

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ АНАЛИЗА И
ВИЗУАЛИЗАЦИИ ТЕЛЕМЕТРИЧЕСКИХ
ДАННЫХ

Студент


(подпись, дата)

05.01.2026 Кочетков Артём Михайлович
(ФИО полностью)

Руководитель


(подпись, дата)

05.06.2026 Федюнин Р.Н.
(фамилия, инициалы)

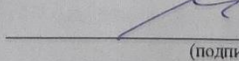
Нормоконтролёр


(подпись, дата)

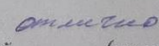
05.06.2026 А.С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

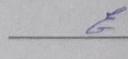
Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от 11.06.2026 № 28)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кочетков Артём Михайлович гр. 22ВВП2 факультета вычислительной техники.
2. Тема работы «Приложение анализа и визуализации телеметрических данных»
Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026 г.
3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Федюнин Р.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение для анализа и визуализации телеметрических данных, обеспечивающее импорт структурированных данных из локальных файлов (форматы CSV и LOG), их обработку, фильтрацию и построение настраиваемых графических представлений (линейных и логарифмических) для последующего анализа пользователем.

Исходные данные:

Форматы входных данных: CSV, LOG. Целевая операционная система: Windows 10 и выше. Требования к интерфейсу: графический пользовательский интерфейс с поддержкой интуитивного взаимодействия. Требования к визуализации: поддержка нескольких подграфиков, настройка сетки, легенды, цветовой кодировки по типам единиц измерения. Требования к импорту: автоматическое распознавание единиц измерения, временных меток; возможность ручной корректировки параметров импорта. Требования к сохранению: экспорт конфигураций графиков в JSON-шаблоны, сохранение настроек пользователя между сеансами.

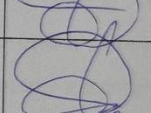
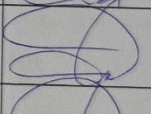
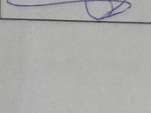
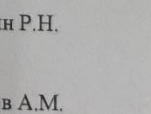
Язык и среда разработки:

Использовать современный язык программирования, поддерживающий ООП и соответствующую среду разработки.

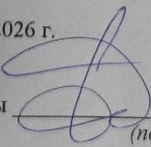
5. Перечень подлежащих разработке вопросов

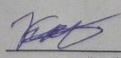
Анализ предметной области. Проектирование архитектуры программного обеспечения. Разработка модуля импорта и предобработки данных. Разработка модуля визуализации и настройки графиков. Разработка пользовательского интерфейса. Реализация механизмов обработки больших объемов данных. Тестирование и отладка. Разработка системы настроек и сохранения состояния. Обеспечение отказоустойчивости. Оформление технической документации.

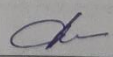
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	07.04.2026-11.04.2026	
2.	Анализ требований	10	12.04.2026-13.04.2026	
3.	Выбор технологий, применяемых при проектировании и разработке	14	14.04.2026-16.04.2026	
4.	Разработка макета графического интерфейса	20	17.04.2026-19.04.2026	
5.	Разработка приложения анализа и визуализации телеметрических данных	150	20.04.2026-06.05.2026	
6.	Тестирование и отладка	30	07.05.2026-10.05.2026	
7.	Оформление пояснительной записки	50	11.05.2026-21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент  Кочетков А.М.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
09.06.2026 (подпись, дата)

Реферат

Бакалаврская работа содержит 103 листа, 23 рисунка, 6 источников и 3 приложения.

АНАЛИЗ ДАННЫХ, ВИЗУАЛИЗАЦИЯ ДАННЫХ, ВРЕМЕННЫЕ РЯДЫ, ГРАФИКИ, ДЕСКТОПНОЕ ПРИЛОЖЕНИЕ, ИМПОРТ ДАННЫХ, ОБРАБОТКА ДАННЫХ, ТЕЛЕМЕТРИЯ, MATPLOTLIB, PANDAS, PYTHON

Объект разработки: программное обеспечение для анализа и визуализации телеметрических данных, предназначенное для работы со структурированными временными рядами, хранящимися в локальных файлах форматов CSV и LOG.

Цель работы: разработка десктопного приложения, обеспечивающего импорт, обработку и наглядное графическое представление телеметрических данных с возможностью гибкой настройки отображения для последующего анализа пользователем.

Методы исследования: анализ предметной области и существующих аналогов (Grafana, OpenObserve, RemusLab); выбор архитектурного паттерна MVP; разработка на языке Python с применением библиотек PyQt5 (графический интерфейс), Matplotlib (визуализация), Pandas и NumPy (обработка данных и численные расчёты).

Полученные результаты:

– спроектирована и реализована модульная архитектура приложения, включающая модуль импорта данных с автоматическим распознаванием структуры файлов, редактор метаинформации, систему управления единицами и пользовательскими типами;

					ПГУ1.090301.38.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Приложение анализа и визуализации телеметрических данных	Лит.	Лист	Листов
Разраб.		Кочетков А.М.		24.06.20				
Провер.		Федюнин Р.Н.		25.06.20				
Н. Контр		Бычков А.С.		25.06.20			2	48
Утверд.		Митрахин М.А.		24.06.20	Пояснительная записка	ФВТ, каф. ВТ гр. 22ВВП2		

– разработан многооконный графический интерфейс с поддержкой нескольких независимых фигур, настраиваемых подграфиков, множественных правых осей;

– реализованы инструменты визуализации: линейные и точечные графики, логарифмический масштаб, настраиваемая сетка, цветовая кодировка по типам единиц;

– реализовано сохранение и загрузка пользовательских конфигураций (шаблоны в формате JSON), поддержка светлой и тёмной темы оформления;

– проведено функциональное тестирование, подтвердившее корректность импорта файлов различной структуры, отказоустойчивость при обработке некорректных данных и стабильность работы приложения.

Область применения: мониторинг и анализ телеметрических данных в промышленности, энергетике, транспортных системах, а также в научных исследованиях, связанных с обработкой временных рядов.

Практическая значимость: разработанное приложение предоставляет универсальный, простой в использовании и легко интегрируемый инструмент для визуального анализа структурированных телеметрических данных, не требующий сложной настройки или использования дополнительного программного стека, что позволяет сократить временные затраты на подготовку данных и построение графиков по сравнению с универсальными офисными пакетами или математическими средами.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		3

Содержание

Перечень сокращений и обозначений.....	4
Введение	6
1 Анализ предметной области	7
1.1 Сведения о телеметрии.....	7
1.2 Обзор приложений аналогов	8
1.3 Анализ актуальности разрабатываемого приложения	11
1.4 Постановка задачи	13
1.5 Выбор и настройка инструментов разработки.....	14
2 Разработка приложения.....	18
2.1 Выбор архитектуры	18
2.2 Описание модулей приложения	19
2.3 Описание пользовательского интерфейса приложения.....	27
2.4 Тестирование приложения	35
3 Охрана труда	43
Заключение	45
Список использованных источников	48
Приложение А – UML-диаграммы.....	49
Приложение Б – Листинг программы.....	55
Приложение В – Презентация	96

Перечень сокращений и обозначений

НОД – наибольший общий делитель

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

ЭВМ – электронно-вычислительная машина

GUI (Graphical User Interface) – графический пользовательский интерфейс

IDE (Integrated Development Environment) – интегрированная среда разработки

MVP (Model-View-Presenter) – модель-представление-представитель

SQL (Structured Query Language) – язык структурированных запросов

UML (Universal Modeling Language) – унифицированный язык моделирования

WPF (Windows Presentation Foundation) – графическая презентационная основа

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Введение

В настоящее время телеметрия применяется во множестве областей человеческой деятельности, таких как сельское хозяйство (мониторинг почвы, погоды, техники), медицина (дистанционное наблюдение за пациентами, носимые устройства), вендинг и торговые автоматы (контроль остатков и неисправностей), оборонная и космическая отрасли (управление беспилотными аппаратами, телеметрия ракет и спутников), а также в автомобилестроении, энергетике, логистике и промышленном производстве. Повсеместная цифровизация привела к тому, что объемы телеметрических данных, генерируемых даже одним предприятием, могут исчисляться гигабайтами в сутки.

Однако сбор и хранение телеметрической информации – лишь половина задачи. Основная ценность заключается в возможности оперативного анализа этих данных и принятия обоснованных решений и именно поэтому ПО, осуществляющее интерпретацию и представление данных для конечного пользователя в удобном и читаемом формате имеет особую важность. Грамотно построенные графики, диаграммы и временные шкалы позволяют инженеру или аналитику быстро выявить аномалию или тренд. Визуализация выступает тем самым «окном» в мир телеметрических данных, превращая абстрактные числовые массивы в наглядные образы, доступные для человеческого восприятия.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

1 Анализ предметной области

1.1 Сведения о телеметрии

Сбор, передача и анализ телеметрии – это задачи, которые высоко востребованы во многих отраслях. Предъявляемые требования о высокой надежности, эффективности, высокой степени автоматизации и масштабируемости напрямую отражаются в предлагаемых продуктах для решения данных задач. Качество и гибкость настройки используемых систем во многом определяют стабильность работы оборудования и удобство контроля за технологическими процессами.

Внедрение телеметрии можно разделить на несколько этапов. Первый – это сбор информации с непосредственных датчиков, установленных или внедренных в готовое оборудование. Сами датчики должны обладать возможностью преобразовывать информацию в пригодную для дальнейшей транспортировки, например, производя конвертацию аналогового сигнала в цифровой.

Второй этап – передача данных по каналам связи. Передача зачастую должна осуществляться непрерывно, каналы связи, как беспроводные, так и физические линии связи должны быть заранее проложены, также используемые каналы должны подходить для передачи конкретного типа информации с минимальными потерями и задержками. Перед отправкой или получением, данные также проходят через приемо-передающие устройства. Все данные в конечном итоге поступают на агрегатор, в роли которого обычно выступает единый сервер или распределенная серверная система, на которых они могут храниться продолжительное время.

Третий этап – анализ данных, эту задачу выполняют алгоритмы, внедренные в ПО, которые также безостановочно фильтрует поток информации, приходящий с датчиков. Приложения, отвечающие за анализ и обработку, выступают интерфейсом для человека, благодаря которому, мы

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

можем здраво оценивать, анализировать и принимать решения, связанные с управлением оборудованием, к которому и подключается все эта система.

Все три этапа необходимы для создания системы передачи и обработки данных телеметрия, но если сенсоры сбора информации и линии передачи данных зачастую имеют устоявшиеся стандарты и проверенные временем решения, то выбор программного обеспечения для обработки конечных данных остается открытым вопросом, так как практически каждое предприятие обладает уникальной рабочей средой и собственными требованиями. Данные, за которым необходим постоянным контроль могут разительно отличаться, так же, как и допуски их отклонения. Желаемые возможности мониторинга и масштабируемости также будут зависеть от конечного заказчика и могут варьировать как от крупномасштабных сетей, так и до минимальных локальных систем.

Все это говорит о необходимости разработки гибких и адаптируемых программных решений для обработки телеметрических данных, способных учитывать особенности конкретной предметной области, обеспечивать удобный мониторинг состояния оборудования и поддерживать дальнейшее расширение функциональных возможностей системы.

1.2 Обзор приложений аналогов

Рассмотрим уже существующие приложения, которые способны работать с статичными и динамическими данными и создавать их графические представления, удобные для последующего анализа пользователем.

Grafana – популярное веб-приложение с открытым исходным кодом, может работать с метриками, логами, трассировками и произвольными данными с разных источников. Имеется встроенная поддержка плагинов, гибкая кастомизация финального представления данных в виде дашбордов.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

Будучи на рынке с 2014 года Grafana, имеет большое количество обучающих материалов, готовых пресетов для метрик в сети, что облегчает развертывание и использование веб-приложения.

Из существенных недостатков стоит выделить ориентированность приложения на работу с данными в реальном времени, что видно по стилю главной приборной панели, показанной на рисунке 1. В Grafana имеется возможность импортировать файлы и также просматривать их, но это потребует настройки новых виджетов на приборной панели, для повторных манипуляций по типу изменения графика, смены единиц измерения также потребуется изменить виджет в настройках. Grafana зарекомендовала себя для телеметрии в web технологиях, анализе трафика и мониторингом серверов и ПК, но требует для этого тщательной настройки и не позволяет сразу интерпретировать данные, которые не подходят по формату к уже настроенной панели приборов.



Рисунок 1 – Пример настроенного дашборда Grafana

OpenObserve – это полнофункциональная платформа наблюдения с открытым исходным кодом, предлагающая унифицированный обзор журналов, метрик, трассировок и фронтенд-телеметрии. Позиционируется как более легковесное решение в сравнении с Grafana и ее стек приложений. В

отличие от Grafana может быть развернута быстрее, т.к. содержит в себе все нужные инструменты сразу. Разработчик также заявляет высокую производительность и возможность работы с данными уровня петабайт в сутки.

Приложение бесплатно для использования в некоммерческом использовании, но имеет и платные варианты, такие как корпоративная и облачная версия.

OpenObserve хорошо подходит для работы с действительно крупными пакетами данных, текстовыми логами и простыми дата дампами, внутренними средствами визуализации являются достаточными для таких видов информации, а многие полезные внедрения по типу запросов на PromQL и обычном SQL, функции пользовательских метрик, поддержки данных на основе временных серий делают это вполне подходящим приложением. Пример окна графического интерфейса OpenObserve представлен на рисунке 2.

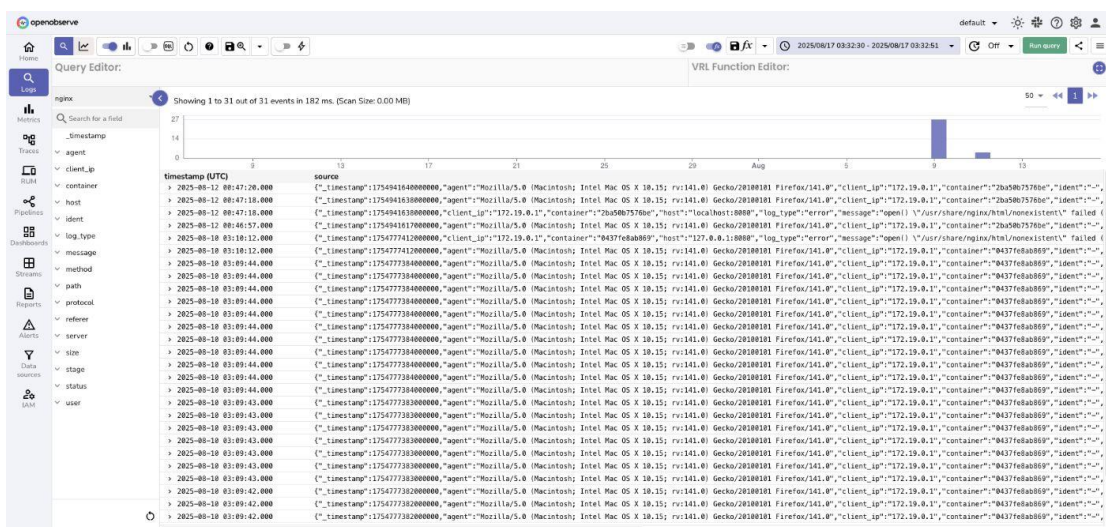


Рисунок 2 – Окно просмотра логов в OpenObserve

RemusLab – программное обеспечение, поставляющееся по лицензии с закрытым исходным кодом, сочетает в себе встроенную поддержку работы с аппаратным обеспечением, обработку и анализ телеметрии с подключенных к ПК интерфейсов.

По сравнению с предыдущими аналогами приложение является более узкоспециализированным, так как предполагает возможность использования ПО на машине, которая будет непосредственно подключена к каналу связи телеметрии по унифицированному интерфейсу. Программа разработана исключительно для Windows, в отличие от двух других аналогов, которые можно использовать с другими операционными системами, либо по облаку. Интерфейс программы показан на рисунке 3.

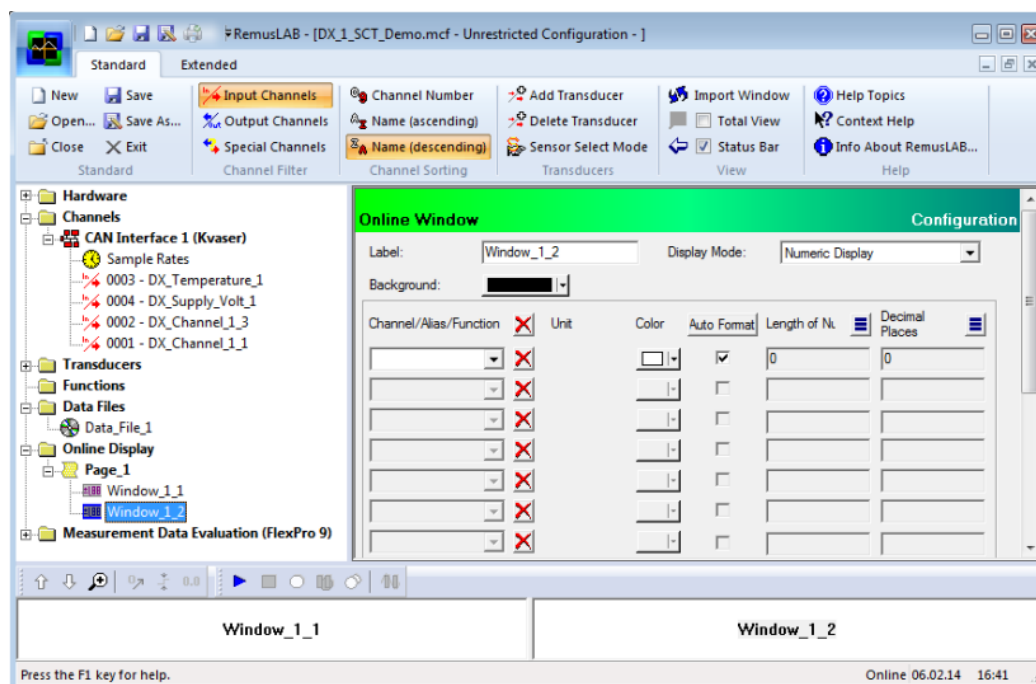


Рисунок 3 – Интерфейс программы на моменте создания виртуального монитора

1.3 Анализ актуальности разрабатываемого приложения

Телеметрия находит применение в крупном спектре областей: от информационных технологий и облачных систем, где необходим постоянный контроль за состоянием серверных мощностей, сбор данных пользователя, мониторинг ошибок в логах, до конкретных промышленных предприятий и бизнес-проектов, транспортных систем, энергетики. Рост объемов собираемой информации приводит к постоянному увеличению количества

телеметрических данных, требующих анализа. Существующие решения по представлению телеметрических данных удовлетворяют потребностям пользователей, но и имеют ряд недостатков:

- обработка данных преимущественно в режиме реального времени при недостаточной поддержке работы с локальными файлами формата .csv, .log, .zip и другими распространенными форматами хранения данных;

- перегруженный пользовательский интерфейс и высокая сложность развертывания системы, необходимость использования целого стека приложений для функционирования;

- высокая ресурсоемкость некоторых решений, представляющая повышенные требования для аппаратной части ЭВМ;

- некоторые приложения разработаны конкретно под продукты компании и пригодны для использования только с определенным оборудованием, что затрудняет их использование в универсальных телеметрических системах.

Отказ от использования специализированных программ для анализа и представления телеметрии влечет за собой усложнение процесса интерпретации конечных данных вплоть до упущения важных деталей в потоке данных. Применение ПО, не предназначенного для данных задач, но имеющих возможность визуального представления данных (Excel, MatLab) позволяет выполнять базовый анализ и строить графики, однако требует значительных временных затрат на подготовку данных, настройку визуализации и создание инструментов анализа для каждого отдельного случая. Кроме того, такой подход не обеспечивает необходимого уровня автоматизации и удобства при регулярной работе с телеметрической информацией.

Актуальность разработки специализированного приложения обусловлена необходимостью создания универсального, простого в использовании и легко интегрируемого инструмента анализа телеметрических

данных. Такое приложение должно обеспечивать удобную загрузку данных из локальных файлов, их обработку, фильтрацию и визуальное представление в форме графиков, диаграмм и статистических показателей.

Данное приложение будет являться частью системы телеметрии. Собранные и обработанные данные, переданные по каналам связи записываются и хранятся на устройстве, эти данные уже можно анализировать с помощью приложения. В данном случае методы сбора и обработки данных обособлены от анализа внутри приложения, что позволяет интегрировать его в уже готовые системы телеметрии.

1.4 Постановка задачи

Необходимо разработать десктопное приложение, способное преобразовывать структурированную информацию, хранящуюся локально в файлах, в графическое и табличное представление, для возможности более удобного анализа пользователем.

К разрабатываемому приложению применяются следующие требования:

- приложение должно уметь работать с .csv и .log файлами большого размера разной структуры;
- приложение должно иметь возможность представления данных в виде линейного и логарифмического графика зависимости величин;
- пользователь должен иметь возможность задавать имена столбцов данных, указывать единицы измерения, заданные таким образом настройки можно сохранить в файл конфигурации и загрузить снова;
- приложение должно иметь возможность обнаружения и указания абнормальных величин, битых файлов данных, неправильной структуры данных в файле;
- при представлении данных в графическом формате у пользователя должна быть возможность гибкой настройки их отображения, а именно:

изменение цветовых обозначений, смена и задание пользовательских имен величинам и данным;

– при импорте данных в приложение, пользователь должен видеть предварительные характеристики данных, которых он выбрал. В приложении должна быть возможность изменения настройки импорта для выбора собственного диапазона данных;

– приложение должно работать на ОС Windows 10 и выше.

1.5 Выбор и настройка инструментов разработки

Исходя из поставленных требований, необходимо определиться с выбором языка программирования для реализации приложения по анализу и визуализации телеметрической информации. На основе этого выбора нужно определить среду разработки и дополнительные инструменты, такие как библиотеки, фреймворки, способ реализации пользовательского интерфейса.

Ключевыми аспектами при выборе языка программирования являются: производительность, заключающаяся в времени выполнения и потреблении ресурсов, наличие библиотек и пакетных менеджеров, предсказуемость поведения, а также порог входа и доступность квалифицированных специалистов на рынке, не менее важен и тип статической типизации.

Приложение должно отображать пользователю в интерфейсе обработанную информацию, поступающую из файлов с данными, рисовать интерактивный график зависимостей, выделять и группировать информацию из файлов данных, а также понятно отображать ее для пользователя, экспортировать настройки отображения и обработанные данные. Требования построения точных графиков на основе различных, не predetermined данных требует написания множества модулей и классов, заточенных лишь на это. Данную проблему можно решить, используя готовые решения в виде встроенных библиотек или фреймворков. Помимо это важную роль играет удобство работы с файлами, возможности их хранения в памяти и чтения,

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

производительность при увеличении их размера и количества однотипных данных.

Для данного проекта мог бы подойти такой язык как C++. C++ это компилируемый язык программирования общего назначения, широко используемый для написания программного обеспечения. В виду его гибкости и возможности создания высокопроизводительных приложений, а также возможности написания пользовательского интерфейса на Qt сильно ускорило бы процесс разработки, но накладные расходы в виде программирования доступа к памяти, решение проблем возможных ее утечек, а также необходимости написания обработки визуальной отрисовки графиков и функций вручную, делает данный выбор не идеальным.

Более высокоуровневой альтернативой может послужить C# – язык программирования высокого уровня, разработанный Microsoft в рамках платформы .NET, относится к семье языков с C-подобным синтаксисом исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов и частично освобождает от контроля за доступом к памяти, который теперь ложится на CLR. Главным минусом при написании данного приложения на C# будет являться реализация графического интерфейса на WPF, что в целом сложнее чем на Qt, также приложения будет привязано к ОС Windows из-за .NET.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения, с объемной библиотекой классов, встроенным сборщиком мусора, легким управлением многопоточностью и построенным вокруг принципа кроссплатформенности. Главной проблемой Java будет неоптимизированной хранение данных после их агрегации из локальных файлов. Также использование JavaFx или Swing для UI вносит дополнительные проблемы в виду их слабой возможности вносить изменения в стиль приложения и устарелого вида интерфейса, иначе придется

использовать альтернативы на Kotlin наподобие TornadoFx, которые никак не улучшают производительность отрисовки при большом количестве объектов, что важно в данном приложении.

Еще одним языком программирования, подходящим для реализации такого приложения, является Python. Это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика, читаемости кода и его качества, он является полностью объектно-ориентированным в том плане, что всё является объектами. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанными на нём программами по сравнению с аналогичным кодом, написанным на компилируемых языках. Несмотря на это, решения, предлагаемые библиотеками, сильно сокращают время разработки и позволяет написать более качественное приложение. Pandas – библиотека для работы с табличными данными, основанная на NumPy, расширяет его возможности для одновременной работы с разными типами размеченных данных. Позволяет выполнять операции сортировки, группировки, агрегации, слияния, заполнения пропусков и другие [1]. Matplotlib – одна из самых популярных библиотек для визуализации данных. Позволяет создавать статические, анимированные и интерактивные графики и диаграммы. Поддерживает различные типы визуализаций: линейные, столбчатые, круговые, точечные, гистограммы и другие. Имеют интеграцию с вышеупомянутыми библиотеками, возможность экспорта в мультимедиа и имеет объектно-ориентированный подход для работы с графиками за счет модуля Pyplot [2]. Все эти факторы делают Python хорошим выбором для разработки данного приложения.

Из этих соображений был выбран язык программирования Python. Достаточно простой для написания и понимания язык, обладающий

некоторыми библиотеками, которые, в сущности, упрощают разработку требуемого приложения.

Выше упоминалось о простоте использования Qt для написания интерфейсов на C++, в случае с Python имеется графический фреймворк Qt: PyQt – набора расширений для его реализации, практически полностью реализует возможности Qt [3]. PyQt также включает в себя Qt Designer (Qt Creator) – дизайнер графического интерфейса пользователя. Программа pyuis генерирует Python код из файлов, созданных в Qt Designer. Это делает PyQt очень полезным инструментом для быстрого прототипирования.

Основываясь на выборе языка программирования и фреймворка интерфейса логичным будет использование среды разработки PyCharm, которая нативно поддерживает PyQt и имеет встроенный дизайнер. Данная IDE специализирована для языка Python что сильно ускоряет написание кода за счет, интегрированных авто дополнений строчек кода для популярных библиотек, возможность полного рефакторинга классов и файлов проекта.

