객체지향 CASE 도구 OODesigner의 플랫폼 이식 사례 연구

홍 의 석† · 김 태 균††

요 약

소프트웨어 개발 기술이 발전함에 따라 객체지향 CASE(Computer Aided Software Engineering) 도구의 중요성이 점차 커지고 있다. 본 논문에서는 객체지향 CASE 도구인 OODesigner[15, 16]를 세가지 플랫폼에서 개발한 공정을 기술하고, 구현된 도구들의 설계 및 구현 결과 유사성을 기술하며, CASE 도구 개발을 위한 설계 및 구현 관점의 일반적인 구조를 제시한다. OODesigner는 초기에 OMT(Object Modeling Technique)를 지원하기 위한 목적으로 1994년부터 개발되기 시작하였으며, 1997년에 Unix 버전이 완성된 이후 UML(Unified Modeling Language)을 지원하는 Java 버전과 Windows 버전의 개발이 이루어졌다. CASE 도구 개발은 Model-View-Controller(MVC) 패러다임을 적용하는 전형적인 예라고 볼 수 있으며 구현 결과 MVC 관점에서 세가지 버전들의 공통된 설계 패턴들을 얻을 수 있었다. 이러한 설계의 유사성들은 해당되는 설계 표기법을 지원하는 다양한 CASE 도구의 개발에 사용될 수 있다.

A Case Study of Platform Migration for an Object-Oriented CASE tool : OODesigner

Euy-Seok Hong† · Tae-Gyun Kim††

ABSTRACT

As software technology has made progress, object-oriented CASE tools have become more important. This paper presents the process and similarity in design and implementation of OODesigner, an object-oriented CASE tool, on three platforms and outlines a kind of generic architecture for the design and the implementation of CASE tools. OODesigner is a tool that was initially developed to support OMT. An initial Unix version has been developed since 1994. In 1997, after the completion of the Unix version, we began developing a Java version and a Windows version supporting UML. The development of a CASE tool is a typical application of the Model-View-Controller(MVC) paradigm. Thus, we obtained a common design pattern among the versions in the MVC point of views. This design similarity can be used to develop several kinds of CASE tools with the corresponding design notations.

1. 서 론

객체지향 개념이 등장한 이후 객체지향에 대한 여러 가지 반론과 다양한 관점들이 제시되었지만 객체지향은 현재 소프트웨어 개발의 여러 분야에서 소프트웨어 위기를 극복할 수 있는 대표적인 패러다임으로 자리잡고 있다[1, 2]. 소프트웨어 엔지니어들은 객체지향의 추상화, 캡슐화, 상속성, 다형성, 분산성 등을 사용하여 보다 높은 질의 소프트웨어를 개발할 수 있으며, 컴포넌트와 설계 패턴 등의 기술을 이용해 소프트웨어 설

※ 본 논문은 한국과학재단 연구비 지원에 의한 결과임(KOSEF 981-0923-123-2)
† 정 회 원 : 안양대학교 영상처리학과 교수
†† 정 회 원 : 부산외국어대학교 컴퓨터공학과 교수
논문접수 : 1999년 11월 19일, 심사완료 : 2000년 9월 7일

계 및 코드의 재사용성을 높일 수 있다.

소프트웨어공학의 관점에서 보았을 때 객체 지향 기술에 대한 연구는 여러 부분에서 행해져 왔다. 우선 객체지향을 소프트웨어 개발 기술에 접목시키기 시작한 객체지향 언어 분야를 들 수 있다. Smalltalk, Object-C, Eiffel, C++, Java 등과 같은 수많은 객체지향 언어들이 개발되었으며 오늘날 사용되고 있다. 또한 Booch의 OOD, Wasserman의 OOSD, Beck과 Cunningham의 CRC, Wirfs Brock의 RDD, Rumbaugh의 OMT, Rational Rose사의 UML과 같은 소프트웨어 개발 방법론들이 등장하여 객체 지향 소프트웨어의 설계 기법으로 이용되고 있다[3-9]. CORBA나 DCOM같은 분산 객체 기술들[10-12]도 각광 받고 있으며 이러한 기술들을 뒷받침하는 개발 도구들인 IPSE, SEE, CASE, meta-CASE 기술들도 발전해왔다[13].

본 연구는 1994년부터 OODesinger라는 객체지향 CASE 도구를 Unix 플랫폼에서 제작하면서 시작되었다. OODesigner의 초기 목표는 OMT 세가지 모델의 제작, 클래스 자원들의 문서화, C++ 코드의 자동 생성 등과 같은 순방향 공학(forward engineering)과 C++ 코드를 자동으로 설계 문서로 변환 해주는 역공학(reverse engineering), C&K 메트릭으로[14] 대표되는 객체지향 메트릭들을 자동 수집해주는 품질 공학(quality engineering), 그리고 자료의 저장 및 검색 기능을 지원하는 것이었다. OODesigner의 Unix 버전은 1996년에 개발 완료 되었으나 시스템의 확장성에 관한 문제가 인식되어 유지보수성을 높이고 다른 플랫폼에 대한 이식성을 높이는 재구조화 작업이 수행되었다[15, 16].

Unix 버전에 대한 재구조화 작업의 완료 후에UML을 지원하는 Java 버전과 Windows 버전을 구현하였으며 개발 이유는 객체 지향 언어 및 설계 기법의 사용 능력을 향상 시키기 위한 것이었다. 즉 객체 지향 기법을 이용하여 UML의 유용성을 확인하고 Java 언어와 Visual C++의 사용 능력을 향상시키는 것이 본 연구의 기본 목적이었으며 결국 본 논문에서와 같은 지속적인 설계 및 구현 연구를 통하여 객체 지향 설계는 근본적으로 특정 언어나 특정 플랫폼에 영향을 받지않는 어플리케이션 영역의 고유의 모델을 제시하는 수단을 제공함을 확인 할 수 있었다.

