A UNIVERSAL VECTOR GRAPHICS EDITING SYSTEM BASED ON DESIGN PATTERN AND JAVA

Limin Ao^{1,2}, Pingping Xia², Yanfeng Jia², Yongchun Wang²

¹ *College of Information and Electrical Engineering, China Agricultural University, Beijing 100083, China*

² *Department of Information Engineering, Northeast China Institute of Electric Power Engineering, Jilin Jilin 132012, China*

Abstract: Design patterns capture successful solutions to recurring problems that arise when building software systems. Firstly, this paper presents how to design a vector graphics editing platform using design pattern. At last, on the base of summarizing all kinds of vector graphics editing systems at home and abroad, this paper presents a new idea on how to develop a universal vector graphics editing system.

Key words: Design Patterns, Vector Graphics, Mediator Pattern, Strategy Pattern

1. INTRODUCTION

The main function of vector graphics editing system is to support all kinds of special graphics' input and output. On the base of summarizing all kinds of vector graphics editing systems at home and abroad, we absorb the merits of others during the development of the vector graphics editing system [1]. Meanwhile, considering all kinds of different and flexible requirements for graphics and symbols in many aspects, such as metallurgy, architecture, power system, etc, a new idea is introduced on how to develop a universal vector graphics editing system.

We developed the system using design patterns. Software design patterns became popular with the wide acceptance of the book (《Design Pattern-elements of reusable software》) by Gamma et al. (1995) [2]. They are

successful design paradigms that have been summarized during practice. The rational use of design patterns can help software developers to comprehend design routes easily, perfect their solution, and accomplish the development of software better.

2. THE DESIGN AND IMPLEMENTATION OF MAIN PLATFORM

We developed the platform based on Java in the environment of IBM Eclipse3.0.1. The main components of vector graphics editing system are presented below Fig.1 [3]. *WinApp* defines the drawing interface and acts as a mediator among components. *ViewPanel* is an area that can show figures and accept user input. The *FigureSet* consist of *Figures*, which in turn can be containers for other *Figures*. Changes in the *FigureSet* are propagated to the *ViewPanel* that is responsible for updating any *Figures*. Each *Figure* has *Handles*, which define access points and determine how to interact with the *Figure*. In *ViewPanel*, you can select several figures and handle them. There are many *Tools* in *Toolbar*, which are used to create or handle *Figures*.

3. THE VECTOR GRAPHICS EDITING SYSTEM AND ITS DESIGN PATTERNS USED

3.1 The Application of Mediator Pattern

Object-Oriented Programming advocates distributing behavior over objects. However, as more of these isolated objects are developed in a program, the problem of communication between these classes becomes more complex. The more each class needs to know about the methods of another class, the more tangled the class structure can become [4]. This makes the program harder to read and harder to maintain. Furthermore, it can become difficult to change the program, since any changes may affect code in several other classes.

In vector graphics editing system objects, including *Command*, *Tool*, *ViewPanel* and their own subclasses, etc, interact with each other. If their interactions are dealt with general measures, it may lead to a pretty intricate network. At the same time, the maintenance and ability to change the system are difficult to expand. As a result, the reusability of the system is weakened.

The Mediator pattern resolves the problem by promoting loose coupling between these classes. Mediator accomplishes this by being the only class that

has detailed knowledge of the methods of other classes. Classes send information to the Mediator when changes occur and the Mediator passes them on to any other classes. Therefore, Mediator pattern is used in the system (UML class diagram refer to Fig. 2). The *DrawingEditor*, as a mediator, takes charge of the interactions among Colleagues. These Colleagues consist of *Command*, *Tool*, *ViewPanel* and so on. The advantage of the Mediator is obvious—it is the only class that knows of the other classes, and thus the only one that would need to be changed if one of the other classes changes or if other interface-control classes are added [4].

Figure 1. Main Framework

3.2 The Application of Strategy Pattern

The operation of a figure is actually to operate its many special points of the figure, which is updated itself according to these points. Then, how do we get these special points? The problem involves particular algorithms. On account of different special points of a figure, there are many differences among these calculating methods. For example, the algorithms are totally different when getting the middle point and end point of a line. Moreover, for different figures, even if getting the middle point or end point, the algorithm is also different. Therefore, we should define different algorithms on every special point to meet different responsibilities.

The Strategy pattern consists of a number of related algorithms encapsulated in a driver class called the *Context*. The client's program can

select one of these differing algorithms or in some cases the *Context* might select the best one for you. The intent is to switch easily between algorithms without any monolithic conditional statements [4]. Strategy lets the algorithm vary independently from the clients that use it.

Figure 2. Mediator design pattern

In the system, the problem about how to choose algorithms of special point could be resolved just by using the Strategy pattern. Firstly, we define an interface *Locator* which has a method *locate()* implemented by *Locator's* subclass [5]. For instance, we get the middle point of line by subclass *CenterLocator*. The concrete algorithm is encapsulated in method *locate()*.