Альтернативой PyCharm может послужить Microsoft Visual Studio Code. Visual Studio Code – это редактор исходного кода. Он имеет многоязычный интерфейс пользователя и поддерживает ряд языков программирования, подсветку синтаксиса, IntelliSense, рефакторинг, отладку, навигацию по коду. Поскольку это редактор, а не полноценная IDE, то его использования несколько ограничено, но возможность установки расширений, в том числе для разработки на Python делают этот вариант рабочим.

Определившись с выбором средств разработки, требуется их установка и дальнейшая настройка. Python устанавливается в систему через репозиторий или инсталлер с официального сайта. Для разработки будет создано изолированное окружение с помощью venv, в него будут установлены все необходимые библиотеки и зависимости. Такой подход аннулирует конфликты уже установленных библиотек.

2 Разработка приложения

2.1 Выбор архитектуры

Опираясь на поставленную задачу по разработке приложения, для его написания необходимо будет создать гибкий пользовательский интерфейс для работы с данными, чтобы максимально облегчить и ускорить аналитику телеметрических данных. Поскольку приложение не завязано на одном конкретном стандарте данных и может принимать файлы различной типа с разной разметкой, необходимо будет создать гибкий метод по открытию таких файлов, скорее всего, вынеся его функционал в отдельное меню и окно, где будет происходить импорт данных, и их подготовка перед разбиением на серии и добавление на основной график.

В данном случае хорошим решением будем опираться на архитектуру MVP (Model-View-Presenter) при разработке приложения, четко разделяя код, завязанный на элементах пользовательского интерфейса и внутреннюю логику работы, построения графиков и вызове оберток функций библиотеки `matplotlib`.

MVP – шаблон проектирования пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения). Модель предоставляет данные для пользовательского интерфейса, представление реализует отображение данных и маршрутизацию пользовательских команд или событий презентеру. Презентер управляет моделью и представлением, например, извлекает данные из модели и форматирует их для отображения в представлении [4].

Таким образом в дальнейшем можно будет связать логику кода GUI, написанного на PyQt5 и внутренних модулей через встроенные сигналы, обеспечивающие слабую связность и позволяющие независимо разрабатывать модули.

При запуске пользователю будет отображаться основное окно программы, в котором он в последующем будет работать со всеми данными. При разработке дизайна в Qt было принято решение разместить на данном экране все элементы управления графиком, его сериями, а также окно добавления серии на график. Проблема загруженности интерфейса отпадает, т.к. вкладки, в которых будут размещены элементы управления решено было сделать скрывающимися по нажатию кнопки.

После написания методов импорта конфигураций и базовых файлов, они были вынесены в отдельные классы под управлением менеджера импорта данных, в который были добавлены функции предпросмотра изначальных данных, более гибкий импорт.

2.2 Описание модулей приложения

Точка входа в приложение находится в файле `__main__.py`, в нем происходит запуск главного окна приложения, которое описано в классе `MainWindow.py`. Дизайн основного окна выполнен на Qt с применением стилей на базе QSS – механизма настройки внешних виджетов Qt. Имеющиеся стили подгружаются вместе с основным окном и файлом конфигурации «`config.json`», в котором описаны словари единиц измерения, настройки цветов и прочее. Далее класс `MainWindow` также создает все остальные Qt виджеты: боковые панели, описанные в `SeriesDisplay.py` и `FigureSettings.py`, создает новую пустую фигуру на графике вместе с новой вкладкой – `EditableTabBar.py`, также в этом классе описаны методы обработки нажатий верхних кнопок меню, через которые пользователь будет взаимодействовать с другими секциями программы и методы для безопасного закрытия приложения.

Дабы продолжить работу с программой, необходимо загрузить телеметрические данные. Это делается путем вызова Менеджера Импорта – `DataManager` через верхнее контекстное меню или нажатием `Ctrl+D`, вызванное отдельное окно содержит в себе несколько вкладок и все

необходимое для правильно импорта информации. Менеджер отвечает за парсинг файлов, извлечение временных рядов, создание объектов классов Group и Series, а также за управление словарями единиц измерения (структуры классов user_units и base_units) и таблицей уточнений (unit_clarify).

Внутри менеджер импорта разделен на три вкладки. При импорте данных используется таблица, позволяющая сопоставить извлечённые из данных строки с выбранным типом и единицей измерений, которая заранее задана в словаре. Для удобного просмотра и редактирования самого словаря единиц в менеджере импорта также присутствует виджет UnitTree с моделью UnitModel. UnitTree отображает иерархию «тип единицы, список единиц», поддерживает добавление, удаление и переименование элементов через двойной щелчок или клавишу Delete, а UnitModel синхронизирует изменения с глобальными словарями.

После того как пользователь завершил импорт, менеджер импорта данных сохраняет созданные объекты Group (каждый содержит свой датафрейм и словарь Series для каждого столбца) в атрибут all_groups главного окна. Затем вызывается метод display_header_info, который для каждой серии из всех групп создаёт виджет, который отображается на второй вкладке менеджера – «Настройка серий». В этом виджете пользователь может поставить галочку "keep" – тогда серия будет включена в набор доступных для построения, а также задать алиас (псевдоним) для канала – при этом проверяется уникальность алиаса среди всех внутри группы и оригинальных заголовков. Выбор типа единицы и конкретной единицы измерения из выпадающих списков, которые динамически заполняются из заранее известных также происходит виджете. Все изменения сразу же записываются в соответствующий объект Series и помечают конфигурацию как изменённую (dm.modified = True).

После завершения работы с менеджером импорта данных, становится возможным построение графиков. В главном окне приложения слева

расположен виджет SeriesDisplay (из series_display.py), который содержит два дерева: «Доступные серии» и «Серии выбранного графика». Изначально все серии с флагом keep = True попадают в первое дерево. Пользователь выделяет нужные серии (можно целые группы или отдельные алиасы) и перетаскивает их на область графика. AxesFrame (центральный виджет с фигурой matplotlib) обрабатывает событие dropEvent, определяет текущий подграфик (который подсвечивается при наведении) и вызывает метод add_to_subplot из SeriesDisplay. Этот метод строит иерархический словарь ContentsDict и передаёт его в текущий SubplotManager через вызов метода sp.add(contents).

Описанные выше модули будут задействованы в такой последовательности каждый раз при новом запуске, дабы начать работу с приложением. На основе взаимодействия этих модулей можно построить соответствующую UML-диаграмму, изображённую на рисунке 4, которая будет наглядно демонстрировать работу приложения.

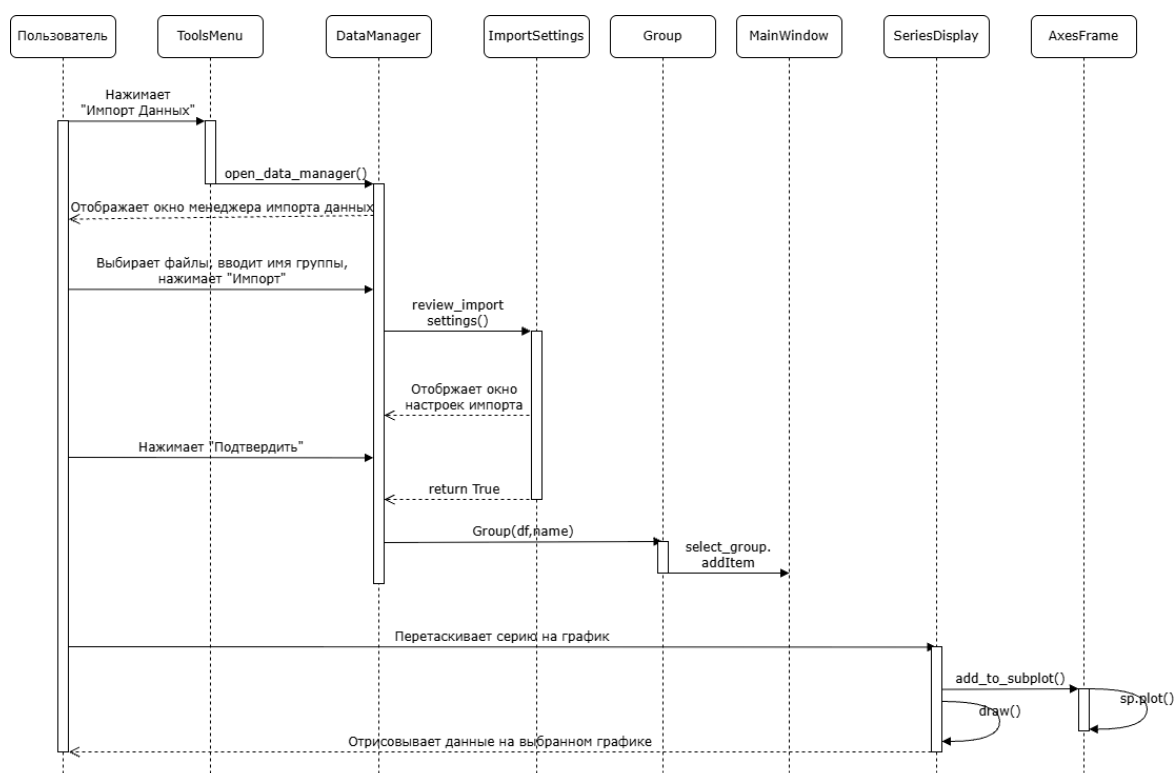


Рисунок 4 – UML-диаграмма последовательности

Класс SubplotManager (из subplot_manager.py) – это обертка над одним подграфиком и его правыми осями. При добавлении содержимого он для каждого алиаса определяет соответствующий объект Series, получает его значения unit_type и unit, после чего среди своих осей ищет ту, у которой id совпадает с этой парой. Если происходит совпадение, то он добавляет алиас в ax.contents, иначе – создаёт новую правую ось с помощью patch_ax(), которая наделяет стандартную ось matplotlib дополнительными атрибутами и методами (set_par, set_host, label и т.д.). Первая ось всегда существует и если она ещё пуста, может быть переиспользована для первого набора данных. После распределения содержимого вызывается sp.plot(), который: для каждой оси извлекает данные из соответствующей группы, обрезает по временному диапазону (значения cf.start – cf.end), применяет плотность графика согласно выставленному параметру в интерфейсе и масштабирование. После отрисовки всех осей вызывается sp.arrange_axes() для позиционирования правых осей относительно главной и sp.show_legend() для построения легенды.

За отрисовку сетки отвечает специальная ось ygrid_proху, которая получает позиции точек от основной оси и рисует горизонтальные линии через mock_gridlines. Это позволяет синхронизировать сетку с масштабом даже при наличии нескольких правых осей с разными единицами измерения.

Взаимодействие с графиком происходит через панель инструментов. Класс SubplotToolbar позволяет добавлять новые пустые подграфики (функция insert_subplot), удалять выбранные (функция delete_subplot), очищать их содержимое (переноса обратно в доступные данные), менять порядок подграфиков (кнопки вверх/вниз) и циклически переставлять правые оси (функция cycle_subplot). Класс LegendToolbar управляет легендой: включает и выключает её отображение на графике, выбирает положение, задаёт количество колонок, решает, показывать ли единицы измерения в подписях, а также включает цветовую координацию (color_coordinate). AxesToolbar позволяет выбрать конкретную активную ось из списка или включить отображение всех,

переключить логарифмический масштаб по оси Y, включить/отключить автоматическое масштабирование и вручную задать пределы оси (поля `yaxis_min/yaxis_max` становятся активными, когда автоподбор выключен).

Правая боковая панель «Настройки элемента» содержит множество параметров всей фигуры: отступы, интервал между подграфиками, смещение правых осей происходит с помощью подчи разных параметром на методы `update_gs_kwargs()` и `adjust_axes_offset()`, которые в свою очередь перестраивают сетку графика.

Весы подграфиков – строка, в которую необходимо вводить данные формата «[2, 1, 2]». Цифры обозначают размеры каждого графика в относительных величинах, т.е. средний график будет в 2 раза меньше двух других. Введенные значения проходят процедуры упрощения на НОД.

Тип графика – непрерывная линия или отдельные точки, размер точек, толщину, плотность, отступ по оси X.

Временной диапазон (значения `select_start, select_end`) – кнопки «Min» и «Max» устанавливают границы по минимуму и максимуму данных в текущей фигуре.

Размеры шрифтов устанавливается отдельно для разных элементов графика, поворот тиков и формат временной метки (настраиваемый через поле `strftime` в диалоге `edit_timestamp_format`).

Таблица цветов для типов единиц (`unit_table`) – каждый тип можно раскрасить в свой цвет, после чего все линии и подписи соответствующих осей перекрасятся.

Если пользователю нужно отредактировать свойства уже построенной серии, он может дважды щёлкнуть по ней в любом из деревьев `SeriesDisplay` – откроется новое окно, вызванное из класса `EditSeriesDialog` (из `edit_series_dialog.py`). В этом диалоге отображается статистика серии: минимум, максимум, их временные метки, количество точек. В диалоге можно изменить алиас, тип единицы измерения, тип данных и масштаб. При

сохранении диалог проверяет уникальность алиаса, затем, если серия уже была на графике, удаляет её из текущего подграфика, применяет изменения и добавляет заново – это необходимо, потому что смена единиц измерения может потребовать переноса на другую ось. После всех изменений фигура перерисовывается.

Для работы с несколькими фигурами в классе AxesFrame используется EditableTabBar – панель вкладок, в которой можно переименовывать вкладки двойным кликом. При смене вкладки вызывается функцией update_dock_widgets(), который обновляет содержимое SeriesDisplay и FigureSettings в соответствии с новой фигурой. Кнопка «+» создаёт новую пустую фигуру через метод FileMenu.new().

Все изменения (новые алиасы, единицы, цветовые схемы, настройки фигур) могут быть сохранены в JSON-шаблон посредством вызова метода ToolsMenu.export_template() (нажатие кнопки «Экспорт шаблона» в выпадающем меню «Опции»), а затем применены к другой фигуре через метод import_template(). Также при закрытии приложения главное окно вызывает метод overwrite_config(), который сохраняет текущие словари единиц, режим темы (светлая или тёмная) и другие глобальные настройки в файле config.json.

Завершение работы происходит через событие closeEvent класса UI: оно проверяет, все ли фигуры сохранены, при необходимости запрашивает подтверждение, затем сохраняет конфигурацию, закрывает все окна matplotlib и завершает приложение. Представление взаимодействия модулей продемонстрировано на рисунке 5.

Таким образом, приложение предоставляет полный цикл работы с телеметрическими данными: импорт, редактирование метаинформации каналов, гибкое построение много панельных графиков с поддержкой множества правых осей, настройку внешнего вида и сохранение результатов.

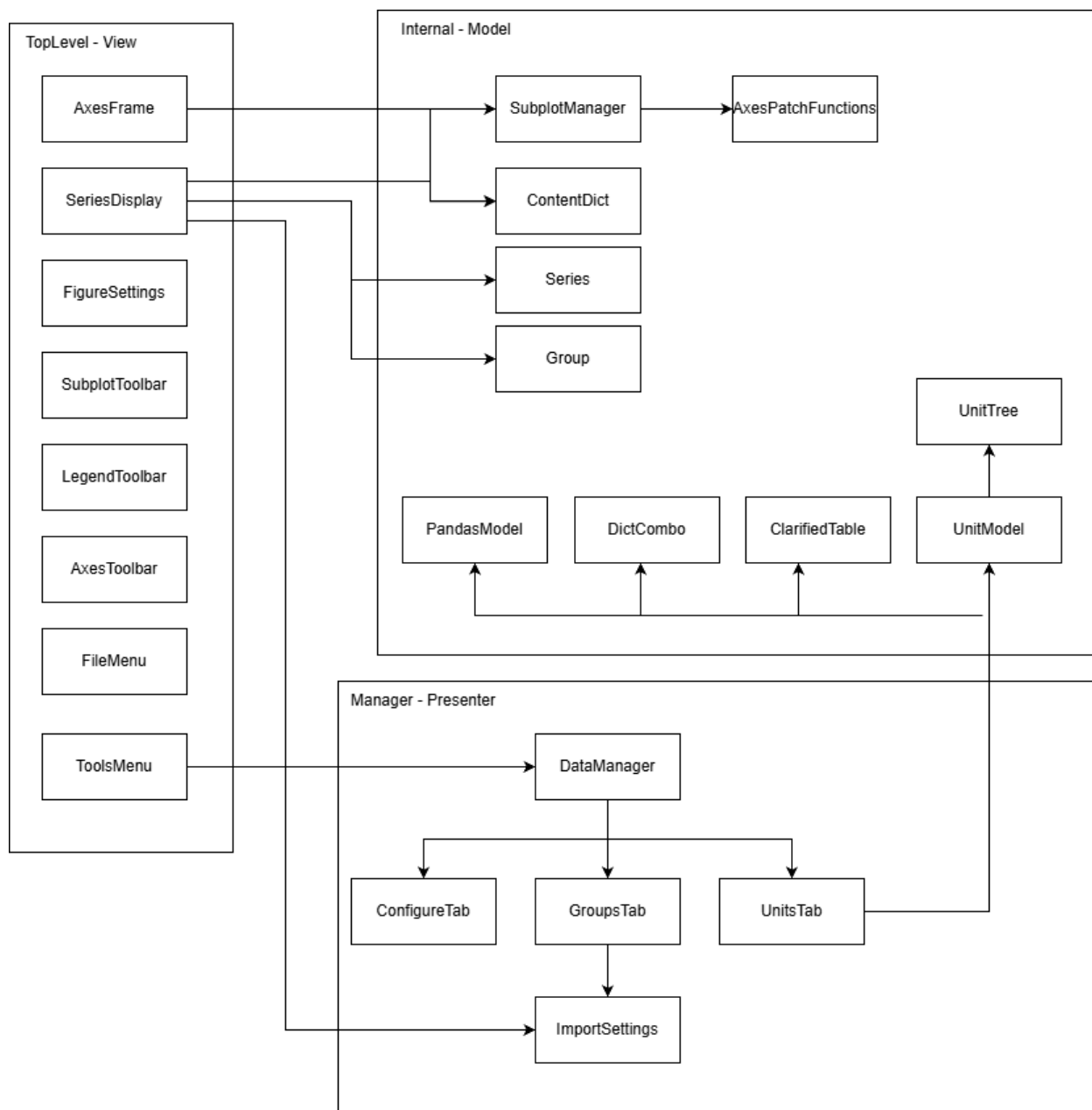


Рисунок 5 – UML-диаграмма модулей приложения

Диаграмма основных внутренних классов приложения и их взаимосвязи показана на рисунке 6 и на рисунке 7.

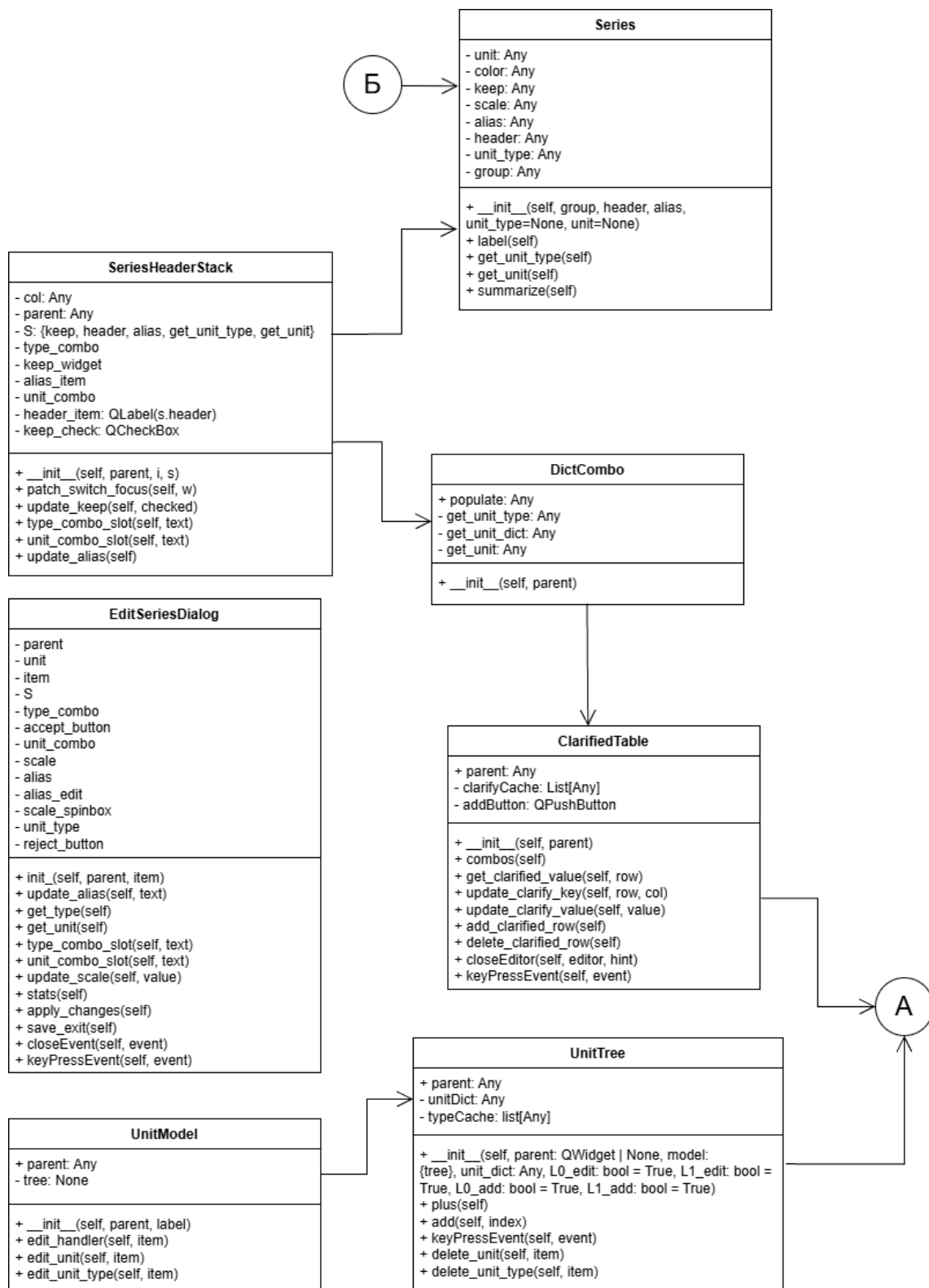


Рисунок 6 – Диаграмма классов приложения часть 1

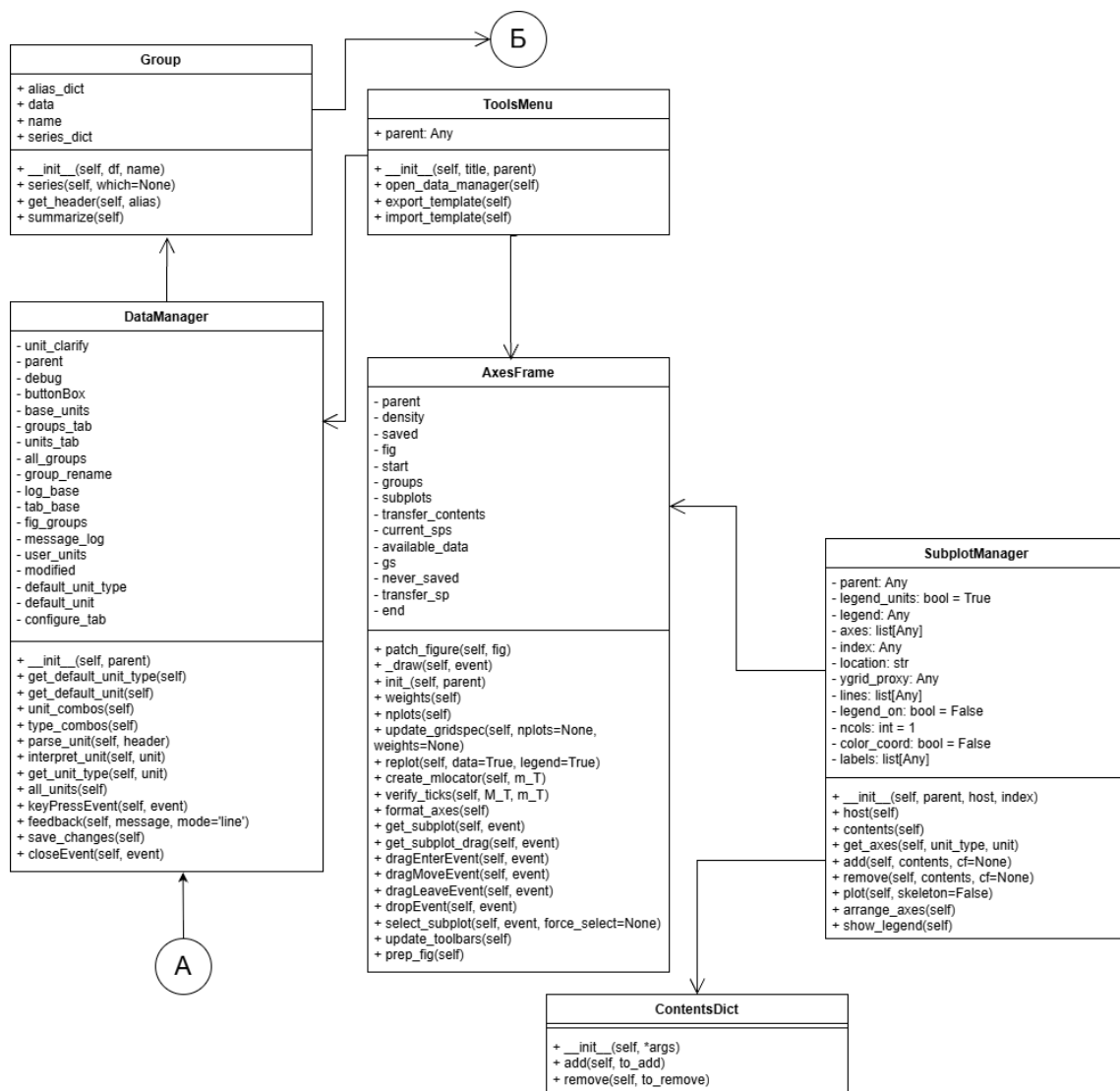


Рисунок 7 – Диаграмма классов приложения часть 2

2.3 Описание пользовательского интерфейса приложения

Основное окно программы – это первое, что видит пользователь после запуска приложения. В центральной области главного окна отображается активная фигура, содержащая один или несколько графиков. Графики можно выделять с помощью клавиш Ctrl или Shift, тем самым можно удалять или редактировать множественное их число. Под главным окном графиков реализована панель вкладок. Настройки содержимого, выделения графиков и прочие настройки, все они являются независимыми для каждой отдельной вкладки. Работа с вкладками включает добавление пустых, удаление и

переименование. Пример настройки графиков на разных вкладках представлен на рисунке 8.

