University of Camerino

School of Science and Technology
Master of Science in Computer Science

Complex System Design Project

OPENMOTION:

A POINTING SYSTEM WITH

OBJECT RECOGNITION

Proposer:

Andrea Monacchi

Academic Year 2010-2011

Contents

List of Figures

Chapter 1

Introduction

Since the advent of mice, it becames clear that a computer was not just a simple calculator but an interface to functionality and informations, previously unthinkable.
Nowadays, we are witnessing the launch of increasingly powerful computers and increasingly complex software. The development of human-machine interfaces is becoming increasingly important to make these technologies more accessible. In this project we focus on a pointing system that uses computer vision to recognize the position of an object in the scene.

1.1 Context

Information Technology offers access to a wide range of services but often weak person like elderly people cannot take advantage from this. In these cases becomes important to provide a different interface with the computer taking into account the low level of knowledge of these systems by those people. Thus, the first requirements come out from the target and is simplicity and low cost. For these reasons we have choosen to use a common webcam to detect the position of a special object in the scene. The object, as we will see in the next chapter, uses LEDs (Light Emitting Diode) to be visible from the system when the user press a special button. However, a pointing system could be used for different purposes and we report few of these:

- game controls
- remote control for media centers
- digital shop windows
- "wireless" business and academic presentation

In the first part of this document we report our implementation of the system and in the last part we discuss about the possibility to use a special dispatcher and our own events to implement more sophisticated applications.

1.2 Organization

In the second chapter we specify the requirements and we make some initial choices about the technologies and methodologies to use.
In the third chapter we design the system taking into account the requirement and the target of this product.
In the fourth chapter we report a case study where we use the pointing system to manage mouse and a special pointer.
In the fifth chapter we verify the system and we evaluate performance and functionalities against the requirements.
Last chapter includes an estimate of construction costs and materials used.

Chapter 2

Requirements analysis

In this chapter we define the aim of the project and then we estabilish design constraints and choices.

2.1 Project requirements

As defined in the previous chapter, we want to provide to our target an easy to use tool that allows the remote control of the graphic interface. We use a special object that can become visible from the system whenever the user wants. The system is composed by a webcam and an interface Agent that recognizes the position of the object (if it is enabled) and from this information produces Events on the GUI.

We should use a framework or a similar tool that would permit us to mask the part closely linked to the webcam management. In this way, we can focus our work on the image processing and event dispatching part without worryng about low-level mechanisms. However, usually it is important to define some requirements and constraints on the system so that next we can make the right choices in the design step.

2.1.1 The controller

We want a controller that can become visible when the user wants. To do this we could make a wireless controller that could use batteries to power a LED. User could use a push-button to toggle the on-off state of that led.

2.1.2 The interface Agent

As we will see in the next chapters, we want to make an interface Agent that detects events from the position of the controller recognized from photos. This means we have to shot and process images to find the position of the controller. We could put many constraints on the system, first at all we suppose that we can use just a single controller per time. In this way the system design become

very easy to do. We can also put constraints on the system performance. We choose to use a single color for the controller in a way that we don't have to check every time different color for the user. System performances also affects our choice on framework and programming language but there are also other parameters to consider like technology and time costs. The important things about this product are constraints on cost and simplicity that comes out from the defined target (families and elderly people). To meet these constraints we have to use opensource resources like IDEs, programming languages, libraries (or packages), existing projects and documentation that can help us to enter into these technologies in the shortest time.

2.1.3 The project

Since we use opensource resources and this project is for an exam we chose to release all the source code with an opensource license like BSD. In fact, we added the project to sourceforge dot net.
An interesting idea about this project is: we could make a website where everyone can join and post his application using the system. In this way we are sure that users appreciate this idea and we hope they will contribute to the project develop adding new functionalities and application.

Figure 2.1: The project logo of OpenMotion

2.1.4 Risk analisis

The main risk in this project is the need to learn too many concept on computer vision. Luckily, one of us has experience with similar problems and it becomes feasible to us use familiar tools. In effect, if we would want optimize the performance we should choose openCV and C++ (or better C). Unfortunately, building application in C/C++ with a library not yet known requires time and this is a cost that we cannot sustain. For these reasons we chose to use Java with the library JMF (Java Media Framework). Java permits us to achieve our results in the shortest time and this is crucial for time-to-market. In future we could pass to openCV with lower risk.

2.1.5 Estimated resources requirements and costs

To make the controller we need:

- red LED 5mm
- resistor (voltage adaption)
- button switch
- battery enclosure
- 2 x AA 1,5 V Battery

To develop the system we need:

- Computer with JRE, JDK, JMF, usb port
- USB Webcam

Fortunately we have these resources to make the prototype, we need to buy just the button switch and the battery enclosure.
To build the system we estimate 7 day/man cost.

2.1.6 Activity plan

The involved activities are:

- build the controller - 1h
- work-environment setup (webcam drivers, JRE, JDK, JMF) - 1h
- InterfaceAgent implementation and test - 7 days
  - PointDetector coding - 5 days
  - EventDetector coding - 2 days
- case study development - 2 days
- writing documentation - 4 days

Chapter 3

Design

In this chapter we make the controller, then we discuss about the architectural design of the system and we report the whole UML diagram.

3.1 Controller construction

To make the controller we take the battery enclosure and we connect the positive to the button. We connect the negative directly to the LED, with a conductor wire. Finally we connect the button to the resistor and the resistor to the anod of LED.
The resistor value is calculated taking into account that a common led lights if is crossed by a current between 8mA and 20mA but some type can tolerate higher currents. To compute the resistor value the equation is:

$$R_{DL} = \frac{V_{dd} - V_{DL}}{I_{DL}} = \frac{3-2}{20 * 10^{-3}} = \frac{1000}{20} = 50\Omega$$

3.2 modeling with UML

In this section we present our UML model of the system and then we explain the classes involved.

3.2.1 The Interface Agent

The package interfaceAgent contains the whole pointing system. The class InterfaceAgent instantiates an EventDetector and a BlobRecognizer and hence this class wraps the whole system. BlobRecognizer takes a photo with webcam and after a process phase returns the controller position in the scene. Event Detector gets the current position of the controller and maintains the past position. In this way it can discriminate between different types of event. Event Detector uses the Java Class Robot to manage the mouse.

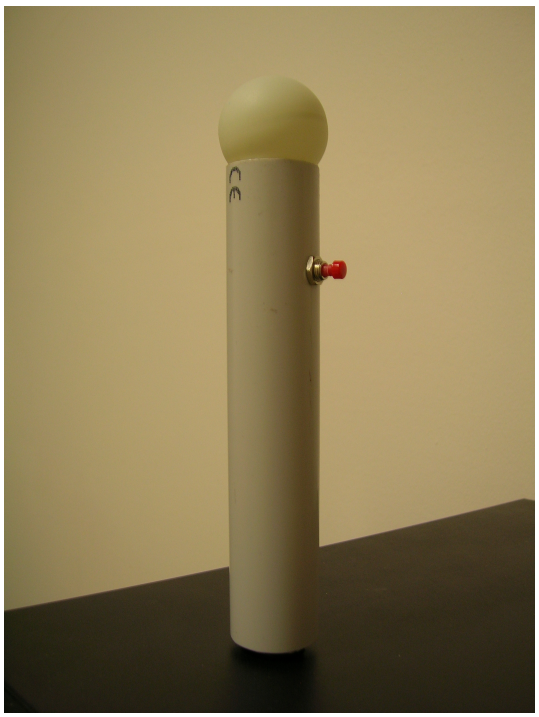

Figure 3.1: The Controller

3.2.2 PointDetector

In the package PointDetector we have five classes: BlobRecognizer, CameraManager, CameraException, Blob and Point.

