ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИММиТ, кафедра «Мехатроника и роботостроение» при ЦНИИ РТК

Отчет о прохождении производственной практики

Водорезов Г.И.
(Ф.И.О. обучающегося)

3 курс, группа 3331506/70401
(номер курса обучения и учебной группы)

Направление подготовки 15.03.06, «Мехатроника и робототехника»
(Направление подготовки (код и наименование)

Место прохождения практики: «Центральный научно-исследовательский и опытно-конструкторский институт робототехники и технической кибернетики»

Россия, 194064, г. Санкт-Петербург, Тихорецкий пр. д.21

Сроки практики: 28.06.2020г. – 18.07.2020г.

Руководитель практики от ФГАОУ ВО «СПбПУ»:

Габриель А.С., ст. преподаватель

Руководитель практики от профильной организации:

Бахшиев А.В, ведущий научный сотрудник

Оценка:

Научный руководитель-консультант:

Руководитель практики
от ФГАОУ ВО «СПбПУ» Габриель А.С.

Дата:

Оглавление

Введение

Современные средства моделирования общего назначения (например Matlab) позволяют описывать сложные системы и обладают мощными библиотеками готовых решений. Однако ряд задач требует, чтобы модели и алгоритмы обеспечивали не только параметрическую, но и структурную адаптацию, то есть позволяли менять, как состав элементов модели, так и структуру связей между элементами. Применительно к робототехнике можно выделить такие задачи, как создание информационно-управляющих систем на базе нейронных сетей с изменяемой топологией; разработка адаптивных подсистем технического зрения и принятия решений и др. Для реализации алгоритмов, решающих такие задачи необходимо создание среды моделирования, которая может обеспечить структурную адаптацию модели онлайн, т.е. во время выполнения, и при этом будет вычислительно эффективна, чтобы обеспечить возможность экспериментальной отработки на реальных робототехнических платформах.

1. Индивидуальное задание

1.1. Описание исходного приложения

RDK (Research Development Kit) – программное обеспечение, предоставляющая средства для создания ядра с заранее неизвестной или изменяемой структурой в процессе работы самой программы. В данном ПО так же реализованы унифицированный доступ к параметрам, входным и выходным данным элементов этой структуры (алгоритмов) и предоставлены средства для загрузки и сохранения структуры ядра.

Программа имеет двухуровневую архитектуру: уровень библиотеки алгоритмов, и уровень конечного ПО, использующего библиотеку.

Унификация интерфейса пользователя ядра, позволит существенно упростить внедрение конечных вариантов ядер, обеспечивающих решение поставленных задач, в состав использующих их продуктов. Помимо интерфейса пользователя библиотеки, также обеспечена унификацию интерфейса разработчика алгоритмов библиотеки. Здесь унификация может преследовать цель снижения требований от разработчика конечных алгоритмов к глубине познаний в архитектуре библиотеки.

RDK представляет собой платформу для создания таких библиотек, и реализует описанный выше функционал. При разработке RDK внимание уделялось в первую очередь снижению порога вхождения для пользователя конечной библиотекой и для разработчика конечных алгоритмов.

1.2. Техническое задание

Задание состоит в разработке программного виджета для платформы RDK, который бы выполнял роль интерфейса пользователя для визуализации данных на графиках.

К данному виджету были предъявлены следующие требования:

1. Виджет должен быть выполнен в виде отдельно, независимого окна.
2. Возможность создания нескольких вкладок, их добавление и удаление.
3. На каждой вкладке можно создавать несколько окон для графиков с независимыми параметрами и с различным расположением графиков на вкладке

4. Возможность настроить в каждом графике его названия, подписи осей, их максимума и минимума, размер графика относительно остальных графиков
5. Добавление несколько серий данных в один график, для каждой серии возможность изменять настройки цвет, толщину и тип линии, название серии в легенде графика
6. Отдельное окно, содержащее настройки параметров графика и их расположения внутри вкладки, время обновления данных для данной вкладки.
7. Отдельное окно для настройки параметров каждой серии.
8. Реализовать способ добавления источника данных, в платформе RDK это какой-либо выбранный параметр отдельного компонента.
9. Возможность сохранения отдельного графика или всей вкладки целиков в формате JPEG.
10. Подключить получившийся виджет в исходное приложение.

Так как платформа RDK разрабатывается на базе фрэймворка Qt, то и реализованный мною виджет был сделан в этой среде разработки.

2. Реализация

Исторически сложилось, что платформа RDK разрабатывалась с помощью фреймворка Qt и языка программирования C++, следовательно и мой виджет выполнен в данной среде разработке. Данный фреймворк так же содержит множество собственно разработанных библиотек, некоторые из которых расширяют возможности языка (QDir, QVector), а некоторые добавляли возможность с графическим интерфейсом пользователя, и, в частности, непосредственно с графиками и представлением серий данных на графиках (QChart, QLineSeries).

2.1 Диаграмма классов.

Диаграмма классов выполнена с помощью языка графического описания UML. Данная диаграмма позволяет наглядно понять структуру системы, поля и методы каждого из классов и зависимости между ними. Диаграмма представлена на Рисунок 1.

