Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт электронного обучения
Специальность 230105
Кафедра автоматизации и компьютерных систем

ДИПЛОМНЫЙ ПРОЕКТ

Тема работы
Сервер сбора данных для гетерогенной системы учета энергоресурсов

УДК 004.383.2:621.31.031

Студент

Группа	ФИО	Подпись	Дата
З-8001	Поспелова Ирина Владимировна		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент	Скирневский И.П.	аспирант		

КОНСУЛЬТАНТЫ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент	Конотопский В.Ю.	к. э. н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент	Невский Е.С.			

ДОПУСТИТЬ К ЗАЩИТЕ:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
АиКС	Фадеев А.С.	к. т. н.		

Томск – 2016 г.

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт электронного обучения
Направление подготовки (специальность) 230105
Кафедра автоматизации и компьютерных систем

УТВЕРЖДАЮ:
Зав. кафедрой

_____ _____ _____
(Подпись) (Дата) (Ф.И.О.)

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

В форме:

Дипломного проекта

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	**ФИО**
З-8001	Поспелова Ирина Владимировна

Тема работы:

Сервер сбора данных для гетерогенной системы учета энергоресурсов	
Утверждена приказом директора (дата, номер)	от 15.04.2016 № 2917/с

Срок сдачи студентом выполненной работы:	06.06.2016

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.).*	Объектом разработки является приложение, обеспечивающее сбор данных с устройств сбора и передачи данных и передачу полученных данных на центральный сервер. Разрабатываемое приложение является частью автоматизированной системы коммерческого учета энергоресурсов. Программа должна быть представлена в виде консольного приложения и работать непрерывно в фоновом режиме. Исходными данными для разработки дипломного проекта являются следующие документы: – техническое задание на выполнение научно-исследовательских, опытно-конструкторских и технологических работ по теме: «Разработка гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов,

	программного обеспечения, а также разработка и реализация проектно-сметной документации на развертывание и проведение натурных испытаний на объектах»; – технические требования к результатам выполнения комплексного проекта по созданию высокотехнологичного производства с участием российского высшего учебного заведения по теме: «Реализация комплексного проекта по созданию высокотехнологичного производства интеллектуальных приборов энергоучета, разработанных и изготовленных на базе отечественных микроэлектронных компонентов, и гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов на их основе»; – договор №02.G25.31.0107 между ЗАО «ПКК Миландр» и Министерством образования и науки Российской Федерации об условиях предоставления и использования субсидии на реализацию комплексного проекта по созданию высокотехнологичного производства интеллектуальных приборов энергоучета, разработанных и изготовленных на базе отечественных микроэлектронных компонентов, и гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов на их основе.
Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).*	Разработке подлежат следующие пункты: – реализация протокола взаимодействия приложения с устройствами сбора и передачи данных; – реализация формирования и отправки на устройство бора и передачи данных запросов и управляющих команд по расписанию; – реализация протокола взаимодействия приложения с центральным сервером; – реализация системы регистрации событий приложения; – реализация временной базы данных и блока по работе с ней.
Перечень графического материала *(с точным указанием обязательных чертежей)*	– блок-схемы алгоритмов программы; – диаграммы вариантов использования приложения; – диаграммы классов приложения; – диаграммы последовательности приложения.

Консультанты по разделам выпускной квалификационной работы

(с указанием разделов)

Раздел	Консультант
	Скирневский Игорь Петрович
	Конотопский Владимир Юрьевич

	Невский Егор Сергеевич
Названия разделов, которые должны быть написаны на русском и иностранном языках:	
Заключение	

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	03 марта 2016 г.

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент	Скирневский И. П.	аспирант		03.03.2016

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
З-8001	Поспелова Ирина Владимировна		03.03.2016

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
З-8001	Поспелова Ирина Владимировна

Институт	электронного обучения	Кафедра	АиКС
Уровень образования	Специалитет	Направление/специальность	230105

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих*	
2. *Нормы и нормативы расходования ресурсов*	
3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого потенциала инженерных решений (ИР)*	
2. *Формирование плана и графика разработки и внедрения ИР*	
3. *Обоснование необходимых инвестиций для разработки и внедрения ИР*	
4. *Составление бюджета инженерного проекта (ИП)*	
5. *Оценка ресурсной, финансовой, социальной, бюджетной эффективности ИР и потенциальных рисков*	
Перечень графического материала *(с точным указанием обязательных чертежей)*	
1. *«Портрет» потребителя* 2. *Оценка конкурентоспособности ИР* 3. *Матрица SWOT* 4. *Модель Кано* 5. *ФСА диаграмма* 6. *Оценка перспективности нового продукта* 7. *График разработки и внедрения ИР* 8. *Инвестиционный план. Бюджет ИП* 9. *Основные показатели эффективности ИП* 10. *Риски ИП*	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент	Конотопский В.Ю.	к. э. н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
З-8001	Поспелова Ирина Владимировна		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	ФИО
З-8001	Поспелова Ирина Владимировна

Институт	электронного обучения	Кафедра	АиКС
Уровень образования	специалитет	Направление/специальность	230105

Социальная ответственность	Предназначение разделов социальной ответственности
1. Анализ возможных сбоев разрабатываемой системы	Возникновение сбоев в работе АСКУЭ может привести к потере, повреждению или получению недостоверных данных. Все сбои, происходящие в ходе работы программы, можно разделить на несколько категорий: – аппаратные сбои, к которым можно отнести выход из строя оборудования; – сбои, возникающие в результате неправильное эксплуатации программы; – программные сбои, возникающие в результате ошибок в алгоритмах программы. К этому же типу сбоев относится возникновение исключительных ситуаций, которые могут привести к аварийному завершению работы программы
2. Анализ причин сбоев в работе системы	К причинам возникновения сбоев в работе программы относится: – нарушение сетевого взаимодействия между компонентами системы; – получение несанкционированного доступа к данным системы злоумышленниками; – получение несанкционированного доступа к аппаратным компонентам системы; – выход из строя оборудования системы; – изменение данных пользователями, не обладающими соответствующими правами; – ошибки в алгоритмах программного обеспечения. Все перечисленные ошибки ведут

	к потере или повреждению данных, а следовательно, к нарушению функционирования всей системы в целом.
3. Меры по аппаратной защите системы	Способы аппаратной защиты системы включают в себя: – способы защиты от несанкционированного доступа к аппаратным компонентам системы: устройствам учета энергоресурсов и устройствам сбора и передачи данных. Это достигается путем опломбирования всех устройств и снабжении всех устройств системой оповещения оператора при вскрытии пломб; – способы защиты данных от выхода из строя аппаратных компонентов системы. К этой категории мер относится размещение программных компонентов системы на разных ЭВМ, размещение центральной базы данных и резервной базы данных на разных физических носителях, снабжение каждой ЭВМ устройством бесперебойного питания; – способы защиты данных от выхода из строя одного из каналов связи между устройствами.
4. Организационные меры, обеспечивающие защиту системы	К организационным мерам, обеспечивающим защиту системы отосятся: – обязательный инструктаж оператора системы перед началом выполнения своих обязанностей; – ведение формуляра системы, в котором должны фиксироваться все события, возникающие при эксплуатации приложения.
5. Меры по программной защите системы	Меры по программной защите системы включают в себя: – меры по защите целостности данных при передаче по сети. Это реализуется посредством внедрения в системы передачи данных механизма гарантированной доставки данных;

	– меры по защите данных от атак злоумышленников. Реализуются при помощи использования технологии VPN; – меры по защите данных от изменения пользователями, не обладающими соответствующими правами доступа. Осуществляется за счет разграничения прав разным группам пользователей; – меры по защите данных в случае повреждения базы данных. Защита базы данных осуществляется за счет регулярного копирования всех данных из базы в резервное хранилище данных; – меры по защите от сбоев в алгоритмах программного обеспечения. Осуществляются за счет тестирования программного обеспечения и обработки исключительных ситуаций; – контроль достоверности измерений УУЭ. Осуществляется посредством реализации механизма проверки корректности полученных данных; – наличие системы оповещений о событиях, произошедших в результате работы системы. Реализуется за счет журналирования всех событий, возникающих в ходе выполнения приложения.
6. Требования к аппаратному и программному обеспечению	Для корректного выполнения приложения установлены требования к аппаратному и программному обеспечению, а также требования к платформе, на которой должно выполняться приложение.

Дата выдачи задания для раздела по линейному графику	29.04.16

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент	Невский Е.С.			

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
З-8001	Поспелова Ирина Владимировна		

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт электронного обучения
Направление подготовки (специальность) 230105
Кафедра автоматизации и компьютерных систем
Уровень образования – специалитет
Период выполнения – весенний семестр 2015/2016 учебного года

Форма представления работы:

Дипломный проект

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	06.06.2016 г.

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
15.03.16	Описание АСКУЭ	10
25.03.16	Описание сервера сбора данных (ССД)	10
15.04.16	Описание программной архитектуры ПО ССД	15
05.05.16	Описание алгоритмов ПО ССД	15
16.05.16	Анализ результатов	10
23.05.16	Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	15
30.05.16	Социальная ответственность	15
01.06.16	Обязательное приложение на иностранном языке	10

Составил преподаватель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент	Скирневский И.П.	аспирант		03.03.2016

СОГЛАСОВАНО:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
АиКС	Фадеев А.С.	к. т. н.		03.03.2016

РЕФЕРАТ

Выпускная квалификационная работа 98 с., 33 рис., 14 табл., 25 источников, 3 прил.

Ключевые слова: сервер сбора данных, автоматизированная система коммерческого учета энергоресурсов, устройство сбора и передачи данных, центральный сервер, опрос по расписанию.

Объектом исследования является (ются) сервер сбора данных автоматизированной системы коммерческого учета энергоресурсов

Цель работы – разработка программного обеспечения для сервера сбора данных автоматизированной системы коммерческого учета энергоресурсов

В процессе проектирования проводилась разработка алгоритмов программного обеспечения для сервера сбора данных

В результате проектирования было разработано программное обеспечение для сервера сбора данных

Основные конструктивные, технологические и технико-эксплуатационные характеристики: рзаработанное приложение позволяет осуществлять опрос устройств сбора и передачи данных по расписанию, полученному от центрального сервера системы. Все сетевое взаимодействие между компонентами системы осуществляется по протоколу HTTP. Для обеспечения целостности даных при передаче по сети реализован механизм гарантированной доставки данных. Взаимодействие приложения с временной базой данных осуществляется посредством блока, построенного на технологии объектно-реляционного отображения. Мониторинг времени вызова задач в соответствии с расписанием осуществляется посредством спроектированного планировщика задач.

Степень внедрения: проект находится на стадии внедрения в состав программного обеспечения атвоматизированной системы коммерческого учета энергоресурсов.

Область применения: разработанное приложение является частью гетерогенной системы учета энергоресурсов, которая в свою очередь будет применяться для объектов жилого, коммерческого и производственного назначения.

Экономическая эффективность/значимость работы заключается в потенциальном снижении потребительской задолженности по оплате электроэнергии и в снижении затрат на персонал, контролирующий показания индивидуальных приборов учета электроэнергии.

В будущем планируется усовершенствовать сервер сбора данных, позволяя ему взаимодействовать с устройствами сбора и передачи данных разных производителей

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Настоящий документ содержит в себе следующие обозначения и сокращения:

АРМ – автоматизированное рабочее место;

АСКУЭ – автоматизированная система контроля и учета энергоресурсов;

БД – база данных;

ВБД – временная база данных;

ПО – программное обеспечение;

СБД – сервер баз данных;

СИ – сервисный инженер;

СП – сервер приложений;

ССД – сервер сбора данных;

УСПД – устройство сбора и передачи данных;

УУЭ – устройство учета энергоресурсов

ЦС – центральный сервер;

ЭВМ – электронно-вычислительная машина.

ВВЕДЕНИЕ

Применение в инженерной инфраструктуре жилищно-коммунального хозяйства гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов с интеллектуальных приборов энергоучета, разработанных и выпускаемых на основе российской элементной базы, обеспечивает реальные предпосылки для усиления технологической независимости государственной социальной инфраструктуры и, как следствие, является актуальной государственной задачей.

Однако, в отечественной практике до настоящего времени нет ни одного примера реализации гетерогенных систем сбора данных о потребляемых энергоресурсах в сфере жилищно-коммунального хозяйства, развернутых на базе интеллектуальных приборов энергоучета с российскими микросхемами. Под гетерогенной организацией сети подразумевается:

– применение интерфейсов передачи данных на центральный сервер, основанных на различных физических принципах: проводные интерфейсы Ethernet, беспроводные системы GSM, GPRS и др.;

– применение интерфейсов передачи данных в сети MI-LAN, основанных на различных физических принципах: беспроводные системы, проводные – выделенные линии связи RS-485, передача данных по силовым проводам PLC;

– применение различных по типу и составу узлов учета энергоресурсов.

В настоящий момент существует потребность в недорогой, простой в обслуживании, надёжной и функциональной автоматизированной системе коммерческого учета энергоресурсов (сокращенно АСКУЭ) для бытовых потребителей.

Создание аппаратно-программного обеспечения системы сбора данных о потребляемых энергоресурсах с приборов энергоучета на базе российских электронных компонентов позволит решить проблему комплексного импортозамещения. Гетерогенная организация сети позволит многократно увеличить достоверность передачи данных.

В настоящей выпускной квалификационной работе в качестве объекта разработки выступает программное обеспечение для сервера сбора данных автоматизированной системы коммерческого учета энергоресурсов. Сервер сбора данных является частью АСКУЭ и предназначен для обеспечения следующих функций:

– опрос устройств сбора и передачи данных (сокращенно УСПД) согласно составленному расписанию;

– хранение полученных от УСПД данных до момента их передачи на центральный сервер системы (сокращенно ЦС);

– передача полученных от УСПД данных на ЦС.

Настоящий дипломный проект включает в себя шесть разделов:

– первый раздел описывает АСКУЭ: ее назначение, область применения, выполняемые функции и структуру;

– второй раздел описывает сервер сбора данных (сокращенно ССД): его назначение, структуру и механизмы функционирования

– третий раздел описывает процесс проектирования ССД: разработку его модулей и алгоритмов;

– четвертый раздел описывает результаты разработки;

– пятый раздел обосновывает экономическую эффективность разрабатываемого проекта;

– шестой раздел описывает возможные уязвимости разрабатываемой системы и их последствия, а также способы их устранения.

1 ОПИСАНИЕ АСКУЭ

Данный раздел дает понятие АСКУЭ, описывает ее особенности, назначение и функции, а также логическую, физическую и программную структуру.

1.1 Цели, назначение и область применения АСКУЭ

Автоматизированная система коммерческого учета энергоресурсов (АСКУЭ) представляет собой систему для сбора показаний с узлов учета энергоресурсов и архивирования, хранения, анализа и визуализации полученных данных, формирования отчетов.

АСКУЭ предназначена для автоматизированного сбора и обработки данных о количестве потребляемой электроэнергии и мониторинга параметров однофазных и трёхфазных электросетей.

Область применения разрабатываемой АСКУЭ – объекты жилого, коммерческого и производственного назначения.

1.2 Особенности разрабатываемой системы

При эксплуатации сложных технических объектов одной из важнейших задач является оценка текущего состояния их подсистем. С помощью дистанционного контроля технического состояния оборудования и сравнения значений измеряемых параметров с допустимыми нормами, можно оценить работоспособность системы и своевременно выявить отказ какой-либо подсистемы объекта, и тем самым повысить эксплуатационную надежность объекта. Главными требованиями к таким системам мониторинга на сегодняшний день являются масштабируемость и гетерогенность. Здесь под гетерогенностью стоит понимать, как информационную гетерогенность – возможность эффективной работы разного типа датчиков в составе единой системы мониторинга, так и канальную – возможность использования различных каналов передачи данных.

Интегрирование в систему возможно при условии использования в приборах и датчиках стандартных протоколов обмена данными.

Еще одной особенностью разрабатываемой системы является то, что она развернута на базе интеллектуальных приборов энергоучета с отечественными микросхемами.

1.3 Функциональная структура АСКУЭ

Описание функциональной структуры АСКУЭ приведено на основании документа «Схема функциональной структуры» [2].

Схема функциональной структуры системы приведена на рисунке 1.1.

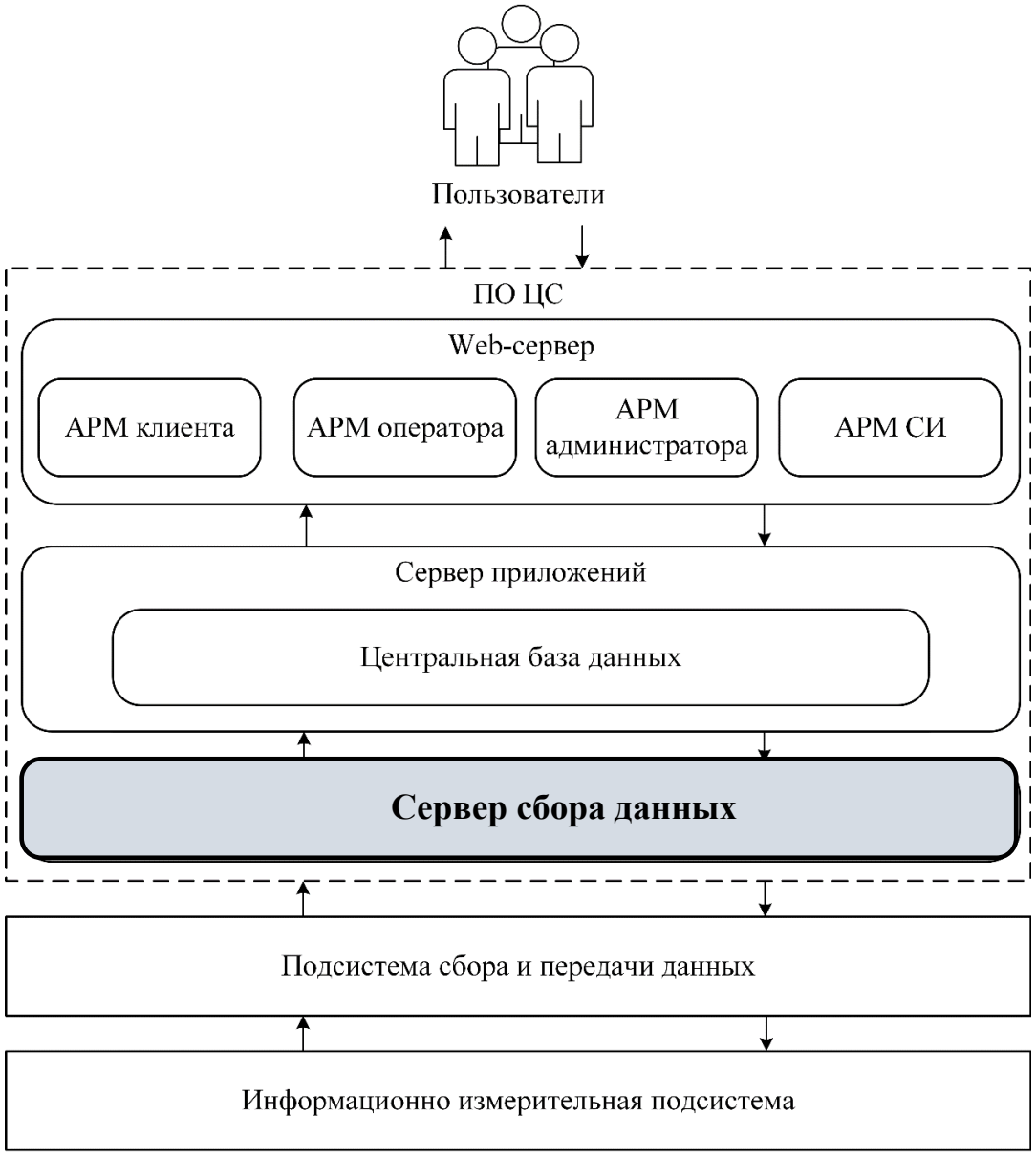


Рисунок 1.1 – Схема функциональной структуры АСКУЭ

В состав функциональной структуры входят следующие подсистемы:

– информационно-измерительная подсистема, реализующая следующие процедуры:

  – процедура измерения количества потреблённой электроэнергии;
  – процедура изменения настроек УУЭ;

– подсистема сбора и передачи данных, реализующая следующие процедуры:

  – процедура сбора и передачи данных;
  – процедура формирования управляющих команд;

– подсистема центрального сервера, реализующая следующие процедуры:

  – процедура ввода и хранения параметров системы учёта;
  – процедура обработки, хранения и визуализации данных.

Информационно-измерительная подсистема состоит из УУЭ.

Подсистема сбора и передачи данных состоит из устройств сбора и передачи данных (УСПД).

Подсистема центрального сервера (ЦС) включает в себя:

– сервер сбора данных;
– сервер приложений;
– web-сервер.

Взаимодействие функциональных подсистем между собой и с внешней средой реализуется непосредственно специальным программным обеспечением АСКУЭ, обеспечивающим автоматизированное функционирование соответствующего оборудования.

Взаимодействие УУЭ и УСПД происходит посредством цифровых интерфейсов (рисунок 1.2):

– по двухпроводному каналу связи по стандарту RS-485;
– RF-433 для передачи данных по радиоканалу;
– PLC для передачи данных по силовым линиям.

Сеть MI-LAN осуществляет взаимосвязь между УУЭ и УСПД, в том числе представляет возможность передачи данных от одного УУЭ до УСПД через другие УУЭ, используя их как ретрансляторы.