BlobRecognizer

BlobRecognizer gets the reference of the CameraManager at creation time and then starts (with its own control flow) to take photo from webcam invoking the getImage method. This method acquires the photo and its dimension from CameraManager. Then it gives a value false to a matrix of boolean that represents visited Pixels of the image. Next, run invokes the method process passing the image. Process method visits through the image (pixel by pixel) and checks for enabled pixels by invoking for each one the method isEnabled which returns true if the pixel color is within a certain value range. We chose red as enabled color.
If the pixel is enabled, the process method instantiates a new Blob with the same dimensions of the original image. Then it invokes the method findBlob to find enabled pixels around the current one. FindBlob method is a recursive function that invokes itself on the adjacent pixels and try to attach these pixels to the blob if they are enabled. FindBlob ends when is impossible to find other enabled pixels or when it reaches the image borders. The process

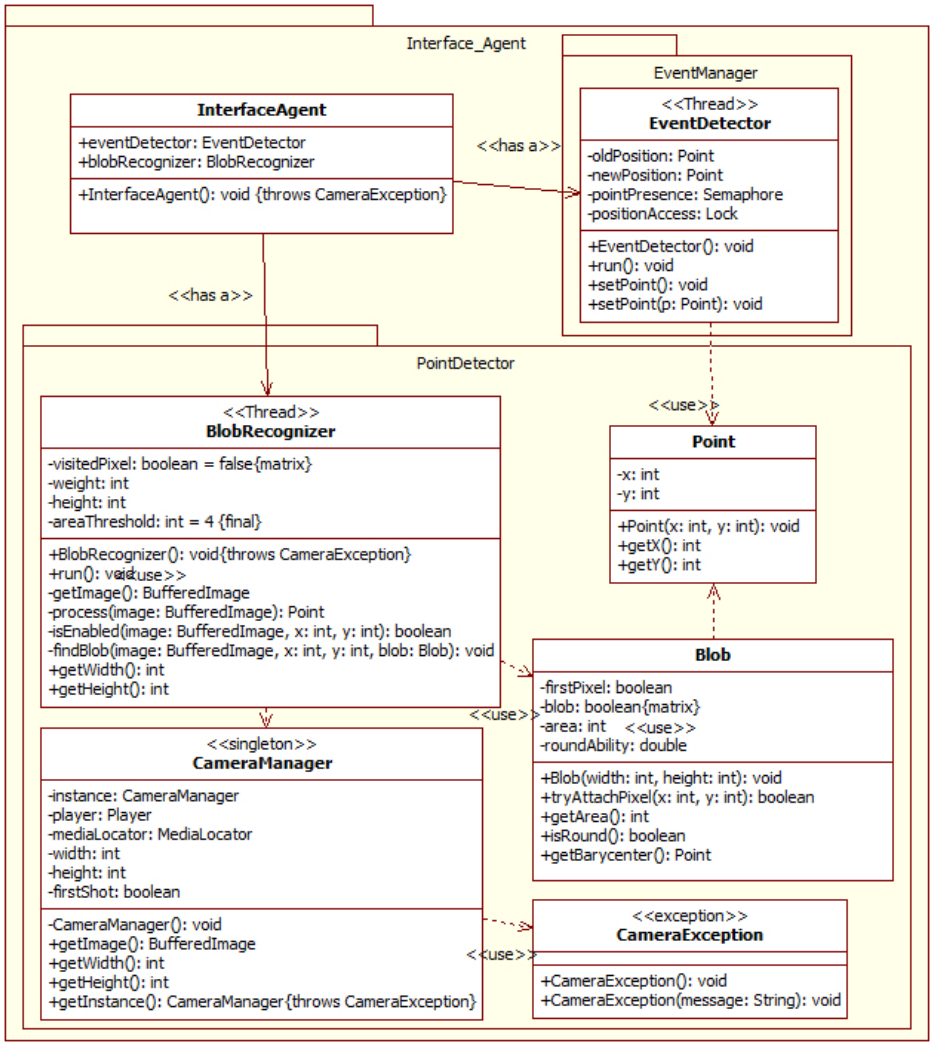

Figure 3.2: Class diagram of the system

method go ahead checking if the area of the found blob is more than the defined threshold. If this is true, it checks for the circularity of the blob (to reject any noisy blob) and in positive case it returns the blob barycenter adapted to the screen resolution. Otherwise the process method returns null. Now, we are in the run method and we go ahead checking if the point returned by process is null or not. If it is null run invokes setPoint passing no parameter to the eventDetector. Otherwise it invokes setPoint passing the found point (methods overloading).

CameraManager

CameraManager is invoked by BlobRecognizer to shot photos. At the creation time, it instantiates a media locator for the webcam and a player to get the data stream. Now it is ready to take photos from that stream. When BlobDetector invokes getImage, it takes a frame from the data flow and checks for the correctness of acquired datas. Then, it gets the dimension of the frame and converts the byte[] that represents the frame in an interlaced way (RGB, with a byte for every channel) to an alfaRGB format which can be passed to a BufferedImage and showed on the screen. GetImage method returns such BufferedImage. To create just an instance of CameraManager we use the singleton design pattern where a special method getInstance looks for CameraManager instance before instantiate a new one. GetInstance method can throws a CameraException, this type of exception, is used to mask the exceptions IOException, NoPlayerException, CannotRealizeException which can be thrown by the constructor (the real one, which is private). In this way we can use personal error messages to specialize the exceptional flow. For instance, if at creation time is not possible to access the webcam a CameraException is thrown.

CameraException

CameraException is an Exception class which extends Exception and passes to its parent an error message. For this reasons two constructor are make, one receives a String (error message) whereas the other one receives no parameters and show the default message.

Blob

Blob is an entity which is used to represent found blob from the processing phase. Every blob has an area and a roundAbility which we use to discriminate noisy shapes from correct ones. At start a boolean matrix which represent the image is set false. When BlobDetector tries to attach pixels to the blob (with the method tryAttachPixel) we check if this is the first pixel of blob (in this case we are sure that the pixel is enabled and is belonged by the blob) or else we check if there is an adjacent pixel that belongs to the blob. In these case we add the pixel to the blob simply setting the boolean location of that pixel

to true. Blob provide several methods to BlobDetector: getArea, isRound and getBarycenter. GetArea returns the blob area whereas isRound checks if the blob is circular. Finally getBarycenter returns the blob central point (centroid) wich can represent it.

Point

Point is used to represent Blob barycenters. A Blob is defined by an x position and a y position. The information hiding leads us to hide these attributes and get them with two methods (getX and getY).

3.2.3 EventManager

In the package EventManager we have just the class EventDetector. EventDetector is an active entity (it has its own execution thread) which receives a point from the BlobDetector and processes these point triggering events on their position. In fact, EventDetector thread is a while(true), after instantiate the Robot Class it blocks itself until a point is available. When BlobDetector pass a point to the EventDetector (with setPoint(Point p) or setPoint()) it wakes up and defines the event taking into account the old position of the Point. If the newPosition is not null we have a move of the pointer, otherwise if the old position was not null and the current it is it triggers a Press and a Release on the mouse interface. To prevent concurrent access to the newPosition attributes it uses a Lock called positionAccess which is acquired before reading/writing the data and released after the operation on in.

Chapter 4

Case Study

In this chapter we report a case study where we make an application to view images using the pointing system. Once we choosen the application, we discuss about the possibility to manage it using the system mouse and then we propose another pointer using events bounded to the application.

4.1 The application

We want to create a simple image viewer which loads images from a source directory and gives to the user the possibility to flow the image list using special buttons. In this way we can evaluate directly the power of the tool and the bounds of the technology used.

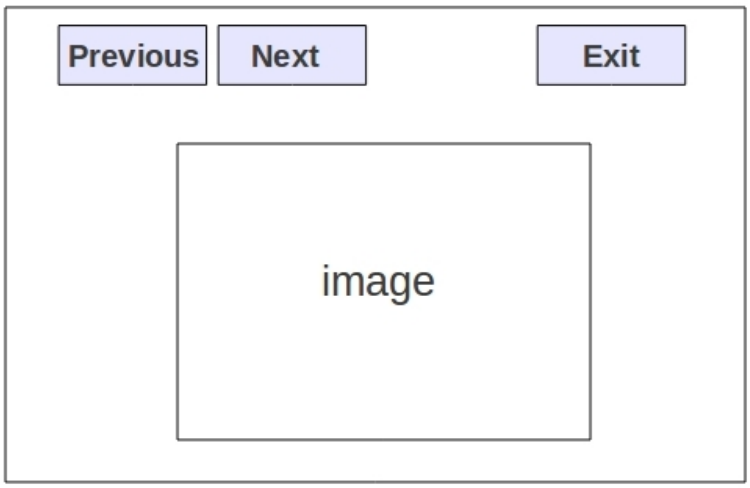

Figure 4.1: the GUI draft

4.1.1 use of mouse

As we have seen in the design chapter we can manage the mouse using the controller. In this way we can do the same things we do with a classical mouse and so we can implement many application using the interface Agent as a sort of mouse manager. This means that we can realize any type of application also using different programming languages. Our example application is a classical application with a class GUI where we define the graphics components to use and a class Listener where we define the action to take for events on these components. A class controller is used to separate the view from the model (according to the architectural pattern model-view-controller). The main method instantiates the GUI, the GUI instantiates the Listener and the Controller, the Controller instantiates the interface Agent or otherwise we can separate the application from the Agent compiling these components in two different program. We report the application code in appendix.

