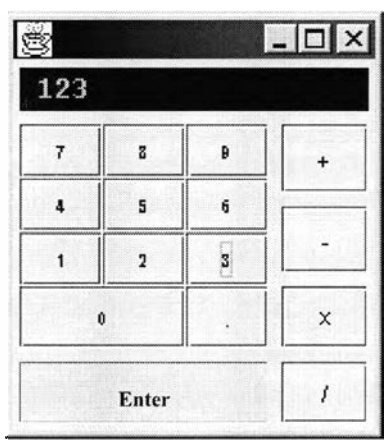
บทที่ 6

ทดสอบการใช้ FOOD ในการสร้างโปรแกรมประยุกต์

หลังจากพัฒนาโครงร่างสำหรับสร้างโปรแกรมเชิงวัตถุด้วยรูปแบบการออกแบบเสร็จสมบูรณ์แล้ว ผู้วิจัยได้ทำการทดสอบโดยการสร้างโปรแกรมเชิงวัตถุ จำนวน 3 โปรแกรม คือ โปรแกรมเครื่องคิดเลข โปรแกรมวาดภาพ และโปรแกรมสัญญาณไฟจราจร ได้ตามที่ออกแบบไว้ดังนี้

6.1 การสร้างโปรแกรมเครื่องคิดเลข

โปรแกรมเครื่องคิดเลขที่สร้างขึ้นนี้ สามารถใช้ในการคำนวณ บวก ลบ คูณ หาร ประกอบด้วยปุ่มตัวเลข 0 ถึง 9 และทศนิยม ปุ่มเครื่องหมายทางคณิตศาสตร์ "+" "-" "x" "/" ปุ่ม "Enter" เพื่อสั่งให้ทำการคำนวณ และลาเบลเพื่อใช้แสดงผลการคำนวณ โปรแกรมเครื่องคิดเลขที่สร้างเสร็จเรียบร้อยแล้ว ดังแสดงในรูปที่ 6.1



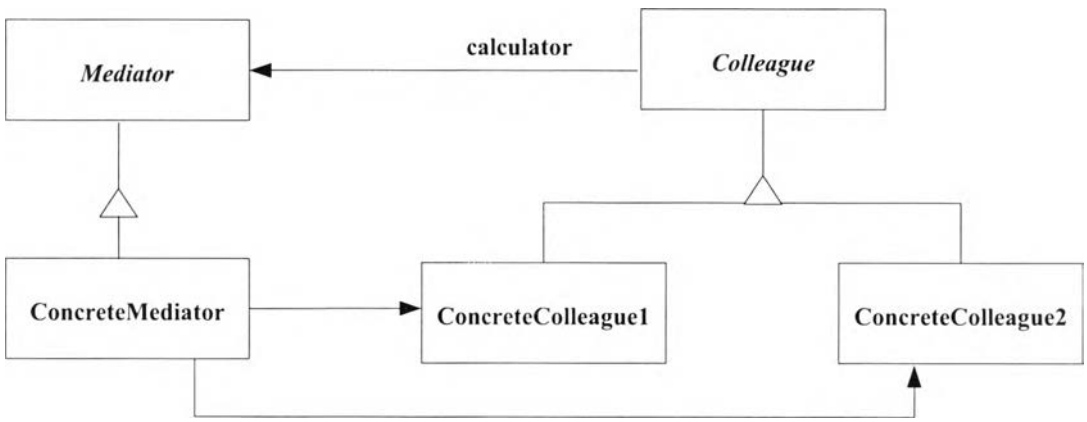
รูปที่ 6.1 ผลการสร้างโปรแกรมเครื่องคิดเลข

6.1.1 รายละเอียดปัญหาในการเขียนโปรแกรมเครื่องคิดเลข

ในการออกแบบโปรแกรมเครื่องคิดเลขต้องมีการสร้างปุ่มต่างๆเป็นจำนวนมาก ซึ่งในการพัฒนาโปรแกรมเชิงวัตถุ ออปเจคที่สร้างขึ้นแต่ละตัวจะมีการกำหนดพฤติกรรมแตกต่างกันไปตามแต่ละออปเจค ดังนั้นการที่มีออปเจคจำนวนมากทำให้การโต้ตอบกันระหว่างออปเจคต่างๆมีมากขึ้นไปด้วย ทุกออปเจคที่อยู่ในระบบต้องรู้ถึงพฤติกรรมของออปเจคอื่นๆด้วย และต้องรอรับเหตุการณ์ที่จะเกิดขึ้น ทำให้มีความซับซ้อนในการจัดการกับออปเจคเหล่านั้น

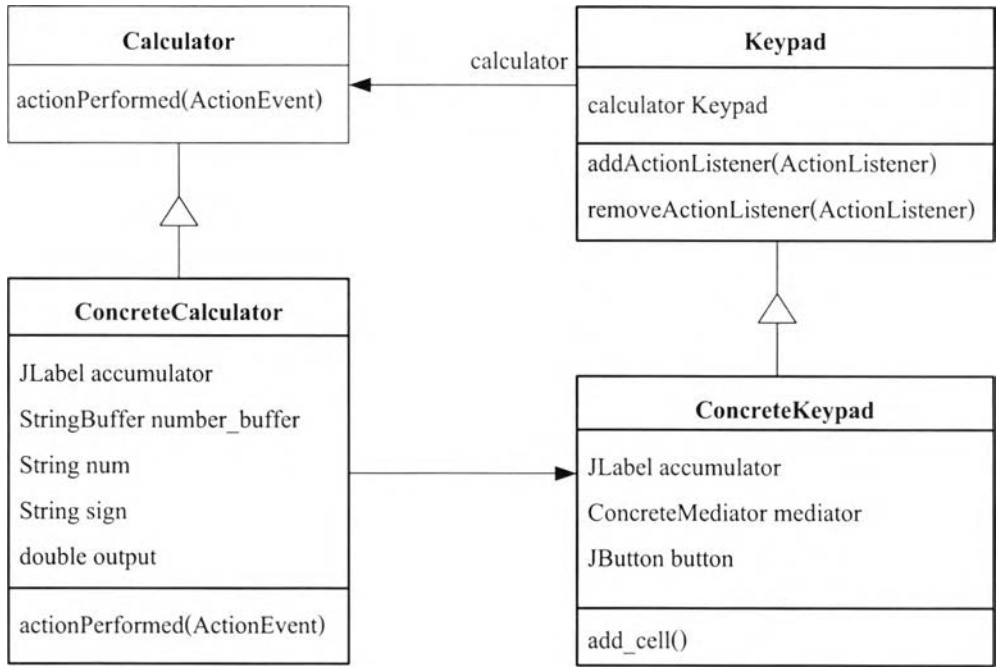
6.1.2 แนวทางการแก้ปัญหา

จากปัญหาดังกล่าว Holub, A [7] ได้เลือกใช้รูปแบบการออกแบบชื่อ Mediator ซึ่งจัดเป็นรูปแบบพฤติกรรม(Behavioral pattern)แบบหนึ่ง โดยได้มีการออกแบบคลาสที่มีศูนย์กลางในการควบคุมและจัดการประสานงานในการโต้ตอบระหว่างออปเจค จากรูปที่ 6.2 คลาสคอลลีก (Colleague) จะเป็นคลาสแม่ของออปเจคที่ต้องการสร้าง โดยมีความสัมพันธ์อ้างอิงไปยังคลาสมีดิเอเตอร์ ที่จะทำหน้าที่ประสานงานในการโต้ตอบกันระหว่างออปเจค ทุกครั้งที่มีเหตุการณ์เกิดขึ้นกับ ออปเจคจะแจ้งให้คลาสมีดิเอเตอร์ทราบ เพื่อจะได้ดำเนินการกับออปเจคที่เกี่ยวข้องต่อไป



รูปที่ 6.2 รูปแบบการออกแบบ Mediator [3]

