



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 *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ

**«Разработка web-приложений, реализующих бизнес-логику
работы пользователя в САПР на основе
графоориентированной методологии»**


Студент РК6-83
(Группа)

В.О. Голубев

(Подпись, дата)

(И.О.Фамилия)

Руководитель ВКР


(Подпись, дата)

17/06/2020

А.П. Соколов

(И.О.Фамилия)

Консультант

А.Ю. Першин

(Подпись, дата)

(И.О.Фамилия)

Нормоконтролер

С.В. Groшев

(Подпись, дата)

(И.О.Фамилия)

Москва, 2020 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой РК-6
(Индекс)

_____ А.П. Карпенко
(И.О.Фамилия)

«____» _____ 2020 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

Студент группы РК6-83Б

_____ Голубев Владислав Олегович
(Фамилия, имя, отчество)

Тема выпускной квалификационной работы

Разработка web-приложений, реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии

Источник тематики (кафедра, предприятие, НИР): кафедра

Тема квалификационной работы утверждена распоряжением по факультету РК № №03.04.01-02/104_ВКР_Б от «25» октября 2019 г. _____

Часть 1. Аналитический обзор литературы.

В рамках аналитического обзора литературы должны быть изучены методы организации и визуализации процессов сложных вычислительных задач. На основе анализа методов необходимо разработать подход к удаленному запуску графоориентированных решателей с возможностью визуализации статуса решения. Обосновать возможность использования графовой модели в качестве инструмента для описания бизнес-логики работы в системе инженерного анализа

Часть 2. Разработка архитектуры программной реализации, программная реализация.

Разработка архитектуры подсистемы приема, переддачи и интерпретации запросов к графовой модели. Программная реализация спроектированных решений в плагине к серверу приложений, который осуществляет запуск произвольного графоориентированного решателя.

Часть 3. Отладка и тестирование

Тестирование полученного программного решения должно осуществляться в среде PBC GCD.

Должны быть протестированы успешный запуск различных тестовых графоориентированных решателей, а также поведение программного обеспечения в случае возникновения исключительных ситуаций.

Оформление выпускной квалификационной работы:

Расчетно-пояснительная записка на 46 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

5 графических листов

Дата выдачи задания «18» февраля 2020 г.

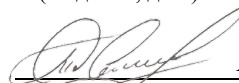
Студент

Руководитель выпускной квалификационной работы

В.О. Голубев

(Подпись, дата)

(И.О.Фамилия)



(Подпись, дата)

А.П. Соколов

(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

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 РК

КАФЕДРА РК-6

ГРУППА РК6-83Б

УТВЕРЖДАЮ

Заведующий кафедрой РК-6
(Индекс)

А.П. Карпенко

(И.О.Фамилия)

«___» _____ 2020 г.



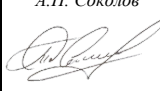
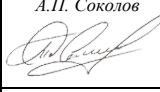
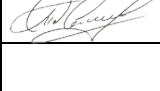
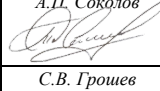
КАЛЕНДАРНЫЙ ПЛАН

выполнения выпускной квалификационной работы

студента: Голубева Владислава Олегович

(Фамилия, имя, отчество)

Тема выпускной квалификационной работы: Разработка web-приложений, реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии

№ п/п	Наименование этапов выпускной квалификационной работы	Сроки выполнения этапов		Отметка о выполнении	
		план	факт	Должность	ФИО, подпись
1.	Задание на выполнение работы. Формулирование проблемы, цели и задач работы	18.02.2020 Планируемая дата	18.02.2020	Руководитель ВКР	А.П. Соколов 
2.	1 часть: <u>аналитический обзор литературы</u>	18.02.2020 Планируемая дата	25.02.2020	Руководитель ВКР	А.П. Соколов 
3.	Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач	28.02.2020 Планируемая дата	28.02.2020	Заведующий кафедрой	А.П. Карпенко
4.	2 часть: <u>математическая постановка задачи, разработка архитектуру программной реализации, программная реализация</u>	31.03.2020 Планируемая дата	31.03.2020	Руководитель ВКР	А.П. Соколов 
5.	3 часть: <u>проведение вычислительных экспериментов, отладка и тестирование</u>	30.04.2020 Планируемая дата	30.04.2020	Руководитель ВКР	А.П. Соколов 
6.	1-я редакция работы	31.05.2020 Планируемая дата	31.05.2020	Руководитель ВКР	А.П. Соколов 
7.	Подготовка доклада и презентации	17.06.2020 Планируемая дата	08.06.2020		
8.	Заключение руководителя	15.06.2020 Планируемая дата	15.06.2020	Руководитель ВКР	А.П. Соколов 
9.	Допуск работы к защите на ГЭК (нормоконтроль)	15.06.2020 Планируемая дата	15.06.2020	Нормоконтролер	С.В. Грошев
10.	Внешняя рецензия	12.06.2020 Планируемая дата	17.06.2020		
11.	Защита работы на ГЭК	18.06.2020 Планируемая дата	19.06.2020		

Студент В.О. Голубев
(подпись, дата)

Руководитель работы А.П. Соколов
(подпись, дата)

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)

**НАПРАВЛЕНИЕ
НА ГОСУДАРСТВЕННУЮ ИТОГОВУЮ АТТЕСТАЦИЮ**

Председателю
Государственной Экзаменационной Комиссии № _____

факультета «Робототехники и комплексной автоматизации» МГТУ им. Н.Э. Баумана

Направляется студент Голубев Владислав Олегович группы РК6-81Б

на защиту выпускной квалификационной работы «Разработка web-приложений, реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии»

Декан факультета

« ____ » _____ 2020 г.

Справка об успеваемости

Студент Фамилия Имя Отчество за время пребывания в МГТУ имени Н.Э. Баумана с 2017 г. по 2020 г. полностью выполнил учебный план со следующими оценками: отлично – _____, хорошо – _____, удовлетворительно – _____.

Инспектор деканата _____

Отзыв руководителя выпускной квалификационной работы
Студент Голубев В.О. в процессе выполнения ВКР изучил архитектурные особенности web-клиента к программному комплексу PBC GCD, а также некоторые особенности системы в целом. В работе представлен результат проведённого обзора литературы в области разработки подсистем реализующих бизнес-логику функционирования отдельных модулей расширения в составе крупных программных систем. Результаты, полученные в работе, планируются использовать на практике, существенно ускоряя процесс создания новых подсистем комплекса PBC GCD. Работа выполнена автором самостоятельно, в полном объёме, в полном соответствии с заданием и календарным планом. Голубев В.О. достоин «отличной» оценки и присвоения звания бакалавр техники и технологий по направлению «Информатика и вычислительная техника».

Руководитель ВКР


(подпись)

А.П. Соколов
(ФИО)

« 17 » _____ июня 2020 г.
(дата)

Студент

(подпись)

В.О. Голубев
(ФИО)

« 17 » _____ июня 2020 г.
(дата)

АННОТАЦИЯ

Работа посвящена исследованию возможности реализации бизнес-логики работы пользователя помощью графоориентированной методологии. Программная реализация этой идеи позволит разработать систему инженерного анализа на основе графоориентированной методологии, где правила и функции системы будут описываться и реализовываться графовой моделью. Для этого были проанализированы научные работы и существующие аналогичные программные разработки и патенты. В результате анализа был предложен и реализован в программном виде подход по коммуникации с графовой моделью во время решения сложной задачи. Разработано приложение в web-клиенте PBC GCD для удаленного запуска произвольного графоориентированного решателя с возможностью визуализации статуса обхода графа в реальном времени.

Тип работы: выпускная квалификационная работа.

Тема работы: Разработка web-приложений, реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии

Объект исследований: интерактивное взаимодействие с графовой моделью в рамках применения графоориентированного подхода (англ. graph-based software engineering (GBSE)) при разработке программных реализаций сложных вычислительных методов.

СОКРАЩЕНИЯ

ООП — Объектно-ориентированное программирование.

САПР — Система автоматизированного проектирования.

PBC — Распределенная вычислительная система.

CASE — Computer-aided software engineering.

GBSE — Graph Based Software Engineering.

SDK — Software Development Kit.

DFD — Data Flow Diagram.

CBD — Component based design.

UML — Uniform Model Language.

PaaS — Platform as a Service.

SaaS — Software as a Service.

IaaS — Infrastructure as a Service.

RPC — Remote Procedure Call.

DCOM — Distributed COM (Component Object Model).

CORBA — Common Object Request Broker Architecture.

SOAP — Simple Object Access Protocol.

REST — Representational State Transfer.

HTTP — HyperText Transfer Protocol.

URL — Uniform Resource Locator.

JSON — JavaScript Object Notation.

HDF — Hubble Deep Field.

netCDF — Network Common Data Form.

INI — Initialization file.

YAML — Yet Another Markup Language.

SSH — Secure Shell.

FTP — File Transfer Protocol.

SFTP — Secure FTP.

MVT — Model-View-Template.

SPA — Single Page Application.

TCP — Transmission Control Protocol.

API — Application Programming Interface.

IPC — Inter Process Communication.

GUI — Graphical User Interface.

CRUD —Create Read Update Delete.

СОДЕРЖАНИЕ

АННОТАЦИЯ	8
СОКРАЩЕНИЯ	9
СОДЕРЖАНИЕ	11
ВВЕДЕНИЕ	12
1. ПОСТАНОВКА ЗАДАЧИ	20
1.1. Концептуальная постановка задачи	20
2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ	21
2.1. Используемые технологии	21
2.2. Коммуникации между клиентом и сервером	23
2.3. Описание алгоритма удаленного запуска решателя	26
2.4. Прием, передача и интерпретация запросов к графовой модели	29
2.5. Обработка исключительных ситуаций	31
3. ТЕСТИРОВАНИЕ И ОТЛАДКА	35
АНАЛИЗ РЕЗУЛЬТАТОВ	40
ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ВВЕДЕНИЕ

При проведении современных исследований возникает необходимость автоматизировать процессы решения сложных вычислительных задач. Достижение подобной цели не представляется возможным без формально определенного метода организации процессов в автоматизированной системе. Проектирование, создание и сопровождение подобных систем является трудоемкой задачей, для решения которой применяют инструментальные средства и среды разработки автоматизированных систем (CASE-системы) [1]

Для эффективного взаимодействия с подобной системой исследователю необходим графический пользовательский интерфейс, в котором модель организации вычислений может быть представлена визуально. Большие перспективы в автоматизации процесса решения сложных задач открывает перед исследователем возможность прямого взаимодействия с вычислительной моделью: остановка вычислительного процесса на определенном этапе, изучение обрабатываемых данных, просмотр истории изменения обрабатываемых данных, возврат к определенному этапу вычислений, ввод дополнительных параметров на определенной стадии вычислений и т.д. О необходимости подобных решений писал В. McCormic [2].

Для достижения подобной цели в GBSE необходимо решить множество технических задач по дополнению существующих инструментов. Одной из главных задач является дополнение функционала в существующий SDK для разработчиков графоориентированной методологии в плане обеспечения возможности приёма, передачи и интерпретации сообщений к графовой модели. Это позволит не только получать статус текущего решения, но также реализовать процесс отладки привычный для большинства инженеров-разработчиков, вводить дополнительные данные во время вычислительного процесса и др. Подобные возможности могут предоставить новый пользовательский опыт для пользователей графоориентированной методологии, поскольку в таком случае графовая модель становится не только инструментом для формального описания алгоритмов сложных

вычислительных задач, но и инструментов описания алгоритмов решения сложных задач, но и реализации бизнес-логики пользователя. На основе таких решений можно построить систему инженерного анализа с возможностью интерактивного взаимодействия с графовой моделью и организации вычислительных экспериментов с помощью графического пользовательского интерфейса. Для заложения основ построения бизнес-логики на основе GBSE и решения задачи получения статуса решения и последующей визуализации в данной работе рассматриваются работы методы визуализации и организации сложных вычислительных задач.

