Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Кафедра информационных технологий
в исследовании дискретных структур (КИТИДиС)

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р. физ.-мат. наук
_____ Д.Я. Суханов
подпись
«22» 06 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА
(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

СИСТЕМА АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПРОГРАММ НА ПЛАТФОРМЕ QT НА ОСНОВЕ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

по направлению подготовки 03.04.03 Радиофизика
направленность (профиль) «Радиофизика, электроника и информационные системы»

Фоминых Александра Федоровна

Руководитель ВКР
канд. физ.-мат. наук, доцент
_____ М.Л. Громов
«09» 06 2023 г.

Автор работы
студент группы № 072176
_____ А.Ф. Фоминых
«09» 06 2023 г.

Томск – 2023

Министерство науки и высшего образования Российской Федерации.

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Наименование учебного структурного подразделения

УТВЕРЖДАЮ
Руководитель ООП
д-р. физ.-мат. наук
_____ Д.Я. Суханов
подпись
«27» 09 2021 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы магистра обучающемуся Фоминых Александре Федоровне

Фамилия Имя Отчество обучающегося

по направлению подготовки 03.04.03 Радиофизика, направленность (профиль) «Радиофизика, электроника и информационные системы»

1 Тема выпускной квалификационной работы
Система автоматизации разработки программ на платформе QT на основе автоматного программирования

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис / деканат – _____ б) в ГЭК – _____

3 Исходные данные к работе:
Объект исследования – разработка программ на платформе QT на основе автоматного программирования
Предмет исследования – автоматное программирование, автоматные композиции, платформа Qt, разработка программ на основе формальных моделей
Цель исследования – создание системы генерации шаблонов программ для автоматизации разработки ПО на платформе Qt на основе автоматного программирования

Задачи:
Изучение автоматного программирования
Разработать автоматное описание программ
Изучение возможностей платформы Qt
Разработка системы генерации шаблонов программ для платформы Qt
Создание пользовательского интерфейса
Внедрение в пользовательский интерфейс систему генерации шаблонов программ

Методы исследования:
Методы описания формальных моделей, принципы автоматного программирования и объектно-ориентированного программирования. А также экспериментальные исследования

Организация или отрасль, по тематике которой выполняется работа, – программная индустрия, специализирующаяся на разработке инструментов и систем программирования,

4 Краткое содержание работы
В работе будут рассмотрены основные принципы автоматного программирования, а

также исследованы возможности и ограничения использования данного подхода в контексте разработки на платформе Qt. Предполагается проведение экспериментов и анализ результатов для оценки эффективности и применимости разработанной системы

Руководитель выпускной квалификационной работы

Доцент, НИ ТГУ		М.Л. Громов
должность, место работы	*подпись*	*И.О. Фамилия*

Задание принял к исполнению

Студент, НИ ТГУ		А.Ф. Фоминых
должность, место работы	*подпись*	*И.О. Фамилия*

РЕФЕРАТ

Магистерская диссертация содержит 58 листов, 3 главы, 12 рисунков, 30 источников.

АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ, АВТОМАТНЫЕ КОМПОЗИЦИИ, ПЛАТФОРМА QT, РАЗРАБОТКА ПРОГРАММ НА ОСНОВЕ ФОРМАЛЬНЫХ МОДЕЛЕЙ

В диссертации рассматриваются основные понятия и определения, которые являются основой работы. Вводятся такие понятия, как автомат, композиция автоматов и ее типы, а также дается определение автоматного программирования и принципы его работы. Это позволяет создать теоретическую основу для разработки системы автоматизации разработки программ на платформе Qt. Также были рассмотрены и описаны основные компоненты, предоставляемые платформой Qt, и рассматриваются ее возможности для разработки программ. Исследование данных компонентов и возможностей позволяет выявить преимущества и потенциал платформы Qt в контексте автоматизации разработки программ

Одним из главных результатов магистерской диссертации является разработка системы генерации шаблонов программ на платформе Qt на основе автоматного описания. В работе описывается архитектура системы, предлагается язык автоматного описания программ, разрабатывается модуль генерации шаблонов и создается пользовательский графический интерфейс. Эта система позволяет автоматизировать процесс разработки программ на платформе Qt.

Таким образом, магистерская диссертация посвящена разработке системы автоматизации разработки программ на платформе Qt на основе автоматного программирования. Результаты исследования позволяют повысить эффективность процесса разработки программного обеспечения, а разработанная система представляет собой ценный инструмент для программистов, работающих на платформе Qt.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность работы. В современном информационном обществе программирование стало неотъемлемой частью многих отраслей. Однако процесс разработки программного обеспечения по-прежнему остается сложным и трудоемким. Одной из основных проблем разработки является необходимость повторного создания структурно схожих программ, что отнимает значительное время и ресурсы. В связи с этим, возникает потребность в автоматизации разработки программного обеспечения.

Существует множество фреймворков и инструментов, позволяющих упростить процесс разработки и повысить эффективность работы программистов. Одним из таких инструментов является платформа Qt, которая предлагает широкий спектр возможностей для разработки кроссплатформенных приложений. Однако даже с Qt разработка программного обеспечения по-прежнему требует большого объема ручной работы.

Поэтому, с точки зрения разработки программного обеспечения, тема «Система автоматизации разработки программ на платформе Qt на основе автоматного программирования» остается актуальной и интересной. Вот некоторые аспекты, подтверждающие ее актуальность:

1.Повышение производительности и эффективности: автоматизация разработки программного обеспечения может значительно ускорить процесс разработки программного обеспечения и повысить производительность труда разработчиков. Автоматизация рутинных задач и генерация кода на основе заранее заданных шаблонов и правил позволяет сэкономить время и уменьшить количество ошибок.

2. Сокращение рутинных задач: автоматное программирования позволяет сократить количество рутинных задач, таких как создание повторяющегося кода или настройка стандартных компонентов пользовательского интерфейса. Это освобождает разработчиков от монотонных задач и позволяет им сосредоточиться на более творческих и сложных аспектах разработки.

3.Снижение вероятности ошибок: использование автоматизированной системы, основанной на автоматном программировании, снижает вероятность ошибок, поскольку процесс генерации кода стандартизирован и опирается на определенные правила. Это повышает качество программного обеспечения и сокращает время, затрачиваемое на отладку и исправление ошибок.

4.Улучшенная масштабируемость и переносимость: автоматное программирование помогает улучшить масштабируемость проектов и переносимость кода. Используя

шаблоны и генерируя правила, программы можно легко адаптировать к различным платформам и архитектурам.

5.Сотрудничество и согласованность: Система автоматизации разработки Qt, основанная на автоматизированном программировании, обеспечивает согласованность и стандартизацию в команде разработчиков. Все участники проекта работают по определенным правилам и шаблонам, что способствует сотрудничеству и обеспечивает согласованность кодовой базы.

Однако следует отметить, что автоматное программирование может подходить не для всех типов проектов и может потребовать дополнительных усилий для внедрения. Разработчикам следует тщательно изучить свои требования и проектную среду, прежде чем принимать решение о внедрении автоматизированной системы на основе автоматного программирования для разработки программ на платформе Qt.

Целью данной магистерской работы является создание системы генерации шаблонов программ, основанной на автоматном программировании, для автоматизации процесса разработки программного обеспечения на платформе Qt.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучение существующих подходов к автоматизации разработки программного обеспечения и особенностей платформы Qt.
2. Исследование методов и инструментов автоматного программирования, а также их применимости для разработки на платформе Qt.
3. Разработка системы генерации шаблонов программ на платформе Qt, основанной на принципах автоматного программирования.
4. Проведение экспериментов и анализ полученных результатов для оценки эффективности разработанной системы и ее сравнение с традиционными методами разработки программного обеспечения на платформе Qt.

Методы исследования.

В данной работе будут использованы такие методы как, анализ научной литературы и источников, анализ существующих подходов и инструментов автоматизации разработки на платформе Qt, изучение принципов автоматного программирования, проектирование и разработка системы генерации шаблонов программ на основе автоматных композиций, а также проведение экспериментов для оценки эффективности и применимости разработанной системы.

Практическая ценность.

Защищаемый программный комплекс позволяет автоматизировать и упростить процесс разработки программ на платформе QT через использование методов автоматного

программирования. Предложенный комплекс будет полезен для разработчиков, ускоряя их работу и позволяя сосредоточиться на решении более сложных задач. Кроме того, результаты работы могут быть применены в различных сферах, где используется платформа Qt, включая разработку приложений для настольных компьютеров, мобильных устройств, встроенных систем и других областей применения.

Достоверность результатов.

Достоверность результатов исследования обеспечена проведением эксперимента и анализом полученных результатов.

Основное положения, выносимое на защиту.

Защищается комплекс программ для автоматизации процесса разработки ПО на платформе Qt, отличающийся от аналогов, тем что разрабатываемая система представляется композицией взаимодействующих автоматов, позволяющий автоматизировать процесс разработки ПО на основе формальной модели (автоматные композиции), а также автоматизировать процесс генерации тестов, в том числе тестов на осцилляции и тупики.

Структура и объем работы.

Магистерская диссертация разделена на 3 глав.

В **первой главе** вводятся основные понятия и определения, которые используются в работе. В частности, вводятся понятия автомата, композиции автоматов и ее типы, а также определение автоматного программирования и его принципов работы.

Во **второй главе** магистерской диссертации рассматривается платформа Qt, которая используется для разработки программного обеспечения. В этой главе изучаются основные компоненты, предоставляемые платформой Qt, а также рассматриваются её возможности для разработки программ.

В **третьей главе** магистерской диссертации разрабатывается система генерации шаблонов программ на платформе Qt на основе автоматного описания. Описывается архитектура системы, язык автоматного описания программ, модуль генерации шаблонов и разработка графического интерфейса пользователя.

1 Основные понятия и определения

1.1 Автомат

Цель этой главы — описать автомат как математическую модель для представления системы, способной выдавать соответствующий выходной отклик на заданный вход. Автоматы широко используются в различных областях, в том числе информационных технологий, автоматического управления и др. Основными элементами автомата являются входы, выходы, состояния и переходы [2].

Входным сигналом является информация, введенная в автомат. Каждый входной сигнал соответствует символу в множестве I. Выходной сигнал — это информация, производимая машиной в ответ на входной сигнал. Каждый выходной сигнал соответствует некоторым символам в наборе выходных символов.

Состояние представляет собой внутреннее состояние автомата [4]. При получении входного сигнала автомат меняет свое состояние. В каждый момент автомат находится в некотором состоянии множества S. Состояние автомата изменяется при переходе из одного состояния в другое.

Переход — это изменение состояния автомата после получения определенного входного сигнала. При переходе автомат меняет свое состояние из одного состояния в другое [3].

Конечный автомат или просто автомат – это система, которая получает входной сигнал и выдает выходной сигнал по определенным правилам [1]. Формально автомат можно определить как пять $A = (S, I, O, \delta, S_0)$, где:

S - множество состояний автомата,

I — набор входных символов автомата,

O — набор выходных символов автомата,

δ — коэффициент перехода автомата,

S_0 — начальное состояние, $S_0 \in S$.

Пример классического автомата приведен на Рисунок 1 –

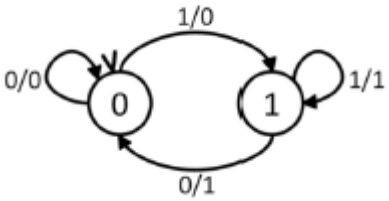


Рисунок 1 – Классический автомат

Отношения перехода автоматов можно распространить на входные и выходные последовательности [6]. Таким образом, для входной последовательности $\alpha = i_1...i_k \in I^*$ и выходной последовательности $\beta = o_1...o_k \in O^*$, если существует последовательность $(s, \alpha, \beta, s') \in \delta$ для состояния $s, s_1, ..., s_{k-1}, s'$ такие, что $(s, i_1, o_1, s_1) \in \delta, ..., (s_{k-1}, i_k, o_k, s') \in \delta$. Здесь выходная функция $out(s, \alpha)$ отображает $S \times I^*$ в набор подмножеств выходной последовательности. Для $\alpha \in I^*$ и $s \in S$, если существует s' такое, что $(s, \alpha, \beta, s') \in \delta$, то выходная последовательность β принадлежит $out(s, \alpha)$. В этом случае пара (α, β) называется входной-выходной последовательностью автомата в состоянии s.

Язык $L \times A(s)$ автомата A в состоянии s есть множество входных и выходных последовательностей автомата в этом состоянии. То есть $L \times A(s) \subset (I \times O)^*$ и $(i_1, o_1)...(i_k, o_k) \in L \times A(s)$ тогда и только тогда, когда $o_1...o_k \in out(s, i_1...i_k)$. Язык начального состояния автомата A называется языком автомата A и обозначается как $L \times A$.