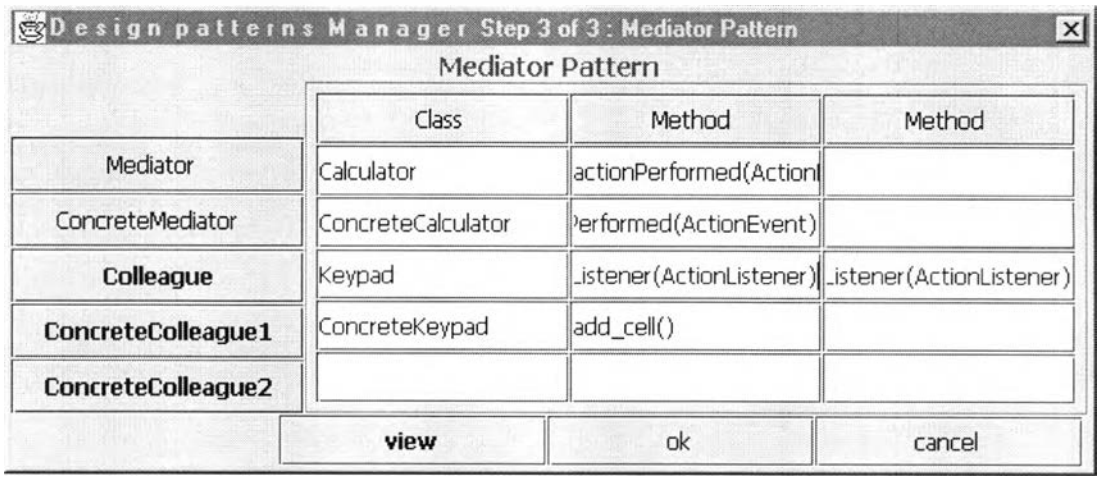
ในการสร้างโปรแกรมเครื่องคิดเลขได้นำรูปแบบ Mediator จากรูปที่ 6.2 มาเป็นแนวทางในการออกแบบ โดยสร้างคลาสคีย์แพด(Keypad)เป็นคลาสแม่แล้วสับคลาสไปเป็นคลาสคอนครีทคีย์แพด(ConcreteKeypad) เพื่อที่จะสร้างออปเจคของปุ่มต่างๆบนเครื่องคิดเลขโดยออปเจคของปุ่มต่างๆ จะใช้ลิสเทนเนอร์(Listener)ชุดเดียวกัน จากนั้นสร้างคลาสแคลคูเลเตอร์ (Calculator) เป็นคลาสแม่ แล้วสับคลาสเป็นคลาสคอนครีทแคลคูเลเตอร์ (ConcreteCalculator) เมทธอดแอคชั่นเพอร์ฟอร์ม(actionPerformed) จะรับเหตุการณ์จากออปเจคเพื่อตรวจสอบและทำการคำนวณ การออกแบบคลาสโปรแกรมเครื่องคิดเลข ดังแสดงในรูปที่ 6.3



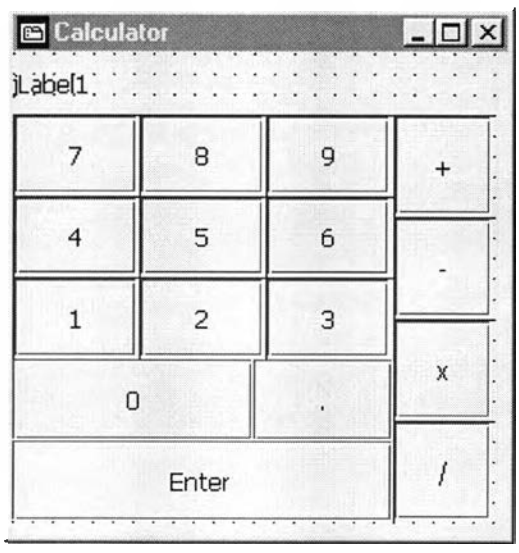
รูปที่ 6.3 การออกแบบคลาสโปรแกรมเครื่องคิดเลข [7]

6.1.3 ขั้นตอนการสร้าง

1) สร้างโครงชุดคำสั่งรูปแบบการออกแบบ โดยเข้าสู่ระบบเลือกเมนู "Design Pattern\Pattern Manager" จะเป็นการเรียกใช้งานส่วนจัดการรูปแบบการออกแบบ เพื่อสร้างโครงชุดคำสั่งโดยเลือกรูปแบบการออกแบบชื่อ Mediator แล้วทำการกำหนดค่าต่างๆตามที่ได้ออกแบบไว้ ดังแสดงในรูปที่ 6.4 การกำหนดค่าให้แก่รูปแบบการออกแบบ
2) สร้างส่วนติดต่อกับผู้ใช้ โดยเลือกเริ่มงานใหม่จากแถบเมนู (File/New) จะมีวินโดว์แสดงเทมเพลส (Templates) ให้ผู้ใช้เลือกออปเจคเฟลม แล้วทำการสร้างส่วนติดต่อกับผู้ใช้โดยนำองค์ประกอบได้แก่ ปุ่มคำสั่ง แถบข้อความมาจัดเรียงกัน ดังแสดงในรูปที่ 6.5 การนำองค์ประกอบซอฟต์แวร์มาสร้างส่วนติดต่อกับผู้ใช้ พร้อมทั้งกำหนดคุณสมบัติขององค์ประกอบต่างๆ โดยระบบจะสร้างชุดคำสั่งให้อัตโนมัติ
3) เพิ่มชุดคำสั่งให้สมบูรณ์ในส่วนชุดคำสั่งของคลาสคอนครีทคีย์แพด (ConcreteKeypad) ได้มีการใช้เมทอดแอดเซล (add_cell) เพื่อสร้างและกำหนดพฤติกรรมให้กับปุ่มต่างๆ ดังแสดงในรูปที่ 6.6 และได้ใช้ลิสเทนเนอร์ ชุดเดียวกันดังแสดงในรูปที่ 6.7 เมื่อทำการเขียนชุดคำสั่งเสร็จแล้วจึงทำการคอมไพล์และรันโปรแกรมที่สมบูรณ์



รูปที่ 6.4 การกำหนดค่าให้แก่รูปแบบการออกแบบ



รูปที่ 6.5 การนำองค์ประกอบซอฟต์แวร์มาสร้างส่วนติดต่อกับผู้ใช้

```
ConcreteKeypad()
    {
        accumulator.setForeground   (Color.green);
        accumulator.setFont         (new Font("Monospaced",Font.BOLD, 14));
        accumulator.setPreferredSize(new Dimension(100, 20) );

        JPanel keypanel = new JPanel();
        keypanel.setLayout( new GridBagLayout() );
        //        cont.  r c w h x y p  label/obj, action
        add_cell( keypanel, 2, 0, 1, 1, 1, 0, "1"       , "1"                );
        add_cell( keypanel, 2, 1, 1, 1, 1, 0, "2"       , "2"       );
        add_cell( keypanel, 2, 2, 1, 1, 1, 0, "3"       , "3"       );
        add_cell( keypanel, 1, 0, 1, 1, 1, 0, "4"       , "4"       );
        add_cell( keypanel, 1, 1, 1, 1, 1, 0, "5"       , "5"       );
        add_cell( keypanel, 1, 2, 1, 1, 1, 0, "6"       , "6"       );
        add_cell( keypanel, 0, 0, 1, 1, 1, 0, "7"       , "7"       );
        add_cell( keypanel, 0, 1, 1, 1, 1, 0, "8"       , "8"       );
        add_cell( keypanel, 0, 2, 1, 1, 1, 0, "9"       , "9"       );
        add_cell( keypanel, 3, 0, 2, 1, 2, 1, 0, "0"       , "0"       );
        add_cell( keypanel, 3, 2, 1, 1, 1, 1, 0, "."       , "."       );

        this.setLayout( new GridBagLayout() );

        add_cell( this,   0, 0, 2, 1, 4, 1, 3, accumulator, null );
        add_cell( this,   1, 1, 1, 1, 1, 1, 3, "+"       , "+"  );
        add_cell( this,   2, 1, 1, 1, 1, 1, 3, "-"       , "-"  );
        add_cell( this,   3, 1, 1, 1, 1, 1, 3, "X"       , "*"  );
        add_cell( this,   4, 1, 1, 1, 1, 1, 3, "/"       , "/"  );
        add_cell( this,   4, 0, 1, 1, 4, 1, 3, "Enter"   , "\n" );
        add_cell( this,   1, 0, 1, 3, 3, 4, 3, keypanel  , null );

        setMinimumSize( getPreferredSize() );
        setMaximumSize( getPreferredSize() );
    }
    //---------------------------------------------------------------
    private void add_cell(  Container container,
                            int row,    int col,
                            int width,  int height,
                            int weightx,int weighty,
                            int pad,
                            Object item,
                            String action_command )
    {
        GridBagConstraints constraints = new GridBagConstraints();

        constraints.fill        = GridBagConstraints.BOTH;
        constraints.gridy       = row;
        constraints.gridx       = col;
        constraints.gridwidth   = width;
        constraints.gridheight  = height;
        constraints.weightx     = weightx;
        constraints.weighty     = weighty;
        constraints.insets      = new Insets(pad,pad,pad,pad);

        JComponent component;
        if( item instanceof JComponent )
            component = (JComponent)item;
        else                        // assume it's a string
        {

            String label = (String) item;
            JButton button = new JButton( label          );
            button.addActionListener    ( calculator     );
            button.setActionCommand     ( action_command );
            button.setFont
            (   new Font("SansSerif",Font.BOLD,
                Character.isLetterOrDigit(label.charAt(0))? 12: 16 )
            );
            component = button;
        }

        container.add( component );
        component.setVisible( true );

        GridBagLayout layout = (GridBagLayout)container.getLayout();
        layout.setConstraints( component, constraints );
    }
```