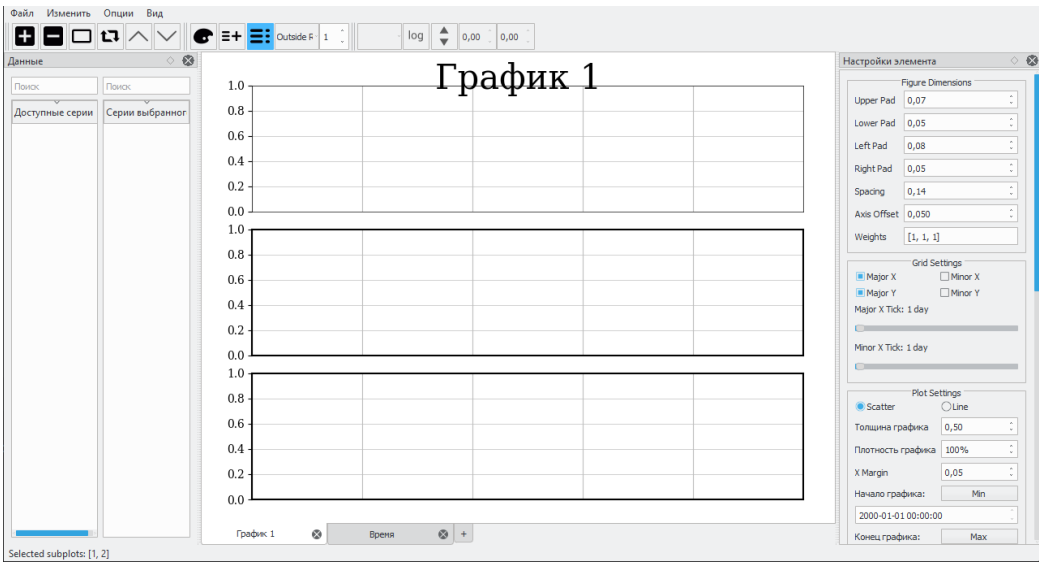


Рисунок 8 – Настройка вкладок в главном окне программы

Боковая панель «Настройки элемента» объединяет параметры оформления текущей фигуры. Внутри группы «Размеры фигуры» находятся поля для изменения отступов от краёв фигуры до области других графиков, заданные в относительных единицах, поля изменения сдвига осей графика и настройки размеров отображения графиков относительно друг друга.

В группе «Сетка» включается линии сетки внутри графика, а также задается частота отображение меток на оси времени. В разделе настройки графика имеется переключатель типа отображение графика с точечного на линейный, настройка толщины маркера и плотность всего графика в процентах. Ниже находятся опции установки начала и конца графика согласно временным меткам внутри импортированных данных, имеются специально выделенные кнопки для установки границ в минимальную и максимальную позицию соответственно. Имеется также настройка отступа от крайних точек данных до границ подграфика, выраженный в долях от общей длительности временного интервала.

Ниже расположены настройки размеров шрифта текста, который отображается вместе с графиком, меняя значения в полях, можно управлять размером шрифта заголовка фигуры, размером шрифта подписей осей Y (и оси X для нижнего подграфика), размером шрифта меток на осях, углом поворота подписей на оси X, а также можно задать собственный формат отображения даты и времени на оси X.

Таблица, представленная ниже сопоставляет каждому типу единиц измерения определённый цвет, используемый при включённой цветовой координации. Изменение цвета осуществляется щелчком по цветному квадрату. Новый цвет применяется немедленно ко всем рядам соответствующего типа во всех подграфиках текущей фигуры.

Боковая панель «Данные» содержит в себе два иерархических списка (дерева). «Доступные серии» содержит все импортированные ряды данных, которые ещё не добавлены на график текущей фигуры. Список «Серии выбранного графика» отображает содержимое выделенного графика. Добавление на график реализовано через перетаскивание элементов на целевой график. Поддерживается перетаскивание нескольких элементов, выделенных с помощью Ctrl или Shift. Удаление – перетаскивание элемента из этого дерева обратно в дерево «Доступные серии». Также можно выделить элементы и нажать клавишу Delete, что вернёт их в доступные.

У обоих списков реализовано поле поиска, которое позволяет найти необходимую серию с учетом регистра.

Редактирование свойств ряда происходит через двойной щелчок по любому ряду в любом из двух списков. Открывается диалог, в котором доступны следующие действия:

- изменение псевдонима, который отображается на графике вместо оригинального заголовка;
- выбор типа единиц (например, «Температура», «Скорость») и конкретной единицы измерения (например, «°C», «m/s»);

– установка масштаба – все значения ряда умножаются на заданный коэффициент при отображении (исходные данные не изменяются). Следует учитывать, что применение масштаба может изменить визуальное восприятие данных, если факт масштабирования не отражён в подписях;

– просмотр основной статистики ряда: количество точек, временные границы, минимальное и максимальное значения с соответствующими временными метками.

Пример работы с серией в окне данных, и настройка линии графика отображены на рисунке 9.

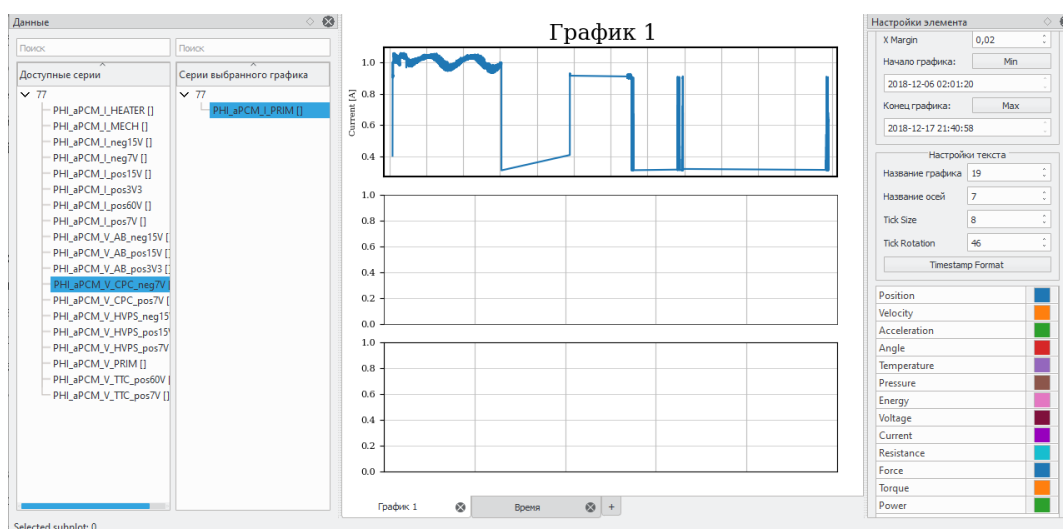


Рисунок 9 – Добавленная серия подвергается изменению через поля интерфейса «Настройки элемента».

В верхней части интерфейса сразу над блоком «Данные» расположены панели инструментов для работы с графиками. Кнопки, выполненные в виде значков, отвечают за следующие действия: добавить график – вставляет пустой график ниже выделенного. Если выделено несколько графиков или ни одного, новый график добавляется вниз. Удалить график – удаляет выделенные графики, а их содержимое возвращается в дерево «Доступные серии». Очистить график – удаляет все ряды из выделенных подграфиков, оставляя сами подграфики пустыми. Поменять оси – циклически переставляет

порядок правых осей в выделенных подграфиках. Поднять график / Опустить график – перемещает выделенный подграфик на одну позицию вверх или вниз (доступно только при выделении одного подграфика).

Следующая за ней панель инструментов отвечает за настройку легенды графика. В этой секции кнопки переключают отображение легенды – показывает или скрывает легенду, включает режим, в котором цвет линий и подписей определяется типом единиц, а не отдельным рядом. Ряды одного типа получают одинаковый цвет, но различаются маркерами или стилями линий. Отображение единиц измерения в легенде – при включении в легенде отображается запись с указанным псевдонимом. Положение легенды – выбор из предопределённых позиций, при этом область графика автоматически сжимается влево с учётом всех правых осей. Колонки легенды – задаёт количество столбцов в легенде (от 1 до 10). Текущая ось – выпадающий список, позволяющий выбрать, какая ось будет подвергаться настройке. Пункт «All» применяет действия ко всем осям подграфика. Логарифмический масштаб – переключает шкалу оси Y между линейной и логарифмической. Авто масштабирование – при включении пределы оси Y подбираются автоматически по отображаемым данным. При выключении становятся активными поля Y-Axis Min / Max, позволяющие задать пользовательские границы. Эти границы сохраняются даже после повторного включения авто масштабирования и восстанавливаются при последующем отключении.

Y-Axis Min / Max – числовые поля для ввода минимального и максимального значения оси. Доступны только для выбранной конкретной оси и при выключенном авто масштабировании.

В верхней части окна в выпадающем меню, вызываемым кнопкой «Файл» можно вызывать базовые функции программы по работе с вкладками – закрытие и сохранение, сохранение визуального отображение графика в формате png и выход из приложения.

В выпадающем меню «Опции» находятся функции экспорта и импорта конфигурации, при экспорте сохраняются все ранее установленные параметры в блоке «Настройки элемента» и другие настройки графиков, а «Импорт данных» открывает новое окно в окно будет происходить вся настройка, связанная с добавлением информации в приложение. В меню «Вид» происходит переключение между цветовыми схемами приложения и быстрое скрывание боковых панелей интерфейса для более удобного просмотра основного окна. Менеджер импорта данных – отдельное окно, предназначенное для загрузки файлов, настройки параметров импорта и формирования групп данных, которые затем становятся доступны в главном окне. Вид полного окна показана на рисунке 10.

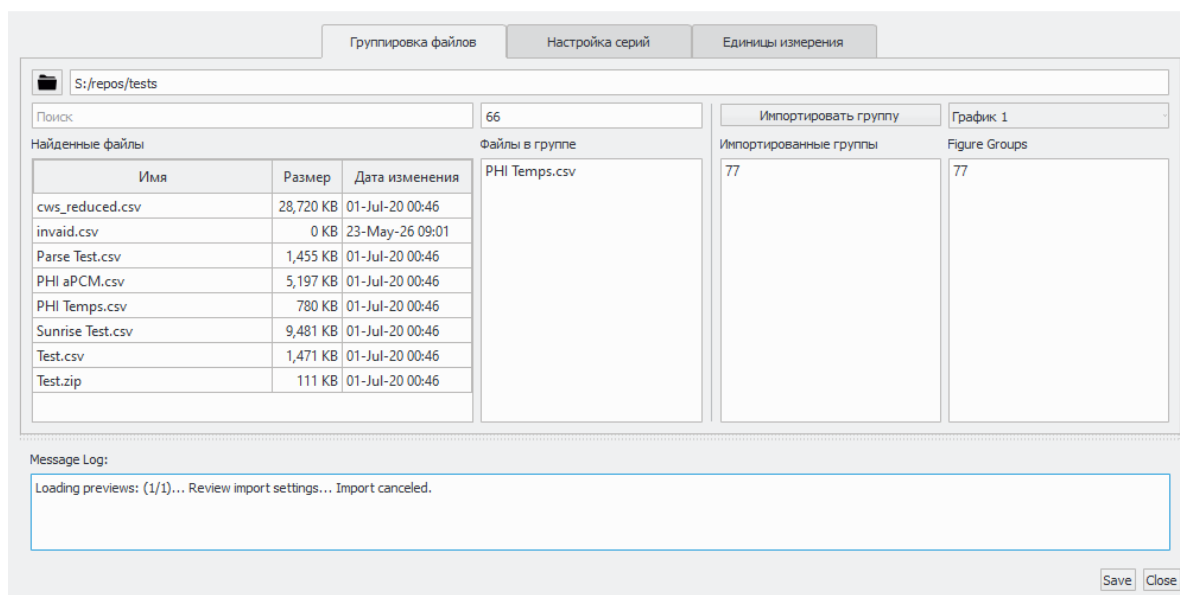


Рисунок 10 – Менеджер импорта данных на вкладке группировки файлов

Во вкладке «Группировка файлов» указывается путь к папке, содержащей CSV-файлы или ZIP-архивы, внутри которых находится ровно один CSV-файл. После нажатия Enter отображаются все файлы с расширениями .csv и .zip в указанной папке. Создаётся группа – задаётся имя и добавляются файлы. Один файл может входить в несколько групп. При нажатии кнопки Импорт Группы открывается диалог предварительный просмотр и настройка

параметров импорта. В левой части диалога отображается список файлов, входящих в группу. При выборе файла в правой части показывается его содержимое в табличном виде. Параметры импорта, строка с заголовками, столбец с временными метками, разделитель, кодировка настраиваются последовательно для каждого файла. Результат применения параметров отображается в виде предпросмотра: синим цветом подсвечиваются столбцы, распознанные как заголовки рядов данных. Оранжевым цветом – столбец, определённый как содержащий временные метки. Кнопки управления: Auto-Detect – автоматический подбор параметров для всех выбранных файлов. Reset – возврат параметров выбранных файлов к состоянию на момент открытия диалога. Поддерживается копирование и вставка параметров из одной строки в другую (выделение ячейки, Ctrl+C, затем выделение ячейки в другой строке того же столбца, Ctrl+V). Допустимо копирование только одной строки за раз, причём ширина выделения (количество столбцов) должна совпадать. После завершения настройки и подтверждения корректности подсветок нажатием кнопки Confirm выполняется импорт группы.

После успешного импорта группа отображается в списке импортированных. Переименование группы осуществляется двойным щелчком по её названию. Если открыто несколько фигур, можно перетаскивать группы из списка импортированных в список текущей фигуры, делая их доступными для построения графиков. Одна группа может использоваться в разных фигурах.

Вкладка «Настройка серии» становится доступна после импорта группы. Внутри нее расположена таблица для просмотра и редактирования метаданных каждого ряда. Таблица содержит следующие столбцы:

Keep – флажок, определяющий, будет ли ряд передан в главное окно (впоследствии его можно добавить на график). Original Header – исходный заголовок столбца из файла (не редактируется). Alias – псевдоним, отображаемый на графиках. При пустом значении используется оригинальный

заголовок. Unit Type – тип единиц (выбирается из выпадающего списка). Unit – конкретная единица измерения, соответствующая выбранному типу.

По нажатию кнопки экспорта происходит сохранение группы в CSV-файл с учётом флажков Keep. Заголовки столбцов формируются с учетом объявленных алиасов, что обеспечивает совместимость с последующим импортом. Визуальное представление загруженных и измененных данных в этой вкладке представлено на рисунке 11.

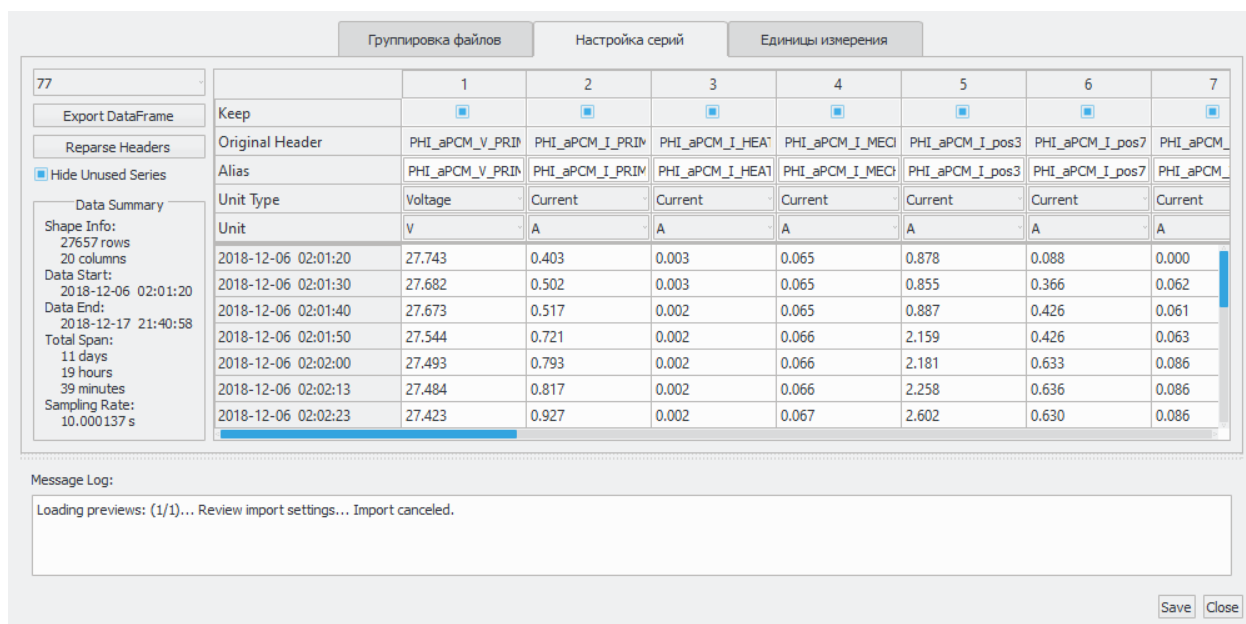


Рисунок 11 – Вкладка «Настройка серий» в менеджере импорта данных

Внутри дерева единиц находятся встроенные так и пользовательские единицы, и типы данных. Вкладка позволяет управлять словарём единиц, который применяется при парсинге заголовков. Все изменения сохраняются в конфигурационном файле config.json и действуют глобально. Разрешено переименование, удаление и добавление новых типов и единиц. Предусмотрено создание собственных типов без обязательной привязки к конкретным единицам. При парсинге пользовательские единицы имеют приоритет перед базовыми. Сводка единиц измерения для конкретного импорта данных и общий вид вкладки представлены на рисунке 12.

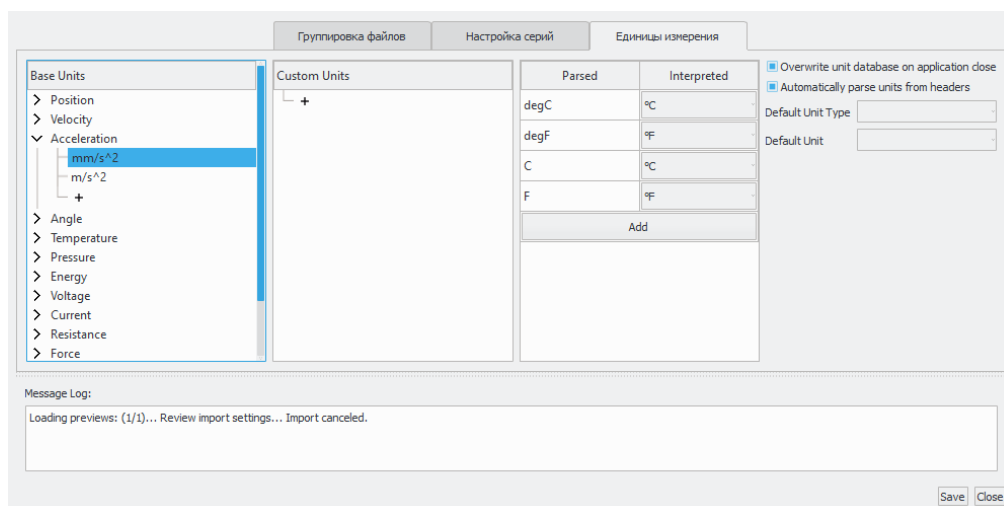


Рисунок 12 – Вкладка «Единицы измерения» в менеджере импорта данных

Таблица «Clarified Units» служит для сопоставления различных строковых обозначений одной и той же единице. Например, градусы Фаренгейта могут обозначаться как «°F», «degF» или «F». Вместо добавления всех вариантов в список единиц типа «Температура» они перечисляются в таблице уточнений, где каждому варианту ставится в соответствие каноническое обозначение единицы. Это позволяет алгоритму парсинга распознавать все варианты и приводить их к единому виду. Добавление нового уточнения осуществляется кнопкой Add, после чего вводится исходная строка и выбирается целевая единица из выпадающего списка. Удаление строки производится её выделением и нажатием Delete.

2.4 Тестирование приложения

В ходе разработки данного приложения и при проверке его работоспособности на финальном этапе было произведено множество тестов, цель которых была проверить корректность работы всех элементов пользовательского интерфейса программы, внутренних алгоритмов работы, импорта и экспорта файлов и конфигураций.

Во время разработки в код был добавлен пользовательский обработчик ошибок, который заменяет стандартный при вызове исключений и логирует стек ошибки в заранее указанное место.

Тестирование работоспособности программы заключалось в использовании всех элементов глобального пользовательского интерфейса при различных этапах работы программы.

Для того чтобы провести полное тестирование, необходимо представлять все возможные варианты работы с приложением со стороны пользователя, на основе описания модулей приложения и опираясь на его графический интерфейс была построена UML-диаграмма вариантов использования, показанная на рисунке 13.

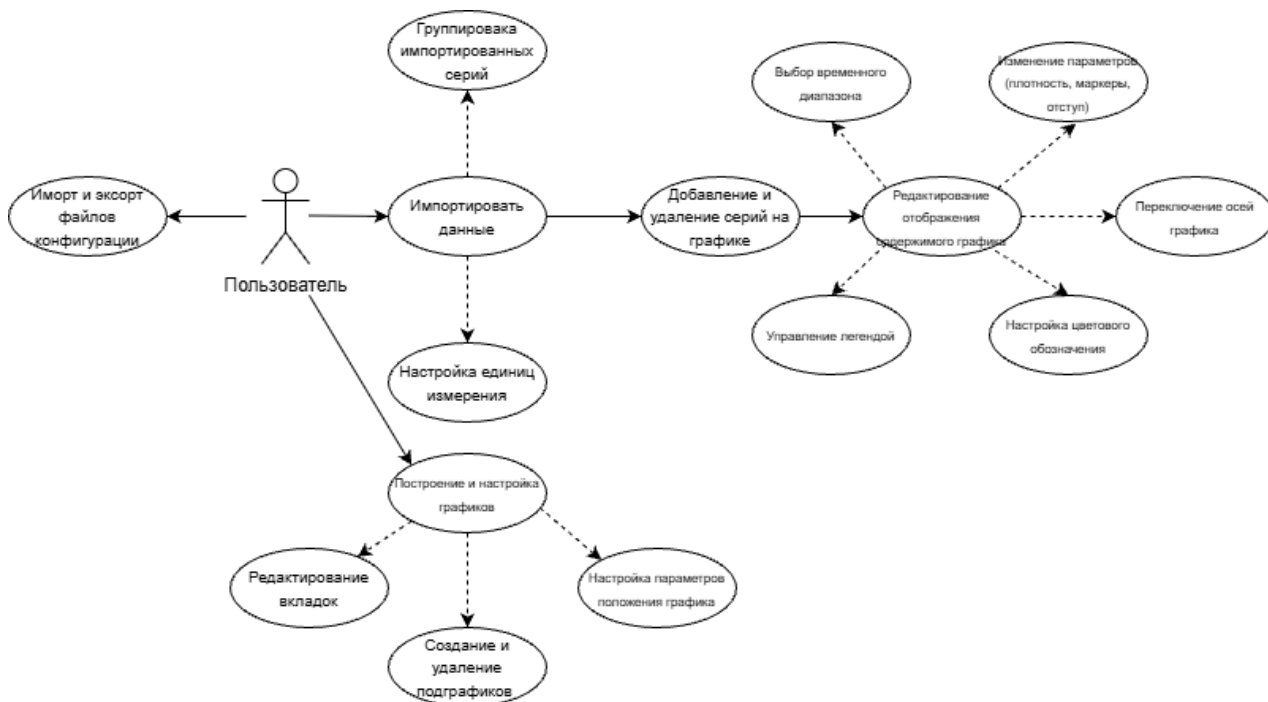


Рисунок 13 – UML-диаграмма вариантов использования приложения

При первом запуске, в главном окне создается вкладка с пустым графиком, как видно на рисунке 14. С таким графиком не должны происходить никакие операции, так как в нем нет добавленных серий.

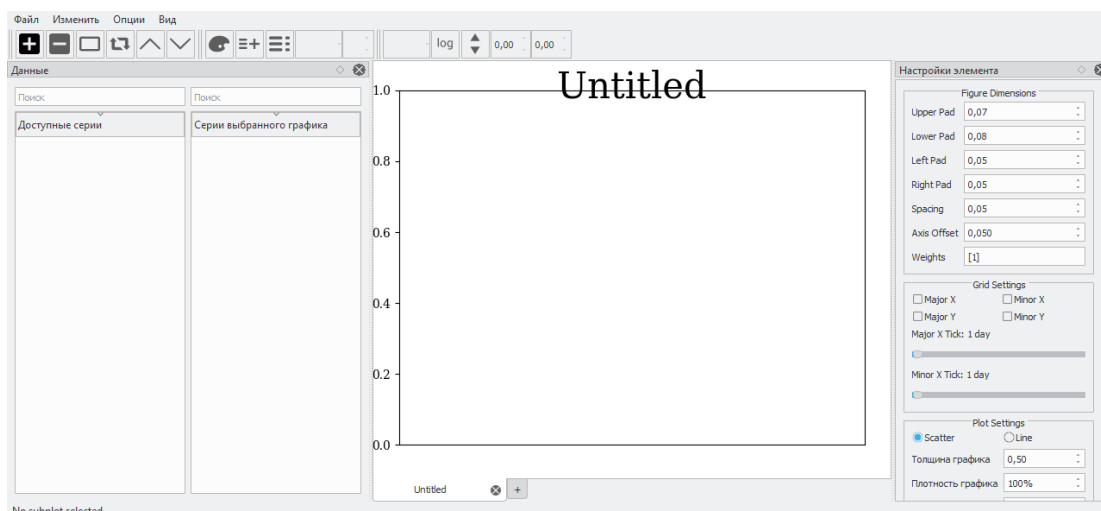


Рисунок 14 – Главное окно программы

Для продолжения работы с программой необходимо перейти в менеджер импорта данных, который показан на рисунке 15 и создать группу из файла или нескольких файлов, содержащих телеметрическую информацию.

