Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ БАКАЛАВРА НА ТЕМУ:

Программная подсистема имитационного моделирования процессов

Студент	ИУ6-83Б		А.А. Бушев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			М.В. Фетисов
		(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер			
		(Подпись, дата)	(И.О. Фамилия)

2023 г.

АННОТАЦИЯ

Расчетно-пояснительная записка выпускной квалификационной работы бакалавра посвящена процессу проектирования и разработки программной подсистемы имитационного моделирования процессов. Для реализации подсистемы был проведен анализ библиотек и фреймворков поддержки межпроцессного взаимодействия. Помимо этого, был разработан протокол взаимодействия между плагином интеграции со средой разработки и сервером имитационного моделирования.

На основе проведенных исследований была спроектирована и реализована программная подсистема имитационного моделирования процессов.

ANNOTATION

Calculation and explanation summary of graduation qualification work of the bachelor is devoted to process of designing and development of software subsystem of process simulation. For subsystem realization analysis of libraries and frameworks supporting interprocess communication is conducted. Also protocol of interaction between the integration plugin with the development environment and the simulation server was developed.

Based on conducted research program, the software subsystem of process simulation was developed.

РЕФЕРАТ

Расчетно-пояснительная записка X с., X рис., X табл., X источн., X прил. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ, ПОДСИСТЕМА, МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ

Объектом разработки является программная подсистема имитационного моделирования процессов.

Целью данной работы является подсистема, которая позволит проводить имитационное моделирование динамических процессов в адаптивной среде разработки.

В ходе выполнения работы были решены следующие задачи:

— исследование библиотек и фреймворков поддержки межпроцессного взаимодействия;

— сравнение аналогов;

— проектирование и разработка диаграмм;

— реализация программной подсистемы.

В результате разработки была спроектирована, реализована и протестирована подсистема, которая предназначена для имитационного моделирования динамических процессов студентами кафедры «Системы автоматического управления» в адаптивной среде разработки SIMODO.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

SIMODO — название программных продуктов, разрабатываемых на кафедре ИУ6 МГТУ им. Баумана.

Git — система контроля версий.

LSP — открытый протокол, разрешающий реализацию и распространение поддержки языка программирования независимо от любого данного редактора или интегрированной среды разработки.

LSP+ — модификация протокола LSP под требования интегрированной среды разработки SIMODO.

ОС — операционная система.

Windows — операционная система для компьютеров от корпорации Microsoft.

Linux — семейство операционных систем на основе открытого ядра.

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде.

UDP — протокол пользовательских датаграмм, один из ключевых элементов набора сетевых протоколов для Интернета.

ПП — программный продукт.

ООП — объектно-ориентированное программирование.

scriptC0 — первый скриптовый язык SIMODO.

ПО — программное обеспечение.

RPM — Red Hat Package Manager — формат пакетов программного обеспечения, а также программа, созданная для управления этими пакетами, используемые в ряде Linux-дистрибутивов.

DEB — расширение имён файлов «бинарных» пакетов для распространения и установки программного обеспечения в операционной системе проекта Debian, и других, использующих систему управления пакетами dpkg.

NSIS — Nullsoft Scriptable Install System — система создания установочных программ для Microsoft Windows с открытым исходным кодом.

MSI — Microsoft Installer — подсистема Microsoft Windows, обеспечивающая установку программ.

QtIFW — Qt Installer Framework — фреймворк для разработки установщиков с графическим интерфейсом.

DevOps — Методология автоматизации технологических процессов сборки, настройки и развёртывания программного обеспечения.

Gitlab — Веб-инструмент жизненного цикла DevOps с открытым исходным кодом.

ТЗ — техническое задание.

Стек технологий — набор инструментов, применяющийся при работе в проектах и включающий языки программирования, фреймворки, СУБД.

ОДУ — обыкновенное дифференциальное уравнение.

СДУ — система дифференциальных уравнений.

ВВЕДЕНИЕ

Заглушка

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В настоящее время для адаптивной системы моделирования SIMODO[1] актуален вопрос межпроцессного взаимодействия. Само моделирование должно происходить отдельно от визуализации результатов моделирования. Однако возникает вопрос, какой инструмент использовать для организации взаимодействия процесса моделирования и процесса визуализации результатов моделирования.

Исследовательская работа посвящена обзору существующих библиотек и фреймворков поддержки межпроцессного взаимодействия. А также проведению сравнительного анализа библиотек и фреймворков поддержки межпроцессного взаимодействия. Помимо этого, в работе разработан протокол взаимодействия с адаптивной системой моделирования SIMODO.

1.1 ОПИСАНИЕ ОТКРЫТОЙ АРХИТЕКТУРЫ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ SIMODO

Основной задачей интегрированная среды разработки SIMODO является обеспечение удобного средства описания моделей (помимо запуска моделирования и анализа результатов моделирования) на базе открытой архитектуры[2]. Открытая архитектура интегрированной среды разработки SIMODO позволяет расширить базовую функциональность для работы с различными документами.

Основной задачей интегрированной среды разработки SIMODO является предоставление простого средства редактирования текста моделей. Поскольку любой язык программирования может быть использован для написания фрагментов модели, то среда предоставляет гибкие средства настройки, не замыкаясь на наборе предметно-ориентированных языков SIMODO.

На рисунке 1 изображён графический интерфейс интегрированной среды разработки SIMODO.

Интегрированная среда разработки SIMODO состоит из расширяемого редактора — оболочка редактора и расширения (плагины) — и процессов, выполняющие разнообразные функции, которые целесообразно отделить от

редактора межпроцессным взаимодействием[1]. Преимуществом такого отделения является отказоустойчивость всей интегрированной среды разработки SIMODO.

Оболочка редактора предоставляет базовую функциональность и механизм расширения. После запуска оболочка редактора загружает плагины. Плагины являются фабриками для соответствующих компонентов. Компоненты, которые поставляются плагинами, расширяют функциональность редактора. В свою очередь оболочка редактора предоставляет для указанных плагинов и порождающих ими компонентов интерфейс доступа к некоторым своим функциям.

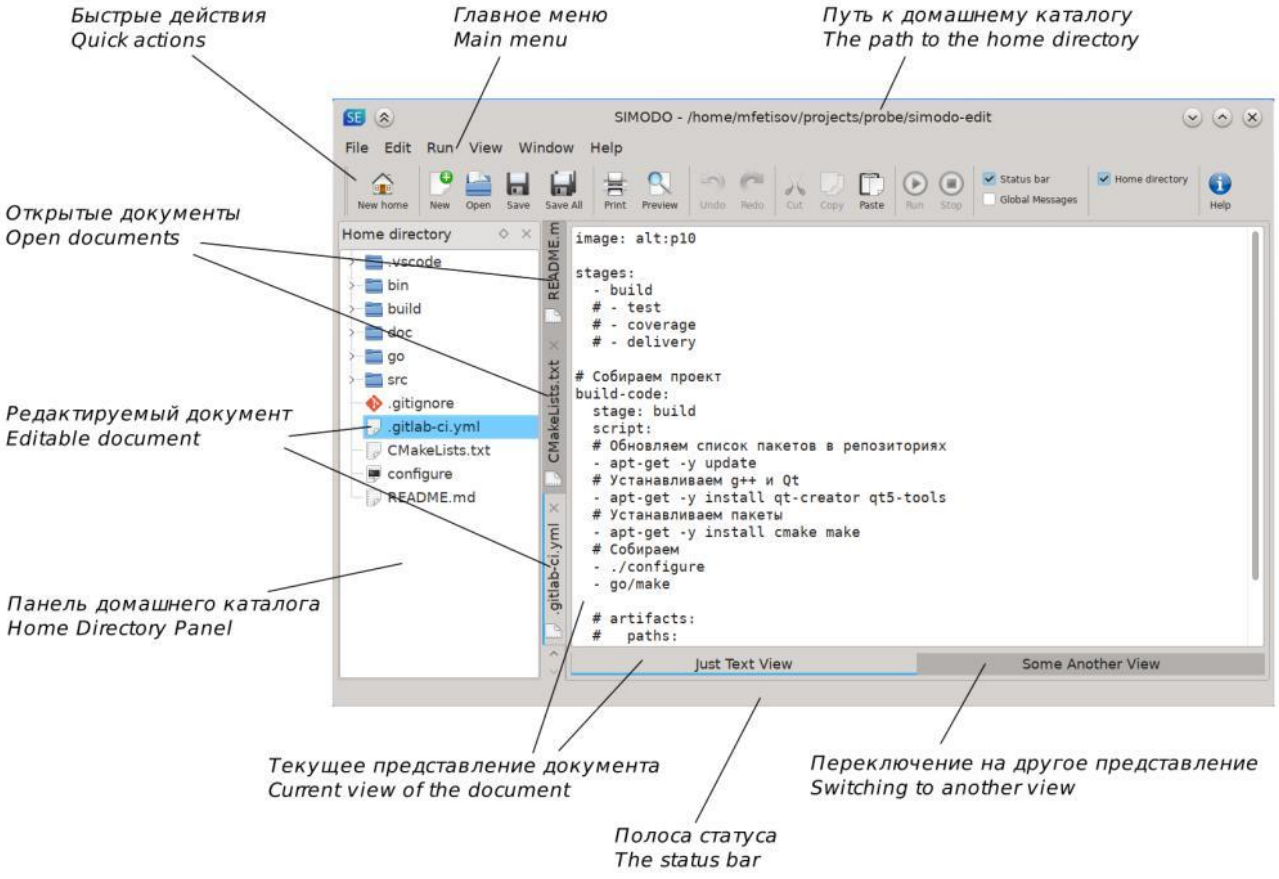

Рисунок 1 — Графический интерфейс интегрированной среды разработки SIMODO.

Сама оболочка не выполняет отображение или управление документом. Вместо этого она предоставляет пространство, где размещает панели и представление документов, отображение которых определяется реализацией плагинов.

На рисунке 2 представлена схема интерфейсов оболочки.

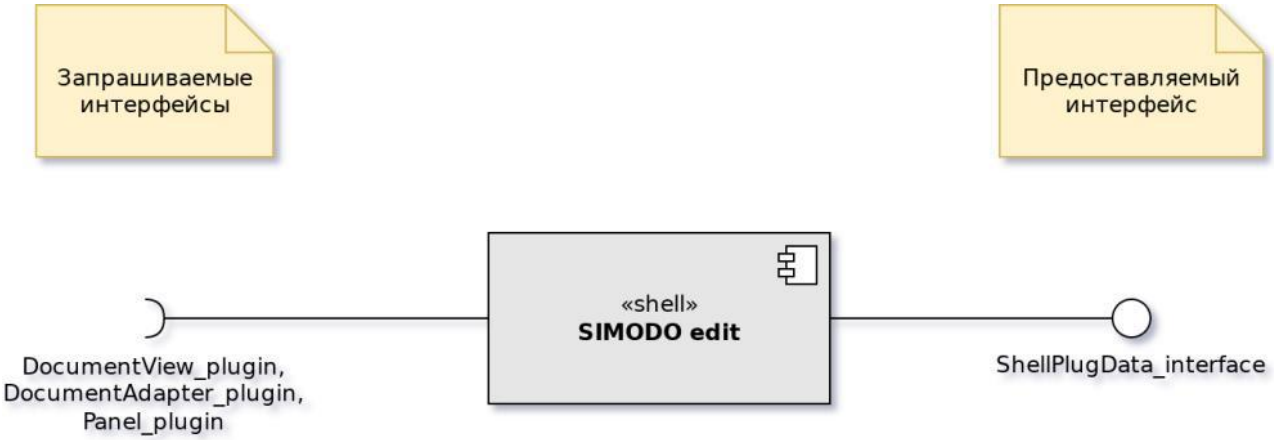

Рисунок 2 — Интерфейсы оболочки

Следующим слоем открытой программной архитектуры являются процессы, выполняющие разнообразные функции, которые целесообразно отделить от редактора межпроцессным взаимодействием. Преимуществом такого отделения является отказоустойчивость всей программы.

Процессы запускаются конкретными плагинами как консольные приложения и обмениваются между ними через каналы — механизм взаимодействия через стандартный ввод-вывод. Возможен как синхронный, так асинхронный обмен.

Примерами выделения в отдельные процессы могут быть: взаимодействие с системами управления версиями (например, git), языковые протоколы (например, LSP[3]; для языков серии SIMODO используется модифицированная версия LSP+[1]), сервер моделирования SIMODO и т.д.

На рисунке 3 представлено взаимодействия слоёв интегрированной среды разработки SIMODO.

Открытая программная архитектура интегрированной среды разработки SIMODO представляет собой вариант многоуровневой архитектуры, где нижележащие слои не зависят от вышележащих, а обмен сообщениями возможет только между соседними слоями.