รูปที่ 6.6 ชุดคำสั่งคอนครีทคีย์แพท เรียกใช้เมทธอด “add_cell”
เพื่อสร้างและกำหนดพฤติกรรมให้กับปุ่มต่างๆ

```
private void add_cell(  Container container,
                        int row,   int col,
                        int width,  int height,
                        int weightx,int weighty,
                        int pad,
                        Object item,
                        String action_command )
    {
        GridBagConstraints constraints = new GridBagConstraints();

        constraints.fill       = GridBagConstraints.BOTH;
        constraints.gridy      = row;
        constraints.gridx      = col;
        constraints.gridwidth  = width;
        constraints.gridheight = height;
        constraints.weightx    = weightx;
        constraints.weighty    = weighty;
        constraints.insets     = new Insets(pad,pad,pad,pad);

        JComponent component;
        if( item instanceof JComponent )
            component = (JComponent)item;
        else                        // assume it's a string
        {

            String label = (String) item;
            JButton button = new JButton( label         );
            button.addActionListener   ( mediator     );
            button.setActionCommand    ( action_command );
            button.setFont
            (   new Font("SansSerif",Font.BOLD,
                Character.isLetterOrDigit(label.charAt(0))? 12: 16 )
            );
            component = button;
        }

        container.add( component );
        component.setVisible( true );

        GridBagLayout layout = (GridBagLayout)container.getLayout();
        layout.setConstraints( component, constraints );
    }
```

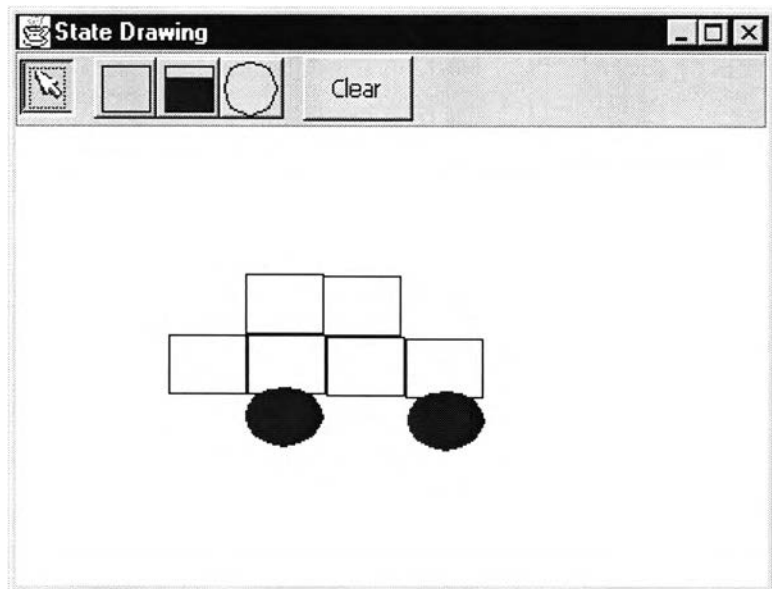
รูปที่ 6.7 เมทธอด “add_cell” สร้างปุ่มโดยใช้ลิสเทนเนอร์ชุดเดียวกัน

6.2 การสร้างโปรแกรมวาดภาพ

โปรแกรมวาดภาพที่สร้างขึ้นนี้ เป็นโปรแกรมที่ใช้วาดภาพอย่างง่ายโดยจะวาดภาพบนพื้นฐานของรูปสี่เหลี่ยม และรูปวงกลม โปรแกรมวาดภาพที่สร้างเสร็จเรียบร้อยแล้ว ดังแสดงในรูปที่ 6.8 คุณสมบัติของโปรแกรมวาดภาพมีดังนี้

1. เลือกปุ่มลูกศรในแถบเครื่องมือ  เป็นการเปลี่ยนสถานะของโปรแกรมให้อยู่ในสถานะเลือกรูปภาพ เมื่อคลิกเม้าส์ที่รูปในส่วนพื้นที่วาดภาพจะเป็นการเลือกรูปนั้น
2. เลือกปุ่มรูปสี่เหลี่ยม  เป็นการเปลี่ยนสถานะของโปรแกรมให้อยู่ในสถานะวาดรูปสี่เหลี่ยม เมื่อคลิกเม้าส์ในส่วนพื้นที่วาดภาพจะเป็นการสร้างรูปสี่เหลี่ยมขึ้นมาใหม่
3. เลือกปุ่มสีน้ำเงิน  เป็นการเปลี่ยนสถานะของโปรแกรมให้อยู่ในสถานะระบายสีน้ำเงินลงบนออปเจครูปภาพที่เลือก

4. เลือกปุ่มรูปวงกลม  เป็นการเปลี่ยนสถานะของโปรแกรมให้อยู่ในสถานะวาดรูปวงกลม เมื่อคลิกเม้าส์ในส่วนพื้นที่วาดภาพจะเป็นการสร้างรูปวงกลมขึ้นมาใหม่
5. เลือกปุ่ม"Clear"  เป็นการลบออปเจครูปภาพทั้งหมดออกจากพื้นที่วาดภาพ



