

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

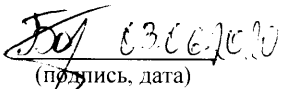
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА

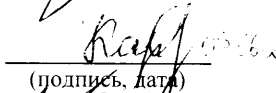
на тему

**Инструментальная система темпорального
анализа моделей параллельных
вычислительных систем**

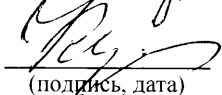
Студент

 Борискин В.В.
(подпись, дата)

Руководитель

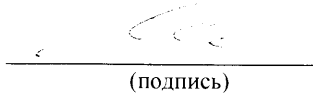
 Карамышева Н.С.
(подпись, дата)

Нормоконтролёр

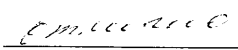
 А.В. Кучин
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от **03.06.2020 г. № 12**)

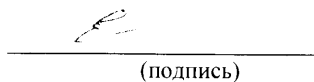
Заведующий кафедрой


(подпись)

М.А. Митрохин

Работа защищена с отметкой  (протокол заседания ГЭК от **11.06.2020 г. № 4**)

Секретарь ГЭК


(подпись)

Е.И. Гурин

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2020 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

1. Студент Борискин Вячеслав Владимирович гр.16BV1 факультета вычислительной техники.

2. Тема работы: «Инструментальная система темпорального анализа моделей параллельных вычислительных систем»

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Карамышева Н.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Инструментальная система темпорального анализа моделей параллельных вычислительных систем», выполняющее проектирование, описание и моделирование поведения вычислительных систем. Приложение должно обладать графическим пользовательским интерфейсом. Система должна включать в себя функционал ввода и разбора системы канонических уравнений, описывающих моделируемую систему, отображать графическую модель системы и предоставлять пользователю возможность взаимодействовать с элементами моделируемой системы и проводить симуляцию ее работы. Разрабатываемое приложение должно выполнять задачи проектирования, общего анализа и моделирования вычислительных систем.

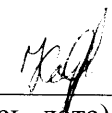
5. Перечень подлежащих разработке вопросов

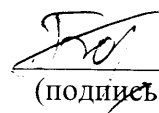
Требуется произвести анализ предметных областей: изучить актуальные на сегодняшний день методы описания вычислительных систем, изучить существующие инструменты, выполняющие описание. Необходимо изучить методологии проектирования объектно-ориентированных приложений, а также изучить функционал фреймворка Windows Forms.

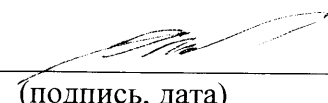
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	30.04.2020 – 07.05.2020	
2.	Анализ требований	10	08.05.2020 – 10.05.2020	
3.	Выбор технологий, применяемых при проектировании и разработке	15	11.05.2020 – 13.05.2020	
4.	Описание алгоритмов	15	14.05.2020 – 16.04.2020	
5.	Разработка приложения «Инструментальная система темпорального анализа моделей параллельных вычислительных систем»	30	17.04.2020 – 27.05.2020	
6.	Оформление пояснительной записки	10	28.05.2020 - 01.06.2020	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы  Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент  30.04.2020 Борискин В.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 149 листов, 44 рисунка, 2 таблицы, 6 формул, 11 источников, 2 приложения.

АВТОМАТЫ, ВРЕМЕННЫЕ АВТОМАТЫ, МОДЕЛИ, МОДЕЛИРОВАНИЕ, СИМУЛЯЦИЯ, ВРЕМЕННЫЕ ЗАДЕРЖКИ.

Цель работы – разработать систему, позволяющую проектировать, анализировать и моделировать сложные вычислительные системы, подразумевающие своей работе временные задержки.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: ввод системы канонических уравнений, описывающих моделируемую систему, анализ и разбор введенной системы канонических уравнений, создание визуальной модели вычислительной системы, описанной с помощью временного автомата, моделирование поведения вычислительной системы. Приложение должно обладать графическим пользовательским интерфейсом.

В результате проведенной работы разработано приложение, удовлетворяющее поставленной задаче, а именно: разработаны алгоритмы парсинга системы канонических уравнений, отображения графических объектов временного автомата, описывающего вычислительную систему, моделирования и симуляции работы вычислительной системы. Приложение успешно протестировано и готово к использованию.

Перечень принятых сокращений

API (Application Programming Interface) – программный интерфейс приложения;

UML (Unified Modeling Language) – унифицированный язык моделирования;

ЕС – Едина система

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств

СКУ – система канонических уравнений

ПЭВМ – персональная электронно-вычислительная машина

ТПВ – таблица переходов и выходов

ВС – вычислительная система

Содержание

Перечень принятых сокращений.....	5
Введение.....	7
1.Анализ предметной области.....	8
1.1.Краткое описание теории автоматов.....	8
1.2.Способы описания вычислительных систем.....	11
1.3.Постановка проблемы.....	16
1.4.Анализ существующих решений.....	17
1.5.Выбор языка программирования.....	21
1.6. Выводы.....	22
2.Реализация.....	24
2.1.Описание работы разработанной среды	24
2.2.Описание алгоритмов.....	26
2.3.Описание классов.....	32
2.4.Выводы.....	43
3.Описание результата.....	45
3.1.Описание работы реальной системы.....	45
3.2.Руководство пользователя.....	47
3.3.Выводы.....	63
4.Охрана труда.....	64
Заключение.....	67
Список используемых источников.....	68
Приложение А. Листинги программы.....	69
Приложение В. Презентация.....	128

Введение

Жизнь современного человека невозможно представить без участия в ней компьютера. Внедрение вычислительных машин происходит во все более и более новые сферы жизнедеятельности человека. Компьютеры используются в роли персональных машин, для удовлетворения личных потребностей пользователя, таких как потребление аудио и видео контента.

Также компьютеры используются в любительской и профессиональной деятельности. Например, компьютер используется для работы с текстом. Для удовлетворения подобных потребностей применяются текстовые редакторы, специальные инструменты, обладающие широкими возможностями для работы с текстовой информацией.

Подобно этому персональные компьютеры также используются при работе с графической информацией. В частности, с помощью использования компьютера представляется возможным создание потрясающих изображений, обработка фотографий, создание и редактирование видео файлов.

В течение последнего столетия четко прослеживается вектор развития индустрии вычислительной техники. С каждым годом вычислительные машины все более совершенствуются, а их физические габариты становятся гораздо меньше. Вместе с последними экспоненциально растут вычислительные мощности, позволяющие обрабатывать все большие объемы информации. Для удовлетворения потребностей пользователя активно разрабатываются и совершенствуются вычислительные алгоритмы, выполняющие ту или иную задачу. На ряду с развитием алгоритмов активно разрабатывались методы их описания.

1. Анализ предметной области

Отправной точкой в проектировании и разработке любого приложения является анализ предметной области. Этот этап необходим для четкой постановки целей, формировании требований к конечному продукту и определении ожидаемых результатов всей работы. В этом разделе проводится обзор основных аспектов и методик описания вычислительных систем.

1.1. Краткое описание теории автоматов

Одним из часто используемых методов описания алгоритмов - моделирование посредством абстрактных автоматов [1].

Абстрактные автоматы - вычислительные машины, представленные в виде математических моделей, осуществляющие обработку информации по заданному алгоритму [2]. Проблемами в сфере абстрактных автоматов призвана заниматься теория автоматов – раздел дискретной математики, изучающий способы преобразования информации.

1.1.1. Задачи теории автоматов

— Задача анализа. По заданному автомату описать его поведение, установить свойства автомата.

— Задача синтеза. Построить автомат с наперед заданным поведением (алгоритмом функционирования). Задачу синтеза принято рассматривать двояко: абстрактный синтез как построение математической модели автомата и структурный синтез как разработку функциональной логической схемы автомата.

— Задача полноты. Пусть M — некоторое множество автоматов. Определить, обладает ли совокупность автоматов, составляющих подмножество M' множества M , свойством полноты. Иными словами, если ко всем автоматам подмножества M' конечное число раз применить операцию суперпозиции, совпадут ли M' и M ?

— Задача минимизации. Построить автомат, минимальный заданному. Минимальный автомат обладает наименьшим числом компонентов модели (в частности, минимальной мощностью множества так

называемых состояний) и при этом функционально эквивалентен заданному автомату.

— Задача эквивалентных преобразований. Определить полную систему правил, позволяющую преобразовывать произвольный автомат в любой эквивалентный ему автомат. Частным случаем данной задачи является переход от одной модели автомата к другой. Два автомата функционально эквивалентны, если их поведение одинаково при воздействии одних и тех же последовательностей входных сигналов. В таком случае говорят о совпадении моделей поведения двух автоматов.

1.1.2. Классификация автоматов

Введем понятие конечного автомата. Конечный автомат — это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд.

Конечные автоматы классифицируются по нескольким параметрам.

— По степени определенности. Автоматы делятся на детерминированные и недетерминированные. Отличие между ними заключается в том, что в один момент времени детерминированные автоматы могут находиться только в одном состоянии, в то время как недетерминированные автоматы могут находиться в нескольких состояниях одновременно. Исходя из этого отличия детерминированный автомат можно описать как последовательность из пяти элементов:

$$M = (Q, \Sigma, \beta, S_0, F)$$

Q – множество состояний автомата

Σ – алфавит языка

β – функция перехода такая, что $\beta: Q \times \Sigma \rightarrow Q$,

$S_0 \in Q$ – начальное состояние

$F \subseteq Q$ – множество конечных состояний.

Недетерминированный автомат описывается похожим образом – набором элементов:

$$M = (Q, \Sigma, \Delta, S, F)$$

Q – множество состояний автомата

Σ - алфавит языка

Δ - отношение перехода

$S \subseteq Q$ – множество начальных состояний

$F \subseteq Q$ – множество конечных состояний.

— По типу. Различают два типа автоматов. Автомат Мили и автомат Мура. Различия между ними заключается в том, что выходной сигнал в автомате Мили зависит от входного сигнала и от предыдущего состояния автомата, тогда как выходной сигнал в автомате Мура зависит только от текущего состояния автомата.

— По наличию зависимости от времени. В этом ключе автоматы подразделяются на временные и не временные. Временные автоматы – те автоматы в работе и функционировании которых подразумевается наличие временных задержек, величина которых определятся в зависимости от каких-либо условий. Другими словами временной автомат – это кортеж из 6 множеств (T, A, S, S_0, P, E) , где T – множество таймеров, A – множество

действий, S – множество состояний управления, S_0 – начальное состояние, $P : L \rightarrow TC \downarrow$ – назначение инвариантов, $E \subseteq L \times TC \times A \times 2C \times L$ – множество переходов. Для временных автоматов выполняются следующие условия.

Каждый временной автомат оборудован внутренним часовым механизмом (временной переменной), который меняет своё значение с течением времени и пригоден для измерения дискретных промежутков времени: 1 такт (единица времени), 2 такта и т.д. При успешном срабатывании какого-либо перехода внутренний таймер устанавливается в 0.

Функция $\sigma S : \lambda S \rightarrow (2F \setminus \emptyset)$ есть функция задержки выходного символа,

которая определяет для каждого перехода непустое, возможно, бесконечное множество допустимых моментов времени, в которые автоматом может быть произведён выходной символ. Это множество моментов времени задаётся конечным набором линейных функций с целыми неотрицательными

коэффициентами. Если $\sigma S (hs1, i, o, s2i) = D$, где $D \subset F$ – конечное множество

линейных функций с целыми неотрицательными коэффициентами, то, находясь в состоянии $s1$ и получив входное воздействие i , автомат сбрасывает значение внутреннего таймера в 0 и может выдать выходную реакцию o на это воздействие в любой момент, когда таймер приобретет значение из $T(D)$. В этот же самый момент состояние автомата станет $s2$, а таймер вновь сбросится в 0.

3. Временной автомат начинает свою работу из начального состояния, а внутренний таймер установлен в 0.

1.2. Способы описания вычислительных систем

Описать какую-либо вычислительную систему можно с помощью конечных автоматов. Чтобы описать систему существует несколько способов

1.2.1. Описание с помощью таблиц переходов и выходов.

При табличном способе задания и описания автомата Мили строятся две таблицы – переходов и выходов. В таблице переходов указывается из какого состояния под действием какого сигнала автомат переходит в то или иное состояние. В таблице выходов указывается формируемый выходной сигнал, выдаваемый автоматом в каком-либо состоянии при поступлении какого-либо входного сигнала. В таблице 1 представлено описание работы RS-триггера, с помощью автомата Мили, где S_0 – состояние, при котором триггер закрыт, S_1 – состояние, при котором триггер открыт, Z_0 – сигнал, подающийся на вход R, Z_1 – сигнал, подающийся на вход S, W_0 – выходной сигнал с инверсного выхода $\sim Q$, W_1 – выходной сигнал с выхода Q.

Таблица 1 - Описание RS-триггера с помощью автомата Мили

Состояние	Входные сигналы	
	Z_0	Z_1
S_0	S_0/W_0	S_1/W_1
S_1	S_0/W_0	S_1/W_1

Автомат Мура описывается с помощью одной отмеченной таблицы, в которой одновременно в столбце с начальным состоянием также указывается и выходной сигнал, вырабатываемый при переходе в это состояние. На пересечении строки и столбца указывается состояние, в которое переходит автомат из данного состояния под действием данного входного сигнала. В таблице 2 представлен результат описание RS-триггера с помощью автомата Мура, где S_0 – состояние, при котором триггер закрыт, S_1 – состояние, при котором триггер открыт, Z_0 – сигнал, подающийся на вход R, Z_1 – сигнал, подающийся на вход S, W_0 – выходной сигнал с инверсного выхода $\sim Q$, W_1 – выходной сигнал с выхода Q.

Таблица 2 - Описание RS-триггера с помощью автомата Мура

Состояние	Выходной сигнал	Входные сигналы	
		Z_0	Z_1
S_0	W_0	S_0	S_1
S_1	W_1	S_0	S_1

1.2.2. Описание посредством графа

При таком способе автомат задается в виде математического графа, граф - абстрактный математический объект, представляющий собой множество вершин и набор ребер – соединений между вершинами. При

описании алгоритма с помощью абстрактного автомата строится граф, множеством вершин которого является множество состояний автомата, а множество ребер - множество переходов между состояниями. При описании автомата Мили в начале дуги, обозначающей переход, записывается входной сигнал, инициирующий данный переход, в конце дуги записывается выходной сигнал. На рисунке 1 изображен граф автомата Мили, описывающего работу RS-триггера, где S_0 – состояние, при котором триггер закрыт, S_1 – состояние, при котором триггер открыт, Z_0 – сигнал, подающийся на вход R, Z_1 – сигнал, подающийся на вход S, W_0 – выходной сигнал с инверсного выхода $\sim Q$, W_1 – выходной сигнал с выхода Q.

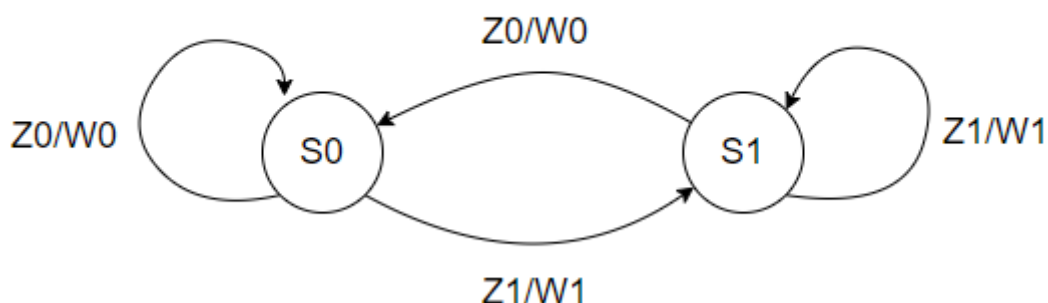


Рисунок 1 - Граф RS-триггера на автомате Мили

Для автомата Мура выходной сигнал записывается рядом с состоянием, в котором он формируется. На рисунке 2 изображен RS-триггер, описанный с помощью автомата Мура, где S_0 – состояние, при котором триггер закрыт, S_1 – состояние, при котором триггер открыт, Z_0 – сигнал, подающийся на вход R, Z_1 – сигнал, подающийся на вход S, W_0 – выходной сигнал с инверсного выхода $\sim Q$, W_1 – выходной сигнал с выхода Q.

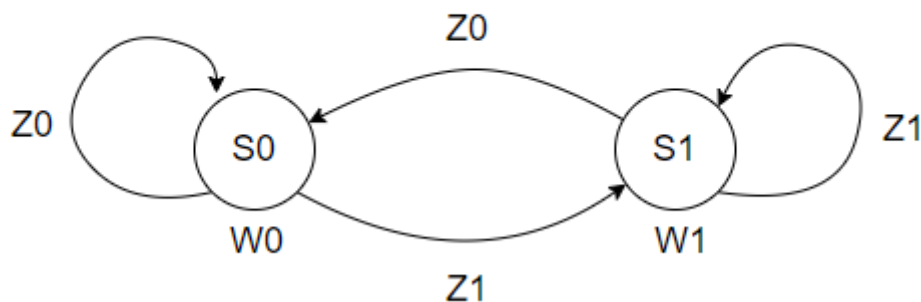


Рисунок 2 - Граф RS-триггера на автомате Мура

1.2.3. Описание систем с временной составляющей.

Исходя из вышеперечисленного, способ описания алгоритмов методом использования абстрактных автоматов – один из самых практичных и широко используемых. Но подобным методом решить все проблемы описания алгоритмов не представляется возможным. Существует множество алгоритмов, подразумевающих в своей работе временную составляющую.

Введем такое понятие как киберсистема. Киберсистема – это вычислительная система, подразумевающая в своей работе интеграцию физических устройств, таких как датчиков и т.д. Киберфизические системы, представляют из себя объединение физического и цифрового мира. Важным подклассом киберфизических систем являются промышленные киберфизические системы, в которых физические системы включают датчики и исполнительные механизмы, а цифровые системы основаны на использовании контроллеров и компьютеров. Отличительными свойствами промышленных киберфизических систем являются сложные динамические взаимодействия между элементами, подсистемами и окружающей средой. Промышленные киберфизические системы управляются множеством интеллектуальных устройств, связанных коммуникационной сетью и взаимодействующих друг с другом для формирования интеллектуальных машин, промышленных и логистических систем.

На сегодняшний день актуальными являются несколько методов описания и апробации вычислительных киберсистем.

— Описание киберсистем методом применения автомата с таймаутами [3]. Под подобным автоматом понимается пятерка $(S, I, O, hS, \Delta S)$, где S – конечное непустое множество состояний, I – входной алфавит, O – выходной алфавит, $hS \subseteq (S \times I \times O \times S \times Z)$ – отношение переходов, Z – множество целых неотрицательных чисел, определяющих число тактов времени между поступлением входного символа и выдачей выходного, $\Delta S: S \rightarrow S \times (N \cup \{\infty\})$ – функция (входного) таймаута, определяющая для каждого состояния максимальное время ожидания входного символа, N – множество натуральных чисел. Принцип работы такого метода заключается в следующем, если в каком-либо состоянии входной сигнал не поступает

какое-то время определенной величины, то автомат переходит в следующее состояние спустя это время, независимо от того поступил ли входной сигнал или нет.

— Описание методом применения автомата с временными ограничениями [4]. Под таким автоматом понимается множество (I, S, O, hS) , где I – входной алфавит, O – выходной алфавит, S – конечное непустое множество состояний, $hS \subseteq (S \times I \times O \times S \times \Pi \times Z)$ – отношение переходов, Π – множество интервалов из промежутка $[0; \infty)$, и Z – множество целых неотрицательных чисел. Принцип работы описывается набором (s, i, o, s', g, d) , где переходы из состояния s в состояние s' происходят под действием входного символа i , поступившего в момент времени t , $t \in g$, после перехода автомата в текущее состояние s выдачей выходного символа o через d тактов времени после поступления входного символа.

Если в некотором состоянии автомата входной сигнал не поступает в течение определенного времени, который превышает (входной) таймаут в текущем состоянии, то автомат может изменить своё состояние.

После перехода в состояние s' по входному символу либо таймауту отсчет времени начинается с 0. Если $s = s'$, то после достижения таймаута в состоянии s отсчет времени начинается с 0. Функция $\text{time}(s, t) = s'$ [7] определяет, в каком состоянии s' находится автомат через t тактов времени, при условии, что входной символ не был подан в течение этого времени. Для временного автомата S через V будем обозначать наибольшую конечную границу для временных интервалов, которая совпадает с максимальной величиной входного таймаута. Временным входным символом называется пара (i, t) , где i – символ входного алфавита, t – время поступления входного символа после выдачи автоматом последнего выходного символа. Временным выходным символом называется пара (o, d) , где o – символ выходного алфавита, d – число единиц времени между подачей входного символа и выдачей выходного символа. Для временного автомата S и входной временной последовательности $\alpha = (i_1, t_1), (i_2, t_2) \dots, (i_n, t_n)$ соответствующая выходная временная последовательность $\gamma = (o_1, d_1), (o_2,$

d2) ..., (on, dn) называется (выходной) реакцией. При этом последовательность пар (временной входной символ / временной выходной символ) называется временной входе-выходной последовательностью и обозначается α/γ . Для нахождения реакции временного автомата на входной временной символ (i, t) в состоянии s, сначала определяется состояние $s' = \text{time}(s, t)$, в котором находится автомат в момент времени t, далее определяется переход (s', i, o, s'', g, d) такой, что $(t - \Sigma) \in g$, где Σ есть сумма таймаутов при переходах по таймаутам из состояния s в состояние s'; соответственно, при подаче входного временного символа (i, t) в состоянии s автомат производит выходную реакцию (o, d) и переходит в следующее состояние s". Таким образом, реакция автомата на временную входную последовательность в состоянии s вычисляется итеративно, начиная с состояния s. Вычитание величины Σ из времени поступления входного символа отражает тот факт, что отсчёт времени устанавливается в 0 при выполнении автоматом переходов по таймауту. На рисунке 3 представлен автомат, работающий описанным образом.

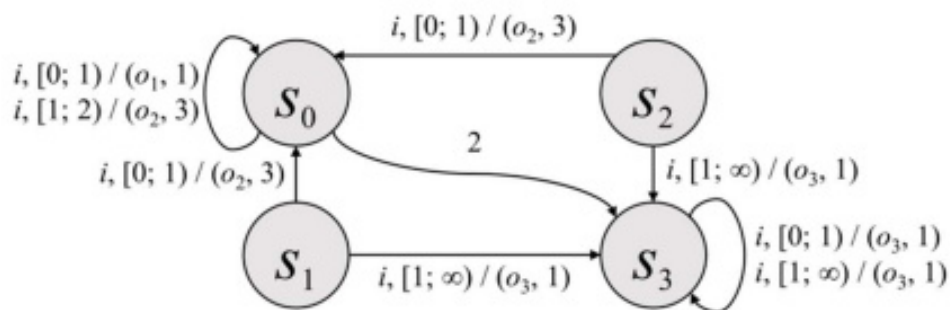


Рисунок 3 - Автомат с временными ограничениями

1.3. Постановка проблемы

В промышленной деятельности человеку может понадобиться наладить и оптимизировать какой-то процесс, внедрить в уже работающую систему новый функционал, добавить к разного рода оборудованию новый модуль, исполняющий некий алгоритм. После внедрения изменений необходимо удостовериться, что все процессы и части системы функционируют должным образом. Появляется необходимость провести анализ работы системы в целом.

С помощью перечисленных методик описания можно описать работу всей вычислительной системы и проанализировать ее. В перечисленных методиках и решениях проблема описания алгоритмов, подразумевающих в своей работе временные задержки исследована мало.

На данный момент не существует актуального метода моделирования вычислительных систем позволяющего симулировать работу системы в зависимости от временных меток. Эта проблема актуальна на сегодняшний день, в следствии чего была предпринята попытка найти и описать один из способов ее решения.

Перечисленные выше методы имеют ряд недостатков, в числе которых такие, как моделирование только детерминированных автоматов, а также сложность в проведении темпорального анализа вычислительных систем.

1.4. Анализ существующих решений

1.4.1. Обзор доступных на рынке решений

На сегодняшний день на рынке технологий существуют несколько решений, позволяющих проводить описание, осуществлять проектирование и разработку информационных систем.

Среди всего множества решений наиболее крупными являются такие инструменты, как CPN Tools, Rational Rose, MagicDraw.

1.4.2. Достоинства и недостатки существующих решений

CPN Tools. CPN Tools - это специальная моделирующая система, которая использует язык сетей Петри для описания моделей. CPN Tools предлагает очень мощный класс сетей Петри для описания моделей. Согласно стандартной классификации, такие сети называют иерархическими временными раскрашенными сетями Петри. Было доказано, что они эквивалентны машине Тьюринга и составляет универсальную алгоритмическую систему. Таким образом, произвольный объект может быть описан с помощью этого класса сетей.

Временные сети Петри используют понятие модельного времени для описания продолжительности действий в реальных объектах. В отличие от классических сетей Петри, где срабатывание перехода происходит мгновенно, срабатывание перехода во временной сети связано с

определенной продолжительностью или временной задержкой. Это позволяет анализировать временные характеристики реальных объектов, например, время отклика как характеристику качества обслуживания сети.

Несмотря на все достоинства данного инструмента, он имеет и ряд недостатков, в числе которых такие, как сложность в использовании, большое потребление ресурсов машины, выполняющей процессы CPN Tools, сильная ориентированность на сети Петри. На рисунке 4 представлен интерфейс данной среды.

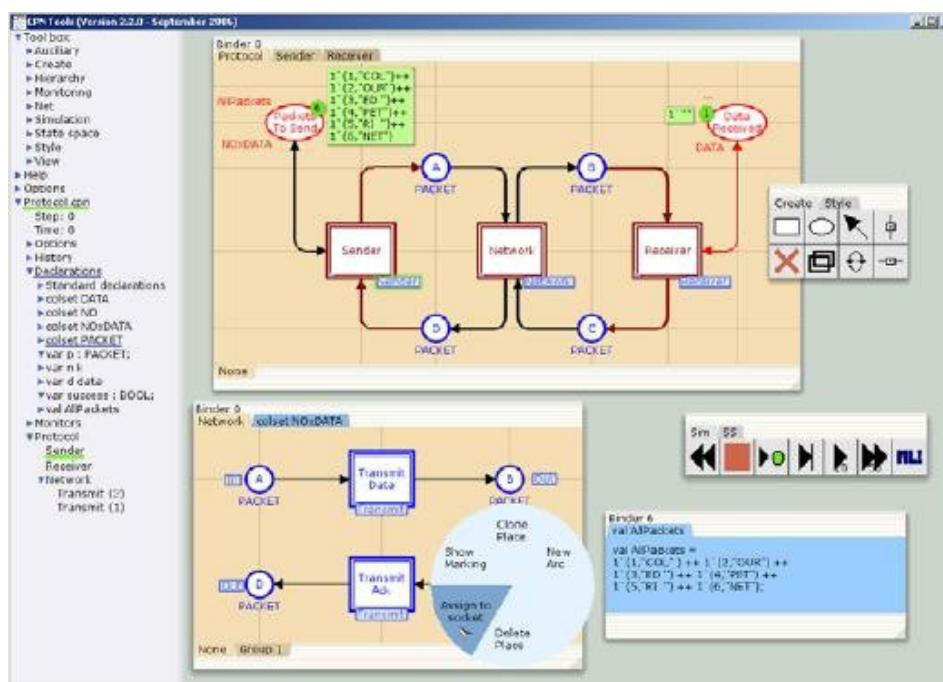


Рисунок 4 - Интерфейс CPN Tools

Rational Rose. Rational Rose представляет собой CASE средство проектирования и разработки информационных систем и программного обеспечения для управления предприятиями. Как и другие CASE средства его можно применять для анализа и моделирования бизнес-процессов.

Принципиальное отличие Rational Rose от других средств заключается в объектно-ориентированном подходе. Графические модели, создаваемые с помощью этого средства, основаны на объектно-ориентированных принципах и языке UML.

UML - унифицированный язык моделирования - это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. При работе с UML используются в основном три

структурные единицы языка. Это диаграммы, сущности и связи. Сущности – это абстракции, которые являются основными элементами модели, связи соединяют их между собой, а диаграммы группируют представляющие интерес наборы сущностей. Диаграммы – это графическое представление множества элементов, изображаемое в виде связного графа.

Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем. Инструменты моделирования Rational Rose позволяют разработчикам создавать целостную архитектуру процессов предприятия, сохраняя все взаимосвязи и управляющие воздействия между различными уровнями иерархии.

К недостаткам данного инструмента можно отнести сложный пользовательский интерфейс, имеющий ряд недоработок, а также невозможность проводить моделирование и симуляцию информационных систем в работе которых присутствует временная составляющая. Еще одним недостатком является моделирование информационных систем только средствами языка UML, метод формализованного описания информационных систем с помощью автоматов различного типа в данном инструменте не поддерживается. На рисунке 5 представлен интерфейс системы.

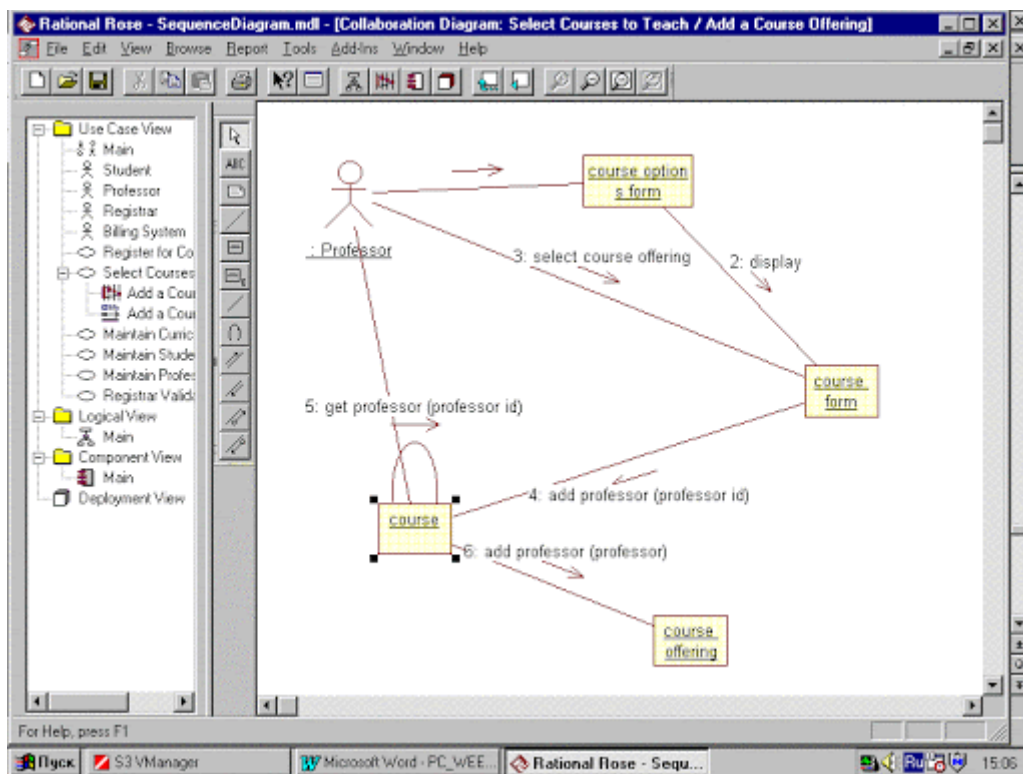


Рисунок 5 - Интерфейс Rational Rose

MagicDraw. MagicDraw – это инструмент для моделирования бизнес-процессов, архитектуры, программного обеспечения и систем с поддержкой совместной работы. Моделирование происходит с помощью средств языка UML. Предназначен в основном для проектирования объектно-ориентированных систем и баз данных.