Основная идея решения сложных вычислительных задач сводится к их декомпозиции на более мелкие процессы [1]. Методики, используемые в CASE, предполагают организацию процесса решения в виде иерархической структуры, которую можно представить в виде диаграмм. Например, диаграммы потоков данных (DFD), граф-схемы, диаграммы перехода состояний. Такое описание позволяет выделять структурные единицы приложения в виде функций или подпрограмм и связывать их между собой в определенной последовательности. Продолжением идей структурного описания проектируемого приложения можно считать объектный подход, в котором стали применять методы объектно-ориентированного программирования, что впоследствии привело к методологии CBD [3]. Эта технология основывалась на унификации интерфейса для интеграции программного обеспечения в сложную систему. Языком описания для CBD является UML [4]. Для визуализации процессов используются диаграммы последовательностей, диаграммы состояний, диаграммы деятельности. Метод организации и выполнения процессов основанный на CBD описан Robert D. Bjornson и Stephen B. Weston [5]. В идее метода лежит организация вычислительных процессов, которые состоят из готовых компонентов, в виде диаграммы потока данных. Для создания и редактирования диаграммы предусмотрен графический интерфейс, в котором можно определить связи между процессами посредством указания точек входа и выхода обрабатываемых данных.

Метод разработки программного обеспечения при помощи проектирования моделей, не зависящих от предметной области запатентовали David TalbyScott и David McMaster [6]. Для описания организации процессов составляется UML-модель, которая интерпретируется для последующей генерации исходного кода на основе шаблонов. Подобная программная разработка может применяться для создания программного обеспечения для реализации сложных вычислительных методов решения задач. Подход к построению моделей обработки данных вычислительных экспериментов предложили российские исследователи Леонтьев, Тарасов, Харитонов [7]. Для построения моделей используются сети Петри. Модель вычислительного эксперимента состоит из моделей вычислительного и управляющего процессов. Построение этих моделей происходит отдельно. Модель вычислительного процесса строится автоматически на основе дерева событий. Модель управляющего процесса строится на определении шаблонов реакции на события. Подобный подход позволяет построить гибкую структуру вычислительного эксперимента и получать сообщения о статусе эксперимента от управляющего процесса.

Решение сложных вычислительных задач обусловлено запуском решения на производительных вычислительных машинах. Обычно к методам высокопроизводительных вычислений относят:

1. параллельные вычисления;
2. распределенные вычисления;
 - GRID;
 - Облачные вычисления (PaaS, IaaS, SaaS).

GRID-системы применяются учеными для вычислительных задач, когда производительности кластеров из мощных серверов недостаточно [8]. Алекперов Р.К. [9] в своем исследовании рассмотрел организацию распределенных вычислительных сред на основе GRID-технологии и проанализировал специфические особенности этих задач. В своей работе он привел практические примеры, когда распределенные вычисления помогли решать задачи, требующие десятки

лет вычислений, за несколько недель. В работе ученых Farkas Z. И Kacsuk P. описана разработка графического пользовательского интерфейса для взаимодействия с различными видами GRID-сетей [10]. В системе предусмотрен сервис для мониторинга выполнения задач в рамках архитектуры и визуализация статуса выполнения. Графический интерфейс представляет из себя поток работ (workflow), который графом описывает структуру GRID-сети, позволяет взаимодействовать с вычислительной системой и визуализировать состояние выполняемых задач. В системе существует Workflow Editor, который позволяет менять структуру вычислительной системы. Взаимодействия с различными видами GRID-сетей реализуется через промежуточный слой. Для каждой архитектуры написаны скрипты, которые переводят действия пользователя в графическом интерфейсе на язык конкретной архитектуры, а также обратно интерпретируют ответы от GRID-систем в workflow.

На сегодняшний день для проведения сложных вычислений многие ученые, исследовательские лаборатории и организации пользуются облачными платформами, которые предоставляют свои вычислительные мощности под любые нужды. Большинство облачных платформ предоставляет пользователю графический интерфейс (workflow) для организации процесса вычислений в виде диаграмм потока данных или граф-схемы [11], [12]. Подобные методы реализуются за счет использования промежуточного программного обеспечения, которое связывает готовые программные компоненты или набором библиотек, с помощью которых можно создавать пользовательские вычислительные процессы и исполнять их в облаке.

В настоящее время организация сложных вычислений все чаще проводится с использованием облачных технологий, поскольку подобный подход снимает большую часть ответственности по инфраструктурному обеспечению вычислительной системы и обходится дешевле, чем использование или разработка GRID-сетей. Существующие облачные вычислительные платформы в основном узко специализированы на решении задач машинного обучения [13], [14]. Существует

открытая разработка системы машинного обучения TensorFlow от компании Google, вычисления в которой выражаются в виде потоков данных через граф состояний [15]. Среди закрытых разработок стоит упомянуть проект вычислительной платформы российской компании Яндекс - Нирвана [16], которая позволяет производить вычисления, не завязанные на конкретной предметной области. Подобный подход позволяет использовать вычислительную платформу для широкого класса задач.

Среди российских разработок стоит отметить разработанную в МГТУ им. Н.Э. Баумана технологию GBSE, позволяющую упростить процессы проектирования, разработки, тестирования, сопровождения программных реализаций сложных вычислительных методов [17].

Все найденные методы были основаны на организации процессов решения в виде графов, сетей Петри или потоков данных. Визуализировать процессы с подобной структурой удобно с помощью соответствующих диаграмм. Для визуализации процесса решения в рассмотренных методах были предусмотрены механизмы отслеживания состояния и управления исполнением процессов (workflow monitor), в которых можно:

- увидеть детали выполнения процессов;
- увидеть историю выполнения процессов;
- остановить, прекратить или возобновить исполнение решения.

Подобные механизмы реализуются с помощью создания параллельного управляющего процесса, который может обмениваться сообщениями с исполняющим процессом [7], [10]. Такой подход позволяет разделить обязанности и не усложнять логику работы исполняющего процесса.

При решении сложных ресурсоемких задач зачастую вычислительных мощностей локального персонального компьютера уже не хватает и возникает потребность в использовании многопроцессорных удалённых машин. Для успеш-

ного задействования мощностей удаленного вычислительного кластера необходимо решить задачи получения соединения, передачи данных и команд по определенному протоколу, сериализации и десериализации данных, распределения нагрузки на процессоры и др [18], [19]. Над решением подобных задач работало много исследователей, а также производителей программного и аппаратного обеспечения.

В 70-х - 80-х годах появился ряд реализаций подхода удаленного запуска процедур (RPC) [20], [21] Подобные технологии предлагали подход запуска удаленных вычислений таким образом, чтобы это было похоже на вызов обычной процедуры в языках программирования. Среди основных можно выделить DCOM [22], CORBA [23], SOAP(XML-RPC) [24] и др. На сегодняшний день существуют более современные аналоги, такие как Thrift [25], Finagle [26] gRPC [27].

Нельзя не упомянуть архитектурный подход к взаимодействию компонентов распределенного приложения в сети — REST [28], который очень хорошо себя зарекомендовал как способ обмена данными и командами в клиент-серверной архитектуре. Поскольку REST использует такие стандарты, как HTTP, URL, JSON и XML, послать запрос на удаленный запуск процедуры на сервере не представляет большого труда.

Реализации RPC предлагали не только способы удаленного вызова, но и технологии сериализации-десериализации данных и даже шифрования, однако для передачи больших данных, которые использовали ученые и исследователи, производительность была недостаточной. Для научных наборов данных, имеющих большой объем, таких как данные, получаемые от спутников, или численные модели климата, погоды и океанов, были разработаны специальные бинарные стандарты сериализации, например HDF [29] и netCDF [30].

Для передачи разнородных данных по сети особую популярность приобрел язык разметки XML, однако, программная обработка XML может оказаться неоправданно затратной, по сравнению с работой с данными более простой структуры. В таком случае разработчики рассматривают средства, изначально ориентированные на данные, такие как INI, YAML, JSON.

Несмотря на большое количество технологий удаленного запуска процедур, одним из самых практичных до сих пор считается удаленный доступ к вычислительному кластеру и запуску на его мощностях необходимых вычислений. Существуют специальные программные системы для запуска вычислений на удаленных кластерах, которые используют протоколы SSH, FTP и SFTP. Например, программа COMSOL Desktop [31] позволяет с помощью графического интерфейса запустить задачу на расчет, просматривать текущий статус вычислений и получить результирующий файл, в котором содержатся результаты вычислений.

Запуском вычислительного процесса на кластере занимается специальная программа — система управления кластером, главными задачами которой является запуск вычислительных процессов, балансировка нагрузки на CPU и GPU, защита от сбоев, максимально быстрый возврат результата вычислений. Примерами таких систем могут служить Microsoft Cluster Server [32], Beowulf, Condor, MOSIX, EnFuzion, PBS [33] и др. Многие системы представлены в виде набора дополнений к ядру операционной системы, позволяющее распределять процессы между узлами в составе кластера.

Одной из альтернатив запуску больших вычислений на кластере являются GRID-сети, которые применяются учеными для вычислительных задач, когда производительности кластеров из мощных серверов недостаточно [34]. Алекперов Р.К. [9] в своем исследовании рассмотрел организацию распределенных вычислительных сред на основе GRID-технологии и проанализировал специфические особенности этих задач. В своей работе он привел исторические примеры, когда распределенные вычисления помогли решать задачи, требующие десятки лет вычислений, за несколько недель. Одной из самых популярных программных

систем для запуска вычисления и развертывания GRID-сетей является BOINC, которое используется в десятках научных проектов.

Несмотря на большую производительность, развертка GRID-сети не является самым лучшим решением для проектов с ограниченными ресурсами и задачами не требующих множества часов вычислений. Более рациональным вариантом может оказаться задействование облачных платформ для запуска вычислительных задач. На сегодняшний день существуют платформы для запуска решателей от всемирно известных производителей программного обеспечения, таких как Siemens, ANSYS, MSC [35], [36], [37] и др. Подобное решение позволит исследователю сосредоточиться на решении прикладных задач, а представитель сервиса берет на себя обязательства по предоставлению необходимых вычислительных мощностей с предустановленным программным обеспечением.

В результате анализа научных работ и существующих программных разработок в сфере организации и визуализации процессов решения сложных вычислительных задач был разработан подход к организации запуска графовой модели таким образом, чтобы иметь возможность интерактивного обращения к ней во время её решения. На основе данного подхода была построена программная инфраструктура для приема, передачи и интерпретации сообщений к графовой модели. Изучив различные подходы к удаленному запуску вычислений, можно сделать вывод, что одним из самых популярных способов на сегодняшний день является удаленный доступ к вычислительной машине. Создание приложения для программного запуска вычислений подразумевает решение множества задач, для чего существуют различные средства в виде сетевых протоколов, форматов данных, готовых библиотек для языков программирования и т.д.