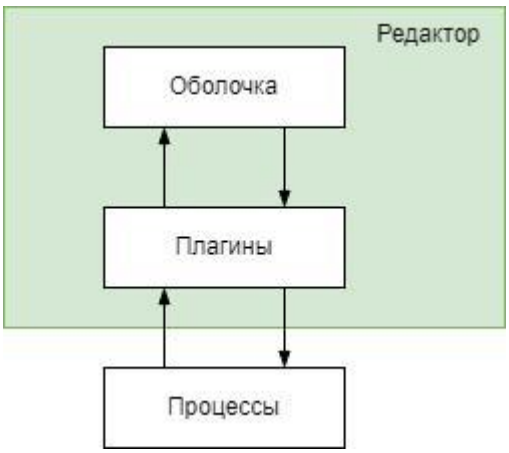

Рисунок 3 — Взаимодействие элементов интегрированной среды разработки SIMODO

Адаптивная система моделирования SIMODO относится к третьему слою архитектуры интегрированной среды разработки SIMODO. Среда должна выполняться в отдельном процессе, с которым плагин расширяемого редактора обменивается сообщениями.

Начало и завершение работы адаптивной системы моделирования SIMODO, а также управление ходом моделирования должны программно контролироваться плагином расширяемого редактора.

Таким образом возникает необходимость выбора инструмента реализации межпроцессного взаимодействия между плагинами и процессами.

1.2 СРАВНИТЕЛЬНЫЙ АНАЛИЗ СУЩЕСТВУЮЩИХ БИБЛИОТЕК И ФРЕЙМВОРКОВ ПОДДЕРЖКИ МЕЖПРОЦЕССНОГО ВЗАИМОДЕЙСТВИЯ

Для выявления достоинств и недостатков существующих библиотек и фреймворков, необходимо провести их сравнительный анализ, на основе которого будет выполнен выбор библиотеки или фреймворка.

Исходя из того, что в качестве целевой операционной системы для интегрированной среды разработки SIMODO является ОС Windows и дистрибутивы Linux[1], при выборе аналогов, будут рассматриваться кросс-платформенные библиотеки и фреймворки, которые разработаны под указанные ОС.

На основании того, что в качестве основного языка разработки интегрированной среды разработки SIMODO используется язык программирования C++[1], при выборе аналогов, будут рассматриваться библиотеки и фреймворки, которые обладают интерфейсом, совместимым с языком программирования C++.

На сегодняшний день существует множество библиотек и фреймворков поддержки межпроцессного взаимодействия. Одними из самых популярных являются [4]:

— Boost;

— eCAL;

— ACE;

— POCO;

— Qt.

BOOST

Библиотека Boost.Process позволяет управлять системными процессами. Она предоставляет следующие возможности[5]:

— Создание дочернего процесса.

— Настройка каналов для дочернего процесса.

— Синхронное и асинхронное взаимодействие с дочерним процессом через каналы.

— Синхронное и асинхронное ожидание завершения процесса.

— Прерывание процесса.

Библиотека Boost.Interprocess снижает сложность использования межпроцессного взаимодействия и синхронизации, а также предлагает ряд механизмов межпроцессного взаимодействия и синхронизации[5]:

— разделяемая память;

— файлы, проецируемые в память;

— семафоры;

— мьютексы;

— условные переменные;

— размещаемые в разделяемой памяти и файлах, проецируемых в память, мьютексы;

— именованные объекты синхронизации;

— блокировка файлов;

— относительные указатели;

— очередь сообщений.

Библиотека Boost.Interprocess также предлагает высокоуровневые межпроцессного механизмы динамического выделения разделяемой памяти или файлов, проецируемых в память. Библиотека не требует компиляции, потому что состоит только из заголовочных файлов.

Фреймворк eCAL (улучшенный абстрактный уровень связи) является связующим программным обеспечением, которое делает возможным масштабируемое, высокопроизводительное межпроцессное взаимодействие на одном вычислительном узле или между несколькими узлами в вычислительной сети[6]. eCAL разработан для характерных сценариев облачных вычислений, когда разные процессы обмениваются данными используя шаблон издатель-подписчик.

Фреймворк eCAL автоматически выбирает лучший механизм передачи данных для каждого соединения:

— разделяемая для связи на одном вычислительном узле;

— UDP протокол для сетевого взаимодействия.

На рисунке 4 показан принцип работы фрейморка eCAL.

Фреймворк eCAL обладает следующими особенностями[6]:

— скорость обмена данными 1–10 Гб/с;

— поддержка шаблона издатель-подписчик и сервер-клиент;

— обмен данных происходит без посредника;

— обладает мощным инструментарием отслеживания и анализа децентрализованных потоков данных;

— не требует настройки перед использование;

— позволяет использовать пользовательский протокол сообщений;

— работает на ОС Windows и Linux.

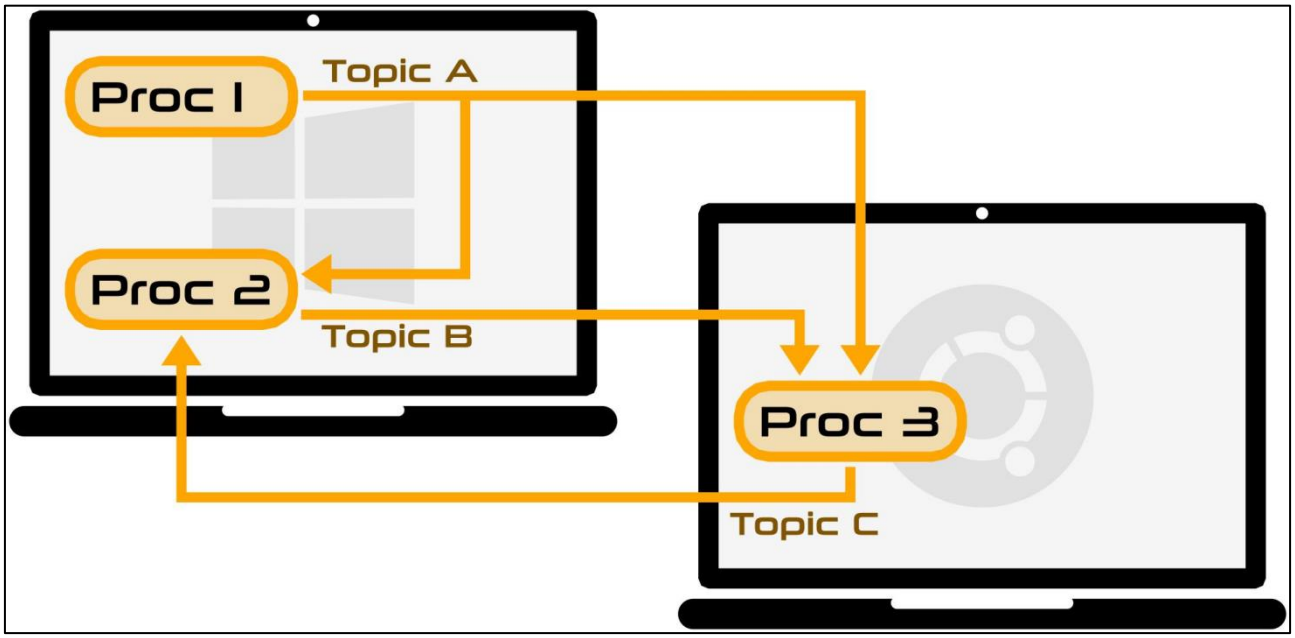

Рисунок 4 — Принцип работы фреймворка eCAL

Фреймворк ACE (адаптивное окружение связи) является объектно-ориентированным фреймворком, который реализует основные шаблоны для программного обеспечения параллельной связи. Фреймворк ACE предоставляет широкий набор C++ фасадов-обёрток, предназначенных для многократного использования, и компонент фреймворка, выполняющих общие для ряда ОС задачи программной связи[7]:

— Демультиплексирование событий и диспетчеризация обработки событий.

— Обработка сигналов.

— Инициализация сервисов.

— Межпроцессное взаимодействие.

— Управление разделяемой памятью.

— Маршрутизация сообщений.

— Динамическая настройка распределённых сервисов.

— Параллельное выполнение и синхронизация.

Фреймворк ACE предназначен для разработчиков высокопроизводительных сервисов связи и приложений реального времени[7]. Фреймворк упрощает разработку объектно-ориентированных сетевых

приложений и сервисов, использующие межпроцессное взаимодействие, демультиплексирование событий, явное динамическое связывание и параллелизм. В дополнение, Фреймворк ACE автоматизирует системную настройку с помощью сервисов динамического связывания во время выполнения приложений и выполняет данные сервисы в одном или нескольких процессах или потоках.

Диаграмма на рисунке 5 иллюстрирует ключевые компоненты в фреймворке ACE и их иерархические связи.

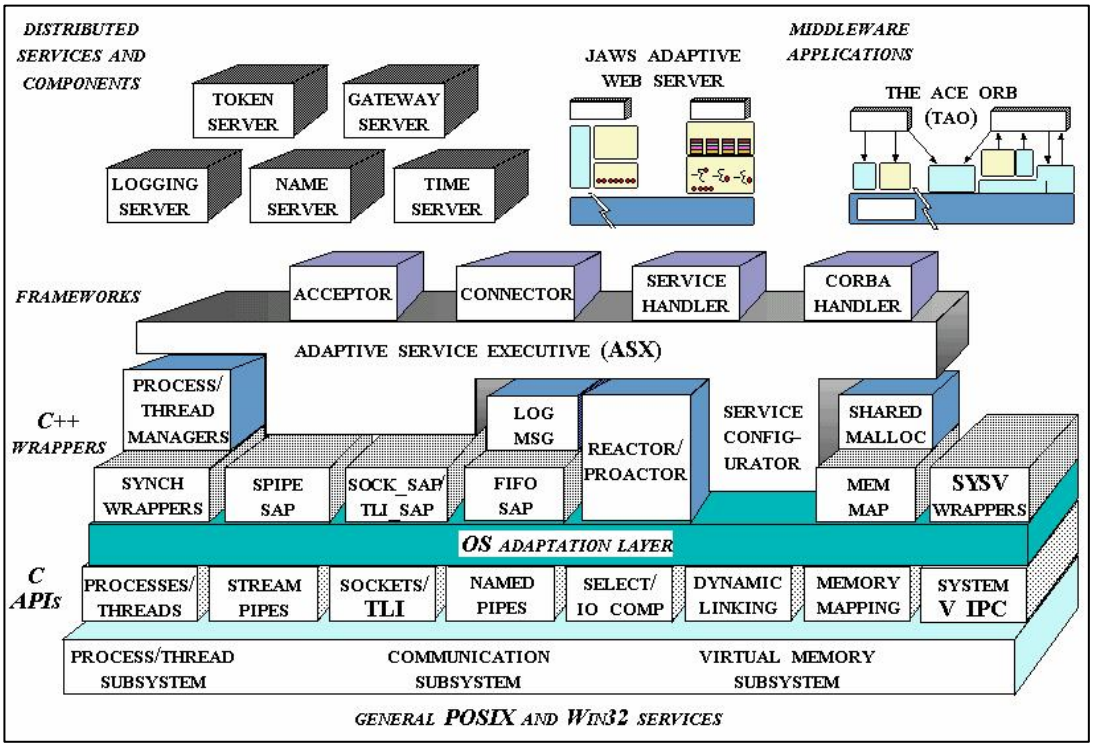

Рисунок 5 — Структура и функциональность фреймворка ACE

Библиотеки POCO являются кросс-платформенными библиотеками для создания ориентированных на сеть или интернет приложений, которые работают на персональных компьютерах, серверах, мобильных устройствах, устройствах интернета вещей и встроенных системах[8].

Библиотека POCO Process позволяет:

— Получить информацию о текущем процессе.

— Запустить новый процесс.

— Прервать другой процесс.

— Перенаправить ввод-вывод запускаемого процесса.

— Синхронизировать процессы, используя примитивы синхронизации: именованный мьютекс и именованное событие.

— Использовать разделяемую память.

Библиотека POCO является кросс-платформенной, модульной и масштабируемой библиотекой, используемой в программах корпоративного уровня[8]. Библиотека обеспечивает целостный и простой для понимания программный интерфейс.

Исходные тексты библиотеки POCO обладают особым качеством кода, читаемостью, полнотой, согласованностью и тестируемостью.

Состав библиотеки POCO представлен на рисунке 6.

Фреймворк Qt является кросс-платформенным программным обеспечением для создания как графических пользовательских интерфейсов, так и кросс-платформенных приложений, запускаемых на различном программном и аппаратном обеспечении, например, операционные системы Linux, Windows, macOS, Android или встраемые системы с минимальными изменениями нижележащей кодовой базы, оставаясь нативным приложением с соответствующими возможностями и скоростью[9].

Фреймворк Qt предоставляет несколько кросс-платформенных способов реализации межпроцессного взаимодействия[10]:

— Использование сетевого TCP/IP межпроцессного взаимодействия в вычислительной сети.

— Использование локального сервера и сокетов для межпроцессного взаимодействия на одном вычислительном узле.

— Разделяемая память.

— Класс QProcess позволяет запускать внешние программы в качестве дочерних процессов и взаимодействовать с ними.

— Управление сессиями, позволяющее распространять события между процессами.

