СОДЕРЖАНИЕ

Определения и сокращения ……. 6
Введение …… 7
1 Аналитический обзор средств, методов и технологий по теме дипломного проекта ……... 8
1.1 Описание проблемной области ……..... 8
1.2 Обзор программных средств управления освещением …….. 16
1.3 Обоснование выбора языка и средств разработки …….. 24
1.4 Спецификация требований ……..... 28
2 Моделирование предметной области ……..…….... 29
2.1 Диаграмма вариантов использования ……..... 29
2.2 Диаграмма развертывания ……..... 29
2.3 Диаграмма потоков данных …….. 31
2.4 Анализ требований ……. 33
3 Проектирование программного средства …….. 38
3.1 Разработка архитектуры ПС ……..... 38
3.2 Разработка схемы алгоритма работы с программой …….. 42
3.3 Разработка алгоритма отправки общегородских команд ….. 42
3.4 Разработка алгоритма отслеживания состояния устройства. 46
4 Разработка программного средства ……... 48
4.1 Используемые средства и модули …… 48
4.2 Структура программного средства …….. 48
5 Тестирование программного средства ……..… 53
5.1 Виды тестирования ……. 53
5.2 Тестирование приложения в режиме конфигурирования ….. 55
5.3 Тестирование приложения в режиме реального времени ….. 58
6 Инструкция по установке и использованию …….. 62
6.1 Установка ……. 62
6.2 Режим реального времени …….. 62
6.3 Режим конфигурирования …….. 69
7 Технико-экономическое обоснование дипломного проекта …… 72
7.1 Характеристика программного продукта ……. 71
7.2 Расчет затрат и отпускной цены …… 71
7.3 Расчет стоимостной оценки результата …… 76
7.4 Расчет показателей эффективности использования программного продукта …….. 77
Заключение …..…… 80
Список использованных источников ……..... 82
Приложение А Текст программы ……... 83

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Инициализация – приведение областей памяти в состояние, исходное для последующей обработки или размещения данных.

Программа – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программирование – научная и практическая деятельность по созданию программ.

Программный модуль – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

Программное обеспечение – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

Спецификация программы – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

ООП – объектно-ориентированное программирование.

UI – (User Interface) – Пользовательский интерфейс.

ОС – Операционная система.

ПО – Программное обеспечение.

ПС – Программное средтсво.

CLR (Common Language Runtime) – Общеязыковая исполняющая среда.

MVVM (Model view viewmodel) – Модель, представление, модуль представления.

API (Application Programming Interface) – Интерфейс программирования приложений.

ВВЕДЕНИЕ

В данной дипломной работе рассмотрены вопросы автоматизации и диспетчеризации управления городским наружным освещением. Работа носит учебно-исследовательский характер и основана на реальном проекте автоматизации управления городским освещением управления перспективных разработок в ОАО «БелЭлектроМонтажНаладка».

Построение надёжных автоматизированных систем контроля и управления наружным освещением на основе единого системного подхода (и с применением новых информационных технологий) позволит решить многие текущие и будущие проблемы отечественных промышленных предприятий.

Наружное освещение является существенным фактором нормального функционирования различных форм человеческой деятельности и жизнеобеспечения.

На сегодняшний день важно не только использовать современное электрооборудование для обеспечения городского освещения, но и применять автоматизированные средства для контроля и управления городским освещением.

В настоящем дипломном проекте задача контроля и управления освещением будет решена посредством разработки программного средства для централизованного управления освещением города на базе устройства РУНО-3 и обеспечение на этой основе оперативности и документальной подтвержденности решений, принимаемых инженерно-техническим персоналом.

Две основные функции, возлагаемые на программные средства данного типа:

– сбор данных о контролируемом технологическом процессе;

– управление технологическим процессом, реализуемое ответственными лицами на основе собранных данных и правил (критериев), выполнение которых обеспечивает наибольшую эффективность и безопасность технологического процесса.

Таким образом, разрабатываемое программное средство должно собирать информацию о технологическом процессе, обеспечивает интерфейс с оператором, сохраняет историю процесса и осуществляет автоматическое управление процессом в том объеме, в котором это необходимо.

.

1 АНАЛИТИЧЕСКИЙ ОБЗОР СРЕДСТВ, МЕТОДОВ И ТЕХНОЛОГИЙ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТА

1.1 Описание проблемной области

1.1.1 Телемеханические и диспетчерские системы управления

Автоматизированная система управления (АСУ) – это система «человек-машина», обеспечивающая эффективное функционирование объекта, в которой сбор, передача и обработка информации, необходимой для реализации функций управления, осуществляются с применением средств автоматизации и вычислительной техники.

В сложных системах полная автоматизация управления предприятием (или его отдельным департаментом) обычно трудно реализовать из-за отсутствия аналитического аппарата управляющих процессов, а также непредсказуемости всех возможных режимов работы. Поэтому наряду с устройствами автоматизации и телемеханики определённые функции выполняет исключительно человек, при этом система управления превращаются в автоматизированную систему диспетчерского управления (АСДУ).

Эти диспетчерские системы управления отличаются от соответствующих систем автоматизации в первую очередь превалирующей ролью человека (диспетчера) в контуре управления. Приёмо-передача сигналов управления осуществляется диспетчером с помощью специально организованных каналов и линий связи. С помощью средств телемеханики диспетчер получает информацию о параметрах режима освещения, электропотребления, состояний ремонта и аварийный ситуаций. С помощью этих устройств осуществляется передача управляющих команд с диспетчерского пункта на объекты.

Режимы работы отдельных элементов в системе управления наружным освещением (СУНО) взаимосвязаны. Согласованное действие всех этих элементов будет обеспечено лишь в случае, если важнейшие из них обладают устойчивыми операциями контроля и управления, сосредоточенные в одном месте (диспетчерском пункте).

В простейшем случае диспетчеризация управления может осуществляться с помощью телефонной связи диспетчера с обслуживающим персоналом удалённых объектов. При телефонной связи диспетчера с контрольными пунктами получается значительный промежуток времени с момента, требующего оперативного вмешательства до момента исполнения. Кроме того, при диспетчеризации только посредством телефонной связи велика вероятность неполучения или недостоверности информации.

Работа диспетчера оказывается болеё эффективной, если информация о режимах работы элементов системы автоматически приходит от приборов, установленных на диспетчерских пунктах. Кроме того, сам диспетчер имеёт возможность изменить режим работы управляемой системы, непосредственно посылая сигналы на контролируемые объекты.

Если контрольных пунктов мало, а расстояние между диспетчерскими пунктами значительно, то можно использовать дистанционное управление. Для этого необходимо перенести аппаратуру управления и сигнализации со щитов местного управления на центральный диспетчерский пункт (ЦДП) (Рисунок 1.1). В случаях большого расстояния между диспетчерскими и контрольными пунктами необходимо использовать устройства телемеханики. Они не требуют постоянного дежурного персонала и позволяют использовать управляющую вычислительную машину.

Отдельной задачей СУНО является операция, выполненная с помощью технических средств и программного обеспечения, в результате решения которой формируются либо отчетный документ, либо одно или серия однотипных сообщений обслуживающему персоналу.

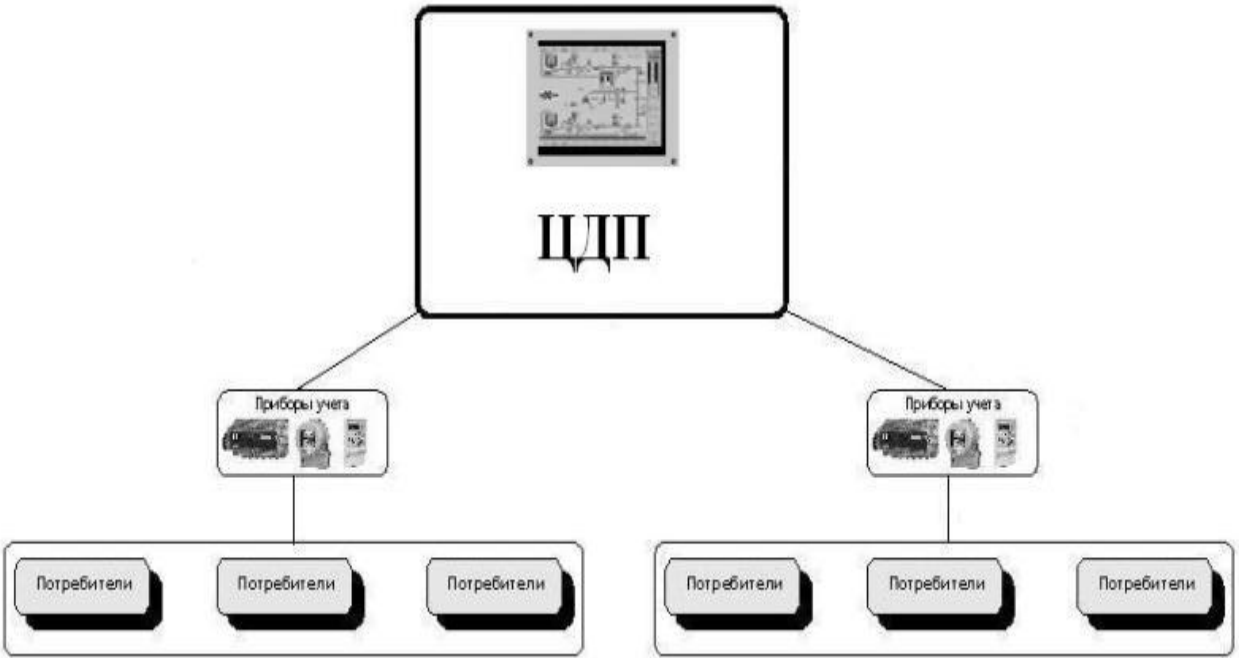

Рисунок 1.1 – Диспетчерская система управления СУНО

Телеуправление – управление положением или состоянием объектов методами и средствами телемеханики. Телеуправление предприятиями применяется тогда, когда это дает возможность улучшить ведение режима и позволяет ускорить локализацию и ликвидацию аварии, нарушение и отклонение от нормальных режимов работы, если это невозможно сделать с помощью местной автоматики.

Телесигнализация (ТС) – это получение информации о состоянии контролируемых и управляемых объектов, имеющих ряд возможных дискретных состояний. ТС должна обеспечивать передачу на пульт управления предупреждающих и аварийных сигналов, а также обеспечивать отображение состояние основных элементов СУНО на диспетчерском пульте.

Телеизмерения (ТИ) – должны обеспечивать возможность измерения

основных параметров, отображающих работу системы и позволяющих правильно управлять ситуацией. Для телеизмерений в СУНО рекомендуют выбирать:

– текущие напряжения на осветительных линиях;

– сила тока на осветительных линиях;

– суммарную мощность, потребленную за требуемый промежуток времени;

Телеизмерения тока и напряжения организуются по вызову, а мощности – по циклическому типу в течение суток. Телеизмерения интегральных параметров (ТИИ) обеспечивают возможность составления энергетических балансов. Кроме того, они используются постоянно для ввода результатов измерений в вычислительную информационную сеть.

Телеизмерения текущих параметров (ТИТ) – должны обеспечивать диспетчеру возможность измерения основных электрических параметров, необходимых для управления системой и восстановления её после аварии.

Телемеханизация (ТМ) должна обеспечивать:

1) отображение на диспетчерском пульте состояний и основных элементов;

2) передача на диспетчерский пульт предупреждающих и аварийных сигналов;

3) управление основными элементами системы и т.п.

В связи с постоянным удорожанием потреблённой электроэнергии и необходимости модернизации производственных мощностей (и их систем автоматизации) у промышленных предприятий возникла необходимость в построении интегрированных решений, в разработке автоматизированных систем контроля и управления электропотребления (АСКУЭ), построенных с применением персональных ЭВМ.

1.1.2 Структура АСКУЭ, построенная с применением ПЭВМ

В числе главных проблем, возникающих при создании АСКУЭ предприятия – оптимальное разделение функций между универсальными и специализированными средствами. Это в конечном итоге определяет конкретный выбор технических средств, суммарные затраты на создание АСКУЭ, её эксплуатацию и достигаемую эффективность.

Одна крайность при решении указанной проблемы заключается в перенесении почти всех функций АСКУЭ на ЭВМ. Полная централизация сбора и обработки измерительных данных на ЭВМ – приводит к уменьшению затрат на специализированное оборудование, но одновременно и к увеличению затрат на кабели связи, снижению надёжности и живучести системы в целом, а также делает проблематичной её метрологическую аттестацию. Другая крайность – построение АСКУЭ исключительно на базе специализированных средств. В данном случае достигается экономия кабельной продукции, успешно решаются вопросы метрологической аттестации, обеспечивается децентрализованный доступ к информации, но

снижается эффективность АСКУЭ в целом за счет ограничения функций систем в плане полноты накопления данных, их обработки, отображения, документирования и анализа информации.

Оптимальный подход при создании АСКУЭ предприятия состоит в согласованном выборе специализированных и универсальных средств с учётом их функций. При этом типовая структура централизованной АСКУЭ предприятия включает, как специализированные системы, так и ПЭВМ (Рисунок 1.2). Устройства сбора и передачи данных (УСПД) выполнены в виде микропроцессорных средств и предназначены для экономии кабельной продукции, а также для контроля каналов связи. Структура АСКУЭ конкретных предприятий отличаются количеством и типом систем, средствами связи, но для всех АСКУЭ характерны взаимозависимость функций ПЭВМ и систем.

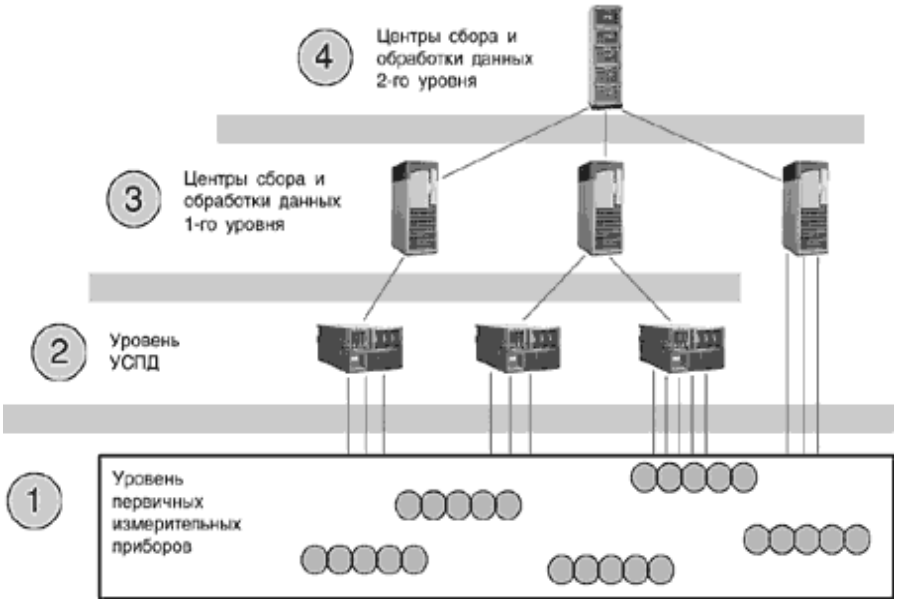

Рисунок 1.2 – Типовая структура централизованной АСКУЭ

Современные специализированные информационно-измерительные системы автоматизированного электроснабжения характеризуются определенным числом измерительных каналов и групп учёта, а также списком штатных энергетических (мощность, расход) и сервисных (неработающие каналы, сбои питания и т.п.) параметров. В группу алгебраически суммируются данные определенных измерительных каналов одного вида учёта (точки учёта) в соответствии со схемой конкретного предприятия. По соответствующей группе и (или) каналу система за определённые интервалы времени накапливает информацию о фактических расходах энергии или энергоносителей (электроэнергии, холодной и горячей воды, пара, газа, воздуха и др.).

Перечень интервалов накопления информации о расходах определяется:

1) требованиями коммерческого учёта в соответствии с действующими и перспективными тарифами;

2) требованиями технического учёта, то есть задачами оперативного прогнозирования и управления нагрузкой;

3) требованиями контроля за показателями электроэнергии и т.п.

Поэтому диапазон интервалов содержит, как правило, интервал краткосрочного накопления (1 – 3 мин), интервалы среднесрочного (30 мин, зоны и смены суток, сутки) и долгосрочного (неделя, декада, месяц, квартал, год) накоплений. Данные о расходах электроэнергии и энергоресурсов в указанных интервалах используются помимо своего прямого назначения и для расчётов мощностей или удельных расходов, а также могут быть использованы в контуре экономического энергопотребления (в задачах АСКУЭ).

Основную информацию о процессах электропотребления предприятия получают на основе изучения комплекса графиков и диаграмм, отражающих в интегральном виде характер и динамику процессов на различных объектах (или их группах) системы электроснабжения предприятия. Указанные графики и диаграммы желательно иметь если не по каждой группе или каналу учёта, то по большинству точек учёта, причём в режиме сопоставления их друг с другом (например, суточный график нагрузки нескольких цехов на фоне графика нагрузки предприятия в целом и т.п.) и с возможностью выбора за любой среднесрочный или долгосрочный интервал текущего года.

Основным видом энергетических параметров для АСКУЭ являются не графики нагрузок, а текущие итоговые суммы расходов и мощностей. Поэтому сбор информации для вышеперечисленных графиков и её накопление (архивирование) являются задачами программного комплекса АСКУЭ верхнего уровня.

Периодичность процесса сбора данных в ПЭВМ с систем нижнего уровня определяется, с одной стороны, срочностью решаемой задачи верхнего уровня, а с другой – списком параметров систем. Для согласования времени принятия решения на разных уровнях управления применяются промежуточные системы человеко-машинного интерфейса (SCADA-системы).

Рассмотрим основную структуру диспетчерского управления и автоматизации системы электроснабжения.

1.1.3 Интегрированные системы управления и автоматизация СУНО

В современных условиях в электроэнергетике происходит постепенное слияние различных систем автоматизации: АСКУЭ, АСДУ и АСУ ТП, и создание на их базе интегрированных автоматизированных систем управления (ИАСУ).

Интегрированные АСУ – это логическое продолжение вертикальной интеграции АС на разных этапах производства (потребления) электроэнергии. Основная цель создания таких систем – дальнейшеё повышение эффективности технических и программных средств

автоматизации и диспетчеризации СЭС для улучшения технико-экономических показателей и повышения качества и надёжности электроснабжения ПП.

Реформирование электроэнергетики республики Беларусь требует создания полномасштабных иерархических систем управления: автоматических систем измерения показателей электроэнергии (АСИЭ); автоматизированных систем учёта потребления и сбыта электроэнергии (АСУПСЭ), АС диспетчерского управления (АСДУ), АС контроля и управления электропотреблением (АСКУЭ).

Основная особенность экономического метода управления – рассмотрение электропотребления как главного звена, управляющего рынком электроэнергии. Этот рынок, в свою очередь, представляет совокупность собственно технологического процесса (производства, передачи, распределения и потребления электроэнергии), учётно-финансового процесса электропотребления, а также политических процессов в государстве и обществе. Эти факторы должны являться основой для создания и развития рынка электроэнергии в России. Причём функционирование такого рынка не возможно без создания интегрированной системы управления электропотреблением на базе систем АСИЭ, АСУПСЭ, АСДУ и АСКУЭ. При этом возникает необходимость чётко разграничить функции указанных систем в рамках единой системы управления энергопотреблением.

Интегрированная система управления электропотреблением в условиях рынка должна охватывать все уровни и стадии управления от производства до реализации, от физических потоков электрической энергии до финансовых и экономических показателей электропотребления (таблица 1.1).

Управление на каждом уровне должно осуществляться соответствующими системами, что обусловлено спецификой выполняемых ими функций (в зависимости от стадии управления) и находит подтверждение в теории и практике создания больших информационно-управляющих систем.

В рамках интегрированной системы АСКУЭ должны быть задействованы различные по функциональному назначению технологические системы, образующие иерархию экономического управления электропотреблением.

Такая единая система АСКУЭ поможет реализовать управление электропотреблением экономическим методом:

1) долгосрочное и краткосрочное планирование режимов (кривой) потребления – на основе экономических параметров электропотребления потребителей и поставщиков (от АСУПСЭ) и действующей системы тарифов на электроэнергию (отражающей внешнеё, косвенное, воздействие на управление потреблением);

2) контроль режимов электропотребления – по параметрам потребления, используемым для расчётов с потребителями (от АСИЭ);

3) принятие управленческих решений по регулированию потребления

и доведение их до системы управления производством и распределением энергии.

Таблица 1.1 – Структура (иерархия) управления электропотреблением

Уровни управления	Стадии управления
1. Внешний (старший) уровень управления.	1. Директивное и косвенное управление.
2. Уровень экономики.	2. Управление (планирование и контроль) экономическим методом.
3. Уровень потребления.	3. Учёт (накопление) экономических параметров для расчёта с потребителями.
4. Уровень присоединения.	4. Измерение (контроль) параметров для расчёта с потребителями.
5. Уровень процесса.	5. Измерение (контроль) технических параметров.

Система АСДУ осуществляет управление на технологическом уровне (уровне процесса и уровне присоединения). Её основными функциями являются:

1) управление и регулирование потреблением на основе исполнения команд системы экономического управления (АСКУЭ) либо посредством исполнения директив внешнего уровня;

2) обеспечение надёжного электроснабжения посредством автоматического измерения (контроля) технических параметров электроэнергии (/, U. W, P) и автоматической коммутации цепей и генерирующих мощностей либо посредством исполнения старшего директивного уровня управления.

Система АСУПСЭ выполняет функции:

1) учёт и накопление экономических параметров потребления – потреблённой энергии и мощности; соответствующих им стоимости и фактической оплаты;

2) взаиморасчёты через выставление платежей непосредственно с поставщиками и потребителями, а также с финансовыми учреждениями для контроля оплаты;

3) подготовку исходной информации об экономических параметрах электропотребления со стороны потребителей и поставщиков для принятия решений.

Система АСИЭ осуществляет измерение и контроль параметров электропотребления для расчётов с потребителями (потреблённой энергии и мощности).

АСИЭ выполняет измерение параметров энергопотребления в точках присоединения потребителей и поставщиков. АСУПСЭ осуществляет преобразование и группировку параметров потребления электроэнергии в экономические параметры конкретных потребителей и поставщиков,

выставление счетов и контроль оплаты, их учёт (накопление) и анализ.

АСДУ является исполнительным органом, осуществляющим непосредственное управление (по командам системы управления экономического уровня) коммутацией цепей и генерацией мощностей, т.е. на уровнях процесса и присоединений.

АСКУЭ должна выполнять одновременно две функции:

1) оперативный контроль и управление по выдерживанию заданного режима (кривой) потребления;

2) формирование нового оптимального режима потребления на основе фактических экономических параметров потребления и тарифов на электроэнергию, а при необходимости – управление переходом на новый режим потребления.

Интегрированные организационно-технологические АСУ энергосистемами создаются на базе функционирующих АСУ как естественное их развитие и характеризуются рядом особенностей, в частности наличием: многомашинного оперативного информационного управляющего комплекса (ОИУК); системой сбора оперативно-диспетчерской и организационно-экономической информации; разветвлённой сетью периферийных пунктов сбора и обработки информации; АСУ различного назначения, автоматизированных систем диспетчерского (АСДУ) и организационно-экономического управления (АСОУ), АСУ технологическими процессами, АСУ энергетическими компаниями и предприятиями.

К объективным трудностям создания такой единой системы АСКУЭ можно отнести продолжающийся процесс реформирования электроэнергетики, только формирующийся рынок электроэнергии, недостаточность правовой базы и отсутствие достаточных инвестиций в отрасль.

1.1.4 SCADA-системы управления

SCADA (аббр. от англ. Supervisory Control And Data Acquisition – диспетчерское управление и сбор данных) – это программное средство, предназначенное для обеспечения работы в реальном времени систем сбора, обработки и отображения информации об объекте мониторинга или управления. Программное средство может являться частью АСУ ТП, АСКУЭ, системы экологического мониторинга, научного эксперимента, автоматизации здания и т. д. Данный тип приложений широко используется во многих отраслях хозяйства, где требуется обеспечивать операторский контроль за технологическими процессами в реальном времени. Данное программное обеспечение устанавливается на компьютеры и, для связи с объектом, использует драйверы ввода-вывода.

Выделим основные особенности средств диспетчерского контроля:

1) обмен данными с «устройствами связи с объектом» (то есть с промышленными контроллерами и платами ввода-вывода) в реальном времени через драйверы;

2) обработка информации в реальном времени;

3) логическое управление;

4) отображение информации на экране монитора в удобной и понятной для человека форме;

5) ведение базы данных реального времени с технологической информацией;

6) аварийная сигнализация и управление тревожными сообщениями.

7) подготовка и генерирование отчетов о ходе технологического процесса;

8) обеспечение связи с внешними приложениями (СУБД, электронные таблицы, текстовые процессоры и т. д.).

На основе выше перечисленного можно выделить основные требования к программному средству управления городским освещением. Подобная платформа должна позволять эффективно управлять освещением города. Это должно достигаться за счет эффективного проектирования и разработки.

Основные функции ПС:

1) сбор информации с контроллеров нижнего уровня;

2) выдачу команд управления для контроллеров нижнего уровня;

3) регистрацию и хранение технологических параметров;

4) самоконтроль и диагностику, вывод информации о техническом состоянии обслуживающему персоналу;

5) функции пульта ручного управления и индикации;

6) функция управления описаниями устройств.

1.2 Обзор программных средств управления освещением

1.2.1 Система централизованного управления освещением города Риги

Типовой системой автоматического управления уличным освещением города является система централизованного управления освещением города Риги. Управление уличным освещением осуществляется централизовано с диспетчерского пульта. В качестве управляющей машины используется ПЭВМ IBM PC. Управляющая программа – типичная SCADA-система, выполняющая следующие функции:

1) автоматическое включение и выключение уличного освещения согласно введенному графику;

2) ручное включение и выключение уличного освещения согласно требованиям диспетчера;

3) селективное включение и выключение уличного освещения по разным графикам для разных районов или улиц;

4) аварийное выключение уличного освещения;

5) контроль за текущим состоянием станций управления;

6) визуализация аварийных состояний;

7) журнал событий и их архивация;

8) статистика качества связи;

9) формирование графиков включения;

10) формирование отчетов.

В настоящее время уличное освещение города Риги состоит из 65 станций управления, распределенных по городу в соответствии с его застройкой. Станция управления осуществляет контроль за питающими линиями уличного освещения в двух режимах – ночном и вечернем.

1.2.2 Система централизованного управления освещением Санкт-Петербурга

В Санкт-Петербурге повседневную работу по эксплуатации сетей наружного освещения и реконструкции электросетей в городе проводит организация ГЭСП «Ленсвет». Для улучшения службы уличного и наружного освещения специалистами «Ленсвета» была разработана целевая программа «Светлый город» (рисунок 1.3). Одним из разделов этой программы является автоматизированное управление наружным освещением. Санкт-Петербургская автоматизированная система управления наружным освещением города носит название «Аврора».

1.2.3 Система централизованного управления освещением Москвы

В Москве большое внимание уделяется не только автоматизации управления уличным освещением, но и архитектурно-художественным и садово-парковым освещением города. Созданием осветительных установок и автоматизацией управления и контроля занимается НПО «Светосервис». Работа всех структур «Светосервиса» направлена на создание комфортной световой среды: осуществляется круглосуточный надзор за состоянием уличного освещения, реализуется программа «Формирование световой среды г. Москвы», ведутся работы по архитектурному освещению в регионах России (Казань, Вологда, Воронеж, Екатеринбург и др.). Принцип работы АСУ наружным освещением такой же, как и в выше описанных примерах.

1.2.4 Система централизованного управления освещением города Минска

В Минске используется АСУ «Горсвет», предназначенная для централизованного и локального управления электрическими сетями уличного, иллюминационного и рекламного освещения. Система была разработана и внедрена специалистами БелЭлектроМонтажНаладки (Минск) и «Ленсвет»(Санкт-Петербург). В качестве технических средств использовались контроллеры собственной разработки, адаптированные для решения таких задач, а также для организации связи по телефонным линиям.

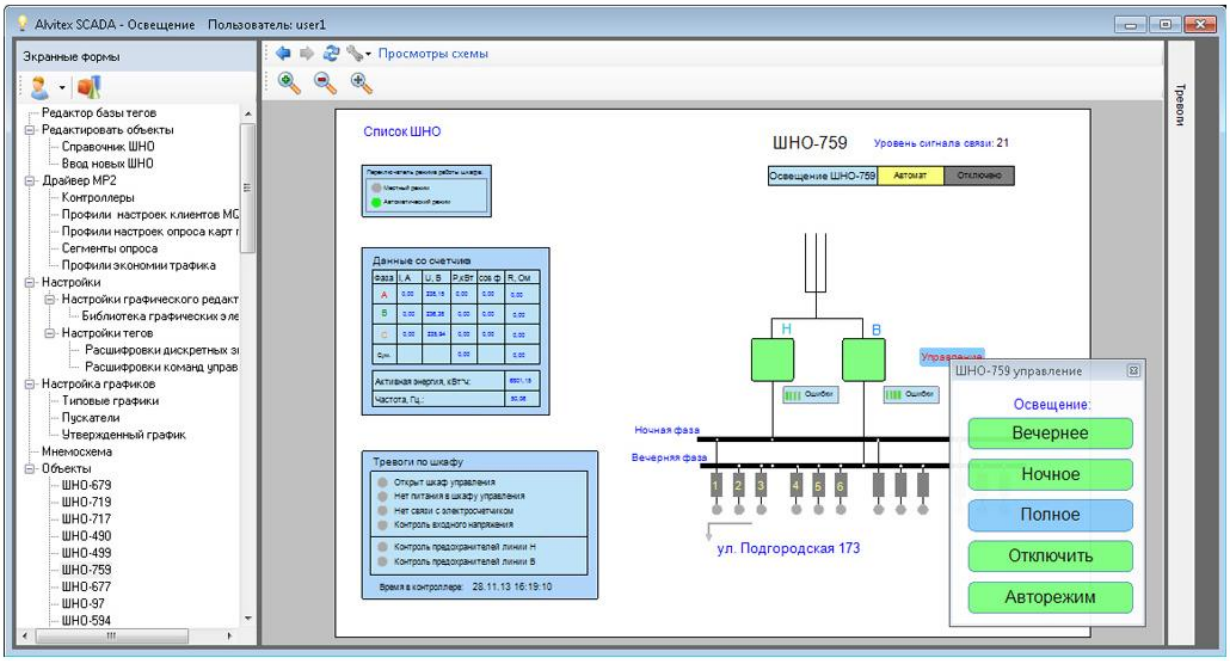

Рисунок 1.3 – Программа «Светлый город»

Обеспечивая непрерывный мониторинг состояния сети наружного освещения, система «Горсвет» позволяет:

1) оптимизировать структуру и режим управления городской сети освещения;

2) обеспечить оптимальный уровень освещенности улиц;

3) оперативно выявлять и устранять повреждения сетей освещения.

Верхний уровень системы – диспетчерский пункт управления, включающий в себя АРМ(ы) диспетчера на базе SCADA-пакета iFIX и пульт диспетчерского управления. Нижний уровень системы – контролируемые пункты (КП) головных пунктов питания (ГПП), включающие шкафы наружного освещения (ШНО) и шкафы управления (ШУ) с контроллерами «ПИКОН-2», либо реле управления наружным освещением «РУНО 3»(обзор данного устройства будет приведен в п.1.2.4.1.1). Обмен информацией между верхним и нижним уровнями управления осуществляется по выделенным телефонным линиям связи или радиоканалу, с использованием протокола «Modbus»(обзор данного устройства будет приведен в п.1.2.4.2.).

Система позволяет осуществлять каскадное подключение контролируемых пунктов (до 32-х) по выделенным линиям связи (разрабатываются технические средства для организации связи по осветительной сети). ШНО, организация связи с которыми невозможна или нецелесообразна, могут управляться реле времени с годовым графиком (РВ720).

Представление информации по каждому контролируемому пункту производится в форме таблицы(Рисунок 1.4), графической мнемосхемы (Рисунок 1.5) и в виде отображения местоположения и основных состояний объекта на карте города (Рисунок 1.6).