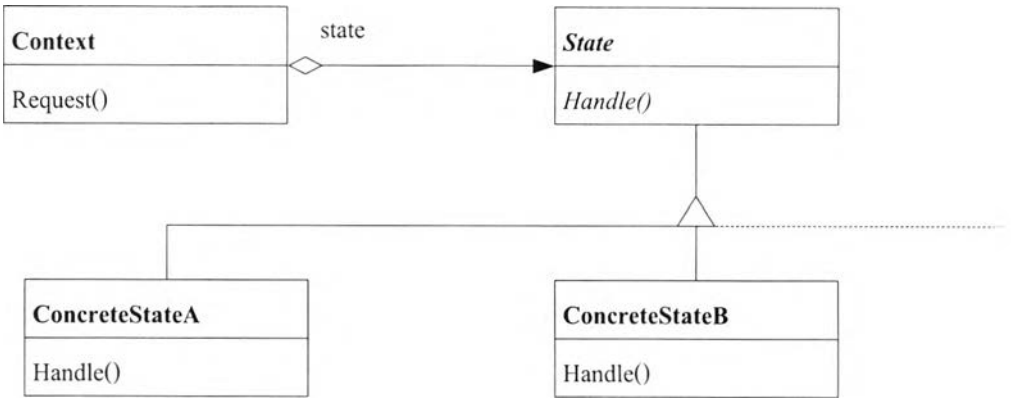
รูปที่ 6.8 ผลการสร้างโปรแกรมวาดภาพ

6.2.1 รายละเอียดปัญหาในการเขียนโปรแกรมวาดภาพ

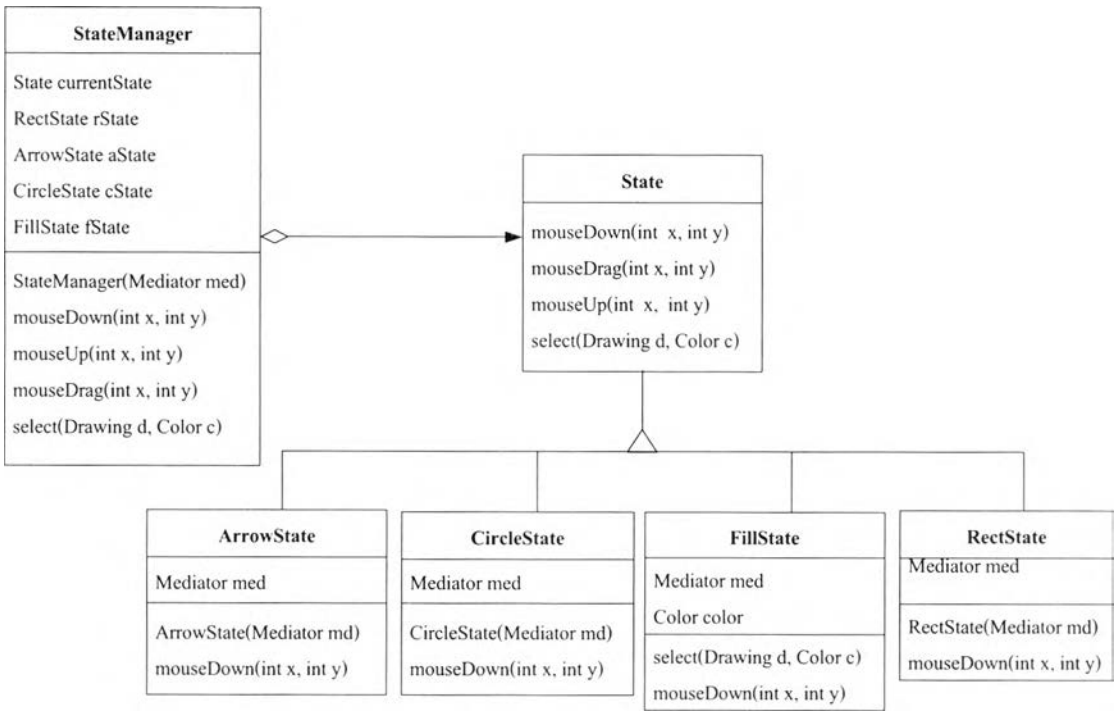
โปรแกรมวาดภาพจะมีปุ่มต่างๆบนแถบเครื่องมือที่จะเปลี่ยนสถานะการทำงานของโปรแกรม โดยทั่วไปการเขียนโปรแกรมลักษณะนี้จะใช้ประโยค "if –else" หรือ "switch" ในการควบคุมการทำงาน ที่จะต้องตรวจสอบสถานะปัจจุบันของออปเจคทุกตัว หรืออาจสร้างตัวแปรเพื่อเก็บสถานะไว้ ซึ่งทำให้โปรแกรมอ่านยากและซับซ้อน การแก้ไขผิดพลาดได้ง่าย

6.2.2 แนวทางการแก้ปัญหา

จากปัญหาดังกล่าว Cooper, W.J. [8] ได้นำรูปแบบการออกแบบชื่อ State ซึ่งจัดเป็นรูปแบบพฤติกรรม(Behavioral pattern)แบบหนึ่ง โดยได้มีการออกแบบคลาสสเตท (State) เป็นคลาสแม่ประกอบไปด้วยเมทธอดพฤติกรรมพื้นฐานทั้งหมดของโปรแกรม เพื่อให้ออปเจคทำการสับคลาสไปใช้ และมีคลาสคอนเท็กซ์(Context) เป็นตัวควบคุมสถานะการทำงาน ดังแสดงในรูปที่ 6.9



รูปที่ 6.9 รูปแบบการออกแบบ State [3]



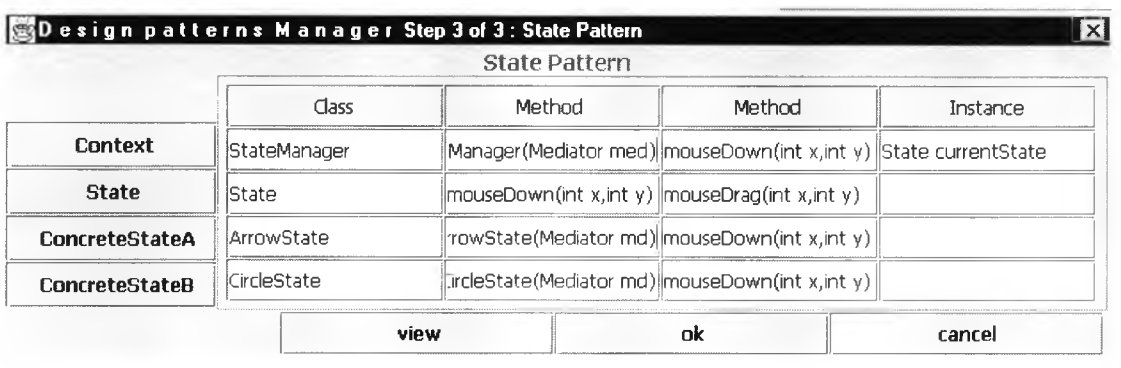
รูปที่ 6.10 การใช้รูปแบบการออกแบบ State ในการออกแบบโปรแกรมวาดภาพ [8]

6.2.3 ขั้นตอนการสร้าง

1) สร้างโครงชุดคำสั่งรูปแบบการออกแบบ โดยเข้าสู่ระบบเลือกเมนู "Design Pattern\Pattern Manager" จะเป็นการเรียกใช้งานส่วนจัดการรูปแบบการออกแบบ เพื่อสร้างโครงชุดคำสั่งโดยเลือกรูปแบบการออกแบบชื่อ State แล้วทำการกำหนดค่าต่างๆตามที่ได้ออกแบบไว้ ดังแสดงในรูปที่ 6.11 การกำหนดค่าให้แก่รูปแบบการออกแบบ

2) สร้างส่วนติดต่อกับผู้ใช้ โดยเลือกเริ่มงานใหม่จากแถบเมนู (File/New) จะมีวินโดว์แสดงเทมเพลส(Templates)ให้ผู้ใช้เลือกออปเจคเฟลม แล้วทำการสร้างส่วนติดต่อกับผู้ใช้โดยนำองค์ประกอบได้แก่ ปุ่มคำสั่ง การนำองค์ประกอบซอฟต์แวร์มาสร้างส่วนติดต่อกับผู้ใช้ พร้อมทั้งกำหนดคุณสมบัติขององค์ประกอบต่างๆโดยระบบจะสร้างชุดคำสั่งให้อัตโนมัติ

3) เพิ่มชุดคำสั่งให้สมบูรณ์ ในการสร้างโปรแกรมวาดภาพได้นำรูปแบบการออกแบบสเตทจากรูปที่ 6.9 มาเป็นแนวทางในการออกแบบ จะได้คลาสของโปรแกรมวาดภาพ ดังแสดงในรูปที่ 6.12 โดยสร้างคลาสแม่ชื่อ “State” ประกอบด้วยเมทธอดพฤติกรรมของระบบ คือ “mouseDown” “mouseUp” “mouseDrag” และ “select” แสดงชุดคำสั่งของคลาส “State” ดังรูปที่ 6.13 จากนั้นทำการสับคลาสออกไปเป็น คลาส “ArrowState” คลาส “CircleState” คลาส “FillState” คลาส “RectState” เพื่อกำหนดรายละเอียดให้แก่พฤติกรรมของแต่ละปุ่ม แสดงตัวอย่างชุดคำสั่งของคลาส“RectState” ดังแสดงในรูปที่ 6.14 ซึ่งจะมีคลาส “StateManager” เป็นคลาสที่ควบคุมการทำงาน แสดงชุดคำสั่งของคลาส “StateManager” ดังแสดงในรูปที่ 6.15 การออกแบบในลักษณะนี้จะไม่มีการใช้ประโยค “if-else” หรือ “switch” มา ควบคุมการทำงานเพราะว่าสถานะได้ถูกรวมไว้ภายในคลาสแล้ว



