Национальный технический университет Украины

«Киевский политехнический институт»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Курсовой проект

по курсу

«Автоматизация проектирования компьютерных систем»

по теме

«Логическое моделирование схем с помощью событийного алгоритма (ЛИД-модель элементов)»

Выполнил

студент группы ИВ-73

Захожий Игорь

Зачетная книжка №7308

Допущен к защите _____

_____ (Чебаненко Т.М.)

Киев-2011

ОПИСЬ АЛЬБОМА

№ строки	Формат	Обозначения	Наименование	Кол. лист.	№ экз.	Прим.
1						
2			Документация общая			
3						
4			Заново разработанная			
5						
6	А4	ИАЛЦ.462637.002 ТЗ	Техническое задание	3		
7						
8	А4	ИАЛЦ.462637.003 ПЗ	Пояснительная записка	21		
9						
10	А4	ИАЛЦ.462637.004	Приложение А. Блок-схема	1		
11			событийного алгоритма			
12	А4	ИАЛЦ.462637.005	Приложение Б. Листинг	47		
13			программы			
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						

Подп. и дата | Инв.№ дубл. | Взам. инв. № | Подп. и дата | Инв.№ подл.

Изм.	Лист	№ документа	Подп.	Дата
Разраб		Захожий И.А.		
Провер.		Чебаненко Т.М.		
Н.контр.				
Утв.				

ИАЛЦ.462637.001 ОА

Опись альбома

Лит.	Лист	Листов
Т	1	1

НТУУ «КПИ», ФИВТ, группа ИВ-73

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Содержание

Подп. и дата | Инв.№ дубл. | Взам. инв. № | Подп. и дата | Инв.№ подл.

Изм.	Лист	№ документа	Подпись	Дата	ИАЛЦ. 467449.002 ТЗ		
Разработал		Захожий И.А.			Техническое задание	Лит.	Лист 1 / Листов 3
Проверил		Чебаненко Т.М.				Т	
Н. контр.						НТУУ «КПИ», ФИВТ, группа ИВ-73	
Утвердил							

1. Наименование и область применения

Необходимо разработать программный продукт для логического моделирования схем с помощью событийного алгоритма (ЛИД-модель элементов). Данный программный продукт может применяться в лабораториях, которые разрабатывают средства для производства схем.

2. Основание для разработки

Основанием для разработки служит индивидуальное задание на курсовой проект, выданное кафедрой Вычислительной Техники НТУУ "КПИ" четвертому курсу специальности "Компьютерные системы и сети" по дисциплине "Автоматизация проектирования компьютерных систем".

3. Цель и назначение разработки

Целью данной работы является закрепление полученных знаний по дисциплине «Автоматизация проектирования компьютерных систем», а также получение навыков и опыта в проектировании законченных программных продуктов, соответствующих всем требованиям технического задания.

4. Технические требования

4.1 Общие требования

Разрабатываемый программный продукт должен обеспечивать интерфейс для ввода логических схем и средства сохранения и загрузки данных о уже введенных схемах, выполнять моделирование работы схемы. Также необходимо ввести контроль вводимых исходных данных, реализовать визуализацию вводимых данных. Результат должен отображаться в виде таблиц и временных диаграмм.

4.2 Требования к уровню стандартизации

Необходимо оформить работу в виде технической документации, которая включает: опись альбома, техническое задание, пояснительную записку, приложения.

4.3 Требования к программному обеспечению

Программный продукт должен работать под управлением операционных систем Microsoft Windows XP/Vista/7 и должен быть реализован на языке программирования Java.

5. Стадии и этапы разработки

- Согласование технического задания.
- Анализ существующих разработок.
- Разработка программного продукта.
- Защита курсового проекта.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Содержание

Введение

Широкое распространение радиоэлектронных устройств с применением цифровой обработки сигналов обуславливает повышенный интерес к вопросам диагностирования их технического состояния.

Одной из разновидностей диагностирования цифровых узлов и блоков является тестовое диагностирование, применение которого на этапе проектирования и изготовления цифровых узлов позволяет определить правильность их функционирования. При разработке тестовой диагностики возникает сложность в определении эталонных реакций при тестировании существующих схем и оптимального числа контрольных точек для снятия выходной реакции диагностируемой цифровой схемы. Это можно сделать либо создавая прототип разрабатываемого цифрового устройства и проводя его диагностику аппаратурными методами, либо осуществляя моделирование на ЭВМ как цифрового устройства, так и процесса диагностики. Наиболее рациональным является второй подход, который предполагает создание автоматизированных систем, позволяющих производить диагностику цифровых схем на стадии проектирования. Одними из таких систем являются системы логического моделирования.

Цель логического моделирования состоит в том, чтобы выполнить функцию проектируемой схемы без её физической реализации. Проверка на правильность моделирования может быть различной в зависимости от уровня представления цифровой схемы в ЭВМ. Если, например, осуществляется проверка только значений логической функции на выходе схемы, то достаточно представить схему на уровне логических элементов. Для того чтобы проверить состояния сигналов в схеме, необходимо точно описать задержки срабатывания всех элементов в условиях синхронизации.

В данной работе рассматривается логическое моделирование схем с помощью событийного алгоритма (ЛИД-модель элементов).

1. Обзор методов логического моделирования

В отношении асинхронных моделей возможны два метода логического моделирования — пошаговый (инкрементный) и событийный [2].

В пошаговом методе время дискретизируется и вычисления по выражениям модели выполняются в дискретные моменты времени t_0, t_1, t_2, ... и т. д. Шаг дискретизации ограничен сверху значением допустимой погрешности определения задержек и потому оказывается довольно малым, а время анализа — значительным.

Для сокращения времени анализа используют событийный метод. В этом методе событием называют изменение любой переменной модели. Событийное моделирование основано на следующем правиле: обращение к модели логического элемента происходит только в том случае, если на входах этого элемента произошло событие. В сложных логических схемах на каждом такте синхронизации обычно происходит переключение всего лишь 2 ... 3 % логических элементов, и соответственно в событийном методе в несколько раз уменьшаются вычислительные затраты по сравнению с пошаговым моделированием.

Методы анализа синхронных моделей представляют собой методы решения систем логических уравнений. К этим методам относятся метод простых итераций и метод Зейделя, которые аналогичны одноименным методам решения систем алгебраических уравнений в непрерывной математике.

При использовании метода простых итераций в исходном состоянии задают начальные (можно произвольные) значения промежуточных и выходных переменных. Новое состояние должно соответствовать указанным в таблице изменившимся значениям входных сигналов. Вычисления заканчиваются, если на очередной итерации изменений переменных нет.

Согласно методу простых итераций, в правые части уравнений модели на каждой итерации подставляют значения переменных, полученные

на предыдущей итерации. В отличие от этого в методе Зейделя, если у некоторой переменной обновлено значение на текущей итерации, именно его и используют в дальнейших вычислениях уже на текущей итерации. Метод Зейделя позволяет сократить число итераций, но для этого нужно предварительно упорядочить уравнения модели так, чтобы последовательность вычислений соответствовала последовательности прохождения сигналов по схеме. Такое упорядочение выполняют с помощью ранжирования.

Ранжирование заключается в присвоении элементам и переменным модели значений рангов в соответствии со следующими правилами: 1) в схеме разрываются все контуры обратной связи, что приводит к появлению дополнительных входов схемы (псевдовходов); 2) все внешние переменные (в том числе на псевдовходах) получают ранг 0; 3) элемент и его выходные переменные получают ранг к, если у элемента все входы проранжированы и старший среди рангов входов равен к – 1.

Для сокращения обьема вычислений в синхронном моделировании возможно использование событийного подхода. Обращение к модели элемента происходит, только если на его входах произошло событие.

2. Описание используемого алгоритма логического моделирования

В данной работе для реализации системы логического моделирования использовался событийный алгоритм моделирования логических схем (ЛИД-модель элементов).

Основным свойством событийного алгоритма является то, что на каждом шаге выполняется моделирование работы не всего множества элементов схемы, а только тех элементов, у которых изменился хотя бы один входной сигнал и, следовательно, может измениться выходной сигнал. Моделирование, при котором используется модель, не учитывающая временных характеристик, называется аналитической статической (Л-модель). Модель, любой элемент которой можно представить функциональным и динамическим блоками, учитывает временные характеристики и называется логико-динамической (ЛД-модель). В данной работе используется ЛИД-модель элементов, которая также учитывает инерционные свойства элементов.

Реализация событийного алгоритма требует наличия Таблицы Будущих Событий (ТБС), Таблицы Текущих Событий (ТТС) и счётчика системного времени (ССВ). В ТБС заносятся номера элементов, на входах которых изменились сигналы. На каждом шаге элементу ставится момент времени Т возможного изменения сигнала на выходе элемента, который определяется суммой ССВ, динамической и инерционной задержек элемента. Таблица Текущих Событий (ТТС) содержит номера элементов, для которых выполняется расчёт выходных сигналов на данном шаге.

Начальное значение ССВ равняется 0. Затем подаются нулевые сигналы, и выполняется моделирование схемы по алгоритму простой итерации. После этого подаётся набор входных сигналов, и в ТБС заносятся номера элементов, на которые поступили сигналы. На каждом следующем шаге из ТБС в ТТС переносятся номера тех элементов, Т которых минимальное, и выполняется расчёт выходных сигналов всех элементов из ТТС. Если выходной сигнал элемента изменил своё значение,

то в ТБС заносятся номера всех элементов, на входы которых подаётся данный сигнал.

Итерации для заданного набора оканчиваются, когда ТБС пуста, либо ССВ превысил значение $ССВ_k + T_{max}$. T_{max} равняется задержке самой длинной цепочки элементов.

Блок-схема использованного алгоритма приведена в Приложении А.

3. Разработка программного обеспечения

3.1 Описание разработанных типов данных и процедур

Для выполнения технического задания были разработаны следующие типы данных и процедуры:

- Пакет zak.adcs - содержит главный класс программы и классы для сохранения настроек программы и загрузки изображений.
  - Класс Program – главный класс программы.
  - Класс Configuration – класс для сохранения настроек программы посредством сериализации в XML.
  - Класс ImageIconLoader – класс для загрузки изображений из jar-дистрибутива.
- Пакет zak.adcs.gui – содержит классы для отображения визуальных окон программы.
  - Класс MainFrame – класс для представления главного окна программы и с помощью элементов интерфейса предоставляет доступ к основным функциям программы.
  - Класс InternalFrame – класс для предоставления интерфейса для работы со схемой и моделирования, а также для возможности одновременной работы с множеством схем.
  - Класс SchemeDialog – класс, представляющий диалоговое окно для задания настроек схемы.
- Пакет zak.adcs.logicsheme.elements – содержит классы и интерфейсы для представления и работы с моделями логических элементов.
  - Интерфейс VisualElement – интерфейс элемента в редакторе схемы.
    - public void draw(Graphics2D g2) – метод для отрисовки элемента в редакторе схем.

- *public void move(int x, int y) — метод для перемещения элемента в редакторе схем.*
- *public void setSelected(boolean selected) — метод для выделения/отмены выделения элемента в редакторе схем.*
- *public boolean contains(Point p) — метод для проверки принадлежности точки области отрисовки элемента.*
- *public boolean isOverlaped(Rectangle r) — метод, чтобы проверить не перекрывает ли прямоугольник область отрисовки элемента.*

o *Абстрактный класс ClosedElement — класс для представления абстракции элемента с замкнутым контуром в редакторе схем.*

o *Интерфейс Enterable — интерфейс элемента, который имеет хотя бы один вход.*

- *public boolean isInEnter(Point point) — метод для проверки, находится ли точка в входе в области отрисовки элемента.*
- *public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException — метод для получения точки входа в области отрисовки элемента.*
- *public String getName() — метод для получения имени элемента.*
- *public void addEnterConnectionName(String name, Point point) — метод для для подсоединения выхода другого элемента к входу данного.*

o *Интерфейс Exitable — интерфейс элемента, который имеет выход.*

- *public boolean isInExit(Point point) — метод для проверки, находится ли точка в выходе в области отрисовки элемента.*
- *public Point getExitPoint(Point point) throws ConnectionAlreadyExistException — метод для получения точки выхода в области отрисовки элемента.*

- *public String getName() – метод для получения имени элемента.*
- *public void addExitConnectionName(String name, Point point) – метод для подсоединения входа другого элемента к выходу данного.*

- *Интерфейс LIDElement – интерфейс логического элемента, который имеет динамическую и инерционную задержки.*
  - *public int getInertiaDelay() – метод для получения инерционной задержки логического элемента.*
  - *public int getDynamicDelay() – метод для получения динамической задержки логического элемента.*
  - *public void setInertiaDelay(int inertiaDelay) – метод для установки инерционной задержки логического элемента.*
  - *public void setDynamicDelay(int dynamicDelay) – метод для установки динамической задержки логического элемента.*
- *Класс ConnectionElement – класс для представления соединения между элемента в редакторе схем.*
- *Класс InElement – класс для представления входа схемы.*
- *Класс OutElement – класс для представления выхода схемы.*
- *Абстрактный класс LogicElement – класс для представления логического элемента.*
- *Класс AndElement – класс для представления логического элемента И.*
- *Класс OrElement – класс для представления логического элемента ИЛИ.*
- *Класс NandElement – класс для представления логического элемента И-НЕ.*
- *Класс NorElement – класс для представления логического элемента ИЛИ-НЕ.*

- Класс XorElement – класс для представления логического элемента Исключительное ИЛИ.
- Класс ConnectionAlreadyExistException – класс для представления исключения при попытке добавить соединения к уже занятому входу/выходу.

- Пакет zak.adcs.logicscheme.editor – содержит классы для отображения редактора схем, представления и сохранения/восстановления схемы в памяти.
  - Класс SchemeEditor – класс для отображения рабочей области редактора схем и элементов управления редактором.
  - Класс SchemePanel – класс для отображения рабочей области редактора схем, а также обработки действий пользователя.
    - Внутренний класс MouseHandler – класс для обработки событий мыши в рабочей области редактора схем.
  - Класс IOElementDialog – класс для отображения диалогового окна с параметрами элемента входа/выхода схемы.
  - Класс LIDElementDialog – класс для отображения диалогового окна с параметрами логического элемента.
  - Класс SchemeModel – класс для представления модели схемы в памяти, а также для ее сохранения/восстановления.
- Пакет zak.adcs.logicscheme.modelling – содержит классы для моделирования логических схем и вывода результатов в табличном виде и в виде временной диаграммы.
  - Класс SchemeTableModel – класс для представления матрицы связности элементов схемы.
  - Класс ModellingPanel – класс для отображения результатов моделирования и элементов интерфейса для управления моделированием.
  - Класс TimeDiagramPanel – класс для отображения результатов моделирования в виде временной диаграммы.

- *Класс LogicFunctionsWorker – класс для моделирования логических элементов схемы.*
  - *Метод public static char getValue(LogicFunction function, String inputSet) – метод для получения значения выхода логического элемента при заданных значениях входных сигналов.*
- *Класс ModellingModel – класс для моделирования схем.*
  - *Метод public boolean isBusy() – метод, определяющий завершено ли моделирование схемы.*
  - *Метод public void step(String inputSet, String startSet) – метод для моделирования схемы при новом входном наборе с установочным набором на один интервал времени.*
  - *Метод public void step() – метод для моделирования схемы на один интервал времени.*
  - *Метод public void step(String inputSet, boolean newSet) – метод для моделирования схемы при новом входном наборе без установочного набора на один интервал времени.*
  - *Метод public void modelling(String inputSet, String startSet) – метод для моделирования схемы.*
  - *Метод public void clear() – метод для завершения моделирования и очистки результатов.*
  - *Приватный метод private String[] getStartCondition(String startSet) – метод для получения начального состояния схемы путем моделирования схемы методом простой итерации.*
  - *Приватный метод private boolean hasFutureEvents() – метод для проверки наличия будущих событий схемы.*

3.2 Описание возможностей программы

При запуске программы на экране появляется главное окно программы, изображенное на рисунке 3.1.

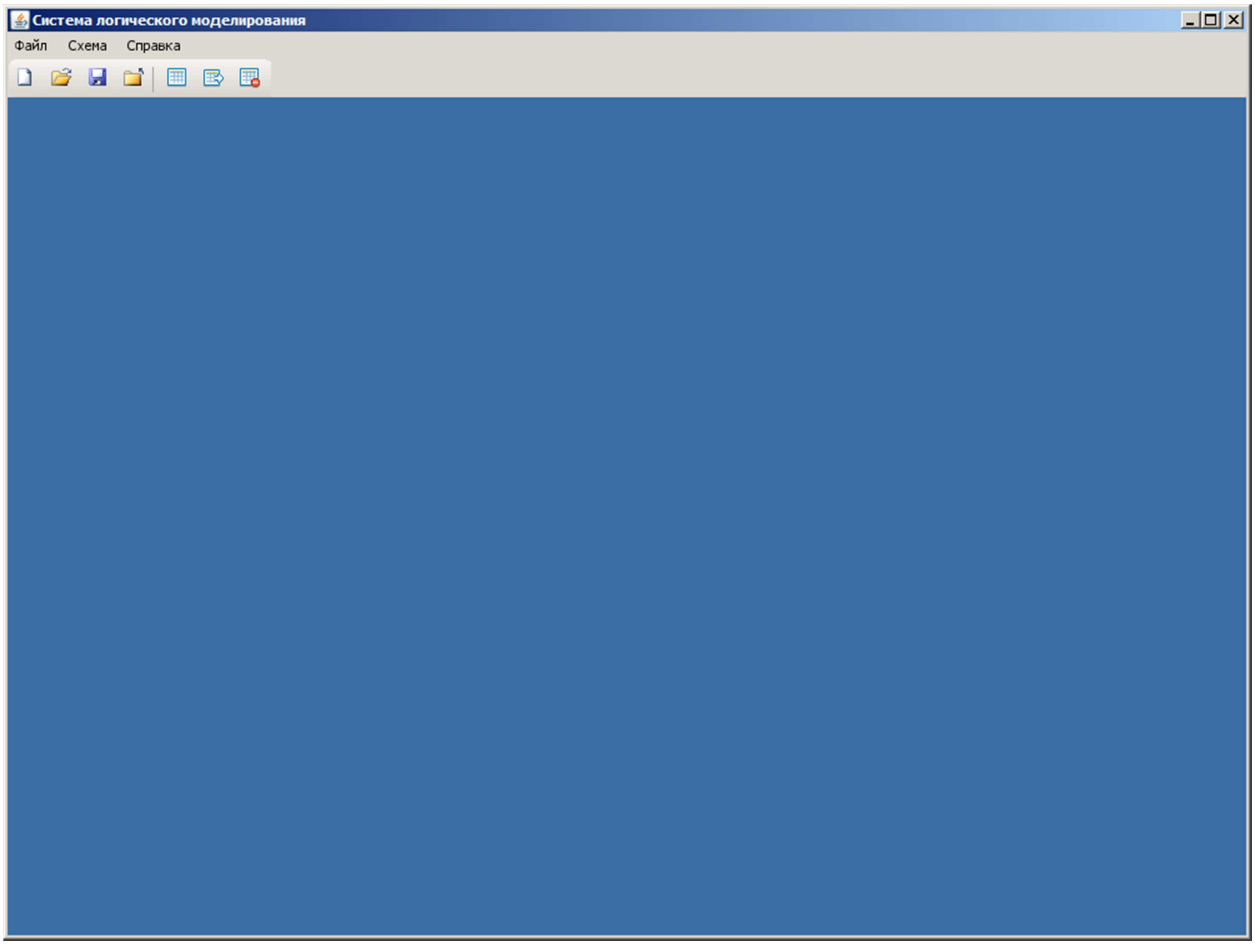


Рис. 3.1 – Главное окно программы

Главное окно содержит главное меню и главную панель инструментов. Программа дает возможность работать с множеством схем одновременно, так как поддерживает интерфейс MDI, и каждая схема имеет свой фрейм внутри главного окна.

Для создания новой схемы можно воспользоваться пунктом главного меню «Файл»-> «Новый…», комбинацией клавиш Ctrl-N или кнопкой на панели инструментов. После этого на экране появится диалоговое окно для указания настроек отображения схемы, которое изображено на рисунке 3.2.

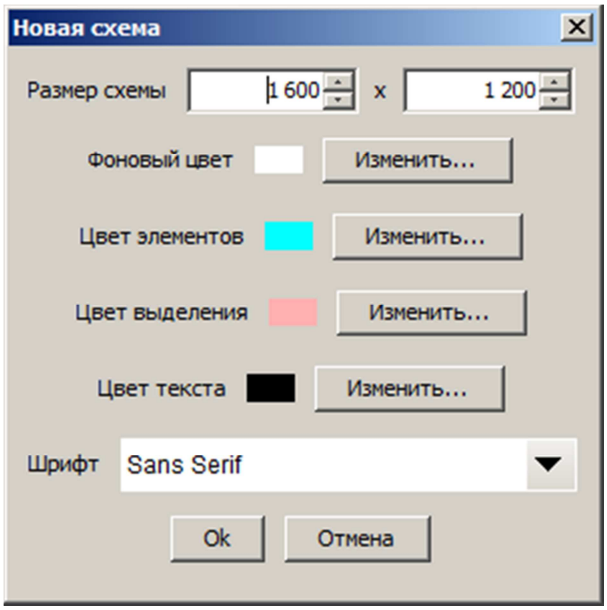


Рис. 3.2 — Диалоговое окно для выбора настроек отображения схемы

После этого в рабочей области главного окна программы откроется фрейм для работы со схемой, который изображен на рисунке 3.3.

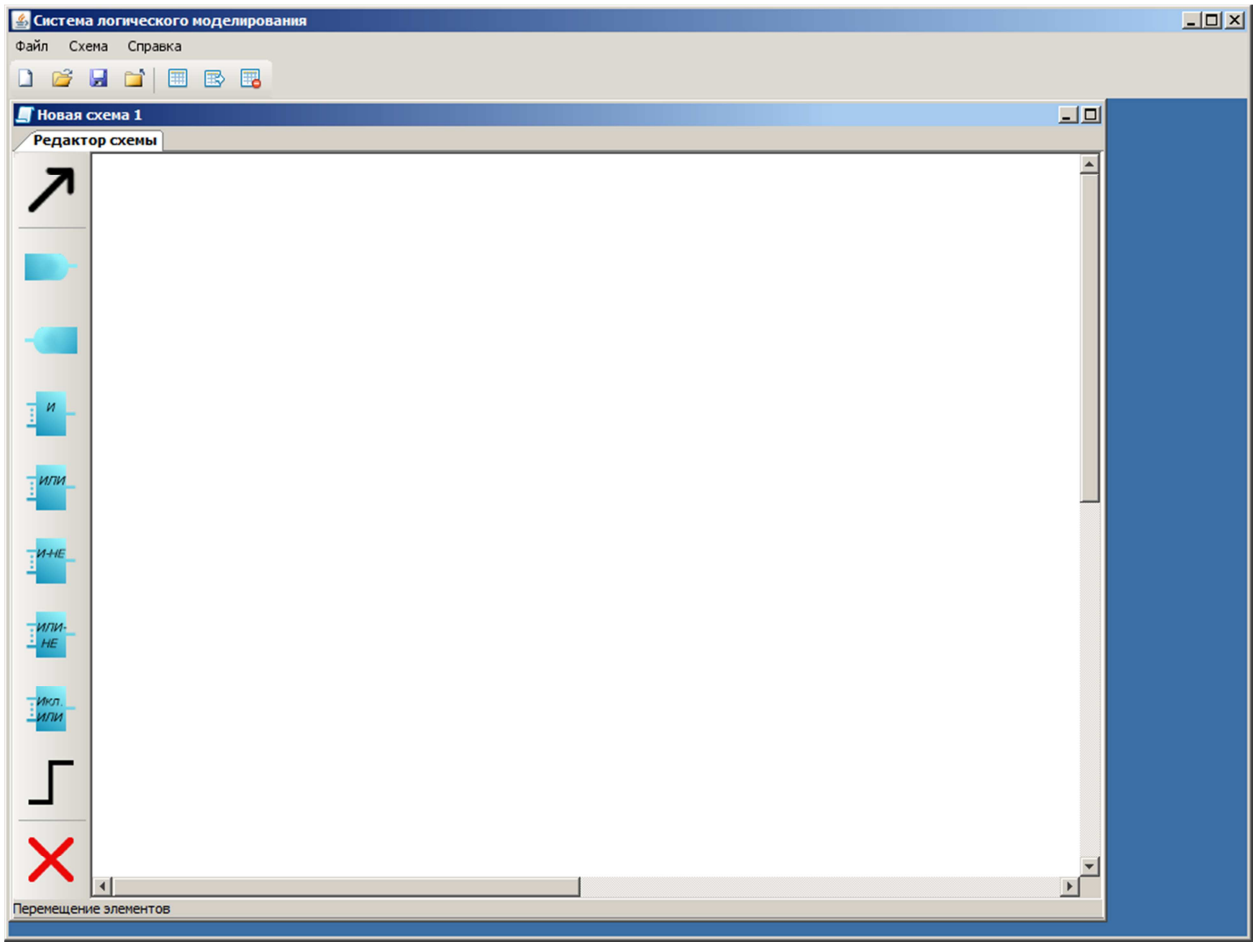


Рис. 3.3 — Главное окно программы с фреймом для работы со схемой

Слева во внутреннем фрейме находится панель инструментов для ввода схемы. С помощью кнопок этой панели можно выбрать необходимое действие. Непосредственно сам ввод производится с помощью мыши в рабочей области редактора схем с поддержкой механизма drag'n'drop. Для добавления элемента на схему необходимо выбрать необходимый элемент на панели инструментов и кликнуть на рабочей области. После этого появится диалоговое окно для ввода параметров элемента (рис. 3.4).