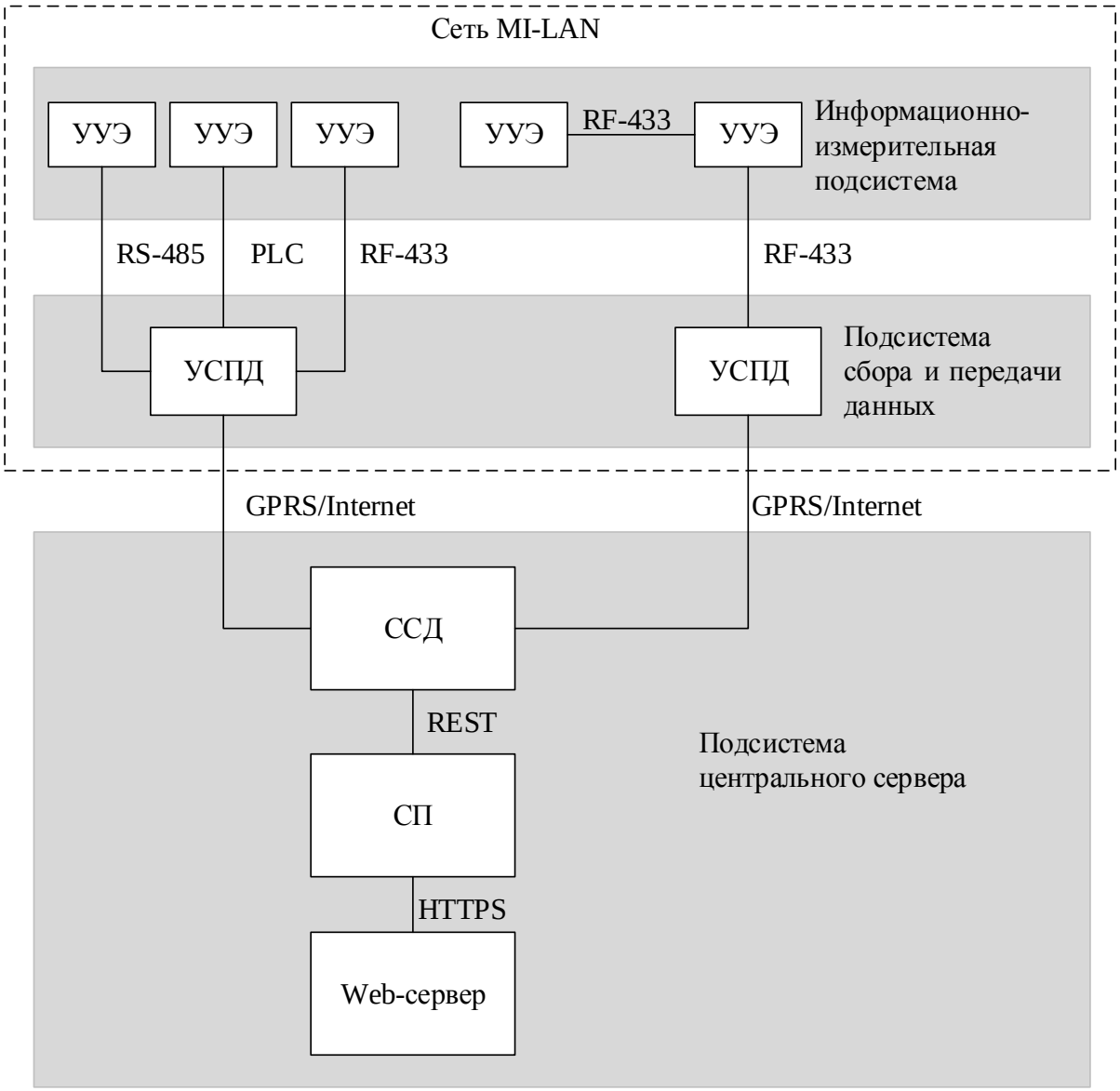


Рисунок 1.2 – Взаимодействие элементов АСКУЭ

УУЭ передают на УСПД:

– текущие показания, согласно установленным тарифным зонам, при их наличии;

– служебную информацию.

УСПД передает на УУЭ управляющие команды.

Взаимодействие между УСиПД и ЦС осуществляется по средствам сети VPN на основе сети Интернет или GSM.

УСПД передает на ЦС:

- текущие показания УУЭ, подключенных к УСПД;
- служебную информацию о работе УУЭ и УСПД.

ЦС передает на УСПД:

- параметры системы учета;
- управляющие команды для УСПД;
- системное время ЦС.

Пользователи взаимодействуют с центральным сервером АСКУЭ посредством автоматизированных рабочих мест (АРМ), реализованных в форме web-интерфейсов.

Для взаимодействия с другими системами учета предусмотрена возможность экспорта и импорта информации в XML-документы в соответствии с моделью документа 80020.

Функции импорта и экспорта позволяют предварительно указать формат XML-документа, что позволяет обеспечить совместимость с множеством систем.

Для корректной работы подсистем АСКУЭ необходима система синхронизации времени. Подсистема центрального сервера и средствами сети Интернет синхронизирует системное время со сторонними сервисами времени. Синхронизация системного времени подсистемы сбора и передачи данных осуществляется посредством:

- получения системного времени от ЦС;
- получения системного времени по сети Интернет от сторонних сервисов синхронизации времени.

Информационно-измерительная подсистема получает системное время командами управления от подсистемы сбора и передачи данных.

1.4 Функции, реализуемые подсистемой сбора и передачи данных

Подсистема сбора и передачи данных состоит из устройств сбора и передачи данных, реализующих следующие функции [3]:

– автоматизированный сбор информации от узлов учёта согласно требованиям п. 161 Постановления Правительства РФ от 04.05.2012 № 442 (ред. 04.09.2016) [1];

– передача агрегированных данных по запросу в подсистему ЦС;

– хранение показаний мощности УУЭ за установленный период;

– хранение значений потребленных ресурсов, подключенных УУЭ, по тарифным зонам.

2 ОПИСАНИЕ СЕРВЕРА СБОРА ДАННЫХ

Сервер сбора данных (ССД) выполняет функцию сбора данных о показаниях потребления электроэнергии из информационно измерительной подсистемы, через подсистему сбора и передачи данных.

Под процессом сбора данных в рамках реализации АСКУЭ понимается:

- сбор данных с УСПД по расписанию;
- передача управляющих команд на УСПД и УУЭ;
- передача данных, полученных от УСПД, на ЦС;
- журналирование событий процесса сбора данных [4].

2.1 Структура и механизмы функционирования ССД

Согласно описанию АСКУЭ, ССД является составной частью подсистемы ЦС. Функциональная структура ССД представлена на рисунке 2.1.

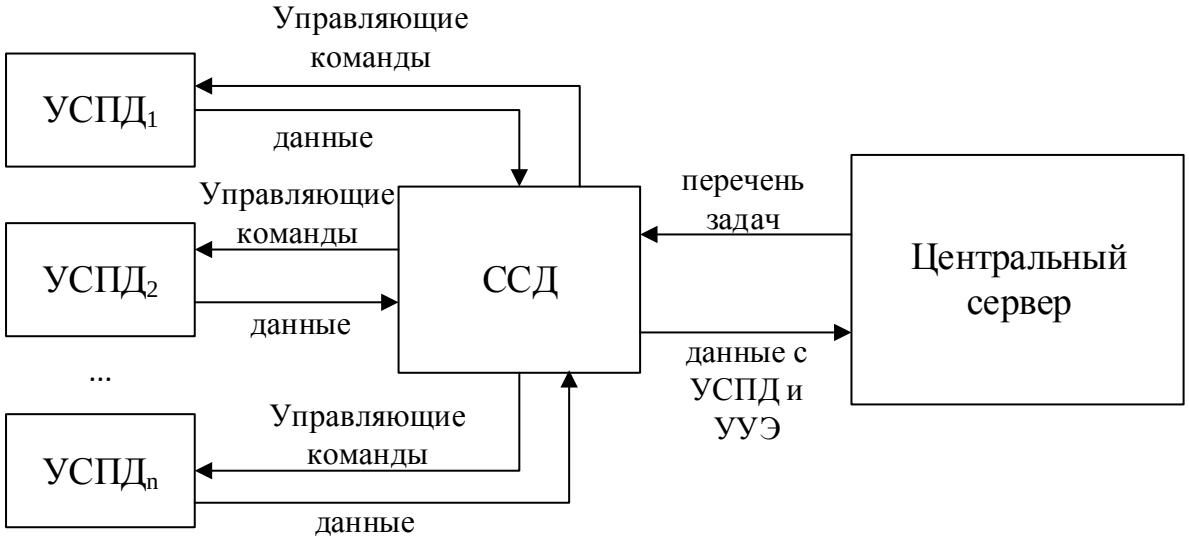


Рисунок 2.1 – Функциональная структура сервера сбора данных

Исходя из рисунка 2.1, ССД является посредником между УСПД и ЦС. ССД принимает от ЦС перечень задач и данные, необходимые для взаимодействия с УСПД. Исходя из полученных задач, ССД формирует управляющие команды, отправляет их на УСПД, получает необходимую информацию и пересылает ее на ЦС.

2.2 Описание набора данных, хранимых на ССД

ССД должен обеспечивать хранение следующей информации [5, 6]:

– данные, поступающие от УСПД и УУЭ, которые предполагается хранить на ССД до тех пор, пока они не будут отправлены в базу ЦС;

– перечень задач, которые необходимо выполнить. Как только задача завершается, она должна быть удалена;

– настройки параметров ССД (логин и пароль для соединения с УСПД, максимальный объем хранимых задач и пр.);

– журнал событий ССД;

– информация о подключенных УСПД.

2.3 Выбор средств разработки ПО ССД

В соответствии с требованиями предприятия, реализация программы выполнена на языке программирования C++ с использованием библиотеки QT 5.4.0. При разработке приложения была использована среда QT creator 3.3.0. Данный выбор обусловлен следующими факторами:

– после установки среды не требуется никакой дополнительной настройки для работы с библиотекой QT;

– простота установки и настройки;

– наличие графического отладчика;

– возможность работы с системами контроля версий;

– наличие подробной документации по использованию среды;

– наличие функции автоматического дополнения кода.

В качестве инструмента по работе с БД была использована СУБД SQLite. Данный выбор сделан на основании следующих факторов:

– в библиотеку QT встроены функции по работе с SQLite, поэтому ее использование не требует дополнительных настроек;

– надежность;

– простота использования;

– высокая производительность;

– хранение созданной БД в отдельном файле.

Основной недостаток SQLite заключается в том, что она не поддерживает крупные БД. Технические требования к ССД не предусматривают хранение

больших объемов информации, поэтому данный недостаток не повлияет на работу выбранной СУБД.

В качестве браузера для работы БД использовалось приложение SQLite browser 3.8.0. Данный выбор обусловлен простотой установки приложения, малыми размерами и наличием удобного и интуитивно понятного интерфейса.

В качестве средства проектирования ПО ССД использовалось приложение StarUML 5.0.

Также в ходе разработки проекта использовалась система контроля версий git [7]. В качестве приложения по работе с системой использовалось графическое приложение GitHub Gui.

3 ПРОЕКТИРОВАНИЕ ПО СЕРВЕРА СБОРА ДАННЫХ

Настоящий раздел рассматривает разработку функциональной и программной структуры ПО ССД в целом и каждого его блока, а также разработку алгоритмов программы. В завершение раздела описываются процессы отладки и тестирования для каждого блока приложения.

3.1 Назначение и функции ПО ССД

Назначение ПО ССД заключается в сборе данных с УСПД и отправке полученной информации на ЦС.

Основные функции, которые выполняет приложение, представлены на рисунке 3.1 в виде диаграммы вариантов использования [8].

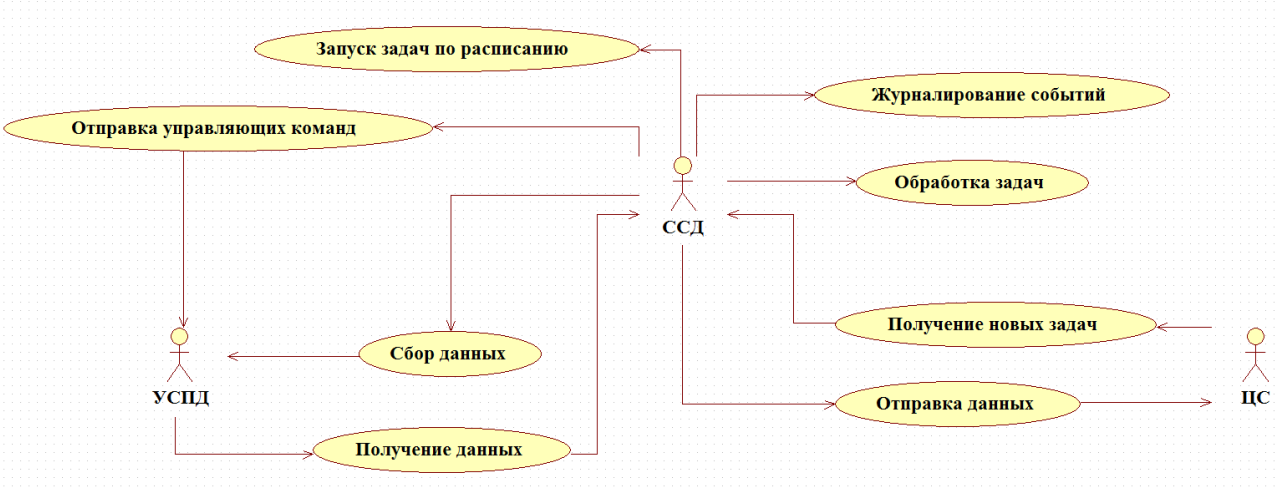


Рисунок 3.1 – Функции ПО ССД

Согласно рисунку 3.1 ПО ССД выполняет следующие функции:

- взаимодействие с ЦС (получение и передача данных);
- взаимодействие с УСПД (отправка команд и сбор данных);
- обработка задач, полученных от ЦС;
- запуск задач согласно расписанию;
- журналирование событий.

3.2 Проектирование функциональной структуры ПО ССД

На основании анализа требований к ПО ССД была спроектирована схема функционирования ССД (рисунок 3.2).

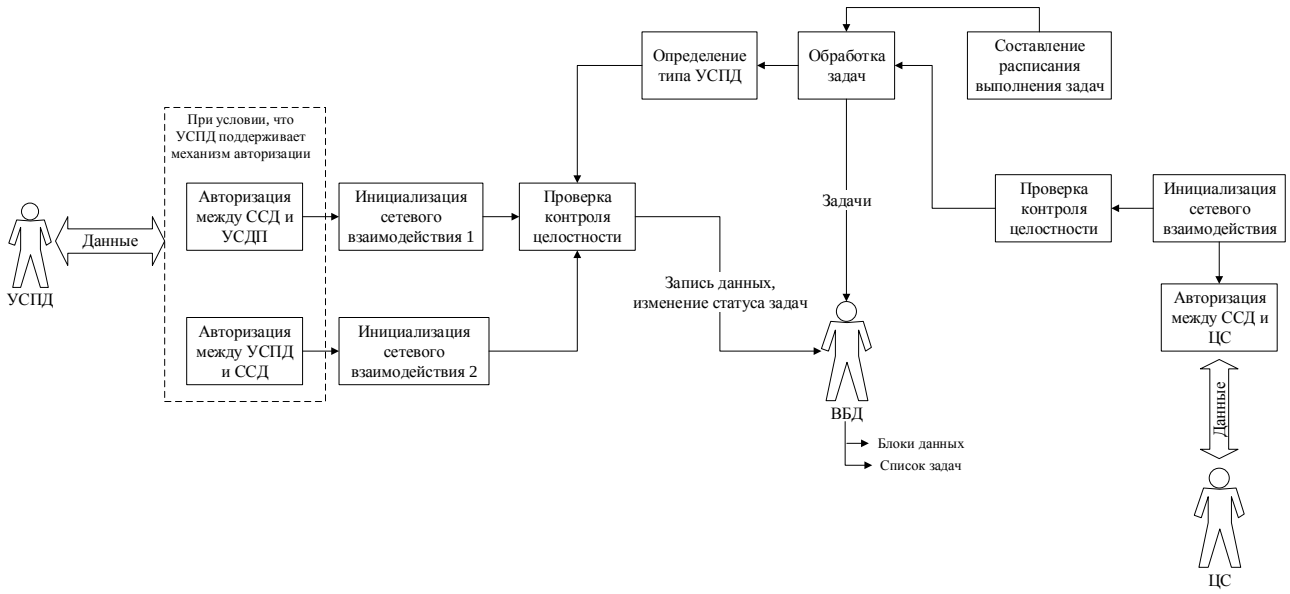


Рисунок 3.2 – Схема функционирования ССД

Согласно рисунку 3.2, ЦС инициирует сетевое соединение, проходит процедуру авторизации и отправляет на ССД пакет, содержащий в себе список задач, которые необходимо выполнить. Полученный пакет проходит контроль целостности и поступает в блок обработки задач.

Далее происходит обработка полученного пакета и осуществляется запись его содержимого во внутреннюю базу данных ССД.

После того, как пакет задач обработан, ССД записывает всю информацию о них в ВБД, составляет расписание выполнения для каждой задачи в соответствии с ее приоритетом и параметрами, а также отслеживает время до начала ее выполнения. Как только наступило время выполнения задачи, ССД определяет тип УСПД, загружает все необходимые для взаимодействия с ним драйвера и библиотеки и запускает задачу.

По умолчанию, в качестве инициатора опроса выступает ССД, но инициация сетевого подключения является настраиваемой опцией (то есть, в качестве инициатора опроса можно установить УСПД или позволить ему инициировать подключение в экстренных ситуациях). Для того, чтобы позволить УСПД инициировать сетевое подключение, ПО ССД предусматривает возможность инициализации еще одного сетевого соединения (рисунок 3.2). Данный блок прослушивает порты устройств, которые могут обратиться к ССД и начать пересылку данных.

Также ССД предусматривает механизм авторизации между ССД и УСПД при создании нового подключения, но так как не все модели УСПД поддерживают механизм авторизации, эта опция является настраиваемой. При инициализации подключения УСПД проходит процедуру авторизации и пересылает данные на ССД. в свою очередь ССД осуществляет проверку целостности полученных данных, после чего записывает их в ВБД.

Согласно требованиям к АСКУЭ, УСПД собирает данные с УУЭ как в автономном режиме, так и по запросу ССД. Данные, полученные от УУЭ, УСПД хранит в своей внутренней базе данных и по запросу пересылает их на ССД.

3.3 Организация хранения данных на ССД

Как было описано выше, ССД должен обеспечивать хранение информации, которую можно условно разбить на несколько категорий:

- информация о настройках и конфигурации ПО ССД;
- информация о событиях ПО ССД;
- информация о задачах;
- данные о подключенных УСПД;
- данные, полученные от УСПД.

Организация хранения настроек и конфигурационной информации

Вся информация о настройках и конфигурации ССД хранится в конфигурационном файле config.cfg.

Файл поделен на секции:

секция authorization хранит настройки подключения к ССД;

секция task_param хранит необходимые настройки для работы с задачами (например, предельное число обрабатываемых заданий).

Проектирование временной базы данных

Анализируя перечень информации, подлежащей хранению на ССД, было принято решение о том, что данные о задачах, подключенных УСПД и полученных показаниях необходимо хранить во временной базе данных (ВБД).

Структура разработанной ВБД изображена на рисунке 3.3 и включает в себя следующие сущности:

- device. Сущность хранит идентификатор УСПД;
- task. Сущность хранит информацию о задачах, которые необходимо выполнить;
- target. Сущность связывает сущность device с сущностью task;
- net_address_type. Сущность хранит типы сетевых адресов устройств;
- net_address. Сущность хранит сетевые адреса устройств;
- income_message. Сущность хранит в себе информацию о входящих сообщениях;
- outcome_message. Сущность хранит в себе информацию об исходящих сообщениях.

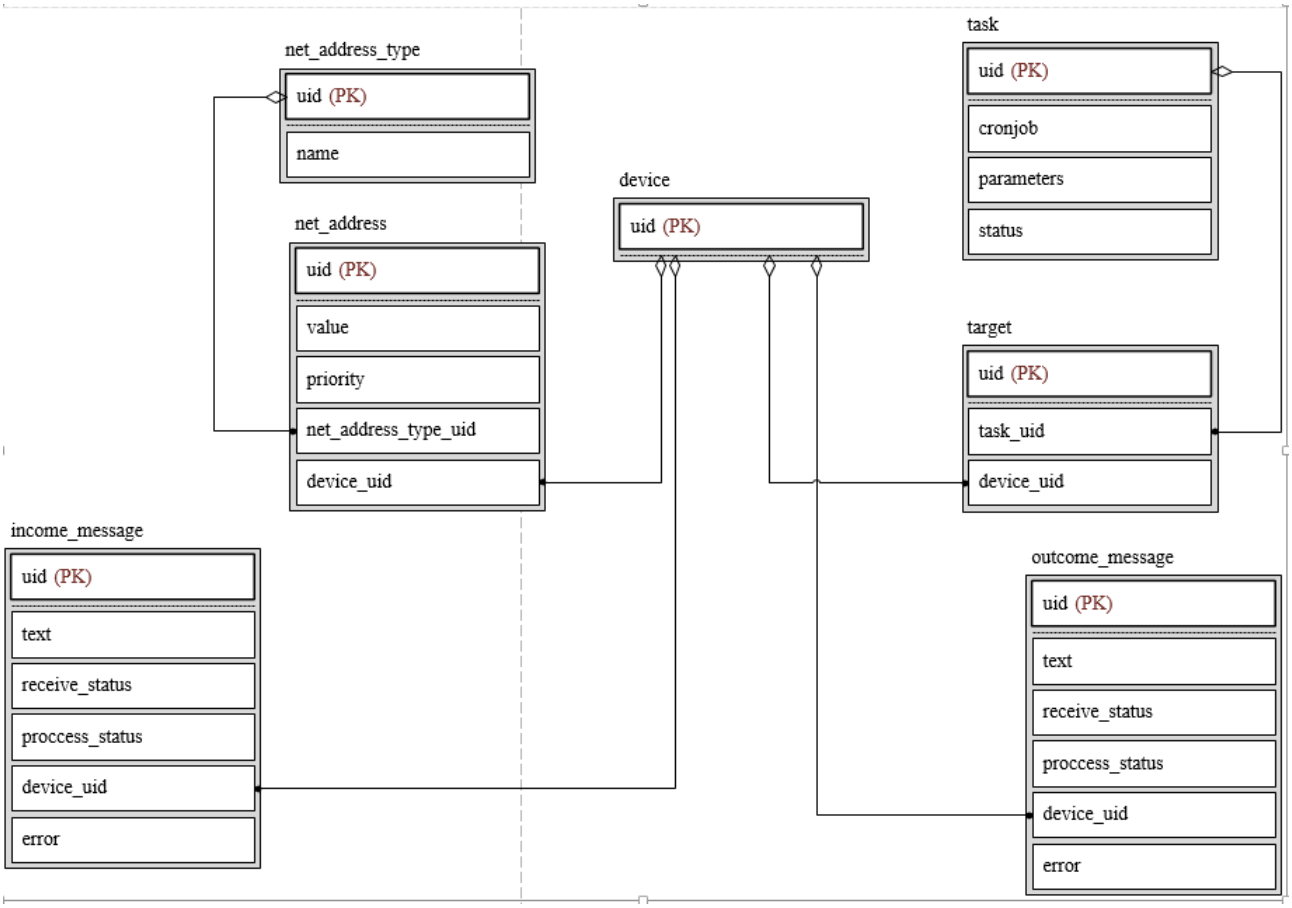


Рисунок 3.3 – Структура временной базы данных

Более подробная информация о структуре ВБД приведена в приложении А.