객체지향 CASE 도구들은 MVC 패러다임[17]의 적용이라는 공통적인 설계 구조를 가지고 있을 뿐만 아니라 객체지향 언어로 제작될 때 구현의 유사성도 가지고 있다. 본 논문에서는 그 동안의 설계 및 정제 과정을 통해 얻어진 세가지 버전들의 유사성을 살펴보고 CASE 도구같이 MVC 패러다임의 적용이 가능한 소프트웨어의 개발을 위해 이용될 수 있는 설계 구조에 대해 기술한다.

2장에서는 각 버전들의 목표와 재구조화, 이식 과정들을 기술하고 3장에서는 몇 가지 예를 통해 세 버전들의 유사성을 기술한다. 각 예들은 MVC 컴포넌트들과 설계 및 구현의 관점에서 구분되어 기술된다. 4장에서는 설계 및 구현 관점에서 도구 개발을 위해 이용될 수 있는 일반적인 구조를 제시하며 5장에서는 결론과 향후 연구 과제에 대해 기술한다.

2. OODesigner

본 장에서는 OODesigner의 세 버전을 각 버전의 구현 목표와 개발 공정을 중심으로 기술한다. 세 버전의 기능들에 대한 목표는 유사하지만 개발 공정에 관련된 목표에는 차이점이 있다.

2.1 Unix 버전

개발 초기 버전은 OMT를 지원하는 데 목표가 있었으며 연구 시작 시에 산출물 목표들(product goals)과 공정 목표들(process goals)이라는 두 가지 형태의 목표들이 설정 되었다. 산출물 목표들은 기능 요구 사항에 해당되며 다음과 같은 내용을 갖는다.

- OMT의 세가지 모델[1] 작성 지원.
- 클래스 자원들의 문서화 지원.
- 설계된 객체 모델의 정보저장소 관리.
- C++ 코드의 자동 생성 및 역공학 기능 지원.
- 재사용을 위한 클래스 저장 및 탐색 기능 지원.
- C++ 프로그램의 메트릭 데이터 수집.

본 연구의 초기 목표 설정 시에 산출물 목표보다는 공점 목표의 설정에 좀더 중요성을 부여하였는데 그 이유는 본 연구가 객체 지향 기술의 적용에 대한 첫번째 시도였기 때문이다. 본 연구의 초기에 설정된 공정 목표들은 다음과 같다.

- 개발팀의 객체지향 설계와 구현 기술을 향상시킨다.
- 객체지향 순환 개발 공정(iterative development

1) 객체 모델, 기능 모델, 동적 모델

process)을 익힌다.

- OODesigner의 추가 개발에 OODesigner를 CASE 도구로 사용한다.
- 향후 산출물의 기능 추가 및 확장, 플랫폼 이식 등을 위해 유지보수성을 높인다.

OODesigner의 첫 버전은 OS-4.1x, X11-R5, Motif-1.2, C++-2.0을 사용하여 Sun 워크스테이션에서 구현되었으며 C++ 코드로 60,000 LOC 정도의 분량으로 구현되었다. 이 도구는 이미 1997년부터 공개된바 있으며 현재도 리눅스 플랫폼 사용자들의 관심을 끌고있다.

초기 버전은 설정된 산출물 목표들을 모두 만족 시켰지만 유지보수성에 관련된 공정 목표들을 만족시키지 못했다. 문제점은 클래스 상속 계층 및 클래스 클러스터들의 배치에 있었으며 이는 당시 객체지향 기술에 대한 이해 부족에 기인한다. 결국 약 일년간의 개발 기간을 투입하여 Unix 버전을 재구조화 하였으며 그 결과로 시스템의 확장성, 이식성, 유지보수성이 향상된 산출물을 얻을 수 있었다[15, 16].

2.2 Java 버전

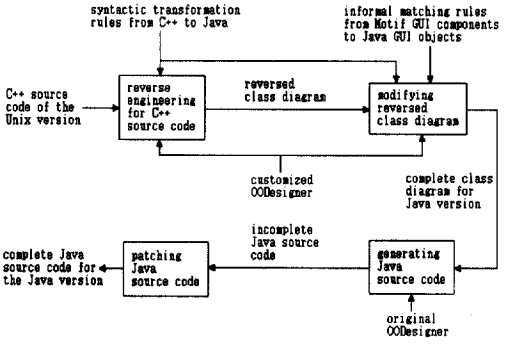
Unix 버전의 재구조화가 완료된 1997년에는 Java와 UML 기술이 주목을 받았으며, 본 연구팀이 참가한 RESORT(Research on object-oriented Software Re-engineering Technology) 프로젝트의 부분 연구로서 UML을 지원하는 CASE 도구의 개발이 필요하였기 때문에 OODesigner의 Unix 버전을 UML을 지원하는 Java 버전으로 이식하는 작업이 수행되었다. Java 버전의 산출물 목표는 Unix 버전의 산출물 목표와 유사하며, 공정 목표의 관점에서는 다음과 같은 내용이 설정되었다.

- Java 언어의 사용 능력을 습득한다.
- 플랫폼 이식을 위해 Unix 버전 OODesigner를 CASE 도구로 이용한다.
- C++과 Java 프로그래밍의 유사성을 인식한다.

Unix 버전에는 이미 C++ 코드의 역공학 기능과 Java 코드 생성 기능이 구현되어 있었으므로 이 기능들을 C++ 코드를 Java 코드로 변환하는 데 사용하였다. Java 버전으로의 이식 프로세스는 다음과 같이 이루어졌다.