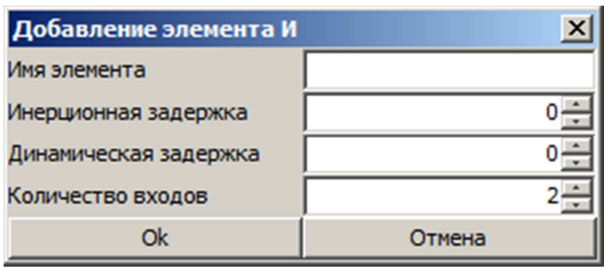


Рис. 3.4 — Диалоговое окно для ввода параметров элемента

Параметры элемента можно изменить позже, сделав на нем двойной клик. При добавлении элемента в недопустимое место будет выведено окно, сообщающее об ошибке (Рис. 3.5).

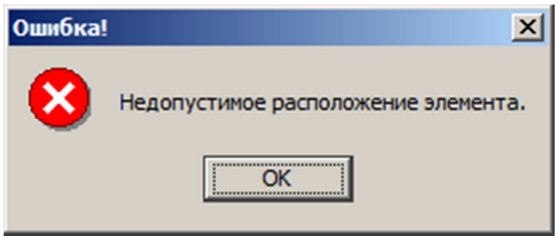


Рис. 3.5 — Диалоговое окно, сообщающее о ошибке ввода

Для соединения элементов необходимо выбрать данное действие на панели инструментов, нажать клавишу мыши на входе/выходе элемента и тянуть мышь к выходу/входу необходимого элемента. Возможные соединения будут подсвечиваться.

Для удаления элемента необходимо выбрать данное действие на панели инструментов и кликнуть на элементе и подтвердить данное действие в диалоговом окне.

Для сохранения схемы можно воспользоваться пунктом главного меню «Файл»-> «Сохранить», комбинацией клавиш Ctrl-S или кнопкой на главной

панели инструментов. Схема сохраняется в XML-файл. Для открытия ранее созданной схемы необходимо воспользоваться пунктом главного меню «Файл»-> «Открыть…», комбинацией клавиш Ctrl-O или кнопкой на главной панели инструментов. Закрыть схему можно с помощью пункта главного меню «Файл»-> «Закрыть», комбинации клавиш Ctrl-W или кнопки на главной панели инструментов.

Пример созданной схемы приведен на рисунке 3.6.

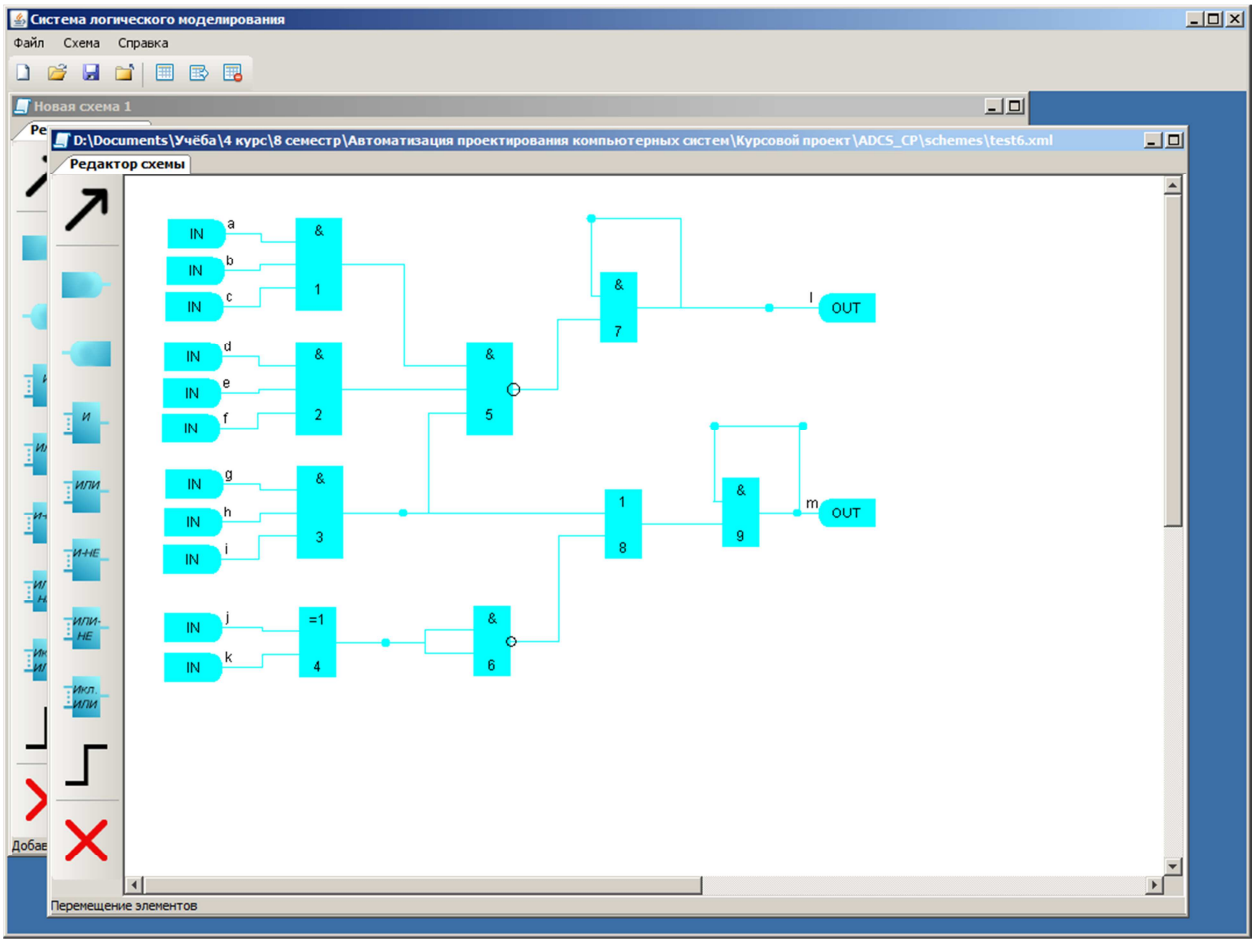
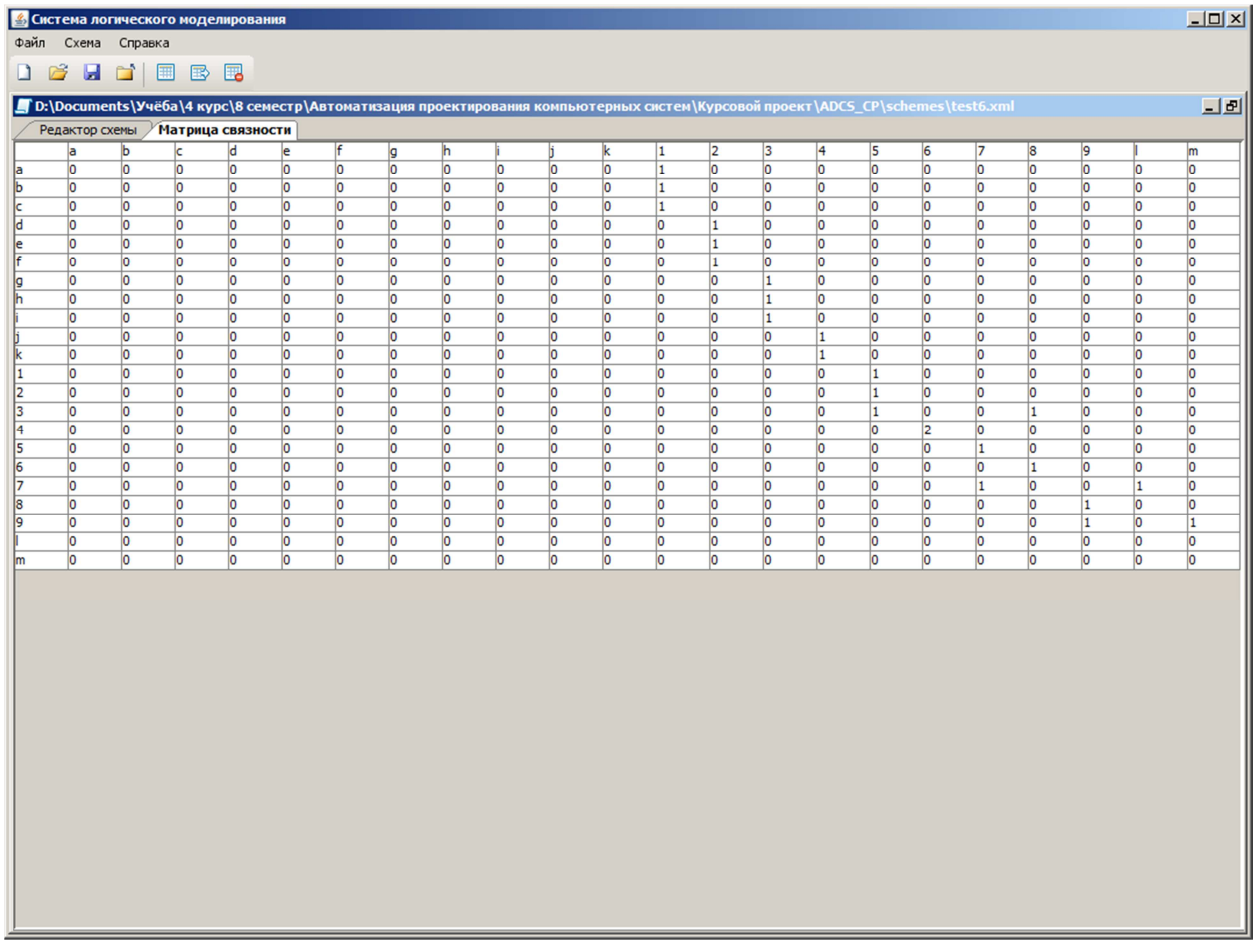


Рис. 3.6 — Пример созданной схемы

Для просмотра матрицы связей между элементами схемы необходимо воспользоваться пунктом главного меню «Схема»-> «Матрица связности», комбинацией клавиш Ctrl-Shift-M или кнопкой на главной панели инструментов. При этом матрица будет выведена в новой вкладке в фрейме схемы. Матрица связей между элементами схемы для схемы на рис. 3.6 приведена на рисунке 3.7.



	a	b	c	d	e	f	g	h	i	j	k	1	2	3	4	5	6	7	8	9	l	m
a	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис. 3.7 — Матрица связей между элементами схемы с рис.3.6

Для перехода в режим моделирования нужно использовать пункт главного меню «Схема»-> «Моделирование», комбинацию клавиш Ctrl-M или кнопку на главной панели инструментов. При этом изменение схемы становится невозможным. Чтобы отредактировать схему нужно выйти из режима моделирования. Сделать это можно с помощью пункта главного меню «Схема»-> «Редактировать схему», комбинации клавиш Ctrl-E или кнопки на главной панели инструментов.

При переходе в режим моделирования для управления моделированием и вывода результатов откроется новая вкладка в фрейме схемы (рис. 3.8). Вверху находится панель управления моделирования. Ниже находятся таблица и временная диаграмма для вывода результатов.

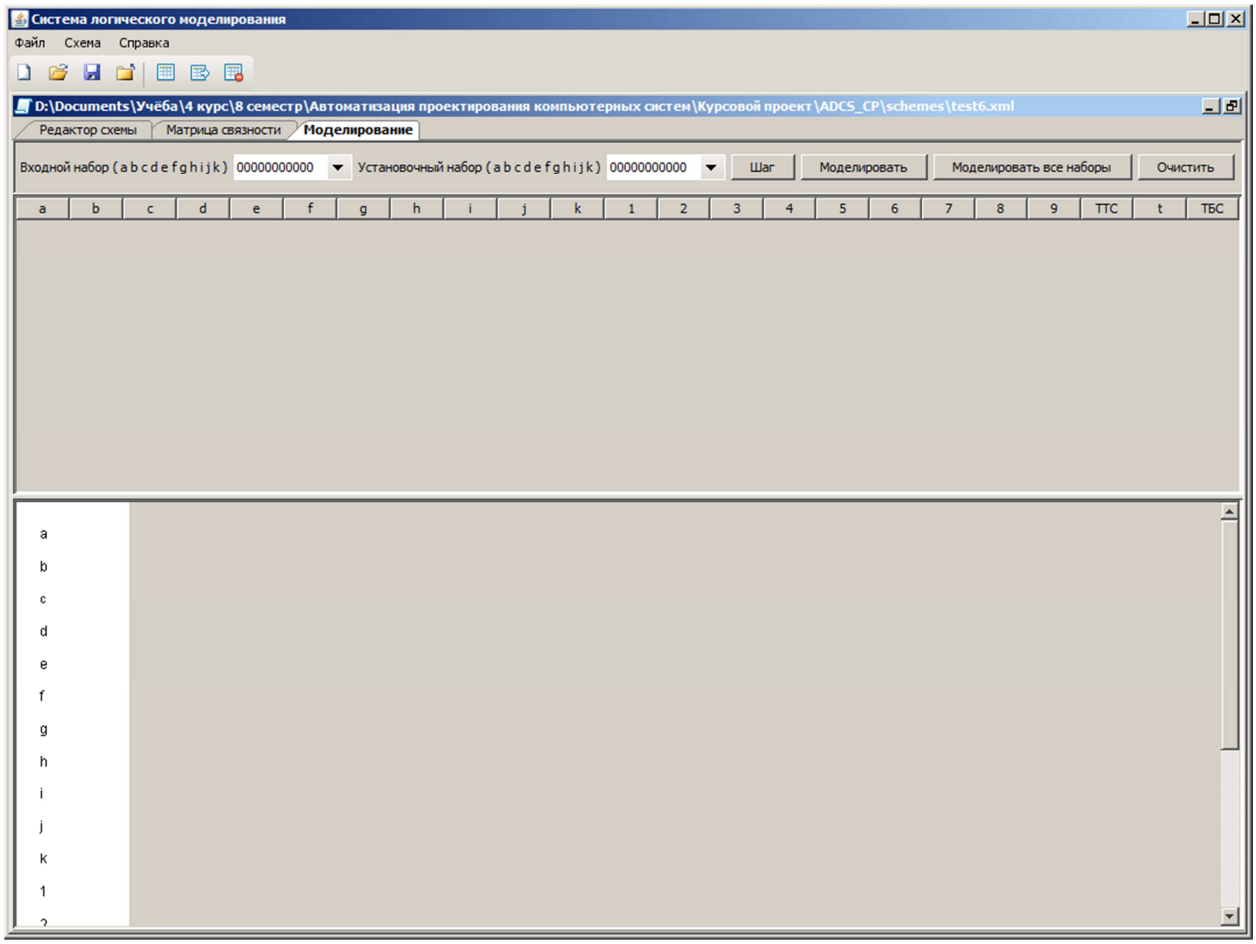


Рис. 3.8 – Вкладка режима моделирования схемы

Для начала моделирования необходимо выбрать из списка установочный входной набор, необходимый входной набор и нажать кнопку «Моделировать». Также имеется возможность моделировать схему пошагово с помощью кнопки «Шаг» и промоделировать схему на всех возможный входных наборах с помощью кнопки «Моделировать все наборы». Для завершения моделирования и очистки результатов нужно воспользоваться кнопкой «Очистить».

Примеры моделирования схемы с рис. 3.6 приведены на рисунках 3.9 и 3.10.

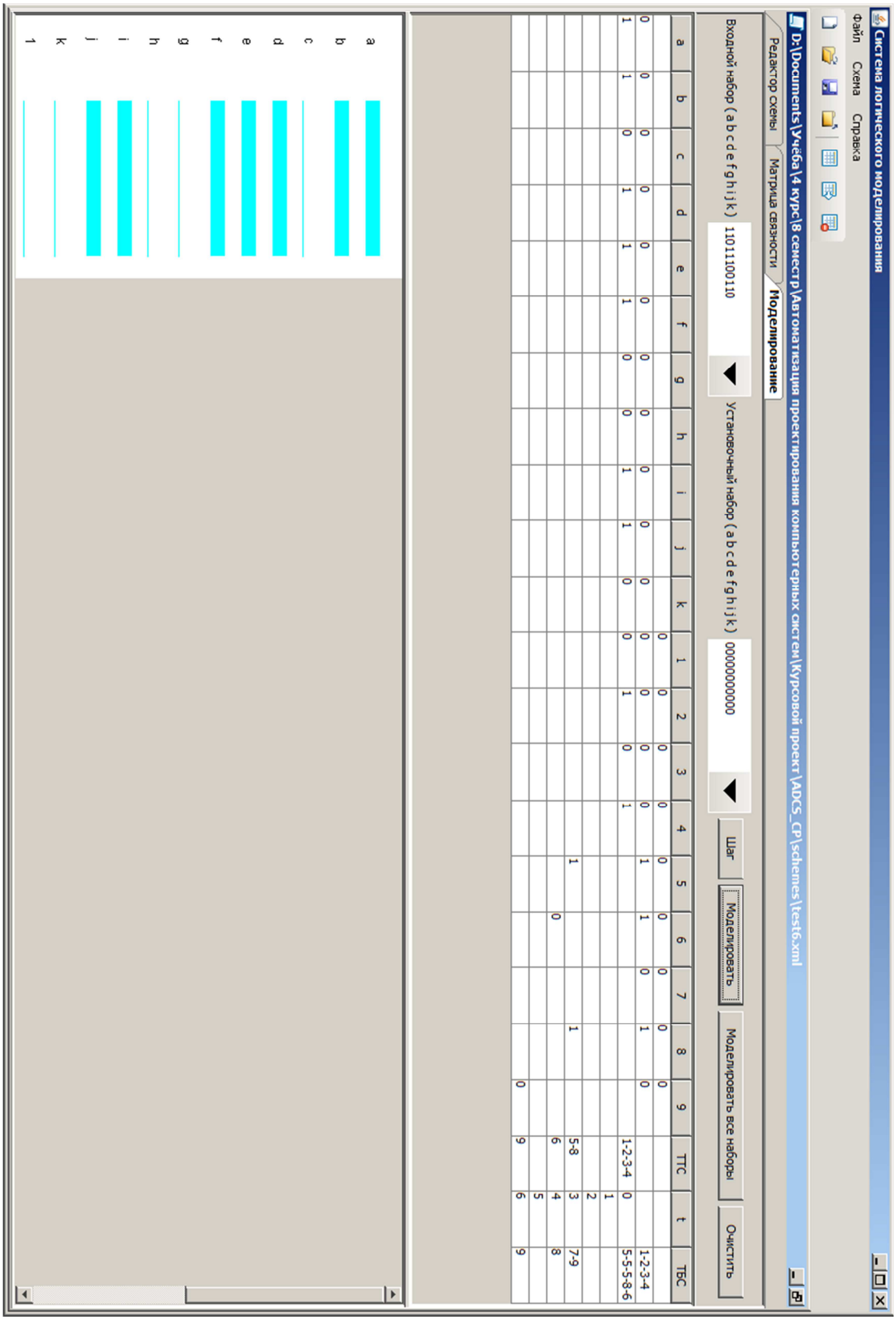


Рис. 3.9 — Пример моделирования схемы с рис.3.6

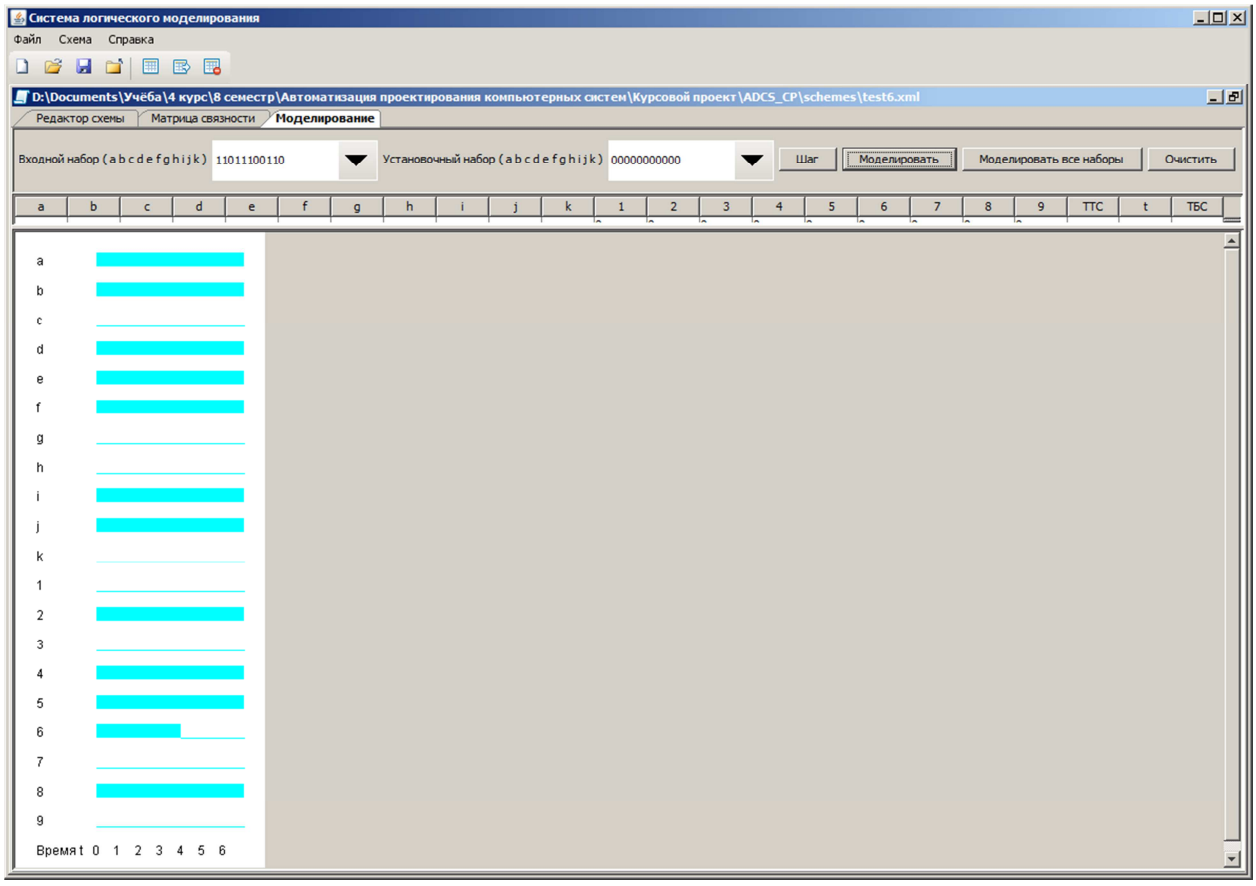


Рис. 3.10 — Временная диаграмма моделирования схемы с рис.3.6

Закрыть программу можно с помощью пункта главного меню «Файл»-> «Выход» или комбинации клавиш Ctrl-Q.

Заключение

В данной работе была разработана программа, позволяющая производить моделирования логических схем с помощью событийного алгоритма (ЛИД-модель элементов).