Основными недостатками является возможность описания системы только с помощью языка UML и ему подобными (UML, SysML, BPMN и UPDM). Отсутствие возможности описания вычислительных систем, в частности систем с временными составляющими. На рисунке 6 представлен интерфейс данной системы.

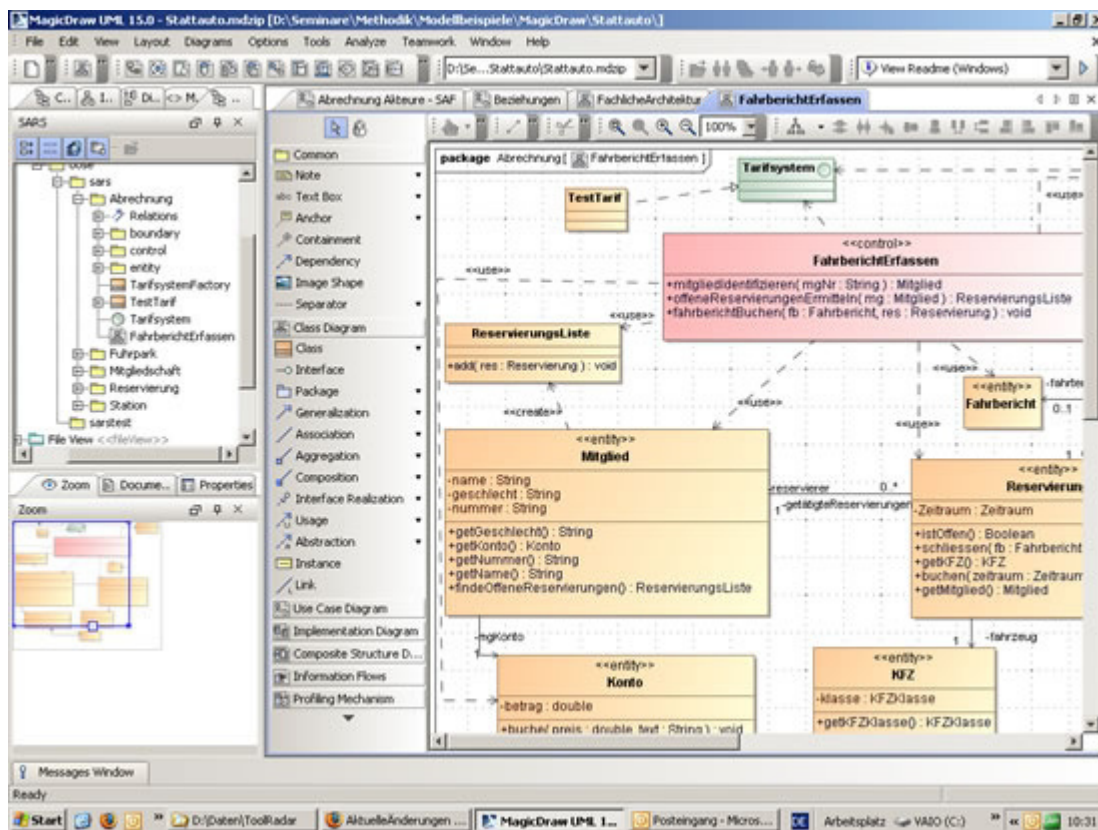


Рисунок 6 - Интерфейс MagicDraw

1.5. Выбор языка программирования

При выборе языка программирования необходимо учитывать поддержку работы с графическими окнами [5]. Также важно учитывать, чтобы язык был объектно-ориентированный. Объектно-ориентированные языки предоставляют разработчику возможность правильно спроектировать структуру приложения для удобной и корректной разработки конечного продукта [6]. Так же немало важными факторами разработки является чистота кода и производительность [7]. В настоящее время наиболее распространенными языками для разработки оконных приложения являются C++, Java, JavaScript, C#.

1.5.1. Язык программирования C++.

C++ - объектно-ориентированный, компилируемый, строго типизированный язык программирования. Этот язык удовлетворяет основным парадигмам объектной ориентированности, таким как полиморфизм, наследование, инкапсуляция. К недостаткам языка можно отнести несовершенство механизма работы с памятью, и сложный синтаксис языка.

1.5.2. Язык программирования Java.

Java – кроссплатформенный строго типизированный, объектно-ориентированный язык программирования. Кроссплатформенность данного языка достигается благодаря наличию нем такого средства, как `jvm – java virtual machine`. При компиляции исходный код, компилируется в так-называемый байт-код, способный исполняться виртуальной машиной на практически любой физической машине. К недостаткам можно отнести слабую ориентированность языка на разработку оконных приложений и малое количество библиотек ориентированных на разработку оконных приложений.

1.5.3. Язык программирования JavaScript.

JavaScript – скриптовый интерпретируемый язык, предназначенный для исполнения в среде браузера. В основном применяется в разработки Web-страничек. JavaScript поддерживает императивный, декларативный и объектно-ориентированный стили программирования. Так как JavaScript интерпретируемый язык безопасность и производительность напрямую зависит от конкретной реализации интерпретатора.

1.5.4. Язык программирования C#

C# – объектно-ориентированный, компилируемый, строго типизированный язык программирования [8]. Является производным от языка C++. В этом языке хорошо развита поддержка оконных приложений. Также C# является объектно-ориентированным языком, в виду чего ему тоже свойственны такие парадигмы как полиморфизм, наследование и инкапсуляция. C# обладает хорошей производительностью и безопасностью. В качестве языка разработки был выбран именно он.

1.6. Выводы

Основываясь на результатах проведенного анализа было принято решение разработать собственную среды проектирования и моделирование вычислительных систем, которая будет решать такие задачи, как проектирование сложных вычислительных систем, моделирование поведения систем, подразумевающих в своей работе временные задержки, анализ

систем с целью выявления узких мест и уязвимостей еще на стадии проектирования.

В качестве языка разработки был выбран язык C# т.к. одним из важных критериев разработанной среды является удобный и понятный для пользователя интерфейс. Такой язык программирования, как C# позволяет удовлетворить данную потребность.

2. Реализация

В качестве решения вышеупомянутой проблемы была разработана и отлажена среда моделирования и симуляции вычислительных систем посредством парсинга системы канонических уравнений (СКУ). Принципиальное отличие разработанной среды проектирования вычислительных систем заключается в том, что с помощью данной среды пользователь может спроектировать и промоделировать поведение систем, в которых важную роль играют временные задержки в функционировании.

2.1. Описание работы разработанной среды

Моделирование предполагается проводить с помощью построения временного автомата, описывающего поведение системы [9]. Для удовлетворения своих потребностей пользователю достаточно описать поведение исходной системы аналитическим образом, то есть с помощью системы канонических уравнений. СКУ задаются как набор формул в нормальной дизъюнктивной форме (ДНФ). В качестве операндов самого уравнения выступают конъюнкты, состоящие из временной составляющей, предыдущего состояния и выходного сигнала предыдущего состояния, само же уравнение приравнивается к новому состоянию. Количество конъюнктов определяет количество вершин графа автомата, из которых осуществляется переход.

После ввода СКУ среда приступает к анализу введенных данных. Происходит построчное считывание информации из текстового пространства и результат считывания передается разработанному парсеру СКУ. Данный алгоритм анализирует переданную информации с помощью механизма регулярных выражений и создает множества, описывающие компоненты графа.

После того как все данные из текстового пространства прочитаны и проанализированы, исходя из сформировавшихся объектов строится модель симулируемой системы, представленная в виде временного автомата. На графическом пространстве поочередно прорисовывается каждый компонент графа, созданный парсером СКУ. После отрисовки элементов графа

управление возвращается пользователю.

Пользователь имеет возможность взаимодействовать с графическими элементами, перемещать, выделять и устанавливать их в сигнальное состояние. После совершения каких-либо действий пользователя, поочередно анализируются множества объектов, и, в случае необходимости, происходит перерисовка компонентов, потерпевших изменение, после чего управление возвращается пользователю.

Если пользователь начал симуляцию, на каждом ее шаге определяются состояния компонентов графа, если какой-то элемент потерпел изменение, происходит перерисовка графа, и затем управление возвращается пользователю. На рисунках 7 и 8 представлены UML-диаграммы последовательности и вариантов использования разработанной среды.

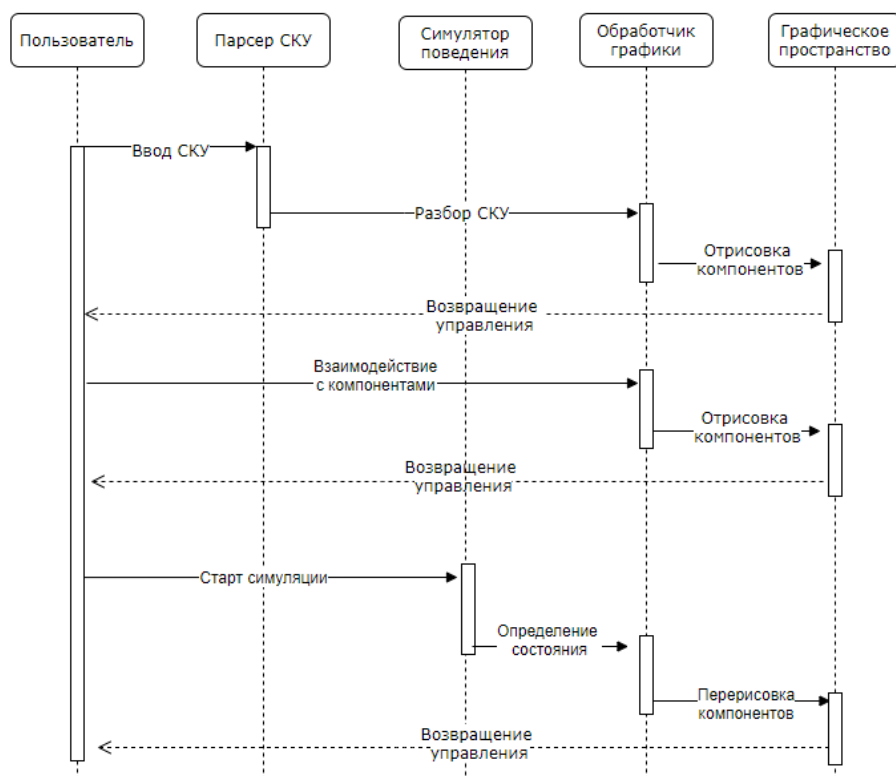


Рисунок 7 - UML-диаграмма последовательности работы алгоритма

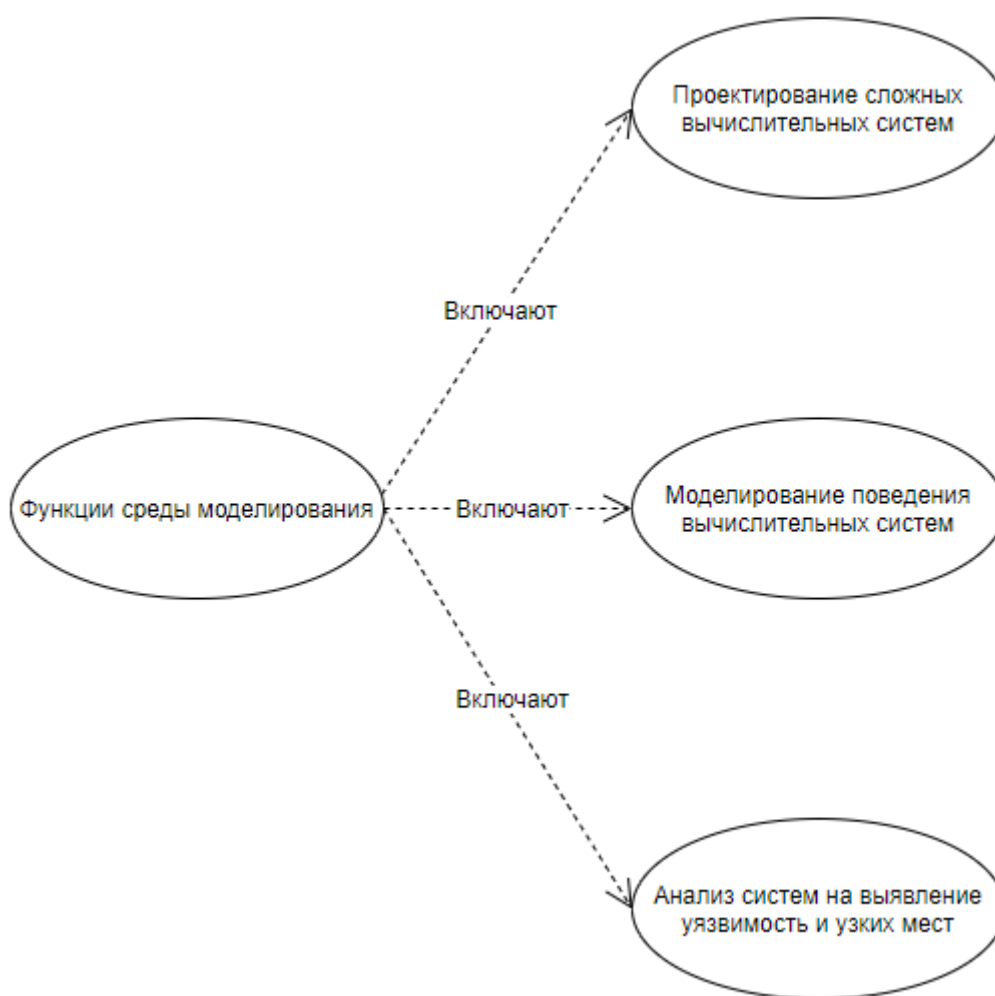


Рисунок 8 - UML-диаграмма вариантов использования функций среды моделирования

2.2. Описание алгоритмов

В ходе выполнения работы, были разработаны и отлажены следующие алгоритмы.

2.2.1. Алгоритм разбора и анализа СКУ.

Алгоритм производит построчное считывание данных из текстового пространства, предназначенного для ввода СКУ. Каждая строка анализируется с помощью механизма обработки регулярных выражений. В среде задается шаблон, согласно которому происходит разбор входной строки [10]. Шаблон разбивается на 4 группы. Первая группа производит анализ строки с целью нахождения конечного состояния, характерного для анализируемой строки. Вторая группа отвечает за поиск начальных состояний, из которых можно перейти в конечное состояние. Третья группа обрабатывает строку с целью поиска входных сигналов, под действием

которых происходит переход. Четвертая группа отвечает за поиск временной метки, при наличии которой переход срабатывает только через заданное количество времени, измеряемого в миллисекундах. Если входная строка содержит все 4 группы, то происходит ее разбор и создаются объекты состояний, переходов, сигналов и т.д., необходимые для работы среды. Схема алгоритма изображена на рисунке 9.

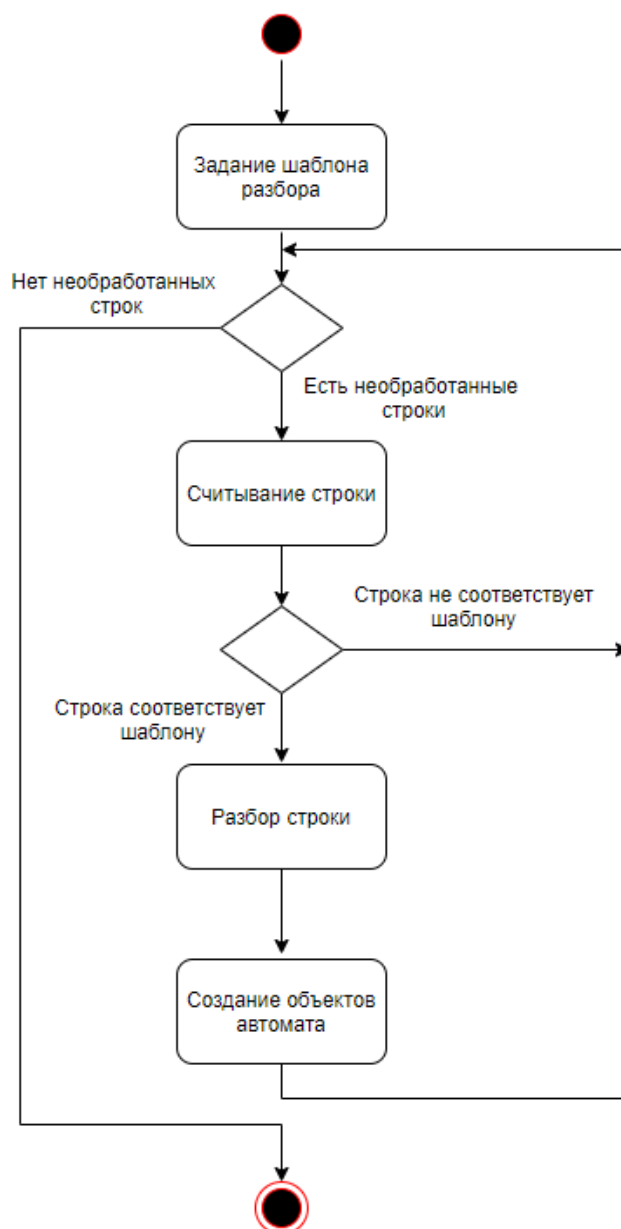


Рисунок 9 – UML-диаграмма деятельности разбора СКУ

2.2.2. Алгоритм отрисовки компонентов.

После того, как проанализированы все строки СКУ, происходит отображение созданных объектов на графическом пространстве. Перед отрисовкой у каждого компонента рассчитываются координаты точек, в

которых происходит отображение данного элемента. При наведении курсора мыши на тот или иной элемент, происходит его подсвечивание. Данный эффект достигается следующим образом.

1. Всякий раз при движении мыши по графическому пространству, в системе устанавливается и обрабатывается соответствующее событие (event).

2. Среди свойств события присутствуют такие свойства, как координата текущего положения мыши по оси x , и координата текущего положения мыши по оси y . Точка начала координат расположена в крайней левой верхней точки графического пространства, т.е. значение каждой координаты означает количество точек от точки начала координат до точки текущего расположения мыши по соответствующей оси.

3. После того, как в среде инициализировались координаты текущего положения мыши, поочередно происходит анализ всех графических компонентов графа.

4. Как описывалось ранее, у каждого элемента графа есть координаты его вершин. Исходя из значений координат этих вершин можно вычислить площадь графического пространства, занимаемого тем или иным элементом, а также крайние точки этой области. Поочередно, у каждого графического компонента вычисляется занимаемая площадь и крайние точки этой площади.

5. После того, как все необходимые расчеты были проведены, данные о расположении каждого элемента на графическом пространстве сравниваются с координатами текущего положения курсора мыши.

6. Если курсор мыши находится в пределах области графического пространства, занимаемого тем или иным элементом, данный элемент устанавливается в специальное состояние.

7. Если в процессе обработки события движения мыши какие-то флаги элементов графа поменяли свое значение, например какой-то элемент был не активен, а стал активен и наоборот, происходит перерисовка изменившихся компонентов, в соответствии с нужными стилями.

У каждой единицы графа свой алгоритм определения площади и крайних точек занимаемого пространства. Например, состояния изображены в виде окружностей, следовательно площадь поверхности на графическом пространстве – это площадь круга внутри окружности.

Исходя из аксиом геометрии у любой окружности есть радиус. Зная радиус окружности несложно вычислить площадь круга по формуле 1, где S – требуемая площадь, π – константа, примерно равная 3,14, r – радиус окружности.

Состояние «выделенности» перехода концептуально определяется подобным образом, т.е. вычисляется площадь, занимаемая отрезком того или иного перехода, вычисляются начальная и конечная точки перехода, и исходя из полученных результатов анализируется, находится ли курсор мыши в пределах активной области. В виду того, что переходы в среде представлены отрезком, алгоритм определяющий, выделен ли переход в данный момент или нет реализован несколько по-другому, в отличие от алгоритма, работающего с состояниями.

Исходя из теорем геометрии, любой отрезок принадлежит какой-то прямой на плоскости. Прямую можно описать с помощью канонического уравнения, представленного в формуле 2, где $A (X_1, Y_1)$, $B (X_2, Y_2)$ – точки, принадлежащих данной прямой.

Примем за точки A и B точки начала и конца отрезка, обозначающего переход. Тогда, подставив координаты любой точки в полученное уравнение, можно определить принадлежит ли данная точка этой прямой или нет. Если равенство выполняется, значит точка принадлежит прямой и объект анализируемого отрезка устанавливается в сигнальное состояние. Далее в работу вступает уже описанный ранее алгоритм перерисовки изменивших свое состояние элементов.

Переходы, изображенные в виде арок, в случаях, когда система переходит из какого-то состояния в это же состояние, обрабатываются с помощью алгоритма, аналогичного алгоритму обработки состояний.

Так как входной сигнал может быть отображен около нескольких переходов, то у него предусмотрено наличие нескольких точек отображения. Координаты этих точек вычисляются следующим образом.

1. Определяется множество переходов, которые срабатывают в зависимости от данного сигнала.

2. Поочередно обрабатывается каждый переход. Исходя из координат вершин начального и конечного состояний, между которыми присутствует переход, по теореме Пифагора вычисляется значение длины отрезка, обозначающего этот переход.

3. Исходя из вычисленного значения и координат вершин начального и конечного состояний, рассчитываются значения синуса и косинуса угла наклона отрезка относительно оси X.

4. Исходя из значений синуса и косинуса угла наклона, вычисляется начальная и точка отрезка, обозначающего переход. Относительно координат центра окружности, представляющей собой начальное состояние вычисляется смещение по оси X, рассчитываемое как $r * \cos \alpha$, где r – это радиус окружности, обозначающей начальное состояние, а α – угол наклона отрезка, относительно оси X. Полученное значение является значением координаты X начальной точки отрезка. С помощью выражения $r * \sin \alpha$ вычисляется смещение по оси Y, относительно центра окружности. Полученное значение будет координатой Y начальной точки отрезка. Конечная точка вычисляется подобным образом, за исключением знака выражения.

5. С помощью координат этих точек вычисляются координаты точки середины данного отрезка. Координата X вычисляется как длина половины отрезка между большей и меньшей координатами X точек начала и конца отрезка. Координата Y вычисляется аналогично.

6. Из координаты Y вычисленной точки вычитается определенная в свойствах системы константа, чтобы точка отображения сигнала не лежала на отрезке перехода и данные не сливались.

Аналогичным образом обрабатываются точки отображения временных меток. Схема алгоритма приведена на рисунке 10.

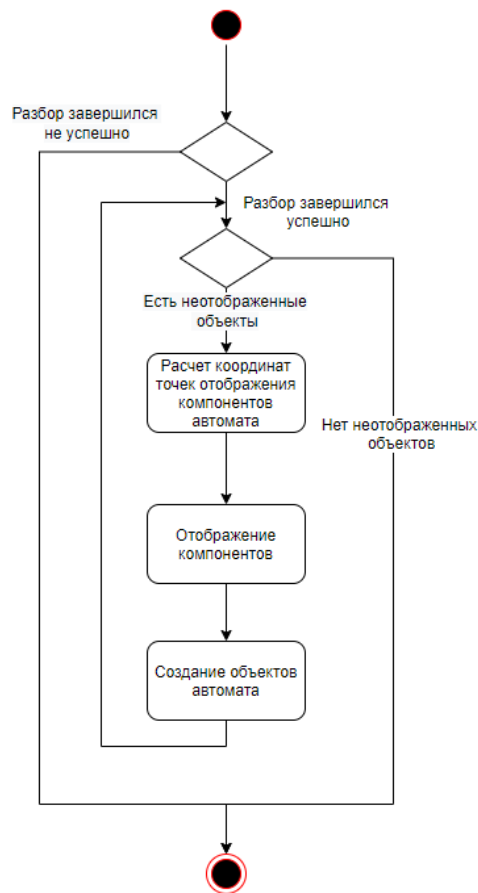


Рисунок 10 – UML-диаграмма деятельности алгоритма отрисовки

2.2.3. Алгоритм моделирования поведения системы

После того, как все инициализирующие операции были завершены, происходит симуляция работы вычислительной системы.

Алгоритм проверяет, указано ли какое-либо начальное состояние. Если начальное состояние не указано, будет выведено сообщение об ошибке.

Если начальное состояние установлено алгоритм приступает к моделированию поведения. Исходя из информации о активности входных сигналов и величины временных задержек алгоритм на каждом шаге определяет состояние в какое переходит моделируемая система, либо остается в исходном состоянии если никакой переход еще не сработал. Цикл продолжается до тех пор, симуляция не остановлена.

Также алгоритм предусматривает возможность пошагового моделирования поведения. После каждого такта моделирования, управление возвращается пользователю для анализа следующего действия.

Цикл повторяется до тех пор, пока симуляция не остановлена. Схема работы алгоритма приведена на рисунке 11.

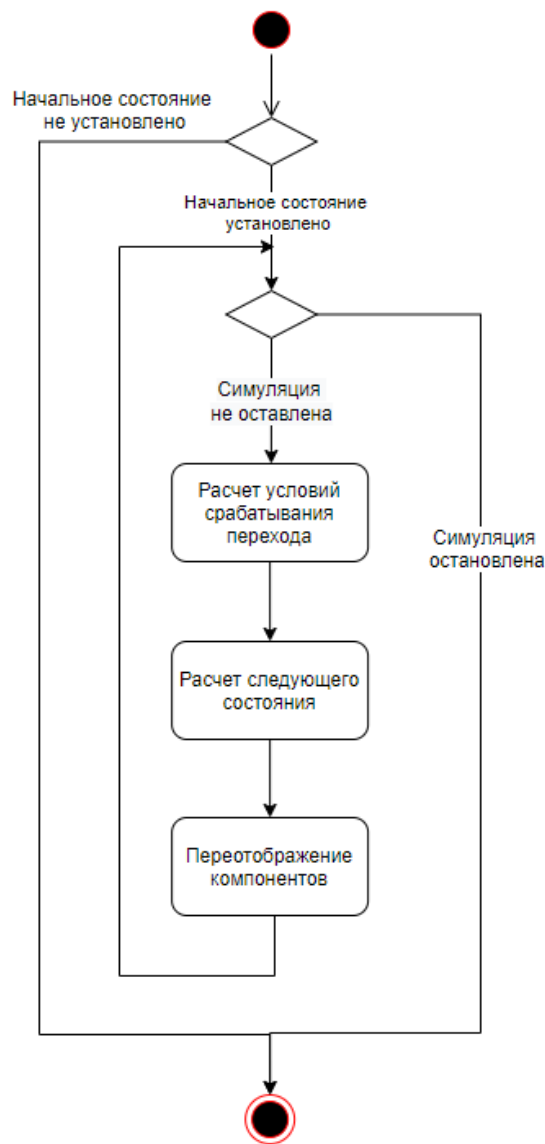


Рисунок 11 – UML-диаграмма деятельности алгоритма симуляции

2.3. Описание классов

В разработанной системе все множество классов разделено на функциональные блоки.

1. Сущности – классы, описывающие какую-либо сущность, над которой производятся действия.
2. Сервисы – классы, в которых расположен функционал для работы с классами сущностями.
3. Интерфейсы – классы, отвечающие за отображение графического интерфейса.
4. Свойства – классы, хранящие в себе свойства среды моделирование, заданные в ее настройках.

Разберем на примере взаимодействие классов между друг другом.

2.3.1. Описание сущностей

Среди классов сущностей присутствуют такие классы, как Rule, State, Signal, Link, Time.

Класс Rule описывает одно правило разбора СКУ, на основе которой строится временной автомат. Этот класс содержит следующие поля:

- *startState* типа State – начальное состояние.
- *endState* типа State – конечное состояние.
- *signal* типа Signal – сигнал, инициирующий переход между состояниями.
- *timeMark* типа Time – временная метка, по истечению которой срабатывает переход. Присутствует опционально.
- *signalInverted* типа bool – флаг инвертирован ли сигнал.

В результате разбора объект этого класса описывает одно правило, которое в свою очередь устанавливает из какого состояния в какое состояние, под действием какого сигнала, инвертирован ли сигнал и по истечению какой временной метки произойдет переход. Структура класса представлена на рисунке 12.

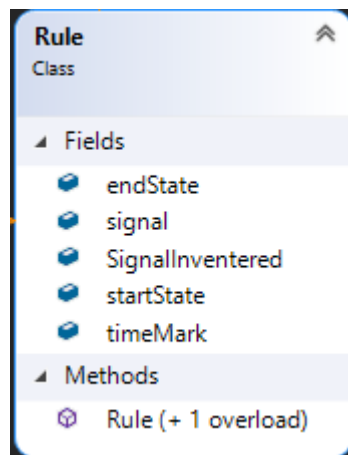


Рисунок 12 - UML-диаграмма класс Rule

Класс State описывает состояние автомата. Данный класс имеет следующие поля:

- *inputs* типа Signal [] – массив входных сигналов, инициирующих переход в это состояние.
- *links* типа Link [] - массив переходов, при срабатывании которых в системе устанавливается это состояние.

- *name* типа `String` – имя состояния.
- *selected* типа `bool` – флаг, обозначающий нужно ли подсветить состояние, устанавливающийся при наведении курсора мыши.
- *active* типа `bool` – флаг, обозначающий является ли активным это состояние.
- *paintDot* типа `Point` – точка отрисовки состояния.
- *nameDot* типа `Point`, точка отрисовки имени этого состояния.

`Point` – структура, описывающая местоположение чего-либо в рабочем пространстве окна. Также класс имеет следующие методы:

- *setNameDot()* – вычисляющий точку отображения имени, исходя из точки отрисовки.

Структура класса приведена на рисунке 13.

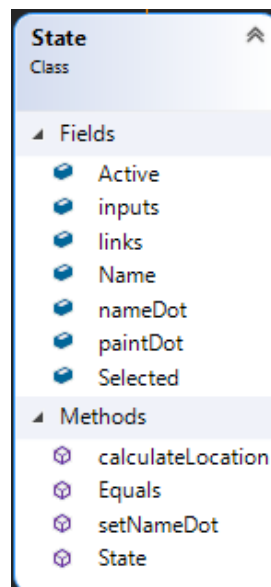


Рисунок 13 – UML-диаграмма класса `State`

Класс `Signal` описывает поведение и состояние какого-либо входного сигнала. Данный класс имеет следующие поля:

- *links* типа `Link []`, массив переходов, обозначающий какие переходы зависят от данного сигнала.
- *name* типа `String` - имя сигнала.
- *selected* типа `bool` - флаг, устанавливающийся при наведении мыши на этот сигнал.

— *active* типа `bool` – флаг, обозначающий находится ли сигнал в единичном состоянии.

— *paintDots* типа `Point []` – массив, описывающий множество точек на графическом пространстве, в котором будет отображено имя данного сигнала,

— *invertedPoints* типа `Point []` – массив, описывающий множество точек, в которых данный сигнал имеет инвертирующее значение. В таких точках перед именем сигнала предшествует символ восклицательного знака.

Также данный класс имеет следующие методы:

— *setPaintDots()* – вычисление точек, в которых отображается данный сигнал. Вычисление происходят исходя из массива `links`.

Структура класса `Signal` представлена на рисунке 14.