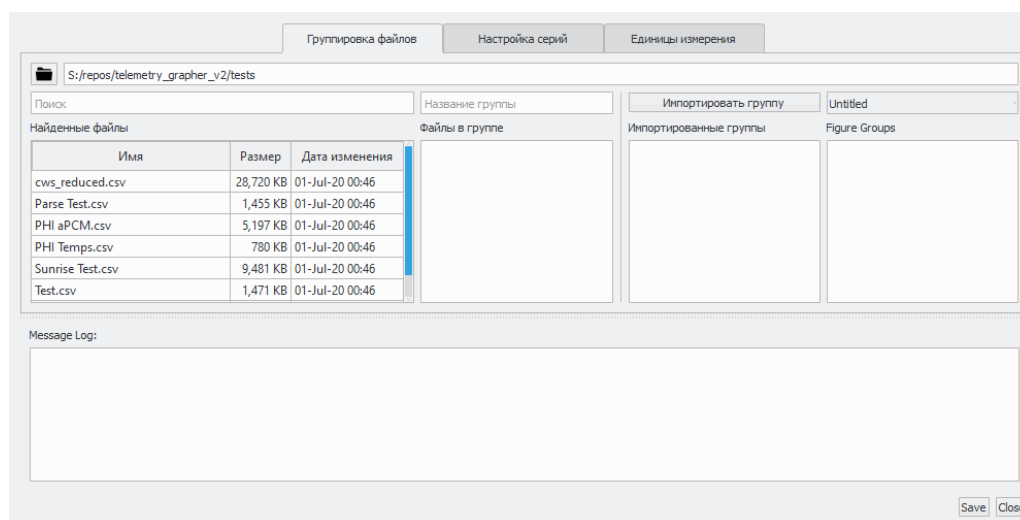


Рисунок 15 – Окно менеджера импорта

При попытке импорта файла некорректного формата, файла с неправильными заголовками, битого файла, пустого файла и прочих вариаций неисправных файлов менеджер покажет в логге ошибку и не импорта не произойдет, что видно на рисунке 16.

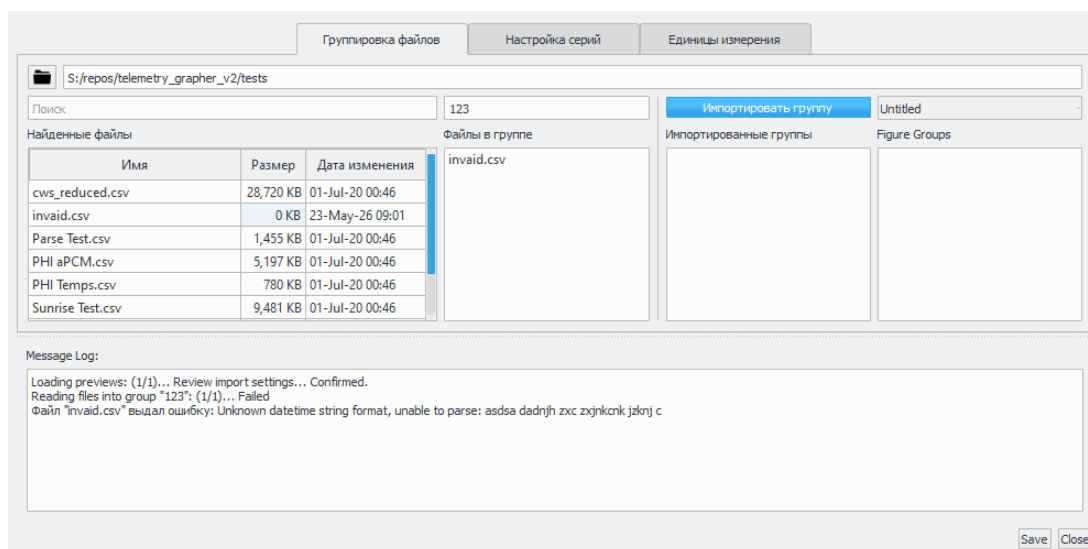


Рисунок 16 – Попытка импорта неправильного файла

Если же файл является корректным, то пользователь увидит окно предпросмотра и настройки импорта. На рисунке 17 происходит импорт файла размер в 28 мегабайт формата csv, полученный из открытого источника и являющийся датасетом реальных телеметрических данных [5]. Таким образом проверяется работоспособность и производительность приложения при работе с большим набором данных. Никаких отклонений замечено не было и импорт был произведен успешно, что видно на рисунке 18.

File					Datetime Format	Header Row	Index Column	Skip Rows
1	cws_reduced.csv				Infer	0	0	None

Рисунок 17 – Окно предпросмотра данных для файла большого размера

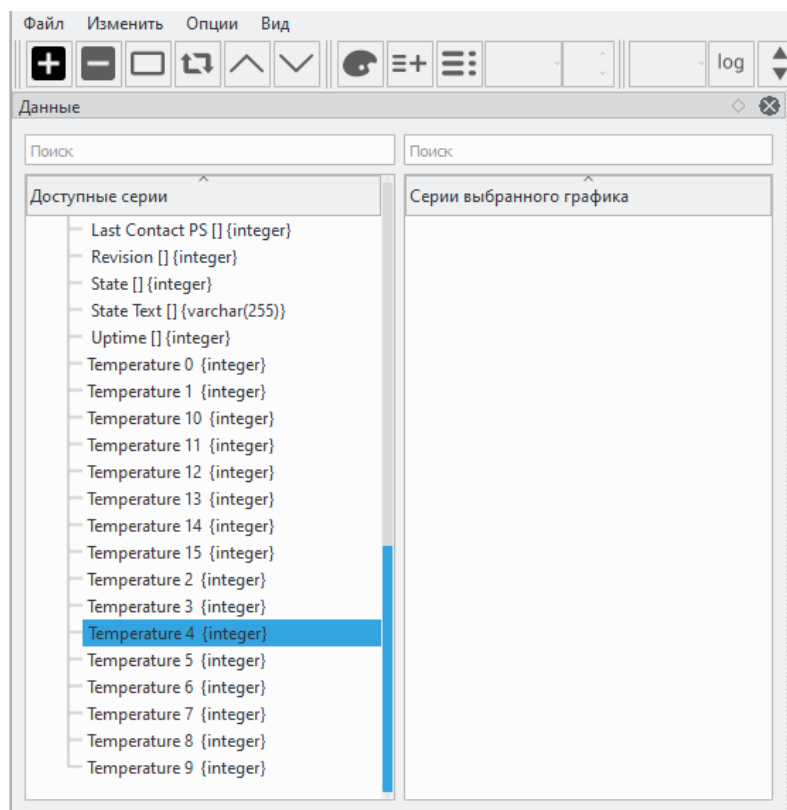


Рисунок 18 – Результат импорта данных в главном окне

Далее пользователь может продолжить работу в основном окне и теперь будет происходить проверка всех внутренних функций приложения, связанных с элементами интерфейса, после произвольной работы над графиком, ошибок зафиксировано не было. Финальный вид главного окна показан на рисунке 19.

Внутри приложения также предусмотрены различные сообщения об ошибке, которые сигнализируют пользователю и неправильном вводе информации, попытке взаимодействия с приложением, которая может вызвать необработанное исключение или просто ошибках, связанных с неправильной работой системы.

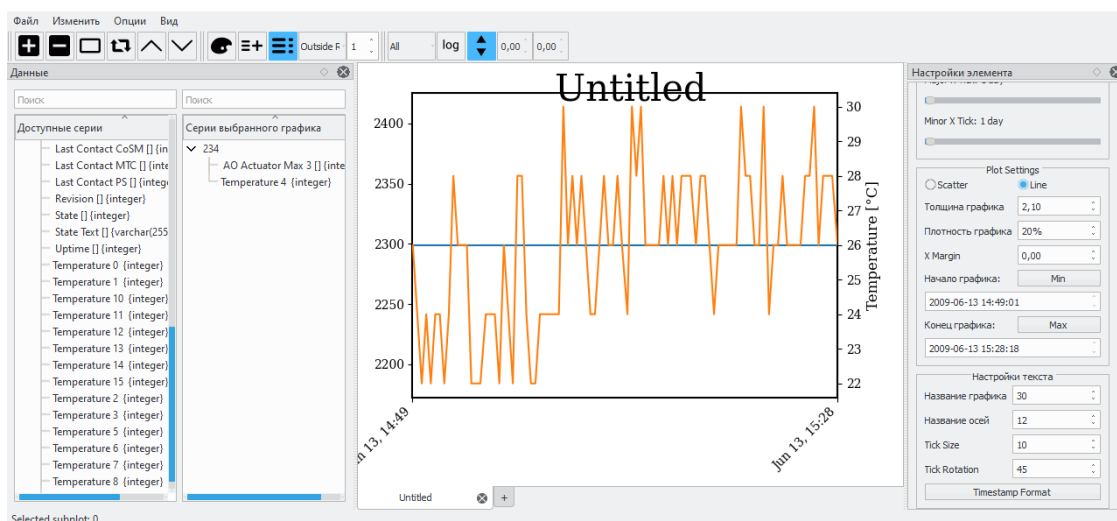


Рисунок 19 – Вид графика после произвольных изменений настроек со стороны пользователя

Одним из таких сообщений является вывод ошибки «неверный путь к файлу» при попытке указать несуществующую директорию файловой системы, вид ошибки показан на рисунке 20.

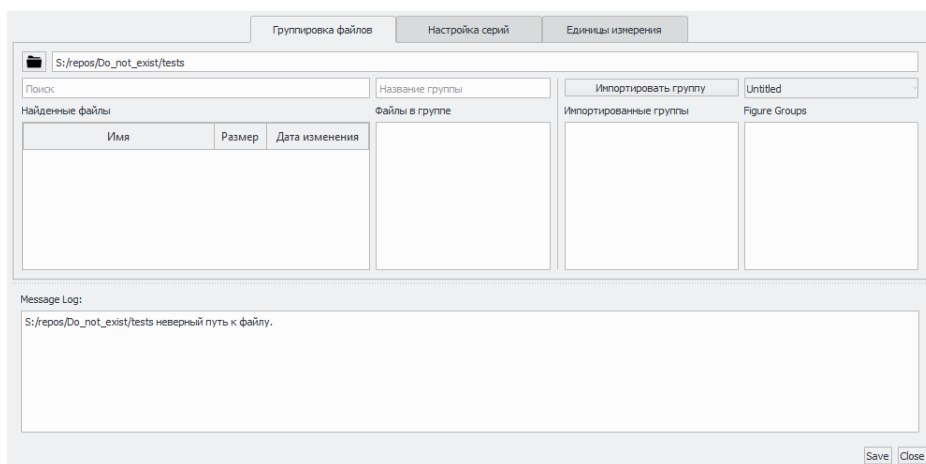


Рисунок 20 – Вызов ошибки «неверный путь к файлу»

Аналогично программа откажет в создании группы, если пользователь не укажет ничего в соответствующем поле ввода, что видно на рисунке 21.

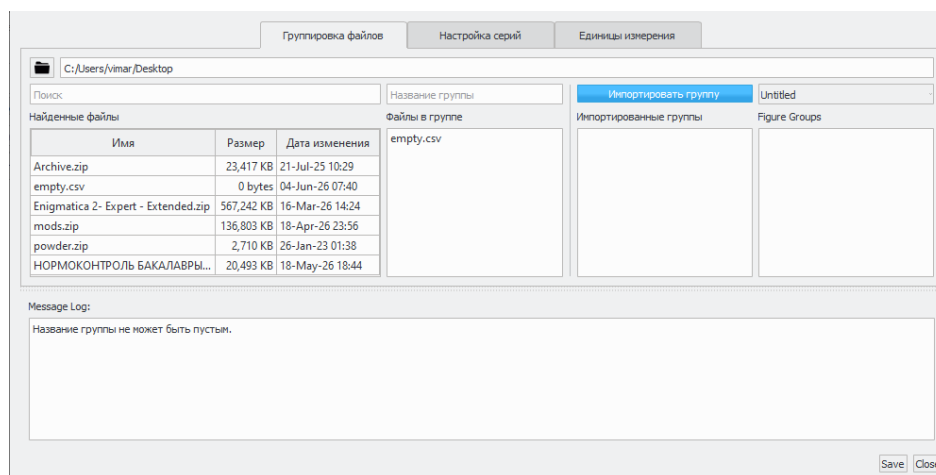


Рисунок 21 – Вызов предупреждения «название группы не может быть пустым»

Еще одним неотъемлемо важным предупреждением для пользователя при работе с любым приложением и особенно в работе с данными телеметрии является предупреждение о несохраненных данных и различные диалоги подтверждения, предотвращающие возможность случайно закрытия приложения и последующей потери данных и изменений в ходе работы.

Если вкладка не подверглась сохранения ни разу за период работы, то при ее закрытии нажатием на крест у окна вкладки, пользователь получает уведомление как на рисунке 22.

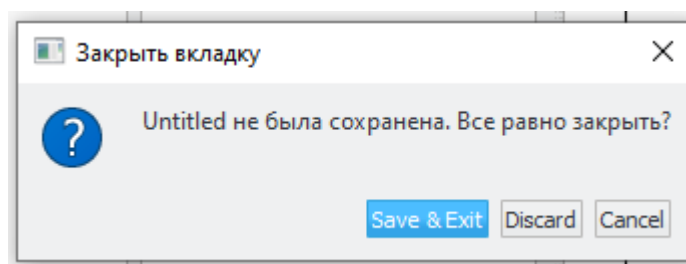


Рисунок 22 – Попытка закрытия несохраненной вкладки

Если пользователь попытается закрыть приложение через стандартные элементы управления Windows, то он получит похожее сообщение с вариантами отмены, закрытия и закрытия с предварительным сохранением, что показано на рисунке 23.

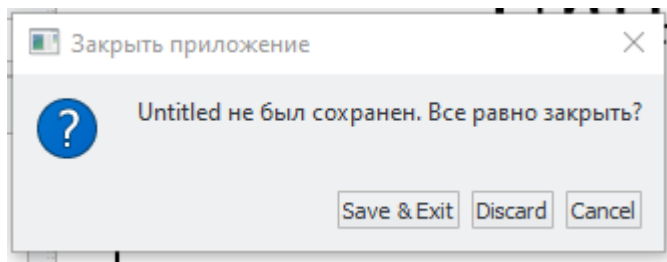


Рисунок 23 – Попытка закрытия с несохраненными изменениями в приложении

3 Охрана труда

Работа с ПК в офисных помещениях может скрывать в себе риски для здоровья при неправильном обращении с техникой, отсутствием контроля за соблюдением норм, описанных в регламентирующих документах и просто доставлять дискомфорт. В связи с этим во время работы в помещениях, где происходит постоянная работа с ЭВМ должны поддерживаться следующие требования:

- освещение должно соответствовать стандартам, а в случае отсутствия естественного освещения, искусственное должно компенсировать его нехватку, что должно быть доказано с использованием измерительных приборов;
- компьютеры не должны размещаться в местах, где есть риск попадания влаги;
- рабочие места с компьютерами должны быть оборудованы защитным заземлением;
- в помещениях с компьютерами следует проводить ежедневную влажную уборку и проветривание каждый час;
- рекомендуемая температура воздуха в помещении с компьютерами составляет $+19^{\circ}\text{C} - +21^{\circ}\text{C}$;
- рабочее место, например, стол, должно быть спроектировано таким образом, чтобы обеспечить оптимальное расположение используемого оборудования в соответствии с выполняемой работой.

Помимо необходимости соблюдения условий для помещений, правильно подготовленное и оборудованное рабочее место не менее важно. При организации рабочих мест с компьютерами необходимо учитывать, что расстояние между рабочими столами, на которых установлены видеомониторы, должно быть не менее 2,0 метров в направлении между ними и не менее 1,2 метра между боковыми поверхностями видеомониторов. Запрещается размещать рабочие места с компьютерами рядом с

силовыми кабелями, высоковольтными трансформаторами и другим оборудованием, способным создавать помехи в работе компьютеров. Системный блок и монитор следует разместить так, чтобы расстояние от вентиляционных отверстий этих устройств до стены (или стенки мебели) составляло не менее 10 сантиметров. Экран видеомонитора должен находиться от глаз пользователя на расстоянии 600 – 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов. Искусственное освещение в помещениях для эксплуатации компьютеров должно осуществляться системой общего равномерного освещения. Освещенность на поверхности рабочего стола в зоне размещения ПК должна быть 300-500 люкс [6].

Заключение

В ходе выполнения выпускной квалификационной работы на тему «Приложение анализа и визуализации телеметрических данных» была успешно достигнута поставленная цель - разработано десктопное приложение, обеспечивающее импорт, обработку и наглядное графическое представление структурированных телеметрических данных, хранящихся в локальных файлах форматов CSV и LOG.

Для достижения указанной цели были решены основные задачи.

Проведён всесторонний анализ предметной области и обзор существующих систем анализа и визуализации телеметрических данных, включая веб-ориентированное решение Grafana, легковесную платформу OpenObserve и узкоспециализированное ПО RemusLab. Выявлены их ключевые недостатки и сформулированы требования к разрабатываемому приложению, определены основные сценарии использования и целевая аудитория.

Обоснован выбор языка программирования Python, обусловленный наличием мощных библиотек для обработки данных (Pandas, NumPy) и визуализации (Matplotlib), а также графического фреймворка PyQt5, обеспечивающего создание интуитивно понятного кроссплатформенного интерфейса. В качестве среды разработки выбрана PyCharm, предоставляющая нативную поддержку PyQt и инструменты для эффективной отладки.

В качестве основы архитектуры выбран паттерн MVP (Model-View-Presenter), обеспечивающий чёткое разделение логики отображения и бизнес-логики, что способствует удобству тестирования и дальнейшего сопровождения кода. Разработан комплект UML-диаграмм (вариантов использования, последовательности, классов, модулей), наглядно отражающих структуру и динамику взаимодействия компонентов системы.

Созданы следующие функциональные блоки приложения:

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		45

– менеджер импорта данных – осуществляет парсинг CSV и LOG файлов (включая ZIP-архивы), автоматическое распознавание заголовков столбцов, разделителей, кодировок и временных меток, а также сопоставление извлечённых строк с пользовательскими единицами измерения через таблицу уточнений;

– система управления единицами измерения – поддерживает создание, редактирование и удаление пользовательских типов и единиц;

– модуль визуализации – построен на базе Matplotlib с кастомизированной обработкой осей (множественные правые оси, поддержка логарифмического масштаба, автоматический и ручной выбор пределов);

– система сохранения настроек – реализована загрузка пользовательских конфигураций (единицы измерения, цветовая кодировка, тема оформления) из JSON-файла при запуске и сохранение изменений при штатном завершении работы; предусмотрен экспорт и импорт шаблонов настроек графиков для повторного использования.

Проведено функциональное тестирование сценариев использования, включая импорт файлов корректной и некорректной структуры (пустые, битые, с неправильными заголовками), возможные операции по настройке графиков и управлению вкладками. Реализована система логирования ошибок с записью стека вызовов в файл errors.log и выводом пользователю понятных сообщений о нештатных ситуациях.

В рамках раздела «Безопасность жизнедеятельности» рассмотрены нормативные требования к организации рабочего места оператора ЭВМ.

Разработанное приложение представляет собой самостоятельное законченное программное решение, которое может быть использовано для анализа телеметрических данных в различных предметных областях: промышленном мониторинге, энергетике, транспортных системах, вендинге, сельском хозяйстве, а также в научных исследованиях, связанных с обработкой временных рядов. В отличие от существующих аналогов,

приложение ориентировано на работу с локальными файлами произвольной структуры, не требует развёртывания дополнительного программного стека, предоставляет интуитивно понятный интерфейс и гибкие средства настройки визуализации.

В качестве перспективных направлений совершенствования системы можно выделить: расширение спектра поддерживаемых форматов входных файлов (Parquet, HDF5, Excel); добавление инструментов статистического анализа данных непосредственно в интерфейс приложения; разработку веб-интерфейса на базе существующего ядра для обеспечения удалённого доступа; интеграцию с системами сбора телеметрии через протоколы MQTT или Modbus.

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		47

Список использованных источников

1. Документация к библиотеке numpy. [Электронный ресурс] / Режим доступа: <https://numpy.org/doc/stable/user/index.html#user> свободный. (дата обращения 16.05.2026).

2. Документация к библиотеке Matplotlib. [Электронный ресурс] / Режим доступа: <https://matplotlib.org/stable/users/index.html> свободный. (дата обращения 16.05.2026).

3. Руководство по разработке десктопных приложений. [Электронный ресурс] / Режим доступа: <https://docs.astralinux.ru/latest/desktop/python/pyqt5/> свободный. (дата обращения 14.05.2026).

4. Документация Python: шаблон проектирования модель представление презентер (MVP). [Электронный ресурс] / Режим доступа: https://github.com/mantidproject/documents/blob/main/Presentations/DeveloperInduction/Python_MVP.md свободный. (дата обращения 14.05.2026).

5. Набор данных для проверки работоспособности приложения. [Электронный ресурс] / Режим доступа: <https://www.kaggle.com/search?q=telemetry+in%3Adatasets> свободный. (дата обращения 16.05.2026).

6. Гигиенические требования к персональным электронно-вычислительным машинам и организация работы. [Электронный ресурс] / Режим доступа: https://10.rospotrebnadzor.ru/upload/medialibrary/95a/2.2.2_2.4.1340_03.pdf свободный. (дата обращения 16.05.2026).

Приложение А

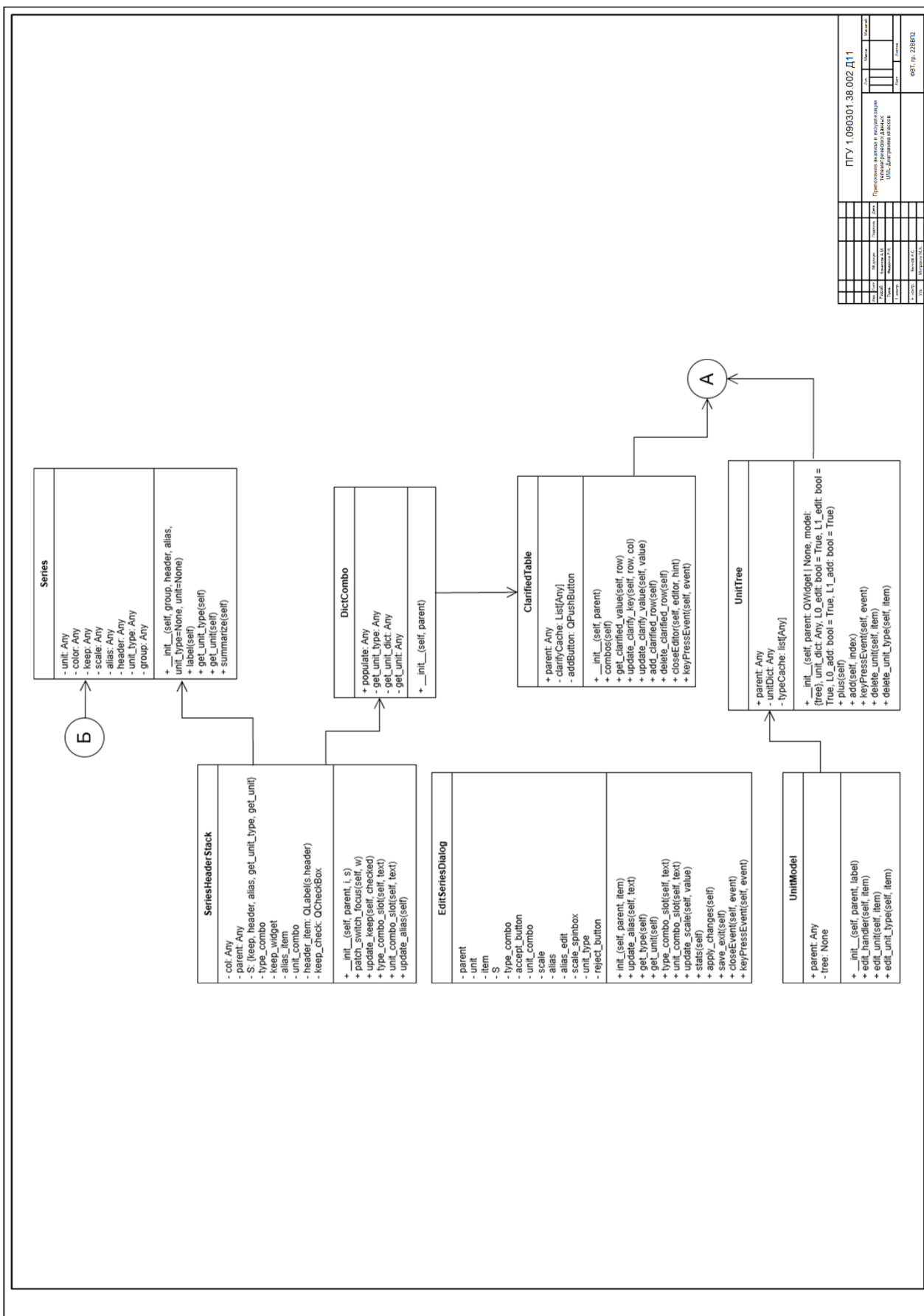
(обязательное)

