Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

з курсу «Автоматизація проектування комп'ютерних систем»

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

Тема роботи

Редактор блок-схем алгоритмів.

Мета роботи

Здобуття навичок з побудови редактора блок-схем алгоритмів. Розробка інтерфейсу користувача та функціонального наповнення. Розробка засобів перетворення форматів зберігання даних.

Завдання на роботу

Номер залікової книжки: $7308_{10} = 1110010001100_2$.

Тип редактора: $n_1 = 0$ => Редактор графічних схем алгоритмів (ГСА).

Тип формату: $n_2 = 0$ => Текстовий.

Реалізувати редактор алгоритму заданого типу з можливістю збереження/відновлення результатів роботи програми у матричному вигляді згідно розробленого формату. Передбачити в редакторі наступні функції:

- Створення нової блок-схеми алгоритму;
- Модифікація алгоритму (створення/видалення початкового, кінцевого, логічних(X) та операційних(Y) вузлів, редагування сигналів вузлів, створення/видалення зв'язків між вузлами);
- Контроль вводу (тільки один початковий та кінцевий вузли, логічні та операційні вузли мають містити не менш одного сигналу, логічні вузли містять тільки вхідні сигнали, а операційні – тільки вихідні тощо);
- Збереження алгоритму у матричному вигляді;
- Відновлення алгоритму з матричного вигляду.

Опис програми

Дана програма призначена для створення і редагування графічних блок-схем алгоритмів (ГСА). Головне вікно програми зображено на рисунку 1. За допомогою панелі, що знаходиться в лівій частині вікна, можна вибрати дію, яку необхідно зробити з блок-схемою. Більша частина вікна призначена для роботи з блок-схемою. Зберегти, створити, відкрити, закрити блок-схему можна за допомогою меню File головного меню.

