論文 92-29B-2-2

C++에서의 객체 지향 모델링을 위한 다이아그래밍 툴

(Diagramming Tool for Object-Oriented Modeling on C++)

河 秀 澈* 元 裕 憲**

(Soo Cheol Ha and Yoo Hun Won)

要 約

최근 소프트웨어 개발을 위한 새로운 패러다임으로 객체 지향화가 주목받고 있다. 본 논문에서는 C++ 프로그램을 효과적으로 개발하기 위한 다이아그래밍 기법과 툴 구축에 관한 사항들을 제안한다. 이 다이아그래밍 기법은 C++의 클래스 성격이 강조되므로 모듈화 및 클래스 상호 작용의 명확한 표현이 가능하다. 또한 다이아그램의 논리적 작성 원리로부터 물리적 이미지 구현 원리로 직접적인 사상이 가능하므로 프로그래머가 설계 단계에서 얻은 자원의 재사용이 가능할 뿐아니라 코드화를 위한 구현 단계로의 변화도 용이하다.

Abstract

In recent years, object-orientation is rising to notice as a new paradigm for developing solftware. This paper suggests the diagramming technique and a tool for developing C++ program effectively. This technique can represent the modularity and the interactions of classes definitely by emphasizing the characteristics of classes of C++ . It can do the direct mapping from the logical idea to the physical screen image, so programmers can reuse the design resources in design phase as well as transforming the resources into chode in the implementation phase.

I. 서 론

소프트웨어 위기 현상을 인식한 이래 소프트웨어 개발시의 생산성 및 품질 향상은 해결해야 할 중요한 과제가 되었다. 전통적인 소프트웨어 개발 방법론들은 개발된 소프트웨어의 확장성, 재사용성, 프로그래머 생산성의 향상에 한계가 있기 때문에 소프트웨어 유지보수의 복잡도 및 비용의 제어가 어려웠다. 구조화 프로그래밍 개념으로부터 구조화 설계로 진화하여 온 기능적 분해 방법의 대안으로써 등장한 것이 본 논문에서 논의하려는 객체 지향 접근법 (object-orientation)이다.

전통적인 소프트웨어 설계 방법론의 대안으로 등장한 객체 지향 접근법이라 하더라도 설계 과정의 정형화가 수학 공식과 같이 정립되어 있는것이 아니고 설계자의 창조적인 능력과 경험에 의존하는 부분이 많다. 이때 프로그램 설계의 가시성을 높이기 위해 다이아그램(diagram)으로 그 과정을 문서화 하는 것은 즉시성의 제공, 설계를 코드로 구현하기 위한 청사진의 제공 등의 역할을 한다.[1,2] 많은 연구가들이 그들이 개발해 온 방법론을 강조하기 위해 다이아그래밍 표기법을 사용하여 왔다. 그 이유는 소프트웨

*正會員, 大田大學校 電子計算學科
(Dept. of Computer Eng., Daejeon Univ.)
**正會員, 弘益大學校 電子計算學科
(Dept. of Computer Eng., Hongik Univ.)
接受日字 : 1991年 12月 9日
(※ 본 논문은 한국과학재단의 지원으로 이루어졌음)

어 공학의 원리에 입각한 개발 방법론들은 다이아그래밍 기법에 의해 실현될 수 있기 때문이다.[3]

초보자나 C언어의 절차적 개념에 익숙한 프로그래머에게는 객체 지향 개념을 일관되게 견지하면서 C++ 프로그램을 개발한다는 것이 쉽지 않다. 따라서 절차적 개념으로부터 객체지향 개념으로의 사상 변환을 위한 방법이 요청된다. 본 논문은 이를 위해 객체지향 개념을 일관되게 유지하면서 프로그램 설계가 가능하며 이를 C++상에서 효율적으로 보조할 수 있는 다이아그래밍 기법을 제안한다. 또한 C++ 객체지향 프로그래밍을 위한 실험적인 CASE(computer aided software engineering) tool을 구현한 결과도 기술한다.

Ⅱ. 객체지향 다이아그래밍 기법과 C++

1. 객체지향 설계의 다이아그래밍 표기법

시스템 설계를 위한 방법론으로 객체지향 설계가 채택되었을 때 설계자에게 직면하는 문제는 요구사항 명세서(requirement specification)가 객체 지향적인 방법으로 표현되어 있지 않을 수 있으며 이러한 명세서로 부터 적절한 객체들과 관련 연산들을 결정하는 설계를 유도해내는 방법에 관한 것들이다. 이것은 설계 과정 자체가 설계자의 숙련도, 교육정도 및 직관과 경험에 의존하는 창조적인 과정이므로 공식과 같은 형태로 표현될 수 없기 때문이다.[1] 또한 전략을 정형화하는 단계가 기계적으로 나타날지라도 그것은 자동화된 절차가 아니기 때문이다.[4] 설계자는 가능한 한 다수의 설계를 가정하고 이것을 테스트하여 인정할 만한 해가 발견될 때까지 반복과정을 거쳐야 한다.

소프트웨어 공학 원리에 입각한 개발 방법론들은 다이아그래밍 기법에 의해 실현될 수 있다. 환언하면 다이아그래밍 기법은 소프트웨어 공학의 원리와 상호 관련있으며 방법론을 강화할 수 있다.[3] 이들의 관계는 그림1과 같다.