Проектирование системы журналирования событий ПО ССД

Все события, происходящие в процессе функционирования ПО ССД записываются в специальный журнал событий. Журнал событий представляет собой файл с именем ssd.log. Формат записи события в журнал имеет следующий вид:

[дд.мм.гг чч:мм:сс]: Текст записи

Вывод всех сообщений дублируется в консоль приложения.

3.4 Проектирование программной архитектуры приложения

В процессе анализа функциональной схемы ПО ССД было принято решение о разбиении приложения на пять блоков:

- блок обработки задач;
- планировщик задач;
- блок взаимодействия с ВБД;
- блок сетевого взаимодействия;
- управляющий блок.

Описание назначения и программной структуры каждого из блоков описано ниже.

Взаимодействие блоков ПО ССД можно изобразить в виде схемы, представленной на рисунке 3.4.

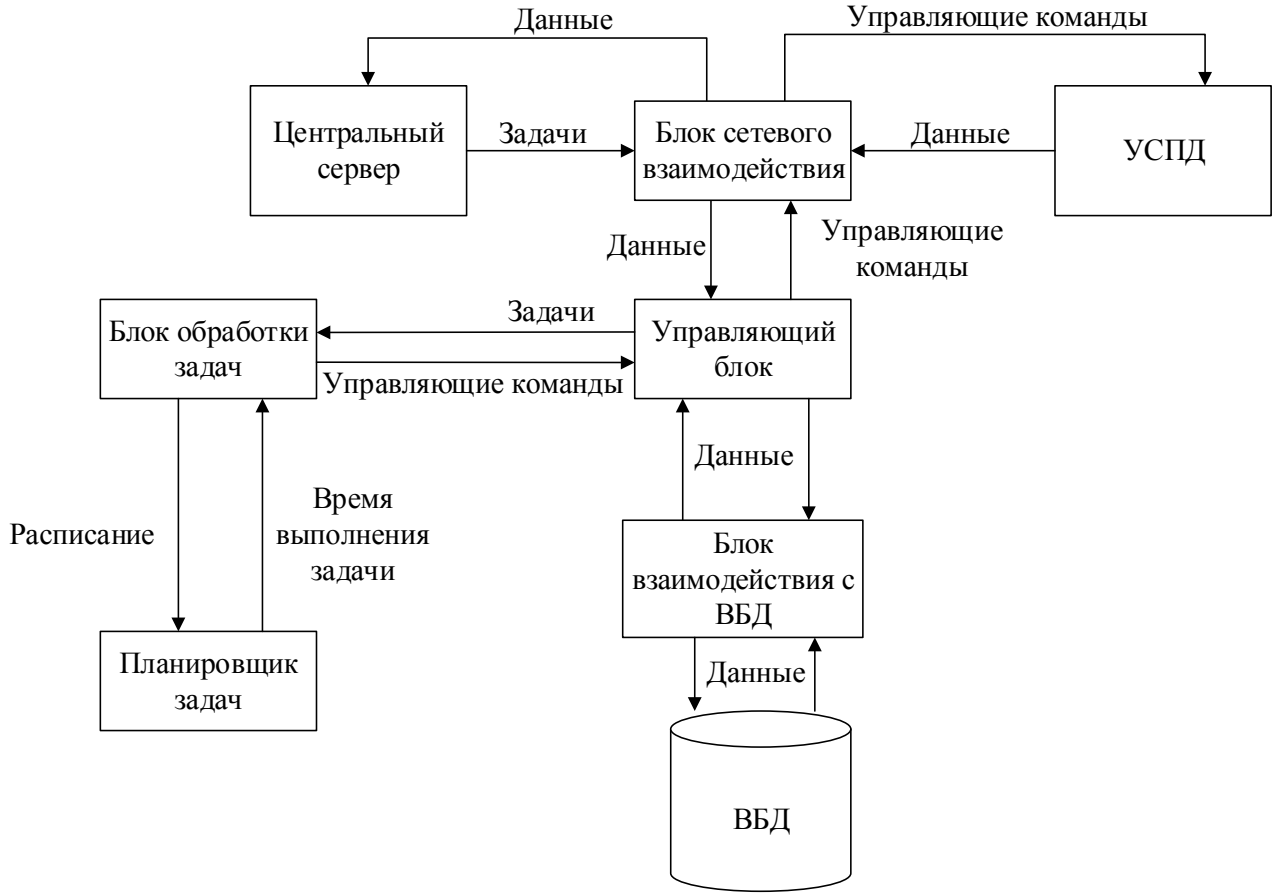


Рисунок 3.4 – Взаимодействие блоков ПО ССД

Назначение блока сетевого взаимодействия заключается в создании сетевых подключений между ССД и другими компонентами системы (ЦС и УСПД), а также в обмене данными между ССД и этими компонентами.

Блок взаимодействия с ВБД позволяет записывать и извлекать необходимые данные из ВБД.

Блок обработки задач обрабатывает задачи, поступающие от ЦС и формирует управляющие команды, отправляемые другим компонентам системы посредством блока сетевого взаимодействия.

Планировщик задач предназначен для отслеживания времени выполнения задач в соответствии с расписанием, полученным от ЦС.

Управляющий блок объединяет все блоки приложения и осуществляет над ними контроль.

3.4.1 Блок обработки задач

Основное назначение блока обработки задач заключается в управлении задачами, получаемыми от ЦС. Функции, выполняемые блоком обработки задач, представлены на рисунке 3.5 и включают в себя:

- получение пакета с задачами;
- обработку пакета с задачами;
- составление списка задач;
- управление списком задач;
- управление таблицей задач в ВБД;
- передача задач в планировщик;
- получение сигнала о вызове задачи из планировщика;
- отправка задачи на выполнение в управляющий блок.

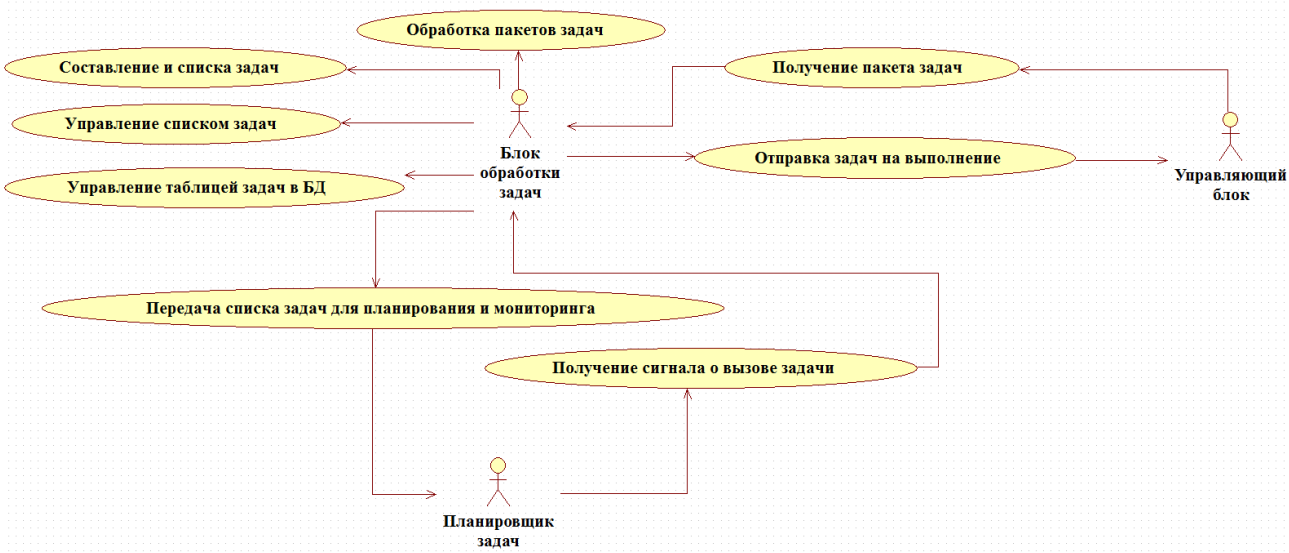


Рисунок 3.5 – Функции блока обработки задач

Анализируя функции блока обработки задач, было принято решение представить данный блок в виде трех классов:

- класс Task;
- класс TaskManager;
- класс TaskModel.

На рисунке 3.6 представлена диаграмма классов [8], описывающая взаимодействие классов блока обработки задач.

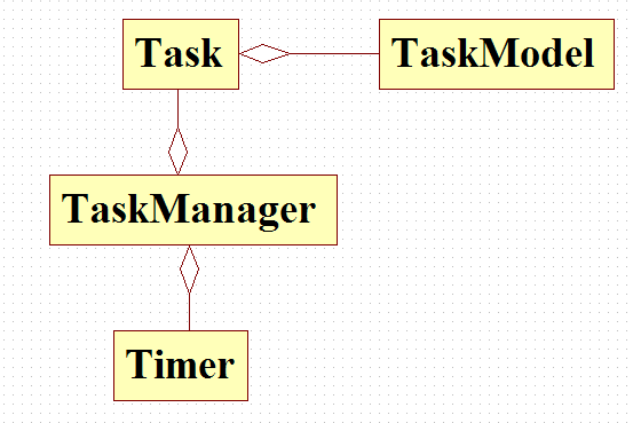


Рисунок 3.6 – Диаграмма классов блока обработки задач

Класс Task представляет собой описание задачи и включает в себя набор методов по управлению задачей. Более детальное описание класса приведено в листинге, представленном в приложении Б.1.

Класс TaskManager предназначен для формирования и управления списком задач. Детальное описание класса приведено в листинге, представленном в приложении Б.2.

Класс TaskModel представляет собой объект для работы с записями таблицы task, являющейся частью ВБД. Более подробное описание данного класса будет дано в пункте по описанию блока взаимодействия с ВБД.

3.4.2 Планировщик задач

В процессе проектирования сервера сбора данных возникла необходимость в разработке модуля, который мог бы в конкретный момент времени осуществлять вызов конкретной процедуры или функции (например, запускать процедуру опроса для УСПД каждые полчаса). Решить данную задачу позволит разработка планировщика задач.

Входные данные модуля планирования представлены в виде пары значений:

– первый элемент пары представляет собой строку, содержащую в себе cron-выражение задачи;

– второй элемент содержит в себе индекс обрабатываемой задачи.

В качестве выходных данных планировщик высылает сигнал в блок обработки задач и передает индекс задачи, которую необходимо выполнить. В

свою очередь блок обработки задач соединяет полученный сигнал с соответствующим слотом, запускающим выполнение задачи [9, 10].

Понятие cron-выражения

Изначально под cron (Command Run On) подразумевался демон-планировщик в UNIX-подобных системах, использующийся для периодического выполнения заданий в определенное время. Здесь регулярные действия описываются инструкциями, помещенными в файл crontab или в специальные директории [11].

В настоящее время cron-формат является мощным и гибким способом описания времени и периодичности действий.

Cron-выражение - это строка, состоящая из 6 или 7 полей, отделенных между собой пробелами, где:

- Seconds, Minutes, Day of month и др. – это хронологический параметр;
- YES – указание обязательности использования данного фрагмента cron-выражения;
- 0-59 – значение, допустимый интервал для заданного элемента cron-выражения;
- (* /- ,) – набор разрешенных спецсимволов.

Поля могут содержать любые из допустимых значений, а также различные комбинации специальных символов.

Формат cron-выражения выглядит следующим образом:

Seconds YES 0-59 , - * /

Minutes YES 0-59 , - * /

Hours YES 0-23 , - * /

Day of month YES 1-31 , - * ? / L W

Month YES 1-12 or JAN-DEC , - * /

Day of week YES 1-7 or SUN-SAT , - * ? / L #

Year NO empty, 1970-2099 , - * /

В cron-выражениях используется перечень специальных символов, представленный в таблице 3.1 [12].

Таблица 3.1 – Перечень специальных символов cron-формата

Символ	Значение символа
*	все значения в пределах одного фрагмента cron-выражения. Например, в поле "Minutes" специальный символ "*" обозначает, что задача будет выполняться каждую минуту
?	неопределенное значение. Если Вам необходимо, чтобы задача запускалась, к примеру 10 числа каждого месяца, но не важно, в какой день недели, то в поле "Day of month" необходимо установить значение "10", а в поле "Day of week" - "?"
-	определение диапазонов. Например, "10-12" в поле "Hours" означает часы 10, 11 и 12
,	указание дополнительных значений. К примеру, значения "ПОНЕДЕЛЬНИК, СРЕДА, ПЯТНИЦА" в поле "Day of week" будут использоваться в cron-выражении в виде запуска задачи в понедельник, среду и пятницу
/	приращение значений. Например, "5/15" в области секунд означает "секунды 5, 20, 35, и 50, то есть значение "5" будет увеличиваться на "15"
L	определение последнего дня. В поле "Day of month", к примеру, значение "L" подразумевает, что задача будет выполняться в последний день каждого месяца. В поле "Month" значение "6L" обозначает, что задача была запущена в прошлую пятницу месяца
W	определение буднего дня (понедельник-пятница). Например, в поле "Day of month" значение "15 W" будет обозначать самый близкий будний день к 15-ому из месяца, то есть если 15-м числом является суббота, задача будет запущена в пятницу 14-го. Если 15-ым числом будет воскресенье, то задача будет запущена в понедельник 16-го

В рамках разработки ПО ССД cron-выражение включает в себя 6 полей (MINUTES, HOURS, DAYS_OF_MONTH, MONTHS, DAYS_OF_WEEK, YEAR) и набор специальных символов будет включать в себя только символы: «*», «/», «-», «,».

Пример Cron-выражения, поступающего на вход планировщика имеет следующий вид: «*/30 10-18 * * * *». Данная запись означает, что действие будет выполняться каждые 30 минут с 10 до 18 часов каждый день.

Проектирование планировщика

Функции, реализуемые планировщиком задач, представлены на рисунке 3.7 и включают в себя:

- получение cron-выражения и индекса задачи от блока обработки задач;

– обработка cron-выражения;

– расчет времени запуска задачи;

– мониторинг времени запуска для каждой задачи по каждому cron-выражению;

– отправка сигнала о вызове конкретной задачи в назначенное время.

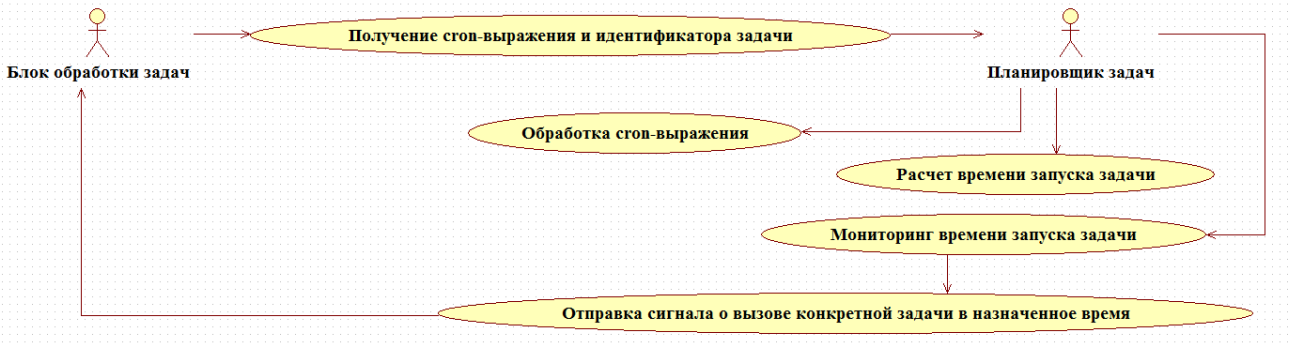


Рисунок 3.7 – Функции, выполняемые планировщиком

Исходя из анализа функций планировщика задач, было решено представить модуль планирования в виде четырех классов, каждый из которых выполняет свой круг задач:

– класс CronParser;

– класс Timer;

– класс TaskManager.

В результате проектирования структуры и интерфейса программы была составлена диаграмма классов, представленная на рисунке 3.8.

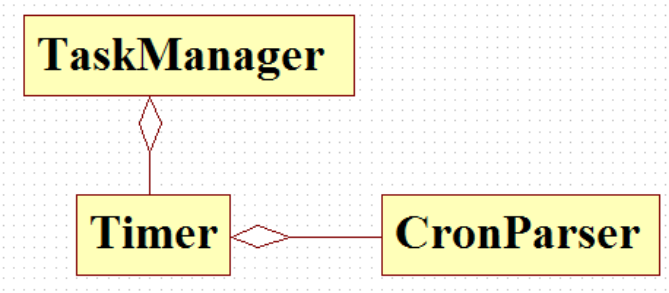


Рисунок 3.8 – Диаграмма классов планировщика задач

Класс CronParser выполняет в программе две основные функции:

– обрабатывает cron-выражение;

– рассчитывает дату ближайшего вызова функции по cron-выражению.

Детальное описание класса CronParser представлено в виде листинга в приложении Б.3.

Класс Timer отслеживает, сколько времени осталось до выполнения функции и в назначенное время отправляет соответствующий сигнал об этом [13]. Детальное описание класса Timer представлено в виде листинга в приложении Б.4**.**

Класс TaskManager для планировщика является управляющим классом и координирует его работу. Описание атрибутов и методов класса представлены в листинге (приложение Б.2).

3.4.3 Блок взаимодействия с ВБД

Большая часть данных, которые использует разрабатываемое приложение, хранится в ВБД. Постоянное обращение к ВБД делает код громоздким, нефункциональным и нечитаемым. В следствие этого было решено вынести все функции по взаимодействию с БД в отдельный блок, спроектированный по технологии ORM (с английского Object-Relational Mapping – «объектно-реляционное отображение»). Суть данной технологии заключается в том, что она связывает базы данных с концепциями объектно-ориентированного программирования, что позволяет записывать созданные в программе объекты в БД или считывать их из нее [14].

Основные функции, которые должен выполнять блок взаимодействия с ВБД представлены в виде схемы. Изображенной на рисунке 3.9 и включают в себя:

- выполнение SQL-запросов (на выборку, добавление и удаление данных);
- получение данных из ВБД по запросу;
- передача полученных данных в блок обработки задач.

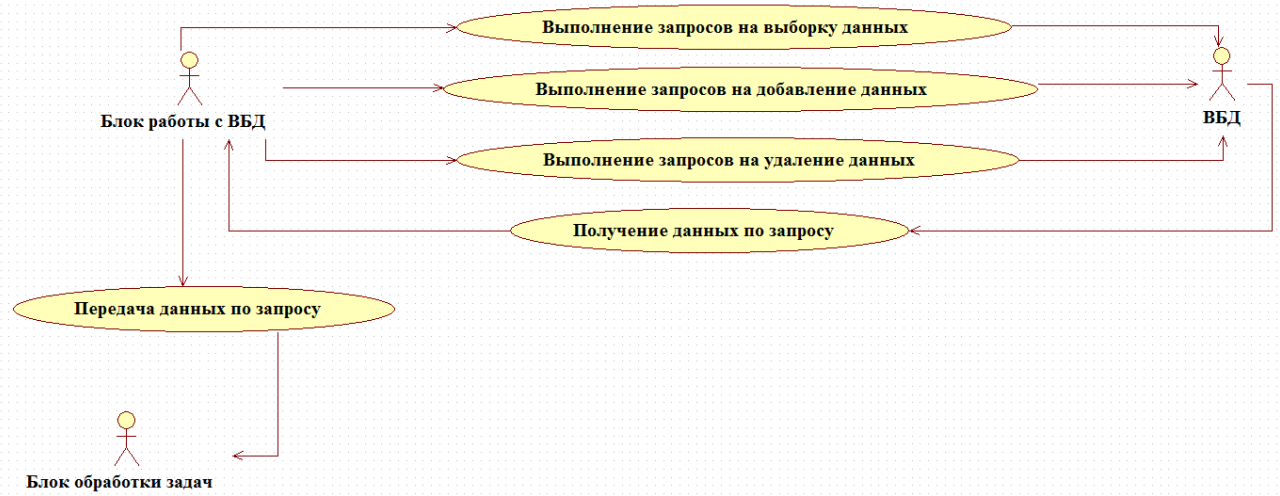


Рисунок 3.9 – Функции блока взаимодействия с ВБД

Существует несколько подходов к созданию блока объектно-реляционного отображения [15]:

– подход Table Gateway;

– подход Row Gateway;

– подход Active Record;

– подход Data Mapper.

При использовании подхода Table Gateway объект выполняет роль шлюза к таблице БД. Один экземпляр объекта обрабатывает все строки в таблице. Схематично данный подход можно изобразить в виде рисунка 3.10.

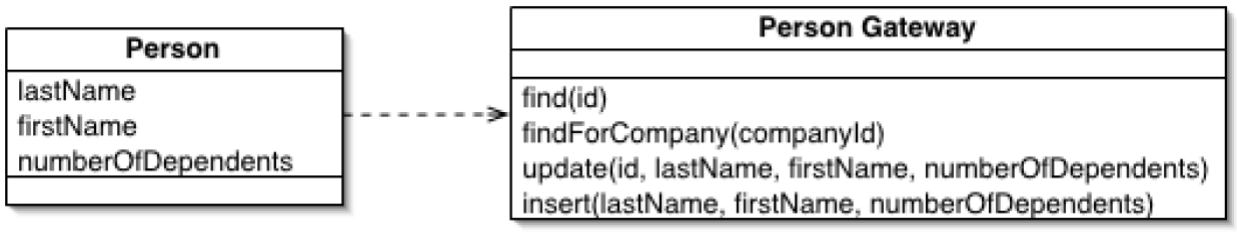


Рисунок 3.10 – Принцип функционирования паттерна Table Gateway

Объект, реализованный по принципу Table Gateway содержит все SQL запросы, необходимые для доступа к одной таблице из БД. Код, работающий с данным объектом, вызывает соответствующие методы для взаимодействия с БД.

К недостаткам метода можно отнести следующее:

– нет разделения логики приложения и логики хранения данных;

– от программиста требуется построение корректных оптимизированных SQL-запросов.