1.2 Автоматные композиции

Автоматные композиции представляют собой процесс объединения нескольких автоматов в единый автомат. Существуют различные виды композиций автоматов, каждый из которых обладает своими особенностями и применяется в разных ситуациях.

Одним из видов композиций является последовательная композиция. При последовательной композиции выходное состояние одного автомата становится входным для следующего автомата. Этот вид композиции широко применяется в ситуациях, когда требуется последовательное выполнение нескольких действий или процессов.

При последовательной композиции автоматы связываются между собой таким образом, что выходные сигналы или состояния первого автомата передаются входным сигналам или состояниям второго автомата, и так далее. Это означает, что результат работы первого автомата определяет начальное состояние или входные данные для второго автомата, и так далее по цепочке.

Пример последовательной композиции можно представить с помощью автоматов, которые моделируют выполнение некоторого процесса. Пусть у нас есть автомат A1, представляющий выполнение действия A1, и автомат A2 представляющий выполнение действия A2. При последовательной композиции автоматов A1 и A2 выходное состояние автомата A1 становится входным состоянием автомата A2. Это означает, что после завершения действия A1 автомат переходит к выполнению действия A2.

Пример структуры последовательной композиции [5] автоматов представлен на Рисунок 2 –

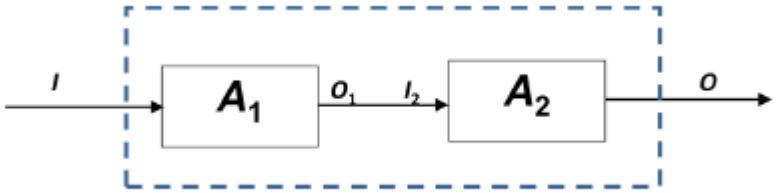


Рисунок 2 – Структура последовательной композиции

Где **[A1]** и **[A2]** обозначают состояния автоматов A1 и A2 соответственно, а стрелка указывает на переход от состояния A1 к состоянию A2.

Важно отметить, что каждый автомат в последовательной композиции может иметь свои собственные входы, выходы и состояния, и взаимодействие между ними осуществляется передачей данных или сигналов между состояниями или переходами. Также возможно использование условий или ограничений для определения переходов или состояний в автоматах.

Последовательная композиция позволяет структурировать и управлять последовательными процессами, где каждый автомат выполняет определенное действие или задачу. Это особенно полезно при моделировании и управлении системами, которые требуют выполнения нескольких этапов или шагов в определенной последовательности.

Преимуществом последовательной композиции является ее простота и интуитивность. Она позволяет разбить сложную задачу на более простые подзадачи, что облегчает понимание и управление процессом. Кроме того, последовательная композиция обеспечивает линейный поток выполнения и упорядоченность действий, что может быть важно во многих сценариях и приложениях.

Однако следует помнить, что последовательная композиция имеет ограничения в том смысле, что выполнение каждого автомата зависит от завершения предыдущего автомата. Это может оказаться неэффективным в случае, когда одно действие требует много времени или ресурсов, и оно блокирует выполнение следующего действия. В таких случаях может быть полезно рассмотреть другие виды автоматных композиций, например, параллельную композицию, чтобы обеспечить более эффективное распараллеливание и выполнение задач.

Другим видом композиции является параллельная композиция, где автоматы работают параллельно, и выходные состояния каждого автомата объединяются в единое выходное состояние композиции. Параллельная композиция используется, когда необходимо выполнить несколько действий одновременно.

При параллельной композиции описывается совместная работа автоматов, функционирующих в режиме диалога [7]. Общая структура параллельной композиции автоматов показана на Рисунок 3 –

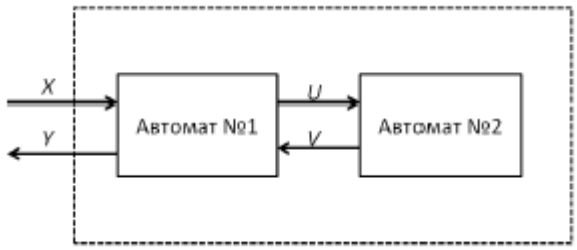


Рисунок 3 – Общая структура параллельной композиции автоматов [6]

В параллельной композиции автоматы работают независимо друг от друга, выполняя свои действия параллельно. Каждый автомат имеет свои входы и выходы, и они обмениваются сообщениями для совместной координации своих действий. Коммуникация между автоматами может осуществляться через разделяемые каналы связи или с использованием событийной модели, где автоматы реагируют на определенные события в системе. Параллельная композиция широко применяется для моделирования и управления сложными системами, такими как телекоммуникационные сети, распределенные вычислительные системы, многопоточные программы и другие.

Преимуществом параллельной композиции является возможность разделения задачи на более мелкие подзадачи, которые могут быть решены параллельно. Это позволяет улучшить производительность системы, сократить время выполнения и повысить отзывчивость. Кроме того, параллельная композиция обеспечивает легкое масштабирование системы путем добавления или удаления автоматов по мере необходимости.

Однако при использовании параллельной композиции необходимо учитывать возможные проблемы, такие как гонки данных (data races) и состояний (state races), возникающие при одновременном доступе к общим ресурсам. Такие проблемы требуют внимательного управления синхронизацией и координацией действий автоматов, чтобы избежать конфликтов и некорректного поведения системы.

Одним из применений автоматных композиций является создание систем управления, основанных на конечных автоматах (FSA, finite-state automaton). Конечные автоматы представляют собой модель поведения, которая подходит для систем с ограниченным числом состояний. В FSA каждое состояние представлено узлом, а переходы между состояниями обозначаются стрелками. Состояния, входы и выходы определяются заранее и задаются в таблице переходов. Автоматы могут работать в

режиме дискретного времени, тогда каждый такт времени соответствует одному состоянию, или в режиме непрерывного времени, тогда состояния меняются непрерывно в зависимости от входных сигналов.

Для создания систем управления на основе FSA используются автоматные композиции, которые позволяют создавать сложные модели систем с помощью простых автоматических компонент. В результате получается эффективная система, которая может значительно упростить управление и контроль над различными процессами и устройствами.

Таким образом, автоматные композиции — это мощный инструмент для моделирования и управления различными системами с помощью автоматов. Они позволяют создавать сложные модели систем с помощью простых компонент и взаимодействовать между собой для решения более сложных задач. Применение автоматных композиций позволяет упростить управление и контроль над различными процессами и устройствами, а также улучшить производительность и эффективность системы в целом.

1.3 Автоматное программирование

Автоматное программирование – это метод разработки программного обеспечения с использованием автоматных моделей, таких как конечные автоматы. Этот метод программирования основан на принципе декомпозиции больших и сложных систем на более простые компоненты, которые могут быть реализованы в виде конечных автоматов и для которых можно использовать автоматные композиции [8].

Преимущество автоматного программирования заключается в том, что оно упрощает процесс создания и поддержки программного обеспечения. Этот метод программирования позволяет разработчикам сосредоточиться на проектировании отдельных компонент программы вместо того, чтобы заботиться о всей системе целиком. Кроме того, автоматные модели позволяют четко определить границы задач, что упрощает отладку и тестирование программы.

Одна из основных задач, которую можно решить с помощью автоматного программирования, это реализация логики управления в программном обеспечении. Конечные автоматы могут быть использованы для описания поведения программы в зависимости от состояний и событий. Например, если мы разрабатываем систему управления состоянием двери, мы можем создать конечный автомат, который будет описывать поведение двери в зависимости от того, открыта она или закрыта, и какие события происходят, такие как нажатие кнопки или детектирование препятствия.

Помимо этого, автоматные модели также могут использоваться для описания бизнес-процессов и протоколов обмена данными. Например, при разработке приложения для онлайн-магазина мы можем использовать конечные автоматы для описания процессов оформления заказа или проведения платежа.

Одним из наиболее распространенных языков для автоматного программирования является UML State Machines. Этот язык позволяет описать конечный автомат как компонент системы и определить его свойства, состояния, входы и выходы. Кроме того, UML State Machines позволяет создавать автоматные композиции, чтобы объединять несколько автоматов в одной модели.

Таким образом, автоматное программирование является эффективным и удобным методом разработки программного обеспечения, особенно для систем с логикой управления и бизнес-процессов. Этот метод программирования позволяет разработчикам упростить процесс создания и поддержки программного обеспечения и улучшить его производительность и эффективность.

Кроме UML State Machines, существует множество других языков и инструментов для автоматного программирования. Например, язык Promela используется для описания параллельных процессов, а генераторы кода, такие как Stateflow и YAKINDU Statechart Tools, позволяют автоматически генерировать код на основе автоматных моделей.

Важным аспектом автоматного программирования является верификация программного обеспечения с использованием автоматных моделей. Верификация позволяет определить, соответствует ли программа заданным требованиям и спецификациям. Существуют различные инструменты для верификации автоматных моделей, такие как Spin и NuSMV.

Но не все задачи могут быть решены с помощью автоматного программирования. Некоторые задачи требуют использования других методов программирования, таких как объектно-ориентированное программирование или функциональное программирование. Кроме того, автоматное программирование имеет свои ограничения в том, что оно применим только к системам с ограниченным числом состояний и не подходит для более сложных систем.

Таким образом, автоматное программирование – это эффективный метод разработки программного обеспечения, особенно для систем с логикой управления и бизнес-процессов. Оно упрощает процесс создания и поддержки программного обеспечения и улучшает его производительность и эффективность, а также позволяет верифицировать программное обеспечение перед его внедрением в реальных условиях эксплуатации.

2 Платформа Qt для разработки программ

Платформа Qt является мощным и многофункциональным фреймворком для разработки программного обеспечения. Она предоставляет разработчикам гибкое и эффективное окружение, позволяющее создавать кроссплатформенные приложения с привлекательными графическими интерфейсами и богатым функциональным набором [9]. Qt включает в себя различные инструменты, такие как Qt Creator, Qt Designer, Qt Linguist и многие другие, которые значительно упрощают разработку и автоматизируют рутинные задачи.

Один из ключевых инструментов, входящих в состав Qt, это Qt Creator - интегрированная среда разработки (IDE), предоставляющая широкий спектр возможностей для создания, отладки и развертывания приложений на платформе Qt. Qt Creator обладает множеством полезных функций, включая автодополнение кода, отладчик, инструменты анализа кода и другие, которые значительно ускоряют процесс разработки. На Рисунок 4 – приведен пример визуального интерфейса Qt Creator.

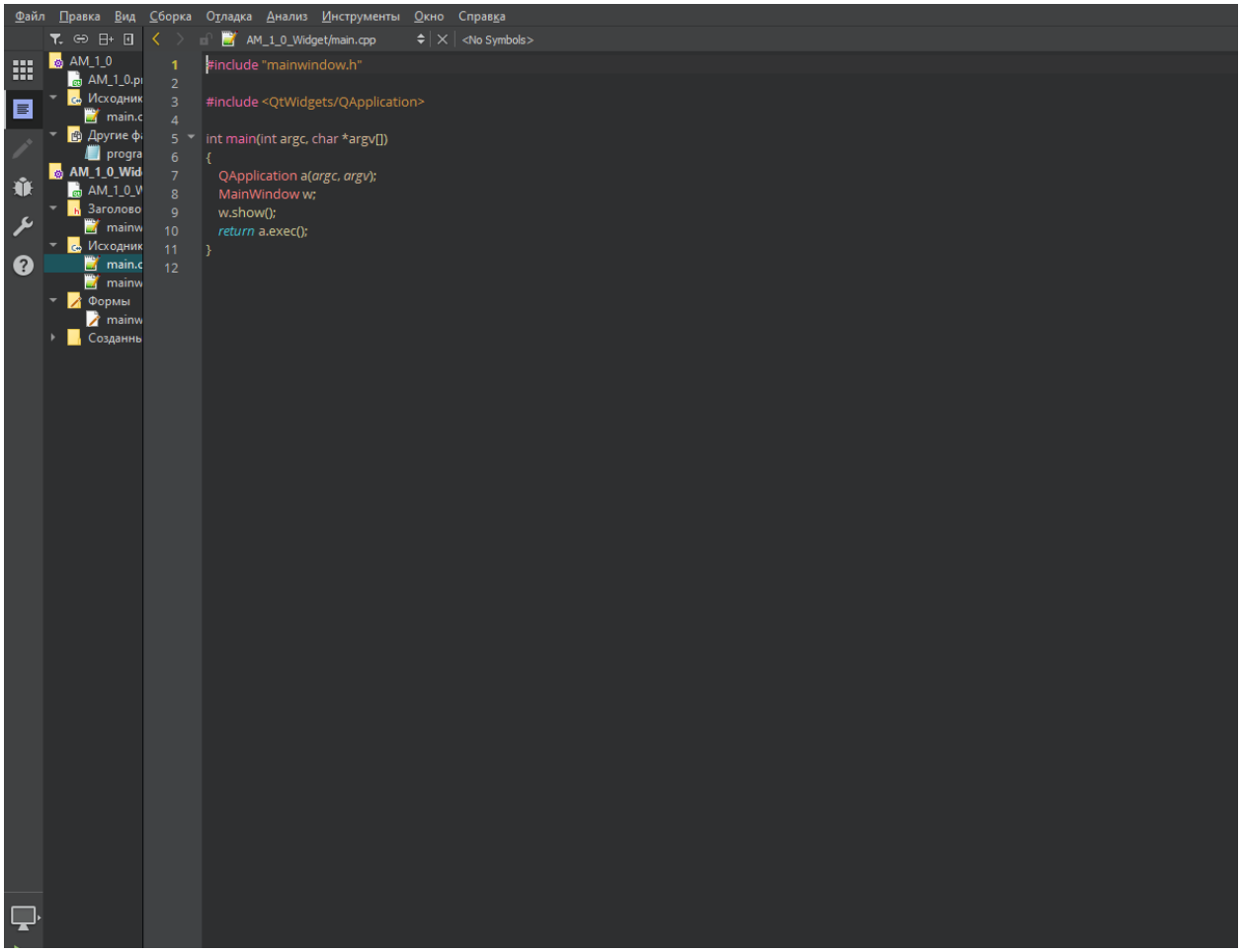


Рисунок 4 – Пример визуального вида Qt Creator

Еще одним важным инструментом является Qt Designer, который представляет собой мощный инструмент для визуального проектирования пользовательского интерфейса. С помощью Qt Designer разработчики могут легко создавать интерфейс приложения, перетаскивая и настраивая готовые элементы управления. Qt Designer позволяет генерировать исходный код для созданного интерфейса, что упрощает и автоматизирует процесс создания графического пользовательского интерфейса. Пример визуального интерфейса Qt Designer показан на Рисунок 5 – .