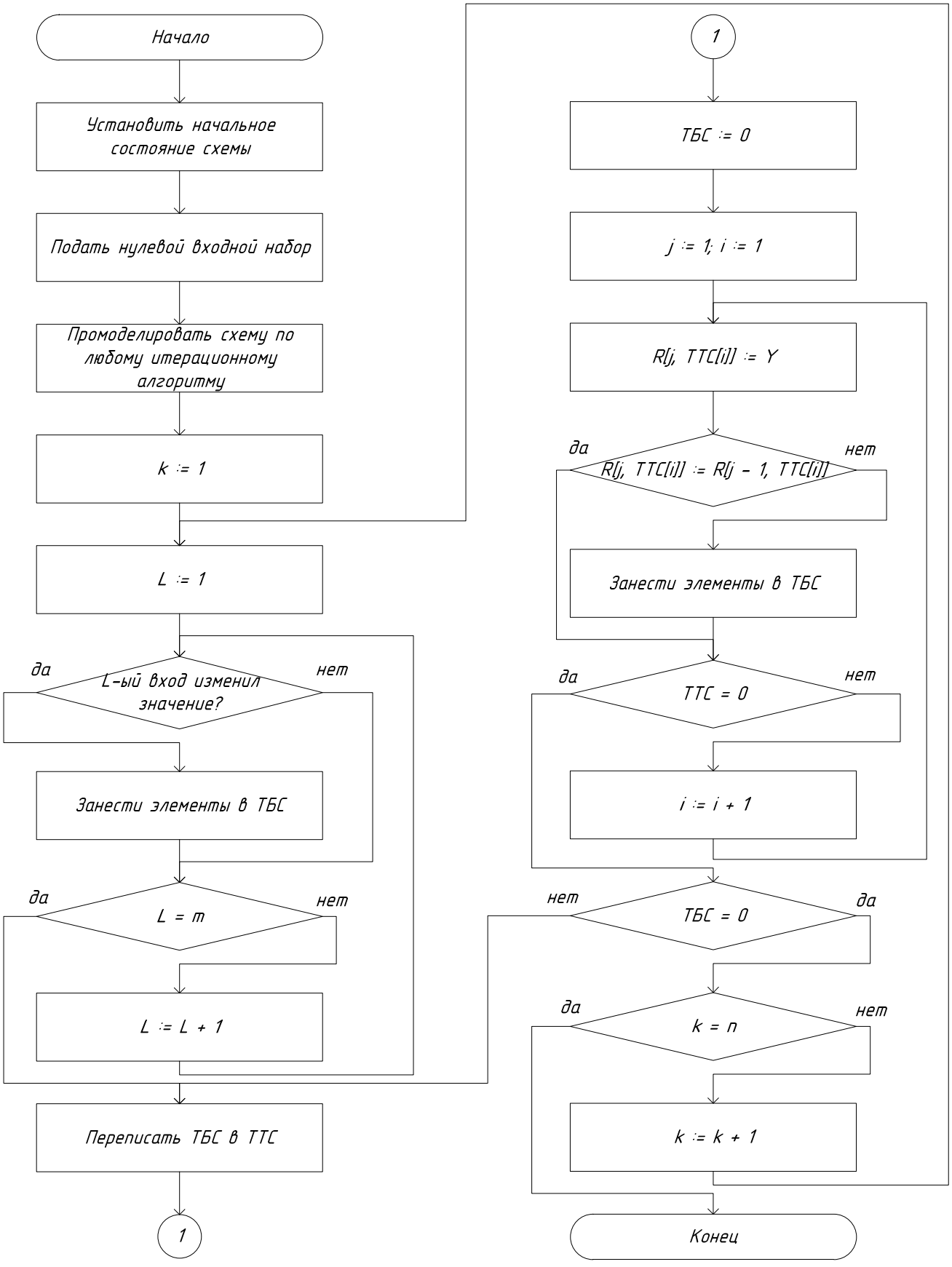
Преимуществом использованного для моделирования алгоритма является то, что на каждом шаге производится моделирование не всех элементов схемы, а только тех, на входах которых возможно изменение сигнала. Следовательно, при моделировании с помощью данного алгоритма производится меньшее количество вычислений, чем при использовании итерационных алгоритмов. Также достоинством событийного алгоритма является то, что он довольно точно отображает поведение схемы, так как дает возможность учитывать динамические и инерционные задержки на элементах схемы.

Недостаток событийного алгоритма – это сложность реализации из-за интенсивного использования динамических таблиц текущих и будущих событий.

Список литературы

1. Конспект лекций по курсу «Автоматизация проектирования компьютерных систем».
2. Норенков И.П. Основы автоматизированного проектирования // М.: Издательство МГТУ имени Н.Э. Баумана, 2002. — 336 с.
3. Жабін В.І., Жуков І.А., Клименко І.А., Ткаченко В.В. Прикладна теорія цифрових автоматів: Навч. посібник // К.: Книжкове видавництво НАУ, 2007. — 364 с.

ПРИЛОЖЕНИЕ А



						ИАЛЦ.462637.004	
Изм.	Кол.	Лист	№	Подпись	Дата		
Разраб.		Захожий И.А.				Приложение А. Блок-схема событийного алгоритма	Лит. / Лист 1 / Листов 1
Провер.		Чебаненко Т.М.					НТУУ «КПИ» ФИВТ гр. ИВ-73
Реценз.							
Н. Контр.							
Утвер.							

ПРИЛОЖЕНИЕ Б

```
package zak.adcs;

import com.thoughtworks.xstream.XStream;
import zak.adcs.gui.MainFrame;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 28.02.11
 *         Time: 0:08
 */
public class Program {

    private static final File CONFIGURATION_FILE = new File("conf.xml");

    private static Configuration configuration;

    private static void readConfigurationFile() {
        try {
            BufferedReader input = new BufferedReader(new FileReader(CONFIGURATION_FILE));
            StringBuilder builder = new StringBuilder();
            String line;
            while ((line = input.readLine()) != null) {
                builder.append(line);
            }
            input.close();
            XStream xStream = new XStream();
            xStream.alias("configuration", Configuration.class);
            configuration = (Configuration) xStream.fromXML(builder.toString());
        } catch (IOException e) {
            Toolkit kit = Toolkit.getDefaultToolkit();
            Dimension screenSize = kit.getScreenSize();
            configuration = new Configuration(new Rectangle((int) ((screenSize.getWidth() - MainFrame.MIN_WIDTH) / 2),
                    (int) ((screenSize.getHeight() - MainFrame.MIN_HEIGHT) / 2), MainFrame.MIN_WIDTH, MainFrame.MIN_HEIGHT));
        }
    }

    public static void writeConfigurationFile(Rectangle mainFrameBounds) {
        try {
            configuration = new Configuration(mainFrameBounds);
            XStream xStream = new XStream();
            xStream.alias("configuration", Configuration.class);
            String xml = xStream.toXML(configuration);
            PrintWriter output = new PrintWriter(CONFIGURATION_FILE);
            output.write(xml);
            output.close();
        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Не удалось сохранить настройки программы.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("org.fife.plaf.VisualStudio2005.VisualStudio2005LookAndFeel");
        } catch (UnsupportedLookAndFeelException e) {
            JOptionPane.showMessageDialog(null,
                    "Не удалось включить OfficeLnFs LookAndFeel. Будет использоваться Metal LookAndFeel.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        } catch (ClassNotFoundException e) {
            JOptionPane.showMessageDialog(null,
                    "Не удалось включить OfficeLnFs LookAndFeel. Будет использоваться Metal LookAndFeel.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        } catch (InstantiationException e) {
            JOptionPane.showMessageDialog(null,
                    "Не удалось включить OfficeLnFs LookAndFeel. Будет использоваться Metal LookAndFeel.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        } catch (IllegalAccessException e) {
            JOptionPane.showMessageDialog(null,
                    "Не удалось включить OfficeLnFs LookAndFeel. Будет использоваться Metal LookAndFeel.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
        readConfigurationFile();
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                MainFrame mainFrame = new MainFrame(configuration.getMainFrameBounds());
                mainFrame.setVisible(true);
            }
        });
    }

}
package zak.adcs;

import javax.swing.*;
import java.net.URL;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 08.03.11
 *         Time: 2:58
 */
public class ImageIconLoader {

    private static ImageIconLoader instance;
```

```
    private ImageIconLoader() {}

    public static ImageIconLoader getInstance() {
        if (instance == null) {
            instance = new ImageIconLoader();
        }
        return instance;
    }

    public ImageIcon getImageIcon(String path) {
        ImageIcon imageIcon = null;
        URL imageURL = getClass().getResource(path);
        if (imageURL == null) {
            imageIcon = new ImageIcon(path.substring(1));
        } else {
            imageIcon = new ImageIcon(imageURL);
        }
        return imageIcon;
    }

}
```

```
package zak.adcs;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 28.02.11
 *         Time: 1:16
 */
public class Configuration {

    private int mainframex;
    private int mainframey;
    private int mainframewidth;
    private int mainframeheight;

    public Configuration(Rectangle mainFrameBounds) {
        mainframex = (int) mainFrameBounds.getX();
        mainframey = (int) mainFrameBounds.getY();
        mainframewidth = (int) mainFrameBounds.getWidth();
        mainframeheight = (int) mainFrameBounds.getHeight();
    }

    public Rectangle getMainFrameBounds() {
        return new Rectangle(mainframex, mainframey, mainframewidth, mainframeheight);
    }

}
```

```
package zak.adcs.gui;

import zak.adcs.ImageIconLoader;
import zak.adcs.Program;
import zak.adcs.logicscheme.editor.SchemeModel;

import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.net.URL;
import java.util.Scanner;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 28.02.11
 *         Time: 2:02
 */
public class MainFrame extends JFrame {

    public static final int MIN_WIDTH = 1000;
    public static final int MIN_HEIGHT = 750;

    private JMenuBar menuBar;
    private JToolBar toolBar;
    private JDesktopPane desktopPane;
    private JFileChooser fileChooser;

    private NewAction newAction;
    private OpenAction openAction;
    private SaveAction saveAction;
    private SaveAsAction saveAsAction;
    private CloseAction closeAction;
    private ExitAction exitAction;
    private ConnectivityAction connectivityAction;
    private ModellingAction modellingAction;
    private EditSchemeAction editSchemeAction;
    private AboutAction aboutAction;

    public MainFrame(Rectangle bounds) {
        super();
        setBounds(bounds);
        setMinimumSize(new Dimension(MIN_WIDTH, MIN_HEIGHT));
        setTitle("Система логического моделирования");
```

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
addWindowListener(new MainFrameListener());
newAction = new NewAction();
openAction = new OpenAction();
saveAction = new SaveAction();
saveAsAction = new SaveAsAction();
closeAction = new CloseAction();
exitAction = new ExitAction();
connectivityAction = new ConnectivityAction();
modellingAction = new ModellingAction();
editSchemeAction = new EditSchemeAction();
aboutAction = new AboutAction(this);
ClassLoader classLoader = getClass().getClassLoader();
URL url = classLoader.getResource("org/fife/plaf/Office2003/new.gif");
ImageIcon newIcon = new ImageIcon(url);
url = classLoader.getResource("org/fife/plaf/Office2003/open.gif");
ImageIcon openIcon = new ImageIcon(url);
url = classLoader.getResource("org/fife/plaf/Office2003/save.gif");
ImageIcon saveIcon = new ImageIcon(url);
url = classLoader.getResource("org/fife/plaf/Office2003/saveas.gif");
ImageIcon saveAsIcon = new ImageIcon(url);
url = classLoader.getResource("org/fife/plaf/Office2003/close.gif");
ImageIcon closeIcon = new ImageIcon(url);
url = classLoader.getResource("org/fife/plaf/Office2003/delete.gif");
ImageIcon exitIcon = new ImageIcon(url);
ImageIcon connectivityIcon = ImageIconLoader.getInstance().getImageIcon("/img/connectivity_table.png");
ImageIcon modellingIcon = ImageIconLoader.getInstance().getImageIcon("/img/modelling.png");
ImageIcon editSchemeIcon = ImageIconLoader.getInstance().getImageIcon("/img/edit_scheme.png");
url = classLoader.getResource("org/fife/plaf/Office2003/about.gif");
ImageIcon aboutIcon = new ImageIcon(url);
menuBar = new JMenuBar();
JMenu fileMenu = new JMenu("Файл");
JMenu schemeMenu = new JMenu("Схема");
JMenu helpMenu = new JMenu("Справка");
JMenuItem tempItem = new JMenuItem(newAction);
tempItem.setText("Новый...");
tempItem.setIcon(newIcon);
tempItem.setMnemonic('N');
tempItem.setAccelerator(KeyStroke.getKeyStroke('N', KeyEvent.CTRL_MASK));
fileMenu.add(tempItem);
tempItem = new JMenuItem(openAction);
tempItem.setText("Открыть...");
tempItem.setIcon(openIcon);
tempItem.setMnemonic('O');
tempItem.setAccelerator(KeyStroke.getKeyStroke('O', KeyEvent.CTRL_MASK));
fileMenu.add(tempItem);
tempItem = new JMenuItem(saveAction);
tempItem.setText("Сохранить");
tempItem.setIcon(saveIcon);
tempItem.setMnemonic('S');
tempItem.setAccelerator(KeyStroke.getKeyStroke('S', KeyEvent.CTRL_MASK));
fileMenu.add(tempItem);
tempItem = new JMenuItem(saveAsAction);
tempItem.setText("Сохранить как...");
tempItem.setIcon(saveAsIcon);
tempItem.setMnemonic('S');
tempItem.setAccelerator(KeyStroke.getKeyStroke('S', (KeyEvent.CTRL_MASK | KeyEvent.SHIFT_MASK)));
fileMenu.add(tempItem);
tempItem = new JMenuItem(closeAction);
tempItem.setText("Закрыть");
tempItem.setIcon(closeIcon);
tempItem.setMnemonic('W');
tempItem.setAccelerator(KeyStroke.getKeyStroke('W', KeyEvent.CTRL_MASK));
fileMenu.add(tempItem);
fileMenu.addSeparator();
tempItem = new JMenuItem(exitAction);
tempItem.setText("Выход");
tempItem.setIcon(exitIcon);
tempItem.setMnemonic('Q');
tempItem.setAccelerator(KeyStroke.getKeyStroke('Q', KeyEvent.CTRL_MASK));
fileMenu.add(tempItem);
tempItem = new JMenuItem(connectivityAction);
tempItem.setText("Матрица связности");
tempItem.setIcon(connectivityIcon);
tempItem.setMnemonic('M');
tempItem.setAccelerator(KeyStroke.getKeyStroke('M', (KeyEvent.CTRL_MASK | KeyEvent.SHIFT_MASK)));
schemeMenu.add(tempItem);
tempItem = new JMenuItem(modellingAction);
tempItem.setText("Моделировать");
tempItem.setIcon(modellingIcon);
tempItem.setMnemonic('M');
tempItem.setAccelerator(KeyStroke.getKeyStroke('M', KeyEvent.CTRL_MASK));
schemeMenu.add(tempItem);
tempItem = new JMenuItem(editSchemeAction);
tempItem.setText("Редактировать схему");
tempItem.setIcon(editSchemeIcon);
tempItem.setMnemonic('E');
tempItem.setAccelerator(KeyStroke.getKeyStroke('E', KeyEvent.CTRL_MASK));
schemeMenu.add(tempItem);
tempItem = new JMenuItem(aboutAction);
tempItem.setText("О программе...");
tempItem.setIcon(aboutIcon);
helpMenu.add(tempItem);
menuBar.add(fileMenu);
menuBar.add(schemeMenu);
menuBar.add(helpMenu);
setJMenuBar(menuBar);
toolBar = new JToolBar(JToolBar.HORIZONTAL);
toolBar.setFloatable(false);
toolBar.setRollover(true);
JButton tempButton = toolBar.add(newAction);
tempButton.setIcon(newIcon);
tempButton.setToolTipText("Новый файл");
```

```
        tempButton = toolBar.add(openAction);
        tempButton.setIcon(openIcon);
        tempButton.setToolTipText("Открыть файл...");
        tempButton = toolBar.add(saveAction);
        tempButton.setIcon(saveIcon);
        tempButton.setToolTipText("Сохранить файл");
        tempButton = toolBar.add(closeAction);
        tempButton.setIcon(closeIcon);
        tempButton.setToolTipText("Закрыть файл");
        toolBar.addSeparator();
        tempButton = toolBar.add(connectivityAction);
        tempButton.setIcon(connectivityIcon);
        tempButton.setToolTipText("Матрица связности");
        tempButton = toolBar.add(modellingAction);
        tempButton.setIcon(modellingIcon);
        tempButton.setToolTipText("Моделирование");
        tempButton = toolBar.add(editSchemeAction);
        tempButton.setIcon(editSchemeIcon);
        tempButton.setToolTipText("Редактировать схему");
        add(toolBar, BorderLayout.NORTH);
        desktopPane = new JDesktopPane();
        add(desktopPane);
        fileChooser = new JFileChooser(".");
        fileChooser.setAcceptAllFileFilterUsed(true);
        fileChooser.addChoosableFileFilter(new XMLFileFilter());
        fileChooser.setMultiSelectionEnabled(false);
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    }

    private class MainFrameListener extends WindowAdapter {

        @Override
        public void windowClosing(WindowEvent e) {
            Program.writeConfigurationFile(e.getWindow().getBounds());
        }

    }

    private class NewAction extends AbstractAction {

        public void actionPerformed(ActionEvent e) {
            SchemeDialog schemeDialog = new SchemeDialog(getX() + ((getWidth() - SchemeDialog.WIDTH) / 2),
                    getY() + ((getHeight() - SchemeDialog.HEIGHT) / 2), "Новая схема");
            schemeDialog.setVisible(true);
            if (schemeDialog.isOkClicked()) {
                InternalFrame internalFrame = new InternalFrame(new Rectangle(0, 0, InternalFrame.MIN_WIDTH, InternalFrame.MIN_HEIGHT),
                        schemeDialog.getSchemeSize(), schemeDialog.getBackgroundColor(), schemeDialog.getStandartColor(),
                        schemeDialog.getSelectedColor(), schemeDialog.getTextColor(), schemeDialog.getTextFont());
                desktopPane.add(internalFrame);
                desktopPane.setSelectedFrame(internalFrame);
                desktopPane.getDesktopManager().activateFrame(internalFrame);
                desktopPane.revalidate();
                desktopPane.repaint();
            }
        }

    }

    private class OpenAction extends AbstractAction {

        public void actionPerformed(ActionEvent e) {
            int answer = fileChooser.showOpenDialog(rootPane);
            if (answer == JFileChooser.APPROVE_OPTION) {
                try {
                    Scanner scanner = new Scanner(new FileReader(fileChooser.getSelectedFile()));
                    StringBuilder builder = new StringBuilder();
                    while (scanner.hasNextLine()) {
                        builder.append(scanner.nextLine());
                    }
                    scanner.close();
                    InternalFrame internalFrame = new InternalFrame(new Rectangle(0, 0, InternalFrame.MIN_WIDTH, InternalFrame.MIN_HEIGHT),
                            fileChooser.getSelectedFile(), SchemeModel.fromXML(builder.toString()));
                    desktopPane.add(internalFrame);
                    desktopPane.setSelectedFrame(internalFrame);
                    desktopPane.getDesktopManager().activateFrame(internalFrame);
                    desktopPane.revalidate();
                    desktopPane.repaint();
                } catch (FileNotFoundException e1) {
                    JOptionPane.showMessageDialog(rootPane, "Выбранный файл не был найден.", "Ошибка!",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }

    private class SaveAction extends AbstractAction {

        public void actionPerformed(ActionEvent e) {
            InternalFrame internalFrame = (InternalFrame) desktopPane.getSelectedFrame();
            if (internalFrame.getFile() == null) {
                saveAsAction.actionPerformed(e);
            } else {
                if (internalFrame.isModified()) {
                    try {
                        PrintWriter writer = new PrintWriter(new FileWriter(internalFrame.getFile()));
                        writer.write(internalFrame.getXMLScheme());
                        writer.close();
                    } catch (IOException e1) {
                        JOptionPane.showMessageDialog(rootPane, "Произошла ошибка при сохранении схемы.", "Ошибка!",
                                JOptionPane.ERROR_MESSAGE);
                    }
                }
```

```
        }
    }

}

private class SaveAsAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        int answer = fileChooser.showSaveDialog(rootPane);
        if (answer == JFileChooser.APPROVE_OPTION) {
            if (!fileChooser.getSelectedFile().getName().toLowerCase().endsWith(XMLFileFilter.XML_EXTENSION)) {
                fileChooser.setSelectedFile(new File(fileChooser.getSelectedFile().getAbsolutePath() +
                        XMLFileFilter.XML_EXTENSION));
            }
            ((InternalFrame) desktopPane.getSelectedFrame()).setFile(fileChooser.getSelectedFile());
            saveAction.actionPerformed(e);
        }
    }

}

private class CloseAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        InternalFrame internalFrame = (InternalFrame) desktopPane.getSelectedFrame();
        if (internalFrame.isModified()) {
            int answer = JOptionPane.showConfirmDialog(rootPane, "Схема \"" + internalFrame.getTitle() +
                    "\" содержит несохраненные изменения. Желаете сохранить их перед закрытием?",
                    "Предупреждение", JOptionPane.YES_NO_OPTION);
            if (answer == JOptionPane.YES_OPTION) {
                saveAction.actionPerformed(e);
            }
        }
        desktopPane.remove(internalFrame);
        desktopPane.revalidate();
        desktopPane.repaint();
    }

}

private class ExitAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        for (JInternalFrame internalFrame : desktopPane.getAllFrames()) {
            desktopPane.setSelectedFrame(internalFrame);
            desktopPane.getDesktopManager().activateFrame(internalFrame);
            closeAction.actionPerformed(e);
        }
        Program.writeConfigurationFile(getBounds());
        System.exit(0);
    }

}

private class ConnectivityAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        JInternalFrame jInternalFrame = desktopPane.getSelectedFrame();
        if (jInternalFrame != null) {
            ((InternalFrame) jInternalFrame).addConnectivityTab();
        }
    }

}

private class ModellingAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        JInternalFrame jInternalFrame = desktopPane.getSelectedFrame();
        if (jInternalFrame != null) {
            ((InternalFrame) jInternalFrame).addModellingTab();
        }
    }

}

private class EditSchemeAction extends AbstractAction {

    public void actionPerformed(ActionEvent e) {
        JInternalFrame jInternalFrame = desktopPane.getSelectedFrame();
        if (jInternalFrame != null) {
            ((InternalFrame) jInternalFrame).editScheme();
        }
    }

}

private class AboutAction extends AbstractAction {

    private MainFrame frame;

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

    public void actionPerformed(ActionEvent e) {
        final JDialog aboutDialog = new JDialog(frame);
        aboutDialog.setTitle("О программе");
        aboutDialog.setModal(true);
        aboutDialog.setLayout(new GridLayout(5, 1));
        aboutDialog.add(new JLabel("Курсовой проект по курсу АПКС"));
        aboutDialog.add(new JLabel("Версия: 1.0"));
        aboutDialog.add(new JLabel("Автор: Захожий Игорь"));
```

```
            aboutDialog.add(new JLabel("Copyright (c) 2011"));
            JButton closeAboutButton = new JButton(new AbstractAction() {
                public void actionPerformed(ActionEvent e) {
                    aboutDialog.setVisible(false);
                }
            });
            closeAboutButton.setText("Закрыть");
            aboutDialog.add(closeAboutButton);
            aboutDialog.pack();
            aboutDialog.setLocation((frame.getX() + (frame.getWidth() - aboutDialog.getWidth()) / 2),
                    (frame.getY() + (frame.getHeight() - aboutDialog.getHeight()) / 2));
            aboutDialog.setResizable(false);
            aboutDialog.setVisible(true);
        }

    }

    private class XMLFileFilter extends FileFilter {

        public static final String XML_EXTENSION = ".xml";

        private static final String XML_DESCRIPTION = "XML файл";

        public XMLFileFilter() {
            super();
        }

        @Override
        public boolean accept(File f) {
            if ((f.getName().toLowerCase().endsWith(XML_EXTENSION)) || (f.isDirectory())) {
                return true;
            }
            return false;
        }

        @Override
        public String getDescription() {
            return XML_DESCRIPTION;
        }