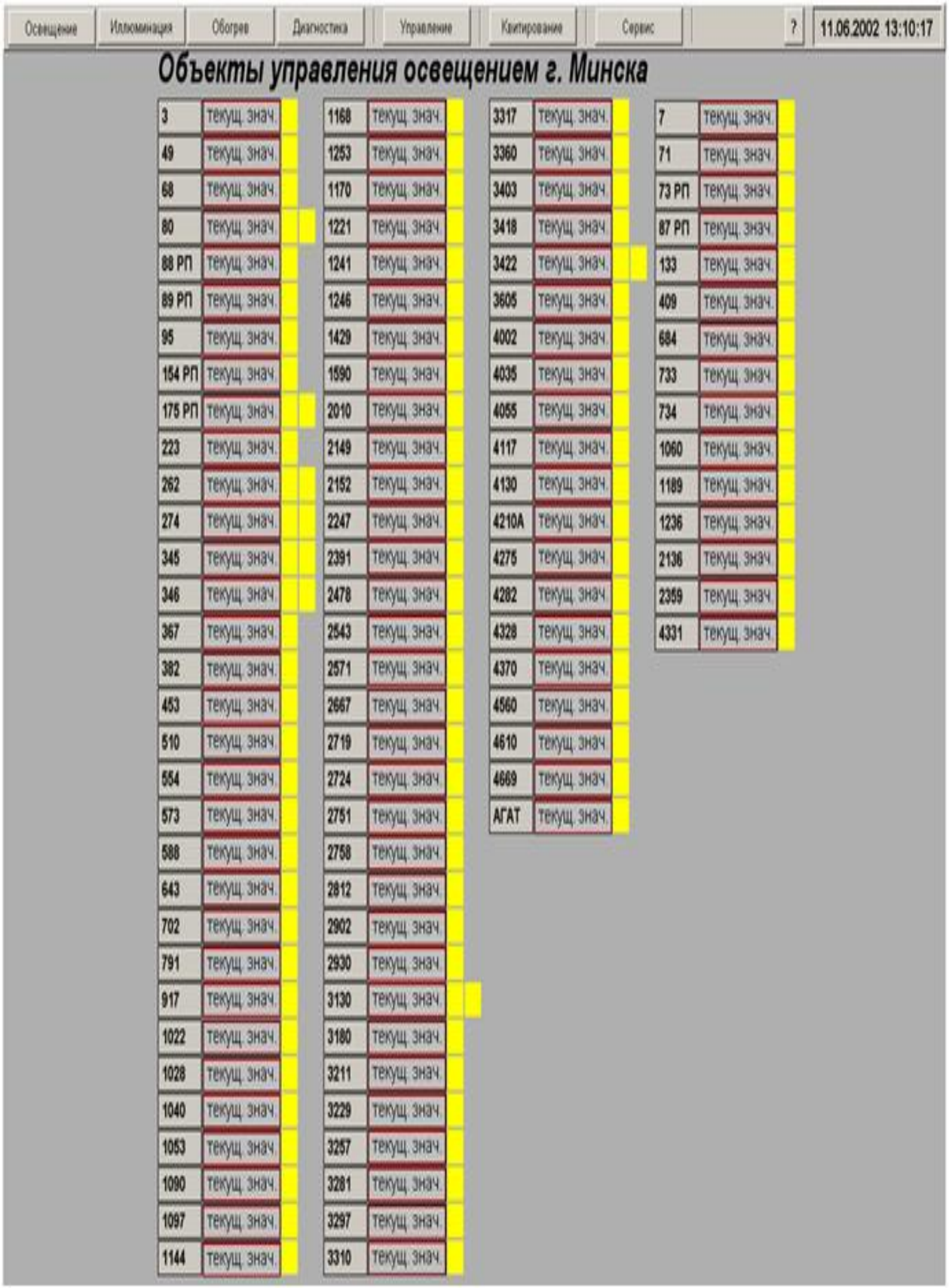

Рисунок 1.4 – Представление информации о освещение города в виде таблицы в программе «Горсвет»

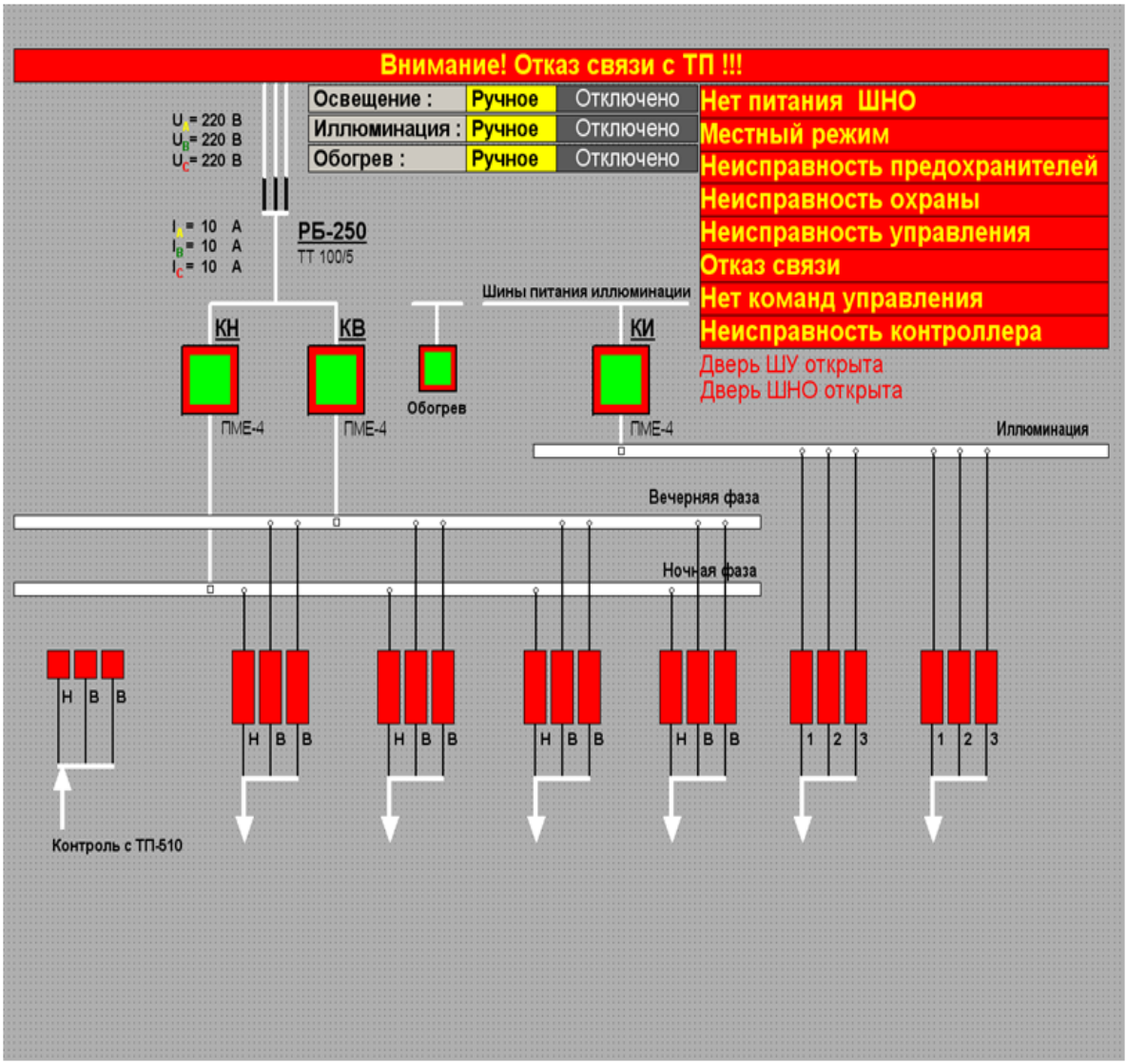

Рисунок 1.5 – Представление информации о освещение города в виде графической мнемосхемы в программе «Горсвет»

1.2.4.1 Обзор реле управления наружным освещением «РУНО 3»

Реле управления наружным освещением «РУНО 3» предназначено для автоматического управления (по годовым графикам) уличным освещением, праздничной иллюминацией и декоративной подсветкой зданий.

«РУНО 3» имеет три независимых канала управления (каждый канал с отдельным годовым графиком) включением/отключением нагрузки каждого канала 1 раз в сутки.

Выполняемые функции**:**

– управление объектом по команде с верхнего уровня и (или) автономно;

– сбор информации с объекта контроля и управления;

– возможность передачи информации на верхний уровень по GPRS каналу связи (для модификации с GSM-модулем).

При отсутствии напряжения питания встроенные часы переходят на питание от литиевой батареи или ионистора (в зависимости от модификации изделия). «РУНО 3» устанавливается в ШНО.

Для стабилизации точности хода часов в рабочем диапазоне температур предусмотрена автоматическая температурная коррекция.

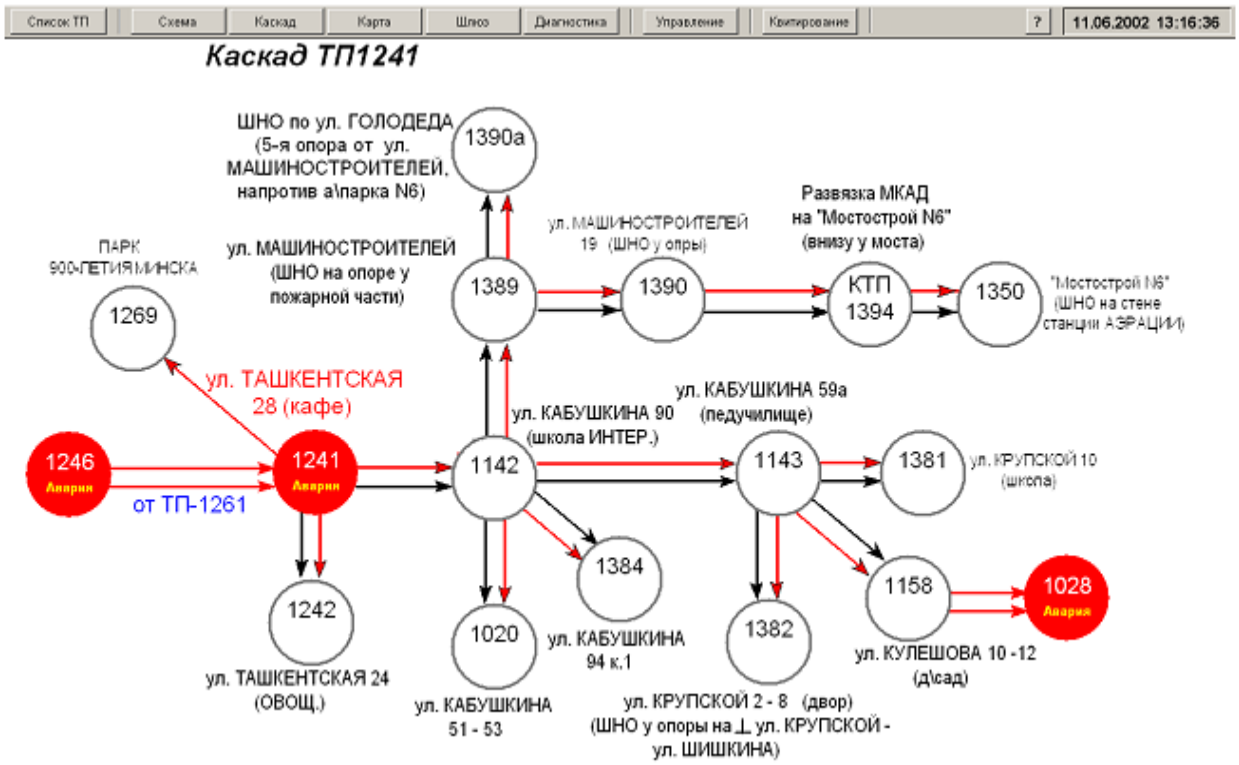

Рисунок 1.6 – Представление информации о освещение города в виде состояний объекта на карте города в программе «Горсвет»

1.2.4.2 Обзор протокола «Modbus TCP»

Протокол Modbus и одноимённая сеть являются самыми распространёнными в мире среди протоколов и сетей. Несмотря на свой возраст (Modbus стал стандартом де-факто ещё в 1979 году) он не только не устарел, но, на- оборот, демонстрирует существенно возросшее количество ориентированных на него новых разработок и увеличивающийся объём организационной поддержки протокола. Миллионы Modbus-устройств по всему миру продолжают успешно работать, обновляются версии описания протокола.

Одним из главных преимуществ Modbus является отсутствие необходимости в специальных интерфейсных контроллерах (PROFIBUS и CAN требуют для своей реализации заказных микросхем), также к преимуществам следует причислить простоту программной реализации и элегантность принципов функционирования. Всё это снижает за- траты на освоение стандарта как системными интегратора- ми, так и разработчиками контроллерного оборудования. Высокая степень открытости протокола

обеспечивается полностью бесплатными текстами стандартов, которые можно скачать с сайта www.modbus.org.

Основным недостатком Modbus является сетевой обмен по типу «ведущий–ведомый», что не позволяет ведомым устройствам передавать данные по мере их появления и поэтому требует интенсивного опроса ведомых устройств ведущим.

Разновидностями Modbus выступают Modbus Plus, представляющий собой многомастерный протокол с кольцевой передачей маркера, и протокол Modbus TCP, рассчитанный на использование в сетях Ethernet и Интернет.

Формат Modbus кадра показан на рисунке 1.7 (где PDU protocol data unit) – элемент данных протокола; ADU (application data unit) – элемент данных приложения). Поле адреса всегда (даже в ответах на команду, посланную ведущим) содержит только адрес ведомого устройства. Благодаря этому ведущее устройство знает, от какого модуля пришёл ответ.

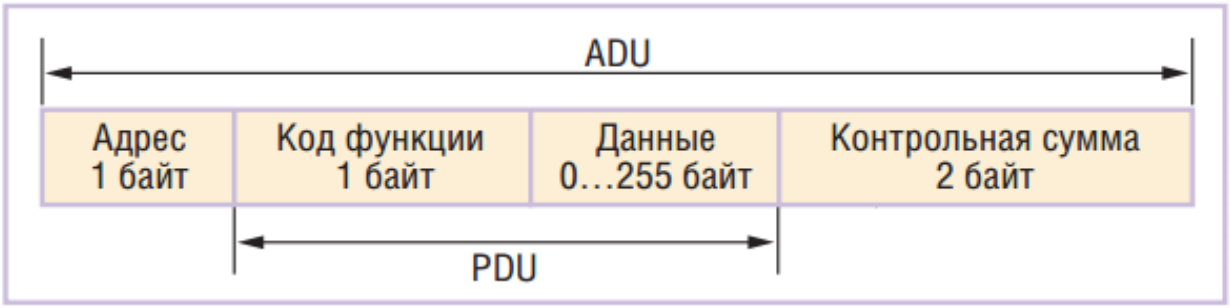

Рисунок. 1.7 – Формат кадра протокола Modbus RTU

Поле «Код функции» говорит модулю о том, какое действие нужно выполнить.

Поле «Данные» может иметь произвольное количество байтов в диапазоне от 0 до 255. В нём может содержаться информация о параметрах, используемых в запросах контроллера или ответах модуля.

Поле «Контрольная сумма» содержит контрольную сумму CRC длиной 2 байта.

Протокол Modbus TCP (или Modbus TCP/IP) используется для того, чтобы подключить устройства с протоколом Modbus к Ethernet или сети Internet. Он использует кадры Modbus RTU на 7-м (прикладном) уровне модели OSI, протоколы Ethernet на 1-м и 2-м уровнях модели OSI и TCP/IP на 3-м и 4-м уровнях, то есть Ethernet TCP/IP используется для транспортировки модифицированного кадра Modbus RTU.

Кадр Modbus RTU в этом случае не имеет поля контрольной суммы, поскольку используется стандартная контрольная сумма Ethernet TCP/IP; нет также поля адреса, поскольку в Ethernet используется иная система адресации. Таким образом, только два поля – «Код функции» и «Данные» (блок PDU) встраиваются в протокол Ethernet TCP/IP. Перед ними вставляется новое поле (Рисунок 1.8) – MBAP (Modbus application protocol –

прикладной протокол Modbus). Поле «Идентификатор обмена» используется для идентификации сообщения в случае, когда в пределах одного TCP-соединения клиент посылает серверу несколько сообщений без ожидания ответа после каждого сообщения. Поле «Идентификатор протокола» содержит нули и зарезервировано для будущих применений. Поле «Длина» указывает количество следующих за ним байтов. Поле «Идентификатор устройства» идентифицирует удалённый сервер, расположенный вне сети Ethernet (например, в сети Modbus RTU, которая соединена с Ethernet с помощью межсетевого моста). Чаще всего это поле содержит нули или единицы, игнорируется сервером и отправляется обратно в том же виде (как эхо). Изображённый на рисунке 1.8 фрейм называется фреймом ADU, встраивается в поле «Данные» фрейма Ethernet.

Таким образом, структура кадра и смысл его полей «Код функции» и «Данные» для Modbus и Modbus TCP совершенно идентичны, поэтому для работы с Modbus TCP не требуется дополнительного обучения при знании Modbus RTU. Те же самые коды функций и данные, что и в Modbus RTU, передаются по очереди с прикладного (7-го) уровня модели OSI на транспортный уровень, который добавляет к блоку PDU кадра Modbus RTU (Рисунок 1.7) заголовок с протоколом TCP. Далее новый полученный кадр передаётся на сетевой уровень, где в него добавляется заголовок IP, затем он передаётся на канальный уровень Ethernet и на физический. Дойдя до физического уровня, блок PDU оказывается «обросшим» заголовками протоколов всех уровней, через которые он прошёл. Пройдя по линии связи, сообщение продвигается снизу вверх по стеку протоколов (уровням модели OSI) в устройстве получателя, где на каждом уровне из него удаляется соответствующий заголовок, а на прикладном уровне выделяется блок PDU (код функции и данные) кадра протокола Modbus RTU.

В сети с протоколом Modbus TCP устройства взаимодействуют по типу «клиент–сервер», где в качестве клиента выступает ведущее устройство, в качестве сервера – ведомое. Сервер не может инициировать связи в сети, но некоторые устройства в сети могут выполнять роль как клиента, так и сервера.

Modbus TCP не имеет широковещательного или многоабонентского режима, он осуществляет соединение только между двумя устройствами.

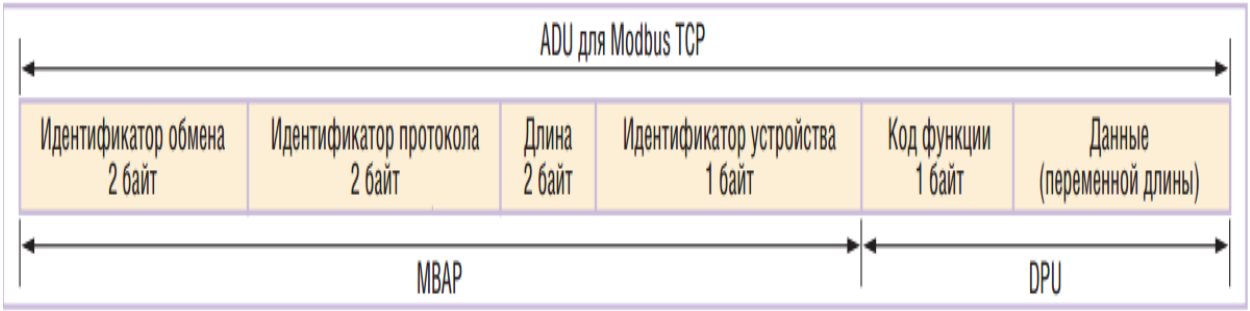

Рисунок 1.8 – Часть фрейма Modbus TCP, встраиваемая в поле «Данные» фрейма Ethernet

1.2.5 Итог обзора программных средств управления освещением

В республике Беларусь планируется развитие системы управления уличным городским освещением. В настоящий момент в республике Беларусь городское освещение автоматизировано лишь в крупных городах, в которых используется программа «Горсвет». Программа «Горсвет» не является собственной разработкой БелЭлектроМонтажНаладки, и требует дополнительной покупки лицензии. Другие рассмотренные ранее аналоги предназначены для управления контролеров отличных от тех, которые производятся в ОАО «БелЭлектроМонтажНаладка» и используются при освещении городов в республике Беларусь. Поэтому было принято решение приступить к созданию программного средства управления городским освещением, взаимодействующего с устройствами, произведенными ОАО «БелЭлектроМонтажНаладка». В связи с тем, что в малых городах управление освещением осуществляется вручную, то было принято решение разработать программное средство управления освещением для малых городов.

Разрабатываемое программное средство должно совмещать в себе основные преимущества рассмотренных аналогов и быть ориентированным на устройства производства ОАО «БелЭлектроМонтажНаладка».

И в итоге разрабатываемое программное средство должно улучшить условия труда оперативного персонала, повысить безопасность ремонтных работ, сократить до минимума расходы на эксплуатацию линий наружного освещения и транспортные расходы, связанные с контролем линий и ламп освещения, повысить безопасность жителей города в вечернее и ночное время, повысить безопасность дорожного движения, создать световой облик города.

Обеспечивая непрерывный мониторинг состояния сети наружного освещения, программное средство позволит:

1) оптимизировать структуру и режим управления городской сети освещения;
2) обеспечить оптимальный уровень освещенности улиц;
3) оперативно выявлять и устранять повреждения сетей освещения.

1.3 Обоснование выбора языка и средств разработки

Дипломный проект «Прораммное средство управления городским освещением» разработан на платформе .NET 4.0 в интегрированной среде разработки Microsoft Visual Studio 2013.

При реализации многопоточности и улучшения интерактивности была использована библиотека TPL и асинхронная модель на основе задач (Task-based Asynchronous Pattern, TAP).

Визуальный интерфейс реализовывался с использованием технологии Windows Presentation Foundation (WPF), а также с помощью библиотек Prism и Modern UI.

При реализации принципов обращение контроля (Inversion of Control, IoC) и внедрение зависимости (англ. Dependency injection) был использован Microsoft Unity контейнер.

Краткое описание использовавшихся технологий и средств для разработки дипломного проекта представлено ниже.

1.3.1 Платформа .NET, язык C# и среда разработки Microsoft Visual Studio

.NET Framework – программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является исполняющая среда Common Language Runtime (CLR), способная выполнять как обычные программы, так и серверные веб-приложения. NET Framework поддерживает создание программ, написанных на разных языках программирования.

Основной идеей при разработке .NET Framework являлось обеспечение свободы разработчика за счёт предоставления ему возможности создавать приложения различных типов, способные выполняться на различных типах устройств и в различных средах. Вторым принципом стало ориентирование на системы, работающие под управлением семейства операционных систем Microsoft Windows.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка (assembly). Затем код исполняется виртуальной машиной Common Language Runtime (CLR), или транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR, встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

Архитектура платформы .NET Framework представлена на рисунке 1.9. Одной из основных идей Microsoft .NET является совместимость программных частей, написанных на разных языках. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Основными поддерживаемыми .NET языками являются C#, VB.NET, C++/CLI, F#.

Для разработки данного дипломного проекта был выбран язык программирования C#, так как он широко распространен, динамично

развиватеся и изначально разрабатывался как язык прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR.

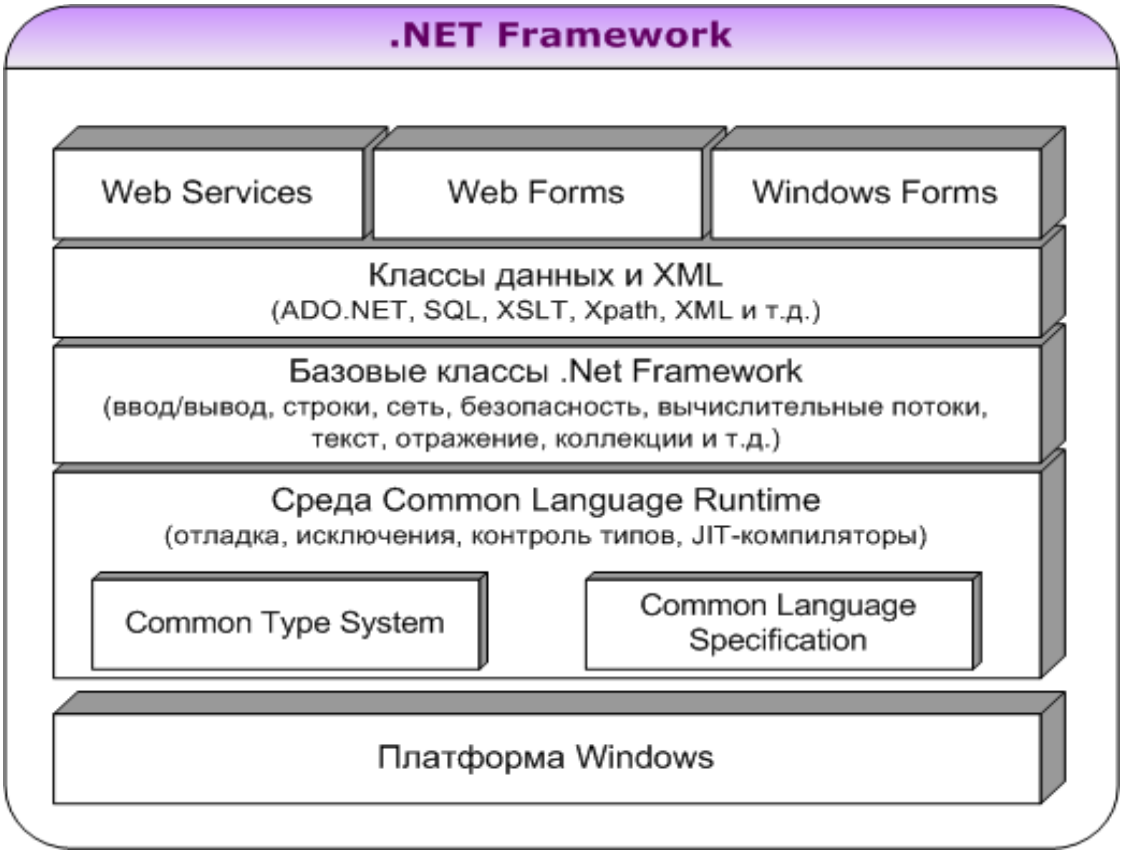

Рисунок 1.9 – Архитектура .NET Framework

Проект разрабатывался в интегрированной среде разработки, поддерживающей .NET, Microsoft Visual Studio 2013. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion, Git и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов цикла разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

1.3.2 Обзор асинхронной модели на основе задач

В эпоху многоядерных машин, которые позволяют параллельно выполнять сразу несколько процессов, стандартных средств работы с

потоками в .NET уже оказалось недостаточно. Поэтому одним из новшеств платформы .NET 4.0 стала библиотека параллельных задач TPL (Task Parallel Library), основной функционал которой располагается в пространстве имен System.Threading.Tasks. Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер. Кроме того, упрощается сама работа по созданию новых потоков. Поэтому начиная с .NET 4.0. рекомендуется использовать именно TPL и ее классы для создания многопоточных приложений, хотя стандартные средства и класс Thread по-прежнему находят широкое применение.

В основе библиотеки TPL лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов .NET задача представлена специальным классом - классом Task, который находится в пространстве имен System.Threading.Tasks. Данный класс описывает отдельную задачу, которая запускается в отдельном потоке. Хотя при желании ее также можно запускать синхронно в текущем потоке.

При установке библиотеки BCL.Async появляется возможность использовать ключевые слова async/await в .Net 4.

Модификатор async используется для указания того, что метод, лямбда-выражение или анонимный метод являются асинхронными. Если этот модификатор используется в методе или выражении, он называется асинхронным методом.

Оператор await применяется к задаче в асинхронных методах, для того, чтобы приостановить выполнение метода до тех пор, пока ожидаемая задача не завершится. Задача представляет собой работу, выполняющуюся в настоящее время.

Асинхронный метод, в котором используется await должен быть помечен модификатором async. Такой метод, определенный с помощью модификатора Async, и обычно содержащий один или несколько выражений Await, называется асинхронным методом.

1.3.3 Обзор Windows Presentation Foundation.

Windows Presentation Foundation (WPF) – система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (Extensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

XAML представляет собой язык декларативного описания интерфейса, основанный на XML. Также реализована модель разделения кода и дизайна, позволяющая кооперироваться программисту и дизайнеру. Кроме того, есть встроенная поддержка стилей элементов, а сами элементы легко разделить на элементы управления второго уровня, которые, в свою очередь, разделяются до уровня векторных фигур и свойств/действий. Это позволяет легко задать стиль для любого элемента, например, Button (кнопка).

1.4 Спецификация требований

Цель дипломного проекта – разработать программное средство для управления городским наружным освещением, что позволит улучшить систему освещения городов республики Беларусь и сократит расходы на обслуживание наружного освещения.

Разрабатываемое программное средство должно совмещать в себе основные преимущества рассмотренных аналогов:

1) Программное средство должно иметь иерархическую древовидную структуру и обеспечивать централизованное управление освещением города. В том числе его оперативное отключение в чрезвычайных случаях, контроль за состоянием сетей наружного освещения и своевременное выявление неисправностей.

2) Должен осуществляться контроль следующих параметров:

а) исправность предохранителей и наличие стороннего напряжения на отходящих линиях;

б) напряжения на каждой фазе;

в) токовой нагрузки по каждой фазе.

3) Должен вестись журнал команд и времени их подачи диспетчером, времени возникновения и ликвидации аварий, типов этих аварий. Необходимо отображение информации о состоянии ГПП на электронной карте населенного пункта. годовые графики освещения.

4) Программное средство позволяет вести наблюдение за текущим состоянием каждого устройства, управлять режимами освещения этих устройств, контролировать наличие аварий на них и переводить их в ручной/автоматический режим работы.

Для достижения цели дипломного проекта были сформулированы следующие задачи:

1) разработка структуры и функций программного средства;

2) построение моделей программного средства;

3) реализация алгоритмов для обеспечения диспетчерского контроля за управления наружным освещением;

4) разработка интуитивно понятного пользовательского интерфейса для конечного пользователя и тестирование ПС.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Диаграмма вариантов использования

Предметом разработки является программное средство управления городским освещением на базе ОС Windows. Программное средство должно быть выполнено в виде графического приложения.

Назначением программного средства является автоматизация и повышение эффективности процесса управления городским наружным освещением.

На рисунке 2.1 представлены возможные варианты использования для разрабатываемого программного средства.

На рисунке 2.1 представлены базовые действия диспетчера, инженера и удаленного устройства с приложением, так инженер может:

1) добавить описание устройства;
2) удалить описание устройства;
3) редактировать описание устройства;
4) настроить подключение к устройству;
5) просмотреть отчет;

Диспетчер может:

1) изменить встроенный в устройство график освещения;
2) отслеживать состояние параметров устройства;
3) отслеживать состояние связи с устройства;
4) отслеживать показания счетчиков;
5) отправлять команды на каждое устройство в отдельности;
6) отправлять общегородские команды;
7) синхронизировать времени;
8) просмотреть отчет;

Удаленные устройства отвечают на запросы приложения, отправляя данные о своем состоянии, либо меняя свое состояние.

2.2 Диаграмма развертывания

На рисунке 2.2 представлены диаграмма развертывания разрабатываемого программного средства. Из неё видно, что программное средство будет распространяться в виде msi-пакета для работы под операционными системами Microsoft Windows 7 или Microsoft Windows 8. Также необходимым условием является установка на рабочую станцию пункта диспетчерского управления .Net Framework версий 4.0 либо .Net Framework версий 4.5. Связь с удаленными устройствами будет осуществляться посредством сети интернет с использованием протокола “Modbus TCP”. На начальном этапе разработки необходимо поддержать связь с устройствами «РУНО 3», но также следует учесть возможность дальнейшей поддержки других устройств.

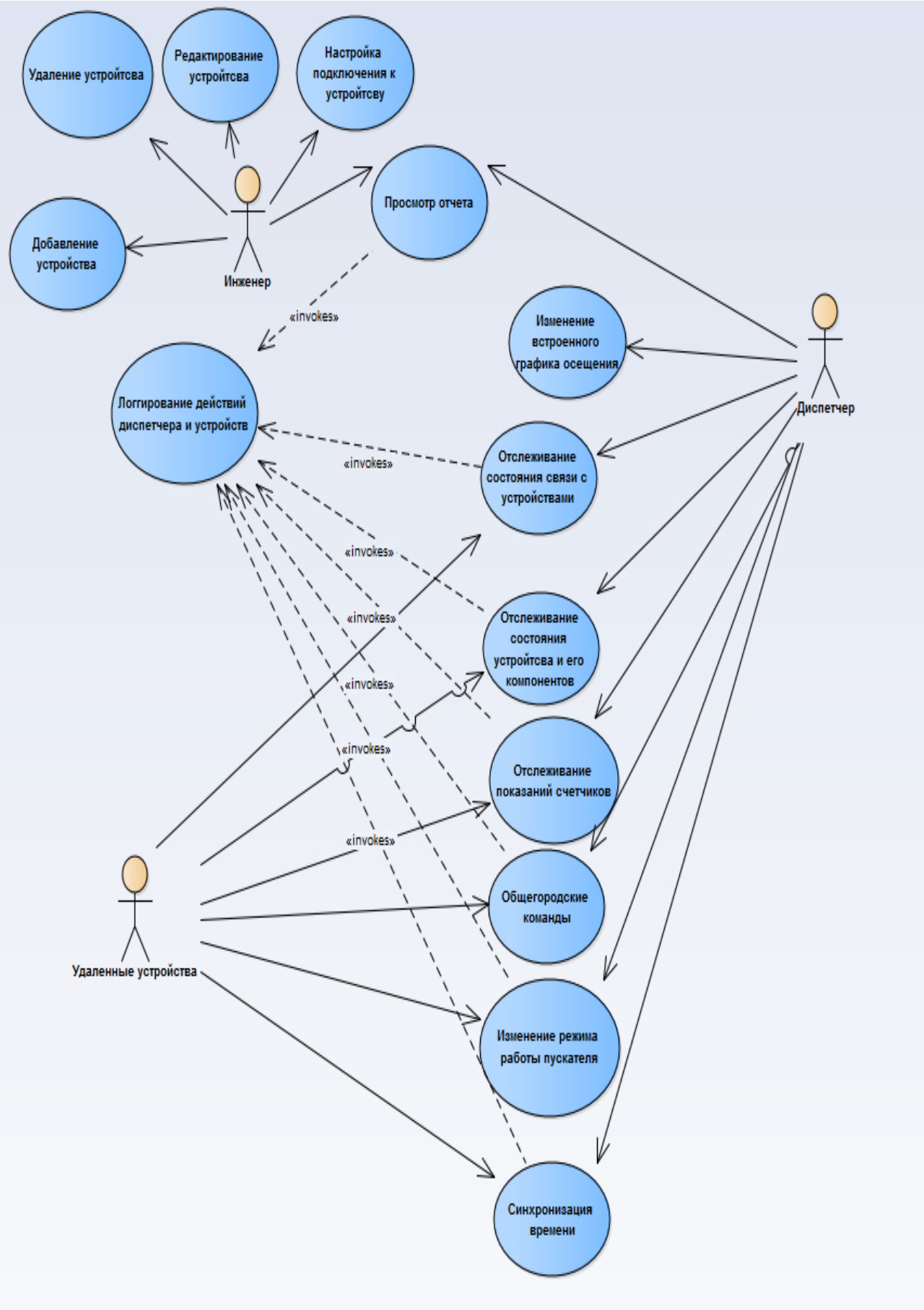

Рисунок 2.1 – Диаграмма вариантов использования

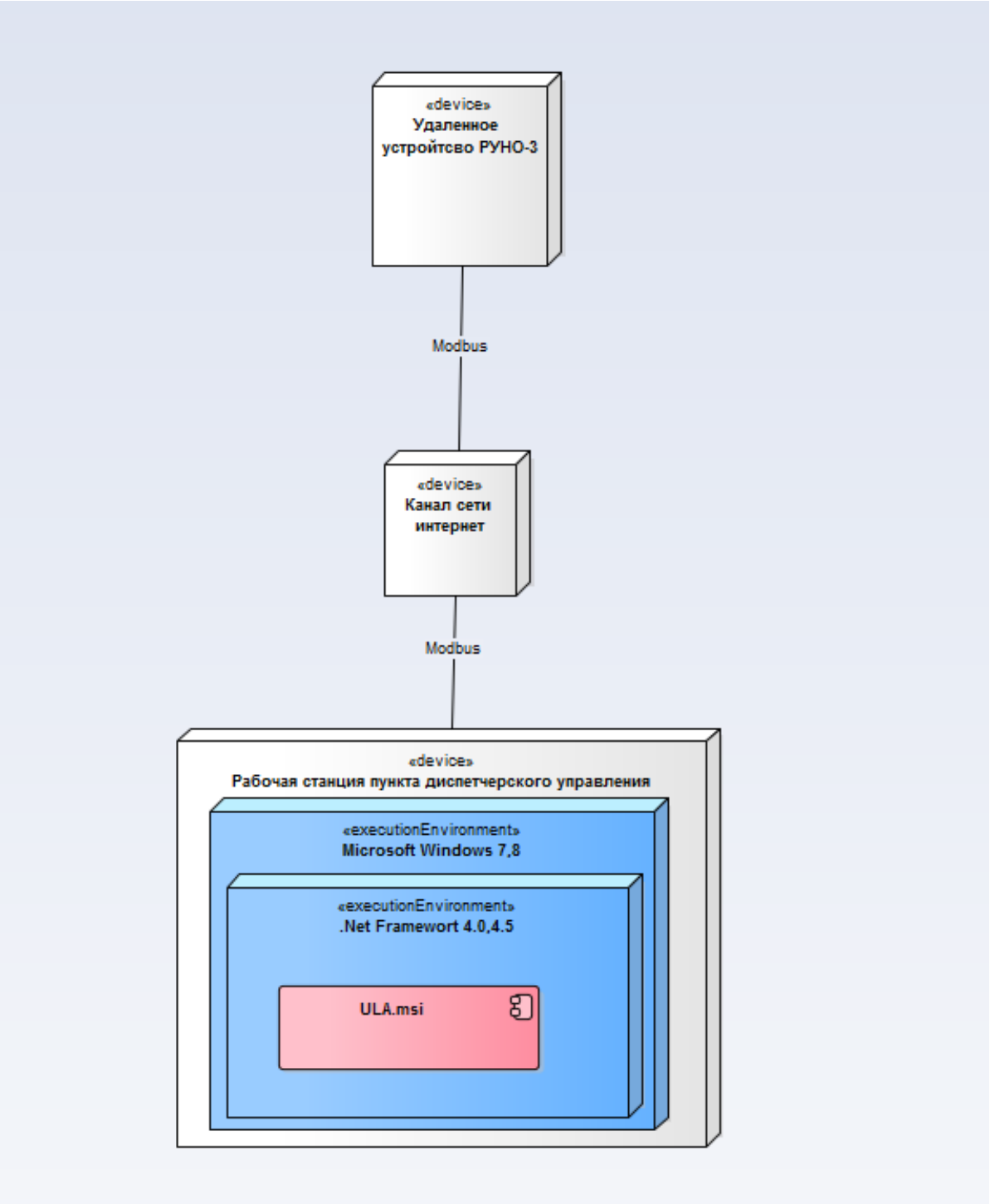

Рисунок 2.2 – Диаграмма развертывания

2.3 Диаграмма потоков данных

На рисунке 2.3. представлена диаграмма потоков данных. На ней можно увидеть три компонента:

1) Файл конфигурации. Он является хранилищем данных описывающих управляемые устройства.