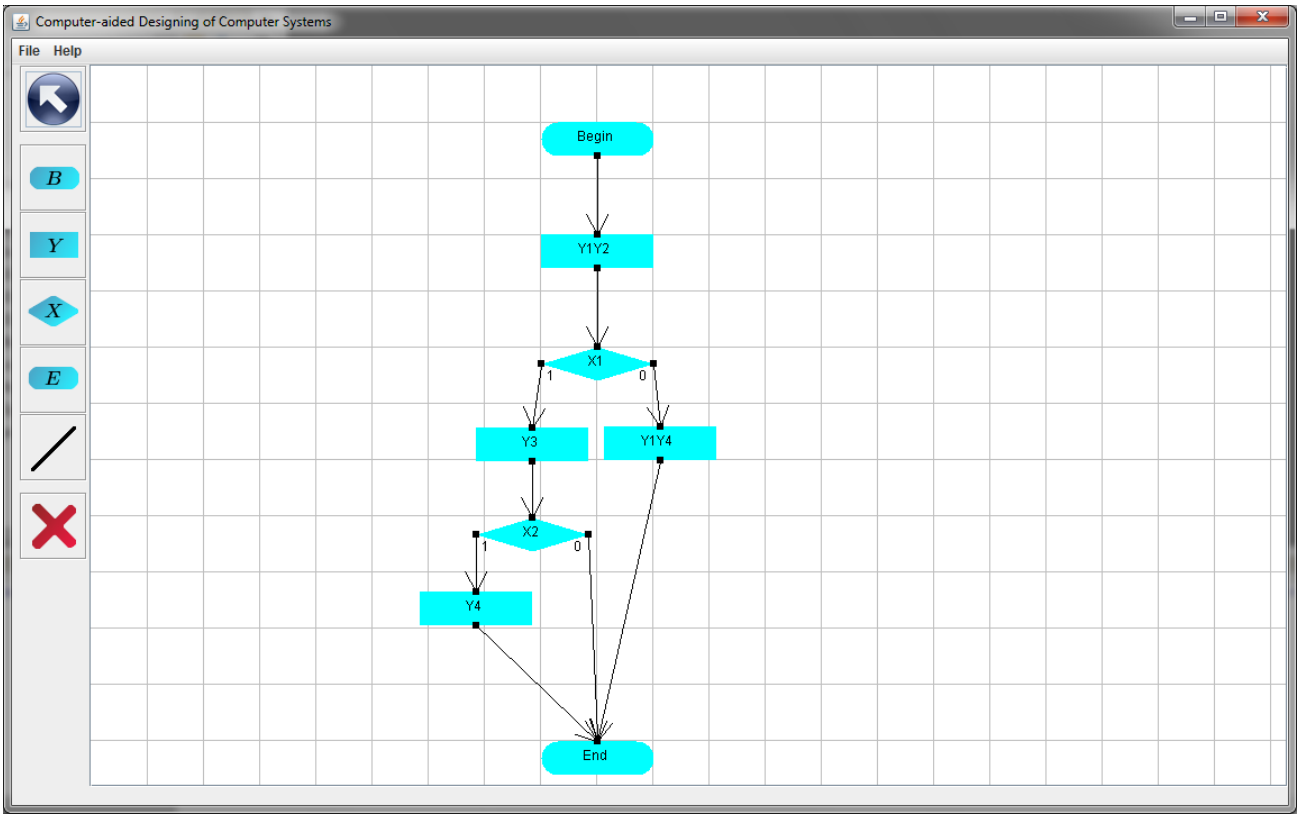

Рисунок 1. Головне вікно програми

Лістинг програми

```
package gsa;

import java.awt.*;
import java.util.LinkedList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:08:19
 * To change this template use File | Settings | File Templates.
 */
abstract class Node {

    protected int x;
    protected int y;
    protected int width;
    protected int height;
    protected Color color;

    public abstract LinkedList<Node> getParents();

    public abstract LinkedList<Node> getChildren();

    public abstract String getText();

    public abstract int[] getSignals();

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getWidth() {
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public Color getColor() {
        return color;
    }

    public void setColor(Color color) {
        this.color = color;
    }

}

package gsa;

import java.awt.*;
import java.util.LinkedList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:22:13
 * To change this template use File | Settings | File Templates.
 */
class BeginNode extends Node {

    private static String TEXT = "Begin";

    private Node child;

    public BeginNode(int x, int y, int width, int height, Color color) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
    }

    @Override
    public LinkedList<Node> getParents() {
        return null;
    }

    @Override
    public LinkedList<Node> getChildren() {
        LinkedList<Node> list = new LinkedList<Node>();
        list.add(child);
        return list;
    }
```

```
    @Override
    public String getText() {
        return TEXT;
    }

    @Override
    public int[] getSignals() {
        int[] result = new int[1];
        result[0] = 0;
        return result;
    }

    public void setChildNode(Node node) {
        child = node;
    }

    public void removeChildNode() {
        child = null;
    }

}

package gsa;

import java.awt.*;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:20:44
 * To change this template use File | Settings | File Templates.
 */
class OperatorNode extends Node {

    static String TEXT = "Y";

    private int[] numbers;
    private LinkedList<Node> parents;
    private Node child;

    public OperatorNode(int x, int y, int width, int height, Color color, int[] numbers) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
        this.numbers = numbers;
        parents = new LinkedList<Node>();
    }

    public void addParentNode(Node node) {
        parents.add(node);
    }

    public void removeParentNode(Node node) {
        ListIterator<Node> iterator = parents.listIterator();
        boolean found = false;
        while ((iterator.hasNext()) && (!found)) {
            if (iterator.next() == node) {
                iterator.remove();
            }
        }
    }

    public void removeParentNodes() {
        parents = new LinkedList<Node>();
    }

    public void setChildNode(Node node) {
        child = node;
    }

    public void removeChildNode() {
        child = null;
    }

    @Override
    public LinkedList<Node> getParents() {
        return parents;
    }

    @Override
    public LinkedList<Node> getChildren() {
        LinkedList<Node> list = new LinkedList<Node>();
        list.add(child);
        return list;
    }

    @Override
    public String getText() {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < numbers.length; i++) {
            builder.append(TEXT);
            builder.append(String.valueOf(numbers[i]));
        }
        return builder.toString();
    }

    @Override
    public int[] getSignals() {
        return numbers;
    }

    public void setSignals(int[] numbers) {
        this.numbers = numbers;
```

```
    }
}

package gsa;

import java.awt.*;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:20:23
 * To change this template use File | Settings | File Templates.
 */
class LogicNode extends Node {

    public static String YES_TEXT = "1";
    public static String NO_TEXT = "0";

    private static String TEXT = "X";

    private int number;
    private LinkedList<Node> parents;
    private Node yesChild;
    private Node noChild;

    public LogicNode(int x, int y, int width, int height, Color color, int number) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
        this.number = number;
        parents = new LinkedList<Node>();
    }

    public void addParentNode(Node node) {
        parents.add(node);
    }

    public void removeParentNodes() {
        parents = new LinkedList<Node>();
    }

    public void removeParentNode(Node node) {
        ListIterator<Node> iterator = parents.listIterator();
        boolean found = false;
        while ((iterator.hasNext()) && (!found)) {
            if (iterator.next() == node) {
                iterator.remove();
            }
        }
    }

    public void setYesChildNode(Node node) {
        yesChild = node;
    }

    public void removeYesChildNode() {
        yesChild = null;
    }

    public void setNoChildNode(Node node) {
        noChild = node;
    }

    public void removeNoChildNode() {
        noChild = null;
    }

    @Override
    public LinkedList<Node> getParents() {
        return parents;
    }

    @Override
    public LinkedList<Node> getChildren() {
        LinkedList<Node> list = new LinkedList<Node>();
        list.add(yesChild);
        list.add(noChild);
        return list;
    }

    @Override
    public String getText() {
        return (TEXT + String.valueOf(number));
    }

    @Override
    public int[] getSignals() {
        int[] result = new int[1];
        result[0] = number;
        return result;
    }

    public void setSignal(int number) {
        this.number = number;
    }

}

package gsa;

import java.awt.*;
import java.util.LinkedList;
import java.util.ListIterator;
```

```
/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:22:44
 * To change this template use File | Settings | File Templates.
 */
class EndNode extends Node {

    private static String TEXT = "End";

    private LinkedList<Node> parents;

    public EndNode(int x, int y, int width, int height, Color color) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.color = color;
        parents = new LinkedList<Node>();
    }

    @Override
    public LinkedList<Node> getParents() {
        return parents;
    }

    @Override
    public LinkedList<Node> getChildren() {
        return null;
    }

    @Override
    public String getText() {
        return TEXT;
    }

    @Override
    public int[] getSignals() {
        int[] result = new int[1];
        result[0] = 0;
        return result;
    }

    public void addParentNode(Node node) {
        parents.add(node);
    }

    public void removeParentNodes() {
        parents = new LinkedList<Node>();
    }

    public void removeParentNode(Node node) {
        ListIterator<Node> iterator = parents.listIterator();
        boolean found = false;
        while ((iterator.hasNext()) && (!found)) {
            if (iterator.next() == node) {
                iterator.remove();
            }
        }
    }

}

package gsa;

import java.awt.*;
import java.util.ArrayList;
import java.util.LinkedList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 14:00:30
 * To change this template use File | Settings | File Templates.
 */
public class GSAModel {

    public static int DEFAULT_NODE_WIDTH = 100;
    public static int DEFAULT_NODE_HEIGHT = 30;
    public static int DEFAULT_ARROW_WIDTH = 20;
    public static double DEFAULT_ARROW_ANGLE = 0.5;
    public static int DEFAULT_CONNECTORS_SIZE = 6;

    private static Color DEFAULT_BACKGROUND_COLOR = Color.WHITE;
    private static Color DEFAULT_NET_COLOR = Color.LIGHT_GRAY;
    private static Color DEFAULT_NODES_COLOR = Color.CYAN;
    private static Color DEFAULT_LINES_COLOR = Color.BLACK;

    private static int DEFAULT_NET_INTERVAL = 50;

    private ArrayList<Node> nodes;
    private boolean hasBeginNode;
    private boolean hasEndNode;
    private Color backgroundColor;
    private Color netColor;
    private Color nodesColor;
    private Color linesColor;
    private int netInterval;
    private boolean isChanged;
    private int actionType;
    private boolean yesNoExit;

    public GSAModel() {
        nodes = new ArrayList<Node>();
        hasBeginNode = false;
        hasEndNode = false;
        backgroundColor = DEFAULT_BACKGROUND_COLOR;
```

```
    netColor = DEFAULT_NET_COLOR;
    nodesColor = DEFAULT_NODES_COLOR;
    netInterval = DEFAULT_NET_INTERVAL;
    linesColor = DEFAULT_LINES_COLOR;
    isChanged = false;
    actionType = 0;
}

public Color getLinesColor() {
    return linesColor;
}

public void setLinesColor(Color linesColor) {
    this.linesColor = linesColor;
}

public int getNetInterval() {
    return netInterval;
}

public void setNetInterval(int netInterval) {
    this.netInterval = netInterval;
}

public int getActionType() {
    return actionType;
}

public void setActionType(int actionType) {
    this.actionType = actionType;
}

public boolean isChanged() {
    return isChanged;
}

public void setChanged(boolean changed) {
    isChanged = changed;
}

public void addBeginNode(int x, int y, int width, int height) throws IllegalNodeException {