① Unix 버전의 소스 코드를 역공학 도구 기능을 사용하여 클래스 다이어그램들로 변환한다.
② 클래스 다이어그램에 새로운 클래스들을 추가하거나 내용을 변경한다. 예를 들면 Motif GUI 컴포넌트들을 해당되는 Java GUI 객체들로 변환한다.
③ Java 코드 자동 생성 기능을 이용하여 수정된 클래스 다이어그램으로부터 Java 코드를 생성한다.
④ 생성된 Java 코드를 이용하여 구현 작업을 완료한다.

(그림 1)은 Java 버전으로의 이식 공정을 SADT[18] 형태로 나타낸 것이다. 이식 공정에서 주목할만한 점은 이미 구현된 Unix 버전을 CASE 도구로 사용하였기 때문에 생산성을 매우 높일 수 있었다는 것이다. 이식 공정 초기 단계에서는 1 MM의 노력으로 53개의 클래스를 가진 약 20,000 LOC 정도의 코드를 작성하였다.



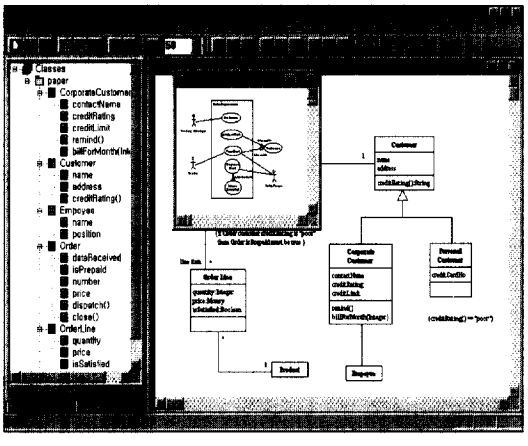
(그림 1) Java 버전으로의 이식 프로세스

2.3 Windows 버전

Windows 환경의 사용자가 급격히 증가하였기 때문에Java 버전의 이식 작업과 함께 Windows 버전으로의 이식 작업이 이루어졌다. Windows 버전의 산출물 목표는 Java 버전의 경우와 유사하며 공정 목표는 다음과 같다.

- Visual C++ 개발 플랫폼 사용 방식을 습득한다.
- Java 버전으로 이식 경우와 같이 Windows버전으로의 이식 공정에서도 OODesigner를 CASE 도구로 사용한다.
- 플랫폼 독립적이고 재사용 가능한 클래스들을 추출한다.

이식 과정을 통해 Unix 버전의 MVC 객체 모델들이 상당 수 Windows버전에 재사용 가능함을 발견할 수 있었다. 비록 두 버전의 View 컴포넌트와 Controller 컴포넌트에는 약간의 차이점이 있었지만 근본적인 기능의 측면에서 매우 유사함을 발견하였다. 이러한 유사성은 Windows 버전 개발의 생산성 향상에 큰 도움이 되었으며 이식 작업의 초기에 1 MM의 노력으로 57개의 클래스를 가진 약 20,000 LOC의 코드를 작성할 수 있었다. 그 동안의 개발 과정을 통하여 구현된 세 버전 중에서 Unix 버전과 Windows 버전은 http://www.comp.pufs.ac.kr/~ktg/ood.shtml 과 ftp://203.230.73.24/OOD 사이트를 통해 배포되고 있다. (그림 2)는 Windows 버전의 사용 화면이다.



(그림 2) Windows 버전 OODesigner의 사용 화면

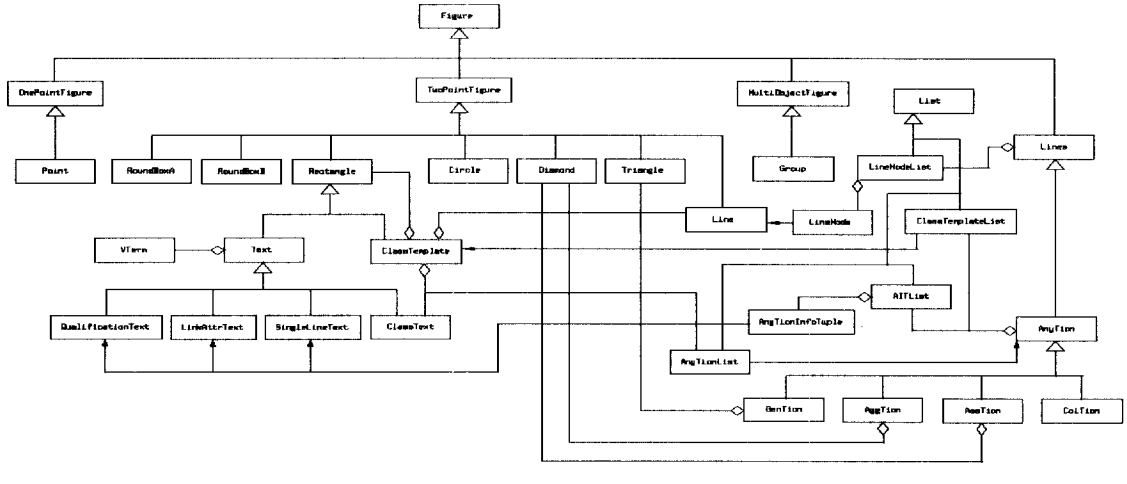
3. 버전의 유사성

MVC 패러다임에서는 시스템이 세 개의 컴포넌트로 나뉘어진다. Model 컴포넌트는 어플리케이션 영역의 객체를 모델링한 부분이며 View 컴포넌트는 데이터를 화면에 보여주는 기능을 담당하고 Controller 컴포넌트는 GUI 이벤트들과 모델 사이의 상호작용을 처리하는 부분이다. CASE 도구는 사용자에게 다이어그램들을 보여주는 View 컴포넌트와 주어진 표기법으로 다이어그램을 제작할 수 있게 하는 Controller 컴포넌트를 가지므로 CASE 도구의 개발은 MVC 패러다임을 이용하는 전형적인 예라 할 수 있다. 본 장에서는 MVC컴포넌트들의 관점에서 OODesigner 세 버전들의 유사성들을 기술한다.