2) Внешняя сущность – управляемое устройство РУНО 3. Данное устройство обменивается данными с разрабатываемым программным

средством по протоколу «Modbus TCP». РУНО 3 получает сообщения от ПС и, если это сообщение чтения данных, возвращает требуемую информацию.

3) Разрабатываемое программное средство. Является центральным компонентом рассматриваемой модели. Получает данные о управляемых устройствах из файла конфигурации, а также отправляет измененные данные обратно. Программное средство ведет интенсивный обмен данными вида запрос-ответ с множеством управляемых устройств.

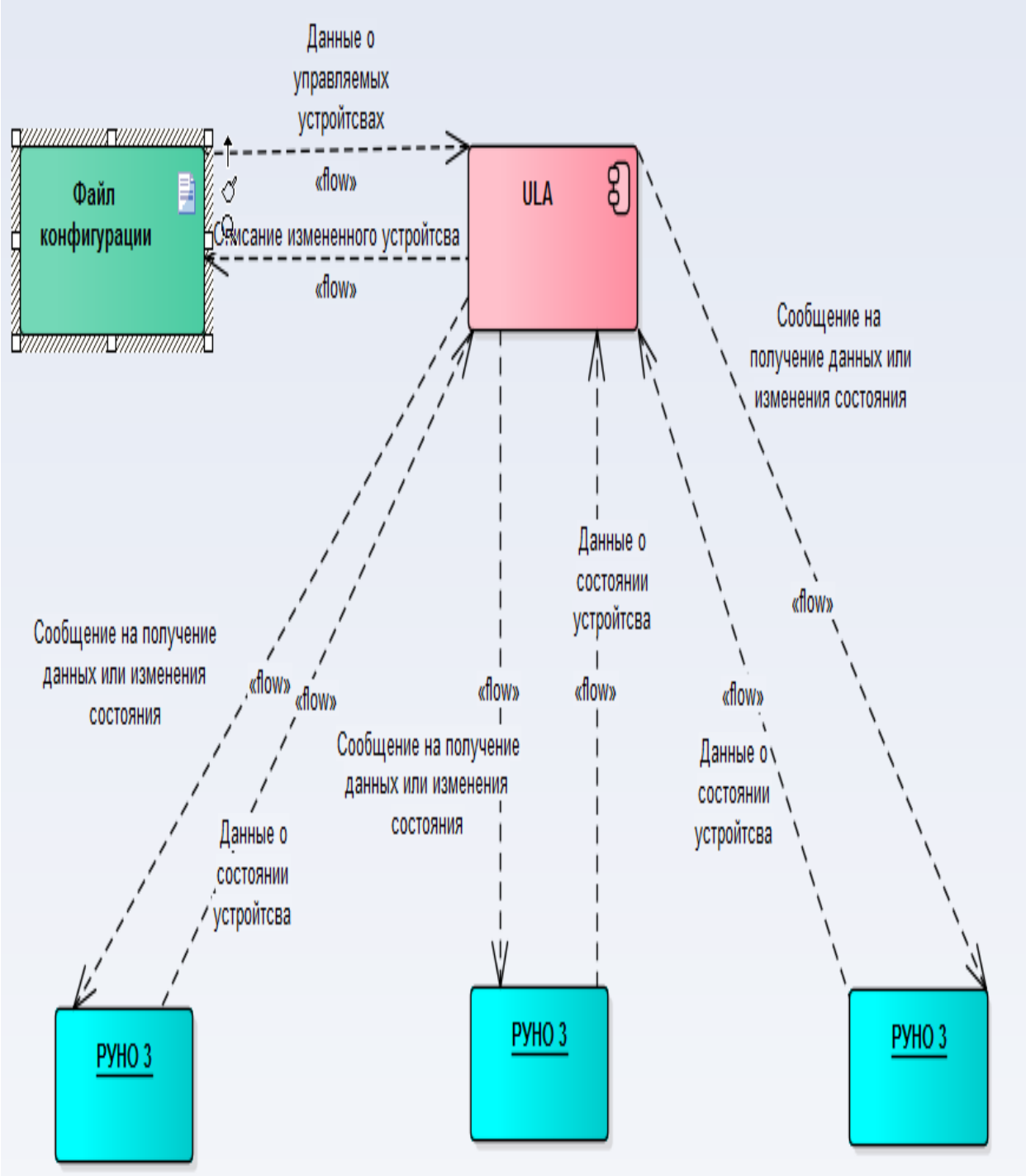

Рисунок 2.3 – Диаграмма потоков данных

2.4 Анализ требований

2.4.1 Требования к режимам работы

Программное средство должно поддерживать два возможных режима:

1) Режим реального времени
2) Режим конфигурирования

Режим реального времени предназначен для управления и отслеживания состояния наружных сетей освещения. Пользователем в данном режиме является диспетчер.

Режим конфигурирования предназначен для управления схемой освещения управляемого участка. Пользователем в данном режиме является инженер, который может добавлять, удалять или изменять описания устройств, на работу с которыми ориентировано разрабатываемое программное средство. Переход в режим конфигурирования должен быть защищен паролем.

2.4.2 Требования к оформлению пользовательского интерфейса

При разработке графического интерфейса должны быть использованы преимущественно светлые стили. Главная форма (Рисунок 2.4) должна обладать минимальным оформлением – основная часть места должна использоваться для отображения управляемых устройств. Все основные функции и соответствующие им дополнительные формы должны быть доступны и легко обнаруживаемы на главной форме.

Информацию о том, в каком режиме находится программное средство можно увидеть в верхней части экрана, слева от названия программы.

При запуске программного средства загружается режим реального времени. При первом запуске и отсутствующей файле конфигурации устройств, в режиме реального времени загрузится пустой экран и будет предложено перейти в режим конфигурирования для добавления новых устройств. Если база устройств уже создана, то на экране схематически отобразятся устройства и их состояния.

Главная форма в режиме реального времени представляет собой поле, на котором схематически изображены устройства, подключенные по GSM связи. Количество отображенных устройств на общем фрагменте не превышает 105 штук. Состояние устройств опрашивается с периодом, указанным в режиме конфигурирования. При необходимости этот период можно изменить.

Каждое устройство схематически изображено как прямоугольник (рисунок 2.5.), с выделенными областями под название (Блок 1), состояние пускателей и посланную команду (Блоки 2, 3 и 4), и состояние устройства (Блок 5).

Различные состояния пускателей устройства на главной форме должно различаться цветом графического элемента. На рисунке 2.6 рассмотрим различные положения и состояния устройства:

В верхней части формы в режиме реального времени находится меню (Рисунок 2.4), из которого можно вызвать общегородские команды, просмотреть историю событий и вызвать файл справки.

Форма общегородских команд должна использоваться для отправки команд управления на все подключенные устройства одновременно.

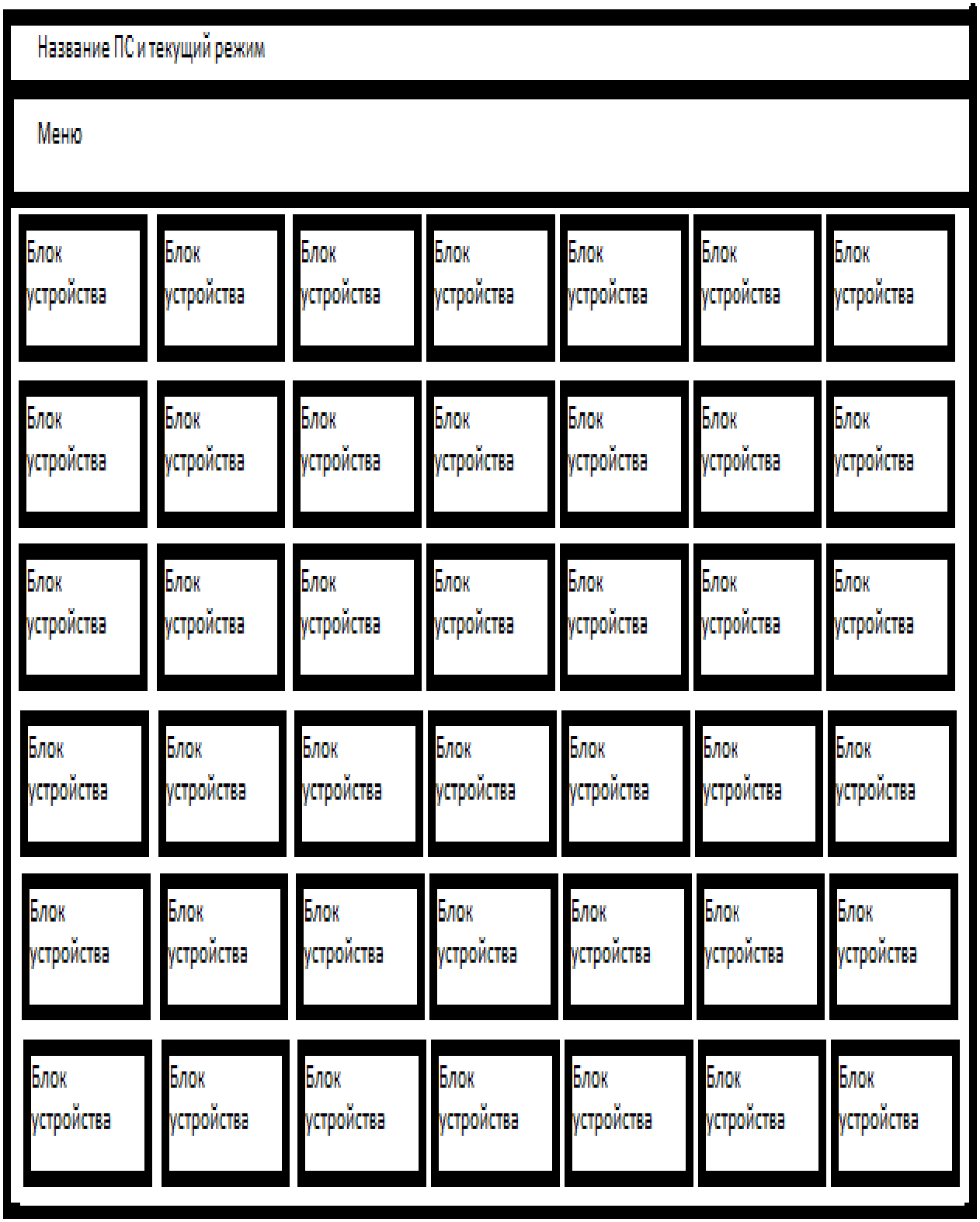

Рисунок 2.4 – Эскиз главной формы

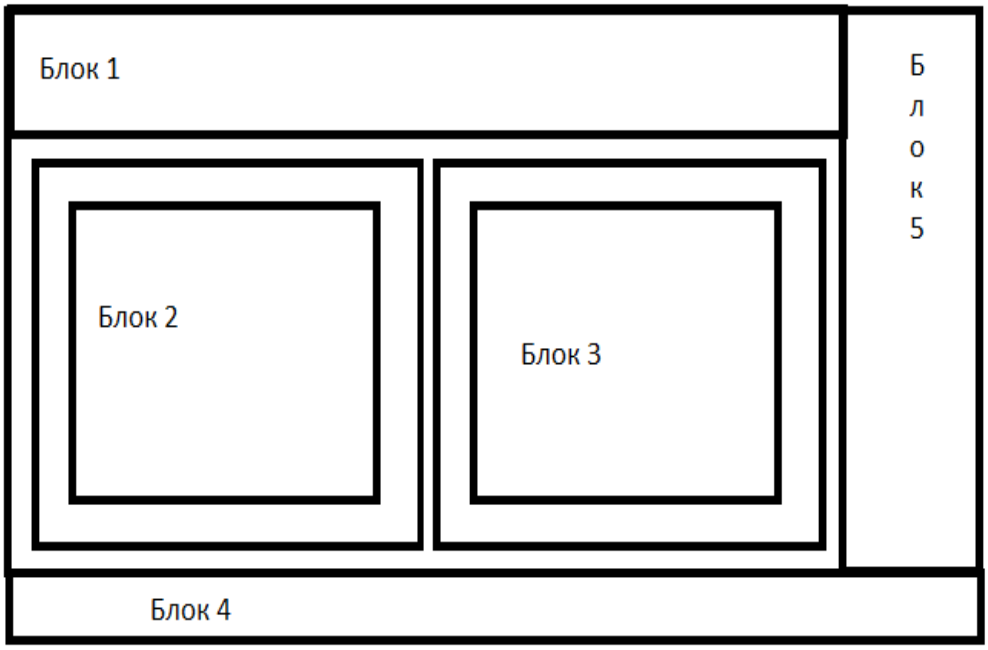

Рисунок 2.5 – Эскиз схематического изображения устройства

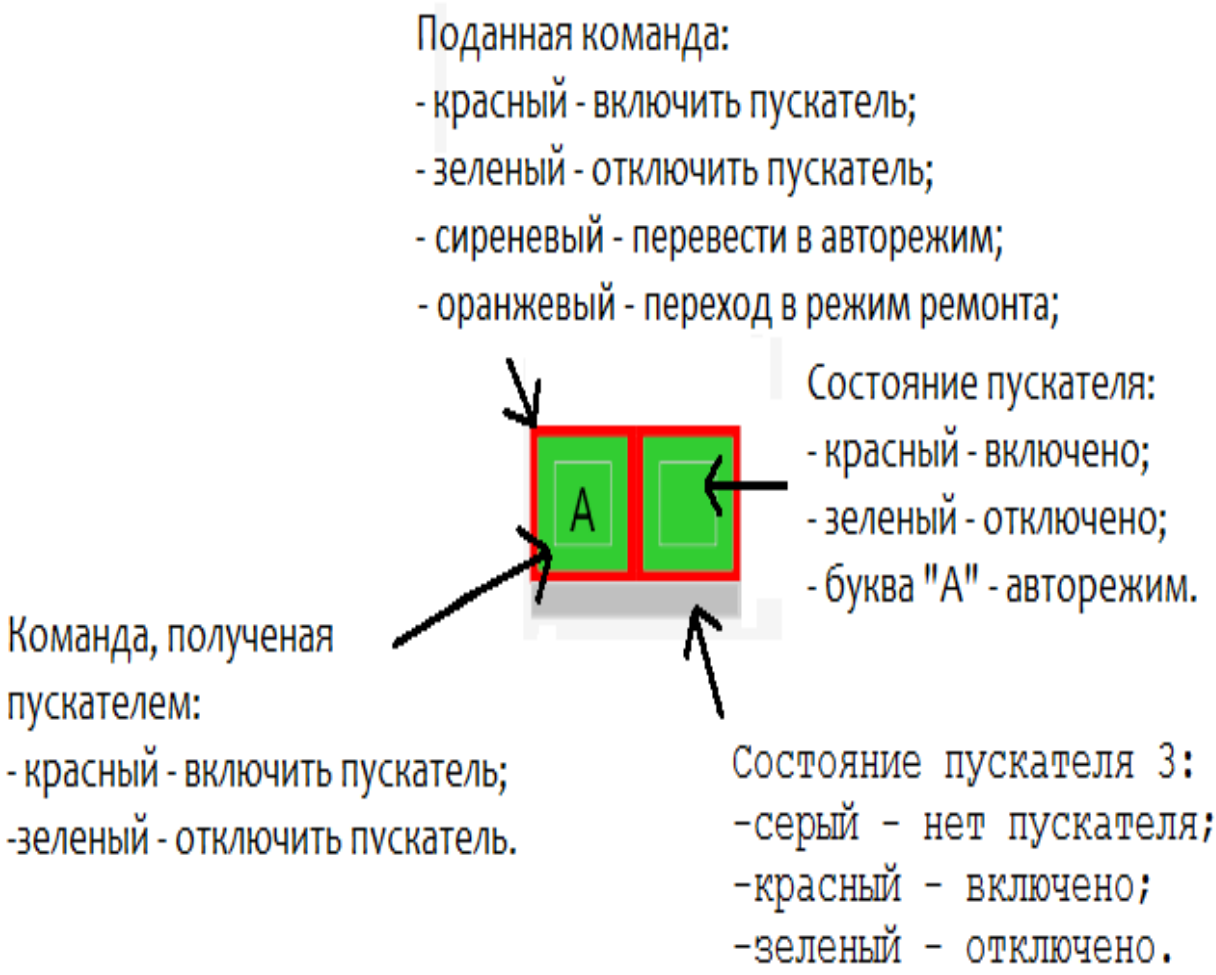

Рисунок 2.6 – Эскиз описания состояний пускателей устройства на главной форме

На данной форме требуется поддержка вызова следующих команд:

1) команды на включение или отключение ночного режима освещения;

2) команды на включение или отключение вечернего режима освещения;

3) команды на включение или отключение полного режима освещения;

4) команды на включение или отключение режима подсветки (срабатывают только если пускатели сконфигурированы соответствующим образом);

5) команды на включение или отключение режима иллюминации (срабатывают только если пускатели сконфигурированы соответствующим образом);

6) команды на включение или отключение режима энергосбережения (срабатывают только если пускатели сконфигурированы соответствующим образом);

7) команда для перевода всех устройств в авторежим;

8) команда «Синхронизация времени» всем устройствам одновременно. Для отправки команды синхронизации времени также должна быть разработана отдельная интерактивная форма.

В режиме реального времени по двойному клику на схематическом изображении ШНО должен осуществляться переход к схеме РУНО-3.

Конфигурация каждого РУНО-3 должна автоматически считывается из устройства. И в соответствии с этой конфигурацией рисуется схема устройства. При открытой схеме устройства запросы к РУНО-3 шлются чаще и в большем количестве (по сравнения с режимом реального времени). Период обновления информации на открытой схеме можно изменить в Режиме конфигурации.

На схеме должны быть изображены:

1) панель управления;

2) пускатели;

3) рубильник;

4) предохранители;

5) дата/время;

6) уровень сигнала;

7) показания счетчиков;

8) неисправности;

9) панель конфигурирования.

Вверху должно быть указано имя устройства. Под ним должна находиться панель управления устройством. Можно, независимо от других устройств:

1) включить или отключить ночное (вечернее, полное) освещение;

2) включить или отключить другие режимы управления (подсветка, иллюминация, энергосбережение), если устройство сконфигурировано нужным образом;

3) перевести устройство в авторежим;

4) перевести в режим ремонта.

В центральной части схематически прорисованы пускатели и прикрепленные к ним предохранители с названиями, которые задаются в режиме конфигурирования.

С пускателями в соответствии с конфигурацией соотнесены предохранители. Они должны изменять свою цветовую схему в зависимости от их состояния:

1) предохранитель не под напряжением;

2) предохранитель под напряжением;

3) предохранитель не под напряжением, но должен быть под напряжением (ненормальное состояние – горит индикатор неисправность);

4) предохранитель под напряжением, но должен быть не под напряжением (ненормальное состояние – горит индикатор неисправность).

В правой части экрана должны отображаться дата и время, считываемые с РУНО-3.

Под датой/временем можно посмотреть значение уровня сигнала РУНО-3 в относительных единицах, которое показывает качество сигнала GSM. Допустимые значения от 0 до 36. Значения 0-5 – недостаточный сигнал, 36 единиц – наилучший сигнал, 99 – невозможно считать значение. Значение вида "- -" будет говорит о том, что в данной версии РУНО-3 невозможно определить качество GSM сигнала.

Еще ниже должны быть расположены индикаторы неисправностей.

По нажатию на кнопку "Показания счетчика", требуется получить данные с подключенного счетчика (фазные напряжения, фазные токи, мощности по фазам и активная потребленная энергия нарастающим итогом суммарно по всем тарифам).

Панель конфигурирования должна давать доступ к конфигурации РУНО-3, к графикам, зашитым в РУНО-3 и к журналу сообщений РУНО-3. Для доступа к этой информации должен использоваться пароль.

2.4.3 Требования к программному и техническому обеспечению

Минимальные системные требования:

1) .Net Framework 4.0 или выше;
2) Windows 7 или выше;
3) 1Gb ОЗУ.

2.4.4 Требования к эргономике и технической эстетике

Программа должна быть оптимизирована для просмотра при разрешении 1024*768, 1280*1024 без горизонтальной и вертикальной полосы прокрутки. Элементы управления должны быть сгруппированы однотипно – горизонтально либо вертикально – на всех формах.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры ПС

Проектирование приложения основано предметно-ориентированном подходе.

Предметно-ориентированное проектирование (англ. Domain-driven design, DDD) – это набор принципов и схем, помогающих разработчикам создавать изящные системы объектов. При правильном применении оно приводит к созданию программных абстракций, которые называются моделями предметных областей. В эти модели входит сложная бизнес-логика, устраняющая промежуток между реальными условиями области применения продукта и кодом.

Предметно-ориентированное проектирование не является какой-либо конкретной технологией или методологией. DDD – это набор правил, которые позволяют принимать правильные проектные решения. Данный подход позволяет значительно ускорить процесс проектирования программного обеспечения в незнакомой предметной области.

Подход DDD особо полезен в ситуациях, когда разработчик не является специалистом в области разрабатываемого продукта. К примеру: программист не может знать все области, в которых требуется создать ПО, но с помощью правильного представления структуры, посредством проблемно-ориентированного подхода, может без труда спроектировать приложение, основываясь на ключевых моментах и знаниях рабочей области.

Предметно-ориентированный подход подразумевает использование объектно-ориентированного проектирования и использования шаблонов проектирования.

Основными преимуществами стиля DDD являются:

1) Обмен информацией. Все участники группы разработки могут использовать модель предметной области и описываемые ею сущности для передачи сведений и требований предметной области с помощью общего языка предметной области, не прибегая к техническому жаргону.
2) Расширяемость. Модель предметной области часто является модульной и гибкой, что упрощает обновление и расширение при изменении условий и требований.
3) Удобство тестирования. Объекты модели предметной области характеризуются слабой связанностью и высокой связность, что облегчает их тестирование.

Основываясь на предметно-ориентированном подходе, была определена трехуровневая модель программного приложения. Согласно данной модели приложение делится на три уровня:

1) уровень графического интерфейса, который будет отображаться на устройстве визуального отображения информации и взаимодействовать с пользователем;

2) уровень бизнес-логики, который представляет объектную модель системы и осуществляет основные функциональные обязанности системы;

3) уровень доступа к данным, который реализует доступ к различным данным, независимо от того, хранятся ли они в локальной базе данных или файле, либо поставляются от удаленных сетевых устройств.

Основными преимуществами трех-уровневого архитектурного стиля являются:

1) Удобство поддержки. Уровни не зависят друг от друга, что позволяет выполнять обновления или изменения, не оказывая влияния на приложение в целом.

2) Гибкость. Управление и масштабирование каждого уровня может выполняться независимо, что обеспечивает повышение гибкости.

3) Конфигурируемость. Изолированность уровней друг от друга позволяет быстро и простыми средствами переконфигурировать систему при возникновении сбоев.

4) Доступность. Приложения могут использовать модульную архитектуру, которая позволяет использовать в системе легко масштабируемые компоненты, что повышает доступность.

В качестве концепции разработки части графического интерфейса была выбрана концепция MVVM (model-view-viewmodel). Основная идея этой концепции состоит в разделении бизнес-логики от её представления.

Для реализации доступа из модуля более высокого уровня к модулям более низкого уровня без обращения к конкретной реализации (работая только с интерфейсом), будут использованы принципы обращения контроля и внедрения зависимостей.

3.1.1 Диаграмма взаимодействия модулей.

При проектировании программного средства было решено разделить его на следующие модули:

1) модуль доступа к данным;
2) модуль бизнес-логики;
3) модуль пользовательского интерфейса;
4) модуль композиции;
5) модуль описания абстрактного устройства;
6) модуль описаний конкретных устройств;

Диаграмма взаимодействия модулей разрабатываемого программного средства представлена на рисунке 3.1.

3.1.2 Модуль доступа к данным.

Слой доступа к данным, представленный проектом ULA.Drivers, должен являться интерфейсом инкапсулирующим процесс сбора данным с удаленных устройств, т.е. являться провайдером данных для остальных частей разрабатываемого программного средства.

Связь между программным средством, установленным на рабочей станции диспетчера, с устройством осуществляется по протоколу «Modbus TCP».

Основным классом данного модуля является DefaultDataEntityStreamProducer. Данный класс представляет собой сервис взаимодействия с удаленными устройствами. Его открытые методы ProcessGetDataAsync и ProcessPostDataAsync принимают описание команды к удаленному устройству, ставят эту команду в очередь и возвращают объект Task представляющий асинхронное описание данной команды.

3.1.3 Модуль бизнес-логики

Модуль бизнес-логики, представленный проектами ULA.Business.Infrastructure и ULA.Business, содержит описание сервисов управления описания удаленных устройтсв и работы с файлом конфигурации и сервиса управления удаленным устройтсвом.

Проект ULA.Business.Infrastructure содержит описание интерфейсов вышеупомянутых сервисов, а проект ULA.Business содержит их реализацию.

3.1.4 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса представлен четырьмя проектами:

1) ULA.Shell. Данный проект является точкой входа в приложение
2) ULA.Presentation.SharedResourses. Данный проект содержит описание графических ресурсов. Таких как картинки, графические элементы управления, WPF-конвертеры и прочие ресурсы используемые из других проектов.
3) ULA.Presentation.Infrastructure. Данный проект содержит программные интерфейсы сервисов взаимодействия с пользователем и интерфейсы ViewModel-ей.
4) ULA.Presentation. Данный проект содержит реализации разметки UserControl-ов и соответствующих им ViewModel-ей, сервисов взаимодействия с пользователем и классов проверкяющих вводимые пользователем данные.

3.1.5 Модуль композиции

Модуль композиции, представленный проектом ULA.CompositionRoot, содержит классы предназначенные для реализации принципов обращения контроля и внедрения зависимостей.

3.1.6 Модуль описания абстрактного устройства

Модуль описания абстрактного устройства, представленный проектами ULA.AddinsHost и ULA.Common, описывает абстрактное логическое управляемое устройство. Реализуя интерфейсы или расширяя классы описанные в данном проекте, возможно подключать новые устройства к

существующей системе управления городским освещением не изменяя исходных код самого программного средства.

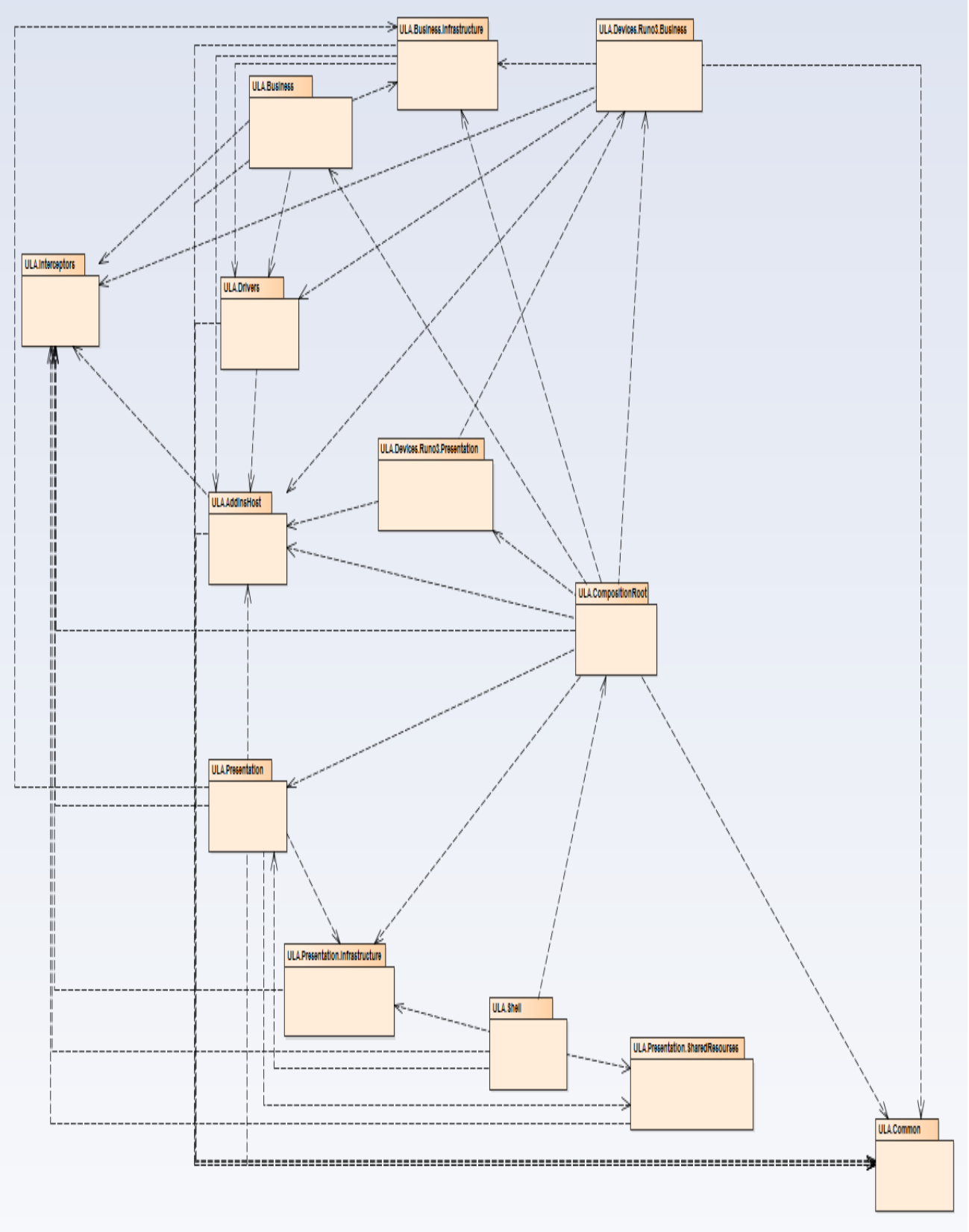

Рисунок 3.1 – Диаграмма взаимодействия модулей

3.1.7 Модуль описаний конкретных устройств

Данный модуль содержит в себе реализации логических удаленных устройств, наследующихся от классов представленных в модуле абстрактного логического устройства. На данный момент данный модуль представлен проектами описывающими устройство РУНО-3. Проекты получили названия ULA.Devices.Runo3.Business и ULA.Devices.Runo3.Presentation. ULA.Devices.Runo3.Business описывает логику работы с устройством Руно-3, а проект ULA.Devices.Runo3.Presentation реализует графические примитивы, с помощью которых данное устройство будет отображаться в графическом интерфейсе разрабатываемого программного средства.

3.2 Разработка схемы алгоритма работы с программой

Схема алгоритма работы программы представлена на рисунке 3.2. Также она выполнена на листе формата А1. На ней мы видим, что при запуске приложения открывается всегда одна и та же форма. Далее пользователь путем нажатия на кнопки или активные области может переходить на другие формы разрабатываемого приложения. При каждом нажатии запрос пользователя обрабатывается и ему предоставляется соответствующая этому запросу форма. Работа с программой считается законченной, когда пользователь закрывает приложение или, когда система уничтожает процесс, в рамках которого это приложение работает.

3.3 Разработка алгоритма отправки общегородских команд

Схема алгоритма отправки общегородских команд представлена на рисунке 3.3. Также она выполнена на листе формата А1.

Одной из важнейших частей программы является возможность управления освещением посредством отправки команд на удаленное устройство. Для пользователей крайне необходима возможность изменять состояния всех контролируемых устройств одной командой.

Для каждого устройства из списка управляемых устройств необходимо проверить применимость общегородской команды к нему. В первую очередь отфильтровываются устройства с которыми в данный момент связь не установлена. Затем следует анализ конфигурации схемы устройства в отношении отправляемой команды и, если устройство в данный момент не находится в состоянии ремонта, формируется сообщение описывающее команду применяемую к данному устройству. Данное сообщение отправляется сервису взаимодействия с удаленными устройствами.