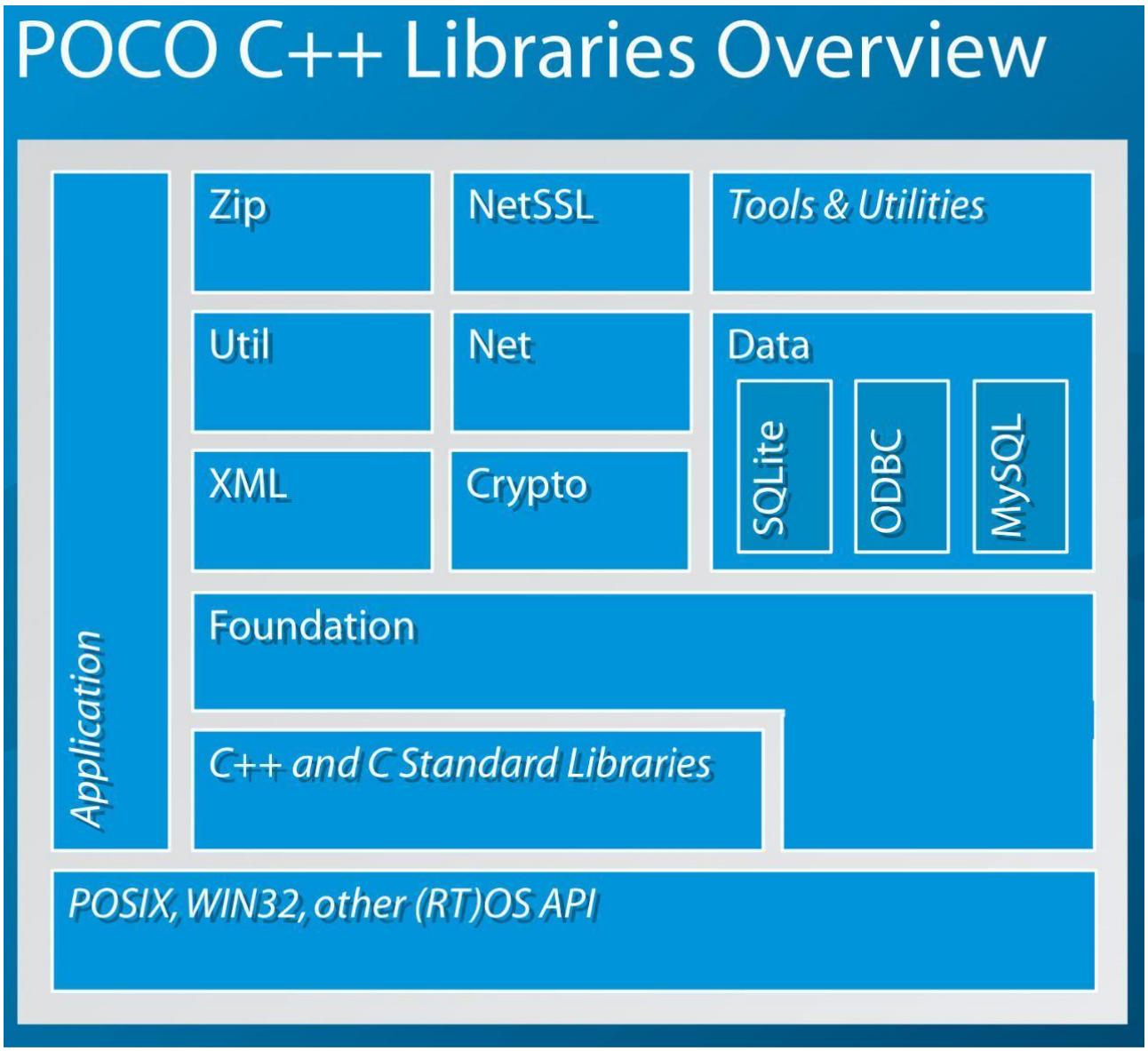

Рисунок 6 — Состав библиотеки POCO.

Терминология для каналов в фреймворке Qt может ввести в заблуждение[10]. Выходные каналы процесса являются каналами чтения, а входные каналы процесса являются каналами записи. Данная терминология обусловлена тем, что из процесса читается результат его работы, а запись в процесс становится его входом.

Фреймворк Qt позволяет перенаправить только один выходной канал.

Интегрированная среда разработки SIMODO разрабатывается с использованием фреймворка Qt[1], поэтому механизм плагинов реализуется с использованием данного фреймворка. Каждый плагин, расширяющий функциональность оболочки будет зависеть от небольшой части фреймворка Qt, связанного с системой плагинов.

Состав фреймворка Qt представлен на рисунке 7.

Анализа данных библиотек и фреймворков показывает, что инструменты реализации межпроцессного взаимодействия предоставляют широкий выбор методов межпроцессного взаимодействия. Однако для поставленной задачи нет необходимость в использовании всего инструментария тяжеловесных библиотек. Выделим критерии выбора:

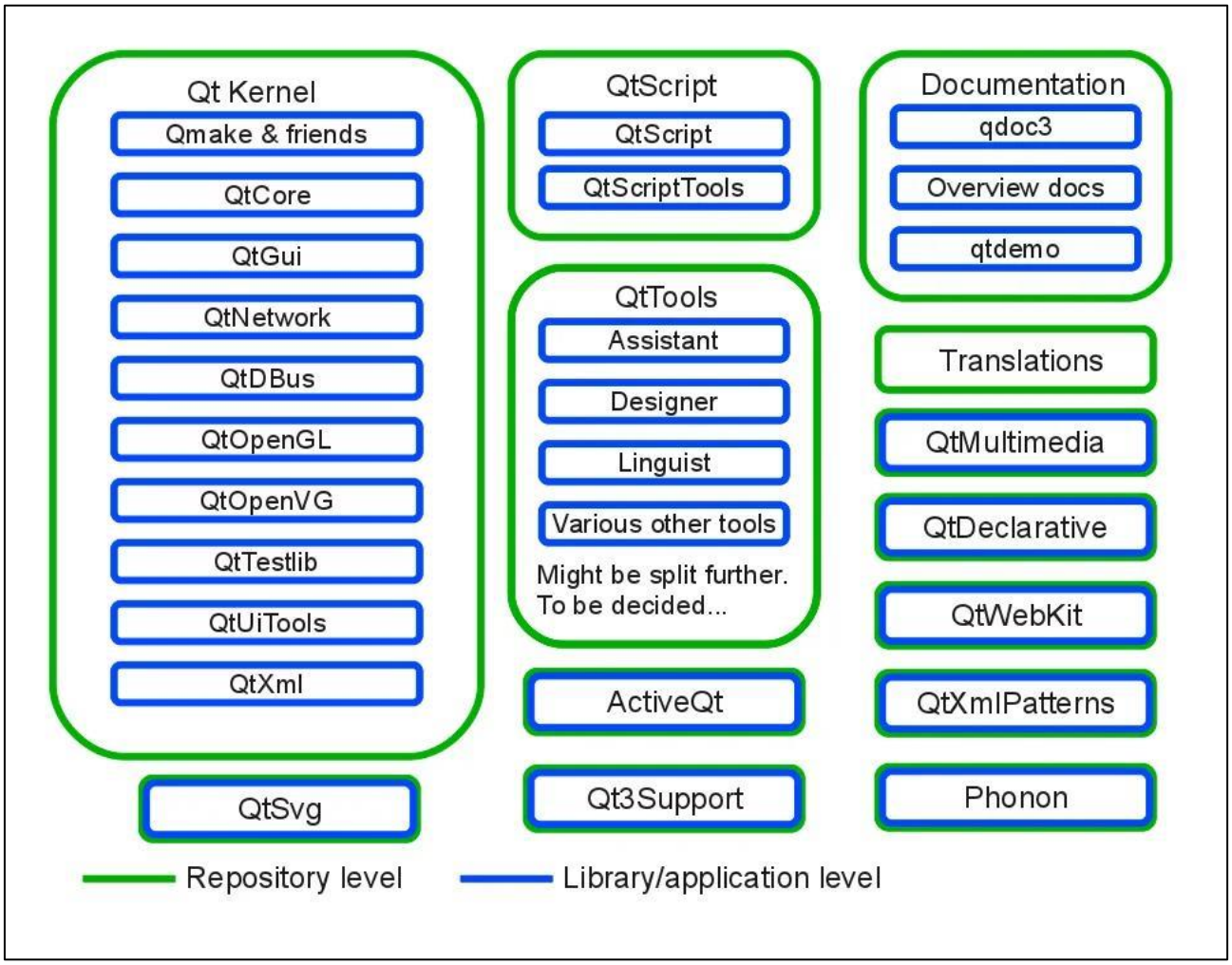

Рисунок 7 — Состав фреймворка Qt.

— Возможность программного создания процесса: интегрированная среда разработки должна самостоятельно запускать сторонние процессы, чтобы этого не приходилось делать пользователю.

— Взаимодействие с процессом через каналы[10]: каналы являются наиболее простым способом взаимодействия с процессом, что позволит ограничить работу адаптивной системы моделирования SIMODO с вводом-выводом до работы со стандартными потоками ввода-вывода.

— Программное завершение процесса: интегрированная среда разработки SIMODO должна иметь возможность преждевременно завершить работу внешних процессов, не дожидаясь завершения работы этих процессов в случае завершения работы среды.

— Библиотека или фреймворк должен состоять только из заголовочных файлов C++, чтобы исключить издержки, связанные с поставкой правильной версией библиотек времени выполнения программы.

— Размер полностью скомпилированной библиотеки должен быть минимален, чтобы итоговый размер приложения занимал как можно места в постоянном запоминающем устройстве пользователя.

— По выделенным критериям проведён сравнительный анализ библиотек, который продемонстрирован в таблице 1

Таблица 1 — сравнительный анализ библиотек поддержки межпроцессного взаимодействия

Библиотека	Программное создание процесса	Взаимодействие через каналы	Программное прерывание процесса	Состоит только из заголовочных файлов	Размер библиотеки, Мб
Boost	+	+	+	+	3
Qt	+	+	+	–	15
POCO	+	+	+	–	8
ACE	+	–	+	–	89
eCal	–	–	–	–	112

Сравнение приложений по выделенным критериям выявило, что среди предложенных библиотек и фреймворков поддержки межпроцессного взаимодействия библиотека. Библиотека Boost обладает самыми лучшими показателями относительно остальных кандидатов.

1.3 ПРОТОКОЛ ВЗАИМОДЕЙСТВИЯ С АДАПТИВНОЙ СИСТЕМОЙ МОДЕЛИРОВАНИЯ

Архитектура интегрированной среды разработки SIMODO предполагает три уровня:

— Расширяемый редактор — оболочка, задача которой состоит в обеспечении механизма расширения и базовой функциональности (например, отображение окна программы и базовые действия работы с файлами).

— Плагины — расширения оболочки, которые обеспечивают основную функциональность (например, возможность редактировать текстовые файлы или работать с файловой системой).

— Процессы — внешние процессы, выполняющие разнообразные функции (например лексический, синтаксический и семантический анализ или выполнение моделирования), которые разумно изолировать от оболочки межпроцессным взаимодействием.

На рисунке 8 приведён пример представления каждого слоя архитектуры интегрированной среды разработки SIMODO.

В архитектуре интегрированной среды разработки SIMODO нижние уровни независимы от верхних, а взаимодействие возможно только между смежными уровнями.

Адаптивная система моделирования SIMODO принадлежит к третьему уровню в архитектуре интегрированной среды разработки SIMODO. Среда выполняется в отдельном процессе, с которым взаимодействует плагин расширяемого редактора для передачи исходных текстов моделей, скриптов запуска моделирования или других программных модулей. В свою очередь, среда отправляет информацию о состоянии моделей и ходе моделирования. Среде может понадобится дополнительная информация (например, модель использует внешние модули), поэтому она может запросить дополнительные ресурсы у плагина (например, исходные тексты внешних модулей).

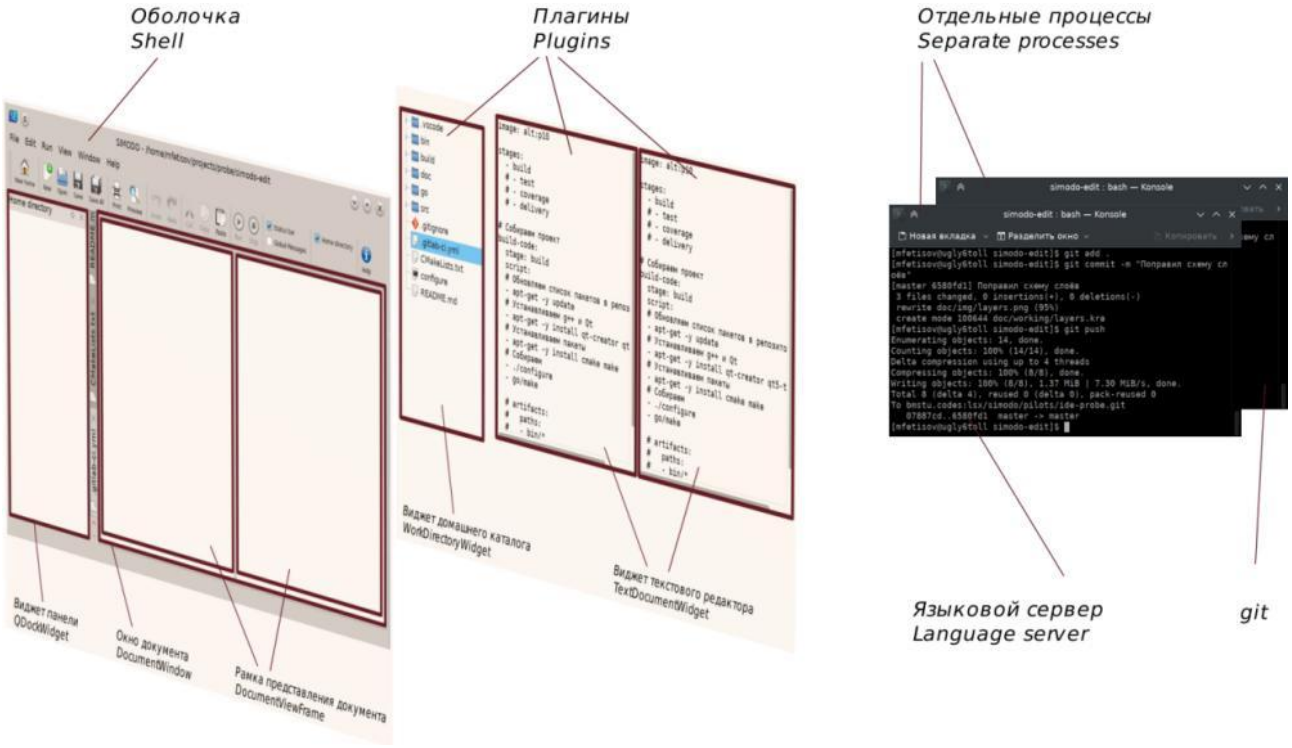

Рисунок 8 — Архитектура интегрированной среды разработки SIMODO

Запуск и остановка адаптивной системы моделирования SIMODO, запуск, временная и полная остановка моделирования должны программно контролироваться плагином расширяемого редактора.