3.1 Model 컴포넌트 관점

3.1.1. 설계 관점

OODesigner의 Model 컴포넌트는 가장 플랫폼 독립적이다. Model 컴포넌트는 OMT와 UML 표기법들을 표현하기 위한 객체 모델들로 구성되며 세 개 버전들에서 공통적으로 사용된다. (그림 3)은 클래스 다이어그램의 심볼들을 모델링한 클래스 다이어그램이며 구체화된 클래스(concrete class)들은 OMT 심볼들을 나타낸다. 예를 들면 우측 하단의 GenTion, AggTion, AssTion, ColTion은 각각 generalization, aggregation, association, collaboration을 나타낸다. 이 모델은 세 버전에 수정 없이 사용되었다.



(그림 3) Model 컴포넌트 예

3.1.2 구현 관점

이식 프로세스 결과 C++와 Java로 구현한 Model 컴포넌트는 매우 강한 구현 유사성을 지니고 있었다. (그림 4)의 예는 두 버전의 구현 유사성을 보여준다.

```
// an example of C++ member function within
// ClassTemplate class
int ClassTemplate : : minimized_width(int& maxchar)
{
      _classname->uncentering(TRUE);
      if (_NofCharsinLine>_classname->width()) {
            maxchar = _NofCharsinLine;
      } else {
            maxchar = _classname->width();
      }
      return (_deltaH*maxchar+2*_GapX+_deltaH);
}

// the correponding Java member function
public final class ClassTemplate
extends Box implements ClassLike {
   ...
private int minimized_width(IntVar maxchar) {
      _classname.uncentering(true);
      if (_NofCharsinLine>_classname.width()) {
            maxchar.v = _NofCharsinLine;
      } else {
            maxchar.v = _classname.width();
      }
      return(_deltaH*maxchar.v+2*_GapX+_deltaH);
}
   ...
}
```

(그림 4) C++과 Java 버전의 구현 유사성 예

3.2 View 컴포넌트 관점

3.2.1 설계 관점

OODesigner의 View 컴포넌트는 모델의 데이터를 화면에 그려주는 역할을 한다. View 컴포넌트와 관련된 객체들은 이벤트 핸들러들과 그래픽 콘텍스트, 폰트들이다. 이러한 객체들은 각 플랫폼에서 내장형 기능으로 제공되며 View 컴포넌트의 기능들이 Controller 컴포넌트에 분산되어 있기 때문에 View 컴포넌트에 대한 객체 모델들은 작성될 필요가 없다. 세개의 버전이 같은 기능을 수행하기 위해 비슷한 이름의 객체들을 사용하기 때문에 Unix 버전을 다른 버전으로 이식하기 위해 비슷한 이름의 내장 객체들로 대치되었다. <표 1>은 세 개의 버전 사이에 대응되는 View 컴포넌트 객체들을 나타낸 것이다. 비록 그래픽 객체들의 속성들간에 약간의 차이점은 있었지만 View 컴포넌트 해당 객체들의 이름과 멤버 함수 이름들을 변경하여 다른 버전에 재사용하였다.

<표 1> 버전간에 대응되는 View 컴포넌트 객체들

기 능	X Toolkit	Java AWT	MFC
graphics	GC	Graphics	CDC
font	Font	Font	Cfont
clipping	Region	Shape	CRgn
event handler for repainting screen	user defined event handler	paint() in Canvas class	OnDraw() in CView class

3.2.2 구현 관점

Model 컴포넌트 경우와 같이 Unix 버전의 소스 코드를 다른 버전을 구현하는 데 재사용하였다. 사소한 문법들이 수정되기는 하였지만 이미 존재하는 알고리즘들이나 전체 구조는 변화되지 않았다. (그림 5)는 화면을 다시 그리기 위한 Windows 버전과 Java 버전의 코드이다.

```
// the member function in the Windows version to repaint screen
void COODView : : OnDraw(CDC* pDC)
{
    COODDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CPoint sp = GetScrollPosition();
    if (sp.x != _originX || sp.y != _originY) {
        _originX = sp.x; _originY = sp.y;
        if (_currentFocus != NIL) {
            _currentFocus->make_region();
            _currentFocus->reset_clip_area();
        }
        Figure *all = _figures->get_first();
        while (all != NIL) {
            all->set_in_canvas(all->check_in_region(_canvasRgn));
            if (all->in_canvas()) all->make_region();
            else all->reset_region();
            all = _figures->get_next();
        }
        recalc_clip_area();
    }
    CPen *oldPen = pDC->SelectObject(_plainPen);
    CBrush *oldBrush = pDC->SelectObject(_plainBrush);
    Figure *ptr = _figures->get_first();
    while(ptr != NIL) {
        ptr->draw(pDC);
        ptr = _figures->get_next();
    }
    if (_currentFocus != NIL) {
        if (_editingTag) _currentFocus->draw(pDC);
        else {
```

```
            if (_doNotRubberband) highlight(_currentFocus);
            else rubberbanding(_currentFocus);
        }
    }
    pDC->SelectObject(oldPen);
    pDC->SelectObject(oldBrush);
    _doNotRubberband = FALSE;
}
// the corresponding Java member function
class GraphicController extends Canvas
implements ActionListener, AdjustmentListener{
  ...
    public void paint(Graphics g) {
      int spx = hscrollBar.getValue();
      int spy = vscrollBar.getValue();
      if (spx != _originX || spy != _originY) {
          _originX = spx; _originY = spy;
          if (_currentFocus != null) {
              _currentFocus.make_region();
              _currentFocus.reset_clip_area();
          }
          Figure all = _figures.get_first();
          while (all != null) {
              all.set_in_canvas(all.check_in_region(_canvasRgn));
              if (all.in_canvas()) all.make_region();
              else all.reset_region();
              all = _figures.get_next();
          }
          recalc_clip_area();
      }
      Figure ptr = _figures.get_first();
      while(ptr != null) {
          ptr.draw(g);
          ptr = _figures.get_next();
      }
```

```
      if (_currentFocus != null) {
          if (_editingTag) {
              _currentFocus.draw(g);
              showCaret();
          } else if (_popupflag == true) {
              draw_dots(_currentFocus);
              _popupflag = false;
          } else rubberbanding(_currentFocus);
      }
    }
  ...
}
```