    if (!hasBeginNode) {
        nodes.add(new BeginNode(x, y, width, height, nodesColor));
        hasBeginNode = true;
    }
    else {
        throw new IllegalNodeException(true);
    }
}

public void addEndNode(int x, int y, int width, int height) throws IllegalNodeException {
    if (!hasEndNode) {
        nodes.add(new EndNode(x, y, width, height, nodesColor));
        hasEndNode = true;
    }
    else {
        throw new IllegalNodeException(false);
    }
}

public void addLogicNode(int x, int y, int width, int height, int signalNumber) {
    nodes.add(new LogicNode(x, y, width, height, nodesColor, signalNumber));
}

public void addOperatorNode(int x, int y, int width, int height, int[] signalNumbers) {
    nodes.add(new OperatorNode(x, y, width, height, nodesColor, signalNumbers));
}

public void removeNode(Node node) {
    if (node.getClass().getName().contains("BeginNode")) {
        hasBeginNode = false;
    }
    else {
        if (node.getClass().getName().contains("EndNode")) {
            hasEndNode = false;
        }
    }
    nodes.remove(node);
}

public boolean isInNode(int x, int y) {
    boolean isIn = false;
    int i = 0;
    while ((i < nodes.size()) && (!isIn)) {
        int nodex1 = nodes.get(i).getX();
        int nodex2 = nodes.get(i).getX() + nodes.get(i).getWidth();
        int nodey1 = nodes.get(i).getY();
        int nodey2 = nodes.get(i).getY() + nodes.get(i).getHeight();
        if ((x >= nodex1) && (x <= nodex2) && (y >= nodey1) && (y <= nodey2)) {
            isIn = true;
        }
        i++;
    }
    return isIn;
}

public Node getInNode(int x, int y) {
    Node inNode = null;
    int i = 0;
    while ((i < nodes.size()) && (inNode == null)) {
        int nodex1 = nodes.get(i).getX();
        int nodex2 = nodes.get(i).getX() + nodes.get(i).getWidth();
        int nodey1 = nodes.get(i).getY();
        int nodey2 = nodes.get(i).getY() + nodes.get(i).getHeight();
        if ((x >= nodex1) && (x <= nodex2) && (y >= nodey1) && (y <= nodey2)) {
            inNode = nodes.get(i);
        }
        i++;
    }
```

```
    return inNode;
}

public Node getNodeInEntrance(int x, int y) {
    for (Node e : nodes) {
        if (!e.getClass().getName().contains("BeginNode")) {
            Rectangle connector = new Rectangle(e.getX() + (e.getWidth() - DEFAULT_CONNECTORS_SIZE) / 2,
                    e.getY() - DEFAULT_CONNECTORS_SIZE / 2, DEFAULT_CONNECTORS_SIZE, DEFAULT_CONNECTORS_SIZE);
            if (connector.contains(x, y)) {
                return e;
            }
        }
    }
    return null;
}

public Node getNotLogicNodeInExit(int x, int y) {
    for (Node e : nodes) {
        if ((!e.getClass().getName().contains("EndNode")) && (!e.getClass().getName().contains("LogicNode"))) {
            Rectangle connector = new Rectangle(e.getX() + (e.getWidth() - DEFAULT_CONNECTORS_SIZE) / 2,
                    e.getY() + e.getHeight() - DEFAULT_CONNECTORS_SIZE / 2, DEFAULT_CONNECTORS_SIZE, DEFAULT_CONNECTORS_SIZE);
            if (connector.contains(x, y)) {
                return e;
            }
        }
    }
    return null;
}

public Node getLogicNodeInExit(int x, int y) {
    for (Node e : nodes) {
        if (e.getClass().getName().contains("LogicNode")) {
            Rectangle yesConnector = new Rectangle(e.getX()- DEFAULT_CONNECTORS_SIZE / 2,
                    e.getY() + (e.getHeight() - DEFAULT_CONNECTORS_SIZE) / 2, DEFAULT_CONNECTORS_SIZE, DEFAULT_CONNECTORS_SIZE);
            if (yesConnector.contains(x, y)) {
                yesNoExit = true;
                return e;
            }
            Rectangle noConnector = new Rectangle(e.getX() + e.getWidth() - DEFAULT_CONNECTORS_SIZE / 2,
                    e.getY() + (e.getHeight() - DEFAULT_CONNECTORS_SIZE) / 2, DEFAULT_CONNECTORS_SIZE, DEFAULT_CONNECTORS_SIZE);
            if (noConnector.contains(x, y)) {
                yesNoExit = false;
                return e;
            }
        }
    }
    return null;
}

public boolean isYesNoExit() {
    return yesNoExit;
}

public Color getNodesColor() {
    return nodesColor;
}

public Color getNetColor() {
    return netColor;
}

public Color getBackgroundColor() {
    return backgroundColor;
}

public ArrayList<Node> getNodes() {
    return nodes;
}

public int[][] getConnectionMatrix() {
    int[][] matrix = new int[nodes.size()][];
    for (int i = 0; i < matrix.length; i++) {
        matrix[i] = new int[nodes.size()];
        for (int j = 0; j < matrix[i].length; j++) {
            matrix[i][j] = 0;
        }
    }
    for (int i = 0; i < nodes.size(); i++) {
        LinkedList<Node> children = nodes.get(i).getChildren();
        if (children != null) {
            for (int j = 0; j < children.size(); j++) {
                int k = 0;
                boolean found = false;
                while ((k < nodes.size()) && (!found)) {
                    if (children.get(j) == nodes.get(k)) {
                        matrix[i][k] = j + 1;
                        found = true;
                    }
                    k++;
                }
            }
        }
    }
    return matrix;
}

public int[] getNodesType() {
    int[] matrix = new int[nodes.size()];
    for (int i = 0; i < nodes.size(); i++) {
        if (nodes.get(i).getClass().getName().contains("BeginNode")) {
            matrix[i] = 0;
        }
        else {
            if (nodes.get(i).getClass().getName().contains("OperatorNode")) {
                matrix[i] = 1;
            }
            else {
                if (nodes.get(i).getClass().getName().contains("LogicNode")) {
                    matrix[i] = 2;
```

```
                }
                else {
                    if (nodes.get(i).getClass().getName().contains("EndNode")) {
                        matrix[i] = 3;
                    }
                }
            }
        }
    }
    return matrix;
}

public int[][] getBoundsMatrix() {
    int[][] matrix = new int[nodes.size()][];
    for (int i = 0; i < matrix.length; i++) {
        matrix[i] = new int[4];
    }
    for (int i = 0; i < nodes.size(); i++) {
        matrix[i][0] = nodes.get(i).getX();
        matrix[i][1] = nodes.get(i).getY();
        matrix[i][2] = nodes.get(i).getWidth();
        matrix[i][3] = nodes.get(i).getHeight();
    }
    return matrix;
}

public int[][] getSignalMatrix() {
    int[][] matrix = new int[nodes.size()][];
    for (int i = 0; i < nodes.size(); i++) {
        matrix[i] = nodes.get(i).getSignals();
    }
    return matrix;
}

public void setGSA(int[] nodesType, int[][] connectivityMatrix, int[][] signalMatrix,
                   int[][] boundsMatrix) throws IllegalNodeException {
    hasBeginNode = false;
    hasEndNode = false;
    nodes = new ArrayList<Node>();
    for (int i = 0; i < nodesType.length; i++) {
        if (nodesType[i] == 0) {
            addBeginNode(boundsMatrix[i][0], boundsMatrix[i][1], boundsMatrix[i][2],
                    boundsMatrix[i][3]);
        }
        else {
            if (nodesType[i] == 1) {
                addOperatorNode(boundsMatrix[i][0], boundsMatrix[i][1], boundsMatrix[i][2],
                        boundsMatrix[i][3], signalMatrix[i]);
            }
            else {
                if (nodesType[i] == 2) {
                    addLogicNode(boundsMatrix[i][0], boundsMatrix[i][1], boundsMatrix[i][2],
                            boundsMatrix[i][3], signalMatrix[i][0]);
                }
                else {
                    addEndNode(boundsMatrix[i][0], boundsMatrix[i][1], boundsMatrix[i][2],
                            boundsMatrix[i][3]);
                }
            }
        }
    }
    for (int i = 0; i < nodes.size(); i++) {
        if (nodes.get(i).getClass().getName().contains("BeginNode")) {
            int j = 0;
            boolean found = false;
            BeginNode node = (BeginNode) nodes.get(i);
            while ((j < connectivityMatrix[i].length) && (!found)) {
                if (connectivityMatrix[i][j] == 1) {
                    node.setChildNode(nodes.get(j));
                    found = true;
                }
                j++;
            }
        }
        else {
            if (nodes.get(i).getClass().getName().contains("OperatorNode")) {
                int j = 0;
                boolean foundChild = false;
                OperatorNode node = (OperatorNode) nodes.get(i);
                while ((j < connectivityMatrix[i].length) && (!foundChild)) {
                    if (connectivityMatrix[j][i] > 0) {
                        node.addParentNode(nodes.get(j));
                    }
                    if (connectivityMatrix[i][j] == 1) {
                        node.setChildNode(nodes.get(j));
                        foundChild = true;
                    }
                    j++;
                }
            }
            else {
                if (nodes.get(i).getClass().getName().contains("LogicNode")) {
                    int j = 0;
                    int foundChildren = 0;
                    LogicNode node = (LogicNode) nodes.get(i);
                    while ((j < connectivityMatrix[i].length) && (foundChildren < 2)) {