รูปที่ 6.11 การกำหนดค่าให้แก่รูปแบบการออกแบบ

```
import java.awt.*;
public class State
{
public void mouseDown(int x, int y) {}
public void mouseUp(int x, int y) {}
public void mouseDrag(int x, int y) {}
public void select (Drawing d, Color c) {}
}
```

รูปที่ 6.12 ชุดคำสั่งของคลาส “State”

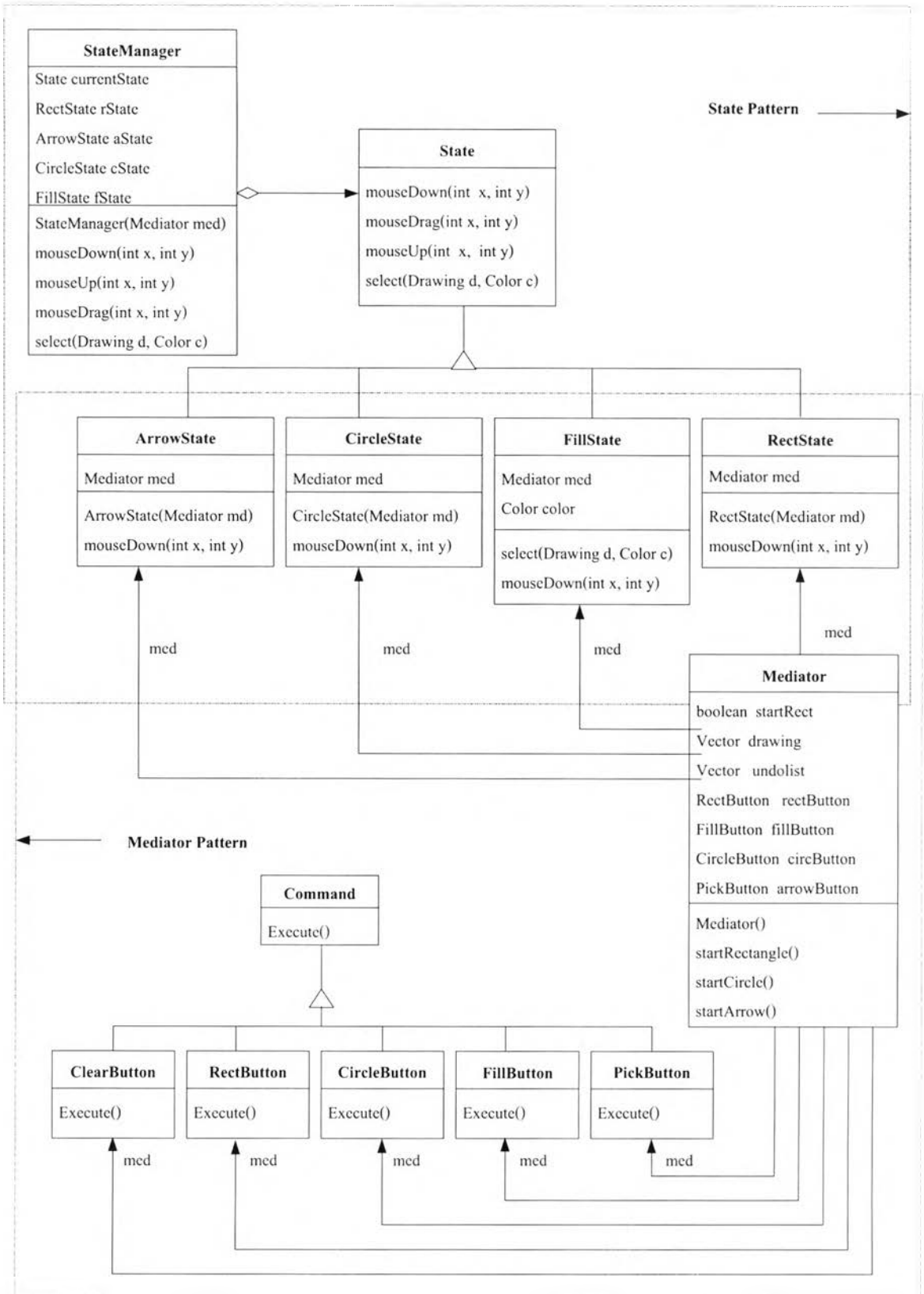
```
public class RectState extends State
{
        private Mediator med;    //save the Mediator here
        public RectState(Mediator md)
        {
                med = md;
        }
        //create a new Rectangle where mode clicks
        public void mouseDown(int x, int y)
        {
        med.addDrawing(new visRectangle(x, y));
        }
}
```

รูปที่ 6.13 ชุดคำสั่งของคลาส “RectState”

```
import java.awt.*;

public class StateManager
{
        private State currentState;

         ArrowState aState;

         public StateManager(Mediator med)
         {
                aState = new ArrowState(med);
                currentState = aState;
                 }
        public void setState(State state) {
                currentState = state;
        }
        public void mouseDown(int x, int y)
        {
                currentState.mouseDown(x, y);
                 }
        public void mouseUp(int x, int y)
                {
                currentState.mouseUp(x, y);
        }
        public void mouseDrag(int x, int y)
                {
                currentState.mouseDrag(x, y);
        }
        public void select(Drawing d, Color c)
                {
                currentState.select(d, c);
         }
}
```

รูปที่ 6.14 ชุดคำสั่งคลาส “StateManager”

ในการสร้างโปรแกรมวาดภาพมีการใช้ออปเจคปุ่มอยู่หลายปุ่ม จึงนำรูปแบบการออกแบบมีดิเอเตอร์มาใช้ทำหน้าที่ประสานงานในการโต้ตอบกันระหว่างออปเจค ดังแสดงในรูปที่ 6.15 แสดงแผนภาพคลาสโปรแกรมวาดรูปที่มีการนำรูปแบบการออกแบบมีดิเอเตอร์มาใช้ ร่วมกับรูปแบบการออกแบบ State ทำให้ทุกครั้งที่มีการกดปุ่มเพื่อเปลี่ยนสถานะของโปรแกรม คลาสมีดิเอเตอร์จะส่งเมสเสจ ให้แก่คลาส “StateManager” เพื่อจัดการเปลี่ยนสถานะของโปรแกรม ชุดคำสั่งของคลาสมีดิเอเตอร์ ดังแสดงในรูปที่ 6.16



รูปที่ 6.15 แผนภาพคลาสโปรแกรมวาดรูปที่มีการนำรูปแบบการออกแบบ Mediator มาใช้ร่วมกับรูปแบบการออกแบบ State [8]

```
public class Mediator
{
  boolean startRect;
  boolean dSelected;
  Vector drawings;
  Vector undoList;
  RectButton rectButton;
  FillButton fillButton;
  CircleButton circButton;
  PickButton arrowButton;
  JPanel canvas;
  Drawing selectedDrawing;
public Mediator()
{
  startRect = false;
  dSelected = false;
  drawings = new Vector();
  undoList = new Vector();
}
public void startRectangle()
{
  arrowButton.setSelected(false);
  circButton.setSelected(false);
  fillButton.setSelected(false);
}
public void startCircle()
{
  rectButton.setSelected(false);
  arrowButton.setSelected(false);
  fillButton.setSelected(false);
}
public void startFill(StateManager stMgr)
{
  rectButton.setSelected(false);
  circButton.setSelected(false);
  arrowButton.setSelected(false);
  stMgr.select(selectedDrawing, Color.blue);
  repaint();
}
public void startArrow()
{
  rectButton.setSelected(false);
  circButton.setSelected(false);
  fillButton.setSelected(false);
}
public Drawing getSelected()
{
  return selectedDrawing;
}
public void fillSelected()
{
  if(dSelected)
  {
    selectedDrawing.setFill(Color.red);
  }
}
public Vector getDrawings()
{
  return drawings;
}
public void addDrawing(Drawing d)
{
  drawings.addElement(d);
}
```