Целью работы является разработка программного обеспечения для взаимодействия с графовой модели в технологии GBSE для обеспечения получения статуса обхода графовой модели и визуализации в реальном времени.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Концептуальная постановка задачи

Разработка Web-приложений, реализующих бизнес-логику работы пользователя в системах инженерного анализа на основе графоориентированной методологии проводится в рамках работы над РВС GCD на кафедре РК6.

Объект разработки: графическое представление статуса обхода графовой модели.

Цель разработки: создать программное обеспечение для приема, передачи и интерпретации сообщений к графовой модели с визуализацией статуса решения.

Задачи выпускной квалификационной работы:

- 1) провести обзор литературы по теме: "Методы визуализации процессов решения сложных вычислительных задач" (не менее 15 источников);
- 2) провести обзор литературы по теме «Технологии и методы удаленного запуска процедур и функций на высокопроизводительных вычислительных системах»;
- 3) разработать функцию системы (в рамках web-клиента), позволяющую визуализировать состояние выполнения удаленного процесса (в частности, обход тестового графа);
- 4) доработка сервера приложений русомарс в части поддержки приема и интерпретации запросов о состоянии выполнения процесса с последующим формированием ответа и его отправки на клиентскую сторону;
- 5) разработать схему архитектуры подсистемы, обеспечивающей прием-передачу-интерпретацию запросов для обеспечения решения задачи визуализации.

2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ

2.1. Используемые технологии

Клиентская часть (comwps) реализована на языке Python с использованием web-фреймворка Django [38]. Сервер приложений реализован на языке Python. Для работы с сокетами используется модуль `ruzmq` — реализация технологии ZeroMQ [39] для языка Python. В качестве сервера баз данных используется PostgreSQL. Для решения задачи были задействованы модули `comsdk` и `ruscomsdk` - SDK для программных реализаций сложных вычислительных методов в рамках графоориентированной технологии GBSE на языках C++ и Python. Диаграмма компонентов системы удаленного запуска представлена на рисунке 1. Django хорошо подходит для разработки веб-приложений разной сложности. Внутри он реализует паттерн MVT. Веб-клиент `comwps` использует собственный подход к разработке и в отличие от предлагаемого Django по-умолчанию синхронного режима обмена информации по HTTP, использует технологию асинхронных запросов AJAX. Таким образом, `comwps` является нетипичным для веб-приложений на Django SPA-приложением, т.е. приложением, которое не требует перезагрузки в браузере при переходе на новую страницу. Веб-клиент общается с единой для PBC GCD базой данных, используя сервер `comaps`, использующийся в качестве проксирующего сервера. Сам процесс коммуникации происходит по специальному протоколу. Передача информации по сети происходит по протоколу TCP.

Сервер приложений принимает сообщения из TCP-сокета, работа с которым осуществляется с помощью библиотеки `ruzmq`, представляющую реализацию технологии ZeroMQ на языке Python.

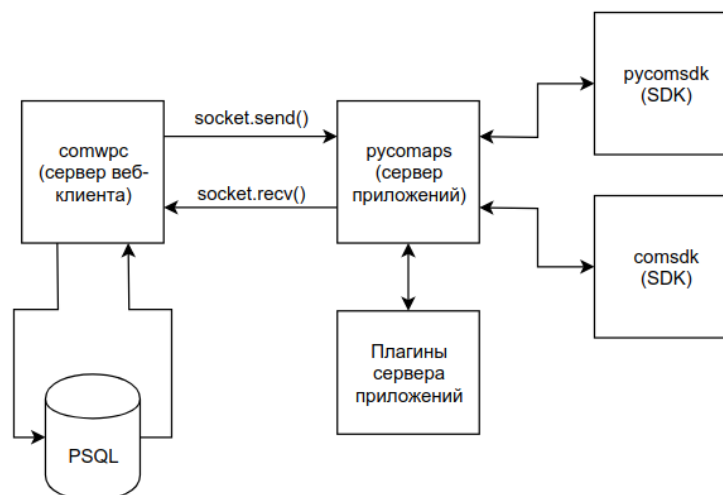


Рисунок 1 - Диаграмма компонентов системы удаленного запуска

Технология ZMQ представляет API для удобной работы с сокетами, который реализует наиболее востребованные паттерны работы в сетевом программировании. Например, самый распространенный паттерн Request-Reply представлен специальной парой сокетов REQ и REP. Функции библиотеки берут на себя сложную работу с очередями сообщений, осуществляя обработку запросов без блокировок. Для этого в ZMQ очереди строятся на неблокируемых структурах данных[40]. Благодаря такому подходу работа с сокетами упрощается до вызова нескольких методов для передачи и принятия сообщений.

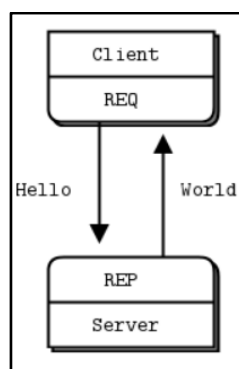


Рисунок 2 - Схема паттерна Request-Reply, реализованного с использованием zmq-сокетов

На сервере приложений русомaps используется сочетание сокетов Router-Dealer. Этот паттерн позволяет разработать асинхронный сервер, который может

принимать множество запросов и распределять нагрузку на параллельные процессы-обработчики. Схематичное изображение работы сокетов Router-Dealer представлено на рисунке 3.

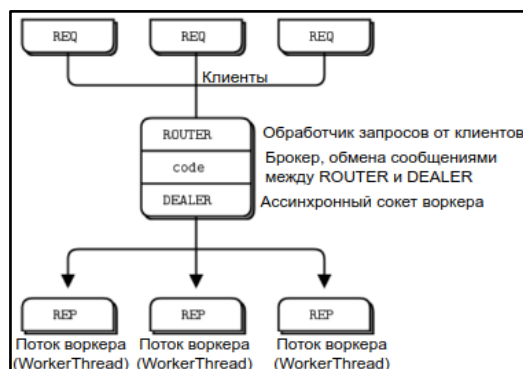


Рисунок 3 - Схема паттерна Extended Request-Reply, реализованного на zmq-сокетах

2.2. Коммуникации между клиентом и сервером

Web-клиент somwps не поддерживал работу с новым сервером приложений русомарс. Требовалось наладить коммуникацию между двумя узлами PBC GCD для дальнейшей работы над разработкой плагина удаленного запуска графоориентированных решателей. Изначально somwps поддерживал коммуникацию с сервером приложений сомарс. Для этого существовал класс `cls_Node`, методы которого содержали логику обращения к серверу приложений, осуществления TCP-handshake, передачи и приема данных. Чтобы оставить поддержку коммуникации сомарс и добавить коммуникацию с русомарс. Архитектурная реализация классов была изменена в соответствии с идеями ООП о наследовании и полиморфизме. Был введен абстрактный класс `Node`, который определял интерфейсы для коммуникации с другими узлами PBC. Данные интерфейсы должны реализовывать классы наследники: `cls_Node` и `ZMQNode`. Для передачи и приема запросов с сервером приложений русомарс были разработаны соответствующие методы, реализующие абстрактные интерфейсы с использованием ZMQ-сокетов. Таким образом удалось в короткое время, не изменив большого количества кода расширить функционал web-клиента для возможности коммуникации с русомарс. На рисунке 4 продемонстрирована UML-диаграмма классов, осуществляющих коммуникацию с русомарс и сомарс.

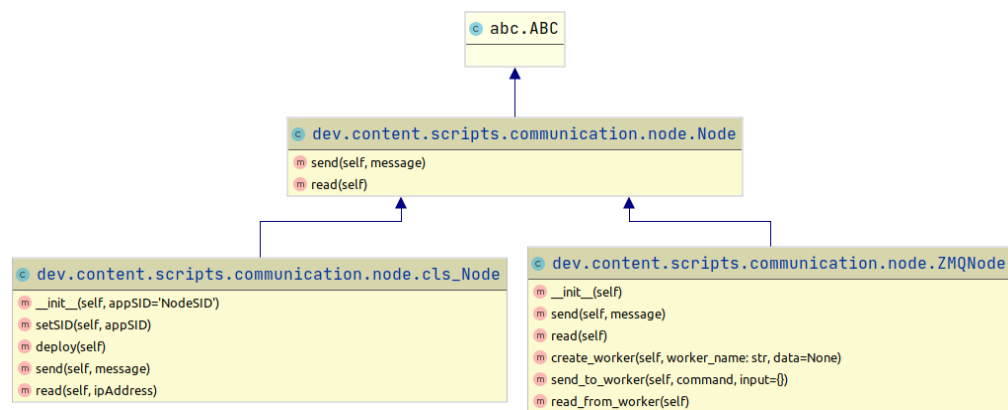


Рисунок 4 - UML-диаграмма классов, осуществляющих коммуникацию с русомaps и сомарс

Текстовый протокол основывается на формате JSON. Пример сообщения на запуск обработчика на сервере приложений приведен на рисунке 5.

```

{
    'command': 'create_worker',
    'worker_name': worker_name,
    'input': data
}
  
```

Рисунок 5 - Пример формата сообщения на создание обработчика

Здесь в значении ключа `command` находится название команды, которую сервер приложений должен исполнить, `worker_name` - название обработчика запроса, `input` — поле, в котором хранится информация с передаваемыми данными. Кроме команды на создание обработчика, сервер приложений поддерживает команду запроса текущего статуса работы обработчика — `get_worker_result`. При этом необходимо обязательно указывать идентификатор обработчика.

Логика обработки запросов в сервере приложений реализуется в классе `ApplicationServer`. Методы этого класса запускают сервер с нужной конфигурацией. На запрос от клиента на создание нового обработчика порождается новый процесс `Worker`, который выполняет определенную задачу. Каждый `Worker` имеет индивидуальный идентификатор, который позволяет однозначно определить его среди множества работающих в данный момент обработчиков. Каждый `Worker` может взаимодействовать с `ApplicationServer` с помощью IPC. Для этого создаются пары сокетов `DEALER-REP`, как было показано на рисунке 3. Самый

простой случай взаимодействия процессов `AppServer` и `Worker (requestHandler)` рассмотрен на рисунке 6. Можно увидеть, как достигается асинхронная обработка запросов от веб-клиента к `requestHandler`. Каждый процесс `requestHandler` для запускает алгоритм обработки в отдельном потоке, когда как главный процесс прослушивает возможные обращения от сервера приложений.

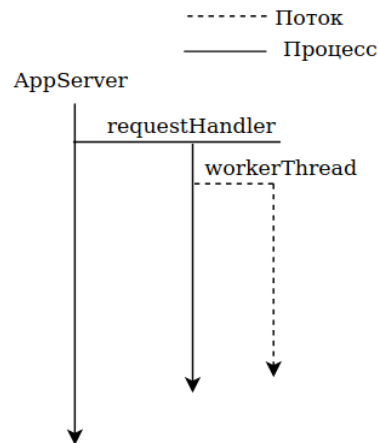


Рисунок 6 - Схема распараллеливания процессов `AppServer` и `Worker`

Структура каждого обработчика строго определена. Каждый `Worker` представляет собой модуль языка `Python` с двумя классами: `WorkerThread` и `RequestHandler`, которые в свою очередь являются наследниками абстрактных классов `AbstractWorkerThread` и `AbstractRequestHandler`. UML-диаграмма классов наследников для плагина удаленного запуска приведена на рисунке 7.