При использовании подхода Raw Gateway объект является шлюзом к одной строке в таблице БД. Для каждой строки БД создается один экземпляр объекта. Схематично работа подхода изображена на рисунке 3.11

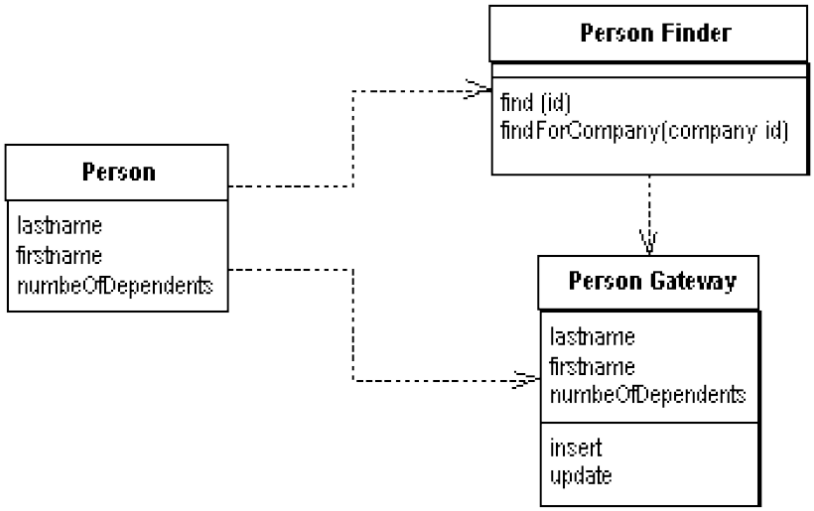


Рисунок 3.11 – Принцип функционирования подхода Row Gateway

Паттерн предоставляет программисту объекты, имеющие ту же структуру, что и строки БД. Все взаимодействие с соответствующей таблицей в БД скрыто за интерфейсом паттерна.

Недостатки метода:

– при больших размерах БД происходит замедление работы приложения, так как происходит постоянное обращение к БД;

– если объекты имеют свою бизнес-логику, увеличивается сложность разработки паттерна.

При использовании паттерна Active Record объект представляет собой обертку для строки БД и добавляет бизнес-логику к этим данным. Принцип функционирования подхода представлен на рисунке 3.12.

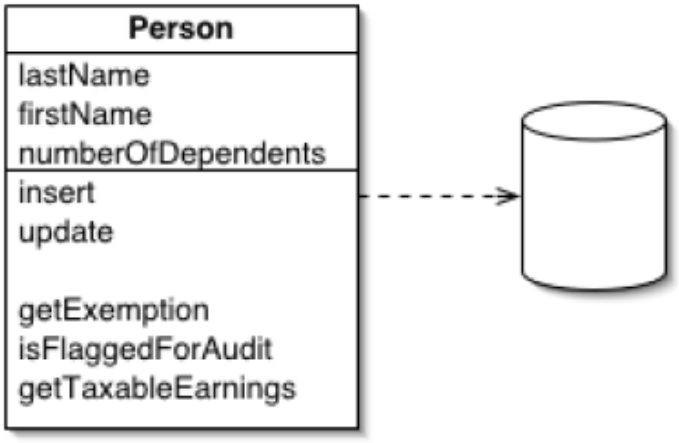


Рисунок 3.12 – Принцип функционирования паттерна Active Record

Объект Active Record содержит в себе как данные, так и бизнес-логику по работе с БД.

К недостаткам метода можно отнести замедление работы приложения при больших размерах БД.

Паттерн Data Mapper представляет собой объект, который перемещает данные между объектной моделью и БД, сохраняя их независимыми друг от друга. Принцип функционирования представлен в виде схемы, изображенной на рисунке 3.13.

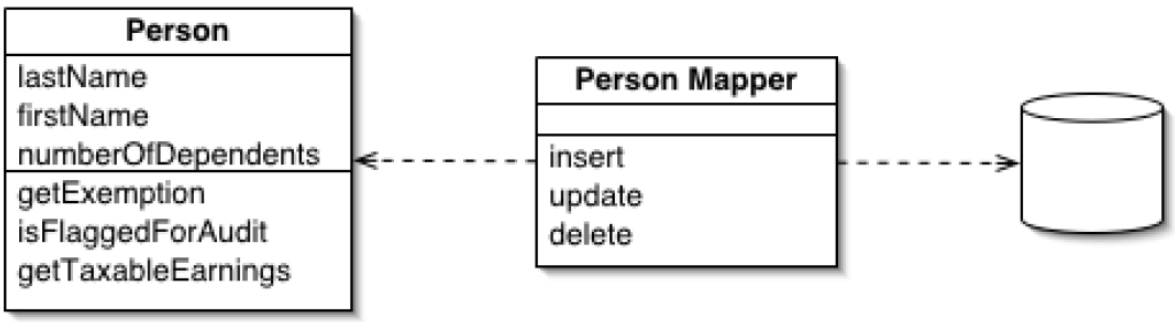


Рисунок 3.13 – Принцип функционирования паттерна Data Mapper

Недостатком метода является сложность его разработки.

При проектировании блока взаимодействия с ВБД был выбран подход Row Gateway. В отличии от паттерна Active Record, Row Gateway скроет за своим интерфейсом взаимодействие с БД, а сложность разработки паттерна ниже, чем у Data Mapper. К тому же, ВБД для ССД имеет небольшой размер, поэтому ощутимого замедления работы приложения не произойдет.

В ходе проектирования было решено создать блок взаимодействия с ВБД на основе следующих классов:

- класс Model;
- класс Query;
- класс TableSchema;
- класс TaskModel;
- класс TargetModel;
- класс NetAddressModel;
- NetAddressTypeModel;
- DeviceModel;
- Message;
- Income;
- Outcome.

Диаграмма классов изображена на рисунке 3.14

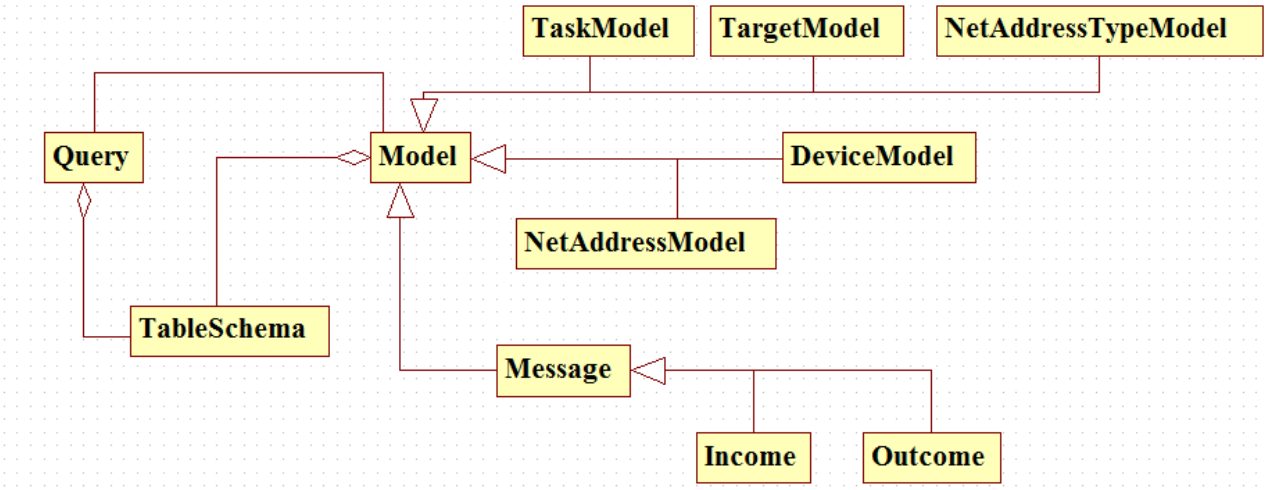


Рисунок 3.14 – Диаграмма классов блока взаимодействия с ВБД

Класс Model представляет собой абстрактную модель записи и методы по работе с ней. Данный класс содержит в себе методы по генерированию и обработке SQL-запросов на удаление, добавление и изменение записи в таблице БД. Более подробное описание класса Model отражено в листинге, представленном в приложении Б.5.

Класс Query генерирует модель записи для конкретной таблицы и позволяет осуществлять запросы на выборку разной сложности. Более подробное описание атрибутов и методов класса представлено в приложении Б.6.

Класс TableSchema представляет собой абстрактную структуру записи в таблице БД. Класс хранит в себе всю информацию о записи: наименование и количество полей, первичные и внешние ключи, имя таблицы, связи с другими таблицами и пр. Более подробное описание атрибутов и методов класса представлено в приложении Б.7.

Класс TaskModel наследуется от класса Model и представляет собой запись таблицы task.

Класс DeviceModel наследуется от класса Model и представляет собой запись таблицы device.

Класс NetAddressTypeModel наследуется от класса Model и представляет собой запись таблицы net_address_type.

Класс NetAddressModel наследуется от класса Model и представляет собой запись таблицы net_address**.**

Класс TargetModel наследуется от класса Model и представляет собой запись таблицы target**.**

Класс Message наследуется от класса Model и представляет собой абстрактную модель сообщения. Помимо атрибутов и методов класса Model класс Message так же содержит в себе атрибуты и методы по работе с сообщениями. Более подробное описание атрибутов и методов класса представлено в приложении Б.8**.**

Класс Income описывает входящее сообщение и содержит в себе методы и атрибуты по работе с ним**.**

Класс Outcome описывает исходящее сообщение и содержит в себе методы и атрибуты по работе с ним**.**

3.4.4 Блок сетевого взаимодействия

Обмен данными как с УСПД, так и с центральным сервером системы осуществляется посредством HTTP-протокола. Данный выбор обусловлен тем, что HTTP протокол позволяет программисту организовать гарантированную доставку данных передающихся по сети.

Функции, выполняемые блоком сетевого взаимодействия, можно изобразить схематично в виде рисунка 3.15.

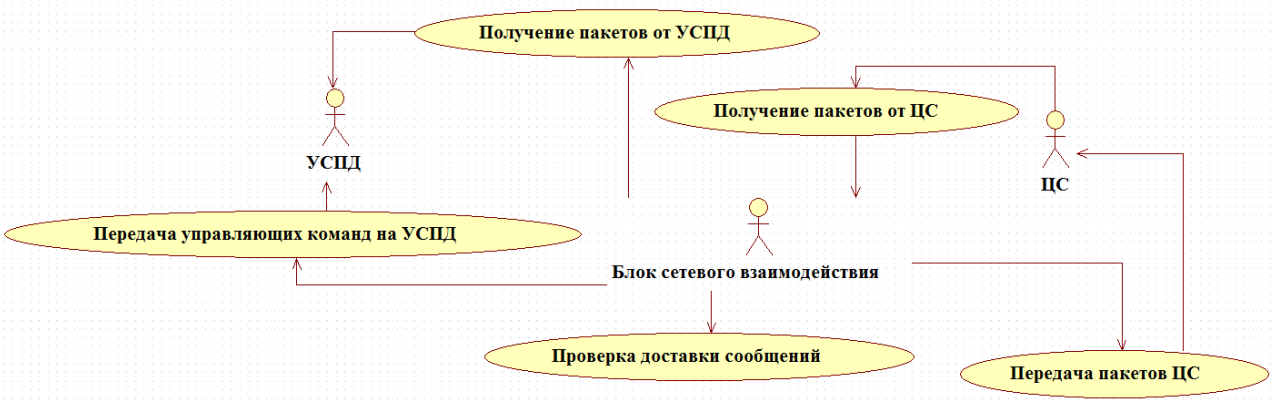


Рисунок 3.15 – Функции блока сетевого взаимодействия

Исходя из рисунка 3.15, основными функциями блока сетевого взаимодействия являются:

- получение пакетов от ЦС и УСПД;
- отправка пакетов на ЦС и УСПД;
- проверка доставки пакетов.

Анализируя функции, выполняемые блоком сетевого взаимодействия, было принято решение спроектировать его на основе следующих классов:

- класс HTTPServer;
- класс Request;
- класс Income;
- класс Outcome.

Диаграмма классов блока сетевого взаимодействия представлена на рисунке 3.16.

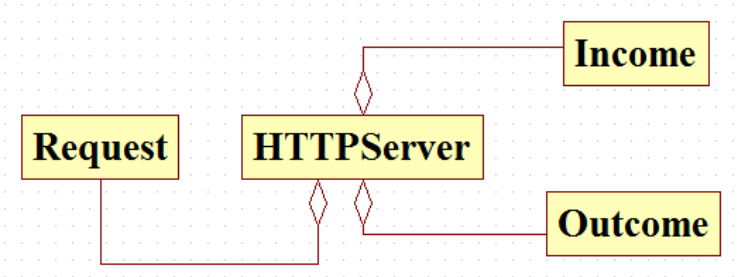


Рисунок 3.16 – Диаграмма классов блока сетевого взаимодействия

Класс Request предназначен для отправки и получения данных по HTTP-протоколу. Более подробное описание атрибутов и методов класса представлено в приложении Б.9.

Класс Income предназначен для управления входящими сообщениями, класс outcome предназначен для управления исходящими сообщениями.

Класс HTTPServer является управляющим классом блока. Данный класс слушает порт 8080. При получении запроса на соединение данный класс устанавливает сетевое соединение и осуществляет обмен данными. Более подробное описание атрибутов и методов класса представлено в приложении Б.10**.**

3.4.6 Управляющий блок

Управляющий блок включает в себя единственный класс DCServer, главным назначением которого является управление всеми блоками приложения. Более подробное описание атрибутов и методов класса представлено в приложении Б.11**.**

3.4.7 Общая структура ПО ССД

Общая структура классов приложения представлена на рисунке 3.17.

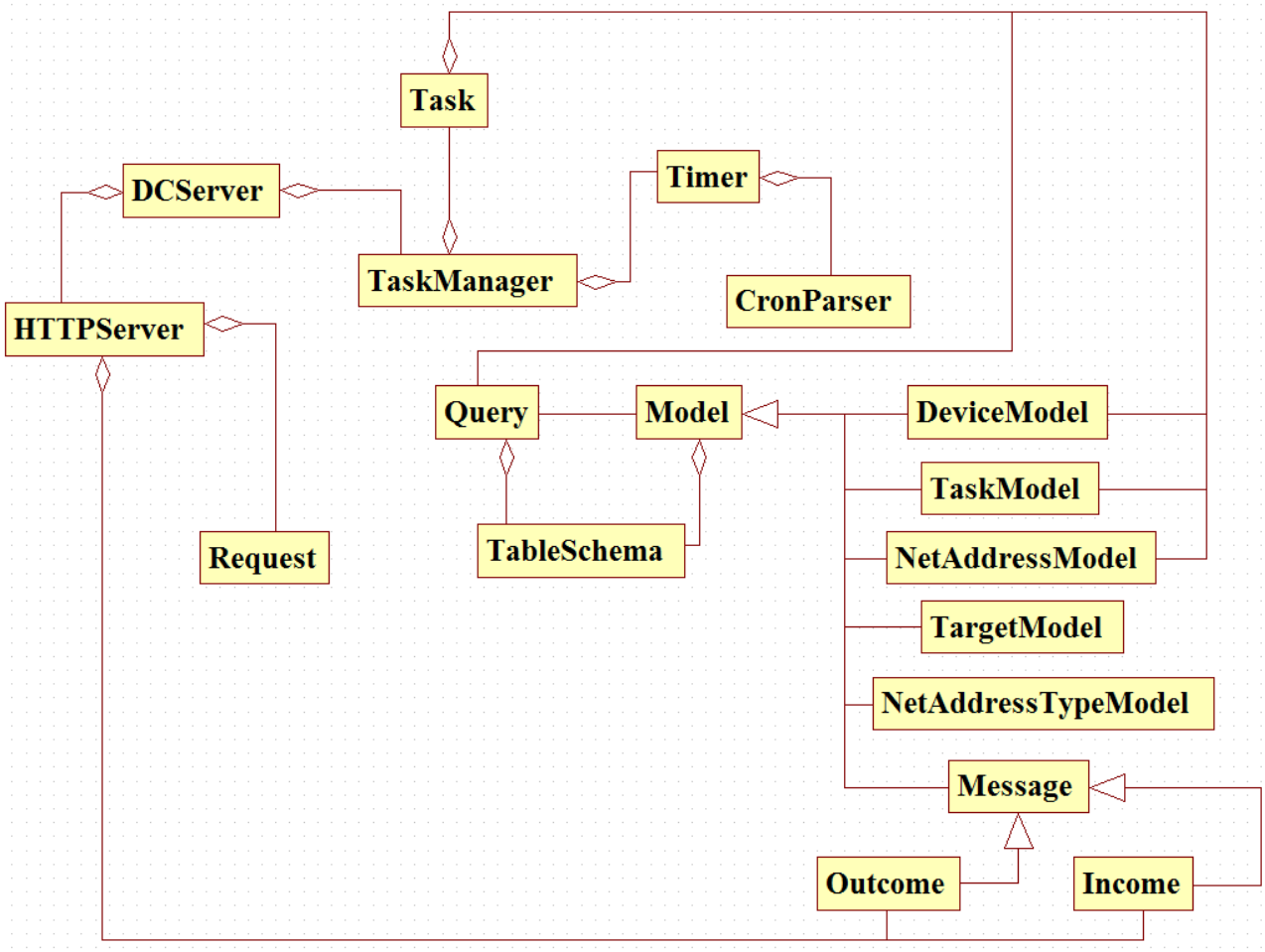


Рисунок 3.17 – Диаграмма классов всего приложения

Общая структура компонентов приложения изображена на рисунке 3.18.

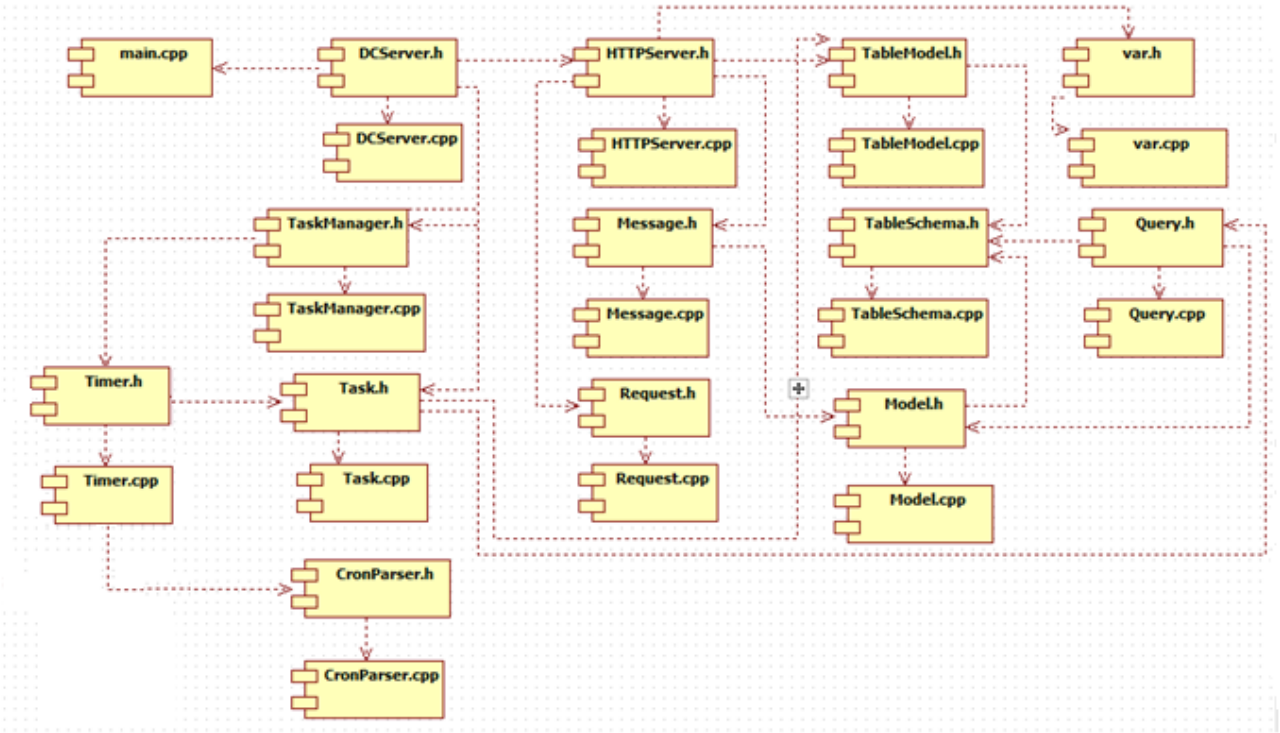


Рисунок 3.18 – Диаграмма компонентов приложения

3.5 Проектирование алгоритмов ПО ССД

3.5.1 Алгоритмы планировщика задач

Взаимодействие классов планировщика задач можно изобразить в виде диаграммы, представленной на рисунке 3.19.

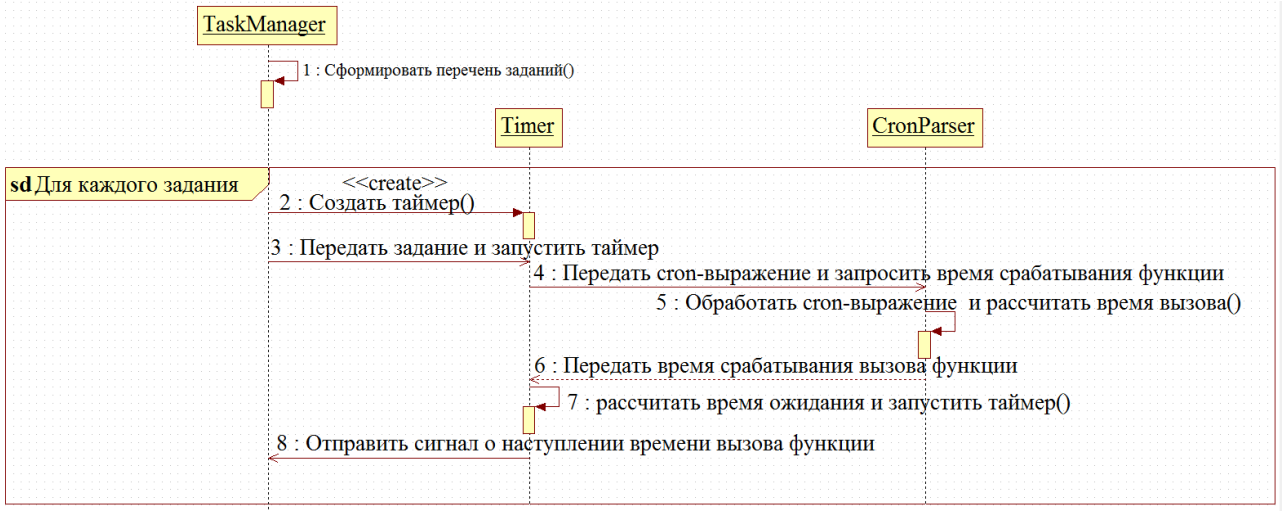


Рисунок 3.19 – Взаимодействие классов планировщика

Исходя из представленной диаграммы, объект класса TaskManager формирует перечень заданий, подлежащих вызову по запланированному времени. Далее объект класса TaskManager для каждого задания создает свой объект класса Timer и передает в него информацию о текущем задании. В свою очередь Timer передает cron-выражение для текущего задания в объект класса CronParser. CronParser обрабатывает полученное cron-выражение, вычисляет ближайшее время вызова функции и передает ее в объект Timer. Timer рассчитывает время ожидания выполнения функции и запускает таймер. По истечении времени ожидания Timer отправляет сигнал объекту TaskManager, в котором передает имя вызываемой функции.