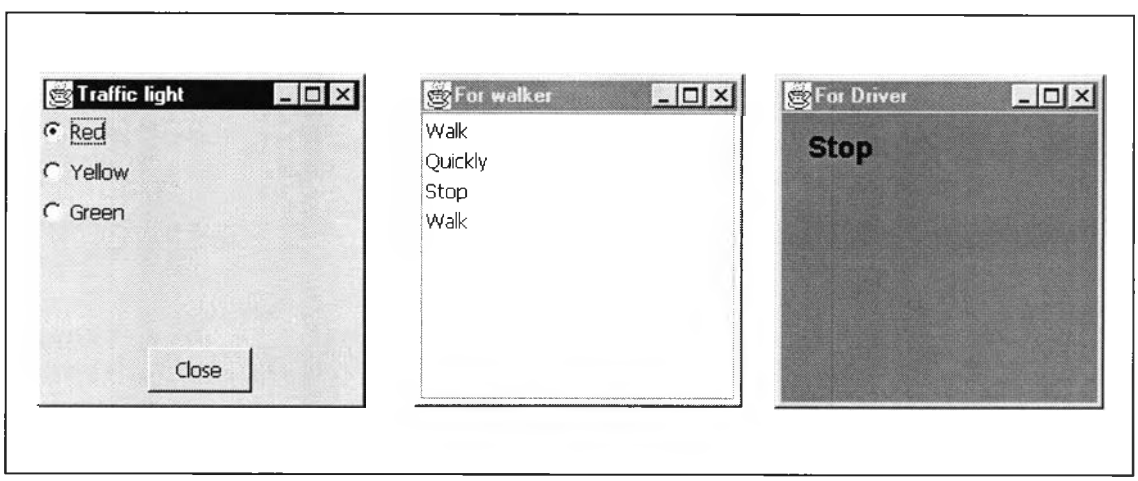
รูปที่ 6.16 ชุดคำสั่งของคลาส “Mediator”

6.3 การสร้างโปรแกรมสัญญาณไฟจราจร

โปรแกรมสัญญาณไฟจราจรประกอบด้วย 3 วินโดว์ วินโดว์แรกจะเป็นการกดเพื่อให้สัญญาณไฟ วินโดว์ที่สองเป็นการแจ้งข้อความสำหรับคนข้ามถนน วินโดว์ที่สามเป็นสัญญาณไฟ

ที่แสดงให้แก่ผู้ขับรถ โปรแกรมสัญญาณไฟจราจรที่สร้างเสร็จแล้ว ดังแสดงในรูปที่ 6.17 มีลักษณะการทำงานดังนี้

1. เลือกปุ่มในวินโดว์แรกเพื่อให้สัญญาณไฟ โดยจะมีสัญญาณสี 3 สี คือ สีแดง สีเหลือง สีเขียว
2. โดยวินโดว์ที่สองเป็นข้อความแจ้งสำหรับคนข้ามถนนโดยจะสัมพันธ์กับการกดให้สัญญาณไฟ เมื่อสัญญาณไฟแดงสามารถข้ามถนนได้จะแสดงข้อความ "Walk" เมื่อสัญญาณไฟเหลืองจะให้คนรีบข้ามถนนโดยเร็วให้พ้นพื้นที่จราจรจะแสดงข้อความ "Quickly" เมื่อสัญญาณไฟสีเขียวจะให้คนหยุดรอข้ามถนนจะแสดงข้อความ "Stop"
3. วินโดว์ที่สามจะเป็นการแสดงสัญญาณไฟจราจรให้แก่ผู้ขับรถ จะสัมพันธ์กับการกดให้สัญญาณไฟ เมื่อกดปุ่ม "Red" จะแสดงสัญญาณไฟสีแดงและข้อความ "Stop" เพื่อให้รถหยุด เมื่อกดปุ่ม "Yellow" จะแสดงสัญญาณไฟสีเหลืองและข้อความ "Slow" เพื่อให้รถเตรียมหยุด เมื่อกดปุ่ม "Green" จะแสดงสัญญาณไฟสีเขียวและข้อความ "Go" เพื่อปล่อยให้รถวิ่งออกไปได้



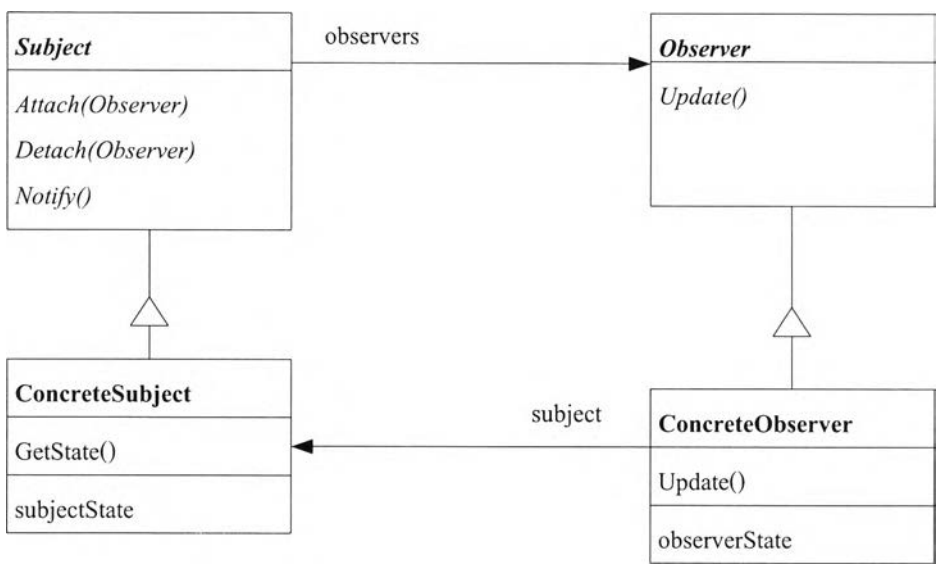
รูปที่ 6.17 ผลการสร้างโปรแกรมสัญญาณไฟจราจร

