를 효율적으로 관리하지 못하는 단점이 있다.

향후 연구 방향은 설계 단계에서 에러를 검출하고 이를 사용자에게 알려 주는 부분에 대한 연구와 OODB에 저장된 데이터를 효율적으로 저장하고 검색할 수 있는 정보 저장소에 대한 연구가 필요할 것이다.

참 고 문 헌

[1] OCS Ltd, "C++ New Standard," http://www.ocsltd.com/ c++/, 1997.
[2] K. Scott and M. Fowler, "UML Distilled," Addison-Wesley, 1997.
[3] S. Khoshafian and R. Abnous. "Object Orientation," John Willey & Sons, 1995.
[4] Rational Software, "Rational Rose 98i," http://www.rational.com/rose, 1999.
[5] Platinum Technology, "Paradigm Plus," http://www. platinum.com/products/appdev/pplus_ps.htm/, 1999.
[6] 최은혁, 최은만, "역공학을 이용한 C 및 C++ 재사용 부품 추출 및 검색", 정보과학회 논문지(C), Vol.2, No.2, pp.197-205, 1996.
[7] 김태균, "C++ 언어에 대한 역공학 도구의 설계와 구현", 정보과학회 논문지(C), Vol.1, No.2, pp.135-145, 1995.
[8] 문양선 외 5명, "C++ 프로그램의 이해도 증진을 위한 역공학 시각화 도구", 정보과학회 논문지(C), Vol.1, No.2, pp.160-171, 1995.
[9] B. Stroustrup, "The C++ Programming Language 3rd," Addson Wesley, 1997.
[10] B. Stroustrup, "The C++ Programming Language

2nd," Addison Wesley, 1991.
[11] Quintessoft's whitepaper, "Code Patterns," http://www. quintessoft.com/whitepap.htm, 1997.
[12] Quintesoft Company, "Code Navigator for C++(TM)," http://www.quintessoft.com/, 1997.
[13] C. Kramer and L. Prechelt, "Design Recovery by Automated Search for Structural Design Patternsin OO Software," Conf. of Reverse Engineering, Nov. IEEE'96, 1996.
[14] E. Gamma, et. al., "Design Patterns - Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.
[15] M. Meijers, et. al., "Tool Support for Object-Oriented Design Patterns," Proceedings of ECOOP'97, 1997.
[16] I. Sommerville, "Software Engineering," Addison-Wesley, 1995.
[17] Rational Software Company, "UML Notation Version 1.1," http://www.rational.com/uml/html/notation, Nov. 1997.
[18] Rational Software Company, "UML Semantics Version 1.1," http://www.rational.com/uml/html/semantics, Nov. 1997.
[19] 김희천, 우치수, "문서화 지원을 위한 CASE 도구의 설계 및 구현", 정보과학회 논문지(C), Vol.3, No.1, pp.37-50, 1997.



조 형 주
1997년 서울대학교 컴퓨터공학과졸업(학사). 1999년 서울대학교 컴퓨터공학과 졸업(석사). 1999년 ~ 현재 한국과학기술원 전산학과 박사과정. 관심분야는 객체지향 기술, OODB, GIS, 공간/시공간 DB, 질의 최적화

정 진 완
정보과학회논문지 : 컴퓨팅의 실제
제 6 권 제 1 호 참조

김 형 주
정보과학회논문지 : 컴퓨팅의 실제
제 6 권 제 1 호 참조