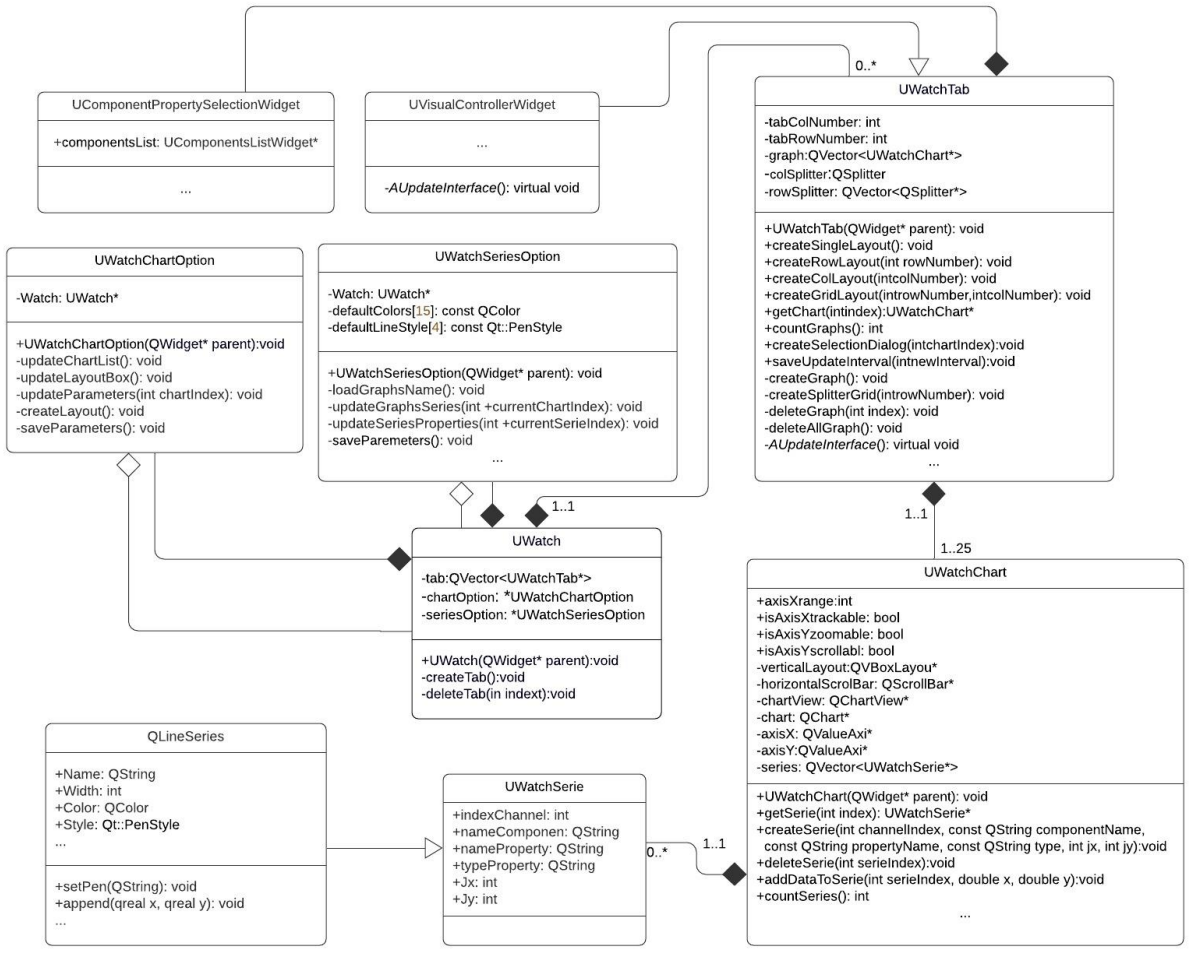

Рисунок 1 - UML-диаграмма классов.

2.2. Структура файлов

В ходе работы были реализованы 6 классов, каждый из которых отвечает за свою логически завершенную часть виджета и имеет интерфейс взаимодействия с другими классами. Каждый класс состоит из 3-х файлов: заголовочный файл, файл с реализацией и файл с реализацией UI (User interface).

2.2.1. UWatch

Основной класс, при создании экземпляра которого происходит создание требуемого виджета и первой вкладке в его основной части. Данный класс имеет функционал по вызову окон настроек для графиков и серий, а также сохранения скриншота для текущей вкладки.

Данный класс имеет следующие публичный метод:

- UWatchTab *getCurrentTab() – возваращет указатель на текущую вкладку

Так же ниже перечислены некоторые приватные поля и методы:

- QVector <UWatchTab*> tab – динамический массив указателей на все созданные вкладки.
- UWatchChartOption *chartOption – указатель на экземпляр класса UWatchChartOption.
- UWatchSeriesOption *seriesOption – указатель на экземпляр класса UWatchSeriesOption.
- void createTab() - создает новую вкладку
- void deleteTab(int index) – удаляет вкладку с индекосм index

Создание экземпляра классов UWatchSeriesOption и UWatchChartOption происходит через соответствующие кнопки в меню, находящимся в верхней части виджета.