Приложение анализа и визуализации телеметрических данных

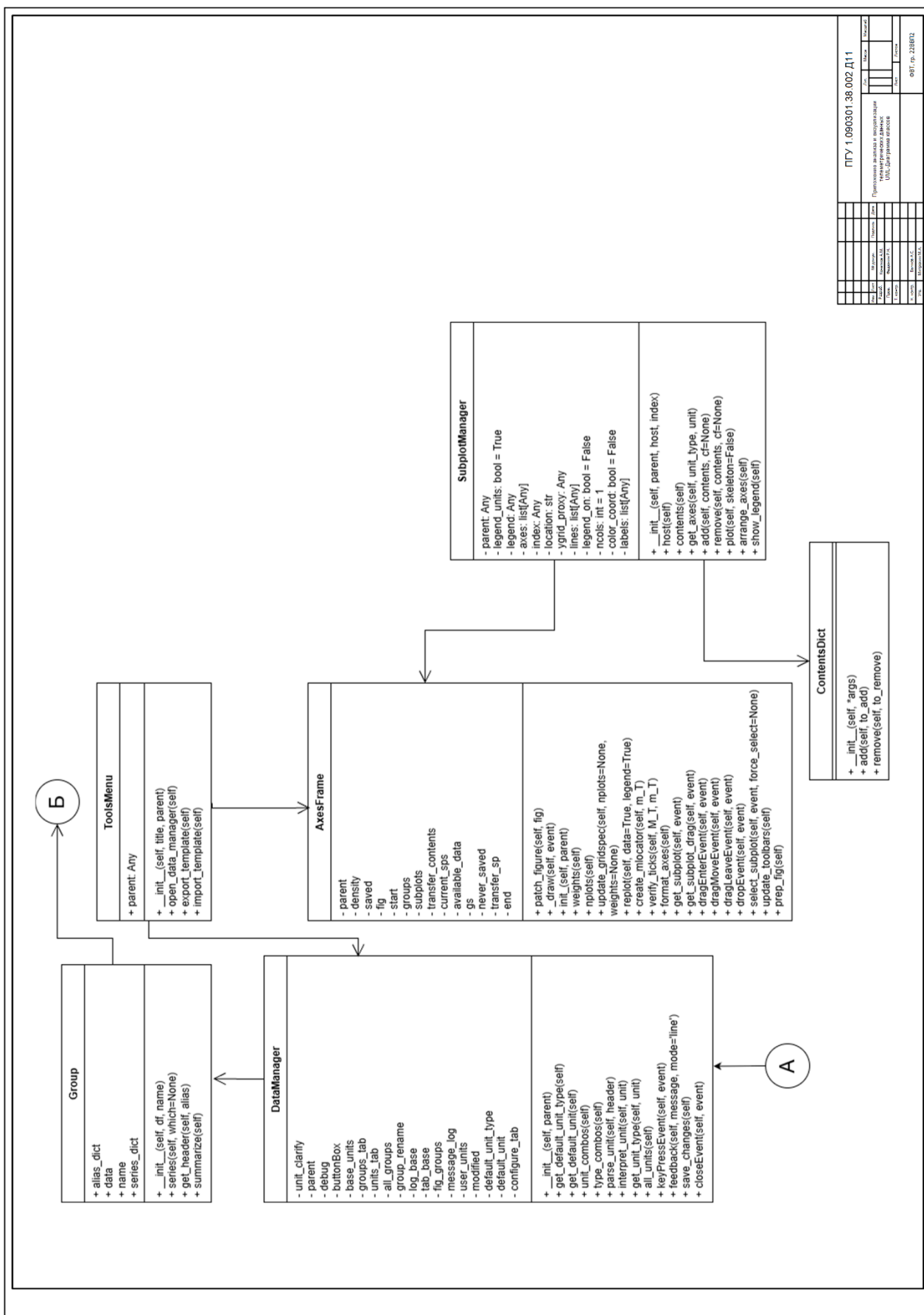
UML-диаграммы

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		49

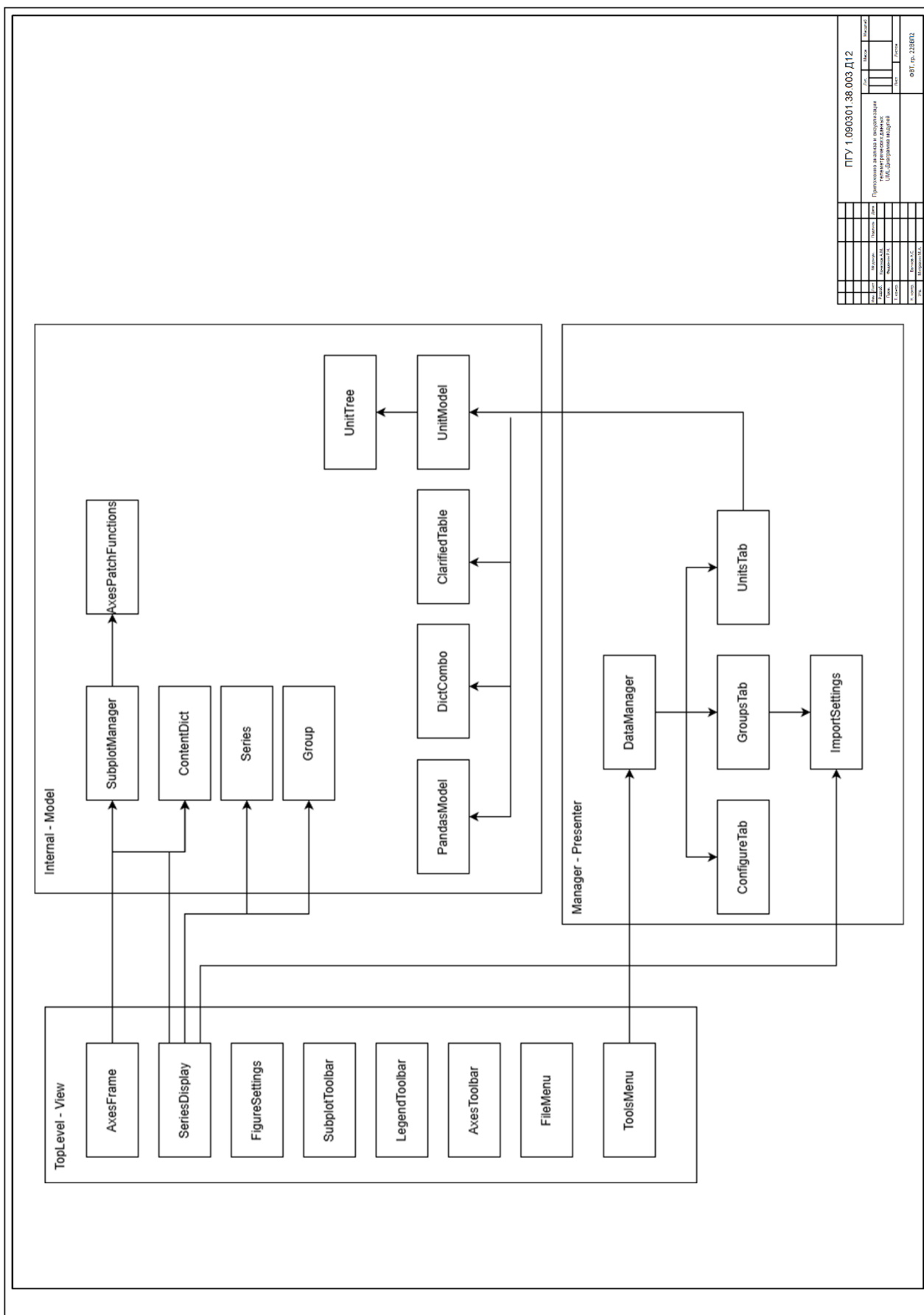
Приложение А.1 – Диаграмма классов



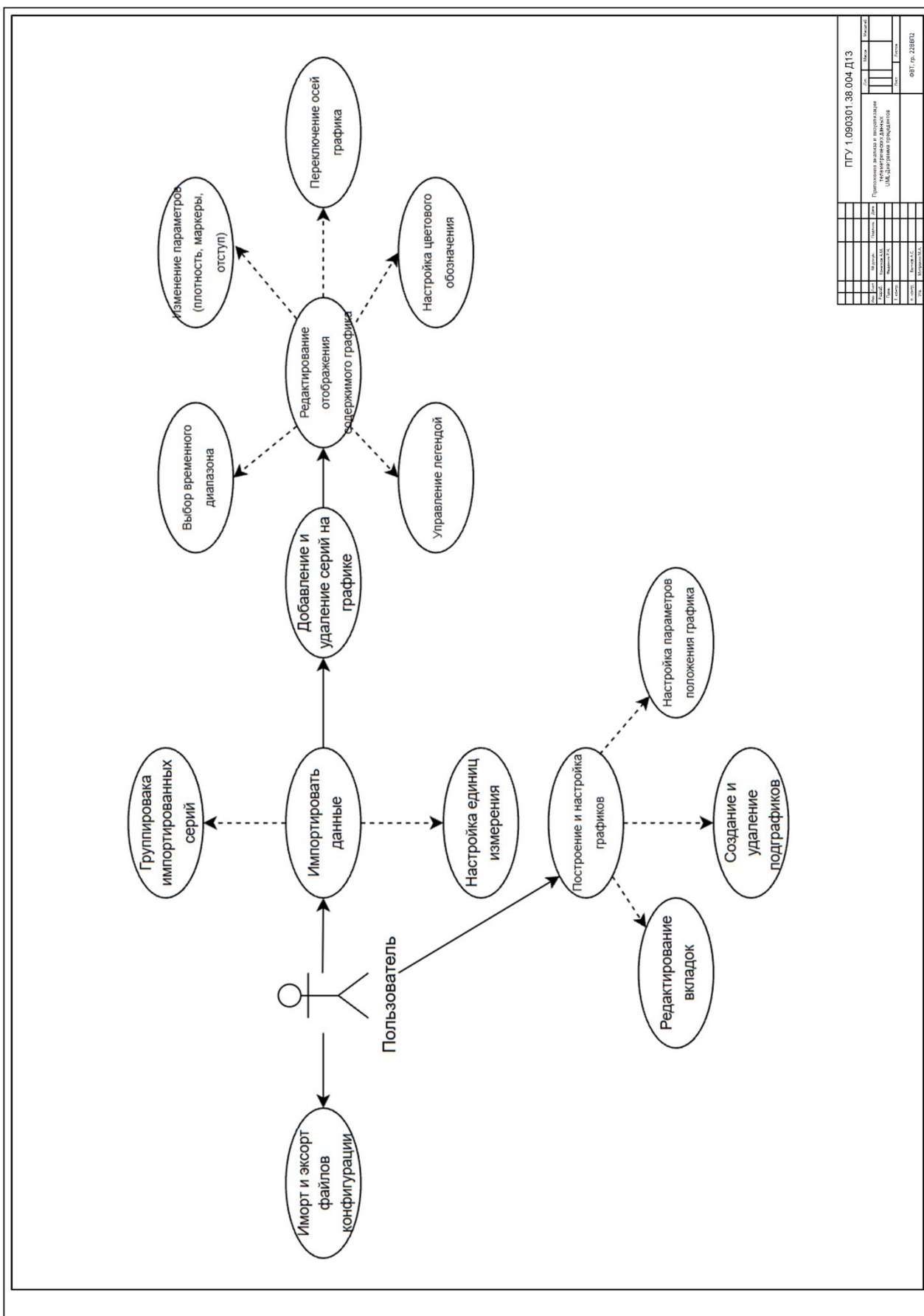
Приложение А.2 – Диаграмма классов



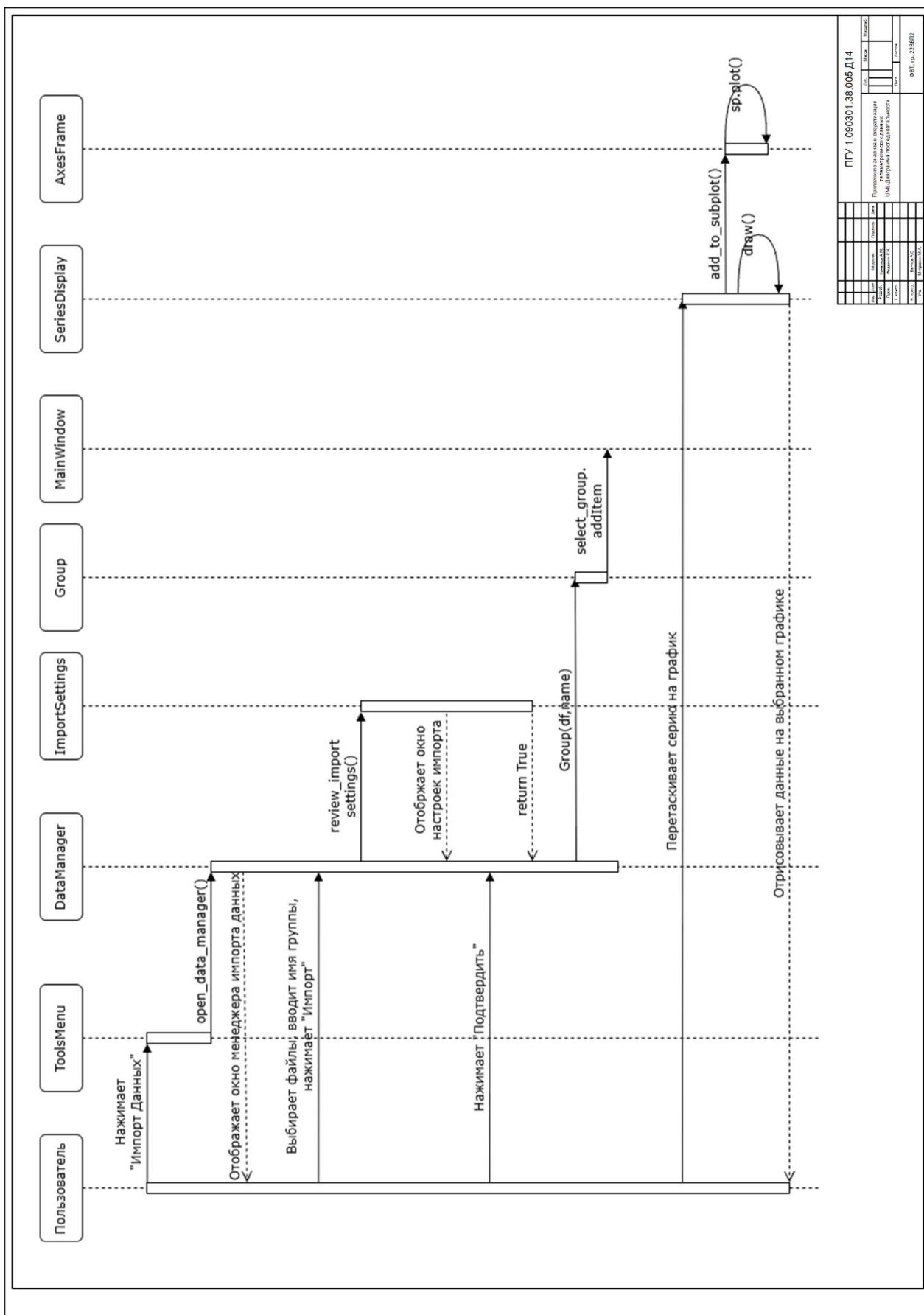
Приложение А.3 – Диаграмма модулей



Приложение А.4 – Диаграмма прецедентов



Приложение А.5 – Диаграмма последовательности



Приложение Б

(обязательное)

Приложение анализа и визуализации телеметрических данных

Листинг программы

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

Файл axes_patch_functions.py

```
# -*- coding: utf-8 -*-

from math import log10
import numpy as np
import matplotlib.pyplot as plt

from main_proj.classes.internal.contents_dict import ContentsDict

def patch_ax(self, ax):
    ax.parent = self
    ax.contents = ContentsDict()
    ax.color = 'k'
    ax.set_adjustable('datalim')
    ax.patch.set_visible(False)

    ax.log = False
    ax.auto_limits = True
    ax.custom_limits = []

    def set_id(unit_type=None, unit=None, self=ax):
        self.unit_type = unit_type
        self.unit = unit
        self._id = (unit_type, unit)
    ax.set_id = set_id
    ax.set_id()

    def label(self=ax):
        # in the future, use labels instead of (type, unit) _id
        if self.unit_type:
            if self.unit:
                ylabel = '{} [{}]' .format(self.unit_type, self.unit)
            else:
                ylabel = self.unit_type
        else:
            if self.unit:
                ylabel = '[{}]' .format(self.unit)
            else:
                ylabel = ''
        return ylabel
    ax.label = label

    def set_par(offset=0.0, self=ax):
        self.yaxis.tick_right()
        self.yaxis.set_label_position('right')
        for spine in self.spines.values():
            spine.set_visible(False)
        self.spines['right'].set_visible(True)
        self.spines['right'].set_position(('axes', offset))
    ax.set_par = set_par

    def set_host(self=ax):
        self.yaxis.tick_left()
        self.yaxis.set_label_position('left')
        for spine in self.spines.values():
            spine.set_visible(True)
        self.spines['right'].set_position(('axes', 1.0))
    ax.set_host = set_host

    def get_data_extents(self=ax):
        try:
            ymin = min([min(line.get_ydata()) for line in self.lines])
```

					ПГУ1.090301.38.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ymax = max([max(line.get_ydata()) for line in self.lines])
        return ymin, ymax
    except ValueError:
        return 0, 0
ax.get_data_extents = get_data_extents

def patch_ygrid_proxy(self, ygp):
    ygp.parent = self
    ygp.patch.set_visible(False)
    ygp.set_frame_on(False)
    ygp.xaxis.set_visible(False)
    ygp.yaxis.set_visible(False)

def mock_gridlines(ax, which='both', linestyle=None, self=ygp):
    y0, y1 = ax.yaxis.get_view_interval()
    M_locs = np.clip(ax.yaxis.get_majorticklocs(), y0, y1)
    m_locs = np.clip(ax.yaxis.get_minorticklocs(), y0, y1)
    if which=='both':
        locs = list(M_locs) + list(m_locs)
    elif which=='major':
        locs = M_locs
    elif which=='minor':
        locs = m_locs
    if ax.log:
        z0, z1 = log10(min(locs)), log10(max(locs))
        normalized = [(log10(loc) - z0)/(z1 - z0) for loc in locs]
    else:
        normalized = [(loc - y0)/(y1 - y0) for loc in locs]
    for loc in set(normalized):
        ygp.axhline(loc,
                    color=plt.rcParams['grid.color'],
                    linestyle=linestyle,
                    linewidth=0.5)
ygp.mock_gridlines = mock_gridlines

```

Файл clarified_table.py

```
# -*- coding: utf-8 -*-
```

```

from PyQt5.QtWidgets import (QTableWidget, QTableWidgetItem,
                             QPushButton, QHeaderView)
from PyQt5.QtCore import Qt, QObject

from main_proj.classes.internal.dict_combo import DictCombo

class ClarifiedTable(QTableWidget):

    def __init__(self, parent):
        super().__init__()
        self.parent = parent
        ut = self.parent
        dm = ut.parent

        self.setColumnCount(2)
        self.setHorizontalHeaderLabels(['Parsed', 'Interpreted'])
        self.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.verticalHeader().hide()
        self.setRowCount(len(dm.unit_clarify)+1)
        self.clarify_cache = []

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

```

for row, key in enumerate(dm.unit_clarify):
    self.clarify_cache.append(key)
    self.setItem(row, 0, QTableWidgetItem(key))
    clarify_combo = DictCombo(dm.all_units,
                               None,
                               self.get_clarified_value)

    clarify_combo.row = row
    clarify_combo.populate()

clarify_combo.currentTextChanged.connect(self.update_clarify_value)
    self.setCellWidget(row, 1, clarify_combo)
    self.add_button = QPushButton('Add')
    self.add_button.clicked.connect(self.add_clarified_row)
    self.setSpan(row+1, 0, 1, 2)
    self.setCellWidget(row+1, 0, self.add_button)
    self.cellChanged.connect(self.update_clarify_key)

def combos(self):
    return [self.cellWidget(row, 1) for row in range(self.rowCount()-1)]

def get_clarified_value(self, row):
    ut = self.parent
    dm = ut.parent
    key = self.item(row, 0).text()
    try:
        return dm.unit_clarify[key]
    except KeyError:
        return ''

def update_clarify_key(self, row, col):
    ut = self.parent
    dm = ut.parent
    item = self.item(row, 0)
    key = item.text()
    if key and key not in self.clarify_cache:
        value = self.cellWidget(row, 1).currentText()
        if row < len(self.clarify_cache):
            # Rename existing entry
            old = self.clarify_cache[row]
            del dm.unit_clarify[old]
            self.clarify_cache[row] = key
        else:
            # Add new entry
            self.clarify_cache.append(key)
            dm.unit_clarify[key] = value
            dm.modified = True
    else:
        try:
            item.setText(self.clarify_cache[row])
        except IndexError:
            self.removeRow(row)
            self.clearSelection()

def update_clarify_value(self, value):
    ut = self.parent
    dm = ut.parent
    clarify_combo = QObject.sender(self)
    key = self.item(clarify_combo.row, 0).text()
    dm.unit_clarify[key] = value
    dm.modified = True

def add_clarified_row(self):

```

```

ut = self.parent
dm = ut.parent
row = self.rowCount()-1
self.insertRow(row)
item = QTableWidgetItem()
self.blockSignals(True)
self.setItem(row, 0, item)
clarify_combo = DictCombo(dm.all_units,
                           None,
                           self.get_clarified_value)

clarify_combo.row = row
clarify_combo.populate()
clarify_combo.currentTextChanged.connect(self.update_clarify_value)
self.clearSelection()
self.setCellWidget(row, 1, clarify_combo)
self.editItem(item)
self.blockSignals(False)

def delete_clarified_row(self):
    ut = self.parent
    dm = ut.parent
    all_rows = [item.row() for item in self.selectedIndexes()]
    for row in sorted(set(all_rows), reverse=True):
        key = self.item(row, 0).text()
        if key in dm.unit_clarify:
            del dm.unit_clarify[key]
        self.clarify_cache.pop(row)
        self.removeRow(row)
        dm.modified = True
    for row in range(self.rowCount()-1):
        clarify_combo = self.cellWidget(row, 1)
        clarify_combo.row = row

def closeEditor(self, editor, hint):
    self.commitData(editor)
    QTableWidgetItem.closeEditor(self, editor, hint)

def keyPressEvent(self, event):
    if event.key() == Qt.Key_Delete:
        ut = self.parent
        dm = ut.parent
        all_rows = [item.row() for item in self.selectedIndexes()]
        for row in sorted(set(all_rows), reverse=True):
            key = self.item(row, 0).text()
            if key in dm.unit_clarify:
                del dm.unit_clarify[key]
            self.clarify_cache.pop(row)
            self.removeRow(row)
            dm.modified = True
        for row in range(self.rowCount()-1):
            clarify_combo = self.cellWidget(row, 1)
            clarify_combo.row = row
    QTableWidgetItem.keyPressEvent(self, event)

```

Файл dict_combo.py

```
# -*- coding: utf-8 -*-
```

```
from PyQt5.QtWidgets import QComboBox
```

```
class DictCombo(QComboBox):
```

					ПГУ1.090301.38.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

```

def __init__(self, get_unit_dict, get_unit_type=None, get_unit=None):
    super().__init__()
    self.get_unit_dict = get_unit_dict
    self.get_unit = get_unit
    self.get_unit_type = get_unit_type

def populate_units():
    # Unit Combos
    reset = self.get_unit()
    self.clear()
    if self.get_unit_type() in self.get_unit_dict():
        self.addItem('')
        self.addItem(self.get_unit_dict()[self.get_unit_type()])
    self.setCurrentText(reset)

def populate_types():
    # Type Combos
    reset = self.get_unit_type()
    self.clear()
    self.addItem('')
    self.addItem(list(self.get_unit_dict().keys()))
    self.setCurrentText(reset)

def populate_all_units():
    # Clarify Combos
    reset = self.get_unit(self.row)
    self.clear()
    for units in self.get_unit_dict().values():
        self.addItem(units)
    self.setCurrentText(reset)

if self.get_unit_type is not None:
    if self.get_unit is not None:
        self.populate = populate_units
    else:
        self.populate = populate_types
else:
    self.populate = populate_all_units

```

Файл edit_series_dialog.py

```
# -*- coding: utf-8 -*-
```

```

from PyQt5.QtWidgets import (QDialog, QFormLayout, QHBoxLayout, QMessageBox,
                             QLabel, QLineEdit, QDoubleSpinBox, QPushButton)

```

```

from PyQt5.QtGui import QIcon
from PyQt5.QtCore import Qt

```

```
from main_proj.classes.internal.dict_combo import DictCombo
```

```
class EditSeriesDialog(QDialog):
```

```

    def __init__(self, parent, item):
        super().__init__()
        self.parent = parent
        ui = self.parent
        self.item = item
        group_name = item.parent().text(0)
        alias = item.text(0)
        group = ui.all_groups[group_name]
        header = group.get_header(alias)
        s = group.series(header)
        self.s = s
        self.alias = s.alias

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		60

```

self.unit_type = s.unit_type
self.unit = s.unit
self.scale = s.scale
self.setWindowTitle('Edit Series')
self.setWindowIcon(QIcon('rc/satellite.png'))

form = QFormLayout()
form.addRow('Group:', QLabel(s.group.name))
form.addRow('Header:', QLabel(s.header))
self.alias_edit = QLineEdit(s.alias)
self.alias_edit.setPlaceholderText(s.header)
self.alias_edit.textChanged.connect(self.update_alias)
form.addRow('Alias:', self.alias_edit)
self.type_combo = DictCombo(ui.all_units, self.get_type)
self.type_combo.populate()
self.type_combo.currentTextChanged.connect(self.type_combo_slot)
form.addRow('Unit Type:', self.type_combo)
self.unit_combo = DictCombo(ui.all_units, self.get_type,
self.get_unit)
self.unit_combo.populate()
self.unit_combo.currentTextChanged.connect(self.unit_combo_slot)
form.addRow('Unit:', self.unit_combo)
self.scale_spinbox = QDoubleSpinBox()
self.scale_spinbox.setRange(0, 1000)
self.scale_spinbox.setValue(s.scale)
self.scale_spinbox.valueChanged.connect(self.update_scale)
form.addRow('Scale:', self.scale_spinbox)
form.addRow(QLabel(self.stats()))
hbox = QHBoxLayout()
self.accept_button = QPushButton('Save & Exit')
self.accept_button.clicked.connect(self.save_exit)
hbox.addWidget(self.accept_button)
self.reject_button = QPushButton('Cancel')
self.reject_button.clicked.connect(self.close)
hbox.addWidget(self.reject_button)
form.addRow(hbox)
self.setLayout(form)
self.setFixedSize(self.sizeHint())

def update_alias(self, text):
    s = self.s
    alias = text.strip()
    if alias in s.group.alias_dict or alias in s.group.data.columns:
        self.alias_edit.blockSignals(True)
        self.alias_edit.setText(s.alias)
        self.alias_edit.blockSignals(False)
    else:
        self.alias = alias

def get_type(self):
    return self.unit_type

def get_unit(self):
    return self.unit

def type_combo_slot(self, text):
    self.unit_type = text
    self.unit_combo.populate()

def unit_combo_slot(self, text):
    self.unit = text

```

```

def update_scale(self, value):
    self.scale = value

def stats(self):
    series = self.s.group.data[self.s.header]
    start = series.index[0].strftime('%Y-%m-%d %H:%M:%S')
    end = series.index[-1].strftime('%Y-%m-%d %H:%M:%S')
    idmin = series.idxmin().strftime('%Y-%m-%d %H:%M:%S')
    idmax = series.idxmax().strftime('%Y-%m-%d %H:%M:%S')
    stats = 'Series Statistics:\n'
    stats += '    Count:\t {}\n'.format(series.size)
    stats += '    Start:\t {}\n'.format(start)
    stats += '    End:\t {}\n'.format(end)
    stats += '    Min:\t {}\n'.format(series.min())
    stats += '    @\t {}\n'.format(idmin)
    stats += '    Max:\t {}\n'.format(series.max())
    stats += '    @\t {}\n'.format(idmax)
#    stats += '    Mean:\t{}\n'.format(series.mean())
#    stats += '    Median:\t{}\n'.format(series.median())
    return stats

def apply_changes(self):
    s = self.s
    if s.alias: del s.group.alias_dict[s.alias]
    s.alias = self.alias
    s.unit_type = self.unit_type
    s.unit = self.unit
    s.scale = self.scale
    self.item.setText(0, s.label)
    if s.alias: s.group.alias_dict[s.alias] = s.header

def save_exit(self):
    ui = self.parent
    sd = ui.series_display
    cf = ui.get_current_figure()
    s = self.s

    if self.item.treeWidget() is sd.available:
        labels = cf.available_data[s.group.name]
        i = labels.index(s.label)
        self.apply_changes()
        labels[i] = s.label
    elif self.item.treeWidget() is sd.plotted:
        sp = cf.current_sps[0]
        if s.unit_type != self.unit_type or s.unit != self.unit:
            sp.remove({s.group.name: [s.alias]})
        else:
            for ax in sp.axes:
                try:
                    ax.contents.remove({s.group.name: [s.alias]})
                except ValueError:
                    pass
            self.apply_changes()
            sp.add({s.group.name: [s.label]})
    cf.replot()
    cf.update_gridspec()
    cf.draw()
    self.close()

def closeEvent(self, event):
    ui = self.parent
    s = self.s

```

```

c1 = self.alias != s.alias
c2 = self.unit_type != s.unit_type
c3 = self.unit != s.unit
c4 = self.scale != s.scale
if any([c1, c2, c3, c4]):
    result = ui.popup('Discard changes?')
    if result == QMessageBox.Cancel:
        event.ignore()
    elif result == QMessageBox.Save:
        self.save_exit()
    else:
        event.accept()
else:
    event.accept()

def keyPressEvent(self, event):
    """Enables dialog closure by escape key."""
    if event.key() == Qt.Key_Escape:
        self.close()

```

Файл `editable_tab_bar.py`