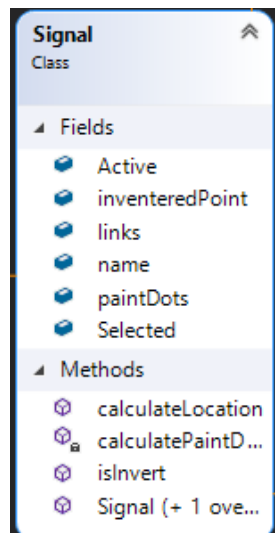


Рисунок 14 - UML-диаграмма класса `Signal`

Класс `Link` отвечает за описание того или иного перехода между состояниями. Графически этот класс представляет собой отрезок. Класс содержит следующие поля:

— *startState* типа `State` – начальное состояние перехода.

— *endState* типа `State` - конечное состояния перехода.

— *startDot* типа `Paint` – начальная точка отрезка.

— *endDot* типа `Point` – конечная точка отрезка.

— *signals* типа `Signal []` – массив, обозначающий множество входных сигналов, влияющих на конкретный переход,

— *timeMark* типа *Time* – временная задержка, по истечению которой срабатывает переход. Поле опционально.

— *arc* типа *bool*, флаг, означающий является ли переход аркой или нет.

— *arcDot* типа *Point* - точка отрисовки арки.

— *arrowDots* типа *Point []* – массив, описывающий точки отрисовки стрелки, указывающей направление перехода,

— несколько полей типа *float*, описывающие математические характеристики, такие как *sin*, *cos*, *length*, обозначающие синус и косинус угла наклона отрезка конкретного перехода относительно оси X и длину отрезка. Эти поля необходимы для корректного расчета местоположения конкретного перехода на графической области.

Также класс имеет следующие методы:

— *setName()* – устанавливает имя перехода, исходя из начального и конечного состояния.

— *calculateLength()* – вычисляет длину отрезка, представляющего переход.

— *setCosX()* – вычисляет косинус угла наклона отрезка, относительно оси X.

— *setSinX()* – вычисляет синус угла наклона отрезка, относительно оси X.

— *calculateArrowDots()* – вычисляет координаты точек, обозначающих стрелку направления перехода.

— *calculateStartAndEndDots()* – вычисляет точки начала и конца отрезка, обозначающего переход.

— *linkActive()* – вычисляет сработает ли данный переход.

Структура класса изображена на рисунке 15.

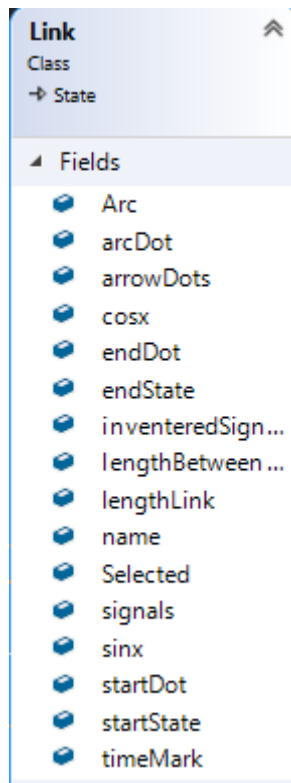


Рисунок 15 – UML-диаграмма класса Link

Класс Time описывает временную задержку при переходе моделируемой системы из одного состояния в другое. Класс имеет следующие поля:

- *links* типа Link [] – массив, описывающий на какие переходы влияет конкретная временная метка.
- *name* типа String - имя временной метки.
- *value* типа float - значение временной метки, измеряемое в миллисекундах.
- *paintDots* типа Point [] – массив, описывающий множество точек, в которых требуется отобразить имя временной метки на графическом пространстве.
- *selected* типа bool – флаг, означающий выделена ли временная метка при наведении мыши.

Структура класса представлена на рисунке 16.

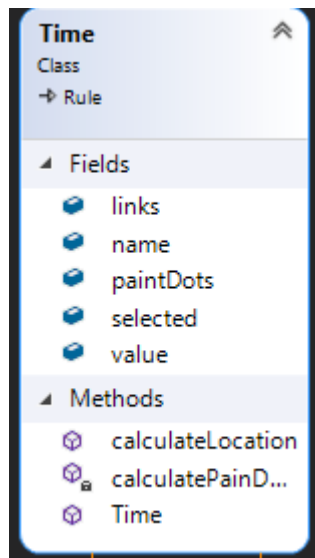


Рисунок 16 - UML-диаграмма класса Time

2.3.2. Описание сервисов.

Среди сервисов присутствуют два крупных класса, отвечающие за общий функционал системы. Это классы Parser и DrawArea.

Класс Parser производит разбор системы канонических уравнений, введенных в текстовое пространство среды. Данные считываются построчно и анализируются с помощью механизма регулярных выражений. На выходе создаются уже настроенные объекты компонентов графа и управление передается классу DrawArea. Класс имеет следующие методы:

- *parseStart()* – начинает разбор введенной SKY.
- *parse(String)* – анализирует входную строку, создавая массивы сущностей, описанных в предыдущем пункте.
- *getStateByName(String)* – получает из массива состояний состояние по имени.
- *getSignalByName(String)* – получает сигнал по имени из массива сигналов.

Структура класса Parser приведена на рисунке 17.

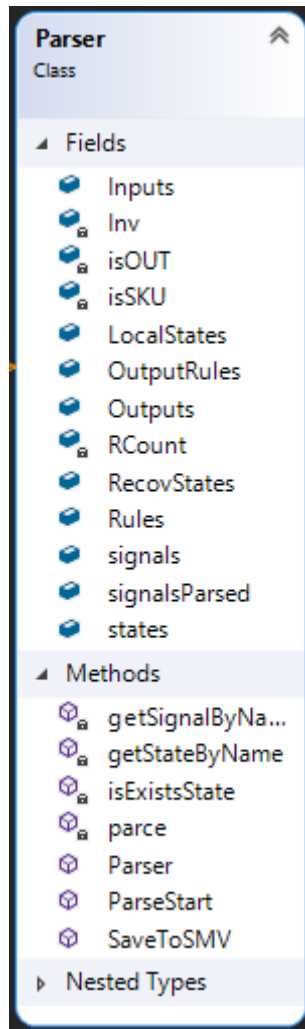


Рисунок 17 – UML-диаграмма класса Parser

Класс DrawArea отвечает непосредственно за отображение графической области взаимодействия с графом и отрисовку компонентов графа на этой области. Данный класс является наследником класса PictureBox и имеет доступ ко всем его не приватным методам и полям, вследствие чего происходит отрисовка необходим компонентов. Среди полей класса присутствует поле g типа Graphics, которое непосредственно предоставляет доступ к механизму отрисовки графики. Класс имеет следующие методы:

- *refreshArea(Graphics)* – перерисовывает компоненты графа.
- *createStates()* – выполняет инициализацию объектов состояний.
- *drawStates(Graphics)* – выполняет отрисовку состояний.
- *createSignals()* – выполняет инициализацию входных сигналов.
- *drawSignals(Graphics)* – выполняет отрисовку сигналов.
- *createLinks()* – выполняет инициализацию объектов переходов.

- *drawLinks()* – выполняет отрисовку переходов.
- *createTimeMarks()* – выполняет инициализацию временных меток.
- *drawTimeMarks(Graphics)* – выполняет отрисовку временных меток.
- *isDotOnLink(Point, Link)* – определяет, принадлежит ли точка текущего положения мыши конкретному переходу.
- *isDotOnState(Point, State)* - определяет, принадлежит ли точка текущего положения мыши конкретному состоянию.
- *isDotOnSignal(Point, Signal)* - определяет, принадлежит ли точка текущего положения мыши конкретному сигналу.
- *isDotOnTimeMark(Point, Time)* - определяет, принадлежит ли точка текущего положения мыши конкретной временной метке.
- *setActiveState()* – устанавливает выделенное состояние активным.
- *setActiveSignal()* – устанавливает выделенный сигнал единичным.
- *setValueTimeMark(Point)* – устанавливает значение выделенной временной метке.
- *relocationStates(Point)* – перерасчитывает координаты передвигаемых состояний.
- *mouseMove()* – обрабатывает событие движения мыши.
- *mouseDown()* – обрабатывает событие нажатия мыши.
- *mouseClick()* – обрабатывает событие клика мыши.

Структура данного класса представлена на рисунке 18.

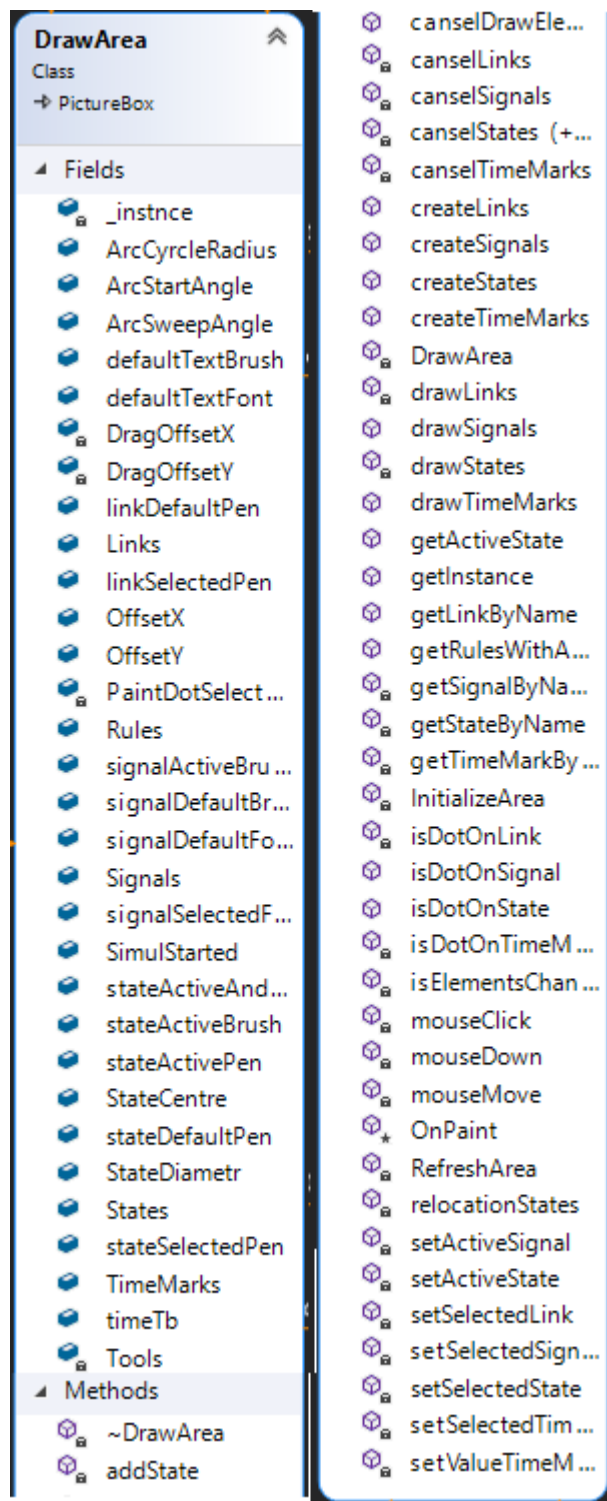


Рисунок 18 - UML-диаграмма класса DrawArea

2.3.3. Описание интерфейсов

Классы-интерфейсы отвечают за отображение основных окон. К таким классам относятся классы Form1 и ToolPanel

Класс Form1 представляет собой общий интерфейс среды. Внутри этого класса расположены основные элементы управления и настройки среды. Данный класс является наследником класса Form, и имеет доступ ко

всем неprivatным компонентам этого класса, благодаря чему реализован механизм управления окнами среды.

Класс `ToolPanel` описывает панель управления симуляцией системы. Данный класс является наследником класса `Control`. Благодаря подобной архитектуре реализован механизм управления симуляцией работы моделируемой системы. Класс имеет следующие методы:

- `onClickSimulStart()` – запускает симуляцию поведения моделируемой системы.

- `onClickStep()` – обрабатывает один шаг симуляции.

- `onClickSimulStop()` – завершает процесс симуляции.

2.3.4. Описание свойств

К классам свойств относится класс `Settings`, хранящий в себе основные настройки системы. Доступ к этому классу реализован средствами самого языка, вследствие чего класс доступен из любого места программы.

Общая схема взаимодействия классов представлена на рисунке 19.

алгоритм разбора введенной пользователем системы канонических уравнений, описывающих моделируемую систему, алгоритм отображения и перерисовки компонентов временного автомата, представляющего исходную систему, алгоритм взаимодействия с элементами автомата, алгоритм симуляции работы моделируемой системы.

3. Описание результата

Данный раздел содержит описание интерфейса и руководство пользователя приложения «Инструментальная система темпорального анализа моделей параллельных вычислительных систем». Рекомендуемый уровень владения компьютером пользователя – уверенный. Пользователю необходимо владеть навыками работы с оконными приложениями и иметь представление о теории автоматов.

3.1. Описание работы реальной системы

Предположим, перед пользователем стоит задача, спроектировать и смоделировать работу некой вычислительной системы. Для простоты понимания в качестве моделируемой системы пусть будет выступать механизм работы стиральной машинки. Описать алгоритм ее работы можно следующим образом.

1. Машинка находится в исходном состоянии и готова к стирке
2. По нажатию пользователем кнопки «Старт» в течении определенного происходит набор воды в барабан
3. После заполнения барабана до должного уровня водой начинается процесс стирки.
4. Процесс стирки продолжается какое-то количество времени и при условии, что в барабане присутствует вода.
5. После того как стирка закончена в течении определенного времени происходит откачка воды из барабана.
6. Как только вся вода откачена, датчик уровня воды посылает сигнал означающий, что стирка закончена и машинка переходит в конечное состояние.
7. Машинка находится в конечном состоянии до тех пор, пока дверца барабана открыта. Как только дверца закрыта, поступает соответствующий сигнал, и машинка переходит вновь в начальное состояние.

Весь описанный алгоритм можно представить в виде временного автомата, изображенного на рисунке 20, где S_0 – начальное состояние машинки, S_1 – состояние стирки, S_2 – состояние откачки воды после стирки,

S_k – конечное состояние. X_1 – сигнал нажатия кнопки «Старт», X_2 – сигнал с уровня датчика воды, свидетельствующий о наличии необходимого уровня воды, X_3 – сигнал с замка машинки, означающий что замок закрыт. T_1 – время набора воды в машинку, T_2 – время стирки.

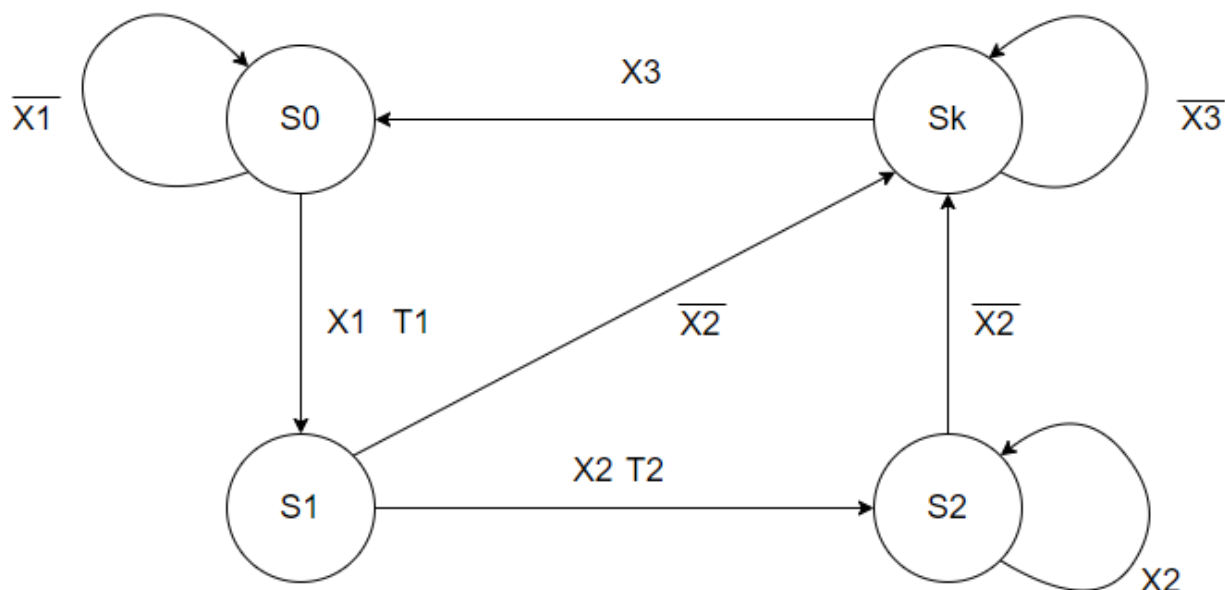


Рисунок 20 - алгоритм работы стиральной машинки, описанный посредством временного автомата

С точки зрения описания работы машинки через временной автомат алгоритм выглядит следующим образом.

1. Система находится в начальном состоянии S_0 .
2. Система переходит из начального состояния само в себя же при отрицательном сигнале X_1 .
3. Система переходит из состояния S_0 в состояние S_1 под действием сигнала X_1 и прошествии T_1 тактов времени.
4. Система переходит из состояния S_1 в состояние S_k под действием отрицательного сигнала X_2 .
5. Система переходит из состояния S_1 в состояние S_2 под действием сигнала X_2 и по прошествии T_2 тактов времени.
6. Система переходит из состояния S_2 в себя же под действием сигнала S_2 .
7. Система переходит из состояния S_2 в состояние S_k под действием отрицательного сигнала X_2 .

8. Система переходит из состояния S_k в себя же под действием отрицательного сигнала X_3 .

9. Система переходит из состояния S_k в состояние S_0 под действием сигнала X_3 .

Система канонических уравнений, описывающий данный автомат будет выглядеть следующим образом.

3.2. Руководство пользователя

В разработанной системе темпорального анализа моделей вычислительных систем весь описанный алгоритм работы можно спроектировать и промоделировать его поведение. Достаточно просто описать вычислительную систему с помощью системы канонических уравнений.

3.2.1. Описание интерфейса среды моделирования

В верхней части окна расположены основные вкладки для работы со средой моделирования.

Во вкладке «Файл» представлены основные инструменты работы с текстовым пространством и средой.

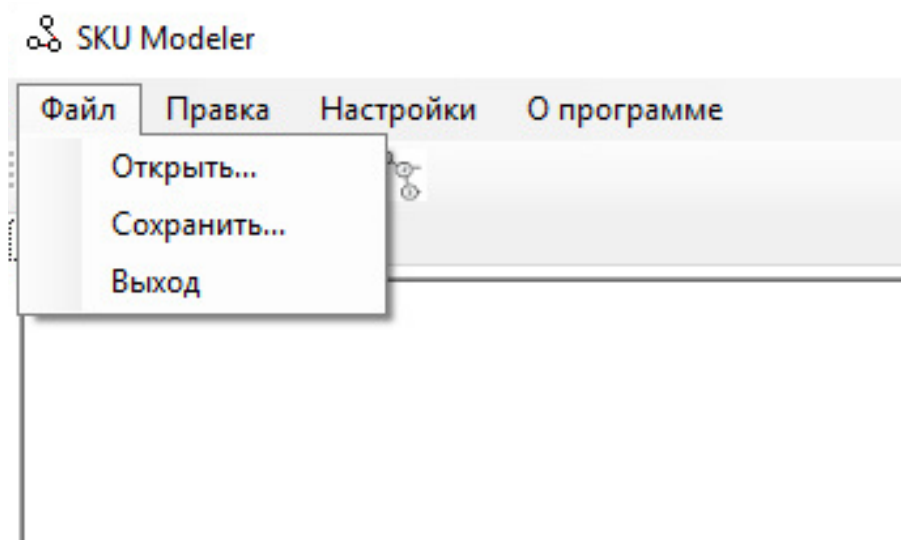


Рисунок 21 – Вкладка «Файл»

1. Пункт «Открыть» открывает текстовый файл для дальнейшего

редактирования.

2. Пункт «Сохранить» сохраняет текущий текст из области для ввода SKU в текстовый файл.

3. Пункт «Выход» завершает рабочий процесс и закрывает программу.

Во вкладке «Правка» расположены инструменты работы с графическим пространством, в котором располагается граф автомата моделируемой системы.

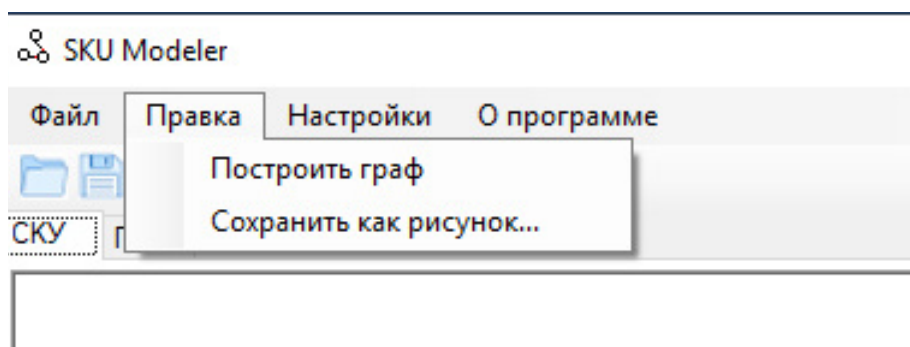


Рисунок 22 - Вкладка «Правка»

1. Пункт «Построить граф» выполняет разбор введенной SKU и на основе результата разбора строит граф автомата моделируемой системы

2. Пункт «Сохранить как рисунок» сохраняет построенный граф в файл с расширением .bmp как картинку.

Вкладка «Настройки» содержит основные настройки среды моделирования.

1. Во вкладке «Общие» расположены настройки автосохранения данных в текстовой области, в частности периодичность сохранения. Также здесь присутствует возможность менять расположение панели инструментов, управляющих графом. Эту панель можно расположить 4 разными образами, слева, сверху, справа и снизу. По умолчанию панель расположена слева. Также здесь можно включить подсветку синтаксиса.

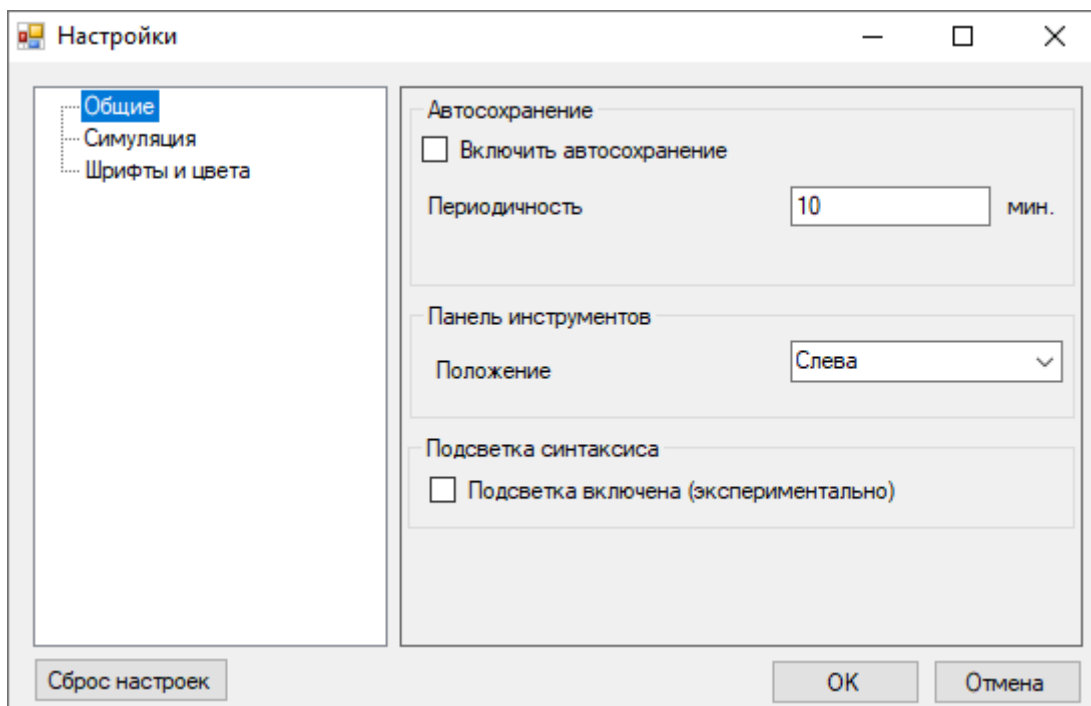


Рисунок 23 – Общие настройки

2. Во вкладке «Симуляция» расположены настройки симуляции поведения моделируемой системы, такие как скорость симуляции, обозначающая сколько реального времени в миллисекундах, занимает такт работы моделируемой системы. Также здесь можно включить запись симуляции с последующим выводом в файл для последующего анализа результатов моделирования.

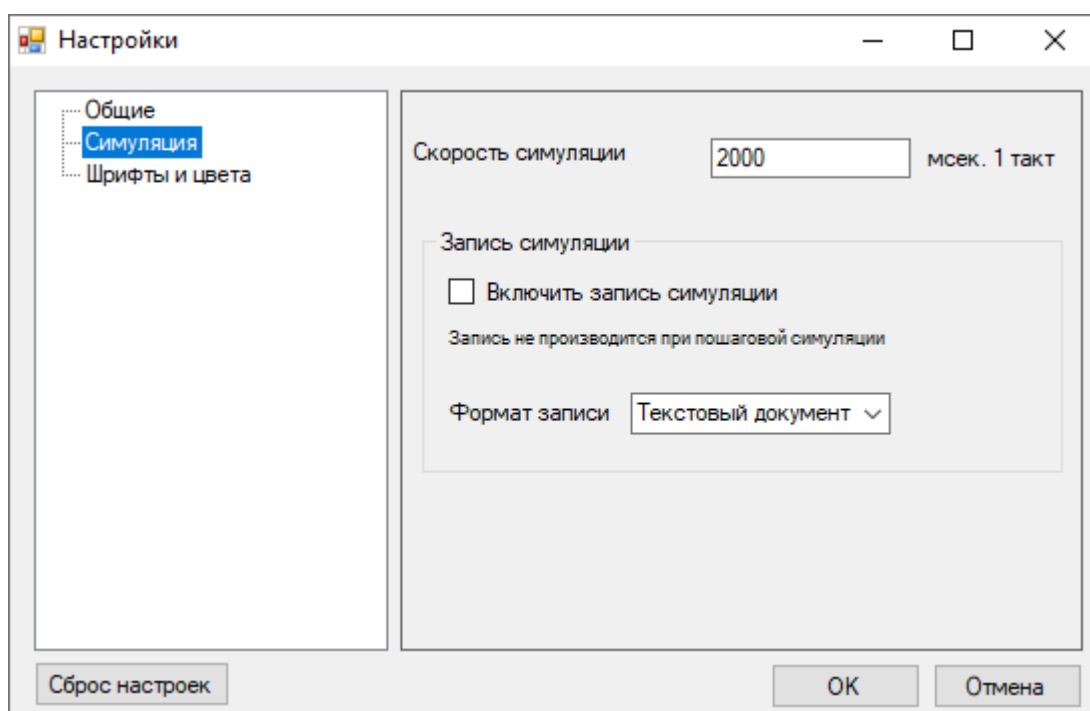


Рисунок 24 - Настройки симуляции моделируемой системы

3. Во вкладке «Шрифты и цвета» расположены настройки

текстовой области системы, где можно выбрать предпочитаемые стили и цвета текста разных элементов описания СКУ.

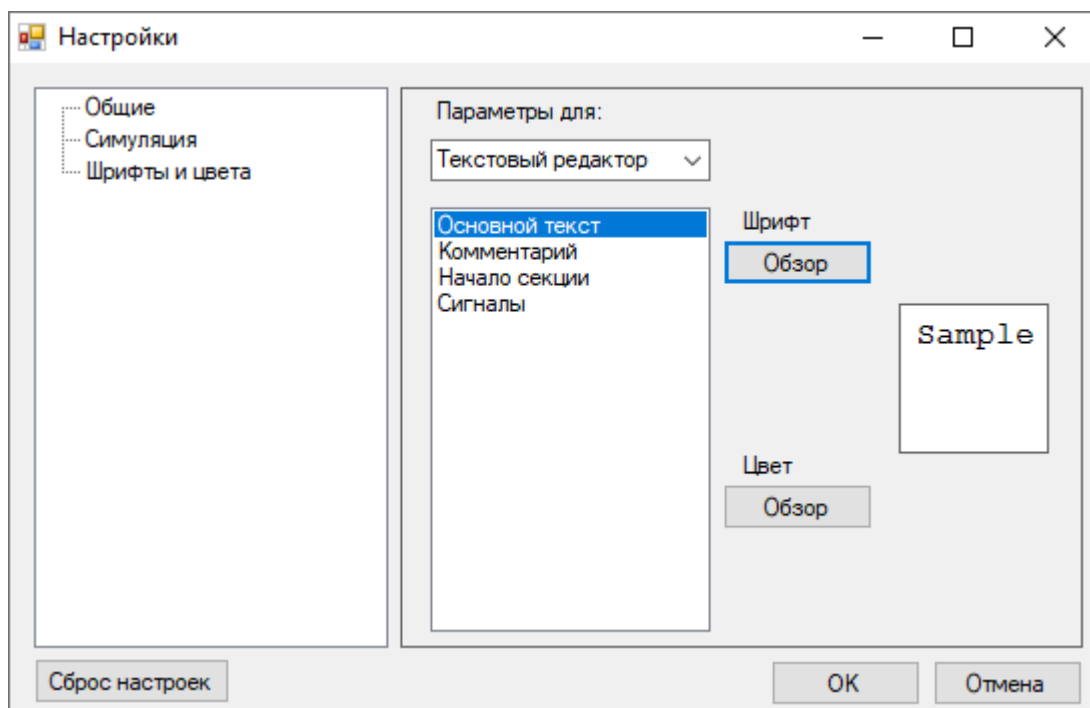


Рисунок 25 - Настройки оформления текстового пространства

Вкладка «О программе» содержит сведения об авторе и разработчике среды моделирования.

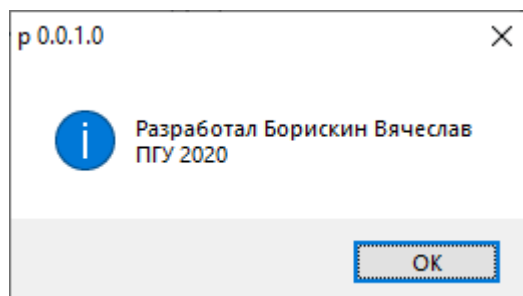


Рисунок 26 - Вкладка "О программе"

В главном окне разработанной среды расположена текстовая область для ввода системы канонических уравнений, описывающих исходную систему.