2.2.2 UWatchTab

Класс, реализующий одну вкладку. Хранит в себе массив всех графиков, расположенных на нем, а также способ их расположения. При создании экземпляра данного класса на нем автоматически создается 1 график. Данный класс унаследован от класса UVisualControllerWidget. Этот класс реализован в исходном приложении и является абстрактным классом для всех будущих классов, реализующих интерфейс приложения. Класс UVisualControllerWidget предоставляет доступ к ядру программы и имеет виртуальные методы для работы с данными ядра.

Данный класс имеет следующие публичный поля и методы:

- void createSingleLayout() – создает один график на вклдке.
- void createRowLayout(int rowNumber) – создает rowNumber графиков на вкладке и располагает их горизонтально.
- void createColLayout(int colNumber) – создает colNumber графиков на вкладке и располагает их вертикально.
- void createGridLayout(int rowNumber, int colNumber) – создает rowNumber * colNumber графиков на вкладке и располагает их вертикально в виде сетки.
- int layoutMode – способ расположения графиков на вкладке (0 – один график, 1 – вертикально, 2 – горизонтально, 3 – сетка).
- UWatchChart *getChart(int index) – возвращает график с индексом index.
- int countGraphs() – возвращает количество графиков
- void createSelectionDialog(int chartIndex) – вызов диалогового окна для добавления новой серии в график с индексом chartIndex.
- void saveUpdateInterval(int newInterval) – устанавливает новый интервал для отрисовки данных на всех графиках вкладки.
- int getColNumber() – возвращает количество вертикально расположенных графиков.
- int getRowNumber() – возвращает количество горизонтально расположенных графиков.

Так же стоит упомянуть о некоторых приватных полях данного класса:

- QVector <UWatchChart*> graph – динамический массив всех графиков на вкладке.
- virtual void AUpdateInterface() – данный метод вызывается каждые UpdateIntervalMs мс и используется для добавления новых данных в каждую серию данных для каждого графика. В данном методе реализован доступ к ядру программы и добавлению текущих значений соответствующего выхода компонента для соответствующей ему серии данных.

2.2.3. UWatchChart

Класс, реализующий один график. Хранит в себе массив серий данных, а также в нем реализованы методы доступа к свойствам графика и его серий.

Публичные методы данного класса:

- QString getChartTitle() – возвращает имя графика.

- QString getAxisXName() -возвращает имя оси X.
- QString getAxisYName()-возвращает имя оси Y.
- double getAxisXmin() – возвращает минимальное значение по оси X.
- double getAxisXmax()– возвращает максимальное значение по оси X.
- double getAxisYmin() – возвращает минимальное значение по оси Y.
- double getAxisYmax() – возвращает максимальное значение по оси Y.

- UWatchSerie *getSerie(int index) – возвращает серию с индексом index.
- QString getSerieName(int serieIndex) – возвращает имя серии с индексом serieIndex.
- QColor getSerieColor(int serieIndex) – возвращает цвет линии серии с индексом serieIndex.
- int getSerieWidth(int serieIndex) – возвращает ширину линии серии с индексом serieIndex.
- Qt::PenStyle getSerieLineType(int serieIndex) – возвращает тип линии серии с индексом serieIndex.
- void setChartTitle(QString title) – устанавливает новое имя title для графика.
- void setChartIndex(int index) – устанавливает номер графика index
- void setAxisXname(QString name) – устанавливает новое имя name для оси X.
- void setAxisYname(QString name) – устанавливает новое имя name для оси Y.
- void setAxisXmin(double value) – устанавливает новый минимум value для оси X.
- void setAxisXmax(double value) – устанавливает новый максимум value для оси X.
- void setAxisYmin(double value) – устанавливает новый минимум value для оси Y.
- void setAxisYmax(double value) – устанавливает новый максимум value для оси Y.

- void setSerieName(int serieIndex, QString name) – устанавливает новое имя name для серии с номером serieIndex.
- void setSerieColor(int serieIndex, int colorIndex) – устанавливает новый цвет из массива defaultColors с индексом colorIndex для серии с номером serieIndex.
- void setSerieLineType(int serieIndex, Qt::PenStyle lineType) – устанавливает новый тип линии lineType для серии с номером serieIndex.

- void setSerieWidth(int serieIndex, int width) – устанавливает новую ширину линии width для серии с номером serieIndex.
- void setSerieStyle(int serieIndex, QColor color, int width, Qt::PenStyle lineType) – устанавливает новый цвет color, ширину линии width и тип линии lineType для серии с номером serieIndex.
- void createSerie(int channelIndex, const QString componentName, const QString propertyName, const QString type, int jx, int jy) – создает новую серию и прикрепляет к ней данные об источнике данных.
- void deleteSerie(int serieIndex) -удаляет серию с индексом serieIndex.
- void addDataToSerie(int serieIndex, double x, double y) -добавляет данные точку с координатами x и y на текущем графике для серии с индексом serieIndex.
- int countSeries() – возвращает количество серий в графике.
- int axisXrange = 5 – ширина области видимость по оси X
- bool isAxisXtrackable = true – автослежение за концом серий по оси X.
- bool isAxisYzoomable = true – возможность масштабировать область видимости по оси Y.
- bool isAxisYscrollable = true – возможность двигать область видимости по оси Y.