Алгоритмы обработки cron-выражения и расчета ближайшего времени вызова

Первичная обработка cron-выражения выполняется при помощи метода calcTaskDate. В качестве входных данных метод принимает строку cron-выражения. Выходным параметром является ближайшая дата вызова функции.

Метод разделяет cron-выражение на части по знаку пробела и каждую часть помещает в массив строк crons в качестве отдельного элемента. Далее идет проверка размера массива. Если размер crons не равен 5, то cron-выражение считается некорректным и программа выдает сообщение об ошибке. Если crons содержит 5 элементов, то каждой временной единице присваивается значение в соответствии с текущим временем. Далее calcTaskDate проверяет по значению переменной isCalled, вызывался ли уже данный метод для текущего времени. Если вызывался, то значение минут вызова увеличивается на единицу. Далее для каждого массива единиц времени (month, dayOfMonth, dayOfWeek, hour, minute) вызывается метод parse. В качестве cron-выражения в параметрах метода parse указывается соответствующий элемент массива crons:

- $crons_0$ – минуты;
- $crons_1$ – часы;
- $crons_2$ – день месяца;
- $crons_3$ – месяц;
- $crons_4$ – день недели.

После того как для каждой временной единицы получено соответствующее значение, calcTaskDate формирует из них дату вызова функции. Более детальное описание алгоритма метода calcTaskDate представлено в виде блок-схемы в приложении В.1:

Входными данными метода parse являются:

- строка cron-выражения для данной временной единицы;
- текущее значение временной единицы;
- минимальный и максимальный пределы допустимых значений для текущей временной единицы.

Выходными данными метода является пара значений следующего вида:

- ближайшее значение для текущей временной единицы согласно ее cron-выражению (перечень допустимых cron-выражений представлен в таблице 3.2);

– флаг переполнения разряда. Если флаг равен true, то последующая временная единица увеличивается на единицу и поиск ближайшего значения рассчитывается уже от нового значения.

Таблица 3.2 – Перечень допустимых cron-выражений

Тип cron-выражения	Возможные принимаемые значения
*	Текущее значение временной единицы, при условии, что она попадает в интервал допустимых значений. Минимальное и максимальное допустимые значения интервала обуславливаются типом временной единицы, для которой ищется возможное значение.
*/step	Возможными значениями являются все значения, которые берутся от текущего значения временной единицы до предельно допустимого с шагом step.
start-finish	Возможными значениями являются все значения от start до finish, берущиеся с шагом 1
start-finish/step	Возможными значениями являются все значения от start до finish, берущиеся с шагом step
$value_1$, $value_2$, …, $value_n$	Возможными значениями являются все значения, отделенные друг от друга знаком «,», при условии, что они попадают в интервал допустимых
start/step	Возможными значениями являются все значения, которые берутся от значения start до предельно допустимого с шагом step
value	Возможное значение – значение value, при условии, что оно попадает в интервал допустимых значений.

Переменные value, start, finish, step, указанные в таблице являются целочисленными десятичными значениями.

Установление соответствия между входным cron-выражением и одним из допустимых выражений происходит при помощи использования регулярных выражений [16]. Регулярные выражения повышают надежность кода

программы, так как при их использовании снижается риск возникновения ошибок при обработке неверного cron-выражения. Если cron-выражение содержит ошибку, то в консоль выводится соответствующее сообщение.

Более детальное описание алгоритма метода parse представлено в виде блок-схемы в приложении В.2. Реализация алгоритмов планировщика задач в виде листингов представлена в приложениях Б.12, Б.13 и Б.14.

Алгоритмы обработки событий таймера

В процессе проектирования планировщика задач возникла необходимость в разработке собственного таймера. Это обусловлено тем, что для отслеживания вызова каждой процедуры или функции должен создаваться отдельный таймер, обладающий идентификатором (идентификатор необходим для того, чтобы у программиста была возможность управлять конкретным таймером). Помимо этого, при выполнении программы могут возникать ситуации, когда два разных таймера должны высылать сигнал в одинаковый момент времени. Это может привести к сбоям в работе программы, поэтому для каждого таймера необходимо использовать специальные блокирующие переменные, называемые мьютексами (от англ. mutex – «взаимное исключение»). Мьютекс – это переменная, служащая для защиты разделяемых данных от одновременного доступа нескольких потоков [17]. Мьютексы могут находиться в одном из двух состояний — отмеченном или неотмеченном. Когда какой-либо поток, принадлежащий любому процессу, становится владельцем объекта mutex, последний переводится в неотмеченное состояние [18]. Если задача освобождает мьютекс, его состояние становится отмеченным. Таким образом, использование мьютексов не позволит двум таймерам сработать в одно и то же время.

Обработка событий таймера осуществляется посредством метода timerEvent. В этой функции посредством методов объекта CronParser обрабатывается cron-выражение и вычисляется время ближайшего срабатывания функции. После этого, при помощи метода calcDiffTime вычисляется время ожидания до следующего срабатывания функции и

запускается таймер. По истечении установленного времени timerEvent посылает сигнал о том, что наступило время вызова функции, а также проверяет тип таймера: если объект таймера не является синглшотом, то рассчитывается время до следующего вызова функции. Также timerEvent осуществляет проверку на то, вызывался ли уже метод getDateTime для текущего времени. Если метод уже вызывался, то timerEvent устанавливает флаг isCalled класса CronParser в состояние «true» при помощи метода setCall.

Более детальное описание алгоритмов функций timerEvent и calcDiffTime представлено в виде блок-схем в приложениях В.3 и В.4 соответственно.

3.5.2 Алгоритмы блока обработки задач

Схематично алгоритм работы блока обработки задач можно изобразить в виде рисунка 3.20.

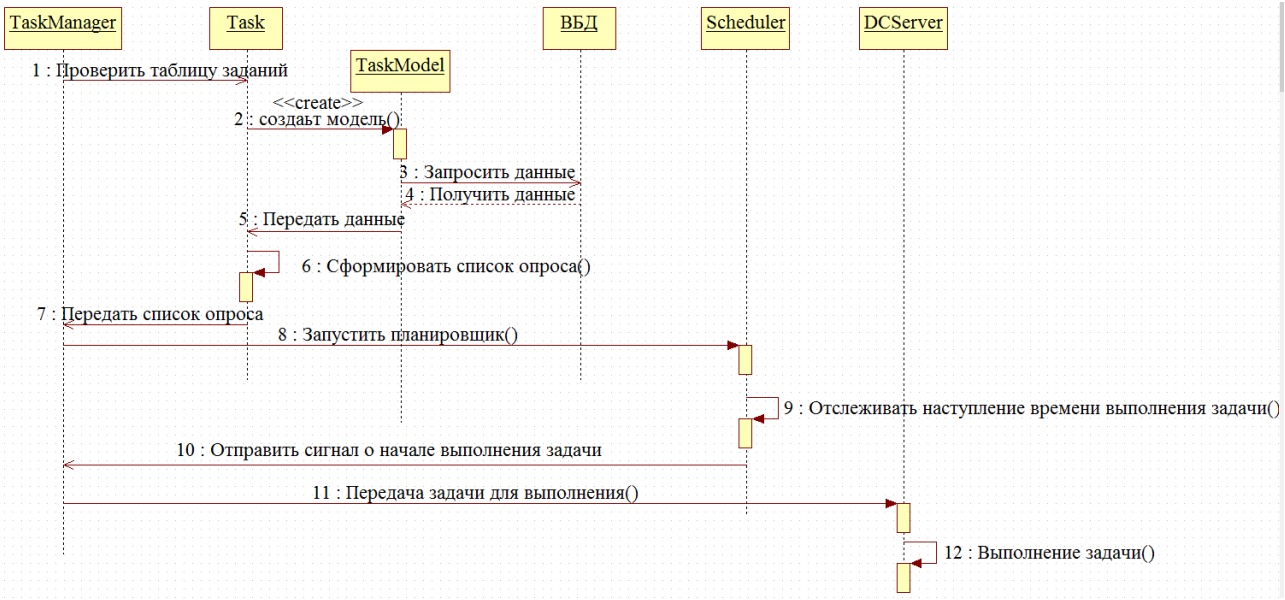


Рисунок 3.20 – Алгоритм работы блока обработки задач

При запуске блока обработки задач объект класса TaskManager посредством методов объекта класса Task проверяет ВБД на наличие невыполненных заданий. В свою очередь Task использует объект класса TaskModel для взаимодействия с таблицей task, находящейся в ВБД. TsakModel получает из ВБД необходимую информацию о задачх и передает ее в TaskManaget через Task в виде списка опроса. После получения списка опроса Task запускает объект планировщика задач (Scheduler). Планировщик

обрабатывает задачи и отслеживает время их выполнения. Как только наступило время выполнения задачи, Scheduler сообщает об этом в TaskManager. В свою очередь TaskManager передает в управляющий объект класса DCServer все данные, необходимые для выполнения задачи. При получении сигнала на выполнение задачи и необходимых для ее выполнения данных, объект DCServer запускает задачу.

Задачи, считающиеся выполненными, удаляются из ВБД.

Реализация алгоритмов блока обработчика задачи представлена в виде листингов в приложениях Б12 и Б15.

3.5.3 Алгоритмы блока сетевого взаимодействия

Передача данных, как между ССД и ЦС, так и между ССД и УСПД осуществляется посредством HTTP-протокола. Принцип сетевого взаимодействия ССД и с ЦС, и с УСПД является одинаковым, поэтому все описание сетевого взаимодействия будет осуществляться на примере взаимодействия ССД с ЦС.

Алгоритм получения данных

Процесс успешного получения данных схематично можно представить в виде диаграммы, изображенной на рисунке 3.21.

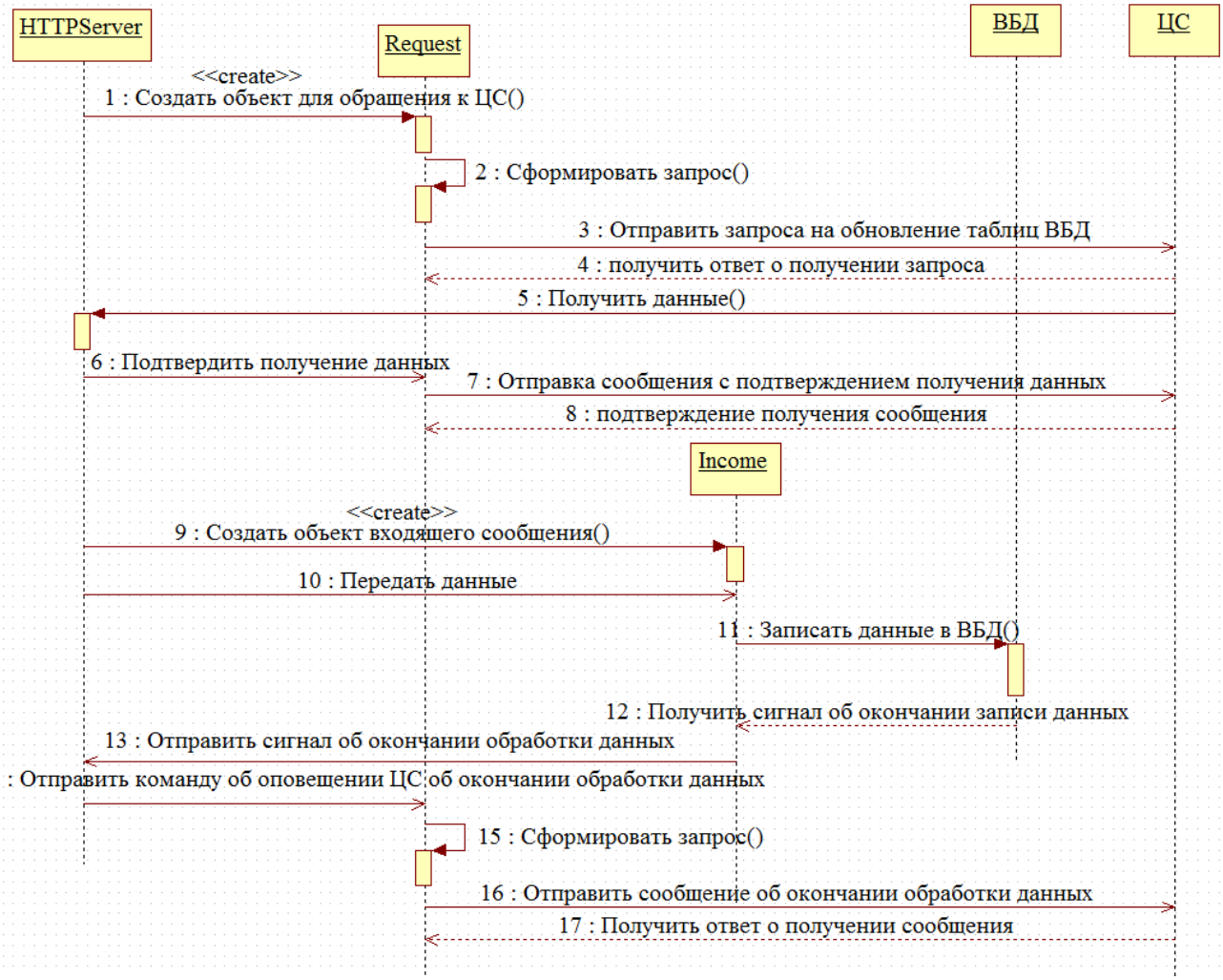


Рисунок 3.21 – Алгоритм успешного получения данных

При запуске ССД в первую очередь необходимо обновить информацию в таблицах ВБД. Для этого создается объект класса HTTPServer. HTTPServer создает объект класса Request для взаимодействия с ЦС. В свою очередь Request формирует запрос на обновление таблиц и отправляет его на ЦС. Если запрос дошел успешно, ЦС возвращает подтверждение получения сообщения. После получения данных от ЦС HTTPServer посредством объекта Request отправляет на ЦС сообщение с подтверждением получения данных и также получает от ЦС ответ о том, что сообщение получено.

Далее HTTPServer создает объект входящего сообщения Income и передает в него полученные данные. Income в свою очередь обрабатывает данные и записывает из в ВБД. По окончании записи данных в ВБД Income отправляет HTTPServer сигнал об окончании обработки данных. Далее HTTPServer посредством объекта Request отправляет на ЦС сообщение об

окончании обработки данных и ожидает ответ о получении сообщения. При получении подтверждения о получении сообщения от ЦС процесс получения данных считается завершенным.

На любом из шагов описанного алгоритма есть риск возникновения ситуаций, в которых ЦС не отвечает на отправленные сообщения. В этом случае объект класса HTTPServer повторяет отправку сообщения с интервалом в каждые пять минут до тех пор, пока ЦС не ответит.

Алгоритм отправки данных

Схематично алгоритм успешной отправки данных можно представить в виде рисунка 3.22.

Рисунок 3.22 – Алгоритм успешной отправки данных

Согласно представленному рисунку перед началом отправки данных объект класса HTTPServer создает объект класса Outcome. В свою очередь Outcome запрашивает данные, необходимые для отправки. По окончании получения данных Outcome отправляет соответствующий сигнал объекту HTTPServer. Как только исходящее сообщение сформировано, HTTPServer создает объект класса Request для отправки данных на ЦС. Request отправляет на ЦС пакет с данными и получает в ответ подтверждение о получении данных. После того, как ЦС обработает полученные данные, он отправляет HTTPServer

сообщение о том, что данные обработаны. При получении данного сообщения HTTPServer посредством объекта класса Request отправляет на ЦС сообщение о подтверждении получения сообщения об окончании обработки данных. Как только ЦС отправить ответное подтверждение получения сообщения, отправка данных считается успешно выполненной.

Реализация сетевого взаимодействия ССД с другими уомпонентами АСКУЭ представлена в виде листинга в приложениях Б.16 и Б.17.

Обработка передаваемых и получаемых данных

Все данные, передающиеся по сети представлены в текстовом формате обмена данными JSON (англ. JavaScript Object Notation). В качестве значений в данном формате могут использоваться:

- объект, представляющий собой множество пар «ключ»: «значение», заключенных в фигурные скобки. Пары отделяются друг от друга запятой;
- массив, представляющий собой упорядоченное множество значений и заключенный в квадратные скобки;
- числа;
- литералы типа true, false и null;
- строки символов [19].

Обработка всех сообщений, представленных в формате JSON осуществлена с использованием класса Var, спроектированного одним из разработчиков АСКУЭ.

3.5.4 Алгоритмы блока взаимодействия с ВБД

Блок взаимодействия с ВБД позволяет осуществлять выполнение следующих типов запросов:

- запросы на выборку данных;
- запросы на добавление данных;
- запросы на изменение данных;
- запросы на удаление данных.

Все запросы, за исключением запросов на выборку данных осуществляются по одному и тому же алгоритму. Рассмотрим данный алгоритм на примере добавления данных в ВБД. Алгоритм добавления данных в ВБД можно представить в виде диаграммы, изображенной на рисунке 3.23.

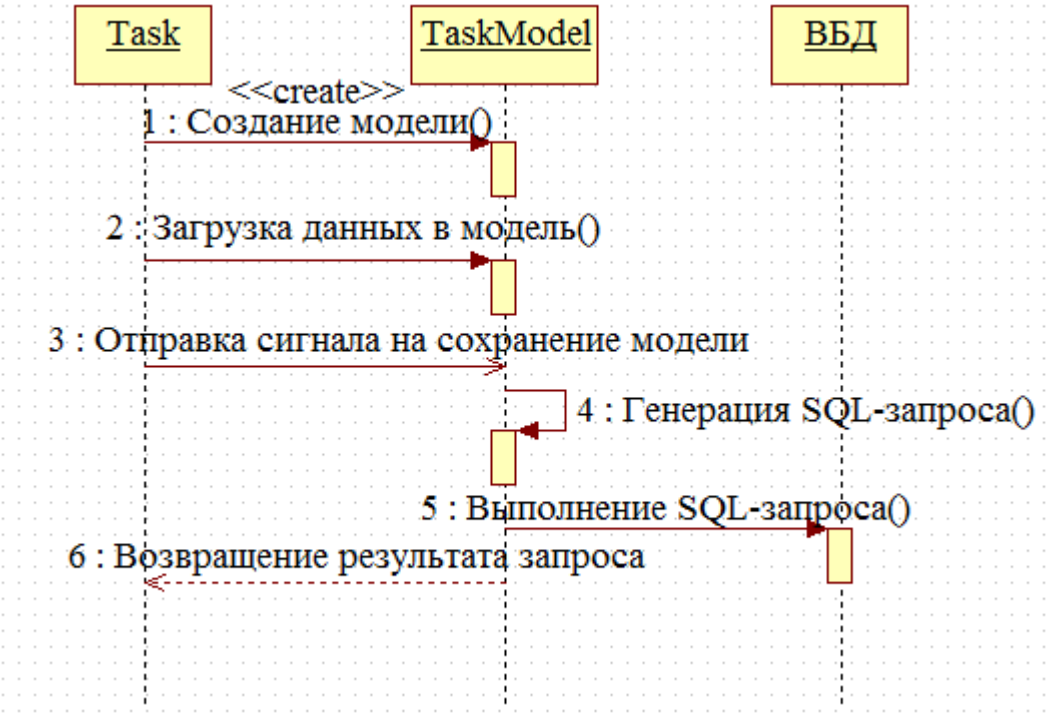


Рисунок 3.23 – Алгоритм добавления данных в ВБД

Для того, чтобы объект класса Task мог добавить данные в ВБД, необходимо создать объект класса TaskModel и загрузить в него данные. После загрузки данных, в объект TaskModel отправляется сигнал на сохранение модели в ВБД. При получении сигнала на сохранение TaskModel генерирует соответствующий SQL-запрос. Стоит отметить, что если TaskModel до этого содержал в себе данные, то будет сгенерирован запрос на обновление таблицы. Если объект класса TaskModel впервые получил набор данных, то при получении сигнала он сгенерирует запрос на добавление записей в ВБД. Результат выполнения SQL-запроса TaskModel вернет в объект Task. В качестве возвращаемого значения будет выступать либо true (при успешном выполнении запроса), либо false (при ошибке выполнения запроса). Если в результате выполнения запроса возникла ошибка, информация о ней будет отображена в консоли и в журнале событий ССД.

Алгоритм выполнения запросов на выборку данных

Все запросы на выборку данных осуществляются посредством объекта класса Query. Процесс получения данных можно представить в виде схемы, изображенной на рисунке 3.24.