    }

}

package zak.adcs.gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 08.03.11
 *         Time: 12:53
 */
public class SchemeDialog extends JDialog {

    public static final int WIDTH = 300;
    public static final int HEIGHT = 300;

    private static final int MIN_SIZE_VALUE = 400;
    private static final int MAX_SIZE_VALUE = 4000;
    private static final int DEFAULT_SCHEME_WIDTH = 1600;
    private static final int DEFAULT_SCHEME_HEIGHT = 1200;

    private static final Color DEFAULT_BACKGROUND_COLOR = Color.WHITE;
    private static final Color DEFAULT_STANDART_COLOR = Color.CYAN;
    private static final Color DEFAULT_SELECTED_COLOR = Color.PINK;
    private static final Color DEFAULT_TEXT_COLOR = Color.BLACK;

    private static final Font DEFAULT_FONT = new Font("Sans Serif", Font.PLAIN, 12);

    private static final String COLOR_PREVIEW_TEXT = "        ";

    private JLabel sizeLabel;
    private JLabel xLabel;
    private JSpinner xSpinner;
    private JSpinner ySpinner;
    private JLabel backgroundLabel;
    private JLabel bColorLabel;
    private JButton bColorButton;
    private JLabel standartLabel;
    private JLabel stColorLabel;
    private JButton stColorButton;
    private JLabel selectedLabel;
    private JLabel slColorLabel;
    private JButton slColorButton;
    private JLabel textLabel;
    private JLabel tColorLabel;
    private JButton tColorButton;
    private JLabel fontLabel;
    private JComboBox fontBox;
    private JButton okButton;
    private JButton cancelButton;

    private boolean okClicked;

    public SchemeDialog(int x, int y, String title) {
        super();
        setLocation(x, y);
```

```
setTitle(title);
setSize(WIDTH, HEIGHT);
setResizable(false);
setModal(true);
okClicked = false;
sizeLabel = new JLabel("Размер схемы");
xLabel = new JLabel(" x ");
xSpinner = new JSpinner(new SpinnerNumberModel(DEFAULT_SCHEME_WIDTH, MIN_SIZE_VALUE, MAX_SIZE_VALUE, 1));
ySpinner = new JSpinner(new SpinnerNumberModel(DEFAULT_SCHEME_HEIGHT, MIN_SIZE_VALUE, MAX_SIZE_VALUE, 1));
backgroundLabel = new JLabel("Фоновый цвет");
bColorLabel = new JLabel(COLOR_PREVIEW_TEXT);
bColorLabel.setOpaque(true);
bColorLabel.setBackground(DEFAULT_BACKGROUND_COLOR);
standartLabel = new JLabel("Цвет элементов");
stColorLabel = new JLabel(COLOR_PREVIEW_TEXT);
stColorLabel.setOpaque(true);
stColorLabel.setBackground(DEFAULT_STANDART_COLOR);
selectedLabel = new JLabel("Цвет выделения");
slColorLabel = new JLabel(COLOR_PREVIEW_TEXT);
slColorLabel.setOpaque(true);
slColorLabel.setBackground(DEFAULT_SELECTED_COLOR);
textLabel = new JLabel("Цвет текста");
tColorLabel = new JLabel(COLOR_PREVIEW_TEXT);
tColorLabel.setOpaque(true);
tColorLabel.setBackground(DEFAULT_TEXT_COLOR);
bColorButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        Color result = JColorChooser.showDialog(rootPane, backgroundLabel.getText(), bColorLabel.getBackground());
        if (result != null) {
            bColorLabel.setBackground(result);
        }
    }
});
bColorButton.setText("Изменить...");
stColorButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        Color result = JColorChooser.showDialog(rootPane, standartLabel.getText(), stColorLabel.getBackground());
        if (result != null) {
            stColorLabel.setBackground(result);
        }
    }
});
stColorButton.setText("Изменить...");
slColorButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        Color result = JColorChooser.showDialog(rootPane, selectedLabel.getText(), slColorLabel.getBackground());
        if (result != null) {
            slColorLabel.setBackground(result);
        }
    }
});
slColorButton.setText("Изменить...");
tColorButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        Color result = JColorChooser.showDialog(rootPane, textLabel.getText(), tColorLabel.getBackground());
        if (result != null) {
            tColorLabel.setBackground(result);
        }
    }
});
tColorButton.setText("Изменить...");
fontLabel = new JLabel("Шрифт");
fontBox = new JComboBox();
fontBox.setEditable(false);
fontBox.addItem("Sans Serif");
fontBox.addItem("Serif");
fontBox.addItem("Monospaced");
fontBox.addItem("Dialog");
fontBox.addItem("Dialog Input");
fontBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        fontBox.setFont(new Font((String) fontBox.getSelectedItem(), Font.PLAIN, 12));
        repaint();
    }
});
fontBox.setSelectedIndex(0);
fontBox.setFont(DEFAULT_FONT);
okButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        okClicked = true;
        setVisible(false);
    }
});
okButton.setText("Ok");
cancelButton = new JButton(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        okClicked = false;
        setVisible(false);
    }
});
cancelButton.setText("Отмена");
Box sizeBox = Box.createHorizontalBox();
sizeBox.add(Box.createHorizontalStrut(10));
sizeBox.add(sizeLabel);
sizeBox.add(Box.createHorizontalStrut(10));
sizeBox.add(xSpinner);
sizeBox.add(Box.createHorizontalStrut(5));
sizeBox.add(xLabel);
sizeBox.add(Box.createHorizontalStrut(5));
sizeBox.add(ySpinner);
sizeBox.add(Box.createHorizontalStrut(10));
Box bColorBox = Box.createHorizontalBox();
bColorBox.add(Box.createHorizontalStrut(10));
```

```
        bColorBox.add(backgroundLabel);
        bColorBox.add(Box.createHorizontalStrut(10));
        bColorBox.add(bColorLabel);
        bColorBox.add(Box.createHorizontalStrut(10));
        bColorBox.add(bColorButton);
        bColorBox.add(Box.createHorizontalStrut(10));
        Box stColorBox = Box.createHorizontalBox();
        stColorBox.add(Box.createHorizontalStrut(10));
        stColorBox.add(standartLabel);
        stColorBox.add(Box.createHorizontalStrut(10));
        stColorBox.add(stColorLabel);
        stColorBox.add(Box.createHorizontalStrut(10));
        stColorBox.add(stColorButton);
        stColorBox.add(Box.createHorizontalStrut(10));
        Box slColorBox = Box.createHorizontalBox();
        slColorBox.add(Box.createHorizontalStrut(10));
        slColorBox.add(selectedLabel);
        slColorBox.add(Box.createHorizontalStrut(10));
        slColorBox.add(slColorLabel);
        slColorBox.add(Box.createHorizontalStrut(10));
        slColorBox.add(slColorButton);
        slColorBox.add(Box.createHorizontalStrut(10));
        Box tColorBox = Box.createHorizontalBox();
        tColorBox.add(Box.createHorizontalStrut(10));
        tColorBox.add(textLabel);
        tColorBox.add(Box.createHorizontalStrut(10));
        tColorBox.add(tColorLabel);
        tColorBox.add(Box.createHorizontalStrut(10));
        tColorBox.add(tColorButton);
        tColorBox.add(Box.createHorizontalStrut(10));
        Box fBox = Box.createHorizontalBox();
        fBox.add(Box.createHorizontalStrut(10));
        fBox.add(fontLabel);
        fBox.add(Box.createHorizontalStrut(10));
        fBox.add(fontBox);
        fBox.add(Box.createHorizontalStrut(10));
        Box buttonsBox = Box.createHorizontalBox();
        buttonsBox.add(Box.createHorizontalStrut(10));
        buttonsBox.add(okButton);
        buttonsBox.add(Box.createHorizontalStrut(10));
        buttonsBox.add(cancelButton);
        buttonsBox.add(Box.createHorizontalStrut(10));
        Box vBox = Box.createVerticalBox();
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(sizeBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(bColorBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(stColorBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(slColorBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(tColorBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(fBox);
        vBox.add(Box.createVerticalStrut(10));
        vBox.add(buttonsBox);
        vBox.add(Box.createVerticalStrut(10));
        add(vBox);
    }

    public boolean isOkClicked() {
        return okClicked;
    }

    public Dimension getSchemeSize() {
        return new Dimension((Integer) xSpinner.getValue(), (Integer) ySpinner.getValue());
    }

    public Color getBackgroundColor() {
        return bColorLabel.getBackground();
    }

    public Color getStandartColor() {
        return stColorLabel.getBackground();
    }

    public Color getSelectedColor() {
        return slColorLabel.getBackground();
    }

    public Color getTextColor() {
        return tColorLabel.getBackground();
    }

    public Font getTextFont() {
        return fontBox.getFont();
    }

    //TODO Maybe to code methods for editing scheme options

}

package zak.adcs.gui;

import zak.adcs.ImageIconLoader;
import zak.adcs.logicscheme.editor.SchemeEditor;
import zak.adcs.logicscheme.editor.SchemeModel;
import zak.adcs.logicscheme.modelling.ModellingPanel;

import javax.swing.*;
import java.awt.*;
import java.io.File;
```

```java
/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 01.03.11
 *         Time: 5:08
 */
public class InternalFrame extends JInternalFrame {

    public static final int MIN_WIDTH = 880;
    public static final int MIN_HEIGHT = 660;

    private static final String NEW_SCHEME_NAME = "Новая схема ";

    private static int NEW_SCHEME_COUNTER = 1;

    private JTabbedPane tabbedPane;
    private SchemeEditor schemeEditor;

    private int connectivityTabIndex;
    private int modellingTabIndex;

    private File file;

    public InternalFrame(Rectangle bounds, Dimension schemeSize, Color schemeBackColor, Color schemeStandColor,
                         Color schemeSelColor, Color textColor, Font font) {
        super(NEW_SCHEME_NAME + String.valueOf(NEW_SCHEME_COUNTER++), true, false, true, true);
        setBounds(bounds);
        setMinimumSize(new Dimension(MIN_WIDTH, MIN_HEIGHT));
        setFrameIcon(ImageIconLoader.getInstance().getImageIcon("/img/document.png"));
        file = null;
        connectivityTabIndex = -1;
        modellingTabIndex = -1;
        tabbedPane = new JTabbedPane();
        schemeEditor = new SchemeEditor(schemeSize, schemeBackColor, schemeStandColor, schemeSelColor, textColor, font,
                rootPane);
        tabbedPane.addTab("Редактор схемы", schemeEditor);
        add(tabbedPane);
        reshape(getX(), getY(), getWidth(), getHeight());
        setVisible(true);
    }

    public InternalFrame(Rectangle bounds, File file, SchemeModel schemeModel) {
        super(file.getAbsolutePath(), true, false, true, true);
        setBounds(bounds);
        setMinimumSize(new Dimension(MIN_WIDTH, MIN_HEIGHT));
        setFrameIcon(ImageIconLoader.getInstance().getImageIcon("/img/document.png"));
        this.file = file;
        connectivityTabIndex = -1;
        modellingTabIndex = -1;
        tabbedPane = new JTabbedPane();
        schemeEditor = new SchemeEditor(schemeModel, rootPane);
        tabbedPane.addTab("Редактор схемы", schemeEditor);
        add(tabbedPane);
        reshape(getX(), getY(), getWidth(), getHeight());
        setVisible(true);
    }

    public void addConnectivityTab() {
        if (connectivityTabIndex == -1) {
            JTable table = new JTable(schemeEditor.getSchemeTableModel());
            table.setTableHeader(null);
            table.setDragEnabled(false);
            tabbedPane.addTab("Матрица связности", new JScrollPane(table));
            connectivityTabIndex = tabbedPane.getTabCount() - 1;
        }
        schemeEditor.setPanelEnableToModify(false);
        tabbedPane.setSelectedIndex(connectivityTabIndex);
    }

    public void addModellingTab() {
        if (modellingTabIndex == -1) {
            tabbedPane.addTab("Моделирование", new ModellingPanel(schemeEditor.getSchemeTableModel(),
                    schemeEditor.getBackgroundColor(), schemeEditor.getStandartColor(), schemeEditor.getTextColor(),
                    schemeEditor.getTextFont()));
            modellingTabIndex = tabbedPane.getTabCount() - 1;
        }
        schemeEditor.setPanelEnableToModify(false);
        tabbedPane.setSelectedIndex(modellingTabIndex);
    }

    public void editScheme() {
        for (int i = tabbedPane.getTabCount() - 1; i > 0; i--) {
            tabbedPane.removeTabAt(i);
        }
        connectivityTabIndex = -1;
        modellingTabIndex = -1;
        schemeEditor.setPanelEnableToModify(true);
    }

    public boolean isModified() {
        return schemeEditor.isModified();
    }

    public File getFile() {
        return file;
    }

    public void setFile(File file) {
        this.file = file;
        setTitle(file.getAbsolutePath());
    }
```

```
    public String getXMLScheme() {
        return schemeEditor.getXMLScheme();
    }

}

package zak.adcs.logicscheme.editor;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 12.03.11
 *         Time: 14:24
 */
public class IOElementDialog extends JDialog {

    public static final int WIDTH = 300;
    public static final int HEIGHT = 65;

    private JLabel nameLabel;
    private JTextField nameField;
    private JButton okButton;
    private JButton cancelButton;

    private boolean okClicked;

    public IOElementDialog(Point location, String title) {
        super();
        setLocation(location);
        setSize(WIDTH, HEIGHT);
        setResizable(false);
        setTitle(title);
        setModal(true);
        okClicked = false;
        nameLabel = new JLabel("Имя элемента");
        nameField = new JTextField();
        okButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                if (nameField.getText().length() == 0) {
                    JOptionPane.showMessageDialog(rootPane, "Вы не ввели имя элемента.", "Ошибка!",
                            JOptionPane.ERROR_MESSAGE);
                } else {
                    okClicked = true;
                    setVisible(false);
                }
            }
        });
        okButton.setText("Ok");
        cancelButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        cancelButton.setText("Отмена");
        setLayout(new GridLayout(2, 2));
        add(nameLabel);
        add(nameField);
        add(okButton);
        add(cancelButton);
    }

    public String getName() {
        return nameField.getText();
    }

    public boolean isOkClicked() {
        return okClicked;
    }

    public void setName(String name) {
        nameField.setText(name);
    }

}

package zak.adcs.logicscheme.editor;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 12.03.11
 *         Time: 14:24
 */
public class LIDElementDialog extends JDialog {

    public static final int WIDTH = 300;
    public static final int HEIGHT = 130;

    private JLabel nameLabel;
    private JTextField nameField;
    private JLabel inertiaLabel;
    private JSpinner inertiaSpinner;
```

```
    private JLabel dynamicLabel;
    private JSpinner dynamicSpinner;
    private JLabel inCountLabel;
    private JSpinner inCountSpinner;
    private JButton okButton;
    private JButton cancelButton;

    private boolean okClicked;

    public LIDElementDialog(Point location, String title) {
        super();
        setLocation(location);
        setSize(WIDTH, HEIGHT);
        setResizable(false);
        setTitle(title);
        setModal(true);
        okClicked = false;
        nameLabel = new JLabel("Имя элемента");
        nameField = new JTextField();
        inertiaLabel = new JLabel("Инерционная задержка");
        inertiaSpinner = new JSpinner(new SpinnerNumberModel(0, 0, Integer.MAX_VALUE, 1));
        dynamicLabel = new JLabel("Динамическая задержка");
        dynamicSpinner = new JSpinner(new SpinnerNumberModel(0, 0, Integer.MAX_VALUE, 1));
        inCountLabel = new JLabel("Количество входов");
        inCountSpinner = new JSpinner(new SpinnerNumberModel(2, 2, Integer.MAX_VALUE, 1));
        okButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                okClicked = true;
                setVisible(false);
            }
        });
        okButton.setText("Ok");
        cancelButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        cancelButton.setText("Отмена");
        setLayout(new GridLayout(5, 2));
        add(nameLabel);
        add(nameField);
        add(inertiaLabel);
        add(inertiaSpinner);
        add(dynamicLabel);
        add(dynamicSpinner);
        add(inCountLabel);
        add(inCountSpinner);
        add(okButton);
        add(cancelButton);
    }

    public String getName() {
        return nameField.getText();
    }

    public int getInertiaDelay() {
        return (Integer) inertiaSpinner.getValue();
    }

    public int getDynamicDelay() {
        return (Integer) dynamicSpinner.getValue();
    }

    public int getInCount() {
        return (Integer) inCountSpinner.getValue();
    }

    public boolean isOkClicked() {
        return okClicked;
    }

    public void setInertiaDelay(int inertiaDelay) {
        inertiaSpinner.setValue(inertiaDelay);
    }

    public void setDynamicDelay(int dynamicDelay) {
        dynamicSpinner.setValue(dynamicDelay);
    }

    public void setInCount(int inCount) {
        inCountSpinner.setValue(inCount);
    }

    public void setName(String name) {
        nameField.setText(name);
    }

}

package zak.adcs.logicscheme.editor;

import zak.adcs.ImageIconLoader;
import zak.adcs.logicscheme.modelling.SchemeTableModel;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
```

```
 *         Time: 13:06
 */
public class SchemeEditor extends JPanel {

    private SchemePanel panel;
    private JToolBar toolBar;
    private JLabel statusLabel;
    private Font textFont;

    private SchemeEditor() {
        super();
        toolBar = new JToolBar(JToolBar.VERTICAL);
        toolBar.setFloatable(false);
        toolBar.setRollover(true);
        ImageIconLoader iconLoader = ImageIconLoader.getInstance();
        JButton tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.MOVING);
                statusLabel.setText("Перемещение элементов");
            }
        });
        tempButton.setToolTipText("Перемещение элементов");
        tempButton.setIcon(iconLoader.getImageIcon("/img/cursor.png"));
        toolBar.addSeparator();
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_IN);
                statusLabel.setText("Добавление входов");
            }
        });
        tempButton.setToolTipText("Добавление входов");
        tempButton.setIcon(iconLoader.getImageIcon("/img/in_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_OUT);
                statusLabel.setText("Добавление выходов");
            }
        });
        tempButton.setToolTipText("Добавление выходов");
        tempButton.setIcon(iconLoader.getImageIcon("/img/out_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_AND);
                statusLabel.setText("Добавление элементов И");
            }
        });
        tempButton.setToolTipText("Добавление элементов И");
        tempButton.setIcon(iconLoader.getImageIcon("/img/and_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_OR);
                statusLabel.setText("Добавление элементов ИЛИ");
            }
        });
        tempButton.setToolTipText("Добавление элементов ИЛИ");
        tempButton.setIcon(iconLoader.getImageIcon("/img/or_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_NAND);
                statusLabel.setText("Добавление элементов И-НЕ");
            }
        });
        tempButton.setToolTipText("Добавление элементов И-НЕ");
        tempButton.setIcon(iconLoader.getImageIcon("/img/nand_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_NOR);
                statusLabel.setText("Добавление элементов ИЛИ-НЕ");
            }
        });
        tempButton.setToolTipText("Добавление элементов ИЛИ-НЕ");
        tempButton.setIcon(iconLoader.getImageIcon("/img/nor_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_XOR);
                statusLabel.setText("Добавление элементов Исключительное ИЛИ");
            }
        });
        tempButton.setToolTipText("Добавление элементов Исключительное ИЛИ");
        tempButton.setIcon(iconLoader.getImageIcon("/img/xor_element.png"));
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.ADDING_CONN);
                statusLabel.setText("Добавление соединений");
            }
        });
        tempButton.setToolTipText("Добавление соединений");
        tempButton.setIcon(iconLoader.getImageIcon("/img/conn_element.png"));
        toolBar.addSeparator();
        tempButton = toolBar.add(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                panel.getModel().setCurrentAction(SchemeModel.CurrentAction.DELETING);
                statusLabel.setText("Удаление эелентов и соединений");
            }
        });
        tempButton.setToolTipText("Удаление эелентов и соединений");
        tempButton.setIcon(iconLoader.getImageIcon("/img/delete_element.png"));
        statusLabel = new JLabel("Перемещение элементов");
    }

    public SchemeEditor(Dimension panelSize, Color panelColor, Color standartColor, Color selectedColor, Color textColor,
                        Font font, JRootPane rootFrame) {
        this();
```

```
        panel = new SchemePanel(new SchemeModel(panelSize, panelColor, standartColor, selectedColor, textColor, font),
                rootFrame);
        setLayout(new BorderLayout());
        add(statusLabel, BorderLayout.SOUTH);
        add(toolBar, BorderLayout.WEST);
        add(new JScrollPane(panel));
    }

    public SchemeEditor(SchemeModel schemeModel, JRootPane rootFrame) {
        this();
        panel = new SchemePanel(schemeModel, rootFrame);
        setLayout(new BorderLayout());
        add(statusLabel, BorderLayout.SOUTH);
        add(toolBar, BorderLayout.WEST);
        add(new JScrollPane(panel));
    }

    public SchemeTableModel getSchemeTableModel() {
        return panel.getModel().getSchemeTableModel();
    }

    public Color getStandartColor() {
        return panel.getModel().getStandartColor();
    }

    public Color getBackgroundColor() {
        return panel.getModel().getBackgroundColor();
    }

    public boolean isModified() {
        return panel.getModel().isModified();
    }

    public String getXMLScheme() {
        return panel.getModel().toXML();
    }

    public void setPanelEnableToModify(boolean enable) {
        panel.setEnableToModify(enable);
    }

    public Font getTextFont() {
        return panel.getModel().getFont();
    }

    public Color getTextColor() {
        return panel.getModel().getTextColor();
    }

}

package zak.adcs.logicscheme.editor;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.annotations.Annotations;
import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamOmitField;
import zak.adcs.logicscheme.elements.*;
import zak.adcs.logicscheme.modelling.LogicFunctionsWorker;
import zak.adcs.logicscheme.modelling.SchemeTableModel;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 4:33
 */
@XStreamAlias("schememodel")
public class SchemeModel {

    public enum CurrentAction {MOVING, ADDING_IN, ADDING_OUT, ADDING_AND, ADDING_OR, ADDING_NAND, ADDING_NOR, ADDING_XOR, ADDING_CONN,
DELETING}

    public static final int DEFAULT_LOGIC_ELEMENT_WIDTH = 50;
    public static final int DEFAULT_LOGIC_ELEMENT_HEIGHT = 60;
    public static final int DEFAULT_IO_ELEMENT_WIDTH = 60;
    public static final int DEFAULT_IO_ELEMENT_HEIGHT = 25;
    public static final int DEFAULT_ELEMENT_IN_DISTANCE = 20;

    private Dimension size;

    @XStreamAlias("backgroundcolor")
    private Color backgroundColor;
    @XStreamAlias("standartcolor")
    private Color standartColor;
    @XStreamAlias("selectedcolor")
    private Color selectedColor;
    @XStreamAlias("textcolor")
    private Color textColor;

    private Font font;

    private ArrayList<VisualElement> elements;

    @XStreamOmitField
    private boolean isModified;
    @XStreamOmitField
    private CurrentAction currentAction;
```