Некоторые приватные методы:

- void addSeriesSlot() – вызывает окно для добавления новой серии в график, вызывается через popup-меню.
- void saveToJpegSlot() – сохраняет график в формате .jpeg в папку «screenshots» в корневой папке программы, вызывается через popup-меню.

2.2.4. UWatchSerie

Данный класс унаследован от класса QLineSeries, который принадлежит стандартной для фреймворка Qt библиотеке QCharts и реализует одну линию на графике. Так же каждый объект данного класса хранит данные об источнике данных:

- int indexChannel – номер канала.
- QString nameComponent – имя компонента.
- QString nameProperty – имя параметра.
- QString typeProperty – тип параметра.
- int Jx – столбец в матрице данных.
- int Jy - строка в матрице данных.

2.2.5. UWatchChartOption

Этот класс реализует диалоговое окно, которое содержит в себе настройки расположения графиков и графический интерфейс для

изменения параметров графиков. Данный класс не содержит публичных методов, но стоит отметить ключевые приватные методы:

- void updateParameters(int chartIndex) – загружает на форму все данные о параметрах графика с индексом chartIndex для текущей вкладки.
- void createLayout() – создание нового расположения на вкладке для выбранного графика.
- void saveParameters() – обновление параметров выбранного графика.

2.2.6. UWatchSeriesOption

Класс, реализующий диалоговое окно, содержащее в себе формы для изменения параметров конкретной серии для конкретного графика и способ добавления новых серий. Этот класс, как и предыдущий не содержит публичных методов, основные приватные методы:

- void updateGraphsSeries(int currentChartIndex) – обновляет список серий для графика с индексом currentChartIndex.

- void updateSeriesProperties(int currentSerieIndex) – обновляет данные в формах для изменения параметров серии с индексом currentSerieIndex.
- void saveParemeters() – сохраняет изменения параметров для текущей выбранной серии для выбранного графика.

2.3 Использование классов.

Для подключение реализованных программных модулей к основной программе необходимо добавить файлы всех классов в основной проект, а также подключить библиотеку QCharts. Далее следует подключить класс UWatch в заголовочном файле класса, отвечающем за инициализацию интерфейса программы, в случае исходной программы – в классе UGEngineControllWidget. Вызов реализованного виджета происходит через заранее реализованную кнопку в меню приложения (Window->Watch Window).

При создании объекта класса UWatch происходит автоматическое создание первой вкладки, а в этой вкладке создание единственного графика. Для создания новых вкладок и удаления текущих реализованы методы createTab() и deleteTab(), для вызова окон настроек графиков и серий – методы on_actionSeries_option_triggered() и on_actionCharts_option_triggered() , для сохранения скриншота вкладки – метод on_actionTake_screenshot_triggered(), и метод slotCustomMenuRequested() , который реализует контекстное меню. Основные действия для работы с вкладками представленные на Рисунок 2.

Рисунок 2- Инициализация виджета.

При создании объекта класса UWatchChartOption вызывается диалоговое окно для настройки расположения графиков и их параметров – названия, подписи и интервалы видимости по осям. Для изменения количества и расположения графиков на вкладки используются методы createSingleLayout(), createRowLayout(), createColLayout(),

createGridLayout(). Возможные сценарии настройки графиков, представленные на Рисунок 3.

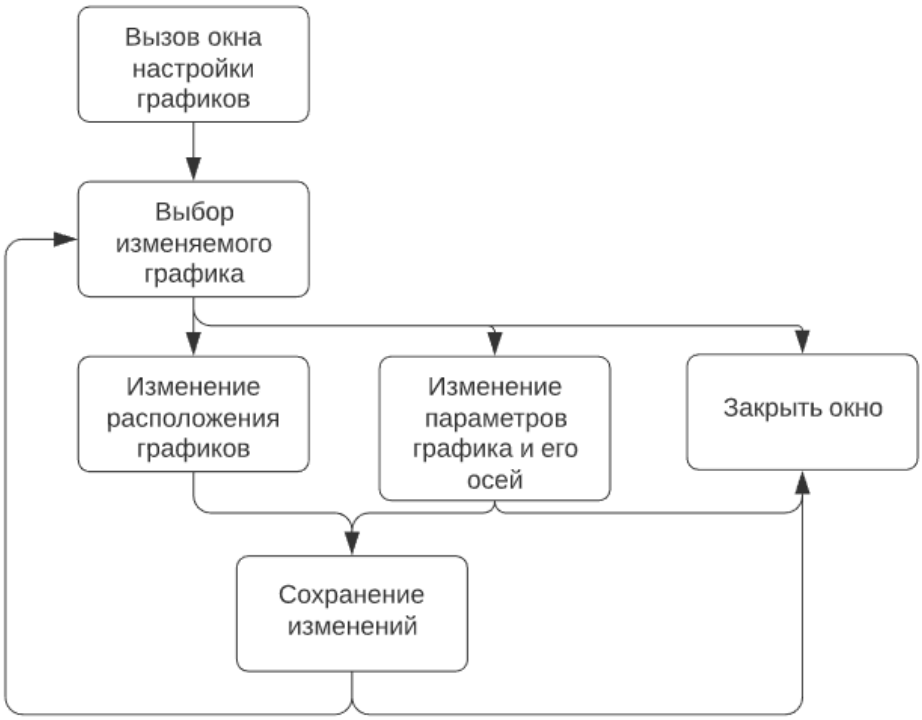

Рисунок 3– Сценарии окна изменения графиков.

