'96 대한전기학회 하계학술대회 논문집 (1996. 7. 22~24)

객체지향 프로그래밍기법을 이용한 공정 감시용 칼라 그래픽 편집기 개발에 관한 연구

박세화
생산기술연구원 생산설비개발센터

A Study on the Development of Color Graphic Editor for Process Monitoring Using Object Oriented Programming

Se-Hwa Park
Industrial Engineering Technology Center, KAITECH

Abstract

Monitoring system is essential part in control system to monitor the circumstances of the industrial equipments. Color graphics is generally introduced in the monitoring system for the purpose of effective human interfaces. Hence, color graphic editor is under the implementation to draw graphic elements easily which are utilized in the monitoring situation. In this study, OOP(object oriented programming) is applied in the programming of the software. OOP enables systematic design of algorithm, easy management of the software and easy extension of additional functions. It is reported that the software is under the implementation. Therefore, preliminary structure of the software is briefly discussed in this paper.

1. 서론

최근, 발전소나 소각로, 하수처리 시설 등 에너지·환경 관련 설비의 도입에 관한 관심이 사회적으로 고조되고 있는 바, 이들 설비의 계통을 칼라 그래픽으로 나타내면 한 눈에 전 공정을 쉽게 이해할 수 있게 된다. 따라서, 이런 산업 설비의 운전 상황을 칼라 그래픽 화면을 통해 효과적으로 감시하기 위한 모니터링 시스템(monitoring system)은 산업 설비에 있어서 필수적이다.

모니터링 시스템은 수많은 그래픽 화면을 통해, 센싱된 값들을 다양한 형태로 나타내 주기도 할뿐만 아니라 공정의 이상 상태를 화면에 표시해 주기도 한다. 그런데, 모니터링되는 수많은 세부 그래픽 화면은 간단한 선(line)이나 원(circle), 다각형, 글자(text) 등의 조합을 통해 이루어지며, 이들 세부 그래픽 화면을 구성할 때 그래픽 편집기를 이용하면 짧은 시간에 효과적으로 화면을 설계할 수가 있겠다[1].

이런 공정 감시를 위한 칼라 그래픽 편집기(color graphic editor)로써 상용화된 제품은 여러 가지 있으나, 전용 모니터링 시스템만을 위해 사용할 수 있거나 또는 특정 칼라 그래픽 편집기가 생성해 낸 출력 파일(file)을 외부에서 알기가 어렵게 되어 있는 등 호환성이 거의 없는 경우가 대부분이다.

한편, 마이크로 소프트사의 MS-Windows는 MS-DOS 환경 하에서 딱딱한 텍스트 위주의 작업을 벗어나, GUI(graphic user interface)라는 개념이 도입되어 1992년 이후 크게 각광 받는 운영체제로써, 전세계적으로 수천만 카피(copy)가 팔렸으며, 기존의 MS-Windows version 3.1을 크게 개선한 Windows 95는 PC(personal computer)의 활용도를 상당히 높임으로써, MS-DOS를 완전히 대체하고 PC의 새로운 표준으로 자리잡게 될 것이라는 관측이 지배적이다. 즉, 기존의 공장 자동화 설비나 공정 감시 설비의 운전 감시 시스템을 MS-Windows 운영체제 환경 하에서 고성능 화된 PC를 이용하여 보다 저렴한 비용으로 원하는 목적을 달성하는 것이 가능하다.

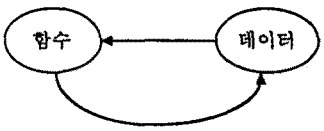
따라서, 본 연구에서는 앞으로 산업계에서 새로운 표준으로 자리 잡게 될 MS-Windows 환경 하에서 산업 설비의 운전 감시 모니터링 시스템을 위해 필요한 수많은 세부 그래픽 공정 화면을 손쉽게 구성하여 Ascii 형식으로 출력해 줌으로써 외부에서 쉽게 데이터를 읽어 처리할 수 있는 칼라 그래픽 편집기를 개발하고자 한다. 이 때, 객체지향 프로그래밍 기법(OOP, Object Oriented Programming)을 이용한다.