```
public SchemeModel(Dimension size, Color backgroundColor, Color standartColor, Color selectedColor, Color textColor,
                   Font font) {
    this.size = size;
    this.backgroundColor = backgroundColor;
    this.standartColor = standartColor;
    this.selectedColor = selectedColor;
    this.textColor = textColor;
    this.font = font;
    elements = new ArrayList<VisualElement>();
    isModified = true;
    currentAction = SchemeModel.CurrentAction.MOVING;
}

private static XStream getXStream() {
    XStream xStream = new XStream();
    Annotations.configureAliases(xStream, SchemeModel.class);
    Annotations.configureAliases(xStream, AndElement.class);
    Annotations.configureAliases(xStream, OrElement.class);
    Annotations.configureAliases(xStream, NandElement.class);
    Annotations.configureAliases(xStream, NorElement.class);
    Annotations.configureAliases(xStream, XorElement.class);
    Annotations.configureAliases(xStream, ConnectionElement.class);
    return xStream;
}

public static SchemeModel fromXML(String xml) {
    XStream xStream = getXStream();
    return (SchemeModel) xStream.fromXML(xml);
}

public Color getBackgroundColor() {
    return backgroundColor;
}

public void setBackgroundColor(Color backgroundColor) {
    this.backgroundColor = backgroundColor;
}

public Color getStandartColor() {
    return standartColor;
}

public void setStandartColor(Color standartColor) {
    this.standartColor = standartColor;
}

public Color getSelectedColor() {
    return selectedColor;
}

public void setSelectedColor(Color selectedColor) {
    this.selectedColor = selectedColor;
}

public Dimension getSize() {
    return size;
}

public ArrayList<VisualElement> getElements() {
    return elements;
}

public void addElement(VisualElement element) {
    elements.add(element);
}

public Font getFont() {
    return font;
}

public void setFont(Font font) {
    this.font = font;
}

public Color getTextColor() {
    return textColor;
}

public void setSize(Dimension size) {
    this.size = size;
}

public void setTextColor(Color textColor) {
    this.textColor = textColor;
}

public String toXML() {
    XStream xStream = getXStream();
    return xStream.toXML(this);
}

public boolean isModified() {
    return isModified;
}

public void setModified(boolean modified) {
    isModified = modified;
}

public CurrentAction getCurrentAction() {
    return currentAction;
}
```

```
public void setCurrentAction(CurrentAction currentAction) {
    this.currentAction = currentAction;
}

public void removeConnection(int number) {
    ConnectionElement connectionElement = null;
    for (VisualElement visualElement : elements) {
        if (visualElement instanceof ConnectionElement && ((ConnectionElement) visualElement).getNumber() == number) {
            connectionElement = (ConnectionElement) visualElement;
            break;
        }
    }
    if (connectionElement != null) {
        for (VisualElement visualElement : elements) {
            if (visualElement instanceof Enterable) {
                ((Enterable) visualElement).removeEnterConnectionName(connectionElement.getElementFrom());
            }
            if (visualElement instanceof Exitable) {
                ((Exitable) visualElement).removeExitConnectionName(connectionElement.getElementTo());
            }
        }
        for (ArrayList<Integer> pointConnections : connectionElement.getConnectionElements()) {
            if (pointConnections != null) {
                for (int subNumber : pointConnections) {
                    removeConnection(subNumber);
                }
            }
        }
        elements.remove(connectionElement);
    }
}

public void removePointConnections(int number, int pointIndex) {
    ConnectionElement connectionElement = null;
    for (VisualElement visualElement : elements) {
        if (visualElement instanceof ConnectionElement && ((ConnectionElement) visualElement).getNumber() == number) {
            connectionElement = (ConnectionElement) visualElement;
            break;
        }
    }
    if (connectionElement != null) {
        ArrayList<Integer> connectionsNumbers = connectionElement.getConnectionElements().get(pointIndex);
        if (connectionsNumbers != null) {
            for (int pointConnection : connectionsNumbers) {
                removeConnection(pointConnection);
            }
        }
    }
}

public boolean containsName(String name, VisualElement visualElement) {
    for (VisualElement element : elements) {
        if (element.getClass().getName() == "zak.adcs.logicscheme.elements.InElement") {
            if ((((InElement) element).getName().compareTo(name) == 0) && (visualElement != element)) {
                return true;
            }
        } else if (element.getClass().getName() == "zak.adcs.logicscheme.elements.OutElement") {
            if ((((OutElement) element).getName().compareTo(name) == 0) && (visualElement != element)) {
                return true;
            }
        } else if (element.getClass().getName() != "zak.adcs.logicscheme.elements.ConnectionElement") {
            if ((((LogicElement) element).getName().compareTo(name) == 0) && (visualElement != element)) {
                return true;
            }
        }
    }
    return false;
}

public SchemeTableModel getSchemeTableModel() {
    int inCount = 0;
    int elementCount = 0;
    int outCount = 0;
    for (VisualElement element : elements) {
        if (element instanceof InElement) {
            inCount++;
        } else if (element instanceof OutElement) {
            outCount++;
        } else if (!(element instanceof ConnectionElement)) {
            elementCount++;
        }
    }
    String[] inNames = new String[inCount];
    String[] elementNames = new String[elementCount];
    LogicFunctionsWorker.LogicFunction[] logicFunctions = new LogicFunctionsWorker.LogicFunction[elementCount];
    int[] inertiaDelay = new int[elementCount];
    int[] dynamicDelay = new int[elementCount];
    String[] outNames = new String[outCount];
    int matrixSize = inCount + elementCount + outCount;
    int[][] connectivityMatrix = new int[matrixSize][];
    for (int i = 0; i < connectivityMatrix.length; i++) {
        connectivityMatrix[i] = new int[matrixSize];
        for (int j = 0; j < connectivityMatrix[i].length; j++) {
            connectivityMatrix[i][j] = 0;
        }
    }
    int inIndex = 0;
    int elementIndex = 0;
    int outIndex = 0;
    for (int i = 0; i < elements.size(); i++) {
        VisualElement element = elements.get(i);
        if (element instanceof InElement) {
            inNames[inIndex++] = ((InElement) element).getName();
```

```
            } else if (element instanceof OutElement) {
                outNames[outIndex++] = ((OutElement) element).getName();
            } else if (!(element instanceof ConnectionElement)) {
                elementNames[elementIndex] = ((LogicElement) element).getName();
                inertiaDelay[elementIndex] = ((LogicElement) element).getInertiaDelay();
                dynamicDelay[elementIndex] = ((LogicElement) element).getDynamicDelay();
                if (element instanceof AndElement && !(element instanceof NandElement)) {
                    logicFunctions[elementIndex++] = LogicFunctionsWorker.LogicFunction.AND;
                } else if (element instanceof OrElement && !(element instanceof NorElement)) {
                    logicFunctions[elementIndex++] = LogicFunctionsWorker.LogicFunction.OR;
                } else if (element instanceof NandElement) {
                    logicFunctions[elementIndex++] = LogicFunctionsWorker.LogicFunction.NAND;
                } else if (element instanceof NorElement) {
                    logicFunctions[elementIndex++] = LogicFunctionsWorker.LogicFunction.NOR;
                } else if (element instanceof XorElement) {
                    logicFunctions[elementIndex++] = LogicFunctionsWorker.LogicFunction.XOR;
                }
            }
        }
        for (VisualElement element : elements) {
            if (element instanceof ConnectionElement) {
                ConnectionElement connectionElement = (ConnectionElement) element;
                boolean foundFrom = false;
                int indexFrom = -1;
                for (int i = 0; i < inNames.length; i++) {
                    indexFrom++;
                    if (inNames[i].compareTo(connectionElement.getElementFrom()) == 0) {
                        foundFrom = true;
                        break;
                    }
                }
                if (!foundFrom) {
                    for (int i = 0; i < elementNames.length; i++) {
                        indexFrom++;
                        if (elementNames[i].compareTo(connectionElement.getElementFrom()) == 0) {
                            foundFrom = true;
                            break;
                        }
                    }
                    if (!foundFrom) {
                        for (int i = 0; i < outNames.length; i++) {
                            indexFrom++;
                            if (outNames[i].compareTo(connectionElement.getElementFrom()) == 0) {
                                foundFrom = true;
                                break;
                            }
                        }
                    }
                }
                boolean foundTo = false;
                int indexTo = -1;
                for (int i = 0; i < inNames.length; i++) {
                    indexTo++;
                    if (inNames[i].compareTo(connectionElement.getElementTo()) == 0) {
                        foundTo = true;
                        break;
                    }
                }
                if (!foundTo) {
                    for (int i = 0; i < elementNames.length; i++) {
                        indexTo++;
                        if (elementNames[i].compareTo(connectionElement.getElementTo()) == 0) {
                            foundTo = true;
                            break;
                        }
                    }
                    if (!foundTo) {
                        for (int i = 0; i < outNames.length; i++) {
                            indexTo++;
                            if (outNames[i].compareTo(connectionElement.getElementTo()) == 0) {
                                foundTo = true;
                                break;
                            }
                        }
                    }
                }
                connectivityMatrix[indexFrom][indexTo]++;
            }
        }
        return new SchemeTableModel(inNames, elementNames, logicFunctions, inertiaDelay, dynamicDelay, outNames,
                connectivityMatrix);
    }

}

package zak.adcs.logicscheme.editor;

import zak.adcs.logicscheme.elements.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 4:32
 */
```

```
public class SchemePanel extends JPanel implements Scrollable {

    private JRootPane rootFrame;

    private SchemeModel model;

    public SchemePanel(SchemeModel model, JRootPane rootFrame) {
        super();
        this.model = model;
        this.rootFrame = rootFrame;
        setSize(this.model.getSize());
        MouseHandler mouseHandler = new MouseHandler();
        addMouseListener(mouseHandler);
        addMouseMotionListener(mouseHandler);
    }

    public SchemeModel getModel() {
        return model;
    }

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

    public void setEnableToModify(boolean enable) {
        if (enable) {
            if (getMouseListeners().length == 0 && getMouseMotionListeners().length == 0) {
                MouseHandler mouseHandler = new MouseHandler();
                addMouseListener(mouseHandler);
                addMouseMotionListener(mouseHandler);
            }
        } else {
            if (!(getMouseListeners().length == 0 && getMouseMotionListeners().length == 0)) {
                removeMouseListener(getMouseListeners()[0]);
                removeMouseMotionListener(getMouseMotionListeners()[0]);
            }
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        setBackground(model.getBackgroundColor());
        if (model.getElements() != null) {
            for (VisualElement ve : model.getElements()) {
                ve.draw(g2);
            }
        }
    }

    @Override
    public Dimension getPreferredSize() {
        return model.getSize();
    }

    public Dimension getPreferredScrollableViewportSize() {
        return getPreferredSize();
    }

    public int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction) {
        return 1;
    }

    public int getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int direction) {
        return 10;
    }

    public boolean getScrollableTracksViewportWidth() {
        return false;
    }

    public boolean getScrollableTracksViewportHeight() {
        return false;
    }

    private class MouseHandler extends MouseAdapter {

        private int oldX;
        private int oldY;
        HashMap<ConnectionElement, Point> oldPoints;

        private Point lastCursorPoint;

        private ClosedElement movingElement;

        private int movingPointIndex;
        private ConnectionElement movingPointElement;
        private ArrayList<ConnectionElement> movingPointConnections;

        private int connectionElementPointCanConnectIndex;

        private ConnectionElement addingConnectionElement;
        private ConnectionElement connectionElementToConnect;
        private ConnectionElement connectionElementCanConnect;
        private ClosedElement elementToConnect;
        private ClosedElement elementCanConnect;

        private boolean enterFlag;

        @Override
        public void mouseClicked(MouseEvent e) {
            if (e.getButton() == MouseEvent.BUTTON1) {
```

```
SchemeModel.CurrentAction currentAction = model.getCurrentAction();
if (currentAction == SchemeModel.CurrentAction.MOVING) {
    if (e.getClickCount() > 1) {
        VisualElement inElement = null;
        for (VisualElement element : model.getElements()) {
            if (element.contains(e.getPoint())) {
                inElement = element;
                break;
            }
        }
        if (inElement != null) {
            inElement.setSelected(true);
            repaint();
            if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.InElement") {
                IOElementDialog dialog = new IOElementDialog(e.getLocationOnScreen(), "Вход");
                dialog.setName(((InElement) inElement).getName());
                dialog.setVisible(true);
                if (dialog.isOkClicked() && dialog.getName() != (((InElement) inElement).getName())) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        ((InElement) inElement).setName(dialog.getName());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.OutElement") {
                IOElementDialog dialog = new IOElementDialog(e.getLocationOnScreen(), "Выход");
                dialog.setName(((OutElement) inElement).getName());
                dialog.setVisible(true);
                if (dialog.isOkClicked() && dialog.getName() != (((OutElement) inElement).getName())) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        ((OutElement) inElement).setName(dialog.getName());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.AndElement") {
                LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "И");
                AndElement andElement = (AndElement) inElement;
                dialog.setName(andElement.getName());
                dialog.setInertiaDelay(andElement.getInertiaDelay());
                dialog.setDynamicDelay(andElement.getDynamicDelay());
                dialog.setInCount(andElement.getInCount());
                dialog.setVisible(true);
                if (dialog.isOkClicked()) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        andElement.setName(dialog.getName());
                        andElement.setInertiaDelay(dialog.getInertiaDelay());
                        andElement.setDynamicDelay(dialog.getDynamicDelay());
                        andElement.setInCount(dialog.getInCount());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.OrElement") {
                LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "ИЛИ");
                OrElement orElement = (OrElement) inElement;
                dialog.setName(orElement.getName());
                dialog.setInertiaDelay(orElement.getInertiaDelay());
                dialog.setDynamicDelay(orElement.getDynamicDelay());
                dialog.setInCount(orElement.getInCount());
                dialog.setVisible(true);
                if (dialog.isOkClicked()) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        orElement.setName(dialog.getName());
                        orElement.setInertiaDelay(dialog.getInertiaDelay());
                        orElement.setDynamicDelay(dialog.getDynamicDelay());
                        orElement.setInCount(dialog.getInCount());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.XorElement") {
                LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Искл. ИЛИ");
                XorElement xorElement = (XorElement) inElement;
                dialog.setName(xorElement.getName());
                dialog.setInertiaDelay(xorElement.getInertiaDelay());
                dialog.setDynamicDelay(xorElement.getDynamicDelay());
                dialog.setInCount(xorElement.getInCount());
                dialog.setVisible(true);
                if (dialog.isOkClicked()) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        xorElement.setName(dialog.getName());
                        xorElement.setInertiaDelay(dialog.getInertiaDelay());
                        xorElement.setDynamicDelay(dialog.getDynamicDelay());
                        xorElement.setInCount(dialog.getInCount());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.NandElement") {
                LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "И-НЕ");
                NandElement nandElement = (NandElement) inElement;
                dialog.setName(nandElement.getName());
                dialog.setInertiaDelay(nandElement.getInertiaDelay());
                dialog.setDynamicDelay(nandElement.getDynamicDelay());
                dialog.setInCount(nandElement.getInCount());
                dialog.setVisible(true);
                if (dialog.isOkClicked()) {
```

```
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        nandElement.setName(dialog.getName());
                        nandElement.setInertiaDelay(dialog.getInertiaDelay());
                        nandElement.setDynamicDelay(dialog.getDynamicDelay());
                        nandElement.setInCount(dialog.getInCount());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.NorElement") {
                LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "ИЛИ-НЕ");
                NorElement norElement = (NorElement) inElement;
                dialog.setName(norElement.getName());
                dialog.setInertiaDelay(norElement.getInertiaDelay());
                dialog.setDynamicDelay(norElement.getDynamicDelay());
                dialog.setInCount(norElement.getInCount());
                dialog.setVisible(true);
                if (dialog.isOkClicked()) {
                    if (model.containsName(dialog.getName(), inElement)) {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    } else {
                        norElement.setName(dialog.getName());
                        norElement.setInertiaDelay(dialog.getInertiaDelay());
                        norElement.setDynamicDelay(dialog.getDynamicDelay());
                        norElement.setInCount(dialog.getInCount());
                    }
                }
            } else if (inElement.getClass().getName() == "zak.adcs.logicscheme.elements.ConnectionElement") {
                ConnectionElement connectionElement = (ConnectionElement) inElement;
                connectionElement.addPoint(e.getPoint());
            }
            inElement.setSelected(false);
            repaint();
        }
    }
} else if (currentAction == SchemeModel.CurrentAction.ADDING_IN) {
    Rectangle bounds = new Rectangle(e.getX(), e.getY(),
            SchemeModel.DEFAULT_IO_ELEMENT_WIDTH, SchemeModel.DEFAULT_IO_ELEMENT_HEIGHT);
    boolean canAdd = true;
    if (model.getElements() != null) {
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
    }
    if (canAdd) {
        IOElementDialog dialog = new IOElementDialog(e.getLocationOnScreen(), "Добавление входа");
        dialog.setVisible(true);
        if (dialog.isOkClicked()) {
            if (model.containsName(dialog.getName(), null)) {
                JOptionPane.showMessageDialog(rootFrame,
                        "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
            } else {
                InElement inElement = new InElement((int) bounds.getX(), (int) bounds.getY(),
                        (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                        model.getSelectedColor(), model.getTextColor(), model.getFont(), dialog.getName());
                model.getElements().add(inElement);
                repaint();
                model.setModified(true);
            }
        }
    } else {
        JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                JOptionPane.ERROR_MESSAGE);
    }
} else if (currentAction == SchemeModel.CurrentAction.ADDING_OUT) {
    Rectangle bounds = new Rectangle(e.getX(), e.getY(),
            SchemeModel.DEFAULT_IO_ELEMENT_WIDTH, SchemeModel.DEFAULT_IO_ELEMENT_HEIGHT);
    boolean canAdd = true;
    if (model.getElements() != null) {
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
    }
    if (canAdd) {
        IOElementDialog dialog = new IOElementDialog(e.getLocationOnScreen(), "Добавление выхода");
        dialog.setVisible(true);
        if (dialog.isOkClicked()) {
            if (model.containsName(dialog.getName(), null)) {
                JOptionPane.showMessageDialog(rootFrame,
                        "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
            } else {
                OutElement outElement = new OutElement((int) bounds.getX(), (int) bounds.getY(),
                        (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                        model.getSelectedColor(), model.getTextColor(), model.getFont(), dialog.getName());
                model.getElements().add(outElement);
                repaint();
                model.setModified(true);
            }
        }
    } else {
        JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                JOptionPane.ERROR_MESSAGE);
    }
} else if (currentAction == SchemeModel.CurrentAction.ADDING_AND) {
    Rectangle bounds = new Rectangle(e.getX(), e.getY(), SchemeModel.DEFAULT_LOGIC_ELEMENT_WIDTH,
```

```
                SchemeModel.DEFAULT_LOGIC_ELEMENT_HEIGHT);
        boolean canAdd = true;
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
        if (canAdd) {
            LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Добавление элемента И");
            dialog.setVisible(true);
            if (dialog.isOkClicked()) {
                if (model.containsName(dialog.getName(), null)) {
                    JOptionPane.showMessageDialog(rootFrame,
                            "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (dialog.getInCount() > 2) {
                        bounds.setSize(SchemeModel.DEFAULT_IO_ELEMENT_WIDTH,
                                SchemeModel.DEFAULT_ELEMENT_IN_DISTANCE * (dialog.getInCount() + 1));
                        for (VisualElement element : model.getElements()) {
                            if (element.isOverlaped(bounds)) {
                                canAdd = false;
                                break;
                            }
                        }
                    }
                    if (canAdd) {
                        AndElement andElement = new AndElement((int) bounds.getX(), (int) bounds.getY(),
                                (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                                model.getSelectedColor(), model.getTextColor(), model.getFont(),
                                dialog.getName(), dialog.getInertiaDelay(), dialog.getDynamicDelay(),
                                dialog.getInCount());
                        model.getElements().add(andElement);
                        repaint();
                        model.setModified(true);
                    } else {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
    } else if (currentAction == SchemeModel.CurrentAction.ADDING_OR) {
        Rectangle bounds = new Rectangle(e.getX(), e.getY(), SchemeModel.DEFAULT_LOGIC_ELEMENT_WIDTH,
                SchemeModel.DEFAULT_LOGIC_ELEMENT_HEIGHT);
        boolean canAdd = true;
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
        if (canAdd) {
            LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Добавление элемента ИЛИ");
            dialog.setVisible(true);
            if (dialog.isOkClicked()) {
                if (model.containsName(dialog.getName(), null)) {
                    JOptionPane.showMessageDialog(rootFrame,
                            "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (dialog.getInCount() > 2) {
                        bounds.setSize(SchemeModel.DEFAULT_IO_ELEMENT_WIDTH,
                                SchemeModel.DEFAULT_ELEMENT_IN_DISTANCE * (dialog.getInCount() + 1));
                        for (VisualElement element : model.getElements()) {
                            if (element.isOverlaped(bounds)) {
                                canAdd = false;
                                break;
                            }
                        }
                    }
                    if (canAdd) {
                        OrElement orElement = new OrElement((int) bounds.getX(), (int) bounds.getY(),
                                (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                                model.getSelectedColor(), model.getTextColor(), model.getFont(),
                                dialog.getName(), dialog.getInertiaDelay(), dialog.getDynamicDelay(),
                                dialog.getInCount());
                        model.getElements().add(orElement);
                        repaint();
                        model.setModified(true);
                    } else {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
    } else if (currentAction == SchemeModel.CurrentAction.ADDING_NAND) {
        Rectangle bounds = new Rectangle(e.getX(), e.getY(), SchemeModel.DEFAULT_LOGIC_ELEMENT_WIDTH,
                SchemeModel.DEFAULT_LOGIC_ELEMENT_HEIGHT);
        boolean canAdd = true;
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
        if (canAdd) {
```

```
            LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Добавление элемента И-НЕ");
            dialog.setVisible(true);
            if (dialog.isOkClicked()) {
                if (model.containsName(dialog.getName(), null)) {
                    JOptionPane.showMessageDialog(rootFrame,
                            "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (dialog.getInCount() > 2) {
                        bounds.setSize(SchemeModel.DEFAULT_IO_ELEMENT_WIDTH,
                                SchemeModel.DEFAULT_ELEMENT_IN_DISTANCE * (dialog.getInCount() + 1));
                        for (VisualElement element : model.getElements()) {
                            if (element.isOverlaped(bounds)) {
                                canAdd = false;
                                break;
                            }
                        }
                    }
                    if (canAdd) {
                        NandElement nandElement = new NandElement((int) bounds.getX(), (int) bounds.getY(),
                                (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                                model.getSelectedColor(), model.getTextColor(), model.getFont(),
                                dialog.getName(), dialog.getInertiaDelay(), dialog.getDynamicDelay(),
                                dialog.getInCount());
                        model.getElements().add(nandElement);
                        repaint();
                        model.setModified(true);
                    } else {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
    } else if (currentAction == SchemeModel.CurrentAction.ADDING_NOR) {
        Rectangle bounds = new Rectangle(e.getX(), e.getY(), SchemeModel.DEFAULT_LOGIC_ELEMENT_WIDTH,
                SchemeModel.DEFAULT_LOGIC_ELEMENT_HEIGHT);
        boolean canAdd = true;
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
        if (canAdd) {
            LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Добавление элемента ИЛИ-НЕ");
            dialog.setVisible(true);
            if (dialog.isOkClicked()) {
                if (model.containsName(dialog.getName(), null)) {
                    JOptionPane.showMessageDialog(rootFrame,
                            "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (dialog.getInCount() > 2) {
                        bounds.setSize(SchemeModel.DEFAULT_IO_ELEMENT_WIDTH,
                                SchemeModel.DEFAULT_ELEMENT_IN_DISTANCE * (dialog.getInCount() + 1));
                        for (VisualElement element : model.getElements()) {
                            if (element.isOverlaped(bounds)) {
                                canAdd = false;
                                break;
                            }
                        }
                    }
                    if (canAdd) {
                        NorElement norElement = new NorElement((int) bounds.getX(), (int) bounds.getY(),
                                (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                                model.getSelectedColor(), model.getTextColor(), model.getFont(),
                                dialog.getName(), dialog.getInertiaDelay(), dialog.getDynamicDelay(),
                                dialog.getInCount());
                        model.getElements().add(norElement);
                        repaint();
                        model.setModified(true);
                    } else {
                        JOptionPane.showMessageDialog(rootFrame,
                                "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                    JOptionPane.ERROR_MESSAGE);
        }
    } else if (currentAction == SchemeModel.CurrentAction.ADDING_XOR) {
        Rectangle bounds = new Rectangle(e.getX(), e.getY(), SchemeModel.DEFAULT_LOGIC_ELEMENT_WIDTH,
                SchemeModel.DEFAULT_LOGIC_ELEMENT_HEIGHT);
        boolean canAdd = true;
        for (VisualElement element : model.getElements()) {
            if (element.isOverlaped(bounds)) {
                canAdd = false;
                break;
            }
        }
        if (canAdd) {
            LIDElementDialog dialog = new LIDElementDialog(e.getLocationOnScreen(), "Добавление элемента Искл. ИЛИ");
            dialog.setVisible(true);
            if (dialog.isOkClicked()) {
                if (model.containsName(dialog.getName(), null)) {
                    JOptionPane.showMessageDialog(rootFrame,
                            "Элемент с данным именем уже существует.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (dialog.getInCount() > 2) {
                        bounds.setSize(SchemeModel.DEFAULT_IO_ELEMENT_WIDTH,
```