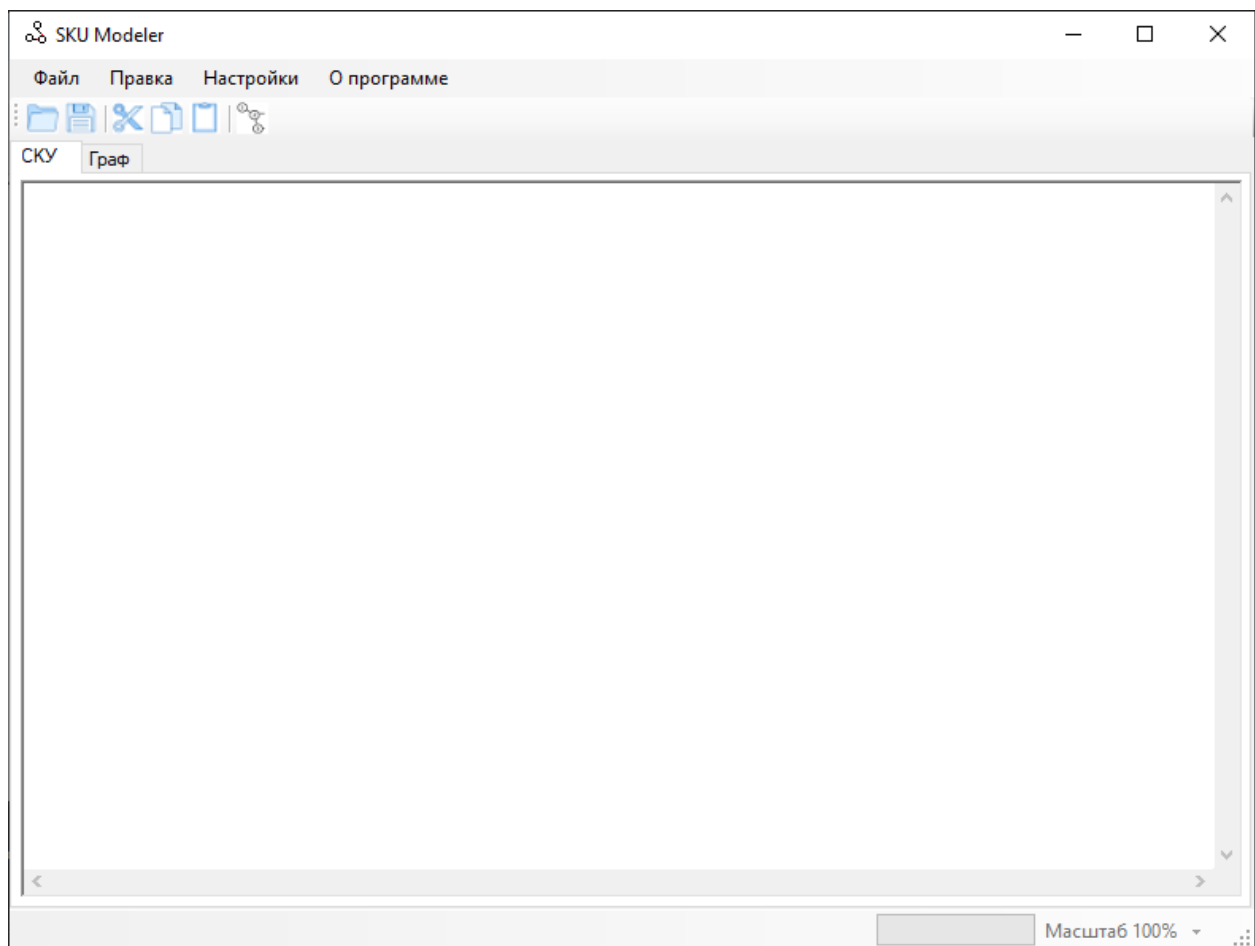


Рисунок 27 – Текстовое пространство среды моделирования

В верхней части текстового пространства расположены основные органы управления текстом.



Рисунок 28 - Органы управления текстовым пространством

1. Открыть текстовый файл для дальнейшего редактирования.
2. Сохранить текущие результаты в текстовый файл.
3. Вырезать выделенный текст в буфер обмена.
4. Копировать выделенный текст в буфер обмена.
5. Вставить текст из буфера обмена.
6. Построить граф моделируемой системы на основе введенной пользователем СКУ.

3.2.2. Описание процесса моделирования

Чтобы промоделировать систему пользователю необходимо ввести СКУ в текстовое поле. Как и в любом языке программирования существуют

ряд синтаксических правил, которые пользователь должен соблюсти для достижения корректного результата разбора SKU.

1. В первой строке указывается ключевое слово «input», после которого в квадратных скобках следует перечисление всех входных сигналов моделируемой системы.

2. Далее следует набор правил, которые указывают из какого состояния и в какое состояние, и под действием каких сигналов и временных задержек должна перейти система.

3. Как и в подавляющем большинстве языков программирования описание каждого правила должно заканчиваться символом точки с запятой. Для удобочитаемости каждое правило следует располагать на новой строке.

На рисунке 13 изображен ввод SKU.

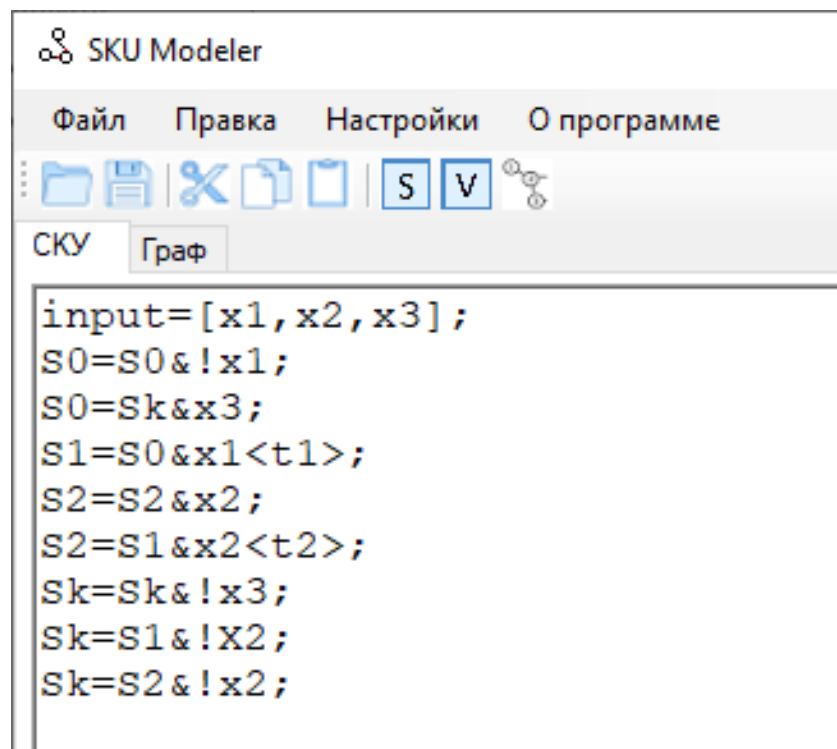


Рисунок 29 - Ввод SKU

После ввода SKU пользователю необходимо нажать кнопку «Построить граф». В этот момент средствами среды происходит разбор введенной SKU, создаются объекты состояний переходов и временных меток, генерируются объекты связей между состояниями. По мере завершения процесса разбора в нижней части интерфейса среды заполняется шкала завершенности, представленная на рисунке 30.

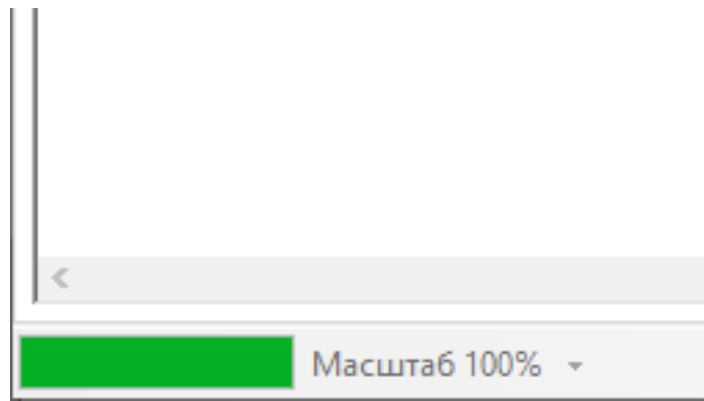


Рисунок 30 - Шкала завершенности разбора

В тот момент, когда разбор завершился, во вкладке граф происходит поочередное визуальное отображение каждого объекта, созданного в процессе разбора. Перед тем, как отрисовать объект, алгоритм среды моделирования рассчитывает координаты вершин каждого графического элемента. Координаты рассчитываются относительно левого верхнего угла графической области. После того, как координаты вершин каждого элемента вычислены, происходит поочерёдное отображение объектов в соответствии с рассчитанными координатами, представленное на рисунке 31.

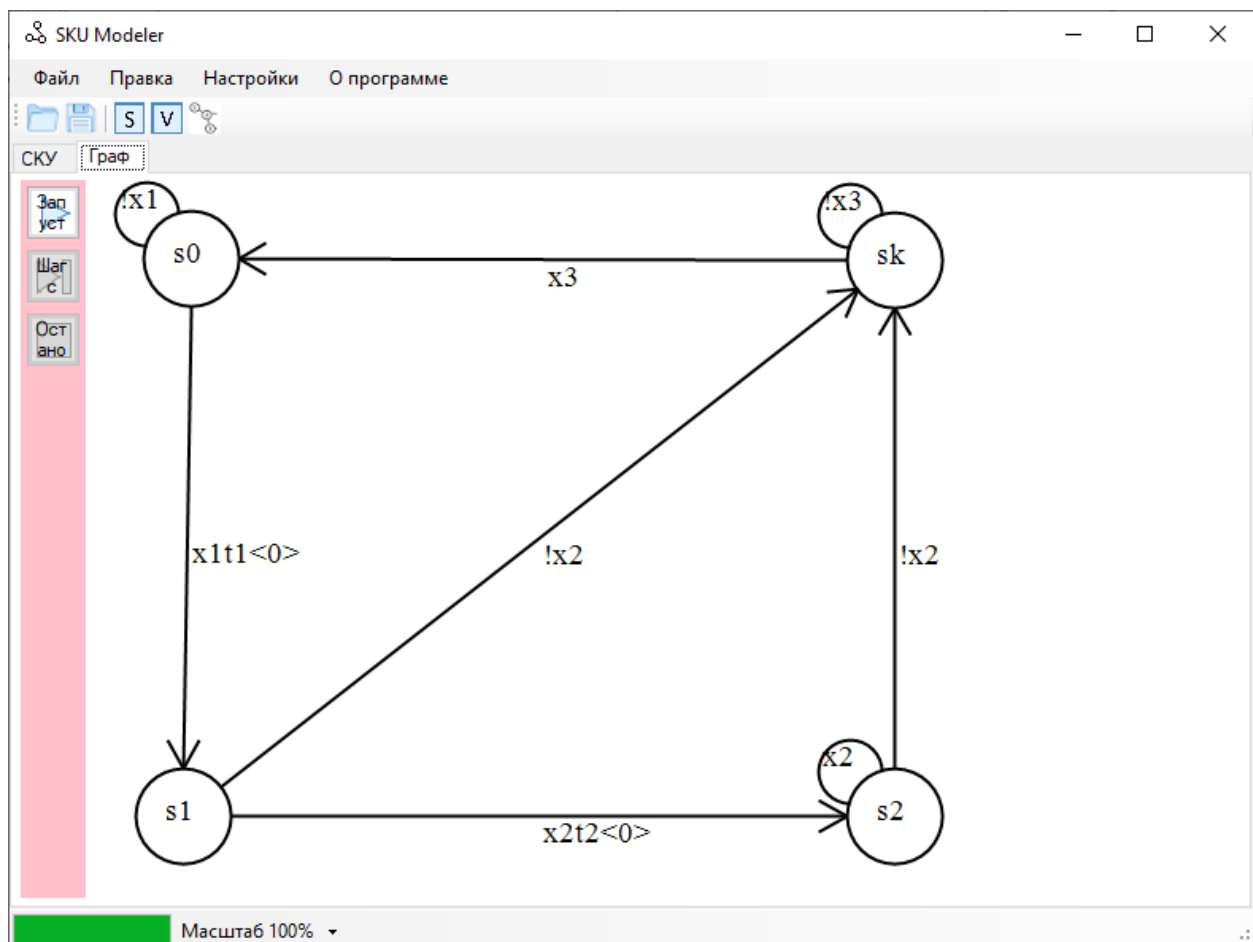


Рисунок 31 - граф автомата, описывающего моделируемую систему

Окружностями представлено множество состояний моделируемой системы. Направленными отрезками – множество переходов. Стрелкой указывается направление перехода, от исходного состояния к конечному. Символами представлено множество входных сигналов. В угловатых скобках указывается имя временной метки и значение, измеряемое в миллисекундах, по истечению которого срабатывает переход.

Элементы графа интерактивны. Пользователь в полной мере может взаимодействовать с ними, перемещать в зависимости от своих предпочтений, выделять, устанавливая элементы в сигнальное состояние и т.д. На рисунке 32 продемонстрирован данный эффект.

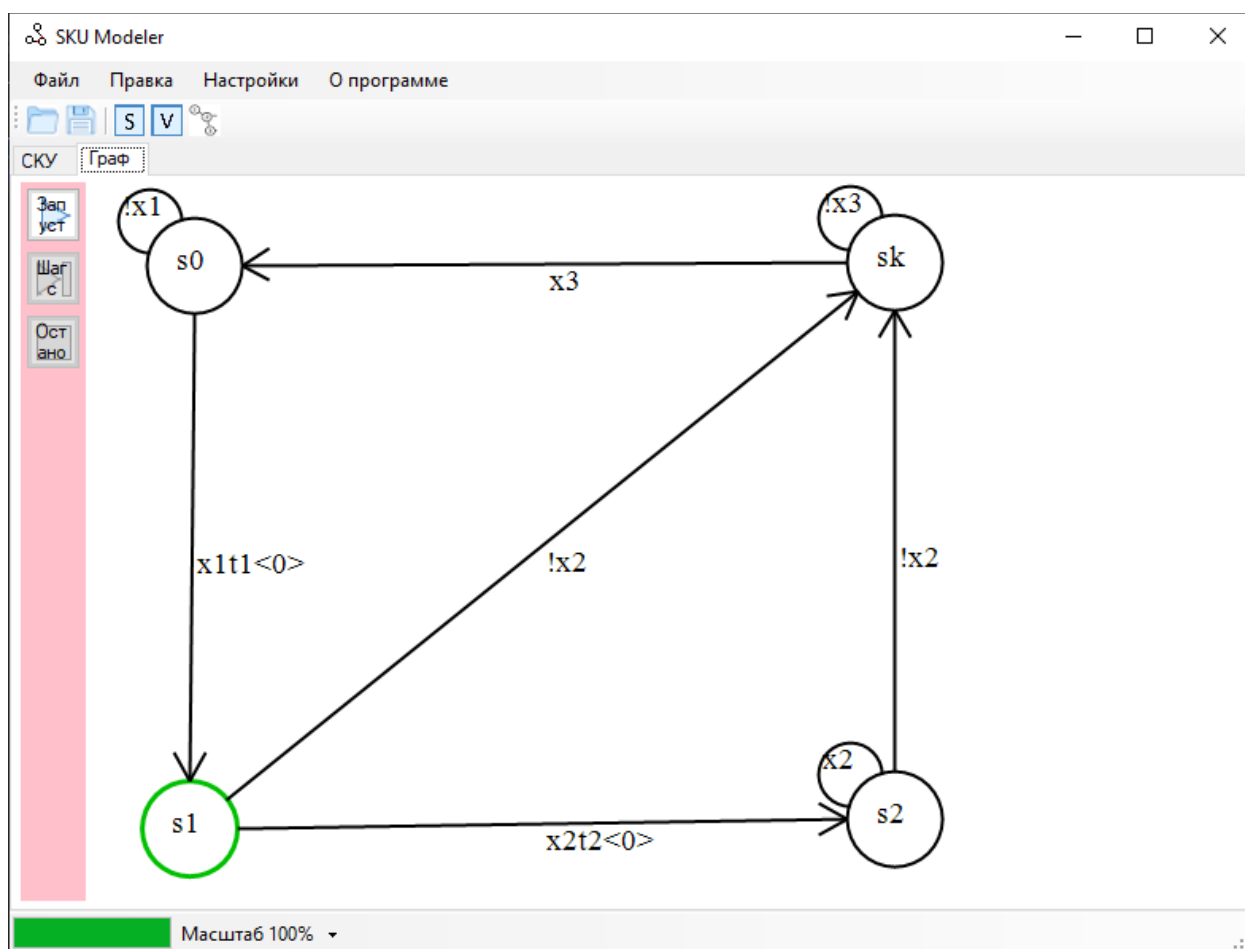


Рисунок 32 - Выделение состояния при наведении курсора мыши

Чтобы состояние автомата выделилось зеленым цветом как на рисунке, пользователю достаточно навести курсор мыши на область графического пространства, принадлежащую данному состоянию.

Подобным образом подсвечиваются отрезки, обозначающие переходы из одного состояния в другое, достаточно навести на них курсор мыши. Данный эффект продемонстрирован на рисунках 33, 34.

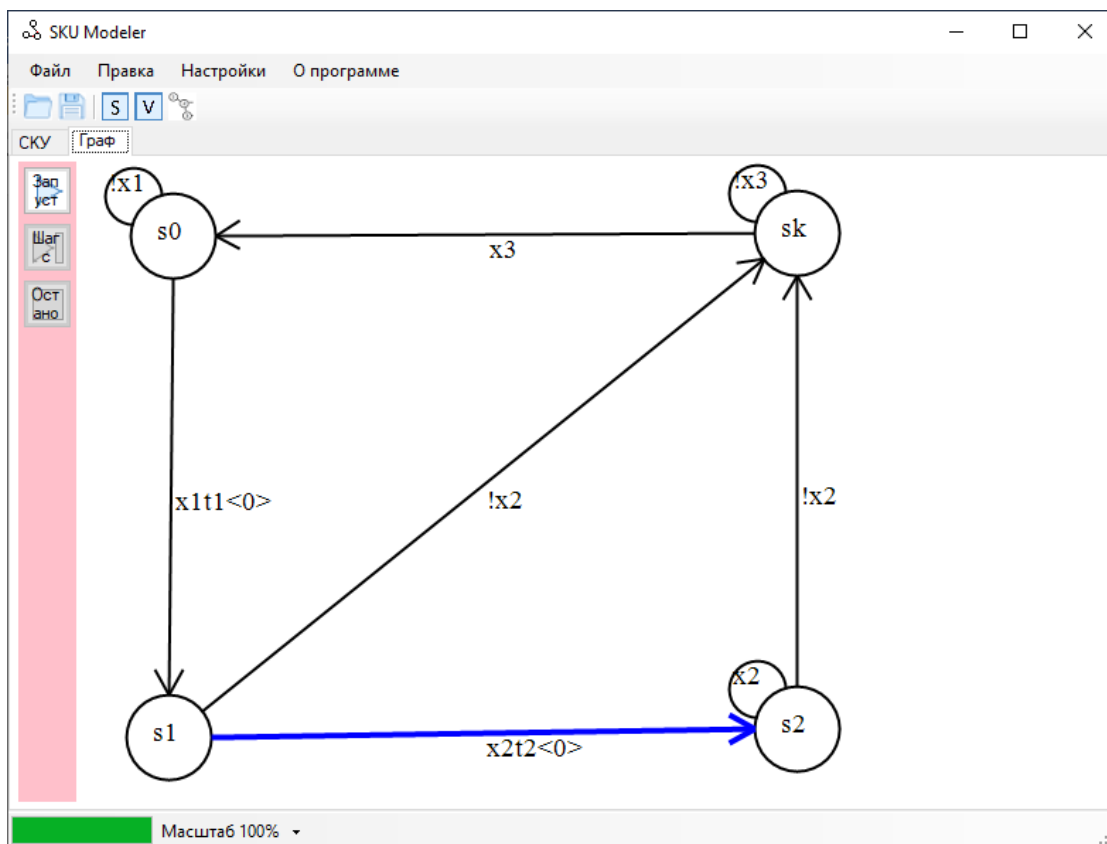


Рисунок 33 - выделение перехода при наведении курсора мыши

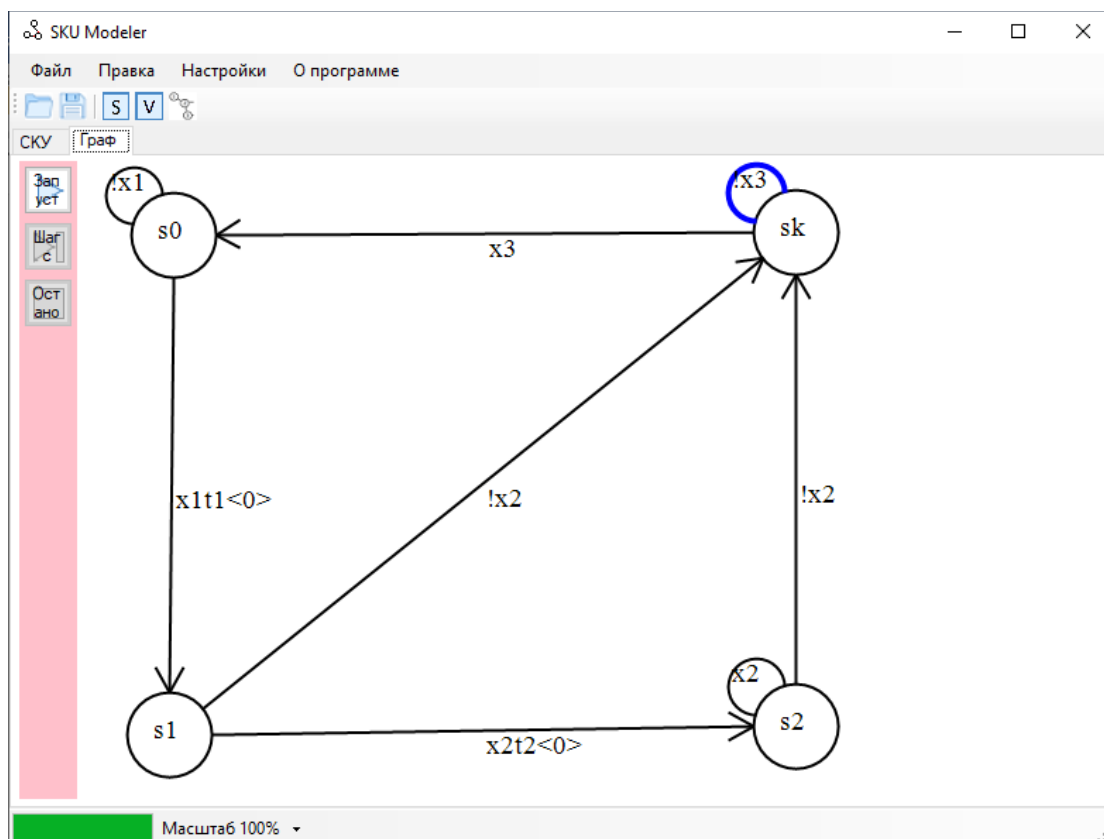


Рисунок 34 - выделение перехода при наведении курсора мыши

Помимо состояний и переходов среда моделирования может взаимодействует со входными сигналами и временными метками. Сигналы и временные метки также могут быть в выделенном состоянии. Этот эффект продемонстрирован на рисунках 35, 36.

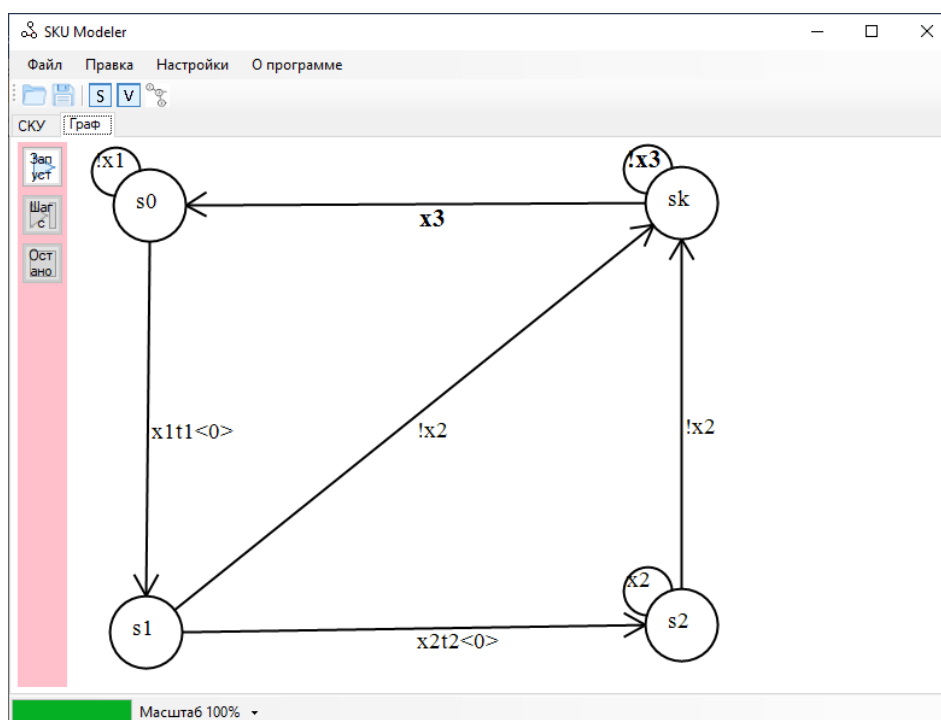


Рисунок 35 - Выделение входного сигнала

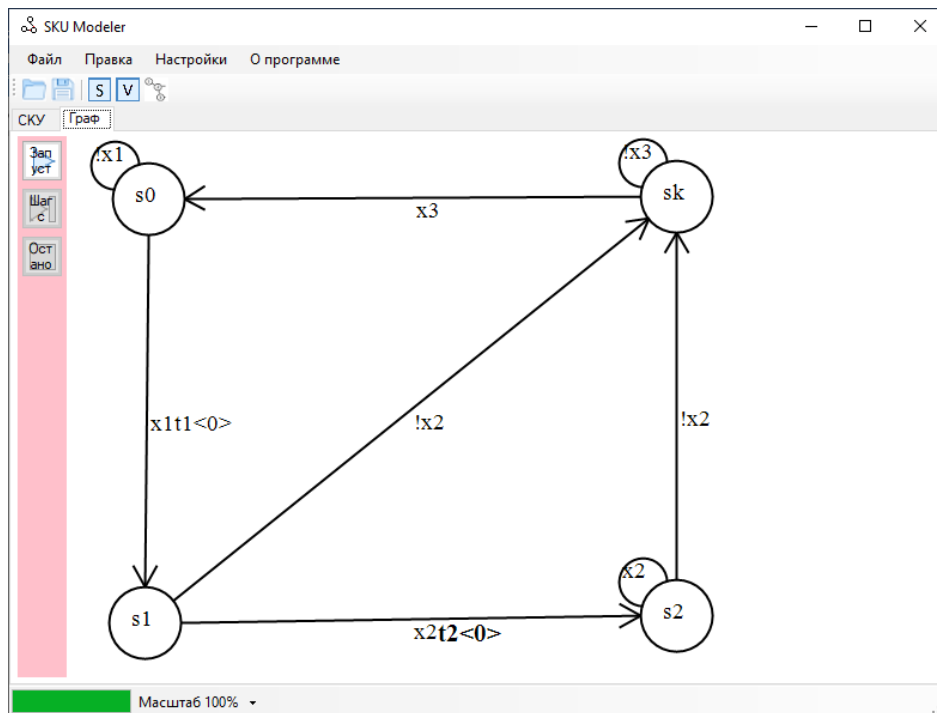


Рисунок 36 - Выделение временной метки по наведению курсора мыши

Помимо выделения компонентов графа наведением мыши, так же у пользователя есть возможность перетаскивать элементы состояний по всей графической области для максимального удобства работы со средой. Данный эффект продемонстрирован на рисунке 37.

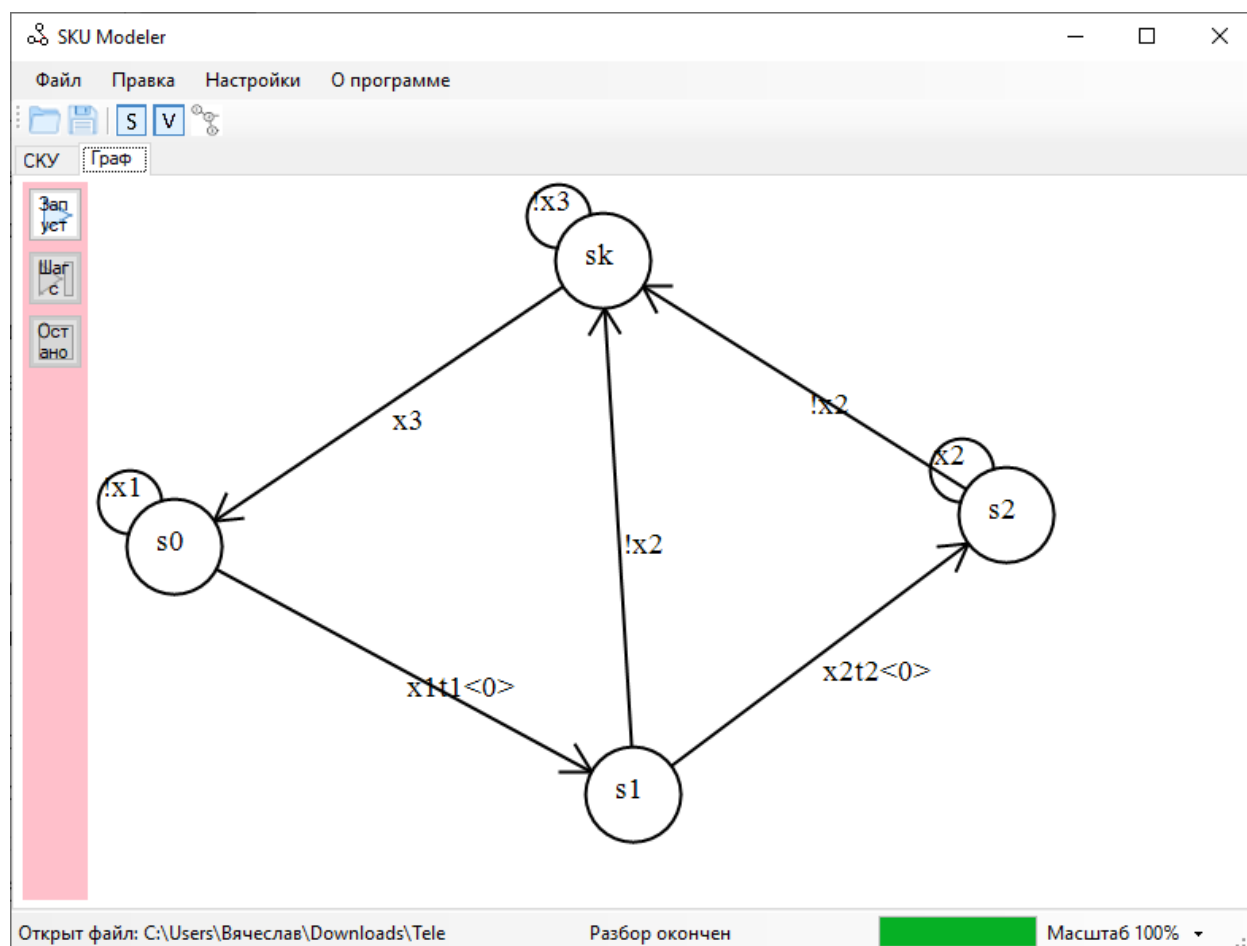


Рисунок 37 - Перемещение элементов состояний

Для того чтобы переместить какое -либо состояние, необходимо навести на него курсор мыши, чтобы оно подсвечивалось и перетаскиванием мыши переместить его в предпочитаемое место.

Для проведения симуляции работы моделируемой системы необходимо реализовать механизм управления состояниями и входными сигналами. Этот механизм реализован следующим образом. Перед началом симуляции пользователь должен выбрать начальное состояние моделируемой системы. Для этого достаточно кликнуть по нужному состоянию, в следствие чего объект этого состояния примет единичное значение, что можно наблюдать на рисунке 38.