Следующим шагом после настройки графиков идет добавление серии данных. Серия данных из себя представляет структуру, состоящую из информации об источнике данных, и методы графического отображения серии, реализованные в стандартном для Qt классе QLineSeries. Добавление серии в график происходит через метод createSerie() класса UWatchChart, в параметрах метода следует передать всю информацию об источнике данных. Данный метод используется только из диалогового окна – объекта класса ComponentPropertySelectionWidget, которое можно вызвать из настройки серий данных или контекстного меню конкретного графика. Для изменения или получения параметров графического отображения серии в классе UWatchChart существует широкий перечень методов доступа к параметрам конкретной серии. Варианты событий при изменении параметров серий представлены на Рисунок 4.

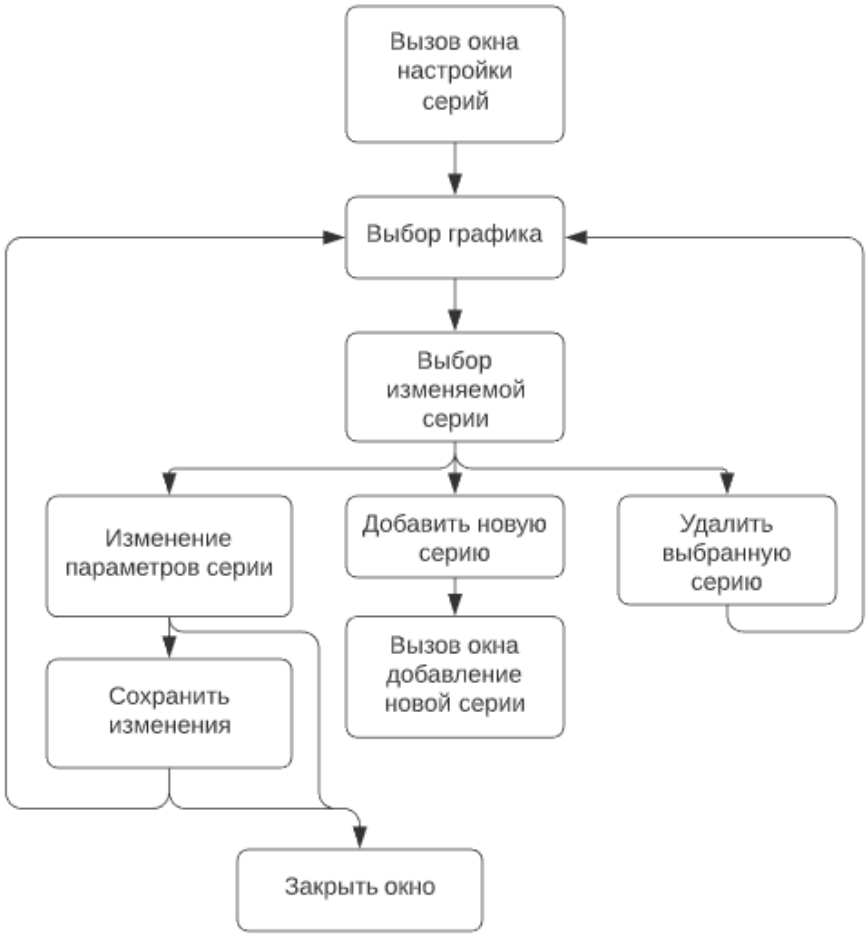

Рисунок 4– Сценарии окна изменения серий.

Для получения данных о сериях и имен графиков реализованы методы loadGraphsName и updateSeriesProperty, для сохранения изменений в сериях – метод saveParameters.

2.4 Графический интерфейс пользователя

Графический интерфейс выполнен с помощью стандартных элементов библиотеки Qt. Дизайн интерфейса соответствует основному стилю приложения и является интуитивно понятным для пользователя.

Пример основного окна виджета представлен на Рисунок 5.