Если сервис принял команду на обработку, то на форме необходимо отобразить факт отправки команды на устройство и запланировать проверку, которая покажет или устройство изменило режим работы.

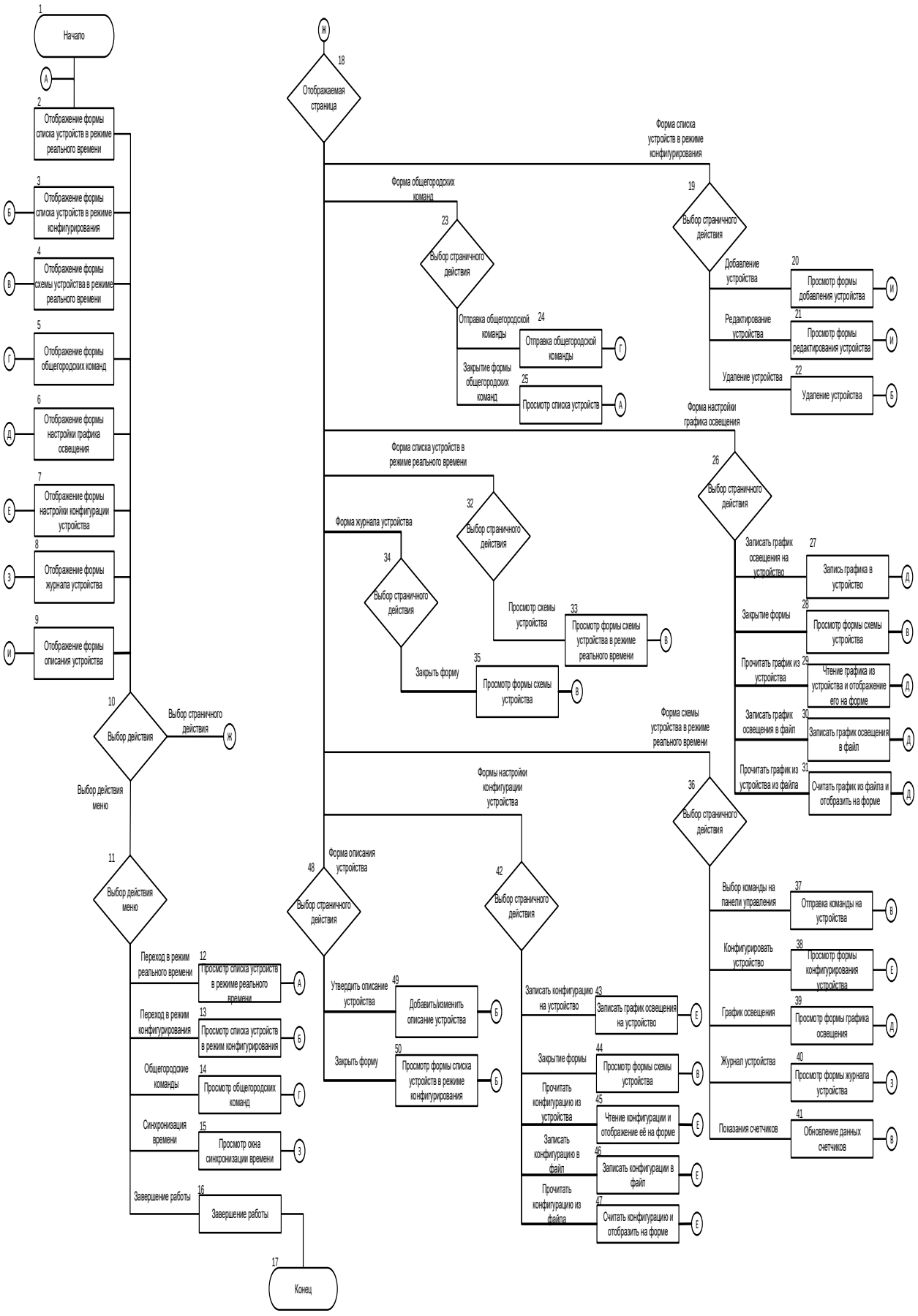

Рисунок 3.2 – Схема алгоритма работы программы

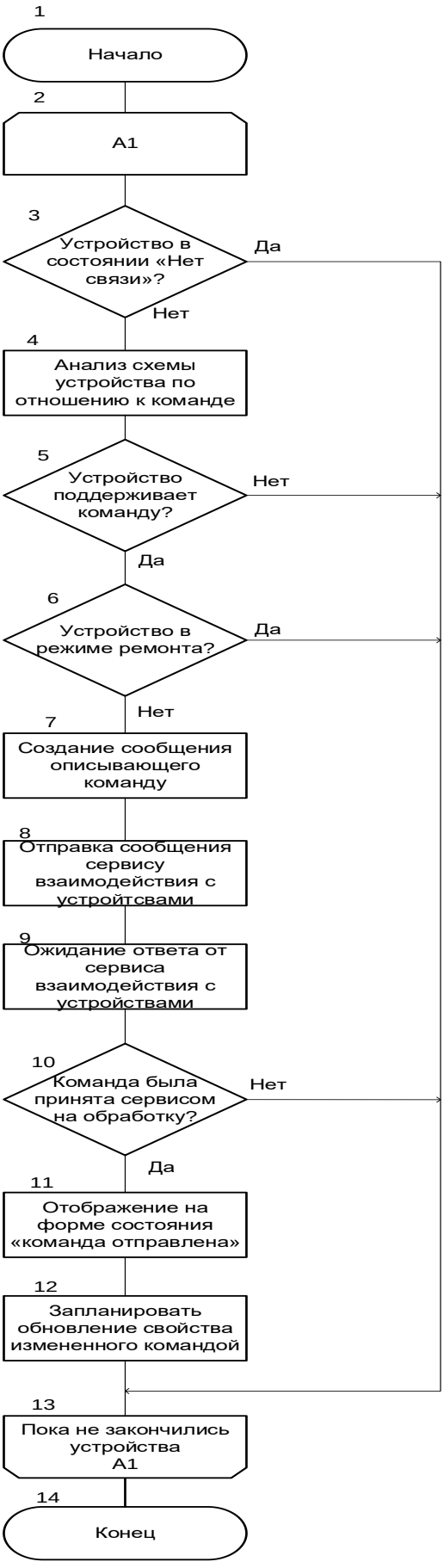

Рисунок 3.3 – Схема алгоритма отправки общегородских команд

3.4 Разработка алгоритма отслеживания состояния устройства

Схема алгоритма отслеживания состояния устройства представлена на рисунке 3.4. Также она выполнена на листе формата А1.

Одним из основных действий, производимых разработанным программным средством, является получение данных описывающих состояние освещения на контролируемом участке. Входные данные представляют собой поток modbus ответов от устройства на запросы считывания определенных свойств. Устройство хранит параметры освещения в массиве байт и характеризуется схемой данных, описывающий соответствие индексов внутреннего массива и свойств находящихся по этим адресам. Устройство хранит различные параметры освещения, и не всех из них требуют постоянного автоматического считывания. Также следует отметить, что максимальная блина блока данных в modbus сообщении не должна превышать 260 байт.

Исходя из вышесказанного, был разработан механизм, обеспечивающий постоянное обновление свойств устройства, описывающих его состояние, в зависимости от текущего состояния приложения. Получение сообщений происходит посредством планировщика обновлений, который позволяет с заданным промежутком времени производить выборку сообщений.

Данный алгоритм включает в себя работы по отправке асинхронных сообщений чтения данных с устройства, анализ ответов от устройства и обновления таблицы описания состояния устройства. Он начинает свою работу при старте программного средства, если оно уже было сконфигурировано, а так же выполняется по требованию планировщика задач на протяжении всего жизненного цикла программного средства, за исключением перехода в режим конфигурирования.

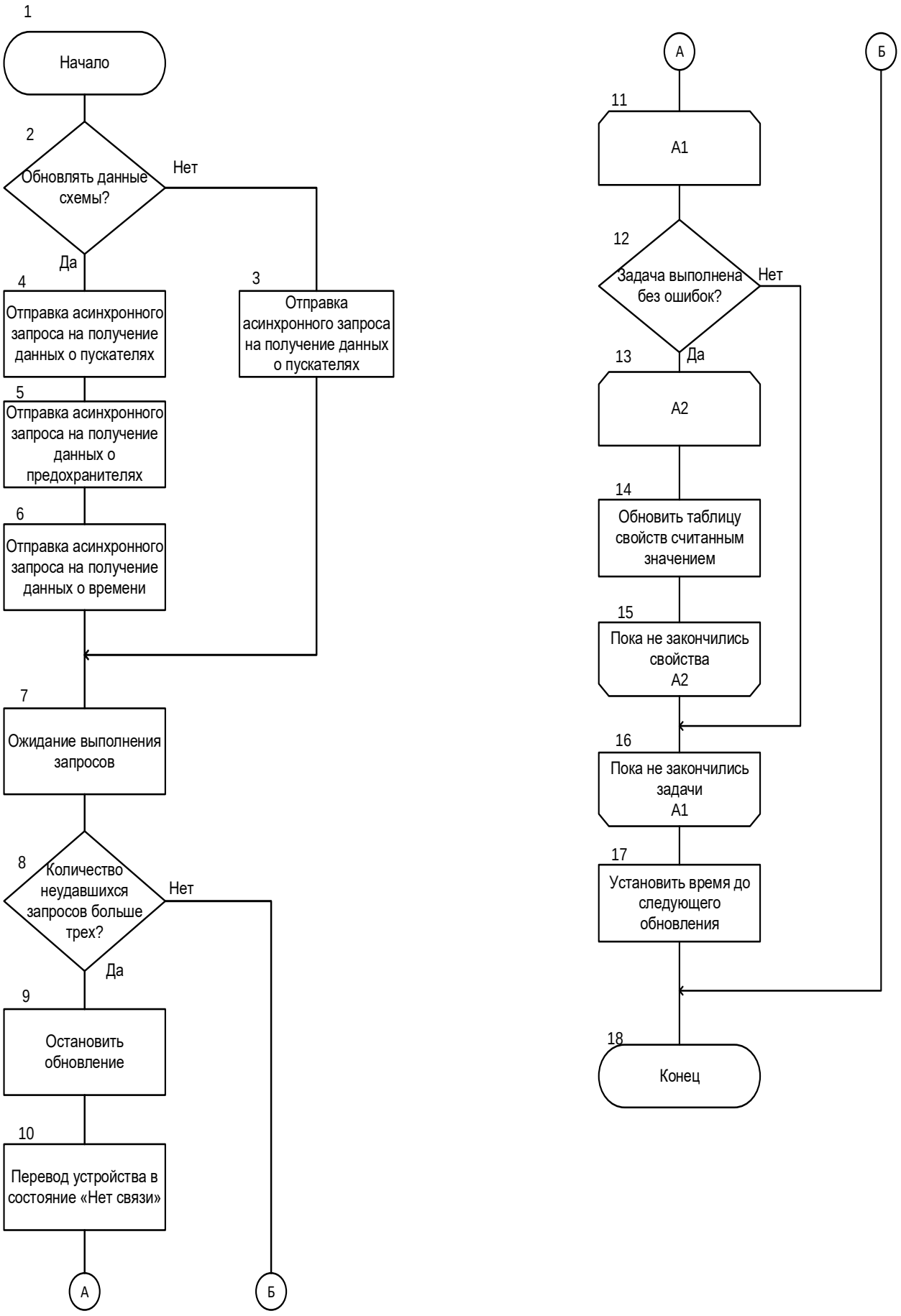

Рисунок 3.4 – Схема алгоритма отслеживания состояния устройства

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

В данном разделе будет рассмотрен переход от абстрактной модели, разработанной во втором разделе, к физической модели, то есть описание процесса адаптации абстрактной модели к конкретной реализации программного средства.

Поскольку выбранной архитектурой системы является трех-уровневая архитектура на основе технологии .Net и языка C#, необходимо кратко рассмотреть программное обеспечение, необходимое для разработки и для работы приложения.

4.1 Используемые средства и модули

Для разработки кода на языке C# можно использовать одну из многих доступных на выбор сред разработки (IDE), в данном случае для разработки была выбрана Visual Studio 2013, т.к. она лучше всего поддерживает написание приложений на языке C# и полностью поддерживает все возможности языка, появившиеся с выходом платформы Microsoft .NET Framework 4.5. Для компиляции кода и работы программного средства на персональном компьютере должна быть установлена платформа Microsoft .NET Framework 4.5.

Операционной системой, на которой будет производиться разработка программного средства, была выбрана Windows 7. Она является одной из наиболее устойчивых и распространенных систем из линейки Windows на сегодняшний день. Выбор линейки ОС Windows основан на том, что среда разработки Visual Studio работает только на данных операционных системах.

4.2 Структура модулей программного средства

Программное средство ориентировано на работу с устройствами, которые контролируют и управляют наружным освещением. Поэтому требовалось разработать объектную модель, описывающую удаленные устройства. На рисунке 4.1 представлены базовые сущности, описывающие устройства.

Базовый интерфейс который реализуют все описания устройств – IlogicalDevice. Он содержит 3 свойства: DeviceDescription (строка содержащая текстовое описание устройства), DeviceId (уникальный идентификатор устройства) и DeviceName (строка содержащая имя устройства).

Интерфейс IConfigLogicalDevice, наследующийся от ILogicalDevice, декларирует описание устройства в режиме конфигурации. Он содержит описание двух методов и одного свойства. Метод CreateMomento должен вернуть объект, описывающий состояние устройства. Метод

SetMomentoAsync предназначен для изменения состояния устройства в соответствии с переданным как аргумент объектом. Свойство RestoreDataTableFromMomento должно возвращать объект описывающий таблицу данных, хранящихся в устройстве.

Абстрактный класс ConfigLogicalDeviceBase реализует IConfigLogicalDevice интерфейс. В данном классе реализованы общие для всех устройств свойства (DeviceDescription, DeviceId, DeviceName), а специфичные для каждого типа устройства методы и свойства помечены как абстрактные.

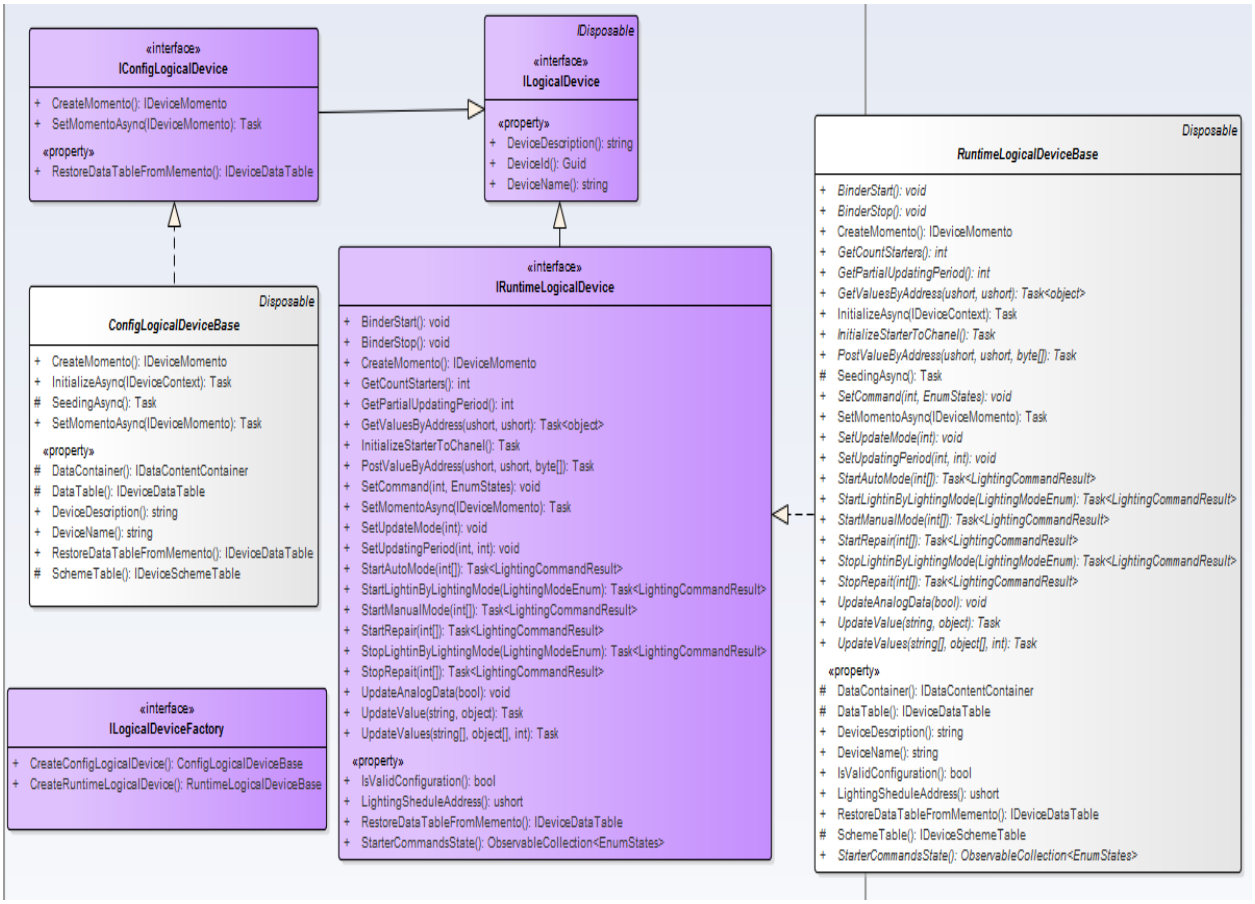

Рисунок 4.1 – Базовые сущности, описывающие устройство

Интерфейс IRuntimeLogicalDevice, наследующийся от ILogicalDevice, декларирует описание устройства в режиме реального времени. Он содержит описание методов и свойств, предназначенных для взаимодействия с удаленным устройством. Метод CreateMomento должен вернуть объект, описывающий состояние устройства. Метод SetMomentoAsync предназначен для изменения состояния устройства в соответствии с переданным как аргумент объектом. Методы BinderStart и BinderStop предназначены для запуска, и остановки соответственно, механизма привязки к удаленному устройству (т.е. открытие сеанса связи и запуск автоматического обновления данных). Метод SetUpdateMode предназначен для изменения режима обновления устройства. Методы GetValueByAddress и PostValueByAddress

возвращают и устанавливают значения находящееся по определенному адресу в устройстве. Методы UpdateValue и UpdateValues предназначены для изменения значений по имени свойства. Методы StartAutoMode, StartLightingByLightingMode, StartManualMode, StartRepair предназначены для запуска определенных режимов работы устройства. Свойство IsValidConfiguration должно показывать или устройство было корректно сконфигурировано. Свойство LightingSheduleAddress возвращает значение адреса по которому находятся данные описывающие график освещения.

Абстрактный класс RuntimeLogicalDeviceBase реализует IRuntimeLogicalDevice интерфейс. В данном классе реализованы общие для всех устройств свойства (DeviceDescription, DeviceId, DeviceName), а специфичные для каждого типа устройства методы и свойства помечены как абстрактные.

Интерфейс ILogicalDeviceFactory описывает методы CreateConfigLogicalDevice и CreateRuntimeLogicalDevice для создания устройств в режимах конфигурирования и реального времени соответственно.

Из описания устройств можно увидеть, что, для представления состояния устройств, применяется паттерн «Хранитель» (англ. Momento). Его реализация в контексте программного средства представлена на рисунке 4.2.

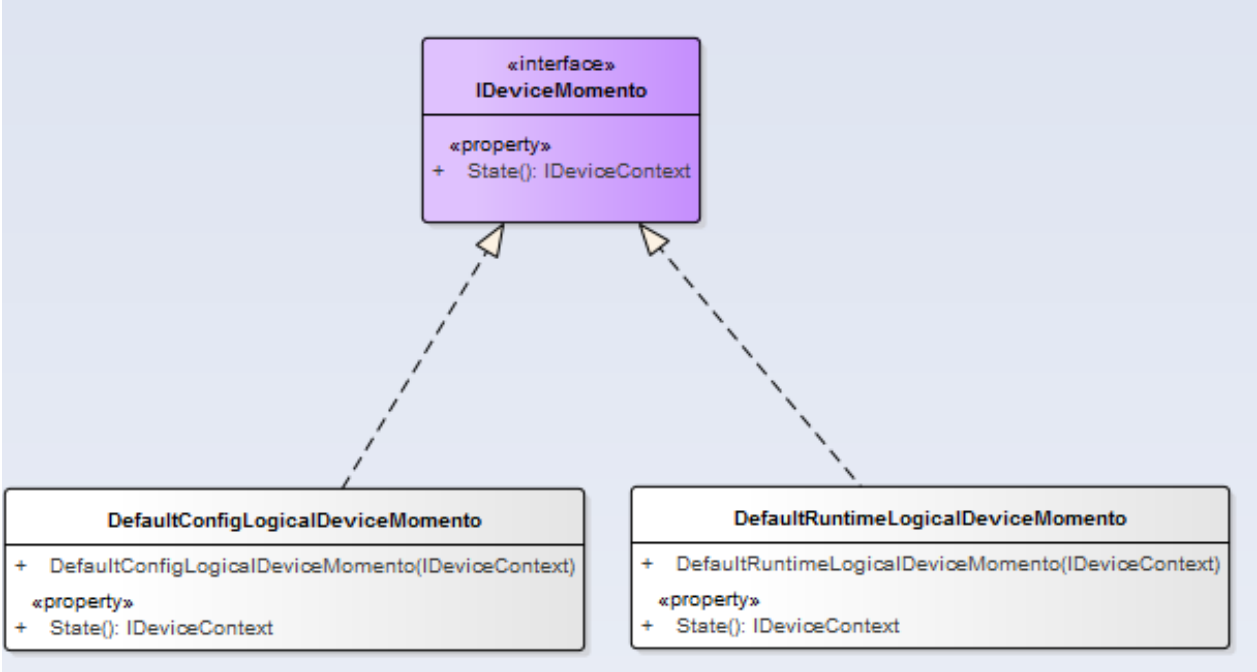

Рисунок 4.2 – Диаграмма классов реализации паттерна «Хранитель»

Был создан базовый интерфейс IDeviceMomento, описывающий единственное свойство State типа IDeviceContext. Классы DefaultConfigLogicalDeviceMomento и DefaultRuntimeLogicalDeviceMomento реализуют интерфейс IDeviceMomento для описаний устройств в режимах конфигурирования и реального времени соответственно.

Упомянутый ранее интерфейс IDeviceContext описывает модель данных, описывающую устройство. Диаграмма классов описывающая контекст устройства представлена на рисунке 4.3. Из не видно, что интерфейс содержит шесть свойств. Свойство DataContainer (типа IDataContentContainer) содержит данные специфичные для устройств определенного типа. Свойство DataTable (типа IDataTable) содержит описывание таблицы данных хранимых на устройстве. Свойство DeviceDescription представляет строку текстового описания устройства. Свойство DeviceName содержит название устройства. Свойство DeviceType хранит тип устройства. Свойство SchemeTable (типа IDeviceSchemeTable) описывает схему устройства.

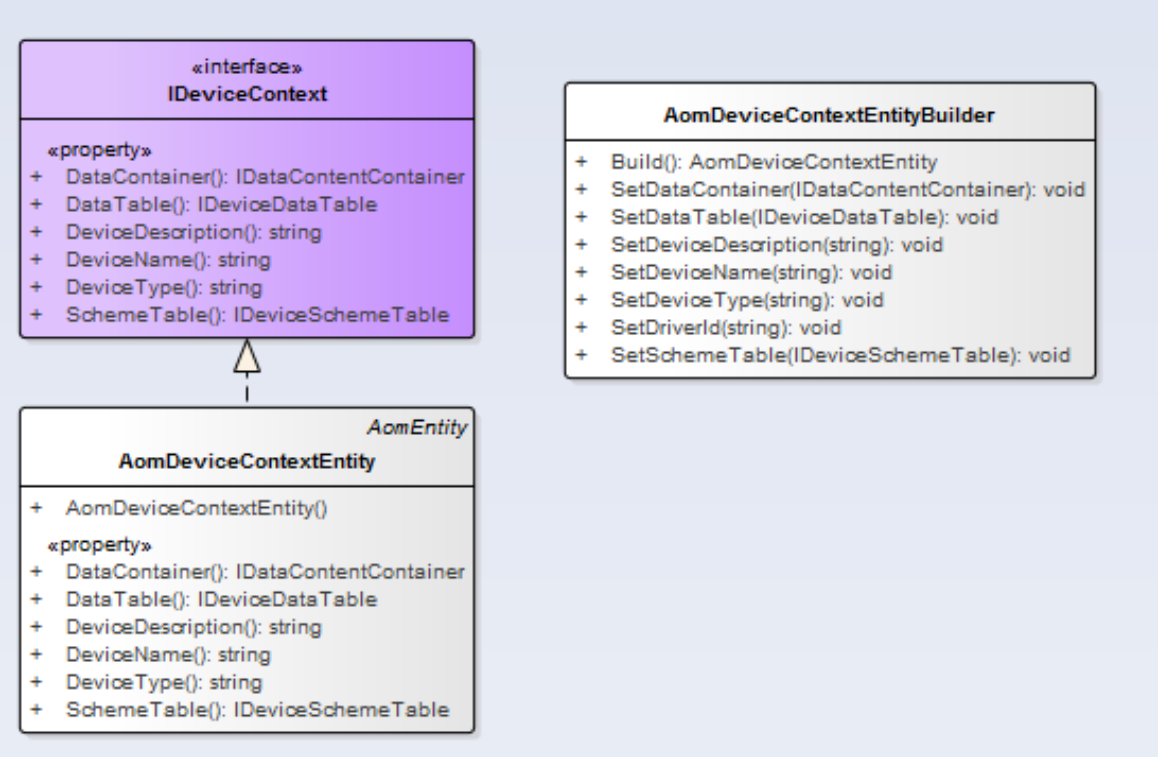

Рисунок 4.3 – Диаграмма классов, описывающая контекст устройства

Класс AomDeviceContextEntity наследуется от AomEntity и реализует IDeviceContext. Для создания объектов данного класса применяется паттерн «Строитель» (англ. Builder), представленный классом AomDeviceContextEntityBuilder. Данный класс содержит набор «Set»-методов соответствующий свойства класса AomDeviceContextEntity и метод Build, возвращающий новый объект класса AomDeviceContextEntity.

Диаграмма классов, описывающая функционал таблицы данных хранимых на устройстве, представлена на рисунке 4.4. Данный функционал описывается в интерфейсе IDeviceDataTable. Интерфейс описывает четыре метода и индексатор. Метод AddRow описывает метод добавления строки данных типа IDeviceDataTableRow, описанного далее. Метод GetDataRow возвращает строку данных соответствующую имени, переданному как

параметр. Для доступа к строке данных по имени, также можно использовать индексатор. Метод GetEnumeratorRows возвращает коллекцию строк данных, хранимых в данной таблице. Класс DefaultDeviceDataTable реализует IDeviceDataTable.

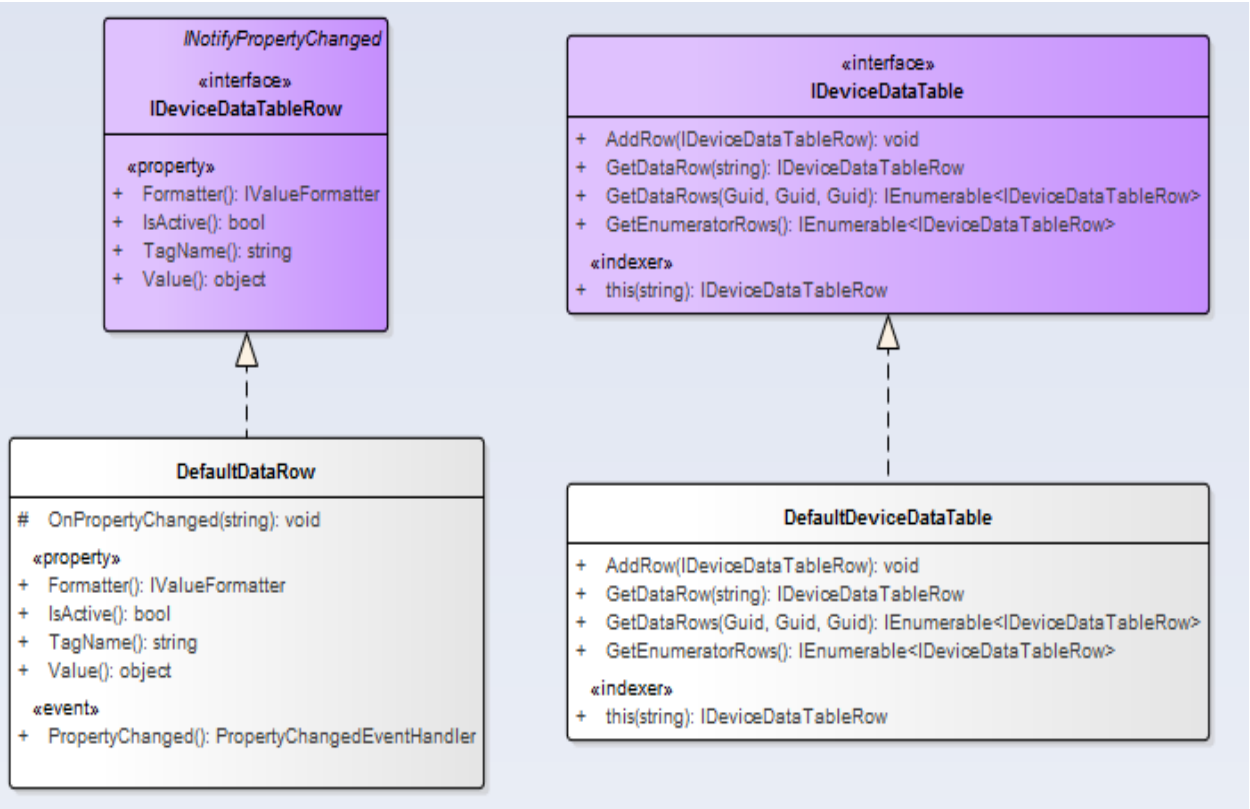

Рисунок 4.4 – Диаграмма классов, описывающая функционал таблицы данных хранимых на устройстве

Интерфейс IDeviceDataTableRow описывает строку данных в таблице хранимых устройством данных. Данные интерфейс описывает четыре свойства. Свойство Formatter содержит класс реализующий функционал перевода значения между байтовым представлением значения и его истинным типом. Свойство IsActive сигнализирует о корректности данных в зависимости от конфигурации устройства. Свойство TagName представляет имя строки данных. Свойство Value содержит последнее считанное значение из этой строки данных.

Класс DefaultDataRow реализует интерфейс IDeviceDataTableRow и представляет собой стандартную реализацию строки данных. Кроме свойств, описанных ранее, данный класс объявляет событие PropertyChanged, которое сигнализирует о изменении свойства данного класса. Метод OnPropertyChanged принимает имя свойства, и потокобезопасно вызывает событие PropertyChanged, если на него кто-либо подписался.

Диаграмма классов, описывающая схему устройства, представлена на рисунке 4.5. Данный функционал описывается в интерфейсе IDeviceSchemeTable. Интерфейс IDeviceSchemeTable описывает восемь

методов. Методы AddResistorRow и AddStarterRow предназначены для добавления в схему устройства объектов пускателя и предохранителя соответственно. Методы GetStarterRow и GetResistorRow предназначены для получения пускателя, и предохранителя соответственно, по уникальному идентификатору сущности. Методы GetStarterByIndex и GetResistorByIndex предназначены для получения пускателя, и предохранителя соответственно, по его индексу в списке сущностей. Методы GetStarterEnumerable и GetResistorEnumerable предназначены для получения списка всех пускателей и предохранителей соответственно.

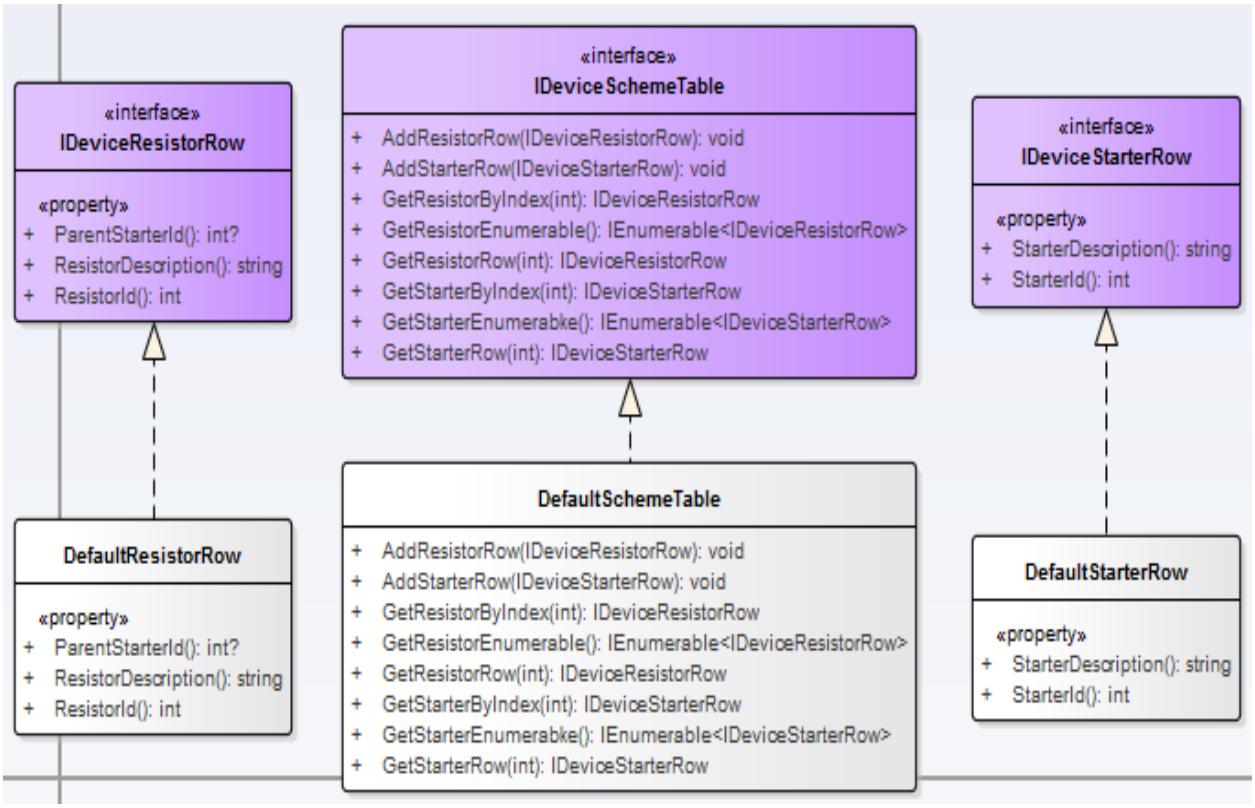

Рисунок 4.5 – Диаграмма классов, описывающая схему устройства

Интерфейс IDeviceResistorRow описывает предохранитель в схеме устройства. Данных интерфейс описывает три свойства. ResistorId является уникальным идентификатором предохранителя. ResistorDescription содержит текстовое описание. ParentStarterId может хранить идентификатор пускателя, с которым связан данный предохранитель.