```
# -*- coding: utf-8 -*-
```

```
import re
```

```
from PyQt5.QtWidgets import QTabBar, QLineEdit
from PyQt5.QtCore import Qt, QPoint, QEvent
```

```

class EditableTabBar(QTabBar):
    def __init__(self, parent):
        QTabBar.__init__(self, parent)
        self._editor = QLineEdit(self)
        self._editor.setWindowFlags(Qt.Popup)
        self._editor.setFocusProxy(self)
        self._editor.editingFinished.connect(self.handleEditingFinished)
        self._editor.installEventFilter(self)

    def eventFilter(self, widget, event):
        if ((event.type() == QEvent.MouseButtonPress and
            not self._editor.geometry().contains(event.globalPos())) or
            (event.type() == QEvent.KeyPress and
            event.key() == Qt.Key_Escape)):
            self._editor.hide()
            return True
        return QTabBar.eventFilter(self, widget, event)

    def mouseDoubleClickEvent(self, event):
        index = self.tabAt(event.pos())
        ui = self.parent().parent()
        if index >= 0 and index != ui.nfigs():
            self.editTab(index)

    def editTab(self, index):
        rect = self.tabRect(index)
        self._editor.setFixedSize(rect.size())
        adjust = self.parent().geometry().height()
        point = QPoint(rect.x(), adjust-rect.height())
        self._editor.move(self.parent().mapToGlobal(point))
        self._editor.setText(self.tabText(index))
        if not self._editor.isVisible():
            self._editor.show()

    def handleEditingFinished(self):

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

```

ui = self.parent().parent()
index = self.currentIndex()
if index >= 0 and index != ui.nfigs():
    cf = self.parent().widget(index)
    fig_title = re.sub('[\\|\\|\\.]', '|', self._editor.text())
    if (fig_title not in ui.open_figure_titles() or
        fig_title == cf.title):
        self._editor.hide()
        self.setTabText(index, self._editor.text())
        cf.fig.suptitle.set_text(fig_title)
        cf.title = fig_title
        cf.draw()
        cf.saved = False

```

Файл file_sort_filter_proxy_model.py

```

# -*- coding: utf-8 -*-
import re
import datetime as dt

from PyQt5.QtCore import QSortFilterProxyModel

class FileSortFilterProxyModel(QSortFilterProxyModel):

    def __init__(self, parent):
        QSortFilterProxyModel.__init__(self, parent)

    def file_size_to_bytes(self, size):
        num, suffix = re.split(' ', size)
        num = re.sub(',', '', num)
        n = self.parent().suffixes.index(suffix)
        return int(num)*1000**n

    def lessThan(self, source_left, source_right):
        left = self.sourceModel().data(source_left)
        right = self.sourceModel().data(source_right)
        if source_left.column() == 1:
            left_size = self.file_size_to_bytes(left)
            right_size = self.file_size_to_bytes(right)
            return left_size < right_size
        elif source_left.column() == 2:
            left_date = dt.datetime.strptime(left, self.parent().date_format)
            right_date = dt.datetime.strptime(right,
self.parent().date_format)
            return left_date < right_date
        else:
            return left < right

```

Файл group.py

```

# -*- coding: utf-8 -*-

from typing import Union, List, Optional, Callable
import pandas as pd

from main_proj_v2.classes.internal.series import Series

class Group:

    def __init__(self, df: pd.DataFrame, name: str):
        self.name = name
        self.data = df
        self.series_dict: dict = {
            header: Series(self, header, '')

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

        for header in self.data.columns
    }
    self.alias_dict: dict = {} # alias -> header mapping

    def series(self, which: Optional[Union[str, Callable, List[str]]] = None)
-> Union[List[Series], Series]:
    if which is None:
        return list(self.series_dict.values())
    elif isinstance(which, str):
        # Try header first, then alias
        if which in self.series_dict:
            return self.series_dict[which]
        elif which in self.alias_dict:
            header = self.alias_dict[which]
            return self.series_dict[header]
        else:
            raise KeyError(f'"{which}" is not a valid series reference')
    elif isinstance(which, list):
        result = []
        for w in which:
            if w in self.series_dict:
                result.append(self.series_dict[w])
            elif w in self.alias_dict:
                result.append(self.series_dict[self.alias_dict[w]])
        return result
    elif callable(which):
        return [s for s in self.series_dict.values() if which(s)]
    else:
        raise TypeError(f'Invalid series reference type: {type(which)}')

    def get_header(self, alias: str) -> str:
        """Returns original header for an alias."""
        return self.alias_dict.get(alias, alias)

    def summarize(self) -> None:
        """Print summary information."""
        print('Headers')
        for series in self.series():
            print(f'\t{series.header}')
            for line in series.summarize():
                print(f'\t\t{line}')
        print('\nAssigned Aliases')
        print(f'\t{list(self.alias_dict.keys())}')

```

Файл item_edit.py

```

# -*- coding: utf-8 -*-
"""item_edit.py - Contains ItemEdit class definition."""

from PyQt5.QtWidgets import QLineEdit

class ItemEdit(QLineEdit):

    def __init__(self, text, ct):
        super().__init__(text)
        self.returnPressed.connect(self.leave)

    def focusInEvent(self, event):
        self.selectAll()
        QLineEdit.focusInEvent(self, event)

    def focusOutEvent(self, event):
        self.home(False)
        QLineEdit.focusOutEvent(self, event)

```

```
def mouseDoubleClickEvent(self, event):
    QLineEdit.mouseDoubleClickEvent(self, event)
    self.selectAll()
```

```
def leave(self):
    self.deselect()
    self.clearFocus()
```

Файл pandas_model.py

```
# -*- coding: utf-8 -*-
"""pandas_model.py - Contains PandasModel class definition."""

from PyQt5.QtGui import QStandardItemModel, QStandardItem
from PyQt5.QtCore import Qt

class PandasModel(QStandardItemModel):

    def __init__(self, data, parent=None):
        QStandardItemModel.__init__(self, parent)
        self._data = data
        for row in data.values.tolist():
            data_row = []
            for x in row:
                try:
                    data_row.append(QStandardItem("{:.3f}".format(x)))
                except ValueError:
                    data_row.append(QStandardItem(x))
            self.appendRow(data_row)
        return

    def rowCount(self, parent=None):
        return len(self._data.values)

    def columnCount(self, parent=None):
        return self._data.columns.size

    def headerData(self, x, orientation, role):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            return self._data.columns[x]
        if orientation == Qt.Vertical and role == Qt.DisplayRole:
            return self._data.index[x]
        return None
```

Файл q_color_button.py

```
# -*- coding: utf-8 -*-

from PyQt5.QtWidgets import QPushButton

class QColorButton(QPushButton):
    """Associates unit type with color."""

    def __init__(self, parent, color, unit_type):
        super(QColorButton, self).__init__()
        self.setFixedSize(20,20)
        self.parent = parent
        self.color = color
        self.setStyleSheet("background-color: {};".format(color))
        self.unit_type = unit_type
```

Файл series.py

```
# -*- coding: utf-8 -*-
"""series.py - Contains Series class definition."""
```

```

from typing import Optional, TYPE_CHECKING

if TYPE_CHECKING:
    from main_proj.classes.internal.group import Group

class Series:
    """Represents a time series data series."""

    def __init__(self, group: 'Group', header: str, alias: str = '',
                  unit_type: Optional[str] = None, unit: Optional[str] =
None):
        self.group = group
        self.header = header
        self.alias = alias
        self.unit_type = unit_type or ''
        self.unit = unit or ''
        self.scale: float = 1.0
        self.keep: bool = True
        self.color: Optional[str] = None

    @property
    def label(self) -> str:
        return self.alias if self.alias else self.header

    def get_unit_type(self) -> str:
        return self.unit_type

    def get_unit(self) -> str:
        return self.unit

    def summarize(self) -> tuple:
        return (
            f'Header: {self.header}',
            f'Alias: {self.alias}',
            f'Unit Type: {self.unit_type}',
            f'Unit: {self.unit}',
            f'Scale: {self.scale}',
            f'Keep: {self.keep}'
        )

    def __repr__(self) -> str:
        return f"Series('{self.label}', unit='{self.unit}')"

```

Файл series_header_stack.py

```
# -*- coding: utf-8 -*-
```

```

from PyQt5.QtWidgets import QWidget, QLabel, QCheckBox, QVBoxLayout
from PyQt5.QtCore import Qt, QObject

```

```

from main_proj.classes.internal.item_edit import ItemEdit
from main_proj.classes.internal.dict_combo import DictCombo

```

```
class SeriesHeaderStack(QWidget):
```

```

    def __init__(self, parent, i, s):
        super().__init__()
        self.parent = parent
        ct = self.parent
        dm = ct.parent
        self.col = i
        self.s = s

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

```

self.keep_check = QCheckBox()
self.keep_check.stack = self
self.keep_check.row = 0
self.keep_check.setChecked(s.keep)
self.keep_check.stateChanged.connect(self.update_keep)
# self.patch_switch_focus(self.keep_check)
self.keep_widget = QWidget()
self.keep_widget.row = 0
vbox = QVBoxLayout(self.keep_widget)
vbox.setAlignment(Qt.AlignCenter)
vbox.setContentsMargins(0, 0, 0, 0)
vbox.addWidget(self.keep_check)

self.header_item = QLabel(s.header)
self.header_item.setStyleSheet('QLabel {padding: 0px 3px 0px 3px;}')
self.header_item.stack = self
self.header_item.row = 1
self.patch_switch_focus(self.header_item)

self.alias_item = ItemEdit(s.alias, parent)
self.alias_item.stack = self
self.alias_item.row = 2
self.alias_item.home(False)
self.alias_item.setPlaceholderText(s.header)
self.alias_item.editingFinished.connect(self.update_alias)
self.patch_switch_focus(self.alias_item)

self.type_combo = DictCombo(dm.all_units, s.get_unit_type)
self.type_combo.stack = self
self.type_combo.row = 3
self.type_combo.populate()
self.type_combo.currentTextChanged.connect(self.type_combo_slot)

self.patch_switch_focus(self.type_combo)

self.unit_combo = DictCombo(dm.all_units, s.get_unit_type,
s.get_unit)
self.unit_combo.stack = self
self.unit_combo.row = 4
self.unit_combo.populate()
self.unit_combo.currentTextChanged.connect(self.unit_combo_slot)
self.patch_switch_focus(self.unit_combo)

def patch_switch_focus(self, w):
def appendFocusInEvent(event, self=w):
    type(w).focusInEvent(w, event)
    ct = w.stack.parent
    ct.header_table.setCurrentCell(w.row, w.stack.col)
w.focusInEvent = appendFocusInEvent

def update_keep(self, checked):
    ct = self.parent
    dm = ct.parent
    keep_check = QObject.sender(self)
    selected = ct.header_table.selectedIndexes()
    stacks = set([keep_check.stack] + [ct.stacks[i.column()] for i in
selected])
    for stack in stacks:
        stack.s.keep = checked
    if ct.hide_unused.isChecked():
        ct.display_header_info()
    else:

```

```

        for stack in stacks:
            stack.keep_check.blockSignals(True)
            stack.keep_check.setChecked(checkered)
            stack.keep_check.blockSignals(False)
        dm.modified = True

def type_combo_slot(self, text):
    ct = self.parent
    dm = ct.parent
    stack = QObject.sender(self).stack
    stack.s.unit_type = text
    stack.unit_combo.populate()
    dm.modified = True

def unit_combo_slot(self, text):
    ct = self.parent
    dm = ct.parent
    stack = QObject.sender(self).stack
    stack.s.unit = text
    dm.modified = True

def update_alias(self):
    ct = self.parent
    dm = ct.parent
    item = QObject.sender(self)
    s = item.stack.s
    group = s.group
    alias = item.text().strip()

    def remove_key_by_value(dictionary, value):
        for key in dictionary:
            if dictionary[key] == value:
                del dictionary[key]
                break

    accept = True
    if alias != s.alias and alias in group.alias_dict:
        dm.feedback("{} is already in use."
                    'Please choose a different alias.'
                    .format(alias))
        accept = False
    if alias in group.data.columns:
        dm.feedback("{} is the name of an original header."
                    'Please choose a different alias.'
                    .format(alias))
        accept = False
    if accept:
        s.alias = alias
        remove_key_by_value(group.alias_dict, s.header)
        if alias:
            group.alias_dict[alias] = s.header
        dm.modified = True
    else:
        ct.header_table.blockSignals(True)
        item.setText(s.alias)
        ct.header_table.blockSignals(False)

```

Файл subplot_manager.py

```
# -*- coding: utf-8 -*-
```

```

import datetime as dt
from copy import copy, deepcopy
import numpy as np

```

					ПГУ1.090301.38.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

import matplotlib.pyplot as plt
from main_proj.classes.internal.contents_dict import ContentsDict
from main_proj.classes.internal.axes_patch_functions import patch_ax,
patch_ygrid_proxy

class SubplotManager():

    def __init__(self, parent, host, index):
        self.parent = parent
        cf = self.parent
        patch_ax(self, host)
        self.axes = [host]

        self.index = index
        self.color_coord = False
        self.legend_on = False
        self.legend = None
        self.location = 'Outside Right'
        self.ncols = 1
        self.legend_units = True
        self.lines = []
        self.labels = []

        self.ygrid_proxy = cf.fig.add_subplot(cf.nplots(), 1, self.index + 1)
        self.ygrid_proxy.set_zorder(-1)
        patch_ygrid_proxy(self, self.ygrid_proxy)

    def host(self):
        return self.axes[0]

    @property
    def contents(self):
        contents = ContentsDict()
        for ax in self.axes:
            contents.add(deepcopy(ax.contents))
        return contents

    def get_axes(self, unit_type, unit):
        for ax in self.axes:
            if hasattr(ax, '_id') and ax._id == (unit_type, unit):
                return ax
        if len(self.axes) == 1 and not self.host().contents:
            ax = self.host()
            ax.set_id(unit_type, unit)
            return ax
        return None

    def add(self, contents, cf=None):
        if cf is None:
            cf = self.parent
        ui = cf.parent
        for group_name in contents:
            group = ui.all_groups[group_name]
            for alias in contents[group_name]:
                header = group.get_header(alias)
                s = group.series(header)
                ax = self.get_axes(s.unit_type, s.unit)
                if ax is None:
                    par = cf.fig.add_subplot(cf.nplots(), 1, self.index + 1)
                    patch_ax(self, par)
                    par.set_id(s.unit_type, s.unit)

```

					ПГУ1.090301.38.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        self.axes.append(par)
        ax = par
        ax.contents.add({group_name: [alias]})

def remove(self, contents, cf=None):
    if cf is None:
        cf = self.parent
    ui = cf.parent
    for group_name in contents:
        group = ui.all_groups[group_name]
        for alias in contents[group_name]:
            header = group.get_header(alias)
            s = group.series(header)
            ax = self.get_axes(s.unit_type, s.unit)
            if ax:
                ax.contents.remove({group_name: [alias]})

# Remove empty axes
for ax in [ax for ax in self.axes if not ax.contents]:
    ax.remove()
    self.axes.remove(ax)

# Ensure at least one axis exists
if not self.axes:
    ax = cf.fig.add_subplot(cf.nplots(), 1, self.index + 1)
    patch_ax(self, ax)
    ax.set_id()
    self.axes = [ax]
    if self in cf.current_sps:
        plt.setp(ax.spines.values(),
                  linewidth=ui.highlight[True])

def plot(self, skeleton=False):
    cf = self.parent
    ui = cf.parent

    color_index = 0
    style_dict = {}
    self.lines = []
    self.labels = []

    for ax in self.axes:
        ax.clear()
        ax.patch.set_visible(False)

    for group_name in list(ax.contents.keys()):
        group = ui.all_groups[group_name]
        df = group.data
        subdf = df[(df.index >= cf.start) & (df.index <= cf.end)]

        for alias in ax.contents[group_name]:
            header = group.get_header(copy(alias))
            s = group.series(header)
            data = subdf[header]
            n = len(data.index)
            d = cf.density / 100

            if n > 0 and d < 1:
                thin = np.linspace(0, n-1, num=max(1, int(n*d))),
dtype=int)
                data = data.iloc[thin]

```

```

else x)

        data = data.map(lambda x: x * s.scale if x is not None

if self.color_coord:
    if s.unit_type not in style_dict:
        style_dict[s.unit_type] = 0
    counter = style_dict[s.unit_type]
    style = ui.markers[counter % len(ui.markers)]
    style_dict[s.unit_type] += 1
    if s.unit_type:
        color = ui.color_dict.get(s.unit_type,
ui.current_rcs['axes.labelcolor'])
    else:
        color = ui.current_rcs['axes.labelcolor']
        labelcolor = color
    else:
        style = 'o'
        color = f'C{color_index % 10}'
        color_index += 1
        labelcolor = ui.current_rcs['axes.labelcolor']

if cf.scatter:
    line, = ax.plot(data, style, color=color,
                    markersize=cf.dot_size,
                    linestyle='None')
else:
    line, = ax.plot(data, color=color, linewidth=1.5)

self.lines.append(line)
self.labels.append((s.label, s.unit))

if ax.contents:
    ax.set_ylabel(ax.label(),
                  fontsize=cf.label_size,
                  color=labelcolor)
    ax.tick_params(axis='y', labelcolor=labelcolor)
    ax.relim()

seconds = (cf.end - cf.start).total_seconds() * cf.x_margin
offset = dt.timedelta(seconds=seconds)
ax.set_xlim([cf.start - offset, cf.end + offset])

if ax.auto_limits:
    ax.autoscale(axis='y')
elif hasattr(ax, 'custom_limits') and ax.custom_limits:
    ax.set_ylim(ax.custom_limits)

if ax.log:
    ax.set_yscale('log')
else:
    ax.set_yscale('linear')

self.arrange_axes()
cf.format_axes()
cf.update_toolbars()

def arrange_axes(self):
    cf = self.parent
    self.host().set_host()
    self.host().set_zorder(0)
    offset = cf.axis_offset
    for i, ax in enumerate(self.axes[1:]):

```

					ПГУ1.090301.38.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ax.set_par(offset=1 + offset * (i))
        ax.set_zorder(i + 1)

def show_legend(self):
    cf = self.parent
    ui = cf.parent
    if self.legend is not None:
        self.legend.remove()
    if self.legend_on and self.contents:
        if self.location == 'Outside Right':
            offset = cf.axis_offset
            npars = len(self.axes[1:])
            bbox = (1 + offset * npars, 0.5)
        elif self.location == 'Outside Top':
            bbox = (0.5, 1)
        else:
            bbox = (0, 0, 1, 1)

        if self.legend_units:
            labels = ['{} {}'.format(*x) for x in self.labels if x[1]]
        else:
            labels = [x[0] for x in self.labels]

        if labels:
            self.legend = self.axes[-1].legend(self.lines, labels,
                                                loc=ui.locations.get(self.location, 'best'),
                                                bbox_to_anchor=bbox,
                                                ncol=self.ncols,
                                                framealpha=1.0)
            self.legend.set_zorder(len(self.labels) + 1)
            for handle in self.legend.legend_handles:
                if hasattr(handle, 'set_markersize'):
                    handle.set_markersize(4)
        else:
            self.legend = None

```

Файл unit_model.py

```

# -*- coding: utf-8 -*-
"""unit_model.py - Contains UnitModel class definition."""

from PyQt5.QtGui import QStandardItemModel

class UnitModel(QStandardItemModel):

    def __init__(self, parent, label):
        super().__init__()
        self.parent = parent
        self.tree = None
        self.setHorizontalHeaderLabels([label])
        self.itemChanged.connect(self.edit_handler)

    def edit_handler(self, item):
        if item.parent():
            self.edit_unit(item)
        else:
            self.edit_unit_type(item)

    def edit_unit(self, item):
        ut = self.parent
        dm = ut.parent
        i = item.row()
        unit_type = item.parent().text()
        unit = item.text().strip()

```

```

        if unit and unit not in self.tree.unit_dict[unit_type]:
            try:
                # Rename existing unit
                old = self.tree.unit_dict[unit_type][i]
                self.tree.unit_dict[unit_type][i] = unit
                for combo in dm.unit_combos():
                    idx = combo.findText(old)
                    combo.setItemText(idx, unit)
            except IndexError:
                # Add new unit
                self.tree.unit_dict[unit_type].append(unit)
                for combo in dm.unit_combos():
                    combo.populate()
                dm.modified = True
        else:
            try:
                item.setText(self.tree.unit_dict[unit_type][i])
            except IndexError:
                self.removeRow(i, item.parent().index())

def edit_unit_type(self, item):
    ut = self.parent
    dm = ut.parent
    i = item.row()
    unit_type = item.text().strip()
    if unit_type and unit_type not in dm.all_units():
        if i < len(self.tree.type_cache):
            # Rename existing unit type
            old = self.tree.type_cache[i]
            self.tree.unit_dict[unit_type] = self.tree.unit_dict[old]
            del self.tree.unit_dict[old]
            self.tree.type_cache[i] = unit_type
            for combo in dm.type_combos():
                idx = combo.findText(old)
                combo.setItemText(idx, unit_type)
        else:
            # Add new unit type
            self.tree.type_cache.append(unit_type)
            self.tree.unit_dict[unit_type] = []
            for combo in dm.type_combos():
                combo.insertItem(i+1, unit_type)
            dm.modified = True
    else:
        try:
            item.setText(self.tree.type_cache[i])
        except IndexError:
            self.removeRow(i, item.index().parent())

```

Файл unit_tree.py

```

# -*- coding: utf-8 -*-
"""unit_tree.py - Contains UnitTree class definition."""