4.1.2 Make an embedded pointer

In many application we could want to bound the events made with the controller into our application. For instance, we can suppose we have a shop in a shopping center and we want to provide to the customer the possibility to look for our offers using the controller to run the album of our goods on sale. If we use the system pointer we give to the user the possibility to access to the computer whereas we would like he can only look for our offers. In this case we have to make a special dispatcher and trigger our own events by the event detector. In this section we present how to do this.

Our Events

We assume we need a click event that is trigger when the user moves the pointer on a graphic component and releases it. We also need an action that refreshes the pointer position whatever event is triggered. To do this we make an abstract class PointerAction that has a position x and a position y to describe the position of the action to perform on the interface. That class is extended by two classes which are Position and Event. The first class is just an instantiable version of PointerAction and provides a costructor to do this. On the contrary, the class Event is again an abstract class but it adds the concept of Component where the Event is triggered. For this reason the Event class provides the methods setSource and getSource to modify that attribute. However, since the Event class is an abstract class it cannot be instantiated. The Event class is useful to mask the different Event type we can have on the interface. Every concrete Event has to specialize that class. In fact, we provide the Event click extending the class Event and defining the constructor. Every required Event can be added by simply specializing that class. We see below a scheme of the hierarchy.

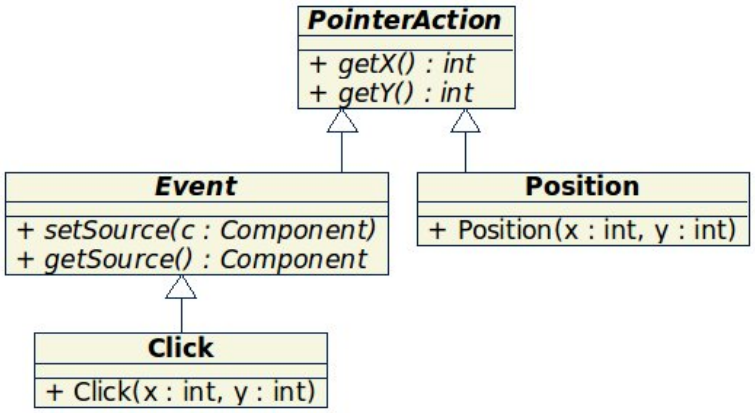

Figure 4.2: the PointerAction hierarchy

The Dispatcher

To manage our own events we need to modify the eventDetector in a way that it instantiates our PointerActions instead of manage the mouse via the Robot Java class. For this reason we use a vector of actions which the EventDetector fills and then passes these events to EventDispatcher to process them. The Dispatcher maintains an Hashtable to refer the Listener from a Component to get the Actions of a certain event on a graphic Component. The EventDispatcher waits until there aren't actions to process. When the EventDetector passes the actions, it unlocks the dispatcher which was waiting on the semaphore and it can go ahead to process all the action received. First it checks the action type, if the action is a position, it moves the pointer position to the new one (we will see how). If the action is an event (in this case we have just the click one), it looks for a top-level Component placed in the same event position and assign that Component to the event via setSource method. To get the top level components we use a recursive function that invoke itself until it reaches a component whose father is null.
Then the dispatcher gets the listener for that component and invokes clickAction (for instance, in case of click) on it. Finally the dispatcher clears its event list and blocks itself until new actions are ready.

The pointer management

We said we don't want to use the system pointer to interact with the GUI. Thus, we define a class Pointer which we use to represent the cursor. This class extends JPanel and has an attribute of Image type. With this attribute we define the appearance of the cursor. To modify it we provide a method setPointer and we ovverride the paint method to draw the image into the component bounds. The pointer is instantiated by the programmer into the application and is passed to the interfaceAgent via a setPointer method. This

method invokes the setPointer method on the EventDispatcher where we assign the reference to the pointer attribute. This means the user application can change its appereance at run-time. In effect InterfaceAgent provide to the programmer the methods addListener, removeListener and setPointer. Both of them invoke namesake methods on the dispatcher to respectively add/remove a reference Component-listener in the Hashtable or set the reference to the Pointer imagePanel.

Graphic Components

To use our events we need to extend the functionality of common graphic components. We create an OpenMotionComponent interface where we declare the abstract method addListener and removeListener. Every graphic component we want to use in our application is a graphic component (for instance a JPanel) and has to implement this interface. In fact, we extend the JButton and JTextField classes implementing on them the OpenMotionComponent interface. These methods use the static reference to the dispatcher to invoke on it the methods addListener and removeListener.

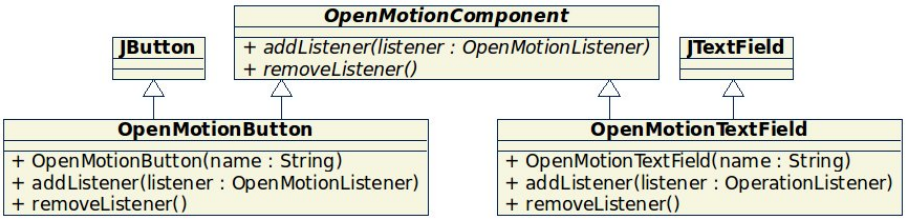

Figure 4.3: the graphic Components hierarchy

4.1.3 The Listener

To provide to the user the possibility to define actions for our events we have to define an interface OpenMotionListener where we insert our event handler. For instance, in our case we use just the click event, so in our interface we have just the abstract method clickAction.
As we will see in the code section the listener is extended by the user to define his actions, it is instantiated by the user application and is referenced by every hot component (a component is hot if can trigger events).

4.1.4 The system

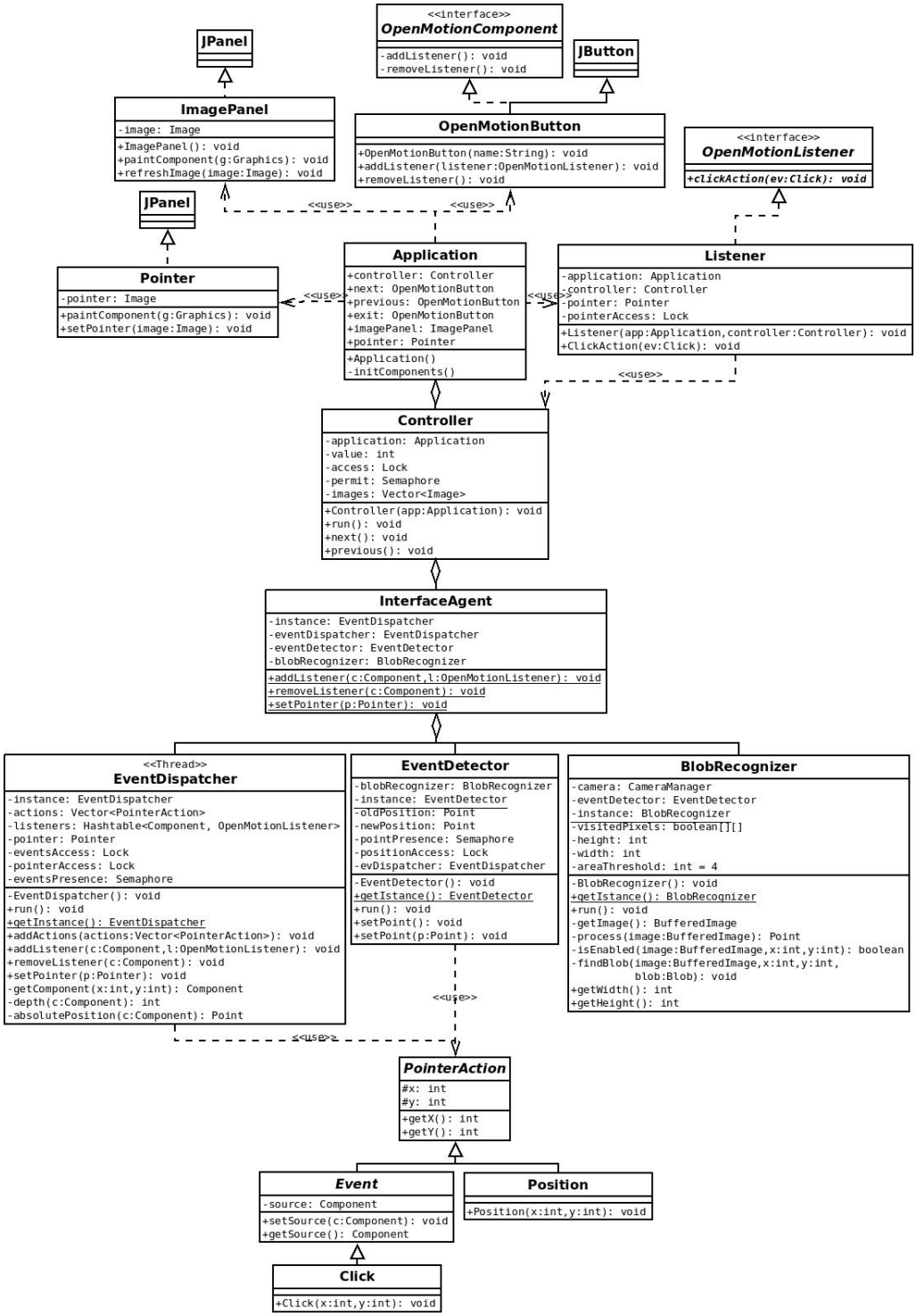

Figure 4.4: the system

Chapter 5

Validation testing

In this chapter we report the results of the functional testing of the system.