                        if (connectivityMatrix[j][i] > 0) {
                            node.addParentNode(nodes.get(j));
                        }
                        if (connectivityMatrix[i][j] == 1) {
                            node.setYesChildNode(nodes.get(j));
                            foundChildren++;
                        }
                        if (connectivityMatrix[i][j] == 2) {
                            node.setNoChildNode(nodes.get(j));
                            foundChildren++;
                        }
                        j++;
                    }
```

```
                    }
                    else {
                        EndNode node = (EndNode) nodes.get(i);
                        for (int j = 0; j < connectivityMatrix[i].length; j++) {
                            if (connectivityMatrix[i][j] > 0) {
                                node.addParentNode(nodes.get(j));
                            }
                        }
                    }
                }
            }
        }
    }

}

package gsa;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 09.09.2010
 * Time: 22:15:18
 * To change this template use File | Settings | File Templates.
 */
public class IllegalNodeException extends Exception {

    private static String TEXT1 = "The ";
    private static String TEXT2 = " node has already exist.";
    private static String BEGIN_TEXT = "Begin";
    private static String END_TEXT = "End";

    private String text;

    public IllegalNodeException(boolean beginNode) {
        if (beginNode) {
            text = TEXT1 + BEGIN_TEXT + TEXT2;
        }
        else {
            text = TEXT1 + END_TEXT + TEXT2;
        }
    }

    @Override
    public String getMessage() {
        return text;
    }
}

package gsa;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionAdapter;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;
import java.util.LinkedList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.09.2010
 * Time: 13:59:04
 * To change this template use File | Settings | File Templates.
 */
public class GSAPanel extends JPanel {

    private static Color DEFAULT_TEXT_COLOR = Color.BLACK;
    private static Font DEFAULT_FONT = new Font("Sans Serif", Font.PLAIN, 14);

    private GSAModel model;
    private JFrame frame;

    private Node dragNode;
    private int dragX;
    private int dragY;

    private Node parentNodeToConnect;
    private boolean isParentNodeLogic;
    private Point fromPoint;
    private Point toPoint;

    public GSAPanel(GSAModel model, JFrame frame) {
        super();
        this.model = model;
        this.frame = frame;
        setBackground(model.getBackgroundColor());
        addMouseListener(new GSAMouseListener(frame));
        addMouseMotionListener(new GSAMouseMotionListener());
        dragNode = null;
        dragX = 0;
        dragY = 0;
    }

    public GSAModel getModel() {
        return model;
    }

    public void setModel(GSAModel model) {
        this.model = model;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(model.getNetColor());
```

```
        for (int i = model.getNetInterval(); i < getWidth(); i += model.getNetInterval()) {
            g2.drawLine(i, 0, i, getHeight());
        }
        for (int i = model.getNetInterval(); i < getHeight(); i += model.getNetInterval()) {
            g2.drawLine(0, i, getWidth(), i);
        }
        for (Node e : model.getNodes()) {
            if (e.getClass().getName().contains("BeginNode")) {
                drawBeginNode((BeginNode) e, g2);
            }
            else {
                if (e.getClass().getName().contains("OperatorNode")) {
                    drawOperatorNode((OperatorNode) e, g2);
                }
                else {
                    if (e.getClass().getName().contains("LogicNode")) {
                        drawLogicNode((LogicNode) e, g2);
                    }
                    else {
                        drawEndNode((EndNode) e, g2);
                    }
                }
            }
        }
        for (Node e : model.getNodes()) {
            LinkedList<Node> children = e.getChildren();
            boolean haveChildren = false;
            if (children != null) {
                for (Node c : children) {
                    if (c != null) {
                        haveChildren = true;
                    }
                }
            }
            if (haveChildren) {
                if (e.getClass().getName().contains("BeginNode") || e.getClass().getName().contains("OperatorNode") ||
e.getClass().getName().contains("EndNode")) {
                    Node child = children.getFirst();
                    drawArrowLine(new Point(e.getX() + e.getWidth() / 2, e.getY() + e.getHeight()),
                            new Point(child.getX() + child.getWidth() / 2, child.getY()), g2);
                } else {
                    if (e.getClass().getName().contains("LogicNode")) {
                        Node yesChild = children.getFirst();
                        Node noChild = children.getLast();
                        if (yesChild != null) {
                            drawArrowLine(new Point(e.getX(), e.getY() + e.getHeight() / 2),
                                    new Point(yesChild.getX() + yesChild.getWidth() / 2, yesChild.getY()), g2);
                        }
                        if (noChild != null) {
                            drawArrowLine(new Point(e.getX() + e.getWidth(), e.getY() + e.getHeight() / 2),
                                    new Point(noChild.getX() + noChild.getWidth() / 2, noChild.getY()), g2);
                        }
                    }
                }
            }
        }
        if (fromPoint != null) {
            drawArrowLine(fromPoint, toPoint, g2);
        }
    }

    private void drawArrowLine(Point p1, Point p2, Graphics2D g2) {
        g2.setColor(model.getLinesColor());
        g2.drawLine((int) p1.getX(), (int) p1.getY(), (int) p2.getX(), (int) p2.getY());
        double temp = Math.atan2(p1.getX() - p2.getX(), p1.getY() - p2.getY());
        g2.drawLine((int) p2.getX(), (int) p2.getY(),
                (int) (p2.getX() + GSAModel.DEFAULT_ARROW_WIDTH * Math.sin(temp + GSAModel.DEFAULT_ARROW_ANGLE)),
                (int) (p2.getY() + GSAModel.DEFAULT_ARROW_WIDTH * Math.cos(temp + GSAModel.DEFAULT_ARROW_ANGLE)));
        g2.drawLine((int) p2.getX(), (int) p2.getY(),
                (int) (p2.getX() + GSAModel.DEFAULT_ARROW_WIDTH * Math.sin(temp - GSAModel.DEFAULT_ARROW_ANGLE)),
                (int) (p2.getY() + GSAModel.DEFAULT_ARROW_WIDTH * Math.cos(temp - GSAModel.DEFAULT_ARROW_ANGLE)));
    }

    private void drawOperatorNode(OperatorNode node, Graphics2D g2) {
        g2.setColor(node.getColor());
        g2.fillRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());
        g2.setColor(DEFAULT_TEXT_COLOR);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = DEFAULT_FONT.getStringBounds(node.getText(), context);
        double x = node.getX() + (node.getWidth() - bounds.getWidth()) / 2;
        double y = node.getY() + node.getHeight() - (node.getHeight() - bounds.getHeight());
        g2.drawString(node.getText(), (int) x, (int) y);
        g2.setColor(model.getLinesColor());
        g2.fillRect(node.getX() + (node.getWidth() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2, node.getY() - GSAModel.DEFAULT_CONNECTORS_SIZE / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
        g2.fillRect(node.getX() + (node.getWidth() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2, node.getY() + node.getHeight() -
GSAModel.DEFAULT_CONNECTORS_SIZE / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
    }

    private void drawLogicNode(LogicNode node, Graphics2D g2) {
        Polygon shape = new Polygon();
        shape.addPoint(node.getX(), (node.getY() + node.getHeight() / 2));
        shape.addPoint((node.getX() + node.getWidth() / 2), node.getY());
        shape.addPoint((node.getX() + node.getWidth()), (node.getY() + node.getHeight() / 2));
        shape.addPoint((node.getX() + node.getWidth() / 2), (node.getY() + node.getHeight()));
        g2.setColor(node.getColor());
        g2.fill(shape);
        g2.setColor(DEFAULT_TEXT_COLOR);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = DEFAULT_FONT.getStringBounds(node.getText(), context);
        double x = node.getX() + (node.getWidth() - bounds.getWidth()) / 2;
        double y = node.getY() + node.getHeight() - (node.getHeight() - bounds.getHeight());
        g2.drawString(node.getText(), (int) x, (int) y);
        x = node.getX() + 5;
        y = node.getY() + node.getHeight();
        g2.drawString(LogicNode.YES_TEXT, (int) x, (int) y);
        bounds = DEFAULT_FONT.getStringBounds(LogicNode.NO_TEXT, context);
        x = node.getX() + node.getWidth() - bounds.getWidth() - 5;
```

```
        g2.drawString(LogicNode.NO_TEXT, (int) x, (int) y);
        g2.setColor(model.getLinesColor());
        g2.fillRect(node.getX() + (node.getWidth() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2, node.getY() - GSAModel.DEFAULT_CONNECTORS_SIZE / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