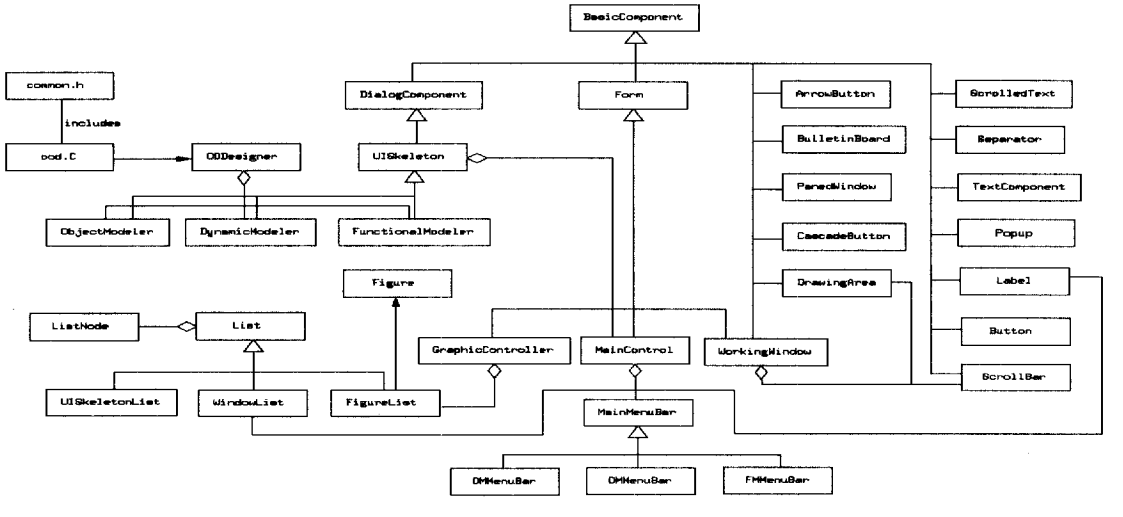
(그림 5) Windows 버전과 Java 버전의 화면 재그리기 예

3.3 Controller 컴포넌트 관점

3.3.1 설계 관점

세 버전의 Model 및 View 컴포넌트 부분은 매우 유사하지만 Controller 컴포넌트는 상대적으로 많은 부분에서 차이가 있다. 그러므로 Unix 버전의 Controller 컴포넌트의 재사용율은 상대적으로 낮았다. Controller 컴포넌트는 물리적인 입력 디바이스들을 다루는 GUI 프레임들을 제공하고 이벤트들을 적당한 View에 대한 메시지들로 변환해주는 역할을 한다. 각 버전간의 주된 차이점은 다음과 같다.

● **Unix 버전** : main() 함수는 어플리케이션 객체와 전역 변수들을 초기화하고 어플리케이션 프레임을 제공하며 Motif widget들을 해당 callback 함수들과 함께 적당한 클래스들로 캡슐화 한다. (그림 6)은 Unix 버전



(그림 6) Unix 버전의 Controller 컴포넌트 객체 모델

의 Controller 컴포넌트를 위한 객체 모델이다. Motif 프로그래밍의 문제점은 이벤트 핸들러들과 callback 함수들을 클래스의 정적 멤버로 선언해야 한다는 것이다. Motif경우 이벤트 핸들러들은 이벤트 발생 시 그 이벤트를 처리할 객체를 찾아야 하며 이벤트 처리 역할을 해당 멤버 함수들에게 전달해야 한다. 이러한 구조는 멤버 함수의 수적 증가를 가져온다.

● **Java 버전** : Java 버전의 경우 어플리케이션 프레임들을 제작하기 위해 Java AWT 클래스들을 사용하였으며, 내장 클래스들은 GUI 컴포넌트 제작 노력을 감소시켰다. (그림 7)은 Java 버전의 Controller 컴포넌트를 위한 객체 모델이다. 그림에서 <<library>>, <<interface>> 스테레오형은 해당 클래스가 내장 클래스와 내장 인터페이스임을 나타낸다. (그림 6)과 (그림 7)은 두 버전의 상속 계층 구조는 다르지만 클래스들의 설계가 상당히 유사함을 나타낸다. Java 프로그래밍에서 이벤트 처리의 장점은 이벤트를 받는 객체를 명시적으로 나타낼 필요가 없다는 것이다. 즉 인터페이스를 구현하는 멤버 함수들이 정적으로 선언될 필요가 없다. 그러므로 이벤트를 실제적으로 다루는 멤버 함수들을 추가할 필요가 없다.