5.1 testing

The testing phase has been evaluated on the systems made. First we executed the mouse manager version and then the bounded one.

5.1.1 Mouse Manager

In this version we use the interface Agent to manage the system mouse. We can see in the picture below the application interface. The application is a classical Java application and is independent from the interface agent.

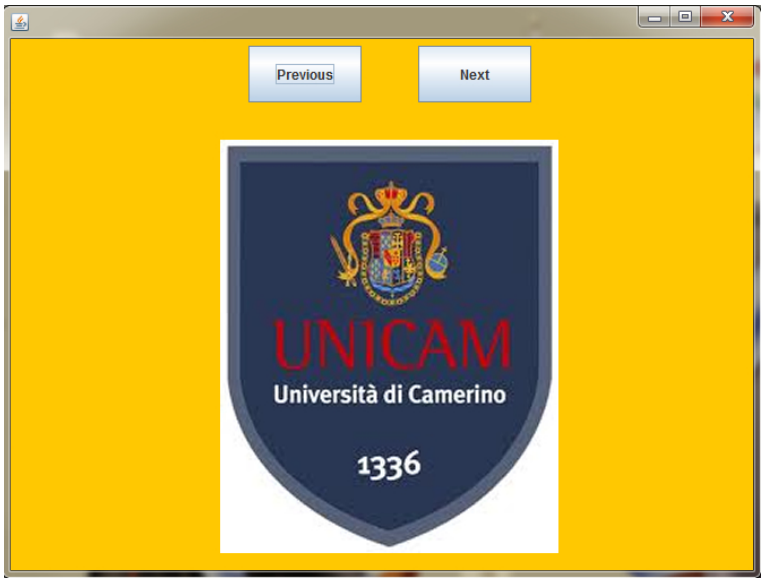

Figure 5.1: GUI

5.1.2 Bounded with event dispatcher

In this version we use the interface manager to produce and dispatch events on the graphic interface (on our components). You can see in the picture our special pointer.

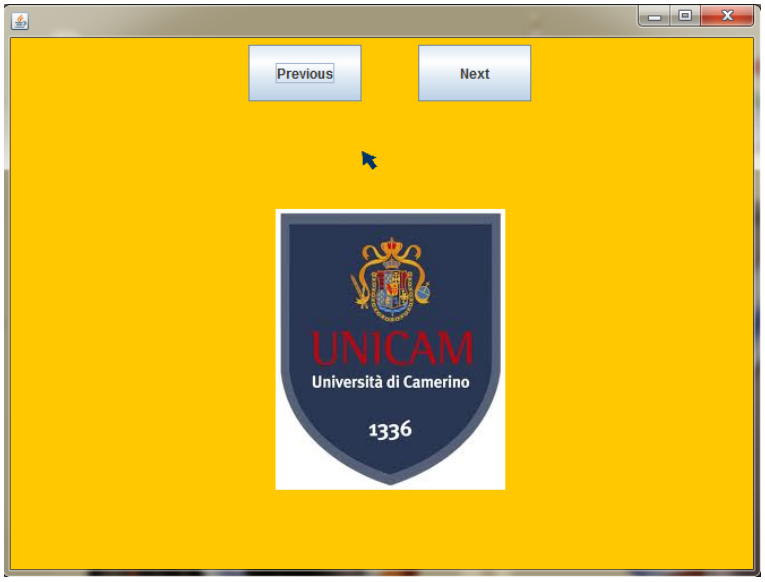

Figure 5.2: GUI with special pointer

5.2 results

We have tried the different versions in several computer. Both of them works with no problem. However, in the first case the performance of the system becames worse if we use an older pc (in a worse way than the second version one). We noticed a latency using the Robot class which has to use low lever mechanism to access the system mouse.
On the contrary the second version is more fluently, probably because it has not to access to the system mouse (via system calls) and it can be executed as a normal Java application. In effect, the second version is just a mover of a JPanel on the interface.

5.3 future works

We would like to improve this work adding new features. We report some of these ideas:

- adaptive color to improve the controller recognition. We already start to implement this technology using color histograms and bins distance.

- use openCV to access to a lot of features
- system porting to C++ to improve performances and use openCV

Chapter 6

Cost Estimate

In this chapter we present a cost estimate taking into account both productive and human resources.

6.1 Production costs

Resource	**Cost (Euro)**
red LED 5mm	0,20
Resistor (voltage adaption)	0,04
button switch	0,40
battery enclosure	0,60
2 x AA 1,5 V Battery	0,50
ping pong ball	0,20

Total: 1,94 euros.
To the table has to be added webcam and computer cost.

6.2 Human work costs

As we planned in the second chapter we report the real project costs.

- build the controller - 1h
- work-environment setup (webcam drivers, JRE, JDK, JMF) - 1h
- InterfaceAgent implementation and test - 9 days
  - PointDetector coding - 5 days
  - EventDetector coding - 4 days
- case study development - 2 days
- writing documentation - 5 days

Total: 16 days, 2 hours

Appendix A

Code

A.1 InterfaceAgent

A.1.1 InterfaceAgent

```
package InterfaceAgent;

import InterfaceAgent.EventManager.EventDetector;
import InterfaceAgent.PointDetector.BlobRecognizer;
import InterfaceAgent.PointDetector.CameraException;

public class InterfaceAgent {
    // *** Attributes ***
    private static EventDetector eventDetector;
    private static BlobRecognizer blobRecognizer;

    // *** Methods ***
    public InterfaceAgent() throws CameraException{
            eventDetector = new EventDetector();
            blobRecognizer = new BlobRecognizer(eventDetector);
    }
}
```

A.1.2 PointDetector

BlobRecognizer

```
package InterfaceAgent.PointDetector;

import InterfaceAgent.EventManager.EventDetector;
import java.awt.Color;
import java.awt.Toolkit;
import java.awt.image.BufferedImage;

public class BlobRecognizer extends Thread{
    // *** Attributes ***
    private CameraManager camera; // istanza webcam
    private EventDetector eventDetector;
    private boolean[][] visitedPixels; // punti visitati
    private static int height, width; // dimensione immagine
    private final int areaThreshold = 4; // soglia sull'area
```

```
    // *** Methods ***
    public BlobRecognizer(EventDetector ed) throws CameraException{
        // recupero l'istanza della USBCam
        camera = CameraManager.getIstance();
        eventDetector = ed;
        // avvio il thread di gestione
        this.start();
    }

    // thread di gestione
    @Override
    public void run(){
        while(true){
            Point point = null;
            long millisecond = System.currentTimeMillis();

            // ottengo l'immagine dalla webcam
            BufferedImage image = getImage();

            // processo l'immagine per ricavarne i blob
            point = process(image);

            if(point == null)    // indico l'assenza di punti all'event detector
                eventDetector.setPoint();
            else                // passo il punto all'event detector
                eventDetector.setPoint(point);

            millisecond = System.currentTimeMillis() - millisecond;
            // a questo punto vado in sleep per non monopolizzare la CPU
            try {
                if(millisecond < 30) Thread.sleep(30 - millisecond);
else  Thread.sleep(1);
            } catch(Exception e){
                System.out.println(e);
            }
        }
    }

    private BufferedImage getImage() {
        BufferedImage image = null;

        while(image == null){
            image = camera.getImage();      // recupero l'immagine
            width = camera.getWidth();      // recupero la larghezza
            height = camera.getHeight();    // recupero l'altezza
        }

        if(visitedPixels == null) visitedPixels = new boolean[width][height];
        // inizializzo la matrice dei pixel visitati a false
        for(int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
                visitedPixels[i][j] = false;

        // ritorno l'immagine
        return image;
    }


    private Point process(BufferedImage image){
        Point point = null;

        for(int y = 0; y < height && point == null; y++)
```

```
        for(int x = 0; x < width && point == null; x++) {
            // se il pixel è abilitato e non è ancora stato visitato
            if(isEnabled(image, x, y) && !visitedPixels[x][y]){
                // allora il pixel fa parte di un blob
                Blob blob = new Blob(width, height); // creo il blob
                findBlob(image, x, y, blob); // trovo gli altri pixel del blob

                // solo se l'area del blob supera quella minima lo considero
                if(blob.getArea() >= areaThreshold){
                    // adatta il baricentro alla risoluzione dello schermo
                    point = blob.getBarycenter();

                    // se il blob ha una forma circolare lo accetto
                    if(blob.isRound()){
                        // adatta la posizione del punto rispetto la risoluzione
                        // posX:relWidth=X:scrWidth => X=posX*scrWidth/relWidth
                        int scrX = (int) (point.getX() *
                        Toolkit.getDefaultToolkit().getScreenSize().width/width);
                        // posY:relHeight=Y:scrHeight => Y=posY*scrHeight/relHeight
                        int scrY = (int) (point.getY() *
                        Toolkit.getDefaultToolkit().getScreenSize().height/height);
                        point = new Point(scrX, scrY);
                    }else point = null;
                }
            }
        }
    // ritorna il baricentro del blob trovato, o null
    return point;
}

private boolean isEnabled(BufferedImage image, int x, int y){
    boolean esito = false;
    // recupera il valore del pixel per i canali RGB
    Color pixel = new Color(image.getRGB(x, y));

    if(pixel.getRed() > 200 &&
       pixel.getBlue() < 120 &&
       pixel.getGreen() < 120){
        esito = true;
    }

    return esito;
}

private void findBlob(BufferedImage image, int x, int y, Blob blob){
    // se il pixel è abilitato e non è stato ancora visitato non fa parte
    // di altre forme e devo visitarlo
    if(isEnabled(image, x, y) && !visitedPixels[x][y]) {
        visitedPixels[x][y] = true; // aggiunge il punto tra quelli visitati
        blob.tryAttachPixel(x, y);  // aggrega il pixel alla forma
        // esplora ricorsivamente i pixel vicini
        if(x + 1 < width) findBlob(image, x+1, y, blob);   // destra
        if(x > 0) findBlob(image, x-1, y, blob);           // sinistra
        if(y > 0) findBlob(image, x, y - 1, blob);         // sopra
        if(y + 1 < height) findBlob(image, x, y + 1, blob);// sotto;
    }
}

public int getWidth(){
    return width;
}
```

```
    public int getHeight(){
        return height;
    }
}
```