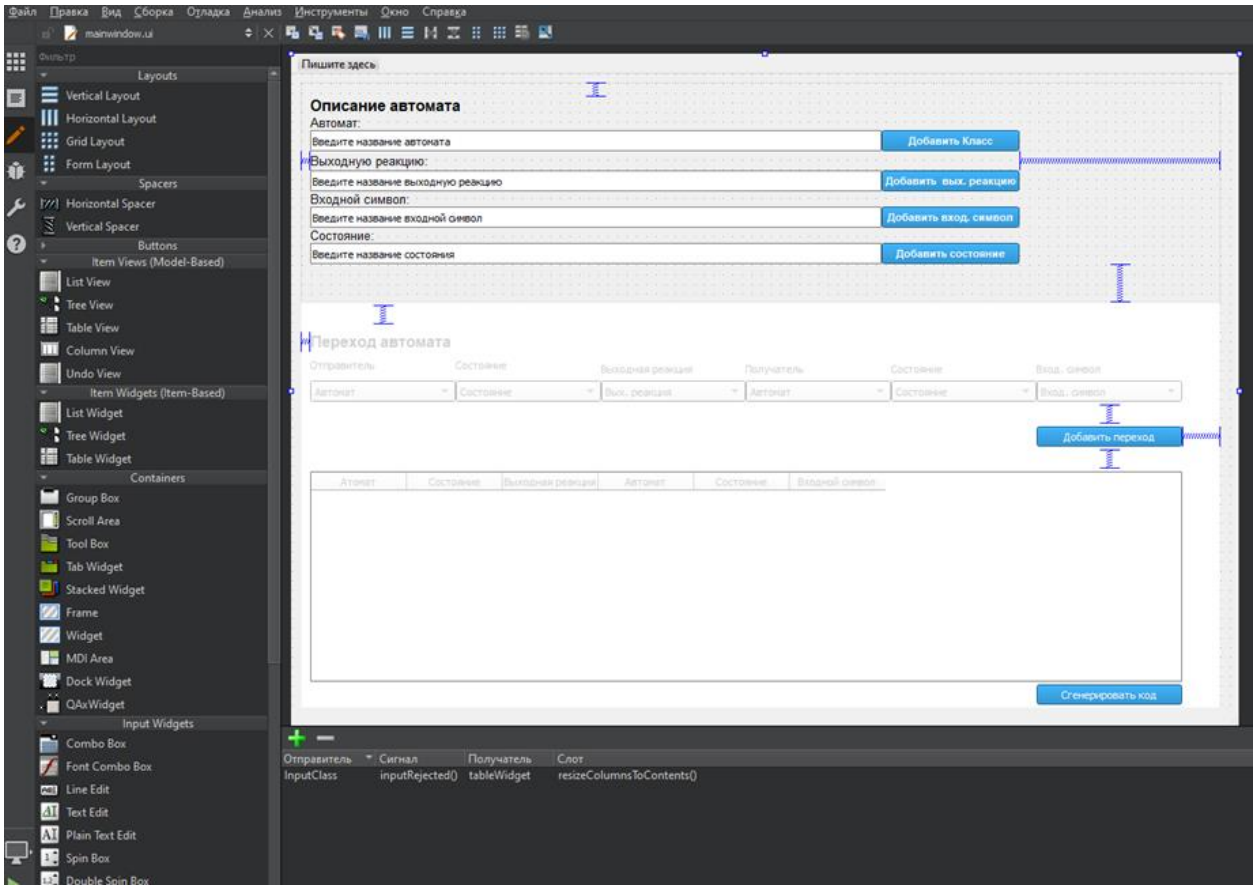


Рисунок 5 – Пример визуального вида Qt Creator

2.1 Основные компоненты платформы Qt

Платформа Qt включает в себя несколько ключевых компонентов, которые обеспечивают разработчикам широкий набор функциональных возможностей.

Qt Core является основным модулем платформы Qt и предоставляет базовые классы и функциональность для разработки программ. Он включает в себя классы для работы с событиями, потоками, контейнерами, файлами, сетью и другими важными аспектами разработки. Qt Core предоставляет множество удобных инструментов для обработки данных и управления потоками выполнения.

Модуль Qt GUI предоставляет разработчикам инструменты для создания графического интерфейса пользователя. Он включает в себя готовые элементы

управления, такие как кнопки, поля ввода, таблицы, списки, а также возможности для настройки внешнего вида и стилей. Qt GUI обеспечивает поддержку множества функций, связанных с отображением и взаимодействием с пользователем.

Модуль Qt Widgets предоставляет набор классов для создания классических настольных приложений с использованием виджетов. Он предлагает широкий выбор готовых элементов управления и возможности для их настройки и расширения. Qt Widgets является одним из основных модулей для создания графического интерфейса на платформе Qt.

Qt QML представляет собой декларативный язык и модуль платформы Qt для создания графического интерфейса на основе QML (Qt Meta-Object Language). QML позволяет разработчикам создавать динамические и интерактивные пользовательские интерфейсы с использованием декларативного подхода. Он обеспечивает простоту разработки и удобство создания пользовательского опыта.

2.2 Возможности Qt для разработки программ

Платформа Qt обладает широким набором возможностей, которые делают ее одним из наиболее популярных инструментов для разработки программного обеспечения. Вот лишь некоторые из ключевых возможностей Qt:

1. Кроссплатформенность: Qt позволяет разрабатывать приложения, которые могут работать на различных операционных системах, таких как Windows, macOS, Linux, Android и iOS. Это обеспечивает гибкость и эффективность разработки, поскольку разработчики могут создавать приложения для нескольких платформ с использованием общего кода.

2. Графический интерфейс пользователя: Qt предоставляет обширный набор инструментов для создания современных и привлекательных графических интерфейсов пользователя. Он включает в себя готовые элементы управления (кнопки, поля ввода, таблицы и другие), а также возможности для настройки внешнего вида и стилей.

3. Мультимедиа: Qt предлагает богатые возможности для работы с мультимедийным контентом. Разработчики могут легко создавать и воспроизводить аудио и видео, а также работать с изображениями. Qt также обеспечивает поддержку графического ускорения и 3D-графики.

4. Сетевое взаимодействие: Qt предлагает широкий спектр возможностей для работы с сетевым взаимодействием. Разработчики могут создавать клиент-серверные приложения, обмениваться данными по сети, работать с протоколами HTTP, FTP, SMTP и многими другими.

5. Многопоточность: Qt предоставляет удобные средства для работы с многопоточностью. Разработчики могут легко создавать приложения, которые выполняют параллельные операции и эффективно использовать многопроцессорные системы.

6. Интеграция с платформенными функциями: Qt обеспечивает простую интеграцию с платформенными функциями операционных систем. Разработчики могут использовать функциональность, такую как уведомления, работа с файлами и доступ к аппаратному обеспечению, без необходимости писать специфичный код для каждой платформы.

Это лишь небольшой обзор возможностей Qt. Фреймворк имеет много других функций, которые помогают разработчикам создавать высококачественное программное обеспечение, сохраняя при этом эффективность и переносимость.

3 Разработка системы генерации шаблонов программ на платформе Qt на основе автоматного описания

3.1 Описание архитектуры системы

Система генерации шаблонов программ на платформе Qt на основе автоматного программирования состоит из трех основных модулей: автоматного описания программы, генератора кода и пользовательского интерфейса.

Модуль автоматного описания программы представляет собой набор автоматов, которые описывают различные аспекты программы. Каждый автомат отвечает за отдельную функцию программы, такую как обработка ввода пользователя или генерация выходных данных. Автоматы могут быть как детерминированными, так и недетерминированными в зависимости от сложности описываемой функции.

Описание каждого автомата включает в себя следующие элементы:

- Набор состояний автомата и начальное состояние.
- Набор входных сигналов, на которые автомат может реагировать.
- Функции переходов, определяющие, какой переход будет выполнен в ответ на конкретный входной сигнал, исходя из текущего состояния автомата.
- Набор выходных реакций, определяющих, какие данные будут сгенерированы при переходе из текущего состояния в новое состояние.

Модуль генератора кода отвечает за генерацию исходного кода программы на основе автоматного описания. Он обрабатывает автоматы, определяет последовательность выполнения функций программы и генерирует соответствующий код на языке программирования Qt. Генератор кода также отвечает за оптимизацию кода и проверку его на соответствие заданным требованиям.

Пользовательский интерфейс предоставляет возможность создания автоматного описания программы. Через интерфейс пользователь может определить состояния, входные сигналы, функции переходов и выходные реакции для каждого автомата.

Таким образом, система генерации шаблонов программ на платформе Qt на основе автоматного описания предоставляет средства для формального описания функциональности программы с использованием автоматов, а затем автоматически генерирует соответствующий исходный код на языке Qt.

3.2 Автоматное описание программ

Автоматное описание программы предоставляет формализованный подход к описанию функциональности программы с использованием конечных автоматов или конечных автоматов с выходами (FSM). Конечный автомат с выходами позволяет генерировать выходные данные на основе текущего состояния.

Для иллюстрации примера, была создана автоматная композиция, представленная на Рисунок 6 –

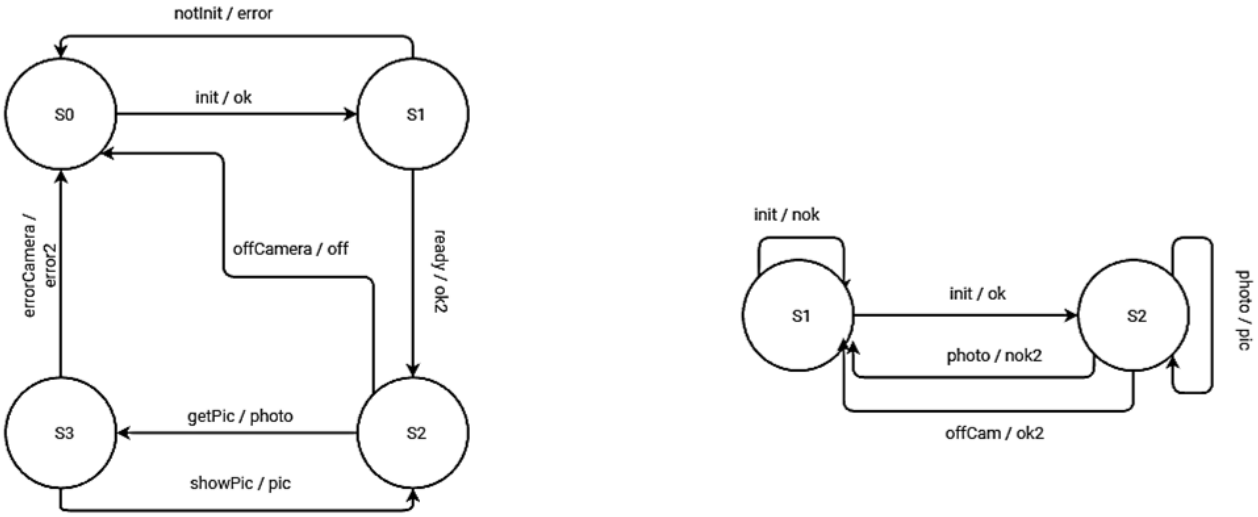


Рисунок 6 – пример автоматной композиции

Описание каждого автомата включает в себя следующие элементы:

1. Набор состояний автомата и начальное состояние:
   - Каждый автомат имеет определенный набор состояний, которые представляют различные состояния функции программы. Например, для автомата "Camera" описаны состояния "S0_Camera", "S1_Camera", "S2_Camera" и "S3_Camera".
   - У каждого автомата также есть определенное начальное состояние, с которого он начинает свою работу. Например, автомат "Camera" начинает с состояния "S0_Camera".
2. Набор входных воздействий, на которые автомат может реагировать:
   - Каждый автомат может реагировать на определенные входные сигналы или события. Эти входные сигналы могут быть инициированы внешними источниками или другими автоматами. Например, автомат "Camera" может реагировать на сигналы "ok", "error", "ok2", "photo" и "off".
3. Функции переходов:
   - Функции переходов определяют, какой переход будет выполнен в ответ на конкретный входной сигнал в зависимости от текущего состояния автомата.