Интерфейс IDeviceStarterRow описывает пускатель в схеме устройства. Данных интерфейс описывает два свойства. StarterId является уникальным идентификатором пускателя. StarterDescription содержит текстовое описание.

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Завершающим этапом разработки программного средства является тестирование, которое представляет собой процесс исследования приложения с целью получения информации о качестве программного продукта.

5.1 Виды тестирования

В современной практике принято выделять следующие методики испытаний приложения:

1) блочное тестирование – тестирование класса, отдельного метода или небольшого приложения, написанного одним программистом, выполняемое отдельно от прочих частей системы;
2) тестирование компонента – это тестирование класса, пакета, небольшого приложения или другого элемента системы, разработанного несколькими программистами, выполняемое в изоляции от остальных частей системы;
3) интеграционное тестирование – это совместное выполнение тестирования двух и более классов, пакетов, компонентов или подсистем, созданных несколькими программистами (данный вид тестирования обычно начинают проводить, как только созданы два класса, которые можно протестировать, и продолжают до завершения работы над системой);
4) регрессивное тестирование – повторное выполнение тестов, направленное на обнаружение дефектов в программе, уже прошедших этот набор тестов;
5) тестирование системы – выполнение программного обеспечения в его окончательной конфигурации, интегрированного с другими программными и аппаратными системами.

При тестировании системы в целом, предметом тестирования являются безопасность, производительность, утечка ресурсов, проблемы синхронизации и прочие аспекты, которые невозможно протестировать на более низких уровнях интеграции.

Тестирование обычно разделяют на две обширных категории:

1) тестирование методом черного ящика;
2) тестирование методом белого (прозрачного) ящика.

В первом случае тестировщик не владеет сведениями о внутренней работе тестируемого элемента, в случае же белого ящика внутренняя реализация тестируемого элемента изначально известна тестировщику. В данном разделе рассматривается тестирование методом прозрачного ящика, так как объектом тестирования являлся разработанный исходный код программного средства.

Одним из примеров тестирования методом белого ящика является структурированное базисное тестирование, в основе которого лежит

следующая идея: необходимо протестировать каждый оператор программы хотя бы один раз. Если оператор является логическим, то должны учитываться сложность анализируемого выражения внутри условия, чтобы оператор был протестирован полностью. Наиболее простым способом покрыть все возможные варианты, является расчет числа возможных путей выполнения программы и создание минимального набора тестов, проверяющих каждый путь.

При возрастании сложности метода число тестов, нужных только для покрытия всех путей, быстро увеличивается. Соответственно, более короткие методы обычно включают меньше путей, которые необходимо протестировать.

Описанный метод тестирования применялся для проверки корректности работы отдельных частей программного средства, таких как загрузка документа с удаленного сервера.

Систематичный подход к тестированию позволяет находить максимальное число дефектов всех типов, при этом должны соблюдаться следующие правила:

1) тестирование программы производится на предмет реализации каждого существенного требования (планируется на стадии выработки требований);

2) тестирование программы производится на предмет реализации каждого значимого аспекта проектирования (тесты планируются на стадии проектирования);

3) используется «базисное тестирование», которым дополняются тесты требований и проекта детальными тестами;

На практике произвести исчерпывающее тестирование не представляется возможным, поэтому происходит выбор отдельных тестов, способных обеспечить максимальную вероятность обнаружения ошибок. Набор тестов определяется для основных модулей и всего приложения в целом.

Целью любого тестирования является нахождение ошибок. Успешным считается тест, нарушающий работу программного средства. Все остальные этапы разработки направлены на предотвращения ошибок и недопущению нарушения работы программы. Сам процесс тестирование не доказывает отсутствие ошибок и не приводит к повышению качества программного средства.

При разработке программного средства управления городским освещением использовалось ручное тестирование. Для функциональных требований разработаны сценарии, как позитивного поведения, так и негативного. Тестирование программного средства производится по методу черного ящика.

5.2 Тестирование приложения в режиме конфигурирования

Сценарии использования программного средства управления городским освещением в режиме конфигурирования не зависят от качества связи с устройством и независимы от возможных ошибок в самом устройстве РУНО-3. Функции, поддерживаемые в данном режиме, направлены на конфигурирование программного средства на работу с удаленными устройствами. В данном режиме можно создать, редактировать или удалить описание удаленного устройства. Также можно изменить положением схематических изображений устройств на рабочей области. Присутствует возможность изменить пароль доступа к функция изменения графиков и конфигурирования на устройствах, а также возможно изменить периоды обновлений.

Разработаем и проведем тесты, показывающие работоспособность функциональности предусмотренной в режиме конфигурирования.

Таблица 5.1 – Тест-кейсы для ПС в режиме конфигурирования

№ тест-кейса	Название тест-кейса	Ожидаемый результат	Результат выполнения
1	2	3	4
1	**Переход на форму создания нового устройства** 1. Выбрать пункт «Устройство» в главном меню 2. Выбрать пункт «Новое» в меню «Устройство»	1. Отображение меню «Устройство» 2. Отображение формы создания нового устройства	Тест пройден
2	**Корректное создание нового устройства** 1. В форме создание нового устройства ввести корректные данные 2. Кликнуть по кнопке «Создать»	1. Поля ввода после введения не данных подсвечиваются красным 2. Схематическое изображение устройства добавляется на форму списка устройств	Тест пройден

Продолжение таблицы 5.1

1	2	3	4
3	**Проверка вводимых данных в форму создания устройства** 1. В форме создание нового устройства ввести некорректные данные 2. Кликнуть по кнопке «создать»	1. Поля ввода с неверно введенными данными подсвечиваются красным 2. Кнопка «Создать» не активна	Тест пройден
4	**Отмена создания нового устройства** 1. В форме создания нового устройства ввести корректные данные 2. Кликнуть по кнопке «Отмена»	1. Нет сообщений о неверно введенных данных 2. Переход к форме списка устройств. На данной форме отсутствуют новые устройства	Тест пройден
5	**Редактирование устройства** 1. Выбрать устройство на форме списка устройств 2. Выбрать пункт «Устройство» в главном меню 3. Выбрать пункт «Редактировать» в меню «Устройство» 4. Корректно изменить данные описывающие устройство 5. Кликнуть по кнопке «Редактировать»	1. Выбранное устройство графически выделяется 2. Отображение меню «Устройство» 3. Отображение формы редактирования с существующими на данный момент данными устройства 4. Нет сообщений о неверно введенных данных 5. Измененные данные отображаются на схематическом изображении устройства на форме списка устройств	Тест пройден

Продолжение таблицы 5.1

1	2	3	4
6	**Удаление устройства** 1. Выбрать устройство на форме списка устройств 2. Выбрать пункт «Устройство» в главном меню 3. Выбрать пункт «Удалить» в меню «Устройство» 4. Подтвердить удаление во всплывающем окне	1. Выбранное устройство графически выделяется 2. Отображение меню «Устройство» 3. Отображение всплывающего окна для подтверждения удаления 4. Удаление устройства из списка всех устройств	Тест пройден
7	**Отмена удаления устройства** 1. Выбрать устройство на форме списка устройств 2. Выбрать пункт «Устройство» в главном меню 3. Выбрать пункт «Удалить» в меню «Устройство» 4. Отменить удаление во всплывающем окне	1. Выбранное устройство графически выделяется 2. Отображение меню «Устройство» 3. Отображение всплывающего окна для подтверждения удаления 4. Удаление устройства остается в списке всех устройств	Тест пройден

Продолжение таблицы 5.1

1	2	3	4
8	**Изменение расположения элементов** 1. Выделить и перетащить устройство в новое положение 2. Выбрать пункт «Устройство» в главном меню 3. Выбрать пункт «Сохранить расположение элементов» в меню «Устройство» 4. Перейти в режим реального времени и вернуться в режим конфигурирования	1. Положение элемента в списке устройств изменилось 2. Отображение меню «Устройство» 3. Форма списка устройств не изменилась 4. После возвращения в режим конфигурирования список устройств соответствует измененному состоянию	Тест пройден
9	**Изменение расположения элементов без сохранения** 1. Выделить и перетащить устройство в новое положение 2. Перейти в режим реального времени и вернуться в режим конфигурирования	1. Положение элемента в списке устройств изменилось 2. После возвращения в режим конфигурирования список устройств соответствует неизмененному состоянию	Тест пройден
10	**Попытка изменить пароль введя неверный текущий** 1. Выбрать пункт «Настройки» в главном меню 2. Выбрать пункт «Изменить пароль» в меню «Настройки» 3. Ввести неверный текущий пароль в всплывающее окно и кликнуть по кнопке «Да»	1. Отображение меню «Настройки» 2. Отображение всплывающего окна ввода текущего пароля. 3. Отображение всплывающего «Нет доступа»	Тест пройден

Продолжение таблицы 5.1

1	2	3	4
11	**Изменение пароля** 1. Выбрать пункт «Настройки» в главном меню 2. Выбрать пункт «Изменить пароль» в меню «Настройки» 3. Ввести текущий пароль в всплывающее окно и кликнуть по кнопке «Да» 4. Ввести новый пароль в всплывающее окно и кликнуть по кнопке «Да» 5. Проверить доступ к защищенным паролем функциям с помощью нового пароля	1. Отображение меню «Настройки» 2. Отображение всплывающего окна ввода текущего пароля. 3. Отображение всплывающего окна ввода нового пароля 4. Возвращение к предыдущей форме 5. Доступ к защищенным паролем функциям осуществляется по измененному паролю	Тест пройден

5.3 Тестирование приложения в режиме реального времени

Сценарии использования программного средства управления городским освещением в режиме реального времени в большинстве случаев связаны с взаимодействием с удаленным устройством. Одной из основных функций данного режима является мониторинг состояния удаленного устройства. В режиме реального времени есть возможность отправки команд на каждое устройство в отдельности, либо на все, которые поддерживают данную команду, посредством общегородских команд. Присутствует возможность управления графиками освещения и конфигурацией устройства с помощью операций чтения и записи на/из устройства или файл. Разработаем и проведем тесты, показывающие работоспособность функциональности предусмотренной в режиме реального времени.

Таблица 5.2 – Тест-кейсы для ПС в режиме реального времени

№ тест-кейса	Название тест-кейса	Ожидаемый результат	Результат выполнения
1	2	3	4
1	**Мониторинг состояний устройства** 1. Соотносить состояние устройства, отображаемое в программе, с его фактическим состоянием	1. Состояние устройства, отображаемое в программе, должно совпадать с фактическим	Тест пройден
2	**Отправка команды на устройство** 1. Отправить команду на устройство	1. Состояние устройства должно измениться	Тест пройден
3	**Отправка общегородской команды** 1. Выбрать пункт «Общегородские команды» в главном меню 2. Выбрать пункт «Общегородские команды» в меню «Общегородские команды» 3. Отправить общегородскую команду	1. Отображение меню «Общегородские команды» 2. Отображение формы общегородских команд. 3. Состояние устройств поддерживающих команду должно изменится	Тест пройден
4	**Отправка команды на устройство находящееся в режиме ремонта** 1. Отправить команду на устройство находящееся в режиме ремонта	1. Отображение окна с сообщением о том, что команда не отправлена, т.к. устройство находится в режиме ремонта	Тест пройден

Продолжение таблицы 5.2

1	2	3	4
5	**Запись графика освещения в устройство** 1. Ввод эталонных данных в форму графиков освещения 2. Отправка графика освещения на устройство 3. Сравнить данные записанные в устройство с эталонными	Данные о графике освещения, записанные на устройство, должны совпадать с ожидаемыми	Тест пройден
6	**Чтение графика освещения из устройства** 1. Кликнуть по кнопке «Читать» на форме графиков освещения 2. Сравнить считанные данные с эталонными	Данные о графике освещения, считанные с устройства, должны совпадать с ожидаемыми	Тест пройден
7	**Запись конфигурации на устройство** 1. Ввод эталонных данных в форму конфигурации устройства 2. Отправка конфигурации на устройство **3.** Сравнить данные записанные в устройство с эталонными	Данные о конфигурации, записанные на устройство, должны совпадать с ожидаемыми	Тест пройден
8	**Чтение конфигурации из устройства** 1. Кликнуть по кнопке «Читать» на форме графиков освещения 2. Сравнить считанные данные с эталонными	Данные о конфигурации считанные с устройства должны совпадать с ожидаемыми	Тест пройден

6 ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

6.1 Установка программы

Установка программы должна выполнятся под учетной записью, которая обладает правами администратора на компьютере. Установка производится путем запуска файла SetupULA.msi. При установке необходимо следовать всем стандартным msi-инструкциям.

6.2 Режим реального времени

При запуске ULA загружается режим реального времени.

Информацию о том, в каком режиме находится ULA можно увидеть в верхней части экрана. Голубым цветом подписан режим работы программы (Рисунок 6.1)

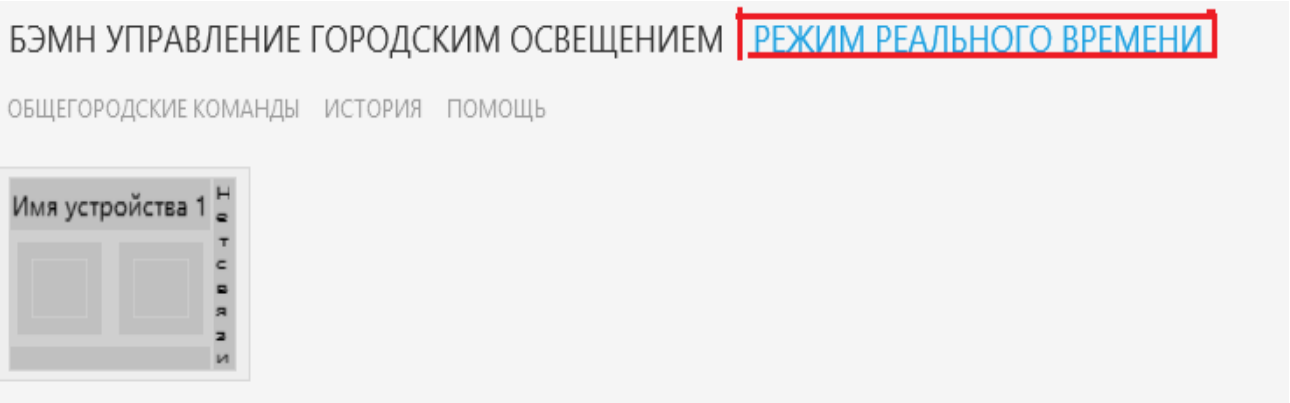

Рисунок 6.1 – Определение режима работы программы.

В режиме реального времени можно:

1) отслеживать изменения состояния устройств в текущий момент времени;

2) отсылать общегородские команды всем устройствам одновременно;

3) изменять время на всех устройствах одновременно;

4) переходить к схеме устройства и управлять им отдельно от остальных.

При первом запуске и отсутствующей базе данных устройств, в режиме реального времени загрузится пустой экран и будет предложено перейти в режим конфигурирования для добавления новых устройств. Если база устройств уже создана, то на экране схематически отобразятся устройства и их состояния.

ULA в Режиме Реального Времени представляет собой поле, на котором схематически изображены устройства, подключенные по GSM связи (Рисунок 6.2). Количество отображенных устройств на общем фрагменте не превышает 105 штук. Состояние устройств опрашивается с

периодом, указанным в Режиме конфигурирования. При необходимости этот период можно изменить.

Каждое устройство схематически изображено как прямоугольник (Рисунок 6.3), с выделенными областями под название, состояние пускателей, посланную команду и состояние устройства (авария, ремонт, нет связи).

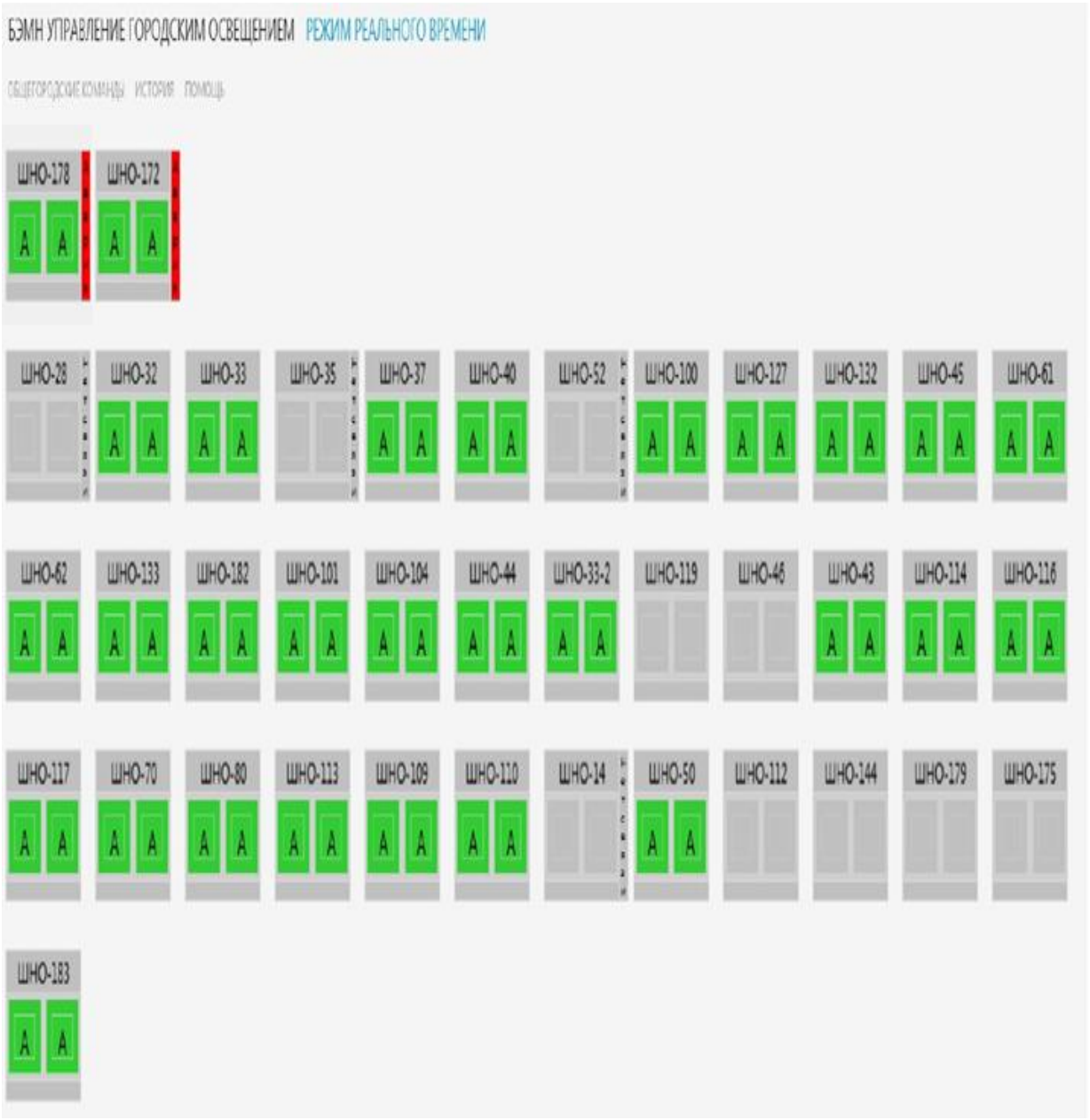

Рисунок 6.2 – Форма списка устройств в режиме реального времени

На рисунке 6.4 рассмотрены различные положения и состояния устройства.

В верхней части экрана в режиме реального времени находится меню, из которого можно вызвать общегородские команды, просмотреть историю событий ULA и вызвать файл справки.

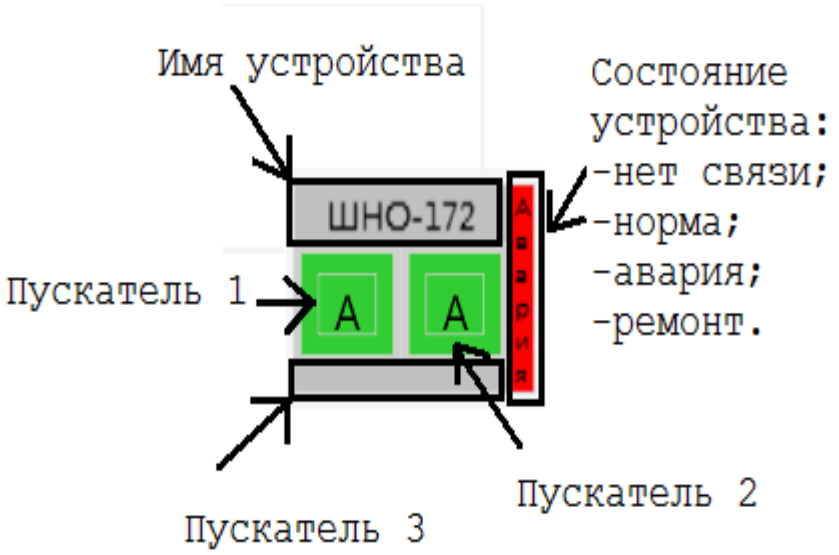

Рисунок 6.3 – Схематическое изображение устройства

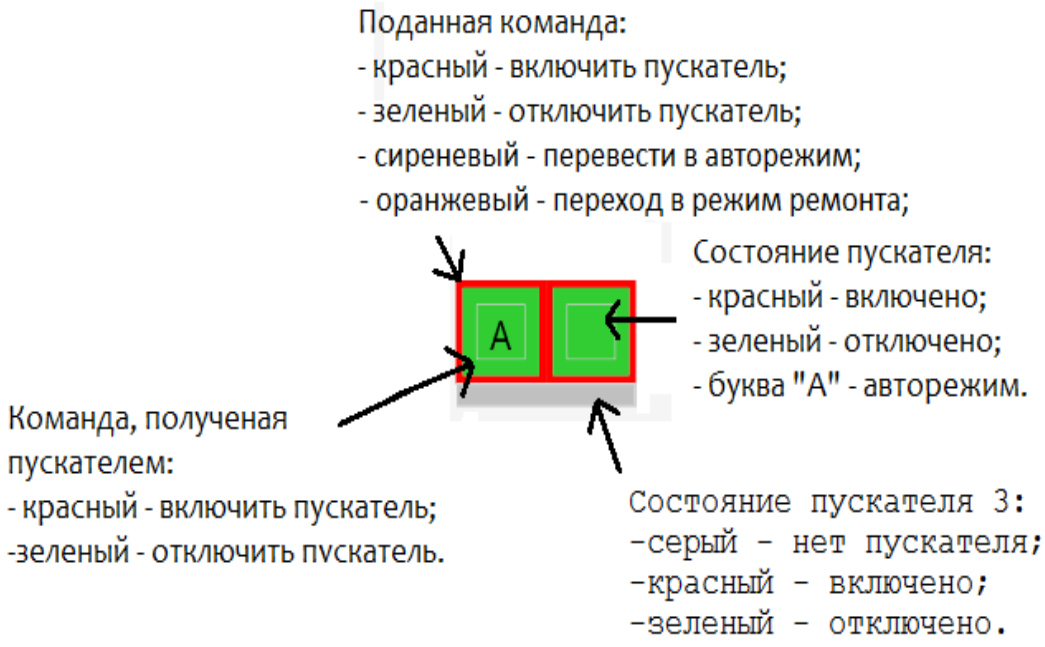

Рисунок 6.4 – Положения и состояния устройства

Общегородские команды используются для отправки команд управления на все подключенные устройства одновременно. Форма общегородских команд представлена на рисунке 6.5.

Могут использоваться следующие команды:

1) команды на включение или отключение ночного режима освещения;

2) команды на включение или отключение вечернего режима освещения;

3) команды на включение или отключение полного режима освещения;

4) команды на включение или отключение режима подсветки (срабатывают только если пускатели сконфигурированы соответствующим образом);

5) команды на включение или отключение режима иллюминации (срабатывают только если пускатели сконфигурированы соответствующим образом);

6) команды на включение или отключение режима энергосбережения (срабатывают только если пускатели сконфигурированы соответствующим образом);

7) команда для перевода всех устройств в авторежим.

При подаче команд управления, на общей схеме тонкой рамкой отображается посланная команда:

1) зеленой – команда на отключение;

2) красной – команда на включение;

3) фиолетовой – переход в авторежим;

4) желтой – переход в режим ремонта.

Если команда посылается на третий пускатель, то на общей схеме это не отображается. Цвет изменяется только от изменения фактического состояния.

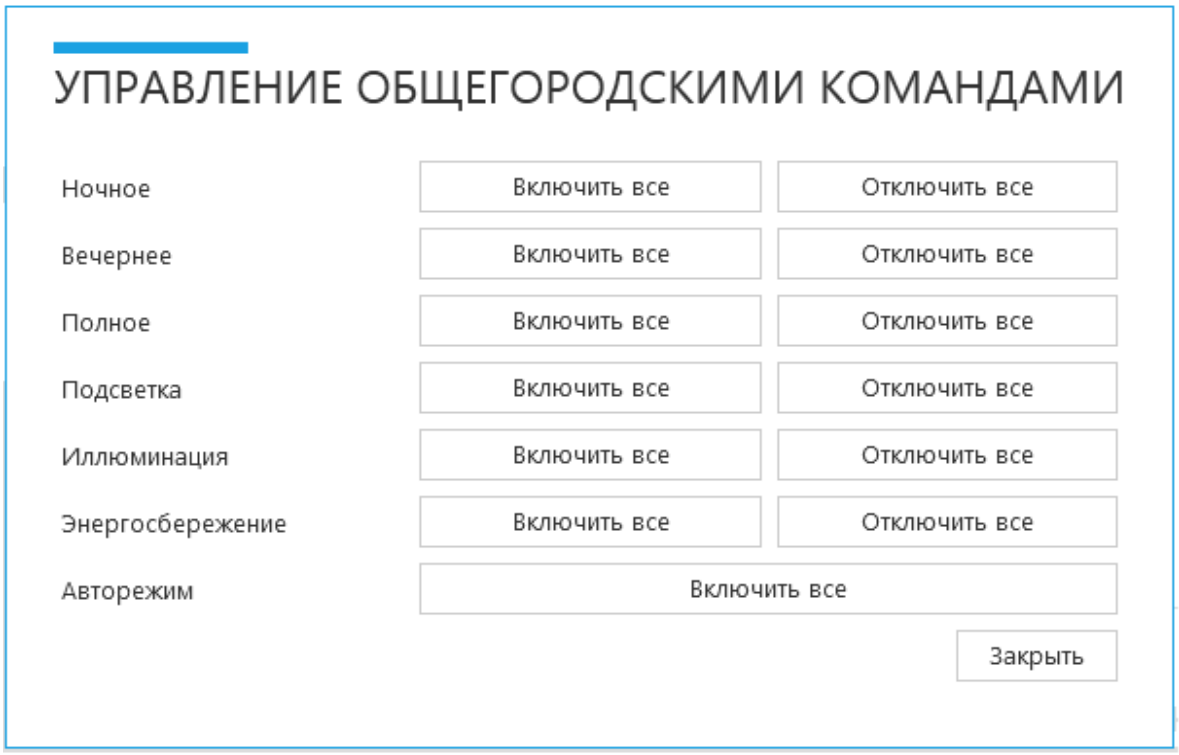

Рисунок 6.5 – Форма общегородских команд

Из пункта меню «общегородские команды» также можно перейти на форму «Синхронизация времени», представленную на рисунке 6.6, и послать соответствующую команду всем устройствам одновременно.

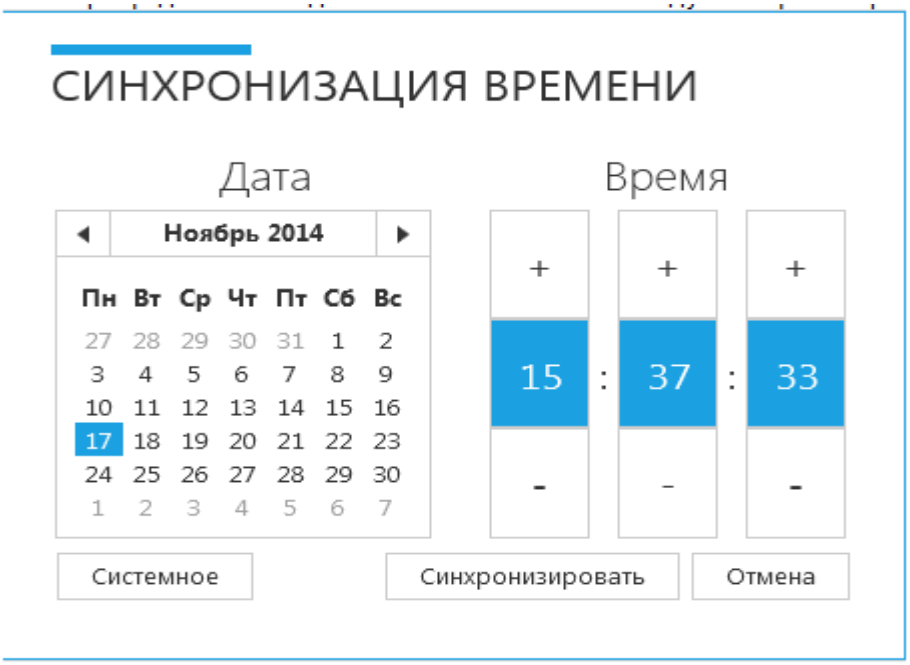

Рисунок 6.6 – Форма синхронизации времени

Чтобы изменить дату/время нажмите кнопку «Установить время». В появившемся окне выберите или введите нужные дату/время а затем кнопку «Синхронизировать». Для установки системного времени просто нажмите «Системное». Запросы на изменение времени идут раз в 30-40 сек, поэтому подождите немного, чтобы увидеть смену даты/времени. Допускается отклонение времени в ULA от системного времени на компьютере до 10 сек. Это связано с очередностью отправки запросов по GSM- каналу.

6.2.1 Схема устройства

В режиме реального времени по двойному клику на схематическом изображении ШНО осуществляется переход к схеме РУНО-3. Форма схемы устройства представлена на рисунке 6.7.

Конфигурация каждого РУНО-3 автоматически считывается из устройства. И в соответствии с этой конфигурацией рисуется схема устройства. При открытой схеме устройства запросы к РУНО-3 шлются чаще и в большем количестве (по сравнения с режимом реального времени). Период обновления информации на открытой схеме можно изменить в режиме конфигурации. Меню-настройки-Период обновлений (схема).

На схеме изображены:

1) панель управления;
2) пускатели;
3) рубильник;

4) предохранители;
5) дата/время;
6) уровень сигнала;
7) показания счетчиков;
8) неисправности;
9) панель конфигурирования.

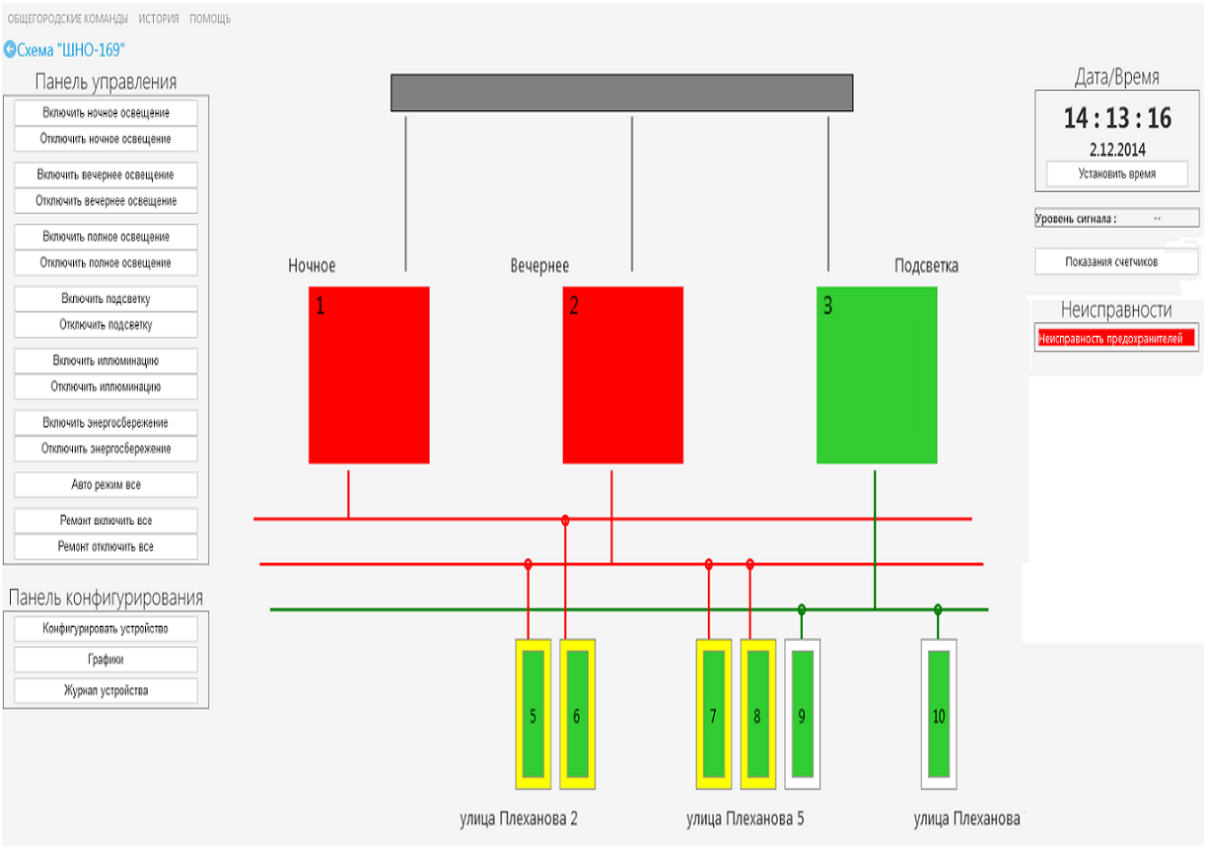

Рисунок 6.7 – Форма схемы устройства

Вверху голубым цветом указано имя устройства. Под ним находится панель управления устройством.