6.3.1 รายละเอียดปัญหาในการเขียนโปรแกรมสัญญาณไฟจราจร

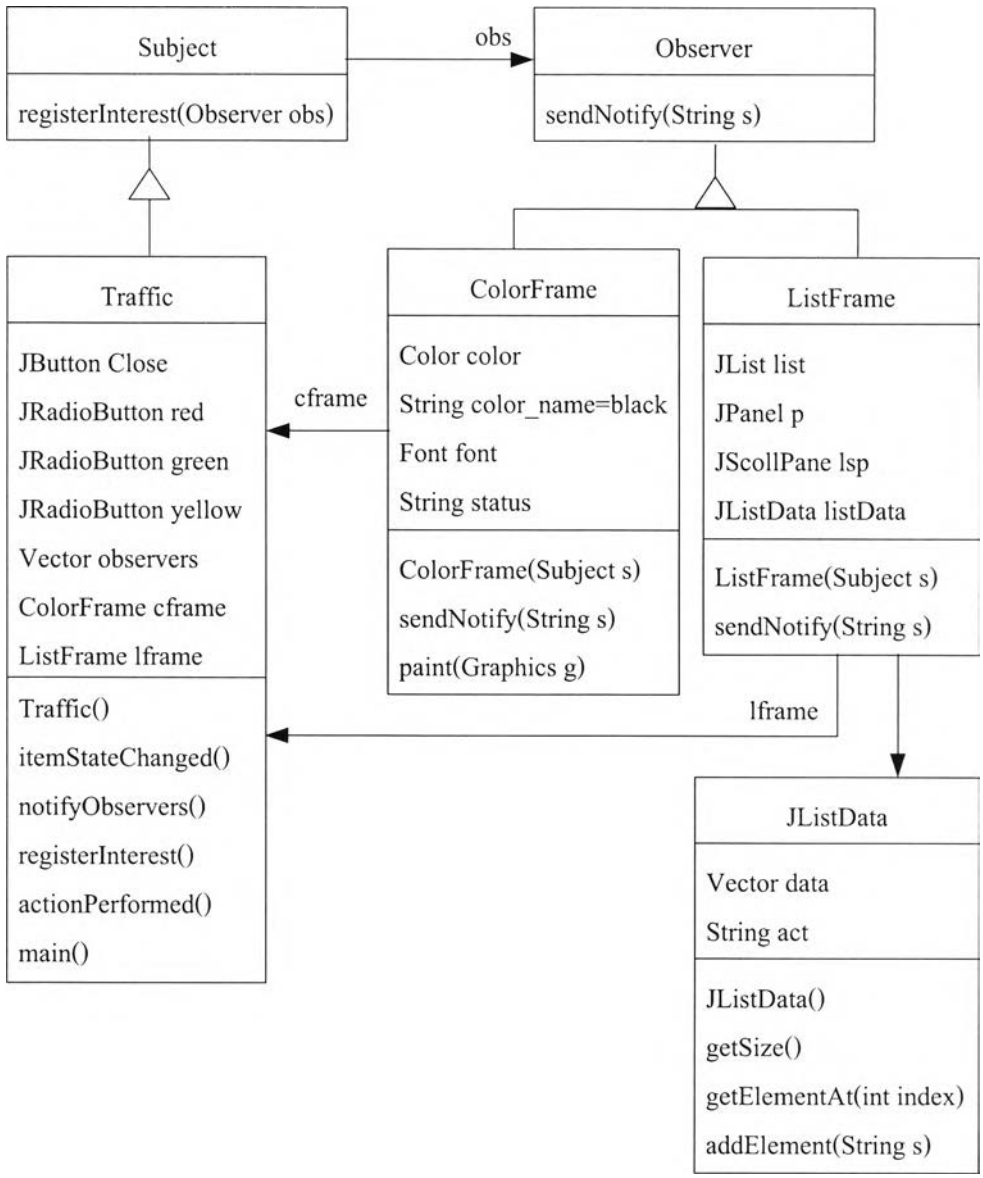
ในการออกแบบโปรแกรมบ่อยครั้งที่ต้องการแสดงผลข้อมูลหลายรูปแบบในเวลาเดียวกันและการแสดงผลในรูปแบบต่างๆต้องสัมพันธ์กับการเปลี่ยนแปลงของข้อมูลด้วย ซึ่งในการเขียนโปรแกรมอาจจะใช้วิธีเขียนชุดคำสั่งเพื่อเช็คการเปลี่ยนแปลงข้อมูล แล้วทำการสั่งให้ปรับปรุงการแสดงผลทั้งหมด ซึ่งจะต้องระวังในการเขียนโปรแกรมให้ปรับปรุงการแสดงผลให้ครบ และยุ่งยากในการแก้ไขโปรแกรมเมื่อมีการเพิ่มหรือลดรูปแบบการแสดงผล

6.3.2 แนวทางการแก้ปัญหา

จากปัญหาดังกล่าว Cooper,W.J.[8] ได้นำรูปแบบการออกแบบชื่อ Observer มาใช้ในการออกแบบโปรแกรม ซึ่งมีหลักการคือกำหนดความสัมพันธ์แบบ "one-to-many" ระหว่างออปเจค เมื่อออปเจคหนึ่งเปลี่ยนสถานะ ออปเจคที่สัมพันธ์ด้วยทั้งหมดจะถูกแจ้งให้ทราบและปรับปรุงโดยอัตโนมัติ รูปแบบการออกแบบออปเซิฟเวอร์ ดังแสดงในรูปที่ 6.18 ในการออกแบบจะถือว่าออปเจคที่เป็นแหล่งข้อมูลแยกออกจากออปเจคที่แสดงผลข้อมูล ในการใช้งานจะกำหนดให้อินเทอร์เฟสคลาสซับเจค (Subject) แทนแหล่งข้อมูล และอินเทอร์เฟสคลาสออปเซิฟเวอร์จะแทนการแสดงผล



รูปที่ 6.18 รูปแบบการออกแบบ Observer [3]



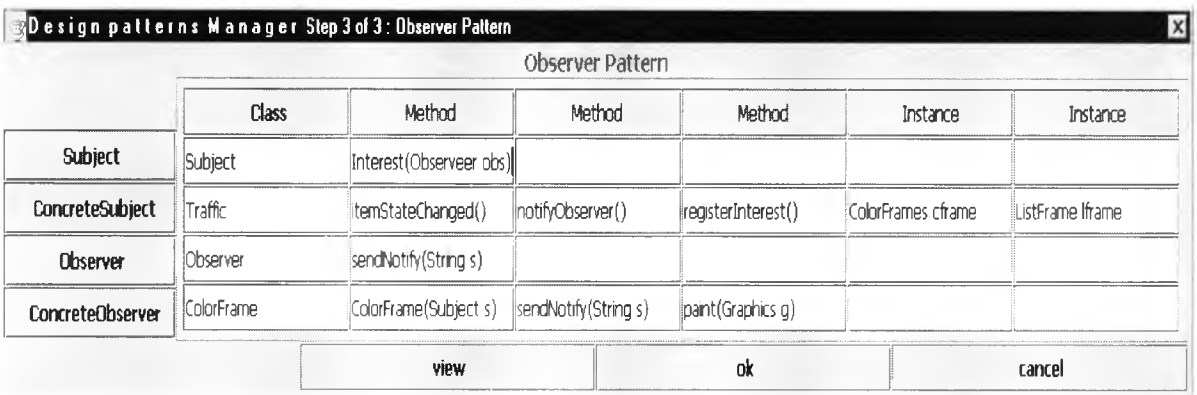
รูปที่ 6.19 แผนภาพคลาสโปรแกรมสัญญาณไฟจราจรที่มีการใช้รูปแบบการออกแบบ Observer มาใช้ [8]

6.3.3 ขั้นตอนการสร้าง

1) สร้างโครงชุดคำสั่งรูปแบบการออกแบบ โดยเข้าสู่ระบบเลือกเมนู "Design Pattern\Pattern Manager" จะเป็นการเรียกใช้งาน ส่วนจัดการรูปแบบการออกแบบ เพื่อสร้างโครงชุดคำสั่งโดยเลือกรูปแบบการออกแบบชื่อ Observer แล้วทำการกำหนดค่าต่างๆตามที่ได้ออกแบบไว้ ดังแสดงในรูปที่ 6.20 การกำหนดค่าให้แก่รูปแบบการออกแบบ

2) สร้างส่วนติดต่อกับผู้ใช้ โดยเลือกเริ่มงานใหม่จากแถบเมนู (File/New) จะมีวินโดว์แสดงเทมเพลส(Templates)ให้ผู้ใช้เลือกออปเจคเฟลม แล้วทำการสร้างส่วนติดต่อกับผู้ใช้โดยนำองค์ประกอบได้แก่ ปุ่มคำสั่ง การนำองค์ประกอบซอฟต์แวร์มา สร้างส่วนติดต่อกับผู้ใช้ พร้อมทั้งกำหนดคุณสมบัติขององค์ประกอบต่างๆโดยระบบจะสร้างชุดคำสั่งให้อัตโนมัติ

3) เพิ่มชุดคำสั่งให้สมบูรณ์ ในการสร้างโปรแกรมสัญญาณไฟจราจร ได้นำรูปแบบการออกแบบ Observer จากรูปที่ 6.18 มาเป็นแนวทางในการออกแบบ จะได้คลาสของระบบโปรแกรมสัญญาณไฟจราจร ดังแสดงในรูปที่ 6.19 โดยสร้างคลาส “Traffic” ทำการอิมพีเม้นท์ อินเทอร์เฟสคลาสซับเจค ซึ่งคลาส “Traffic” จะเป็นแหล่งข้อมูลคือเป็นการให้สัญญาณไฟจราจรสีต่างๆ คลาส “ColorFrame” และ คลาส “ListFrame” ทำการอิมพีเม้นท์อินเทอร์เฟสคลาส “Observer” ซึ่งเป็นคลาสแสดงไฟสัญญาณจราจรและคลาสแสดงข้อความให้คนข้ามถนน ตามลำดับ