Таким образом, можно выделить следующие сообщения, передаваемые между плагином расширяемого редактора и процессом адаптивной системы моделирования SIMODO:

— init: инициализация адаптивной системы моделирования SIMODO.

— info: запрос информации о текущем состояния адаптивной системы моделирования SIMODO

— start: запуск моделирования с передачей исходного текста сценария, описывающего ход моделирования.

— request: запрос дополнительного ресурса системой моделирования.

— process: передача исходного текста модуля.

— send: передача данных.

— pause: временная остановка моделирования.

— resume: продолжение моделирования.

— stop: полная остановка моделирования.

— terminate: остановка адаптивной системы моделирования SIMODO

На рисунке 9 изображена диаграмма последовательности сообщений между плагином и адаптивной системой моделирования SIMODO (сервером) при нормальном ходе моделирования, т.е. без возникновения ошибок при обмене сообщениями.

Основная концепция взаимодействия между процессом плагина и процессом сервера моделирования заключается в том, что после запуска процесса, на котором работает сервер моделирования, плагин инициирует запуск, управление и остановку моделирования, в свою очередь предоставляя серверу необходимые ресурсы.

В первую очередь плагин запускает процесс сервера моделирования, после чего плагин должен отправить сообщение инициализации серверу. О готовности сервера плагин может узнать отправив запрос на получение информации о состоянии сервера. В ответ на запрос сервер отправит запрашиваемую информацию стандартным сообщением данных.

Моделирование запускается на инициализированном сервере моделирования сообщением start и передаётся сценарий моделирования. После этого нормальным считается следующее поведение:

— Запрос сервером моделирования дополнительных ресурсов и отправка соответствующих ресурсов плагином.

— Отправка плагину данных о ходе моделирования.

— Отправка серверу моделирования команд управления, которые обрабатываются в соответствии со сценарием моделирования.

— Временная приостановка хода моделирования и возобновления хода моделирования.

Завершение хода моделирования возможно в соответствии с логикой сценария моделирования (не приведено на рисунке 9) или при получении сообщения stop.

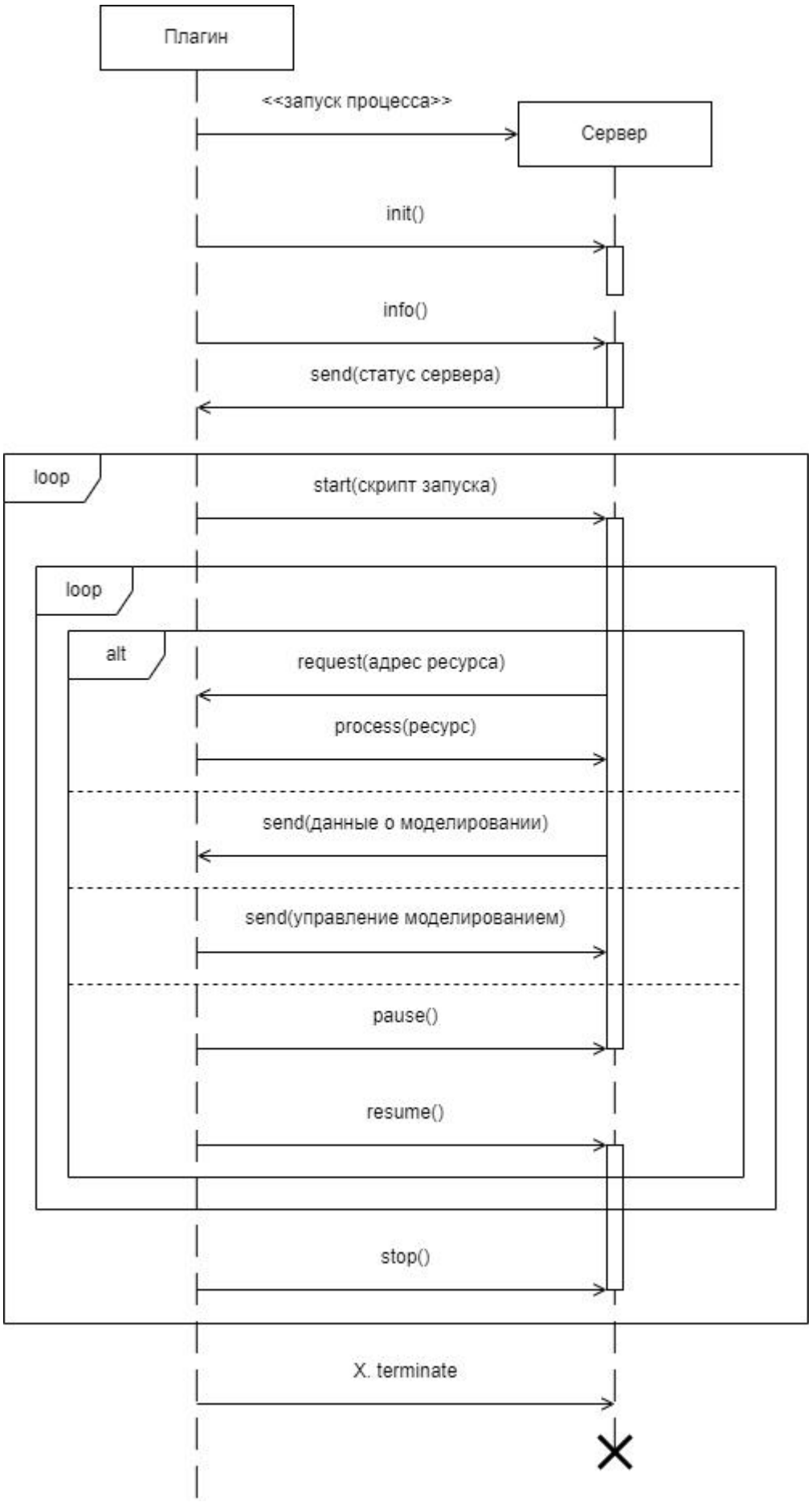

Рисунок 9 — Пример нормального хода моделирования

Сервер моделирования позволяет проводить моделирование неограниченное число раз в общем случае с разными сценариями моделирования.

1.4 ВЫБОР МЕТОДОВ РЕШЕНИЯ СИСТЕМ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

При математическом моделировании ряда технических устройств используются системы дифференциальных нелинейных уравнений. Такие модели используются не только в технике, они находят применение в экономике, химии, биологии, медицине, управлении[11].

Исследование функционирования таких устройств требуют решения указанных систем уравнений. Поскольку основная часть таких уравнений являются нелинейными и нестационарными, часто невозможно получить их аналитическое решение.

Возникает необходимость использовать численные методы, наиболее известными из которых является метод Рунге-Кутта и метод Фельберга с переменным шагом.

Классический метод Рунге-Кутта обладает фиксированным шагом и точностью четвёртого порядка, но он прост в реализации. Метод Фельберга обладает точностью пятого порядка и переменным шагом, что усложняет формулы, однако приводит к более точным результатам[11–12].

Было решено, что в реализации модуля сцены будет использоваться классический метод Рунге-Кутта, потому что он проще в реализации, а разница в точности между четвёртым и пятым порядком не носит критический характер.

Для одного дифференциального уравнения n–го порядка, задача Коши состоит в нахождении функции, удовлетворяющей равенству[11] — формуле (1).

$$y^{(n)} = f(t, y', \ldots, y^{(n-1)}) \qquad (1)$$

Искомая функция также должна удовлетворять начальным условиям — формула (2).

$$y(t_0) = y_1^0, y'(t_0) = y_2^0, \ldots, y^{(n-1)}(t_0) = y_n^0 \quad (2)$$

Перед решением эта задача должна быть переписана в виде следующей СДУ[11] — формула (3).

$$\begin{cases} \frac{dy_1}{dt} = f_1(y_1, y_2, \ldots, y_n, t) \\ \frac{dy_2}{dt} = f_2(y_1, y_2, \ldots, y_n, t) \\ \cdots \\ \frac{dy_n}{dt} = f_n(y_1, y_2, \ldots, y_n, t) \end{cases} \quad (3)$$

Начальные условия СДУ должны соответствовать формуле (4).

$$y_1(t_0) = y_1^0, y_2(t_0) = y_2^0, \ldots, y_n(t_0) = y_n^0 \quad (4)$$

По методу Рунге—Кутта четвёртого порядка общее решение [11–12] имеет вид формулы (5).

$$y^{n+1} = y^n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \quad (5)$$

где расчётные коэффициенты [12] соответствуют формулам (6)–(9).

$$k_1 = \tau f(t_n, y^n) \quad (6)$$

$$k_2 = \tau f\left(t_n + \frac{\tau}{2}, y^n + \frac{1}{2}k_1\right) \quad (7)$$

$$k_3 = \tau f\left(t_n + \frac{\tau}{2}, y^n + \frac{1}{2}k_2\right) \quad (8)$$

$$k_4 = \tau f(t_n + \tau, y^n + k_3) \quad (9)$$

1.5 ВЫВОДЫ

Сравнительный анализ библиотек и фреймворков поддержки межпроцессного взаимодействия, показал, что наиболее подходящей библиотекой для организации межпроцессного взаимодействия между плагином

интегрированной среды разработки SIMODO и сервером адаптивной системы моделирования SIMODO является библиотека Boost.

На основе описания открытой архитектуры интегрированной среды разработки SIMODO разработан протокол взаимодействия с адаптивной системой моделирования SIMODO.

Рассмотрены численные методы численного интегрирования для решения систем дифференциальных уравнений. Выбран классический метод Рунге-Кутта четвёртого порядка точности.

2 РАЗРАБОТКА ПРОГРАММНОЙ ПОДСИСТЕМЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ПРОЦЕССОВ

2.1 ВЫБОР ТЕХНОЛОГИИ И ЯЗЫКА ПРОГРАММИРОВАНИЯ

В качестве технологии программирования было выбрано объектно-ориентированное программирование. ООП является основой всех современных приложений и имеет удобное и практическое применение. При использовании этого метода вся программа разбивается на объекты, с каждым из которых работа происходит по отдельности, что позволяет в будущем расширять программный продукт путем добавления новых объектов.

Для написания настольных приложений существует множество языков программирования. Наиболее популярные[13] – это C#, C++ и Python, которые являются кроссплатформенными языками.

Для разрабатываемого приложения был выбран язык C++, так как он используется в разработке адаптивной среды разработки SIMODO[14].

2.2 ВЫБОР ПОДХОДА РАЗРАБОТКИ

В качестве жизненного цикла разработки была выбрана спиральная модель. Этот метод позволяет в конце каждого цикла иметь работающий продукт, который можно продемонстрировать[15]. Это дает возможность своевременно оценить и протестировать продукт, чтобы сразу вносить какие-либо правки и исправлять ошибки, не дожидаясь окончания разработки. Нахождение багов на каждом этапе позволяет избежать «волнового» исправления ошибок. А также этот метод позволяет детальнее подойти к каждому этапу разработки по отдельности

Также при разработке было решено использовать нисходящий подход[16], реализуя сначала модули верхнего уровня (интерфейс пользователя), а после переходя к модулям нижнего уровням (логика работы программы).