- Каждая функция перехода определяет пару (входной сигнал, целевое состояние). Например, для автомата "Camera" определены следующие функции переходов: при сигнале "ok" в состоянии "S0_Camera" переход осуществляется в состояние "S1_Camera", при сигнале "error" в состоянии "S1_Camera" переход осуществляется в состояние "S0_Camera" и так далее.

4. Набор выходных реакций:
   - Выходные реакции определяют, какие данные будут сгенерированы при переходе из текущего состояния автомата в новое состояние.
   - Каждая выходная реакция определяет пару (слот, целевое состояние). Например, для автомата "Camera" определены выходные реакции: при переходе из состояния "S1_Camera" в состояние "S0_Camera" выполняется слот "notInit", а при переходе из состояния "S3_Camera" в состояние "S2_Camera" выполняется слот "showPic".

Для эффективного хранения и обработки данных об автоматах необходимо иметь удобный формат описания и чтения композиций. В этом контексте было принято решение использовать формат XML, который обладает широкими возможностями для структурирования данных и является универсальным для программирования.

Одним из обсуждаемых принципов использования XML является его способность формализовать процесс описания и использования автоматов. В результате разработано XML-представление [11], предназначенное для описания структуры автоматных композиций.

Файл с автоматной композицией представлен в формате .xml и содержит информацию о композиции в четко определенной структуре. Благодаря этому формату процесс работы с автоматами становится более систематизированным и удобным для программирования.

Все это позволяет сохранять и обрабатывать данные об автоматах более эффективным и профессиональным способом.

Автоматная композиция записывается в файл в формате .xml следующим образом:

```
<program>
    <class name="Camera">
        <state name="S0_Camera">
            <transition signal="ok" slot="init" targetState="S1_Camera"/>
        </state>
        <state name="S1_Camera">
            <transition signal="error" slot="notInit" targetState="S0_Camera"/>
            <transition signal="ok2" slot="ready" targetState="S2_Camera"/>
        </state>
        <state name="S2_Camera">
            <transition signal="photo" slot="getPic" targetState="S3_Camera"/>
            <transition signal="off" slot="offCamera" targetState="S0_Camera"/>
        </state>
        <state name="S3_Camera">
```

```
            <transition signal="error2" slot="errorCam"
targetState="S0_Camera"/>
            <transition signal="pic" slot="showPic" targetState="S2_Camera"/>
        </state>
    </class>
    <class name="Photo">
        <state name="S1_Photo">
            <transition signal="nok" slot="init" targetState="S1_Photo"/>
            <transition signal="ok" slot="init" targetState="S2_Photo"/>
        </state>
        <state name="S2_Photo">
            <transition signal="pic" slot="photo" targetState="S2_Photo"/>
            <transition signal="ok2" slot="offCam" targetState="S1_Photo"/>
            <transition signal="nok" slot="photo" targetState="S1_Photo"/>
        </state>
    </class>
</program>
```

Где: *«class»* - является автоматом описываемой автоматной композиции, *«signal»* — выходная реакция, *«slot»* — входное воздействие, *«state»* — текущее состояние автомата, *«targetState»* — следующее состояние автомата

Таким образом, XML-представление автоматной композиции предоставляет структурированный и удобный способ описания и использования автоматов в программировании.

3.3 Модуль генерации шаблонов программ на платформе Qt

Модуль генерации шаблонов программ на платформе Qt ответственен за автоматическое создание части кода на основе заранее определенных шаблонов и правил. Генерация кода на основе шаблонов позволяет ускорить процесс разработки программного обеспечения, обеспечивая согласованность и стандартизацию в кодовой базе. В данной главе рассматривается использование генерации кода на основе шаблонов в контексте разработки на платформе Qt.

Модуль генерации шаблонов программ на платформе Qt выполняет генерацию исходного кода программы на основе автоматного описания. Он преобразует описание автоматных композиций в последовательность команд на языке программирования C++ для платформы Qt. Сгенерированный код может быть скомпилирован и запущен на целевой платформе.

В целом, модуль генерации шаблонов программ на платформе Qt является важным компонентом системы генерации кода, отвечая за преобразование автоматного описания программы в полноценный исходный код. Он позволяет получить структуру и связи между классами и состояниями из внешнего XML-файла, которые затем используются для генерации соответствующего программного кода.

Для этого модуля созданы различные функции и структуры. Например, программный код включает необходимые библиотеки для считывания файлов формата

XML, такие как fstream, QFile и QXmlStreamReader. Также в коде создаются структуры, такие как "State", "Class" и "Transition", которые представляют состояния, классы и переходы между состояниями.

Для разбора XML-файла и извлечения информации о классах и переходах создана функция **parseXML**. Ниже представлено описание этой функции:

Функция **parseXML** принимает путь к XML-файлу (*filePath*) в формате *QString* и два вектора: *classes* для хранения классов и *transitions* для хранения переходов. Она открывает XML-файл, создает объект *QXmlStreamReader* для чтения XML и последовательно обрабатывает каждый элемент внутри файла. В процессе обработки функция извлекает информацию о классах, состояниях и переходах, заполняет соответствующие структуры данных и добавляет их в соответствующие векторы. Если возникают ошибки при чтении XML, они выводятся в стандартный поток ошибок.

Эта функция является важной частью модуля генерации шаблонов программ на платформе Qt, так как позволяет получить структуру и связи между классами и состояниями из внешнего XML-файла, которые затем могут быть использованы для генерации соответствующего программного кода.

Для генерации файлов **.cpp** и **.h** автоматов созданы функции **generateHeaderFile** и **generateCppFile**. Ниже представлено описание каждой из функций:

```
void generateHeaderFile(const Class& cls, const std::vector<Transition>& transitions) {
    std::ofstream headerFile(cls.name + ".h");

    headerFile << "#ifndef " << cls.name << "_H\n"
               << "#define " << cls.name << "_H\n\n"
               << "#include <QObject>\n\n"
               << "enum " << cls.name << "States\n"
               << "{\n";

    for (const auto& state : cls.states) {
        headerFile << "    " << state.name << ",\n";
    }

    headerFile << "};\n\n"
               << "enum " << cls.name << "InputSignals\n"
               << "{\n";

    for (const auto& transition : transitions) {
        headerFile << "    " << "s_" << transition.slot << ",\n";
    }

    headerFile << "};\n\n"
               << "class " << cls.name << " : public QObject\n"
               << "{\n"
               << "    Q_OBJECT\n\n"
               << "public:\n"
               << "    explicit " << cls.name << "(QObject* parent = nullptr);\n\n";

    for (const auto& transition : transitions) {
        headerFile << "    void " << transition.slot << "();\n";
    }

    headerFile << "\nsignals:\n";

    for (const auto& transition : transitions) {
        headerFile << "    void " << transition.signal << "();\n";
    }
```

```
            headerFile << "\npublic slots:\n"
                       << "    void initState();\n";

            for (const auto& state : cls.states) {
                headerFile << "    void enter_" << state.name << "();\n"
                           << "    void exit_" << state.name << "();\n";
            }

            headerFile << "\nprivate:\n"
                       << "    " << cls.name << "States m_state;\n"
                       << "};\n\n"
                       << "#endif // " << cls.name << "_H\n";

            headerFile.close();
        }
```

Данная функция генерирует заголовочный файл на основе переданных данных о автомате и переходах. Файл создается с именем автомата, соответствующим имени класса, и имеет расширение .h. В заголовочном файле объявляются перечисления состояний и сигналов, а также класс, наследующийся от *QObject* и содержащий слоты и сигналы.

```
        void generateCppFile(const Class& cls, const std::vector<Transition>& transitions) {
            std::ofstream cppFile(cls.name + ".cpp");

            cppFile << "#include \"" << cls.name << ".h\"\n\n"
                    << cls.name << "::" << cls.name << "(QObject* parent)\n"
                    << "    : QObject(parent), m_state(" << cls.name << "States::" << cls.name <<
"_STATE)\n"
                    << "{\n"
                    << "    initState();\n"
                    << "}\n\n";

            cppFile << "void " << cls.name << "::initState()\n"
                    << "{\n";

            for (const auto& state : cls.states) {
                cppFile << "    if (m_state == " << cls.name << "States::S" << state.name << ")\n"
                        << "    {\n"
                        << "        enter_" << state.name << "();\n"
                        << "    }\n";
            }

            cppFile << "}\n\n";

            for (const auto& state : cls.states) {
                cppFile << "void " << cls.name << "::enter_" << state.name << "()\n"
                        << "{\n"
                        << "    // TODO: Add enter_" << state.name << " implementation\n"
                        << "}\n\n"
                        << "void " << cls.name << "::exit_" << state.name << "()\n"
                        << "{\n"
                        << "    // TODO: Add exit_" << state.name << " implementation\n"
                        << "}\n\n";
            }

            for (const auto& transition : transitions) {
                cppFile << "void " << cls.name << "::" << transition.slot << "()\n"
                        << "{\n"
                        << "    // TODO: Add " << transition.slot << " implementation\n"
                        << "    exit_" << transition.fromState << "();\n"
                        << "    // Handle transition\n"
                        << "     " << transition.toClass << "* " << transition.toClass << "_ptr =
new " << transition.toClass << "(this);\n"
                        << "      " << transition.toClass << "_ptr->" << transition.toState <<
"();\n"
                        << "    delete " << transition.toClass << "_ptr;\n"
                        << "}\n\n";
            }
```

```
        cppFile.close();
    }
```

Эта функция генерирует файл реализации на основе переданных данных о автомате и переходах. Файл создается с именем, соответствующим имени автомата, и имеет расширение *.cpp*. В файле реализуются функции-члены класса, включая конструктор и методы для входа и выхода из состояний, а также методы, обрабатывающие переходы между состояниями.

Обе функции используют переданные структуры и векторы для генерации кода. Вместо полного копирования кода в тексте, вы можете указать, что функции **generateHeaderFile** и **generateCppFile** выполняют генерацию соответствующих файлов на основе данных о классах и переходах, без предоставления конкретных деталей реализации.