The definition of interface *Locator* as follows:

```

public interface Locator{

    public Point locate(Figure owner){    }

}
  
```

The definition of subclass *CenterLoator* which get middle point is as follows:

```

public class CenterLocator implements Locator{

    public Point locate(Figure owner){

        Rectangle r=owner.displayBox();

        return

        new Point(r.x+r.width/2,r.y+r.height/2);

    }

}
  
```

```

    }

}

```

The contents mentioned above introduce several main patterns for the design of vector graphics editing system. These patterns can improve largely the reusability and expansibility of the system. Meanwhile, other design patterns, such as State pattern, Factory pattern, Composite pattern, etc, also have been used in the system. Their introduction is omitted because of the restrictions on the paper's length [7].

4. THE PRINCIPLE OF UNIVERSAL VECTOR GRAPHICS EDITING SYSTEM

Any graphics are composed of dots, lines and circles, so we define these graphics' Class as basic Class whose constructed functions have no parameters. Graphics' Class is the key to implement a universal vector graphics editing system [6]. We define a class *DynamicFigure* whose constructed function has a parameter (*int ID*). Using the class *DynamicFigure*, we can draw all kinds of different figures according as the difference of its ID value. Looking from the point of Object-Oriented Programming, we consider two factors: One is object's state; the other is object's behavior. As far as the object's state is concerned, the Class *DynamicFigure* has graphics' all kinds of properties, such as graphics' ID value, coordinates, size, color, stroke and so on. As far as the object's behavior is concerned, Class *DynamicFigure* have graphics' all kinds of methods, such as *public void move (int x, int y)* (whose function is to move a figure to a new place) and *public void drawFrame(Graphics g)* (whose function is to draw a figure using Java2D). After we draw a compositive graphics using basic graphics, clicking the Mouse right-button to show a Pop-Menu (See Fig.3.) and selecting the "composite" Menu-Item, then the system computes the smallest rectangle which encircles the selected basic graphics. Finally, the system connects database *SQLServer2000* by *JDBC* driver and stores the selected basic graphics' properties, such as the basic graphics' ID value, the distance from basic graphics to the smallest rectangle, color(R,G,B), stroke, compositive graphics' ID value and the smallest rectangle's left X and top Y coordinate. In this way, when we select a *DynamicFigure* in the left Toolbar(See Fig.3.), even if we handle the same Class *DynamicFigure*, the system reads properties fields values from database according as the difference of graphics' ID value, we can draw different compositive graphics using the method *public void drawFrame(Graphics g)*.

Meanwhile, after we create a vector graphics, the system must build automatically a little icon whose shape is same to the compositive graphics in the left toolBar. We analyze the interface `java.swing.Icon`'s code, implement the interface *Icon* and override the method *paintIcon(Component comp, Graphics g, int x, int y)* to display the icon in the left toolBar [7]. In this way, we can create automatically any vector graphics what we want.

If we double click the little icon in the left toolBar, the compositive graphics will be deleted from `SQLServer2000` and the icon will also be deleted from the toolBar.

The result can be seen in Fig. 3.

Figure 3. The Main interface of the editing system

5. CONCLUSIONS

Improving software adaptability is a formidable task in reality. One rather intriguing concept that strongly promotes adaptability is design patterns. This paper presented a new idea that aims to help software developers using design patterns. In particular, this paper has presented how to analyze and use design

patterns as potential adaptability enhancers in developing software systems [8].

Meanwhile, in virtue of Java' merit, the system's maintenance and reusability are improved largely.

Based on the universal vector graphics editing system, we can shorten the designing time and reduce costs largely during the development of vector graphics system. The software developer can quickly design new drawing diagram software that accords with special industry need. Moreover, it also can be used in mechanization, manufacture, chemical plant, and so on. The application fields are very extensive.

REFERENCES

- [1] Gardner, John A.; Bulatov, Vladimir: Directly Accessible Mainstream Graphical Information, In: Klaus, Joachim; Miesenberger, Klaus, Burger, Dominique; Zagler, Wolfgang (Eds.): Proceedings of the 9th International Conference on Computers Helping People with Special Needs (ICCHP), Springer, Berlin, 2004
- [2] Eric Gamma, Richard Helm. Design Patterns-elements of reusable object-oriented software [M].Addison Wesley Publishing Company, 1995
- [3] <http://www.di.uniovi.es/~cueva/investigacion/lineas/patrones/JHotDraw.pdf>
- [4] James W. Cooper. Java Design Patterns: A Tutorial [M].USA: Pearson Education, 2003.
- [5] J.Mylopoulos, L.Chung, S.S.Y.Liao et al., Extending object-oriented analysis to explore alternatives, IEEE Software (2001 January/February) 2-6
- [6] Rotard, Martin; Otte, Kerstin; Ertl, Thomas: Exploring Scalable Vector Graphics for Visually Impaired Users, In: Klaus, Joachim; Miesenberger, Klaus; Burger, Dominique; Zagler, Wolfgang (Eds.): Proceedings of the 9th International Conference on Computers Helping People with Special Needs (ICCHP), Springer, Berlin, 2004
- [7] D.C.Schmidt, M.Stal, Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Wiley, New York, NY, 2000.

Limin Ao was born in 1956. She is a Ph.D. student at College of Information and Electrical Engineering, China Agricultural University, where she has studied for the last four years. She is currently Professor in the Department of Information Engineering at the Northeast China Institute of Electric Power Engineering. Hers research interests include Software Engineering, with application of computer in electric power system.

Pingping Xia was born in 1975. He is a postgraduate student at department of Information Engineering of Northeast China Institute of Electric Power Engineering. His research areas include design pattern, software engineering.