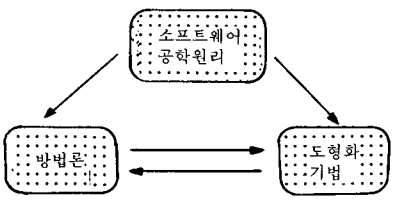

그림 1. 소프트웨어 공학 원리, 방법론, 다이아그래밍 기법

Fig. 1. Principle of software engineering, methodology, diagramming technique.

이것은 Fairley[6]의 지적처럼 소프트웨어 품질과 생산성에 영향을 미치는 요인이 된다. 객체지향 설계의 그래픽적 표현 역시 매우 유용하고 객체에 대한 다이아그램은 객체지향 설계를 연구하는 사람들에게 큰 도움을 제공한다. 즉, 다이아그램은 인터페이스 컴퍼넌트의 객체 성격을 강조할 수 있다. 이것은 객체 지향적 계산의 관점에서 본다면 객체들간에 수행되는 대화를 명확하게 표현할 수 있다는 의미이다.

객체지향 설계를 위한 다이아그래밍 연구들에는 Booch의 Boochgram[7,8,9,10], Seidewitz와 Stark의 Object Diagram[11], Sincovec와 Wiener의 Modula Design Chart[12,13], Cunningham과 Beck[14], Loomis와 그의 동료들의 연구[15], Sommerville[1] 등이 있다.

이들 연구를 방법론에 관해 평가를 하면, 대부분 Abbott와 Booch의 방법론을 따르고 있다. 그러나, 이 방법론은 Ada와 같은 프로그래밍 언어로 소프트웨어를 개발하는데 치우친 방법론이며, 객체 지향의 중요한 개념인 상속성과 메시지들에 관한 명백한 사용 절차가 준비되어 있지 않다는 단점을 갖고 있다.[23] 다이아그래밍 기법에 대한 평가를 하면, Boochgrram과 Modular Design Chart등 이들 대부분은 상위 추상화 수준에 있는 프로그램 컴포넌트를 표현하기 위해 사용하며, 세부설계 구현이 시작된 때에는 그래픽 표기는 중지되고 PDL이 사용된다. 이들이 개발한 다이아그램들은 설계의 단계에서 작성되며 프로그램을 위한 코딩으로의 사상시에는 설계자와 개발자가 다른 경우 프로그래머의 다이아그램 판독 능력과 프로그램 작성 능력에 좌우될 가능성이 있으며, 설계자와 개발자가 같은 경우에도 다이아그래밍을 위한 방법론과 동일한 방법으로 프로그래밍을 적용하기가 어렵다.[20] 따라서 본 논문에서는 제안하는 다이아그래밍 기법은 구조 설계와 세부 설계에 모두 적용될 뿐만아니라 상속성, 클래스들간의 관계 및 메시지에 관한 개념을 표기할 수 있도록 새로운 표기법을 제안하였다.

툴에 관한 연구로는 PEREAM,[16] HyperHood++,[17] PARADISE,[18] Trellis[19] 등이 있지만 C++에 관한 연구는 미비하다. PEREAM 시스템은 객체 지향 설계 단계가 프로그래밍 툴과 결합되어 Smalltalk-80의 전반부에 위치하면서 설계와 구현을 지원하는 환경이며, HyperHood++는 Ada로 개발되는 응용제품의 품질을 향상시키기 위해 사용되는 툴이다. PARADISE 시스템은 정보 시스템의 모듈 설계 단계를 소프트웨어 공학 환경으로 통합한 툴이며, Trellis 프로그래밍 환경은 객체 기반 언어인 Trellis/

Owl에서의 프로그래밍을 지원하면서 객체 지향 프로그래밍 방법론을 지원하기 위한 툴이다.

2. C++ 를 위한 다이아그래밍 표기법

객체 지향 설계가 구현 단계로 이동시 적용하는 프로그래밍 언어가 C++ 임을 전제로 한다. C++에 적합한 표기법은 객체지향 설계를 위한 다이아그래밍 기법들의 분석 결과를 토대로 제안하는데 제시하는 표기법은 다음 사항을 만족하려 한다.[20]

① 상위 추상화 레벨로부터 구현 세부사항까지를 표현할 수 있을 것
② 클래스 중심의 철학이 부각될 것
③ 복합하지 않은 단순화된 다이아그램을 사용할 것
④ CASE 도구 구현시를 고려할 것

이 요구사항을 목표로 하는 다이아그램을 C++ gram이라 명명한다.[22] 제안하는 표기법은 C++ 언어의 가장 중요한 객체 지향적 특성들을 중심으로 설계 지침을 쉽게 구현 단계로 이동하기 위한 다이아그래밍 표기법이다. 이 다이아그램들은 예비 설계와 세부 설계 단계에 모두 적용 가능하므로 다이아그램의 내부에 코드화가 예상되는 코드부분을 미리 상세히 기재한 것으로 논의를 전개한다.

1) 클래스

클래스 정의를 위한 다이아그램은 그림2와 같다.