2.3 РАЗРАБОТКА СХЕМЫ СТРУКТУРНОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Структурная схема программного продукта показывает разделение программы на её главные составляющие. На основе анализа технического

задания, в разрабатываемом приложении, которое выполняет имитационное моделирование, выявлено две подсистемы:

— подсистема интеграции с адаптивной средой разработки: подсистема интеграции с адаптивной средой разработки преобразует входные данные в понятный для сервера моделирования вид и преобразует исходящие от сервера моделирования данные в приемлемый для среды разработки вид; включает в себя две подсистемы:

1) в подсистеме ввода вывода происходит работа с процессом, в котором работает сервер моделирования;

2) в подсистеме обработки сообщений происходит обработка сообщений, приходящих от сервера;

— подсистема имитационного моделирования включает в себя три подсистемы:

1) в подсистеме ввода вывода происходит получение и отправка данных в рамках протокола сервера моделирования;

2) в подсистеме обработки сообщений происходит обработка сообщений, приходящих от редактора;

3) подсистема интерпретации отвечает за процесс имитационного моделирования в рамках интерпретатора scriptC0; включает в себя три подсистемы: интерпретатор, менеджер модулей и модуль сцены.

Рассмотрим подробнее рассмотрим подсистему интерпретации:

— интерпретатор отвечает за исполнение скриптов на языке scriptC0

— менеджер модулей обеспечивает интерпретатор исходными текстами модулей, запрашиваемых в процессе интерпретации

— модуль сцены является модулем scriptC0; отвечает за расчёт моделей в виде систем дифференциальных уравнений с помощью численных методов.

На основе выявленных подсистем была составлена структурная схема ПП, показанная на рисунке 10.

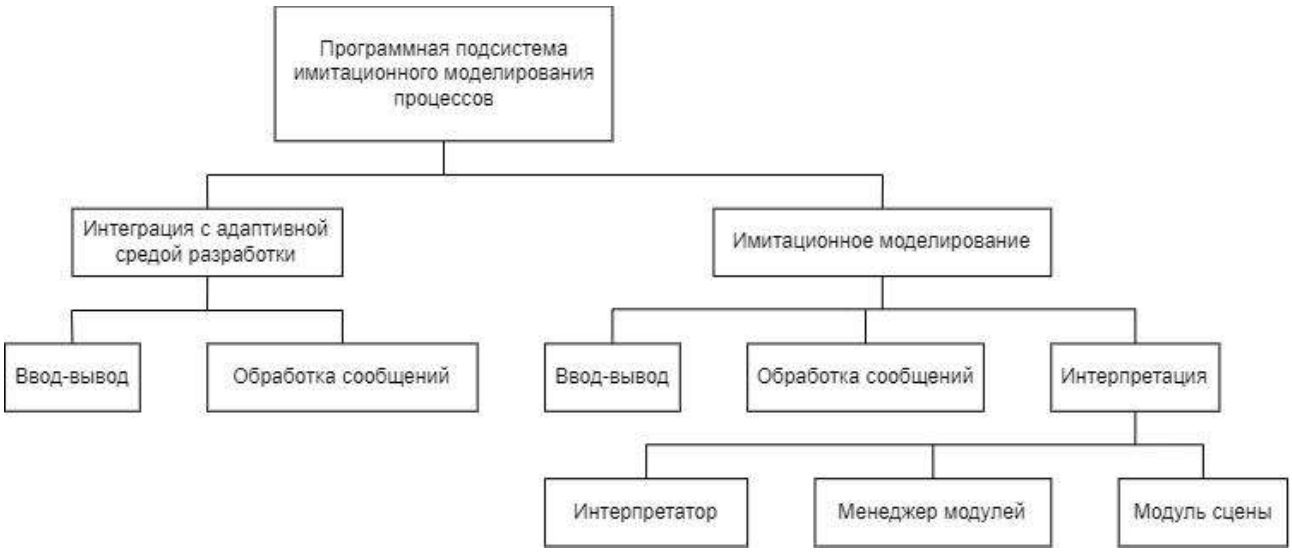

Рисунок 10 – Структурная схема информационной системы

2.4 РАЗРАБОТКА ДИАГРАММ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Сервер имитационного моделирования принадлежит к третьему уровню в архитектуре интегрированной среды разработки SIMODO. Сервер выполняется в отдельном процессе, с которым взаимодействует плагин расширяемого редактора для передачи исходных текстов моделей, скриптов запуска моделирования или других программных модулей. В свою очередь, сервер отправляет информацию о состоянии моделей и ходе моделирования. Серверу может понадобится дополнительная информация (например, модель использует внешние модули), поэтому он может запросить дополнительные ресурсы у плагина (например, исходные тексты внешних модулей). Запуск и остановка сервера имитационного моделирования SIMODO, запуск, временная и полная остановка моделирования должны программно контролироваться плагином расширяемого редактора.

Таким образом, можно выделить следующие сообщения, передаваемые между плагином расширяемого редактора и процессом сервера имитационного моделирования SIMODO:

— Инициализация с грамматиками языков программирования в качестве аргументов: инициализация сервера имитационного моделирования SIMODO с предоставление исходных текстов грамматик языков программирования, используемых во время имитационного моделирования.

— Запуск моделирования с основным сценарием в качестве аргумента: запуск интерпретации исходного текста основного сценария моделирования

— Запрос внешнего ресурса по его адресу: сигнал адаптивной среде разработке о необходимости в исходном тексте модуля, который указан в основном сценарии моделирования или в уже полученных модулях.

— Отправка исходного текста ресурса: передача серверу имитационного моделирования исходного текста модуля, который был запрошен по определённому адресу.

— Установка пары ключ-значение: передача информации серверу имитационного моделирования в процессе имитационного моделирования для управления ходом имитационного моделирования.

— Сообщение информации: передача информации адаптивной среде разработки о ходе имитационного моделирования.

— Приостановка моделирования: сигнал серверу имитационного моделирования об обратимом прерывании хода имитационного моделирования.

— Возобновление моделирования: сигнал серверу имитационного моделирования о возобновлении хода имитационного моделирования.

— Остановка: сигнал серверу имитационного моделирования о необратимом прерывании хода имитационного моделирования.

— Остановка сервера: сигнал серверу имитационного моделирования о прерывании работы сервера имитационного моделирования; требуется ожидание момента завершения процесса, в котором исполняется сервер имитационного моделирования.

На рисунке 11 изображена диаграмма последовательности[17] сообщений между плагином и сервером имитационного моделирования SIMODO при нормальном ходе моделирования, т.е. без возникновения ошибок при обмене сообщениями.

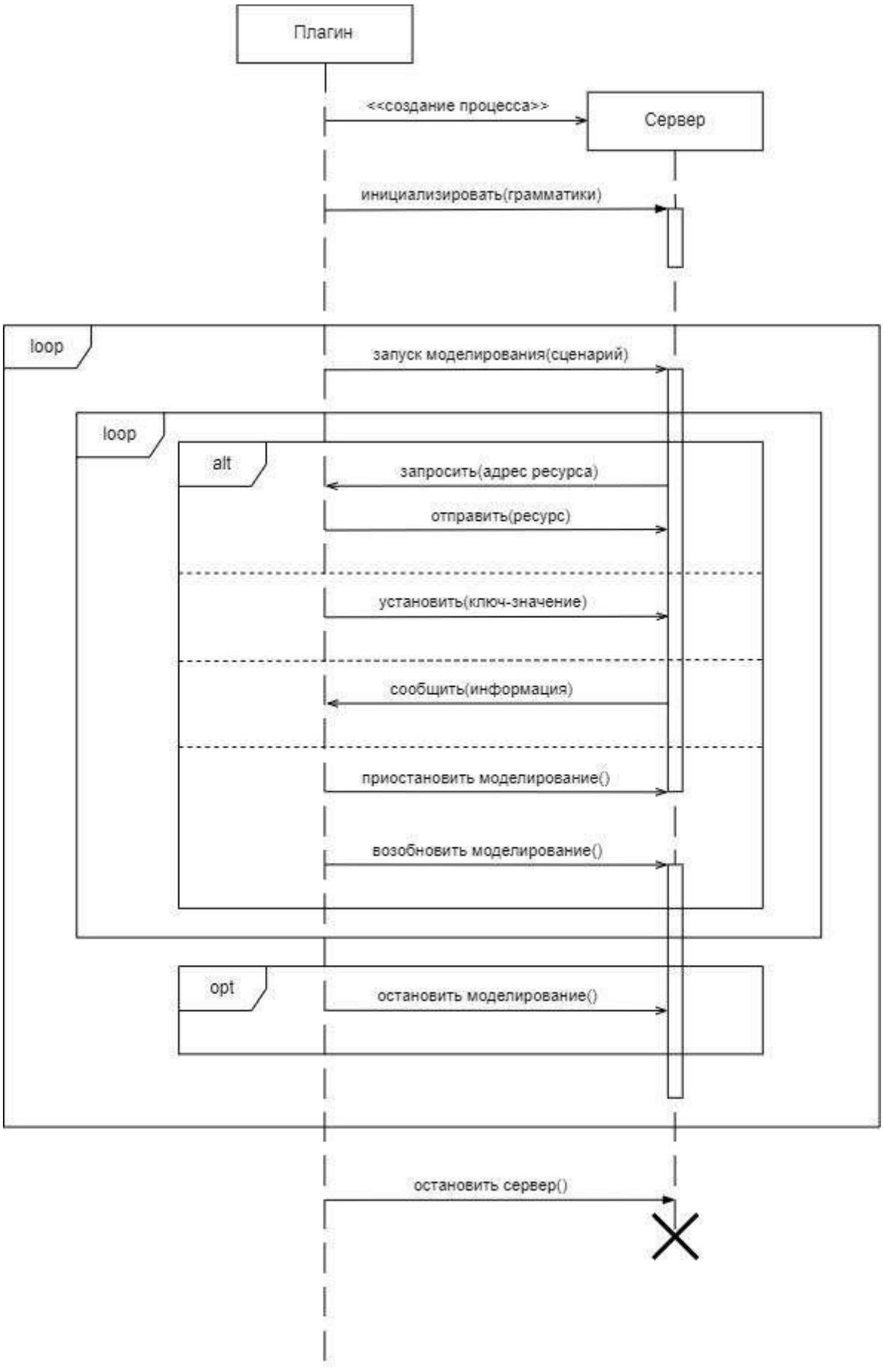

Рисунок 11 – Диаграммы последовательностей

Основная концепция взаимодействия между процессом плагина и процессом сервера моделирования заключается в том, что после запуска процесса, на котором работает сервер моделирования, плагин инициирует запуск, управление и остановку моделирования, в свою очередь предоставляя серверу необходимые ресурсы.

В первую очередь плагин запускает процесс сервера моделирования, после чего плагин должен отправить сообщение инициализации серверу. О готовности сервера плагин может узнать отправив запрос на получение информации о состоянии сервера. В ответ на запрос сервер отправит запрашиваемую информацию стандартным сообщением данных.

Моделирование запускается на инициализированном сервере моделирования и передаётся сценарий моделирования. После этого нормальным считается следующее поведение:

— Запрос сервером моделирования дополнительных ресурсов и отправка соответствующих ресурсов плагином.

— Отправка плагину данных о ходе моделирования.

— Отправка серверу моделирования команд управления, которые обрабатываются в соответствии со сценарием моделирования.

— Временная приостановка хода моделирования и возобновления хода моделирования. Завершение хода моделирования возможно в соответствии с логикой сценария моделирования (не приведено на рисунке 11) или при получении сообщения остановки моделирования.

Сервер моделирования позволяет проводить моделирование неограниченное число раз в общем случае с разными сценариями моделирования.

2.5 РАЗРАБОТКА ДИАГРАММ КЛАССОВ ПРЕДМЕТНОЙ ОБЛАСТИ

После окончания разработки ПО была разработана диаграмма классов предметной области, чтобы представить созданные классы с их полями и методами. Диаграмма классов[18] сервера имитационного моделирования

показана на рисунке 12. Диаграмма классов модуля сцены показана на рисунке 13.

Основные классы сервера имитационного моделирования:

— Сервер — класс, организующий конфигурацию сервера имитационного моделирования.

— СервисВводаВывода — интерфейс, обеспечивающий получение и отправку сообщений через абстрактный канал.

— ОбработчикСобытий — интерфейс, позволяющий наращивать функционал сервера созданием нового объекта.

— Интерпретатор — интерфейс, обеспечивающий ход имитационного моделирования.

Основные классы модуля сцены:

— Сцена — класс, отвечающий за управление имитационным моделированием и обеспечением режима реального времени, если он активирован.

— Актор — интерфейс участника сцены; позволяет потенциально использовать любые модели.

— Модель — класс, отвечающий за представление системы дифференциальных уравнений в понятном для сцене виде.

— Композиция — класс, отвечающий за объединение моделей в единицу, которую можно трактовать как отдельную модель; позволяет выстраивать иерархическую структуру сложных моделей.

2.6 РАЗРАБОТКА СХЕМЫ АЛГОРИТМОВ МОДУЛЯ СЦЕНЫ

Одной из главных задач модуля является пересчёт всех акторов, а при активации режима реального времени — выполнение пересчёта акторов с адекватной реальному времени задержкой.

Для решения данных задач были разработаны схемы алгоритмов[17] модуля сцены, которые представлены на рисунках 14-15.