```
                                        SchemeModel.DEFAULT_ELEMENT_IN_DISTANCE * (dialog.getInCount() + 1));
                                for (VisualElement element : model.getElements()) {
                                    if (element.isOverlaped(bounds)) {
                                        canAdd = false;
                                        break;
                                    }
                                }
                            }
                            if (canAdd) {
                                XorElement xorElement = new XorElement((int) bounds.getX(), (int) bounds.getY(),
                                        (int) bounds.getWidth(), (int) bounds.getHeight(), model.getStandartColor(),
                                        model.getSelectedColor(), model.getTextColor(), model.getFont(),
                                        dialog.getName(), dialog.getInertiaDelay(), dialog.getDynamicDelay(),
                                        dialog.getInCount());
                                model.getElements().add(xorElement);
                                repaint();
                                model.setModified(true);
                            } else {
                                JOptionPane.showMessageDialog(rootFrame,
                                        "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                            }
                        }
                    }
                } else {
                    JOptionPane.showMessageDialog(rootFrame, "Недопустимое расположение элемента.", "Ошибка!",
                            JOptionPane.ERROR_MESSAGE);
                }
            } else if (currentAction == SchemeModel.CurrentAction.DELETING) {
                VisualElement inElement = null;
                for (VisualElement element : model.getElements()) {
                    if (element.contains(e.getPoint())) {
                        inElement = element;
                        break;
                    }
                }
                if (inElement != null) {
                    boolean connectionPointFlag = false;
                    if (inElement instanceof ConnectionElement) {
                        ConnectionElement connectionElement = (ConnectionElement) inElement;
                        int pointIndex = connectionElement.getInPointIndex(e.getPoint());
                        if (pointIndex > -1) {
                            connectionPointFlag = true;
                            connectionElement.setSelectedPoint(pointIndex, true);
                            repaint();
                            int answer = JOptionPane.showConfirmDialog(rootFrame,
                                    "Вы уверены что хотите удалить выбранный элемент?", "Подтверждение",
                                    JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
                            if (answer == JOptionPane.YES_OPTION) {
                                model.removePointConnections(connectionElement.getNumber(), pointIndex);
                                connectionElement.removePoint(pointIndex);
                                model.setModified(true);
                            } else {
                                connectionElement.setSelectedPoint(pointIndex, false);
                            }
                        } else {
                            connectionPointFlag = true;
                            connectionElement.setSelected(true);
                            repaint();
                            int answer = JOptionPane.showConfirmDialog(rootFrame,
                                    "Вы уверены что хотите удалить выбранный элемент?", "Подтверждение",
                                    JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
                            if (answer == JOptionPane.YES_OPTION) {
                                model.removeConnection(connectionElement.getNumber());
                                model.setModified(true);
                            } else {
                                connectionElement.setSelected(false);
                            }
                        }
                    }
                    if (!connectionPointFlag) {
                        inElement.setSelected(true);
                        repaint();
                        int answer = JOptionPane.showConfirmDialog(rootFrame,
                                "Вы уверены что хотите удалить выбранный элемент?", "Подтверждение",
                                JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
                        if (answer == JOptionPane.YES_OPTION) {
                            model.getElements().remove(inElement);
                            if (inElement instanceof ClosedElement) {
                                ClosedElement closedElement = (ClosedElement) inElement;
                                for (int i = model.getElements().size() - 1; i >= 0; i--) {
                                    if (model.getElements().get(i) instanceof ConnectionElement) {
                                        if (((ConnectionElement)
model.getElements().get(i)).getElementTo().compareTo(closedElement.getName()) == 0 ||
                                                ((ConnectionElement)
model.getElements().get(i)).getElementFrom().compareTo(closedElement.getName()) == 0) {
                                            ConnectionElement connectionElement = (ConnectionElement) model.getElements().get(i);
                                            model.removeConnection(connectionElement.getNumber());
                                        }
                                    }
                                }
                            }
                            model.setModified(true);
                        } else {
                            inElement.setSelected(false);
                        }
                    }
                    repaint();
                }
            }
        }
    }

    @Override
```

```
public void mousePressed(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON1) {
        SchemeModel.CurrentAction currentAction = model.getCurrentAction();
        if (currentAction == SchemeModel.CurrentAction.MOVING) {
            VisualElement inElement = null;
            for (VisualElement element : model.getElements()) {
                if (element.getClass().getName() != "zak.adcs.logicscheme.elements.ConnectionElement" &&
                        element.contains(e.getPoint())) {
                    inElement = element;
                    break;
                } else if (element instanceof ConnectionElement && element.contains(e.getPoint())) {
                    ConnectionElement connectionElement = (ConnectionElement) element;
                    if (connectionElement.isInExit(e.getPoint())) {
                        movingPointIndex = connectionElement.getInPointIndex(e.getPoint());
                        movingPointElement = connectionElement;
                        movingPointElement.setSelectedPoint(movingPointIndex, true);
                        repaint();
                        oldX = connectionElement.getPoint(movingPointIndex).x;
                        oldY = connectionElement.getPoint(movingPointIndex).y;
                        lastCursorPoint = e.getPoint();
                        ArrayList<Integer> movingPointConnectionsNumbers =
                                connectionElement.getConnectionNumbers(movingPointIndex);
                        if (movingPointConnectionsNumbers != null && movingPointConnectionsNumbers.size() > 0) {
                            movingPointConnections = new ArrayList<ConnectionElement>();
                            for (int number : movingPointConnectionsNumbers) {
                                for (VisualElement iElement : model.getElements()) {
                                    if (iElement instanceof ConnectionElement) {
                                        if (((ConnectionElement) iElement).getNumber() == number) {
                                            movingPointConnections.add((ConnectionElement) iElement);
                                            break;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            if (inElement != null) {
                movingElement = (ClosedElement) inElement;
                movingElement.setSelected(true);
                repaint();
                oldX = movingElement.getX();
                oldY = movingElement.getY();
                oldPoints = new HashMap<ConnectionElement, Point>();
                int tempCounter = 0;
                for (VisualElement element : model.getElements()) {
                    if (element instanceof ConnectionElement) {
                        ConnectionElement connectionElement = (ConnectionElement) element;
                        if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0) {
                            tempCounter++;
                        }
                    }
                }
                for (VisualElement element : model.getElements()) {
                    if (element instanceof ConnectionElement) {
                        ConnectionElement connectionElement = (ConnectionElement) element;
                        if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0 &&
                                tempCounter < 2) {
                            Point copyPoint = new Point(connectionElement.getFirstPoint());
                            oldPoints.put(connectionElement, copyPoint);
                        } else if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0 &&
                                connectionElement.hasChildrenConnections()) {
                            boolean flag = false;
                            for (VisualElement cElement : model.getElements()) {
                                if (cElement instanceof ConnectionElement) {
                                    ConnectionElement cConnectionElement = (ConnectionElement) cElement;
                                    flag = cConnectionElement.hasChild(connectionElement.getNumber());
                                }
                            }
                            if (!flag) {
                                Point copyPoint = new Point(connectionElement.getFirstPoint());
                                oldPoints.put(connectionElement, copyPoint);
                            }
                        } else if (connectionElement.getElementTo().compareTo(movingElement.getName()) == 0) {
                            Point copyPoint = new Point(connectionElement.getLastPoint());
                            oldPoints.put(connectionElement, copyPoint);
                        }
                    }
                }
                lastCursorPoint = e.getPoint();
            }
        } else if (currentAction == SchemeModel.CurrentAction.ADDING_CONN) {
            Point enterPoint = null;
            Point exitPoint = null;
            VisualElement inElement = null;
            for (VisualElement element : model.getElements()) {
                try {
                    if (element instanceof Enterable) {
                        enterPoint = ((Enterable) element).getEnterPoint(e.getPoint());
                        if (enterPoint != null) {
                            inElement = element;
                            break;
                        }
                    }
                    if (element instanceof Exitable) {
                        exitPoint = ((Exitable) element).getExitPoint(e.getPoint());
                        if (exitPoint != null) {
                            inElement = element;
                            break;
                        }
                    }
                } catch (ConnectionAlreadyExistException e1) {
```

```
                            JOptionPane.showMessageDialog(rootFrame, e1.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    if (inElement instanceof ConnectionElement) {
                        connectionElementToConnect = (ConnectionElement) inElement;
                        connectionElementToConnect.setSelected(true);
                        addingConnectionElement = new ConnectionElement(connectionElementToConnect.getElementFrom(), "",
                                exitPoint, e.getPoint(), model.getStandartColor(), model.getSelectedColor());
                        addingConnectionElement.setSelected(true);
                        model.getElements().add(addingConnectionElement);
                        lastCursorPoint = e.getPoint();
                        enterFlag = false;
                        repaint();
                    } else {
                        if (enterPoint != null) {
                            addingConnectionElement = new ConnectionElement("", ((Enterable) inElement).getName(), e.getPoint(),
                                    enterPoint, model.getStandartColor(), model.getSelectedColor());
                            addingConnectionElement.setSelected(true);
                            model.getElements().add(addingConnectionElement);
                            lastCursorPoint = e.getPoint();
                            elementToConnect = (ClosedElement) inElement;
                            elementToConnect.setSelected(true);
                            elementCanConnect = null;
                            enterFlag = true;
                            repaint();
                        } else if (exitPoint != null) {
                            addingConnectionElement = new ConnectionElement(((Exitable) inElement).getName(), "", exitPoint,
                                    e.getPoint(), model.getStandartColor(), model.getSelectedColor());
                            addingConnectionElement.setSelected(true);
                            model.getElements().add(addingConnectionElement);
                            lastCursorPoint = e.getPoint();
                            elementToConnect = (ClosedElement) inElement;
                            elementToConnect.setSelected(true);
                            elementCanConnect = null;
                            enterFlag = false;
                            repaint();
                        }
                    }
                }
            }
        }

        @Override
        public void mouseReleased(MouseEvent e) {
            if (e.getButton() == MouseEvent.BUTTON1) {
                SchemeModel.CurrentAction currentAction = model.getCurrentAction();
                if (currentAction == SchemeModel.CurrentAction.MOVING) {
                    if (movingElement != null) {
                        boolean canMove = true;
                        for (VisualElement element : model.getElements()) {
                            if (element != movingElement && element.isOverlaped(
                                    new Rectangle(movingElement.getX(), movingElement.getY(), movingElement.getWidth(),
                                            movingElement.getHeight()))) {
                                canMove = false;
                                break;
                            }
                        }
                        if (!canMove) {
                            movingElement.setX(oldX);
                            movingElement.setY(oldY);
                            for (ConnectionElement connectionElement : oldPoints.keySet()) {
                                if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0) {
                                    connectionElement.setFirstPoint(oldPoints.get(connectionElement));
                                } else {
                                    connectionElement.setLastPoint(oldPoints.get(connectionElement));
                                }
                            }
                            JOptionPane.showMessageDialog(rootFrame,
                                    "Недопустимое расположение элемента.", "Ошибка!", JOptionPane.ERROR_MESSAGE);
                        } else {
                            model.setModified(true);
                        }
                        movingElement.setSelected(false);
                        repaint();
                        movingElement = null;
                        oldPoints = null;
                        lastCursorPoint = null;
                    } else if (movingPointElement != null) {
                        movingPointElement.setSelectedPoint(movingPointIndex, false);
                        repaint();
                        model.setModified(true);
                        movingPointElement = null;
                        movingPointIndex = -1;
                        movingPointConnections = null;
                    }
                } else if (currentAction == SchemeModel.CurrentAction.ADDING_CONN) {
                    if (addingConnectionElement != null) {
                        VisualElement inElement = null;
                        Point enterPoint = null;
                        Point exitPoint = null;
                        for (VisualElement element : model.getElements()) {
                            try {
                                if (enterFlag) {
                                    if (element instanceof Exitable && (!(element instanceof ConnectionElement && connectionElementToConnect
!= null))) {
                                        exitPoint = ((Exitable) element).getExitPoint(e.getPoint());
                                        if (exitPoint != null) {
                                            inElement = element;
                                            break;
                                        }
                                    }
                                } else {
                                    if (element instanceof Enterable) {
```

```
                                    enterPoint = ((Enterable) element).getEnterPoint(e.getPoint());
                                    if (enterPoint != null) {
                                        inElement = element;
                                        break;
                                    }
                                }
                            }
                        } catch (ConnectionAlreadyExistException e1) {
                            elementToConnect.setSelected(false);
                            model.getElements().remove(addingConnectionElement);
                            addingConnectionElement = null;
                            lastCursorPoint = null;
                            elementToConnect = null;
                            JOptionPane.showMessageDialog(rootFrame, e1.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    if (connectionElementCanConnect != null) {
                        if (exitPoint != null) {
                            addingConnectionElement.setFirstPoint(exitPoint);
                            addingConnectionElement.setElementFrom(connectionElementCanConnect.getElementFrom());
                            ((Exitable) elementToConnect).addExitConnectionName(connectionElementCanConnect.getElementFrom(),
e.getPoint());
                            connectionElementCanConnect.addConnection(addingConnectionElement, addingConnectionElement.getFirstPoint());
                            connectionElementCanConnect.addExitConnectionName(elementToConnect.getName(), e.getPoint());
                            model.setModified(true);
                        } else {
                            model.getElements().remove(addingConnectionElement);
                        }
                        connectionElementCanConnect.setSelectedPoint(connectionElementPointCanConnectIndex, false);
                        addingConnectionElement.setSelected(false);
                        repaint();
                        addingConnectionElement = null;
                        lastCursorPoint = null;
                        elementToConnect.setSelected(false);
                        elementToConnect = null;
                        connectionElementCanConnect = null;
                        connectionElementPointCanConnectIndex = -1;
                    } else if (connectionElementToConnect != null) {
                        if (enterPoint != null) {
                            addingConnectionElement.setLastPoint(enterPoint);
                            addingConnectionElement.setElementTo(((Enterable) inElement).getName());
                            connectionElementToConnect.addExitConnectionName(((Enterable) inElement).getName(), e.getPoint());
                            connectionElementToConnect.addConnection(addingConnectionElement, addingConnectionElement.getFirstPoint());
                            ((Enterable) inElement).addEnterConnectionName(connectionElementToConnect.getElementFrom(), e.getPoint());
                            model.setModified(true);
                        } else {
                            model.getElements().remove(addingConnectionElement);
                        }
                        connectionElementToConnect.setSelected(false);
                        addingConnectionElement.setSelected(false);
                        if (elementCanConnect != null) {
                            elementCanConnect.setSelected(false);
                        }
                        repaint();
                        addingConnectionElement = null;
                        elementCanConnect = null;
                        lastCursorPoint = null;
                        connectionElementToConnect = null;
                    } else {
                        if (enterPoint != null) {
                            addingConnectionElement.setLastPoint(enterPoint);
                            addingConnectionElement.setElementTo(((Enterable) inElement).getName());
                            ((Exitable) elementToConnect).addExitConnectionName(((Enterable) inElement).getName(), e.getPoint());
                            ((Enterable) inElement).addEnterConnectionName(((Exitable) elementToConnect).getName(), e.getPoint());
                            model.setModified(true);
                        } else if (exitPoint != null) {
                            addingConnectionElement.setFirstPoint(exitPoint);
                            addingConnectionElement.setElementFrom(((Exitable) inElement).getName());
                            ((Exitable) inElement).addExitConnectionName(((Enterable) elementToConnect).getName(), e.getPoint());
                            ((Enterable) elementToConnect).addEnterConnectionName(((Exitable) inElement).getName(), e.getPoint());
                            model.setModified(true);
                        } else {
                            model.getElements().remove(addingConnectionElement);
                        }
                        elementToConnect.setSelected(false);
                        addingConnectionElement.setSelected(false);
                        if (elementCanConnect != null) {
                            elementCanConnect.setSelected(false);
                        }
                        if (connectionElementCanConnect != null) {
                            connectionElementCanConnect.setSelectedPoint(connectionElementPointCanConnectIndex, false);
                            connectionElementCanConnect = null;
                            connectionElementPointCanConnectIndex = -1;
                        }
                        repaint();
                        addingConnectionElement = null;
                        elementCanConnect = null;
                        lastCursorPoint = null;
                        elementToConnect = null;
                    }
                }
            }
        }
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        SchemeModel.CurrentAction currentAction = model.getCurrentAction();
        if (currentAction == SchemeModel.CurrentAction.MOVING) {
            if (movingElement != null) {
                movingElement.move(e.getX() - (int) lastCursorPoint.getX(), e.getY() - (int) lastCursorPoint.getY());
                int tempCounter = 0;
                for (VisualElement element : model.getElements()) {
```

```
                        if (element instanceof ConnectionElement) {
                            ConnectionElement connectionElement = (ConnectionElement) element;
                            if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0) {
                                tempCounter++;
                            }
                        }
                    }
                    for (VisualElement element : model.getElements()) {
                        if (element instanceof ConnectionElement) {
                            ConnectionElement connectionElement = (ConnectionElement) element;
                            if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0 && tempCounter < 2) {
                                connectionElement.moveFirstPoint(e.getX() - (int) lastCursorPoint.getX(),
                                        e.getY() - (int) lastCursorPoint.getY());
                            } else if (connectionElement.getElementFrom().compareTo(movingElement.getName()) == 0 &&
                                    connectionElement.hasChildrenConnections()) {
                                connectionElement.moveFirstPoint(e.getX() - (int) lastCursorPoint.getX(),
                                        e.getY() - (int) lastCursorPoint.getY());
                            }
                            if (connectionElement.getElementTo().compareTo(movingElement.getName()) == 0) {
                                connectionElement.moveLastPoint(e.getX() - (int) lastCursorPoint.getX(),
                                        e.getY() - (int) lastCursorPoint.getY());
                            }
                        }
                    }
                    repaint();
                    lastCursorPoint = e.getPoint();
                } else if (movingPointElement != null) {
                    movingPointElement.movePoint(movingPointIndex, e.getX() - lastCursorPoint.x, e.getY() - lastCursorPoint.y);
                    if (movingPointConnections != null) {
                        for (ConnectionElement ce : movingPointConnections) {
                            ce.manageTemporaryPoints();
                        }
                    }
                    movingPointIndex = movingPointElement.getInPointIndex(e.getPoint());
                    repaint();
                    lastCursorPoint = e.getPoint();
                }
            } else if (currentAction == SchemeModel.CurrentAction.ADDING_CONN) {
                if (addingConnectionElement != null) {
                    if (enterFlag) {
                        addingConnectionElement.moveFirstPoint(e.getPoint().x - lastCursorPoint.x,
                                e.getPoint().y - lastCursorPoint.y);
                        if (elementCanConnect != null) {
                            if (!((Exitable) elementCanConnect).isInExit(e.getPoint())) {
                                elementCanConnect.setSelected(false);
                                elementCanConnect = null;
                            }
                        }
                        if (connectionElementCanConnect != null) {
                            if (!connectionElementCanConnect.isInExit(e.getPoint())) {
                                connectionElementCanConnect.setSelectedPoint(connectionElementPointCanConnectIndex, false);
                                connectionElementCanConnect = null;
                                connectionElementPointCanConnectIndex = -1;
                            }
                        }
                        for (VisualElement element : model.getElements()) {
                            if (element instanceof Exitable) {
                                if (((Exitable) element).isInExit(e.getPoint())) {
                                    if (element instanceof ConnectionElement) {
                                        connectionElementCanConnect = (ConnectionElement) element;
                                        connectionElementPointCanConnectIndex = connectionElementCanConnect.getInPointIndex(e.getPoint());
                                    } else {
                                        elementCanConnect = (ClosedElement) element;
                                    }
                                    break;
                                }
                            }
                        }
                    } else {
                        addingConnectionElement.moveLastPoint(e.getPoint().x - lastCursorPoint.x,
                                e.getPoint().y - lastCursorPoint.y);
                        if (elementCanConnect != null) {
                            if (!((Enterable) elementCanConnect).isInEnter(e.getPoint())) {
                                elementCanConnect.setSelected(false);
                                elementCanConnect = null;
                            }
                        }
                        for (VisualElement element : model.getElements()) {
                            if (element instanceof Enterable) {
                                if (((Enterable) element).isInEnter(e.getPoint())) {
                                    elementCanConnect = (ClosedElement) element;
                                    break;
                                }
                            }
                        }
                    }
                    if (elementCanConnect != null) {
                        elementCanConnect.setSelected(true);
                    }
                    if (connectionElementCanConnect != null) {
                        connectionElementCanConnect.setSelectedPoint(connectionElementPointCanConnectIndex, true);
                    }
                    repaint();
                    lastCursorPoint = e.getPoint();
                }
            }
        }

    }

}

package zak.adcs.logicscheme.elements;
```

```
import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 0:22
 */
@XStreamAlias("andelement")
public class AndElement extends LogicElement {

    private static final String AND_TEXT = "&";

    public AndElement(int x, int y, int width, int height, Color standartColor, Color selectedColor,
                      Color textColor, Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,
                inCount);
    }

    @Override
    public void draw(Graphics2D g2) {
        super.draw(g2);
        g2.setColor(textColor);
        g2.setFont(font);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(AND_TEXT, context);
        g2.drawString(AND_TEXT, x + width / 6 + (int) ((width / 3 * 2 - stringBounds.getWidth()) / 2),
                (int) (y + stringBounds.getHeight()));
    }

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamAsAttribute;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 05.03.11
 *         Time: 13:10
 */
public abstract class ClosedElement implements VisualElement {

    @XStreamAsAttribute
    protected int x;
    @XStreamAsAttribute
    protected int y;
    @XStreamAsAttribute
    protected int width;
    @XStreamAsAttribute
    protected int height;

    @XStreamAlias("standartcolor")
    protected Color standartColor;
    @XStreamAlias("selectedcolor")
    protected Color selectedColor;
    @XStreamAlias("textcolor")
    protected Color textColor;
    protected Color color;

    protected Font font;

    protected String name;

    protected ClosedElement(int x, int y, int width, int height, Color standartColor, Color selectedColor,
                            Color textColor, Font font, String name) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.standartColor = standartColor;
        this.selectedColor = selectedColor;
        this.textColor = textColor;
        this.font = font;
        this.name = name;
        color = standartColor;
    }

    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 getStandartColor() {
        return standartColor;
    }

    public void setStandartColor(Color standartColor) {
        this.standartColor = standartColor;
    }

    public Color getSelectedColor() {
        return selectedColor;
    }

    public void setSelectedColor(Color selectedColor) {
        this.selectedColor = selectedColor;
    }

    public Color getTextColor() {
        return textColor;
    }

    public void setTextColor(Color textColor) {
        this.textColor = textColor;
    }

    public Font getFont() {
        return font;
    }

    public void setFont(Font font) {
        this.font = font;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void move(int x, int y) {
        this.x += x;
        this.y += y;
    }

    public void setSelected(boolean selected) {
        if (selected) {
            color = selectedColor;
        } else {
            color = standartColor;
        }
    }

    public boolean contains(Point p) {
        Rectangle r = new Rectangle(x, y, width, height);
        return r.contains(p);
    }

    public boolean isOverlaped(Rectangle r) {
        if ((r.contains(new Point(x, y))) || (r.contains(new Point(x + width, y))) ||
                (r.contains(new Point(x, y + height))) || (r.contains(new Point(x + width, y + height)))) {
            return true;
        }
        Rectangle bounds = new Rectangle(x, y, width, height);
        if ((bounds.contains(new Point((int) r.getX(), (int) r.getY()))) ||
                (bounds.contains(new Point((int) (r.getX() + r.getWidth()), (int) r.getY()))) ||
                (bounds.contains(new Point((int) r.getX(), (int) (r.getY() + r.getHeight())))) ||
                (bounds.contains(new Point((int) (r.getX() + r.getWidth()), (int) (r.getY() + r.getHeight()))))) {
            return true;
        }
        return false;
    }

}

package zak.adcs.logicscheme.elements;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 16.03.11
 *         Time: 5:41
 */
public class ConnectionAlreadyExistException extends Exception {
```

```
    private static final String ENTER = "Вход";
    private static final String EXIT = "Выход";
    private static final String MESSAGE = " уже занят другим соединением.";

    private boolean isEnter;

    public ConnectionAlreadyExistException(boolean isEnter) {
        this.isEnter = isEnter;
    }

    @Override
    public String getMessage() {
        StringBuilder builder = new StringBuilder();
        if (isEnter) {
            builder.append(ENTER);
        } else {
            builder.append(EXIT);
        }
        builder.append(MESSAGE);
        return builder.toString();
    }

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

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;
import com.thoughtworks.xstream.annotations.XStreamAsAttribute;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 1:13
 */
@XStreamAlias("connectionelement")
public class ConnectionElement implements VisualElement, Exitable {

    private static final int POINT_DIAMETER = 8;
    private static final int HALF_RECTANGLE = 5;

    private static final int DEFAULT_NUMBER = 1;

    @XStreamAlias("nextnumber")
    private static int nextNumber;

    @XStreamAsAttribute
    @XStreamAlias("number")
    private int number;

    @XStreamAlias("standartcolor")
    private Color standartColor;
    @XStreamAlias("selectedcolor")
    private Color selectedColor;
    private Color color;