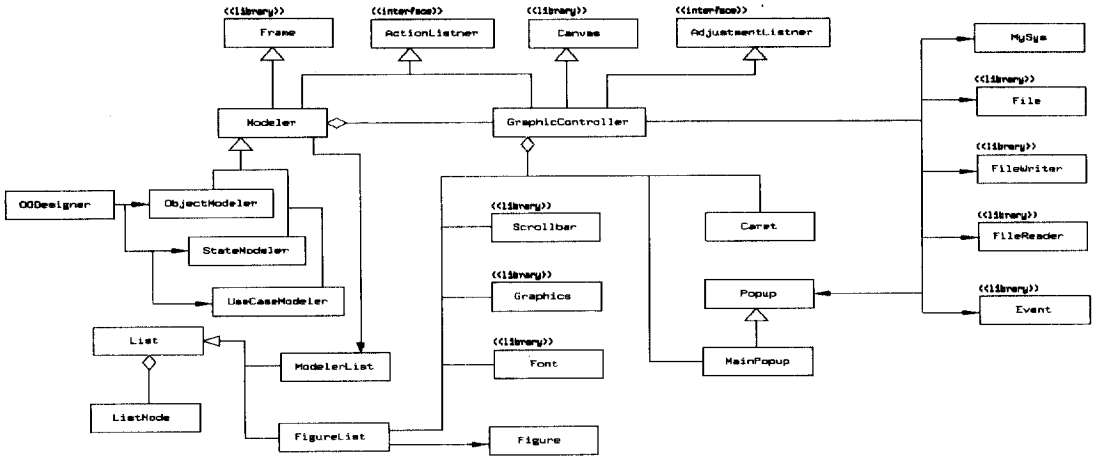
● **Windows버전** : 어플리케이션 프레임들을 제작하기 위해 MFC 라이브러리를 사용하였으며 이벤트 처리기들은 정적으로 선언될 필요 없으나 동적 이벤트 처리를 위해 플래그 변수들을 사용해야 하는 문제점이 있었다.

3.3.2 구현 관점

Model 컴포넌트와 View 컴포넌트와는 달리 Unix 버전의 Controller 컴포넌트의 C++ 코드는 많은 부분 재사용할 수 없었다. 즉 이벤트 처리기 부분에 많은 수정이 가해졌다. (그림 8)은 버튼 관련 이벤트 처리를 위한 Unix 버전과 Windows 버전의 예이다.

```
// an example of event handler of the Unix version
void GraphicController : : start_draw(
        Widget,XtPointer thisController,XEvent *event,BOOL *)
{
    if (event->xbutton.button != Button1) return;
    if (OODesigner : : EditingTag) return;
    GraphicController *controller =
            (GraphicController*) thisController;
    if (controller->_currentDrawingType == NIL) return;
    controller->_isFixed = FALSE;
    XtRemoveEventHandler(controller->_canvas_widget,
            PointerMotionMask,FALSE,
            (XtEventHandler)&GraphicController : :
            fix_pointer, thisController);
    XtRemoveEventHandler(controller->_canvas_widget,
            PointerMotionMask,FALSE,
            (XtEventHandler)&GraphicController : :
            trace_enter, thisController);
    controller->local_start_draw(event);
}

// the corresponding Visual C++ member function
void COODView : : start_draw(CPoint event)
```



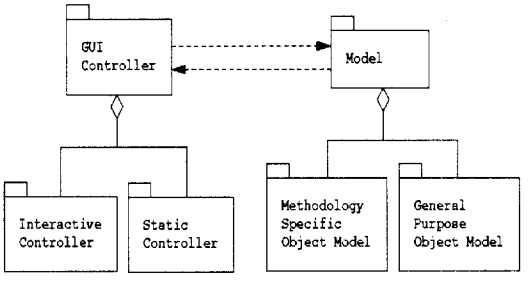
(그림 7) Java 버전의 Controller 컴포넌트 객체 모델

```
{
    if (F_start_draw == FALSE) return;
    if (EditingTag) return;
    if (_currentDrawingType == 0) return;
    _isFixed = FALSE;
    F_fix_pointer = FALSE;
    F_trace_enter = FALSE;
    local_start_draw(event);
}
```

(그림 8) Unix 버전과 Windows 버전의 이벤트 처리 예

4. 객체지향 CASE 도구 구조

본 장에서는 OODesigner의 세가지 버전 개발로 얻어진 경험을 바탕으로 일반적인 객체지향 CASE 도구 설계 구조를 제안한다. 일반적으로 MVC 패러다임에서 View 컴포넌트는 시스템을 구성하는 독립적인 컴포넌트로 여겨지지만 OODesigner 개발 경험에 의하면 View 컴포넌트는 Controller 컴포넌트와 매우 강하게 결합되어 있었다. 즉 View 컴포넌트는 개별적인 객체로 식별되기 보다는 Controller 컴포넌트의 일부 멤버 함수로서 구현될 수 있다. 그러므로 CASE 시스템은 MVC 컴포넌트들보다는 MC 컴포넌트들로 나뉘어 설계되는 것이 바람직하다고 판단된다. MVC 컴포넌트로 시스템을 분석하는 이유는 구현하고자 하는 소프트웨어를 기능적인 관점에서 접근하기 때문이다. 그러나 본 연구의 경험상 시스템을 객체 지향적인 관점에서 설계하고 구현하고자 할 때 시스템이 제공하는 View 기능만을 따로 수행하는 객체를 식별할 필요가 없었다. 이러한 점에서 CASE 도구의 설계를 위한 클래스 다이어그램의 작성을 위해서는 크게 Controller를 위한 패키지와 Model을 위한 패키지를 구성하는 것이 바람직하다. (그림 9)는 본 논문에서 제안하는 CASE 도구 구조를 패키지 별로 나타낸 것이다.