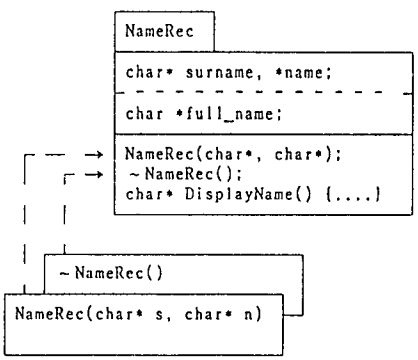

그림 2. 클래스와 멤버함수의 다이아그래밍 표기
Fig. 2. Diagram of class and member function.

하나의 클래스에 캡슐화되어 있는 객체를 표현하기 위한 클래스 표기 다이아그램은 3개의 부분으로 구성된다. 즉, 작성된 한 클래스 다이아그램과 다른 나머지 다이아그램의 부분에서 전적으로 사용 가능한 공개 부분(public part), 클래스 외부에서는 접근이 불가능한 비공개 부분(private part), 그리고 주어진 부모(수퍼) 클래스로부터 유도된 부속 클래스 내부로부터를 제외하고 클래스 외부에서는 접근할 수 없는 보호된 부분(protected part)으로 구성된다. 이 다이아그램이 프로그램으로 변환될 때 생성되는 코드는 다음과 같다.

[코드 1] 클래스 정의

```
class NameRec{
      char *surname, *name;
protected:
      char *full_name;
public;
      NameRec(char*, char*);
      ~NameRec();
      char *DisplayName()
      {.....}
};
```

//멤버 정의

```
NameRec:NameRec(char *s,
  char*n)
{
  ...=new...;
  ...
}
NameRec::~NameRec()
{
  delete...;
  ...
}
```

2) 메시지

이는 객체의 생성과 선택하려는 메소드(method) 또는 메시지의 성격을 규정하기 위한 행동 결정을 표현하는데 있다. 메시지를 표현하는 다이아그램은 그림3에 보인다. 여기에서 아래로 향한 화살표는 메시지의 흐름을 나타내고 있다.

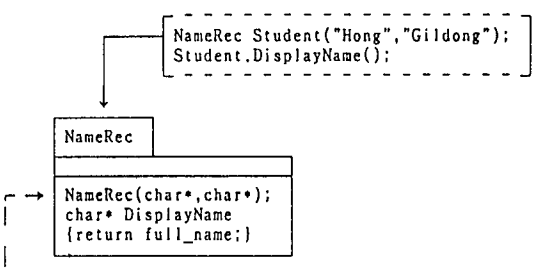

그림 3. 메시지 처리의 다이아그래밍 표기 (메시지 처리기)
Fig. 3. Diagram for handling message (Message handler).

3) 유도된 클래스

설계 추상화의 계층성을 위한 표기는 C++가 추상화의 계층성을 부속(sub) 클래스 또는 유도된(derived) 클래스를 사용하여 설정한다는 점을 이용하였다. 유도된 클래스(부속 클래스)는 자체의 메소드를 구현하지 않고 그 클래스의 수퍼 클래스(super, parent 또는 base class라도 함)로부터 모든 메소드 (구조나 특성 등)를 상속받을 수 있다. 또 수퍼 클래스

내부에는 보유하고 있지않지만 유도된 클래스에만 특정적인 즉, 새로운 메소드들을 지정할 수 있다. 수퍼 클래스와 유도된 클래스로 정의되는 다이아그래밍 표기법은 그림4에 보인다.

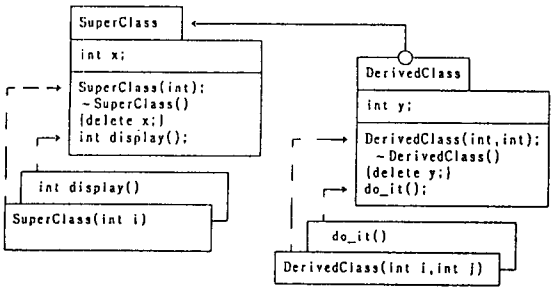

그림 4. 수퍼 클래스와 부속 클래스의 다이아그래밍 표기(비공개형)

Fig. 4. Diagram of superclass and subclass (private)

유도된 클래스의 어떤 멤버도 수퍼 클래스 공개부의 멤버함수를 접근할 수 있지만, 다른 클래스에서는 유도된 클래스를 통하더라도 접근할 수 없다. 이것은 비록 두 클래스는 서로 밀접하고 종속적인 관계를 가지고 있지만 수퍼 클래스 공개부는 유도된 클래스의 공개부가 아니라는 의미이다. 그림 4로부터 변환되는 프로그램의 예상 코드는 다음과 같다.

[코드 2] 클래스 상속

```
//수퍼클래스
class SuperClass{
  int x;
public:
      SuperClass(int);
      ~SuperClass(){delete x;}
      int display();
};
```

```
//유도된 부속클래스
class DerivedClass;
  SuperClass{
  int y;
public:
  DerivedClass(int, int);
  ~DerivedClass()
  {delety y;}
  do_it();
};
```

여기에서 클래스 DerivedClass의 어떤 멤버도 클래스 SuperClass의 멤버함수 display()에 접근할 수 있다. 그러나 그림4의 표기만으로는 또 다른 클래스들이 DerivedClass를 통하더라도 display()함수에 접근할 수 없다. 이 상황을 변경하기 위해서는 수퍼 클래스가 공개적이 되도록 하는 그림5와 같은 다이아그램 표기가 필요하다.