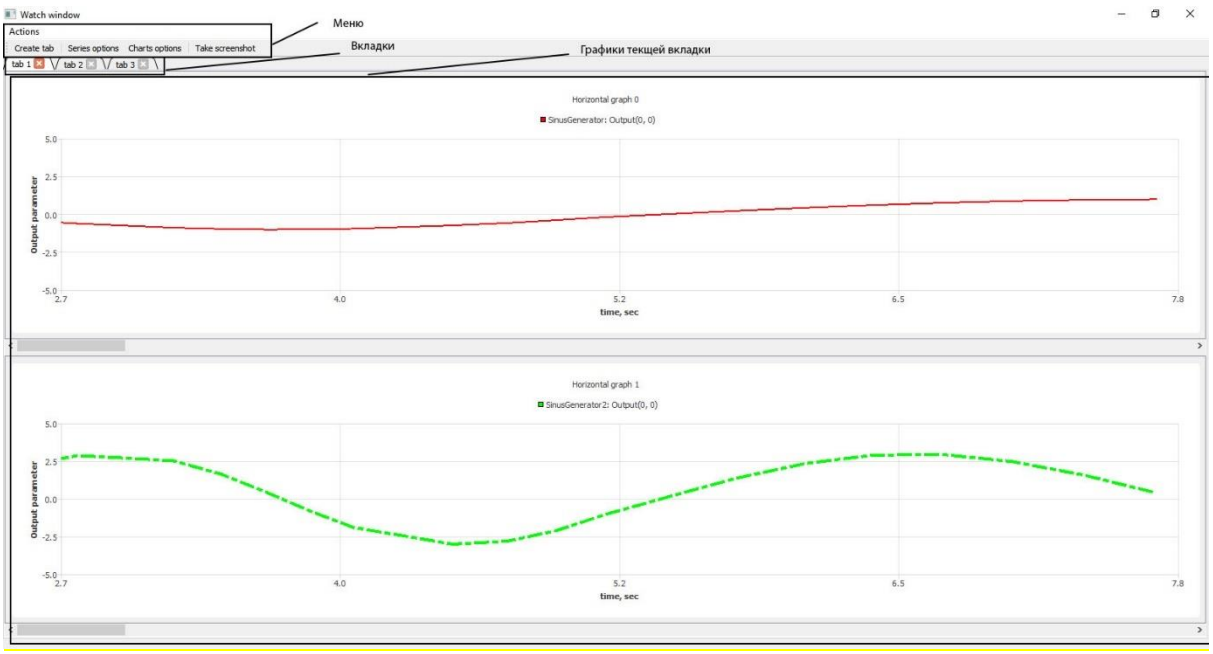

Рисунок 5– Основное окно виджета.

Каждый график имеет элементы, представленные на Рисунок 6.

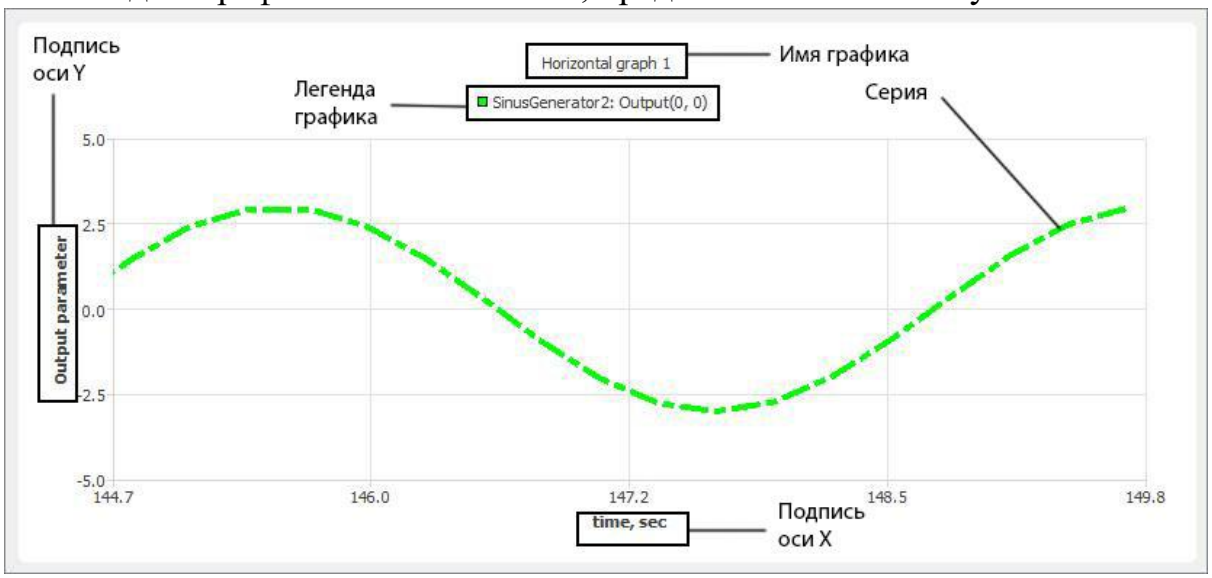

Рисунок 6 – График и его элементы.

Помимо основного виджета рассмотрим диалоговые окна настроек. Окно настройки параметров графиков и их расположения представлено на Рисунок 7.