Для генерации файлов **.cpp** и **.h** базового класса создаются функции **generateBaseClassCppFile** и **generateBaseClassHeaderFile**. Ниже представлено их описание:

```
    // Функция для генерации файла .cpp базового класса
    void generateBaseClassCppFile(const vector<Class>& classes, const vector<Transition>&
transitions) {
        ofstream cppFile("baseclass.cpp");

        cppFile << "#include \"baseclass.h\"\n\n";

        cppFile << "\nBaseClass::BaseClass(QObject* parent) : QObject(parent) {\n";

        for (const auto& cls : classes) {
            cppFile << "    m_" << cls.name << " = new " << cls.name << ";\n";
            cppFile << "    m_" << cls.name << "->moveToThread(&" << cls.name << "Thread);\n";
            cppFile << "\n";
        }

        if (!transitions.empty()) {
            cppFile << "    // Подключение сигналов и слотов\n";
            for (const auto& transition : transitions) {
                cppFile << "    connect(m_" << transition.from << ", &" << transition.signal
                        << ", m_" << transition.to << ", &" << transition.to << "::" <<
transition.slot << ");\n";
            }
            cppFile << "\n";
        }

        // Отправка сигнала для выполнения соединений сигналов и слотов
        cppFile << "    emit doWork();\n";

        for (const auto& cls : classes) {
            cppFile << "    m_" << cls.name << "Thread.start();\n";
        }

        cppFile << "}\n\n";
        cppFile << "BaseClass::~BaseClass() {\n";

        for (const auto& cls : classes) {
            cppFile << "    delete m_" << cls.name << ";\n";
            cppFile << "    " << cls.name << "Thread.quit();\n";
            cppFile << "    " << cls.name << "Thread.wait();\n";
        }

        cppFile << "}\n\n";
        cppFile.close();
    }
```

Функция generateBaseClassCppFile генерирует файл baseclass.cpp, который включает заголовочный файл "baseclass.h". Затем функция создает экземпляры классов, указанных в векторе classes, и перемещает их в соответствующие потоки QThread. Если вектор transitions не пуст, функция подключает сигналы и слоты для переходов между классами. Затем функция отправляет сигнал doWork() для выполнения соединений сигналов и слотов, и запускает потоки для каждого класса. В конце функция освобождает ресурсы, удаляя экземпляры классов и завершает потоки.

```
// Функция для генерации файла .h базового класса
void generateBaseClassHeaderFile(const vector<Class>& classes) {
    ofstream headerFile("baseclass.h");

    headerFile << "#ifndef BASECLASS_H\n";
    headerFile << "#define BASECLASS_H\n\n";

    headerFile << "#include <QObject>\n";
    headerFile << "#include <QThread>\n\n";

    for (const auto& cls : classes) {
        headerFile << "#include \"" << cls.name << ".h\"\n";
    }

    headerFile << "\n";

    headerFile << "class BaseClass : public QObject {\n";
    headerFile << "    Q_OBJECT\n\n";

    headerFile << "public:\n";
    headerFile << "    explicit BaseClass(QObject* parent = nullptr);\n";
    headerFile << "    ~BaseClass();\n\n";

    headerFile << "private:\n";
    for (const auto& cls : classes) {
        headerFile << "    QThread " << cls.name << "Thread;\n";
        headerFile << "    " << cls.name << "* m_" << cls.name << ";\n";
    }

    headerFile << "\n";

    headerFile << "signals:\n";
    headerFile << "    void doWork();\n\n";

    headerFile << "};\n\n";

    headerFile << "#endif // BASECLASS_H\n";

    headerFile.close();
}
```

Функция **generateBaseClassHeaderFile** генерирует файл *baseclass.h*, который содержит объявление класса *BaseClass*, производного от *QObject*. Заголовочный файл включает необходимые заголовочные файлы и объявляет экземпляры классов и потоки, указанные в векторе *classes*. Класс *BaseClass* имеет публичные и приватные члены, включая конструктор, деструктор и сигнал *doWork()*.

Класс **BaseClass** в данном контексте представляет базовый класс, который служит основой для создания и управления другими классами на платформе Qt. Он играет роль центрального координатора и обеспечивает инициализацию, управление потоками выполнения и соединение сигналов и слотов между различными классами.

Таким образом, модуль генерации шаблонов программ на платформе Qt предоставляет мощный инструмент для автоматического создания кода на основе шаблонов и правил. Он ускоряет процесс разработки, обеспечивает согласованность и стандартизацию в кодовой базе, а также предоставляет удобные функции для работы с XML-файлами, генерации заголовочных и реализационных файлов, а также управления классами и состояниями в приложениях, разработанных на платформе Qt.

3.4 Разработка графического интерфейса пользователя

Пользовательский интерфейс представляет собой визуальное представление программы или приложения, которое позволяет пользователю взаимодействовать с программой. В данном случае, код представляет собой описание пользовательского интерфейса, созданного с использованием библиотеки Qt.

Этот интерфейс содержит различные элементы, такие как кнопки (QPushButton), метки (QLabel), текстовые поля (QLineEdit), выпадающие списки (QComboBox) и таблицу (QTableWidget). Он организован с помощью различных компоновщиков (layout), таких как QGridLayout, чтобы упорядочить и расположить элементы интерфейса.

Взаимодействуя с этим пользовательским интерфейсом, пользователь может описывать автоматные композиции и добавлять автоматы, входные воздействия, выходные реакции и состояния, выбирать их из выпадающих списков, а также сохранять результаты. Интерфейс позволяет пользователям создавать и управлять элементами, связанными с программной архитектурой и событийной моделью программы.

Таким образом, пользовательский интерфейс, представленный на Рисунок 7 – , предоставляет удобные инструменты для создания и управления автоматными композициями, а модуль генерации шаблонов позволяет преобразовать эти композиции в программный код, который можно использовать для реализации конкретных программных проектов.

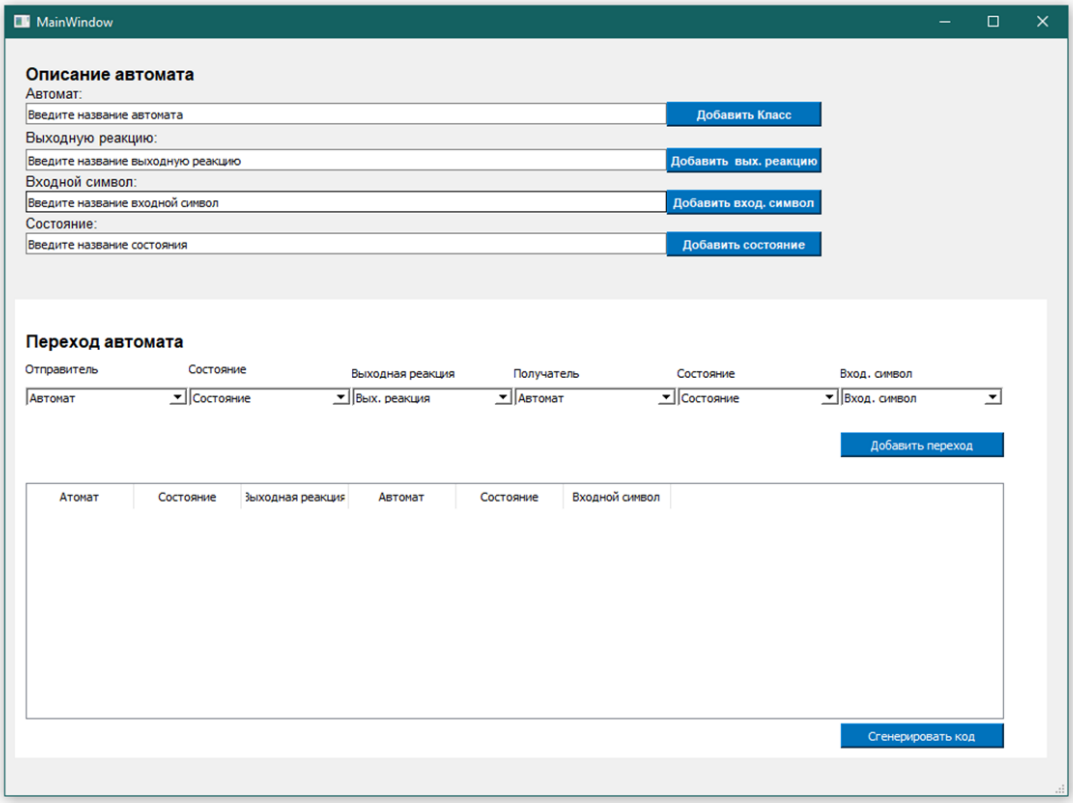


Рисунок 7 – пользовательский интерфейс

Интерфейс разделен на 3 блока:

1. Создание компонентов автоматов
2. Создание связи/переходов автоматов, между состояниями
3. Отображения списка переходов

На Рисунок 8 – отображена область создания компонентов автоматов:

- название автомата
- название состояний автомата
- название выходных реакций
- название входных воздействий

Все названия вводятся в соответствующее поле ввода и в соответствии со компонентом по нажатию кнопки сохраняется.

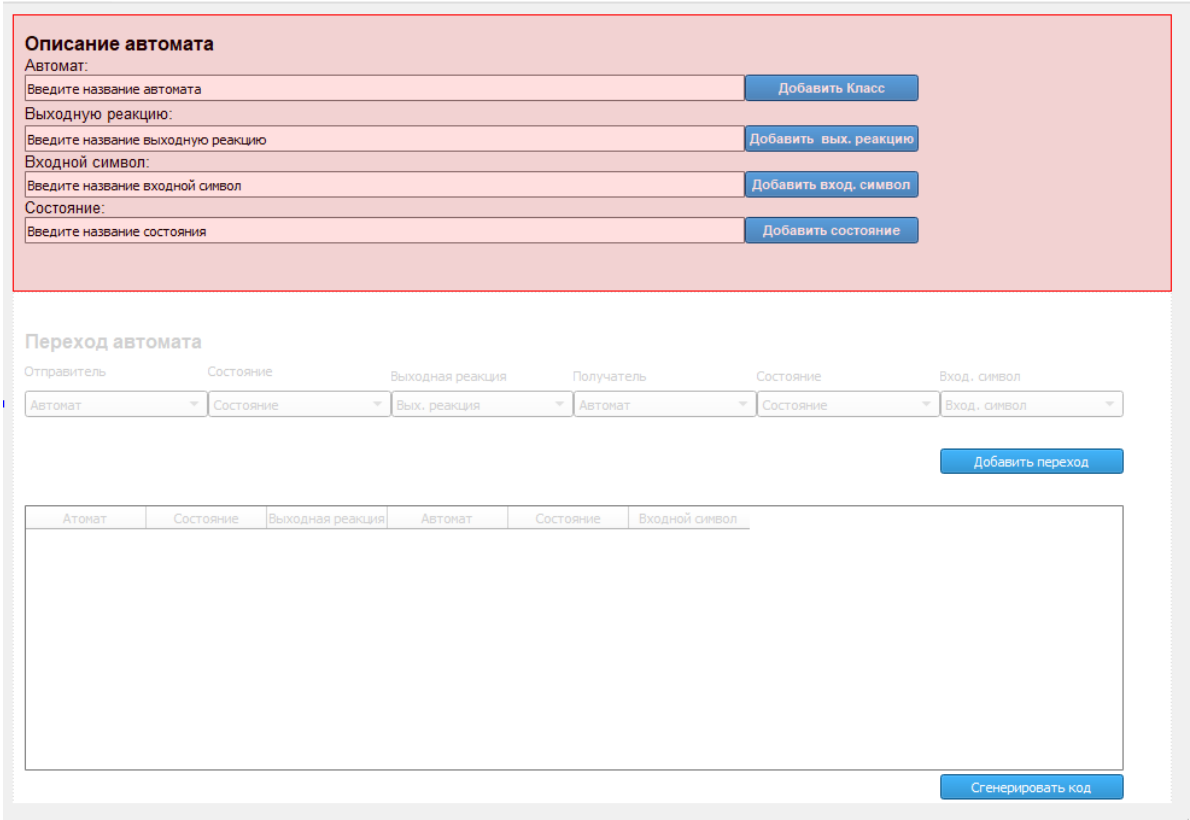


Рисунок 8 – область добавления элементов автоматной композиции

На Рисунок 9 – приведен пример отображения списка созданных названий автоматов в QComboBox. В этом списке можно выбрать название автомата, который описывает пользователь.

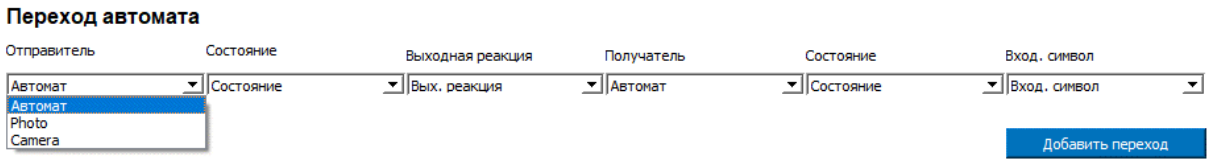


Рисунок 9 – пример выбора автомата для создания перехода

На Рисунок 10 – приведен пример отображения списка созданных названий состояний в QComboBox. В этом списке можно выбрать название состояния, из которого начинается переход в следующее состояние в выбранном автомате. По аналогии выбором автомата и состояния, выбираются выходные реакции, входные воздействия и следующее состояние в которое переходит автомат.

Рисунок 10 – пример выбора состояния для создания перехода

Третий блок интерфейса представляет собой список переходов в автоматной композиции, для проверки правильности внесенных данных область. Область данного блока показана на Рисунок 11 – . А на Рисунок 12 – представлен пример отображения списка переходов.