이 표기가 프로그램으로 변환될 때에는 public 선언이 필요하며 이후부터 수퍼 클래스의 공개부가 유도된 클래스의 공개부가 될 수 있다.

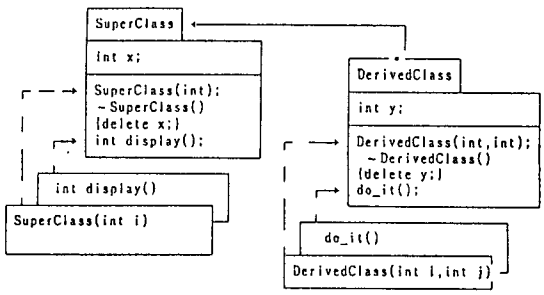

그림 5. 수퍼 클래스와 부속 클래스의 다이아그래밍 표기(공개형)

Fig. 5. Diagram of superclass and subclass (public).

Ⅲ. C++gram

1. C++gram의 의미론

본 논문에서 제안한 C++gram은 설계 과정에서 얻어진 일반화, 집합화, 관련화의 관계성을 C++ 클래스들 사이의 논리적 결합으로 사상할 수 있도록 표기하고, 추상화의 계층성 및 상속성에 기준하여 적용할 수 있도록 개발하였다.

1) 일반화

유사한 객체들의 가계(families)나 "a kind of," "is a," "category" 또는 "specialization" 관계를 일반화(generalization)라 하는데 이를 C++gram으로 표기할 수 있다. $CT_1, \cdots, CTn(n>2)$을 C++의 클래스 형이라 하면, 수퍼 클래스 CT_1과 부속 클래스 $CT_2, \cdots, CTn$은 부분집합 관계를 나타내는 일반화가 된다. 이 일반화의 개념은 1) 특수화(specialization), 2) 제한화(restriction)로 간주할 수도 있다. 즉, 수퍼 클래스보다 더 특징적인 서술을 필요로할 때 수퍼 클래스로부터 부속 클래스들을 얻을 수 있다. 이것은 특수화의 개념과 같다. 또 수퍼 클래스를 먼저 정의하여 가능한 객체들 집합의 토대를 마련한 후 그 집합으로 부터 부분 집합을 정의하는 방식도 가능하며, 이것은 제한화의 개념과 같다.

일반화는 (subclass) is a kind of (superclass)
(superclass) can be a (subclass1) or (subclass2)

라 해석될 수 있으므로,[15] 사용자의 요구사항 명세서에서 다음 예와 같은 사항을 발견하면 일반화의 사상으로 설계 사항을 표기한다.

예) Truck is a kind of Vehicle.
Bicycle is a kind of Vehicle.
Vechicle can be a Truck or a Bicycle.

① 클래스 관점

클래스 관점의 일반화는 상속성이 설계될 때 단계에서 객체의 성격에 관한 정형적인 추론을 지원함과 동시에 프로그램 개발시 프로그래머의 비정형적 추론에 도움이 된다.

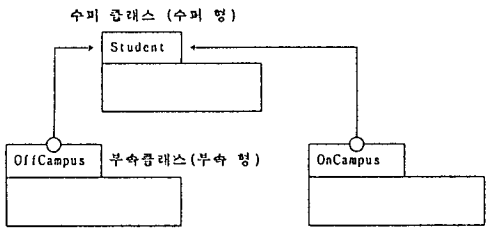

그림 6. 일반화의 표기 예(클래스 관점)
Fig. 6. Example of generalization(a class point of view).

설계 과정에서 클래스들이 공통 특징들을 공유하도록 표기될 수 있고 더 일반적인 수퍼 클래스를 만들기 위해 공통성이 표기될 수 있다. 이와 반대로 어떤 형태보다 더 서술이 필요하면 기존의 클래스를 특수화하는 부속 클래스의 생성도 가능하다.

② 메소드 관점

C++gram으로는 메소드(멤버함수) 관점의 일반화도 표현할 수 있다. 수퍼 클래스의 속성, 메소드와의 관계성은 일반화를 통하여 상속된다. 즉, 수퍼 클래스에 적용된 의미론은 부속 클래스에도 마찬가지로 적용된다.

2) 집합화

클래스 형을 $CT_1, \cdots, CTn$ ($n<\_2$)이라 하자. aggregate-type CT_1과 component-type $CT_2, \cdots, CTn$을 가진 컴포넌트 관계를 집합화(aggregation)로 정의하면, 객체들의 그룹이나 "a part of," "has a part" 관계가 표현되며 이 개념을 C++gram으로 표기할 수 있다. 요구사항 명세서에 다음과 같은 형태의 문장 또는 문맥을 발견하면, 설계 지침을 작성할 때나 방법론 적용에서 얻어진 예비 설계 지침을 상세 설계로 이동하기 위해 설계 사상을 다이아그램으로 표기할 때에는 집합화로 표기할 수 있다.

예) 〈component〉 is part of 〈assembly〉.
〈assembly〉 contains 〈cardinalityl〉〈component
〈component1〉 and
〈cardinality2〉 〈component2〉…

① 클래스 관점

조립품(assembly)과 컴포넌트 역할을 하는 클래스들의 관계 예는 다음과 같다.