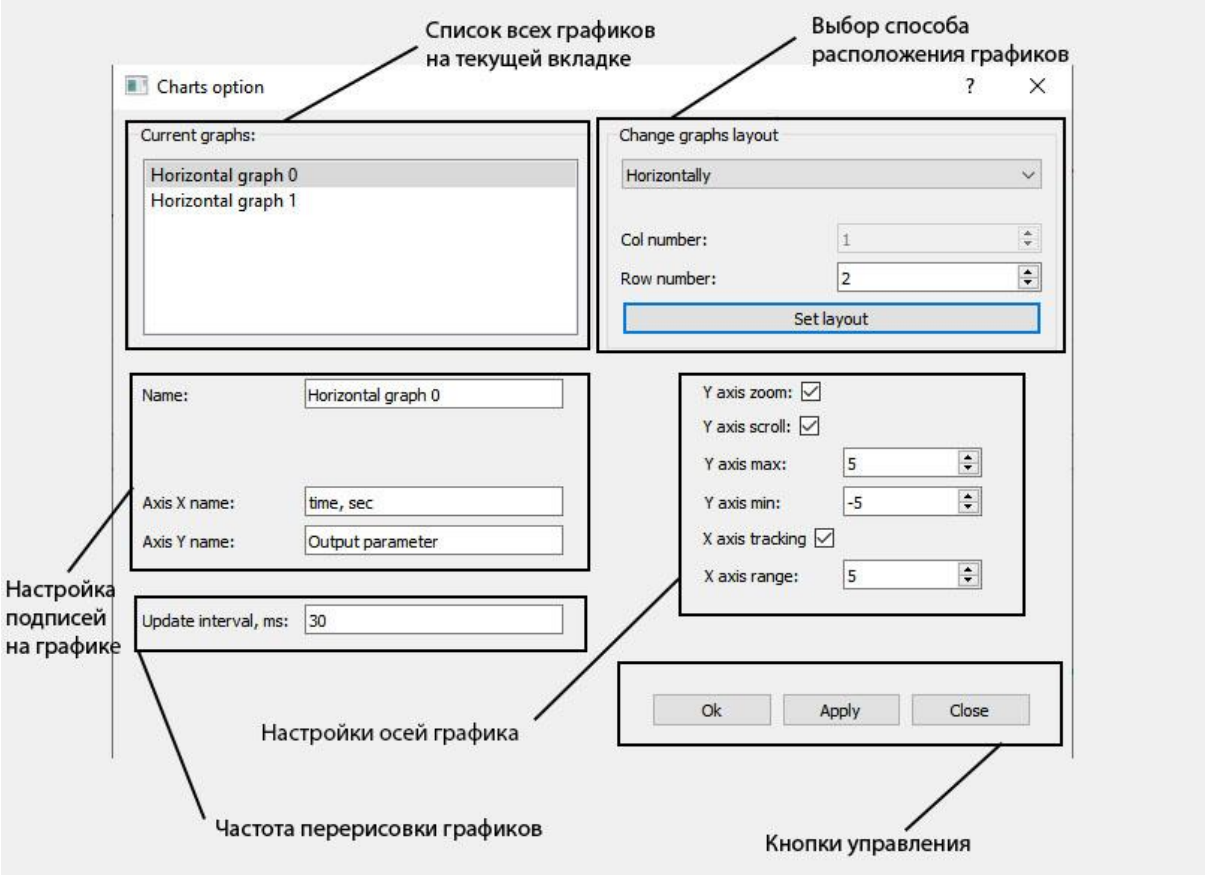

Рисунок 7– Окно настройки серий.

Окно настройки параметров серий, интерфейс добавления новых и удаления имеющихся изображен на Рисунок 8.

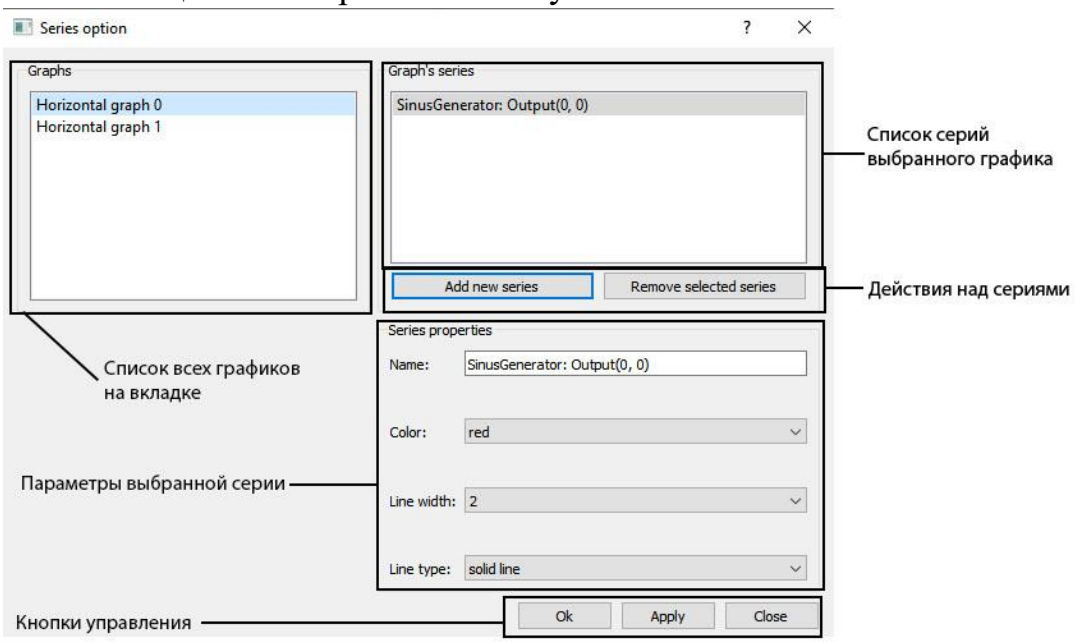

Рисунок 8– Окно настройки графиков.