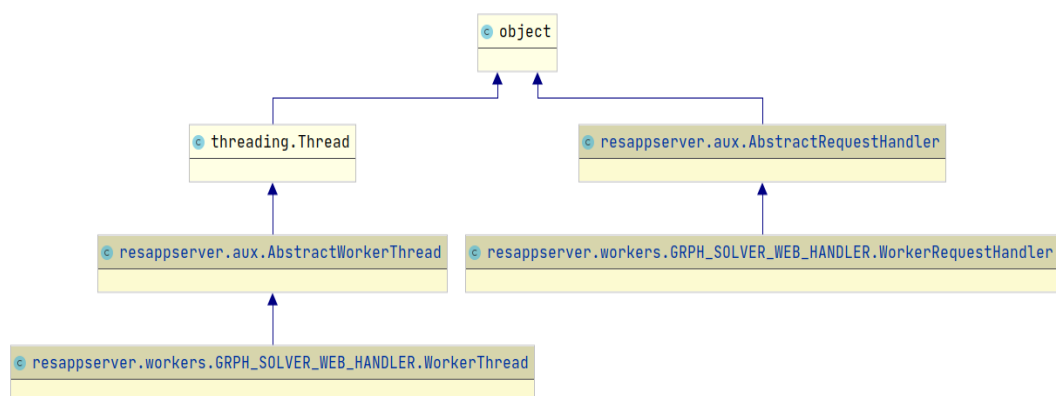


Рисунок 7 - UML-диаграмма классов-наследников для плагина удаленного запуска

`AbstractWorkerThread` является наследником класса `Thread`, который в языке `Python` описывает поток распараллеливания. Данный класс содержит стандарт-

ную логику для всех обработчиков, определяющую создание и перезапуск потока. WorkerThread конкретного обработчика определяет логику обработки конкретной задачи. Наследники класса AbstractRequestHandler определяют все возможные команды для обработчика.

Для того, чтобы обратиться к обработчику с web-клиента, необходимо указать его идентификатор, команду и входные данные. Сервер приложений перенаправляет это сообщение напрямую к обработчику, чтобы тот в свою очередь выполнил необходимую команду. Пример такого запроса представлен на рисунке 8.

```
{
    'worker_id': worker_id,
    'command': command,
    'input': input
}
```

Рисунок 8 - Пример запроса к обработчику сервера приложений

Для передачи сообщений по сети необходимо «сериализовать» встроенные в Python типы данных, такие как словарь, в JSON. Для этого на web-клиенте и сервере приложений используется стандартная библиотека для работы с форматом JSON. Её методы позволяют «сериализовать» и «десериализовать» строки и словари.

2.3. Описание алгоритма удаленного запуска решателя

На стороне web-клиента необходимо разработать функцию для ввода данных, какой решатель необходимо запустить. Плагины к web-клиенту в PBC называются Action Items. Для построения пользовательского интерфейса comwpc использует собственный генератор GUI (wpc_gui_ini_builder), который на основе файла исходных данных в формате aINI создаёт GUI пользователя для ввода входных данных, таких как имя решателя и имя файла входных данных для него также в формате aINI. Генератор создает кнопку «Обработать» и привязывает к ней необходимый субплагин на сервере веб-клиента. Для удаленного запуска это субплагин GRPH_SOLVER_WEB. В субплагине формируется запрос на сервер

приложений. Схема работы автоматического построения GUI приведена в приложении.

Для обработки пользовательских действий применялись скрипты на языке JavaScript, которые загружались при старте web-клиента и описывали обработчики возникающих браузерных событий. Однако данный подход был найден неудобным и трудно расширяемым, поскольку в нем отсутствовала систематизация, присутствующая во многих других частях PBC GCD. Многие скрипты, которые загружались были не востребованы пользователем, поскольку он мог выбрать для работы функции системы, которые данные скрипты не поддерживают. Это негативно сказывалось на производительности веб-клиента. Данный подход не систематизирован так как это сделано во многих частях PBC GCD, из-за чего трудно было написать новый код для обработки взаимодействия пользователя с веб-клиентом без дублирования или усложнения уже реализованных скриптов.

Чтобы исправить эти недостатки, была реализована следующая идея. При загрузке субплагина плагином для автоматического построения GUI в интерфейс web-клиента загружается привязанный скрипт для обработки пользовательских действий при взаимодействии с субплагином. Скрипт содержит описание класса на языке JavaScript, который наследуется от базового класса Action Item, который был разработан для того, чтобы задать общие для всех субплагинов интерфейсы, которые классы наследники могли реализовывать и расширять. В созданном скрипте обработчик субплагина были реализованы функции AJAX-запроса к comwps, реализован интерфейс обработки кнопки «Обработать», построения визуализации графовой модели и визуализации статуса решения. Таким образом удалось динамически добавлять необходимые скрипты, которые необходимы пользователю при работе с функциями системы, а также создать единый подход для разработчиков написания логики обработки браузерных событий и обработки событий во время работы функции системы.

Сервер приложений принимает запрос на создание обработчика и создает отдельный процесс (Worker). Worker представляет собой плагин к серверу приложений для запуска решателя. Он связывается с процессом сервера с помощью

сокета, таким образом он способен принимать запросы и обрабатывать их. В начале работы плагин валидирует присланные для обработки данные. В частности, проверяет, что заданный решатель существует в системе. Он пытается найти файл с описанием графового решателя в формате aDOT. Если плагину не удастся найти файл, он отправляет на web-клиент ответ с сообщением о возникновении исключительной ситуации. Если находит, то подает данный файл на вход парсеру из `pycomsdk`. Парсер представляет из себя экземпляр класса `Parser`, который имеет несколько методов для обработки файла графовой модели. Метод `Parser.parse_file()` формирует Python-объект `Graph`, в котором содержится информация о ребрах и вершинах графа. Метод `Parser.generate_cpp()` генерирует исходный код решателя на языке C++. UML-диаграмма описываемых классов приведена на рисунке 9.

После разбора файла aDOT становится возможным запросить у сервера структуру исполняющейся графовой модели. Для этого в функции парсера была добавлена возможность формирования графовой модели в виде структуры данных языка Python — словарь, которая представляет из себя хранилище типа ключ-значение. Такая структура хорошо сериализуется для передачи по сети в формате JSON. Пример формат хранения такой структуры представлен на рисунке 10.

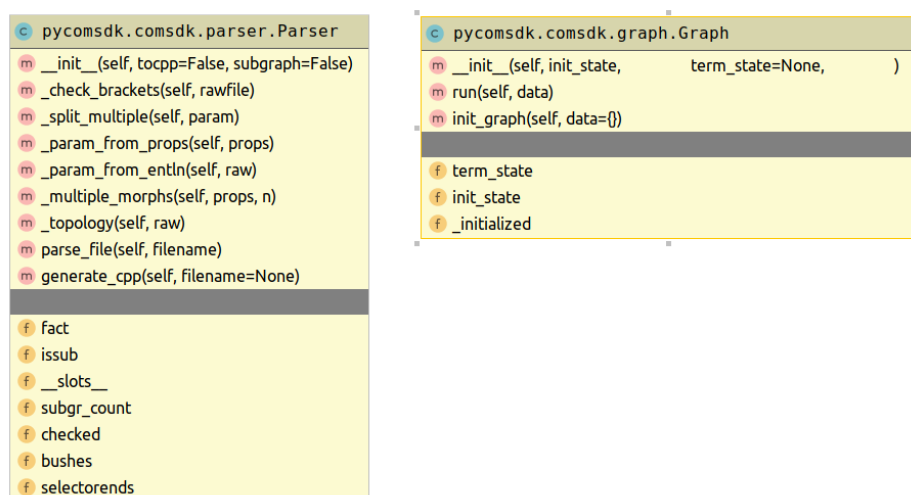


Рисунок 9 - UML-диаграмма классов `Parser` и `Graph`

Непосредственный запуск решателя может осуществляться одним из двух способов. Если функции-обработчики и функции-предикаты, привязанные к ребрам могут быть импортированы как Python-модули, то производится обход графа напрямую из Python с помощью метода Graph.run(). Если функции не могут быть найдены, то генерируется исходный код решателя на языке C++, в котором подключаются необходимые динамические библиотеки из модуля comsdk. Далее происходит компиляция и запуск решателя. Блок-схема алгоритмы удаленного запуска приведена в Приложении А.

```
graph: {
  'init_state_name': {
    'subgraph': None/{subgraph},
    'connect_to': [{
      'next': next_state_name,
      'morph_f': morphism,
      'pred_f': predicate
    }, ...]
  },
  'next_state_name': {...}, ... ,
  'term_state_name': {...}
}
```

Рисунок 10 - Пример описания структуры графовой модели в формате JSON

2.4. Прием, передача и интерпретация запросов к графовой модели

Исходя из анализа научных работ и существующих программных разработок, был предложен подход к созданию архитектуры подсистемы приема, передачи и интерпретации запросов к графовой модели. Была взята идея разделения запуска процесса вычислений на два процесса: управляющий и исполнительный.

Функции управляющего процесса:

1. порождение исполнительного процесса;
2. прием запросов и выдача ответов;
3. управление вычислениями.

Функции исполнительного процесса:

1. обход графовой модели;
2. передача статуса решения.

Такой подход позволит не усложнять логику обхода графовой модели, поскольку обязанности по коммуникации и управлению вычислениями берет на себя управляющий процесс. На рисунке 11 представлена визуальная схема распараллеливания процессов запуска графоориентированного решателя.

Чтобы не вносить существенных изменений в логику обхода графовой модели, было решено задействовать потоки `stdout` и `stderr` для передачи статуса обхода графа. Пусть во время обхода графовая модель записывает в `stdout` успешно пройденные состояния, таким образом их можно однозначно интерпретировать, как текущий статус решения. Если в `stderr` не пуст, то это можно интерпретировать как ошибку в результате обхода графовой модели и обработать как исключительную ситуацию. Просмотр информации из управляющего процесса осуществляется с помощью объекта исполнительного процесса, у которого существуют поля `stdout` и `stderr`.

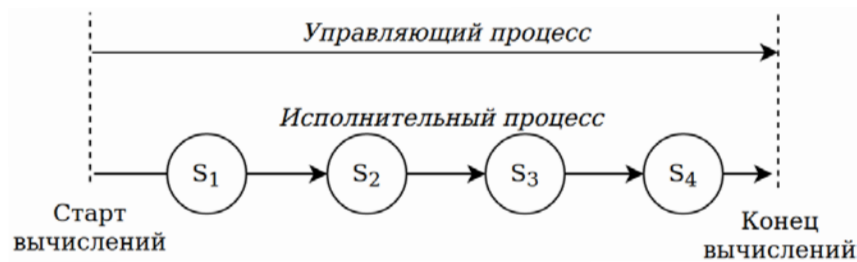


Рисунок 11 - Схема распараллеливания процесса запуска графоориентированного решателя

Данные от плагина с сервера приложений передаются на web-клиент, где визуализируются на построенной графовой модели с помощью раскраски ребер в случае успешного прохождения. Если была обнаружена ошибка, то на web-клиент отправляется сообщение, поясняющее причину ошибки. Более подробно обработка ошибок описана в разделе 2.5. Для того, чтобы получать данные в реальном времени весь процесс получения статуса является итеративным. Чтобы получить статус обхода графа, web-клиент с заданным периодом по времени генерирует запросы на сервер приложений и визуализирует полученный ответ. На рисунках 12, 13 приведены схемы обработки запросов получения статуса с web-клиента.

