Министерство образования и науки 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» (РК)

КАФЕДРА «Системы автоматизированного проектирования» (РК6)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ
по дисциплине:
«Модели и методы анализа проектных решений»

на тему:
«Разработка Web-приложений, реализующих бизнес-логику работы пользователя всистемах инженерного анализа на основе графоориентированной методологии»

Студент РК6-73 (Группа) _____ (Подпись, дата) **В.О. Голубев** (И.О.Фамилия)

Руководитель курсовой работы (проекта) _____ (Подпись, дата) **А.П. Соколов** (И.О.Фамилия)

Консультант _____ (Подпись, дата) **А. Ю. Першин** (И.О.Фамилия)

2020 г.

Министерство образования и науки 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)
_____Карпенко А.П.
(И.О.Фамилия)
« ___ » _____ 20___ г.

З А Д А Н И Е
на выполнение курсового проекта

Студент группы РК6-73_____

___Голубев Владислав Олегович_____
(Фамилия, имя, отчество)

Тема курсового проекта «Разработка Web-приложений, реализующих бизнес-логику работы пользователя в системах инженерного анализа на основе графоориентированной методологии»_____

Направленность курсового проекта (учебная, исследовательская, практическая, производственная, др.)

___практическая_____

Источник тематики (кафедра, предприятие, НИР) кафедра_____

График выполнения КР: 25% к *4* нед., 50% к *8* нед., 75% к *12* нед., 100% к *16* нед.

Техническое задание Создать программную инфраструктуру для приёма, интерпретации и передачи запросов о текущем состоянии процесса выполнения (обхода) графовой модели

Оформление курсового проекта:

Расчетно-пояснительная записка на _21__ листе формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
работа включает 7 рисунков_____
_____
_____

Дата выдачи задания « *9* » сентября 2019 г.

Руководитель курсовой работы _____ А.П. Соколов
(Подпись, дата) (И.О.Фамилия)

Студент _____ В.О. Голубев
(Подпись, дата) (И.О.Фамилия)

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

АННОТАЦИЯ

Данная работа проводилась в рамках разработки Web-приложений, реализующих бизнес-логику работы пользователя в системах инженерного анализа на основе графоориентированной методологии. В работе проведен обзор литературы по теме "Методы визуализации и организации процессов решения сложных вычислительных задач" и на основе анализа существующих разработок представлен и разработан метод решения задачи в рамках технологии GBSE.

СОКРАЩЕНИЯ

CASE — Computer-Aided Software Engineering

UML — Unifed Modeling Language

CBD — Component Based Design

PaaS — Platform as a Service

IaaS — Infrastructure as a Service

SaaS — Software as a Service

СОДЕРЖАНИЕ

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 B.McCormic [2]

Ниже представлен обзор основных работ, в которых рассматриваются 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 (Data Flow Diogram), 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 в последствии привело к методологии Component Based Design [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 вычислительный 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 раздельно. 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 ученные, исследовательские лаборатории и организации пользуются облачными

платформами, которые предоставляют свои вычислительные мощности под любые нужны. 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 подобных 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 специализированны на решении задач машинного обучения [13]. Существует открытая разработка системы машинного обучения TensorFlow от компании Google, вычисления в которой выражаются в виде потоков данных через граф состояний [14]. Среди закрытых разработок стоит упомянуть проект вычислительной платформы российской компании Яндекс - Нирвана [15], 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 [16].

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.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Концептуальная постановка задачи

Аналитический обзор литературы проводился в рамках выполнения работ по разработке Web-приложений, реализующих бизнес-логику работы пользователя в системах инженерного анализа на основе графоориентированной методологии.

