|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 программной реализация удалённого***

***запуска графоориентированных решателей систем инженерного анализа»***

Студент \_ РК6-82\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_**И.С.Громов\_\_

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_**А.П.Соколов\_

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_С.В.Грошев\_\_

(Подпись, дата) (И.О.Фамилия)

*2019 г.*

# **АННОТАЦИЯ**

Выпускная квалификационная работа выполнялась в рамках разработки модуля РВС GCD, реализующего удалённый запуск произвольного графоориентированного решателя.

Реализованная технология позволяет систематизировать процесс разработки программных реализаций сложных вычислительных методов и ускорить процесс проведения вычислительных экспериментов и различных исследований.

Тип работы: выпускная квалификационная работа.

Тема работы: Разработка программной реализации удалённого запуска графоориентированных решателей систем инженерного анализа.

Объект исследований: Решения в области реализации удалённого запуска процедур на распределённых вычислительных системах.

Работа содержит 40 страниц, 5 разделов, 7 иллюстраций, 4 листинга, 1 блок-схему. Выполнена с использованием 15 источников.

# **СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ**

РВС GCD – Распределённая Вычислительная Система GCD;

GUI – графический пользовательский интерфейс (англ. Graphical User Interface);

ОС – операционная система;

БД – база данных;

GBSE – графоориентированная программная инженерия (англ. Graph Based Software Engineering);

СУБД – система управления базами данных;

aDOT – формат описания графовых моделей сложных вычислительных методов (англ. Advanced DOT);

aINI – формат представления исходных данных (англ. Advanced INI);

RPC – Удалённый запуск процедур (англ. Remote Procedure Call).;

CMake – Кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода;

Boost – набор библиотек для языка программирования *C*++;

Плагин – автоматически подключаемый программный модуль (англ. Plug-IN).

# **СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 9](#_Toc12233154)

[**1.** **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ** 10](#_Toc12233155)

[**1.1.** **Актуальность объекта исследований и его развитие** 10](#_Toc12233156)

[**1.2.** **Современные разработки в области объекта исследований** 13](#_Toc12233157)

[**1.3.** **Описание Графоориентированной модели** 15](#_Toc12233158)

[**2.** **КОНЦЕПТУАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ** 18](#_Toc12233159)

[**3.** **АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ** 19](#_Toc12233160)

[**3.1.** **Схема архитектуры** 19](#_Toc12233161)

[**3.2.** **Описание архитектуры** 20](#_Toc12233162)

[**3.3.** **Использованные технологии** 20](#_Toc12233163)

[**4.** **ПРИНЦИП РАБОТЫ ПРИЛОЖЕНИЯ** 22](#_Toc12233164)

[**4.1.** **Описание реализованного плагина** 22](#_Toc12233165)

[**4.2.** **Алгоритм работы плагина** 23](#_Toc12233166)

[**4.3.** **Программная реализация плагина** 25](#_Toc12233167)

[**4.4.** **Регистрация плагина в системе** 25](#_Toc12233168)

[**5.** **ТЕСТИРОВАНИЕ И ОТЛАДКА** 28](#_Toc12233169)

[**5.1.** **Общий принцип тестирования** 28](#_Toc12233170)

[**5.2.** **Описание тестового решателя** 28](#_Toc12233171)

[**5.3.** **Описание функции тестирования** 30](#_Toc12233172)

[**5.4.** **Порядок проведения тестирования** 31](#_Toc12233173)

[**5.5.** **Результаты тестирования** 32](#_Toc12233174)

[**ЗАКЛЮЧЕНИЕ** 36](#_Toc12233175)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 37](#_Toc12233176)

[**ПРИЛОЖЕНИЕ А** 39](#_Toc12233177)

[**ПРИЛОЖЕНИЕ Б** 40](#_Toc12233178)

[**ПРИЛОЖЕНИЕ В** 43](#_Toc12233179)

[**ПРИЛОЖЕНИЕ Г** 45](#_Toc12233180)

[**ПРИЛОЖЕНИЕ Д** 47](#_Toc12233181)

**ВВЕДЕНИЕ**

Необходимость реализации удалённого выполнения вычислительных процессов возникла тогда, когда ПК и различные вычислительные системы стали использоваться для решения ресурсоёмких задач и мощностей одной локальной машины перестало хватать для выполнения сложных вычислительных операций. Для решения этой проблемы была реализована технология удалённого запуска процедур RPC.

Применение этой технологии необходимо, при разработке распределённых вычислительных систем, ключевой особенностью которых является выполнение одной программы на различных, физически удалённых друг от друга машинах, чтобы распределить вычислительные затраты, на выполнение вычислений равномерно распределить между узлами системы и добиться наивысшей скорости работы и производительности.

При решении сложных ресурсоёмких задач зачастую вычислительных мощностей локального персонального компьютера уже не хватает и возникает потребность в использовании многопроцессорных вычислительных систем. Задача направлена на задействование разработанной в МГТУ им. Н.Э. Баумана специальной технологии построения программных реализаций сложных вычислительных методов, основанной на применении понятий теории графов.

1. **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**
   1. **Актуальность объекта исследований и его развитие**

Протокол RPC построен на модели вызовов удалённых процедур, подобному механизму вызовов локальных процедур. При вызове локальной процедуры мы помещаем аргументы в определенное место памяти, в стек или переменные окружения и передаём управление процессом по определенному адресу. После завершения работы мы читаем результаты по конкретному адресу и продолжаем свой процесс.

В случае работы с удалённой процедурой основное отличие состоит в том, что вызов удалённой функции обслуживают два процесса: клиентский процесс и серверный процесс.

Процесс клиента отправляет серверу сообщение, в которое включены параметры вызываемой процедуры и ожидает ответного сообщения с результатами её работы. При получении ответа результат считывается, и процесс продолжает работу. Со стороны сервера процесс-обработчик вызовов находится в состоянии ожидания, и, при поступлении сообщения, считывает параметры процедуры, выполняет её, отправляет ответ и становится в состояние ожидания следующего вызова.

RPC-протокол не накладывает каких-либо требований на дополнительные связи между процессами и не требует синхронности выполняемых функций, т. е. вызовы могут быть асинхронными и взаимонезависимыми, так что клиент во время ожидания ответа может выполнять другие процедуры. Сервер RPC может выделять для каждой функции отдельный процесс или виртуальную машину, поэтому, не дожидаясь окончания работы предыдущих запросов, сразу же может принимать следующие.