Можно, независимо от других устройств:

1) включить или отключить ночное (вечернее, полное) освещение;

2) включить или отключить другие режимы управления (подсветка, иллюминация, энергосбережение), если устройство сконфигурировано нужным образом;

3) перевести устройство в авторежим;

4) перевести устройство в режим ремонта.

В центральной части схематически прорисованы пускатели и прикрепленные к ним предохранители с названиями, которые задаются в режиме Конфигурирования. Над пускателями находится рубильник. Он может быть серым, если не введен в конфигурации. Зеленым, если

установлен в конфигурации и отключен. Красным, если установлен в конфигурации и включен.

С пускателями в соответствии с конфигурацией соотнесены предохранители. Они могут иметь следующий вид:

1) предохранитель не под напряжением;

2) предохранитель под напряжением;

3) предохранитель не под напряжением, но должен быть под напряжением (ненормальное состояние – горит индикатор неисправность).;

4) предохранитель под напряжением, но должен быть не под напряжением (ненормальное состояние – горит индикатор неисправность).

В правой части экрана отображаются дата и время, считываемые с РУНО-3.

Под датой/временем можно посмотреть значение уровня сигнала РУНО-3 в относительных единицах, которое показывает качество сигнала GSM. Допустимые значения от 0 до 36. Значения 0-5 – недостаточный сигнал, 36 единиц – наилучший сигнал, 99 – невозможно считать значение. Значение вида "- -" говорит о том, что в данной версии РУНО-3 невозможно определить качество GSM сигнала.

Еще ниже расположены индикаторы неисправностей.

По нажатию на кнопку "Показания счетчика", можно получить данные с подключенного счетчика (фазные напряжения, фазные токи, мощности по фазам и активная потребленная энергия нарастающим итогом суммарно по всем тарифам).

Панель конфигурирования дает доступ к конфигурации РУНО-3, к графикам, зашитым в РУНО-3 и к журналу сообщений РУНО-3. Для доступа к этой информации используется пароль.

Из схемы устройств осуществляется доступ к графикам, хранящимся в РУНО-3. Пример формы графика освещения представлен на рисунке 6.8. В вкладках «ГРАФИК ОСВЕЩЕНИЯ», «ГРАФИК ПОДСВЕТКИ», «ГРАФИК ЭНЕРГОСБЕРЕЖЕНИЯ» задается время выключения/включения по дням и месяцам года. В левой верхней части окна перечислены месяцы года. Кликая по ним, справа отображается график для каждого числа этого месяца. Для удобства в программе имеется функция сохранять записанный график в файл, а также открывать уже сохраненные данные по графикам. Чтобы записать созданный график в устройство, необходимо нажать кнопку «Записать», предварительно открыв его. Чтобы просмотреть график, записанный в устройстве необходимо нажать кнопку «Прочитать». Также имеется возможность, к основному графику добавить график «Экономия». Он задается на определенный период года. Чтобы добавить его, необходимо поставить флаг в графе «Экономия». Далее изменить период, в котором будет применена экономия: выбрать дату включения режима экономии и дату выключения (число/месяц). Тоже сделать со временем: время включения/выключения режима экономии – это фактическое время

отключения канала. В указанный диапазон дат экономия будет осуществляться на выбранном промежутке времени.

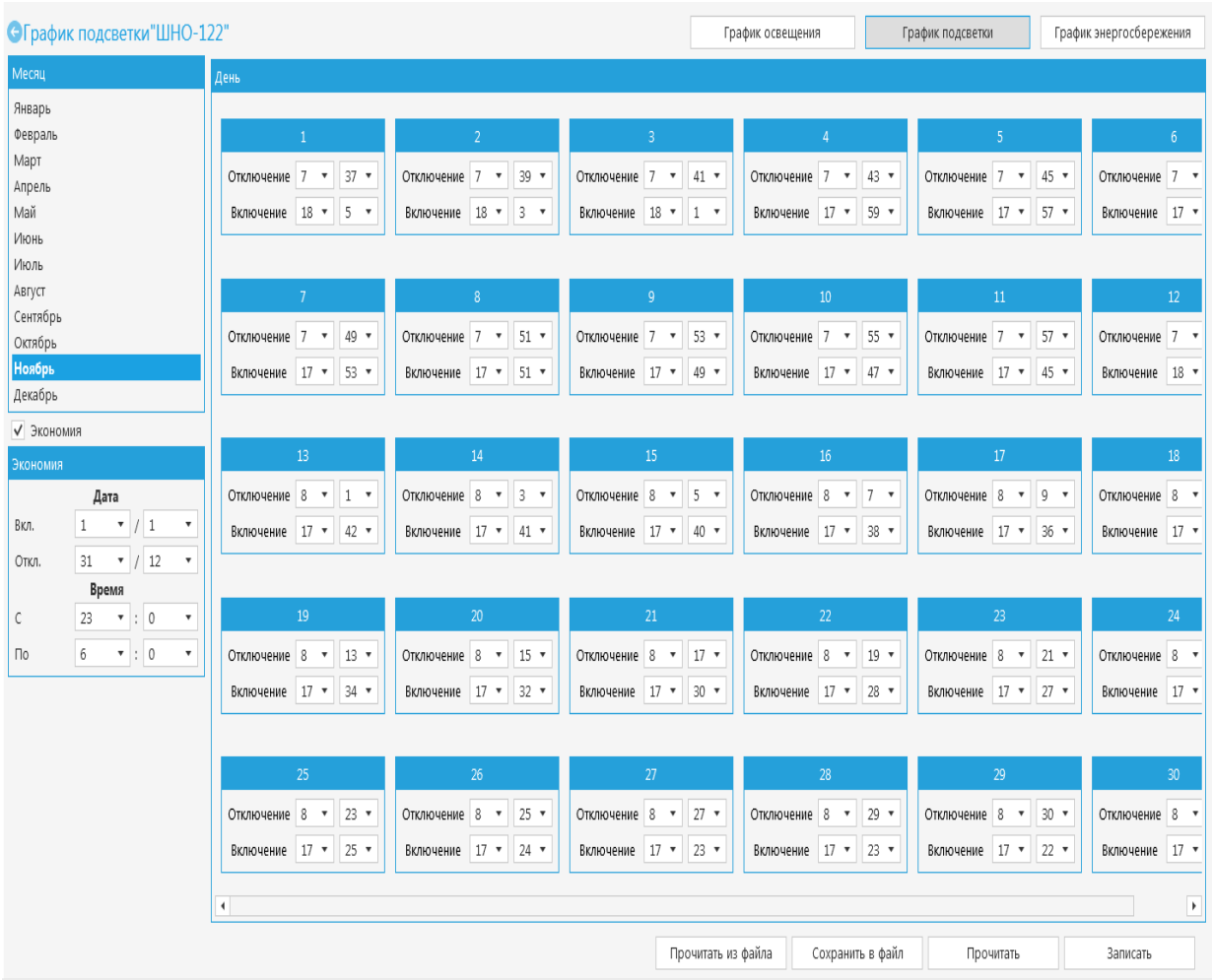

Рисунок 6.8 – Форма графика освещения

6.3 Режим конфигурирования

Режим конфигурирования используется для добавления новых устройств. Переход в этот режим осуществляется из режима реального времени по сочетанию клавиш Ctrl+F5. Возврат в режим реального времени по нажатию F5.

В режиме конфигурирования можно выполнять следующие операции:

1) добавление нового устройства;
2) удаление устройства;
3) редактирование существующих устройств;
4) перемещение по экрану элементов;
5) изменение периода опроса устройств;
6) изменения пароля для доступа к конфигурации РУНО;
7) просмотр журнала событий ULA.

Для добавления нового устройства выбираем из меню «Устройство» пункт «Новое». Форма создания нового устройства представлена на рисунке 6.9.

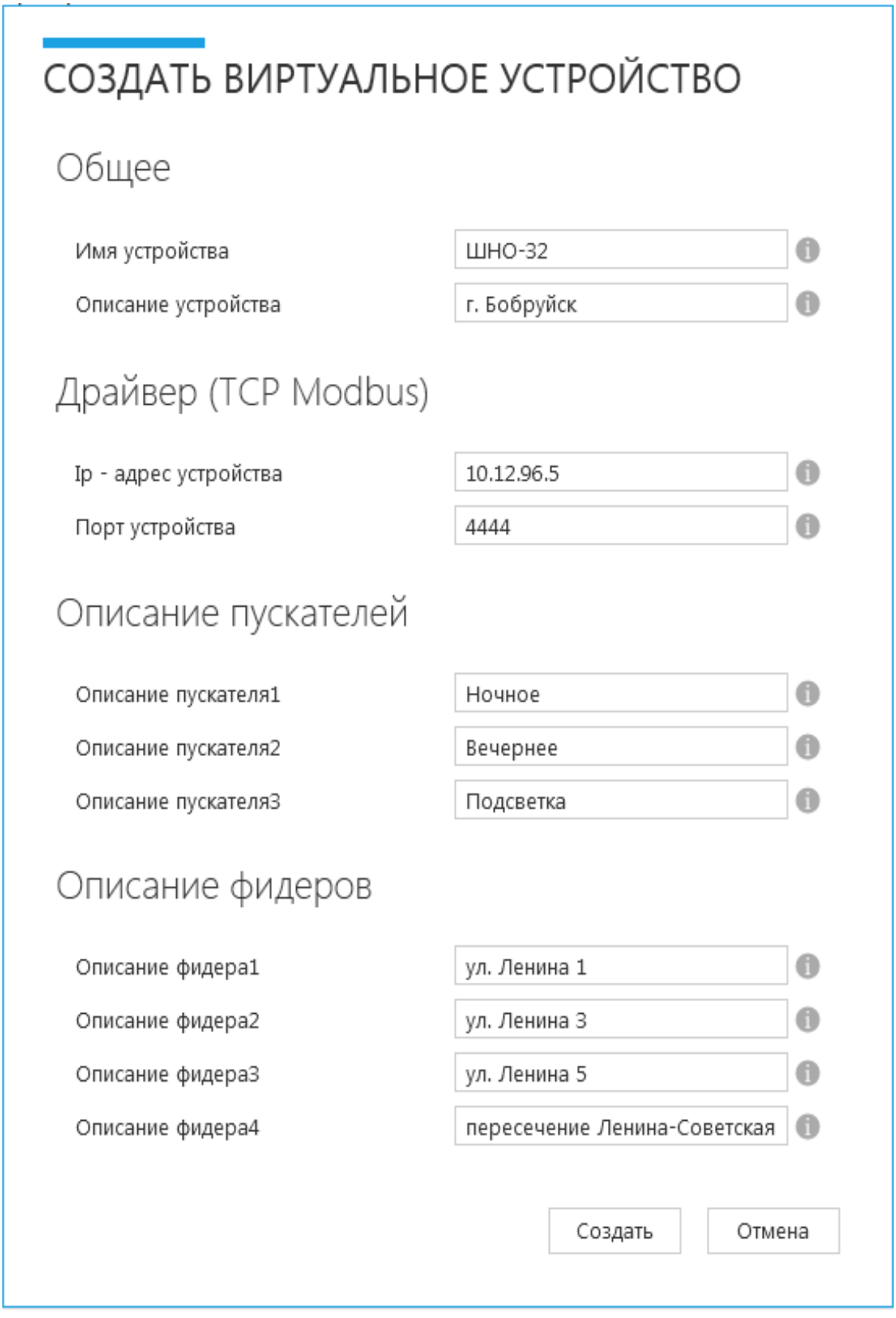

Рисунок 6.9 – Форма создания нового устройства

В открывшемся окне вносим необходимую информацию:

Имя устройства – наименование, которое будет отображаться в верхней части схематического изображения устройства. Описание устройства – будет показыватся при наведении курсора на устройство на общей схеме. Здесь может задавата информация о местоположении ШНО. IP-адрес – адрес РУНО-3 с которым нужно связаться. Порт устройства указывается по умолчанию 4444. Описание пускателей отображается на схеме устройства. Используется для обозначения назначения каждого пускателя. Описание фидеров (группы предохранителей) – используется для обозначения местоположения ШНО. Заполнять можно все поля, если какой-то из пускателей или фидеров не присутствует в схеме, то это название не будет нигде отображаться. В будущем, при изменении конфигурации, нужное название отобразится на экране.

Для удаления или редактирования существующего устройства, нужно выделить его кликом мышки, а затем выбрать нужный пункт меню. На рисунке ниже выделен элемент с именем ШНО-40.

Для более удобного визуального контроля, все элементы можно перемещать по экрану и, при необходимости группировать. Для перетаскивания устройства – наведите на него курсор, нажмите левую клавишу мыши, и не опуская клавишу, переместите курсор в нужное место. Устройство переместится. Для сохранения нужного расположения элементов, выберите в меню Устройство пункт «Сохранить расположение элементов».

В программе, по умолчанию, используется период опроса равный двадцати секундам для подключенных устройств, для режима реального времени и пять секунд опрос данных одного устройства, если открыта схема устройства. Для изменения этих значений выберите в меню Настройки пункт Период обновлений (список) или Период обновлений (схема).

В открывшемся окне поставьте требуемое значение в пределах от пяти до шестидесятисекунд. Чем меньше это значение, тем чаще будут посылаться запросы от программы к устройствам, и тем чаще будет обновляться информация, но трафик при этом будет увеличиваться.

В меню Настройки – «Изменить пароль» можно задать свой пароль для доступа к такой информации как Конфигурация РУНО-3, Графики РУНО-3. Пароль, используемый по умолчанию – bemn. Вначале нужно ввести пароль по умолчанию (или старый пароль), затем новый пароль.

Все действия, выполняемые в программе ULA, сохраняются в журнал событий. Из меню История - Журнал системы можно просмотреть события за определенный период. В строке Действие можно ввести слово для поиска и найти все события по этому слову. Например, просмотреть все события для определенного ШНО. Пример формы журнала системы представлен на рисунке 6.10.

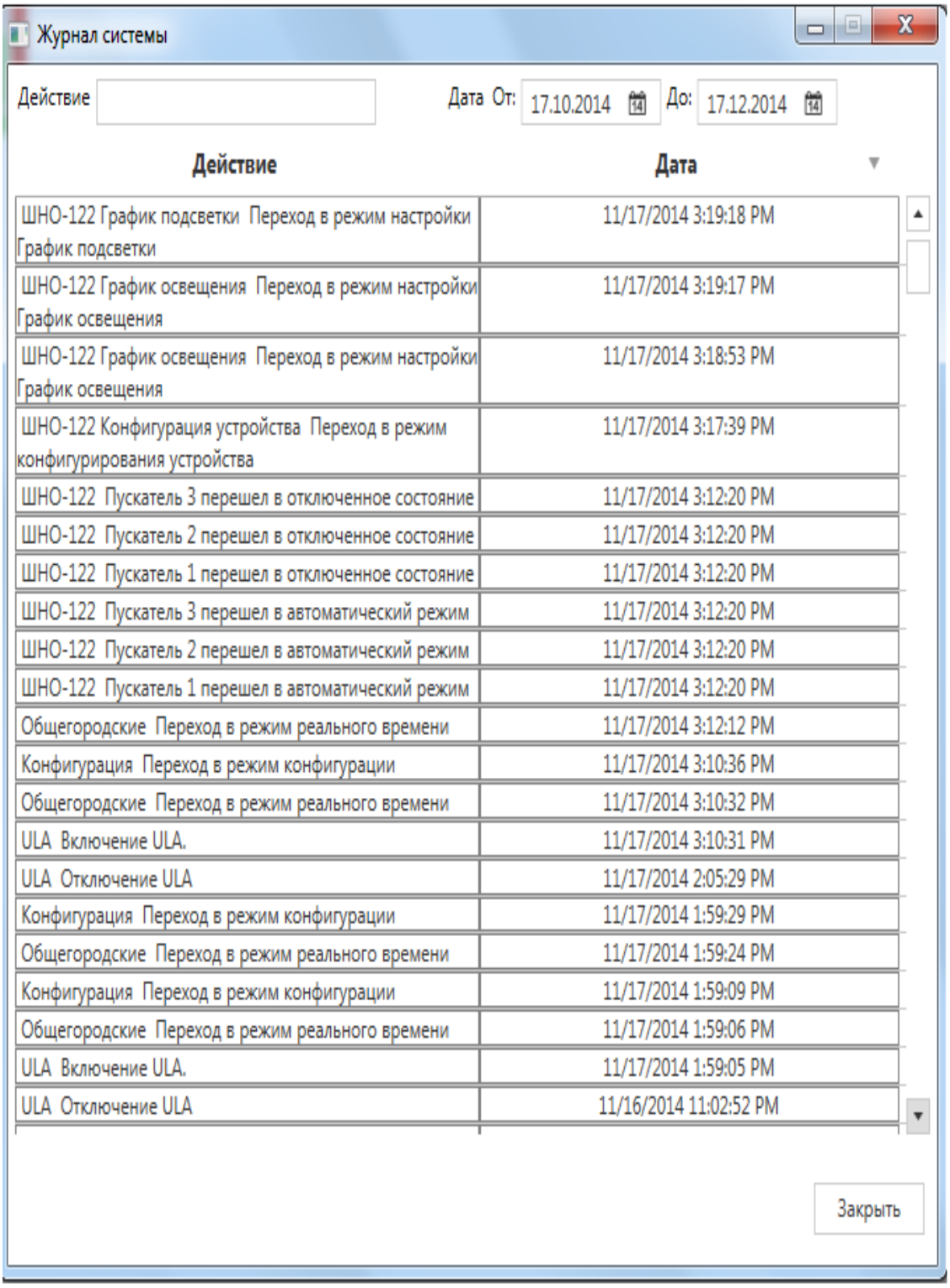

Рисунок 6.10 – Форма журнала системы

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА

7.1 Характеристика программного продукта

Наружное освещение является существенным фактором нормального функционирования различных форм человеческой деятельности и жизнеобеспечения.

На сегодняшний день важно не только использовать современное электрооборудование для обеспечения городского освещения, но и применять автоматизированные средства для контроля и управления городским освещением.

Построение надёжных автоматизированных систем контроля и управления наружным освещением на основе единого системного подхода (и с применением новых информационных технологий) позволит решить многие текущие и будущие проблемы отечественных промышленных предприятий. В настоящий момент регистрация недвижимого имущества осуществляется вручную.

Программный продукт будет разрабатываться Федюковичем Сергеем Александровичем для ОАО «БелЭлектроМонтажНаладка».

Разработка и внедрение данной программы позволят:

– снизить трудоемкость сбора данных о контролируемом технологическом процессе;

– обеспечивать интерфейс с оператором;

– сохраняет историю процесса и осуществляет автоматическое управление процессом;

Экономическая целесообразность инвестиций в разработку и использование программного продукта осуществляется на основе расчета и оценки следующих показателей:

− чистая дисконтированная стоимость (ЧДД);

− срок окупаемости инвестиций (ТОК);

− рентабельность инвестиций (Ри).

В результате разработки данного программного продукта снизится трудоемкость выполнения задачи контроля городского освещения, что и будет являться результатом от внедрения программного продукта.

7.2 Расчет затрат и отпускной цены программного средства

Основная заработная плата исполнителей проекта определяется по формуле:

$$З_о = \sum_{i=1}^{n} Т_{чi} \cdot Т_ч \cdot Ф_{эi} \cdot К, \qquad (7.1)$$

где n – количество исполнителей, занятых разработкой ПС;

$Т_{чi}$ – часовая тарифная ставка i-го исполнителя (руб.);

$Ф_{Эi}$ – эффективный фонд рабочего времени i-го исполнителя (дней);

$Т_{ч}$ – количество часов работы в день (ч);

К – коэффициент премирования (1,2).

В настоящий момент тарифная ставка 1-го разряда на предприятии составляет один миллион четыреста двадцать тысяч руб.

Расчет основной заработной платы представлен в табл. 7.1.

Таблица 7.1 – Расчет основной заработной платы

Категория исполнителя	Разряд	Тарифный коэффициент	Часовая тарифная ставка, тыс. руб.	Трудоемкость, дн.	Основная заработная плата, тыс. руб.
Ведущий программист	15	3,04	24,53	120	23548,8
Руководитель проекта	16	3,72	30,01	65	15607,1
Итого с премией (20%), Зо	-	-	-	-	46987,08

Дополнительная заработная плата исполнителей проекта определяется по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \qquad (7.2)$$

где $Н_Д$ – норматив дополнительной заработной платы (20%)

Дополнительная заработная плата составит:

Зд = 46987,08 тыс ·20/100= 9397,42 тыс руб.

Отчисления в фонд социальной защиты населения и на обязательное страхование ($З_С$) определяются в соответствии с действующими законодательными актами по формуле:

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100}, \quad (7.3)$$

где $Н_{СЗ}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 +0,6%).

Зсз = (46987,08 тыс +9397,42тыс) ·34,6/100=19509,03 тыс руб.

Расходы по статье «Машинное время» ($Р_М$) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

$$Рм=Цм \cdot Тч \cdot Ср, \quad (7.4)$$

где Цм – цена одного машино-часа;
Тч – количество часов работы в день;
Ср – длительность проекта.

Стоимость машино-часа на предприятии составляет 14 тыс руб.. Разработка проекта займет 120 дней Определим затраты по статье “Машинное время”:

Рм=14 тыс·8·120=13440 тыс руб.

Затраты по статье «Накладные расходы» ($Р_Н$), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды ($Р_Н$), определяются по формуле

$$Р_н = \frac{З_о \cdot Н_{рн}}{100}, \quad (7.5)$$

где $Н_{РН}$ – норматив накладных расходов (50%).

Накладные расходы составят:

Рн=46987,08 тыс ·0,5=23493,54 руб.

Общая сумма расходов по всем статьям сметы ($С_п$) на ПО рассчитывается по формуле:

$$С_р = З_{оі} + З_{ді} + З_{сзі} + Р_{мі} + Р_{ні}, \quad (7.6)$$

Ср = 46987,08 тыс + 19509,03 тыс +19509,03 тыс + 13440 тыс + 23493,54 тыс = 122938,68 тыс руб.

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС ($Р_{СА}$), которые определяются по нормативу ($Н_{РСА}$), где $Н_{рас}$ – норматив расходов на сопровождение и адаптацию (20%).

$$H_{рас} = \frac{P_{са}}{C_{р}} \cdot 100, \qquad (7.7)$$

где $Р_{са}$ – расходы на сопровождение и адаптацию ПС в целом по организации (руб.);

$С_Р$ – смета расходов в целом по организации без расходов на сопровождение и адаптацию (руб.).

Рса=122938,68 тыс ·20/100= 24587,74 тыс руб.

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость ПС ($С_П$) определяется по формуле:

$$C_{п} = C_{р} + P_{са}, \qquad (7.8)$$

Сп= 122938,68 тыс + 24587,74тыс = 147526,41 тыс руб.

Прибыль ПС рассчитывается по формуле:

$$П_{о} = \frac{C_{п} \cdot У_{рп}}{100\%}, \qquad (7.9)$$

где $П_{пс}$ – прибыль от реализации ПС заказчику (руб.);

$У_Р$ – уровень рентабельности ПС (25%);

$С_П$ – себестоимость ПС (руб.).

Ппс=147526,41 тыс ·25/100=36887,6 тыс руб.

Прогнозируемая отпускная цена ПС:

$$Ц_{п} = C_{п} + П_{с}, \qquad (7.10)$$

Цп=147526,41 тыс +36887,6 тыс = 184414,01 тыс руб.

7.3 Расчет стоимостной оценки результата

Результатом (Р) в сфере использования программного продукта является прирост чистой прибыли и амортизационных отчислений.

7.3.1 Расчет прироста чистой прибыли

Прирост прибыли за счет экономии расходов на заработную плату в результате снижения трудоемкости выполнения работ по управлению городским освещением.

Экономия затрат на заработную плату при использовании ПС в расчете на объем выполняемых работ определяется по формуле:

$$Э_з = K_{пр} \cdot (t_c \cdot T_c - t_н \cdot T_н) \cdot N_п \cdot (1 + \frac{H_{дп}}{100\%}) \cdot (1 + \frac{H_{нпо}}{100\%}), \qquad (7.11)$$

где Nп – плановый объем работ по проведению опроса и составления отчета, сколько раз выполнялись в году (48 раз);

tc - трудоемкость выполнения работы до внедрения программного продукта 45,3 нормо.ч;

tn - трудеемкость выполнения работы после вднедрения програмнного продукта (12 нормо часов);

Тс - часовая тарифная ставка, соответсвующая разряду выполеняемых работ до внедрения программного продукта (20 тыс. руб/ч.);

Tn - часовая тарифная ставка, соответсвующая разряду выполеняемых работ после внедрения программного продукта (20 тыс. руб. /ч);

Кпр - коэффициент премий 1.2;

Нд - номратив дополнительной заработной платы 20%;

Нпо - ставка отчислений в ФСЗН и обязательное страхование 34+0,6%.

Экономия на заработной плате и начисления на заработную плату составит

Эз=1,2· (45,3·20 тыс -12·20 тыс) ·48· (1,2) ·1,346=61961,66 тыс руб.

Прирост чистой прибыли ($\Delta П_ч$) определяется по формуле:

$$\Delta П_ч = C_o - \frac{C_o \cdot H_п}{100}, \qquad (7.12)$$

где $Н_п$ – ставка налога на прибыль, (18%).

Таким образом, прирост чистой прибыли составит:

ΔПч=61961,66 тыс - 61961,66 тыс ·18/100=50808,56 тыс руб.

7.3.2. Расчет прироста амортизационных отчислений

Расчет амортизационных отчислений осуществляется по формуле:

$$A = H_A \cdot З/100, \qquad (7.13)$$

где З – затраты на разработку программы, руб.;

H_A – норма амортизации программного продукта, (20%);

A=184414,01 тыс · 0,2=36882,8 тыс руб.

7.4 Расчет показателей эффективности использования программного продукта

Для расчета показателей экономической эффективности использования программного продукта необходимо полученные суммы результата (прироста чистой прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2016 год) путем умножения результатов и затрат за каждый год на коэффициент привидения ($ALFA_t$), который рассчитывается по формуле:

$$ALFA_t = (1 + E_н)^{t_p - t}, \qquad (7.14)$$

где $E_н$ – норматив привидения разновременных затрат и результатов;

t_p – расчетный год, t_p = 1;

t – номер года, результаты и затраты которого приводятся к расчетному (2016-1, 2017-2, 2018-3, 2019-4).

$E_н$ = 25%

$$ALFA_1 = (1 + 0{,}25)^{1-1} = 1 - 2016;$$
$$ALFA_2 = (1 + 0{,}25)^{1-2} = 0{,}8 - 2017 \text{ год};$$
$$ALFA_3 = (1 + 0{,}25)^{1-3} = 0{,}64 - 2018 \text{ год};$$
$$ALFA_4 = (1 + 0{,}25)^{1-4} = 0{,}512 - 2019 \text{ год};$$

Результаты расчета показателей эффективности приведены в таблице 7.2.

Проект планируется внедрить в организации во второй половине 2016 года, поэтому в 2016 году организация может получить половину прибыли.

Таблица 7.2 − Расчет экономического эффекта от использования нового ПС

Наименование показателей	Един. измер.	Усл. обоз.	*По годам производства*			
			1-й	2-й	3-й	4-й
Результат						
1. Прирост чистой прибыли	тыс руб	$\Delta П_ч$	25404,28	50808,56	50808,56	50808,56
2.Прирост амортизационных отчислений	тыс руб	ΔA	36882,8	36882,8	36882,8	36882,8
3.Прирост результата	тыс руб	ΔP_t	62287,08	87691,36	87691,36	87691,36
4.Коэффициент дисконтирования	тыс руб	α_t	1	0,8	0,64	0,512
5.Результат с учетом фактора времени	тыс руб	$P_t \alpha_t$	62287,08	70153,09	56122,47	44897,98
Затраты (инвестиции)						
6.Инвестиции в разработку программного продукта	тыс руб	Иоб	184414,01			
7.Инвестиции с учетом фактора времени	тыс руб	$И_t \alpha_t$	184414,01			
8.Чистый дисконтированный доход по годам (п.4 - п.6)	тыс руб	$ЧДД_t$	307,92	70153,09	56122,47	44897,98
9.ЧДД нарастающим итогом	тыс руб	ЧДД	-122126,93	-51973,84	4148,63	49046,61

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта ($Р_И$) по формуле:

$$Р_и = \frac{П_{чср}}{З} \cdot 100\%, \quad (7.15)$$

где $П_{чср}$ – среднегодовая величина чистой прибыли за расчетный период, руб., которая определяется по формуле:

$$П_{чср} = \frac{\sum_{i=1}^{n} П_{чt}}{n}, \quad (7.16)$$

где $П_{чt}$ – чистая прибыль, полученная в году t, тыс руб.

Пчср = (25404,28+50808,56+50808,56+50808,56)/4= 44457,49 тыс руб.

Ри=44457,49/184414,01·100%=24%

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

1) чистый дисконтированный доход за четыре года работы программы составит 49046,61 тыс руб;

2) затраты на разработку программного продукта окупятся в третий год его использования;

3) рентабельность инвестиций составляет 24%.

Таким образом, применение программного продукта является эффективным и инвестиции в его разработку целесообразно осуществлять.

ЗАКЛЮЧЕНИЕ

Целью данного дипломного проекта является разработка программного средства управления городским освещением. Программное средство предназначено для взаимодействия с устройствами РУНО-3, производства ОАО «Белэлектромонтажналадка», посредством GSM сети и протокола Modbus TCP. Введение в эксплуатацию программного средства позволит увеличить производительность труда работников компании, предоставляя им удобную площадку для управления освещением города. Также разработанное программное средство способствует улучшению эффективности системы городского освещения.

В ходе выполнения дипломного проекта была изучена предметная область в сфере программных средств диспетчерского управления, проанализированы аналоги, существующие на рынке, и литературные источники. При анализе предметной области были рассмотрены различные способы управления и мониторинга технического процесса, применяемые при в системах «человек-машина», наиболее подходящие из которых были выбраны для разработки архитектуры программного средства. На этапе изучения предметной области были проанализированы различные технологии и средства, из которых далее были отобраны наиболее подходящие для реализации поставленной задачи.

На этапе моделирования предметной области были разработаны функциональная и информационная модели. Для построения функциональной модели были проанализированы все необходимые возможности, предоставляемые пользователям в системах подобного рода. Разработка информационной модели была выполнена на основе спецификаций входных и выходных данных, и функций, выполняемых программным средством. На основе информации полученной при изучении и моделировании предметной области была составлена функциональная спецификация требований, с помощью которой определяется перечень всех функций, выполняемых разработанным программным средством.

Архитектура программного средства была разработана с учетом всех функциональных требований и знаний, полученных при изучении предметной области диспетчерских программных средств. На этапе разработки архитектуры программного средства была определена структура проектов программного средства.

Во время технического проектирования программного средства управления городским освещением были разработаны и реализованы алгоритмы, применяемые при решении поставленных задач. Для наиболее важных алгоритмов были построены блок-схемы, описывающие принципы их работы в наглядном графическом виде.

На протяжении всех этапов разработки и реализации программного средства производилось его тестирование с применением различных техник и подходов. Был составлен перечень тестовых сценариев, по которым

определяется работоспособность программного средства. В результате контрольного тестирования все тесты были пройдены успешно.

Для установки программного средства был создан установочный msi-пакет. Для последующего сопровождения программного средства было создано руководство по использованию программного средства конечными пользователями разработанного программного средства.

При выполнении дипломного проекта в системе было также выполнено экономическое технико-обоснование целесообразности разработки программного средства управления городским освещением, в результате которого были определены сроки окупаемости и размер прибыли, которая будет получена после ввода программного средства в эксплуатацию.

Таким образом, задачи дипломного проекта были выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Троелсен, Э. Язык программирования C# 2008 и платформа .NET 3.5, 4-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 1344 с.: ил. – Парал. тит. англ.

[2] Мак-Дональд, М. WPF: Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов, 2-е издание: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 928 с.: ил. – Парал. тит. англ.

[3] Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.

[4] Компьютер и режимы энергосбережения [Электронный ресурс]. - Электронные данные. - Режим доступа: http://energohelp.net/articles/energy-tools/61799/

[5] Реле времени РВ720.Руководство по эксплуатации. – БЭМН,2005г. – 38 с.

[6] Официальный сайт компании «ТЕХНОКОНТ» [Электронный ресурс] / ред. Рудко Р.И.; Web-мастер Зиновьев А. Л – Электрон. Дан. – М.: Рос. Гос. Б-ка, 1999 . – Режим доступа: http://www.technocont.ru com, свободный. – Загл. С экрана. – Яз. Рус.

[7] Поспелов Г.Е. АСУ и оптимизация режимов энергосистем. Минск: Энергия. 1979, 467 с., 2 экз

[8] Гамма, Э. Приёмы объектно-ориентированного проектирования / Э. Гамма. – СПб: Питер, 2006. – 348 с.

[9] Algorithmic Performance Comparison Between C, C++, Java and C# Programming Languages [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.cherrystonesoftware.com/doc/AlgorithmicPerformance.pdf

[10] Schildt, H. C# 4.0 The Complete Reference / H. Schildt. – NY: McGraw Hill, 2010. – 976 p.

[11] Smith, J. Advanced MVVM / J. Smith – London: Lulu Enterprises, Inc, – 2010. – 52 p.

[12] Tableau [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.tableausoftware.com/public/.

[13] Хассан, Г. UML-проектирование систем реального времени параллельных и распределенных приложений / Г. Хассан. – М.: ДМК Пресс, 2011. – 704с.

[14] Технико-экономическое обоснование дипломных проектов: Методическое пособие для студентов всех специальностей БГУИР. В 4-ч. Ч. 4.