```

```

from PyQt5.QtWidgets import QTreeView
from PyQt5.QtGui import QStandardItem, QIcon
from PyQt5.QtCore import Qt

```

```

class UnitTree(QTreeView):

```

```

    def __init__(self, parent, model, unit_dict,
                 L0_edit=True, L1_edit=True, L0_add=True, L1_add=True):
        super().__init__()
        self.parent = parent
        self.doubleClicked.connect(self.add)

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

self.setModel(model)
model.tree = self
self.unit_dict = unit_dict
self.type_cache = []
for unit_type in self.unit_dict:
    self.type_cache.append(unit_type)
    L0 = QStandardItem(unit_type)
    L0.setEditable(L0_edit)
    for unit in self.unit_dict[unit_type]:
        L1 = QStandardItem(unit)
        L1.setEditable(L1_edit)
        L0.appendRow([L1])
    if L1_add: L0.appendRow([self.plus()])
    self.model().appendRow(L0)
if L0_add: self.model().appendRow([self.plus()])

def plus(self):
    ut = self.parent
    dm = ut.parent
    item = QStandardItem()
    item.setIcon(QIcon(dm.parent.current_icon_path+'/plus.png'))
    return item

def add(self, index):
    row = index.row()
    item = self.model().itemFromIndex(index)
    if item.data(1) is not None:
        if item.parent():
            new_item = QStandardItem()
            item.parent().insertRow(row, new_item)
        else:
            new_item = QStandardItem()
            new_item.appendRow([self.plus()])
            self.model().insertRow(row, new_item)
    self.scrollTo(new_item.index())
    self.edit(new_item.index())

def keyPressEvent(self, event):
    if event.key() == Qt.Key_Delete:
        if self.selectedIndexes():
            index = self.selectedIndexes()[0]
            item = self.model().itemFromIndex(index)
            if item.data(1) is None:
                if item.parent():
                    self.delete_unit(item)
                else:
                    self.delete_unit_type(item)
            else:
                QTreeView.keyPressEvent(self, event)

def delete_unit(self, item):
    ut = self.parent
    dm = ut.parent
    level0 = item.parent()
    i = item.row()
    unit = item.text()
    unit_type = level0.text()
    self.unit_dict[unit_type].remove(unit)
    self.model().removeRow(i, level0.index())
    for combo in dm.unit_combos():
        combo.populate()
    dm.modified = True

```

```

def delete_unit_type(self, item):
    ut = self.parent
    dm = ut.parent
    i = item.row()
    unit_type = item.text()
    self.type_cache.remove(unit_type)
    del self.unit_dict[unit_type]
    self.model().removeRow(i, item.index().parent())
    for combo in dm.type_combos():
        combo.removeItem(i+1)
        combo.setCurrentText('')
    dm.modified = True

```

Файл axes_frame.py

```

# -*- coding: utf-8 -*-
"""axes_frame.py - Contains AxesFrame class definition."""

#from dateutil.rrule import DAILY
import datetime as dt
from matplotlib.dates import HourLocator, MinuteLocator, DateFormatter
import matplotlib.pyplot as plt
from matplotlib.ticker import NullLocator, AutoMinorLocator
from matplotlib.gridspec import GridSpec
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg

from PyQt5.QtWidgets import QApplication
from PyQt5.QtCore import Qt

from main_proj.classes.internal.subplot_manager import SubplotManager
from main_proj.classes.internal.contents_dict import ContentsDict

class AxesFrame(FigureCanvasQTAgg):

    def patch_figure(self, fig):
        def get(key, self=fig):
            return None
        fig._axstack.get = get

    def _draw(self, event):
        self.draw()

    def __init__(self, parent):
        self.fig = plt.figure()
        self.patch_figure(self.fig)
        super().__init__(self.fig)
        self.transfer_contents = ContentsDict()
        self.transfer_sp = None
        self.setAcceptDrops(True)

        self.saved = True
        self.never_saved = True
        self.parent = parent
        ui = self.parent
        for k,v in ui.af_init.items():
            setattr(self, k, v)

        self.gs = GridSpec(1, 1,
                           left=self.left_pad,
                           right=1-self.right_pad,
                           bottom=self.lower_pad,
                           top=1-self.upper_pad,

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

```

        hspace=self.spacing,
        height_ratios=[1])
    ax0 = self.fig.add_subplot(self.gs[0])
    sp = SubplotManager(self, ax0, 0)
    sp.ygrid_proxy.set_position(self.gs[0].get_position(self.fig))
    self.subplots = [sp]
    self.current_sps = []
    self.groups = []
    self.available_data = ContentsDict()

    self.fig.canvas.mpl_connect('button_press_event',
self.select_subplot)
    self.fig.canvas.mpl_connect('button_press_event', self._draw)
    self.fig.suptitle(self.title,
        fontsize=self.title_size)

    self.start = dt.datetime.strptime('2000-01-01 00:00:00',
        '%Y-%m-%d %H:%M:%S')
    self.end = dt.datetime.now()
    self.draw()

def weights(self):
    return self.gs.get_height_ratios()

def nplots(self):
    return self.gs.get_geometry()[0]

def update_gridspec(self, nplots=None, weights=None):
    if nplots is None:
        nplots = self.nplots()
    if weights is None:
        weights = self.weights()
    n = max([len(sp.axes[2:]) for sp in self.subplots])
    # If any subplot has a shown legend, treat like another secondary
axis
    for sp in self.subplots:
        c1 = sp.legend_on
        c2 = sp.contents
        c3 = sp.location=='Outside Right'
        if c1 and c2 and c3:
            n += 1
            break
    left = self.left_pad
    right = 1 - self.right_pad - self.axis_offset*n
    bottom = self.lower_pad
    top = 1 - self.upper_pad
    self.gs = GridSpec(nplots, 1, height_ratios=weights,
        left=left, right=right, bottom=bottom, top=top,
        hspace=self.spacing)
    for i, sp in enumerate(self.subplots):
        sp.index = i
        for ax in sp.axes:
            ax.set_position(self.gs[i].get_position(self.fig))
        sp.ygrid_proxy.set_position(self.gs[i].get_position(self.fig))
    self.saved = False

def replot(self, data=True, legend=True):
    for sp in self.subplots:
        if data:
            sp.plot()
        if legend:
            sp.show_legend()

```

```

self.saved = False

# def create_rrule(self, num_major_ticks, num_minor_ticks):
#     period = 24*60//num_major_ticks//num_minor_ticks
#     div = num_minor_ticks*60
#     byminute = sorted(set([(i*period%div)%60 for i in range(60)]))
#     byhour = list(range(24))
#     c = 0
#     bysetpos = []
#     for hour in byhour:
#         for minute in byminute:
#             if (hour*60 + minute)%period == 0:
#                 bysetpos.append(c+1)
#                 c += 1
#     return rrulewrapper(DAILY, bysetpos=bysetpos, byhour=byhour,
#                         byminute=byminute, bysecond=0)

def create_mlocator(self, m_T):
    hours = m_T//60
    minutes = m_T - 60*hours
    if hours:
        mlocator = HourLocator(byhour=range(0, 24, hours))
    else:
        mlocator = MinuteLocator(byminute=range(0, 60, minutes))
    mlocator.MAXTICKS = 10000
    return mlocator

def verify_ticks(self, M_T, m_T):
    # print('verify_ticks')
    try:
        if any([sp.contents for sp in self.subplots]):
            days = (self.end - self.start).days
            if self.mx:
                self.create_mlocator(m_T)
                ticks_per_day = 24/M_T + 1440/m_T
            else:
                ticks_per_day = 24/M_T
            if days*ticks_per_day > 500:
                raise ValueError
    except ValueError:
        return False
    return True

def format_axes(self):
    linestyle = ['-', '--', ':', '-.']
    self.fig.suptitle.set_fontsize(self.title_size)
    Mlocator = HourLocator(byhour=range(0, 24, self.M_T))
    mlocator = self.create_mlocator(self.m_T)

    for sp in self.subplots:
        if sp.ygrid_proxy.get_figure() is not None:
            sp.ygrid_proxy.clear()
            sp.ygrid_proxy.set_ymargin(0)
        for i, ax in enumerate(sp.axes):
            ls = linestyle[i%len(linestyle)]
            ax.tick_params(axis='x', which='both',
                          labelbottom=False, bottom=False)
            ax.yaxis.label.set_size(self.label_size)
            ax.tick_params(axis='y', labelsize=self.tick_size)

            if self.MY and not self.my:
                sp.ygrid_proxy.mock_gridlines(ax, which='major',

```

```

                                linestyle=ls)

    if self.my:
        if ax.log:
            ax.minorticks_on()
        else:
            ax.yaxis.set_minor_locator(AutoMinorLocator())
            ax.tick_params(axis='y', which='minor',
                           labelleft=False, labelright=False)
            sp.ygrid_proxy.mock_gridlines(ax, which='both',
                                           linestyle=ls)

    else:
        ax.yaxis.set_minor_locator(NullLocator())
    if sp.contents:
        if sp.index == len(self.subplots)-1:
            sp.host().tick_params(axis='x', which='both',
                                   labelbottom=True, bottom=True)
            for tick in sp.host().xaxis.get_ticklabels():
                tick.set_size(self.tick_size)
                tick.set_rotation(self.tick_rot)
                tick.set_horizontalalignment('right')
            sp.host().xaxis_date()
            sp.host().xaxis.set_major_locator(Mlocator)
            if self.mx:
                sp.host().xaxis.set_minor_locator(mlocator)
            else:
                sp.host().xaxis.set_minor_locator(NullLocator())
            sp.host().xaxis.set_major_formatter(
                DateFormatter(self.tsf))
            sp.host().xaxis.grid(which='major', visible=(self.MX or self.mx))
            sp.host().xaxis.grid(which='minor', visible=self.mx)
            self.saved = False

def get_subplot(self, event):
    for sp in self.subplots:
        if event.inaxes in sp.axes + [sp.ygrid_proxy]:
            return sp
    return None

def get_subplot_drag(self, event):
    x, y = event.pos().x(), event.pos().y()
    bbox = self.fig.get_window_extent().transformed(
        self.fig.dpi_scale_trans.inverted())
    height = bbox.height*self.fig.dpi
    for sp in self.subplots:
        bbox = sp.host().get_window_extent().transformed(
            self.fig.dpi_scale_trans.inverted())
        xmin = bbox.xmin*self.fig.dpi
        ymin = height-bbox.ymin*self.fig.dpi
        xmax = bbox.xmax*self.fig.dpi
        ymax = height-bbox.ymin*self.fig.dpi
        if xmin <= x <= xmax and ymin <= y <= ymax:
            return sp
    return None

def dragEnterEvent(self, event):
    ui = self.parent
    sd = ui.series_display
    if QApplication.focusWidget() is sd.plotted:
        selected = sd.plotted.selectedItems()
        self.transfer_contents = sd.tree_items_to_contents(selected)
        self.transfer_sp = self.current_sps[0]
        self.select_subplot(None, force_select=[])

```

```

self.draw()
self.dragMoveEvent(event)

def dragMoveEvent(self, event):
    sp = self.get_subplot_drag(event)
    if sp is not None:
        self.select_subplot(None, force_select=[sp])
    else:
        self.select_subplot(None, force_select=[])
    self.draw()
    event.accept()

def dragLeaveEvent(self, event):
    self.select_subplot(None, force_select=[])
    self.draw()
    event.accept()

def dropEvent(self, event):
    ui = self.parent
    sd = ui.series_display
    if self.current_sps:
        if QApplication.focusWidget() is sd.available:
            sd.add_to_subplot()
        elif QApplication.focusWidget() is sd.plotted:
            if self.transfer_sp is not self.current_sps[0]:
                sd.transfer_plotted()
            self.transfer_contents = None
            self.transfer_sp = None

def select_subplot(self, event, force_select=None):
    ui = self.parent
    sd = ui.series_display
    sd.plotted.clear()
    modifiers = QApplication.keyboardModifiers()

    def select(sps, add=True):
        """Toggles subplot border thickness
        and membership in self.current_sps."""
        if sps:
            for sp in sps.copy():
                plt.setp(sp.host().spines.values(),
                        linewidth=ui.highlight[add])
            if add:
                self.current_sps.append(sp)
            else:
                self.current_sps.remove(sp)
        else:
            self.current_sps = []

    if force_select is not None:
        select(self.current_sps, False)
        select(force_select)
    else:
        if event.inaxes is None:
            select(self.current_sps, False)
        else:
            sp = self.get_subplot(event)
            if modifiers == Qt.ControlModifier:
                select([sp], add=(sp not in self.current_sps))
            elif modifiers == Qt.ShiftModifier:
                if not self.current_sps:
                    select([sp])

```

```

        elif len(self.current_sps) == 1:
            current_i = self.current_sps[0].index
            select(self.current_sps, False)
            first = min([current_i, sp.index])
            last = max([current_i, sp.index])
            select(self.subplots[first:last+1])
        else:
            select(self.current_sps, False)
            select([sp])
    else:
        select(self.current_sps, False)
        select([sp])
self.update_toolbars()

def update_toolbars(self):
    ui = self.parent
    sd = ui.series_display
    st = ui.subplot_toolbar
    lt = ui.legend_toolbar
    at = ui.axes_toolbar
    b = (len(self.current_sps) >= 1)

    # Subplot Toolbar
    st.cycle.setEnabled(b)
    # st.insert.setEnabled(b)
    st.delete.setEnabled(b)
    st.clear.setEnabled(b)
    st.reorder_up.setEnabled(len(self.current_sps) == 1)
    st.reorder_down.setEnabled(len(self.current_sps) == 1)

    # Legend Toolbar
    lt.legend_toggle.setEnabled(b)
    any_legend = any([sp.legend_on for sp in self.current_sps])
    lt.legend_toggle.setChecked(any_legend)

    lt.legend_location.setEnabled(b)
    lt.legend_location.blockSignals(True)
    lt.legend_location.clear()
    if b:
        lt.legend_location.addItem(list(ui.locations.keys()))
        lt.legend_location.setCurrentText(self.current_sps[0].location)
    lt.legend_location.blockSignals(False)

    lt.legend_columns.setEnabled(b)
    lt.legend_columns.blockSignals(True)
    lt.legend_columns.clear()
    if b:
        lt.legend_columns.setValue(self.current_sps[0].ncols)
    lt.legend_columns.blockSignals(False)

    lt.legend_units.setEnabled(b)
    any_units = any([sp.legend_units for sp in self.current_sps])
    lt.legend_units.setChecked(any_units)

    lt.color_toggle.setEnabled(b)
    any_colored = any([sp.color_coord for sp in self.current_sps])
    lt.color_toggle.setChecked(any_colored)

    # Axes Toolbar
    at.selector.setEnabled(b)
    at.selector.clear()
    if b:

```

```

        if len(self.current_sps) == 1:
            sp = self.current_sps[0]
            sd.populate_tree('plotted', sp.contents)
            if sp.contents:
                at.selector.addItem('All')
                at.selector.addItems([ax.label() for ax in sp.axes])
            if len(at.current_axes) == 1:
                at.selector.setCurrentText(at.current_axes[0].label())
            else:
                at.selector.setCurrentText('All')
            ui.statusBar().showMessage(
                'Selected subplot: {}'.format(sp.index))
        else:
            sd.plotted.clear()
            selected = sorted([sp.index for sp in self.current_sps])
            ui.statusBar().showMessage(
                'Selected subplots: {}'.format(selected))
    else:
        ui.statusBar().showMessage('No subplot selected')

def prep_fig(self):
    ui = self.parent
    fs = ui.figure_settings
    self.select_subplot(None, force_select=[])
    fs.density.blockSignals(True)
    fs.density.setValue(100)
    fs.density.blockSignals(False)
    self.density = 100
    self.replot()
    self.draw()

```

Файл axes_toolbar.py

-*- coding: utf-8 -*-

```

from PyQt5.QtWidgets import (QToolBar, QComboBox, QAction,
                             QDoubleSpinBox, QAbstractSpinBox)

```

```

class AxesToolbar(QToolBar):

```

```

    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent

        self.current_axes = []
        self.selector = QComboBox()
        self.selector.setToolTip('Current Axis')
        self.selector.currentIndexChanged.connect(self.update_current_axes)
        self.selector.setMinimumHeight(33)
        self.addWidget(self.selector)

        self.log_toggle = QAction('log', self)
        self.log_toggle.setToolTip('Toggle Log Scale')
        self.log_toggle.setCheckable(True)
        self.log_toggle.setEnabled(False)
        self.log_toggle.triggered.connect(self.toggle_log_scale)
        self.addAction(self.log_toggle)

        self.autoscale_toggle = QAction('auto', self)
        self.autoscale_toggle.setToolTip('Toggle Autoscaling')
        self.autoscale_toggle.setCheckable(True)
        self.autoscale_toggle.setEnabled(False)
        self.autoscale_toggle.toggled.connect(self.manage_limit_indicators)
        self.autoscale_toggle.triggered.connect(self.toggle_autoscale_limits)

```

```

self.addAction(self.autoscale_toggle)

self.yaxis_min = QDoubleSpinBox()
self.yaxis_min.setToolTip('Y-Axis Min')
# setStepType added after 5.12, currently on 5.9
# Permission error encountered when trying to upgrade
if hasattr(self.yaxis_min, 'setStepType'):
    self.yaxis_min.setStepType(
        QAbstractSpinBox.AdaptiveDecimalStepType)
self.yaxis_min.setEnabled(False)
self.yaxis_min.valueChanged.connect(self.update_custom_limits)
self.yaxis_min.setMinimumHeight(33)
self.addWidget(self.yaxis_min)

self.yaxis_max = QDoubleSpinBox()
self.yaxis_max.setToolTip('Y-Axis Max')
if hasattr(self.yaxis_max, 'setStepType'):
    self.yaxis_max.setStepType(
        QAbstractSpinBox.AdaptiveDecimalStepType)
self.yaxis_max.setEnabled(False)
self.yaxis_max.valueChanged.connect(self.update_custom_limits)
self.yaxis_max.setMinimumHeight(33)
self.addWidget(self.yaxis_max)

def update_current_axes(self, i):
    # This signal emitted twice when:
    # - Click on an already selected subplot with contents
    # - Add/remove series from subplot
    ui = self.parent
    cf = ui.get_current_figure()
    if i == -1:
        # Selector is blank
        self.current_axes = []
        b = False
    elif i == 0:
        # Selector set to 'All'
        sp = cf.current_sps[0]
        self.current_axes = sp.axes
        b = bool(sp.host().contents)
    else:
        sp = cf.current_sps[0]
        ax = sp.axes[i-1]
        self.current_axes = [ax]
        b = bool(sp.host().contents)
    self.log_toggle.setEnabled(b)
    self.autoscale_toggle.setEnabled(b)
    any_log = any([ax.log for ax in self.current_axes])
    self.log_toggle.setChecked(any_log)
    any_auto = any([ax.auto_limits for ax in self.current_axes])
    self.autoscale_toggle.setChecked(any_auto and b)
    self.manage_limit_indicators(any_auto and b)

def toggle_log_scale(self, checked):
    ui = self.parent
    cf = ui.get_current_figure()
    log_dict = {True: 'log', False: 'linear'}
    for ax in self.current_axes:
        ax.log = checked
        ax.set_yscale(log_dict[ax.log])
    cf.format_axes()
    cf.draw()

```

```

def manage_limit_indicators(self, checked):
    i = self.selector.currentIndex()
    enable = (not checked and i > 0)
    self.yaxis_min.setEnabled(enable)
    self.yaxis_max.setEnabled(enable)
    self.yaxis_min.blockSignals(True)
    self.yaxis_max.blockSignals(True)
    if i > 0:
        ax = self.current_axes[0]
        data_min, data_max = ax.get_data_extents()
        ymin, ymax = ax.get_ylim()
        span = data_max - data_min
        margin = ax.margins()[1]
        lower = data_min - span*(1 + margin)
        upper = data_max + span*(1 + margin)
        self.yaxis_min.setMinimum(lower)
        self.yaxis_min.setMaximum(upper)
        self.yaxis_max.setMinimum(lower)
        self.yaxis_max.setMaximum(upper)
        self.yaxis_min.setValue(ymin)
        self.yaxis_max.setValue(ymax)
    else:
        self.yaxis_min.setMinimum(0)
        self.yaxis_min.setMaximum(0)
        self.yaxis_max.setMinimum(0)
        self.yaxis_max.setMaximum(0)
        self.yaxis_min.setValue(0)
        self.yaxis_max.setValue(0)
    self.yaxis_min.blockSignals(False)
    self.yaxis_max.blockSignals(False)

def toggle_autoscale_limits(self, checked):
    ui = self.parent
    cf = ui.get_current_figure()
    for ax in self.current_axes:
        ax.auto_limits = checked
        ax.autoscale(enable=ax.auto_limits, axis='y')
        if not ax.auto_limits:
            if not ax.custom_limits:
                ax.custom_limits = ax.get_ylim()
            ax.set_ylim(ax.custom_limits)

    self.manage_limit_indicators(checked)
    cf.format_axes()
    cf.draw()

def update_custom_limits(self):
    ui = self.parent
    cf = ui.get_current_figure()
    ax = self.current_axes[0]
    ymin = self.yaxis_min.value()
    ymax = self.yaxis_max.value()
    if ymin >= ymax: return
    ax.custom_limits = [ymin, ymax]
    ax.set_ylim(ax.custom_limits)
    cf.format_axes()
    cf.draw()

```

Файл edit_menu.py

```

# -*- coding: utf-8 -*-
"""edit_menu.py - Contains EditMenu class definition."""

from PyQt5.QtWidgets import QMenu, QAction

```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

```

class EditMenu(QMenu):
    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent
        ui = parent
        undo_action = QAction('Отменить', ui)
        undo_action.setShortcut('Ctrl+Z')
        undo_action.triggered.connect(self.undo)
        self.addAction(undo_action)

        redo_action = QAction('Вернуть', ui)
        redo_action.setShortcut('Ctrl+Y')
        redo_action.triggered.connect(self.redo)
        self.addAction(redo_action)

#         refresh_action = QAction('Refresh Figure', ui)
#         refresh_action.setShortcut('Ctrl+R')
#         refresh_action.triggered.connect(ui.axes_frame._draw)
#         self.addAction(refresh_action)

    def undo(self):
        pass

    def redo(self):
        pass

```

Файл file_menu.py

```

# -*- coding: utf-8 -*-
"""file_menu.py - Contains FileMenu class definition."""

import os

from PyQt5.QtWidgets import QMenu, QAction, QFileDialog

from main_proj.classes.toplevel.axes_frame import AxesFrame

class FileMenu(QMenu):
    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent
        ui = parent
        new_action = QAction('Создать', ui)
        new_action.setShortcut('Ctrl+N')
        new_action.triggered.connect(self.new)
        self.addAction(new_action)

#         open_action = QAction('Открыть', ui)
#         open_action.setShortcut('Ctrl+O')
#         open_action.triggered.connect(self.open_fig)
#         self.addAction(open_action)

        close_action = QAction('Закрыть', ui)
        close_action.setShortcut('Ctrl+W')
        close_action.triggered.connect(self.close_tab)
        self.addAction(close_action)

        save_action = QAction('Сохранить', ui)
        save_action.setShortcut('Ctrl+S')
        save_action.triggered.connect(self.save)
        self.addAction(save_action)

        save_as_action = QAction('Сохранить как', ui)

```

```

save_as_action.setShortcut('Ctrl+Shift+S')
save_as_action.triggered.connect(self.save_as)
self.addAction(save_as_action)

save_all_action = QAction('Сохранить все', ui)
save_all_action.setShortcut('Ctrl+Alt+S')
save_all_action.triggered.connect(self.save_all)
self.addAction(save_all_action)

exit_action = QAction('Выход', ui)
exit_action.setShortcut('Ctrl+Q')
exit_action.triggered.connect(ui.close)
self.addAction(exit_action)

def new(self):
    ui = self.parent
    i = ui.nfigs()
    cf = AxesFrame(ui)
    title = cf.title
    count = 1
    while cf.title in ui.open_figure_titles():
        cf.title = title + str(count)
        count += 1
    cf.fig._suptitle.set_text(cf.title)
    ui.tab_base.insertTab(i, cf, cf.title)
    ui.tab_base.setCurrentIndex(i)
    cf.replot()
    cf.saved = True

def open_fig(self):
    pass
#
# print(self.figure_settings.unit_table.size())
#
# print(self.figure_settings.majorXgrid.isChecked())
#
# pass
#
# cf = self.get_current_figure()
#
# dlg_output = QFileDialog.getOpenFileName(self,
#                                           "Open Saved Figure",
#                                           self.fig_dir,
#                                           "(*.pickle)")
#
# if dlg_output[0]:
#     fig = pickle.load(open(dlg_output[0], 'rb'))
#     _ = plt.figure()
#     manager = _.canvas.manager
#     manager.canvas.figure = fig
#     cf.fig.set_canvas(manager.canvas)
#     print(fig)
#     for ax in fig.axes:
#         print(ax)
#         print(ax.lines)
#         print(ax.get_legend())
#         h, l = ax.get_legend_handles_labels()
#         print(h)
#         print(l)
#     cf.fig.show()
#     cf.draw()
#

def close_tab(self):
    ui = self.parent
    i = ui.tab_base.currentIndex()
    ui.close_tab(i)

def save(self, _=None, cf=None):

```

```

        ui = self.parent
        if cf is None:
            cf = ui.get_current_figure()
        filename = cf.title + '.jpg'
        if cf.never_saved or filename not in os.listdir(ui.fig_dir):
            return self.save_as(cf=cf)
        else:
            cf.prep_fig()
            cf.fig.savefig(ui.fig_dir + '/' + filename,
                           dpi=300, transparent=True, bbox_inches='tight')
#         with open(self.filename + '.pickle', 'wb') as f:
#             pl.dump(cf.fig, f)
            ui.statusBar().showMessage('Saved to {}'.format(ui.fig_dir))
            cf.saved = True
            return True

def save_as(self, _=None, cf=None):
    ui = self.parent
    if cf is None:
        cf = ui.get_current_figure()
    cf.prep_fig()
    dlg = QFileDialog()
    dlg.setFileMode(QFileDialog.AnyFile)
    dlg.setViewMode(QFileDialog.Detail)
    dlg_out = dlg.getSaveFileName(ui, 'Save Figure',
                                  ui.fig_dir + '/' + cf.title,
                                  'PNG Image (*.png)')

    # JPG export in matplotlib is bugged as of Jun 30 2020
    if dlg_out[0]:
        savepath = dlg_out[0]
        # store saving directory
        ui.fig_dir = os.path.dirname(savepath)
        cf.fig.savefig(savepath,
                        dpi=300, transparent=True, bbox_inches='tight')
#         with open(self.filename + '.pickle', 'wb') as f:
#             pkl.dump(cf.fig, f)
        ui.statusBar().showMessage('Saved to {}'.format(savepath))
        cf.saved = True
        cf.never_saved = False
        return True
    else:
        return False

def save_all(self, _=None):
    ui = self.parent
    for cf in ui.all_figures():
        self.save(cf)

```

Файл legend_toolbar.py

```

# -*- coding: utf-8 -*-
"""legend_toolbar.py - Contains LegendToolbar class definition."""

from PyQt5.QtWidgets import (QToolBar, QComboBox, QAction, QSpinBox)

class LegendToolbar(QToolBar):

    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent

        self.color_toggle = QAction('cc', self)
        self.color_toggle.setToolTip('Color Coordinate')
        self.color_toggle.setCheckable(True)

```

```

self.color_toggle.setEnabled(False)
self.color_toggle.triggered.connect(self.color_coordinate)
self.addAction(self.color_toggle)

self.legend_toggle = QAction('legend', self)
self.legend_toggle.setToolTip('Отображение легенды')
self.legend_toggle.setCheckable(True)
self.legend_toggle.setEnabled(False)
self.legend_toggle.triggered.connect(self.toggle_legend)
self.addAction(self.legend_toggle)

# self.legend_units = QToolButton(self)
# def iconText():
#     return 'units'
# self.legend_units.setIconText = iconText
self.legend_units = QAction('units', self)
self.legend_units.setToolTip('Отображение единиц измерения в
легенде')
self.legend_units.setCheckable(True)
self.legend_units.setEnabled(False)
self.legend_units.triggered.connect(self.toggle_legend_units)
self.addAction(self.legend_units)

self.legend_location = QComboBox()
self.legend_location.setToolTip('Положение легенды')
self.legend_location.setEnabled(False)
self.legend_location.addItem('None selected')
# !!! Figure out how to make these comboboxes the right size!
# self.legend_location.addItem(list(self.locations.keys()))
self.legend_location.clear()
self.legend_location.currentTextChanged.connect(
    self.set_legend_location)
self.legend_location.setMinimumHeight(33)
self.addWidget(self.legend_location)

self.legend_columns = QSpinBox()
self.legend_columns.setToolTip('Колонки легенды')
self.legend_columns.setEnabled(False)
self.legend_columns.setRange(1, 10)
self.legend_columns.clear()
self.legend_columns.valueChanged.connect(self.set_legend_columns)
self.legend_columns.setMinimumHeight(33)
self.addWidget(self.legend_columns)

def toggle_legend(self, checked):
    ui = self.parent
    cf = ui.get_current_figure()
    for sp in cf.current_sps:
        sp.legend_on = checked
    cf.replot(data=False)
    cf.update_gridspec()
    cf.draw()

def set_legend_location(self, text):
    ui = self.parent
    cf = ui.get_current_figure()
    for sp in cf.current_sps:
        sp.location = text
    cf.replot(data=False)
    cf.update_gridspec()
    cf.draw()

```