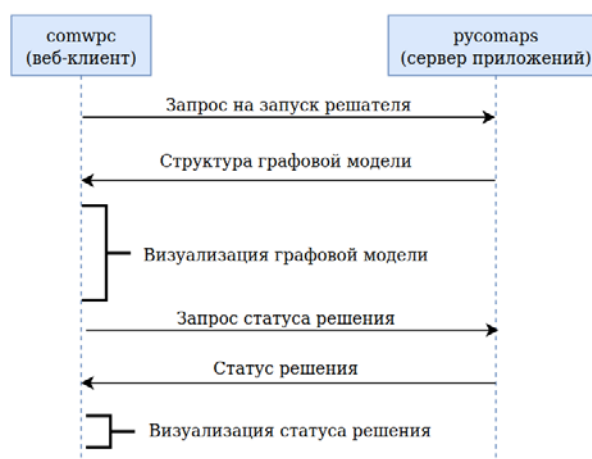


Рисунок 12 - Схема отправки запроса на получение статуса решения

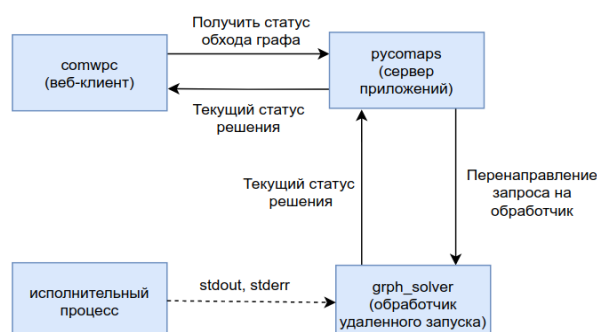


Рисунок 13 - Схема запросов при удаленном запуске

2.5. Обработка исключительных ситуаций

Дополнительной задачей при разработке подсистемы приема, передачи и интерпретации запросов к графовой модели стала разработка подсистемы уведомлений о событиях в системе на web-клиенте. Решение этой задачи позволило корректно обрабатывать исключительные ситуации при удаленном запуске графоориентированных решателей, однако разработанная подсистема решает более широкий класс задач.

Все сообщения, которые необходимо показать пользователю можно разделить на три категории:

1. информационные сообщения;
2. предупреждающие сообщения;
3. сообщения о возникшей ошибке.

Это позволяет систематизировать подход к обработке пользовательских уведомлений и создать единообразный подход к обработке возникающих событий в

системе. В других узлах PBC GCD, таких как desktop-клиент, данная задача решена и построенная информационная инфраструктура для обработки событий возникающих в системе. На рисунке 14 представлена информационная модель системных сообщений в PBC GCD. Главной является таблица «Стандартные сообщения». В ней находятся уникальный идентификатор формата сообщения и строковое поле формата сообщения. Формат сообщения – это форматная строка, которая обычно используется для вывода форматных сообщений в языках C/C++. Чтобы сформировать итоговое сообщение для лога, в форматную строку подставляются параметры (числовые или строковые). Каждое стандартное сообщение имеет свой тип, язык вывода и метод вывода (в каком узле PBC сообщение должно выводиться и каким образом). В PBC GCD поддерживаются два языка вывода: русский и английский.

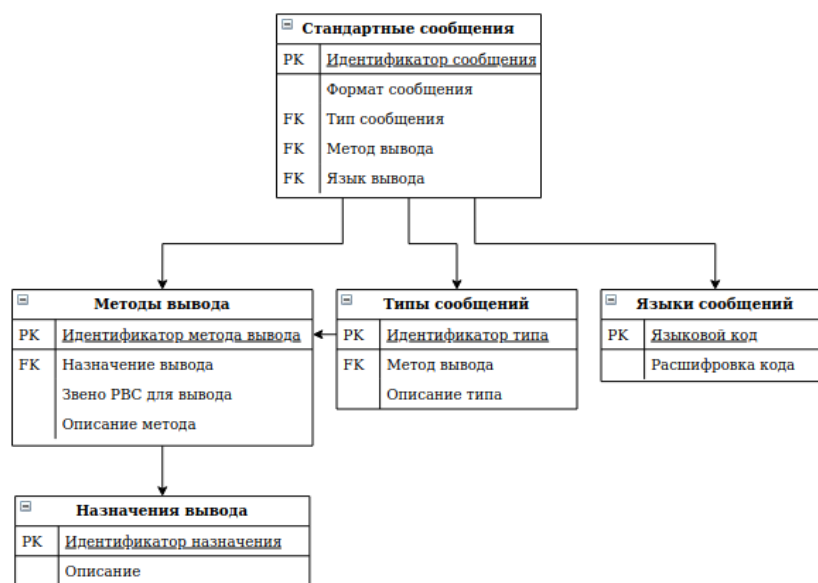


Рисунок 14 - Информационная модель системных сообщений PBC GCD

Для web-клиента требовалось разработать систему, которая сможет выполнять следующий функционал:

- 1) зарегистрировать возникшее событие;
- 2) формировать сообщение о возникшем событии;
- 3) привязывать сообщение к текущей сессии пользователя web-клиента;
- 4) поддержка широковещательных сообщений;

- 5) показывать сообщение с указанием типа события;
- 6) позволять пользователю управлять всеми полученными сообщениями.

В результате исследований по теме и требованиям к разработке было решено, что каждое событие в системе должно быть зарегистрировано в едином хранилище. Это верно для критических ошибок в системе на стороне сервера приложений. Если возникает ошибка в результате обработки запроса от пользователя, то такие события важны только для пользователя и информация о них является временной, поэтому было решено, что пользовательские события должны храниться во временном хранилище на стороне клиента. Такое решение, также, обусловлено требованием связывать возникающее событие с сессией текущего пользователя.

Для регистрации события была разработана Django-модель Events. Django-модель представляет собой объектно-ориентированную обертку над инфологической единицей данных. Все CRUD-операции с объектом модели транслируются на базу данных. В качестве базы данных сервер веб-клиента использует SQLite. Методы класса модели Events реализуют функции регистрации и обработки события, а также подготовки данных для отображения сообщения пользователю. На рисунке 15 представлена UML-диаграмма, описывающая разработанную Django-модель. На рисунке 16 представлена блок-схема обработки возникшего на стороне сервера приложений пользовательского события в веб-клиенте.

События	
PK	Идентификатор сообщения
FK	идентификатор пользователя
	идентификатор сообщения
	тип сообщения
	текст сообщения
	метод вывода
	время
	метка о прочтении сообщения

Рисунок 15 - UML-диаграмма Django-модели

Для разработчиков был разработан интерфейс в виде двух публичных методов класса Events. Метод Events.registr(event_msg, user, is_broadcast) позволяет зарегистрировать событие event_msg и связать его с пользователем user. Опционально можно создать широковещательное сообщение, если передать параметр is_broadcast=True. Метод Events.get_message() возвращает словарь с готовыми полями для отображения данных пользователю. Сообщение будет выдано на языке, который указан у пользователя предпочтительным в браузере. Это достигается за счет анализа полей HTTP-запроса.

Для показа сообщений были разработаны функции на языке JavaScript, которые показывают всплывающее сообщение, а также функции для управления оповещениями в разработанной панели уведомлений.

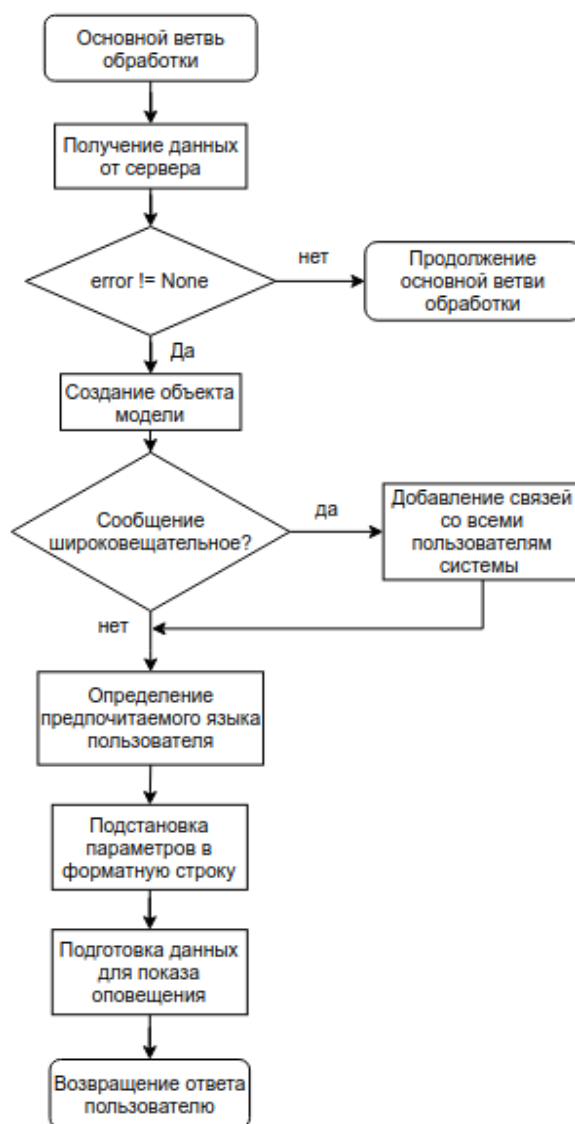


Рисунок 16 - Блок-схема алгоритма обработки возникшего события

3. ТЕСТИРОВАНИЕ И ОТЛАДКА

В рамках тестирования реализованного функционала был локально развернут сервер приложений русомарс и веб-клиент comwps. Инструментарий разработчика для работы с графовыми моделями на языке C++ был, также, локально собран, чтобы иметь доступ к библиотекам функций-обработчиков и функций-предикатов. Тестирование производилось вручную. Из web-клиента была вызвана функция GRPH_SOLVER_WEB. Функция WPC_GUI_INI_BUILDER сгенерировала графический пользовательский интерфейс ввода данных для запуска решателя. Автоматически сгенерированный интерфейс функции показан на рисунке 17. Динамически были загружены скрипты, отвечающие за обработку браузерных событий, вызванных пользователем.

Для тестирования необходимо выбрать тестовый графовоориентированный решатель. Требовалось получить визуализированную графовую модель и увидеть статус выполнения решателя в виде закрашенных ребер графа в случае успешного решения, а также протестировать возникновение исключительных ситуаций и их обработку с помощью разработанной системы уведомлений. Для этого были протестированы несколько сценариев:

- 1) успешный сценарий выполнения;
- 2) неверное задание решателя;
- 3) ошибка во время обхода графовой модели.

Дополнительно требуется протестировать разработанный функционал системы уведомлений о событиях. В случае возникновения ошибок вычисления должны быть корректно завершены, а пользователю показано уведомление описанием возникшей ошибки. Пользователь должен увидеть, что в панели уведомлений был инкрементирован счетчик и он может прочесть уведомление, отметить как прочитанное и удалить все уведомления.

В случае выполнения всех требований разработанное web-приложение может быть интегрировано в PBC GCD.