ПРИЛОЖЕНИЕ А
(обязательное)
Текст программы

```
using System;
using System.Diagnostics.CodeAnalysis;
using System.Windows;
using ULA.CompositionRoot;
using ULA.Interceptors.Application;
using ULA.Shell.Bootstrapping;

namespace ULA.Shell
{
   /// <summary>
   ///     Interaction logic for App.xaml
   /// </summary>
   public partial class App
   {
      #region [Private fields]

      private IApplicationContentContainer _applicationContent;

      #endregion [Private fields]

      #region [Override members]

      /// <summary>
      ///     Raises the <see cref="E:System.Windows.Application.Startup" /> event.
      /// </summary>
      /// <param name="e">
      ///     A <see cref="T:System.Windows.StartupEventArgs" /> that contains the event data.
      /// </param>
      [SuppressMessage("Microsoft.Performance", "CA1804:RemoveUnusedLocals", MessageId = "exception")]
      protected override void OnStartup(StartupEventArgs e)
      {
         var stateManager = new DefaultApplicationStateManger();
         stateManager.GotToNewState(ApplicationState.BOOTSTRAPPING);
         try
         {
            base.OnStartup(e);
            var bootstrapper = new ApplicationBootstrapper(stateManager);
            bootstrapper.Run();
            this._applicationContent = bootstrapper.ContentContainer;
            stateManager.GotToNewState(ApplicationState.RUNTIME);
            Current.MainWindow = (Window) bootstrapper.CurrentShell;
            Current.MainWindow.Show();
         }
#pragma warning disable 168
         catch (Exception exception)
#pragma warning restore 168
         {
            stateManager.GotToNewState(ApplicationState.FATAL_ERROR);
            throw;
         }
      }


      /// <summary>
      ///     Raises the <see cref="E:System.Windows.Application.Exit" /> event.
      /// </summary>
```

```
        /// <param name="e">
        ///     An <see cref="T:System.Windows.ExitEventArgs" /> that contains the event data.
        /// </param>
        protected override void OnExit(ExitEventArgs e)
        {
            this._applicationContent.Dispose();
            this._applicationContent = null;
            base.OnExit(e);
        }

        #endregion [Override members]
    }
}


using System;
using System.Globalization;
using System.Linq;
using ULA.Interceptors.Application;
using ULA.Shell.Properties;

namespace ULA.Shell.ApplicationLevelServices
{
    /// <summary>
    ///     Represents an application settings facade service functionality
    ///     Представляет фасад для сервиса настроек
    /// </summary>
    public class ApplicationSettingsService : IApplicationSettingsService
    {
        private static int _applicationNumber;

        #region [IApplicationSettingsService members]

        /// <summary>
        ///     Gets a full path of the devices storage file
        ///     Получает путь к хранилищу устройств
        /// </summary>
        public string DevicesStoragePath
        {
            get { return String.Format(CultureInfo.CurrentCulture, Settings.Default.DeviceStoragePath,
this.ApplicationNumber); }
        }

        /// <summary>
        ///     Описывает номер экземпляра программы
        /// </summary>
        public int ApplicationNumber
        {
            get { return ApplicationSettingsService._applicationNumber; }
            set { ApplicationSettingsService._applicationNumber = value; }
        }

        /// <summary>
        ///     Возвращает путь к файлу с данными о обновлении времени
        /// </summary>
        public string UpdateTimeFilePath
        {
            get
            {
                return String.Format(CultureInfo.CurrentCulture, Settings.Default.UpdateTimePath,
this.ApplicationNumber);
            }
```

```
        }

        /// <summary>
        ///     Возвращает путь к файлу журнала
        /// </summary>
        public string JournalFilePath
        {
            get
            {
                return String.Format(CultureInfo.CurrentCulture, Settings.Default.JournalPath,
DateTime.Now.Month.ToString("D2", CultureInfo.CurrentCulture),
                    DateTime.Now.Year.ToString("D4", CultureInfo.CurrentCulture), this.ApplicationNumber);
            }
        }

        #endregion [IApplicationSettingsService members]
    }
}



using System.Windows;
using Microsoft.Practices.Unity;
using ULA.CompositionRoot.Bootstrapping;
using ULA.Interceptors.Application;
using ULA.Presentation.Infrastructure.Services;
using ULA.Shell.ApplicationLevelServices;

namespace ULA.Shell.Bootstrapping
{
    /// <summary>
    ///     Represents the application bootstrapper
    ///     Представляет бутстрапер приложения
    /// </summary>
    public class ApplicationBootstrapper : Bootstrapper
    {
        #region [Ctor's]

        /// <summary>
        ///     Creates an instance of <see cref="ApplicationBootstrapper" />
        /// </summary>
        /// <param name="currentApplicationStateManager">
        ///     An instance of <see cref="IApplicationStateManager" /> that represents application management
functionality
        /// </param>
        public ApplicationBootstrapper(IApplicationStateManager currentApplicationStateManager)
            : base(currentApplicationStateManager)
        {
        }

        #endregion [Ctor's]

        #region [Properties]

        /// <summary>
        ///     Gets an instanhce of <see cref="DependencyObject" /> that represents current shell
        /// </summary>
        public DependencyObject CurrentShell
        {
            get { return Shell; }
        }
```

```
        #endregion [Properties]

        #region [Override members]

        /// <summary>
        ///     Creates the shell or main window of the application.
        /// </summary>
        /// <returns>
        ///     The shell of the application.
        /// </returns>
        /// <remarks>
        ///     If the returned instance is a <see cref="T:System.Windows.DependencyObject" />, the
        ///     <see cref="T:Microsoft.Practices.Prism.Bootstrapper" /> will attach the default
        ///     <seealso cref="T:Microsoft.Practices.Prism.Regions.IRegionManager" /> of
        ///     the application in its <see
cref="F:Microsoft.Practices.Prism.Regions.RegionManager.RegionManagerProperty" />
        ///     attached property
        ///     in order to be able to add regions by using the
        ///     <seealso cref="F:Microsoft.Practices.Prism.Regions.RegionManager.RegionNameProperty" />
        ///     attached property from XAML.
        /// </remarks>
        protected override DependencyObject CreateShell()
        {
            return this.Container.Resolve<Shell>();
        }

        /// <summary>
        ///     Configures the <see cref="T:Microsoft.Practices.Unity.IUnityContainer" />. May be overwritten in a
derived class to
        ///     add specific
        ///     type mappings required by the application.
        /// </summary>
        protected override void ConfigureContainer()
        {
            this.Container.RegisterType<IResourcesService, DefaultResourceService>(
                new ContainerControlledLifetimeManager());

            base.ConfigureContainer();

            this.Container.RegisterType<IApplicationSettingsService, ApplicationSettingsService>(
                new ContainerControlledLifetimeManager());
        }

        #endregion [Override members]
    }
}


namespace ULA.Presentation.Infrastructure.DTOs
{
    /// <summary>
    ///     Represetns default device data transfer object model
    ///     Представляет модель данных для устройства
    /// </summary>
    public class DeviceModel
    {
        #region [IDeviceModel members]

        /// <summary>
        ///     Gets or sets the name of the device
        ///     Имя устройства
        /// </summary>
```

```
public string Name { get; set; }

/// <summary>
///     Gets or sets the description of the device
///     Описание устройства
/// </summary>
public string Description { get; set; }

/// <summary>
///     Gets or sets the tcp-ip address of the remote device
///     TCP/IP адрес устройства
/// </summary>
public string TcpAddress { get; set; }

/// <summary>
///     Gets or sets the tcp-ip port of the remote device
///     TCP порт устройства
/// </summary>
public int TcpPort { get; set; }

/// <summary>
///     Gets or sets the type name of the device
///     Имя типа устройства
/// </summary>
public string DeviceFactoryTypeName { get; set; }

/// <summary>
///     Gets or sets the type name of the driver that current device is going to consume
///     Имя типа драйвера
/// </summary>
public string DriverFactoryTypeName { get; set; }

/// <summary>
///     описание первого пускателя
/// </summary>
public string Starter1Description { get; set; }

/// <summary>
///     описание второго пускателя
/// </summary>
public string Starter2Description { get; set; }

/// <summary>
///     описание третьего пускателя
/// </summary>
public string Starter3Description { get; set; }

///// <summary>
/////     Описание первого фидера
///// </summary>
//public string Fider1Description { get; set; }

/// <summary>
///     Описание второго фидера
/// </summary>
public string FiderDescription1 { get; set; }

/// <summary>
///     Описание третьего фидера
/// </summary>
public string FiderDescription2 { get; set; }
```

```
        /// <summary>
        ///     Описание четвертого фидера
        /// </summary>
        public string FiderDescription3 { get; set; }

        #endregion [IDeviceModel members]
    }
}


using System;

namespace ULA.Presentation.Infrastructure.DTOs
{
    /// <summary>
    ///   Log information
    ///   Описывает информацию для журналирования
    /// </summary>
    public interface ILogInformation
    {
        /// <summary>
        ///     Gets or sets the description action
        ///     Описание действия
        /// </summary>
        string ActionDescription { get; set; }

        /// <summary>
        ///     Gets or sets thr action date
        ///     Дата действия
        /// </summary>
        DateTime ActionDate { get; set; }
    }
}


using System;
using ULA.Presentation.Infrastructure.Services.Interactions;
using ULA.Presentation.Infrastructure.ViewModels.Interactions;

namespace ULA.Presentation.Infrastructure.Services
{
    /// <summary>
    ///     Exposes an interaction service functionality
    /// </summary>
    public interface IInteractionService
    {
        /// <summary>
        ///     Sends an interaction request to a user
        /// </summary>
        /// <typeparam name="TViewModel"><see cref="Type" /> that represents the view model of the interaction
request</typeparam>
        /// <param name="provider">
        ///     An instance of <see cref="IInteractionProvider{TViewModel}" /> that represents current
        ///     interaction request provider
        /// </param>
        /// <param name="interactionInterceptor">An instance of <see cref="InteractionInterceptor{TViewModel}" />
to use</param>
        /// <returns>An instance of <see cref="IInteractionToken" /> that represents interaction request
token</returns>
        IInteractionToken Interact<TViewModel>(IInteractionProvider<TViewModel> provider,
            InteractionInterceptor<TViewModel> interactionInterceptor)
            where TViewModel : IInteractionViewModel;
```

```
        /// <summary>
        ///     Sends an interaction request to a user
        /// </summary>
        /// <typeparam name="TViewModel"><see cref="Type" /> that represents the view model of the interaction
request</typeparam>
        /// <param name="provider">
        ///     An instance of <see cref="IInteractionProvider{TViewModel}" /> that represents current
        ///     interaction request provider
        /// </param>
        /// <param name="onInitialized">An instance of <see cref="Action{T}" /> that represents an initialization
callback</param>
        /// <param name="onUninitialized">An instance of <see cref="Action{T}" /> that represents an uninitialization
callback</param>
        /// <returns>An instance of <see cref="IInteractionToken" /> that represents interaction request
token</returns>
        IInteractionToken Interact<TViewModel>(IInteractionProvider<TViewModel> provider,
            Action<TViewModel> onInitialized = null, Action<TViewModel> onUninitialized = null)
            where TViewModel : IInteractionViewModel;
    }
}


using System;

namespace ULA.Presentation.Infrastructure.Services
{
    /// <summary>
    ///     Exposes global resources service
    /// </summary>
    public interface IResourcesService
    {
        /// <summary>
        ///     Injects a resource into a system
        /// </summary>
        /// <param name="packUri">The uri of the resurce to inject</param>
        void AddResourceAsGlobal(Uri packUri);
    }
}


using System.ComponentModel;
using YP.Toolkit.System.Tools.Patterns;

namespace ULA.Presentation.Infrastructure.ViewModels
{
    /// <summary>
    ///     Represents a basic view model functionality
    /// </summary>
    public class ViewModelBase : Disposable, INotifyPropertyChanged, INotifyPropertyChanging
    {
        #region [INotifyPropertyChanged members]

        /// <summary>
        ///     Occurs when a property value changes.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        #endregion [INotifyPropertyChanged members]

        #region [INotifyPropertyChanging members]
```

```
        /// <summary>
        ///     Occurs when a property value is changing.
        /// </summary>
        public event PropertyChangingEventHandler PropertyChanging;

        #endregion [INotifyPropertyChanging members]

        #region [Virtual mmebers]

        /// <summary>
        ///     Notifies property changed
        /// </summary>
        /// <param name="propertyName">The name of the property that has been changed</param>
        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
        }

        /// <summary>
        ///     Notifies property changing
        /// </summary>
        /// <param name="propertyName">The name of the property that is being changed</param>
        protected virtual void OnPropertyChanging(string propertyName)
        {
            PropertyChangingEventHandler handler = PropertyChanging;
            if (handler != null) handler(this, new PropertyChangingEventArgs(propertyName));
        }

        #endregion [Virtual mmebers]
    }
}


namespace ULA.Presentation.Infrastructure.Views
{
    /// <summary>
    ///     Exposes an interaction view functionality
    /// </summary>
    public interface IInteractionView
    {
        /// <summary>
        ///     Gets or sets an instance of <see cref="object" /> that represents current view's data context
        /// </summary>
        object DataContext { get; set; }

        /// <summary>
        ///     Gets or sets an instance of <see cref="string" /> that represents current interaction view's title
        /// </summary>
        string Title { get; set; }
    }
}



using System.Windows;
using System.Windows.Controls;

namespace ULA.Presentation.Infrastructure.Views
{
    /// <summary>
    ///     Represents base entity for the all interaction request views
```

```
    /// </summary>
    public class InteractionViewBase : UserControl, IInteractionView
    {
        #region [Dependency properties]

        /// <summary>
        ///     Gets a TitleProperty metadata
        /// </summary>
        public static readonly DependencyProperty TitleProperty =
            DependencyProperty.Register("Title", typeof (string), typeof (InteractionViewBase),
                new PropertyMetadata(string.Empty));

        #endregion [Dependency properties]

        #region [IInteractionView members]

        /// <summary>
        ///     Gets or sets the title of current interaction request view
        /// </summary>
        public string Title
        {
            get { return (string) GetValue(TitleProperty); }
            set { SetValue(TitleProperty, value); }
        }

        #endregion [IInteractionView members]
    }
}


using System.Windows;
using System.Windows.Controls;

namespace ULA.Presentation.Infrastructure.Views
{
    /// <summary>
    ///     Represents the main content view control
    ///     Представляет общие данные вьюшки
    /// </summary>
    public class MainContentView : UserControl
    {
        #region [Dependency properties]

        /// <summary>
        ///     Gets a TitleProperty metadata
        /// </summary>
        public static readonly DependencyProperty TitleProperty =
            DependencyProperty.Register("Title", typeof (string), typeof (MainContentView),
                new FrameworkPropertyMetadata(string.Empty, OnTitleChanged));

        /// <summary>
        ///     Gets or sets the title of current main content view
        /// </summary>
        public string Title
        {
            get { return (string) GetValue(TitleProperty); }
            set { SetValue(TitleProperty, value); }
        }

        private static void OnTitleChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)
        {
            Application.Current.MainWindow.Title = (string) e.NewValue;
```

```
        }

        #endregion [Dependency properties]
    }
}


namespace ULA.Presentation.Infrastructure
{
    /// <summary>
    ///     Represents a storage entity for holding global application names
    /// </summary>
    public static class ApplicationGlobalNames
    {
        /// <summary>
        ///     Показывает количество колонок в сетке листа устройств
        /// </summary>
        public const int WidgetListColumnCount = 13;

        /// <summary>
        ///     Показывает количество строк в сетке листа устройств
        /// </summary>
        public const int WidgetListRowCount = 8;

        /// <summary>
        ///     Gets the name of the system create new device view
        /// </summary>
        public const string CREATE_NEW_DEVICE_VIEW_NAME = "CreateNewDeviceView";

        /// <summary>
        ///     Gets the name of the system create new device request provider
        /// </summary>
        public const string CREATE_NEW_DEVICE_PROVIDER_NAME = "CreateNewDeviceProvider";

        /// <summary>
        ///     Gets the name of the system busy interaction request provider
        /// </summary>
        public const string BUSY_INTERACTION_PROVIDER_NAME = "BusyInteractionProvider";

        /// <summary>
        ///     Gets the name of the system error messagebox interaction request provider
        /// </summary>
        public const string ERROR_MESSAGEBOX_INTERACTION_PROVIDER_NAME =
"ErrorMessageBoxInteractionProvider";

        /// <summary>
        ///     Gets the name of the system information messagebox interaction request provider
        /// </summary>
        public const string INFORMATION_MESSAGEBOX_INTERACTION_PROVIDER_NAME =
            "InformationMessageBoxInteractionProvider";

        /// <summary>
        ///     Gets the name of the system question messagebox interaction request provider
        /// </summary>
        public const string QUESTION_MESSAGEBOX_INTERACTION_PROVIDER_NAME =
            "QuestionMessageBoxInteractionProvider";

        /// <summary>
        ///     Gets the name of the system warning messagebox interaction request provider
        /// </summary>
        public const string WARNING_MESSAGEBOX_INTERACTION_PROVIDER_NAME =
            "WarningMessageBoxInteractionProvider";
```

```
        /// <summary>
        ///     Gets the name of the citywide commands interaction request provider
        /// </summary>
        public const string CITYWIDE_COMMANDS_INTERACTION_PROVIDER_NAME =
           "CitywideCommandsInteractionProvider";

        /// <summary>
        ///     Gets the name of the about interaction request provider
        /// </summary>
        public const string ABOUT_INTERACTION_PROVIDER_NAME =
           "AboutInteractionProvider";

        /// <summary>
        ///     Gets the name of the update timeout request provider
        /// </summary>
        public const string UPDATE_TIMEOUT_PROVIDER_NAME =
           "UpdateTimeoutInteractionProvider";

        /// <summary>
        ///     Gets the name of the log interaction request provider
        /// </summary>
        public const string LOG_INTERACTION_PROVIDER_NAME = "LogInteractionProvider";

        /// <summary>
        ///     Gets the name of the synchronization time interaction request provider
        /// </summary>
        public const string SYNCHRONIZATION_TIME_PROVIDER_NAME =
"SynchronizationTimeInteractionProvider";

        /// <summary>
        ///     Gets the name of password interaction request provider
        /// </summary>
        public const string PASSWORD_INTERACTION_PROVIDER_NAME = "PasswordInteractionProvider";

        /// <summary>
        ///     Gets the name of the failure application mode view model
        /// </summary>
        public const string FAILURE_MODE_VIEW_MODEL_NAME = "FailureModeViewModel";

        /// <summary>
        ///     Gets the name of the application runtime mode view model
        /// </summary>
        public const string RUNTIME_MODE_VIEW_MODEL_NAME = "RuntimeModeViewModel";

        /// <summary>
        ///     Gets the name of the root view
        /// </summary>
        public const string ROOT_VIEW_NAME = "RootView";

        /// <summary>
        ///     Gets the name of the application configuration mode view model
        /// </summary>
        public const string CONFIGURATION_MODE_VIEW_MODEL_NAME =
"ConfigurationModeViewModel";

        /// <summary>
        ///     Gets the name of the application busy dialog view
        /// </summary>
        public const string BUSY_INTERACTION_VIEW_NAME = "BusyIneractionView";

        /// <summary>
```

```
        ///     Gets the name of the error messagebox dialog view
        /// </summary>
        public const string ERROR_MESSAGEBOX_VIEW_NAME = "ErrorMessageBoxInteractionView";

        /// <summary>
        ///     Gets the name of the information messagebox dialog view
        /// </summary>
        public const string INFORMATION_MESSAGEBOX_VIEW_NAME =
"InformationMessageBoxInteractionView";

        /// <summary>
        ///     Gets the name of the question messagebox dialog view
        /// </summary>
        public const string QUESTION_MESSAGEBOX_VIEW_NAME = "QuestionMessageBoxInteractionView";

        /// <summary>
        ///     Gets the name of the warning messagebox dialog view
        /// </summary>
        public const string WARNING_MESSAGEBOX_VIEW_NAME = "WarningMessageBoxInteractionView";

        /// <summary>
        ///     Gets the name of the citywide commands dialog view
        /// </summary>
        public const string CITYWIDE_COMMANDS_VIEW_NAME = "CitywideCommandsInteractionView";

        /// <summary>
        ///     Gets the name of the about program dialog view
        /// </summary>
        public const string ABOUT_VIEW_NAME = "AboutInteractionView";

        /// <summary>
        ///     Gets the name of the update timeout dialog view
        /// </summary>
        public const string UPDATE_TIMEOUT_VIEW_NAME = "UpdateTimeoutInteractionView";

        /// <summary>
        ///     Gets the name of the log dialog view
        /// </summary>
        public const string LOG_VIEW_NAME = "LogInteractionView";

        /// <summary>
        ///     Gets the name of the synchronization time dialog view
        /// </summary>
        public const string SYNCHRONIZATION_TIME_VIEW_NAME = "SynchronizationTimeInteractionView";

        /// <summary>
        ///     Gets the name of the list widget view in runtime mode
        /// </summary>
        public const string LIST_WIDGET_RUNTIME_VIEW_NAME = "ListWidgetModeRuntimeView";

        /// <summary>
        ///     Gets the name of the list widget view in configuration mode
        /// </summary>
        public const string LIST_WIDGET_CONFIGURATION_VIEW_NAME =
"ListWidgetModeConfigurationView";

        /// <summary>
        ///     Gets the name of the scheme view in runtime mode
        /// </summary>
        public const string SCHEME_RUNTIME_VIEW_NAME = "SchemeModeRuntimeView";

        /// <summary>
```

```
        ///     Gets the name of the scheme view in configuration mode
        /// </summary>
        public const string SCHEME_CONFIGURATION_VIEW_NAME = "SchemeModeonfigurationView";

        /// <summary>
        ///     Gets the name of the confugurate runo view name
        /// </summary>
        public const string RUNO_CONFIGURATION_VIEW_NAME = "RunoConfigurationViewName";

        /// <summary>
        ///     Gets the name of the lighting shedule view name
        /// </summary>
        public const string LIGHTING_SHEDULER_VIEW_NAME = "LightingSheduleView";

        /// <summary>
        ///     Gets the name of the password interaction view name
        /// </summary>
        public const string PASSWORD_VIEW_NAME = "PasswordInteractionView";

        /// <summary>
        ///     Gets the name of the region in the Configuration mode
        /// </summary>
        public const string CONFIGURATION_REGION_NAME = "ModeConfigurationRegion";

        /// <summary>
        ///     Gets the name of the region in the Runtime mode
        /// </summary>
        public const string RUNTIME_REGION_NAME = "ModeRuntimeRegion";

        /// <summary>
        ///     Gets the name of the modify starter view
        /// </summary>
        public const string MODIFY_STARTER_VIEW_NAME = "ModifuStarterViewName";

        /// <summary>
        ///     Get the name of modify starter provider
        /// </summary>
        public const string MODIFY_STARTER_PROVIDER_NAME = "ModifyStarterProvider";

        /// <summary>
        ///     Get the name modify resistor provider
        /// </summary>
        public const string MODIFY_RESISTOR_PROVIDER_NAME = "ModifyResistorProvider";

        /// <summary>
        ///     Get the name modify resistor view
        /// </summary>
        public const string MODIFY_RESISTOR_VIEW_NAME = "ModifyResistorView";
    }
}


using System.Reflection;
using System.Windows;
using System.Windows.Data;
using System.Windows.Input;

namespace ULA.Presentation.Behaviors
{
    /// <summary>
    ///     Represents a binding helper that provides some basic binding help methods
```

```
    ///     Представляет Помощник привязки, который обеспечивает некоторые базовые методы помогающие при привязке
    /// </summary>
    public static class BindingHelper
    {
        #region [Public members]

        /// <summary>
        ///     Updates a binding for an object
        ///     Обновляет привязку к объекту
        /// </summary>
        /// <param name="item">An object to update binding for</param>
        /// <param name="property">The property to update</param>
        /// <param name="dataContext">A binding source to use</param>
        public static void UpdateBinding(DependencyObject item, DependencyProperty property, object dataContext)
        {
            BindingExpression bindingExpression = BindingOperations.GetBindingExpression(item, property);
            if (bindingExpression == null) return;
            Binding parentBinding = bindingExpression.ParentBinding;

            string propertyName = parentBinding.Path.Path;
            if (dataContext == null) return;
            PropertyInfo dataContextProperty = dataContext.GetType().GetProperty(propertyName);
            if (dataContextProperty == null) return;
            var commandValue = dataContextProperty.GetValue(dataContext, null) as ICommand;
            if (commandValue == null) return;
            item.SetValue(property, commandValue);
        }

        #endregion [Public members]
    }
}


using System.Collections.Generic;
using System.Diagnostics;
using System.Windows;
using System.Windows.Input;
using System.Windows.Interactivity;
using System.Windows.Media;

namespace ULA.Presentation.Behaviors
{
    /// <summary>
    ///     Represents the behavior for registering application shortcuts globaly
    ///     Представляет поведение для глобальной регистрации горячих клавиш в приложении
    /// </summary>
    public class RegisterAppShortcutKeysBehavior : Behavior<FrameworkElement>
    {
        #region [Static Fields]

        private static readonly Dictionary<Window, List<FrameworkElement>> _associatedWindows =
            new Dictionary<Window, List<FrameworkElement>>();

        private static readonly object _associatedWindowsLock = new object();
        private static readonly List<InputBinding> _appInputBindings = new List<InputBinding>();
        private static InputBinding[] _readOnlyAppInputBindings = new InputBinding[0];
        private static readonly object _appInputBindingsLock = new object();
        private static readonly HashSet<Key> _shortcutKeys = new HashSet<Key>();

        #endregion [Static Fields]
```

```
#region [Fields]

private FrameworkElement _frameworkElement;

#endregion [Fields]

#region [Dependency properties]

/// <summary>
///     Gets a InputBindingsProperty metadata
///
/// </summary>
public static readonly DependencyProperty InputBindingsProperty =
   DependencyProperty.Register("InputBindings", typeof (InputBindingCollection),
      typeof (RegisterAppShortcutKeysBehavior),
      new FrameworkPropertyMetadata(null));


/// <summary>
///     Gets or sets the input bindings.
/// </summary>
/// <value>The input bindings.</value>
public InputBindingCollection InputBindings
{
   get { return (InputBindingCollection) GetValue(InputBindingsProperty); }
   set { SetValue(InputBindingsProperty, value); }
}

#endregion [Dependency properties]

#region [Ctor's]

/// <summary>
///     Initializes a type <see cref="RegisterAppShortcutKeysBehavior" />
/// </summary>
static RegisterAppShortcutKeysBehavior()
{
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F1);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F2);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F3);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F4);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F5);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F6);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F7);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F8);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F9);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F10);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F11);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F12);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F13);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F14);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F15);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F16);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F17);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F18);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F19);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F20);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F21);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F22);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F23);
   RegisterAppShortcutKeysBehavior._shortcutKeys.Add(Key.F24);
}
```

```
        /// <summary>
        ///     Initializes a new instance of the <see cref="RegisterAppShortcutKeysBehavior" /> class.
        /// </summary>
        public RegisterAppShortcutKeysBehavior()
        {
            InputBindings = new InputBindingCollection();
        }

        #endregion [Ctor's]

        #region [Override members]

        /// <summary>
        ///     Called after the behavior is attached to an AssociatedObject.
        ///     Вызывается после того как поведение привяжется к AssociatedObject
        /// </summary>
        /// <remarks>Override this to hook up functionality to the AssociatedObject.</remarks>
        protected override void OnAttached()
        {
            base.OnAttached();

            this._frameworkElement = AssociatedObject;
            this._frameworkElement.DataContextChanged += OnDataContextChanged;
            this._frameworkElement.Loaded += OnElementLoaded;
        }

        /// <summary>
        ///     Called when the behavior is being detached from its AssociatedObject, but before it has actually occurred.
        ///     Вызывается когда поведение отвязывается от AssociatedObject, но до полной потери связи м/у
поведение и AssociatedObject
        /// </summary>
        /// <remarks>Override this to unhook functionality from the AssociatedObject.</remarks>
        protected override void OnDetaching()
        {
            if (this._frameworkElement != null)
            {
                UnregisterAssociatedWindow(this._frameworkElement);
                this._frameworkElement.DataContextChanged -= OnDataContextChanged;
                this._frameworkElement.Loaded -= OnElementLoaded;
            }
            base.OnDetaching();
        }

        #endregion [Override members]

        #region [Help members]

        /// <summary>
        ///     Регистрируе привязку
        /// </summary>
        /// <param name="inputBindings"></param>
        /// <param name="commandTarget"></param>
        private static void RegisterBindings(InputBindingCollection inputBindings, FrameworkElement
commandTarget)
        {
            Debug.Assert(inputBindings != null);
            Debug.Assert(commandTarget != null);

            lock (RegisterAppShortcutKeysBehavior._appInputBindingsLock)
            {
                foreach (InputBinding inputBinding in inputBindings)
```

```
            {
                if (inputBinding.Command == null)
                {
                    BindingHelper.UpdateBinding(inputBinding, InputBinding.CommandProperty,
                        commandTarget.DataContext);
                }
                // I only add input bindings that are KeyGestures and have a command.
                if ((!(inputBinding.Gesture is KeyGesture)) || (inputBinding.Command == null)) continue;
                inputBinding.CommandTarget = commandTarget;
                RegisterAppShortcutKeysBehavior._appInputBindings.Add(inputBinding);
            }

            RegisterAppShortcutKeysBehavior._readOnlyAppInputBindings =
RegisterAppShortcutKeysBehavior._appInputBindings.ToArray();
        }
    }

    /// <summary>
    ///     Возвращает окно, которому принадлежит элемент
    /// </summary>
    /// <param name="child"></param>
    /// <returns></returns>
    private static Window GetParentWindow(DependencyObject child)
    {
        DependencyObject parentObject = VisualTreeHelper.GetParent(child);

        if (parentObject == null)
            return null;

        var parent = parentObject as Window;
        return parent ?? GetParentWindow(parentObject);
    }

    /// <summary>
    ///     Привязывает команду к окну
    /// </summary>
    /// <param name="commandTarget"></param>
    private static void RegisterAssociatedWindow(FrameworkElement commandTarget)
    {
        Window window = GetParentWindow(commandTarget);
        if (window == null) return;
        lock (RegisterAppShortcutKeysBehavior._associatedWindowsLock)
        {
            // I reference count the windows to prevent double subscribing to PreviewKeyDown
            List<FrameworkElement> frameworkElements;
            if (!RegisterAppShortcutKeysBehavior._associatedWindows.TryGetValue(window, out
frameworkElements))
            {
                frameworkElements = new List<FrameworkElement>();
                RegisterAppShortcutKeysBehavior._associatedWindows[window] = frameworkElements;
                window.PreviewKeyDown += Window_PreviewKeyDown;
            }

            if (!frameworkElements.Contains(commandTarget))
            {
                frameworkElements.Add(commandTarget);
            }
        }
    }

    /// <summary>
    ///     Отвязывает команду от окна
```

```
        /// </summary>
        /// <param name="commandTarget"></param>
        private static void UnregisterAssociatedWindow(FrameworkElement commandTarget)
        {
            Debug.Assert(commandTarget != null);

            Window window = Window.GetWindow(commandTarget);
            if (window == null) return;
            lock (RegisterAppShortcutKeysBehavior._associatedWindowsLock)
            {
                List<FrameworkElement> frameworkElements;
                if (!RegisterAppShortcutKeysBehavior._associatedWindows.TryGetValue(window, out
frameworkElements)) return;
                frameworkElements.Remove(commandTarget);
                if (frameworkElements.Count != 0) return;
                RegisterAppShortcutKeysBehavior._associatedWindows.Remove(window);
                window.PreviewKeyDown -= Window_PreviewKeyDown;
            }
        }

        /// <summary>
        ///     обработчик события перед нажатием клавиши
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private static void Window_PreviewKeyDown(object sender, KeyEventArgs e)
        {
            bool isShortcutKeystroke = (Keyboard.Modifiers != ModifierKeys.None) ||
RegisterAppShortcutKeysBehavior._shortcutKeys.Contains(e.Key);

            if (!isShortcutKeystroke) return;
            var window = sender as Window;
            if (window == null) return;
            // I hold the lock for the shortest time possible
            InputBinding[] inputBindings;
            lock (RegisterAppShortcutKeysBehavior._appInputBindingsLock)
            {
                inputBindings = RegisterAppShortcutKeysBehavior._readOnlyAppInputBindings;
            }

            foreach (InputBinding inputBinding in inputBindings)
            {
                if (!inputBinding.Gesture.Matches(inputBinding.CommandTarget, e)) continue;
                if (InvokeCommandIfTargetVisible(inputBinding))
                {
                    e.Handled = true;
                }
            }
        }

        /// <summary>
        ///     Проверка на видимость элемента и если видимый, то вызов
        /// </summary>
        /// <param name="inputBinding"></param>
        /// <returns></returns>
        private static bool InvokeCommandIfTargetVisible(InputBinding inputBinding)
        {
            Debug.Assert(inputBinding != null);

            var frameworkElement = inputBinding.CommandTarget as FrameworkElement;

            // I only execute when the command target is visible.
```

```
            if ((frameworkElement != null) && (frameworkElement.IsVisible))
            {
                return InvokeCommand(inputBinding);
            }

            return false;
        }

        /// <summary>
        ///     Вызов команды
        /// </summary>
        /// <param name="inputBinding"></param>
        /// <returns></returns>
        private static bool InvokeCommand(InputBinding inputBinding)
        {
            Debug.Assert(inputBinding != null);

            // Routed commands are static, so we need to provide the command target
            // otherwise the default is to use the focused element.
            var routedCommand = inputBinding.Command as RoutedCommand;
            if (routedCommand != null)
            {
                if (!routedCommand.CanExecute(inputBinding.CommandParameter, inputBinding.CommandTarget))
return false;
                routedCommand.Execute(inputBinding.CommandParameter, inputBinding.CommandTarget);
                return true;
            }
            // Other commands are instance commands and can be called directly.
            if (!inputBinding.Command.CanExecute(inputBinding.CommandParameter)) return false;
            inputBinding.Command.Execute(inputBinding.CommandParameter);
            return true;
        }

        /// <summary>
        ///     Обработчик события для загрузки элемента
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void OnElementLoaded(object sender, RoutedEventArgs e)
        {
            RegisterAssociatedWindow(this._frameworkElement);
        }

        /// <summary>
        ///     Обработчик события для изменения контекста