При проектировании и создании первых реализаций технологии RPC препятствовал ряд проблем и трудностей, таких как:

1. Неоднородность сред выполнения процедур. На разных машинах могут быть установлены различные ОС, организация работы которых может различаться;
2. Разные адресные пространства двух машин;
3. Вероятность большого объёма передаваемых данных, что очень сильно нагружает механизм передачи информации;
4. Механизм обработки ошибок, чтобы предусмотреть случаи аварийного завершения работы на одной из машин.

В результате был спроектирован классический механизм удалённого запуска процедур. Принцип его работы изображён на рисунке 1. В его основе лежало использование «стабов» (англ. Stub – заглушка). При такой реализации машина-клиент и машина-сервер не знают о существовании друг друга и работают в обычном режиме, а в «стабах» хранится задаваемая разработчиком логика обработки событий и ответа на них [5].

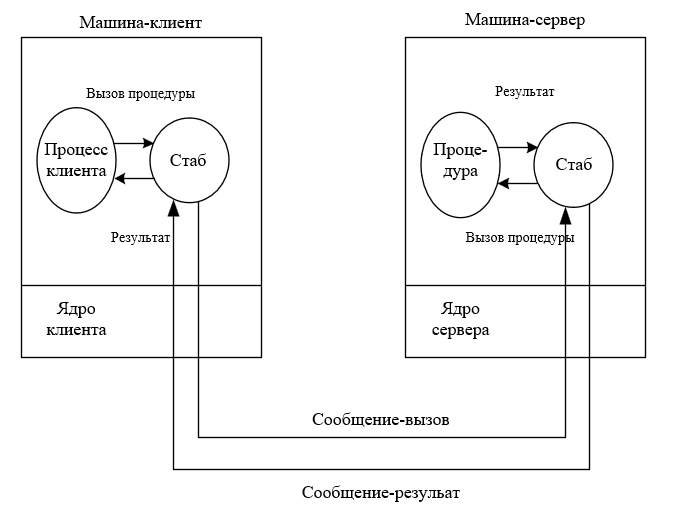


Рис. 1 – Принцип работы RPC

Известны различные варианты архитектур распределённых вычислительных систем, использующих механизмы удалённого запуска ресурсоёмких вычислительных процессов. В основном, все рассмотренные реализации основаны на применении уже готовых технологиях удалённого вызова процедур, таких как: GridMD [1], XML-RPC [2], собственных библиотек на C++, реализующих удалённый запуск процедур [3], COM и CORBA [6].

Первые реализации RPC были созданы в 70-x - 80-х годах 20 века с помощью C/C++. В общем случае механизм включает в себя:

1. транспортную подсистему – механизм передачи данных по сети (TCP, UDP, реже HTTP);
2. сериализатор – механизм преобразования входных данных процедуры, вызываемой удалённо, в формат, пригодный для передачи по сети;
3. очередь потоков вызываемой стороны – механизм определения процедуры удалённого сервера, которая должна быть вызвана.
4. Шифрование пакетов и наложение на них цифровой подписи.
5. Аутентификация и авторизация. Передача по сети информации, идентифицирующей субъект, осуществляющий вызов.

Позже механизм RPC был усовершенствован. Взамен подсистемы сериализации была создана подсистема «маршалинга» - процесс более высокого уровня, который позволяет передавать по сети помимо данных их кодовую базу данных, а также ссылку на объект, не передавая сам объект. Для передачи данных стали использоваться современные структуры данных (JSON, XML).

В РВС GCD используется самостоятельная реализация RPC на языке C++ с использованием формата представления данных *AnyMap* для передачи информации между различными модулями системы.

* 1. **Современные разработки в области объекта исследований**

Параллельно с усовершенствованиями технологии RPC были разработаны более прогрессивные реализации удалённого вызова процедур. Основное их различие в областях применения. Каждая из реализаций создавалась под конкретную задачу. Не существует абсолютно универсальной реализации, тестирования производительности различных RPC проводились многократно [4]. В конкретном классе задач какая-то из них показывает себя наилучшим лучшим образом. Наиболее современные и перспективные из них представлены ниже:

1. Protocol Buffers – независимый от среды выполнения и языка реализации механизм Google для сериализации структурированных данных. Основная его особенность в том, что разработчик определяет формат данных для конкретной задачи, а механизм формирует код для работы с этими данными на любом из популярных языков программирования, также выбранном пользователем;
2. Cap`n Proto - развитие технологии Protocol Buffers, однако в данной реализации был усовершенствован ещё и механизм передачи данных между клиентом и сервером. Основная особенность которого заключается в том, что результаты вызова функций немедленно возвращаются клиенту, однако, это целесообразно только для составных запросов, где результат одной части используется для выполнения других частей запроса [7]. Для понимания, на рисунке 2 представлен принцип работы Cap`n Proto;

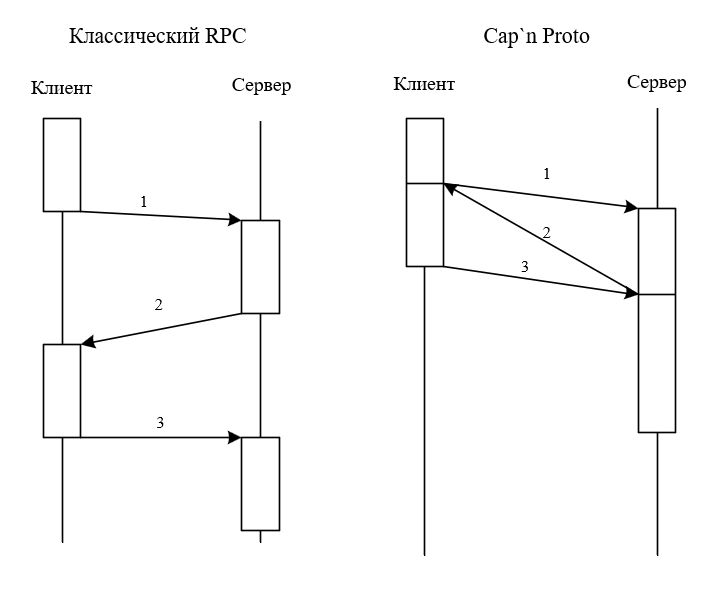


Рис 2. Сравнение Cap`n Proto c обычным RPC, где  
1- вызов какой-то функции  
2 – возврат значения этой функции

3 – вызов новой функции, использующей возвращённое значение

1. gRPC - современная реализация устаревших RPC от Google. Основные принципы в данной реализации были сохранены, однако были использованы современные технологии для их реализации, что позволило получить высокую производительность и независимость от платформы и используемого языка программирования [8];
2. Finagle – RPC от компании Twitter. Основная особенность – высокая безопасность, отказоустойчивость, стабильная работа при больших нагрузках, что делает данный механизм удобным для использования в высоконагруженных системах.
3. Thrift – Механизм, разработанный компанией Apache. Основная особенность – возможность использования различных форматов передачи данных и протоколы связи, что делает его достаточно гибким и адаптируемым под различные решения [9].
   1. **Описание Графоориентированной модели**

Способ и система графоориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов, позволяющая упростить процессы проектирования, разработки, тестирования, сопровождения программных реализаций сложных вычислительных методов [12].