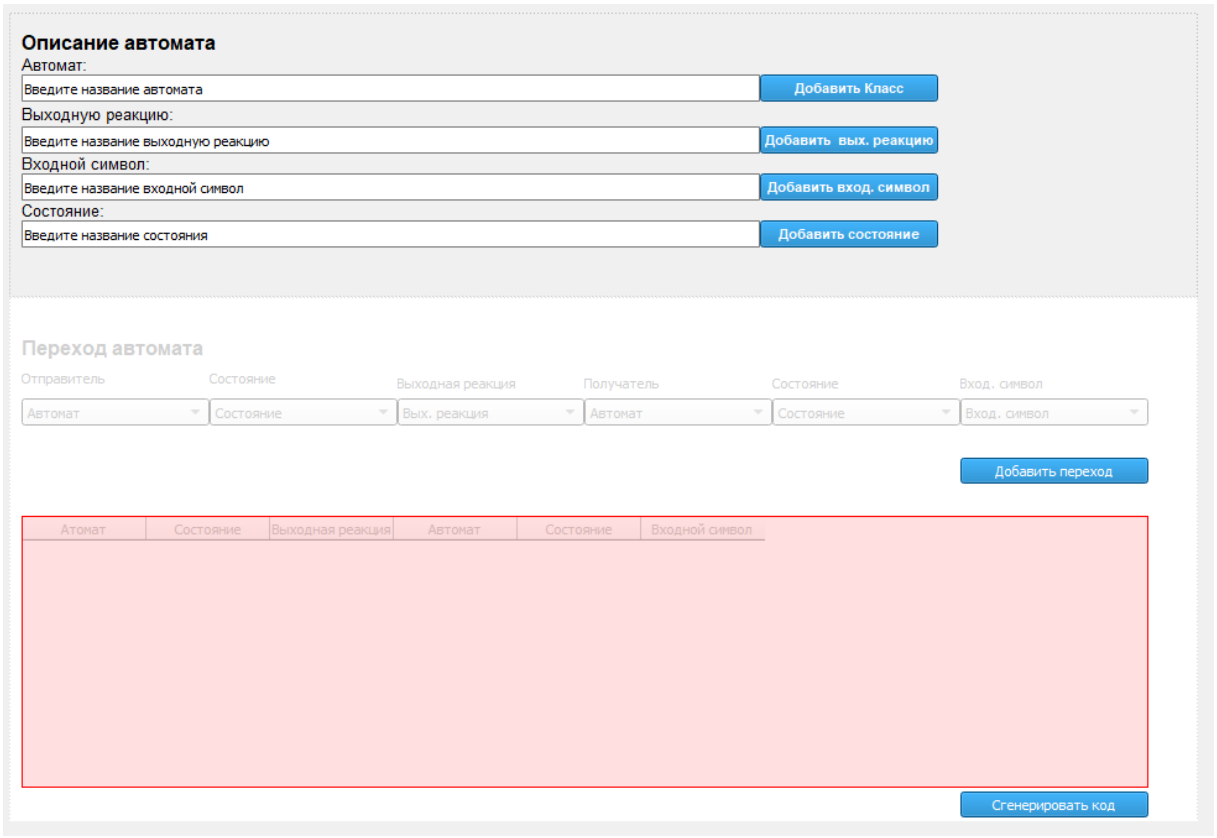


Рисунок 11 – область для отображения списка переходов

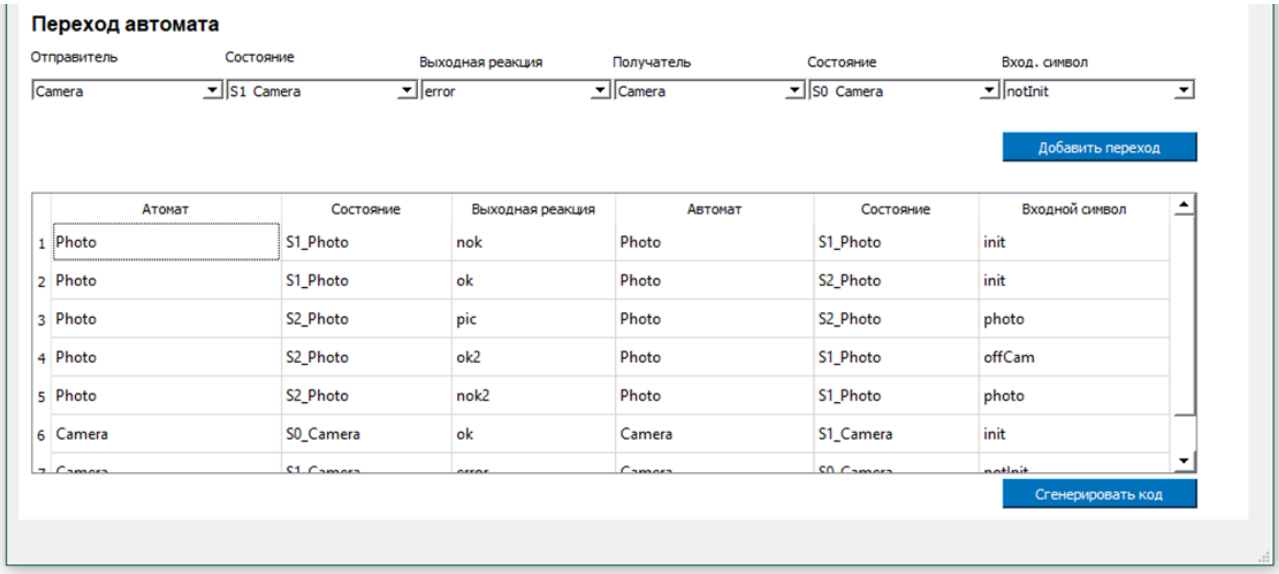


Рисунок 12 – пример отображения списка переходов

После нажатия на кнопку «Сгенерировать код» все данные записываются в XML файл и далее модуль генерации шаблонов считывает этот файл и генерирует шаблоны программы

3.5 Результаты работы системы генерации шаблонов программ на платформе Qt на основе автоматного описания

Система генерации шаблонов программ на платформе Qt, основанная на автоматном описании, была успешно применена для генерации кода на основе представленного автоматного описания. В результате работы системы были сгенерированы файлы .cpp и .h для автоматов Camera (далее – класс Camera) и Photo (далее – класс Photo).

В рамках системы генерации шаблонов программ на платформе Qt были созданы структурированные файлы для класса Camera и Photo. В заголовочном файле объявлен класс Camera, который наследуется от QObject и содержит перечисления состояний и сигналов. Методы-члены класса Camera служат для обработки сигналов и переходов между состояниями. Реализационный файл содержит методы-члены класса Camera, включая конструктор, инициализации, получения фото, перехода в режим ожидания и выключения камеры.

Аналогично для класса Photo были созданы файлы Photo.h и Photo.cpp, в заголовочном файле объявлен класс Photo, который наследуется от QObject и содержит перечисления состояний и сигналов. Методы-члены класса Photo служат для обработки сигналов и переходов между состояниями. Реализационный файл содержит методы-члены класса Photo, включая конструктор, инициализации, снятия фото и выключения камеры.

Также был создан файл BaseClass.h с объявлением класса BaseClass, который наследуется от QObject. Класс BaseClass создает экземпляры классов Camera и Photo, а также потоки для их выполнения. Класс BaseClass координирует и управляет работой классов Camera и Photo, а также устанавливает связи между сигналами и слотами для переходов между классами. Кроме того, у класса BaseClass имеются методы для инициализации системы и освобождения ресурсов.

Подводя итог, можно сказать, что система генерации шаблонов программ на платформе Qt, основанная на автоматном описании, позволяет создавать автоматически эффективный и согласованный код по заданным шаблонам и правилам. Ее результативность и способность упростить процесс разработки программного обеспечения на платформе Qt, обеспечивая стандартизацию и удобство работы с состояниями и переходами между ними, подтверждается практической реализацией.

Код, который получен в итоге после генерации представлен в Приложении А.

ЗАКЛЮЧЕНИЕ

В рамках данной магистерской диссертации была разработана и успешно реализована система автоматизации разработки программ на платформе Qt, основанная на принципах автоматного программирования. Целью данной работы было создание высокоэффективного инструмента, способного автоматически создавать код на основе заранее определенных шаблонов и правил, что ускоряет и облегчает процесс разработки программного обеспечения.

В ходе исследования были проанализированы существующие подходы к разработке программного обеспечения на платформе Qt и изучены основы автоматного программирования, определены требования к системе автоматизации разработки и разработана соответствующая архитектура.

Основной вклад данной работы заключается в разработке и реализации модуля генерации шаблонов программ на платформе Qt, который отвечает за автоматическое создание части кода на основе заранее определенных шаблонов и правил, обеспечивая согласованность и стандартизацию в кодовой базе.

Результаты исследования подтверждают эффективность и практическую ценность разработанной системы. Новый инструмент, основанный на автоматном описании, позволил успешно генерировать код, что значительно ускорило и упростило процесс разработки программного обеспечения и минимизировало потенциальные ошибки. Код, созданный таким образом, соответствует стандартам и требованиям платформы Qt, что гарантирует высокое качество и однородность разработки.

Кроме того, были проведены эксперименты для проверки производительности и эффективности системы, которые подтвердили её способность существенно ускорять процесс разработки программного обеспечения и сокращать трудозатраты благодаря автоматизации генерации кода на основе шаблонов и автоматного описания.

В заключение, следует отметить, что система автоматизации разработки программ на платформе Qt на основе автоматного программирования является ценным инструментом для разработчиков, позволяющим повысить эффективность и качество разработки, а также обеспечить согласованность и стандартизацию в кодовой базе. В дальнейшем стоит рассмотреть возможность расширения функциональности системы, включая поддержку дополнительных возможностей платформы Qt и оптимизацию процесса генерации кода.

СПИСОК ЛИТЕРАТУРЫ

1. Евтушенко Н.В. Недетерминированные автоматы: анализ и синтез: учебное пособие, ч.1 / Н. В. Евтушенко, А.Ф. Петренко, М. В.Ветрова. – Томск: Том. гос. ун-т, 2006. – 142 с.
2. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Automata Theory, Languages, and Computation" / John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. — 3rd Edition. — Ithaca NY and Stanford CA: , 2006 — 550 с.
3. Linz, Peter An introduction to formal languages and automata / Linz, Peter — XVIII. — Lexington, Mass. : D.C. Heath: , 1990 — 373 с.
4. John E. Hopcroft, Jeffrey D. Ullman Introduction to Automata Theory, Languages, and Computation / John E. Hopcroft, Jeffrey D. Ullman — XVIII. — Lexington, Mass. : D.C. Heath: Addison-Wesley Publishing Company, 1979 — 537 с
5. Prokopenko S. Locating a faulty component of an EFSMcomposition // Труды ИСП РАН. 2014. Vol. 26, № 6. P. 47-56.
6. Sotnikov A. P. Experiments On Parallel Composition of Timed Finite State Machines / A. P. Sotnikov, N. V. Shabaldina, M. L. Gromov // Труды ИСП РАН 29:3 (2017). С. 233-246.
7. М. Л. Громов, Н. В. Шабалдина, Построение каскадной параллельной композиции временных автоматов с использованием BALM-II, Модел. и анализ информ. систем, 2016, том 23, номер 6, 715–728
8. Поликарпова, Н.И., Шалыто, А.А. Автоматное программирование / Н.И. Поликарпова, А.А. Шалыто — Санкт-Петербург: , 2009 — 176 с.
9. Qt [Электронный ресурс] : Cross-platform software development for embedded & desktop. – URL: https://www.qt.io/ (дата обращения 01.06.2023).
10. Кузнецова О.В. Применение Qt в разработке кроссплатформенных приложений. Издательский дом "БХВ-Петербург", 2012
11. Болтова В.С. Разработка XML-представления для описания структуры автоматной композиции / Болтова В.С., Шабалдина Н.В. // Материалы Двенадцатой конференции с международным участием “Новые информационные технологии в исследовании сложных структур”, Издательство Национальный исследовательский Томский государственный университет, Томск, 2018 – 68 с.
12. . Конечный автомат [Электронный ресурс]. – URL: https://tproger.ru/translations/finite-state-machines-theory-and-implementation/ (дата обращения 27.11.2022).