Blob

```
package InterfaceAgent.PointDetector;

public class Blob {
    // *** Attributes ***
    private boolean firstPixel;
    private boolean[][] blob;
    private int area;
    private double roundAbility;

    // *** Methods ***
    public Blob(int width, int height) {
        area = 0;
        roundAbility = 0;
        firstPixel = true;
        // Inizializza a false la matrice di ricerca della forma
        blob = new boolean[width][height];
        for(int i = 0; i < width; i++)
                for(int j = 0; j < height; j++)
                        blob[i][j] = false;
    }

    public boolean tryAttachPixel(int x, int y) {
        if(firstPixel || // se è il pixel da cui ho creato la forma
          (x > 0 && blob[x -1][y]) || // sinistra
          (x < blob.length - 1 && blob[x+1][y]) ||  // destra
          (y > 0 && blob[x][y-1]) ||                 // sopra
          (y < blob[x].length - 1 && blob[x][y+1])) // sotto
        {
            blob[x][y] = true;
            firstPixel = false;
            area++;
        }
        return blob[x][y];
    }

    public int getArea(){
        return area;
    }

    public boolean isRound(){
        return roundAbility > 0.5 && roundAbility < 2;
    }

    public Point getBarycenter() {
        int x = 0;
        int y = 0;
        int numPoint = 0;
        int maxX = 0, maxY = 0, minX = blob.length, minY = blob[0].length;

        // Sommo ascisse e ordinate e incrementa il numero di punti
        for(int i = 0; i < blob.length; i++)
            for(int j = 0; j < blob[i].length; j++)
                    if(blob[i][j]) {
                        if(i > maxX) maxX = i;
```

```
                        if(i < minX) minX = i;
                        if(j > maxY) maxY = j;
                        if(j < minY) minY = j;

                        x += i;
                        y += j;
                        numPoint++;
                    }

        // calcolo l'eccentricità
        roundAbility = (maxX - minX + 1) / (double) (maxY - minY + 1);

        // calcolo il baricentro
        x = x / numPoint;
        y = y / numPoint;

        return new Point(x,y);
    }
}
```

CameraException

```
package InterfaceAgent.PointDetector;

public class CameraException extends Exception{

    public CameraException() {

    }

    public CameraException(String messaggio){
        super(messaggio);
    }
}
```

CameraManager

```
package InterfaceAgent.PointDetector;

import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.media.CannotRealizeException;
import javax.media.Manager;
import javax.media.MediaLocator;
import javax.media.NoPlayerException;
import javax.media.Player;
import javax.media.control.FrameGrabbingControl;
import javax.media.format.VideoFormat;

public class CameraManager {
    // *** Attributes ***
    private static CameraManager istance;
    private Player player;
    private MediaLocator mediaLocator;
    private int width, height;
    private boolean firstShot;

    // *** Methods ***
    private CameraManager() throws IOException, NoPlayerException,
                                            CannotRealizeException {
        // inizializzo le variabili
```

```
        player = null;
        mediaLocator = null;
        width = 0;
        height = 0;
        firstShot = true;
        // inizializza il media locator sul bus della webcam
        mediaLocator = new MediaLocator("vfw://0");
        // creo il player
        player = Manager.createRealizedPlayer(mediaLocator);
        // avvio l'acquisizione dello stream
        player.start();

        // ciclo sullo stato della webcam in attesa che diventi disponibile
        while(player.getState() != player.Started){
            try {
                Thread.sleep(500);
            }
            catch(InterruptedException ex){
                System.out.println(ex);
            }
        }
    }

    public BufferedImage getImage(){
        // creo una buffered image
        //byte[] image = null;
        byte[][] imageArr = null;
        BufferedImage image = null;

        // catturo un immagine dal player
        FrameGrabbingControl fbc = (FrameGrabbingControl) player.getControl(
                                   "javax.media.control.FrameGrabbingControl");
        javax.media.Buffer buffer = fbc.grabFrame();

        // controllo che ci siano dati validi nel BufferImage acquisito
        if(buffer != null && buffer.getData() != null){
            // se ci sono dati nel buffer li controllo
            if(buffer.getData() instanceof byte[]){
                // se il formato dei dati è definito
                if(firstShot && buffer.getFormat() != null){
                    // allora ho catturato una immagine
                    // e ho il formato della foto che mi è utile
                    // a valorizzare gli Attributes height e width
                    VideoFormat format = (VideoFormat) buffer.getFormat();
                    width = format.getSize().width;
                    height = format.getSize().height;
                    firstShot = false;
                }
                byte[] rawImage = (byte []) buffer.getData();

                imageArr = new byte[3][height*width];
                for(int i = 0; i < height*width; i++){
                    imageArr[0][i] = rawImage[i * 3];
                    imageArr[1][i] = rawImage[i * 3 + 1];
                    imageArr[2][i] = rawImage[i * 3 + 2];
                }

                image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
                // l'immagine catturata dalla camera è un array che interlaccia
                // i 3 canali Red Green Blue quindi otteniamo il numero dei pixel
                // dividendo il numero di campioni per 3
                int PixelNumber = rawImage.length / 3;
```

```
                // creo un array di interi
                int[] rawRGB = new int[PixelNumber + 1];

                // ciclo sul numero di pixel dell'immagine
                for(int i = 0; i < PixelNumber; i++) {
                    // setto a 11111111 il canale alfa ovvero trasparenza assente
                    rawRGB[PixelNumber - i] = 0xFF000000 | // canale alfa a 11111111
                        ((rawImage[3 * i + 2] & 0xFF) << 16) |
                        ((rawImage[3 * i + 1] & 0xFF) << 8) |
                        ((rawImage[3 * i] & 0xFF));

                    // alfa - red - green - blue
                }
                // imposto l'immagine RGB come BufferedImage
                image.setRGB(0, 0, width, height, rawRGB, 0, width);
            }
        }
        return image;
    }

    public int getWidth(){
        return width;
    }

    public int getHeight(){
        return height;
    }

    public static CameraManager getIstance() throws CameraException {
        if(istance == null)
            try {
            istance = new CameraManager();
        } catch (IOException ex) {
            throw new CameraException("Error during webcam managing");
        } catch (NoPlayerException ex) {
            throw new CameraException("Error during the use of JMF");
        } catch (CannotRealizeException ex) {
            throw new CameraException("Error during webcam managing");
        }
        return istance;
    }
}
```