Рисунок 17 - Форма ввода, сгенерированная из файла в формате aINI

Тестирование начинается с нажатия пользователем кнопки «Обработать». Отправляется запрос на сервер приложений и вызывается функция GRPH_SOLVER_WEB_HANDLER для обработки удаленного запуска тестового решателя. Тестовый решатель описывается файлом в формате aDOT. Исходный текст файла представлен на рисунке 18. Данный файл описывает простой граф из 3 узлов. К ребрам графа привязаны тестовые функции-обработчики и функции-предикаты из динамической библиотеки libcomsdk.

```
digraph gcdfem_gbse_model
{
// Определение функций-обработчиков
PASS_PROCESSOR [module=libcomsdk, entry_func=pass_processor]

// Определение функций-предикатов
PASS_PREDICATE [module=libcomsdk, entry_func=pass_predicate]

// Определение морфизмов
PASS_MORPHISM [predicate=PASS_PREDICATE, function=PASS_PROCESSOR]

// Определение топологии графовой модели метода конечных элементов
BEGIN__ -> S_1
S_1 -> S_2 [morphism=PASS_MORPHISM, comment = "Препроцессинг завершён, осуществляем подготовку к расчету..."]
S_2 -> S_3 [morphism=PASS_MORPHISM, comment = "Расчет завершён, осуществляем подготовку к постпроцессингу..."]
S_3 -> __END__ [comment = "Постпроцессинг завершён. Расчет завершён."]
}
```

Рисунок 18 - Графовая модель тестового решателя в формате aDOT

Сначала тестировался успешный сценарий выполнения. Лог сервера приложений представленный на рисунке 19 показывает, что во время парсинга файла не удалось найти модуль libcomsdk, поэтому построенная графовая модель описывается исходным кодом на языке C++, который был автоматически сгенерирован с помощью инструментов русcomsdk. Структура графовой модели в формате

JSON была отправлена на web-клиент, где она была интерпретирована и на её основе построена визуализация графовой модели в виде ориентированного графа с множеством состояний и функций перехода, которые соответствуют заданному графоориентированному решателю. Результат визуализации представлен на рисунке 20.

```

BEGIN -> S_1
LOADING function pass_predicate from libcomsdk module
Module was not found in python modules. Generating .cpp
BEGIN -> S_1
S_1 -> S_2
S_2 -> S_3
S_3 -> END
BUILDING gcdfem_gbse_model
States:
    BEGIN
    S_1
    S_2
    S_3
    END
Generating done!

```

Рисунок 19 - Лог сервера приложений, показывающий процедуру парсинга и генерации исходного кода решателя

После визуализации с web-клиента были посланы запросы на получение статуса обхода графовой модели. Запросы были приняты и перенаправлены на плагин-обработчик удаленного запуска, который в свою очередь отдал собранные из stdout исполнительного процесса данные в ответ.

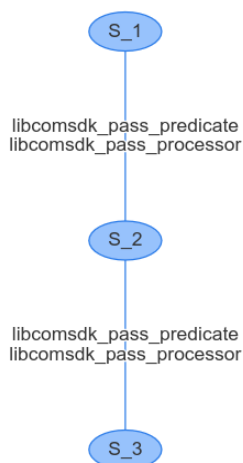


Рисунок 20 — Построенная визуализация графовой модели в web-клиенте

В результате интерпретации полученных данных были закрашены ребра построенной визуальной графовой модели, что можно увидеть на рисунке 21. Плагин на сервер приложений корректно завершил свою работу.

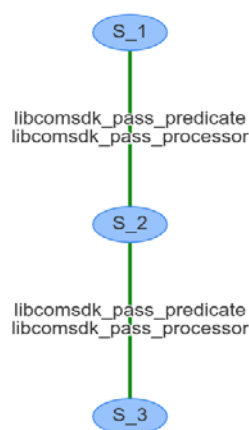


Рисунок 21 — Визуализация статуса решения в web-клиенте.

Для тестирования сценария неверного задания решателя был указан несуществующий в хранилище графовых моделей файл aDOT. В результате обработки плагин не обнаружил файл и сформировал и отправил сообщение об ошибке в виде идентификатора сообщения и параметров, которые должны быть подставлены в форматную строку. Web-клиент обнаружил в ответе сервера сообщение о возникшей ошибке и зарегистрировал событие типа «ошибка» связанное с сессией текущего пользователя, сформировал сообщение пригодное для визуализации в пользовательском интерфейсе и показал пользователю всплывающее окно с информацией о возникшем событии. Результат представлен на рисунке 22. При тестировании ошибки обхода графового решателя было показано уведомление о возникновении ошибки при работе графоориентированного решателя и показаны сообщения из stderr.

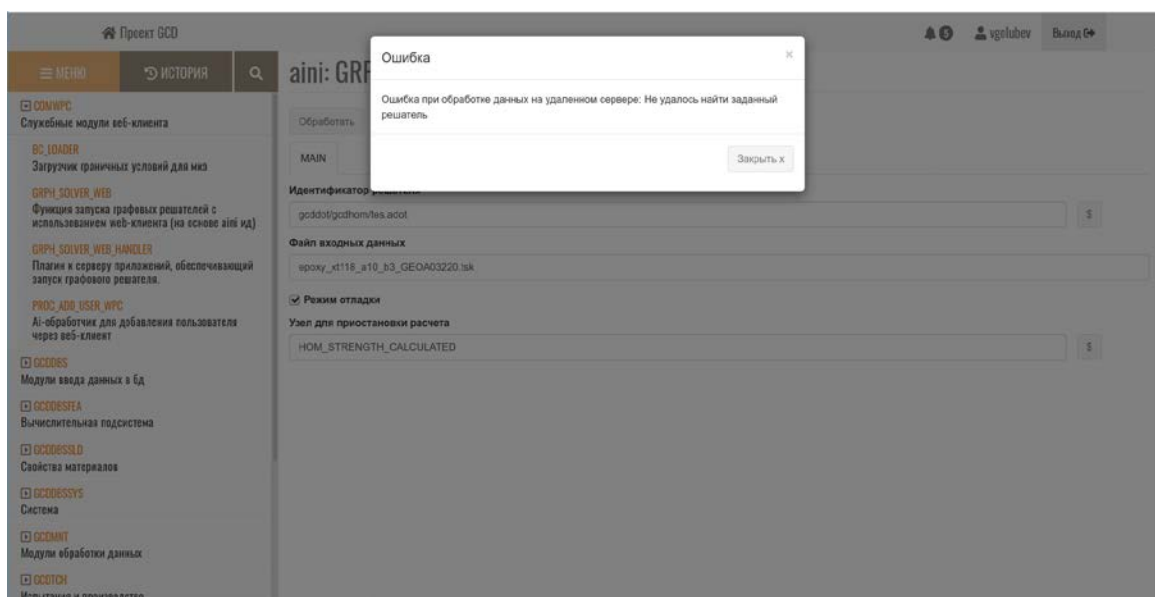


Рисунок 22 - Уведомление о возникновении ошибки

Для тестирования функционала подсистемы уведомлений была открыта панель уведомлений. На ней отражаются полученные уведомления. С помощью функции «отметить все уведомления как прочитанные» обнуляется счетчик новых уведомлений, что показано на рисунке 23. С помощью функции «удалить уведомления» панель уведомлений очищается и все данные о полученных сообщениях стираются из базы данных web-клиента.

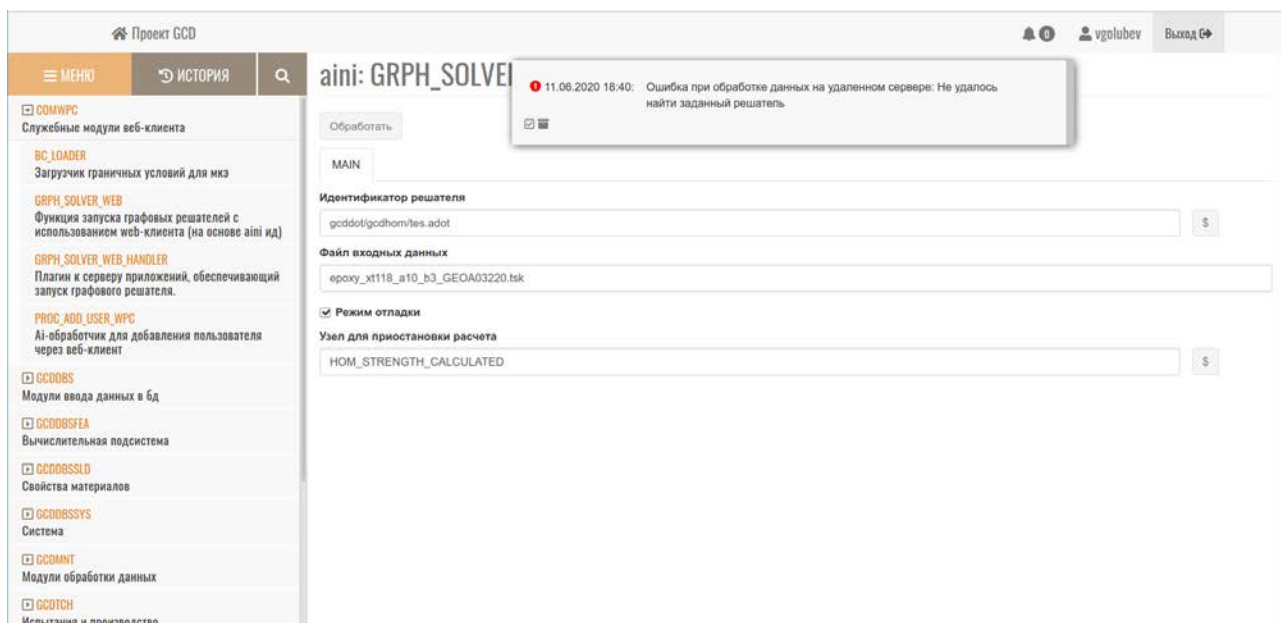


Рисунок 23 - Панель уведомлений после нажатия на кнопку «отметить уведомления как прочитанные»

АНАЛИЗ РЕЗУЛЬТАТОВ

В результате проделанной работы удалось получить программную реализацию web-приложения, которое воплощает идею возможности прямого обращения к графоориентированному решателю во время исполнения процесса решения. Полученное решение имеет свои отличия и недостатки.

Среди положительных сторон работы стоит отметить разработанный подход к запуску графовых моделей с разделением сложных вычислений на два процесса операционной системы: управляющий и исполнительный. Это позволило реализовать программную инфраструктуру для приема, передачи и интерпретации запросов к графовой модели без значительного вмешательства в существующий SDK для разработчиков графоориентированных решателей, а также не усложнять логику обхода графовой модели, поскольку все функции по взаимодействию с клиентскими запросами обрабатываются в отдельном процессе параллельно. Разработанная система удаленного запуска учитывает наличие SDK на разных языках программирования и задействует все возможные средства, чтобы обеспечить запуск любого графоориентированного решателя в системе. Использование полученной программной разработки удобно для инженеров, которые не посвящены в детали графоориентированного подхода, поскольку детали реализации скрыты от пользователя. Вместо этого ему предоставляется удобный графический пользовательский интерфейс для задания нужного решателя и файла исходных данных. В ответ на запрос удаленного запуска пользователь получает визуальное отображение статуса решения, что может оказаться полезным в различных ситуациях, когда решение сложной вычислительной задачи занимает много времени.

Другим положительным пунктом работы является систематизация программного кода в плагине автоматического построения GUI и задание некоторого образца для других разработчиков к обработке действий пользователя системы в интерфейсе субплагинов с помощью разработанной системы динамической загрузки скриптов, а также применения принципов инкапсуляции и полиморфизма для избежания дублирования кода.