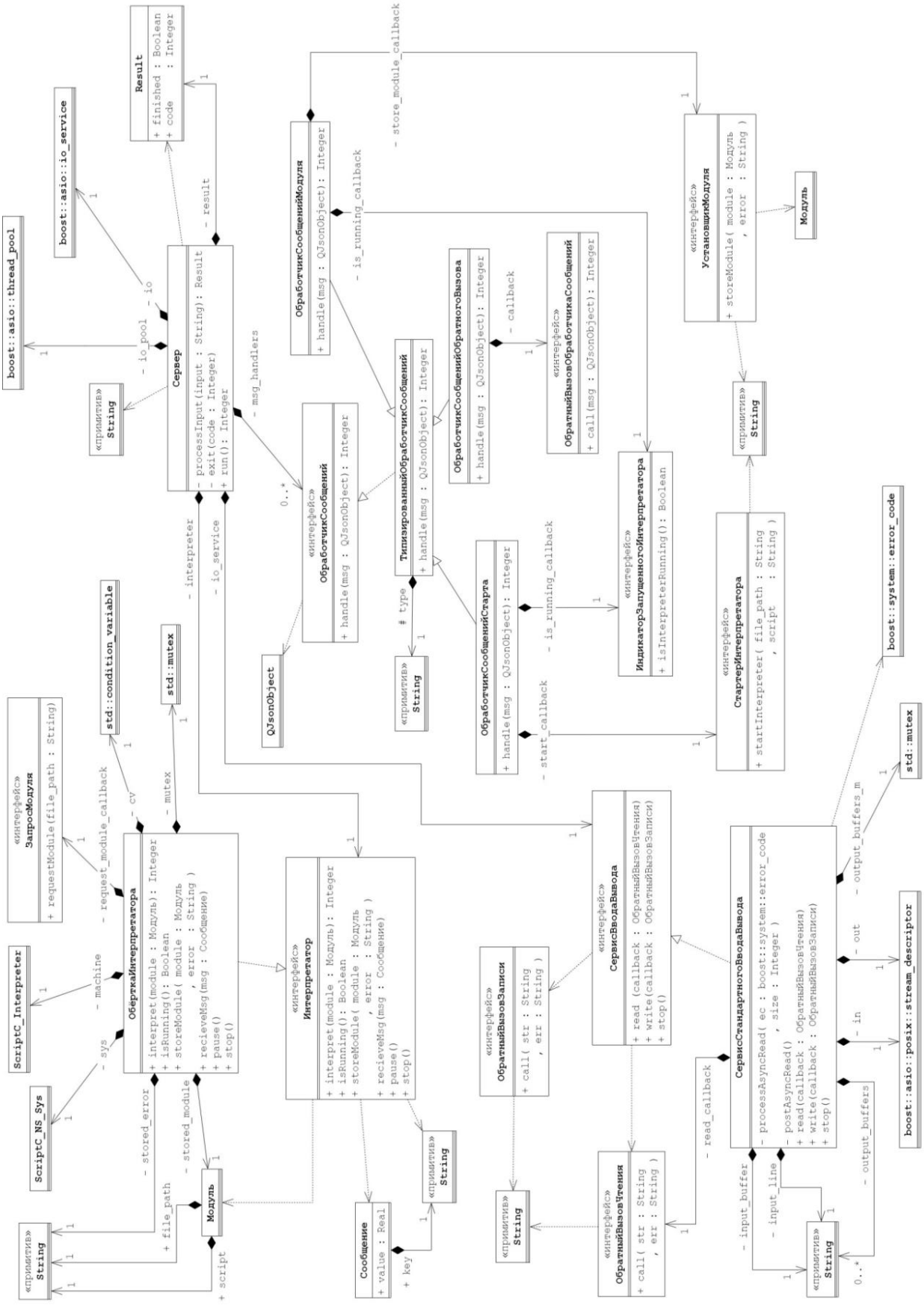

Рисунок 12 – Диаграммы классов сервера имитационного моделирования

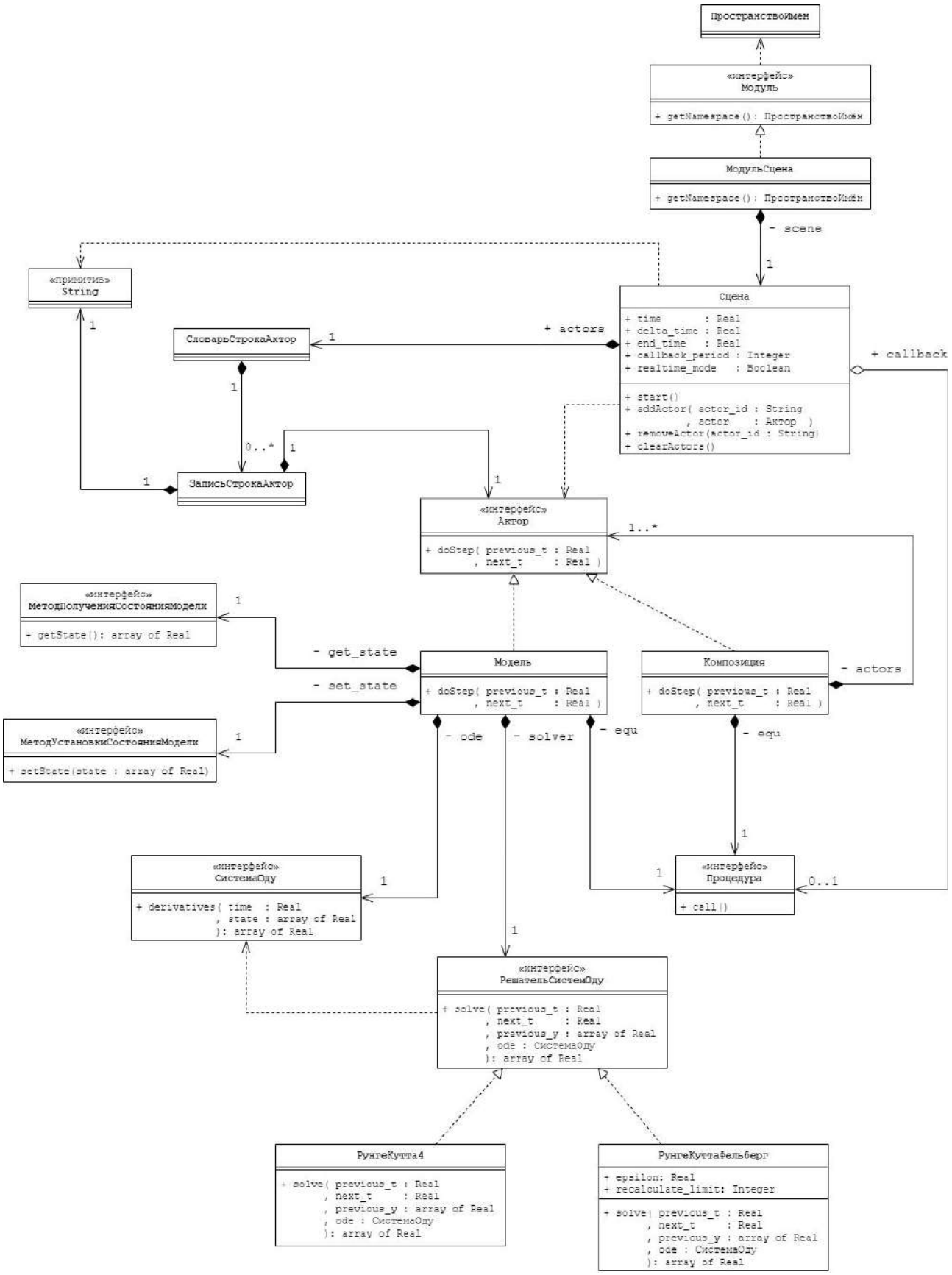

Рисунок 13 – Диаграммы классов модуля сцены

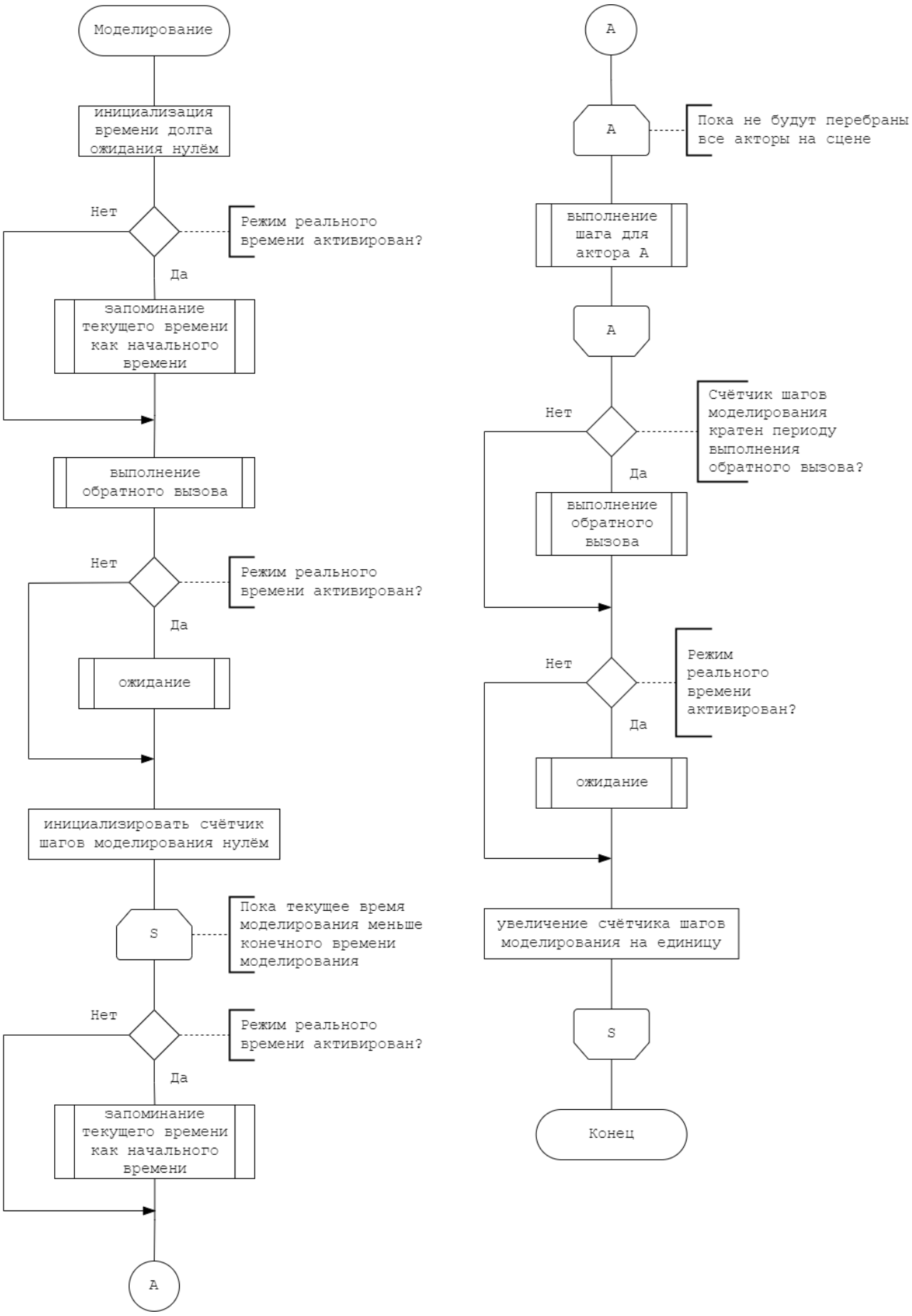

Рисунок 14 – Схемы алгоритмов модуля сцены

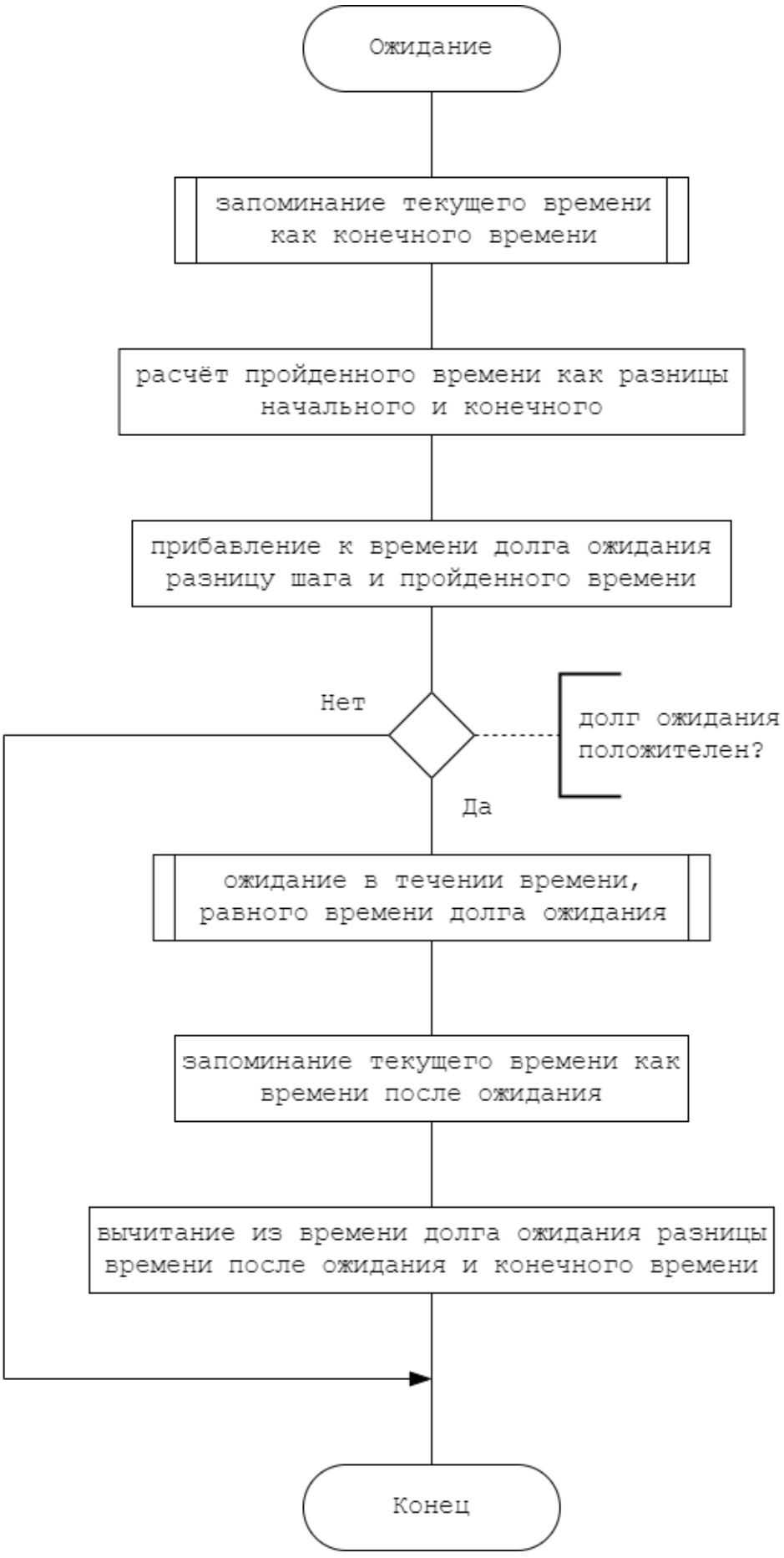

Рисунок 15 – Схемы алгоритмов модуля сцены

Первая схема алгоритма показывает процесс пересчёта акторов в обычном режиме работы сцены. Вторая схема алгоритма демонстрирует пересчёт акторов с учётом режима реального времени.

Алгоритм работы модуля сцены в обычном режиме работы приложения включает в себя следующие шаги:

— Получение списка всех акторов в сцене.

— Проверка состояния каждого актора и выполнение необходимых операций для изменения его состояния.

— Обновление позиции каждого актора на основе его текущего состояния.

— Обновление состояния сцены.

Алгоритм работы модуля сцены в режиме реального времени включает в себя дополнительные шаги, связанные с учётом задержки:

— Получение списка всех акторов в сцене.

— Проверка состояния каждого актора и выполнение необходимых операций для изменения его состояния.

— Расчёт времени задержки на основе текущего времени и времени последнего обновления сцены.

— Обновление позиции каждого актора на основе его текущего состояния и времени задержки.

— Обновление состояния сцены с учётом времени задержки.

Таким образом, алгоритм работы модуля сцены обеспечивает корректное функционирование моделирования в обычном режиме и в режиме реального времени с учётом задержки.

Данный алгоритм использует обратную связь для компенсации ошибок, возникающих из-за неточности ожидания времени задержки. Обратная связь - это процесс, при котором информация о результатах выполнения действий передаётся обратно к исходной системе, чтобы она могла адаптироваться и улучшать свою производительность. В данном случае, если задержка оказалась меньше, чем ожидалось, то алгоритм увеличивает задержку на следующей итерации, чтобы компенсировать ошибку и сохранить стабильность работы

системы. Если задержка оказалась больше, чем ожидалось, то алгоритм уменьшает задержку на следующей итерации, чтобы также компенсировать ошибку и сохранить стабильность работы системы.

Таким образом, алгоритм использует информацию о предыдущих ошибках, чтобы адаптироваться к текущей ситуации и уменьшить влияние ошибок на работу системы. Это позволяет улучшить производительность и стабильность работы системы в условиях изменяющейся нагрузки и темпа реального времени.

2.7 ВЫБОР СТРАТЕГИИ ТЕСТИРОВАНИЯ И РАЗРАБОТКА ТЕСТОВ

В процессе разработки программной подсистемы имитационного моделирования процессов была тщательно продумана стратегия тестирования, которая включает в себя методы структурного контроля и модульного тестирования. Этот выбор обосновывается тем, что структурный контроль позволяет выявлять ошибки кодирования еще на ранних этапах разработки программного продукта. Это в свою очередь сокращает время на их исправление и предотвращает возможные проблемы в будущем.

Модульное тестирование, в свою очередь, дает возможность проверять работу небольших частей программы независимо друг от друга. Это упрощает процесс отладки и повышает качество программного продукта в целом. Кроме того, модульное тестирование позволяет обеспечить более широкий охват тестирования, что в свою очередь увеличивает надежность и стабильность работы программы.

Однако, не следует забывать, что ни один метод тестирования и никакое число тестов не может гарантировать безошибочной работы программного продукта.

2.8 ТЕСТИРОВАНИЕ СТРУКТУРНЫМ КОНТРОЛЕМ

Первым этапом тестирования программного продукта был выбран метод тестирования структурным контролем. Данный вид тестирования помогает

определить типовые ошибки, часто совершаемые в коде. Результаты тестирования представлены в таблице 2.

Тестирование структурным контролем показало, что ошибки, распространённые среди программистов, отсутствуют. После этого можно приступить к тестированию функциональности продукта с помощью других методов тестирования.

Таблица 2 — Тестирование структурным контролем

Вопрос	Результат проверки	Вывод
Обращения к данным		
Все ли переменные инициализированы?	Да, компилятор вывел бы ошибку, если бы были неинициализированные переменные.	Все переменные инициализированы.
Не превышены ли максимальные (или реальные) размеры массивов и строк?	У классов стандартной библиотеки STL нет ограничений на размеры массивов и строк.	Размеры массивов и строк не превышены.
Присутствуют ли переменные со сходными именами?	Нет, компилятор отобразил бы ошибку именования.	Нет переменных со сходными именами.
При вводе из файла проверяется ли завершение файла?	При чтении из файла в цикле проверяется завершение файла.	Нет ошибок при работе с файлами.
Соответствуют ли типы записываемых и читаемых значений?	В C++ при несоответствии типов записываемых и читаемых значений компилятор выдаёт ошибку, чего не наблюдается.	Типы соответствуют.
Не выходят ли индексы за границы массивов?	Нет, иначе мы бы увидели сообщение об ошибке в консоли терминала.	За границу массивов программа не выходит.

Продолжение таблицы 2

Вопрос	Результат проверки	Вывод
Передача управления		
Будут ли корректно завершены циклы?	Некорректное завершение циклов привело бы к некорректному поведению программу или к зависанию. Зависаний не происходит, а функциональность будет проверена в следующем пункте.	Циклы завершены корректно.
Будет ли завершена программа?	Программа будет работать без остановки, пока запущено моделирование.	Программа будет работать до отправки сообщения stop, если запущено моделирование, и пустой строки.
Интерфейс		
Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?	Соответствуют, иначе бы компилятор отобразил ошибку.	Соответствуют.