Point

```
package InterfaceAgent.PointDetector;

public class Point {
    // *** Attributes ***
    private int x, y;

    // *** Methods ***
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }
```

```
    public int getY() {
        return y;
    }
}
```

Mouse manager

```
package InterfaceAgent.EventManager;

import InterfaceAgent.PointDetector.Point;
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class EventDetector extends Thread {
    // *** Attributes ***
    private Point oldPosition; // posizione passata
    private Point newPosition; // posizione attuale
    private Semaphore pointPresence;
    private Lock positionAccess;
    private Robot robot;

    // *** Methods ***
    public EventDetector(){
        pointPresence = new Semaphore(0);
        positionAccess = new ReentrantLock();
        this.start(); // avvio il thread
    }

    @Override
    public void run(){
        boolean en = true;

        try {
            robot = new Robot();
        } catch (AWTException ex) {
            en = false;
        }

        while(en){
            try {
                // mi blocco in attesa che il blobrecognizer processi un frame
                pointPresence.acquire();
                // appena vengo sbloccato vado a vedere se ci sono o meno nuovi punti
                // accedendo ai punti in maniera esclusiva
                positionAccess.lock();

                // aggiorno la posizione del puntatore
                if(newPosition != null){
                    robot.mouseMove(newPosition.getX(), newPosition.getY());
                }

                // controllo la presenza di eventi click
                if(oldPosition != null && newPosition == null){
                    robot.mousePress(InputEvent.BUTTON1_MASK);
                    robot.mouseRelease(InputEvent.BUTTON1_MASK);
                }
```

```
                // la posizione attuale diviene vecchia
                oldPosition = newPosition;
                newPosition = null;

                // rilascio il lock sugli Attributes position
                positionAccess.unlock();

            } catch (InterruptedException ex) {
                // l'attesa del thread sul semaforo è stata interrotta
                // in quanto il thread che lo ha bloccato è stato terminato
                // in questo caso vengo sbloccato automaticamente e non mi resta
                // che terminare il thread
                en = false;
            }
        }
    }

    public void setPoint(){
        // non ci sono punti da processare nel frame corrente
        positionAccess.lock();
        newPosition = null;
        positionAccess.unlock();

        // avviso il detector rilasciando un permesso sul semaforo
        if(pointPresence.availablePermits() == 0) pointPresence.release();
    }

    public void setPoint(Point p){
        // ci sono punti da processare nel frame corrente
        positionAccess.lock();
        newPosition = p;
        positionAccess.unlock();

        // sveglio il detector eventualmente in attesa sul semaforo
        if(pointPresence.availablePermits() == 0) pointPresence.release();
    }
}
```

Home made Events

A.2 Image viewer

A.2.1 Mouse Manager version

Main

```
package Applications;

public class Main {
    public static void main(String args[]){
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Application().setVisible(true);
            }
        });
    }
}
```

Application

```
package Applications;

import InterfaceAgent.PointDetector.CameraException;
import java.awt.Color;
import java.awt.Container;
import java.awt.Toolkit;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Application extends JFrame{
    // *** Attributes ***
    private Controller controller;

    public ImagePanel imagePanel;
    public JButton next;
    public JButton previous;
    public JButton exit;

    // *** Methods ***
    public Application(){
        try {
            controller = new Controller(this);
            initComponents();
        } catch (CameraException ex) {
            JOptionPane.showMessageDialog(null, "Error! "+ex);
        }
    }

    private void initComponents(){
        next = new JButton("Next");
        previous = new JButton("Previous");
        exit = new JButton("Exit");
        imagePanel = new ImagePanel();

        int width, height;
        width = Toolkit.getDefaultToolkit().getScreenSize().width;
        height = Toolkit.getDefaultToolkit().getScreenSize().height;

        setSize(width, height);

        Container pane = this.getContentPane();
        pane.setLayout(null);
        pane.setBackground(Color.ORANGE);

        pane.add(next);
        pane.add(previous);
        pane.add(exit);
        pane.add(imagePanel);

        imagePanel.setSize(width - 200, height - 200);
        imagePanel.setLocation(100, 100);
        next.setLocation((width / 2) + 50, 5);
        next.setSize(100, 50);
        previous.setLocation((width / 2) - 100, 5);
        previous.setSize(100, 50);
        exit.setSize(100,50);
        exit.setLocation(width - 100, 5);

        Listener listener = new Listener(this, controller);
```

```
        addWindowListener(listener);

        next.addActionListener(listener);
        previous.addActionListener(listener);
        exit.addActionListener(listener);
        next.setVisible(true);
        previous.setVisible(true);
        exit.setVisible(true);
    }
}
```

ImagePanel

```
package Applications;

import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JPanel;

public class ImagePanel extends JPanel{
    // *** Attributes ***
    private Image image;

    // *** Methods ***
    public ImagePanel(){
        setOpaque(false);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        if(image != null){
            g.drawImage(image,
                        (getWidth() - image.getWidth(this)) / 2,
                        (getHeight() - image.getHeight(this)) / 2,
                        image.getWidth(this),
                        image.getHeight(this),
                        this);
        }else{
            g.drawString("NO Loaded Image", getWidth() / 2 , getHeight() / 2);
        }
    }

    public void refreshImage(Image image){
        this.image = image;
        repaint();
    }
}
```

Listener

```
package Applications;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class Listener implements WindowListener,
                                 ActionListener{
```

```
    // *** Attributes ***
    private Application app;
    private Controller controller;

    // *** Methods ***
    public Listener(Application app, Controller controller){
        this.app = app;
        this.controller = controller;
    }

    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}

    public void actionPerformed(ActionEvent e) {
        Object component = e.getSource();
        if(component == app.next) controller.next();
        else if(component == app.previous) controller.previous();
        else if(component == app.exit) System.exit(0);
    }
}
```

Controller

```
package Applications;

import InterfaceAgent.InterfaceAgent;
import InterfaceAgent.PointDetector.CameraException;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.Vector;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import javax.imageio.ImageIO;

public class Controller extends Thread{
    // *** Attributes ***
    private Application application;
    private int value;
    private Lock access;
    private Semaphore permit;
    private InterfaceAgent agent;
    private Vector<Image> images;

    // *** Methods ***
    public Controller(Application app) throws CameraException{
        application = app;
        value = 0;
        access = new ReentrantLock();
        permit = new Semaphore(0);
        agent = new InterfaceAgent();

        images = new Vector<Image>();
```

```
        File directory = new File("dist/images");

        if(directory.isDirectory() && directory.canRead()){
            // acquisisco le immagini da visualizzare
            String[] FileNames = directory.list();

            for(String FileName : FileNames ){
                if(FileName.endsWith(".jpg")){
                    try {
                        images.add(ImageIO.read(new File(directory, FileName)));
                    }
                    catch (IOException ex){
                        System.out.println(ex);
                    }
                }
            }
        }
        this.start();
    }

    @Override
    public void run(){
        while(true){
            try {
                permit.acquire();
            } catch (InterruptedException ex) {
                // il thread che mi ha bloccato è stato terminato
            }

            access.lock();
            if(!images.isEmpty()){
                if(value == images.size()) value = 0;
                application.imagePanel.refreshImage(images.get(value % images.size()));
            }
            access.unlock();
        }
    }

    public void next(){
        access.lock();
        value++;
        access.unlock();
        permit.release();
    }

    public void previous(){
        access.lock();
        if(value > 0) value--;
        else value = images.size() - 1;
        access.unlock();
        permit.release();
    }
}
```

A.2.2 Home-made events version

Main

```
package Applications;

public class Main {
```

```
    public static void main(String args[]){
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Application().setVisible(true);
            }
        });
    }
}
```