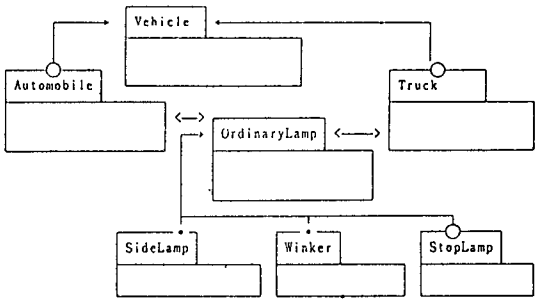

그림 7. 집합화의 표기 예(클래스 관점)
Fig. 7. Example of aggregation(a class point of view).

② 메소드 관점

집합화의 의미에 근거하면 클래스 형이 CT_1이 된 클래스로부터 유도된 부속 클래스 역시 그 메소드의 집합화를 표현할 수 있다. 예를 들어 메소드 A가 조립품 역할을 하고 메소드 2와 3이 컴포넌트의 역할임을 C++gram으로 표기하면 그림8이 된다.

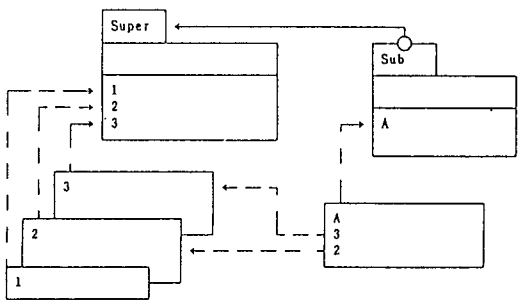

그림 8. 집합화의 표기 예(메소드 관점)
Fig. 8. Example of aggregation (methods point of view)

3) 관련화

CT_1, CT_2를 클래스의 형이라 하고 set-type CT_1과 member-type CT_2가 powerset 관계를 나타낼 때를 관련화(association)라 하자. 즉, 클래스 형 CT_1의 각 객체는 클래스형 CT_2의 객체들의 집합이다. 요구사항으로 부터 C++ 프로그래밍 언어가 보유하

고 있는 정의가 아닌 사용자 고유의 정의를 생성해야 할 때 이 관련화의 관계성을 이용한다. C++gram은 C++ 언어 자체가 직접 지원하지 않는 관련화 개념을 그 표기로써 간접 지원한다. 이는 C++ 클래스 속성간에서는 교환이 되므로 가능하다. 그림 9는 관련화를 표기한 C++gram의 한 예이다.

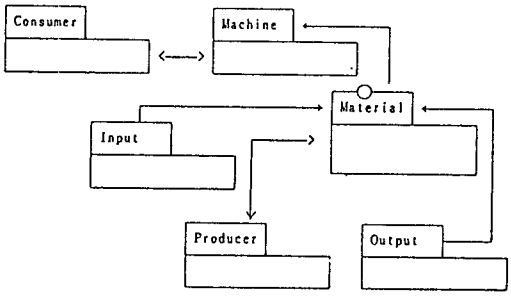

그림 9. 관련화의 표기 예
Fig. 9. Example of association.

관련화는 〈role〉 of 〈class1〉 is 〈cardinality1〉〈class2〉로 해석할 수 있으므로,[15] 명세서에 다음과 같은 사항이 발견되면 집합화 개념에 의한 설계 표기를 하면 된다.

예) Input of Machine is many Materials.
Output of Machine is many Materials.
Consumer of Material is a Machine.
Producer of Material is a Machine.

4) 추상화 계층성과 상속성

객체지향 paradigm은 부속 클래스를 통하여 형의 계층성을 제공한다. 이 관계는 점진적인 문제 해결을 가능케 한다. 이것의 다이아그램 표기법은 수퍼 클래스와 유도된 클래스와의 계층성, 즉, C++의 클래스 메카니즘을 충분히 활용하고 있다. 이것은 외부 명세서로부터 클래스의 관점에서 프로그램의 초기 모델링이 가능하도록 하며 응용 영역에 관한 세부적인 개념들은 보다 더 구체화된 유도 클래스를 작성하면서 생성할 수 있게 한다. 이러한 새로운 부속 클래스터들로 부터 얻어진 객체들은 소프트웨어 구조의 토대를 형성하게 된다. 이 점이 C++gram 표기법이 견지하는 클래스 및 메소드 실현을 위한 추상화 계층성이다. 본 논문에서 제안한 다이아그램은 메소드의 실현을 위하여 추상화 계층의 관점을 견지한다. 이것은 실제적인 형 계층 사상에 완전하게 대응되는 것은 아니다. 그렇지만 형 계층의 기본 원리를 이용하였으며 작업결과 얻은 클래스 및 메소드들은 프로그래머의 입장에서 계층을 지정할 때 수퍼 클래스로부터 부속 클래스로 상속되는 것을 가능케하는 구현의 범위까지 설계 지침을 세부화 하는데 유용한 역할을 하게 된다. 즉, 구현의 편의를 고려한 것이다. 또, 이것은 재사용 라이브러리(reusable library)에 멤버 함수를 저장하거나 기존의 저장된 멤버함수를 재사용할 수 있는 토대가 된다.