Объект разработки:	Web-приложения, реализующие бизнес-логику в системах инженерного анализа на основе GBSE.
Цель разработки:	Создание программного обеспечения реализующую бизнес-логику в PBC GCD.
Ключевые слова:	Визуализация процессов решения задач, методы представления продолжительных операций, computational processes visualization, complex computations visualization, computational pipeline visualization

Задачи курсового проекта:

1. Провести обзор литературы по теме: "Методы визуализации процессов решения сложных вычислительных задач" (не менее 15 источников).
2. Разработать функцию системы (в рамках web-клиента), позволяющую визуализировать состояние выполнения удалённого процессе (в частности, обход тестового графа)
3. Доработка сервера приложени pycomaps в части поддержки приёма и интерпретации запросов о состоянии выполнения процесса с последующим формированием ответа и его отправки на клиентскую сторону.
4. Разработать схему архитектуры подсистемы, обеспечивающей приём-передачу-интерпретацию запросов для обеспечения решения задачи визуализации.

2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Решение реализовано в рамках клиент-серверной архитектуры PBC GCD.

Клиентская часть (comwpc) реализована на языке Python с использованием web-фреймворка Django [21]. Сервер приложений реализован на языке Python. Для работы с сокетами используется модуль pyzmq — реализация технологии ZeroMQ [22] для языка Python. В качестве сервера баз данных используется PostgreSQL Для решения задачи были задействованы модули comsdk и pycomsdk - SDK для программных реализаций сложных вычислительных методов в рамках графоориентированной технологии GBSE на языках C++ и Python. Диаграмма компонентов системы удаленного запуска представлена на рис.1.

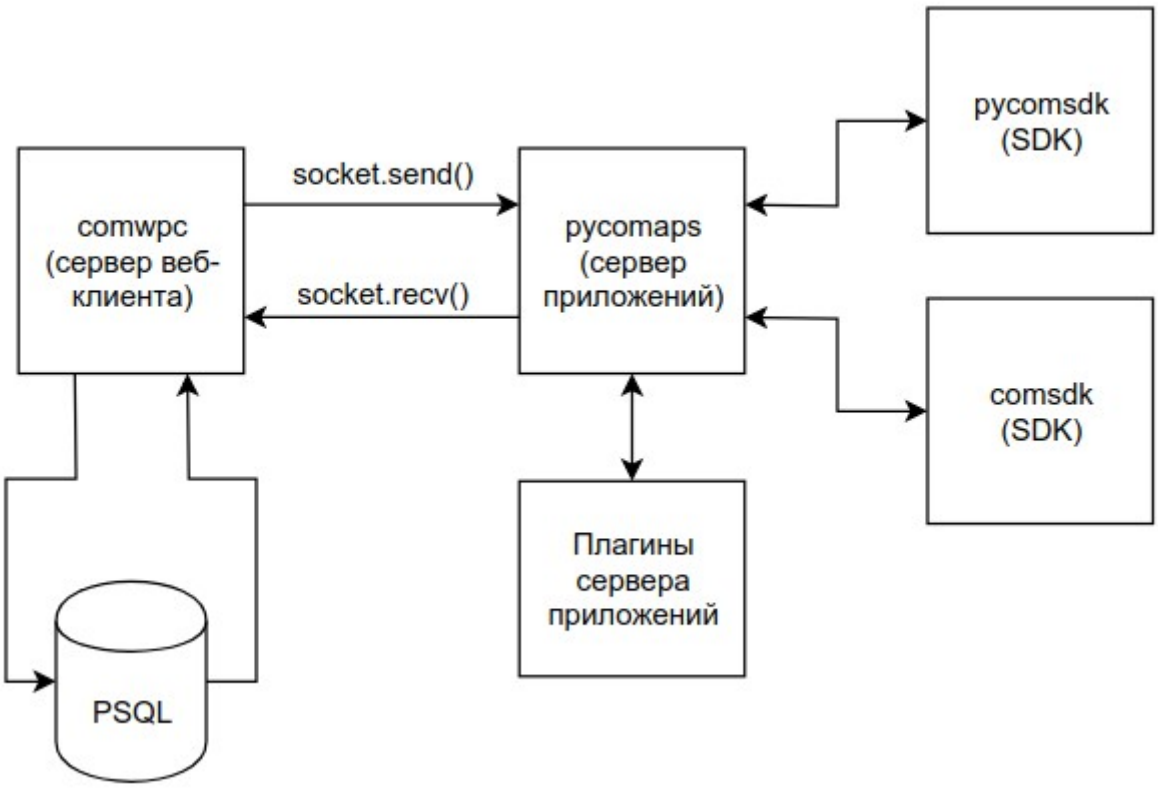

Рис. 1. Диаграмма сетевого взаимодействия компонентов PBC GCD.