        g2.fillRect(node.getX() - GSAModel.DEFAULT_CONNECTORS_SIZE / 2, node.getY() + (node.getHeight() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
        g2.fillRect(node.getX() - GSAModel.DEFAULT_CONNECTORS_SIZE / 2 + node.getWidth(), node.getY() + (node.getHeight() -
GSAModel.DEFAULT_CONNECTORS_SIZE) / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
    }

    private void drawBeginNode(BeginNode node, Graphics2D g2) {
        g2.setColor(node.getColor());
        g2.fillOval(node.getX(), node.getY(), node.getHeight(), node.getHeight());
        g2.fillOval(node.getX() + node.getWidth() - node.getHeight(), node.getY(), node.getHeight(), node.getHeight());
        g2.fillRect(node.getX() + node.getHeight() / 2, node.getY(), node.getWidth() - node.getHeight(), node.getHeight());
        g2.setColor(DEFAULT_TEXT_COLOR);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = DEFAULT_FONT.getStringBounds(node.getText(), context);
        double x = node.getX() + (node.getWidth() - bounds.getWidth()) / 2;
        double y = node.getY() + node.getHeight() - (node.getHeight() - bounds.getHeight());
        g2.drawString(node.getText(), (int) x, (int) y);
        g2.setColor(model.getLinesColor());
        g2.fillRect(node.getX() + (node.getWidth() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2, node.getY() + node.getHeight() -
GSAModel.DEFAULT_CONNECTORS_SIZE / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
    }

    private void drawEndNode(EndNode node, Graphics2D g2) {
        g2.setColor(node.getColor());
        g2.fillOval(node.getX(), node.getY(), node.getHeight(), node.getHeight());
        g2.fillOval(node.getX() + node.getWidth() - node.getHeight(), node.getY(), node.getHeight(), node.getHeight());
        g2.fillRect(node.getX() + node.getHeight() / 2, node.getY(), node.getWidth() - node.getHeight(), node.getHeight());
        g2.setColor(DEFAULT_TEXT_COLOR);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = DEFAULT_FONT.getStringBounds(node.getText(), context);
        double x = node.getX() + (node.getWidth() - bounds.getWidth()) / 2;
        double y = node.getY() + node.getHeight() - (node.getHeight() - bounds.getHeight());
        g2.drawString(node.getText(), (int) x, (int) y);
        g2.setColor(model.getLinesColor());
        g2.fillRect(node.getX() + (node.getWidth() - GSAModel.DEFAULT_CONNECTORS_SIZE) / 2, node.getY() - GSAModel.DEFAULT_CONNECTORS_SIZE / 2,
                GSAModel.DEFAULT_CONNECTORS_SIZE, GSAModel.DEFAULT_CONNECTORS_SIZE);
    }

    private class GSAMouseListener implements MouseListener {

        private JFrame frame;

        public GSAMouseListener(JFrame frame) {
            super();
            this.frame = frame;
        }

        public void mouseClicked(MouseEvent e) {
            if ((e.getButton() == MouseEvent.BUTTON1) && (e.getClickCount() == 1)) {
                if (!model.isInNode(e.getX(), e.getY())) {
                    if (model.getActionType() == 1) {
                        try {
                            model.addBeginNode(e.getX(), e.getY(), GSAModel.DEFAULT_NODE_WIDTH, GSAModel.DEFAULT_NODE_HEIGHT);
                            repaint();
                        } catch (IllegalNodeException e1) {
                            JOptionPane.showMessageDialog(frame, "Error! There is already one Begin node.",
                                    "Error", JOptionPane.ERROR_MESSAGE);
                        }
                    } else {
                        if (model.getActionType() == 2) {
                            String signalCountString = JOptionPane.showInputDialog(frame, "Please, enter the number of signals:",
                                    "Input", JOptionPane.INFORMATION_MESSAGE);
                            int signalCount = Integer.valueOf(signalCountString);
                            if (signalCount > 0) {
                                int[] signalNumbers = new int[signalCount];
                                for (int i = 0; i < signalCount; i++) {
                                    String signalNumberString = JOptionPane.showInputDialog(frame, "Please, enter the number of signal:",
                                            "Input", JOptionPane.INFORMATION_MESSAGE);
                                    int signalNumber = Integer.valueOf(signalNumberString);
                                    if (signalNumber > 0) {
                                        signalNumbers[i] = signalNumber;
                                    } else {
                                        JOptionPane.showMessageDialog(frame, "Error! Incorrect number of signal.",
                                                "Error", JOptionPane.ERROR_MESSAGE);
                                    }
                                }
                                model.addOperatorNode(e.getX(), e.getY(), GSAModel.DEFAULT_NODE_WIDTH, GSAModel.DEFAULT_NODE_HEIGHT,
signalNumbers);
                                repaint();
                            } else {
                                JOptionPane.showMessageDialog(frame, "Error! The node must contain at least one signal.",
                                        "Error", JOptionPane.ERROR_MESSAGE);
                            }
                        } else {
                            if (model.getActionType() == 3) {
                                String signalNumberString = JOptionPane.showInputDialog(frame, "Please, enter the number of signal:",
                                        "Input", JOptionPane.INFORMATION_MESSAGE);
                                int signalNumber = Integer.valueOf(signalNumberString);
                                if (signalNumber > 0) {
                                    model.addLogicNode(e.getX(), e.getY(), GSAModel.DEFAULT_NODE_WIDTH, GSAModel.DEFAULT_NODE_HEIGHT,
signalNumber);
                                    repaint();
                                } else {
                                    JOptionPane.showMessageDialog(frame, "Error! Incorrect number of signal.",
                                            "Error", JOptionPane.ERROR_MESSAGE);
                                }
                            } else {
                                if (model.getActionType() == 4) {
                                    try {
                                        model.addEndNode(e.getX(), e.getY(), GSAModel.DEFAULT_NODE_WIDTH, GSAModel.DEFAULT_NODE_HEIGHT);
                                        repaint();
                                    } catch (IllegalNodeException e1) {
                                        JOptionPane.showMessageDialog(frame, "Error! There is already one End node.",
```

```
                                "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        }
    }
} else {
    if (model.getActionType() == 5) {
        int result = JOptionPane.showConfirmDialog(frame, "Are you sure to delete this node?",
            "Confirm", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
        if (result == JOptionPane.YES_OPTION) {
            model.removeNode(model.getInNode(e.getX(), e.getY()));
            repaint();
        }
    }
}
if (model.getActionType() == 5) {
    Node node = model.getNodeInEntrance(e.getX(), e.getY());
    if (node != null) {
        int result = JOptionPane.showConfirmDialog(frame, "Are you sure to delete all these connections?",
            "Confirm", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
        if (result == JOptionPane.YES_OPTION) {
            for (Node n : node.getParents()) {
                if (n.getClass().getName().contains("BeginNode")) {
                    BeginNode bNode = (BeginNode) n;
                    bNode.removeChildNode();
                }
                else {
                    if (n.getClass().getName().contains("OperatorNode")) {
                        OperatorNode oNode = (OperatorNode) n;
                        oNode.removeChildNode();
                    }
                    else {
                        if (n.getClass().getName().contains("LogicNode")) {
                            LogicNode lNode = (LogicNode) n;
                            if (lNode.getChildren().getFirst() == node) {
                                lNode.removeYesChildNode();
                            }
                            else {
                                lNode.removeNoChildNode();
                            }
                        }
                    }
                }
            }
            if (node.getClass().getName().contains("OperatorNode")) {
                OperatorNode oNode = (OperatorNode) node;
                oNode.removeParentNodes();
            }
            else {
                if (node.getClass().getName().contains("LogicNode")) {
                    LogicNode lNode = (LogicNode) node;
                    lNode.removeParentNodes();
                }
                else {
                    if (node.getClass().getName().contains("EndNode")) {
                        EndNode eNode = (EndNode) node;
                        eNode.removeParentNodes();
                    }
                }
            }
            repaint();
        }
    }
    else {