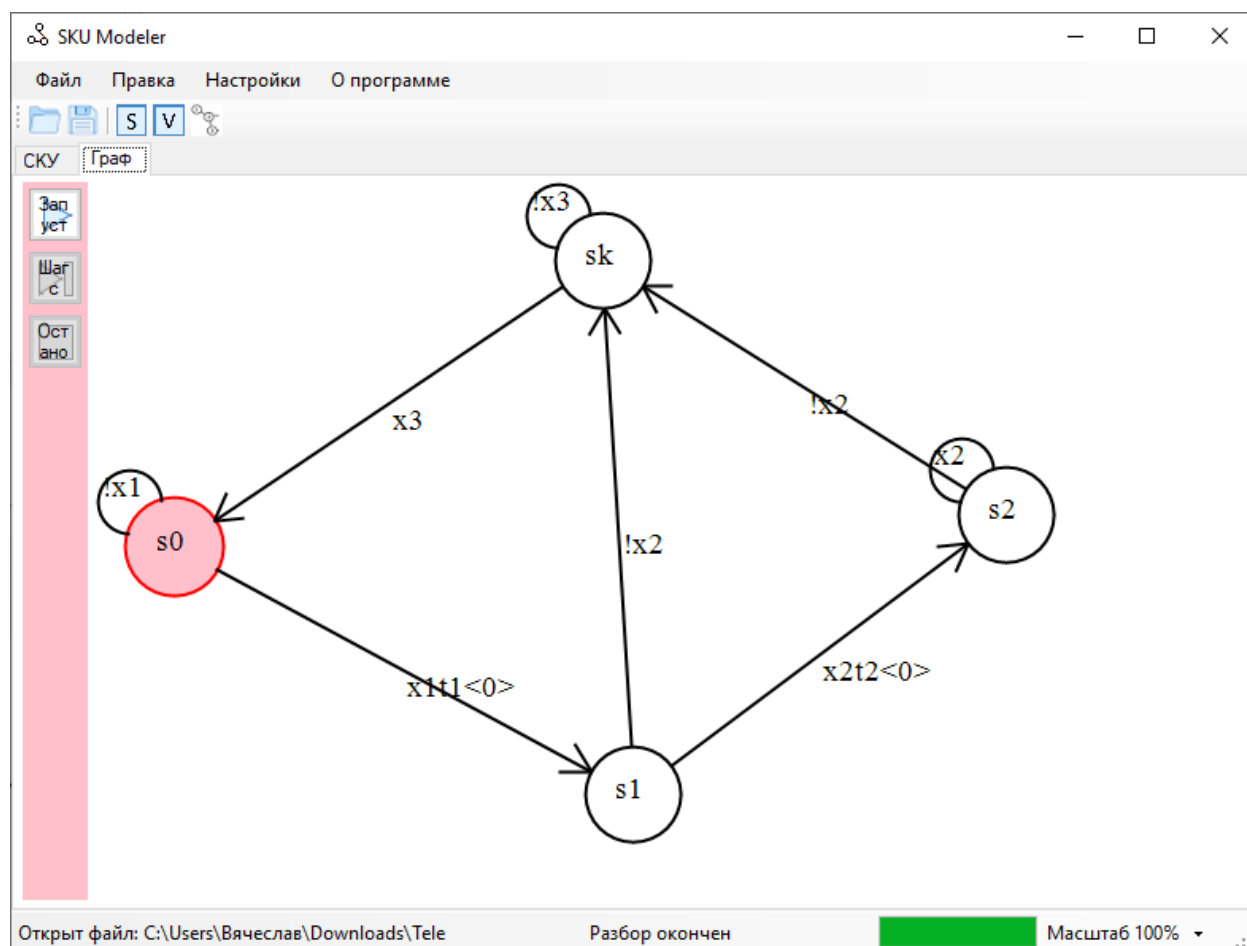


Рисунок 38 - Выбор начального состояния

В один момент времени может быть только одно начальное состояние. Если в среде моделирования выбрано какое-либо начальное состояние, и пользователь попытается выбрать еще одно, то в качестве начального состояния среда выберет последнее, по которому было произведено нажатие левой кнопки мыши, а предыдущее состояние сбросится.

Также пользователю предоставляется возможность указать значения входных сигналов. Сделать это можно аналогично, кликом левой кнопки мыши по нужному сигналу. Результат изображен на рисунке 39.

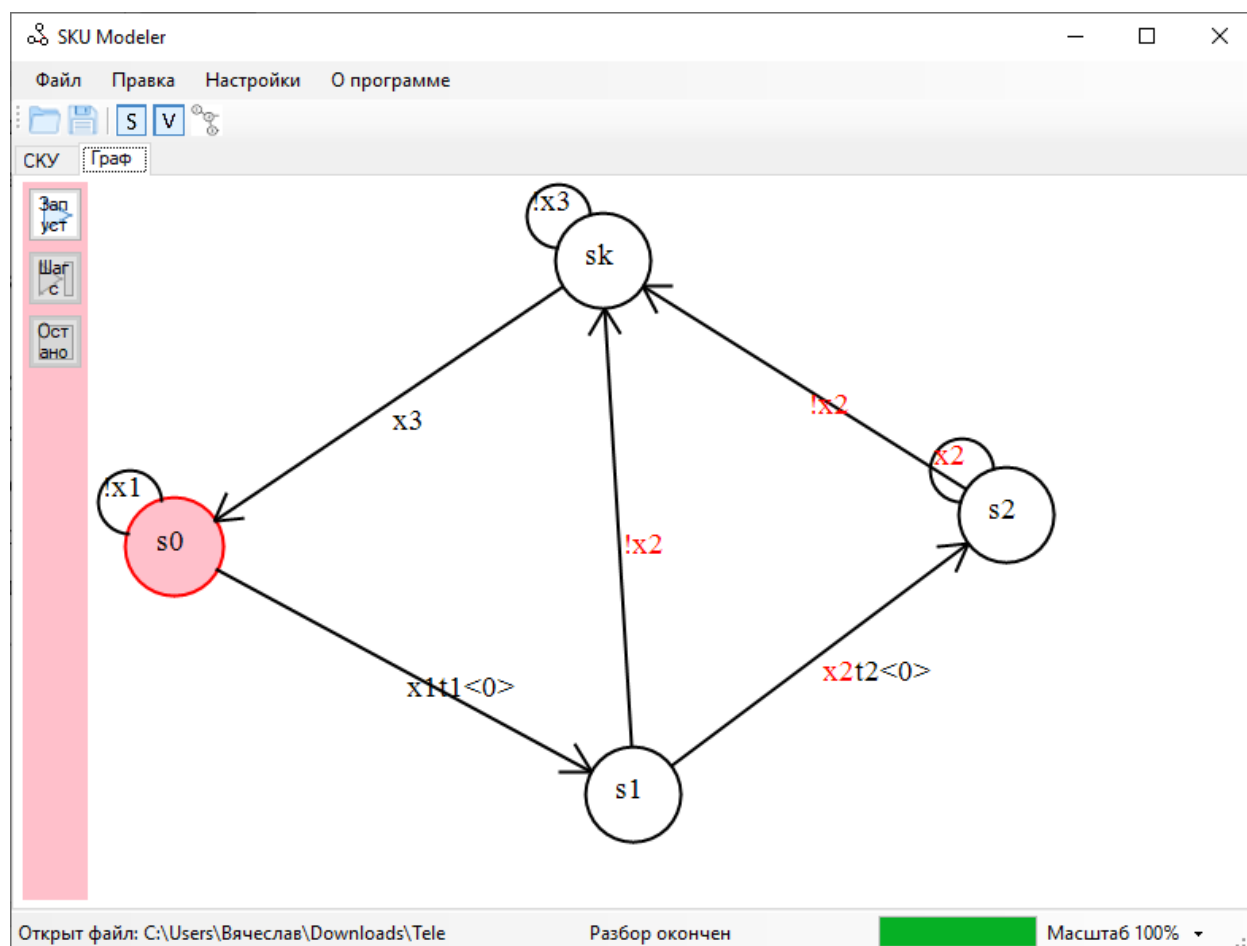


Рисунок 39 - Установка значения входного сигнала

Подобным образом пользователь может установить значение временной метки, по истечению которого должен сработать переход. Значение измеряется в миллисекундах. Чтобы установить необходимое значение достаточно кликнуть по требуемой временной метки, после чего рядом с выделенной меткой отобразится текстовое поле для ввода значения. Результат продемонстрирован на рисунке 40.

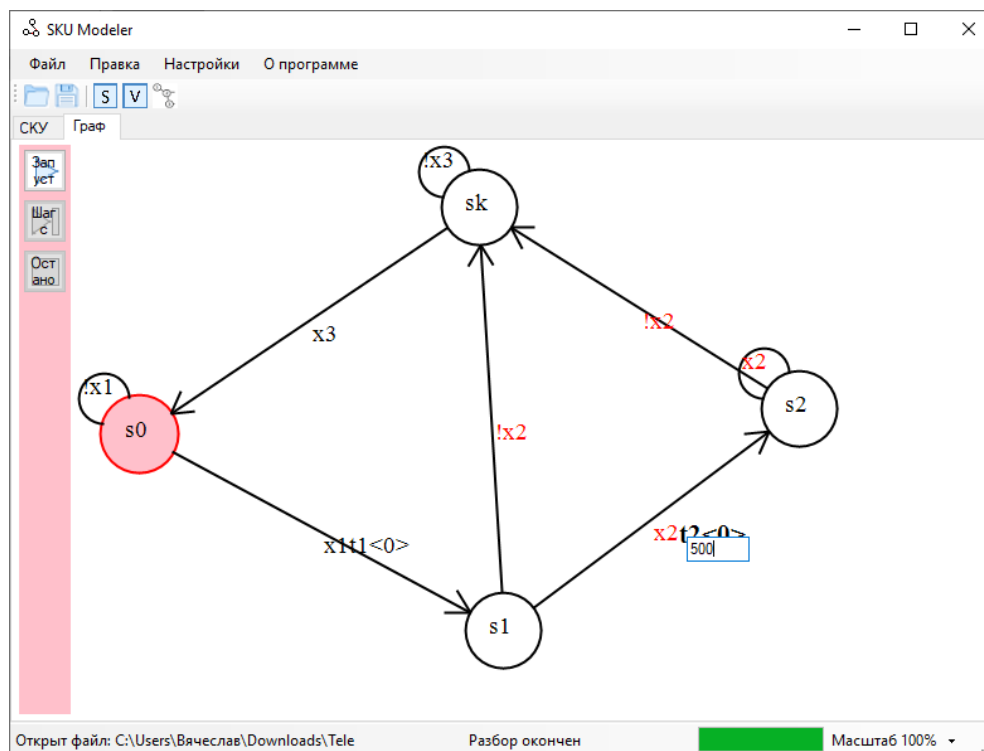


Рисунок 40 - Установка значения временной метке

После того, как поле потеряет фокус, введенное значение будет применено к данному объекту временной метки, что можно видеть на рисунке 41.

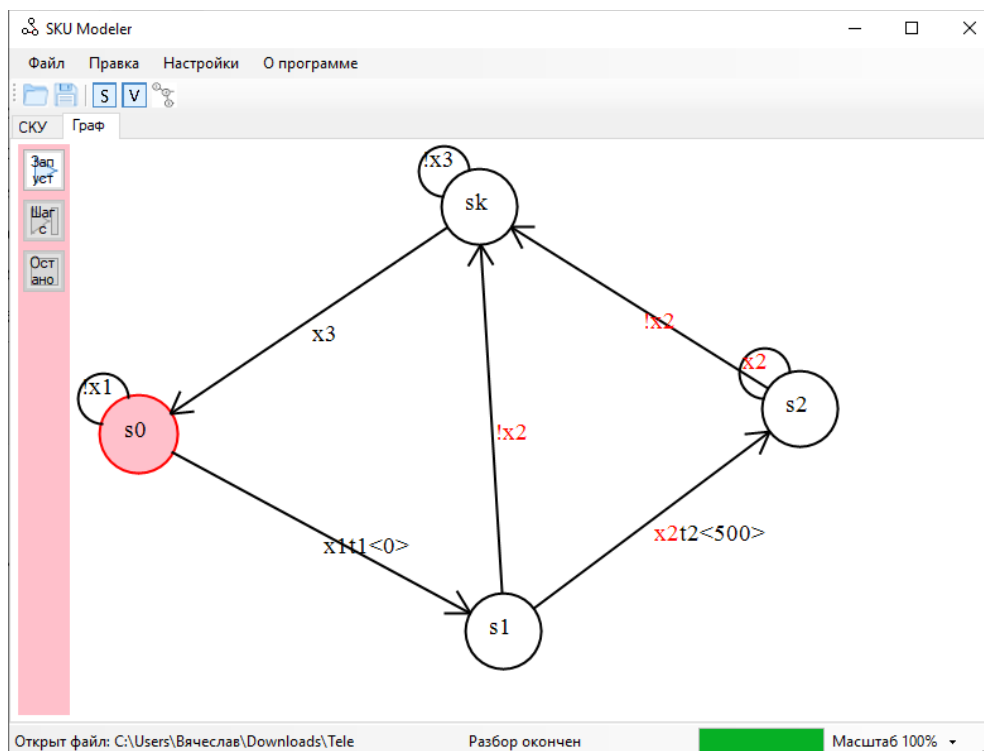


Рисунок 41 - Применение установленного значения к временной метке

После того, как все инициализирующие операции были завершены, пользователь может приступить к симуляции работы моделируемой системы.

Для начала симуляции необходимо нажать кнопку «Запустить», расположенную на панели в левой части окна.

После нажатия кнопки среда моделирования проверяет, указана ли какое-либо начальное состояние. Если начальное состояние не указано, среда предупредит об этом. Результат можно наблюдать на рисунке 42.

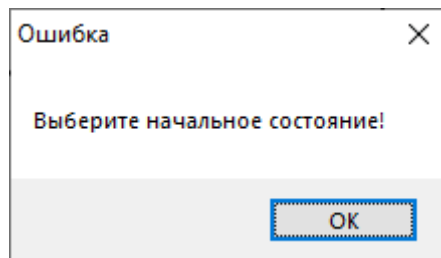


Рисунок 42 - Ошибка при отсутствии начального состояния

Если начальное состояние установлено среда приступает к моделированию поведения. Исходя из информации о активности входных сигналов и величины временных задержек система на каждом шаге определяет состояние в какое она переходит, либо среда остается в исходном состоянии если никакой переход еще не сработал. Цикл продолжается до тех пор, пока пользователь не нажмет кнопку «Стоп», расположенную там же, где и кнопка «Старт». На рисунке 43 видно, как система переходит из одного состояния в другое под действием сигналов.

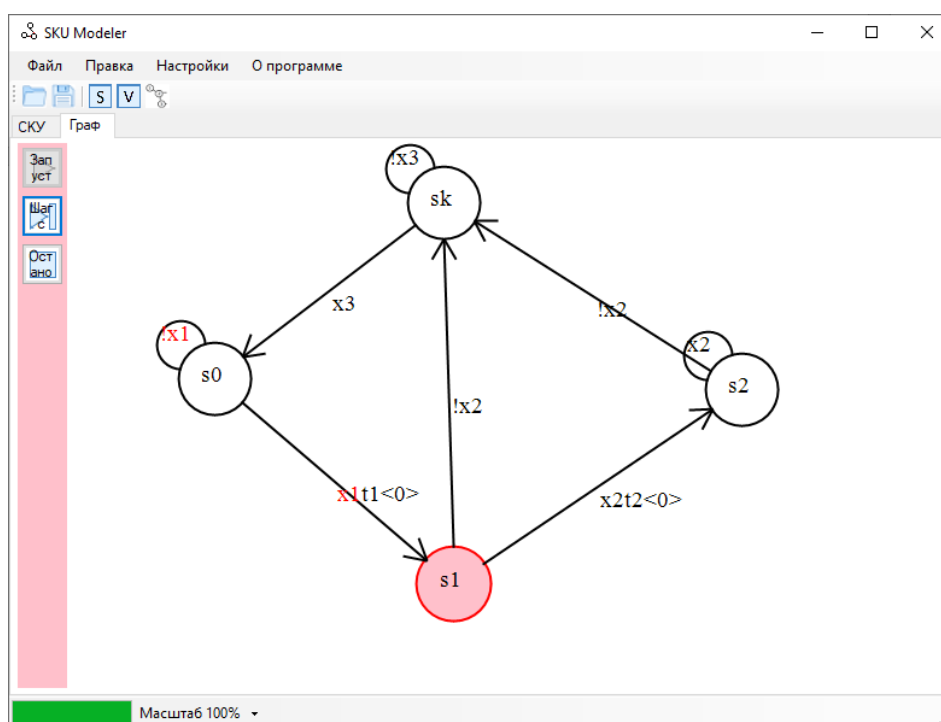


Рисунок 43 - Переход системы в другое состояние

Также в среде предусмотрена возможность пошагового моделирования поведения. Чтобы активировать данный режим достаточно нажать кнопку «Шаг с остановом». После чего станет доступным пошаговое моделирование. Результат продемонстрирован на рисунке 44.

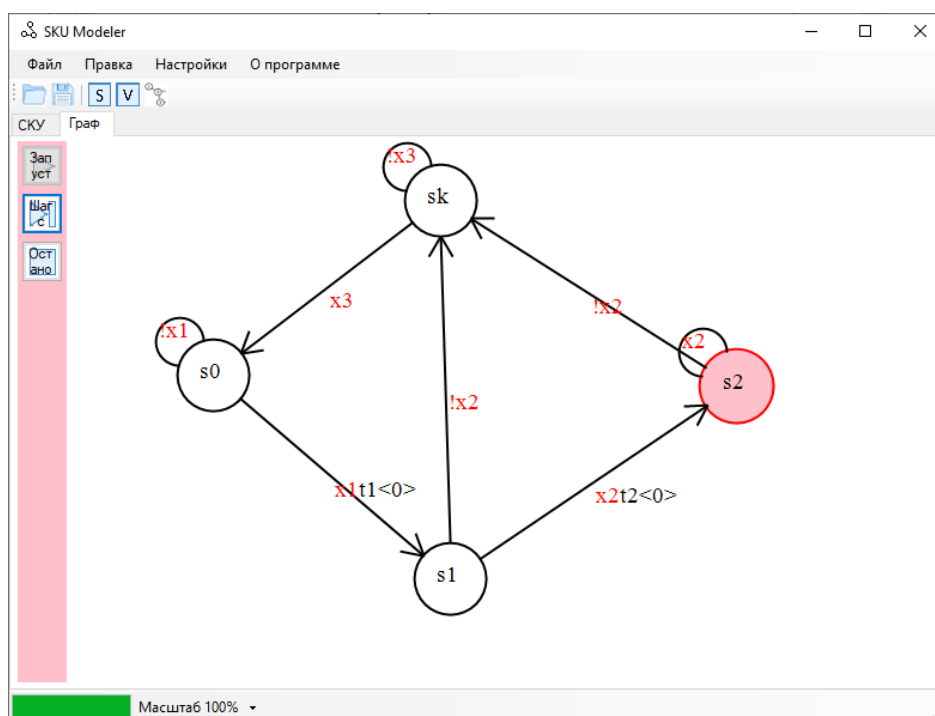


Рисунок 44 - Пошаговое моделирование поведения

Цикл повторяется до тех пор, пока пользователь не нажмет кнопку «Остановить», расположенную там же, на панели слева.

3.3. Выводы

Результатом разработки собственной среды моделирования вычислительных систем, стал инструмент, позволявший проводить проектирование сложных вычислительных систем, моделирование поведения систем, подразумевающих в своей работе временные задержки, анализ систем с целью выявления узких мест и уязвимостей еще на стадии проектирования.

4. Охрана труда

Воздействие персональных компьютеров на здоровье человека является важной проблемой современного общества. Чтобы огородить себя от вредного влияния компьютера на зрительную систему необходимо соблюдать соответствующие меры безопасности, правильно планировать режим работы и выбирать рабочее место.

Искусственное освещение, используемое в помещениях, где работники взаимодействуют с персональными компьютерами, обязано советовать определённым требованиям.

Для этого выделяют следующие требования к освещению на рабочих местах, оборудованных персональными электронно-вычислительными машинами:

- освещенность рабочего места должна составлять около 300 – 500 лк [11]. Запрещается работать в недостаточно освещенном помещении, а также в помещении с избыточным количеством света;
- искусственное освещение в помещениях для эксплуатации ПЭВМ должно быть распределено равномерно;
- предпочтительнее всего, чтобы солнечный свет падал слева от рабочего стола. Нельзя располагать рабочее место с настольным экраном таким образом, чтобы в поле зрения пользователя не попадали окна или осветительные приборы, а также они не должны находиться позади работника;
- яркость бликов на дисплее ПЭВМ не должна превышать 40 кд / м² и яркость потолка не должна превышать 200 кд / м².

При выборе источников общего искусственного освещения лучше обратить внимание на люминесцентные лампы с рассеивателями или экранирующими решетками, т.к. они распределяют вырабатываемый свет равномерно и исключают блики на экране монитора и клавиатуре. Соотношение яркости экрана и непосредственно ближайшего окружения не должно превышать 3:1.

Помимо перечисленного, на рабочие места, оборудованные ПЭВМ, рекомендуется устанавливать локальные источники света – настольные лампы с регулируемой яркостью и положением источника света.

Эти требования обязательно соблюдать, т.к. без правильного распределения источников света в помещении, зрение сотрудников может значительно ухудшаться, что приводит к проблемам со здоровьем, в частности со зрительным аппаратом и усталости.

Помимо правил организации освещенности рабочего места существуют правила организации рабочего стола:

- расстояние между рабочими столами с видеодисплеями должно быть не менее 2 м, а расстояние между боковыми сторонами видеодисплеев – не меньше 1,2 м;
- экран видеодисплея должен находиться на расстоянии 60 - 70 см от глаз оператора, но не ближе 50 см с учетом размеров графических объектов, расположенных на видеодисплее;
- конструкция рабочего стола должна удовлетворять условиям оптимального размещения на поверхности рабочего стола различного вида оборудования с учетом его особенностей его конструкции и характера выполняемой работы;
- обязательно иметь рабочее кресло с возможностью регулировать свое положение вокруг своей оси, по высоте от пола и относительно плоскости пола;
- сиденье и спинка стула должны быть покрыты не электризующимися дышащими материалами;

Помимо перечисленных условий, также желательным является факт использования защитных очков, при работе с видеодисплеем. Такие очки защищают пользователя от вредного воздействия монитора на зрительную систему.

Для продуктивной и комфортной работы с ПЭВМ необходимо обеспечить оптимальные значения температуры воздуха в помещении. Нормальный диапазон температур составляет от +18 до +23 градусов

цельсия. Не стоит упускать из внимания показатели относительной влажности воздуха, среднее значение которой должно составлять примерно 55%. Помещение, где находятся ПЭВМ, должно быть хорошо вентилируемым.

По причине высокой нагрузки на нервную систему во время рабочего процесса, пользователям ПЭВМ рекомендуется организовать рабочую смену таким образом, чтобы происходило чередование работ с использованием персонального компьютера и без него. Интервал рабочей деятельности между отдыхами не должен превышать более 1 часа, а длительность перерыва должна варьироваться от 10 до 15 минут. При работе с компьютером в ночное время суток длительность перерывов следует увеличить на 35 процентов.

Помимо этого, рекомендуется проводить упражнения, стимулирующие кровообращение в области зрительного аппарата и снижающие общее напряжение глаз. Из-за непрерывного процесса просмотра в монитор, слезистая оболочка глаз теряет значительное количество влаги и пересыхает.

Существует ряд рекомендаций и упражнений, снижающих нагрузку на зрительную систему:

- легкими движениями нажимайте на веки;
- открывайте и закрывайте глаза с периодичностью 3-5 секунд;
- часто моргайте в течение пары минут;

переводите взгляд с объектов, расположенных вдали на близко расположенные предметы, фокусируясь на протяжении нескольких секунд.

Заключение

В ходе выполнения данной работы были приобретены навыки описания вычислительных систем, навыки разработки оконных приложений с помощью фреймворка Windows Form. Результатом выполнения работы стала среда моделирования поведения и симуляции работы вычислительных систем, подразумевающих наличие временных задержек в функционировании.

В разработанной среде моделирования описание систем происходит с помощью временного автомата, который задается через систему канонических уравнений.

Разработанная среда моделирования способна решать такие задачи, как проектирование сложных вычислительных систем, моделирование их поведения, общий анализ вычислительных систем, выявление узких и проблемных мест в их работе еще на стадии проектирования.

Основными достоинствами разработанной среды являются наличие удобного и понятного для пользования интерфейса, ресурсоемкость и производительность.

Приложение «Инструментальная система темпорального анализа моделей параллельных вычислительных систем» было представлено на семинаре по электронным и сетевым технологиям MWENT-2020, проходившего в стенах Пензенского Государственного Университета. Результаты трудов семинара были проиндексированы рефератной базой данных SCOPUS.

Список используемых источников

1. Глушков В. М. Абстрактная теория автоматов «Успехи математических наук». – 1961. – Т. 16. – №. 5 (101. – С. 3-62.
2. Ожиганов А. А. Теория автоматов «Учебное пособие». –СПб: НИУ ИТМО. – 2013.
3. Твардовский А.С. Евтушенко Н.В., Громов М.Л. Минимизация автоматов с таймаутами и временными ограничениями. —2017.
4. Твардовский А. С., Лапутенко А. В. О возможностях автоматного описания параллельной композиции временных автоматов «Труды Института системного программирования РАН». – 2018. – Т. 30. – №. 1.
5. Brown E. Windows forms programming with C. – Manning Publications Co., 2002.
6. Гамма, Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования Паттерны проектирования. — СПб.: «Питер», 2001. — С. 18-20.
7. Лафоре Р. Объектно-ориентированное программирование в C++. – «Издательский дом Питер», 2013.
8. Sharp J. Microsoft Visual C# 2013 Step by Step. – Pearson Education, 2013.
9. Романчук В. А., Ручкин В. Н. Оценка результатов моделирования вычислительных систем на базе нейропроцессоров «Известия тульского государственного университета. Технические науки». – 2013. – №. 9-2.
10. Никлаус В. Алгоритмы и структуры данных. – Litres, 2017.
11. Симакова, Н.Н. Организация рабочих мест с персональными электронновычислительными машинами (ПЭВМ) – Новосибирск: «СибГУТИ», 2010. – 40-54 с.

Приложение А. Листинги программы

Файл «Parser.cs»

```
using System;
using System.Text;
using System.IO;
using sku_to_smv.Properties;
using System.Text.RegularExpressions;
using sku_to_smv.src;
using System.Collections.Generic;

namespace sku_to_smv
{
    public enum parceResult { PARSE_OK, PARCE_ERROR };

    public class Parser
    {
        enum state { H, S1, S2, S3, S4, S5, S6, S7, S8, S9,
S10, S11, S12, S13, ERR, END, EXT, OPT, OPT_END, TIME, O1, O2,
O3, O4, O5 };
        enum parseStates { TARGET_STATE, SOURCE_STATE, TIME,
SIGNAL, OPERATION, INPUT_SIGNALS, SKU }

        int RCount; //Количество правил
полученных после анализа
        bool Inv;
        bool isSKU;
        bool isOUT;

        public List<Rule> Rules; //Массив правил,
строющийся при анализе
        public Rule[] OutputRules; //Массив правил,
строющийся при анализе
        public string[] LocalStates; //Список локальных
состояний
        public string[] Inputs; //Список входных
сигналов
        public string[] RecovStates; //Список
повторновходимых состояний
        public string[] Outputs; //Список выходных
сигналов

        public bool signalsParsed;
        public List<Signal> signals;
        public List<State> states;
        public Parser()
        {
            //Инициализируем массивы
            Rules = new List<Rule>();
            LocalStates = new string[0];
            Inputs = new string[0];
            Outputs = new string[0];
            signals = new List<Signal>();
            states = new List<State>();
            RecovStates = new string[0];
            RCount = 0;
        }
    }
}
```

```

        signalsParsed = false;
    }
    /// <summary>
    /// Добавляет новое правило в массив
    /// </summary>
    /*private void AddRule()
    {
        Array.Resize(ref Rules, Rules.Length + 1);
        Rules[Rules.Length - 1] = new Rule();
        RCount++;
    }
    private void AddOutRule()
    {
        Array.Resize(ref OutputRules, OutputRules.Length
+ 1);

        OutputRules[Rules.Length - 1] = new Rule();
        //RCount++;
    }*/

    private bool isExistsState(String stateName)
    {
        foreach (State state in this.states)
        {
            if (state.Name.Equals(stateName)) return
true;
        }
        return false;
    }

    private State getStateByName(String name)
    {
        foreach (State state in this.states)
        {
            if (state.Name.Equals(name)) return state;
        }
        throw new Exception("State with name " + name + "
not found");
    }

    private Signal getSignalByName(String name)
    {
        foreach (Signal signal in this.signals)
        {
            if (signal.name.Equals(name)) return signal;
        }
        throw new Exception("State with name " + name + "
not found");
    }
    /// <summary>
    /// Запуск парсера
    /// </summary>
    /// <param name="InputString">Текст для
разбора</param>
    /// <returns>Результат разбора типа
parceResult</returns>
    public parceResult ParseStart(String InputString)

```

```

{
    int k = 0;
    string STR;
    string inputSTR = InputString;
    int[] EndOfRules = new int[0];
    bool b_Comment = false;
    isSKU = true;
    isOUT = false;
    StringBuilder sb1 = new StringBuilder();
    //Rules = new Rule[0];
    LocalStates = new string[0];
    Inputs = new string[0];
    Outputs = new string[0];
    RecovStates = new string[0];
    GC.Collect();

    //Убираем пробелы, знаки табуляции, перехода на
    новую строку и комментарии
    for (int i = 0; i < inputSTR.Length; i++)
    {
        if (inputSTR[i] == '#' && !b_Comment)
        b_Comment = true;
        else if (inputSTR[i] == '#' && b_Comment)
        b_Comment = false;
        if (inputSTR[i] != 32 && inputSTR[i] != '\t'
        && inputSTR[i] != '\n' && !b_Comment && inputSTR[i] != '#')
        {
            sb1.Append(inputSTR[i]);
        }
    }
    inputSTR = sb1.ToString();
    //Поиск всех знаков ';'
    for (int i = 0; i < inputSTR.Length; i++)
    {
        if (inputSTR[i] == ';')
        {
            Array.Resize(ref EndOfRules,
            EndOfRules.Length + 1);
            EndOfRules[EndOfRules.Length - 1] = i;
        }
    }
    //Переводим в нижний регистр
    inputSTR = inputSTR.ToLower();

    sb1.Clear();
    k = 0;
    for (int i = 0; i < EndOfRules.Length; i++)
    {
        //Выделение одного правила
        STR = inputSTR.Substring(k, EndOfRules[i] - k
+ 1);

        k = EndOfRules[i] + 1;
        //Разбор правила
        if (parce(STR) == parceResult.PARCE_ERROR)
        {
            return parceResult.PARCE_ERROR;