Не изменяет ли подпрограмма аргументов, которые не должны изменяться?	Все изменения с данными происходят в локальных переменных.	Подпрограммы не изменяют аргументы.
Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?	Глобальные переменные не используются.	Ошибок областей видимости нет.

2.9 МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Данный вид тестирования предназначен для проверки корректности функционирования отдельных компонентов библиотеки и удобен тем, что является автоматизированным. Были разработаны тесты, проверяющие работоспособность основных компонентов. В качестве примера написанных модульных тестов приведены некоторые тесты в приложении Г.

Результат модульного тестирования представлен в таблице 3.

Таблица 3 — Отчёт о прохождении всех модульных тестов

Номер теста	Файл теста	Успешность прохождения
1	printLineE-01.sbl	+
2	printLineE-02.sbl	+
3	printLineE-03.sbl	+
4	printLineE-04.sbl	+
5	printUndefinedE-01.sbl	+
6	printUndefinedE-02.sbl	+
7	printUndefinedE-03.sbl	+
8	printUndefinedE-04.sbl	+
9	scene-01.sbl	+
10	scene-02.sbl	+
11	scene-03.sbl	+
12	scene-04.sbl	+
13	scene-05.sbl	+
14	scene-06.sbl	+
15	math-01.sbl	+
16	tableFunctionAt-01.sbl	+
17	tableFunctionAt-02.sbl	+

Продолжение таблицы 3

Номер теста	Файл теста	Успешность прохождения
18	tableFunctionAt-03.sbl	+
19	tableFunctionAt-04.sbl	+
20	tableFunctionAt-05.sbl	+
21	tableFunctionAt-06.sbl	+

На основании полученного тестового отчёта был сделан вывод, что все модульные тесты были пройдены успешно и компоненты программного продукта работают корректно.

2.10 ОЦЕНОЧНОЕ ТЕСТИРОВАНИЕ НА ПРЕДЕЛЬНЫХ НАГРУЗКАХ

3 РАЗРАБОТКА ТЕХНОЛОГИИ НЕПРЕРЫВНОЙ ДОСТАВКИ И ТЕСТИРОВАНИЯ

На кафедре ИУ-6 МГТУ им. Баумана ведётся разработка системы SIMODO — адаптивной системы моделирования, предназначенной для упрощения создания моделей.

Система SIMODO решает проблему построения сложных моделей, которые зачастую объединяют несколько предметных областей. Для решения этой проблемы предлагается использование предметно-ориентированных языков, каждый из которых описывает ту или иную предметную область и должен быть интуитивно понятен специалистам этой области. Система SIMODO предоставляет способ унификации поддержки предметно- ориентированных языков [19].

Практическая значимость системы SIMODO заключается в возможности эффективно описывать и просчитывать сложные модели.

Система SIMODO предназначена для платформ Windows и Linux и не обладает автоматизированной системой развёртывания.

Развёртывание ПО — это все действия, которые делают программную систему готовой к использованию. Данный процесс является частью жизненного цикла программного обеспечения.

Целью данной эксплуатационной практики является создание и развитие системы развёртывания для системы SIMODO.

Задачами ставятся:

— Исследование видов установочных пакетов приложений.

— Выбор установочного пакета для системы SIMODO.

— Разработка системы формирования установочного пакета.

— Автоматизация формирования установочного пакета.

3.1 ИССЛЕДОВАНИЕ ВИДОВ УСТАНОВОЧНЫХ ПАКЕТОВ

Целевыми платформами системы SIMODO являются Windows и Linux. Дистрибутив Linux существует великое множество. Решено было остановится на одном из самых распространённых дистрибутиве Ubuntu версии Jammy [20],

использующий установочные пакеты DEB и дистрибутиве отечественной разработки Alt Linux версии P10, использующий установочные пакеты RPM. Первый критерий к установочным пакетам – поддержка целевых платформ Ubuntu Jammy и Alt Linux P10.

Исходные тексты программ системы SIMODO расположены в репозитории системы DevOps Gitlab [21]. Для непрерывного развёртывания в данной системе используются виртуальные машины на основе Linux дистрибутивов, что определяет второй критерий для установочного пакета системы SIMODO — возможность сборки пакета на дистрибутиве Linux, в частности под целевую платформу Windows.

Если поставлять динамические библиотеки, необходимые для работы системы SIMODO, вместе с системой SIMODO, то размер установочного пакета будет значительно больше, чем установочный пакет, к которому операционная система сможет самостоятельно предоставить необходимые зависимости. Третьим и необязательным критерием выбора установочного пакета является наличие системы разрешения зависимостей.

Были выбраны и исследованы установочные пакеты RPM, DEB, NSIS, MSI, QtIFW. В таблице 2 представлены результаты исследования.

Таблица 2 — Сравнение установочных пакетов

Установоч-ный пакет	Целевые платформы	Возможность сборки под Windows в Linux дистрибутиве	Наличие системы разрешения зависимостей
RPM	Linux дистрибутивы с пакетным менеджером rpm	–	+
DEB	Linux дистрибутивы с пакетным менеджером dpkg	–	+

Продолжение таблицы 2

Установоч-ный пакет	Целевые платформы	Возможность сборки под Windows в Linux дистрибутиве	Наличие системы разрешения зависимостей
NSIS	Windows	+	–
MSI	Windows	–	+
QtIFW	Windows, Linux	+	–

Из таблицы 2 видно, что установочный пакет QtIFW наиболее полно соответствует выставленным критериям.

3.2 РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ РАЗВЁРТЫВАНИЯ

Выбранный установочный пакет QtIFW собирается с помощью одноимённого фреймворка, который прежде, чем собирать установочный пакет, необходимо собрать из исходников [22]. Сам фреймворк требует аналогичной процедуры для статической библиотеки Qt. Для дистрибутивов Linux, в которых нет возможности установить утилиту linuxdeployqt, так же приходится собирать из исходников. Для сборки фреймворков под платформу Windows используется среда кросскомпиляции mxe, которую было решено заранее скомпилировать из исходников и разместить в репозитории Gitlab[23].

В приложении Б приведён листинг скрипта YAML, который используется для сборки установочного пакета в процессе непрерывного развёртывания Gitlab.