2.1. Описание алгоритма удаленного запуска решателя

Веб-клиент использует собственный генератор GUI, который на основе файла исходных данных в формате .slv создаёт GUI пользователя для ввода входных данных, таких как имя решателя и имя файла входных данных для него в формате .tsk. Генератор, также, создает кнопку «Обработать» и привязывает к ней необходимый субплагин на сервере веб-клиента - GRPH_SOLVER_WEB . В субплагине формируется запрос на сервер приложений.

Сервер приложений принимает запрос и создаёт отдельный процесс (worker) для обработки запроса. Worker запускает плагин к серверу приложений для запуска решателя. На основании интерпретации тела запроса плагин получает название aDOT-файла с представлением графовой модели решателя и название файла с исходными данными в формате .tsk. Далее управление передается парсеру файлов формата aDOT из модуля pycomsdk. Парсер представляет из себя экземпляр класса Parser, который имеет несколько методов для обработки файла графовой модели (см. рис. 2). Метод Parser.parse_file() формирует Python-объект Graph, в котором содержится информация о рёбрах и вершинах графа. Метод Parser.generate_cpp() генерирует исходный код решателя на языке C++.

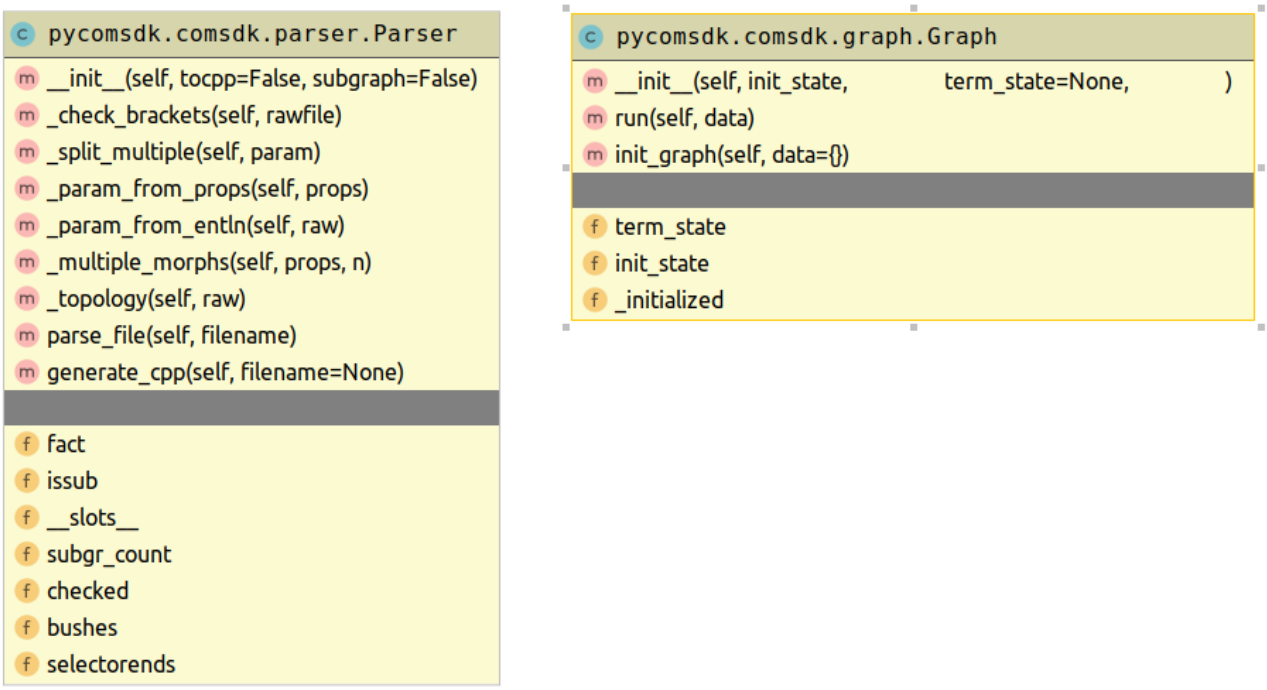

Рисунок 2. UML-диаграмма классов Parser и Graph.