상속성은 형 계층과 밀접한 개념이다. 부속 클래스는 클래스(들)의 여러 특징들을 공유하거나 상속받을 수 있다. 이를 C++gram으로 표기함으로써 명세의 구조화 메카니즘과 사고의 자연스러움, 상식적 관념에 기반을 둔 명세서의 재사용 수단이 제공된다. 객체의 정의에서 상속성을 사용하면 설계자는 특성을 상속하는 객체와 비교하여 새로운 것만을 지정하면 된다.

2. 다이아그래밍 툴

Ⅱ-2절에서의 표기법 제안을 위한 요구사항에 만족하는 CASE tool을 구현하였다. 다이아그램의 논리적 설계 layer가 물리적 스크린 layer와 사상되는 세부 원리는 다음과 같다.

사용한 컴퓨터 환경은 IBM호환 기종인 삼보 Trigem 286+상이며, EGA board가 장착된 이 컴퓨터상에서 다이아그램화 도구를 작성하였다. 또한 Turbo-C (2.0version)를 구현 언어로 이용하여 프로그램을 작성하였다.

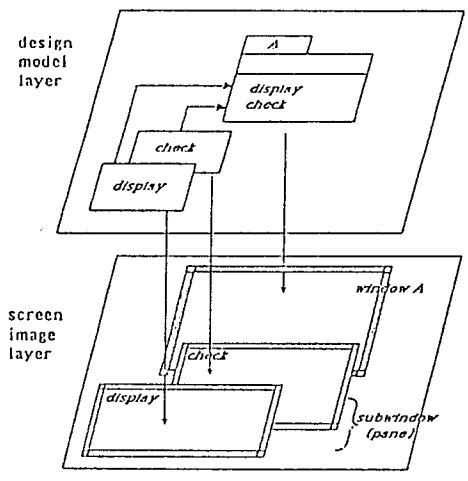

그림 10. 다이화그램화 기법의 세부 계층
Fig. 10. The detailed layer of diagramming technique.

기본 자료 구조는 WINDOW로서 C++gram의 클래스 레이블, 속성부(비공개부), 멤버함수(메소드), 메시지 처리기 등이 모두 window (또는 pane)의 형태로 취급된다.

WINDOW의 자료구조는 다음과 같다.

```
struct_window {
        int_windvisi;          int_wndhd;
        char *_wndasve;        char *_wndtdtlt;
        int_wndx;              int_wndy;
        int_wndw;              int_wndh;
        int_cursor;            int btype;
        int wcolor[4];         int_prenorm;
        struct_window *_next; struct_window *_prev;
} WINDOW;
```

이 WINDOW는 클래스 레이블, 멤버함수, 속성 표현부(비공개부), 메시지 처리기에 모두 적용되는데 각각의 특성과 크기가 다르므로 윈도우 설정함수가 필요하며, 이렇게 설정된 윈도우를 통하여 설계 및 코딩 수정시에 필요한 텍스트를 입력해야 하므로 텍스트 자료 엔트리용 윈도우를 설정하였다. 생성된 다이아그램과 그 내부의 텍스트는 각각 클래스 명의 화일로 저장되어 라이브러리를 형성하게 된다. 이때 각 다이아그램의 화면상의 위치도 별도의 좌표 화일에 저장되어 전체 클래스의 배치도나 상호 관계를 알아볼 수 있으며, 수정 및 재사용이 가능하다.

주 프로그램의 실행시 얻어지는 메뉴는 설계를 위한 Design, 메시지 처리에 관한 시나리오를 위한 Scenario, 다이아그램으로 부터 C++ 원시코드를 얻어내기 위한 Transform/Coding, 이렇게 하여 얻어진 코드화된 내용의 추가 또는 수정을 위한 Editing, 컴파일을 위한 Compile, 실행을 위한 Run, 환경설정을 위한 Miscellany 작업 중지를 위한 Quit 등이다.

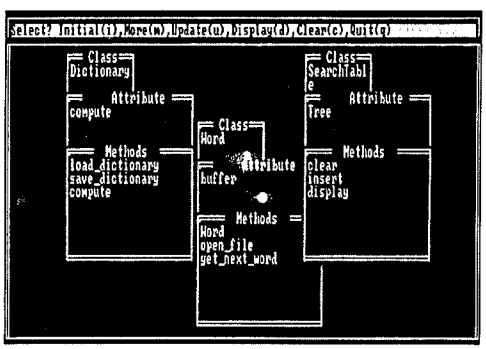

그림11. CASE tool의 작업 화면
Fig. 11. Working screen of CASE tool.

이들 각 메뉴들이 별도로 수행될 수 있도록 다중 타스킹(multi tasking)으로 구성하였다. 따라서 주 프로그램의 실행은 부모 프로세스(parent process)의 생성 및 처리중에 자식 프로세스(child process)가 생성 및 처리된 후 다시 부모 프로세스로 변환되도록 구성하였다. 그림11은 C++gram을 위한 CASE tool의 작동 화면의 예이다.

Ⅳ. 분석 및 평가

본 논문에서 제안한 C++gram의 적용 범위는 표 1과 같다.

표 1. 다이아그래밍 기법의 적용범위 비교
Table 1. Application area of diagramming techniques.