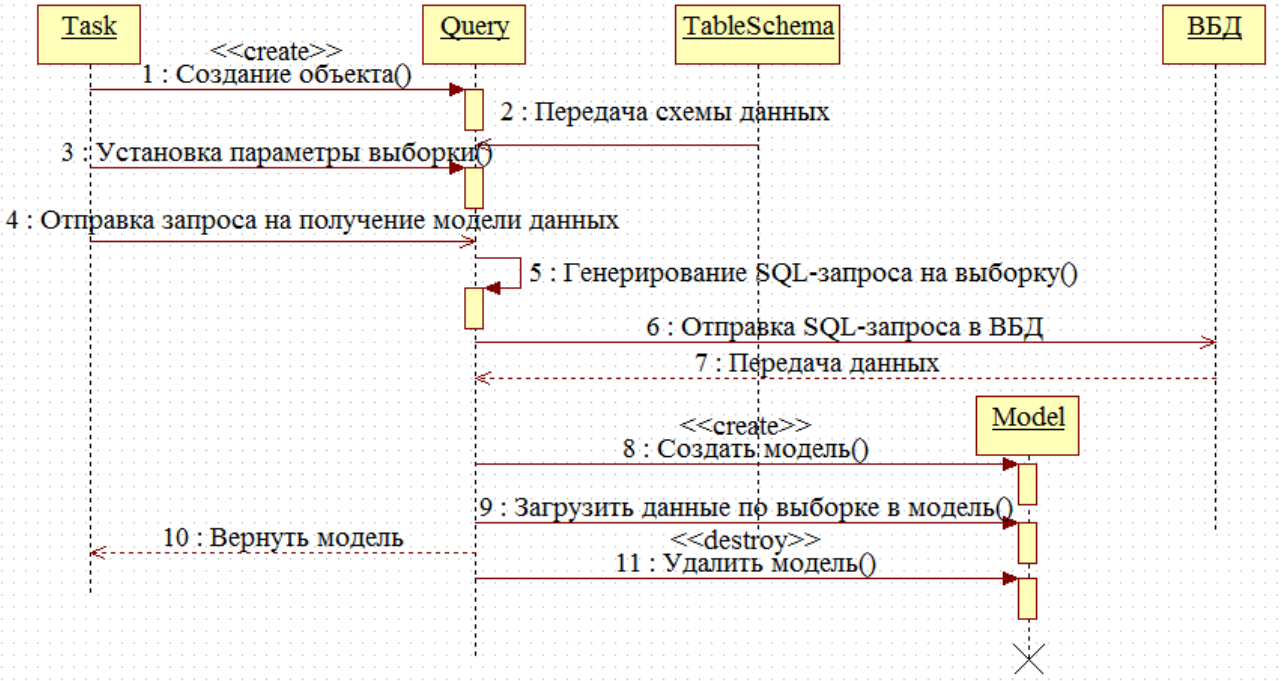


Рисунок 3.24 – Алгоритм выполнения запросов на выборку

Для получения данных по выборке необходимо создать объект класса Query и передать в него схему данных для конкретной таблицы, после чего необходимо установить параметры выборки.

К параметрам выборки относятся:

– перечень выбранных полей;

– условие совпадения поля с конкретным значением;

– лимит на количество выбранных значений;

– упорядочение выбранных значений по возрастанию или убыванию.

Также объект класса Query позволяет выполнять запросы на объединение данных типа INNER_JOIN и LEFT_OUTER_JOIN.

Установив необходимые параметры выборки, объекту Query отправляется запрос на получение модели данных по выборке. В свою очередь Query генерирует строку SQL-запроса и запрашивает по ней данные из ВБД. При

получении данных Query создает объект класса Model, загружает в него данные по выборке и возвращает полученную модель вызвавшему его объекту.

Реализация алгоритмов блока взаимодействия с ВБД отражена в листингах, представленных в приложениях Б.18 и Б.19.

4 РЕЗУЛЬТАТЫ РАЗРАБОТКИ ПО СЕРВЕРА СБОРА ДАННЫХ

В настоящем разделе описаны результаты разработки приложения.

Сервер сбора данных при запуске получает от центрального сервера данные для обновления таблиц ВБД. Процесс получения данных от центрального сервера изображен на рисунке 4.1.

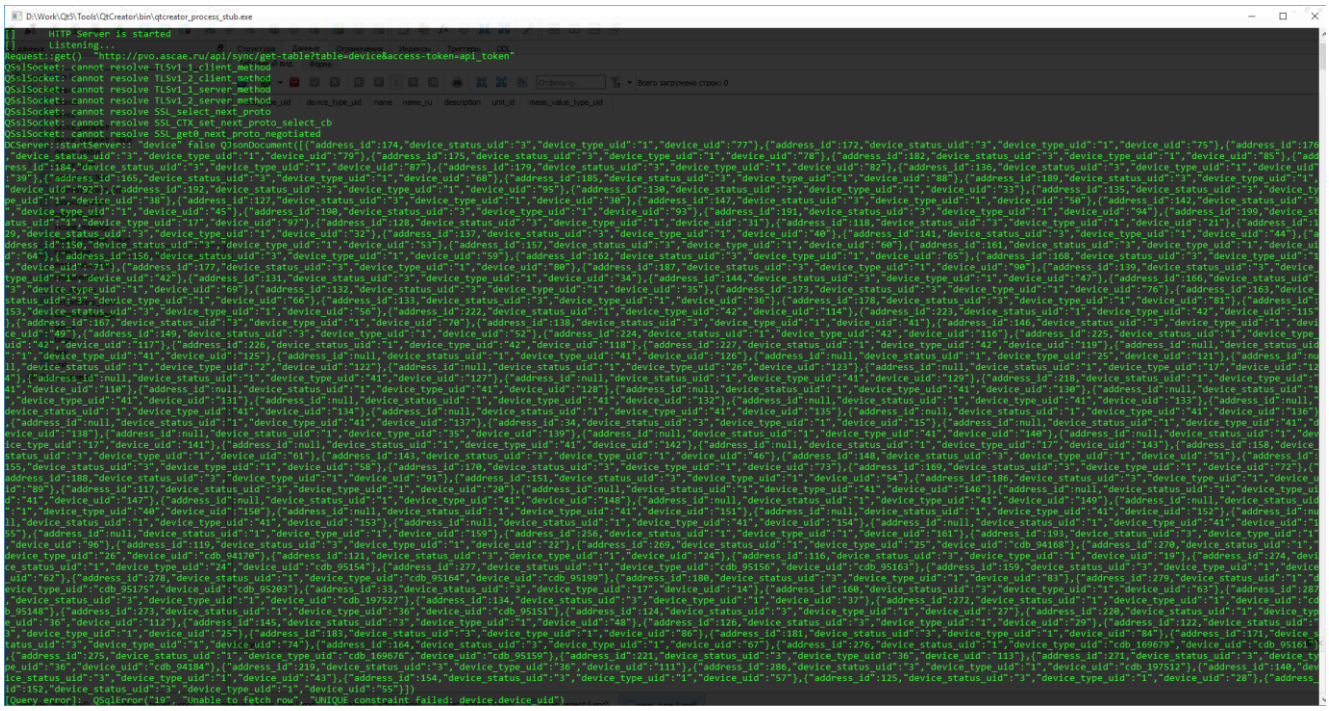


Рисунок 4.1 – процесс получения данных от ЦС

По окончании процесса получения данных ССД обрабатывает их и заполняет соответствующие таблицы временной базы данных, сопровождая процесс записи соответствующими сообщениями (рисунок 4.2).

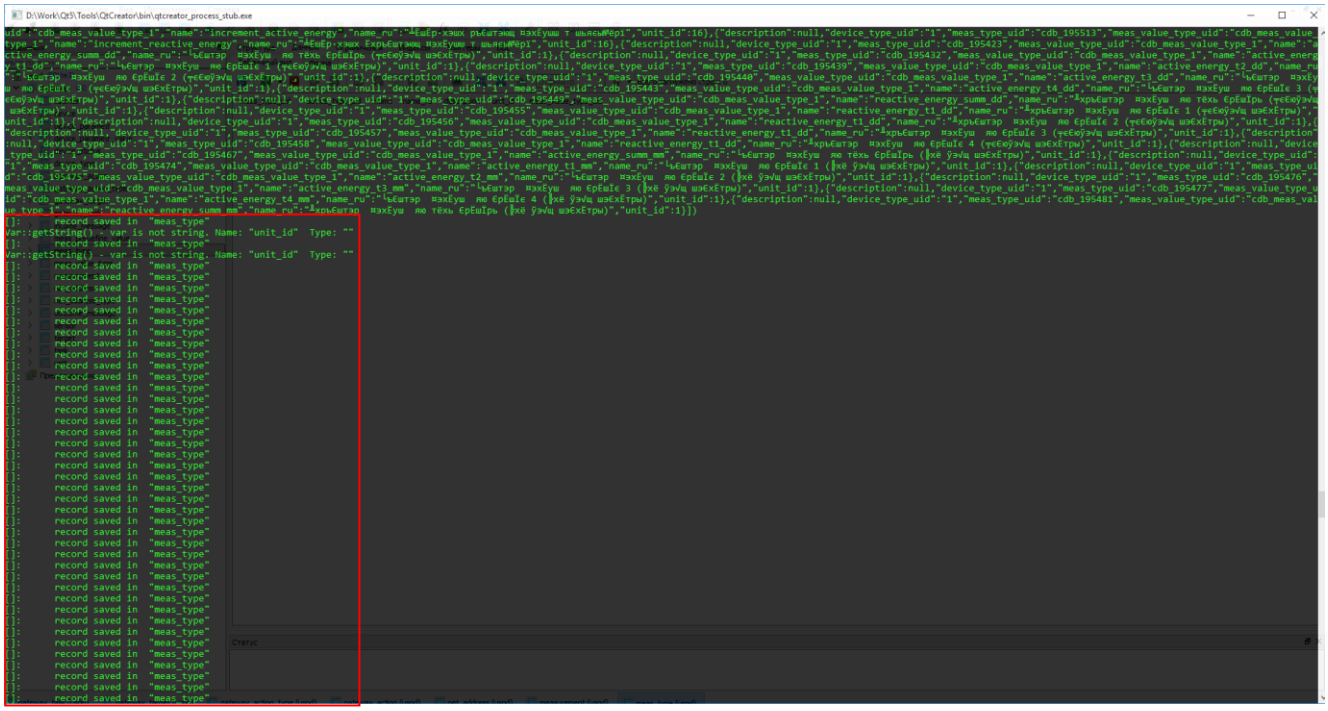
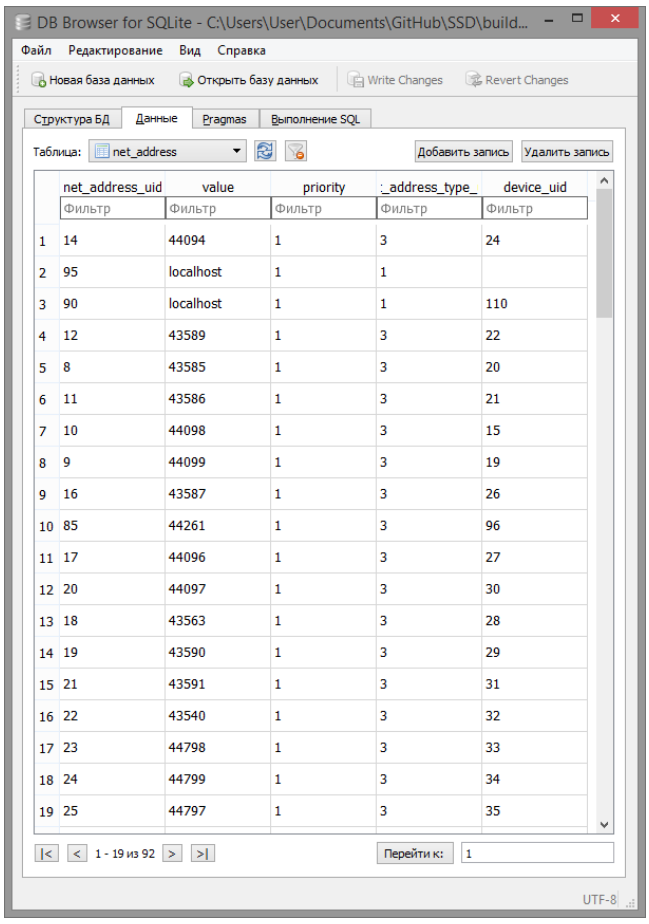


Рисунок 4.2 – Процесс записи данных в ВБД

Результат автоматического заполнения ВБД изображен на примере таблицы net_address и представлен на рисунке 4.3.



	net_address_uid	value	priority	_address_type_	device_uid
1	14	44094	1	3	24
2	95	localhost	1	1	
3	90	localhost	1	1	110
4	12	43589	1	3	22
5	8	43585	1	3	20
6	11	43586	1	3	21
7	10	44098	1	3	15
8	9	44099	1	3	19
9	16	43587	1	3	26
10	85	44261	1	3	96
11	17	44096	1	3	27
12	20	44097	1	3	30
13	18	43563	1	3	28
14	19	43590	1	3	29
15	21	43591	1	3	31
16	22	43540	1	3	32
17	23	44798	1	3	33
18	24	44799	1	3	34
19	25	44797	1	3	35

Рисунок 4.3 – результат автоматического заполнения таблицы net_address

По окончании записи данных в ВБД запускается планировщик. Как только наступило время срабатывания, ССД осуществляет подключение к соответствующему УСПД и начинает опрос (рисунок 4.4).

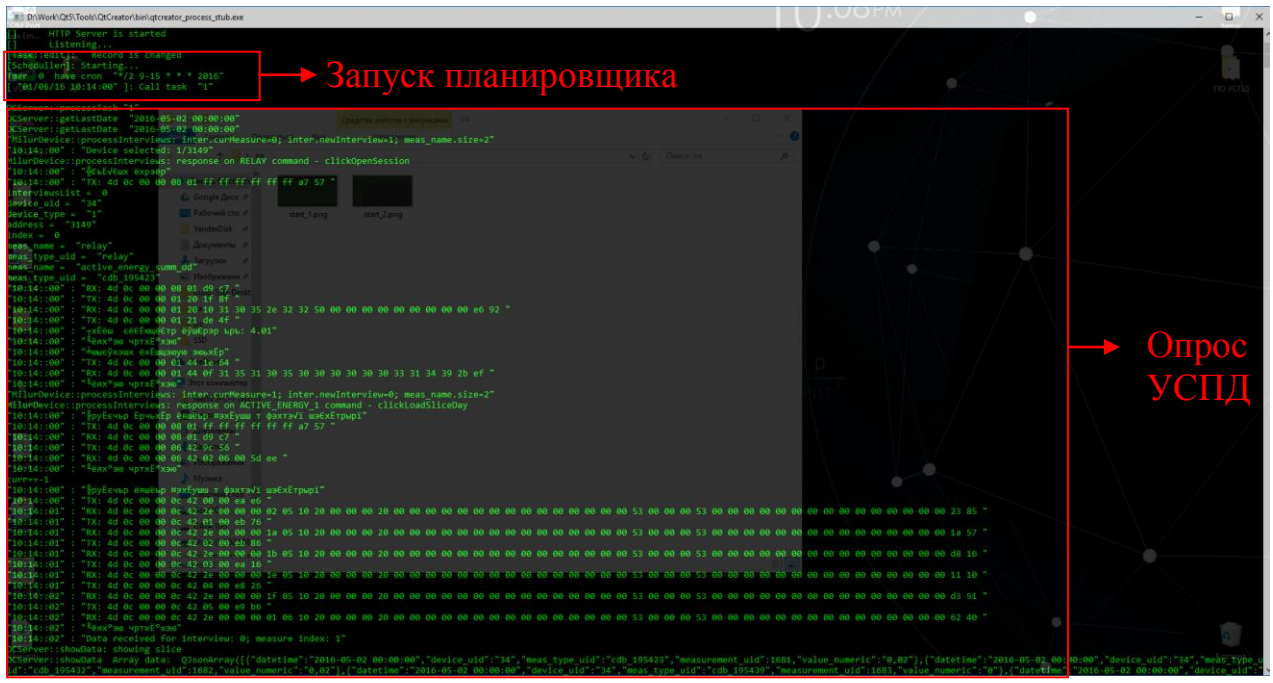


Рисунок 4.4 – Процесс опроса УСПД по расписанию

УСПД отвечает на отправленный запрос и пересылает данные. Процесс передачи данных от УСПД к ЦС изображен на рисунке 4.5.

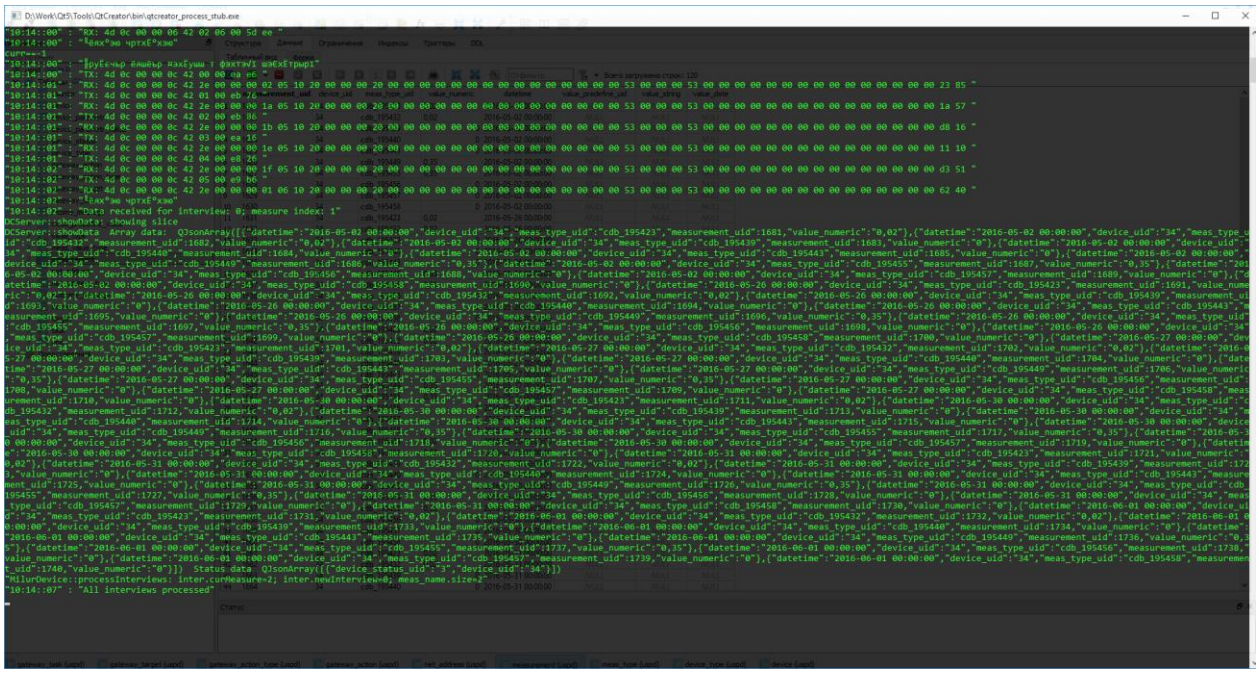


Рисунок 4.5 – Результат опроса УСПД

По окончании приема данных от УСПД, они заносятся в ВБД.

5 ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ

Целью данного раздела является комплексное описание и анализ экономических аспектов выполненной работы. В качестве основного источника для написания данного раздела использовано методическое пособие к выполнению раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» [20].

Основная задача данного раздела заключается в том, чтобы оценить полные денежные затраты на разрабатываемый проект, а также дать хотя бы приближенную экономическую оценку результатов ее внедрения. Это в свою очередь позволит с помощью традиционных показателей эффективности инвестиций оценить экономическую целесообразность осуществления работы.

5.1 Организация и планирование работ

В процессе реализации конкретного проекта необходимо рационально спланировать занятость каждого из его участников. Также необходимо спланировать сроки проведения каждой из работ, входящих в состав проекта.

В настоящем пункте описывается полный перечень проводимых работ, определяются исполнители проекта, а также устанавливаются сроки проведения отдельных работ.

В качестве наглядного результата планирования работ в данном пункте представлен линейный график реализации проекта.

Для выполнения этапов ВКР требуется инженер (И) и научный руководитель (НР). В качестве инженера выступает исполнитель ВКР. Перечень работ по написанию ВКР и продолжительность их выполнения представлены в таблице 5.1.

Таблица 5.1 – Перечень работ и продолжительность их выполнения

Этапы работы	Исполнители	Загрузка исполнителей
1 Постановка целей и задач, получение исходных данных	НР	НР – 100%
2 Составление и утверждение ТЗ	НР, И	НР – 100%, И – 10%
3 Подбор и изучение материалов по тематике	НР, И	НР – 30%, И – 100%
4 Разработка календарного плана	НР, И	НР – 100%, И – 10%
5 Разработка алгоритмов программы	НР, И	НР – 30%, И – 100%
6 Кодирование программы	НР, И	НР – 10%, И – 100%
7 Отладка и тестирование программы	НР, И	НР – 10%, И – 100%
8 Оформление пояснительной записки	И	И – 100%
9 Оформление графического материала	И	И – 100%
10 Подведение итогов	НР, И	НР – 60%, И – 100%

5.1.1 Продолжительность этапов работ

Расчет продолжительности этапов работ по написанию данной ВКР проводился при использовании опытно-статистического метода и реализовывался экспертным способом.

Для определения ожидаемых значений продолжительности работ $t_{ож}$ применяется по усмотрению исполнителя одна из двух формул:

$$t_{ож} = \frac{3 \cdot t_{min} + 2 \cdot t_{max}}{5}, \quad (5.1)$$

$$t_{ож} = \frac{t_{min} + 4 \cdot t_{prob} + t_{max}}{6}, \quad (5.2)$$

где t_{min} – минимальная продолжительность работы в днях;

t_{max} – максимальная продолжительность работы в днях;

t_{prob} – наиболее вероятная продолжительность работы в днях.

Расчет ожидаемого значения продолжительности работ по каждому этапу проводился с использованием второй формулы, так как она дает более

надежные оценки. Исходные данные для расчета $t_{ож}$ по каждому этапу представлены в таблице 5.2.

Таблица 5.2 – Исходные данные для расчета $t_{ож}$

Этапы работы	t_{min}, дни	t_{max}, дни	t_{prob}, дни
1 Постановка целей и задач, получение исходных данных	1	3	2
2 Составление и утверждение ТЗ	1	5	3
3 Подбор и изучение материалов по тематике	3	7	5
4 Разработка календарного плана	2	5	3
5 Разработка алгоритмов программы	10	15	12
6 Кодирование программы	10	15	12
7 Отладка и тестирование программы	10	15	12
8 Оформление пояснительной записки	10	17	12
9 Оформление графического материала	5	10	8
10 Подведение итогов	7	12	10

Рассчитаем $t_{ож}$ для каждого этапа:

$$t_{ож1} = \frac{1 + 4 \cdot 2 + 3}{6} = 2;$$

$$t_{ож2} = \frac{1 + 4 \cdot 3 + 5}{6} = 3;$$

$$t_{ож3} = \frac{3 + 4 \cdot 5 + 7}{6} = 5;$$

$$t_{ож4} = \frac{2 + 4 \cdot 3 + 5}{6} = 3{,}17;$$

$$t_{ож5} = \frac{10 + 4 \cdot 12 + 15}{6} = 12{,}17;$$

$$t_{ож6} = \frac{10 + 4 \cdot 12 + 15}{6} = 12{,}17;$$

$$t_{ож7} = \frac{10 + 4 \cdot 12 + 15}{6} = 12{,}17;$$

$$t_{ож8} = \frac{10 + 4 \cdot 17 + 12}{6} = 12{,}5;$$

$$t_{ож9} = \frac{5 + 4 \cdot 10 + 8}{6} = 7{,}83;$$

$$t_{ож10} = \frac{7 + 4 \cdot 10 + 12}{6} = 9{,}83;$$

Для того, чтобы построить линейный график работ, необходимо рассчитать длительность этапов в рабочих днях и перевести полученные значения в календарные дни.