Способ графоориентированного создания программных реализаций сложных вычислительных методов включает построение и последующую автоматическую интерпретацию графического представления алгоритма в форме ориентированного графа, реализующего рассматриваемый сложный вычислительный метод, и заключается в выполнении последовательности действий [15]:

1. создание формата файлов входных данных для создаваемой программной реализации сложного вычислительного метода;
2. создание графовой модели алгоритма, реализующего сложный вычислительный метод, в форме ориентированного графа, для которого каждый узел имеет уникальное имя и определяет фиксированное состояние общих данных, а с каждым ребром связана функция, называемая функцией перехода, которая определяет преобразование общих данных из одного состояния в другое;
3. разработка программных реализаций функций перехода, определяемых графовой моделью, причём каждая функция перехода определяется парой функций с унифицированными сигнатурами: функцией-предикатом и функцией-обработчиком;
4. отладка и тестирование разработанной программной реализации сложного вычислительного метода [14].

Концептуальная схема обработки графовой модели алгоритма, реализующего конкретный вычислительный метод представлена на рисунке 3.

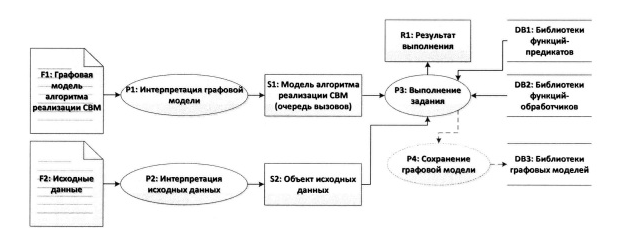


Рис. 3 - Концептуальная схема обработки графовой модели алгоритма, реализующего конкретный вычислительный метод

На рисунке 4 представлен пример графовой модели алгоритма, реализующего конкретный вычислительный метод, где буквами “S” обозначены состояния данных на текущем этапе выполнения алгоритма, и выделены начальное – “S1” и конечное - “S7” состояния, а буквами “F” – функции перехода модели данного алгоритма [11].

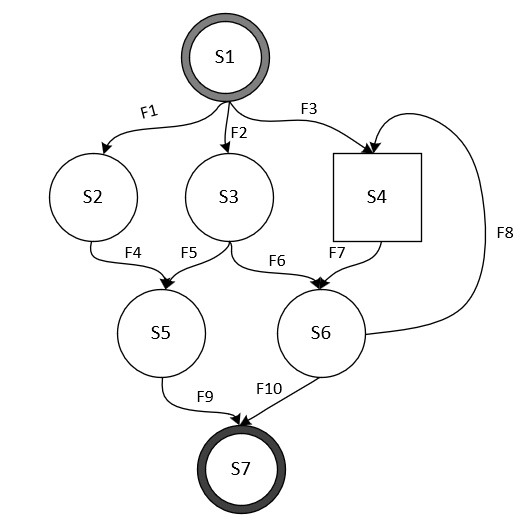


Рис. 4 – Пример графовой модели алгоритма, реализующего конкретный вычислительный метод.

В РВС GCD графовая модель алгоритмов, реализующих вычислительные методы, была расширена добавлением функций-предикатов, которые определяют возможность выполнения функций перехода (возможность перехода модели в следующее состояние). Для хранения самой графовой модели используется разработанный в МГТУ им. Н.Э. Баумана формат файла – *aDOT* [12, 13].

1. **КОНЦЕПТУАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ**

Целью разработки является: создать программное обеспечение удалённого запуска произвольных графоориентированных решателей РВС GCD, а именно:

* 1. Разработать плагин к серверу приложений РВС GCD на языке С++, позволяющий запустить произвольный решатель системы РВС GCD удалённо.
  2. Создать тестовый решатель и протестировать реализованный плагин, запустив этот решатель.

Были выделены следующие задачи:

* 1. Проведение аналитический обзор литературы по теме “ Технологии и методы удалённого запуска процедур и функций на высокопроизводительных вычислительных системах”.
  2. Разработка архитектуры программного обеспечения удалённого запуска графоориентированных решателей.
  3. Разработка реализации программного обеспечения в соответствии с созданной архитектурой.
  4. Тестирование полученных результатов.

1. **АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ**
   1. **Схема архитектуры**

В приложении А представлена процедура удалённого запуска решателя в рамках РВС GCD. Она состоит из трёх компонентов:

1. локальная клиентская машина пользователя, через которую осуществляется доступ к web-клиенту системы и на которой определяются исходные данные для выполнения задач в РВС GCD;
2. удалённый сервер, на котором располагается web-клиент системы РВС GCD. На этом узле системы реализуется формирование GUI, который отображается на локальной машине пользователя при доступе к РВС GCD, определение исходных данных, путём заполнения специальной сформированной на основе файла aINI формы, сериализация этих данных и отправка их на сервер приложений (*comaps*), а также отправка команды на запуск функций на сервере приложений;
3. удалённая вычислительная система, состоящая из множества узлов, на которых развернут сервер приложений РВС GCD и дополнительные его модули, в виде плагинов. В этой части системы выполняется приём данных от web-клиента (comwpc) РВС GCD, десериализация данных, обработка запроса на выполнение указанной в запросе функции, запуск её выполнения, получение сопутствующих данных из хранилищ, формирование ответа, содержащего результаты выполнения функции и отправка его обратно на web-клиент. Также здесь хранятся бинарные пакеты существующих в системе решателей;
4. База данных РВС GCD, где содержится информация обо всех существующих решателях системы и соответствующих им графовых моделях, а также информация о связи функций сервера приложений с соответствующими им функциями на web-клиенте.
5. также компонентами системы являются удалённые хранилища, содержащие исходные данные для решателей (файлы в формате aINI), графовые модели существующих в системе решателей (файлы в формате aDOT) и результаты работы выполненных решателей.
   1. **Описание архитектуры**

Web-клиент РВС GCD (*comwpc*) использует собственный генератор GUI, который на основе файла исходных данных в формате .*slw* создаёт GUI пользователя для ввода входных данных, таких как имя решателя и имя файла входных данных для него в формате .*tsk*. Затем эти данные «сериализуются» путём помещения в объект класса *AnyMap,* который содержит помимо ссылки на файл с входными данными в хранилище подобных файлов ссылку на ID «решателя» в Базе данных «решателей» (*slvid*), и отправляются в удалённую вычислительную систему.