```

```

    }
}
//Составление списка локальных состояний//
//LocalStates = new string[Rules.Length];
/*for (int i = 0; i < Rules.Length; i++)
{
    if (Rules[i].Elems[0].Output)
    {
        Array.Resize(ref Outputs, Outputs.Length
+ 1);
        Outputs[Outputs.Length - 1] =
Rules[i].Elems[0].Value;
    }
    else
    {
        Array.Resize(ref LocalStates,
LocalStates.Length + 1);
        LocalStates[LocalStates.Length - 1] =
Rules[i].Elems[0].Value;
    }
}
SearchForInputs();//Составление списка входных
сигналов//
SearchForRecov();//Составление списка
повторновходимых состояний//*/

return parceResult.PARSE_OK;
}
/// <summary>
/// Разбор очередной строки
/// </summary>
/// <param name="InputString">Строка для
разбора</param>
/// <param name="StartState">Начальное состояние
парсера</param>
/// <returns>Результат разбора типа
parceResult</returns>
/*private parceResult Parse(String InputString, state
StartState)
{
    state st1 = StartState;
    int k = 0;
    string tmp;
    string time = "";
    for (int i = 0; i < InputString.Length; i++)
    {
        if (!signalsParsed) parce(InputString);
        switch (st1)
        {
            //Буква
            case state.H:
                if (InputString[i] == '[')
                    st1 = state.OPT;
                else if
(Regex.IsMatch(InputString[i].ToString(), "[a-z]"))
                {

```

```

k = i;
//Rules[Rules.Length - 1].output

= isOUT;

state.O1;

    }
    else st1 = state.ERR;
    break;
//Если новая секция
case state.OPT:
    if (InputString[i] == 's')
    {
        isSKU = true;
        isOUT = false;
        //Rules[Rules.Length - 1].output

= false;

        st1 = state.OPT_END;
    }
    else if (InputString[i] == 'o')
    {
        isSKU = false;
        isOUT = true;
        //Rules[Rules.Length - 1].output

= true;

        st1 = state.OPT_END;
    }
    else st1 = state.ERR;
    break;
//Ожидаем конца определения секции
case state.OPT_END:
    if (InputString[i] == ']')
        st1 = state.H;
    break;
case state.O1:
    if
(Regex.IsMatch(InputString[i].ToString(), "[a-z0-9]"))
    {
        st1 = state.O2;
    }
    else if (InputString[i] == '{')
    {
        st1 = state.TIME;
    }
    else st1 = state.ERR;
    break;
case state.O2:
    if (InputString[i] == '<')
    {
        st1 = state.O3;
        tmp = InputString.Substring(k, i
- k);

        if (tmp.Contains("{")) tmp =
tmp.Substring(0, tmp.IndexOf("{"));
        //Rules[Rules.Length -
1].AddData("State", tmp, false, isOUT);
        k = i + 1;

```

```

        }
        else if
(Regex.IsMatch(InputString[i].ToString(), "[a-z0-9]"))
        {
            st1 = state.02;
        }
        else st1 = state.ERR;
        break;
    case state.03:
        if (InputString[i] == '=')
        {
            st1 = state.04;
            //Rules[Rules.Length -
1].AddData("<=", "<=", false, isOUT);
            k = i + 1;
        }
        else st1 = state.ERR;
        break;
    case state.04:
        if
(Regex.IsMatch(InputString[i].ToString(), "[a-z]"))
        {
            k = i;
            st1 = state.05;
        }
        else st1 = state.ERR;
        break;
    case state.05:
        if
(Regex.IsMatch(InputString[i].ToString(), "[a-z0-9]"))
        {
            st1 = state.05;
        }
        else if (InputString[i] == ';')
        {
            tmp = InputString.Substring(k, i
- k);

            if (tmp.Contains("{")) tmp =
tmp.Substring(0, tmp.IndexOf("{"));
            Rules[Rules.Length -
1].AddData("State", tmp, false, isOUT);
            st1 = state.END;
        }
        else st1 = state.ERR;
        break;
    //Буква или цифра или '(' или '='
    case state.S1:
        if
(Regex.IsMatch(InputString[i].ToString(), "[a-z0-9]"))
        {
            st1 = state.S1;
        }
        else if (InputString[i] == '(')
        {
            st1 = state.S2;

```

```

        tmp = InputString.Substring(k, i
- k);
        if (tmp.Contains("{")) tmp =
tmp.Substring(0, tmp.IndexOf("{"));
        Rules[Rules.Length -
1].AddData("State", tmp, false, isOUT);
        k = i + 1;
    }
    else if (InputString[i] == '=')
    {
        st1 = state.S7;
        tmp = InputString.Substring(k, i
- k);
        if (tmp.Contains("{")) tmp =
tmp.Substring(0, tmp.IndexOf("{"));
        Rules[Rules.Length -
1].AddData("State", tmp, false, isOUT);
        Rules[Rules.Length -
1].AddData("=", "=", false);
        k = i + 1;
    }
    else st1 = state.ERR;
    break;
case state.S2:
    if (InputString[i] == 't')
    {
        st1 = state.S3;
    }
    else st1 = state.ERR;
    break;
case state.S3:
    if (InputString[i] == '+')
    {
        st1 = state.S4;
    }
    else st1 = state.ERR;
    break;
case state.S4:
    if (InputString[i] == '1')
    {
        st1 = state.S5;
    }
    else st1 = state.ERR;
    break;
case state.S5:
    if (InputString[i] == ')')
    {
        Rules[Rules.Length -
1].AddData("t+1", "(t+1)", false);
        k = i + 1;
        st1 = state.S6;
    }
    else st1 = state.ERR;
    break;
case state.S6:
    if (InputString[i] == '=')

```

```

        {
            Rules[Rules.Length -
1].AddData("=", "=", false);
            k = i + 1;
            st1 = state.S7;
        }
        else st1 = state.ERR;
        break;
        //Буква или '(' или '~'
        case state.S7:
            if
(Regex.IsMatch(InputString[i].ToString(), "[0-9]"))
            {
                st1 = state.S7;
            }
            else if
(Regex.IsMatch(InputString[i].ToString(), "[a-z]"))
            {
                k = i;
                Inv = false;
                st1 = state.S11;
            }
            else if (InputString[i] == '(')
            {
                Inv = false;
                Rules[Rules.Length -
1].AddData("(", "(", Inv);
                int n = 0;
                for (int j = i; j <
InputString.Length; j++)
                {
                    if (InputString[j] == '(')
                    {
                        n++;
                    }
                    if (InputString[j] == ')')
                    {
                        n--;
                    }
                    if (n == 0)
                    {
                        k = j;
                        break;
                    }
                }
                if (Parse(InputString.Substring(i
+ 1, k - i - 1) + ";", state.S7) == parceResult.PARCE_ERROR)
                {
                    st1 = state.ERR;
                }
                i = k - 1;
                st1 = state.S8;
            }
            else if (InputString[i] == '~')
            {
                st1 = state.S10;
            }
        }
    }
}

```

```

        Inv = true;
    }
    else st1 = state.ERR;
    break;
// ')'
case state.S8:
    if (InputString[i] == ')')
    {
        Rules[Rules.Length -
1].AddData(")", ")", false);
        st1 = state.S9;
    }
    else st1 = state.ERR;
    break;
// '&' или '|' или ';'
case state.S9:
    if (InputString[i] == '&')
    {
        Rules[Rules.Length -
1].AddData("&", "&", false);
        st1 = state.S7;
    }
    else if (InputString[i] == '|')
    {
        Rules[Rules.Length -
1].AddData("|", "|", false);
        st1 = state.S7;
    }
    else if (InputString[i] == ';')
    {
        st1 = state.END;
    }
    break;
// Буква или '(' или
case state.S10:
    if
(Regex.IsMatch(InputString[i].ToString(), "[a-z]"))
    {
        k = i;
        st1 = state.S11;
    }
    else if (InputString[i] == '(')
    {
        Inv = false;
        Rules[Rules.Length -
1].AddData("(", "(", Inv);

        int n = 0;
        for (int j = i; j <
InputString.Length; j++)
        {
            if (InputString[j] == '(')
            {
                n++;
            }
            if (InputString[j] == ')')
            {

```

```

        n--;
    }
    if (n == 0)
    {
        k = j;
        break;
    }
}
if (Parse(InputString.Substring(i
+ 1, k - i - 1) + ";", state.S7) == parceResult.PARCE_ERROR)
{
    st1 = state.ERR;
}
i = k - 1;
st1 = state.S8;
}
break;
case state.S11:
    if
(Regex.IsMatch(InputString[i].ToString(), "[a-z0-9]"))
    {
        st1 = state.S11;
    }
    else if (InputString[i] == '{')
    {
        st1 = state.TIME;
    }
    else if (InputString[i] == '&')
    {
        st1 = state.S7;
        string stateName =
InputString.Substring(k, i - k);
        if (stateName.Contains("{"))
stateName = stateName.Substring(0, stateName.IndexOf("{"));
        Rules[Rules.Length -
1].AddData("State", stateName, Inv);
        Rules[Rules.Length -
1].AddData("&", "&", false);
    }
    else if (InputString[i] == '|')
    {
        st1 = state.S7;
        string stateName =
InputString.Substring(k, i - k);
        if (stateName.Contains("{"))
stateName = stateName.Substring(0, stateName.IndexOf("{"));
        Rules[Rules.Length -
1].AddData("State", stateName, Inv);
        Rules[Rules.Length -
1].AddData("|", "|", false);
    }
    else if (InputString[i] == ';')
    {
        st1 = state.END;
        string stateName =
InputString.Substring(k, i - k);

```

```

        if (stateName.Contains("{"))
stateName = stateName.Substring(0, stateName.IndexOf("{"));
        Rules[Rules.Length -
1].AddData("State", stateName, Inv);
    }
    else st1 = state.ERR;
    break;
    case state.TIME:
        if (InputString[i] == '}')
        {
            Rules[Rules.Length -
1].AddData("TimeTransfer", time, Inv);
            st1 = state.S11;
            time = "";
            break;
        }
        time += InputString[i].ToString();
        st1 = state.TIME;
        break;
    case state.ERR: //PrintText("Ошибка
разбора");

        //st1 = state.EXT;
        break;
    case state.END: //j++;
        //AddRule();
        //st1 = state.H;
        break;
    case state.EXT:
        break;
    }
}
if (st1 == state.ERR)
{
    return parceResult.PARCE_ERROR;
}
return parceResult.PARSE_OK;
}*/

private parceResult parce(String inputStr)
{
    if (inputStr.Contains("input="))
    {
        int firstBracket = inputStr.IndexOf("[") + 1;
        int lastBracket = inputStr.LastIndexOf("]");
        String signalStr =
inputStr.Substring(firstBracket, lastBracket - firstBracket);
        String[] parsedSignals =
signalStr.Split(',');
        foreach (String s in parsedSignals)
        {
            Signal signal = new Signal();
            signal.name = s;
            this.signals.Add(signal);
        }
        return parceResult.PARSE_OK;
    }
}

```

```

        String pattern = "(\\w+)=\\((?![\\w\\|]+\\))?)
([&!?\\w+<>]+)";
        String statePattern = "\\((?\\w+\\|?\\))?"
        String signalPattern = "&(!?\\w+)"
        String timePattern = "<(\\w+)>"
        String str = inputStr;
        Regex reg = new Regex(pattern,
RegexOptions.Compiled);
        Regex stateReg = new Regex(statePattern,
RegexOptions.Compiled);
        Regex signalReg = new Regex(signalPattern,
RegexOptions.Compiled);
        Regex timeReg = new Regex(timePattern,
RegexOptions.Compiled);
        Match match = reg.Match(str);
        Time timeMark = null;

        State endState = null;
        List<State> beginStates = new List<State>();
        List<Signal> signals = new List<Signal>();
        List<Rule> rules = new List<Rule>();
        List<Signal> inventeredSignals = new
List<Signal>();
        for (int i = 1; i < match.Groups.Count; i++)
        {
            switch (i)
            {
                case 1:
                {
                    String stateName =
match.Groups[i].Value;
                    if (isExistsState(stateName))
                    {
                        endState =
getStateByName(stateName);
                    }
                    else
                    {
                        endState = new State();
                        endState.Name = stateName;
                        states.Add(endState);
                    }
                    break;
                }
                case 2:
                {
                    MatchCollection matches =
stateReg.Matches(match.Groups[i].Value);
                    foreach (Match m in matches)
                    {
                        for (int j = 1; j <
m.Groups.Count; j++)
                        {
                            String stateName =
m.Groups[j].Value;
                            State beginState;

```

```

        if
        (isExistsState(stateName))
        {
            beginState =
getStateByName(stateName);
        }
        else
        {
            beginState = new
State();
            beginState.Name =
stateName;
            states.Add(beginState);
        }

beginStates.Add(beginState);
    }
    }
    break;
}
case 3:
{
    Match timeMatch =
timeReg.Match(match.Groups[i].Value);
    for (int j = 1; j <
timeMatch.Groups.Count; j++)
    {
        timeMark = new
Time(timeMatch.Groups[j].Value);
    }
    MatchCollection matches =
signalReg.Matches(match.Groups[i].Value);
    foreach (Match m in matches)
    {
        for (int j = 1; j <
m.Groups.Count; j++)
        {
            String signalName =
m.Groups[j].Value;
            try
            {
                if
                (signalName.Contains("!"))
                {
                    signalName =
signalName.Substring(1, signalName.Length - 1);
                    inventeredSignals.Add(getSignalByName(signalName));
                }
            }
            catch (Exception)
            {

```

```

return
parceResult.PARCE_ERROR;
    }
    }
    }
    break;
}
}
}
if (endState == null)
{
    return parceResult.PARCE_ERROR;
}
if (signals.Count != 0)
{
    foreach (Signal signal in signals)
    {
        foreach (State state in beginStates)
        {
            Rule rule = new Rule(state, endState,
signal, timeMark, inventeredSignals.Contains(signal));
            rules.Add(rule);
        }
    }
}
else
{
    foreach (State state in beginStates)
    {
        Rule rule = new Rule(state, endState,
null, timeMark, false);
        rules.Add(rule);
    }
}

this.Rules.AddRange(rules);
return parceResult.PARSE_OK;
}

/// <summary>
/// Функция поиска входных сигналов
/// </summary>
/*private void SearchForInputs()
{
    bool NotInput = false;
    bool Exists = false;
    for (int i = 0; i < Rules.Length; i++)
    {
        for (int j = 0; j < Rules[i].Elems.Length; j+
+)
        {
            if (Rules[i].Elems[j].Type == "State")
            {
                for (int k = 0; k <
LocalStates.Length; k++)
                {

```

```

LocalStates[k])
        if (Rules[i].Elems[j].Value ==
        {
            NotInput = true;
            break;
        }
    }
    for (int k = 0; k < Inputs.Length; k+
+)
    {
        if (Rules[i].Elems[j].Value ==
Inputs[k])
        {
            Exists = true;
            Rules[i].Elems[j].Local =
false;
            break;
        }
    }
    if (!NotInput && !Exists && !
Rules[i].Elems[j].Output)
    {
        Array.Resize(ref Inputs,
Inputs.Length + 1);
        Inputs[Inputs.Length - 1] =
Rules[i].Elems[j].Value;
        Rules[i].Elems[j].Local = false;
    }
    NotInput = false;
    Exists = false;
}
}
}
}*/
/// <summary>
/// Функция поиска повторновходимых состояний
/// </summary>
/*private void SearchForRecov()
{
    for (int i = 0; i < Rules.Length; i++)
    {
        for (int j = 1; j < Rules[i].Elems.Length; j+
+)
        {
            if (Rules[i].Elems[0].Value ==
Rules[i].Elems[j].Value)
            {
                Array.Resize(ref RecovStates,
RecovStates.Length + 1);
                RecovStates[RecovStates.Length - 1] =
Rules[i].Elems[j].Value;
                break;
            }
        }
    }
}*/

```

```

/// <summary>
/// Функция сохранения результата в SMV
/// </summary>
/// <param name="Path">Путь для сохранения</param>
public void SaveToSMV(string Path)
{
    StreamWriter sw = new StreamWriter(Path, false);
    sw.Write("module main(");
    sw.Flush();
    for (int i = 0; i < Inputs.Length; i++)
    {
        sw.Write(Inputs[i]);
        if ((i != Inputs.Length - 1) ||
Outputs.Length > 0)
        {
            sw.Write(",");
        }
        sw.Flush();
    }
    for (int i = 0; i < Outputs.Length; i++)
    {
        sw.Write(Outputs[i]);
        if (i != Outputs.Length - 1)
        {
            sw.Write(",");
        }
        sw.Flush();
    }
    sw.Write(")\r\n{\r\n");
    sw.Write("\tinput ");
    for (int i = 0; i < Inputs.Length; i++)
    {
        sw.Write(Inputs[i]);
        if (i != Inputs.Length - 1)
        {
            sw.Write(",");
        }
        sw.Flush();
    }
    sw.Write(": boolean;\r\n");
    if (Outputs.Length > 0)
    {
        sw.Write("\toutput ");
        for (int i = 0; i < Outputs.Length; i++)
        {
            sw.Write(Outputs[i]);
            if (i != Outputs.Length - 1)
            {
                sw.Write(",");
            }
            sw.Flush();
        }
        sw.Write(": boolean;\r\n");
    }
    //sw.Write("\toutput ");
    sw.Write("\tVAR ");

```

```

for (int i = 0; i < LocalStates.Length; i++)
{
    sw.Write(LocalStates[i]);
    if (i != LocalStates.Length - 1)
    {
        sw.Write(",");
    }
    sw.Flush();
}
sw.Write(": boolean;\r\n");
sw.Write("ASSIGN\r\n");
for (int i = 0; i < Inputs.Length; i++)
{
    sw.Write("\tinit(" + Inputs[i] + ")
:=0;\r\n");
    sw.Flush();
}
for (int i = 0; i < LocalStates.Length; i++)
{
    sw.Write("\tinit(" + LocalStates[i] + ")
:=0;\r\n");
    sw.Flush();
}
sw.Write("default\r\n{");
for (int i = 0; i < LocalStates.Length; i++)
{
    sw.Write("\tnext(" + LocalStates[i] + ")
:=0;\r\n");
    sw.Flush();
}
for (int i = 0; i < Outputs.Length; i++)
{
    sw.Write("\tnext(" + Outputs[i] + ")
:=0;\r\n");
    sw.Flush();
}
sw.Write("}\r\n");
sw.Write("in\r\n{\r\n");
for (int i = 0; i < Rules.Count; i++)
{
    //Rules[i].PrintRule(sw);
    sw.Write("\r\n");
    sw.Flush();
}
sw.Write("}\r\n\r\n");
sw.Write("--Reachability of states\r\n");
for (int i = 0; i < LocalStates.Length; i++)
{
    sw.Write("SPEC EF " + LocalStates[i] +
";\r\n");
    sw.Flush();
}
for (int i = 0; i < Outputs.Length; i++)
{
    sw.Write("SPEC EF " + Outputs[i] + ";\r\n");
    sw.Flush();
}

```

```

    }
    sw.Write("--Recoverability of states\r\n");
    for (int i = 0; i < RecovStates.Length; i++)
    {
        sw.Write("SPEC AG ( " + RecovStates[i] + " ->
AF " + RecovStates[i] + " );\r\n");
        sw.Flush();
    }
    sw.Write("{}");
    sw.Flush();
    sw.Close();
}
/*public void SaveToVHDL(string Path, bool CreateBus,
int OuputSigCount, OutputTableElement[] OutTable)
{
    String resultCode, tmp;
    int index;
    StringBuilder sb = new StringBuilder();
    StreamWriter sw = new StreamWriter(Path, false);
    FileInfo fi = new FileInfo(Path);
    resultCode =
global::sku_to_smv.Properties.Resources.vhd_tmpl;
    ////////////////////////////////////////////
    ////////////////////////////////////////////
    index = resultCode.IndexOf("$inputDescription");
    resultCode = resultCode.Remove(index,
"$inputDescription".Length);
    if (CreateBus)
    {
        tmp = "-- Входные сигналы\n";
        resultCode = resultCode.Insert(index, tmp);
        index += tmp.Length;
        for (int i = 0; i < Inputs.Length; i++)
        {
            tmp = "-- " + Inputs[i] + "\t->\tinputs["
+ i.ToString() + "]\n";
            resultCode = resultCode.Insert(index,
tmp);
            index += tmp.Length;
        }
        int j = 0;
        tmp = "-- Выходные сигналы\n";
        resultCode = resultCode.Insert(index, tmp);
        index += tmp.Length;
        for (int i = 0; i < OutTable.Length; i++)
        {
            if (OutTable[i].HasOutput)
            {
                tmp = "-- " + OutTable[i].OutputName
+ "\t->\toutputs[" + j.ToString() + "]\n";
                resultCode = resultCode.Insert(index,
tmp);
                index += tmp.Length;
                ++j;
            }
        }
    }
}

```

```

    }
    //////////////////////////////////////
    //////////////////////////////////////
    while ((index = resultCode.IndexOf("$name")) !=
-1)
    {
        resultCode = resultCode.Remove(index,
"$name".Length);
        resultCode = resultCode.Insert(index,
fi.Name.Replace(fi.Extension, ""));
    }
    //////////////////////////////////////
    //////////////////////////////////////
    index = resultCode.IndexOf("$ports");
    resultCode = resultCode.Remove(index,
"$ports".Length);
    if (CreateBus)
    {
        if (OuputSigCount > 0) tmp = "inputs
: in STD_LOGIC_VECTOR(" + (Inputs.Length -
1).ToString() + " downto 0);\n";
        else tmp = "inputs : in
STD_LOGIC_VECTOR(" + (Inputs.Length - 1).ToString() + "
downto 0)";
        resultCode = resultCode.Insert(index, tmp);
        index += tmp.Length;
    }
    else
    {
        for (int i = 0; i < Inputs.Length - 1; i++)
        {
            tmp = "\t\t\t" + Inputs[i] + "
: in STD_LOGIC;\n";
            resultCode = resultCode.Insert(index,
tmp);
            index += tmp.Length;
        }
        if (OuputSigCount > 0) tmp = "\t\t\t" +
Inputs[Inputs.Length - 1] + " : in
STD_LOGIC;\n";
        else tmp = "\t\t\t" + Inputs[Inputs.Length -
1] + " : in STD_LOGIC\n";
        resultCode = resultCode.Insert(index, tmp);
        index += tmp.Length;
    }

    if (OuputSigCount > 0)
    {
        if (CreateBus)
        {
            tmp = "\t\t\toutputs :
out STD_LOGIC_VECTOR(" + (OuputSigCount -
1).ToString() + " downto 0)";
            resultCode = resultCode.Insert(index,
tmp);
        }
    }
}

```

```

else
{
    for (int i = 0; i < OutTable.Length - 1;
i++)
    {
        if (OutTable[i].HasOutput)
        {
            tmp = "\t\t\t" +
OutTable[i].OutputName + "                :      out
            STD_LOGIC;\n";
            resultCode =
resultCode.Insert(index, tmp);
            index += tmp.Length;
        }
        if (OutTable[OutTable.Length -
1].HasOutput)
        {
            tmp = "\t\t\t" +
OutTable[OutTable.Length - 1].OutputName + "                :
            out      STD_LOGIC\n";
            resultCode = resultCode.Insert(index,
tmp);
        }
        else resultCode = resultCode.Remove(index
- 2, 1);
    }
}
////////////////////////////////////
////////////////////////////////////
while ((index = resultCode.IndexOf("$width")) !=
-1)
{
    resultCode = resultCode.Remove(index,
"$width".Length);
    resultCode = resultCode.Insert(index,
(Rules.Length - 1).ToString());
}
////////////////////////////////////
////////////////////////////////////
for (int i = 0; i < Rules.Length; i++)
{
    sb.Append("0");
}
while ((index = resultCode.IndexOf("$zero")) !=
-1)
{
    resultCode = resultCode.Remove(index,
"$zero".Length);
    resultCode = resultCode.Insert(index,
sb.ToString());
}
sb.Clear();
////////////////////////////////////
////////////////////////////////////
index = resultCode.IndexOf("$localDescription");

```

```

        resultCode = resultCode.Remove(index,
"$localDescription".Length);

        tmp = "-- Состояния автомата\n";
        resultCode = resultCode.Insert(index, tmp);
        index += tmp.Length;
        for (int i = 0; i < Rules.Length; i++)
        {
            if (Rules[i].output)
            {
                tmp = "-- not used\t->\tcurState[" +
i.ToString() + "]\n";
            }
            else tmp = "-- " + Rules[i].Elems[0].Value +
"\t->\tcurState[" + i.ToString() + "]\n";
            resultCode = resultCode.Insert(index, tmp);
            index += tmp.Length;
        }
        ////////////////////////////////////////////
        ////////////////////////////////////////////
        index = resultCode.IndexOf("$rules");
        resultCode = resultCode.Remove(index,
"$rules".Length);
        for (int i = 0; i < Rules.Length; i++)
        {
            if (!Rules[i].output)
            {
                //sb.Append("\t\tif (");
                sb.Append("\n\t\tnewState(" +
i.ToString() + ") := ");
                for (int j = 1; j <
Rules[i].Elems.Length; j++)
                {
                    if ((Rules[i].Elems[j].Type != "=")
&& (Rules[i].Elems[j].Type != "t+1") && (!
Rules[i].Elems[j].Empty))
                    {
                        if (Rules[i].Elems[j].Type ==
"State")
                        {
                            if
(Rules[i].Elems[j].Inverted)
                            {
                                if
(Rules[i].Elems[j].Local)
                                {
                                    for (int n = 0; n <
Rules.Length; n++)
                                    {
                                        if
(Rules[i].Elems[j].Value == Rules[n].Elems[0].Value)
                                        {
                                            //sb.Append("curState(" + n.ToString() + ") = '0'");
                                            sb.Append("(not curState(" + n.ToString() + "))");

```

```

                                break;
                            }
                        }
                    }
                else
                {
                    for (int n = 0; n <
Inputs.Length; n++)
                    {
                        if (Inputs[n] ==
Rules[i].Elems[j].Value)
                        {
                            //
                            //
                            sb.Append("inputs(" + n.ToString() + ")  = '0'");
                            //
                            else sb.Append(Inputs[n] + "  = '0'");
                            if
                                (CreateBus)
                                sb.Append("(not inputs(" + n.ToString() + "))");
                                else
                                sb.Append("(not " + Inputs[n] + ")");
                                break;
                            }
                        }
                    }
                }
            else
            {
                if
                    (Rules[i].Elems[j].Local)
                    {
                        for (int n = 0; n <
Rules.Length; n++)
                        {
                            if
                                (Rules[i].Elems[j].Value == Rules[n].Elems[0].Value)
                                {
                                    //sb.Append("curState(" + n.ToString() + ")  = '1'");
                                    sb.Append("curState(" + n.ToString() + ")");
                                    break;
                                }
                            }
                        }
                    }
                else
                {
                    for (int n = 0; n <
Inputs.Length; n++)
                    {
                        if (Inputs[n] ==
Rules[i].Elems[j].Value)
                        {

```

```

//
if (CreateBus)
//
sb.Append("inputs(" + n.ToString() + ") = '1'");
//
else sb.Append(Inputs[n] + " = '1'");
if
(CreateBus)
sb.Append("inputs(" + n.ToString() + ")");
else
sb.Append(Inputs[n]);
break;
}
}
}
}
else
{
if (Rules[i].Elems[j].Type ==
if (Rules[i].Elems[j].Type ==
if (Rules[i].Elems[j].Type ==
{
if
{
sb.Append(" not ");
}
sb.Append("(");
}
if (Rules[i].Elems[j].Type ==
{
sb.Append(")");
}
}
}
}
int co = 0;
int k = 0;
for (k = 0; k < OutTable.Length; k++)
{
if (Rules[i].Elems[0].Value ==
OutTable[k].StateName && OutTable[k].HasOutput)
{
tmp = OutTable[k].OutputName;
break;
}
else if (OutTable[k].HasOutput)
co++;
}
if (CreateBus)

```



```

{
    enum state { START, COMMENT, SIGNAL, OPT };

    public SympleElement[] elements;
    public SympleElement brackets;
    public bool b_Brackets;
    state CurrentState;
    int currentElement;
    String str;

    public SympleParser()
    {
        b_Brackets = false;
        elements = new SympleElement[0];
        CurrentState = state.START;
        brackets = new SympleElement();
        brackets.StartIndex = -1;
        brackets.EndIndex = -1;
    }
    static public int FindBracket(String Input, int
StartIndex, bool b_UpDown)
    {
        int counter = 0;
        if (b_UpDown)
        {
            for (int i = StartIndex; i <
Input.Length; i++)
            {
                if (Input[i] == '(')
                {
                    counter++;
                }
                if (Input[i] == ')')
                {
                    counter--;
                }
                if (counter == 0)
                {
                    return i;
                }
            }
        }
        else
        {
            for (int i = StartIndex; i > 0; i--)
            {
                if (Input[i] == ')')
                {
                    counter++;
                }
                if (Input[i] == '(')
                {
                    counter--;
                }
                if (counter == 0)
                {

```

```

        return i;
    }
}
}
return StartIndex;
}
public parceResult Start(String Input)
{
    elements = new SympleElement[0];
    CurrentState = state.START;
    currentElement = -1;
    str = Input.ToLower();

    for (int i = 0; i < str.Length; i++)
    {
        switch (CurrentState)
        {
            case state.START:
                if (str[i] == '#')
                {
                    Array.Resize(ref elements,
elements.Length + 1);

                    currentElement++;
                    elements[currentElement] =
new SympleElement();

                    elements[currentElement].StartIndex = i;
                    CurrentState = state.COMMENT;
                }
                else if
(Regex.IsMatch(str[i].ToString(), "[a-z]"))
                {
                    Array.Resize(ref elements,
elements.Length + 1);

                    currentElement++;
                    elements[currentElement] =
new SympleElement();

                    elements[currentElement].StartIndex = i;
                    CurrentState = state.SIGNAL;
                }
                else if (str[i] == '(')
                {
                    Array.Resize(ref elements,
elements.Length + 1);

                    currentElement++;
                    elements[currentElement] =
new SympleElement();

                    elements[currentElement].StartIndex = i;

                    elements[currentElement].EndIndex = i;

                    elements[currentElement].Style = System.Drawing.FontStyle.Bold;
                    CurrentState = state.START;
                }

```

```

else if (str[i] == ')')
{
    Array.Resize(ref elements,
elements.Length + 1);
    currentElement++;
    elements[currentElement] =
new SympleElement();
    elements[currentElement].StartIndex = i;
    elements[currentElement].EndIndex = i;
    elements[currentElement].Style = System.Drawing.FontStyle.Bold;
    CurrentState = state.START;
}
else if (str[i] == '[')
{
    Array.Resize(ref elements,
elements.Length + 1);
    currentElement++;
    elements[currentElement] =
new SympleElement();
    elements[currentElement].StartIndex = i;
    CurrentState = state.OPT;
}
break;
case state.COMMENT:
    if (str[i] == '#')
    {
        elements[currentElement].EndIndex = i;
        elements[currentElement].TextColor =
Settings.Default.TextFieldCommentColor;
        elements[currentElement].Style = 0;
        CurrentState = state.START;
    }
    break;
case state.OPT:
    if (str[i] == ']')
    {
        elements[currentElement].EndIndex = i;
        elements[currentElement].TextColor =
Settings.Default.TextFieldOptionColor;
        elements[currentElement].Style = 0;
        CurrentState = state.START;
    }
    break;
case state.SIGNAL:

```

```

        if (str[i] != ';' && str[i] !=
'(' && str[i] != ')') && str[i] != '&' && str[i] != '|' && str[i]
!= '~' && str[i] != '#' && str[i] != '=')
        {
            break;
        }
        else
        {

elements[currentElement].EndIndex = i - 1;
            if
(str.Substring(elements[currentElement].StartIndex,
elements[currentElement].EndIndex -
elements[currentElement].StartIndex + 1) == "t+1")
            {

elements[currentElement].TextColor = System.Drawing.Color.Red;

elements[currentElement].Style = 0;
                }
            else
            {

elements[currentElement].TextColor =
Settings.Default.TextFieldSignalColor;

elements[currentElement].Style = 0;
                }
            if (str[i] == '#')
            {
                Array.Resize(ref
elements, elements.Length + 1);
                currentElement++;
                elements[currentElement]
= new SympleElement();

elements[currentElement].StartIndex = i;
                CurrentState =
state.COMMENT;
            }

            if (str[i] == '(')
            {
                Array.Resize(ref
elements, elements.Length + 1);
                currentElement++;
                elements[currentElement]
= new SympleElement();

elements[currentElement].StartIndex = i;

elements[currentElement].EndIndex = i;

elements[currentElement].Style = System.Drawing.FontStyle.Bold;
                CurrentState =
state.START;

```

```

        }
        if (str[i] == ' ')
        {
            Array.Resize(ref
elements, elements.Length + 1);

            currentElement++;
            elements[currentElement]
= new SympleElement();

            elements[currentElement].StartIndex = i;

            elements[currentElement].EndIndex = i;

            elements[currentElement].Style = System.Drawing.FontStyle.Bold;
            CurrentState =
state.START;

        }

        CurrentState = state.START;
    }
    break;
}
}
return parceResult.PARSE_OK;
}
}
}
}
}

```

Файл «Rule.cs»

```

using sku_to_smv.src;
using System;
using System.IO;

namespace sku_to_smv
{
    // Summary:
    //      Описывает элемент списка правил автомата
    //      создающегося при разборе его описания.
    public class Rule
    {
        public State startState;
        public State endState;
        public Signal signal;
        public Time timeMark;
        public bool SignalInventered;

        public Rule()
        {
            startState = new State();
            endState = new State();
            signal = new Signal();
            timeMark = null;
        }
    }
}

```

```

        SignalInventered = false;
    }

    public Rule(State startState, State endState, Signal
signal, Time timeMark, bool InventeredSignal)
    {
        this.startState = startState;
        this.endState = endState;
        this.signal = signal;
        this.timeMark = timeMark;
        this.SignalInventered = InventeredSignal;
    }

    public DrawArea DrawArea
    {
        get => default;
        set
        {
        }
    }
}

```

Файл «State.cs»

```

using sku_to_smv.src;
using System.Collections.Generic;
using System.Drawing;

namespace sku_to_smv
{
    public enum STATE_TYPE { NONE, INPUT, OUTPUT};
    public class State
    {
        public string Name;           //Имя состояния
        public Point paintDot;
        public Point nameDot;
        public bool Selected;         //если выбрано = true
        public bool Active;
        public List<Signal> inputs;
        public List<Link> links;
        public State()
        {
            Name = null;
            paintDot = Point.Empty;
            nameDot = Point.Empty;
            Selected = false;
            inputs = new List<Signal>();
            links = new List<Link>();
        }

        public Rule Rule
        {
            get => default;
            set
            {
            }
        }
    }
}

```

```

    }

    public Link Link
    {
        get => default;
        set
        {
        }
    }

    public DrawArea DrawArea
    {
        get => default;
        set
        {
        }
    }

    public override bool Equals(object obj)
    {
        return obj is State state &&
            Name.Equals(state.Name);
    }

    public void setNameDot()
    {
        int radius =
Properties.Settings.Default.StateDiametr / 2;
        Font style =
Properties.Settings.Default.DefaultText;
        nameDot = new Point(
            paintDot.X + radius - (int)style.Size,
            paintDot.Y + radius - (int)style.Size);
    }

    public void calculateLocation()
    {
        setNameDot();
    }
}
}

```

Файл «Signal.cs»

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;

namespace sku_to_smv.src
{
    public class Signal
    {
        public String name;
        public Point[] paintDots;
        public List<Point> inventeredPoint;
    }
}

```

```

public bool Selected;
public bool Active;
public Link[] links;

public Signal()
{
    name = null;
    paintDots = new Point[0];
    inventeredPoint = new List<Point>();
    Selected = false;
    Active = false;
    links = new Link[0];
}

public Signal(string name)
{
    this.name = name;
}

public Rule Rule
{
    get => default;
    set
    {
    }
}

public State State
{
    get => default;
    set
    {
    }
}

public DrawArea DrawArea
{
    get => default;
    set
    {
    }
}

private void calculatePaintDots()
{
    paintDots = new Point[links.Length];
    for (int i = 0; i < links.Length; i++)
    {
        Link l = links[i];
        for (int j = 0; j < links[i].signals.Length;
j++)
        {
            Signal s = l.signals[j];
            if (s.name.Equals(name))
            {
                Point p = (l.Arc)

```



```

public Signal[] signals;
public Signal[] inventeredSignals;
public float cosx;
public float sinx;
public Link()
{
    startDot = Point.Empty;
    endDot = Point.Empty;
    arcDot = Point.Empty;
    arrowDots = new Point[2];
    startState = null;
    endState = null;
    lengthLink = 0;
    lengthBetweenStatesCentre = 0;
    name = null;
    Arc = false;
    Selected = false;
    timeMark = null;
    signals = new Signal[0];
    inventeredSignals = new Signal[0];
    cosx = 1;
    sinx = 0;
}

this()
public Link(State startState, State endState) :
{
    this.startState = startState;
    this.endState = endState;
}

public Time Time
{
    get => default;
    set
    {
    }
}

public State State
{
    get => default;
    set
    {
    }
}

public DrawArea DrawArea
{
    get => default;
    set
    {
    }
}

public void setName()

```

```

        {
            this.name = startState.Name + endState.Name;
        }

        private void setCosX()
        {
            if (!Arc) cosx = startState.paintDot.Y ==
endState.paintDot.Y
                ? (endState.paintDot.X -
startState.paintDot.X) / Math.Abs(endState.paintDot.X -
startState.paintDot.X)
                : (float)(endState.paintDot.X -
startState.paintDot.X) / lengthBetweenStatesCentre;
        }

        private void setSinX()
        {
            if (!Arc) sinx = startState.paintDot.X ==
endState.paintDot.X
                ? (endState.paintDot.Y -
startState.paintDot.Y) / Math.Abs(endState.paintDot.Y -
startState.paintDot.Y)
                : (float)(endState.paintDot.Y -
startState.paintDot.Y) / lengthBetweenStatesCentre;
        }

        private void calculateLength()
        {
            int radius =
Properties.Settings.Default.StateDiametr / 2;
            lengthBetweenStatesCentre = (int)Math.Sqrt(
                Math.Pow(endState.paintDot.X -
startState.paintDot.X, 2) +
                Math.Pow(endState.paintDot.Y -
startState.paintDot.Y, 2));
            lengthLink = -radius * 2 +
lengthBetweenStatesCentre;
        }

        private void caluclateArrowDots()
        {
            if (Arc) return;
            int arrowLength = 20; //Длина наконечника стрелки
            float cos30 = (float)Math.Sqrt(3) / 2; //Косинус
30 т.к. угол между линией и наконечником = 30
            float sin30 = 0.5F; //Синус 30 т.к. угол между
линией и наконечником = 30
            float cos1 = cosx * cos30 - sinx * sin30;
//cos(a+b) = cos(a)*cos(b) - sin(a)*sin(b);
            float sin1 = sinx * cos30 + cosx * sin30;
//sin(a+b) = sin(a)*cos(b) + cos(a)*sin(b);
            float cos2 = cosx * cos30 + sinx * sin30;
//cos(a-b) = cos(a)*cos(b) + sin(a)*sin(b);
            float sin2 = sinx * cos30 - cosx * sin30;
//sin(a-b) = sin(a)*cos(b) - cos(a)*sin(b);

```

```

        arrowDots[0] = new Point((int)(endDot.X - cos1 *
arrowLength), (int)(endDot.Y - sin1 * arrowLength));
        arrowDots[1] = new Point((int)(endDot.X - cos2 *
arrowLength), (int)(endDot.Y - sin2 * arrowLength));
    }

    private void calculateStartAndEndDots()
    {
        if (Arc)
        {
            int arcOffset = 18;
            arcDot = new Point(
                startState.paintDot.X - arcOffset,
                startState.paintDot.Y - arcOffset);
        }
        else
        {
            int radius =
Properties.Settings.Default.StateDiameter / 2;
            float x = radius * cosx;
            float y = radius * sinx;
            startDot.X = startState.paintDot.X + radius +
(int)x;
            startDot.Y = startState.paintDot.Y + radius +
(int)y;
            endDot.X = endState.paintDot.X + radius -
(int)x;
            endDot.Y = endState.paintDot.Y + radius -
(int)y;
        }
    }

    public void calculateLocation()
    {
        calculateLength();
        setCosX();
        setSinX();
        calculateStartAndEndDots();
        caluclateArrowDots();
    }

    private bool isInvert(Signal s)
    {
        foreach (Signal invS in inventeredSignals)
        {
            if (invS.Equals(s)) return true;
        }
        return false;
    }

    public bool linkActive()
    {
        bool isActive = signals.Length > 0;
        foreach (Signal s in signals)
        {
            bool signalActive = s.Active;

```

```

        if (isInvert(s)) signalActive = !
signalActive;
        isActive = isActive && signalActive;
    }
    return isActive;
}

public override bool Equals(object obj)
{
    return obj is Link link &&
        name.Equals(link.name);
}
}
}

```

Файл «Time.cs»

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace sku_to_smv.src
{
    public class Time : Rule
    {
        public String name;
        public int value;
        public Link[] links;
        public Point[] paintDots;
        public bool selected;

        public Time(String name)
        {
            this.name = name;
            value = 0;
            paintDots = new Point[0];
            links = new Link[0];
            selected = false;
        }

        public TimeTextBox TimeTextBox
        {
            get => default;
            set
            {
            }
        }

        public DrawArea DrawArea
        {
            get => default;
            set
            {
            }
        }
    }
}

```

```

    }

    private void calculatePainDots()
    {
        paintDots = new Point[links.Length];
        for (int i = 0; i < links.Length; i++)
        {
            Link l = links[i];
            paintDots[i] = new Point(
                (int)(l.startDot.X + (l.endDot.X -
l.startDot.X) / 2 + l.signals.Length * 20),
                (int)(l.startDot.Y + (l.endDot.Y -
l.startDot.Y) / 2));
        }
    }

    public void calculateLocation()
    {
        calculatePainDots();
    }
}
}

```

Файл «DrawArea.cs»

```

using System;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.IO.Pipes;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using Microsoft.CSharp;
using sku_to_smv.Properties;
using sku_to_smv.src;

namespace sku_to_smv
{
    public class DrawArea : PictureBox
    {
        //States
        public static readonly Pen stateDefaultPen = new
Pen(Settings.Default.ColorDefaultState, 2);
        public static readonly Pen stateSelectedPen = new
Pen(Settings.Default.ColorSelectedState, 3);
        public static readonly Pen stateActivePen = new
Pen(Settings.Default.ColorActiveState, 4);
        public static readonly Pen stateActiveAndSelectedPen
= new Pen(Settings.Default.ColorActiveState, 6);
        public static readonly Brush stateActiveBrush = new
SolidBrush(Settings.Default.ColorFillActiveState);
        public static readonly Font defaultTextFont =
Settings.Default.DefaultText;
    }
}

```

```

        public static readonly Brush defaultTextBrush = new
SolidBrush(Settings.Default.ColorDefaultTextBrush);

        public static readonly int StateCentre =
Settings.Default.StateCentre;
        public static readonly int OffsetX =
Settings.Default.OffsetStateX;
        public static readonly int OffsetY =
Settings.Default.OffsetStateY;
        public static readonly int ArcCyrcleradius =
Settings.Default.ArcCyrcleradius;
        public static readonly int ArcStartAngle =
Settings.Default.ArcStartAngle;
        public static readonly int ArcSweepAngle =
Settings.Default.ArcSweepAngle;
        public static readonly int StateDiametr =
Settings.Default.StateDiametr;
        //Links
        public static readonly Pen linkDefaultPen = new
Pen(Settings.Default.ColorDefaultLink, 2);
        public static readonly Pen linkSelectedPen = new
Pen(Settings.Default.ColorSelectedLink, 4);
        //Signals
        public static readonly Font signalDefaultFont =
Settings.Default.SignalDefaultFont;
        public static readonly Font signalSelectedFont =
Settings.Default.SignalSelectedFont;
        public static readonly Brush signalDefaultBrush = new
SolidBrush(Settings.Default.SignalDefaultColor);
        public static readonly Brush signalActiveBrush = new
SolidBrush(Settings.Default.SignalActiveColor);
        //TimeMarks
        public TimeTextBox timeTb;
        private static DrawArea _instnce;

        public static DrawArea getInstance()
        {
            if (_instnce == null)
            {
                _instnce = new DrawArea();
            }
            return _instnce;
        }

        public State[] States;
        public Link[] Links;
        public Signal[] Signals;
        public Time[] TimeMarks;
        public Rule[] Rules;
        private float DragOffsetX, DragOffsetY;
        private Point PaintDotSelectedState;
        public bool SimulStarted;
        ToolPanel Tools;

        private DrawArea()

```

```

    {
        InitializeArea();
    }
    ~DrawArea()
    {
        Array.Clear(States, 0, States.Length);
        Array.Clear(Links, 0, Links.Length);
        Array.Clear(Signals, 0, Links.Length);
    }
    private void InitializeArea()
    {
        timeTb = TimeTextBox.GetInstance(this);
        Controls.Add(timeTb);
        DragOffsetX = 0;
        DragOffsetY = 0;
        PaintDotSelectedState = Point.Empty;
        SimulStarted = false;
        ((System.ComponentModel.ISupportInitialize)(this)).BeginInit();
        this.SuspendLayout();
        Links = new Link[0];
        States = new State[0];
        Signals = new Signal[0];
        TimeMarks = new Time[0];
        Rules = new Rule[0];
        this.DoubleBuffered = true;
        Tools = ToolPanel.GetInstance();
        Controls.Add(Tools);
        this.MouseClick += new
MouseEventHandler(this.mouseClick);
        this.MouseDown += new
MouseEventHandler(this.mouseDown);
        this.MouseMove += new
MouseEventHandler(this.mouseMove);
        ((System.ComponentModel.ISupportInitialize)(this)).EndInit();
        this.ResumeLayout(false);
    }

    private void RefreshArea(Graphics g)
    {
        g.Clear(Color.White);
        drawStates(g);
        drawLinks(g);
        drawSignals(g);
        drawTimeMarks(g);
    }

    public void createStates()
    {
        int stateDefaultCentreX = StateCentre;
        int stateDefaultCentreY = StateCentre;
        int offsetStateX = OffsetX;
        int offsetStateY = OffsetY;
        foreach (Rule rule in this.Rules)

```

```

        {
            State startState = rule.startState;
            State endState = rule.endState;
            State[] startAndEndStates = { startState,
endState };

            foreach (State state in startAndEndStates)
            {
                State s = getStateByName(state.Name);
                if (s == null)
                {
                    state.paintDot.X = States.Length % 2
== 0 ? stateDefaultCentreX : stateDefaultCentreX + offsetStateX;
                    state.paintDot.Y =
stateDefaultCentreY + States.Length / 2 * offsetStateY;
                    addState(state);
                }
            }
        }

        private void drawStates(Graphics g)
        {
            int stateDiametr = StateDiametr;
            foreach (State state in States)
            {
                state.calculateLocation();
                Pen statePen;
                if (!state.Active && !state.Selected)
statePen = stateDefaultPen;
                else if (!state.Active && state.Selected)
statePen = stateSelectedPen;
                else if (state.Active && !state.Selected)
statePen = stateActivePen;
                else statePen = stateActiveAndSelectedPen;
                g.DrawEllipse(statePen, state.paintDot.X,
state.paintDot.Y, stateDiametr, stateDiametr);
                if (state.Active)
g.FillEllipse(stateActiveBrush, state.paintDot.X,
state.paintDot.Y, stateDiametr, stateDiametr);
                g.DrawString(state.Name, defaultTextFont,
defaultTextBrush, state.nameDot.X, state.nameDot.Y);
            }
        }

        private void cancelStates()
        {
            States = new State[0];
        }

        public void createLinks()
        {
            foreach (Rule rule in Rules)
            {
                Link link =
getLinkByName(rule.startState.Name + rule.endState.Name);
                if (link == null)

```

```

        {
            Array.Resize(ref Links, Links.Length +
1);
            link = new Link(rule.startState,
rule.endState);
            link.setName();
            link.Arc =
link.startState.Equals(link.endState);
            link.startState.links.Add(link);
            link.endState.links.Add(link);
            Links[Links.Length - 1] = link;
        }
        if (rule.SignalInventered)
        {
            Array.Resize(ref link.inventeredSignals,
link.inventeredSignals.Length + 1);

link.inventeredSignals[link.inventeredSignals.Length - 1] =
rule.signal;
        }
        if (rule.signal != null)
        {
            Array.Resize(ref rule.signal.links,
rule.signal.links.Length + 1);

rule.signal.links[rule.signal.links.Length - 1] = link;
            Array.Resize(ref link.signals,
link.signals.Length + 1);
            link.signals[link.signals.Length - 1] =
rule.signal;
        }
    }

    private void drawLinks(Graphics g)
    {
        int arcRadius = ArcCyrcleradius;
        int arcStartAngle = ArcStartAngle;
        int arcSweepAngle = ArcSweepAngle;
        foreach (Link link in Links)
        {
            Pen linkPen = link.Selected ? linkSelectedPen
: linkDefaultPen;
            link.calculateLocation();
            if (link.Arc)
            {
                g.DrawArc(linkPen, new
RectangleF(link.arcDot.X, link.arcDot.Y, arcRadius * 2,
arcRadius * 2), arcStartAngle, arcSweepAngle);
            }
            else
            {
                g.DrawLine(linkPen, link.startDot.X,
link.startDot.Y, link.endDot.X, link.endDot.Y);
                g.DrawLine(linkPen, link.endDot.X,
link.endDot.Y, link.arrowDots[0].X, link.arrowDots[0].Y);
            }
        }
    }
}

```

```

        g.DrawLine(linkPen, link.endDot.X,
link.endDot.Y, link.arrowDots[1].X, link.arrowDots[1].Y);
    }
}
private void cancelLinks()
{
    Links = new Link[0];
}

public void createSignals()
{
    foreach (Rule rule in Rules)
    {
        Signal s = getSignalByName(rule.signal.name);
        if (s == null)
        {
            s = rule.signal;
            Array.Resize(ref Signals, Signals.Length
+ 1);

            Signals[Signals.Length - 1] = s;
        }
    }
}

public void drawSignals(Graphics g)
{
    foreach (Signal s in Signals)
    {
        s.calculateLocation();
        foreach (Point d in s.paintDots)
        {
            Font font = (s.Selected) ?
signalSelectedFont : signalDefaultFont;
            bool isActive = s.isInvert(d) ? !s.Active
: s.Active;
            Brush brush = (isActive) ?
signalActiveBrush: signalDefaultBrush;
            String printedName =
(s.invertedPoint.Contains(d)) ? "!" + s.name : s.name;
            g.DrawString(printedName, font, brush,
d.X, d.Y);
        }
    }
}

private void cancelSignals()
{
    Signals = new Signal[0];
}

public void createTimeMarks()
{
    foreach (Rule r in Rules)
    {
        if (r.timeMark == null) continue;

```

```

        Time timeMark =
getTimeMarkByName(r.timeMark.name);
        if (timeMark == null)
        {
            timeMark = r.timeMark;
            Array.Resize(ref TimeMarks,
TimeMarks.Length + 1);
            TimeMarks[TimeMarks.Length - 1] =
timeMark;
        }
        Link l = getLinkByName(r.startState.Name +
r.endState.Name);
        if (l == null) continue;
        Array.Resize(ref timeMark.links,
timeMark.links.Length + 1);
        timeMark.links[timeMark.links.Length - 1] =
l;
    }
}

public void drawTimeMarks(Graphics g)
{
    foreach (Time t in TimeMarks)
    {
        t.calculateLocation();
        foreach (Point d in t.paintDots)
        {
            Font font = (t.selected) ?
signalSelectedFont : signalDefaultFont;
            Brush brush = signalDefaultBrush;
            g.DrawString(t.name + "<" + t.value +
">", font, brush, d.X, d.Y);
        }
    }
}

private void cancelTimeMarks()
{
    TimeMarks = new Time[0];
}

public void cancelDrawElements()
{
    cancelStates();
    cancelLinks();
    cancelSignals();
    cancelTimeMarks();
}

public Link getLinkByName(String name)
{
    foreach (Link link in this.Links)
    {
        if (link.name.Equals(name)) return link;
    }
    return null;
}

```

```

private Signal getSignalByName(String name)
{
    foreach (Signal signal in Signals)
    {
        if (signal.name.Equals(name)) return signal;
    }
    return null;
}

private State getStateByName(String name)
{
    foreach (State state in States)
    {
        if (state.Name.Equals(name)) return state;
    }
    return null;
}

private Time getTimeMarkByName(String name)
{
    foreach(Time t in TimeMarks)
    {
        if (t.name.Equals(name)) return t;
    }
    return null;
}

public State getActiveState()
{
    foreach (State s in States)
    {
        if (s.Active) return s;
    }
    return null;
}

public List<Rule> getRulesWithActiveState()
{
    List<Rule> activeRules = new List<Rule>();
    State activeState = getActiveState();
    foreach(Rule r in Rules)
    {
        if (r.startState.Equals(activeState))
            activeRules.Add(r);
    }
    return activeRules;
}

private void addState(State state)
{
    State currentState = getStateByName(state.Name);
    if (currentState == null)
    {
        Array.Resize(ref States, States.Length + 1);
        States[States.Length - 1] = state;
    }
}

```

```

    }
}

private bool isDotOnLink(Point dot, Link link)
{
    float x = dot.X;
    float y = dot.Y;
    if (link.Arc)
    {
        int arcRadius = ArcCyrcleradius;
        Point centreArc = new Point(link.arcDot.X +
arcRadius, link.arcDot.Y + arcRadius);
        //Уравнение окружности вида (x-x.centre)^2 +
(y-y.centre)^2 = r^2, представляющей часть арки
        int result = (int)(Math.Pow(x - centreArc.X,
2) + Math.Pow(y - centreArc.Y, 2));
        int squareRadius = (int)Math.Pow(arcRadius,
2);
        bool dotOnPaintedArc = !(x >= centreArc.X &&
y >= centreArc.Y); //Переменная отвечающая, что точка лежит на
отрисованной части окружности - арке
        return result >= (squareRadius - 100) &&
result <= (squareRadius + 100) && dotOnPaintedArc;
    }
    else
    {
        //Каноническое уравнение прямой на плоскости
        типа (x-x1)/(x2-x1) = (y-y1)/(y2-y1)
        float result = (link.startDot.Y -
link.endDot.Y) * x + (link.endDot.X - link.startDot.X) * y +
(link.startDot.X * link.endDot.Y - link.endDot.X *
link.startDot.Y);
        int offset = 10;
        float maxX = Math.Max(link.startDot.X,
link.endDot.X) + offset;
        float minX = Math.Min(link.startDot.X,
link.endDot.X) - offset;
        float maxY = Math.Max(link.startDot.Y,
link.endDot.Y) + offset;
        float minY = Math.Min(link.startDot.Y,
link.endDot.Y) - offset;
        return (result >= -2500
&& result <= 2500
&& x >= minX && x <= maxX
&& y >= minY && y <= maxY);
    }
}

public bool isDotOnState(Point dot, State state)
{
    int diametr = StateDiametr;
    return dot.X >= state.paintDot.X
&& dot.X <= state.paintDot.X + diametr
&& dot.Y >= state.paintDot.Y
&& dot.Y <= state.paintDot.Y + diametr;
}

```

```

public bool isDotOnSignal(Point dot, Signal signal)
{
    float offset = 20;
    bool isOnSignal = false;
    foreach (Point pDot in signal.paintDots)
    {
        isOnSignal = isOnSignal || (dot.X >= pDot.X
            && dot.X <= pDot.X + offset
            && dot.Y >= pDot.Y
            && dot.Y <= pDot.Y + offset);
    }
    return isOnSignal;
}

private bool isDotOnTimeMark(Point dot, Time tm)
{
    int offset = 10 * tm.name.Length;
    bool isOnMark = false;
    foreach (Point pd in tm.paintDots)
    {
        isOnMark = isOnMark
            || dot.X >= pd.X
            && dot.X <= pd.X + offset
            && dot.Y >= pd.Y
            && dot.Y <= pd.Y + offset;
    }
    return isOnMark;
}

private bool setSelectedLink(Point dot)
{
    bool isChanged = false;
    foreach (Link link in Links)
    {
        bool isLink = isDotOnLink(dot, link);
        isChanged = isChanged || link.Selected ^
isLink;

        link.Selected = isLink;
    }
    return isChanged;
}

private bool setSelectedState(Point dot)
{
    bool isChanged = false;
    foreach (State state in States)
    {
        bool isState = isDotOnState(dot, state);
        isChanged = isChanged || state.Selected ^
isState;

        state.Selected = isState;
    }
    return isChanged;
}

```

```

private bool setActiveState()
{
    bool isChanged = false;
    foreach (State s in States)
    {
        bool isNeedChangeActivity = s.Selected &&
PaintDotSelectedState.Equals(s.paintDot);
        isChanged = isChanged ||
isNeedChangeActivity;
        if (isNeedChangeActivity)
        {
            s.Active = !s.Active;
            cancelStates(s);
        }
    }
    return isChanged;
}

private void cancelStates(State state)
{
    foreach (State s in States)
    {
        s.Active = s.Equals(state) && s.Active;
    }
}

private bool setSelectedTimeMarks(Point dot)
{
    bool isChanged = false;
    foreach (Time tm in TimeMarks)
    {
        bool isOnMark = isDotOnTimeMark(dot, tm);
        isChanged = isChanged || tm.selected ^
isOnMark;
        tm.selected = isOnMark;
    }
    return isChanged;
}

private bool setActiveSignal()
{
    bool isChanged = false;
    foreach (Signal s in Signals)
    {
        bool newActivityStatus = s.Selected && !
s.Active || !s.Selected && s.Active;
        isChanged = isChanged || s.Active ^
newActivityStatus;
        s.Active = newActivityStatus;
    }
    return isChanged;
}

private bool setSelectedSignal(Point dot)
{
    bool isSignalsChanged = false;

```

```

        foreach (Signal signal in Signals)
        {
            bool isOnSignal = isDotOnSignal(dot, signal);
            isSignalsChanged = isSignalsChanged ||
signal.Selected ^ isOnSignal;
            signal.Selected = isOnSignal;
        }
        return isSignalsChanged;
    }

    private bool setValueTimeMark(Point dot)
    {
        bool tbVisible = false;
        Point tbLocation = Point.Empty;
        Time selectedTm = null;
        foreach (Time tm in TimeMarks)
        {
            tbVisible = tbVisible || tm.selected;
            if (tm.selected)
            {
                tbLocation = dot;
                selectedTm = tm;
            }
        }
        timeTb.Visible = tbVisible;
        timeTb.Location = tbLocation;
        timeTb.timeMark = selectedTm;
        return true;
    }

    private bool isElementsChanged(Point dot)
    {
        bool isLinkChanged = setSelectedLink(dot);
        bool isStateChanged = setSelectedState(dot);
        bool isSignalChanged = setSelectedSignal(dot);
        bool isTimeMarkChanged =
setSelectedTimeMarks(dot);
        return isLinkChanged
            || isStateChanged
            || isSignalChanged
            || isTimeMarkChanged;
    }

    private void relocationStates(Point dot)
    {
        foreach (State state in States)
        {
            if (state.Selected)
            {
                state.paintDot.X = dot.X -
(int)DragOffsetX;
                state.paintDot.Y = dot.Y -
(int)DragOffsetY;
                foreach (Link l in state.links)
                {
                    l.calculateLocation();
                }
            }
        }
    }

```

```

        if (l.lengthLink < 0 && !l.Arc)
        {
            int direction =
state.Equals(l.startState) ? -1 : 1;
            state.paintDot.X += (int) (l.cosx
* Math.Abs(l.lengthLink) * direction);
            state.paintDot.Y += (int) (l.sinx
* Math.Abs(l.lengthLink) * direction);
        }
    }
}

e)
private void mouseMove(object sender, MouseEventArgs
{
    Point dot = e.Location;
    if (e.Button.Equals(MouseButtons.Left))
    {
        relocationStates(dot);
    }
    if (isElementsChanged(dot) ||
e.Button.Equals(MouseButtons.Left))
    {
        Refresh();
    }
}

e)
private void mouseDown(object sender, MouseEventArgs
{
    foreach (State state in States)
    {
        if (state.Selected)
        {
            DragOffsetX = e.X - state.paintDot.X;
            DragOffsetY = e.Y - state.paintDot.Y;
            PaintDotSelectedState = new
Point(state.paintDot.X, state.paintDot.Y);
        }
    }
}

e)
private void mouseClicked(object sender, MouseEventArgs
{
    DragOffsetX = 0;
    DragOffsetY = 0;
    bool isStateChangeActivity = !SimulStarted ?
setActiveState() : false;
    bool isSignalChangeActivity = setActiveSignal();
    bool s = setValueTimeMark(e.Location);
    if (isStateChangeActivity ||
isSignalChangeActivity || s) Refresh();
}

```

```

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            e.Graphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.HighQuality;//Включаем
сглаживание графических объектов
            e.Graphics.TextRenderingHint =
System.Drawing.Text.TextRenderingHint.ClearTypeGridFit;//Включае
м сглаживание шрифтов
            e.Graphics.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;//
Включаем интерполяцию
            RefreshArea(e.Graphics);
        }

        public Parser Parser
        {
            get => default;
            set
            {
            }
        }

        public Form1 Form1
        {
            get => default;
            set
            {
            }
        }
    }
}

```

Файл «ToolPanel.cs»

```

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using sku_to_smv.Properties;
using sku_to_smv.src;

namespace sku_to_smv
{
    enum MouseState { NONE, CLICK, HOVER };
    enum Alignment { TOP, BOTTOM, LEFT, RIGHT };
    class ToolPanel : Control
    {
        public Orientation PanelOrientation;
        public Alignment PanelAlignment;
        public Collection<Button> Buttons;
        //public DrawArea area;
        public static readonly Brush ToolPanelBrush = new

```

```

SolidBrush(SystemColors.Control);
    private static ToolPanel _instance;

    public static ToolPanel getInstance()
    {
        if (_instance == null)
        {
            _instance = new ToolPanel();
        }
        return _instance;
    }

    public delegate void cToolButtonEventHandler(object
sender, ToolButtonEventArgs a);
    public event cToolButtonEventHandler ButtonClicked;

    private ToolPanel()
    {
        Buttons = new Collection<Button>();
        PanelOrientation = Orientation.Vertical;
        Location = new Point(0, 0);
        Size = new Size(41, 500);
        BackColor = Color.Pink;
        PanelOrientation = Orientation.Vertical;
        PanelAlignment = Alignment.RIGHT;

        Button startButtnon = new ToolButton();
        //startButtnon.InactiveImage =
Resources.create_simulation;
        startButtnon.Image = Resources.play;
        startButtnon.Name = "start";
        startButtnon.Text = "Запустить симуляцию";
        //startButtnon.Location = Point.Empty;
        startButtnon.Click += new
EventHandler(onClickSimulStart);
        CalculateLocation(ref startButtnon);
        Buttons.Add(startButtnon);

        /*Button runButton = new ToolButton();
        //runButton.SetInactiveImage(Resources.play_grey
);

        runButton.Image = Resources.play;
        runButton.Name = "run";
        runButton.Text = "Запуск симуляции";
        CalculateLocation(ref runButton);
        Buttons.Add(runButton);*/

        Button stepButton = new ToolButton();
        //stepButton.SetInactiveImage(Resources.step_gre
y);

        stepButton.Image = Resources.step;
        stepButton.Name = "step";
        stepButton.Text = "Шаг с остановом";
        CalculateLocation(ref stepButton);
        stepButton.Enabled = false;
        stepButton.Click += new