Application

```
package Applications;

import Graphics.InterfaceAgent;
import Graphics.OpenMotionButton;
import Graphics.Pointer;
import PointDetector.CameraException;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Image;
import java.awt.Toolkit;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JOptionPane;
import javax.swing.JWindow;

public class Application extends JWindow{
    // *** Attributi ***
    private Controller controller;

    public OpenMotionButton next;
    public OpenMotionButton previous;
    public OpenMotionButton exit;
    public ImagePanel imagePanel;

    public Pointer pointer;

    // *** Metodi ***
    public Application(){
        try {
            controller = new Controller(this);
            initComponents();
            pack();
        } catch (CameraException ex) {
            JOptionPane.showMessageDialog(null, "Error! "+ex);
        }
    }

    private void initComponents(){
        next = new OpenMotionButton("Next");
        previous = new OpenMotionButton("Previous");
        exit = new OpenMotionButton("Exit");
        pointer = new Pointer();
        imagePanel = new ImagePanel();

        int width, height;
        width = Toolkit.getDefaultToolkit().getScreenSize().width;
        height = Toolkit.getDefaultToolkit().getScreenSize().height;
```

```
        setMinimumSize(new Dimension(width, height));

        Container pane = this.getContentPane();
        pane.setLayout(null);
        pane.setBackground(Color.ORANGE);

        pane.add(pointer);
        pane.add(next);
        pane.add(previous);
        pane.add(exit);
        pane.add(imagePanel);

        imagePanel.setSize(width - 200, height - 200);
        imagePanel.setLocation(100, 100);
        next.setLocation((width / 2) + 50, height - 50);
        next.setSize(100, 50);
        previous.setLocation((width / 2) - 100, height - 50);
        previous.setSize(100, 50);
        exit.setSize(100,50);
        exit.setLocation(width - 100, 5);

        Listener listener = new Listener(this, controller);
        addWindowListener(listener);

        next.addActionListener(listener);
        next.addListener(listener);
        previous.addActionListener(listener);
        previous.addListener(listener);
        exit.addActionListener(listener);
        exit.addListener(listener);
        next.setVisible(true);
        previous.setVisible(true);
        exit.setVisible(true);

        try {
            // infine definisco il puntatore della applicazione
            Image pointerFace = ImageIO.read(new File("dist/", "pointer.png"));
            // se l'immagine viene trovata l'eccezione non è lanciata
            // in questo caso setto l'immagine come puntatore
            pointer.setPointer(pointerFace);
            pointer.setSize(width / 100, width / 100);
        } catch (IOException ex) {
            // altrimenti uso il puntatore standard (cerchio)
            pointer.setSize(width / 100, width / 100);
        }
        InterfaceAgent.setPointer(pointer);
        // abilito la visibilità del puntatore
        // il dispatcher si occupa solo di definire la posizione del puntatore
        // sta al programmatore scegliere se usare o meno questa funzionalità
        pointer.setVisible(true);
    }
}
```

Listener

```
package Applications;

import EventManager.Click;
import EventManager.OpenMotionListener;
import Graphics.Pointer;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Listener implements WindowListener,
                                 ActionListener,
                                 OpenMotionListener{
    // *** Attributi ***
    private Application app;
    private Controller controller;
    private Pointer pointer;
    private Lock pointerAccess;

    // *** Metodi ***
    public Listener(Application app, Controller controller){
        this.app = app;
        this.controller = controller;
        pointerAccess = new ReentrantLock();
    }

    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}

    public void actionPerformed(ActionEvent e) {
        Object component = e.getSource();
        if(component == app.next) controller.next();
        else if(component == app.previous) controller.previous();
        else if(component == app.exit) System.exit(0);
    }

    public void ClickAction(Click ev) {
        Object component = ev.getSource();
        if(component == app.next) controller.next();
        else if(component == app.previous) controller.previous();
        else if(component == app.exit) System.exit(0);
    }
}
```

Controller

```
package Applications;

import Graphics.InterfaceAgent;
import PointDetector.CameraException;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.Vector;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import javax.imageio.ImageIO;
```

```
public class Controller extends Thread{
    // *** Attributi ***
    private Application application;
    private int value;
    private Lock access;
    private Semaphore permit;
    private InterfaceAgent agent;
    private Vector<Image> images;

    // *** Metodi ***
    public Controller(Application app) throws CameraException{
        application = app;
        value = 0;
        access = new ReentrantLock();
        permit = new Semaphore(0);
        agent = new InterfaceAgent();

        images = new Vector<Image>();
        File directory = new File("dist/images");

        if(directory.isDirectory() && directory.canRead()){
            // acquisisco le immagini da visualizzare
            String[] FileNames = directory.list();

            for(String FileName : FileNames ){
                if(FileName.endsWith(".jpg")){
                    try {
                        images.add(ImageIO.read(new File(directory, FileName)));
                    }catch (IOException ex){
                        System.out.println(ex);
                    }
                }
            }
        }
        this.start();
    }

    @Override
    public void run(){
        while(true){
            try {
                permit.acquire();
            } catch (InterruptedException ex) {
                // il thread che mi ha bloccato è stato terminato
            }
            access.lock();
            if(!images.isEmpty()){
                if(value == images.size()) value = 0;
                application.imagePanel.refreshImage(images.get(value % images.size()));
            }
            access.unlock();
        }
    }

    public void next(){
        access.lock();
        value++;
        access.unlock();
        permit.release();
    }
```

```
    public void previous(){
        access.lock();
        if(value > 0) value--;
        else value = images.size() - 1;
        access.unlock();
        permit.release();
    }
}
```

ImagePanel

```
package Applications;

import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JPanel;

public class ImagePanel extends JPanel{
    // *** Attributi ***
    private Image image;

    // *** Metodi ***
    public ImagePanel(){
        setOpaque(false);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        if(image != null){
            g.drawImage(image,
                        (getWidth() - image.getWidth(this)) / 2,
                        (getHeight() - image.getHeight(this)) / 2,
                        image.getWidth(this),
                        image.getHeight(this),
                        this);
        }else{
            g.drawString("NO Loaded Image", getWidth() / 2 , getHeight() / 2);
        }
    }

    public void refreshImage(Image image){
        this.image = image;
        repaint();
    }
}
```

Event

```
package EventManager;

import java.awt.Component;

public abstract class Event extends PointerAction{
    // *** Attributi ***
    private Component source;
    // il riferimento al componente è utile sull'ascoltatore
    // per discriminare le diverse azioni da intraprendere
    // in occasione di eventi scatenati su componenti di tipo diverso
```

```
    // *** Metodi ***
    public void setSource(Component c){
        source = c;
    }
    public Component getSource(){
        return source;
    }
}
```

Click

```
package EventManager;

public class Click extends Event{
    // *** Attributi ***

    // *** Metodi ***
    public Click(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

PointerAction

```
package EventManager;

public abstract class PointerAction {
    // *** Attributi ***
    protected int x;
    protected int y;

    // *** Metodi ***
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }
}
```

Position

```
package EventManager;

public class Position extends PointerAction {
    // the class position is useful to refresh
    // the pointer position onto the window
    // *** Attributes ***

    // *** Methods ***
    public Position(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

OpenMotionListener

```
package EventManager;

public interface OpenMotionListener {
    public void ClickAction(Click ev);
}
```