Чтобы собрать установочный пакет системы SIMODO, необходимо интегрировать данный процесс в проект системы SIMODO. В приложении А представлен листинг скрипта интеграции сборки установочного пакета в проект графического редактора системы SIMODO. С помощью переменных окружения контролируется вариант сборки под Windows или Linux.

Реализованная автоматизированная система развёртывания системы SIMODO формирует установочные пакеты системы SIMODO как артефакты конвейера системы непрерывного развёртывания Gitlab[23].

Получаемые установочные пакеты позволяют автономно устанавливать систему SIMODO под целевые платформы.

3.3 РАЗРАБОТКА ТЕХНОЛОГИИ ТЕСТИРОВАНИЯ ПОДСИСТЕМЫ

Программная подсистема имитационного моделирования процессов — это комплекс программных средств, предназначенных для создания имитационных моделей процессов, анализа их работы и оптимизации. Важным этапом разработки такой системы является тестирование, которое позволяет проверить работу системы на соответствие требованиям к ней и выявить возможные ошибки.

В рамках работы была разработана модель тестирования, которая включает в себя следующие этапы:

— Подготовка тестовых данных

— Подготовка сценариев тестирования

— Проведение тестирования

— Анализ результатов тестирования

На первом этапе были подготовлены тестовые данные. Данные состоят из файлов с различными параметрами, включая параметры, которые могут быть изменены пользователем.

Примерами таких файлов могут быть:

— Файл с данными для тестирования функции загрузки данных из внешнего источника.

— Файл с данными для тестирования функции изменения параметров моделирования.

— Файл с данными для тестирования функции экспорта результатов моделирования.

На втором этапе были подготовлены сценарии тестирования. Было решено, что каждый сценарий будет покрывать определенный функционал программной подсистемы. Сценарии были составлены в соответствии с требованиями, предъявляемыми к основным функциям программы.

Примеры сценариев тестирования:

— Сценарий тестирования функции загрузки данных из внешнего источника.

— Сценарий тестирования функции изменения параметров моделирования.

— Сценарий тестирования функции экспорта результатов моделирования.

На третьем этапе проводилось тестирование с помощью автоматизированных тестов. Весь процесс тестирования был описан с помощью скриптов. Также были подготовлены тесты, которые требовали ручного ввода данных. Были протестированы все основные функции программы. Результаты тестирования были записаны в файл.

Примеры автоматизированных тестов:

— Тестирование функции загрузки данных из внешнего источника.

— Тестирование функции изменения параметров моделирования.

— Тестирование функции экспорта результатов моделирования.

На четвертом этапе был проведен анализ результатов тестирования. Все ошибки были занесены в базу данных. Для каждой ошибки был составлен отчет, который включал описание ошибки, ее причину и рекомендации по устранению.

Примеры отчетов об ошибках:

— Ошибка при загрузке данных из внешнего источника.

— Ошибка при изменении параметров моделирования.

— Ошибка при экспорте результатов моделирования.

Каждый из этих этапов необходим для тщательного тестирования программной подсистемы имитационного моделирования процессов и помогает выявить возможные ошибки и проблемы в ее работе.

Примеры наборов сообщений для сценариев тестирования:

— Сценарий тестирования функции запуска моделирования приведён на листинге 1. Ожидаемый вывод представлен на листинге 2.

— Сценарий тестирования функции приостановки и возобновления моделирования приведён на листинге 3. Ожидаемый вывод представлен на листинге 4.

— Сценарий тестирования функции остановки моделирования приведён на листинге 5. Ожидаемый вывод представлен на листинге 6.

— Сценарий тестирования функции запроса дополнительных ресурсов приведён на листинге 7. Ожидаемый вывод представлен на листинге 8.

— Сценарий тестирования функции приема запрошенных ресурсов приведён на листинге 9. Ожидаемый вывод представлен на листинге 10.

Листинг 1 — Сценарий тестирования функции запуска моделирования

```
{
    "type" : "start",
    "path" : "/main.scriptc0",
    "script" : ""
}
```

Листинг 2 — Ожидаемый вывод сценария тестирования функции запуска моделирования

```
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "stop"
}
```

Листинг 3 — Сценарий тестирования функции запуска моделирования

```
{
    "type" : "start",
    "path" : "/main.scriptc0",
```

```
    "script" : "for (int i = 0; i < 1000000000; ++i);"
}
{
    "type" : "pause"
}
{
    "type" : "pause"
}
```

Листинг 4 — Ожидаемый вывод сценария тестирования функции запуска моделирования

```
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "stop"
}
```

Листинг 5 — Сценарий тестирования функции запуска моделирования

```
{
    "type" : "start",
    "path" : "/main.scriptc0",
    "script" : "
```

```
        for (int i = 0; i < 1000000000; ++i)
        { if (i % 1000000 == 0) sys.print(i); }
    "
}
{
    "type" : "stop"
}
```

Листинг 6 — Ожидаемый вывод сценария тестирования функции запуска моделирования

```
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "message",
    "brief" : "0",
    "atlarge" : "",
    "severity" : "info"
}
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "stop"
}
```

Листинг 7 — Сценарий тестирования функции запуска моделирования

```
{
    "type" : "start",
    "path" : "/main.scriptc0",
```

```
    "script" : "
        import \"module.scriptc0\" M m;
    "
}
{
    "type" : "module",
    "path" : "/module.scriptc0",
    "script" : ""
}
```

Листинг 8 — Ожидаемый вывод сценария тестирования функции запуска моделирования

```
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "request",
    "path" : "/module.scriptc0"
}
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "stop"
}
```

Листинг 9 — Сценарий тестирования функции запуска моделирования

```
{
    "type" : "start",
    "path" : "/main.scriptc0",
```

```
    "script" : "
        import \"module.scriptc0\" M m;
        sys.print(m.a);
    "
}
{
    "type" : "module",
    "path" : "/module.scriptc0",
    "script" : "int a = 2;"
}
```

Листинг 10 — Ожидаемый вывод сценария тестирования функции запуска моделирования

```
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "request",
    "path" : "/module.scriptc0"
}
{
    "type" : "result",
    "code" : 0
}
{
    "type" : "message",
    "brief" : "2",
    "atlarge" : "",
    "severity" : "info"
}
```

```
{
    "type" : "stop"
}
```

Проведение тестирования с помощью автоматизированных тестов позволяет значительно ускорить процесс проверки работы программной подсистемы и повысить ее качество. Разработанная технология тестирования может быть использована при дальнейшей разработке и совершенствовании программной подсистемы имитационного моделирования процессов.

Таким образом, разработанная технология тестирования позволила проверить работу программной подсистемы моделирования процессов на соответствие требованиям и выявить ошибки в работе системы.

ЗАКЛЮЧЕНИЕ

Заглушка

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. SIMODO в репозитории МГТУ им. Н.Э. Баумана. [Электронный ресурс]. URL: https://bmstu.codes/lsx/simodo (дата обращения: 07.07.2022).

2. Иванова Г.С., Жильцов А.И., Фетисов М.В., Чулин Н.А., Юдин А.Е. Адаптивная система моделирования. — Автоматизация. Современные технологии, номер 11 за 2020 год, стр. 500.

3. Официальная страница LSP [Электронный ресурс]. — URL: https://microsoft.github.io/language-server-protocol/ (дата обращения: 30.11.2022)

4. Популярные библиотеки C++ [Электронный ресурс]. — URL: https://en.cppreference.com/w/cpp/links/libs (дата обращения: 30.11.2022)

5. Библиотека Boost [Электронный ресурс]. — URL: https://www.boost.org/ (дата обращения: 30.11.2022)

6. Документация на фреймворк eCal [Электронный ресурс]. — URL: https://continental.github.io/ecal/_api/ecal_root.html (дата обращения: 30.11.2022)

7. Фреймворк ACE [Электронный ресурс]. — URL: http://www.dre.vanderbilt.edu/~schmidt/ACE.html (дата обращения: 30.11.2022)

8. Библиотека POCO [Электронный ресурс]. — URL: https://pocoproject.org/ (дата обращения: 30.11.2022)

9. Документация на фреймворк Qt [Электронный ресурс]. — URL: https://doc.qt.io/qt-5/ (дата обращения: 30.11.2022)

10. Межпроцессное взаимодействие в Qt [Электронный ресурс]. — URL: https://doc.qt.io/qt-6/ipc.html (дата обращения: 30.11.2022) 11. Неименованный канал [Электронный ресурс]. — URL: https://www.baeldung.com/linux/anonymous-named-pipes (дата обращения: 30.11.2022)

11. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. — М.: Наука, 1987. — 630 с.

12. Арушанян О. Б., Залеткин С. Ф. Решение задачи Коши для обыкновенных дифференциальных уравнений одношаговыми разностными

методами: практикум на ЭВМ по вычислительным методам. — М.: МГУ, 2002. — 51 с.

13. Десять популярных языков программирования для настольных приложений в 2021 году. [Электронный ресурс]. URL: https://www.decipherzone.com/blog-detail/top-programming-languages-for-desktop-apps-in-2021 (дата обращения: 07.02.2023)

14. SIMODO в репозитории МГТУ им. Н.Э. Баумана. [Электронный ресурс]. URL: https://bmstu.codes/lsx/simodo (дата обращения: 07.02.2023).

15. Ещё раз про семь основных методологий разработки / Хабр [Электронный ресурс]. URL: https://habr.com/ru/company/edison/blog/269789 (дата обращения: 07.02.2023)

16. Иванова, Г.С. Технология программирования: Учебник для вузов – М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. 238 с.

17. IBM Developer [Электронный ресурс]. URL: https://developer.ibm.com (дата обращения: 07.02.2023).

18. ГОСТ 19.701-90 «ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения»

19. Иванова Г.С., Жильцов А.И., Фетисов М.В., ЧулинН.А., ЮдинА.Е. Адаптивная система моделирования. – Автоматизация. Современные технологии, номер 11 за 2020 год, стр. 500.

20. Рейтинг дистрибутивов Linux. [Электронный ресурс]. URL: https://www.tecmint.com/top-most-popular-linux-distributions (дата обращения: 07.07.2022).

21. SIMODO в репозитории МГТУ им. Н.Э. Баумана. [Электронный ресурс]. URL: https://bmstu.codes/lsx/simodo (дата обращения: 08.07.2022).

22. Документация QtIFW. [Электронный ресурс]. URL: https://doc.qt.io/qtinstallerframework/ifw-getting-started.html (дата обращения: 05.07.2022).

23. Документация Gitlab. [Электронный ресурс]. URL: https://docs.gitlab.com/ (дата обращения: 10.07.2022).

ПРИЛОЖЕНИЕ А

Техническое задание

Листов X

ПРИЛОЖЕНИЕ Б

Фрагмент исходного кода

Листов X

ПРИЛОЖЕНИЕ В

Графический материал

Листов X

ПРИЛОЖЕНИЕ Г

Примеры модульных тестов

Листов 2

tableFunctionAt-05.sbl

```
import "tf" tf;
// Задание точки на функции
float x [1] = [4];
// Задание таблицы функции
float f [3, 2] = [
[1, 1],
[2, 4],
[3, 9]
];
// Вывод результата вычисления табличной функции
print tf.tableFunctionAt(x, f);
```

tableFunctionAt-06.sbl

```
import "tf" tf;
// Задание точки на функции
float x1 [2] = [0, -1];
// Задание точки на функции
float x2 [2] = [0, -2];
// Задание таблицы функции
float f [4, 3] = [
[0, 0, 1],
[0, 1, 2],
[1, 0, 2],
[1, 1, 3]
];
// Вывод результата вычисления табличной функции
print tf.tableFunctionAt(x1, f);
print tf.tableFunctionAt(x2, f);
```

writeFileLineE-01.sbl

```
import "io" io;
// Запись строки в файл с удалением старого файла
wait io.writeFileLineE(
"test/tmp/writeFileLine-01.out", "First", false);
// Добавление строк в конец файла
wait io.writeFileLineE(
"test/tmp/writeFileLine-01.out", "Second", true);
wait io.writeFileLineE(
"test/tmp/writeFileLine-01.out", "Third", true);
// Чтение содержимого файла
[file, error] = io.readFileE("test/tmp/writeFileLine-
01.out")
// Вывод содержимого файла
wait io.printUndefinedE(file);
```

ode-01.sbl

```
// Задание второстепенных параметров модели
float sigma;
float R;
float b;
float x;
float y;
float z;
// Задание интегрируемых параметров
float __dt_x;
float __dt_y;
float __dt_z;
// Задание СДУ
func __ODE[*]()
```

```
{
__dt_x = sigma*(y-x);
__dt_y = R*x - y - x*z;
__dt_z = -b*z + x*y;
}
```

scene-04.sbl

```
import "test/source/sbl/modules/scene/ode-01.sbl" ("sbl")
type LS;
// Инициализация модели
LS ls = {
sigma : 10, b : 8./3, R : 28,
x : 10, y : 1, z : 1,
};
// Инициализации сцены
import "scene" scene = {
t : 0.0, tk : 10.0, dt : 0.01,
callback_period : 100,
callback : func [scene, ls] () { print scene.t + ", " +
ls.x +
", " + ls.y + ", " + ls.z; },
actors : [ls],
};
// Запуск моделирования
wait scene.start();
```