Расчет длительности этапов в рабочих днях производится в соответствии со следующей формулой:

$$Т_{РД} = \frac{t_{ож}}{К_{ВН}} \cdot К_{Д}, \quad (5.3)$$

где $t_{ож}$ – продолжительность работы в днях;

$К_{ВН}$ – коэффициент выполнения работ, учитывающих влияние внешних факторов на соблюдение предварительно определенных длительностей, в частности, возможно $К_{ВН}$ равно единице;

$К_{Д}$ – коэффициент, учитывающий дополнительное время на компенсацию непредвиденных задержек и согласование работ (коэффициент $К_{Д}$ может варьироваться от 1 до 1,2. Конкретное значение коэффициента принимает сам исполнитель).

Исходные данные для расчета длительности этапов в рабочих днях приведены в таблице 5.3.

Таблица 5.3 – Исходные данные для расчета длительности этапов в рабочих днях

Этапы работы	$t_{ож}$, дни	$К_{ВН}$	$К_{Д}$
1 Постановка целей и задач, получение исходных данных	2,00	1	1
2 Составление и утверждение ТЗ	3,00	1	1
3 Подбор и изучение материалов по тематике	5,00	1	1
4 Разработка календарного плана	3,17	1	1
5 Разработка алгоритмов программы	12,17	1	1,1
6 Кодирование программы	12,17	1	1,2
7 Отладка и тестирование программы	12,17	1	1,2
8 Оформление пояснительной записки	12,50	1	1,1
9 Оформление графического материала	7,83	1	1
10 Подведение итогов	9,83	1	1

Рассчитаем $Т_{РД}$ для каждого этапа:

$$Т_{РД1} = \frac{2}{1} \cdot 1 = 2;$$

$$T_{РД2} = \frac{3}{1} \cdot 1 = 3;$$

$$T_{РД3} = \frac{5}{1} \cdot 1 = 5;$$

$$T_{РД4} = \frac{3{,}17}{1} \cdot 1 = 3{,}17;$$

$$T_{РД5} = \frac{12{,}17}{1} \cdot 1{,}1 = 13{,}38;$$

$$T_{РД6} = \frac{12{,}17}{1} \cdot 1{,}2 = 14{,}6;$$

$$T_{РД7} = \frac{12{,}17}{1} \cdot 1{,}2 = 14{,}6;$$

$$T_{РД8} = \frac{12{,}5}{1} \cdot 1{,}1 = 13{,}75;$$

$$T_{РД9} = \frac{7{,}83}{1} \cdot 1 = 7{,}83;$$

$$T_{РД10} = \frac{9{,}83}{1} \cdot 1 = 9{,}83;$$

Расчет продолжительности этапа в календарных днях осуществляется по следующей формуле:

$$T_{КД} = T_{РД} \cdot T_{К}, \quad (5.4)$$

где $T_{КД}$ – продолжительность выполнения этапа в календарных днях;

$T_{К}$ – коэффициент календарности, позволяющий перейти от длительности работ в рабочих днях к их аналогам в календарных днях.

Коэффициент календарности $T_{К}$ рассчитывается по следующей формуле:

$$T_{К} = \frac{T_{КАЛ}}{T_{КАЛ} - T_{ВД} - T_{ПД}}, \quad (5.5)$$

где $T_{КАЛ}$ – календарные дни ($T_{КАЛ}$ = 365);

$T_{ВД}$ – выходные дни ($T_{ВД}$ = 52);

$T_{ПД}$ – праздничные дни ($T_{ПД}$ = 10).

Рассчитаем коэффициент $T_{К}$:

$$T_{К} = \frac{365}{365 - 52 - 10} = 1{,}205.$$

Рассчитаем продолжительность каждого этапа в календарных днях:

$$Т_{КД1} = 2 \cdot 1{,}205 = 2{,}41;$$

$$Т_{КД2} = 3 \cdot 1{,}205 = 3{,}62;$$

$$Т_{КД3} = 5 \cdot 1{,}205 = 6{,}03;$$

$$Т_{КД4} = 3{,}17 \cdot 1{,}205 = 3{,}82;$$

$$Т_{КД5} = 13{,}38 \cdot 1{,}205 = 16{,}13;$$

$$Т_{КД6} = 14{,}6 \cdot 1{,}205 = 17{,}59;$$

$$Т_{КД7} = 14{,}6 \cdot 1{,}205 = 17{,}59;$$

$$Т_{КД8} = 13{,}75 \cdot 1{,}205 = 16{,}57;$$

$$Т_{КД9} = 7{,}83 \cdot 1{,}205 = 9{,}44;$$

$$Т_{КД10} = 9{,}83 \cdot 1{,}205 = 11{,}85;$$

Трудозатраты на выполнение проекта для каждого из его участников представлены в таблице 5.4.

Таблица 5.4 – Трудозатраты на выполнение проекта

Этап	Ис-пол-ни-тели	Продолжительность работ, дни				Трудоемкость работ по исполнителям чел.-дн.			
						$Т_{РД}$		$Т_{КД}$	
		t_{min}	t_{max}	t_{prob}	$t_{ож}$	НР	И	НР	И
1	2	3	4	5	6	7	8	9	10
Постановка целей и задач, получение исходных данных	НР	1	3	2	2,00	2,00	-	2,41	-
Составление и утверждение ТЗ	НР, И	1	5	3	3,00	3,00	0,30	3,62	0,36
Подбор и изучение материалов по тематике	НР, И	3	7	5	5,00	1,50	5,00	1,81	6,03
Разработка календарного плана	НР, И	2	5	3	3,17	3,17	0,32	3,82	0,38
Разработка алгоритмов программы	НР, И	10	15	12	12,17	4,02	13,38	4,84	16,13
Кодирование	НР,	10	15	12	12,17	1,46	14,60	1,76	17,59

программы	И								
Отладка и тестирование программы	НР, И	10	15	12	12,17	1,46	14,60	1,76	17,59
Оформление пояснительной записки	И	10	17	12	12,50	-	13,75	-	16,57
Оформление графического материала	И	5	10	8	7,83	-	7,83	-	9,44
Подведение итогов	НР, И	7	12	10	9,83	5,90	9,83	7,11	11,85
Итого:					79,83	22,50	79,62	27,11	95,94

Исходя из данных, представленных в таблице 5.4 был составлен линейный график работ (таблица 5.5).

Таблица 5.5 – Линейный график работ

Этап	НР	И	Март			Апрель			Май			Июнь	
			10	20	30	40	50	60	70	80	90	100	110
1	2,41	-											
2	3,62	0,36											
3	1,81	6,03											
4	3,82	0,38											
5	4,84	16,13											
6	1,76	17,59											
7	1,76	17,59											
8	-	16,57											
9	-	9,44											
10	7,11	11,85											

– научный руководитель (НР);

– исполнитель (И).

5.1.2 Расчет накопления готовности проекта

Целью данного пункта является оценка текущих состояний работы над проектом. Величина накопления готовности работы показывает, на сколько процентов по окончании текущего (i-го) этапа выполнен общий объем работ по проекту в целом.

Степень готовности проекта определяется следующей формулой:

$$\text{СГ}_i = \frac{\text{ТР}_i^{\text{Н}}}{\text{ТР}_{\text{общ}}} = \frac{\sum_{k=1}^{i} \text{ТР}_k}{\text{ТР}_{\text{общ}}} = \frac{\sum_{k=1}^{i} \sum_{j=1}^{m} \text{ТР}_{km}}{\sum_{k=1}^{I} \sum_{j=1}^{m} \text{ТР}_{km}}, \quad (5.6)$$

где $\text{ТР}_{\text{общ}}$ – общая трудоемкость проекта;

ТР_i (ТР_k) – трудоемкость i-го (k-го) этапа проекта;

$i = \overline{1, I}$ – индекс этапа;

$ТР_i^Н$ – накопленная трудоемкость i-го этапа проекта по его завершении;

$ТР_{ij}$ ($ТР_{ik}$) – трудоемкость работ, выполняемых j-м участником на i-м этапе;

$j = \overline{1, m}$ – индекс исполнителя.

Исходя из таблицы 4 величины $ТР_{ij}$ ($ТР_{kj}$) находятся в девятом и десятом столбцах таблицы. $ТР_{общ}$ равна сумме чисел из итоговых ячеек этих столбцов.

Таким образом, степень готовности проекта отражена в таблице 5.6.

Таблица 5.6 – Нарастание технической готовности проекта и удельный вес каждого этапа

Этап	$ТР_i$, %	$СГ_i$, %
Постановка целей и задач, получение исходных данных	1,96	1,96
Составление и утверждение ТЗ	3,23	5,19
Подбор и изучение материалов по тематике	6,37	11,56
Разработка календарного плана	3,41	14,97
Разработка алгоритмов программы	17,04	32,01
Кодирование программы	15,73	47,74
Отладка и тестирование программы	15,73	63,47
Оформление пояснительной записки	13,47	76,94
Оформление графического материала	7,67	84,61
Подведение итогов	15,39	100

5.1.3 Расчет сметы затрат на выполнение проекта

В состав затрат на создание проекта включается стоимость всех расходов, необходимых для его реализации. Расчет сметной стоимости ее выполнения производится по следующим статьям затрат:

- материалы и покупные изделия;
- заработная плата;
- социальный налог;
- расходы на электроэнергию (без освещения);
- амортизационные отчисления;
- командировочные расходы;

– оплата услуг связи;

– арендная плата за пользование имуществом;

– прочие услуги (сторонних организаций);

– прочие (накладные расходы) расходы.

5.1.4 Расчет затрат на материалы

К данной статье расходов относится стоимость материалов и других материальных ценностей, расходуемых непосредственно в процессе выполнения работ над проектом. Расчет затрат на материалы, использовавшиеся в ходе выполнения проекта, представлены в таблице 5.7.

Таблица 5.7 – Расчет затрат на материалы

Наименование материалов	Цена за единицу, руб.	Количество	Сумма, руб.
Бумага для принтера формата А4	240,00	1 упаковка	240,00
Картридж для принтера	4645,00	1 штука	4645,00
Ручка шариковая	10,00	1 штука	10,00
Qt 5.4.0 коммерческая лицензия	14244,31	1 штука	10421,00
Итого:			19139,31

Допустим, что транспортно-заготовительные расходы (сокращенно ТЗР) составляют пять процентов от отпускной цены материалов. Исходя из этого рассчитаем расходы на материалы с учетом ТЗР:

$С_{мат}$ = 19139,31· 1,05 = 20096,28.

Таким образом, расходы на материалы ($С_{мат}$) составляют 20096,28 рублей.

5.1.5 Расчет заработной платы

Данная статья расходов включает заработную плату научного руководителя и исполнителя проекта, а также премии, входящие в фонд заработной платы.

Среднедневная тарифная заработная плата рассчитывается по следующей формуле:

$$ЗП_{дн-т} = \frac{МО}{24,83}, \quad (5.7)$$

где МО – величина месячного оклада.

Данная формула учитывает, что в году 298 рабочих дней и, следовательно, в месяце в среднем 24,83 рабочих дня (при шестидневной рабочей неделе).

Для учета в заработной плате премий, дополнительной зарплаты и районной надбавки используется следующий ряд коэффициентов:

$К_{ПР}$ = 1,1;

$К_{доп\ ЗП}$ = 1,188;

$К_{р}$ = 1,3.

Для того, чтобы перейти от тарифной суммы заработка исполнителя проекта к соответствующему полному нужно воспользоваться следующей формулой:

$$ЗП_{полн} = ЗП_{дн-т} \cdot К_{ПР} \cdot К_{доп\ ЗП} \cdot К_{р} = ЗП_{дн-т} \cdot 1{,}699 \qquad (5.8)$$

Расчет затрат на полную заработную плату представлен в виде таблицы 5.8. Затраты времени по каждому исполнителю в рабочих днях взяты из таблицы 5.4.

Таблица 5.8 – Затраты на заработную плату

Исполнитель	Оклад, руб./мес.	Среднеквартальная ставка, руб./день	Затраты времени, раб. дни	Коэффициент	Фонд заработной платы, руб.
НР	14584,32	587,37	23	1,699	22952,66
И	20000,00	805,48	80	1,699	109480,84

Продолжение таблицы 5.8

Исполнитель	Оклад, руб./мес.	Среднеквартальная ставка, руб./день	Затраты времени, раб. дни	Коэффициент	Фонд заработной платы, руб.
Итого:					132433,50

Таким образом, затраты на заработную плату составили 132433,50 рублей.

5.1.6 Расчет затрат на социальный налог

Затраты на единый социальный налог (сокращенно ЕСН), включающий в себя отчисления в пенсионный фонд, на социальное и медицинское страхование, составляют 30 процентов от полной заработной платы по проекту:

$$С_{соц} = С_{зп} \cdot 0{,}3 \quad (5.9)$$

Рассчитаем затраты на ЕСН, используя формулу 3.9:

$С_{соц} = 132433{,}50 \cdot 0{,}3 = 39730{,}05.$

Таким образом, затраты на ЕСН составят 39730,05 рублей.

5.1.7 Расчет затрат на электроэнергию

Данный вид расходов включает в себя затраты на электроэнергию, потраченную в ходе выполнения проекта на работу используемого оборудования. Затраты на электроэнергию рассчитываются по следующей формуле:

$$С_{эл.об.} = Р_{об} \cdot t_{об} \cdot Ц_{э} \quad (5.10)$$

где $Р_{об}$ – мощность, потребляемая оборудованием, кВт;

$Ц_{э}$ – тариф на 1 кВт·час;

$t_{об}$ – время работы оборудования, час.

Для Центра системного проектирования ТУСУР тариф на электроэнергию составляет 6,09 рублей за киловатт в час (с учетом НДС).

Время работы оборудования вычисляется на основе итоговых данных таблицы 3.4 для инженера ($Т_{РД}$) из расчета, что продолжительность рабочего дня составляет 8 часов:

$$t_{об} = Т_{РД} \cdot К_{t} \quad (5.11)$$

Где $К_{t}$ – коэффициент использования оборудования по времени, равный отношению времени его работы в процессе выполнения проекта к ТРД. Данный коэффициент не должен быть больше единицы и определяется исполнителем самостоятельно. В рамках выполнения данного проекта коэффициент $К_{t}$ для персонального компьютера составил 0,9, для лазерного принтера – 0,03.

Рассчитаем время работы персонального компьютера:

$t_{об} = 79{,}62 \cdot 0{,}9 \cdot 8 = 574.$

Рассчитаем время работы лазерного принтера:

$t_{об} = 79{,}62 \cdot 0{,}03 \cdot 8 = 20$

Мощность, потребляемая оборудованием, рассчитывается по формуле 5.12:

$$P_{ОБ} = P_{ном} \cdot К_С \quad (5.12)$$

где $P_{ном}$ – номинальная мощность оборудования в киловаттах;

$К_С$ – коэффициент загрузки, зависящий от средней степени использования номинальной мощности. Для технологического оборудования малой мощности $К_С$ равен единице.

Номинальная мощность оборудования, используемого в ходе реализации проекта, составляет:

– для персонального компьютера – 0,4 киловатта;

– для лазерного принтера – 0,48 киловатта.

Расчет затрат на электроэнергию для технологических целей представлен в таблице 5.9.

Таблица 5.9 – Затраты на электроэнергию технологическую

Наименование оборудования	Время работы оборудования $t_{об}$, час	Потребляемая мощность оборудования $P_{ОБ}$, кВт	Затраты, руб.
Персональный компьютер	574	0,4	1398,26
Лазерный принтер	20	0,48	58,46
Итого:			1456,72

Таким образом, затраты на электроэнергию в процессе выполнения проекта составили 1456,72 рублей.

5.1.8 Расчет амортизационных расходов

В статье «Амортизационные отчисления» рассчитывается амортизация используемого оборудования за время выполнения проекта.

Расчет амортизационных расходов осуществляется по формуле 3.13.

$$С_{АМ} = \frac{Н_А \cdot Ц_{ОБ} \cdot t_{рф} \cdot n}{F_Д}, \quad (5.13)$$

где $Н_А$ – годовая норма амортизации единицы оборудования;

$Ц_{ОБ}$ – балансовая стоимость единицы оборудования с учетом ТЗР;

$F_Д$ – действительный годовой фонд времени работы соответствующего оборудования;

$t_{рф}$ – фактическое время работы оборудования в ходе выполнения проекта;

n – число задействованных однотипных единиц оборудования.

Годовая норма амортизации рассчитывается по следующей формуле:

$$Н_А = \frac{1}{СА}, \quad (5.14)$$

где СА – срок амортизации оборудования.

Из постановления правительства «О классификации основных средств, включенных в амортизационные группы» [21] срок амортизации для персональных компьютеров и печатающих устройств составляет свыше двух лет до трех лет включительно. Зададим значение срока амортизации, равное 2,5 года.

Рассчитаем годовую норму амортизации:

$$Н_А = \frac{1}{2{,}5} = 0{,}4.$$

Таким образом, годовая норма амортизации для персонального компьютера и для лазерного принтера составляет 0,4.

Действительный годовой фонд времени работы оборудования можно принять как произведение количества рабочих дней в году на продолжительность использования оборудования в течение одного рабочего дня (в часах).

Если учесть, что в году 298 рабочих дней и восьмичасовой рабочий день, то действительный годовой фонд времени работы персонального компьютера составляет:

$F_Д = 298 \cdot 8 = 2384$ часа.

Среднее время использования принтера составляет 1,5 часа за рабочий день. Таким образом, годовой фонд времени работы лазерного принтера составляет:

$F_Д = 298 \cdot 1,5 = 447$ часов.

Расчет амортизационных расходов представлен в таблице 5.10.

Таблица 5.10 – Амортизационные расходы на оборудование

Оборудование	Годовая норма амортизации $Н_А$	Балансовая стоимость $Ц_{ОБ}$, руб.	Фактическое время работы $t_{рф}$, часы	Действительный годовой фонд времени работы, часы	Амортизационные расходы $С_{АМ}$, руб.
Персональный компьютер	0,4	42850,00	574	2384	4126,83
Лазерный принтер	0,4	13790,00	20	447	246,80
Итого:					4373,63

Таким образом амортизационные расходы на оборудование, используемое в ходе реализации проекта, составили 4373,63 рубля.

5.1.9 Расчет прочих расходов

В статье «Прочие расходы» отражены расходы на выполнение проекта, которые не учтены в предыдущих статьях. Данный тип расходов рассчитывается по следующей формуле:

$$С_{проч} = (С_{мат} + С_{ЗП} + С_{соц} + С_{эл.об.} + С_{АМ}) \cdot 0,1, \quad (7.15)$$

Рассчитаем затраты на прочие расходы при выполнение проекта:

$С_{проч} = (19139,31 + 132433,50 + 39730,05 + 1456,72 + 4373,63) \cdot 0,1;$

$С_{проч} = 19713,32.$

Таким образом, затраты на прочие расходы при выполнении дипломного проекта составили 19713,32 рубля.

5.1.10 Расчет общей себестоимости разработки

Проведя расчет по всем статьям сметы затрат на разработку, можно определить общую себестоимость проекта «Сервер сбора данных для гетерогенной системы учета энергоресурсов» (таблица 5.11).

Таблица 5.11 – Смета затрат на разработку проекта

Статья затрат	Условное обозначение	Сумма, руб.
Материалы и покупные изделия	$С_{мат}$	19139,31
Основная заработная плата	$С_{зп}$	132433,50
Отчисления в социальные фонды	$С_{соц}$	39730,05
Расходы на электроэнергию	$С_{эл.об.}$	1456,72
Амортизационные отчисления	$С_{АМ}$	4373,63
Прочие расходы	$С_{проч}$	19713,32
Итого:		216846,53

Таким образом, затраты на разработку составили 216846,53 рублей.

5.1.11 Расчет прибыли

В связи с недостатком данных для применения более точных методов расчета прибыли от реализации проекта, было решено принять прибыль как двадцать процентов от полной себестоимости проекта.

Рассчитаем объем прибыли от реализации проекта:

$С_{пр} = 216846,53 \cdot 0,2 = 43369,31$.

Исходя из полученных расчетов, прибыль от реализации проекта составляет 43369,31 рубля.

5.1.12 Расчет НДС

НДС составляет восемнадцать процентов от суммы затрат на разработку и прибыли.

Рассчитаем НДС:

$С_{НДС} = (216846,53 + 43369,31) \cdot 0,18 = 46838,85$.

Таким образом, НДС составляет 46838,85 рублей.

5.1.13 Цена разработки НИР

Цена разработки НИР равна сумме полной себестоимости, прибыли и НДС.

Рассчитаем цену разработки проекта:

$Ц_{НИР} = 216846,53 + 43369,31 + 46838,85 = 307054,69$.

Таким образом, цена разработки НИР составляет 307054,69 рублей.

5.2 Оценка экономической эффективности проекта

В связи с тем, что разрабатываемый проект является лишь частью АСКУЭ, а также в связи с недостатком информации для полноценного анализа, полная оценка экономической эффективности проекта невозможна.

Разрабатываемая система коммерческого учета энергоресурсов должна осуществлять взаимодействие с управляющими компаниями, поставляющими энергию конечным потребителям.

Внедрение автоматизации сбора показаний с устройств учета электроэнергии позволит:

– организацию достоверного учета и оперативного контроля за потреблением электроэнергии по каждой квартире и жилому дому в целом в связи с переходом от системы выписки счетов за потребленную электроэнергию жильцами к системе выписки счетов энергоснабжающей организацией;

– исключение хищений электроэнергии за счет оперативного контроля за балансом энергопотребления;

– сокращение затрат на персонал, контролирующий показания квартирных счетчиков, а также снижение затрат на содержание их рабочего места (приблизительно 50 процентов от заработной платы сотрудника);

– потенциальное снижение задолженностей по оплате электроэнергии в связи с тем, что разрабатываемая система подразумевает наличие механизмов оповещения должников (посредством смс-сообщений, сообщений в личном кабинете и сообщений по электронной почте) и наличие механизмов автоматического отключения электроэнергии (устройства учета электроэнергии снабжены механизмом дистанционного отключения с пульта управления оператора АСКУЭ).