Непосредственный запуск решателя может осуществляться одним из двух способов. Если функции-обработчики и функции-предикаты, привязанные к рёбрам 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.

2.2. Визуализация графовой модели

После удаленного запуска вычислений сервер приложений присылает клиенту текстовое описание графовой модели в формате JSON. Оно составляется из информации, полученной после разбора aDOT-файла на этапе удаленного запуска. Веб-клиент интерпретирует полученное сообщение и визуализирует граф. Визуализация происходит с использованием библиотеки vis.js — реализации мощной библиотеки для визуализации графов для языка JavaScript. Результат визуализации представлен на рисунке .

2.3. Получение статуса решения

Поскольку выполнение решения происходит в отдельном процессе, можно легко следить за его статусом, просматривая потоки stdout и stdterr. Для этого была дополнена реализация генерации исходного кода решателя таким образом, чтобы на выходе после обработки очередного ребра в stdout выводилось сообщение с информацией, которую можно будет интерпретировать как успешное выполнение , а в stderr сообщения об ошибках. Просмотр информации из управляющего процесса осуществляется с помощью объекта исполнительного процесса, у которого существуют поля stdout и stderr. При запросе клиента информации о статусе решения может возникнуть несколько ситуаций:

1. Обход графа ещё не начался
2. Обход графа начался, в потоках stdout и stderr присутствует информация о статусе решения.
3. Обход графа завершен, воркер закончил свою работу и сохранил результат в файл.
4. Воркер закончил работу с ошибкой и сохранил состояние в файл.

В случае первой ситуации, на клиент возвращается сообщение с информацией о том, что ни одно ребро ещё не было пройдено. Во втором случае, возвращается информация из потоков stdout и stderr. В третьем случае, просматривается дамп воркера на присутствие информации о статусе решения и отправляется на клиент. В случае ошибки, на клиент отправляется уведомление об ошибке.

2.4. Интерпретация и визуализация статуса решения

Полученное сообщение состоит из частей, которые описывают порядок обхода графа. После разбора сообщения составляется массив пройденных узлов и с помощью функций библиотеки vis.js находятся соответствующие узлы и прилегающие ребра. Визуализация статуса происходит посредством закрашивания ребер в зеленый цвет в случае успешного прохождения и в

красный в случае не успешного. Не пройденные ребра остаются в начальном состоянии. Результат представлен на рисунке 7.

3. ТЕСТИРОВАНИЕ И ОТЛАДКА

3.1. Общий принцип тестирования

В рамках тестирования реализованного функционала был локально развернут сервер приложений pycomaps и веб-клиент comwpc. Из клиента была вызвана функция GRPH_SOLVER_WEB. Функция WPC_GUI_INI_BUILDER сгенерировала графический пользовательский интерфейс ввода данных для запуска решателя (см. рис. 3). С помощью нажатия кнопки «Обработать» был отправлен запрос на сервер приложений и вызвана функция GRPH_SOLVER_WEB_HANDLER для обработки тестового решателя. Текст aDOT-файла тестового решателя представлен на рисунке 4.