        node = model.getNotLogicNodeInExit(e.getX(), e.getY());
        if (node != null) {
            int result = JOptionPane.showConfirmDialog(frame, "Are you sure to delete this connection?",
                "Confirm", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
            if (result == JOptionPane.YES_OPTION) {
                Node child = node.getChildren().getFirst();
                if (child.getClass().getName().contains("OperatorNode")) {
                    OperatorNode oNode = (OperatorNode) child;
                    oNode.removeParentNode(node);
                }
                else {
                    if (child.getClass().getName().contains("LogicNode")) {
                        LogicNode lNode = (LogicNode) child;
                        lNode.removeParentNode(node);
                    }
                    else {
                        if (child.getClass().getName().contains("EndNode")) {
                            EndNode eNode = (EndNode) child;
                            eNode.removeParentNode(node);
                        }
                    }
                }
                if (node.getClass().getName().contains("BeginNode")) {
                    BeginNode bNode = (BeginNode) node;
                    bNode.removeChildNode();
                }
                else {
                    if (node.getClass().getName().contains("OperatorNode")) {
                        OperatorNode eNode = (OperatorNode) node;
                        eNode.removeChildNode();
                    }
                }
                repaint();
            }
        }
        else {
            node = model.getLogicNodeInExit(e.getX(), e.getY());
            if (node != null) {
                int result = JOptionPane.showConfirmDialog(frame, "Are you sure to delete this connection?",
                    "Confirm", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
                if (result == JOptionPane.YES_OPTION) {
                    LogicNode lNode = (LogicNode) node;
                    Node child;
                    if (model.isYesNoExit()) {
```

```
                                child = lNode.getChildren().getFirst();
                            }
                            else {
                                child = lNode.getChildren().getLast();
                            }
                            if (child.getClass().getName().contains("OperatorNode")) {
                                OperatorNode oChild = (OperatorNode) child;
                                oChild.removeParentNode(node);
                            }
                            else {
                                if (child.getClass().getName().contains("LogicNode")) {
                                    LogicNode lChild = (LogicNode) child;
                                    lChild.removeParentNode(node);
                                }
                                else {
                                    if (child.getClass().getName().contains("EndNode")) {
                                        EndNode eChild = (EndNode) child;
                                        eChild.removeParentNode(node);
                                    }
                                }
                            }
                            if (model.isYesNoExit()) {
                                lNode.removeYesChildNode();
                            }
                            else {
                                lNode.removeNoChildNode();
                            }
                        }
                        repaint();
                    }
                }
            }
        }
    }
    else {
        if ((e.getButton() == MouseEvent.BUTTON1) && (e.getClickCount() == 2) && (model.getActionType() == 0)) {
            if (model.isInNode(e.getX(), e.getY())) {
                Node node = model.getInNode(e.getX(), e.getY());
                if (node.getClass().getName().contains("OperatorNode")) {
                    OperatorNode oNode = (OperatorNode) node;
                    int[] oldSignals = oNode.getSignals();
                    String signalCountString = JOptionPane.showInputDialog(frame, "Please, enter the number of signals:",
                            String.valueOf(oldSignals.length));
                    int signalCount = Integer.valueOf(signalCountString);
                    if (signalCount > 0) {
                        int[] newSignals = new int[signalCount];
                        for (int i = 0; i < signalCount; i++) {
                            int initialValue = 1;
                            if (i < oldSignals.length) {
                                initialValue = oldSignals[i];
                            }
                            String signalNumberString = JOptionPane.showInputDialog(frame, "Please, enter the number of signal:",
                                    String.valueOf(initialValue));
                            int signalNumber = Integer.valueOf(signalNumberString);
                            if (signalNumber > 0) {
                                newSignals[i] = signalNumber;
                            } else {
                                if (i < oldSignals.length) {
                                    newSignals[i] = oldSignals[i];
                                }
                                else {
                                    newSignals[i] = 1;
                                }
                                JOptionPane.showMessageDialog(frame, "Error! Incorrect number of signal.",
                                        "Error", JOptionPane.ERROR_MESSAGE);
                            }
                        }
                        oNode.setSignals(newSignals);
                        repaint();
                    } else {
                        JOptionPane.showMessageDialog(frame, "Error! The node must contain at least one signal.",
                                "Error", JOptionPane.ERROR_MESSAGE);
                    }
                }
                else {
                    if (node.getClass().getName().contains("LogicNode")) {
                        LogicNode lNode = (LogicNode) node;
                        String signalNumberString = JOptionPane.showInputDialog(frame, "Please, enter the number of signal:",
                                 String.valueOf(lNode.getSignals()[0]));
                        int signalNumber = Integer.valueOf(signalNumberString);
                        if (signalNumber > 0) {
                            lNode.setSignal(signalNumber);
                            repaint();
                        } else {
                            JOptionPane.showMessageDialog(frame, "Error! Incorrect number of signal.",
                                    "Error", JOptionPane.ERROR_MESSAGE);
                        }
                    }
                }
            }
        }
    }
}

public void mousePressed(MouseEvent e) {
    if ((e.getButton() == MouseEvent.BUTTON1) && (model.getActionType() == 0) && (model.isInNode(e.getX(), e.getY()))) {
        dragNode = model.getInNode(e.getX(), e.getY());
        dragX = e.getX();
        dragY = e.getY();
    }
    else {
        if ((e.getButton() == MouseEvent.BUTTON1) && (model.getActionType() == 6)) {
            Node node = model.getNotLogicNodeInExit(e.getX(), e.getY());
            if ((node != null) && (node.getChildren().getFirst() == null)) {
                parentNodeToConnect = node;
                isParentNodeLogic = false;
                fromPoint = new Point(e.getX(), e.getY());
            }
            else {
```

```
                        node = model.getLogicNodeInExit(e.getX(), e.getY());
                        if ((node != null) && (((model.isYesNoExit()) && (node.getChildren().getFirst() == null)) || ((!model.isYesNoExit()) &&
(node.getChildren().getLast() == null)))) {
                            parentNodeToConnect = node;
                            isParentNodeLogic = true;
                            fromPoint = new Point(e.getX(), e.getY());
                        }
                    }
                }
            }
        }

        public void mouseReleased(MouseEvent e) {
            if ((dragNode != null) && (model.getActionType() == 0)) {
                dragNode = null;
            }
            else {
                if ((parentNodeToConnect != null) && (model.getActionType() == 6)) {
                    Node node = model.getNodeInEntrance(e.getX(), e.getY());
                    if (node != null) {
                        if (node.getClass().getName().contains("OperatorNode")) {
                            OperatorNode oNode = (OperatorNode) node;
                            oNode.addParentNode(parentNodeToConnect);
                        }
                        else {
                            if (node.getClass().getName().contains("LogicNode")) {
                                LogicNode lNode = (LogicNode) node;
                                lNode.addParentNode(parentNodeToConnect);
                            }
                            else {