Диалоговое окно, отображающее список компонентов моделей и их свойств, было реализовано в основном приложении, пример этого окна представлен на Рисунок 9.

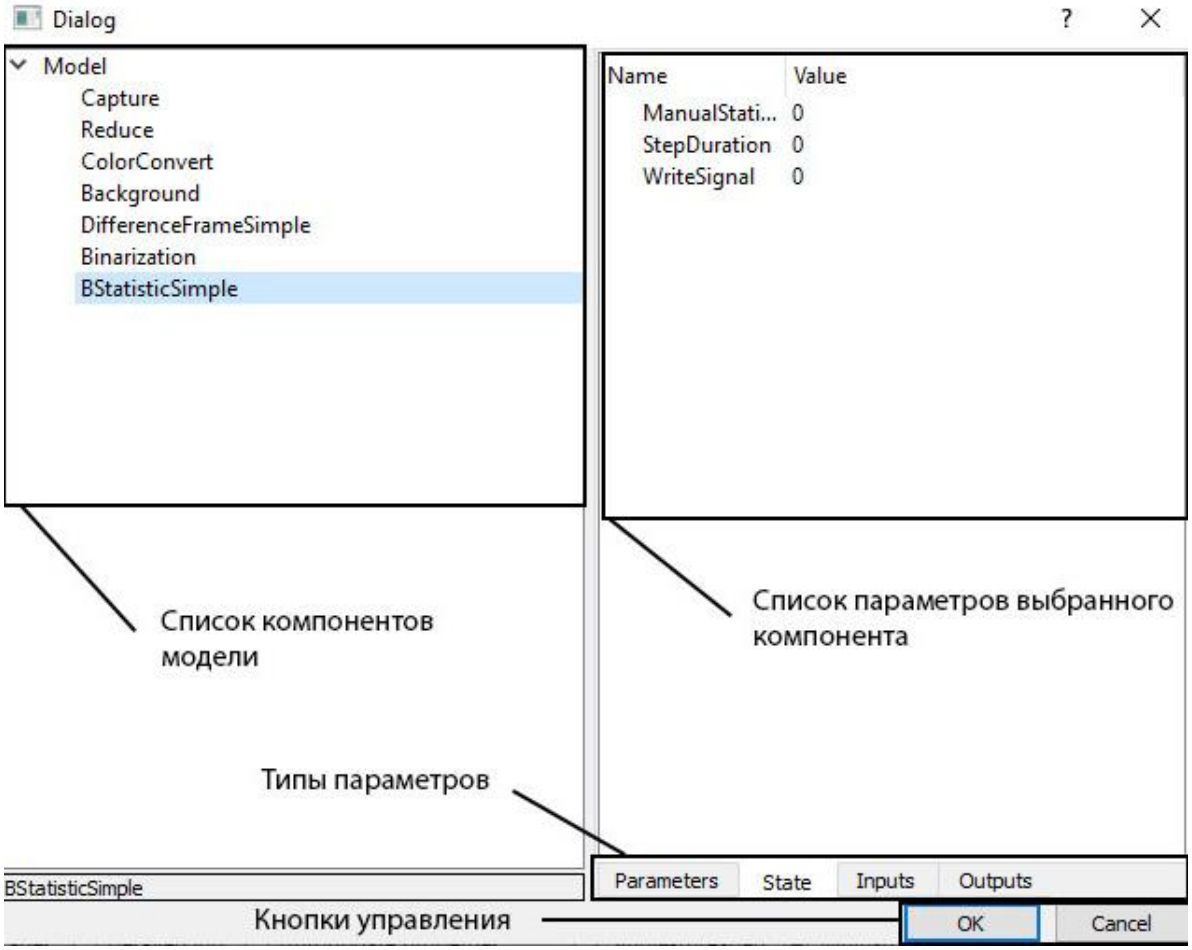

Рисунок 9– Окно со списком компонентов.

Для удобства пользователя было реализовано popup-меню, с довольно скромным функционалом, данное окно представлено на Рисунок 10.

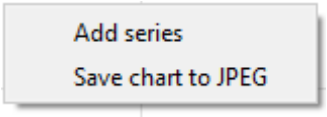

Рисунок 10 – Контекстное меню.

Заключение

В ходе работы была изучена программа для расчёта адаптивных моделей ядра RDK, принцип ее работы и особенности. В процессе работы был познакомился и освоил фреймворк для создания кроссплатформенных приложений Qt.

Был разработан виджет для построения пользовательских графиков, данный виджет соответствует всем пунктам технического задания, является интуитивно понятным для пользователя и соответствует общему стилю приложения. Разработанный виджет был встроен в основную программу и не вызывает конфликтов с другими частями пользовательского интерфейса.

В ходе написания отчета, была составлена документация на разработанные программные модули, которая содержит описание классов модулей и их компонентов, UML-схема классов, способ их подключения в программе, взаимодействия классов между собой и описание графического интерфейса пользователя.

Список литературы

1. Qt [Электронный ресурс]
   URL: https://www.qt.io
2. Бахшиев А.В., Михайлов В.В.: Программный комплекс для разработки систем технического зрения и принятия решений с динамической модульной архитектурой.