13. Daphne A. Norton. Algorithms for testing equivalence of finite automata, with a grading tool for jflap // Technical report, Rochester of finite Technology, Departament of Computer Science (2009). – 69 с.
14. Евтушенко Н.В. Некоторые задачи идентификации состояний для недетерминированных автоматов / Н. В. Евтушенко, Н.Г. Кушик. Томск: STT, 2018. – С. 15-21.
15. Гилл А. Введение в теорию конечных автоматов (серия “Теоретические основы технической кибернетики”). М.: Наука, 1966 – С. 8-118.
16. Л. К. Левит-Гуревич, Д. М. Ярошевский, Схема динамического программирования с многомерной индексацией шагов, Автомат. и телемех., 2006, выпуск 9, 23–40
17. Введение в теорию автоматов, языков и вычислений / журнал «Автоматика и телемеханика». — 2006. — № выпуск 9. — С. 23-40.
18. Хиршберг Д., Лукверт А., Ульман Д.Ж. Алгоритмы. Построение и анализ : учебник / Хиршберг Д., Лукверт А., Ульман Д.Ж. ; пер. с англ. В.В. Минаев и др.; под ред. д-ра физ.-мат. наук А.Д. Вербицкой. – 2-е изд. – М. : Вильямс, 2006. – 1192 с.
19. Ульман Д.Ж. Автоматы и грамматики / Ульман Д.Ж. – М. : Мир, 1977. – 344 с.
20. Мухамедиев А.С. Моделирование временных автоматов с использованием языка Verilog, Сборник научных трудов XIII международной научно-практической конференции «Разработка и перспективы использования сетевых технологий в науке, образовании и индустрии» (WEB-2018). – М.: Краснодарский край, Краснодар, 2018. – 165 с.
21. Коренер Г., Михалис Яннакакис. Формализация процесса синтеза цифровых схем. Дискретные устройства и цифровые устройства. М.: Мир, 1983. 297 с.
22. Михалис Яннакакис. Обобщённый язык цифровой разработки. IEEE DESIGN & Test of Computers, 1986. Vol. 3. №2. PP. 38-43.
23. Hopcroft J.E., Ullman J.D. Formal Languages and Their Relation to Automata. Addison-Wesley. – Reading, Mass.: 1969. – 423 p.
24. Zisman G.A. Введение в теорию автоматов. – М. : Физматлит, 2006. – 352 с.
25. Шубин Г.Г. Труды по теории автоматов. - М. : Наука, 1969. – 704 с.
26. Макаренко О.С. Описание интерфейсов между пользовательской надстройкой и формализацией задач принятия решений в виде конечных автоматов / О.С. Макаренко, М.Л. Громов // Труды ИСП РАН. – 2018. – №10. – С. 13-26.
27. Jeffrey Shallit. Ontogeny of Finite Automata // A Celebration of the Life and Work of Thomas Garrity, World Scientific Press, 2016, pp. 89-106.

28. Ginzburg L. Разработка баз знаний на естественном языке: создание корпуса текстов в парадигме дистрибутивной семантики / Л. Гинзбург // Научно-техническая информация. Сер. 2. Информационные процессы и системы. – 2019. – Т. 7, № 4. – С. 243-251.
29. Ambler A. W. Prentice Hall International series in computer science [Электронный ресурс]: Flexible Life Cycle Management for Software Development — URL: https://ccr.sigsoft.org/invited/ambler.pdf (дата обращения: 20.10.2022).
30. Громов М.Л., Шабалдина Н.В. Методы упрощения временных автоматных моделей, описывающих бизнес-процессы, на основе правил индукции / М.Л. Громов, Н.В. Шабалдина // Труды ИСП РАН. – 2016. – №7. – С. 85-94.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

КОД СГЕНЕРИРОВАННЫХ ФАЙЛОВ

baseclass.h:

```
#ifndef BASECLASS_H
#define BASECLASS_H

#include <QObject>
#include <QThread>

#include <iostream>
#include "Camera.h"
#include "Photo.h"

class BaseClass : public QObject {
    Q_OBJECT

public:
    explicit BaseClass(QObject* parent = nullptr);
    ~BaseClass();
    void init();
private:
    QThread CameraThread;
    Camera* m_Camera;
    QThread PhotoThread;
    Photo* m_Photo;

signals:
    void doWork();

};

#endif // BASECLASS_H
```

baseclass.cpp:

```
#include "baseclass.h"
```

```
BaseClass::BaseClass(QObject* parent) : QObject(parent) {
    m_Camera = new Camera;
    m_Photo = new Photo;
    m_Photo->moveToThread(&PhotoThread);
    m_Camera->moveToThread(&CameraThread);

    // Подключение сигналов и слотов
    connect(this, &BaseClass::doWork, m_Camera,
&Camera::initCam);
    connect(m_Camera, &Camera::error, m_Camera,
&Camera::notInit);
    connect(m_Camera, &Camera::error2, m_Camera,
&Camera::errorCam);
    connect(m_Camera, &Camera::photo, m_Camera, &Camera::getPic);
    connect(m_Camera, &Camera::ok2, m_Camera, &Camera::ready);
    connect(m_Camera, &Camera::pic, m_Camera, &Camera::showPic);
      connect(m_Camera, &Camera::off, m_Camera,
&Camera::offPhoto);

    connect(m_Photo, &Photo::ok, m_Photo, &Photo::init);
    connect(m_Photo, &Photo::nok, m_Photo, &Photo::init);
    connect(m_Photo, &Photo::pic, m_Photo, &Photo::photo);
    connect(m_Photo, &Photo::nok2, m_Photo, &Photo::photo);
    connect(m_Photo, &Photo::ok2, m_Photo, &Photo::offCam);

    CameraThread.start();
    PhotoThread.start();
}

BaseClass::~BaseClass() {
    delete m_Camera;
    CameraThread.quit();
    CameraThread.wait();
    delete m_Photo;
    PhotoThread.quit();
    PhotoThread.wait();
}
void BaseClass::init() {
```

```
    emit doWork();
}
```

Photo.h:

```
#ifndef Photo_H
#define Photo_H

#include <QObject>

enum PhotoStates
{
     S1_Photo,
     S2_Photo
};

enum PhotoInputSignals{
     S_photo,
     S_offCam,
     S_init,
};

const PhotoStates Photo_STATE = S1_Photo;

class Photo : public QObject
{
    Q_OBJECT

     PhotoStates myState;
    PhotoInputSignals i_signal;
public:
    explicit Photo(QObject* parent = nullptr);
    void mainSignalHandler();

public slots:
    void init();
    void photo();
    void offCam();
```

```
signals:
     void ok();
     void ok2();
     void pic();
     void nok();
     void nok2();
};

#endif // Photo_H
```

Photo.cpp:

```
#include "Photo.h"
#include <QDebug>

Photo::Photo(QObject* parent) : QObject(parent),
myState(Photo_STATE)
{ }

void Photo::init(){
    this->i_signal = S_init;
    mainSignalHandler();
qDebug() << "S_init";
}

void Photo::photo(){
    this->i_signal = S_photo;
    mainSignalHandler();
qDebug() << "S_photo";
}

void Photo::offCam(){
    this->i_signal = S_offCam;
    mainSignalHandler();
qDebug() << "S_offCam";
}

void Photo::mainSignalHandler(){
```

```
switch(myState) {
case  S1_Photo:
    switch(i_signal) {
    case S_init:
        //TODO implement

        if(/*TODO implement*/){
            emit ok();
            myState =  S2_Photo;
        } else {
            emit nok();
            myState =  S1_Photo;
        }

        break;
    case S_photo:
        //TODO implement
        break;
    case S_offCam:
        //TODO implement
        break;

    default:  emit nok();
    }
    break;

case S2_Photo:
    switch(i_signal) {
    case S_init:
        //TODO implement
        break;
    case S_photo:
        //TODO implement

        if(/*TODO implement*/){
            emit nok2();
            myState =  S1_Photo;
        } else {
            emit pic();
```

```
                myState =  S2_Photo;
            }

            break;
        case S_offCam:
            //TODO implement

            if(/*TODO implement*/){
                emit ok2();
                myState =  S1_Photo;
            }  else {
                emit pic();
                myState =  S2_Photo;
            }

            break;

        default:  emit nok();
        }
        break;

    default:
        emit nok();
    }
}
```

Camera.h:

```
#ifndef Camera_H
#define Camera_H

#include <QObject>

enum CameraStates
{
    S0_Camera,
    S1_Camera,
    S2_Camera,
    S3_Camera
```

```
};

enum CameraInputSignals{
    S_notInit,
    S_errorCam,
    S_ready,
    S_getPic,
    S_offPhoto,
    S_showPic,
    S_initCam,
};

const CameraStates CameraSTATE = S0_Camera;

class Camera : public QObject
{
    Q_OBJECT

     CameraStates myState;
    CameraInputSignals i_signal;

public:
    explicit Camera(QObject* parent = nullptr);
     void mainSignalHandler();

public slots:
    void notInit();
    void errorCam();
    void ready();
    void getPic();
    void offPhoto();
    void showPic();
    void initCam();

signals:
    void ok();
    void error();
    void error2();
    void ok2();
```

```
        void photo();
        void off();
        void pic();

    };

    #endif // Camera_H
```

Camera.cpp:

```
    #include "Camera.h"
    #include <QDebug>
    Camera::Camera(QObject* parent) : QObject(parent),
myState(CameraSTATE)
    { }


    void Camera::initCam() {
        this->i_signal = S_initCam;
        mainSignalHandler();
         qDebug() << "S_initCam";
    }

    void Camera::errorCam() {
        this->i_signal = S_errorCam;
        mainSignalHandler();
        qDebug() << "S_errorCam";
    }

    void Camera::notInit() {
        this->i_signal = S_notInit;
        mainSignalHandler();
        qDebug() << "S_notInit";
    }
    void Camera::ready() {
        this->i_signal = S_ready;
        mainSignalHandler();
        qDebug() << "S_ready";
    }
```

```
void Camera::getPic() {
    this->i_signal = S_getPic;
    mainSignalHandler();
    qDebug() << "S_getPic";
}
void Camera::offPhoto() {
    this->i_signal = S_offPhoto;
    mainSignalHandler();
}
void Camera::showPic() {
    this->i_signal = S_showPic;
    mainSignalHandler();
}

void Camera::mainSignalHandler(){
    switch(myState) {
    case  S0_Camera:
        switch(i_signal) {
        case S_initCam:
            //TODO implement

            if(/*TODO implement*/){
                emit ok();
                myState =  S1_Camera;
            }

            break;
        case S_notInit:
            //TODO implement
            break;
        case S_errorCam:
            //TODO implement
            break;

        case S_ready:
            //TODO implement
            break;
        case S_getPic:
            //TODO implement
```

```
            break;
        case S_offPhoto:
            //TODO implement
            break;
        case S_showPic:
            //TODO implement
            break;

        default:  emit error();
        }
        break;

    case S1_Camera:
        switch(i_signal) {
        case S_initCam:
            //TODO implement
            break;
        case S_notInit:
            //TODO implement

            if(/*TODO implement*/){
                emit error();
                myState =  S0_Camera;
            }

            break;
        case S_errorCam:
            //TODO implement
            break;

        case S_ready:
            //TODO implement

            if(/*TODO implement*/){
                emit ok2();
                myState =  S2_Camera;
            }

            break;
```

```
        case S_getPic:
            //TODO implement
            break;
        case S_offPhoto:
            //TODO implement
            break;
        case S_showPic:
            //TODO implement
            break;

        default:  emit error();
        }
        break;


    case S2_Camera:
        switch(i_signal) {
        case S_initCam:
            //TODO implement
            break;
        case S_notInit:
            //TODO implement
            break;
        case S_errorCam:
            //TODO implement
            break;

        case S_ready:
            //TODO implement
            break;
        case S_getPic:
            //TODO implement

            if(/*TODO implement*/){
                emit photo();
                myState =  S3_Camera;
            }

            break;
```

```
    case S_offPhoto:
        //TODO implement

        if(/*TODO implement*/){
            emit off();
            myState =  S0_Camera;
        }

        break;
    case S_showPic:
        //TODO implement
        break;

    default:  emit error2();
    }
    break;


case S3_Camera:
    switch(i_signal) {
    case S_initCam:
        //TODO implement
        break;
    case S_notInit:
        //TODO implement
        break;
    case S_errorCam:
        //TODO implement

        if(/*TODO implement*/){
            emit error2();
            myState =  S0_Camera;
        }

        break;

    case S_ready:
        //TODO implement
        break;
```

```
    case S_getPic:
        //TODO implement
        break;
    case S_offPhoto:
        //TODO implement
        break;
    case S_showPic:
        //TODO implement

        if(/*TODO implement*/){
            emit pic();
            myState =  S2_Camera;
        }

        break;

    default:  emit error2();
    }
    break;


default:
    emit error2();
}
```

ПРИЛОЖЕНИЕ

Отчет о патентных исследованиях

УТВЕРЖДАЮ

_____/Д.Я. Суханов/

д-р физ.-мат. наук

"_____" _____ 2021г.