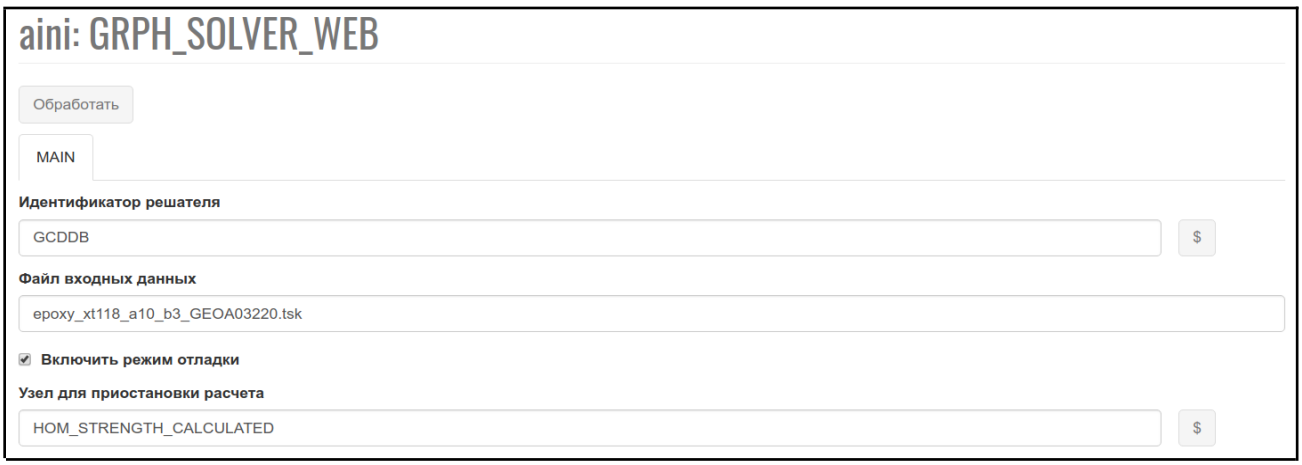

Рисунок 3. Форма ввода сгенерированная из файла в формате .slv.

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 4. Графовая модель тестового решателя в формате aDOT.

3.2. Результаты тестирования

Ожидаемый результат после запуска обработки на удаленный запуск — успешная компиляция сгенерированного cpp-файла исходного кода решателя и успешный запуск программы.