2. 객체지향 프로그래밍

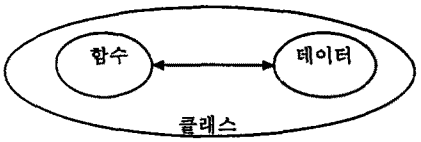
기존의 프로그램 방식은 최종적인 목표를 설정한 후에 이와 관련되는 상세한 동작을 수행하는 함수를 작성해 나간다. 그래서, 프로그램 시에는 쉽게 이해할 수 있는 상세화된 기능을 수행하는 작은 크기의 함수를 작성하게 된다. 이러한 방법을 사용하면 전체적인 작업이 작은 크기의 함수를 작성하는 간단한 것으로 분할이 되어, 초보적인 경험을 가진 사람도 아주 복잡한 프로그램을 쉽게 작성할 수 있게 된다. 그러나, 이러한 프로그래밍 방식에서는 일단 계획을 수립하고 나서 진행중일 때 소프트웨어의 요구 사항이 변경되면 그것을 쉽게 반영할 수 없다. 즉, 프로그램을 작성하는 동안 기본적인 소프트웨어의 구조가 효율적이지 않거나 문제가 있을 경우 작성된 프로그램을 완전히 수정하거나 처음부터 다시 작성해야 한다. 이것은 많은 시간과 노력을 낭비하게 될 뿐만 아니라, 다시 작성하는 프로그램 또한 잠재적으로 문제가 발생할 수 있는 많은 가능성을 증가시킨다. 하지만 소프트웨어는 그것의 이름이 의미하듯이 어떤 상황에서도 융통성 있게 변경될 수 있어야 한다. 이미 작성한 유용한 함수를 언제든지 반복해서 사용할 수 있어야 하고, 계획에 약간의 변화가 가해지더라도 그것을 프로그램이 반영할 수 있

는 융통성 있는 프로그래밍이 되어야 한다.

이런 요구상황을 만족하는 프로그램 방식이 OOP이다. OOP는 프로그램 요소들을 오브젝트로 간주할 수 있도록 하여 그것의 동작 방법이나 개별적인 데이터를 알 필요도 없이 관련된 다양한 함수의 호출 방법을 알면 된다. OOP 기법은 화면의 버튼, 드로잉, 수정 작업 등 어떤 대상에 중점을 두어 처리가 가능한데 이런 항목을 클래스(Class)라고 한다. 클래스는 구조체가 C와 C++에서 독특한 데이터형인 것과 마찬가지로 OOP에서 독특한 것이다. 이는 함수와 그러한 함수가 사용하는 데이터를 결합하거나 캡슐화(encapsulation)한다. 그림 1에서 보듯이 일반적인 프로그래밍 방법을 사용할 때는 그것을 변경하거나 새로운 값을 돌려주는 함수에 데이터가 전달된다. OOP의 클래스는 프로그램 내에서 관련성을 생성하며 데이터와 함수를 캡슐화한다.



(a) 전형적인 프로그래밍 방법



(b) 객체지향 프로그래밍

그림 1. 프로그래밍 방법

요약하면, OOP기법은 클래스를 기본으로 하여 캡슐화가 가능하고 상속성이 있고 다형성을 갖는 특색이 있기 때문에, 결과적으로 프로그램 변경이 용이하고 재사용성이 뛰어나서 공정 감시용 칼라 그래픽 편집기 같이 다양한 기능이 요구되고 복잡한 프로그램의 개발에 적합하다고 할 수 있겠다. 본 연구에서는 구조체 언어인 C를 확장하여 OOP기법을 이용 가능한 C++[2]를 채택하여 개발을 하고자 한다.

3. 공정 감시용 칼라 그래픽 편집기 소프트웨어

공정 감시용 칼라그래픽 편집기의 개발을 위하여 전체적으로 중요한 몇 가지 클래스를 정의하면 다음과 같다.

I) Window Class
전체 프로그램의 중심이 되어 화면 내에서 일어나는 모든 일들을 처리한다. 예로 마우스의 누름이나 움직임에 따라 적절한 작업을 수행할 수 있도록 한다.

II) DrawingTool Class
화면에 다양한 형상이 그려질 수 있도록 툴을 제공한다.

III) EditPicture Class
화면에 그려진 형상들을 편집하여 수정할 때 이용된다.

IV) Shape Class
선, 원, 사각형, 문자 등 여러 형상의 기본 데이터를 가지고 실제 화면의 특정 위치에 형상을 그릴 때 이용된다.

Window Class가 중심이 되어 마우스의 누름이나 움직임에 따라 다양한 클래스를 생성하여 여러 기능들을 수행하게 된다. 한 클래스는 그 내에서 또 수많은 다른 클래스를 부르게(call) 되지만, 데이터나 함수의 사용범위가 미리 정해져 있기 때문에, 전체적으로 수많은 데이터와 함수가 존재해도 상관관계를 분명히 알 수 있다.