Сервер приложений(*comaps*) «десериализует» принятые данные и обращается к плагину, отвечающему за запуск «решателей» с командой на запуск. После чего плагин проверяет входные данные, соответственно спроектированному алгоритму ищет необходимый «решатель» в Базе данных решателей. Если «решатель» найден, то запрашивает его функцию из хранилища библиотек функций решателей и переходит к стадии запуска. На этом этапе «решателю» на вход подаются файл исходных данных в формате *.tsk*, который был выбран пользователем, а также ID решателя в системе РВС GCD на основе которого определяется имя файла графовой модели этого «решателя» в формате .*adot* [13], который взят из хранилища графовых моделей решателей. После выполнения «решателя» результат его работы помещается в хранилище результатов работы решателей, и работа плагина завершается.

* 1. **Использованные технологии**

В качестве инструментов подразумевается использование специально разработанного контейнера - словаря *AnyMap*, позволяющего хранить одновременно данные различного типа. Данный словарь будет хранить информацию, считываемую из файла исходных данных специального формата aINI [10]. Формат позволяет интуитивно понятно задавать исходные данные различных типов в текстовом файле, что используется при выполнении инженерного анализа, вычислительных экспериментов в РВС GCD конечными пользователями системы. Инструментарий перевода содержимого aINI-файла в контейнер *AnyMap* также реализован в библиотеках РВС GCD.

Для представления графовой модели решателя был использован формат *aDOT,* специально разработанный для представления моделей вычислительных методов в РВС GCD [13].

1. **ПРИНЦИП РАБОТЫ ПРИЛОЖЕНИЯ**
   1. **Описание реализованного плагина**

В момент компиляции сервера приложений РВС GCD производится сканирование его директорий на наличие новых исполняемых модулей, после чего они регистрируются в приложении и могут быть использованы. Реализуемый плагин имеет название *comaps\_cpl\_GraphSolverWebHandler*. При его использовании вызывается метод *execute()*, который принимает на вход объект класса *AnyMap* со входными данными, а также сообщение для обратной связи с вызывающей его стороной. В объекте *AnyMap* находится имя выбранного решателя, его Id в базе данных и имя файла с исходными данными, необходимыми для его выполнения. На стороне WEB-клиента вызов инициируется посредством кнопки «обработать», на странице функции запуска графоориентированных решателей с использованием WEB-клиента (функция GRPH\_SOLVER\_WEB) вычислительной подсистемы (GCDDBSFEA) Связывание плагина с функцией WEB-клиента происходит на уровне Базы данных РВС GCD. В таблицу функций WEB-клиента добавлено название функции на сервере приложений, которая обрабатывает данный запрос.

Структура проекта *comaps* с учетом реализованного плагина представлена на рисунке 5:

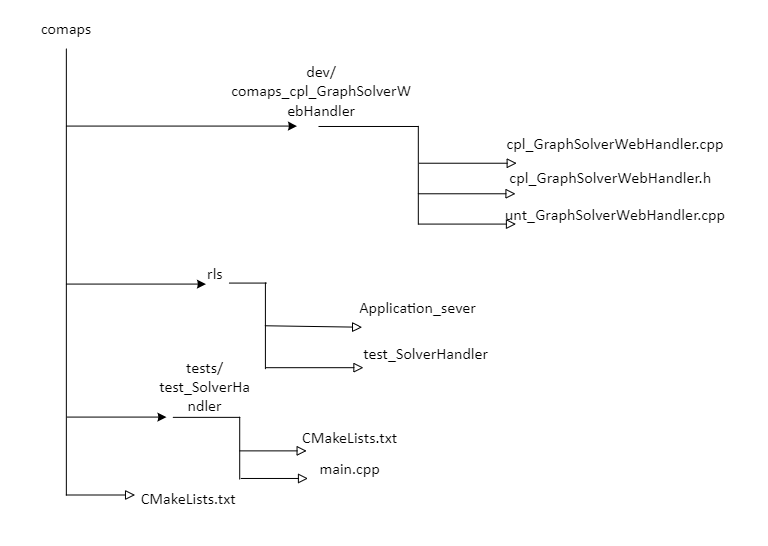


Рис. 5 Структура файлов плагина на сервере приложений

* 1. **Алгоритм работы плагина**

После начала работы плагина, внутри него происходит проверка входных данных на их наличие и целостность (алгоритм проверки описан блок-схемой 1) и при их отсутствии возвращается ошибка, притом для каждой из возможных ошибок возвращается свой код, чтобы можно было определить место их возникновения. В случае корректных входных данных происходит запуск и выполнение «решателя». А также, в процессе выполнения происходит определения файла графовой модели выполняемого решателя, на основе ID этого самого решателя путём выполнения SQL-запроса к базе данных РВС GCD. В таблице com.slvrs хранятся записи обо всех существующих в системе решателях, поэтому, зная ID решателя (com.slvrs.slver) Мы легко можем узнать имя файла его графовой модели (com.slvers.adotn).

Результат работы плагина, как положительный, так и отрицательный, сохраняются в специальном сообщении p\_callback, которое будет возвращено вызывающей стороне после выполнения.

Page-1

Блок-схема 1. Алгоритм работы плагина.

* 1. **Программная реализация плагина**

Для реализации плагина были использованы структуры данных из библиотеки *comSDK*:

1. *AnyMap*, используемый для представления данных, взятых из формата *aINI*;
2. *graph::Nod*e для хранения начального состояния графовой модели решателя и его запуска;
3. *onMessageCallback* для хранения сообщений, возвращаемых сервером приложений, будь то ошибка или результат выполнения решателя.

Для работы с БД *PostgreSQL* была использована библиотека *postgresql/libpq-fe.h*, с помощью которой был реализован доступ к БД РВС GCD и выполнение запроса на получение файла графовой модели решателя, а именно:

1. установление соединения с базой данных, посредством функции *PQconnectdb*;
2. выполнение SQL-запроса на получение имени файла графовой модели решателя из БД, посредством функции *PQexec*;
3. получение имени файла из объекта, возвращённого при выполнении запроса с помощью функции *PQgetvalue*.

Для выполнения проверок на наличие необходимых для запуска решателя файлов и содержимого в них, а также хранения промежуточных данных и их передачи различным функциям и методам были использованы средства работы с файловыми потоками из C++ Standart Library.  
Исходный код плагина представлен на листинге 1 (Приложение А).