(그림 9) 제안된 CASE 도구 설계 구조

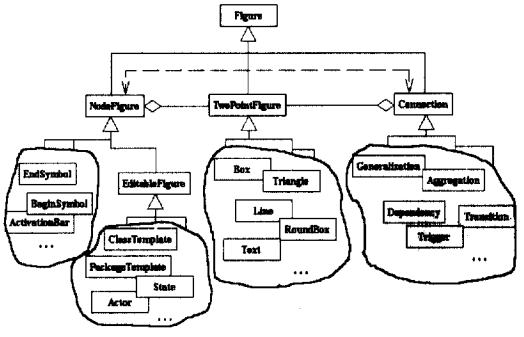
이 구조의 상위 수준에는 Controller 컴포넌트와 Model 컴포넌트에 관련된 두 종류의 패키지가 있다. Controller 패키지는 두가지 종류의 패키지로 구성된다. Interactive Controller 패키지는 입력 장치가 발생시키는 이벤트들을 처리하는 부분이고 부속 기능으로 View 기능을 포함한다. Static Controller 패키지는 풀다운 메뉴 선택에 의한 callback들을 처리하는 기능들로서 사용자와의 상호작용에 의한 것이 아니라 순차적으로 수행되는 것이므로 병행적으로 개발 가능하다. 파일 불러오기, 저장과 코드 자동 생성 및 역공학 기능 등이 Static Controller 패키지로 구현된다.

Model 컴포넌트는 일반적으로 사용되는 Stack, List, OrderedCollection, SymbolTable 등을 포함하는 General Purpose Object Model과 개발하는 도구가 지원할 방법론의 표기들을 나타내는 Methodology Specific Object Model로 구성된다. 전자는 해당 플랫폼에서 내장형으로 지원될 수도 있으며 그렇지 않다면 개발되어야 한다. 후자의 경우는 본 연구에서 지속적으로 재사용되면서 재구조화 되고 있는 표기법 객체 모델로서 (그림 10)과 같은 클래스 내용을 갖는다.

(그림 10)의 개략적인 설계 구조는 OODesigner의 세 버전을 구현하는 과정에서 사용되었던 (그림 3)과 같은 모델을 정제하여 얻은 클래스 다이어그램이다. CASE 도구가 지원해야 하는 표기법의 구성은 근본적으로 그래프(graph)[2]와 일치한다. 따라서 표기법을 지원하기 위한 클래스 다이어그램은 그래프의 구성 요소를 반영하는데 (그림 10)의 NodeFigure 클래스는 노드 객체들을 대표하는 추상 클래스이고 Connection 클래스는 연결선을 대표하는 추상 클래스이다. NodeFigure는 UML 상태 다이어그램의 start 상태 심볼과 end 상태 심볼과 같은 문자열을 동반하지 않는 클래스의 기능을 제공하며 EditableFigure 클래스는 문자열에 대한 편집 기능을 제공하는 노드 표기법을 위한 기능을 제공한다. 아울러 Connection 클래스는 표기법 중에서 연결선 모양을 갖는 aggregation, generalization, transition 등의 클래스가 필요로 하는 공통 기능을 제공한다. 이 그림에서 TwoPointFigure로부터 상속되는 클래스들은 모든 그래픽 편집기에서 공통적으로 필요한 표기법인 사각형, 삼각형, 선분들과 같은 객체들을 만들기 위한 것이며 이들은 NodeFigure 객체와 Connec-

2) 그래프 G는 (V,E)로 표현된다. 여기에서 V(G)는 노드의 집합을 나타내며 E(G)는 연결선의 집합을 나타낸다.

tion 객체의 부속 객체로 사용된다. (그림 10)의 클래스 다이어그램은 UML외의 표기법에 대한 모델링 시에도 재사용의 가능성이 있다.



(그림 10) 표기법 객체 모델을 위한 클래스 다이어그램

비록 서로 다른 객체지향 언어들은 다른 문법 구조들을 가지고 있지만 그들이 지원하는 주된 객체지향 개념은 같다. 그러므로 구현 시 객체지향 프로그래밍의 기본 특징들의 일관성을 유지하는 것이 매우 중요하다. 본 연구의 이식 연구가 성공한 것도 Unix 버전이 재구조화 과정을 거쳐 매우 잘 구조화 되었기 때문이다. 재구조화 과정에서 개발팀이 고려했던 점과 그 성과는 OODesigner의 재구조화에 관련된 논문[19]에 제시되어 있다.

5. 결 론

본 논문에서는 객체지향 CASE 도구인 OODesigner의 세가지 버전들을 개발하고 이식하는 과정과 그로부터 얻은 경험들을 기술하였다. Unix 버전의 개발과 재구조화 경험은 Java 버전과 Windows 버전으로의 이식 공정을 개선시켰다. 두 가지 버전으로의 이식은 초기 버전의 개발에서 얻은 객체지향 패러다임의 이해와 함께 초기 버전의 OODesigner를 사용하여 행해졌다는 점에서 의미가 있다. 또한 이들 개발 결과 소프트웨어 개발 방법론을 지원하는 CASE 도구 개발에는 일반적인 공통점과 패턴을 사용할 여지가 있음을 발견하였다. 본 연구의 결론을 간단히 기술하면 다음과 같다.