요구사항 분석 및 정의	소프트웨어 설계	생산		
		세부설계	코딩	단위 테스트
PDL ↔				
SADT →	→			
SREM ↔				
JSD ←			→	
	SD ↔			
	JPS ↔		→	
	HDM ↔			→
		SP ↔		
	BOOCH,OD,MDC ↔	OMT, SOM		
		C-B ↔	→	
본 논문의 제안 도형	C++gram ↔		→	

C++gram이 설계와 코딩의 모든 범위에 가장 효율적으로 적용될 조건은 자동화 도구의 지원하에서 다이아그램이 설계와 프로그램 개발에 같이 적용될 때이며, 이때 최대의 재사용도를 얻게 된다.

절차 설계에 대한 표기법은 올바르게 사용된다면 설계 과정에서 매우 귀중한 도움이 되며, 최상의 표기법이라 하더라도 서투른 적용시에는 그 반대 결과가 될 것이다라는 전제하에서 평가하였다. 설계에 대한 다이아그램 표기법의 일반적인 특성은 1) 이해력이 좋으며 검토가 용이한 절차적 표현을 유도해야 하며, 2) 코딩능력을 향상할 수 있는 것으로 설계로부터 자연스럽게 얻어지는 부산물이 되어야 한다. 3)

설계에 대한 다이아그램 표기는 항상 프로그램을 올바르게 표현할 수 있게하는 표현력을 가지고 유지보수가 용이해야 한다.[21] 이러한 일반적인 속성에 최적한 표기법은 무엇인가에 대한 논란은 계속되고 있다. 그 이유는 표기법의 선택은 기술적인 속성보다는 인간적인 요인에 더 밀접한 관련이 있기 때문이다. 설계에 더한 제안된 다이아그래밍 기법의 평가 기준이 된 세부 사항은 표2와 같다.

표 2. C++gram의 평가
Table 2. Evaluation of C++gram.

기 준	C++gram과의 대응 내용
모듈화	C++의 클래스 자체가 프로그램을 모듈화 하는 강력한 도구이므로 클래스 중심의 표현은 모듈화를 지원함. 이 사항은 다른 객체 지향 표기법들도 동일.
간편성	문제 영역의 객체들 나타내는 클래스와 메소드 중심의 기본 도형 사용.
수정력	CASE 도구 구축시 클래스 도형은 그 자체가 윈도우로 구성되어 윈도우의 삽입/삭제 또는 절차 명세의 변경력이 용이함. 단 전체 도형의 네크워크형 및 계층성의 수정법을 위한 편집기 개발이 필요.
기계 판독성	도형 자체가 윈도우로 간주되도록 구현되면 작성된 명세는 시제품(prototype)으로 재빠른 변형이 가능.
자료 표현력	지역자료는 모두 클래스 도형 내부에 표현되며, 전역 자료는 메시지 처리기로 처리되어 세부 설계후의 코드 변환 용이(이때 특별한 표기법은 사용하지 않으며 언어 보유의 자료 표현력을 그대로 사용함).
논리 검증력	설계(예비 설계→상세 설계)과정 반복후 얻어진 절차 논리를 표현한 도형 및 도형 내부 명세는 C++처리 환경에서 재빠른 시제품 처리로 그 논리의 검증을 시도함. 도형 자체의 논리 검증을 위한 시뮬레이터의 개발이 필요함.
코드화 능력	도형 내부에 기재된 절차 명세는 직접 코드화로 연결됨

클래스 자체가 프로그램을 모듈화 하는 강력한 도구이므로 클래스를 중심으로 표현된 다이아그래밍기법은 모듈화의 표현도구가 된다.[22,23] 이것은 클래스 정의를 가지고 사용자 정의의 새로운 형을 생성할 수 있기 때문에 객체 지향 개념에 익숙치 않은 프로그래머 (초보자 또는 절차 지향 언어에 익숙해 있는 자)에게 대규모 프로그램을 다룰 수 있는 능력을 주게 된다.

CASE는 소프트웨어 수명주기의 자동화를 강조하기 위한 통합도구(integrated tool)와 방법론과의 결합이다. 이 기술의 궁극적 목표가 일련의 통합 도구로써 전 소프트웨어 수명주기를 자동화 하는 것이지만 현재는 도달 과제로 남아 있다. 이러한 CASE기술 중 구조화 다이아그램과 그림으로 나타내는 명세의 생성을 위한 다이아그래밍은 소프트웨어 개발 작업을 위한 CASE 도구중의 하나로 중요하다.[2] 본 연구를 토대로 구현할 수 있는 CASE 도구의 범주는 툴키트(toolkits) 형태가 된다.

V. 결 론

초보자나 절차적 개념에 익숙한 개발자가 객체 지향 개념을 일관되게 견지하면서 C++ 프로그램을 개발한다는 것은 쉬운 일이 아니다. 따라서 본 논문에서는 절차적 개념으로 부터 객체 지향 개념으로의 사상 변환을 위해 다이아그래밍 툴을 구축한 결과를 논의하였다. 본 연구의 중요한 결과는 다음과 같다.

(1) 상위 추상화 레벨에서만 사용되던 다이아그래밍 기법을 세부 설계시에도 그대로 적용이 가능하도록 개발하였다.

(2) 클래스, 메소드, 메시지 전달을 위주로 표현하는 표기법의 사용은 객체 지향 설계의 개념과 과정 파악에 적합한 정보 제공자의 역할을 하였다.