* 1. **Регистрация плагина в системе**

Функции для регистрации плагина на сервере приложений, его деинициализации и хранения кодов, возвращаемых плагином, был написан файл *comfrm\_cls\_GraphSolverWebHandler.h.* Его содержимое представлено на листинге 2 (Приложение В). В нём происходит присваивание регистрируемому в системе плагину его уникального идентификатора в системе (В данном случае “GRAPH\_SOLVER”) и идентификатор принадлежности модулю РВС GCD (В данном случае “COMAPS”). А также описан прототип функции запуска плагина и возможные коды, возвращаемые плагином при завершении его работы. Возможные коды, возвращаемые плагином представлены в таблице 1:

Таблица 1. Коды завершения работы плагина

|  |  |  |
| --- | --- | --- |
| Код возврата | Название кода | Описание кода |
| 0 | ecSuccess | Успешное завершение работы плагина. |
| 100 | ecEmptyData | Ошибка чтения содержимого входного объекта класса *AnyMap.* |
| 101 | exFileNotFound | Ошибка открытия файла исходных данных или файла графовой модели выбранного решателя (такого файла не существует). |
| 102 | ecFileNotRead | Ошибка чтения данных из файла исходных данных или файла графовой модели выбранного решателя (файл повреждён). |
| 200 | exExecError | Ошибка выполнения выбранного решателя. |

1. **ТЕСТИРОВАНИЕ И ОТЛАДКА**
   1. **Общий принцип тестирования**

В рамках тестирования реализованного функционала была написана функция сервера приложений, которая тестирует разработанный плагин, подавая на вход ему структуру данных *AnyMap,* которая, в случае корректного запуска плагина, должна выглядеть как показано на рисунке 6.

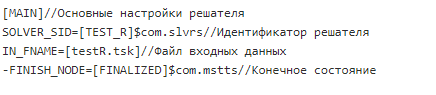


Рис.6 Структура файла testR.slw

В рамках тестирования был локально скомпилирован и развернут сервер приложений РВС GCD и на нём был произведён запуск тестового решателя под названием test\_R с помощью тестирующей функции, структура объекта *AnyMap,* используемого при тестировании находится в файле testR.slw, его структура представлена на рис.6, где:

SOLVER\_SID – ID решателя в БД РВС GCD.

IN\_FNAME – имя файла входных данных для решателя.

Данный файл настроек для решателя не нуждается в указании файла графовой модели выбранного решателя, так как это имя будет получено путём SQL-запроса при выполнении плагина.

* 1. **Описание тестового решателя**

На основе описанного выше файла должен удалённо запускаться тестовый решатель TEST\_R, исходный код которого представлен в листинге 3 (Приложение Г).

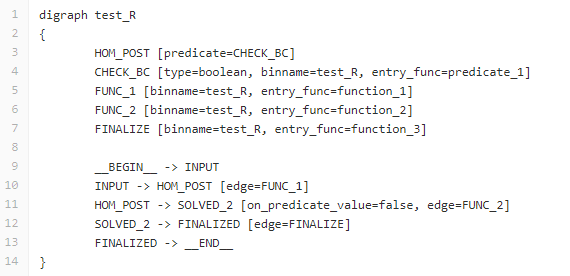
Созданному тестовому решателю соответствует файл графовой модели *digraph\_test.аdot*, содержимое которого представлено на рисунке 6. А также файл исходных данных, *testR.tsk*, который не используется конкретно в этом решателе, однако нужен для проверки доставки данных на сервер приложений. 

Рис. 7 Содержимое файла графовой модели тестового решателя

Визуальное представление графовой модели тестового решателя представлено на рисунке 8.

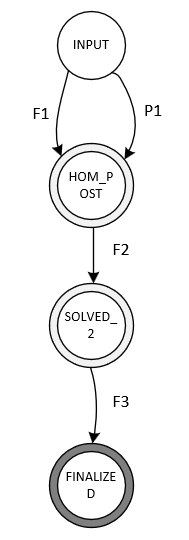


Рис 8. Визуальное представление графовой модели тестового решателя.

В данной модели определены 4 состояния: INPUT, HOM\_POST, SOLVED\_2, FINALIZED. Для этих состояний определены функции перехода F1, F2, F3, а также функция-предикат P1, которым соответствуют функции из текстового представления графовой модели:

F1 – FUNC\_1;

F2 – FUNC\_2;

F3 – FINALIZE;

P1 – CHECK\_BC.

В данной графовой модели все функции реализованы, как функции-заглушки, которые не принимают и не передают входных данных, а лишь сигнализируют о своём выполнении в текстовом виде.

* 1. **Описание функции тестирования**

Для тестирования реализованного плагина была написана функция для сервера приложений *test\_solverHandler* на языке С++, которая запускает выполнение плагина с произвольным объектом *AnyMap,* определяемым пользователем, для задания входного объекта необходимо изменить содержимое структуры данных *LauncAI,* записав туда все необходимые для выполнения плагина данные. Исходный код этой функции представлен на листинге 4 (Приложение Д). Принцип работы тестовой функции:

1. Проверка наличия корректности данных, поданных на вход (ID тестовой функции в системе IP сервера приложений и его ID в системе);
2. Подключение к серверу приложений;
3. Инициализация объекта *LaunchAI* с исходными данными для тестируемого плагина;
4. Запуск тестируемого плагина с подачей ему на вход объекта *LaunchAI;*
5. Проверка результата выполнения тестируемого плагина, обработка ошибок;
6. Вывод сообщения, об успешном выполнении тестируемого плагина или возврат кода ошибки.

Для запуска проверки необходимо задать в исходном коде объект *LaunchAI* который будет предан тестируемому плагину, а также, при запуске тестирующей функции, в консоли указать IP сервера приложений (для локально развёрнутого это 192.168.1.100), ID сервера приложений на конкретной машине (может быть любым) и ID программы тестирования (может быть любым).

* 1. **Порядок проведения тестирования**

Для тестирования реализованного плагина необходимо скомпилировать сервер приложений *comaps* с предварительно добавленными в его директории файлами исходного кода решателя, а также функции тестирования в соответствии со структурой приложения *comapps,* представленной на рисунке 5. После чего необходимо в корневой папке приложения создать папку *build*, перейти в неё и выполнить следующую команду:

cmake -DCMAKE\_BUILD\_TYPE=Release ..

где флаг DCMAKE\_BUILD\_TYPE отвечает за тип сборки проекта, возможные типы:

Debug – бинарные файлы сервера приложений будут находиться в папке dbg и будут доступны для запуска и отладки.

Release - бинарные файлы сервера приложений будут находиться в папке rls и будут доступны для запуска и отладки.

После чего необходимо выполнить команду:

Make

Которая соберёт исходные файлы сервера приложений в исполняемые бинарные пакеты.