Лог сервера приложений представленный на рисунке 8 показывает, что во время парсинга файла не удалось найти модуль libcomsdk, поэтому был сгенерирован исходный код решателя на языке C++.

```
__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 5. Лог сервера приложений, показывающий процедуру парсинга и генерации исходного кода решателя.

На клиенте был построен граф и через несколько секунд был визуализирован статус решения. Результат представлен на рисунках 9, 10.

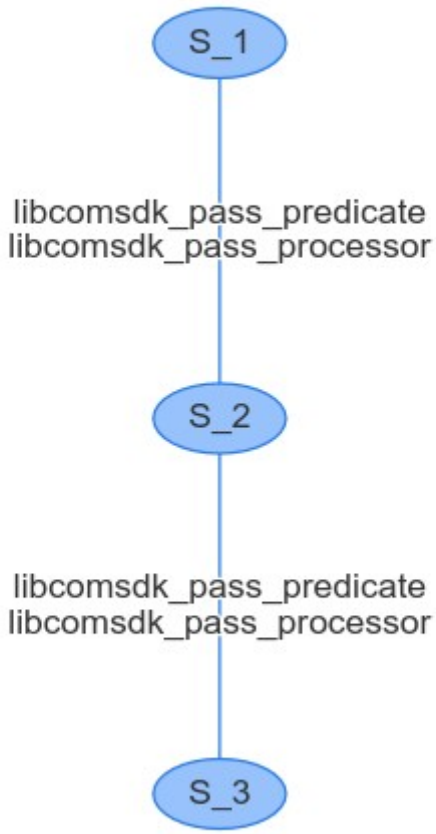

Рисунок 6. Визуальное представление графового решателя. Ребра графа подписаны соответствующими функциями-придикатами и функциями обработчиками.

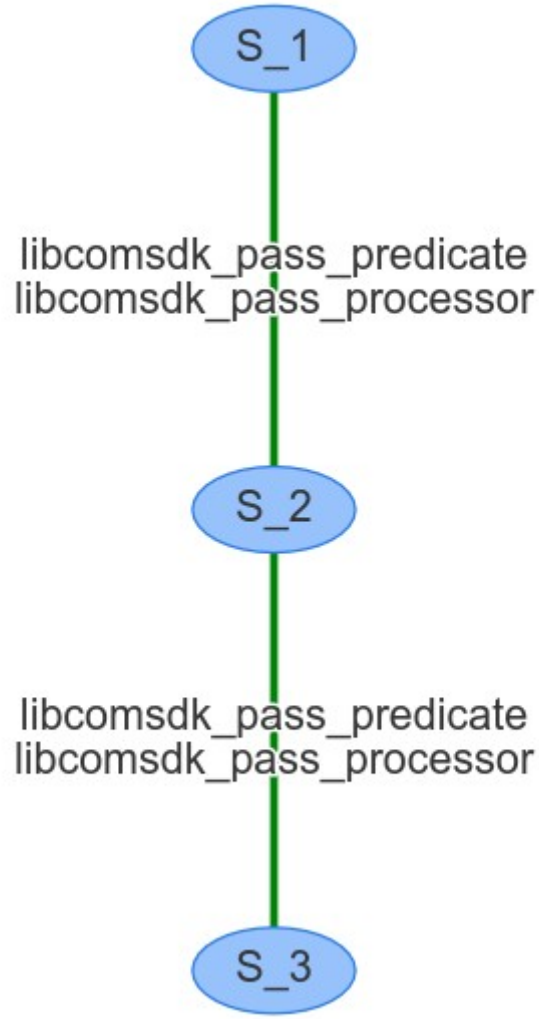

Рисунок 7. Успешный обход всех ребер графа.

4. АНАЛИЗ РЕЗУЛЬТАТОВ

Реализация возможности запускать графоориентированные решатели удалённо позволяет существенно упростить решение сложных вычислительных задач с использованием удалённых многопроцессорных вычислительных машин, имеющих большие мощности, чем домашние компьютеры. Это в свою очередь позволит существенно ускорить и упростить проведение вычислительных экспериментов и научных исследований. Разработанный подход поддается легкой модификации, что позволит в будущем усовершенствовать его или адаптировать под новый функционал PBC GCD. На основе разработанного подхода можно спроектировать архитектуру удаленного запуска решателе на любой вычислительной машине или кластере вне

серверной среды PBC GCD. Предпоссылками к этому являются разработанный парсер aDOT-файлов с возможностью генерации исходного кода решателя и большое количество библиотек функций-обработчиков.

Разработанный метод получения статуса рещения и визуализации процесса решения сложной вычислительной задачи позволяет наглядно продемонстрировать специалисту процесс решения задачи. Архитектура решения позволяет разработать специальные функции, которые смогут использовать визуальное представление графа для построение бизнес-логики в системе PBC GCD.

5. ЗАКЛЮЧЕНИЕ

1. Был проведен обзор научно-технических публикаций и патентов по теме исследования.
2. Приведены современные перспективные разработки в области организации процессов сложных вычислительных задач.
3. Проанализированы существующие методы, выделены их достоинства.
4. Предложен иразработан метод визуализации и организации процессов сложных вычислительных задач в рамках технологии GBSE.

СПИСОК ЛИТЕРАТУРЫ

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. Mcllroy 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 c.
5. US 2004/0056908 Al - Turbo Worx, Inc. (New Haven, CT, US) - Method and system for dataflow creation and execution (2004).

6. US 8,949,772 Bl - 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.: ИПМ 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 интелект. - 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 programming through integration of deep models and small-scale machine learning // Nat Commun. - Volume 10, 2019, C. 33-40.
12. Kudryavtsev A.O., Koshelev V.K., Izbyshev 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.
13. Janez Demsar, Tomaz Curk, Ales Erjavec Orange: Data Mining Toolbox in Python // Journal of Machine Learning Research, 2013 (14), С. 2349–2353.
14. Официальный сайт проекта TensorFlow [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org.
15. Познаём Нирвану – универсальную вычислительную платформу Яндекса [Электронный ресурс]. –Режим доступа: https://habr.com/ru/company/yandex/blog/351016/
16. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. Программирование. – Т.47, No5– 2019, с. 43-55.