```

def set_legend_columns(self, value):
    ui = self.parent
    cf = ui.get_current_figure()
    for sp in cf.current_sps:
        sp.ncols = value
    cf.replot(data=False)
    cf.draw()

def toggle_legend_units(self, checked):
    ui = self.parent
    cf = ui.get_current_figure()
    for sp in cf.current_sps:
        sp.legend_units = checked
    cf.replot(data=False)
    cf.draw()

def color_coordinate(self, checked):
    """Coordinates colors of selected subplot(s) by unit type (not unit).
    Each unit type gets its own color for lines, labels, and ticks."""
    ui = self.parent
    cf = ui.get_current_figure()
    for sp in cf.current_sps:
        sp.color_coord = checked
    cf.replot()
    cf.draw()

```

Файл series_display.py

```
# -*- coding: utf-8 -*-
```

```
"""series_display.py - Contains SeriesDisplay class definition."""
```

```
import re
```

```

from PyQt5.QtWidgets import (QDockWidget, QGridLayout, QLineEdit,
                             QWidget, QTreeWidget, QTreeWidgetItem)
from PyQt5.QtCore import QObject

```

```

from main_proj.classes.internal.contents_dict import ContentsDict
from main_proj.classes.internal.edit_series_dialog import EditSeriesDialog

```

```
class SeriesDisplay(QDockWidget):
```

```
    """Displays hierarchical string references to imported data
    groups/aliases.
```

```
    Available tree shows series which have not yet been plotted.
    Plotted tree shows contents of selected subplot."""
```

```
def __init__(self, parent, title):
```

```
    super().__init__(title)
```

```
    self.parent = parent
```

```
    grid = QGridLayout()
```

```
    w = QWidget()
```

```
    ncols = 1
```

```
    self.available = QTreeWidget()
```

```
    self.available.setSelectionMode(QTreeWidget.ExtendedSelection)
```

```
    self.available.setSortingEnabled(True)
```

```
    self.available.setHeaderLabels(["Доступные серии"])
```

```
    self.available.setColumnCount(ncols)
```

```
    self.available.setDragEnabled(True)
```

```
    self.available.setAcceptDrops(True)
```

```
    self.available.dragEnterEvent = self.available_drag_enter
```

```
    self.available.dropEvent = self.available_drop_event
```

```
    self.available.itemDoubleClicked.connect(self.series_double_clickeded)
```

```
    self.plotted = QTreeWidget()
```

					ПГУ1.090301.38.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		89

```

self.plotted.setSelectionMode(QTreeWidget.ExtendedSelection)
self.plotted.setSortingEnabled(True)
self.plotted.setHeaderLabels(["Серии выбранного графика"])
self.plotted.setColumnCount(ncols)
self.plotted.setDragEnabled(True)
self.plotted.setAcceptDrops(True)
self.plotted.dragEnterEvent = self.plotted_drag_enter
self.plotted.dropEvent = self.plotted_drop_event
self.plotted.itemDoubleClicked.connect(self.series_double_clicked)

self.search_available = QLineEdit()
self.search_available.setPlaceholderText('Поиск')
self.search_available.textChanged.connect(self.search)

self.search_plotted = QLineEdit()
self.search_plotted.setPlaceholderText('Поиск')
self.search_plotted.textChanged.connect(self.search)

grid.addWidget(self.search_available, 0, 0)
grid.addWidget(self.search_plotted, 0, 1)
grid.addWidget(self.available, 1, 0)
grid.addWidget(self.plotted, 1, 1)
w.setLayout(grid)
self.setWidget(w)

self.available.expandAll()
self.available.resizeColumnToContents(0)

def available_drag_enter(self, event):
    if event.source() is self.plotted:
        event.accept()
    else:
        event.ignore()

def available_drop_event(self, event):
    self.return_to_available()

def plotted_drag_enter(self, event):
    ui = self.parent
    cf = ui.get_current_figure()
    if len(cf.current_sps) == 1:
        event.accept()
    else:
        event.ignore()

def plotted_drop_event(self, event):
    self.add_to_subplot()

def series_double_clicked(self, item, column):
    ui = self.parent
    if item.parent():
        dlg = EditSeriesDialog(ui, item)
        dlg.exec_()

def cleanup(self):
    """Sorts both trees and deletes any empty group references."""
    for tree in [self.available, self.plotted]:
        tree.sortItems(0, 0) # sort by column 0 in ascending order
        to_delete = []
        for i in range(tree.topLevelItemCount()):
            if tree.topLevelItem(i).childCount() == 0:
                to_delete.append(i)

```

```

        else:
            tree.topLevelItem(i).sortChildren(0, 0)
    for x in to_delete:
        tree.takeTopLevelItem(x)

def populate_tree(self, which, contents, apply_filter=True):
    """Clears target_tree and repopulates with contents"""
    if which == 'available':
        target_tree = self.available
        w = self.search_available
    elif which == 'plotted':
        target_tree = self.plotted
        w = self.search_plotted
    target_tree.clear()
    if contents:
        for group in contents:
            level0 = QTreeWidgetItem([group])
            target_tree.addTopLevelItem(level0)
            # add series to their corresponding level0 entry
            for alias in contents[group]:
                level1 = QTreeWidgetItem([alias])
                level0.addChild(level1)
            level0.setExpanded(True)
        target_tree.resizeColumnToContents(0)
        if apply_filter:
            w.textChanged.emit(w.text())
        self.cleanup()

def tree_items_to_contents(self, selected):
    # Read contents from selected items
    contents = ContentsDict()
    if all(item.parent() is None for item in selected):
        for item in selected:
            count = item.childCount()
            children = [item.child(i) for i in range(count)]
            for child in children: item.removeChild(child)
            group = item.text(0)
            aliases = [child.text(0) for child in children]
            contents.add({group: aliases})
    else:
        for item in selected:
            if item.parent():
                # if selected item is level1 (ignore if level0)
                parent = item.parent()
                i = parent.indexOfChild(item)
                child = parent.takeChild(i)
                group = parent.text(0)
                aliases = [child.text(0)]
                contents.add({group: aliases})
    return contents

def return_to_available(self):
    ui = self.parent
    cf = ui.get_current_figure()
    selected = self.plotted.selectedItems()
    if selected:
        contents = self.tree_items_to_contents(selected)
        sp = cf.current_sps[0]
    else:
        contents = cf.transfer_contents
        sp = cf.transfer_sp
    sp.remove(contents)

```

```

cf.available_data.add(contents)
sp.plot()
sp.show_legend()
cf.select_subplot(None, force_select=[sp])
cf.update_gridspec()
cf.draw()
# Populate both trees and reapply search filter
self.populate_tree('available', cf.available_data)
self.populate_tree('plotted', sp.contents)
cf.transfer_contents = None
cf.transfer_sp = None

def add_to_subplot(self):
    ui = self.parent
    cf = ui.get_current_figure()
    selected = self.available.selectedItems()
    contents = self.tree_items_to_contents(selected)
    sp = cf.current_sps[0]
    sp.add(contents)
    cf.available_data.remove(contents)
    sp.plot()
    sp.show_legend()
    cf.select_subplot(None, force_select=[sp])
    cf.update_gridspec()
    cf.draw()
    # Populate both trees and reapply search filter
    self.populate_tree('available', cf.available_data)
    self.populate_tree('plotted', sp.contents)

def transfer_plotted(self):
    ui = self.parent
    cf = ui.get_current_figure()
    contents = cf.transfer_contents
    sp = cf.current_sps[0]
    cf.transfer_sp.remove(contents)
    sp.add(contents)
    cf.replot()
    cf.select_subplot(None, force_select=[sp])
    cf.update_gridspec()
    cf.draw()
    # Populate both trees and reapply search filter
    self.populate_tree('plotted', sp.contents)

def get_sp_contents(self):
    """Returns contents of selected subplot if exactly one is selected.
    Otherwise returns an empty ContentsDict object."""
    ui = self.parent
    cf = ui.get_current_figure()
    if len(cf.current_sps) == 1:
        return cf.current_sps[0].contents
    else:
        return ContentsDict()

def search(self, text):
    """Displays series in tree which match input to search bars.
    (case insensitive)"""
    search_bar = QObject.sender(self)
    if search_bar == self.search_available:
        which = 'available'
        cf = self.parent.get_current_figure()
        data_set = cf.available_data
    elif search_bar == self.search_plotted:

```

```

        which = 'plotted'
        data_set = self.get_sp_contents()
        # There's probably a more efficient way to do this. More pythonic.
        user_input = re.compile(text, re.IGNORECASE)
        matches = ContentsDict()
        if data_set:
            for group, aliases in data_set.items():
                matches.add({group:
                    [a for a in aliases if user_input.search(a)]})
#             # add an input method to group.series(which)
#             # for which=function
#             # ie lambda x: x.keep
#             # or lambda x: user_input.search(x)
        self.populate_tree(which, matches, apply_filter=False)

```

Файл tools_menu.py

```
# -*- coding: utf-8 -*-
```

```
import json
import os
```

```
from PyQt5.QtWidgets import QMenu, QAction, QFileDialog
from PyQt5.QtCore import Qt
```

```
from main_proj.classes.manager.data_manager import DataManager
```

```
class ToolsMenu(QMenu):
```

```

    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent
        ui = parent
        data_action = QAction('Импорт данных', ui)
        data_action.setShortcut('Ctrl+D')
        data_action.triggered.connect(self.open_data_manager)
        self.addAction(data_action)

```

```

        export_template_action = QAction('Экспорт шаблона', ui)
        export_template_action.setShortcut('Ctrl+E')
        export_template_action.triggered.connect(self.export_template)
        self.addAction(export_template_action)

```

```

        import_template_action = QAction('Импорт шаблона', ui)
        import_template_action.setShortcut('Ctrl+I')
        import_template_action.triggered.connect(self.import_template)
        self.addAction(import_template_action)

```

```

    def open_data_manager(self):
        ui = self.parent
        ui.statusBar().showMessage('Открыто окно импорта данных')
        ui.dlg = DataManager(ui)
        ui.dlg.setWindowFlags(Qt.Window)
        ui.dlg.setModal(True)
        ui.dlg.show()

```

```

    def export_template(self):
        ui = self.parent
        dlg = QFileDialog()
        dlg.setFileMode(QFileDialog.AnyFile)
        dlg.setViewMode(QFileDialog.Detail)
        dlg_out = dlg.getSaveFileName(ui, 'Экспорт шаблона',
                                     'templates',
                                     'JSON (*.json)')

        if dlg_out[0]:

```

					ПГУ1.090301.38.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

export_path = dlg_out[0]
cf = ui.get_current_figure()
template = {
    'tsf': cf.tsf,
    # 'title': 'Untitled',
    'upper_pad': cf.upper_pad,
    'lower_pad': cf.lower_pad,
    'left_pad': cf.left_pad,
    'right_pad': cf.right_pad,
    'spacing': cf.spacing,
    'axis_offset': cf.axis_offset,
    'MX': cf.MX,
    'mx': cf.mx,
    'MY': cf.MY,
    'my': cf.my,
    'M_T': cf.M_T,
    'm_T': cf.m_T,
    'scatter': cf.scatter,
    'dot_size': cf.dot_size,
    'density': cf.density,
    'x_margin': cf.x_margin,
    'title_size': cf.title_size,
    'label_size': cf.label_size,
    'tick_size': cf.tick_size,
    'tick_rot': cf.tick_rot,
    'nplots': cf.nplots(),
    'weights': cf.weights()
}
with open(export_path, 'w', encoding='utf-8') as f:
    json.dump(template, f, indent=2, ensure_ascii=False)
ui.statusBar().showMessage('Exported to {}'.format(export_path))
return True
return False

def import_template(self):
    ui = self.parent
    dlg = QFileDialog()
    dlg.setFileMode(QFileDialog.AnyFile)
    dlg.setViewMode(QFileDialog.Detail)
    dlg_out = dlg.getOpenFileName(ui, 'Импорт шаблона',
                                  'templates',
                                  'JSON (*.json)')

    if dlg_out[0]:
        import_path = dlg_out[0]
        cf = ui.get_current_figure()
        with open(import_path, 'r', encoding='utf-8') as f:
            template = json.load(f)
            nplots = template['nplots']
            weights = template['weights']
            del template['nplots'], template['weights']

            for k,v in template.items():
                setattr(cf, k, v)
            if cf.nplots() > nplots:
                filename = os.path.split(import_path)[1].split('.')[0]
                ui.popup(
                    text='Too few subplots in template', title='Import
Template',
                    informative='\"{}\" has fewer subplots than the current
figure.'
                    'The number of subplots and their weights are
unchanged.'.format(filename),

```

```

        mode='alert'
    )
    else:
        while cf.nplots() < nplots:
            ui.subplot_toolbar.insert_subplot()
            cf.update_gridspec(nplots, weights)
            ui.figure_settings.update_fields(cf)
            cf.draw()
            return True
        return False

```

Файл view_menu.py

```

# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QMenu, QAction

class ViewMenu(QMenu):
    def __init__(self, title, parent):
        super().__init__(title)
        self.parent = parent
        ui = parent
        docks_action = QAction('Видимость боковых панелей', ui)
        docks_action.setShortcut('Ctrl+H')
        docks_action.triggered.connect(self.toggle_docks)
        self.addAction(docks_action)

        dark_action = QAction('Включить темную тему', ui)
        dark_action.setShortcut('Ctrl+B')
        dark_action.triggered.connect(self.toggle_dark_mode)
        self.addAction(dark_action)

    def toggle_docks(self):
        ui = self.parent
        docks = [ui.series_display, ui.figure_settings]
        if any([not dock.isVisible() for dock in docks]):
            for dock in docks: dock.show()
        else:
            for dock in docks: dock.hide()

    def toggle_dark_mode(self):
        ui = self.parent
        if ui.mode == 'light':
            ui.mode = 'dark'
            ui.current_qss = ui.dark_qss
            ui.current_rcs = ui.dark_rcs
            ui.current_icon_path = 'rc/entypo/dark'
        else:
            ui.mode = 'light'
            ui.current_qss = ui.light_qss
            ui.current_rcs = ui.light_rcs
            ui.current_icon_path = 'rc/entypo/light'
        default_color = ui.current_rcs['axes.labelcolor']
        ui.color_dict[None] = default_color
        ui.color_dict[''] = default_color
        ui.set_app_style(ui.current_qss,
                        ui.current_rcs,
                        ui.current_icon_path)

```

Приложение В
(обязательное)

Приложение анализа и визуализации телеметрических данных

Презентация

					ПГУ1.090301.38.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение анализа и визуализации телеметрических данных

Автор работы:

студент гр. 22ВВП2 Кочетков Артём Михайлович

Научный руководитель:

к.т.н., доцент каф. ВТ ПГУ Федюнин Роман Николаевич

Пенза, 2026

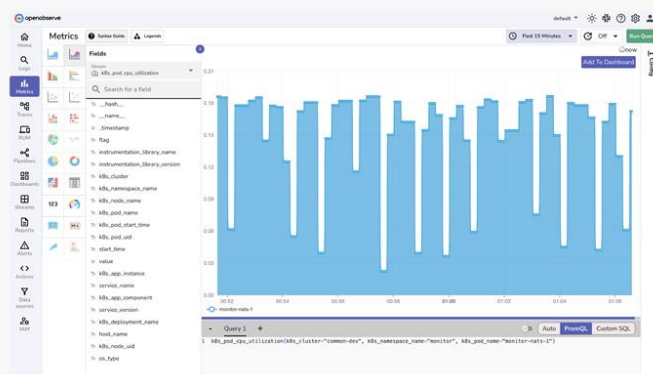
Цели работы

- Разработка десктопного приложения, обеспечивающего импорт, обработку и наглядное графическое представление телеметрических данных с возможностью гибкой настройки отображения для последующего анализа пользователем.
- Поддержка нескольких подграфиков, настройка сетки, легенды, цветовой кодировки по типам единиц измерения
- Автоматическое распознавание единиц измерения, временных меток; возможность ручной корректировки параметров импорта, экспорт конфигураций графиков в JSON-шаблоны, сохранение настроек пользователя между сеансами.

Обзор аналогов Grafana



Обзор аналогов OpenObserve



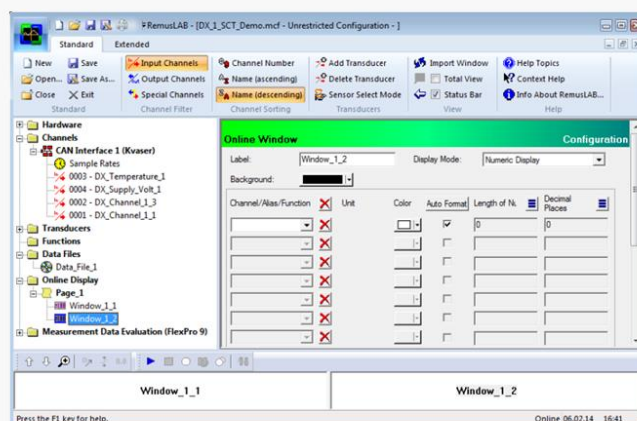
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

Лист

98

Обзор аналогов RemusLab



Стек технологий

■ Python

- Matplotlib
- NumPy
- Pandas
- PyQt5



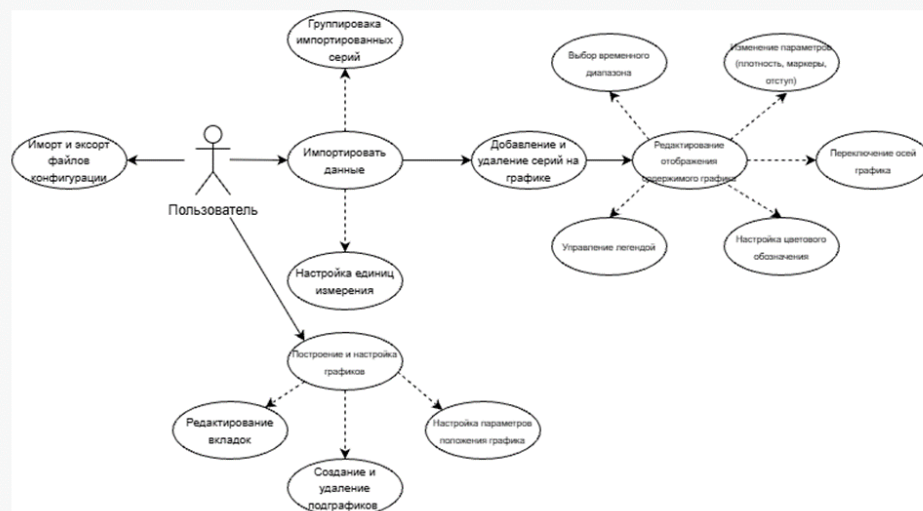
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.38.001 ПЗ

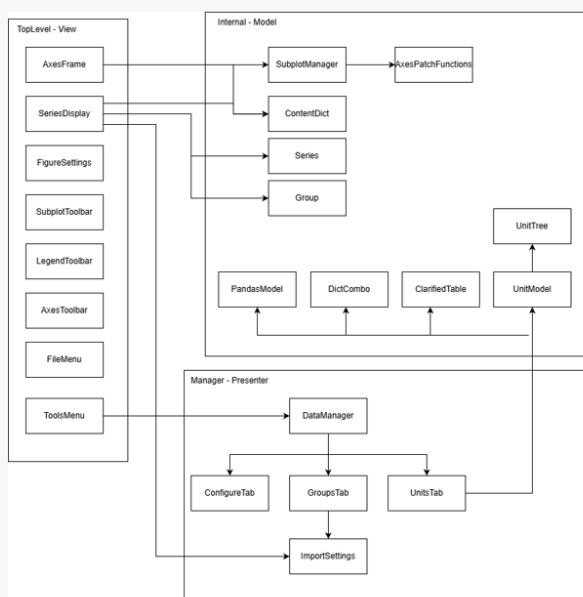
Лист

99

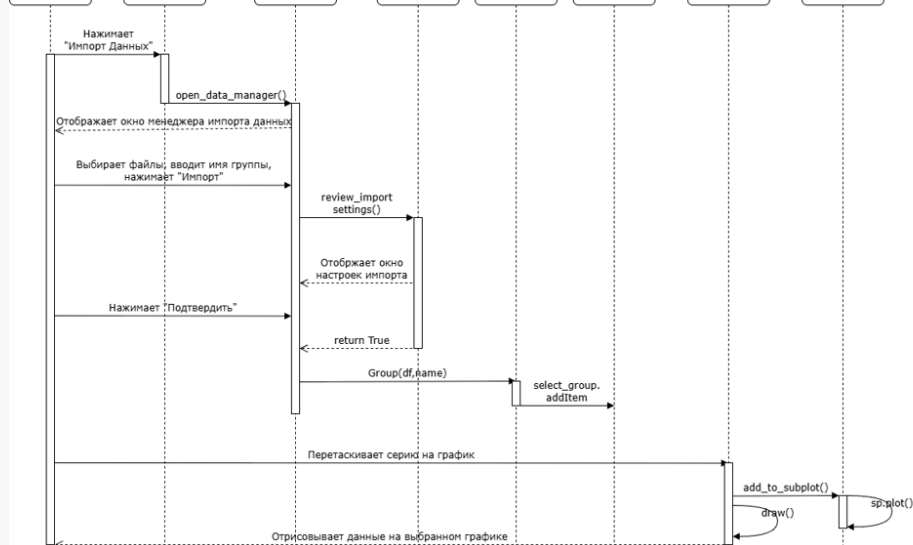
UML диаграмма использования



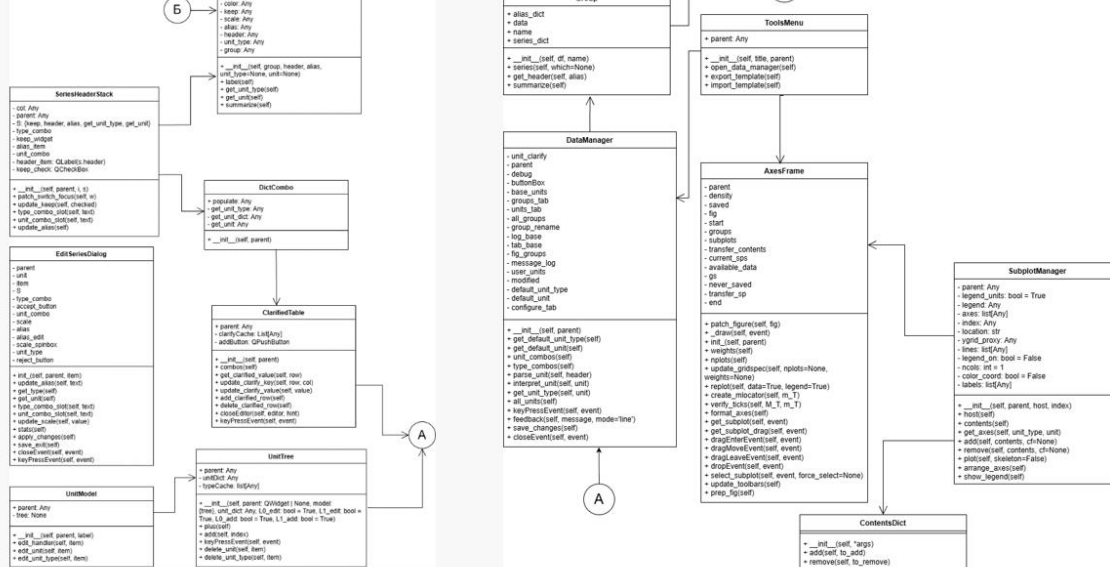
UML диаграмма модулей



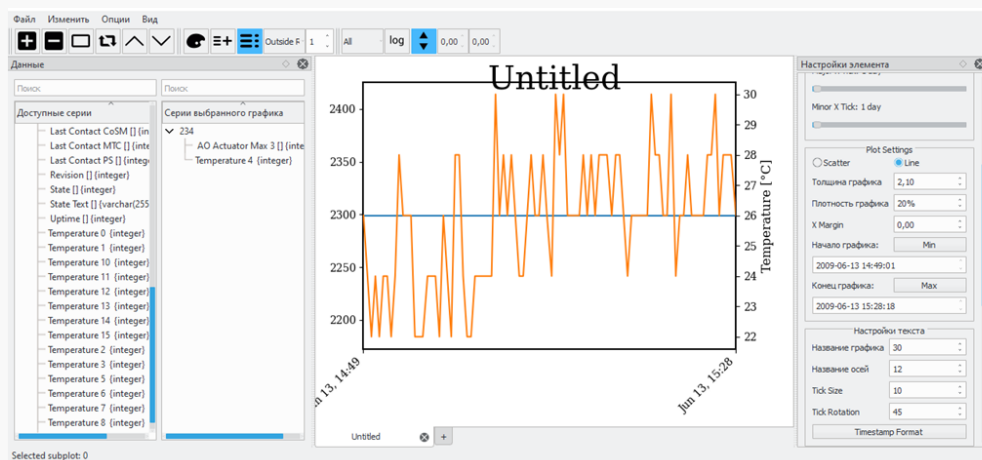
Пользователь ToolsMenu DataManager ImportSettings Group MainWindow SeriesDisplay AxesFrame



Series
- unit: Any



Приложение анализа и визуализации телеметрических данных



Приложение анализа и визуализации телеметрических данных

Группировка файлов			Настройка серий		Единицы измерения	
S:\repos\tests			66		Импортировать группу	
Найденные файлы			Файлы в группе		Figure Groups	
Имя	Размер	Дата изменения	PHI Temps.csv		77	77
cws_reduced.csv	28,720 KB	01-Jul-20 00:46				
invalid.csv	0 KB	23-May-26 09:01				
Parse Test.csv	1,455 KB	01-Jul-20 00:46				
PHI aPCM.csv	5,197 KB	01-Jul-20 00:46				
PHI Temps.csv	780 KB	01-Jul-20 00:46				
Sunrise Test.csv	9,481 KB	01-Jul-20 00:46				
Test.csv	1,471 KB	01-Jul-20 00:46				
Test.zip	111 KB	01-Jul-20 00:46				
Message Log:			Loading previews: (1/1)... Review import settings... Import canceled.			
			Save Close			

Заключение

- Разработанное приложение представляет собой самостоятельное законченное программное решение, которое может быть использовано для анализа телеметрических данных в различных предметных областях: промышленном мониторинге, энергетике, транспортных системах, вендинге, сельском хозяйстве, а также в научных исследованиях, связанных с обработкой временных рядов. В отличие от существующих аналогов, приложение ориентировано на работу с локальными файлами произвольной структуры, не требует развёртывания дополнительного программного стека, предоставляет интуитивно понятный интерфейс и гибкие средства настройки визуализации.

Спасибо за внимание!