EventDispatcher

```
package EventManager;

import Graphics.Pointer;
import java.awt.Component;
import java.awt.Point;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class EventDispatcher extends Thread {
    // *** Attributi ***
    private static EventDispatcher istance;
    private Vector<PointerAction> actions; // eventi in arrivo dall'event Detector
    private Hashtable<Component, OpenMotionListener> listeners;
    private Pointer pointer;
    private Lock eventsAccess;
    private Lock pointerAccess;
    private Lock listenersAccess;
    private Semaphore eventsPresence;

    // *** Metodi ***
    private EventDispatcher(){
        listeners = new Hashtable<Component, OpenMotionListener>();
        eventsAccess = new ReentrantLock();
        pointerAccess = new ReentrantLock();
        listenersAccess = new ReentrantLock();
        eventsPresence = new Semaphore(0);
        this.start();
    }

    @Override
    public void run(){
        while(true){
            try {
                // attendo che mi siano notificati degli eventi dall'eventDetector
                eventsPresence.acquire();
                // accedo in maniera esclusiva al vettore degli eventi
                eventsAccess.lock();

                pointerAccess.lock();

                for (int i = 0; i < actions.size(); i++) {
                    // recupero l'azione
                    PointerAction action = actions.get(i);

                    if(action instanceof Position){
                        pointer.setLocation(action.getX(), action.getY());
                    }else{
                        // ho un evento da risolvere
```

```
                    Event event = (Event) action;
                    // recupero il componente su cui è stato scatenato l'evento
                    Component target = getComponent(event.getX(), event.getY());
                    // se l'evento è stato scatenato sopra un componente di top level
                    // allora invoco l'azione su quel componente
                    if(target != null){
                        // assegno il componente all'evento
                        event.setSource(target);
                        // notifico l'evento sul listener del componente
                        if (action instanceof Click) {
                            listeners.get(target).ClickAction((Click) action);
                        }
                    }
                }

            }

            // pulisco il vettore avendo gia processato tutti gli eventi presenti
            actions.clear();

            // rilascio il permesso sul puntatore
            pointerAccess.unlock();

            // rilascio il permesso sul vettore degli eventi
            eventsAccess.unlock();

        } catch (InterruptedException ex) {
            // l'attesa sul semaforo è stata interrotta in quanto il thread
            // che lo ha bloccato è stato terminato
        }
    }
}

public static EventDispatcher getIstance(){
    if(istance == null)
        istance = new EventDispatcher();

    return istance;
}

// utilizzato dall'eventDetector per aggiungere eventi
public void addActions(Vector<PointerAction> actions){
    // accedo in maniera esclusiva al vettore degli eventi
    eventsAccess.lock();
    this.actions = actions;
    // rilascio il permesso esclusivo e incremento il numero di permessi
    eventsAccess.unlock();
    if(eventsPresence.availablePermits() == 0) eventsPresence.release();
}

// interfaccia con l'applicazione
public void addListener(Component c, OpenMotionListener l){
    listenersAccess.lock();
    listeners.put(c, l);
    listenersAccess.unlock();
}
public void removeListener(Component c){
    listenersAccess.lock();
    listeners.remove(c);
    listenersAccess.unlock();
}
public void setPointer(Pointer p){
```

```
        pointerAccess.lock();
        pointer = p;
        pointerAccess.unlock();
    }

    // --- gestione della posizione dei componenti ---
    private Component getComponent(int x, int y){
        // scorro l'hashTable per trovare il componente nella posizione
        // dell'evento e con la profondità più alta (componente top level)
        Component topLevelComponent = null;

        // si parte dalla profondità zero e via via si aumenta mano a mano che
        // si trovano componenti con profondità più alta
        int depth = 0;

        for(Enumeration e = listeners.keys(); e.hasMoreElements(); ){
            Component c = (Component) e.nextElement();
            // controllo che il componente sia visibile
            if(c.isVisible()){
                // controllo che il componente sia più profondo della profondità precedente
                if(depth(c) >= depth){
                    // controllo che il componente sia nella posizione dell'evento
                    Point p = AbsolutePosition(c);
                    if(x >= p.getX() && x <= (p.getX() + c.getWidth()) &&
                       y >= p.getY() && y <= (p.getY() + c.getHeight())){
                        // confermo il componente come di toplevel tra quelli
                        // visitati fino a questo momento nell'hashtable
                        topLevelComponent = c;
                        depth = depth(c);
                    }
                }
            }
        }
        // ritorno il componente di top-level
        return topLevelComponent;
    }

    private int depth(Component c){
        int depth = 1;

        if(c.getParent() == null) // caso base, il componente è la radice
            depth = 1;
        else // caso induttivo, il componente è un nodo o una foglia
            depth += depth(c.getParent());

        return depth;
    }

    private Point AbsolutePosition(Component c){
        Point position;

        if(c.getParent() == null) // caso base, il componente è la radice
            position = new Point(c.getX(), c.getY());
        else{ // caso induttivo, il componente è un nodo o una foglia
            // la posizione del componente è calcolata rispetto al padre che lo
            // contiene per cui si agisce ricorsivamente per trovare la posizione assoluta
            Point parentPosition = AbsolutePosition(c.getParent());
            position = new Point(c.getX()+parentPosition.x, c.getY()+parentPosition.y);
        }
        return position;
    }
}
```

EventDetector

```
package EventManager;

import PointDetector.BlobRecognizer;
import PointDetector.CameraException;
import PointDetector.Point;
import java.util.Vector;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class EventDetector extends Thread {
    // *** Attributi ***
    private BlobRecognizer blobRecognizer;
    // istanza della classe
    private static EventDetector istance;
    private Point oldPosition; // posizione passata
    private Point newPosition; // posizione attuale
    private Semaphore pointPresence;
    private Lock positionAccess;
    private EventDispatcher evDispatcher; // riferimento al dispatcher

    // *** Metodi ***
    private EventDetector(){
        pointPresence = new Semaphore(0);
        positionAccess = new ReentrantLock();
    }

    public static EventDetector getIstance() throws CameraException{
         if(istance == null) istance = new EventDetector();
         return istance;
    }

    @Override
    public void run(){
        boolean en = true;
        Vector<PointerAction> actions = new Vector<PointerAction>();

        while(en){

            try {
                // mi blocco in attesa che il blobrecognizer processi un frame
                pointPresence.acquire();

                // appena vengo sbloccato vado a vedere se ci sono o meno nuovi punti
                // accedendo ai punti in maniera esclusiva
                positionAccess.lock();
                // aggiorno la posizione del puntatore
                if(newPosition != null){
                    actions.add(new Position(newPosition.getX(), newPosition.getY()));
                }

                // controllo la presenza di eventi click
                if(oldPosition != null && newPosition == null){
                    actions.add(new Click(oldPosition.getX(), oldPosition.getY()));
                }

                // la posizione attuale diviene vecchia
                oldPosition = newPosition;
                newPosition = null;
                // rilascio il lock sugli attributi position
```

```
                positionAccess.unlock();

                evDispatcher.addActions(actions);
            } catch (InterruptedException ex) {
                // l'attesa del thread sul semaforo è stata interrotta
                // in quanto il thread che lo ha bloccato è stato terminato
                // in questo caso vengo sbloccato automaticamente e non mi resta
                // che terminare il thread
                en = false;
            }
        }
    }

    public void setDependencies() throws CameraException{
        // recupero il riferimento all'eventDispatcher
        evDispatcher = EventDispatcher.getIstance();

        // recupero il riferimento al BlobRecognizer
        blobRecognizer = BlobRecognizer.getIstance();
        this.start(); // avvio il thread
    }

    public void setPoint(){
        // non ci sono punti da processare nel frame corrente
        positionAccess.lock();
        newPosition = null;
        positionAccess.unlock();

        // avviso il detector rilasciando un permesso sul semaforo
        if(pointPresence.availablePermits() == 0) pointPresence.release();
    }

    public void setPoint(Point p){
        // ci sono punti da processare nel frame corrente
        positionAccess.lock();
        newPosition = p;
        positionAccess.unlock();

        // sveglio il detector eventualmente in attesa sul semaforo
        if(pointPresence.availablePermits() == 0) pointPresence.release();
    }
}
```

InterfaceAgent

```
package Graphics;
import EventManager.EventDetector;
import EventManager.EventDispatcher;
import EventManager.OpenMotionListener;
import PointDetector.BlobRecognizer;
import PointDetector.CameraException;
import java.awt.Component;

public class InterfaceAgent {
    // *** Attributi ***
    private static EventDispatcher eventDispatcher;
    private static EventDetector eventDetector;
    private static BlobRecognizer blobRecognizer;

    // *** Metodi ***
    public InterfaceAgent() throws CameraException{
```

```
            eventDispatcher = EventDispatcher.getIstance();
            eventDetector = EventDetector.getIstance();
            blobRecognizer = BlobRecognizer.getIstance();

            // connetto i componenti istanziati
            eventDetector.setDependencies();
            blobRecognizer.setDependencies();
    }

    // la comunicazione con l'agente coinvolge il solo eventDispatcher
    public static void addListener(Component c, OpenMotionListener l){
        eventDispatcher.addListener(c, l);
    }

    public static void removeListener(Component c){
        eventDispatcher.removeListener(c);
    }

    public static void setPointer(Pointer p){
        eventDispatcher.setPointer(p);
    }
}
```

OpenMotionComponent

```
package Graphics;

import EventManager.OpenMotionListener;

public interface OpenMotionComponent {
    public void addListener(OpenMotionListener listener);
    public void removeListener();
}
```

OpenMotionButton

```
package Graphics;

import EventManager.OpenMotionListener;
import javax.swing.JButton;

public class OpenMotionButton extends JButton implements OpenMotionComponent{

    public OpenMotionButton(){}

    public OpenMotionButton(String name) {
        super(name);
    }

    public void addListener(OpenMotionListener listener) {
        InterfaceAgent.addListener(this, listener);
    }

    public void removeListener() {
        InterfaceAgent.removeListener(this);
    }
}
```

OpenMotionTextField

```
package Graphics;

import EventManager.OpenMotionListener;
import javax.swing.JTextField;

public class OpenMotionTextField extends JTextField implements OpenMotionComponent{

    public OpenMotionTextField(){}

    public OpenMotionTextField(String name) {
        super(name);
    }

    public void addListener(OpenMotionListener listener) {
        InterfaceAgent.addListener(this, listener);
    }

    public void removeListener() {
        InterfaceAgent.removeListener(this);
    }
}
```

Pointer

```
package Graphics;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JPanel;

public class Pointer extends JPanel{
    // *** Attributi ***
    private Image pointer;

    // *** Metodi ***
    public Pointer(){
        setVisible(false);
        setOpaque(false);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        if(pointer != null){
            g.drawImage(pointer, 0, 0, getWidth(), getHeight(), this);
        }else{
            g.setColor(Color.blue);
            g.drawOval(0, 0, getWidth(), getHeight());
            g.fillOval(0, 0, getWidth(), getHeight());
        }
    }

    public void setPointer(Image image){
        pointer = image;
    }
}
```