Чтобы запустить сервер приложений, необходимо перейти в директорию rls и выполнить команду:

./Application\_server 1104, где

*1104* – ID сервера приложений в системе.

*Application\_server* – имя бинарного исполняемого пакета, скомпилированного из файлов исходного кода сервера приложений *comaps.*

После чего запустить исполняемый бинарный пакет, скомпилированный из исходного кода функции тестирования работы плагина командой:

./test\_SolverHandler 1105 192.168.1.100/1104, где

*1105* – ID функции тестирования в системе.

*192.168.1.100/1104* – IP адрес запущенного сервера приложений/ его ID в системе.

*Test\_SolverHandler* – имя исполняемого бинарного пакета, собранного из исходных файлов функции тестирования разработанного плагина.

* 1. **Результаты тестирования**

В результате тестирования были проверены три возможных варианта выполнения плагина, а именно:

1. На вход плагину подан объект класса *AnyMap*, не содержащий данных – В данном случае работа плагина должна завершиться с кодом ошибки 100 (таблица 1), а тестирующая функция должна вернуть сообщение об ошибке в работе плагина с указанием кода ошибки;
2. На вход плагину подан объект класса , содержащий неверное имя файла входных данных – В данном случае работа плагина должна завершиться с кодом ошибки 102 (таблица 1), а тестирующая функция должна вернуть сообщение об ошибке в работе плагина с указанием кода ошибки;
3. На вход плагину подан объект класса , содержащий корректные данные – В данном случае работа плагина должна завершиться без ошибок, а тестирующая функция должна вывести в консоль сообщение об успешном завершении работы плагина.

Результаты работы тестирующей функции представлены ниже:

igromov@igromov-01pc:~$ ./test\_solverHandler 1105 192.168.1.100/1104

Test client SID: 1105

Application server SID: 192.168.1.100/1104

Test client starting an io service…

Something went wrong while executing Graph Solver Plugin. Error code: 100

Test client stopping an io service…

где

*Test client SID: 1105* – сообщение, означающее инициализацию функции тестирования и присвоение ей ID процесса равного 1105.

*Application server SID: 192.168.1.100/1104* – сообщение, означающее, что тестовая функция определила сервер приложений и подключилась к нему.

*Test client starting an io service…* - сообщение, означающее, что тестируемый плагин начал своё выполение.

*Something went wrong while executing Graph Solver Plugin. Error code: 100* – сообщение, означающее, что тестируемый плагин в процессе своего выполнения вернул сообщение об ошибке с кодом ошибки 100 (таблица 1).

*Test client stopping an io service…* - сообщение, означающее, что выполнение тестируемого плагина завершено.



igromov@igromov-01pc:~$ ./test\_solverHandler 1105 192.168.1.100/1104

Test client SID: 1105

Application server SID: 192.168.1.100/1104

Test client starting an io service…

Something went wrong while executing Graph Solver Plugin. Error code: 102

Test client stopping an io service…

где

*Test client SID: 1105* – сообщение, означающее инициализацию функции тестирования и присвоение ей ID процесса равного 1105.

*Application server SID: 192.168.1.100/1104* – сообщение, означающее, что тестовая функция определила сервер приложений и подключилась к нему.

*Test client starting an io service…* - сообщение, означающее, что тестируемый плагин начал своё выполение.

*Something went wrong while executing Graph Solver Plugin. Error code: 100* – сообщение, означающее, что тестируемый плагин в процессе своего выполнения вернул сообщение об ошибке с кодом ошибки 102 (таблица 1).

*Test client stopping an io service…* - сообщение, означающее, что выполнение тестируемого плагина завершено.



igromov@igromov-01pc:~$ ./test\_solverHandler 1105 192.168.1.100/1104

Test client SID: 1105

Application server SID: 192.168.1.100/1104

Test client starting an io service…

Plugin was executed without errors.

Everything works well!

Test client stopping an io service…

где

*Test client SID: 1105* – сообщение, означающее инициализацию функции тестирования и присвоение ей ID процесса равного 1105.

*Application server SID: 192.168.1.100/1104* – сообщение, означающее, что тестовая функция определила сервер приложений и подключилась к нему.

*Test client starting an io service…* - сообщение, означающее, что тестируемый плагин начал своё выполение.

*Plugin was executed without errors.*

*Everything works well!* – сообщение, означающее, что тестируемый плагин в процессе своего выполнения не вернул никакой ошибки, а, следовательно – выполнился корректно.

*Test client stopping an io service…* - сообщение, означающее, что выполнение тестируемого плагина завершено.

В последнем случае сервер приложений вернул результат выполнения тестового плагина, представленный ниже:

predicate\_1

function\_1

function\_2

function\_3

где:

predicate\_1 – результат выполнения соответствующей функции-предиката. Проверяет возможность перехода графовой модели из состояния INPUT в состояние HOM\_POST.

function\_1 – результат выполнения соответствующей функции перехода. Соответствует переходу графовой модели из состояния INPUT в состояние HOM\_POST.

function\_2 – результат выполнения соответствующей функции перехода. Соответствует переходу графовой модели из состояния HOM\_POST в состояние SOLVED\_2.

function\_3 – результат выполнения соответствующей функции. Соответствует переходу графовой модели в состояние FINALIZED и завершению её обработки.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения выпускной квалификационной работы был проведён тщательный анализ подходов к реализации технологии удалённого запуска процедур в различных системах, обзор современных и перспективных технологий.

На основе полученных данных была сформирована архитектура собственной реализации, применяемой для удалённого запуска графоориентированных решателей в РВС GCD. В соответствии с предложенной архитектурой была осуществлена программная реализация технологии, проведено успешное тестирование и отладка полученной программы. Тестирование показало, что предложенное решение является гибким и поддаётся лёгкой модификации, что позволит в будущем усовершенствовать его или адаптировать под новый функционал РВС GCD.