    @XStreamAlias("elementfrom")
    private String elementFrom;
    @XStreamAlias("elementto")
    private String elementTo;
    private ArrayList<Point> points;
    @XStreamAlias("ispointtemporary")
    private ArrayList<Boolean> isPointTemporary;
    @XStreamAlias("ispointselected")
    private ArrayList<Boolean> isPointSelected;
    @XStreamAlias("connectionelements")
    private ArrayList<ArrayList<Integer>> connectionElements;

    public ConnectionElement(String elementFrom, String elementTo, Point pointFrom, Point pointTo,
                             Color standartColor, Color selectedColor) {
        this.elementFrom = elementFrom;
        this.elementTo = elementTo;
        this.standartColor = standartColor;
        this.selectedColor = selectedColor;
        if (nextNumber == 0) {
            nextNumber = DEFAULT_NUMBER + 1;
        }
        number = nextNumber++;
        color = standartColor;
        points = new ArrayList<Point>();
        isPointTemporary = new ArrayList<Boolean>();
        isPointSelected = new ArrayList<Boolean>();
        connectionElements = new ArrayList<ArrayList<Integer>>();
        points.add(pointFrom);
        isPointTemporary.add(false);
        isPointSelected.add(false);
        connectionElements.add(null);
        points.add(pointTo);
        isPointTemporary.add(false);
        isPointSelected.add(false);
        connectionElements.add(null);
        manageTemporaryPoints();
```

```
}

public void manageTemporaryPoints() {
    for (int i = points.size() - 2; i > 0; i--) {
        if (isPointTemporary.get(i)) {
            points.remove(i);
            isPointTemporary.remove(i);
            isPointSelected.remove(i);
            connectionElements.remove(i);
        }
    }
    for (int i = 0; i < points.size() - 1; i++) {
        if (points.get(i).x != points.get(i + 1).x && points.get(i).y != points.get(i + 1).y) {
            if (points.get(i).x < points.get(i + 1).x) {
                points.add(i + 1, new Point(points.get(i).x + (points.get(i + 1).x - points.get(i).x) / 2, points.get(i).y));
                points.add(i + 2, new Point(points.get(i).x + (points.get(i + 2).x - points.get(i).x) / 2, points.get(i + 2).y));
            } else {
                points.add(i + 1, new Point(points.get(i).x - (points.get(i).x - points.get(i + 1).x) / 2, points.get(i).y));
                points.add(i + 2, new Point(points.get(i).x - (points.get(i).x - points.get(i + 2).x) / 2, points.get(i + 2).y));
            }
            isPointTemporary.add(i + 1, true);
            isPointSelected.add(i + 1, false);
            connectionElements.add(i + 1, null);
            isPointTemporary.add(i + 2, true);
            isPointSelected.add(i + 2, false);
            connectionElements.add(i + 2, null);
            i += 2;
        }
    }
}

public void addConnectionToPoint(ConnectionElement connectionElement, Point point) {
    int pointIndex = getInPointIndex(point);
    connectionElements.get(pointIndex).add(connectionElement.getNumber());
}

public String getElementFrom() {
    return elementFrom;
}

public String getElementTo() {
    return elementTo;
}

public void setElementFrom(String elementFrom) {
    this.elementFrom = elementFrom;
}

public void setElementTo(String elementTo) {
    this.elementTo = elementTo;
}

public void addPoint(Point p, Point afterPoint) {
    points.add(points.indexOf(afterPoint) + 1, p);
    isPointTemporary.add(points.indexOf(afterPoint) + 1, false);
    isPointSelected.add(points.indexOf(afterPoint) + 1, false);
    connectionElements.add(points.indexOf(afterPoint) + 1, new ArrayList<Integer>());
    manageTemporaryPoints();
}

public void addPoint(Point point) {
    int afterNumber = -1;
    for (int i = 0; i < points.size() - 1; i++) {
        Rectangle lineBounds = null;
        if (points.get(i).x == points.get(i + 1).x) {
            if (points.get(i).y < points.get(i + 1).y) {
                lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i).y,
                        2 * HALF_RECTANGLE, points.get(i + 1).y - points.get(i).y);
            } else {
                lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i + 1).y,
                        2 * HALF_RECTANGLE, points.get(i).y - points.get(i + 1).y);
            }
        } else if (points.get(i).y == points.get(i + 1).y) {
            if (points.get(i).x < points.get(i + 1).x) {
                lineBounds = new Rectangle(points.get(i).x, points.get(i).y - HALF_RECTANGLE,
                        points.get(i + 1).x - points.get(i).x, 2 * HALF_RECTANGLE);
            } else {
                lineBounds = new Rectangle(points.get(i + 1).x, points.get(i).y - HALF_RECTANGLE,
                        points.get(i).x - points.get(i + 1).x, 2 * HALF_RECTANGLE);
            }
        }
        if (lineBounds.contains(point)) {
            afterNumber = i;
            break;
        }
    }
    if (afterNumber > -1) {
        addPoint(point, points.get(afterNumber));
    }
}

public int getInPointIndex(Point p) {
    for (int i = 1; i < points.size() - 1; i++) {
        if (!isPointTemporary.get(i)) {
            Ellipse2D.Double ellipse = new Ellipse2D.Double(points.get(i).getX() - POINT_DIAMETER / 2,
                    points.get(i).getY() - POINT_DIAMETER / 2, POINT_DIAMETER, POINT_DIAMETER);
            if (ellipse.contains(p)) {
                return i;
            }
        }
    }
    return -1;
}
```

```
public void movePoint(int index, int x, int y) {
    points.get(index).setLocation(points.get(index).getX() + x, points.get(index).getY() + y);
    manageTemporaryPoints();
}

public void draw(Graphics2D g2) {
    g2.setColor(color);
    for (int i = 0; i < points.size() - 1; i++) {
        g2.drawLine((int) points.get(i).getX(), (int) points.get(i).getY(), (int) points.get(i + 1).getX(),
                (int) points.get(i + 1).getY());
    }
    if (points.size() > 2) {
        for (int i = 1; i < points.size() - 1; i++) {
            if (!isPointTemporary.get(i)) {
                if (isPointSelected.get(i)) {
                    g2.setColor(selectedColor);
                } else {
                    g2.setColor(color);
                }
                g2.fillOval((int) points.get(i).getX() - POINT_DIAMETER / 2,
                        (int) points.get(i).getY() - POINT_DIAMETER / 2,
                        POINT_DIAMETER,
                        POINT_DIAMETER);
            }
        }
    }
}

public void move(int x, int y) {
    for (Point p : points) {
        p.setLocation((int) p.getX() + x, (int) p.getY() + y);
    }
}

public void removePoint(int index) {
    points.remove(index);
    isPointTemporary.remove(index);
    isPointSelected.remove(index);
    manageTemporaryPoints();
}

public void moveFirstPoint(int x, int y) {
    movePoint(0, x, y);
}

public void moveLastPoint(int x, int y) {
    movePoint(points.size() - 1, x, y);
}

public void setFirstPoint(Point point) {
    points.set(0, point);
    manageTemporaryPoints();
}

public void setLastPoint(Point point) {
    points.set(points.size() - 1, point);
    manageTemporaryPoints();
}

public Point getFirstPoint() {
    return points.get(0);
}

public Point getLastPoint() {
    return points.get(points.size() - 1);
}

public Point getPoint(int index) {
    return points.get(index);
}

public void setSelectedPoint(int index, boolean selected) {
    isPointSelected.set(index, selected);
}

public void setSelected(boolean selected) {
    if (selected) {
        color = selectedColor;
    } else {
        color = standartColor;
    }
}

public boolean contains(Point p) {
    for (int i = 0; i < points.size() - 1; i++) {
        Rectangle lineBounds = null;
        if (points.get(i).x == points.get(i + 1).x) {
            if (points.get(i).y < points.get(i + 1).y) {
                lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i).y,
                        2 * HALF_RECTANGLE, points.get(i + 1).y - points.get(i).y);
            } else {
                lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i + 1).y,
                        2 * HALF_RECTANGLE, points.get(i).y - points.get(i + 1).y);
            }
        } else if (points.get(i).y == points.get(i + 1).y) {
            if (points.get(i).x < points.get(i + 1).x) {
                lineBounds = new Rectangle(points.get(i).x, points.get(i).y - HALF_RECTANGLE,
                        points.get(i + 1).x - points.get(i).x, 2 * HALF_RECTANGLE);
            } else {
                lineBounds = new Rectangle(points.get(i + 1).x, points.get(i).y - HALF_RECTANGLE,
                        points.get(i).x - points.get(i + 1).x, 2 * HALF_RECTANGLE);
            }
```

```
            }
            if (lineBounds != null && lineBounds.contains(p)) {
                return true;
            }
        }
        return false;
    }

    public boolean isOverlaped(Rectangle r) {
        for (int i = 0; i < points.size() - 1; i++) {
            Rectangle lineBounds = null;
            if (points.get(i).x == points.get(i + 1).x) {
                if (points.get(i).y < points.get(i + 1).y) {
                    lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i).y,
                            2 * HALF_RECTANGLE, points.get(i + 1).y - points.get(i).y);
                } else {
                    lineBounds = new Rectangle(points.get(i).x - HALF_RECTANGLE, points.get(i + 1).y,
                            2 * HALF_RECTANGLE, points.get(i).y - points.get(i + 1).y);
                }
            } else if (points.get(i).y == points.get(i + 1).y) {
                if (points.get(i).x < points.get(i + 1).x) {
                    lineBounds = new Rectangle(points.get(i).x, points.get(i).y - HALF_RECTANGLE,
                            points.get(i + 1).x - points.get(i).x, 2 * HALF_RECTANGLE);
                } else {
                    lineBounds = new Rectangle(points.get(i + 1).x, points.get(i).y - HALF_RECTANGLE,
                            points.get(i).x - points.get(i + 1).x, 2 * HALF_RECTANGLE);
                }
            }
            if (lineBounds.intersects(r)) {
                return true;
            }
        }
        return false;
    }

    public boolean isInExit(Point point) {
        int pointIndex = getInPointIndex(point);
        if (pointIndex > 0 && pointIndex < points.size() - 1 && !isPointTemporary.get(pointIndex)) {
            return true;
        }
        return false;
    }

    public Point getExitPoint(Point point) throws ConnectionAlreadyExistException {
        if (isInExit(point)) {
            return points.get(getInPointIndex(point));
        }
        return null;
    }

    public String getName() {
        return null;
    }

    public int getNumber() {
        return number;
    }

    public ArrayList<Integer> getConnectionNumbers(int index) {
        return connectionElements.get(index);
    }

    public void addExitConnectionName(String name, Point point) {
    }

    public void removeExitConnectionName(String name) {
    }

    public boolean hasChildrenConnections() {
        for (ArrayList<Integer> e : connectionElements) {
            if (e != null && e.size() > 0) {
                return true;
            }
        }
        return false;
    }

    public boolean hasChild(int number) {
        for (ArrayList<Integer> e : connectionElements) {
            if (e != null && e.size() > 0) {
                for (int n : e) {
                    if (n == number) {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public ArrayList<ArrayList<Integer>> getConnectionElements() {
        return connectionElements;
    }

    public void addConnection(ConnectionElement connectionElement, Point point) {
        int index = getInPointIndex(point);
        if (connectionElements.get(index) == null) {
            connectionElements.add(index, new ArrayList<Integer>());
        }
        connectionElements.get(index).add(connectionElement.getNumber());
    }

}
```

```
package zak.adcs.logicscheme.elements;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 16.03.11
 *         Time: 5:35
 */
public interface Enterable {

    public boolean isInEnter(Point point);
    public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException;
    public String getName();
    public void addEnterConnectionName(String name, Point point);
    public void removeEnterConnectionName(String name);

}
```

```
package zak.adcs.logicscheme.elements;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 16.03.11
 *         Time: 5:36
 */
public interface Exitable {

    public boolean isInExit(Point point);
    public Point getExitPoint(Point point) throws ConnectionAlreadyExistException;
    public String getName();
    public void addExitConnectionName(String name, Point point);
    public void removeExitConnectionName(String name);

}
```

```
package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 05.03.11
 *         Time: 16:15
 */
@XStreamAlias("inelement")
public class InElement extends ClosedElement implements Exitable {

    private static final String IN_TEXT = "IN";

    @XStreamAlias("outname")
    private String outName;

    public InElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                     Font font, String name) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name);
        outName = null;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(color);
        g2.fillRect(x, y, width / 6 * 4, height);
        g2.fillOval(x + width / 6 * 3, y, width / 3, height);
        g2.drawLine(x + width / 6 * 5, y + height / 2, x + width, y + height / 2);
        g2.setFont(font);
        g2.setColor(textColor);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(IN_TEXT, context);
        g2.drawString(IN_TEXT, x + (int) ((width / 6 * 5 - stringBounds.getWidth()) / 2),
                y + (int) ((height - stringBounds.getHeight()) / 2 - stringBounds.getY()));
        stringBounds = font.getStringBounds(name, context);
        g2.drawString(name, x + width / 6 * 5 + (int) ((width / 6 - stringBounds.getWidth()) / 2),
                y + (int) ((height - stringBounds.getHeight() / 2) / 2));
    }

    public String getOutName() {
        return outName;
    }

    public void setOutName(String outName) {
        this.outName = outName;
    }

    public Point getExitPoint(Point point) throws ConnectionAlreadyExistException {
        if ((point.x >= x + width / 6 * 5) && (point.x <= x + width) && (point.y >= y) && (point.y <= y + height)) {
            if (outName != null) {
                throw new ConnectionAlreadyExistException(false);
            }
            return new Point(x + width, y + height / 2);
        } else {
```

```
            return null;
        }
    }

    public void addExitConnectionName(String name, Point point) {
        outName = name;
    }

    public void removeExitConnectionName(String name) {
        outName = null;
    }

    public boolean isInExit(Point point) {
        try {
            if (getExitPoint(point) != null) {
                return true;
            }
            return false;
        } catch (ConnectionAlreadyExistException e) {
            return false;
        }
    }

}
```

```
package zak.adcs.logicscheme.elements;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 12.03.11
 *         Time: 14:26
 */
public interface LIDElement {

    public int getInertiaDelay();
    public int getDynamicDelay();
    public void setInertiaDelay(int inertiaDelay);
    public void setDynamicDelay(int dynamicDelay);

}
```

```
package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zak
 *         Date: 05.03.11
 *         Time: 17:39
 */
public abstract class LogicElement extends ClosedElement implements LIDElement, Enterable, Exitable {

    @XStreamAlias("inertiadelay")
    protected int inertiaDelay;
    @XStreamAlias("dynamicdelay")
    protected int dynamicDelay;

    @XStreamAlias("incount")
    protected int inCount;
    @XStreamAlias("innames")
    protected String[] inNames;
    @XStreamAlias("outnames")
    protected String outName;

    protected LogicElement(int x, int y, int width, int height, Color standartColor, Color selectedColor,
                           Color textColor, Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name);
        this.inertiaDelay = inertiaDelay;
        this.dynamicDelay = dynamicDelay;
        this.inCount = inCount;
        inNames = new String[this.inCount];
    }

    public void draw(Graphics2D g2) {
        g2.setColor(color);
        g2.fillRect(x + width / 6, y, width / 3 * 2, height);
        int inDistance = height / (inCount + 1);
        int inY = y;
        for (int i = 0; i < inCount; i++) {
            inY += inDistance;
            g2.drawLine(x, inY, x + width / 6, inY);
        }
        g2.drawLine(x + width / 6 * 5, y + height / 2, x + width, y + height / 2);
        g2.setColor(textColor);
        g2.setFont(font);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(name, context);
        g2.drawString(name, x + width / 6 + (int) ((width / 3 * 2 - stringBounds.getWidth()) / 2), y + height / 12 * 11);
    }

    public int getInertiaDelay() {
        return inertiaDelay;
    }

    public int getDynamicDelay() {
```

```
        return dynamicDelay;
    }

    public void setInertiaDelay(int inertiaDelay) {
        this.inertiaDelay = inertiaDelay;
    }

    public void setDynamicDelay(int dynamicDelay) {
        this.dynamicDelay = dynamicDelay;
    }

    public int getInCount() {
        return inCount;
    }

    public void setInCount(int inCount) {
        if (inCount < this.inCount) {
            String[] newInNames = new String[inCount];
            for (int i = 0; i < inCount; i++) {
                newInNames[i] = inNames[i];
            }
            inNames = newInNames;
        }
        this.inCount = inCount;
    }

    public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException {
        int enterDistance = height / (inCount + 1);
        if ((point.x >= x) && (point.x <= x + width / 6) &&
                (point.y >= y + enterDistance / 2) && (point.y <= y + height - enterDistance / 2)) {
            int inNumber = (point.y - y - enterDistance / 2) / enterDistance;
            if (inNames[inNumber] != null) {
                throw new ConnectionAlreadyExistException(true);
            }
            return new Point(x, y + enterDistance * (inNumber + 1));
        } else {
            return null;
        }
    }

    public Point getExitPoint(Point point) throws ConnectionAlreadyExistException {
        if ((point.x >= x + width / 6 * 5) && (point.x <= x + width) && (point.y >= y) && (point.y <= y + height)) {
            if (outName != null) {
                throw new ConnectionAlreadyExistException(false);
            }
            return new Point(x + width, y + height / 2);
        } else {
            return null;
        }
    }

    public boolean isInEnter(Point point) {
        try {
            if (getEnterPoint(point) != null) {
                return true;
            }
            return false;
        } catch (ConnectionAlreadyExistException e) {
            return false;
        }
    }

    public boolean isInExit(Point point) {
        try {
            if (getExitPoint(point) != null) {
                return true;
            }
            return false;
        } catch (ConnectionAlreadyExistException e) {
            return false;
        }
    }

    public void addEnterConnectionName(String name, Point point) {
        int enterDistance = height / (inCount + 1);
        if ((point.x >= x) && (point.x <= x + width / 6) &&
                (point.y >= y + enterDistance / 2) && (point.y <= y + height - enterDistance / 2)) {
            int inNumber = (point.y - y - enterDistance / 2) / enterDistance;
            inNames[inNumber] = name;
        }
    }

    public void addExitConnectionName(String name, Point point) {
        outName = name;
    }

    public void removeEnterConnectionName(String name) {
        for (int i = 0; i < inNames.length; i++) {
            if (inNames[i] != null && inNames[i] == name) {
                inNames[i] = null;
            }
        }
    }

    public void removeExitConnectionName(String name) {
        outName = null;
    }

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;
```

```
import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 1:06
 */
@XStreamAlias("nandelement")
public class NandElement extends AndElement {

    public NandElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                       Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,
                inCount);
    }

    @Override
    public void draw(Graphics2D g2) {
        super.draw(g2);
        g2.setColor(textColor);
        g2.drawOval(x + width / 6 * 5 - width / 12, y + height / 2 - width / 12, width / 6, width / 6);
    }
}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 1:12
 */
@XStreamAlias("norelement")
public class NorElement extends OrElement {

    public NorElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                      Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,
                inCount);
    }

    @Override
    public void draw(Graphics2D g2) {
        super.draw(g2);
        g2.setColor(textColor);
        g2.drawOval(x + width / 6 * 5 - width / 12, y + height / 2 - width / 12, width / 6, width / 6);
    }

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 0:32
 */
@XStreamAlias("orelement")
public class OrElement extends LogicElement {

    private static final String OR_TEXT = "1";

    public OrElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                     Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,
                inCount);
    }

    @Override
    public void draw(Graphics2D g2) {
        super.draw(g2);
        g2.setColor(textColor);
        g2.setFont(font);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(OR_TEXT, context);
        g2.drawString(OR_TEXT, x + width / 6 + (int) ((width / 3 * 2 - stringBounds.getWidth()) / 2),
                (int) (y + stringBounds.getHeight()));
    }

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
```

```
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 05.03.11
 *         Time: 16:50
 */
@XStreamAlias("outelement")
public class OutElement extends ClosedElement implements Enterable {

    private static final String OUT_TEXT = "OUT";

    @XStreamAlias("inname")
    private String inName;

    public OutElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                      Font font, String name) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name);
        inName = null;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(color);
        g2.fillRect(x + width / 3, y, width / 6 * 4, height);
        g2.fillOval(x + width / 6, y, width / 3, height);
        g2.drawLine(x, y + height / 2, x + width / 6, y + height / 2);
        g2.setColor(textColor);
        g2.setFont(font);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(OUT_TEXT, context);
        g2.drawString(OUT_TEXT, x + width / 6 + (int) ((width / 6 * 5 - stringBounds.getWidth()) / 2),
                y + (int) ((height - stringBounds.getHeight()) / 2 - stringBounds.getY()));
        stringBounds = font.getStringBounds(name, context);
        g2.drawString(name, x + (int) ((width / 6 - stringBounds.getWidth()) / 2),
                y + (int) ((height - stringBounds.getHeight() / 2) / 2));
    }

    public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException {
        if ((point.x >= x) && (point.x <= x + width / 6) && (point.y >= y) && (point.y <= y + height)) {
            if (inName != null) {
                throw new ConnectionAlreadyExistException(true);
            }
            return new Point(x, y + height / 2);
        } else {
            return null;
        }
    }

    public void addEnterConnectionName(String name, Point point) {
        inName = name;
    }

    public void removeEnterConnectionName(String name) {
        inName = null;
    }

    public boolean isInEnter(Point point) {
        try {
            if (getEnterPoint(point) != null) {
                return true;
            }
            return false;
        } catch (ConnectionAlreadyExistException e) {
            return false;
        }
    }

}

package zak.adcs.logicscheme.elements;

import java.awt.*;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 01.03.11
 *         Time: 5:59
 */
public interface VisualElement {

    public void draw(Graphics2D g2);
    public void move(int x, int y);
    public void setSelected(boolean selected);
    public boolean contains(Point p);
    public boolean isOverlaped(Rectangle r);

}

package zak.adcs.logicscheme.elements;

import com.thoughtworks.xstream.annotations.XStreamAlias;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;

/**
 * Created by IntelliJ IDEA.
 *
```

```
 * @author Zakhozhyy Ihor
 *         Date: 06.03.11
 *         Time: 0:34
 */
@XStreamAlias("xorelement")
public class XorElement extends LogicElement {

    private static final String XOR_TEXT = "=1";

    public XorElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,
                      Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {
        super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,
                inCount);
    }

    @Override
    public void draw(Graphics2D g2) {
        super.draw(g2);
        g2.setColor(textColor);
        g2.setFont(font);
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(XOR_TEXT, context);
        g2.drawString(XOR_TEXT, x + width / 6 + (int) ((width / 3 * 2 - stringBounds.getWidth()) / 2),
                (int) (y + stringBounds.getHeight()));
    }

}

package zak.adcs.logicscheme.modelling;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 21.03.11
 *         Time: 11:03
 */
public class LogicFunctionsWorker {

    public enum LogicFunction {AND, OR, NAND, NOR, XOR}

    private static final char ONE = '1';
    private static final char ZERO = '0';

    public static char getValue(LogicFunction function, String inputSet) {
        switch (function) {
            case AND: {
                return and(inputSet);
            }
            case OR: {
                return or(inputSet);
            }
            case NAND: {
                return nand(inputSet);
            }
            case NOR: {
                return nor(inputSet);
            }
            case XOR: {
                return xor(inputSet);
            }
            default: {
                return ZERO;
            }
        }
    }

    private static char and(String inputSet) {
        for (int i = 0; i < inputSet.length(); i++) {
            if (inputSet.charAt(i) == ZERO) {
                return ZERO;
            }
        }
        return ONE;
    }

    private static char or(String inputSet) {
        for (int i = 0; i < inputSet.length(); i++) {
            if (inputSet.charAt(i) == ONE) {
                return ONE;
            }
        }
        return ZERO;
    }

    private static char nand(String inputSet) {
        if (and(inputSet) == ONE) {
            return ZERO;
        }
        return ONE;
    }

    private static char nor(String inputSet) {
        if (or(inputSet) == ONE) {
            return ZERO;
        }
        return ONE;
    }

    private static char xor(String inputSet) {
        char result = xor2(inputSet.charAt(0), inputSet.charAt(1));
        for (int i = 2; i < inputSet.length(); i++) {
            result = xor2(result, inputSet.charAt(i));
```

```
        }
        return result;
    }

    private static char xor2(char ch1, char ch2) {
        if (ch1 == ch2) {
            return ZERO;
        }
        return ONE;
    }