Shape Class는 전술한 바와 같이 화면에 사각형, 원, 선 등의 형상에 대한 기본 데이터를 각 형상에 맞게 정의하여 그려주거나 편집할 때 이용된다. 즉,

```
class Shape {
  public:
    virtual void DataInsert(TState *state) = 0;
    virtual void Draw(TDC *DC, TPoint scrollPos,float scale)
= 0;
    virtual void Move(TPoint offset, int select) = 0;
    virtual void DrawMark(TDC *DC, TPoint scrollPos ,float
                          scale) = 0;
    virtual void ChangeBrushColor(int color){};
    Shape       *prior;
    Shape       *next;
    int  ID;
          ⋮
  protected:
    TRect       ShapeRegion;
    TPoint      InitPnt;
    TPoint      FinalPnt;
    int         Pencolor;
};
```

와 같이 기본 클래스가 정의되면 선, 사각형은 이로부터 파생이 되어 각각 다음과 같이 정의를 할 수 있겠다.

```
class LineShape : public Shape {
  public:
    virtual void DataInsert(TState *state);
    virtual void Draw(TDC *DC,TPoint scrollPos,float scale);
    virtual void Move(TPoint offset, int select);
    virtual void DrawMark(TDC *DC, TPoint scrollPos,
                          float scale);
  protected:
    int  Penstyle;
    int  Pensize;
```

```
class RectangleShape : public Shape {
  public:
    virtual void DataInsert(TState *state);
    virtual void Draw(TDC *DC, TPoint scrollPos,
                      float scale);
    virtual void Move(TPoint offset, int select);
    virtual void DrawMark(TDC *DC, TPoint scrollPos,
                      float scale);
    void ChangeBrushColor(int color){
       Brushcolor = color;
    }
  protected:
    int  Penstyle;
    int  Pensize;
    int  Brushcolor;
    int  Brushmode;
    int  Brushhatchstyle;
    int  Brushhatchcolor;
};
```

위에서 보듯이 LineShape Class나 RectangleShape Class는 각각에 필요한 데이터를 독자적으로 정의하여 이용을 하고 공통적으로 필요한 데이터는 Shape Class에 정의된 데이터를 이용한다. 이렇게 클래스의 구조를 정한 다음에는 각 Shape에 맞는 함수의 기능을 프로그래밍 하게 되는데, 한 예로 형상을 그리는 작업이 필요할 시에는 다음과 같이 처리 할 수 있겠다.

```
        ⋮
1.    Shape *shape[2];
2.    shape[0] = new LineShape();
3.    shape[1] = new RectangleShape();
        ⋮
4.  for(int i=0; i<2; i++) shape[i]->Draw( ··· );
```

위의 4번째 줄에서 보듯이, 형상을 그릴 때 각 형상에 따라 각기 다른 Draw()함수를 구별하지 않아도 되는 편리함이 있다. 따라서, 새로운 형상을 추가하여 넣고 싶을 경우에는 Shape Class에서 새로운 클래스를 파생하여 적절한 데이터와 함수를 정의하고 원하는 기능을 부여함으로써 손쉽게 기능 확장이 가능하다. 또한, 이미 프로그램된 것에 대한 기능을 바꾸는 경우에도 해당되는 오브젝트에 관계되는 부분만 고치면 되므로 유지보수 측면에서 상당히 유리하다.

4. 화면 구성 예

현재는 공정 감시용 그래픽 편집기의 기본 구조와 부분적인 기능만이 구현되어 있어서 추후에 좀 더 보완이 요구된다. 그림 2는 개발중인 공정감시용 그래픽 편집기의 초기 화면이며, 그림 3은 몇 가지 형상을 조합하여 간단한 그림을 그린 예이다. 프로그램은 BC C++ 4.5에서 OWL(Object Window Library)를 이용하여 개발중이다.

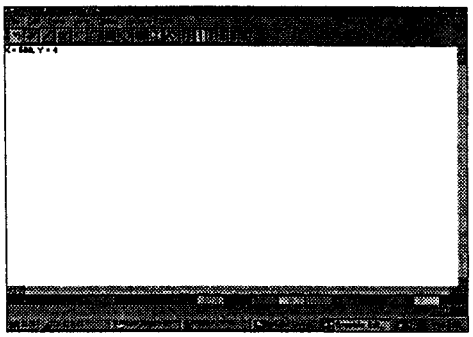


그림 2. 초기 화면

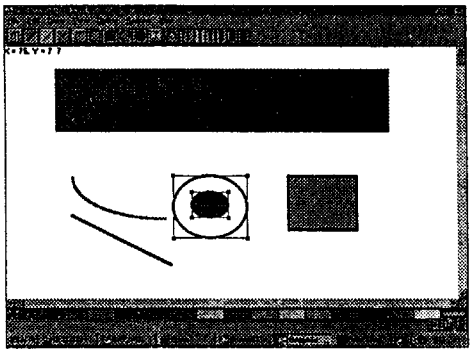


그림 3. 시험 그림 화면

참고 문헌

[1] "발전소 계통 운전 감시용 칼라 그래픽 에디터의 개발", 한국과학기술원 연구 보고서, 1990

[2] Tom Swan, "C++ Primer", Prentice Hall Inc., 1993

[3] Borland C++ 4.5 User's Guide