```

```

EventHandler(onClickStep);
    Buttons.Add(stepButton);

    Button stopSimulation = new ToolButton();
    //stopSimulation.SetInactiveImage(Resources.stop
_grey);

    stopSimulation.Image = Resources.stop;
    stopSimulation.Name = "stop";
    stopSimulation.Text = "Остановить симуляцию";
    CalculateLocation(ref stopSimulation);
    stopSimulation.Enabled = false;
    stopSimulation.Click += onClickStop;
    Buttons.Add(stopSimulation);

    /*Button signalsTable = new ToolButton();
    //signalsTable.SetInactiveImage(Resources.table_
grey);

    signalsTable.Image = Resources.table;
    signalsTable.Name = "table";
    signalsTable.Text = "Таблица сигналов";
    CalculateLocation(ref signalsTable);
    Buttons.Add(signalsTable);

    Button cancelSignals = new ToolButton();
    //cancelSignals.SetInactiveImage(Resources.table
_grey);

    cancelSignals.Image = Resources.reset_all;
    cancelSignals.Name = "reset";
    cancelSignals.Text = "Сбросить все сигналы";
    CalculateLocation(ref cancelSignals);
    Buttons.Add(cancelSignals);

    Button showLog = new ToolButton();
    //showLog.SetInactiveImage(Resources.table_grey)
;

    showLog.Image = Resources.show_log;
    showLog.Name = "showlog";
    showLog.Text = "Показать лог-файл";
    CalculateLocation(ref showLog);
    Buttons.Add(showLog);*/
    Controls.AddRange(Buttons.ToArray());
}
public void CalculateLocation(ref Button button)
{
    int x = 3;
    int y = 3;

    if
    (PanelOrientation.Equals(Orientation.Vertical))
    {
        y += (button.Size.Height + 5) *
Buttons.Count;
    }
    else
    {
        x += (button.Size.Width + 5) *
Buttons.Count;
    }
}

```

```

    }
    button.Location = new Point(x, y);
}
/*public void UpdateControlsLocation()
{
    switch (Settings.Default.ToolsPanelLocation)
    {
        case 0:
            PanelOrientation = Orientation.Vertical;
            PanelAlignment = Alignment.LEFT;
            break;
        case 1:
            PanelOrientation =
Orientation.Horizontal;
            PanelAlignment = Alignment.TOP;
            break;
        case 2:
            PanelOrientation = Orientation.Vertical;
            PanelAlignment = Alignment.RIGHT;
            break;
        case 3:
            PanelOrientation =
Orientation.Horizontal;
            PanelAlignment = Alignment.BOTTOM;
            break;
    }
}*/

private void onClickSimulStart(object sender,
EventArgs e)
{
    DrawArea area = DrawArea.GetInstance();
    State activeState = area.getActiveState();
    if (activeState == null)
    {
        MessageBox.activeStateIsNull();
        return;
    }
    area.SimulStarted = true;
    Buttons[0].Enabled = false;
    Buttons[1].Enabled = true;
    Buttons[2].Enabled = true;
}

private void onClickStep(object sender, EventArgs e)
{
    DrawArea area = DrawArea.GetInstance();
    State activeState = area.getActiveState();
    if (activeState == null) return;
    List<Rule> activeRules =
area.getRulesWithActiveState();
    bool anythingTransferred = false;
    bool transferredArc = false;
    foreach (Rule r in activeRules)
    {
        Link l =

```

```

area.getLinkByName(r.startState.Name + r.endState.Name);
    if (l == null) return;
    bool linkActive = l.linkActive();
    r.endState.Active = linkActive;
    transferedArc = l.Arc && linkActive;
    anythingTransfered = anythingTransfered ||
linkActive;
    }
    activeState.Active = !anythingTransfered ||
transferedArc;
    area.Refresh();
}

private void onClickStop(object sender, EventArgs e)
{
    DrawArea area = DrawArea.GetInstance();
    area.SimulStarted = false;
    Buttons[0].Enabled = true;
    Buttons[1].Enabled = false;
    Buttons[2].Enabled = false;
}
}

public class ToolButtonEventArgs : EventArgs
{
    public ToolButtonEventArgs(string s)
    {
        ButtonName = s;
    }
    private string ButtonName;
    public string Name
    {
        get { return ButtonName; }
    }
}

class ToolButton : Button
{
    Bitmap image;
    Bitmap imageInactive;
    Bitmap frame;
    Bitmap frame_hover;
    Bitmap frame_click;
    MouseState ms;

    public ToolButton()
    {
        image = null;
        imageInactive = null;
        frame = null;
        frame_hover = null;
        frame_click = null;
        Location = Point.Empty;
        Size = new Size(35, 35);
        BackColor = SystemColors.Control;
    }
}

```

```

        Name = "Button";
        Text = "toolButton";
        Enabled = true;
        Visible = true;
        ms = MouseState.NONE;
        BackColor = Color.FromArgb(0, 0, 0, 0);
        SetFrame(Resources.frame);
        SetFrameHover(Resources.frame_hover);
        SetFrameClick(Resources.frame_click);
    }

    public Form1 Form1
    {
        get => default;
        set
        {
        }
    }

    public void Draw(Graphics g)
    {
        g.Clear(Color.White);
        if (Visible)
        {
            //g.FillRectangle(new SolidBrush(BackColor),
            new Rectangle(Location, Size));
            if (true)
            {
                if (Image != null)

                    g.DrawImage(Image, new Rectangle(new
                    Point(Location.X + 4, Location.Y + 4), new Size(22, 22)));
                /*switch (ms)
                {
                    case MouseState.NONE:
                        if (frame != null)
                            g.DrawImage(frame, new
                            Rectangle(Location, Size));
                        break;
                    case MouseState.HOVER:
                        if (frame_hover != null)
                            g.DrawImage(frame_hover, new
                            Rectangle(Location, Size));
                        break;
                    case MouseState.CLICK:
                        if (frame_click != null)
                            g.DrawImage(frame_click, new
                            Rectangle(Location, Size));
                        break;
                }*/
            }
            else
            {
                /*if (imageInactive != null)
                    g.DrawImage(imageInactive, new
                    Rectangle(new Point(Location.X + 4, Location.Y + 4), new

```

```

Size(22, 22)));
                                if (frame != null)
                                    g.DrawImage(frame, new
Rectangle(Location, Size));*/
                                }
                            }
public void SetImage(String path)
{
    try
    {
        image = new Bitmap(path);
        image.MakeTransparent(System.Drawing.Color.M
agenta);
    }
    catch (System.IO.FileNotFoundException)
    {
        image = null;
    }
}
public void SetImage(Bitmap im)
{
    image = im;
    image.MakeTransparent(System.Drawing.Color.Magen
ta);
}
public void SetInactiveImage(String path)
{
    try
    {
        imageInactive = new Bitmap(path);
        imageInactive.MakeTransparent(System.Drawing
.Color.Magenta);
    }
    catch (System.IO.FileNotFoundException)
    {
        imageInactive = null;
    }
}
public void SetInactiveImage(Bitmap im)
{
    imageInactive = im;
    imageInactive.MakeTransparent(System.Drawing.Col
or.Magenta);
}
public void setFrame(String path)
{
    try
    {
        frame = new Bitmap(path);
        frame.MakeTransparent(System.Drawing.Color.M
agenta);
    }
    catch (System.IO.FileNotFoundException)
    {
        frame = null;
    }
}

```

```

    }
}
public void SetFrame(Bitmap im)
{
    frame = im;
    frame.MakeTransparent(System.Drawing.Color.Magenta);
}
public void SetFrameHover(String path)
{
    try
    {
        frame_hover = new Bitmap(path);
        frame_hover.MakeTransparent(System.Drawing.Color.Magenta);
    }
    catch (System.IO.FileNotFoundException)
    {
        frame_hover = null;
    }
}
public void SetFrameHover(Bitmap im)
{
    frame_hover = im;
    frame_hover.MakeTransparent(System.Drawing.Color.Magenta);
}
public void SetFrameClick(String path)
{
    try
    {
        frame_click = new Bitmap(path);
        frame_click.MakeTransparent(System.Drawing.Color.Magenta);
    }
    catch (System.IO.FileNotFoundException)
    {
        frame_click = null;
    }
}
public void SetFrameClick(Bitmap im)
{
    frame_click = im;
    frame_click.MakeTransparent(System.Drawing.Color.Magenta);
}
public MouseState CheckMouseState(MouseEventArgs e)
{
    if (e != null)
        if ((e.X > Location.X && e.X < Location.X +
Size.Width && e.Y > Location.Y && e.Y < Location.Y +
Size.Height) && Visible)
        {
            if (e.Button == MouseButton.Left &&
Enabled)
            {

```

```
        return ms = MouseState.CLICK;
    }
    return ms = MouseState.HOVER;
}
return ms = MouseState.NONE;
}
}
```

Приложение В. Презентация

Инструментальная система темпорального анализа моделей параллельных вычислительных систем

Автор работы: студент группы 16ВВ1 Борискин Вячеслав Владимирович
Научный руководитель: к.т.н., доцент Карамышева Надежда Сергеевна

Пенза, 2020

Цель и объект работы

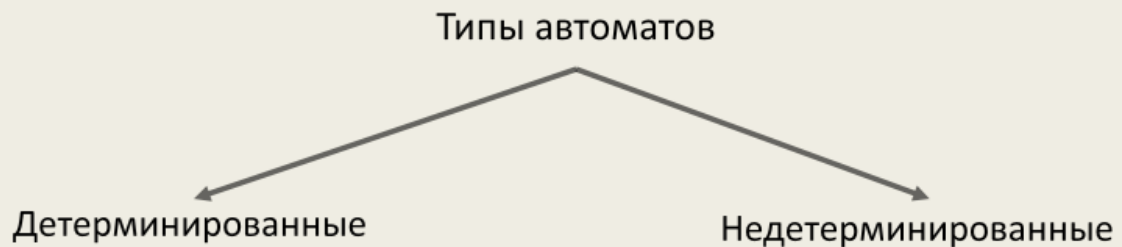
Цель работы – разработать инструментальную среду темпорального анализа моделей параллельных вычислительных систем

Объект работы – приложение, выполняющее следующие функции

- Проектирование вычислительных систем
- Общий анализ вычислительных систем
- Моделирование поведения вычислительных систем

Автоматы

Абстрактные автоматы – вычислительные машины, представленные в виде математических моделей, осуществляющие обработку информации по заданному алгоритму



3

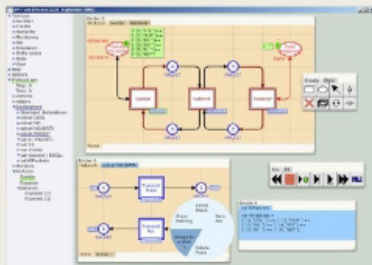
Методы темпорального анализа ВС

1. Метод применения **автомата с таймаутами**
2. Метод применения **автомата с временными ограничениями**

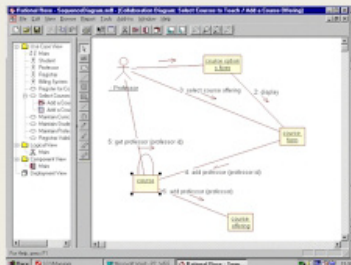
4

Доступные аналоги

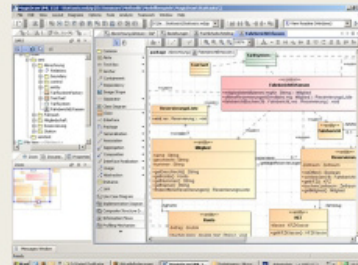
CPN Tools



Rotation Rose

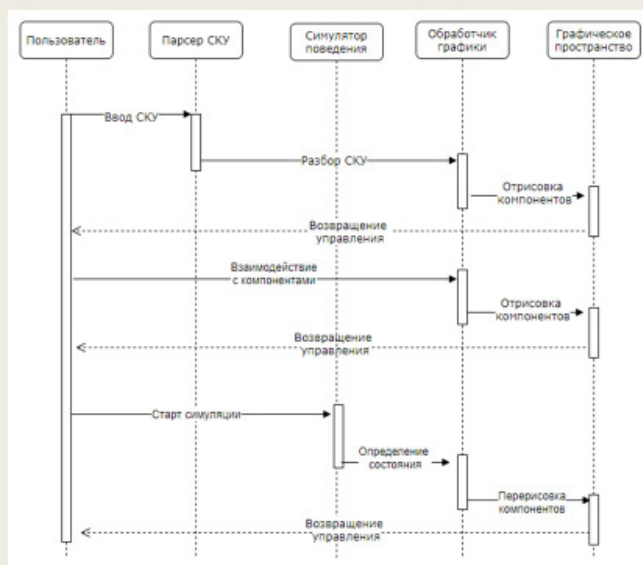


MagicDraw



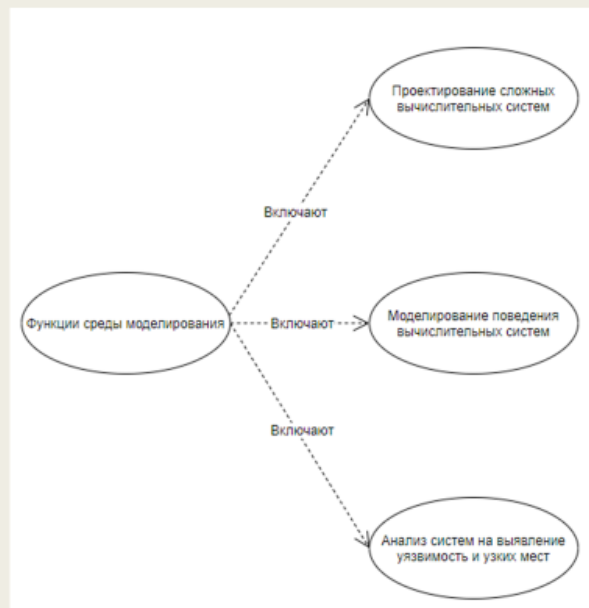
5

UML-диаграмма последовательности



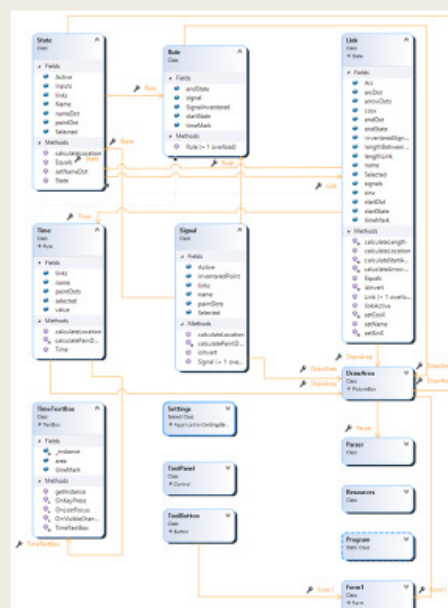
6

UML-диаграмма вариантов использования



7

UML-диаграмма классов



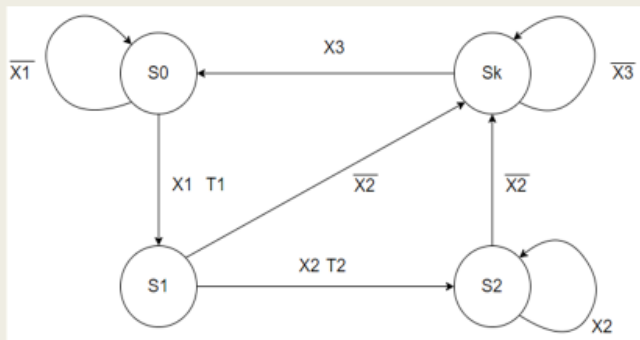
8

Стек технологий



9

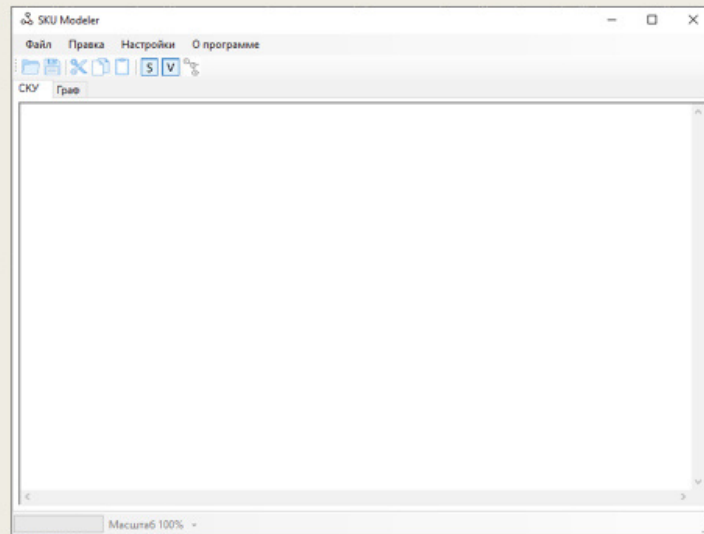
Временной автомат моделируемой системы



S_0	Начальное состояние
S_1	Состояние стирки
S_2	Откачка воды
S_k	Стирка завершена
X_1	Сигнал начала стирки
X_2	Сигнал с датчика воды
X_3	Сигнал закрытия дверцы барабана
T_1	Время набора воды
T_2	Время стирки

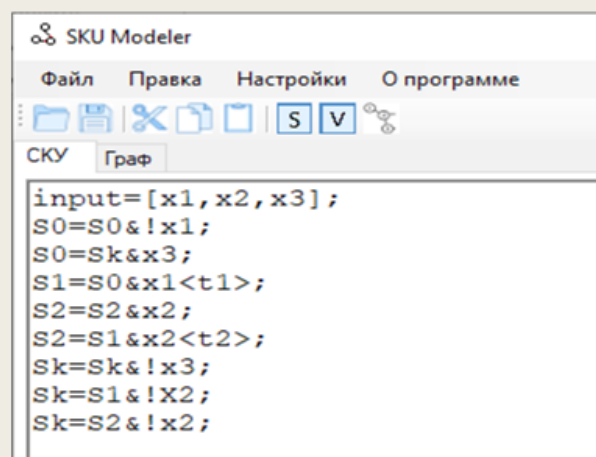
10

Текстовое пространство



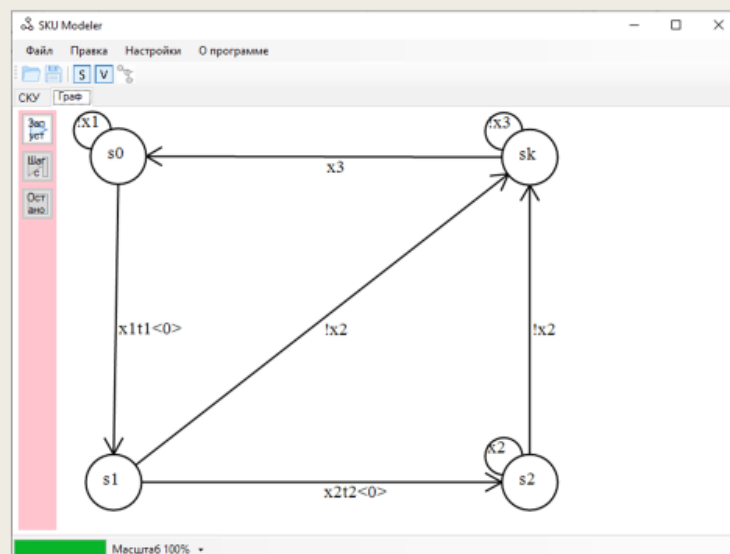
11

Ввод SKU



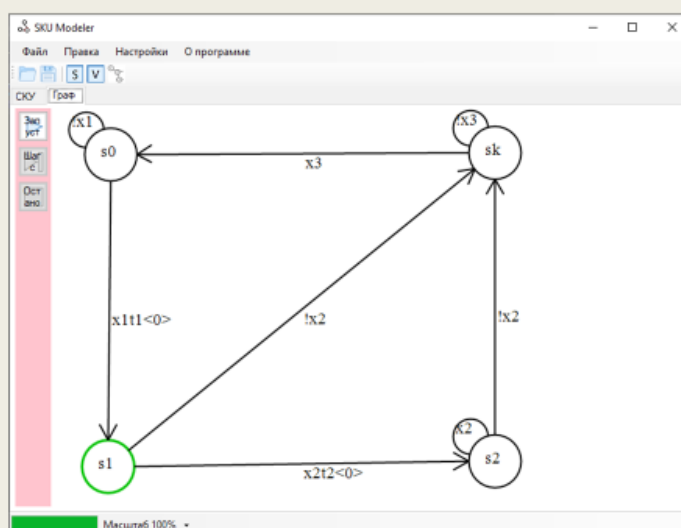
12

Временной автомат



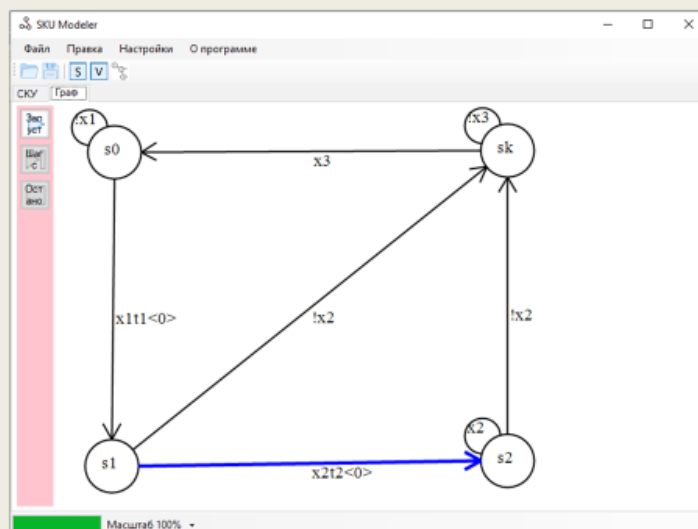
13

Взаимодействие с автоматом



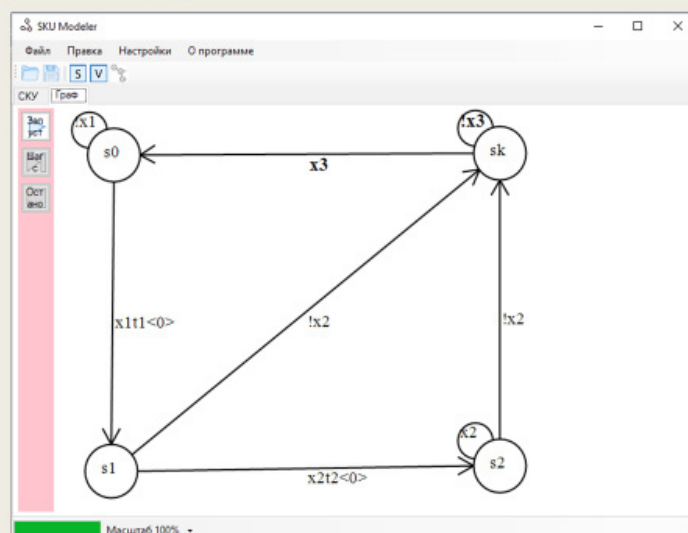
14

Взаимодействие с автоматом



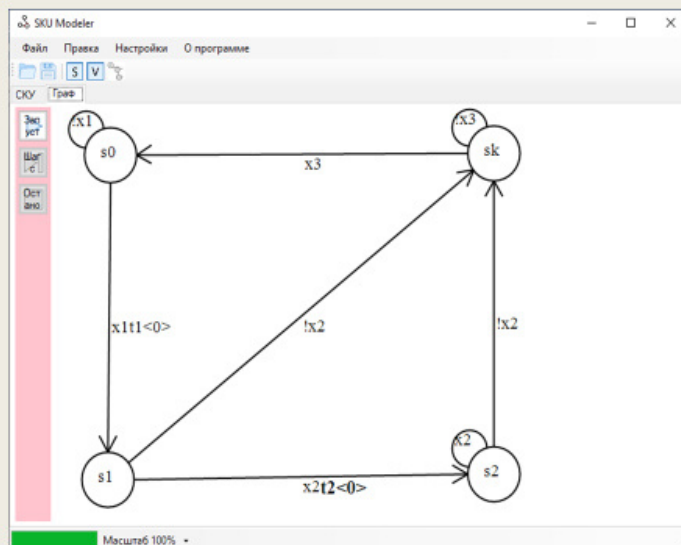
15

Взаимодействие с автоматом



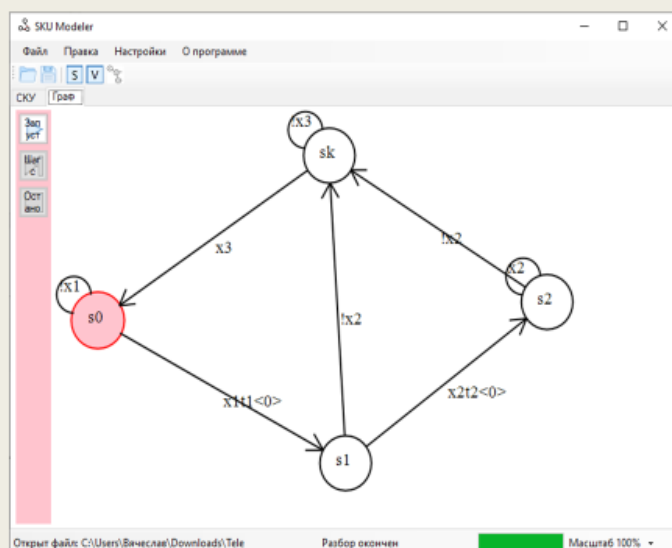
16

Взаимодействие с автоматом



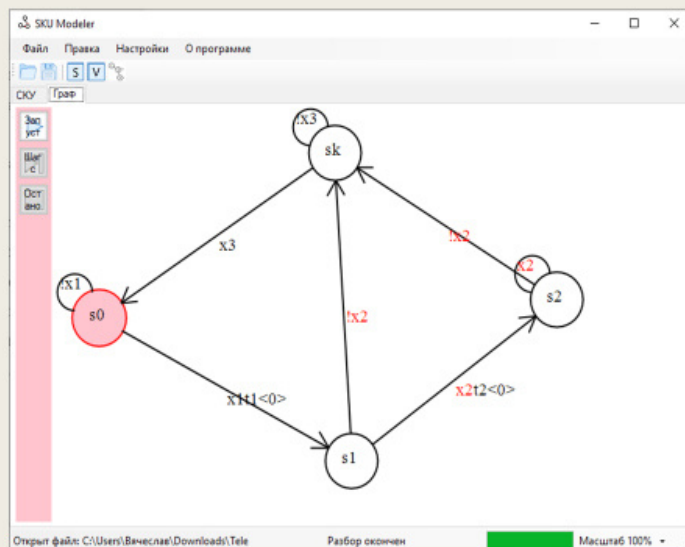
17

Установка начального состояния



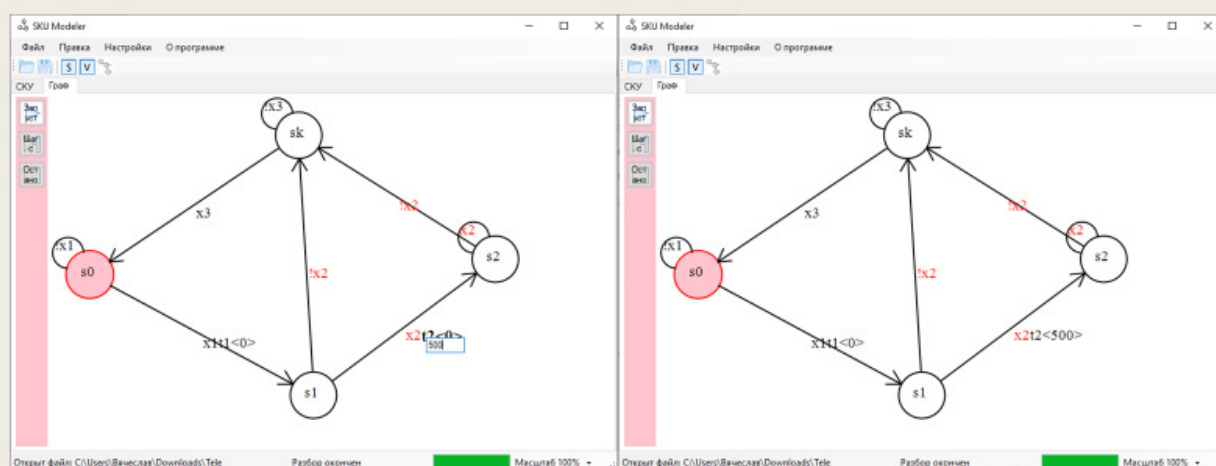
18

Установка входных сигналов



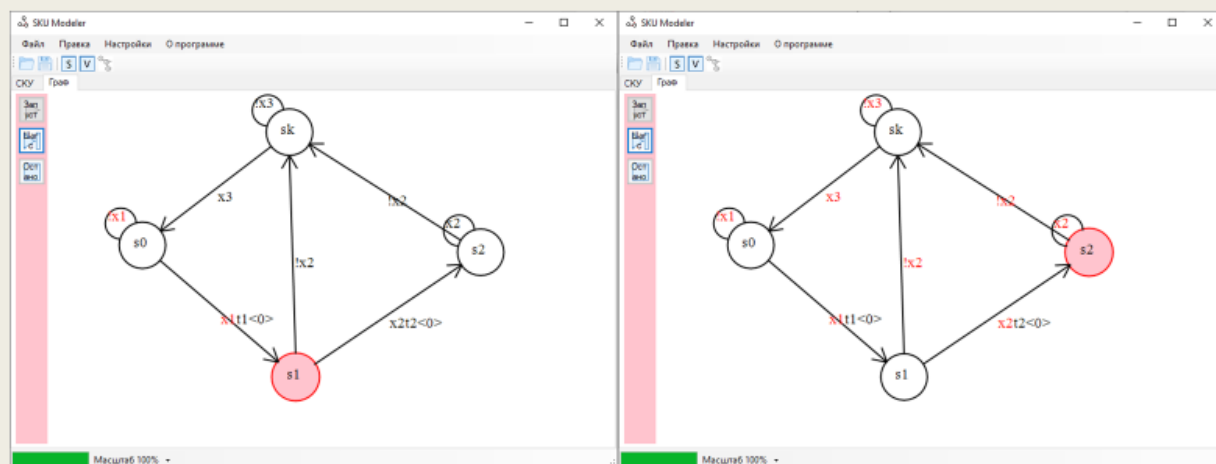
19

Установка временной задержки



20

Симуляция работы



21

Результаты работы

- Разработано и отлажено приложение, выполняющее задачи проектирования анализа и моделирования поведения вычислительных систем;
- Разработан и отлажен алгоритм разбора СКУ, описывающей моделируемую ВС;
- Разработан и отлажен алгоритм отображения графической модели ВС, представленной в виде временного автомата;
- Разработан и отлажен алгоритм симуляции работы ВС;
- Приложение было представлено на семинаре по электронным и сетевым технологиям MWENT-2020. Результаты трудов семинара проиндексированы рефератной базой данных SCOPUS.

22

Спасибо за внимание!