ЗАДАНИЕ
на проведение патентных исследований

Наименование работы (темы) Система автоматизации разработки программ на платформе QT с проверкой зацикливаний на основе автоматного программирования

Шифр работы (темы) MP-S

Этап работы первый, сроки его выполнения 18.09.2021- 20.12.2021

Задачи патентных исследований: Поиск алгоритмов и методов верификации; поиск систем генерации шаблонов программ; поиск программ по созданию модуля проверки автоматных композиций; поиск устройств, применяющих в своей работе;

КАЛЕНДАРНЫЙ ПЛАН

Виды патентных исследований	Подразделения-исполнители (соисполнители)	Ответственные исполнители (Ф.И.О.)	Сроки выполнения патентных исследований. Начало. Окончание	Отчетные документы
1. Поиск алгоритмов и методов верификации	КИТИДиС ТГУ	А.Ф. Фоминых	18.09.2021-11.10. 2021	отчет о поиске
2. Поиск программ, применяющих для генерации шаблонов программ и созданию модуля проверки автоматных композиций	КИТИДиС ТГУ	А.Ф. Фоминых	12.10.2021-18.11. 2021	отчет о поиске
3. Вывод по найденным алгоритмам, методам и устройствам	КИТИДиС ТГУ	А.Ф. Фоминых	19.11.2021-20.12. 2021	отчет о поиске

Руководитель патентного подразделения	_____ личная подпись	В. П. Беличенко расшифровка подписи	25.12.2021 дата
Руководитель ВКР магистра	_____ личная подпись	М. Л. Громов расшифровка подписи	25.12.2021 дата

Регламент поиска

22.09.2021
дата составления регламента

Наименование работы (темы) Система автоматизации разработки программ на платформе QT на основе автоматного программирования
Шифр работы (темы) МОР
Номер и дата утверждения задания от 18.09.2021 Этап работы первый
Цель поиска информации (в зависимости от задач патентных исследований, указанных в задании)
Задачей является поиск алгоритмов и методов автоматного программироватния; поиск систем генерации шаблонов программ; поиск программ по созданию модуля проверки автоматных композиций

Обоснование регламента поиска поиск провести в базах ФИПС и USPTO
Начало поиска 18.09.2021 Окончание поиска 20.12.2021

Предмет поиска (объект исследования, его составные части, товар)	Страна поиска	Источники информации, по которым будет проводиться поиск				Ретроспективность	Наименование информационной базы
		Патентные		НТИ			
		Наименование	Классификационные рубрики МПК	Наименование	Рубрики УДК		
1	2	3	4	5	6	7	8
Алгоритмы и методы автоматного программирования; Генерации шаблонов программ model checking	Россия, США	База данных ФИПС (Россия) База данных USPTO (США)	МПК G06F 15/16 G06F 11/36 G06F 11/3692; G06F 11/3684; G06F 11/3676; G06F 11/3688 G06F 3/0482; G06F 9/451; G06F 16/901	Сайт ЕГИСУ НИОКТР Научная сессия ГУАП ООО "Издательство Молодой ученый" Программные продукты и системы Научно-технический вестник СПбГУ ИТМО Моделирование и анализ информационных систем (МАИС) Science amp; Business Media, Springer, Berlin, Heidelberg В мире науки Вестник Воронежского государственного технического университета ООО "Информационно-управляющие системы"	004.42.02; 004.052.42 004.942.001.57 004.896 004.4'22 681.3.06 004.414.28 + 004.415.52	1998-2021 (Россия) 1976-2021 (США)	База данных федерального института промышленной собственности, Россия База данных патентного ведомства США

Руководитель ВКР магистра — личная подпись — М. Л. Громов (расшифровка подписи) — 25.12.2021 (дата)

Руководитель патентного подразделения — личная подпись — В. П. Беличенко (расшифровка подписи) — 25.12.2021 (дата)

ОТЧЕТ О ПОИСКЕ

В.1 Поиск проведен в соответствии с заданием

от 18.09.2021 и Регламентом поиска № от 22.09.2021

В.2 Этап работы первый

В.3 Начало поиска 18.09.2021 Окончание поиска 20.12.2021

В.4 Сведения о выполнении регламента поиска (указывают степень выполнения регламента поиска,

отступления от требований регламента, причины этих отступлений) – регламент поиска выполнен полностью.

В.5 Предложения по дальнейшему проведению поиска и патентных исследований – провести поиск патентов аналогов наиболее значимых патентов, обнаруженных при проведении патентных исследований. Расширить поиск текущей научной и патентной информации в смежных областях.

В.6 Материалы, отобранные для последующего анализа

Таблица В.6.1 – Патентная документация

Предмет поиска (объект исследования, его составные части)	Страна выдачи, вид и номер охранного документа. Классификационный индекс	Заявитель (патентообладатель), страна. Номер заявки, дата приоритета.	Название изобретения	Сведения о действии охранного документа
1	2	3	4	5
Поиск алгоритмов и методов автоматного программирования; поиск систем генерации шаблонов программ; поиск программ по созданию модуля проверки автоматных композиций	1. RU 2604431 C2 G06F 15/16	МАЙКРОСОФТ ТЕКНОЛОДЖИ ЛАЙСЕНСИНГ, ЭлЭлСи (US) 2013155469/08, 12.06.2012	Автоматизированное преобразование объекта интерфейса пользователя и	Не действует
	2. RU 2021619945 (свидетельство о регистрации программы для ЭВМ)	Федеральное государственное бюджетное образовательное учреждение высшего образования «Кабардино-Балкарский государственный университет им. Х.М. Бербекова» (КБГУ) (RU) 2021618785 09.06.2021	«RecoWorkArea»	Действует
	3. US 20200026641 A1 G06F 11/36; G06F 3/0482; G06F 9/451; G06F 16/901	INTERNATIONAL BUSINESS MACHINES CORPORATION (Armonk, NY) US 61070134, 30.09.2019	Automated test input generation for integration testing of microservice-based web applications	Действует
	4. US 20200242016 A1 G06F 11/36; G06F 11/3692; G06F 11/3684; G06F 11/3676; G06F 11/3688	INTERNATIONAL BUSINESS MACHINES CORPORATION (Armonk, NY) US 71732533, 24.01.2019	Test space analysis across multiple combinatoric models	Действует
	5. US 20200175128 A1 G06F 30/30; G06F 30/33; G06F 30/331	International Business Machines Corporation (Armonk, NY) US 70850279, 29.11.2018	Hardware incremental model checking	Действует
	6. US 20200278922 A1 G06F 11/3688; G06F 11/3664; G06F 11/3696; G06F 11/3608;	B. G. NEGEV TECHNOLOGIES AND APPLICATIONS LTD., AT BEN-GURION UNIVERSITY (Beer Sheva, IL)	Scenario based method for testing software	Действует

	G06F 11/3684; G06F 11/36	72236680, 24.02.2020		
	7. US 20190377664 A1 G06F 11/36; G06F 9/44; G06F 11/00; G06F 11/34	International Business Machines Corporation (Armonk, NY) US 51534654, 26.08.2019	Variant modeling elements in graphical programs	Действует
	8.	https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=34&f=G&l=50&co1=AND&d=PTXT&s1=%22model+checking%22&OS=%22model+checking%22&RS=%22model+checking%22 https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=29&f=G&l=50&co1=AND&d=PTXT&s1=%22model+checking%22&OS=%22model+checking%22&RS=%22model+checking%22 https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=37&f=G&l=50&co1=AND&d=PTXT&s1=%22model+checking%22&OS=%22model+checking%22&RS=%22model+checking%22 https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=45&f=G&l=50&co1=AND&d=PTXT&s1=%22model+checking%22&OS=%22model+checking%22&RS=%22model+checking%22 https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=30&f=G&l=50&co1=AND&d=PTXT&s1=%22model+checking%22&OS=%22model+checking%22&RS=%22model+checking%22		

Таблица В.6.2 – Научно-техническая, конъюнктурная, нормативная документация и материалы государственной регистрации (отчеты о научно-исследовательских работах)

Предмет поиска	Наименование источника информации с указанием страницы источника	Автор, фирма (держатель) технической документации	Год место и орган издания (утверждения, депонирования источника)
1	2	3	4
Поиск алгоритмов и методов автоматного программирования; поиск систем генерации шаблонов программ; поиск программ по созданию модуля проверки автоматных композиций	1. Анализ особенностей использования гибридных суперкомпьютерных архитектур при решении фундаментальных и прикладных задач и разработка методики оценки гибридных суперкомпьютерных систем	Автор: Лукашин А. А ФГАОУ ВО "СПбПУ"	23 мая 2017 г. Сайт ЕГИСУ НИОКТР: https://www.rosrid.ru/ikrbs/detail/BHN13UHSPOUY9OSECENDWGKV Ссылка на отчет: blob:https://www.rosrid.ru/b8b126b6-c37b-4b90-b5c8-d3a6b5c7f868
	2. Методы формальной верификации для анализа крупноблочных параллельных программ Стр: 190-197	Пахарев С.М., Сыщиков А.Ю. СПбГУ аэрокосмического приборостроения	2017, г.Санкт-Петербург Источник: Научная сессия ГУАП Издательство: СПбГУ аэрокосмического приборостроения
	3. Методы верификации программного обеспечения Стр: 138-141	Егоров В.В., Томилова Н.И., Амиров А.Ж., Касылкасова К.Н. Карагандинский государственный технический университет	2016 г. Журнал: Молодой Ученый Учредители: ООО "Издательство Молодой ученый"
	4. Автоматизация верификации программ с использованием графоаналитических моделей вычислительного процесса Стр: 398-402	Зыков А.Г., Поляков В.И., Голованев Я.С. СПбГУ ИТМО	2019 г. Журнал: «Программные продукты и системы.
	5. Введение в верификацию автоматных программ на основе метода model checking Стр: 33-48	Вельдер С.Э., Шалыто А.А. СПбГУ ИТМО	2007 г. Журнал: Научно-технический вестник СПбГУ ИТМО
	6. Автоматизация верификации С-программ с использованием символического метода элиминации инвариантов циклов Стр: 491-505	Кондратьев Д.А., Марьясов И.В., Непомнящий В.А. При поддержке: РФФИ № 17-01-00789	2018 г. Журнал: Моделирование и анализ информационных систем (МАИС) DOI: 10.18255/1818-1015-2018-5-491-505

	7. Systems and software verification: model-checking techniques and tools	Berard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P.	Original French by Vuibert, Paris, 1999 Science amp; Business Media, Springer, Berlin, Heidelberg DOI: 10.1007/978-3-662-04558-9
	8. Программы проверяют программы Стр. 52-57.	Джексон Д.	2006 г. В мире науки. № 10.
	9. Методика верификации автоматных программ Стр: 15-21	Шалыто А.А., Егоров К.В. СПбГУ ИТМО	2008 г. ООО "Информационно-управляющие системы"
	10. Автоматическая генерация автоматного кода Стр: 35-42	Канжелев С.Ю., Шалыто А.А. СПбГУ ИТМО	2006 г. ООО "Информационно-управляющие системы"
	11. Построение генератора программного кода для решения инженерных задач Стр: 14-19	Минакова О.В., Трубников И.В., Курипта О.В. Воронежский государственный технический университет	2020 г. Журнал: Вестник Воронежского государственного технического университета

Выводы по результатам патентного поиска

Целью магистерской диссертации, в рамках которой проводились патентные исследования, является создание системы генерации шаблонов программ

По результатам патентного поиска можно сказать, что в последние десять лет в области исследования методов автоматного программирования и генерации программного кода встречаются не часто, в основном патенты с упоминанием автоматного программирования или генерации кода встречаются в Американской базе USPTO. В Российской базе ФИПС был найден 1 патент, наиболее относящий к теме данной магистерской диссертации, а также было найдено 1 свидетельство о регистрации программы для ЭВМ. По результатам литературного обзора было замечено, что за последние 20 лет тема автоматного программирования является актуальной. И можно встретить не одну работу, близко относящуюся к данной магистерской работе.

Из вышеизложенное можно сделать вывод что по темам, которые поднимаются для решения поставленной цели проводится много исследований, но не часто можно встретить зарегистрированные программы для ЭВМ и запатентованные работы. Но если посмотреть на количество патентов в базе USPTO за последние 5 лет, то можно заметить, что тема проверки моделей и верификации ПО встречается чаще. Это наводит на мысль, что со временем интерес к данной теме может возрасти.

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Томский Государственный Университет

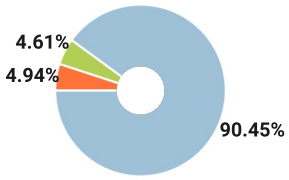
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Фоминых Александра Федоровна
Самоцитирование рассчитано для: Фоминых Александра Федоровна
Название работы: Отчет_по_НИР_Фоминых
Тип работы: Магистерская диссертация
Подразделение:

РЕЗУЛЬТАТЫ

СОВПАДЕНИЯ	4.94%
ОРИГИНАЛЬНОСТЬ	90.45%
ЦИТИРОВАНИЯ	4.61%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 22.06.2023



Структура документа: Проверенные разделы: основная часть с.2-3, 5-32, приложение с.36-58

Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс*; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по коллекции Гарант: аналитика; Переводные заимствования по коллекции Интернет в английском сегменте; Переводные заимствования по Интернету (EnRu); Переводные заимствования по коллекции Интернет в русском сегменте; Переводные заимствования издательства Wiley ; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика; Перефразирования по Интернету; Перефразирования по Интернету (EN); Перефразированные заимствования по коллекции Интернет в английском сегменте; Перефразированные заимствования по коллекции Интернет в русском сегменте; Перефразирования по коллекции

Работу проверил: Лапутенко Андрей Владимирович

ФИО проверяющего

Дата подписи: 22-06-2023

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.