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void OnDataContextChanged(object sender, DependencyPropertyChangedEventArgs e)
        {
            if (this._frameworkElement == null) return;
            if (InputBindings != null)
            {
                RegisterBindings(InputBindings, this._frameworkElement);
            }
        }

        #endregion [Help members]
    }
}
```

```
using ULA.AddinsHost.Business.Device;
using ULA.Business.Infrastructure.DTOs;
using ULA.Presentation.Infrastructure.DTOs;

namespace ULA.Presentation.DTOs
{
   /// <summary>
   ///     Represents extension methods registry for <see cref="DeviceModel" />
   ///     Расширения для клаасов представляющих устройство
   /// </summary>
   public static class DeviceModelExtensions
   {
      /// <summary>
      ///     Maps the <see cref="IConfigLogicalDevice" /> to <see cref="DeviceModel" />
      ///     Конвертирует <see cref="IConfigLogicalDevice" /> в <see cref="DeviceModel" />
      /// </summary>
      /// <param name="device">
      ///     An instance of <see cref="IConfigLogicalDevice" /> to map to <see cref="DeviceModel" />
      /// </param>
      /// <returns>
      ///     Resulted instance of <see cref="DeviceModel" />
      /// </returns>
      public static DeviceModel ToDeviceModel(this IConfigLogicalDevice device)
      {
         return new DeviceModel
            {
               Name = device.DeviceName,
               Description = device.DeviceDescription,
               //Fider1Description =
device.CreateMomento().State.DataContainer.GetValue<string>("Fider1Description"),
               FiderDescription1 =
device.CreateMomento().State.DataContainer.GetValue<string>("FiderDescription1"),
               FiderDescription2 =
device.CreateMomento().State.DataContainer.GetValue<string>("FiderDescription2"),
               FiderDescription3 =
device.CreateMomento().State.DataContainer.GetValue<string>("FiderDescription3"),
               Starter1Description =
device.CreateMomento().State.DataContainer.GetValue<string>("Starter1Description"),
               Starter2Description =
device.CreateMomento().State.DataContainer.GetValue<string>("Starter2Description"),
               Starter3Description =
device.CreateMomento().State.DataContainer.GetValue<string>("Starter3Description")
            };
      }

      /// <summary>
      ///     Maps the <see cref="DeviceModel" /> to the <see cref="LogicalDeviceInformation" />
      ///     Конвертирует <see cref="DeviceModel" /> в <see cref="LogicalDeviceInformation" />
      /// </summary>
      /// <param name="model">
      ///     An instance of <see cref="DeviceModel" /> to map to <see cref="LogicalDeviceInformation" />
      /// </param>
      /// <returns>
      ///     Resulted instance of <see cref="LogicalDeviceInformation" />
      /// </returns>
      public static LogicalDeviceInformation ToLogicalDeviceInfo(this DeviceModel model)
      {
         return new LogicalDeviceInformation
            {
               DeviceName = model.Name,
               DeviceDescription = model.Description,
               DeviceTypeName = model.DeviceFactoryTypeName,
```

```
                DriverTypeName = model.DriverFactoryTypeName,
                DriverTcpAddress = model.TcpAddress,
                DriverTcpPort = model.TcpPort,
                Starter1Description = model.Starter1Description,
                Starter2Description = model.Starter2Description,
                Starter3Description = model.Starter3Description,
                //Fider1Description = model.Fider1Description,
                FiderDescription1 = model.FiderDescription1,
                FiderDescription2 = model.FiderDescription2,
                FiderDescription3 = model.FiderDescription3,
            };
        }
    }
}


using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

namespace ULA.Presentation.Markups
{
    /// <summary>
    ///     Класс с расширениями для Grid. используется для функционала "перетаскивания" виджетов по
списку устройств
    /// </summary>
    public static class GridExtentions
    {
        /// <summary>
        ///     Метод ищет в родителях свойства указанное свойства типа "T"
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="root"></param>
        /// <returns></returns>
        public static T Parent<T>(this DependencyObject root) where T : class
        {
            if (root is T) { return root as T; }

            DependencyObject parent = VisualTreeHelper.GetParent(root);
            return parent != null ? parent.Parent<T>() : null;
        }

        /// <summary>
        ///     Возвращает точку как номер строк и колонок
        /// </summary>
        /// <param name="obj"></param>
        /// <param name="relativePoint"></param>
        /// <returns></returns>
        public static Point GetColumnRow(this Grid obj, Point relativePoint) { return new Point(GetColumn(obj,
relativePoint.X), GetRow(obj, relativePoint.Y)); }
        private static int GetRow(Grid obj, double relativeY) { return GetData(obj.RowDefinitions, relativeY); }
        private static int GetColumn(Grid obj, double relativeX) { return GetData(obj.ColumnDefinitions, relativeX); }

        private static int GetData<T>(IEnumerable<T> list, double value) where T : DefinitionBase
        {
            var start = 0.0;
            var result = 0;

            var property = typeof(T).GetProperties().FirstOrDefault(p => p.Name.StartsWith("Actual"));
            if (property == null) { return result; }
```

```
            foreach (var definition in list)
            {
               start += (double)property.GetValue(definition, null);
               if (value < start) { break; }

               result++;
            }

            return result;
         }
      }
   }
```

```
using System;
using Microsoft.Practices.Prism.Events;
using ULA.Presentation.Behaviors.Interactions;
using ULA.Presentation.Infrastructure.Services;
using ULA.Presentation.Infrastructure.Services.Interactions;
using ULA.Presentation.Infrastructure.ViewModels.Interactions;
using ULA.Presentation.Services.Interactions;
using YP.Toolkit.System.Validation;

namespace ULA.Presentation.Services
{
   /// <summary>
   ///    Represents a default system interaction service functionality
   ///    Представляет функционал сервиса интерактивного вызова
   /// </summary>
   public class InteractionService : IInteractionService
   {
      #region [Private fields]

      private readonly IEventAggregator _eventAggregator;

      #endregion [Private fields]

      #region [Ctor's]

      /// <summary>
      ///    Creates an instance of <see cref="InteractionService" />
      /// </summary>
      /// <param name="eventAggregator">
      ///    An instance of <see cref="IEventAggregator" /> to use
      /// </param>
      public InteractionService(IEventAggregator eventAggregator)
      {
         this._eventAggregator = eventAggregator;
      }

      #endregion [Ctor's]

      #region [IInteractionService members]

      /// <summary>
      ///    Sends an interaction request to a user
      ///    Отправляет интерактивный запрос пользователю(Вызов модального окна)
      /// </summary>
      /// <typeparam name="TViewModel">
      ///    <see cref="Type" /> that represents the view model of the interaction request
```

```
/// </typeparam>
/// <param name="provider">
///     An instance of <see cref="IInteractionProvider{TViewModel}" /> that represents current
///     interaction request provider
/// </param>
/// <param name="interactionInterceptor">
///     An instance of <see cref="InteractionInterceptor{TViewModel}" /> to use
/// </param>
/// <returns>
///     An instance of <see cref="IInteractionToken" /> that represents interaction request token
/// </returns>
public IInteractionToken Interact<TViewModel>(IInteractionProvider<TViewModel> provider,
   InteractionInterceptor<TViewModel> interactionInterceptor)
   where TViewModel : IInteractionViewModel
{
   Guard.AgainstNullReference(provider, "provider");

   var interactionEvent = this._eventAggregator.GetEvent<InteractionRequestEvent>();
   var result = new InteractionToken(interactionEvent);

   interactionEvent.Publish(new InteractionRequestEventArgs
   {
      IsCanceled = false,
      InteractionProvider = provider,
      InterceptorsInvoker = new InteractionInterceptorsInvoker<TViewModel>(interactionInterceptor),
      Callback = result.Dispose
   });

   return result;
}


/// <summary>
///     Sends an interaction request to a user
///  Отправляет интерактивный запрос пользователю(Вызов модального окна)
/// </summary>
/// <typeparam name="TViewModel">
///     <see cref="Type" /> that represents the view model of the interaction request
///     Тип вью-модели на которую осуществляется переход
/// </typeparam>
/// <param name="provider">
///     An instance of <see cref="IInteractionProvider{TViewModel}" /> that represents current
///     interaction request provider
///     Провайдер интерактивного запроса
/// </param>
/// <param name="onInitialized">
///     An instance of <see cref="Action{T}" /> that represents an initialization callback
///     Функция инициализации
/// </param>
/// <param name="onUninitialized">
///     An instance of <see cref="Action{T}" /> that represents an uninitialization callback
/// Функция деинициализации
/// </param>
/// <returns>
///     An instance of <see cref="IInteractionToken" /> that represents interaction request token
/// </returns>
public IInteractionToken Interact<TViewModel>(IInteractionProvider<TViewModel> provider,
   Action<TViewModel> onInitialized = null,
   Action<TViewModel> onUninitialized = null)
   where TViewModel : IInteractionViewModel
{
   return Interact(provider, new InteractionInterceptor<TViewModel>
```

```
            {
                OnAfterInitialization = onInitialized,
                OnAfterUnInitialization = onUninitialized
            });
        }

        #endregion [IInteractionService members]
    }
}


using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;
using System.Windows.Forms;
using System.Windows.Input;
using Microsoft.Practices.Prism.Commands;
using Microsoft.Practices.Prism.Regions;
using Microsoft.Practices.ServiceLocation;
using NLog;
using ULA.AddinsHost.Business.Device;
using ULA.Business.Infrastructure.ApplicationModes;
using ULA.Business.Infrastructure.PersistanceServices;
using ULA.Interceptors.Application;
using ULA.Localization;
using ULA.Presentation.DTOs;
using ULA.Presentation.Infrastructure.DTOs;
using ULA.Presentation.Infrastructure.Services;
using ULA.Presentation.Infrastructure.Services.Interactions;
using ULA.Presentation.Infrastructure.ViewModels;
using ULA.Presentation.Infrastructure.ViewModels.Interactions;
using ULA.Presentation.Services.Interactions;
using ULA.Presentation.Views.Interactions;
using YP.Toolkit.System.ParallelExtensions;
using YP.Toolkit.System.SystemExtensions;
using YP.Toolkit.System.Validation;
using Application = System.Windows.Application;
using MessageBoxResult = ULA.Presentation.Infrastructure.ViewModels.MessageBoxResult;

namespace ULA.Presentation.ViewModels
{
    /// <summary>
    ///     Represents an application configuration mode view model functionality
    ///     Представляет вью-модель для режима конфигурации приложения
    /// </summary>
    public class ApplicationConfigurationModeViewModel : ViewModelBase,
IApplicationConfigurationModeViewModel
    {
        #region [Private fields]

        private readonly IConfigurationModeDevicesService _configService;
        private readonly IInteractionService _interactionService;
        private readonly IApplicationSettingsService _settings;
```

```
        private ICommand _createNewDeviceCommand;
        private IConfigLogicalDevice _currentDevice;
        private ICommand _deleteCurrentDeviceCommand;
        private ObservableCollection<IConfigLogicalDevice> _devices;
        private ICommand _editCurrentDeviceCommand;
        private ICommand _navigateToAboutCommand;
        private ICommand _navigateToJournalOfSystemCommand;
        private ICommand _navigateToUpdatePartialTimeoutCommand;
        private ICommand _navigateToUpdateFullTimeoutCommand;
        private ICommand _navigateToEditPasswordCommand;
        private ICommand _saveNewPositionCommand;
        private ICommand _navigateToHelpCommand;
        private Logger _logger;

        //это чтобы достать Port и Address. Переделать!
        private IPersistanceService _persistanceService;

        #endregion [Private fields]

        #region [Properties]

        /// <summary>
        ///     Gets or sets an instance of <see cref="IConfigLogicalDevice" /> that represents current virtual device
model
        ///     Текущее(в данный момент выбранное) устройство
        /// </summary>
        public IConfigLogicalDevice CurrentDevice
        {
            get { return this._currentDevice; }
            set
            {
                if (this._currentDevice != null && this._currentDevice.Equals(value)) return;
                this._currentDevice = value;
                OnPropertyChanged("CurrentDevice");
                CommandManager.InvalidateRequerySuggested();
            }
        }

        /// <summary>
        ///     Gets or sets a collection of devices
        ///     Коллекция устройств
        /// </summary>
        public ObservableCollection<IConfigLogicalDevice> Devices
        {
            get { return this._devices; }
            set
            {
                this.OnPropertyChanging("Devices");
                this._devices = value;
                this.OnPropertyChanged("Devices");
            }
        }

        #endregion [Properties]

        #region [Ctor's]

        /// <summary>
        ///     Creates an instance of <see cref="ApplicationConfigurationModeViewModel" />
        /// </summary>
        /// <param name="interactionService">
        ///     An instance of <see cref="IInteractionService" /> to use
```

```
        /// </param>
        /// <param name="configService">
        ///     An instance of <see cref="IConfigurationModeDevicesService" /> to use
        /// </param>
        /// <param name="persistanceService"></param>
        /// <param name="settings">Сервис с настройками приложения</param>
        public ApplicationConfigurationModeViewModel(IInteractionService interactionService,
                                        IConfigurationModeDevicesService configService,
                                        IPersistanceService persistanceService,
                                        IApplicationSettingsService settings)
        {
            Guard.AgainstNullReference(interactionService, "interactionService");
            Guard.AgainstNullReference(configService, "configService");
            Guard.AgainstNullReference(persistanceService, "persistanceService");
            Guard.AgainstNullReference(settings, "settings");

            this._interactionService = interactionService;
            this._configService = configService;
            this._persistanceService = persistanceService;
            this._settings = settings;

            this._logger = LogManager.GetLogger("Конфигурация");
        }

        #endregion [Ctor's]

        #region [Public members]
        /// <summary>
        ///     Изменяет расположение устройства и решает коллизии перемещения устройств
        /// </summary>
        /// <param name="device">перемещаемое устройство</param>
        /// <param name="newPosition">новая позиция устройства</param>
        public void ChangeDevicePositionNumber(IConfigLogicalDevice device, long newPosition)
        {
            var repeatPositionDevice = this.Devices.FirstOrDefault(dev => dev.Number.Equals(newPosition));
            if (repeatPositionDevice != null)
            {
                repeatPositionDevice.Number = device.Number;
            }
            device.Number = newPosition;
        }

        #endregion [Public members]

        #region [IApplicationConfigurationModeViewModel members]
        /// <summary>
        ///  Навигация на окно помощи
        /// </summary>
        public ICommand NavigateToHelpCommand
        {
            get
            {
                return this._navigateToHelpCommand ??
                      (this._navigateToHelpCommand = new DelegateCommand(OnNavigateToHelpCommand));
            }
        }

        /// <summary>
        ///     Команда навигации на водтверждение сохранения местоположения виджетов и  сохранения при
согласии
        /// </summary>
        public ICommand SaveNewPositionCommand
```

```
        {
            get
            {
                return this._saveNewPositionCommand ??
                    (this._saveNewPositionCommand = new DelegateCommand(OnSaveNewPositionCommand));
            }
        }

        /// <summary>
        ///     Gets an instance of <see cref="ICommand" /> that represents navigate to journal of system view
command
        ///     Команда навигации на журнал системы
        /// </summary>
        public ICommand NavigateToJournalOfSystemCommand
        {
            get
            {
                return this._navigateToJournalOfSystemCommand ??
                    (this._navigateToJournalOfSystemCommand = new
DelegateCommand(this.OnNavigateToJournalOfSystem));
            }
        }

        /// <summary>
        ///     Gets an instance of <see cref="ICommand" /> that represents create new virtual device command
        ///     Команда навигации на создание устройства
        /// </summary>
        public ICommand CreateNewVirtualDeviceCommand
        {
            get
            {
                return this._createNewDeviceCommand ??
                    (this._createNewDeviceCommand = new DelegateCommand(this.OnCreateNewDeviceCommand));
            }
        }

        /// <summary>
        ///     Gets an instance of <see cref="ICommand" /> that represents edit current virtual device command
        ///     Команда навигации на редактирование устройства
        /// </summary>
        public ICommand EditCurrentDeviceCommand
        {
            get
            {
                return this._editCurrentDeviceCommand ??
                    (this._editCurrentDeviceCommand =
                     new DelegateCommand<IConfigLogicalDevice>(this.OnEditDeviceCommand));
            }
        }

        /// <summary>
        ///     Gets an instance of <see cref="ICommand" /> that represents delete virtual device command
        ///     Команда удаление текущего устройства
        /// </summary>
        public ICommand DeleteCurrentDeviceCommand
        {
            get
            {
                return this._deleteCurrentDeviceCommand ??
                    (this._deleteCurrentDeviceCommand =
                     new DelegateCommand<IConfigLogicalDevice>(this.OnDeleteCurrentDevice));
            }
```

```
    }

    /// <summary>
    ///     Gets an instance of <see cref="ICommand" /> that represents navigate to about view command
    ///     Команда навигации на овно "О программе"
    /// </summary>
    public ICommand NavigateToAboutCommand
    {
       get
       {
          return this._navigateToAboutCommand ??
                (this._navigateToAboutCommand = new DelegateCommand(this.OnNavigateToAbout));
       }
    }

    /// <summary>
    ///     Возвращает команду, которая описывает навигацию на вьюшку установки частичного интервала
обновлений
    ///
    ///     Навигация на окно установки периодаобновлений
    /// </summary>
    public ICommand NavigateToTimeoutPartialPeriodCommand
    {
       get
       {
          return this._navigateToUpdatePartialTimeoutCommand ??
                (this._navigateToUpdatePartialTimeoutCommand =
                   new DelegateCommand(this.OnNavigateToPartialTimaoutPeriod));
       }
    }


    /// <summary>
    ///     Возвращает команду, которая описывает навигацию на вьюшку установки полного интервала
обновлений
    ///     Навигация на окно установки периодаобновлений
    /// </summary>
    public ICommand NavigateToTimeoutFullPeriodCommand
    {
       get
       {
          return this._navigateToUpdateFullTimeoutCommand ??
                (this._navigateToUpdateFullTimeoutCommand =
                   new DelegateCommand(this.OnNavigateToFullTimaoutPeriod));
       }
    }


    /// <summary>
    ///     Команда навигации на вьюшку изменения пароля привелигированного доступа
    /// </summary>
    public ICommand NaigateToEditPassword
    {
       get
       {
          return this._navigateToEditPasswordCommand ??
                (this._navigateToEditPasswordCommand = new DelegateCommand(OnNavigateToEditPassword));
       }
    }

    /// <summary>
    ///     Initializes current application mode asynchronously
```

```
        ///     Асинхронная инициализация
        /// </summary>
        /// <returns>
        ///     An instance of System.Threading.Tasks.Task that represents current asynchronous operation
        /// </returns>
        public async Task InitializeAsync()
        {
            var devices = await this._configService.GetAllDevicesAsync();
            this.Devices =
                new ObservableCollection<IConfigLogicalDevice>(
                    devices.OrderBy(dev => dev.CreateMomento().State.DataContainer.GetValue<long>("Number")));
            this.CurrentDevice = this.Devices.FirstOrDefault();
            this._logger.Trace(DateTime.Now.ToString(new CultureInfo("de-DE")) + " Переход в режим
конфигурации");
        }

        /// <summary>
        ///     Un-initializes current application mode asynchronously
        ///     Асинхронная деинициализация
        /// </summary>
        /// <returns>
        ///     An instance of System.Threading.Tasks.Task that represents current asynchronous operation
        /// </returns>
        public Task UninitializeAsync()
        {
            var regionManager = ServiceLocator.Current.GetInstance<IRegionManager>();
            regionManager.Regions.Remove("ModeRuntimeRegion");
            regionManager.Regions.Remove("ModeConfigurationRegion");
            return Task<object>.Factory.FromResult(null);
        }

        /// <summary>
        ///     Invokes when initialization process is over
        ///     Вызывается, когда процесс инициализации закончен
        /// </summary>
        public void OnInitialized()
        {
            if (this.Devices.Count != 0) return;
            this._interactionService
                .Interact(ApplicationInteractionProviders.QuestionMessageBoxInteractionProvider,
                        onInitialized: dialog =>
                            {
                                dialog.Title = LocalizationResources.Instance.NoDeviceDetected;
                                dialog.Message = LocalizationResources.Instance.WantToCreateNewDevice;
                            },
                        onUninitialized: dialog =>
                            {
                                if (dialog.Result == MessageBoxResult.YES)
                                    this.CreateNewDeviceWithInteraction();
                            });
        }

        #endregion [IApplicationConfigurationModeViewModel members]

        #region [Help members]

        private void OnNavigateToHelpCommand()
        {
            var path = Directory.GetCurrentDirectory();
            Help.ShowHelp(null, "file://" + path + "\\Files\\ULA.chm");
        }
```

```
private void OnSaveNewPositionCommand()
{
   this._interactionService
      .Interact(ApplicationInteractionProviders.QuestionMessageBoxInteractionProvider,
         viewModel =>
         {
            viewModel.Title = "Вы уверены?";
            viewModel.Message = "Вы действительно хотите сохранить текущее положение устройств";
         },
         viewModel =>
         {
            if (viewModel.Result == MessageBoxResult.YES)
            {
               this.SaveNewWidgetPosition();
            }
         });
}

private async void SaveNewWidgetPosition()
{
   var busyToken = this.InteractWithBusy();
   try
   {
      for (int i = 0; i < this.Devices.Count; i++)
      {
         if (this.Devices[i].ModifyNumberToSave)
         {
            await this._configService.UpdateDevicePositionAsync(this.Devices[i]);
         }
      }
   }
   catch (Exception error)
   {
      busyToken.Dispose();
      this.InteractWithError(error);
   }
   busyToken.Dispose();
}

private bool OnCanEditCurrentDevice(IConfigLogicalDevice currentDevice)
{
   return currentDevice != null;
}

private void OnEditDeviceCommand(IConfigLogicalDevice currentDevice)
{
   if (!OnCanEditCurrentDevice(this.CurrentDevice))
   {
      this._interactionService
         .Interact(ApplicationInteractionProviders.WarningMessageBoxInteractionProvider,
            viewModel =>
            {
               viewModel.ButtonType = MessageBoxButtonType.OK;
               viewModel.Title = "Команда не может быть выполнена";
               viewModel.Message = "Устройство для редактирования не выбрано";
            },
            viewModel => { });
      return;
   }
   this._interactionService
      .Interact(ApplicationInteractionProviders.CreateNewDeviceProvider,
         viewModel =>
```

```
            {
                viewModel.SetAlreadyExistingDevices(
                    this.Devices.Select(s => s.ToDeviceModel()));
                var editDeviceModel = this.CurrentDevice.ToDeviceModel();
                var state = this._persistanceService.GetDriverPersistableContextAsync(
                    Guid.Parse(currentDevice.CreateMomento().State.DataContainer.GetValue<string>("DriverId")))
                    .Result.GetDriverMomentoAsync().Result.State;

                editDeviceModel.TcpAddress = state.GetTcpAddress();
                editDeviceModel.TcpPort = state.GetTcpPort();
                viewModel.SetEditingDevice(editDeviceModel);
            },
            viewModel =>
            {
                if (!viewModel.IsCanceled)
                {
                    this.OnEditDevice(viewModel.ResultedDeviceModel);
                }
            });
}

private void OnCreateNewDeviceCommand()
{
    this.CreateNewDeviceWithInteraction();
}

private void OnNavigateToPartialTimaoutPeriod()
{
    string fileName = _settings.UpdateTimeFilePath;
    this._interactionService.Interact(ApplicationInteractionProviders.UpdateTimeoutInteractionProvider,
        viewModel =>
        {
            int timeout = 20;

            if (File.Exists(fileName))
            {
                using (var reader = new StreamReader(fileName))
                {
                    var readData = reader.ReadToEnd();
                    var updateTimes = readData.Split(' ');
                    timeout = Convert.ToInt32(updateTimes[0]);
                }
            }
            viewModel.Timeout = timeout;
        },
        viewModel =>
        {
            if (viewModel.Result == MessageBoxResult.OK)
            {
                if (File.Exists(fileName) == false)
                {
                    using (var stream = File.Create(fileName))
                    {
                        using (var writer = new StreamWriter(stream))
                        {
                            writer.Write("20 5");
                        }
                    }
                }
                var fullTimeout = 0;
                using (var reader = new StreamReader(fileName))
                {
```

```
                var readData = reader.ReadToEnd();
                var updateTimes = readData.Split(' ');
                //partialTimeout = Convert.ToInt32(updateTimes[0]);
                fullTimeout = Convert.ToInt32(updateTimes[1]);
            }
            var fileInfo = new FileInfo(fileName);
            using (var writer = new StreamWriter(fileInfo.Open(FileMode.Truncate)))
            {
                writer.Write(viewModel.Timeout.ToString(CultureInfo.InvariantCulture) + " " + fullTimeout);
            }
        }
    });
}

private void OnNavigateToFullTimaoutPeriod()
{
    string fileName = this._settings.UpdateTimeFilePath;
    this._interactionService.Interact(ApplicationInteractionProviders.UpdateTimeoutInteractionProvider,
        viewModel =>
        {
            int timeout = 20;

            if (File.Exists(fileName))
            {
                using (var reader = new StreamReader(fileName))
                {
                    var readData = reader.ReadToEnd();
                    var updateTimes = readData.Split(' ');
                    timeout = Convert.ToInt32(updateTimes[1]);
                }
            }
            viewModel.Timeout = timeout;
        },
        viewModel =>
        {
            if (viewModel.Result == MessageBoxResult.OK)
            {
                if (File.Exists(fileName) == false)
                {
                    using (var stream = File.Create(fileName))
                    {
                        using (var writer = new StreamWriter(stream))
                        {
                            writer.Write("20 5");
                        }
                    }
                }
                var partialTimeout = 0;
                using (var reader = new StreamReader(fileName))
                {
                    var readData = reader.ReadToEnd();
                    var updateTimes = readData.Split(' ');
                    partialTimeout = Convert.ToInt32(updateTimes[0]);
                    //fullTimeout = Convert.ToInt32(updateTimes[1]);
                }
                var fileInfo = new FileInfo(fileName);
                using (var writer = new StreamWriter(fileInfo.Open(FileMode.Truncate)))
                {
                    writer.Write(partialTimeout + " " + viewModel.Timeout.ToString(CultureInfo.InvariantCulture));
                }
            }
        });
```

```
}

private void OnNavigateToEditPassword()
{
   this._interactionService.Interact(ApplicationInteractionProviders.PasswordInteractionProvider,
      viewModel => { viewModel.Title = "Введите пароль"; },
      viewModel =>
      {
         var validPassword = GetPasswordFromFile();
         var inputPassword = new byte[] { 0 };
         if (viewModel.Password != null)
         {
            inputPassword = MD5.Create().ComputeHash(Encoding.Unicode.GetBytes(viewModel.Password));
         }

         if (viewModel.Result == MessageBoxResult.OK && viewModel.Password != null &&
            !inputPassword.Where((t, i) => t != validPassword[i]).Any())
         {
            this._interactionService.Interact(ApplicationInteractionProviders.PasswordInteractionProvider,
               newViewModel => { newViewModel.Title = "Введите новый пароль"; },
               newViewModel =>
               {
                  if (newViewModel.Result == MessageBoxResult.OK && newViewModel.Password != null)
                  {
                     StorePassword(newViewModel.Password);
                  }
               });
         }
         else if (viewModel.Result == MessageBoxResult.OK)
         {
            this._interactionService.Interact(
               ApplicationInteractionProviders.InformationMessageBoxInteractionProvider,
               viewModel2 =>
               {
                  viewModel2.Title = "Нет доступа";
                  viewModel2.Message = "Пароль введен не верно";
               }, viewModel2 => { });
         }
      });
}

private void StorePassword(string password)
{

   var filePath = "password.txt";
   if (!File.Exists(filePath))
   {
      using (var newFile = File.Create(filePath))
      {
         byte[] newInfo = Encoding.Unicode.GetBytes(password);
         var newResult = MD5.Create().ComputeHash(newInfo);
         newFile.Write(newResult, 0, newResult.Length);
      }
   }
   using (var file = new FileStream(filePath, FileMode.Open, FileAccess.Write))
   {
      byte[] info = Encoding.Unicode.GetBytes(password);
      var result = MD5.Create().ComputeHash(info);
      file.Write(result, 0, result.Length);
   }
}
```

```
        private byte[] GetPasswordFromFile()
        {
            var filePath = "password.txt";
            if (!File.Exists(filePath))
            {
                using (var newFile = File.Create(filePath))
                {
                    byte[] info = Encoding.Unicode.GetBytes("bemn");
                    var result = MD5.Create().ComputeHash(info);
                    newFile.Write(result, 0, result.Length);
                }
            }
            var file = new FileStream(filePath, FileMode.Open, FileAccess.Read);
            return file.ReadAllBytes();
        }

        private bool OnCanDeleteCurrentDevice(IConfigLogicalDevice currentDevice)
        {
            return currentDevice != null;
        }

        private void OnDeleteCurrentDevice(IConfigLogicalDevice currentDevice)
        {
            if (!OnCanDeleteCurrentDevice(this.CurrentDevice))
            {
                this._interactionService
                    .Interact(ApplicationInteractionProviders.WarningMessageBoxInteractionProvider,
                        viewModel =>
                        {
                            viewModel.ButtonType = MessageBoxButtonType.OK;
                            viewModel.Title = "Команда не может быть выполнена";
                            viewModel.Message = "Устройство для удаления не выбрано";
                        },
                        viewModel => { });
                return;
            }
            this._interactionService
                .Interact(ApplicationInteractionProviders.QuestionMessageBoxInteractionProvider,
                    viewModel =>
                    {
                        viewModel.Message = "Вы действительно хотите удалить устройство с именем \"" +
currentDevice.DeviceName + "\"?";
                        viewModel.Title = "Вы уверены?";
                    },
                        viewModel =>
                        {
                            if (viewModel.Result == MessageBoxResult.YES)
                                this.OnDeleteDevice(this.CurrentDevice);
                        });
        }

        private void OnNavigateToAbout()
        {
            this._interactionService
                .Interact(ApplicationInteractionProviders.AboutInteractionProvider);
        }

        private void OnNavigateToJournalOfSystem()
        {
            try
            {
                var logData = GetDataForJournalOfSystem();
```

```
            logData.Reverse(0, logData.Count);
            var logWnd = new Window();
            logWnd.Owner = Application.Current.MainWindow;
            var logView = new LogInteractionView();
            var vm = ServiceLocator.Current.GetInstance<ILogInteractionViewModel>();
            vm.LogCollection = CollectionViewSource.GetDefaultView(
                        new ObservableCollection<LogInformation>(logData));
            logView.DataContext = vm;
            logView.Margin = new Thickness(5, 5, 5, 5);
            logWnd.Content = logView;
            logWnd.Title = "Журнал системы";
            logWnd.ResizeMode = ResizeMode.CanMinimize;
            logWnd.Height = 625;
            logWnd.Width = 670;
            logWnd.Show();
            //var dataForJournal = GetDataForJournalOfSystem();
            //this._interactionService
            //.Interact(ApplicationInteractionProviders.LogInteractionProvider,
            //viewModel =>
            //{
            //    viewModel.LogCollection = CollectionViewSource.GetDefaultView(new
ObservableCollection<LogInformation>(dataForJournal));
            //});
         }
         catch (Exception error)
         {
            this.InteractWithError(error);
         }
      }

      private List<LogInformation> GetDataForJournalOfSystem()
      {
         var fileName = _settings.JournalFilePath;
         if (!File.Exists(fileName))
         {
            using (File.Create(fileName)) { }
            return new List<LogInformation>();
         }
         var allLog = File.ReadAllLines(fileName,
            Encoding.Unicode).ToList();
         const string DATE_TIME_PATTERN = @"\d{2}\.\d{2}\.\d{4}\s\d{1,2}:\d{2}:\d{2}";
         var resultData = new List<LogInformation>();
         foreach (var log in allLog)
         {
            var dateTimeString = Regex.Match(log, DATE_TIME_PATTERN).Value;
            var dateTime = Convert.ToDateTime(dateTimeString, new CultureInfo("de-DE"));
            int dateTimeFirstWhitespace = log.IndexOf(" ", StringComparison.InvariantCulture);
            var logWithoutMagicNumber = log.Substring(dateTimeFirstWhitespace);
            var description = Regex.Replace(logWithoutMagicNumber, DATE_TIME_PATTERN, String.Empty);
            resultData.Add(new LogInformation(description, dateTime));
         }
         return resultData;
      }

      private void CreateNewDeviceWithInteraction()
      {
         this._interactionService
            .Interact(ApplicationInteractionProviders.CreateNewDeviceProvider,
                  viewModel => viewModel.SetAlreadyExistingDevices(
                     this.Devices.Select(s => s.ToDeviceModel())),
                  viewModel =>
                  {
```

```
                if (!viewModel.IsCanceled)
                    this.OnCreateNewDevice(viewModel.ResultedDeviceModel);
            });
    }

    private async void OnCreateNewDevice(DeviceModel deviceModel)
    {
        var busyToken = this.InteractWithBusy();
        try
        {
            var resultedDevice = await this._configService.CreateDeviceAsync(deviceModel.ToLogicalDeviceInfo());
            this.Devices.Add(resultedDevice);
            this.SaveNewWidgetPosition();
        }
        catch (Exception error)
        {
            busyToken.Dispose();
            this.InteractWithError(error);
        }
        busyToken.Dispose();
    }

    private async void OnEditDevice(DeviceModel deviceModel)
    {
        var busyToken = this.InteractWithBusy();
        try
        {
            var editingDevice = await this._configService.UpdateDeviceAsync(deviceModel.ToLogicalDeviceInfo(),
CurrentDevice);
            this.Devices.Remove(this.CurrentDevice);
            this.Devices.Add(editingDevice);
            this.Devices =
            new ObservableCollection<IConfigLogicalDevice>(
                Devices.OrderBy(dev => dev.CreateMomento().State.DataContainer.GetValue<long>("Number")));
            this.CurrentDevice = editingDevice;
        }
        catch (Exception error)
        {
            busyToken.Dispose();
            this.InteractWithError(error);
        }
        busyToken.Dispose();
    }

    private async void OnDeleteDevice(IConfigLogicalDevice deviceModel)
    {
        var busyToken = this.InteractWithBusy();
        try
        {
            await this._configService.RemoveDeviceAsync(deviceModel);
            this.Devices.Remove(deviceModel);
        }
        catch (Exception error)
        {
            busyToken.Dispose();
            this.InteractWithError(error);
        }
        busyToken.Dispose();
    }

    private void InteractWithError(Exception error)
    {
```

```
            this._interactionService
                .Interact(ApplicationInteractionProviders.ErrorMessageBoxInteractionProvider,
                        dialog =>
                        {
                            dialog.ButtonType = MessageBoxButtonType.OK;
                            dialog.Title = LocalizationResources.Instance.ErrorHeader;
                            dialog.Message =
                                string.Format(LocalizationResources.Instance.UnexpectedErrorMessagePattern,
                                        error.Message);
                        });
        }

        private IInteractionToken InteractWithBusy()
        {
            return this._interactionService
                    .Interact(ApplicationInteractionProviders.BusyInteraction, dialog =>
                        {
                            dialog.Title = LocalizationResources.Instance.BusyDialogTitle;
                            dialog.Message = LocalizationResources.Instance.WaitingBusyDialogMessage;
                        });
        }

        #endregion [Help members]
        }
}
```