รูปที่ 6.20 การกำหนดค่าให้แก่รูปแบบการออกแบบ

```
public class Traffic extends JxFrame
implements ActionListener, ItemListener, Subject {
    private JButton Close;
    private JRadioButton red, green, yellow;
    private Vector observers;
    private ColorFrame cframe;
    private ListFrame lframe;

    public Traffic() {
        super("Traffic light");
        observers = new Vector();          //list of observing frames

        JPanel p = new JPanel(true);       //add panel to content pane
        p.setLayout(new BorderLayout());
        getContentPane().add("Center", p);

        Box box = new Box(BoxLayout.Y_AXIS);  //vertical box layout
        p.add("Center", box);
        box.add(red = new JRadioButton("Red"));     //and 3 radio buttons
        box.add(yellow = new JRadioButton("Yellow"));
        box.add(green = new JRadioButton("Green"));

        yellow.addItemListener(this);        //listen for clicks
        red.addItemListener(this);         //on radion buttons
        green.addItemListener(this);

        //make all part of same button group
        ButtonGroup bgr = new ButtonGroup();
        bgr.add(red);
        bgr.add(yellow);
        bgr.add(green);

        //put a  Close button at the bottom of the frame
        JPanel p1 = new JPanel();
        p.add("South", p1);
        p1.add( Close =new JButton("Close"));
        Close.addActionListener(this);     //listen for clicks on it
        setBounds(100, 100, 200, 200);
        //pack();
        //---------create observers---------
        cframe = new ColorFrame(this);
        lframe = new ListFrame(this);
        setVisible(true);
    }
    //-----------------------------------------
    public void itemStateChanged(ItemEvent e) {
        //responds to radio button clicks
        //if the button is selected
        if (e.getStateChange() == ItemEvent.SELECTED)
            notifyObservers((JRadioButton)e.getSource());
    }
    //-----------------------------------------
    private void notifyObservers(JRadioButton rad) {
        //sends text of selected button to all observers
        String color = rad.getText();
        for (int i=0; i< observers.size(); i++) {
            ((Observer)(observers.elementAt(i))).sendNotify(color);
        }
    }
    //-----------------------------------------
    public void registerInterest(Observer obs) {
        //adds observer to list
        observers.addElement(obs);
    }
    //-----------------------------------------
    public void actionPerformed(ActionEvent e) {
        //responds to close button
        Object obj = e.getSource();
        if (obj == Close)
            System.exit(0);
    }
    //-----------------------------------------
    static public void main(String[] argv) {
        new Traffic();
    }
}
```

รูปที่ 6.21 ชุดคำสั่งของคลาส “Traffic”

```
public class ColorFrame extends JFrame
implements Observer {
   private Color color;
   private String color_name="black";
   private Font font;
   private String status ="";

   private JPanel p = new JPanel(true);
   public ColorFrame(Subject s) {
      super("For Driver");
      getContentPane().add("Center",  p);
      s.registerInterest(this);
      setBounds(550, 100, 200, 200);
      font = new Font("Sans Serif", Font.BOLD, 18);
      setVisible(true);
   }
   public void sendNotify(String s) {
      color_name = s;
      if (s.equalsIgnoreCase ("RED")){
         color = Color.red;
         status = "Stop";
               }
      if (s.equalsIgnoreCase ("YELLOW")){
         color =Color.yellow;
         status = "Slow";
               }
      if (s.equalsIgnoreCase ("GREEN")){
         color = Color.green;
         status = "Go";
               }

      //p.repaint();
      setBackground(color);
   }
   public void paint(Graphics g) {

      g.setFont(font);
      //g.drawString(color_name, 20, 50);
      g.drawString(status, 20, 50);
   }
}
```

รูปที่ 6.22 ชุดคำสั่งของคลาส “ColorFrame”

```
public class ListFrame extends JFrame
implements Observer {
   private JList list;
   private JPanel p;
   private JScrollPane lsp;
   private JListData listData;
   private Font font;

   public ListFrame(Subject s) {
      super("For walker");
      //put panel into the frmae
      p = new JPanel(true);
      getContentPane().add("Center", p);
      p.setLayout(new BorderLayout());
      //Tell the Subject we are interested
      s.registerInterest(this);

      //Create the list
      listData = new JListData();  //the list model
      list = new JList(listData);  //the visual list
      lsp = new JScrollPane();     //the scroller
      lsp.getViewport().add(list);
      p.add("Center", lsp);
      lsp.setPreferredSize(new Dimension(200,200));
      setBounds(330, 100, 200, 200);
      font = new Font("Sans Serif", Font.BOLD, 18);
      setVisible(true);
   }
   //---------------------------------
   public void sendNotify(String s) {
      listData.addElement(s);
   }
}
```

รูปที่ 6.23 ชุดคำสั่งของคลาส “ListFrame”

6.4 การวิเคราะห์ผลการทดสอบ

หลังจากทำการทดสอบสร้างโปรแกรม โดยใช้โครงร่างสำหรับสร้างโปรแกรมประยุกต์เชิงวัตถุ จะได้ชุดคำสั่งอัตโนมัติเป็นลักษณะโครงชุดคำสั่ง(skeleton code)เพื่อให้ผู้ใช้เติมชุดคำสั่งให้สมบูรณ์ การสร้างชุดคำสั่งอัตโนมัตินี้ทำให้ลดเวลาในการเขียนชุดคำสั่งเอง ผลปรากฏว่ามีสัดส่วนของจำนวนโครงชุดคำสั่งที่สร้างได้โดยอัตโนมัติวัดโดยบรรทัดของชุดคำสั่งที่สร้างได้โดยอัตโนมัติเทียบกับจำนวนบรรทัดของชุดคำสั่งที่ใช้งานจริง ดังแสดงในตารางที่ 6.1

จากผลการทดสอบแสดงให้เห็นว่า โครงร่างที่พัฒนาขึ้นที่สามารถช่วยแบ่งเบาภาระในการสร้างชุดคำสั่ง ถ้าโปรแกรมมีการใช้องค์ประกอบของซอฟต์แวร์สร้างส่วนติดต่อกับผู้ใช้ สัดส่วนของชุดคำสั่งที่สร้างอัตโนมัติจะมีจำนวนมาก แต่ถ้าโปรแกรมมีความซับซ้อนไม่สามารถสร้างโปรแกรมใช้งานจากองค์ประกอบซอฟต์แวร์ที่มีอยู่ได้จำเป็นต้องเขียนชุดคำสั่งเพื่อสร้างคลาสเองก็จะทำให้สัดส่วนของชุดคำสั่งที่สร้างอัตโนมัติน้อยไปด้วย จากการสร้างโปรแกรมเครื่องคิดเลข

มีการใช้ออปเจคปุ่มจำนวนมาก ซึ่งถ้าใช้การสร้างปุ่มต่าง ๆ จากองค์ประกอบมาวางโดยตรงจะมีชุดคำสั่งที่สร้างให้อัตโนมัติจำนวนมาก แต่ก็จะมีจำนวนบรรทัดชุดคำสั่งที่ใช้งานทั้งหมดมากขึ้นตามไปด้วยซึ่งในโปรแกรมนี้ได้ออกแบบให้มีการใช้รูปแบบมีดิเอเตอร์ ที่ต้องการสร้างคลาสออปเจคปุ่มเอง จึงมีจำนวนบรรทัดที่ต้องเขียนเองมากแต่จำนวนบรรทัดที่ใช้งานจริงก็น้อยลงไปด้วย

ตารางที่ 6.1 จำนวนบรรทัดของโครงชุดคำสั่งที่สร้างอัตโนมัติเปรียบเทียบกับจำนวนบรรทัดคำสั่งทั้งหมดที่ใช้งานจริง

โปรแกรม	จำนวนบรรทัดคำสั่งทั้งหมดที่ใช้งานจริง	จำนวนบรรทัดโครงชุดคำสั่งที่สร้างอัตโนมัติ	ร้อยละของบรรทัดคำสั่งที่สร้างอัตโนมัติต่อการใช้งานจริง
เครื่องคิดเลข	216	90	41.70
วาดรูป	1002	338	33.70
สัญญาณไฟจราจร	183	64	34.97