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. Что позволит существенно ускорить и упростить проведение вычислительных экспериментов и научных исследований.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

* + - 1. I.V. Morozov I. A. Valuev A. S. Automatic distributed workflow generation with GridMD library // [Computer Physics Communications](https://www.scopus.com/sourceid/13184?origin=resultslist) - 182(9), - p. 2052-2058 - 2011.
      2. Tyng-Yeu Liang, Yu-Wei Chang, Hung-Fu Li. A CUDA programming toolkit on grids. //[International Journal of Grid and Utility Computing](https://www.scopus.com/sourceid/14100154713?origin=resultslist) - 3(2-3) - p. 97-111 - 2012.
      3. Ahmed. L., Abhari A. A. multi-agent-based simulator for a transmission control protocol/internet protocol network. // [SIMULATION](https://www.scopus.com/sourceid/14452?origin=resultslist) 90(5) - p. 511-521 - 2014
      4. Kiraly S., Szekely S. Analysing RPC and testing the performance of solutions. // [Informatica (Slovenia)](https://www.scopus.com/sourceid/25507?origin=resultslist) 42(4) - p. 555-561 - 2018.
      5. Влацкая И. В., Сормов С. И. Управление и обработка информации в распределённых системах // Вестник науки Сибири – 2010. - №4. – С. 1-100
      6. Вичугова А. А., Вичугов В. Н., Дмитриева Е. А., Цапко Г. П. Методы и средства интеграции информационных систем в рамках единого информационного пространства проектирования // Вестник науки Сибири №5. – 2012.
      7. Kenton. Is Cap’n Proto Secure? Cap`n Proto[электронный ресурс]. – Режим доступа: https://capnproto.org/faq.html#is-capn-proto-secure
      8. Grosu, P., Abdul Rehman, M., Anderson, E., Pai, V., & Miller, H. gRPC. Programming Models for Distributed Computation. Github. - Dist Prog Book[электронный ресурс] - Режим доступа: http://dist-prog-book.com/chapter/1/gRPC.html
      9. Thrift. Finagle-Quickstart. - Twitter[электронный ресурс]. - Режим доступа: https://twitter.github.io/finagle/guide/Quickstart.html
      10. Соколов А.П., Першин А.Ю. Формат данных Advanced INI (aINI) // Каркас системы. 2007-2017. SA2. с.18.
      11. Соколов А.П., Першин А.Ю., Щетинин В.Н., Сапелкин А.С. Реверсивная многомасштабная гомогенизация физико-механических характеристик гетерогенных периодических сред с использованием графоориентированного программного подхода // Композиты и наноструктуры. 2017. № 3-4. с. 25-38.
      12. Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов // Программирование. 2019. Т. X, No X. С. XX–XX (готовится).
      13. The DOT Language – Graphviz[Электронный ресурс]. - Режим доступа: <https://www.graphviz.org/doc/info/lang.html>
      14. Соколов А.П., Першин А.Ю. Программный инструментарий для создания подсистем ввода данных при разработке систем инженерного анализа // Программная инженерия. 2017. Т. 8, №12. С. 543–555.
      15. Способ и система графо-ориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов – FindPatent.ru[Электронный ресурс]. - Режим доступа: https://findpatent.ru/patent/268/2681408.html

**ПРИЛОЖЕНИЕ А**

**Схема архитектуры подсистемы удалённого запуска решателей РВС GCD**



**ПРИЛОЖЕНИЕ Б**

**Листинг 1. Исходный код плагина, реализующего удалённый запуск решателей**

1. #include "comaps\_cls\_GraphSolverWebHandler.h"
2. #include "comfrm\_cls\_Kernel.h"
3. #include "comfrm\_unt\_SQLTools.h"
4. #include "comfrm\_unt\_Logger.h"
5. #include "comfrm\_cls\_Node.h"
6. #include "comfrm\_ifc\_Kernel.h"
7. #include "comfrm\_cls\_DatabaseRemoteAccess.h"
8. #include "comfrm\_cls\_PostgresqlBackend.h"
9. #include "comfrm\_cls\_Message.h"
10. #include "comfrm\_unt\_StringTools.h"
11. #include "comfrm\_cls\_AnyMap.h"
12. #include "comfrm\_unt\_INIOarser.h"
13. #include <boost/bind.hpp>
14. #include <postgresql/libpq-fe.h>
15. #include <fstream>
16. #include <sstream>
17. using namespace std;
18. using namespace ini;
19. using namespace com;
20. void cpl\_GraphSolverWebHandler::execute(cls\_AnyMap& p\_input, ifc\_ActionItem::tdf\_onMessageClb p\_MessCallback)
21. {
22. m\_callback = p\_MessCallback;
23. cls\_AnyMap\* answer = new cls\_AnyMap();
24. int err = 0;
25. if(!p\_input){
26. err = ecEmptyData;
28. COUT\_LOG << "Solver execution didn`t done. Error code: " << err << endl;
29. (\*answer)[ "ERROR\_CODE" ] = static\_cast< int >(err);
30. m\_callback(boost::shared\_ptr< cls\_AnyMap >(answer));
31. return;
32. }
33. else{
34. std::stringstream sqlRequest;
35. sqlRequest << "SELECT TOP 1 adotn FROM GCDDB.com.slvrs WHERE slver = \"" << p\_input[ "SOLVER\_SID" ] << "\"";
36. string finalRequest = sqlRequest.str();
38. dbConnect = PQconnectdb("host=89.108.120.130 port=5432 dbname=GCDDB user=gcdpublic password=gcdpublic!123");
39. PGresult \*result = PQexec(dbConnect, finalRequest);
40. const string adotFile = PQgetvalue(result, 0, 0);
41. if(!adotFile){
42. err = ecFileNotFound;
43. COUT\_LOG << "Solver execution didn`t done. Error code: " << err << endl;
44. (\*answer)[ "ERROR\_CODE" ] = static\_cast< int >(err);
45. m\_callback(boost::shared\_ptr< cls\_AnyMap >(answer));
46. return;
47. }
48. else{
49. ifstream file1(adotFile), file2(p\_input["IN\_FNAME"]);
50. string test\_read;
51. if(!(getline(file1, test\_read)){
52. err = ecFileRead;
54. COUT\_LOG << "Solver execution didn`t done. Error code: " << err << endl;
55. (\*answer)[ "ERROR\_CODE" ] = static\_cast< int >(err);
56. m\_callback(boost::shared\_ptr< cls\_AnyMap >(answer));
57. return;
58. }
59. else if(!(getline(file2, test\_read)){
60. err = ecFileRead;
61. COUT\_LOG << "Solver execution didn`t done. Error code: " << err << endl;
62. (\*answer)[ "ERROR\_CODE" ] = static\_cast< int >(err);
63. m\_callback(boost::shared\_ptr< cls\_AnyMap >(answer));
64. return;
66. }
67. std::shared\_ptr<com::graph::Node> adotNode = com::graph::loadFromADot(adotFile);
68. if(!(adotNode->run(p\_input["IN\_FNAME"])){
69. err = ecExecError;
71. COUT\_LOG << "Solver execution didn`t done. Error code: " << err << endl;
72. (\*answer)[ "ERROR\_CODE" ] = static\_cast< int >(err);
73. m\_callback(boost::shared\_ptr< cls\_AnyMap >(answer));
74. return;
75. }
76. }
77. }
78. }