(3) 다이아그램의 표기 의미를 클래스 및 메소드 관점의 논리적 관계성 원리로 규명하여 명세서로 부터 클래스 및 메소드를 추출하는데 도움이 되도록 하여 설계 지침의 세부화에 유용하였다.

본 논문은 C++에서의 객체지향 설계를 위한 CASE 툴 개발의 시작점으로서 중요한 역할을 하고 있다. 이 결과를 보다 효과적으로 사용할 수 있도록 하기 위한 향후 연구 과제는 문서화 및 객체에 관한 디버깅 정보를 표시해 주기 위한 객체 표시기, 완성된 설계의 재빠른 시제품화를 위한 시뮬레이터 등이다.

參 考 文 献

[1] I. Sommerville, "Software engineering," Third Ed. Addison-Wesely ,1989.
[2] C. Mc Clure, Case is software automation prentice-Hall, 1989.
[3] Raymond T. Yeh, P.A. Ng, Nordern Software Engineering Foundations and, Current Perspectives, Van Nostrand Reinhold pp. 23-51, 1990.
[4] R.J. Abbott, "Program design by informal

english descriptions," *CACM*, vol. 26, no. 11, pp. 882-894, Nov. 1983.
[5] G.E. Peterson, "Tutorial: object-oriented computing," IEEE Computer Society
[6] R. Fairley, *Software Engineering Concepts*, McGraw-Hill 1985.
[7] G. Booch, "Object-Oriented Design," *Ada. Letters*, vol. 1, no. 3, pp. 64-76, (Mar./Apr.), 1982.
[8] G. Booch, "Software engineering with Ada," Benjamin-Cummings, 1983.
[9] G. Booch, "Object-oriented development," *IEEE Trans.* on S/E, pp. 211-221, vol. SE-12, no. 2, Feb. 1986.
[10] G. Booch, Software Components with Ada (Structures, Tools, and Subsystems), Denjamin/Cummings, 1987.
[11] E. Seidewitz and N. Stark, "Towards a general object-oriented software development methodology," Proc. of First Int'l Conference on Ada Programming Application for the NASA Space Station, p. D. 4.6.1-D.4.6.14, 1986.
[12] R.F. Sincovec and R.S. Wiener, "Modular software construction and object-oriented design using Ada," *Journal of Pascal, Ada,* and Modular-2, Mar./Apr., pp. 50-64, 1984.
[13] R. Wiener and R. Shincovec, Software Engineering with Modula-2 and Ada, John Wiley and Son, 1984.
[14] W. Cunningham and K. Beck, "A diagram for object-oriented programs," OOPSLA'86 Proc., pp. 361-367, 1986.
[15] M.E. S. Loomis and et al, "An object modeling tecnique for conceptual design," ECOOP'87, ibid, pp. 192-202, 1987.
[16] R.K. Ege, et al., "An object-oriented design environment," TOOLS '89 proc., pp. 49-58, 1989.
[17] M. Lai, "HyperHood++:an object-oriented design tool for developments in object-oriented programming language," *ibid.*, pp. 295-308, 1989.
[18] M. Barstschi, et al., "Information systems design and prototyping using and object-oriented software engineering environment," *Ibid.*, pp. 423-436, 1989.
[19] P.D. O.Brien, D.C. Halbert, and M.F. Kilian, "The trellis programming environment," *OOPSLA '87 proc.*, pp. 91-102, 1987.
[20] 하수철, C++ 객체지향 프로그래밍을 위한 다이아그래밍 기법, 홍익 대학원 Ph. D. Thesis 1990.
[21] R.S. Pressman, "Software engineering: A practitioner's approach," 2ed., McGRAW-HILL, 1987.
[22] Soo Cheol HA, Object-oriented Modeling for program design in C++", Proc. of JTC-CSCC'89 (Sapporo Japan), pp. 329-334, June. 1989.
[23] 하수철, 원유헌, "C++를 위한 객체 지향적 설계 방법론과 다이아그램을 사용한 개발 기법에 관한 연구," 정보과학회 학술발표 논문집 vol. 16, no. 2, pp. 139－142, 1989.
[24] 하수철, 원유헌, "객체지향 설계에 관한 연구," 정보과학회 학술발표 논문집, vol. 16, no. 1, pp. 93－96, 1989.

著 者 紹 介

河 秀 澈(正會員)

1981年 2月 홍익대학교 공과대학 전자계산학과 졸업(이학사). 1986年 2月 홍익대학교 공과대학 전자계산학과 이학석사. 1990年 8月 홍익대학교 공과대학 전자계산학과 이학박사. 1981年～1984年 Army Logistics Command EDPS System Analyst. 1987年～현재 대전대학교 공과대학 전자계산학과 조교수. 주관심분야는 프로그래밍 언어, 소프트웨어 공학, 인공지능언어, 객체지향방법론 등임.

元 裕 憲(正會員)

1972年 2月 성균관대학교 수학과 졸업(B. S). 1975年 8月 한국과학원 전자계산학과 졸업 (M. S). 1985年 8月 고려대학교 이학박사. 1975年～1976年 한국과학기술연구소 연구원. 1986年 ～1987年 R, P. I 객원교수. 1976年～현재 홍익대학교 전자계산학교 교수. 주관심분야는 컴파일러, 프로그래밍 언어디자인, 소프트웨어 공학, 분산언어, 객체지향언어, 하드웨어기술언어 등임.