Среди недостатков или недоработок стоит отметить, что предложенный подход за счет своей простоты не позволяет напрямую влиять на процесс вычислений. Полученная программная разработка способна решать задачу получения статуса решения, что обеспечивается за счет логирования пройденных стадий в стандартный поток вывода. Для получения значимых результатов в проектировании приложений, которые могут реализовывать бизнес-логику работы пользователя в САПР на основе графоориентированной методологии, необходима возможность взаимодействия графовой модели с пользователем. Это означает разработку функций графовой модели, которые будут способны запрашивать данные у пользователя для обработки, функции остановки вычислений на определенном этапе с возможностью пошагового выполнения просмотра истории обхода, а может даже возвращение на шаг назад и выполнение функции перехода заново с новыми данными. Для реализации подобных функций недостаточно просто собирать информацию обхода, необходимо значительно переработать SDK для описания алгоритмов решения сложных задач в виде графовой модели таким образом, чтобы имело действие реальное межпроцессное взаимодействие между управляющим и исполнительным процессом, например, с помощью программного канала (pipe), а синхронизация достигалась бы с помощью известных подходов с использованием захвата блокировок. Решение такой задачи требует тщательного перепроектирования программной архитектуры графовой модели, что не входило в задачи в данной работы.

На основе проведенных исследований можно возможно воплотить идею использования графоориентированной методологии как основы для построения бизнес-логики работы пользователя в системе инженерного анализа, если учесть недостатки разработанного подхода. Решение основной задачи удаленного запуска произвольного графоориентированного решателя, получения и визуализации статуса решения соответствует заявленным требованиям. Одним из возможных дополнительных сценариев применения полученного решения в настоящий момент — визуальная валидация и отладка графоориентированных решателей.

ЗАКЛЮЧЕНИЕ

В результате исследований по теме было разработано web-приложение реализующее удаленный запуск произвольного графоориентированного решателя и визуализацию статуса обхода графовой модели. Для решения задачи были проанализированы научные работы и программные разработки в области организации и визуализации процессов решения сложных вычислительных задач, а также изучены классические и современные подходы к удаленному запуску процедур. В результате анализа источников была разработана программная архитектура плагина для удаленного запуска графоориентированных решателей и подсистемы приема, передачи и интерпретации запросов к графовой модели. Дополнительно разработана система уведомлений пользователя о событиях в системе, которая позволила корректно обрабатывать исключительные ситуации при удаленном запуске решателей. В результате всех разработок были улучшены и систематизированы некоторые аспекты в разработке web-клиента PBC GCD.

Полученная программная реализация была протестирована на разных графовых моделях и полностью удовлетворяет заявленным требованиям. При тестировании исключительных ситуаций удаленного запуска все вычисления корректно завершались и пользователю были показаны сообщения в возникших ошибочных событиях в системе при обработке графоориентированного решателя.

При анализе полученного решения были выделены положительные и отрицательные стороны разработки, составлены рекомендации по улучшению существующего решения для достижения цели построения бизнес-логики пользователя в системе инженерного анализа на основе графоориентированной методологии.

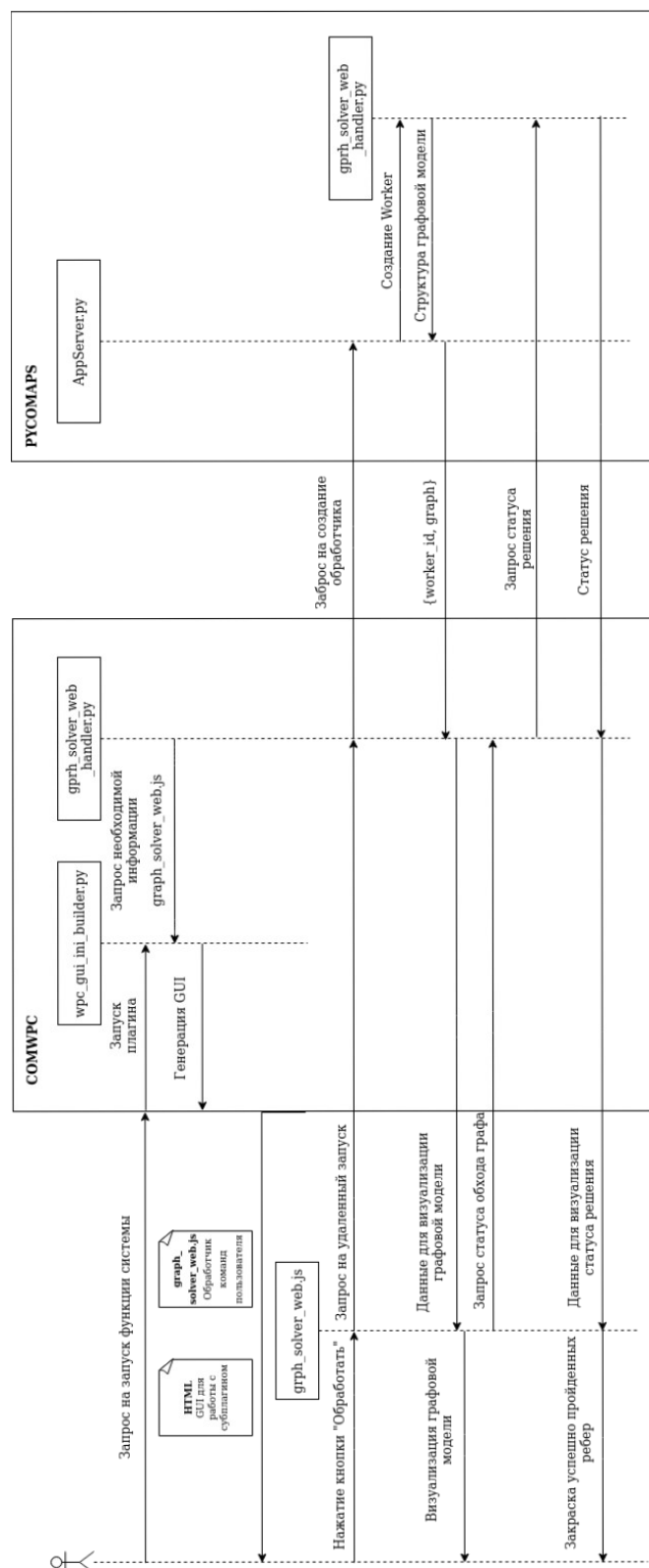
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Норенков И.П Основы автоматизированного проектирования, М: Изд-во МГТУ им. Н. Э. Баумана, 2006. - 448 с.
2. McCormic B. Visualization in Scientific Computing // New York: Computer Graphics. 1987, 81 p.
3. McIlroy M. Mass produced software components : NATO Software Engineering Conference (7-11 Oct. 1969), Garmisch, Germany: Scientific Affairs Division, 1969,79 p.
4. ISO/IEC 19501:2005 - Information technology - Open Distributed Processing - Unified Modeling Language (UML) Version 1.4.2., Iso.org, 2005. - 456 с.
5. US 2004/0056908 A1 - Turbo Worx, Inc. (New Haven, CT, US) - Method and system for dataflow creation and execution (2004).
6. US 8,949,772 B1 - Amazon Technologies, Inc., (Reno, NV, US) - Dynamic model based software application development (2015).
7. Леонтьев Д.В., Тарасов В.Г., Харитонов И.Д. Модель обработки данных вычислительных экспериментов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции. М.: ИПИМ им. М.В.Келдыша, 2018. - С. 373–386.
8. Zurek R.W, Martin L.J. GridPP: development of the UK computing grid for particle physics // New York: Journal of Physics G: Nuclear and Particle Physics, 2006, p. 1–20.
9. Алекперов А. Организация распределенных вычислений на базе GRID-технологии // Баку: Искусственный интеллект. - 2011. - С. 6-14.
10. Farkas Z., Kacsuk P. P-GRADE Portal: A generic workflow system to support user communities // Elsevier. - Volume 27 (Future Generation Computer Systems), 2010, p. 455–465.
11. Godec, P., Pančur, M., Democratized image analytics by visual

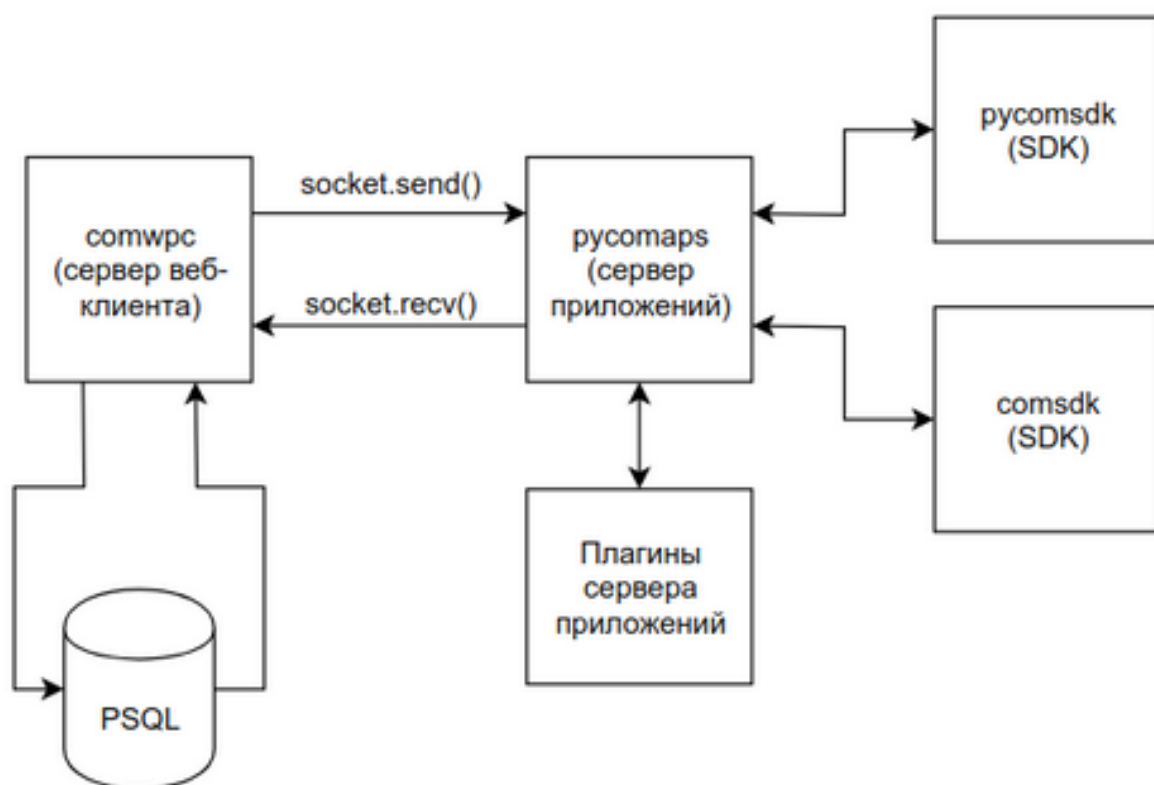
12. programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, С. 33-40.
13. Kudryavtsev A.O., Koshelev V.K., Izbyshchev A.O. Development and implementation of the cloud system for solving problems of high // Proceedings of the Institute for System Programming of Russian Academy of Sciences, Volume 24, 2011, p. 13-33.
14. Janez Demsar, Tomaz Curk, Ales Erjavec Orange: Data Mining Toolbox in Python // Journal of Machine Learning Research, 2013 (14), С. 2349–2353.
15. Официальный сайт проекта TensorFlow [Электронный ресурс]. – Режим доступа: <https://www.tensorflow.org>.
16. Познаём Нирвану – универсальную вычислительную платформу Яндекса [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/company/yandex/blog/351016/>
17. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – Т.47, No5– 2019, с. 43-55.
18. Tanenbaum, Andrew S. Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall, 2006, 702 p.
19. Magnoni, L. Modern Messaging for Distributed Systems (sic). Journal of Physics: Conference Series, 2015, p. 10-15.
20. RFC-1050: RPC: Remote Procedure Call [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc1050>
21. RFC-1057: RPC: Remote Procedure Call Protocol Specification Version 2 [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc1057>
22. Distributed Component Object Model [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/4a893f3d-bd29-48cd-9f43-d9777a4415b0