                                if (node.getClass().getName().contains("EndNode")) {
                                    EndNode eNode = (EndNode) node;
                                    eNode.addParentNode(parentNodeToConnect);
                                }
                            }
                        }
                        if (isParentNodeLogic) {
                            LogicNode lNode = (LogicNode) parentNodeToConnect;
                            if (model.isYesNoExit()) {
                                lNode.setYesChildNode(node);
                            }
                            else {
                                lNode.setNoChildNode(node);
                            }
                        }
                        else {
                            if (parentNodeToConnect.getClass().getName().contains("BeginNode")) {
                                BeginNode bNode = (BeginNode) parentNodeToConnect;
                                bNode.setChildNode(node);
                            }
                            else {
                                if (parentNodeToConnect.getClass().getName().contains("OperatorNode")) {
                                    OperatorNode oNode = (OperatorNode) parentNodeToConnect;
                                    oNode.setChildNode(node);
                                }
                            }
                        }
                    }
                }
                parentNodeToConnect = null;
                isParentNodeLogic = false;
                fromPoint = null;
                toPoint = null;
                repaint();
            }
        }

        public void mouseEntered(MouseEvent e) {}

        public void mouseExited(MouseEvent e) {}

    }

    private class GSAMouseMotionListener extends MouseMotionAdapter {

        @Override
        public void mouseDragged(MouseEvent e) {
            if ((dragNode != null) && (model.getActionType() == 0)) {
                dragNode.setX(dragNode.getX() + (e.getX() - dragX));
                dragNode.setY(dragNode.getY() + (e.getY() - dragY));
                dragX = e.getX();
                dragY = e.getY();
                repaint();
            }
            else {
                if ((parentNodeToConnect != null) && (model.getActionType() == 6)) {
                    toPoint = new Point(e.getX(), e.getY());
                    repaint();
                }
            }
        }

    }

}

package gsa;

import javax.swing.filechooser.FileFilter;
import java.io.File;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 13.09.2010
 * Time: 18:59:48
 * To change this template use File | Settings | File Templates.
 */
public class GSAFileFilter extends FileFilter {
```

```
    public static String GSA_EXTENSION = ".gsa";

    private static String GSA_DESCRIPTION = "GSA File";

    public boolean accept(File pathname) {
        return (pathname.getName().toLowerCase().endsWith(GSA_EXTENSION) || pathname.isDirectory());
    }

    public String getDescription() {
        return GSA_DESCRIPTION;
    }

}

package face;

import gsa.GSAFileFilter;
import gsa.GSAModel;
import gsa.GSAPanel;
import gsa.IllegalNodeException;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.Scanner;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 12.09.2010
 * Time: 1:57:43
 * To change this template use File | Settings | File Templates.
 */
public class MainFrame extends JFrame {

    private JMenuBar menuBar;
    private JToolBar toolBar;
    private GSAPanel gsaPanel;
    private JLabel statusLabel;
    private JFileChooser chooser;

    private NewAction newAction;
    private OpenAction openAction;
    private SaveAction saveAction;
    private SaveAsAction saveAsAction;
    private CloseAction closeAction;
    private ExitAction exitAction;
    private AboutAction aboutAction;

    private File openedFile;

    public MainFrame(Rectangle bounds) {
        super();
        setBounds(bounds);
        setTitle("Computer-aided Designing of Computer Systems");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        newAction = new NewAction(this);
        openAction = new OpenAction(this);
        saveAction = new SaveAction(this);
        saveAsAction = new SaveAsAction(this);
        closeAction = new CloseAction(this);
        exitAction = new ExitAction(this);
        aboutAction = new AboutAction(this);
        addWindowListener(new WindowHandler(exitAction));
        menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenu helpMenu = new JMenu("Help");
        JMenuItem tempItem = new JMenuItem(newAction);
        tempItem.setText("New");
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(openAction);
        tempItem.setText("Open...");
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(saveAction);
        tempItem.setText("Save");
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(saveAsAction);
        tempItem.setText("Save As...");
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(closeAction);
        tempItem.setText("Close");
        fileMenu.add(tempItem);
        fileMenu.addSeparator();
        tempItem = new JMenuItem(exitAction);
        tempItem.setText("Exit");
        fileMenu.add(tempItem);
        tempItem = new JMenuItem(aboutAction);
        tempItem.setText("About...");
        helpMenu.add(tempItem);
        menuBar.add(fileMenu);
        menuBar.add(helpMenu);
        setJMenuBar(menuBar);
        setLayout(new BorderLayout());
        toolBar = new JToolBar(JToolBar.VERTICAL);
        toolBar.setFloatable(false);
        toolBar.setRollover(true);
        JButton tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(0);
                statusLabel.setText(" ");
            }
        });
        tempButton.setIcon(new ImageIcon("img/no_action.png"));
        tempButton.setToolTipText("No action");
        toolBar.addSeparator();
```

```
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(1);
                statusLabel.setText("Adding Begin Node");
            }
        });
        tempButton.setIcon(new ImageIcon("img/begin_node.png"));
        tempButton.setToolTipText("Begin Node");
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(2);
                statusLabel.setText("Adding Operator Node");
            }
        });
        tempButton.setIcon(new ImageIcon("img/operator_node.png"));
        tempButton.setToolTipText("Operator Node");
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(3);
                statusLabel.setText("Adding Logic Node");
            }
        });
        tempButton.setIcon(new ImageIcon("img/logic_node.png"));
        tempButton.setToolTipText("Logic Node");
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(4);
                statusLabel.setText("Adding End Node");
            }
        });
        tempButton.setIcon(new ImageIcon("img/end_node.png"));
        tempButton.setToolTipText("End Node");
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(6);
                statusLabel.setText("Connecting nodes");
            }
        });
        tempButton.setIcon(new ImageIcon("img/line.png"));
        tempButton.setToolTipText("Connect nodes");
        toolBar.addSeparator();
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                gsaPanel.getModel().setActionType(5);
                statusLabel.setText("Deleting");
            }
        });
        tempButton.setIcon(new ImageIcon("img/delete_node.png"));
        tempButton.setToolTipText("Delete");
        add(toolBar, BorderLayout.WEST);
        statusLabel = new JLabel(" ");
        add(statusLabel, BorderLayout.SOUTH);
        gsaPanel = new GSAPanel(new GSAModel(), this);
        add(new JScrollPane(gsaPanel));
        chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));
        chooser.addChoosableFileFilter(new GSAFileFilter());
        chooser.setMultiSelectionEnabled(false);
        openedFile = null;
    }

    private class NewAction extends AbstractAction {

        private MainFrame frame;

        public NewAction(MainFrame frame) {
            super();
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            if (gsaPanel.getModel().isChanged()) {
                closeAction.actionPerformed(e);
            }
            gsaPanel.setModel(new GSAModel());
            gsaPanel.setVisible(true);
            statusLabel.setText(" ");
            frame.repaint();
        }

    }

    private class OpenAction extends AbstractAction {

        private MainFrame frame;

        public OpenAction(MainFrame frame) {
            super();
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            int result = chooser.showOpenDialog(frame);
            if (result == JFileChooser.APPROVE_OPTION) {
                try {
                    Scanner input = new Scanner(new BufferedReader(new FileReader(chooser.getSelectedFile())));
                    int n = input.nextInt();
                    int[] nodesTypeMatrix = new int[n];
                    int[][] boundsMatrix = new int[n][];
                    for (int i = 0; i < boundsMatrix.length; i++) {
                        boundsMatrix[i] = new int[4];
                    }