- 다중 플랫폼을 지원하는 CASE 도구는 MVC 관점에서 볼 때 설계와 구현에 매우 높은 유사성이 존재한다.
- CASE 도구를 설계할 때 시스템을 MVC 컴포넌트들로 나누는 것보다 MC 컴포넌트들로 나누는 것이 바람직하다.
- 객체지향 소프트웨어 개발 시 객체지향 프로그래밍의 기본 특징들의 일관성을 유지하는 것이 매우 중요하다.

본 연구 결과 클래스 크기를 가능하면 작게 하는 것이 소프트웨어 이식성을 높이는 데 매우 중요한 역할을 한 것으로 판단된다. 또한 객체지향 언어의 특징이나 특정 개발 플랫폼의 클래스 라이브러리의 기능 등을 잘못 이해하여 오용하는 경우에는 시스템에 매우 치명적인 위험을 준다는 사실을 확인하였다. 그러므로 새로운 환경에서 새로운 프로젝트를 시작할 때는 매우 길고 잘 계획된 사전 교육과정이 반드시 필요하다.

현재 진행되고 있는 연구는 초기 Unix 버전과 이를 재구조화한 버전을 잘 알려진 객체지향 메트릭들을 이용하여 정량적으로 비교 평가를 해보는 것이다. 향후 연구 과제는 OODesigner를 소프트웨어 컴포넌트나 설계 패턴들을 지원하는 도구로 확장하는 것과 소프트웨어 개발 프로세스 모델들을 지원하도록 하는 것이다.

참 고 문 헌

[1] R. G. Fishman, C. F. Kemerer. "Object-Oriented and Conventional Analysis and Design Methodologies," *IEEE Computer*, Vol.25, No.10, pp.22-40, Oct. 1992.

[2] P. H. Loy, "A Comparison of Object-Oriented and Structured Development Method," *ACM SIGSOFT SE Notes*, Vol.15, No.1, pp.44-48, Jan. 1990.

[3] G. Booch, '*Object-Oriented Design with Applications*', The Benjamin/Cummings Publishing Company, 1991.

[4] A. I. Wasserman, P. A. Pircher, R. J. Muller, "An Object-Oriented Structured Design Method for Code Generation," *ACM SIGSOFT SE Notes*, Vol.14, No.1, pp.32-55, Jan. 1989.

[5] P. Coad, E. Yourdon, '*Object-Oriented Analysis*', Yourdon Press, 1990.

[6] K. Beck, W. Cunningham, "A Laboratory for Teaching Object-Oriented Thinking," *Proc. OOPSLA'89*, pp.1-6, Oct. 1989.

[7] R. J. Wirfs-Brock, R. E. Johnson, "Surveying Current Research in Object-Oriented Design," *Commun. ACM*, Vol.33, No.9, pp.104-124, Sept. 1990.

[8] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenzen, '*Object-Oriented Modeling and Design*', Prentice Hall, 1991.

[9] M. Fowler, K. Scott, '*UML Distilled : Applying the Standard Object-Oriented Modeling Language*', Addison-Wesley, 1997.

[10] T. J. Mowbray, R. Zahavi, '*The Essential CORBA*', John Willey & Sons, 1995.

[11] D. Chappell, '*Understanding ActiveX and OLE*', Microsoft Press, 1997.

[12] K. Wallnau, E. Morris. P. Feiler, A. Earl, and E. Litvak, "Engineering Computer-Based Systems with Distributed Object Technology," *Proc. International Conf. on Worldwide Computing and Its Applications*, 1997, *Lecture Notes on Computer Science #1274*, Springer-Verlag.

[13] A. Finkelstein, J. Kramer, and B. Nuseibeh, '*Software Process Modelling and Technology*', John Wiley & Sons, 1994.

[14] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Soft. Eng.*, Vol.20, No.6, June 1994.

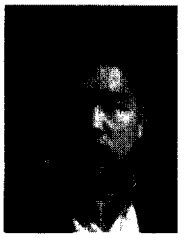
[15] T. Kim, N. Boudjlida, "An Experience Report Related to Restructuring OODesigner : A CASE Tool for OMT," *Proc. Asia-Pacific Software Engineering Conf.*, pp.220-227, Dec. 1998.

[16] T. Kim, G. Shin, "Restructuring OODesigner : A CASE Tool for OMT," *Proc. ICSE'98*, pp. 449-451, April 1998.

[17] Y. P. Shan, "An Event-Driven Model-View-Controller Framework for Smalltalk," *Proc. OOPSLA '89*, pp.347-352, Oct. 1989.

[18] D. A. Marca, C. L. McGowan, '*SADT, Structured Analysis and Design Techniques*', McGraw Hill, 1987.

[19] 조장우, 김태균, "객체 지향 CASE 도구에 대한 재구조화 실험", 한국정보처리학회 논문지, 제6권, 제4호, pp.932-940, 1999.



홍 의 석

e-mail : hes@aycc.anyang.ac.kr

1992년 서울대학교 계산통계학과 졸업(학사)

1994년 서울대학교 대학원 계산통계학과(이학석사)

1999년 서울대학교 대학원 전산과학과(이학박사)

1999년~현재 안양대학교 영상처리학과 전임강사

관심분야 : 메트릭 기반 소프트웨어 품질 예측 모델, 웹 기반 멀티미디어 응용 기술 등

김 태 균

e-mail : ktg@taejo.pufs.ac.kr

1985년 서울대학교 지질학과 졸업(학사)

1987년 서울대학교 대학원 계산통계학과(이학석사)

1995년 서울대학교 대학원 계산통계학과 전산과학전공(이학박사)

1988년~현재 부산외국어대학교 컴퓨터공학과 교수

관심분야 : 컴포넌트 기반 소프트웨어 공학, 객체지향 CASE 도구