23. CORBA to WSDL/SOAP Interworking Specification [Электронный ресурс]. – Режим доступа: <https://www.omg.org/spec/C2WSDL/About-C2WSDL/>
24. SOAP Version 1.2 [Электронный ресурс]. – Режим доступа: <https://www.w3.org/TR/soap12/>
25. Apache Thrift Documentation [Электронный ресурс]. – Режим доступа: <https://thrift.apache.org/docs/>
26. Finagle Official Page [Электронный ресурс]. – Режим доступа: <https://twitter.github.io/finagle/>
27. gRPC Documentation [Электронный ресурс]. – Режим доступа: <https://www.grpc.io/docs/>
28. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. // Springer Science & Business Media, 2011, 528 p.
29. hHDF5 Software Documentation [Электронный ресурс]. – Режим доступа: <https://support.hdfgroup.org/HDF5/doc/>
30. Белоушко, К. Е. Формат NetCDF как стандарт для обмена данными в атмосферных исследованиях // Тезисы II конференции «Базы данных, инструменты и информационные основы полярных геофизических исследований» (POLAR-2012), Троицк, ИЗМИРАН, 2012.
31. Как работать с кластерами через графический интерфейс COMSOL Desktop [Электронный ресурс]. – Режим доступа: <https://www.comsol.ru/blogs/how-to-run-on-clusters-from-the-comsol-desktop-environment/>
32. Raj Rajagopal. Introduction to Microsoft Windows NT Cluster Server: Programming and Administration, 2000, 320 p.
33. Аветисян А.И., Грушин Д.А., Рыжов А.Г. Системы управления кластерами // Труды Института Системного Программирования РАН, Москва, 2002.

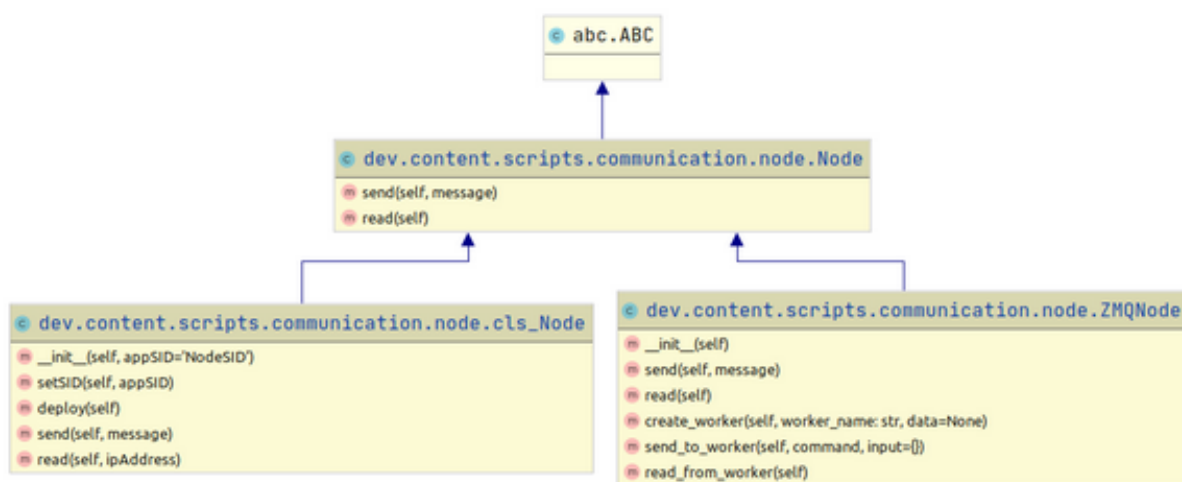
34. Zurek R.W, Martin L.J. GridPP: development of the UK computing grid for particle physics // New York: Journal of Physics G: Nuclear and Particle Physics, 2006, p. 1–20.
35. Алекперов А. Организация распределенных вычислений на базе GRID-технологии // Баку: Искусственный интеллект. - 2011. - С. 6-14.
36. Gompute High Performance Computing [Электронный ресурс]. – Режим доступа: <https://www.gompute.com/>
37. TotalCAE Supported Engineering Applications [Электронный ресурс]. – Режим доступа: <https://www.totalcae.com/engineering-applications.php>
38. Django Web Framework [Электронный ресурс]. – Режим доступа: <https://www.djangoproject.com/>
39. ØMQ - The Guide [Электронный ресурс]. – Режим доступа: <http://zguide.zeromq.org/py:all>
40. M. Herlih. A Methodology for Implementing Highly Concurrent Data Object // ACM Transactions on Programming Languages and Systems, 1993, p.746-770.




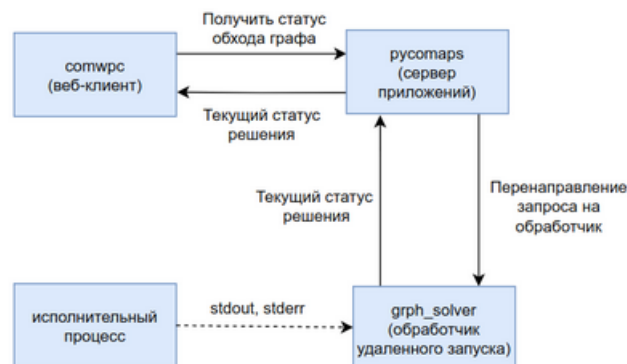
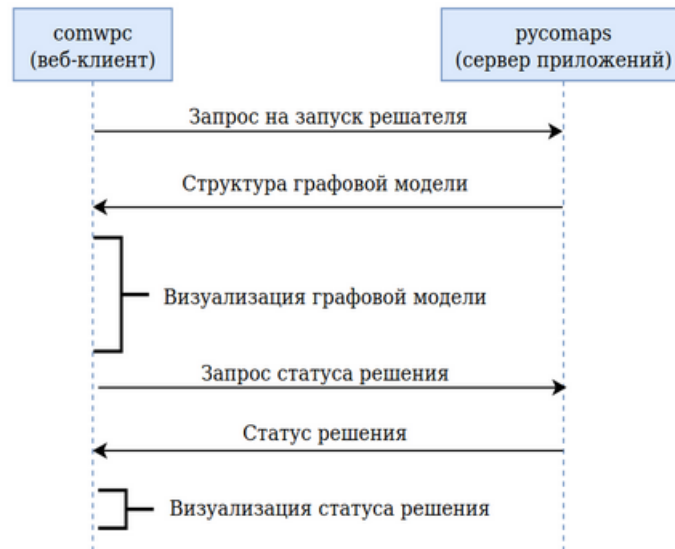
Разработка web-приложений реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии							
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.	Голубев В.О.				Лит.	Лист	Листов
Провер.	Соколов А.П.			17.06.2020		1	5
Н. Контр.					МГТУ им. Н.Э. Баумана		
Утв.							
Схема запросов к серверу приложений							




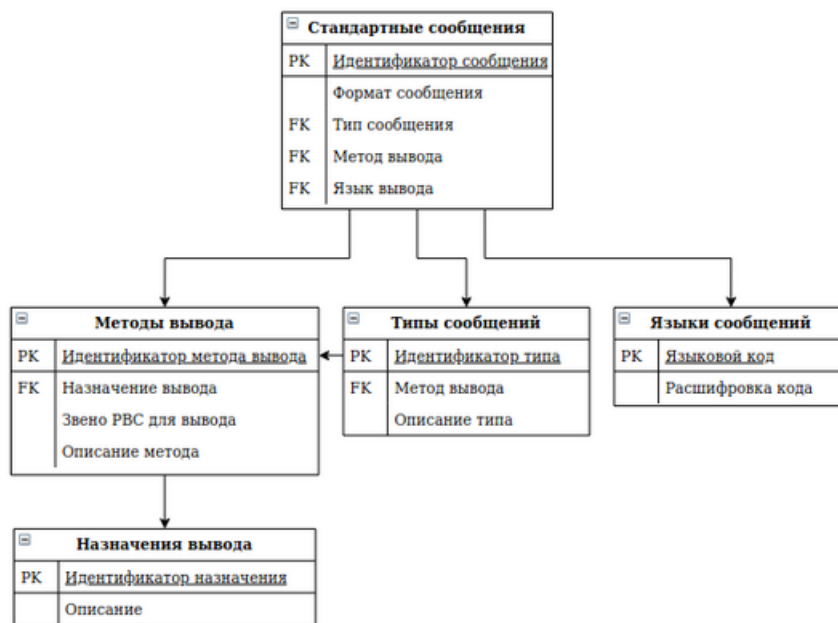
					Разработка web-приложений реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Голубев В.О.			Приложение. Схема основных узлов РВС GCD, участвующих в удаленном запуске		
Провер.		Соколов А.П.		17.06.2020			
Н. Контр.							
Утв.							
					Лит.	Лист	Листов
						2	5
					МГТУ им. Н.Э. Баумана		



					Разработка web-приложений реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Голубев В.О.			Приложение. UML- диаграмма классов, описывающих коммуникацию на web- клиенте		
Провер.		Соколов А.П.		17.06.2020			
Н. Контр.							
Утв.							
						Лит.	Лист
							3
							5
						МГТУ им. Н.Э. Баумана	



					Разработка web-приложений реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии							
Изм.	Лист	№ докум.	Подпись	Дата	Приложение. Схемы, описывающие процесс получения статуса решения задачи				Лит.		Лист	Листов
Разраб.	Голубев В.О.										4	5
Провер.	Соколов А.П.			17.06.2020								
Н. Контр.									МГТУ им. Н.Э. Баумана			
Утв.												



					Разработка web-приложений реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Голубев В.О.			Приложение. Информационная модель подсистемы журнала событий в РВС GCD			
Провер.		Соколов А.П.		17.06.2020				
Н. Контр.								
Утв.								
						Лит.	Лист	Листов
							5	5
						МГТУ им. Н.Э. Баумана		

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)

АКТ
проверки выпускной квалификационной работы

Студент группы РК6-83Б

Голубев Владислав Олегович
(Фамилия, имя, отчество)

Тема выпускной квалификационной работы: Разработка web-приложений, реализующих бизнес-логику работы пользователя в САПР на основе графоориентированной методологии

Выпускная квалификационная работа проверена, размещена в ЭБС «Банк ВКР» в полном объеме и соответствует / не соответствует требованиям, изложенным в Положении о порядке ненужное зачеркнуть

подготовки и защиты ВКР.

Объем заимствования составляет 1.3% текста, что с учетом корректного заимствования соответствует / не соответствует требованиям к ВКР ненужное зачеркнуть

бакалавра, специалиста, магистра

Нормоконтролёр

С.В. Грошев
(подпись) (ФИО)

Согласен:

Студент

В.О. Голубев
(подпись) (ФИО)

Дата: 14.06.20