                    int[][] connectivityMatrix = new int[n][];
                    for (int i = 0; i < connectivityMatrix.length; i++) {
                        connectivityMatrix[i] = new int[n];
                    }
                    int[][] signalMatrix = new int[n][];
                    for (int i = 0; i < nodesTypeMatrix.length; i++) {
                        nodesTypeMatrix[i] = input.nextInt();
```

```
                }
                for (int i = 0; i < boundsMatrix.length; i++) {
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        boundsMatrix[i][j] = input.nextInt();
                    }
                }
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        connectivityMatrix[i][j] = input.nextInt();
                    }
                }
                for (int i = 0; i < signalMatrix.length; i++) {
                    signalMatrix[i] = new int[input.nextInt()];
                }
                for (int i = 0; i < signalMatrix.length; i++) {
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        signalMatrix[i][j] = input.nextInt();
                    }
                }
                try {
                    gsaPanel.getModel().setGSA(nodesTypeMatrix, connectivityMatrix, signalMatrix, boundsMatrix);
                    gsaPanel.setVisible(true);
                    openedFile = chooser.getSelectedFile();
                    input.close();
                    statusLabel.setText(" ");
                    frame.repaint();
                } catch (IllegalNodeException e1) {
                    JOptionPane.showMessageDialog(frame, "Error! Incorrect GSA.",
                            "Error", JOptionPane.ERROR_MESSAGE);
                }
            } catch (FileNotFoundException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't open selected file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

}

private class SaveAction extends AbstractAction {

    private MainFrame frame;

    public SaveAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (openedFile == null) {
            saveAsAction.actionPerformed(e);
        }
        else {
            openedFile.delete();
            try {
                PrintWriter output = new PrintWriter(new FileWriter(openedFile));
                int[] nodesTypeMatrix = gsaPanel.getModel().getNodesType();
                int[][] boundsMatrix = gsaPanel.getModel().getBoundsMatrix();
                int[][] connectivityMatrix = gsaPanel.getModel().getConnectionMatrix();
                int[][] signalMatrix = gsaPanel.getModel().getSignalMatrix();
                output.println(nodesTypeMatrix.length);
                output.println();
                for (int i = 0; i < nodesTypeMatrix.length; i++) {
                    output.print(nodesTypeMatrix[i]);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < boundsMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        output.print(boundsMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.println();
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        output.print(connectivityMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.print("\n\n");
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.print(signalMatrix[i].length);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        output.print(signalMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.close();
                gsaPanel.getModel().setChanged(false);
            } catch (IOException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

}

private class SaveAsAction extends AbstractAction {

    private MainFrame frame;
```

```
    public SaveAsAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        int result = chooser.showSaveDialog(frame);
        if (result == JFileChooser.APPROVE_OPTION) {
            try {
                if (!chooser.getSelectedFile().getName().endsWith(GSAFileFilter.GSA_EXTENSION)) {
                    chooser.setSelectedFile(new File(chooser.getSelectedFile().getAbsolutePath() + GSAFileFilter.GSA_EXTENSION));
                }
                PrintWriter output = new PrintWriter(new FileWriter(chooser.getSelectedFile()));
                int[] nodesTypeMatrix = gsaPanel.getModel().getNodesType();
                int[][] boundsMatrix = gsaPanel.getModel().getBoundsMatrix();
                int[][] connectivityMatrix = gsaPanel.getModel().getConnectionMatrix();
                int[][] signalMatrix = gsaPanel.getModel().getSignalMatrix();
                output.println(nodesTypeMatrix.length);
                output.println();
                for (int i = 0; i < nodesTypeMatrix.length; i++) {
                    output.print(nodesTypeMatrix[i]);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < boundsMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        output.print(boundsMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.println();
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        output.print(connectivityMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.print("\n\n");
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.print(signalMatrix[i].length);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        output.print(signalMatrix[i][j]);
                        output.print(" ");
                    }
                }
                openedFile = chooser.getSelectedFile();
                output.close();
                gsaPanel.getModel().setChanged(false);
            } catch (IOException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

}

private class CloseAction extends AbstractAction {

    private MainFrame frame;

    public CloseAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (gsaPanel.getModel().isChanged()) {
            int result = JOptionPane.showConfirmDialog(frame,
                    "GSA has unsaved changes. Do you want to save them before closing?", "Warning",
                    JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE);
            if (result == JOptionPane.CANCEL_OPTION) {
                return;
            }
            else {
                if (result == JOptionPane.YES_OPTION) {
                    saveAction.actionPerformed(e);
                }
            }

        }
        gsaPanel.setVisible(false);
        frame.remove(gsaPanel);
        openedFile = null;
        statusLabel.setText(" ");
        frame.repaint();
    }

}

private class ExitAction extends AbstractAction {

    private MainFrame frame;

    public ExitAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (gsaPanel.getModel().isChanged()) {
```

```
                closeAction.actionPerformed(e);
            }
            System.exit(0);
        }

    }

    private class AboutAction extends AbstractAction {

        private MainFrame frame;

        public AboutAction(MainFrame frame) {
            super();
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,
                    "Computer-aided Designing of Computer Systems\nCopyright (c) 2010 Zakhozhyy Ihor",
                    "About", JOptionPane.INFORMATION_MESSAGE);
        }

    }

    private class WindowHandler extends WindowAdapter {

                    private ExitAction exitAction;

                    public WindowHandler(ExitAction exitAction) {
                            this.exitAction = exitAction;
                    }

                    public void windowClosing(final WindowEvent e) {
            final ActionEvent e2 = new ActionEvent(this, EXIT_ON_CLOSE, "close");
            exitAction.actionPerformed(e2);
            super.windowClosing(e);
                    }

          }

}

import face.MainFrame;

import javax.swing.*;
import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 12.09.2010
 * Time: 1:56:55
 * To change this template use File | Settings | File Templates.
 */
public class Program {

    private static int MIN_WIDTH = 800;
    private static int MIN_HEIGHT = 600;

    private static Rectangle getDefaultBounds() {
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int width = (int) (screenSize.getWidth() / 10 * 8);
        if (width < MIN_WIDTH) {
            width = MIN_WIDTH;
        }
        int height = (int) (screenSize.getHeight() / 10 * 8);
        if (height < MIN_HEIGHT) {
            height = MIN_HEIGHT;
        }
        return new Rectangle(((int) screenSize.getWidth() - width) / 2, ((int) screenSize.getHeight() - height) / 2,
                width, height);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                MainFrame frame = new MainFrame(getDefaultBounds());
                frame.setMinimumSize(new Dimension(MIN_WIDTH, MIN_HEIGHT));
                frame.setVisible(true);
            }
        });
    }

}
```

Висновки

При виконанні даної лабораторної роботи мною було побудовано редактор графічних блок-схем алгоритмів (ГСА). Я розробив інтерфейс користувача даної програми та його функціональне наповнення та засоби перетворення форматів зберігання даних. Створені за допомогою даного редактора блок-схеми при збереженні перетворюються в матричну форму і в текстовому виді записуються у файл. Дана програма була написана на мові програмування Java. Інтерфейс користувача реалізований за допомогою пакету Swing.