**ПРИЛОЖЕНИЕ В**

**Листинг 2. Исходный код модуля, реализующего регистрацию плагина в РВС GCD**

1. #ifndef comfrm\_cls\_GraphSolverWebHandlerH
2. #define comfrm\_cls\_ GraphSolverWebHandlerH
3. #include "comfrm\_ifc\_ExtendedPlugin.h"
4. #include "comfrm\_ifc\_ActionItem.h"
5. #include "comfrm\_cls\_SQLTable.h"
6. #include <boost/bind.hpp>
7. class cpl\_GraphSolverWebHandler : public ifc\_ExtendedPlugin
8. {
9. public:
10. enum enu\_ErrorCode
11. {
12. ecSuccess = 0,
13. ecEmptyData = 100,
14. ecFileNotFound,
15. ecFileRead,
16. ecExecError = 200,
17. };
18. virtual ~cpl\_GraphSolverWebHandler(){}
19. /\*!
20. Возвращает уникальный строковый идентификатор плагина
21. \return Строковый идентификатор
22. \*/
23. virtual std::string getPluginSID() const {return "GRAPH\_SOLVER";}
24. /\*!
25. Возвращает имя плагина
26. \return Имя плагина
27. \*/
28. virtual std::string getPluginName() const {return "GRAPH\_SOLVER";}
29. /\*!
30. Возвращает тип плагина
31. \return Тип плагина
32. \*/
33. virtual std::string getPluginType() const {return "COMAPS";}
34. /\*!
35. Деинициализирует плагин. Выполняется при удалении плагина из системы
36. \*/
37. virtual void destroyPlugin(){return;}
38. /\*!
39. Запускает плагин, передавая на вход AnyMap с входными данными
40. \param[in] p\_input Входные данные
41. \*/
42. virtual void execute(cls\_AnyMap& p\_input, ifc\_ActionItem::tdf\_onMessageClb p\_MessCallback);
43. private:
44. ifc\_ActionItem::tdf\_onMessageClb m\_callback;
45. };
46. #endif

**ПРИЛОЖЕНИЕ Г**

**Листинг 3. Исходный код тестового решателя TEST\_R**

1. #include "test\_R.h"
2. #include "stringtools.h"
3. #include <anymap.h>
4. #include <iniparser.h>
5. #include <mutex>
6. #include <thread>
7. using namespace std;
8. using namespace com;
9. using namespace ini;
10. int test\_R::loadDataFromTSK(com::Anymap &data) {
11. printf( "loading data...\n" );
12. return 0;
13. }
14. bool test\_R::predicate\_1(com::Anymap &data) {
15. printf( "predicate\_1\n" );
16. return false;
17. }
18. bool test\_R::predicate\_2(Anymap &data) {
19. printf( "predicate\_2\n" );
20. return false;
21. }
22. bool test\_R::predicate\_3(Anymap &data) {
23. printf( "predicate\_3\n" );
24. return true;
25. }
26. int test\_R::function\_2(Anymap &data) {
27. printf( "function\_2\n" );
28. return 2;
29. }
30. int test\_R::function\_1(Anymap &data) {
31. printf( "function\_1\n" );
32. return 1;
33. }
34. int test\_R::function\_3(Anymap &data) {
35. printf( "function\_3\n" );
36. return 3;
37. }
38. return 3;
39. }

**ПРИЛОЖЕНИЕ Д**

**Листинг 4. Исходный код программы тестирования реализованного плагина**

1. #include "comaps\_cls\_GraphSolverWebHandler.h"
2. #include "comfrm\_cls\_Kernel.h"
3. #include "comfrm\_cls\_Node.h"
4. #include "comfrm\_ifc\_Kernel.h"
5. #include "comfrm\_cls\_DatabaseRemoteAccess.h"
6. #include "comfrm\_cls\_PostgresqlBackend.h"
7. #include "comfrm\_cls\_Message.h"
8. #include "comfrm\_unt\_StringTools.h"
9. #include "comfrm\_cls\_AnyMap.h"
10. #include "comfrm\_unt\_INIOarser.h"
11. #include <boost/detail/lightweight\_test.hpp>
12. #include <boost/shared\_ptr.hpp>
13. #include <boost/thread.hpp>
14. #include <string>
15. #include <iostream>
16. using namespace std;
17. using namespace com;
18. namespace com
19. {
20. namespace sys
21. {
22. ifc\_Kernel\* glb\_kernel;
23. boost::asio::io\_service\* glb\_ioservice;
24. stc\_NodeAddress glb\_appServerNodeAddr;
25. }
26. }
27. void onGraphSloverExecuted(boost::shared\_ptr< cls\_Message > p\_msg, cls\_Node::enu\_CommunicationError p\_error)
28. {
29. cls\_AnyMap reply = p\_msg->getAnyMap();
30. if(replly[ "ERROR\_CODE" ].toInt() == 0)
31. {
32. cout << \_\_PRETTY\_FUNCTION\_\_ << "Graph Solver Executed" << endl;
33. cout << \_\_PRETTY\_FUNCTION\_\_ << "Everything works well!" << endl;
34. }
35. else
36. {
37. cout << \_\_PRETTY\_FUNCTION\_\_ << "Something went wrong while executing Graph Solver Plugin. Error code is: " << reply[ "ERROR\_CODE" ].toInt() << endl;
38. }
39. com::sys::glb\_ioservice->stop();
40. }
41. int main(int argc, char\* argv[])
42. {
43. if(argc != 3)
44. {
45. cout << "Usage: ./graphSolverTest CLIENT\_SID 192.1668.1.1000/APP\_SERVER\_SID" << endl;
46. return -1;
47. }
48. com::sys::glb\_ioservice = new boost::asio::io\_service();
49. string appSid(argv[1]);
50. cout << "Test client SID: " << appSid << endl;
51. string appServerSid(argv[2]);
52. cout << "Test client SID: " << appServerSid << endl;
53. cls\_Node\* Node = new cls\_Node(appSid, \*com::sys::glb\_ioservice);
54. com::sys::glb\_appSeverNodeAddr = stc\_NodeAddress(appServerSid);
55. cls\_AnyMap launchAI;
56. launchAI[ "CMD" ] = 4;
57. launchAI[ "ACTION\_ITEM\_SID" ] = "GRAPH\_SOLVER";
58. cls\_Message msg(com::sys::glb\_appServerNodeAddr, launchAI);
59. node->send(msg, boost::bind(&onGraphSloverExecuted, \_1, \_2));
60. cout << \_\_PRETTY\_FUNCTION\_\_ << "Starting an io srevice... " << endl;
61. com::sys::glb\_ioservice->run();
62. cout << \_\_PRETTY\_FUNCTION\_\_ << "Stopping an io service... " << endl;
63. return 0;
64. }