}

package zak.adcs.logicscheme.modelling;

import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 21.03.11
 *         Time: 19:04
 */
public class ModellingModel extends AbstractTableModel {

    public static final int INTERVAL_WIDTH = 20;
    public static final int DIAGRAM_BORDER = 20;
    public static final int DIAGRAM_Y_INTERVAL = 15;

    public static final String TIME_STRING = "Время t";

    private static final String[] LAST_COLUMN_NAMES = {"ТТС", "t", "ТБС"};

    private Color backgroundColor;
    private Color standartColor;
    private Color textColor;

    private Font font;

    private String[] inNames;
    private String[] elementNames;
    private int[] inertiaDelay;
    private int[] dynamicDelay;
    private LogicFunctionsWorker.LogicFunction[] logicFunctions;
    private int[][] connectivityMatrix;

    private int t;
    private HashMap<Integer, LinkedList<Integer>> futureEventsTable;
    private int lastEventT;
    private LinkedList<String> previousFutureEvents;

    private Dimension diagramSize;
    private int graphicHeight;
    private ArrayList<HashMap<Integer, Character>> diagrams;

    private LinkedList<String[]> rows;
    private int columnCount;

    private boolean isBusy;
    private boolean isClear;

    public ModellingModel(String[] inNames, String[] elementNames, int[] inertiaDelay, int[] dynamicDelay,
                          LogicFunctionsWorker.LogicFunction[] logicFunctions, int[][] connectivityMatrix,
                          Color backgroundColor, Color standartColor, Color textColor, Font font) {
        this.inNames = inNames;
        this.elementNames = elementNames;
        this.inertiaDelay = inertiaDelay;
        this.dynamicDelay = dynamicDelay;
        this.logicFunctions = logicFunctions;
        this.connectivityMatrix = connectivityMatrix;
        this.backgroundColor = backgroundColor;
        this.standartColor = standartColor;
        this.textColor = textColor;
        this.font = font;
        columnCount = inNames.length + elementNames.length + 3;
        rows = new LinkedList<String[]>();
        isBusy = false;
        isClear = true;
        diagramSize = new Dimension(100, 100);
        t = 0;
        lastEventT = 0;
        futureEventsTable = new HashMap<Integer, LinkedList<Integer>>();
        previousFutureEvents = new LinkedList<String>();
        diagrams = new ArrayList<HashMap<Integer, Character>>();
        for (int i = 0; i < inNames.length + elementNames.length; i++) {
            diagrams.add(new HashMap<Integer, Character>());
        }
    }

    private boolean hasFutureEvents() {
        if (futureEventsTable.isEmpty()) {
            return false;
        }
        return true;
    }

    private void addDiagramValues(int t) {
        int index = 0;
```

```
    for (int i = 0; i < inNames.length; i++) {
        int j = rows.size() - 1;
        boolean found = false;
        while (!found && j >= 0) {
            if (rows.get(j)[index].length() > 0 && rows.get(j)[index].compareTo(" ") != 0) {
                found = true;
            }
            j--;
        }
        diagrams.get(index).put(t, rows.get(++j)[index].charAt(0));
        index++;
    }
    for (int i = 0; i < elementNames.length; i++) {
        int j = rows.size() - 1;
        boolean found = false;
        while (!found && j >= 0) {
            if (rows.get(j)[index].length() > 0) {
                found = true;
            }
            j--;
        }
        diagrams.get(index).put(t, rows.get(++j)[index].charAt(0));
        index++;
    }
}

private String getInputSet(ArrayList<Integer> numbers) {
    StringBuffer buffer = new StringBuffer();
    for (int index : numbers) {
        int j = rows.size() - 1;
        boolean found = false;
        while (!found && j >= 0) {
            if (rows.get(j)[index].length() > 0 && rows.get(j)[index].compareTo(" ") != 0) {
                found = true;
            }
            j--;
        }
        buffer.append(rows.get(++j)[index]);
    }
    return buffer.toString();
}

private String[] getStartCondition(String startSet) {
    ArrayList<String[]> conditions = new ArrayList<String[]>();
    String[] condition = new String[elementNames.length];
    for (int i = 0; i < condition.length; i++) {
        condition[i] = "0";
    }
    conditions.add(condition);
    boolean endFlag;
    do {
        condition = new String[elementNames.length];
        for (int i = 0; i < elementNames.length; i++) {
            ArrayList<Integer> inputElements = new ArrayList<Integer>();
            for (int j = 0; j < connectivityMatrix.length; j++) {
                if (connectivityMatrix[j][i + inNames.length] > 0) {
                    for (int k = connectivityMatrix[j][i + inNames.length]; k > 0; k--) {
                        inputElements.add(j);
                    }
                }
            }
            StringBuffer inputBuffer = new StringBuffer();
            for (int index : inputElements) {
                if (index < inNames.length) {
                    inputBuffer.append(startSet.substring(index, index + 1));
                } else {
                    inputBuffer.append(conditions.get(conditions.size() - 1)[index - inNames.length]);
                }
            }
            StringBuffer buffer = new StringBuffer();
            buffer.append(LogicFunctionsWorker.getValue(logicFunctions[i], inputBuffer.toString()));
            condition[i] = buffer.toString();
        }
        String[] previousCondition = conditions.get(conditions.size() - 1);
        endFlag = true;
        for (int i = 0; i < elementNames.length; i++) {
            if (condition[i].compareTo(previousCondition[i]) != 0) {
                endFlag = false;
                break;
            }
        }
        if (!endFlag) {
            conditions.add(condition);
        }
    } while (!endFlag);
    return conditions.get(conditions.size() - 1);
}

public boolean isBusy() {
    return isBusy;
}

public boolean isClear() {
    return isClear;
}

public void step(String inputSet, String startSet) {
    String[] row = new String[columnCount];
    int index = 0;
    for (int i = 0; i < inNames.length; i++) {
        row[index++] = "";
    }
    for (int i = 0; i < elementNames.length; i++) {
```

```
        row[index++] = "0";
    }
    for (int i = 0; i < LAST_COLUMN_NAMES.length; i++) {
        row[index++] = "";
    }
    rows.add(row);
    row = new String[columnCount];
    index = 0;
    for (int i = 0; i < startSet.length(); i++) {
        row[index++] = startSet.substring(i, i + 1);
    }
    String[] startCondition = getStartCondition(startSet);
    for (int i = 0; i < startCondition.length; i++) {
        row[index++] = startCondition[i];
    }
    row[index++] = "";
    row[index++] = "";
    StringBuilder stringBuilder = new StringBuilder();
    boolean isFirst = true;
    for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {
        for (int j = 0; j < inNames.length; j++) {
            if (connectivityMatrix[j][i] > 0) {
                if (!isFirst) {
                    stringBuilder.append("-");
                }
                stringBuilder.append(elementNames[i - inNames.length]);
                if (futureEventsTable.get(0) == null) {
                    futureEventsTable.put(0, new LinkedList<Integer>());
                }
                futureEventsTable.get(0).add(i);
                isFirst = false;
                break;
            }
        }
    }
    row[index] = stringBuilder.toString();
    previousFutureEvents.add(stringBuilder.toString());
    rows.add(row);
    isClear = false;
    step(inputSet, false);
}

public void step() {
    StringBuffer buffer = new StringBuffer();
    for (int i = 0; i < inNames.length; i++) {
        buffer.append(" ");
    }
    step(buffer.toString(), false);
}

public void step(String inputSet, boolean newSet) {
    if (newSet) {
        for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {
            for (int j = 0; j < inNames.length; j++) {
                if (connectivityMatrix[j][i] > 0) {
                    if (futureEventsTable.get(t) == null) {
                        futureEventsTable.put(t, new LinkedList<Integer>());
                    }
                    futureEventsTable.get(t).add(i);
                    break;
                }
            }
        }
    }
    String[] row = new String[columnCount];
    for (int i = 0; i < inNames.length; i++) {
        row[i] = inputSet.substring(i, i + 1);
    }
    for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {
        row[i] = "";
    }
    LinkedList<Integer> currentEvents = futureEventsTable.get(t);
    futureEventsTable.remove(t);
    if (currentEvents != null) {
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < currentEvents.size(); i++) {
            stringBuilder.append(elementNames[currentEvents.get(i) - inNames.length]);
            if (i < currentEvents.size() - 1) {
                stringBuilder.append("-");
            }
        }
        row[columnCount - 3] = stringBuilder.toString();
        String tempString = stringBuilder.toString();
        row[columnCount - 2] = String.valueOf(t);
        rows.add(row);
        stringBuilder = new StringBuilder();
        boolean isFirst = true;
        for (int index : currentEvents) {
            int delay = inertiaDelay[index - inNames.length] + dynamicDelay[index - inNames.length];
            ArrayList<Integer> inputElements = new ArrayList<Integer>();
            for (int i = 0; i < connectivityMatrix.length; i++) {
                if (connectivityMatrix[i][index] > 0) {
                    for (int j = connectivityMatrix[i][index]; j > 0; j--) {
                        inputElements.add(i);
                    }
                }
            }
            StringBuffer buffer = new StringBuffer();
            buffer.append(LogicFunctionsWorker.getValue(logicFunctions[index - inNames.length], getInputSet(inputElements)));
            row[index] = buffer.toString();
            for (int i = inNames.length; i < connectivityMatrix[index].length; i++) {
                if (connectivityMatrix[index][i] > 0) {
                    if (!isFirst) {
```

```
                        stringBuilder.append("-");
                    }
                    stringBuilder.append(elementNames[i - inNames.length]);
                    isFirst = false;
                    boolean addFlag = true;
                    for (int eventT = t; eventT <= lastEventT; eventT++) {
                        if (futureEventsTable.get(eventT) != null) {
                            if (futureEventsTable.get(eventT).contains(i)) {
                                if (eventT < t + delay) {
                                    futureEventsTable.get(eventT).remove(new Integer(i));
                                } else {
                                    addFlag = false;
                                }
                            }
                        }
                    }
                    if (addFlag) {
                        if (delay == 0) {
                            currentEvents.add(i);
                        } else {
                            if (futureEventsTable.get(t + delay) == null) {
                                futureEventsTable.put(t + delay, new LinkedList<Integer>());
                            }
                            if (!futureEventsTable.get(t + delay).contains(i)) {
                                futureEventsTable.get(t + delay).add(i);
                                if (t + delay > lastEventT) {
                                    lastEventT = t + delay;
                                }
                            }
                        }
                    }
                }
            }
        }
        row[columnCount - 1] = stringBuilder.toString();
        isBusy = hasFutureEvents();
        previousFutureEvents.add(tempString);
        for (String e : previousFutureEvents) {
            if (e.compareTo(stringBuilder.toString()) == 0 && stringBuilder.toString().length() > 0) {
                isBusy = false;
                break;
            }
        }
        if (!isBusy) {
            previousFutureEvents = new LinkedList<String>();
            lastEventT = 0;
        } else {
            previousFutureEvents.add(stringBuilder.toString());
        }
    } else {
        row[inNames.length + elementNames.length + 1] = String.valueOf(t);
        rows.add(row);
    }
    addDiagramValues(t);
    t++;
}

public void modelling(String inputSet, String startSet) {
    if (!isBusy) {
        if (isClear) {
            step(inputSet, startSet);
        } else {
            step(inputSet, true);
        }
    }
    while (isBusy) {
        step();
    }
}

public void clear() {
    t = 0;
    lastEventT = 0;
    futureEventsTable = new HashMap<Integer, LinkedList<Integer>>();
    previousFutureEvents = new LinkedList<String>();
    isBusy = false;
    isClear = true;
    rows = new LinkedList<String[]>();
}

public int getT() {
    return t;
}

public void setDiagramSize(Dimension diagramSize) {
    this.diagramSize = diagramSize;
    graphicHeight = (int) (diagramSize.getHeight() / (inNames.length + elementNames.length));
}

public ArrayList<String> getNames() {
    ArrayList<String> names = new ArrayList<String>();
    for (int i = 0; i < inNames.length; i++) {
        names.add(inNames[i]);
    }
    for (int i = 0; i < elementNames.length; i++) {
        names.add(elementNames[i]);
    }
    return names;
}

public ArrayList<HashMap<Integer, Character>> getDiagrams() {
    return diagrams;
}
```

```
    public Dimension getDiagramSize() {
        return diagramSize;
    }

    public int getGraphicHeight() {
        return graphicHeight;
    }

    public Color getBackgroundColor() {
        return backgroundColor;
    }

    public Color getStandartColor() {
        return standartColor;
    }

    public Color getTextColor() {
        return textColor;
    }

    public Font getFont() {
        return font;
    }

    @Override
    public String getColumnName(int column) {
        int temp = column;
        if (temp < inNames.length) {
            return inNames[temp];
        } else {
            temp -= inNames.length;
            if (temp < elementNames.length) {
                return elementNames[temp];
            } else {
                temp -= elementNames.length;
                return LAST_COLUMN_NAMES[temp];
            }
        }
    }

    public int getRowCount() {
        return rows.size();
    }

    public int getColumnCount() {
        return columnCount;
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        return rows.get(rowIndex)[columnIndex];
    }

}
```

```
package zak.adcs.logicscheme.modelling;

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

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 21.03.11
 *         Time: 17:25
 */
public class ModellingPanel extends JPanel {

    private JLabel inputSetLabel;
    private JComboBox inputSetBox;
    private JLabel startSetLabel;
    private JComboBox startSetBox;
    private JButton stepButton;
    private JButton modellingButton;
    private JButton allModellingButton;
    private JButton clearButton;
    private JSplitPane splitPane;
    private JTable modellingTable;
    private TimeDiagramPanel diagramPanel;

    private ModellingModel modellingModel;

    public ModellingPanel(SchemeTableModel schemeTableModel, Color backgroundColor, Color standartColor, Color textColor,
                          Font font) {
        super();
        modellingModel = new ModellingModel(schemeTableModel.getInNames(), schemeTableModel.getElementNames(),
                schemeTableModel.getInertiaDelay(), schemeTableModel.getDynamicDelay(),
                schemeTableModel.getLogicFunctions(), schemeTableModel.getConnectivityMatrixExceptOut(),
                backgroundColor, standartColor, textColor, font);
        StringBuilder builder = new StringBuilder();
        builder.append("Входной набор ( ");
        for (String s : schemeTableModel.getInNames()) {
            builder.append(s);
            builder.append(" ");
        }
        builder.append(")");
        inputSetLabel = new JLabel(builder.toString());
        inputSetBox = new JComboBox();
        inputSetBox.setEditable(false);
        for (int i = 0; i < Math.pow(2, schemeTableModel.getInCount()); i++) {
            String temp = Integer.toBinaryString(i);
```

```
            while (temp.length() < schemeTableModel.getInCount()) {
                temp = "0" + temp;
            }
            inputSetBox.addItem(temp);
        }
        builder = new StringBuilder();
        builder.append("Установочный набор ( ");
        for (String s : schemeTableModel.getInNames()) {
            builder.append(s);
            builder.append(" ");
        }
        builder.append(")");
        startSetLabel = new JLabel(builder.toString());
        startSetBox = new JComboBox();
        startSetBox.setEditable(false);
        for (int i = 0; i < Math.pow(2, schemeTableModel.getInCount()); i++) {
            String temp = Integer.toBinaryString(i);
            while (temp.length() < schemeTableModel.getInCount()) {
                temp = "0" + temp;
            }
            startSetBox.addItem(temp);
        }
        stepButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                if (modellingModel.isBusy()) {
                    modellingModel.step();
                } else {
                    if (modellingModel.isClear()) {
                        modellingModel.step((String) inputSetBox.getSelectedItem(), (String) startSetBox.getSelectedItem());
                    } else {
                        modellingModel.step((String) inputSetBox.getSelectedItem(), true);
                    }
                }
                modellingModel.fireTableDataChanged();
                revalidate();
                repaint();
            }
        });
        stepButton.setText("Шаг");
        modellingButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                modellingModel.modelling((String) inputSetBox.getSelectedItem(), (String) startSetBox.getSelectedItem());
                modellingModel.fireTableDataChanged();
                revalidate();
                repaint();
            }
        });
        modellingButton.setText("Моделировать");
        allModellingButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                for (int i = 0; i < inputSetBox.getItemCount(); i++) {
                    modellingModel.modelling((String) inputSetBox.getItemAt(i), (String) startSetBox.getSelectedItem());
                }
                modellingModel.fireTableDataChanged();
                revalidate();
                repaint();
            }
        });
        allModellingButton.setText("Моделировать все наборы");
        clearButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                modellingModel.clear();
                modellingModel.fireTableDataChanged();
                revalidate();
                repaint();
            }
        });
        clearButton.setText("Очистить");
        modellingTable = new JTable(modellingModel);
        modellingTable.setDragEnabled(false);
        diagramPanel = new TimeDiagramPanel(modellingModel);
        splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        splitPane.setTopComponent(new JScrollPane(modellingTable));
        splitPane.setBottomComponent(new JScrollPane(diagramPanel));
        splitPane.setDividerLocation(splitPane.getPreferredSize().height / 2);
        int strutSize = 5;
        Box hBox = Box.createHorizontalBox();
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(inputSetLabel);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(inputSetBox);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(startSetLabel);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(startSetBox);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(stepButton);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(modellingButton);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(allModellingButton);
        hBox.add(Box.createHorizontalStrut(strutSize));
        hBox.add(clearButton);
        hBox.add(Box.createHorizontalStrut(strutSize));
        Box vBox = Box.createVerticalBox();
        vBox.add(Box.createVerticalStrut(2 * strutSize));
        vBox.add(hBox);
        vBox.add(Box.createVerticalStrut(2 * strutSize));
        setLayout(new BorderLayout());
        add(new JScrollPane(vBox), BorderLayout.NORTH);
        add(splitPane);
    }
```

```
}

package zak.adcs.logicscheme.modelling;

import javax.swing.table.AbstractTableModel;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 21.03.11
 *         Time: 10:48
 */
public class SchemeTableModel extends AbstractTableModel {

    private String[] inNames;
    private String[] elementNames;
    private LogicFunctionsWorker.LogicFunction[] logicFunctions;
    private int[] inertiaDelay;
    private int[] dynamicDelay;
    private String[] outNames;
    private int[][] connectivityMatrix;

    public SchemeTableModel(String[] inNames, String[] elementNames, LogicFunctionsWorker.LogicFunction[] logicFunctions,
                            int[] inertiaDelay, int[] dynamicDelay, String[] outNames, int[][] connectivityMatrix) {
        this.inNames = inNames;
        this.elementNames = elementNames;
        this.logicFunctions = logicFunctions;
        this.inertiaDelay = inertiaDelay;
        this.dynamicDelay = dynamicDelay;
        this.outNames = outNames;
        this.connectivityMatrix = connectivityMatrix;
    }

    public int getInCount() {
        return inNames.length;
    }

    public String[] getInNames() {
        return inNames;
    }

    public String[] getElementNames() {
        return elementNames;
    }

    public LogicFunctionsWorker.LogicFunction[] getLogicFunctions() {
        return logicFunctions;
    }

    public int[] getInertiaDelay() {
        return inertiaDelay;
    }

    public int[] getDynamicDelay() {
        return dynamicDelay;
    }

    public int[][] getConnectivityMatrixExceptOut() {
        int[][] result = new int[inNames.length + elementNames.length][];
        for (int i = 0; i < result.length; i++) {
            result[i] = new int[result.length];
            for (int j = 0; j < result[i].length; j++) {
                result[i][j] = connectivityMatrix[i][j];
            }
        }
        return result;
    }

    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return false;
    }

    public int getRowCount() {
        return connectivityMatrix.length + 1;
    }

    public int getColumnCount() {
        return (inNames.length + elementNames.length + outNames.length + 1);
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        if (rowIndex == 0 && columnIndex == 0) {
            return "";
        }
        if (rowIndex == 0) {
            int temp = columnIndex - 1;
            if (temp >= inNames.length) {
                temp -= inNames.length;
                if (temp < elementNames.length) {
                    return elementNames[temp];
                } else {
                    temp -= elementNames.length;
                    return outNames[temp];
                }
            } else {
                return inNames[temp];
            }
        }
        if (columnIndex == 0) {
            int temp = rowIndex - 1;
            if (temp >= inNames.length) {
```

```
                temp -= inNames.length;
                if (temp < elementNames.length) {
                    return elementNames[temp];
                } else {
                    temp -= elementNames.length;
                    return outNames[temp];
                }
            } else {
                return inNames[temp];
            }
        }
        return connectivityMatrix[rowIndex - 1][columnIndex - 1];
    }

    public int getElementCount() {
        return elementNames.length;
    }

}

package zak.adcs.logicscheme.modelling;

import javax.swing.*;
import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.font.LineMetrics;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * Created by IntelliJ IDEA.
 *
 * @author Zakhozhyy Ihor
 *         Date: 21.03.11
 *         Time: 17:26
 */
public class TimeDiagramPanel extends JPanel implements Scrollable {

    private ModellingModel modellingModel;

    public TimeDiagramPanel(ModellingModel modellingModel) {
        this.modellingModel = modellingModel;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        setBackground(modellingModel.getBackgroundColor());
        FontRenderContext context = g2.getFontRenderContext();
        g2.setFont(modellingModel.getFont());
        ArrayList<String> names = modellingModel.getNames();
        int maxWidth = 0;
        int maxHeight = 0;
        Rectangle2D bounds;
        for (String name : names) {
            bounds = getFont().getStringBounds(name, context);
            if (bounds.getWidth() > maxWidth) {
                maxWidth = (int) bounds.getWidth();
            }
            if (bounds.getHeight() > maxHeight) {
                maxHeight = (int) bounds.getHeight();
            }
        }
        bounds = getFont().getStringBounds(ModellingModel.TIME_STRING, context);
        if (bounds.getWidth() > maxWidth) {
            maxWidth = (int) bounds.getWidth();
        }
        modellingModel.setDiagramSize(new Dimension(maxWidth + ModellingModel.INTERVAL_WIDTH * modellingModel.getT() +
ModellingModel.DIAGRAM_BORDER * 3,
                maxHeight * (names.size() + 1) + ModellingModel.DIAGRAM_Y_INTERVAL * names.size() + ModellingModel.DIAGRAM_BORDER * 2));
        setSize(modellingModel.getDiagramSize());
        g2.setColor(modellingModel.getTextColor());
        int y = ModellingModel.DIAGRAM_BORDER;
        for (int i = 0; i < names.size(); i++) {
            LineMetrics metrics = modellingModel.getFont().getLineMetrics(names.get(i), context);
            g2.drawString(names.get(i), ModellingModel.DIAGRAM_BORDER, (int) (y + metrics.getAscent()));
            y += maxHeight + ModellingModel.DIAGRAM_Y_INTERVAL;
        }
        if (modellingModel.getT() > 0) {
            ArrayList<HashMap<Integer, Character>> values = modellingModel.getDiagrams();
            y = ModellingModel.DIAGRAM_BORDER;
            g2.setColor(modellingModel.getStandartColor());
            for (int i = 0; i < values.size(); i++) {
                int x = 2 * ModellingModel.DIAGRAM_BORDER + maxWidth;
                for (int t = 0; t < modellingModel.getT(); t++) {
                    if (values.get(i).get(t).compareTo('1') == 0) {
                        g2.fillRect(x, y, ModellingModel.INTERVAL_WIDTH, maxHeight);
                    } else {
                        g2.drawLine(x, y + maxHeight, x + ModellingModel.INTERVAL_WIDTH, y + maxHeight);
                    }
                    x += ModellingModel.INTERVAL_WIDTH;
                }
                y += maxHeight + ModellingModel.DIAGRAM_Y_INTERVAL;
            }
        }
        g2.setColor(modellingModel.getTextColor());
        int x = ModellingModel.DIAGRAM_BORDER;
        LineMetrics metrics = modellingModel.getFont().getLineMetrics(ModellingModel.TIME_STRING, context);
        g2.drawString(ModellingModel.TIME_STRING, x, y + metrics.getAscent());
        x += ModellingModel.DIAGRAM_BORDER + maxWidth;
        for (int t = 0; t < modellingModel.getT(); t++) {
```

```
            String tString = String.valueOf(t);
            bounds = modellingModel.getFont().getStringBounds(tString, context);
            metrics = modellingModel.getFont().getLineMetrics(tString, context);
            g2.drawString(tString, (int) (x - bounds.getWidth() / 2), (int) (y + metrics.getAscent()));
            x += ModellingModel.INTERVAL_WIDTH;
        }
    }

    @Override
    public Dimension getPreferredSize() {
        return modellingModel.getDiagramSize();
    }

    public Dimension getPreferredScrollableViewportSize() {
        return getPreferredSize();
    }

    public int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction) {
        return 1;
    }

    public int getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int direction) {
        return 10;
    }

    public boolean getScrollableTracksViewportWidth() {
        return false;
    }

    public boolean getScrollableTracksViewportHeight() {
        return false;
    }

}
```