Расчет экономической эффективности разрабатываемого проекта, связанной с сокращением затрат на персонал, контролирующий показания индивидуальных устройств учета электроэнергии, проводился на примере управляющей компании ООО «Компания «Управа» [22]. По данным компании,

в штате сотрудников насчитывается два инспектора по контролю индивидуальных приборов учета. Средняя заработная плата одного такого инженера составляет 20000 рублей в месяц [23].

За один год управляющая компания начисляет инженерам заработную плату суммой:

$С_{ЗП} = 20000 \cdot 12 \cdot 2 = 480000$.

Таким образом, за один год управляющая компания расходует на оплату труда инспекторам по контролю индивидуальных приборов учета 480000 рублей.

При вводе в эксплуатацию системы автоматического сбора данных у управляющей компании исчезает потребность в инспекторах по контролю индивидуальных приборов учета. На замену им необходимо взять в штат одного сервисного инженера и одного оператора ПК.

Средняя заработная плата сервисного инженера составляет 20000 рублей в месяц [23], средняя заработная плата оператора ПК составляет 12000 рублей в месяц [23].

Рассчитаем затраты на оплату заработной платы для нового персонала за один год:

$С_{ЗП} = (20000 \cdot 12) + (12000 \cdot 12) = 384000$.

Таким образом, затраты на заработную плату нового персонала составили 384000 рублей.

Рассчитаем разницу между затратами:

$480000 – 384000 = 96000$.

Также стоит учесть снижение затрат на содержание рабочего места замененного персонала, которое составляет в среднем пятьдесят процентов от заработной платы сотрудника.

Рассчитаем затраты на содержание рабочего места двух инспекторов по контролю индивидуальных приборов учета за один год:

$10000 \cdot 2 \cdot 12 = 240000$.

Рассчитаем затраты на содержание рабочих мест для нового персонала за один год:

$10000 \cdot 12 + 6000 \cdot 12 = 120000 + 72\,000 = 192000$.

Рассчитаем разницу между затратами:

$240000 – 192000 = 48000$.

Рассчитаем общую разницу между затратами:

$48000 + 96000 = 144000$

Таким образом, используя систему автоматического сбора данных управляющая компания экономит 144000 рублей в год.

Расчет экономической эффективности проекта, связанной с потенциальным снижением задолженности по оплате электроэнергии проводился на примере ПАО «Томскэнергосбыт» [24].

Основными факторами возникновения задолженности являются:

– отсутствие у плательщика возможностей для оплаты электроэнергии (временные финансовые трудности, трудности, сложившиеся в результате обстоятельств непреодолимой силы);

– отсутствие у плательщика желания оплаты электроэнергии.

По данным ПАО «Томскэнергосбыт» средняя текущая сумма задолженности по оплате электроэнергии частными лицами по Томской области за 2015 год составила 42 млн. рублей.

Так как разрабатываемая система обладает механизмом автоматического отключения электроэнергии и механизмом оповещения потребителей электроэнергии о появлении задолженности по оплате, то ее внедрение окажет существенное влияние на дисциплину платежей населения. По экспертным оценкам сумма задолженности по оплате электроэнергии сократится примерно на двадцать процентов (более точные оценки сокращения уровня задолженности требуют проведения масштабных исследований, выходящих за рамки настоящей выпускной квалификационной работы).

Опираясь на данные ПАО «Томскэнергосбыт» можно рассчитать примерную сумму по снижению задолженности:

$$\frac{42000000 \cdot 20}{100} = 8400000.$$

Таким образом, при внедрении разрабатываемой системы будет наблюдаться единовременный экономический эффект в размере около 8400000 рублей.

5.3 Оценка научно-технического уровня НИР

Научно-технический уровень характеризует влияние проекта на уровень и динамику обеспечения научно-технического прогресса в данной области. Для оценки научной ценности, технической значимости и эффективности использовался метод балльных оценок.

Сущность метода заключается в том, что на основе оценок признаков работы определяется интегральный показатель ее научно-технического уровня, рассчитываемый по следующей формуле:

$$I_{\text{НТУ}} = \sum_{i=1}^{3} R_i \cdot n_i \qquad (5.16)$$

где $I_{\text{НТУ}}$ – интегральный индекс научно-технического уровня;

R_i – весовой коэффициент i-го признака научно-технического эффекта;

n_i – количественная оценка i-го признака научно-технического эффекта в баллах.

Оценки научно-технического уровня данной работы представлены в виде таблицы 5.11.

Таблица 5.11 – Оценка научно-технического уровня НИР

Значимость	Фактор НТУ	Уровень фактора	Выбранный балл	Обоснование выбранного балла
0,4	Уровень новизны	Относи-тельно новая	4	Система предоставляет возможность работы с датчиками и устройствами разных типов и разных производителей
0,1	Теоретический уровень	Разработка алгоритма	6	Разработка новых алгоритмов, позволяющих взаимодействовать с датчиками и устройствами разных типов и разных производителей
0,5	Возможность реализации	В течение первых лет	10	Автоматизированная система сбора данных начинает функционировать сразу же после внедрения АСКУЭ

Используя данные таблицы 5.11 рассчитаем показатель научно-технического уровня для данного проекта:

$I_{НТУ} = 0,4 \cdot 4 + 0,1 \cdot 6 + 0,5 \cdot 10 = 7,2.$

В таблице 5.12 указано соответствие качественных уровней НИР значениям показателя, рассчитываемого по формуле 7.16.

Таблица 5.12 – Соответствие качественных уровней НИР значению интегрального индекса научно-технического уровня

Уровень НТЭ	Показатель НТЭ
Низкий	1-4
Средний	4-7
Высокий	8-10

Исходя из данных, указанных в таблице 5.12, данный проект имеет средний уровень научно-технического эффекта.

6 СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ

В данном разделе выпускной квалификационной работы проведен анализ возможных сбоев разрабатываемой системы и последствия, возникающие в результате этих сбоев, а также выявлены причины возможных сбоев. На основании полученного анализа разработаны меры по предотвращению аварийных ситуаций и сбоев как для системы в целом, так и для сервера сбора данных в частности.

6.1 Анализ возможных сбоев разрабатываемой системы

Возникновение сбоев в работе автоматизированной системы коммерческого энергоресурсов (АСКУЭ) может привести к потере, повреждению или получению недостоверных данных.

Все данные, обрабатываемые АСКУЭ можно разделить на несколько категорий:

- данные о потребителях электроэнергии и подключенных устройствах;
- данные о потребленной электроэнергии;
- данные об оплате за потребленную электроэнергию.

Если в результате сбоя были повреждены данные о потребителях электроэнергии или данные о подключенных устройствах учета, то не будут учтены показания потребленных энергоресурсов, что повлечет за собой убытки управляющей компании.

Повреждение данных о потребленной электроэнергии повлечет за собой неверные расчеты потребленных ресурсов, что может привести либо к убыткам потребителей, либо к убыткам управляющей компании.

АСКУЭ обладает системой дистанционного отключения электроэнергии при возникновении задолженности у конкретных потребителей, а также системой подачи электроэнергии при ее погашении. При возникновении повреждения данных об оплате за потребленную электроэнергию система может ошибочно отключить электроэнергию добросовестному потребителю,

либо включить ее потребителю, имеющему задолженности по оплате. В последнем случае управляющая компания понесет убытки.

Все сбои, происходящие в ходе работы программы, можно разделить на несколько категорий:

- аппаратные сбои;
- сбои, возникающие в результате неправильное эксплуатации программы;
- программные сбои.

6.2 Анализ причин сбоев в работе системы

АСКУЭ представляет собой аппаратно-программную систему, включающую в себя множество устройств, общающихся друг с другом посредством сетевого соединения. При повреждении сетевой магистрали может произойти нарушение сетевого взаимодействия между компонентами системы, что приведет либо к повреждению передаваемых данных, либо к их полной потере. При выходе из строя одного из передающих узлов системы также происходит утеря и искажение данных.

При выходе из строя центрального сервера системы возникает риск полной утери всех хранимых данных о системе, так как именно в этом блоке АСКУЭ располагается центральная база данных, хранящая основную информацию, касающуюся функционирования системы.

Так как все данные о потребленных ресурсах передаются по сети, то возникает риск их перехвата и изменения злоумышленниками. Также существует риск несанкционированного подключения злоумышленниками к отдельным аппаратным узлам системы для получения или изменения хранимых данных. В ходе функционирования системы могут возникать ситуации, в которых сами пользователи АСКУЭ могут осуществлять неправомерное изменение хранящейся в базе данных информации. Последствиями этих действий служит утеря или искажение информации, что может привести к убыткам пользователей системы и даже к сбою в работе как отдельных модулей системы, так и всей системы в целом.

Также искажение данных может возникать в результате ошибок, возникающих при работе программного обеспечения.

Таким образом, можно выявить следующие причины возникновения сбоев в работе АСКУЭ:

– нарушение сетевого взаимодействия между компонентами системы;

– получение несанкционированного доступа к данным системы злоумышленниками;

– получение несанкционированного доступа к аппаратным компонентам системы;

– выход из строя оборудования системы;

– изменение данных пользователями, не обладающими соответствующими правами;

– ошибки в алгоритмах программного обеспечения.

6.3 Меры по аппаратной защите системы

Меры по аппаратной защите системы включают в себя:

– способы защиты от несанкционированного доступа к аппаратным компонентам системы: устройствам учета энергоресурсов и устройствам сбора и передачи данных;

– способы защиты данных от выхода из строя аппаратных компонентов системы;

– способы защиты данных от выхода из строя одного из каналов связи между устройствами.

В целях обеспечения безопасности от несанкционированного доступа извне, все устройства, подключаемые к сети, должны быть опломбированы. Также все устройства должны быть снабжены системой, реагирующей на вскрытие пломбы. При вскрытии крышки устройства система отправляет тревожный сигнал на пульт оператора системы.

В целях защиты данных от выхода из строя аппаратного обеспечения системы предлагаются следующие меры:

– размещение отдельных программных компонентов системы (сервер приложений и сервер сбора данных) на разных ЭВМ. Данная мера предотвратит полную утерю данных при выходе из строя одной из ЭВМ, а также предотвратит полную остановку работы АСКУЭ;

– размещение центральной и архивной баз данных на разных жестких носителях данных. Данная мера позволит восстановить утерянные данные при выходе из строя одного из носителей данных;

– снабжение каждой из ЭВМ системы блоком бесперебойного питания. В случае незапланированного отключения электроэнергии данная мера позволит корректно завершить работу программного блока и сохранить все данные;

– при выходе из строя одного из устройств сбора и передачи данных (УСПД) может остановиться опрос устройств учета электроэнергии, что может привести к остановке работы системы. Для предотвращения подобных ситуаций АСКУЭ обладает системой, прокладывающей новый маршрут передачи данных через исправно функционирующие сетевые узлы системы;

В случае отказа основного канала связи АСКУЭ обеспечивает выполнение автоматического перехода на резервный канал связи (при его наличии) с документацией этого события в журнале событий системы.

6.4 Организационные меры, обеспечивающие защиту системы

Обеспечение организационных мер по защите системы подразумевает наличие:

– инструкции по эксплуатации АСКУЭ;

– формуляра АСКУЭ.

Перед тем как приступить к работе с АСКУЭ оператору системы необходимо пройти инструктаж и ознакомиться с инструкцией по эксплуатации. Инструкция по эксплуатации должна находиться на рабочем месте оператора и быть в свободном доступе.

Формуляр АСКУЭ предназначен для фиксирования в нем следующих событий:

– события по изменению кода программы;

- внештатные и аварийные ситуации;
- сведения по ремонту аппаратных компонентов системы.

Формуляр должен храниться в одном и том же месте и заполняться лицом, ответственным за эксплуатацию АСКУЭ.

6.5 Меры по программной защите системы.

Программная защита системы включает в себя:

- защиту целостности данных при передаче по сети;
- защиту данных от атак злоумышленников;
- защиту данных от изменения пользователями, не обладающими соответствующими правами доступа;
- защиту данных в случае повреждения базы данных;
- защиту от сбоев в алгоритмах программного обеспечения;
- контроль достоверности измерений узлов учета энергоресурсов (УУЭ);
- наличие системы оповещений о событиях, произошедших в результате работы системы.

6.6 Защита целостности данных

При передаче данных от одного узла АСКУЭ к другому по сети может возникнуть сбой, в результате которого может нарушиться целостность передаваемых данных, либо их полная утеря. Для предотвращения подобных ситуаций в программном обеспечении системы предусмотрен механизм гарантированной доставки данных.

Общий принцип механизма заключается в следующем: отправитель данных содержит объект исходящего сообщения outcome, получатель данных содержит объект входящего сообщения income. Объекты income и outcome обладают уникальными идентификаторами (идентификатор income должен соотноситься с идентификатором outcome), а также содержат в себе флаги об отправке и обработке сообщения (received и processed). Каждый флаг может принимать значение true или false в зависимости от текущего статуса

сообщения. Алгоритм функционирования механизма схематично представлен на рисунке 8.1.

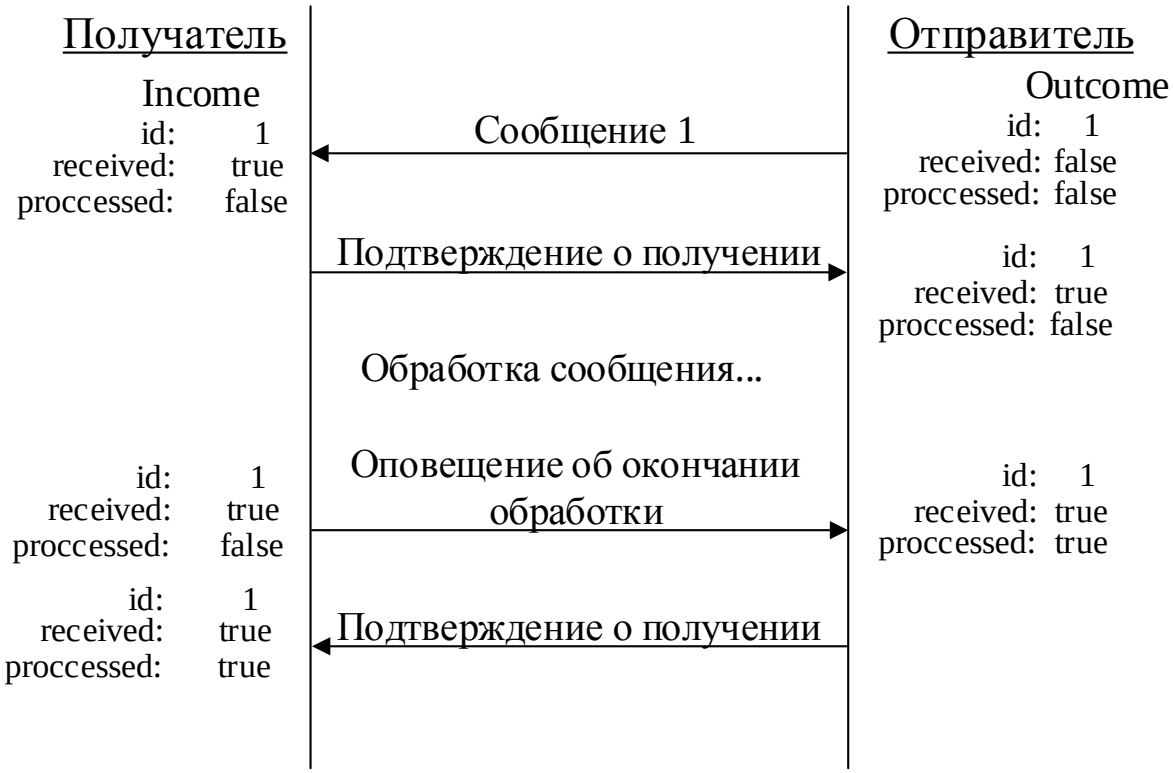


Рисунок 6.1 – Механизм работы системы гарантированной доставки

При отправке сообщения получателю оба флага объекта outcome имеют значение false. При получении сообщения получатель устанавливает флаг received в состояние true и высылает отправителю подтверждение о получении сообщения. Как только получатель получает подтверждение о получении сообщения, он устанавливает флаг received в состояние true. Далее происходит обработка сообщения получателем. Как только получатель заканчивает обработку полученного сообщения, он отправляет отправителю оповещение об окончании обработки. При получении оповещения об окончании обработки отправитель устанавливает флаг processed в состояние true и отправляет получателю подтверждение о получении оповещения. Получатель, получив подтверждение от отправителя, устанавливает флаг processed в состояние true. На этом алгоритм гарантированной доставки завершает свою работу. После того, как доставка осуществлена, отправитель может удалить переданные данные со своего носителя без риска их утери.

6.7 Защита данных от атак злоумышленников

Для того, чтобы обеспечить защиту данных от возможных атак злоумышленников все сетевое взаимодействие между компонентами АСКУЭ происходит с использованием технологии VPN. За счет использования VPN передача информации по сети осуществляется в зашифрованном виде, а также при передаче пакетов по сети подтверждается целостность передаваемых и получаемых данных.

6.8 Защита информационной базы системы

Защиту данных от изменения пользователями обеспечивает разграничение доступа к базам данных для разных групп пользователей.

АСКУЭ подразумевает наличие четырех групп пользователей:

- оператор;
- клиент;
- администратор;
- сервисный инженер.

Права оператора подразумевают доступ к просмотру и изменению информации о клиентах системы, а также к просмотру данных о потребленной электроэнергии.

Права администратора подразумевают добавление/удаление/изменение учетных записей пользователей АСКУЭ, а так же редактирование базы данных.

Права клиента позволяют просмотр информации о потребленных энергоресурсах и предоставляют возможность оплаты потребленных ресурсов через личный кабинет.

АСКУЭ обладает резервным хранилищем данных. Администратор системы обязан осуществлять процедуру резервного копирования базы данных по расписанию, регламентируемому организацией, эксплуатирующей АСКУЭ. Эта мера предотвратит утерю информации при повреждении центральной базы данных.

6.9 Защита от сбоев в алгоритмах программы

Основными причинами ошибок, возникающих в ходе выполнения программ, являются:

– ошибки в алгоритмах, которые могут привести к непредсказуемому поведению программы;

– ошибки, возникающие в связи с возникновением исключительных ситуаций в ходе работы программы, которые могут привести к аварийному завершению.

Для поиска и исправления алгоритмических ошибок в коде применяется тестирование программного обеспечения.

Тестирование обычно преследует две цели:

– подтверждение того, что программа соответствует всем установленным требованиям;

– выявление ситуаций, в которых поведение программы является неправильным или не соответствующим спецификации.

При выполнении данного проекта применялось модульное тестирование, так же называемое юнит-тестированием. Модульное тестирование позволяет проверить на корректность отдельные модули исходного кода программы [25].

Тестирование проекта проводилось с использованием методов, предоставляемых библиотекой QT, так как само приложение разработано на языке Qt/C++.

Для профилактики ошибок, связанных с исключительными ситуациями, в коде языка была использована конструкция try-catch, позволяющая выявлять ошибки, возникающие во время выполнения программы. Использование данной конструкции сообщает о возникновении ошибок без аварийного завершения приложения.

Конструкция try-catch использовалась для обработки следующих ситуаций:

– в структурах с динамическим распределением памяти для того, чтобы избежать утечек памяти;

– при обращении к базе данных и при выполнении SQL-запросов для того, чтобы отслеживать ошибки при взаимодействии с базой данных;

– при осуществлении сетевого взаимодействия с другими блоками системы для того, чтобы отслеживать ошибки, связанные с сетевыми подключениями и передачей данных по сети.

6.10 Контроль достоверности измерений

АСКУЭ осуществляет контроль достоверности потребляемых энергоресурсов. Проверка корректности данных осуществляется программным обеспечением УСПД при получении данных от УУЭ.

Данные считаются корректными, если они:

– неотрицательны;

– меньше максимально допустимого значения (максимальный уровень потребления энергоресурсов рассчитывается статистическими методами);

– меньше предельно допустимого значения, обусловленного типом double (если значение измерения превышает размер double, это свидетельствует о том, что УУЭ неисправно);

– не равно значению 8018 (значение 8018 так же свидетельствует о неисправности УУЭ).

6.11 Журналирование событий АСКУЭ

Для того, чтобы отслеживать корректность выполнения программы, необходима система хранения сообщений, фиксирующих ход ее выполнения. Сообщения о функционировании АСКУЭ хранятся в виде системных журналов, создаваемых системой журналирования событий. Система журналирования событий АСКУЭ хранит информацию обо всех программных событиях.

Каждый модуль АСКУЭ содержит свой собственный журнал. Также система имеет журнал, хранящий в себе все действия пользователей АСКУЭ.

Все журналы фиксируют события двух типов:

– события сети;

– события данных.

К событиям сети относятся события, описывающие ошибки доступа к сетевым узлам системы на сетевом и транспортном уровнях (по модели OSI).

К событиям данных относятся следующие события:

– получение корректных данных;

– получение заведомо недостоверных данных;

– получение данных с нарушенной структурой.

Также журналы содержат в себе записи о возникновениях ошибок:

– дата, время;

– тип ошибки;

– источник ошибки;

– информация об ошибке.

Ниже представлены типы фиксируемых ошибок:

– ошибки формата входных данных;

– ошибки чтения/отсутствия конфигурационного файла;

– ошибки взаимодействия с базами данных.

6.12 Требования к аппаратному и программному обеспечению

Минимальные системные требования к аппаратному обеспечению, необходимые для корректной работы программы, включают в себя:

Для выполнения своих функций ПО ЦС необходимо:

– не менее 4 Гб оперативной памяти;

– не менее 500 Гб дискового пространства;

– процессор Intel Core i5 и выше;

– канал связи с сетью Интернет скоростью не ниже 100 Мбит/с.

Системные программные средства, используемые ПО ЦС, должны предоставляться операционной системой Ubuntu (версия 14.0.4 и выше) либо операционной системой Debian (версия 7.0 и выше).

Также для запуска и корректной работы программного обеспечения необходима установка дополнительного программного обеспечения, включающего в себя:

- сервер Apache (версия 2.0 и выше);
- система управления базами данных PostgreSQL (версия 9.4.0 и выше);
- модуль Apache-PHP (версия 5.4);
- модуль для Apache-memcashe.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта было разработано программное обеспечение сервера сбора данных для гетерогенной системы учета энергоресурсов. Разработанное программное обеспечение включает в свой состав следующие блоки:

- блок обработки задач;
- планировщик задач;
- блок взаимодействия с временной базой данных;
- блок сетевого взаимодействия;
- управляющий блок.

Разработанное приложение ведет регистрацию событий, происходящих во время его работы, и записывает их в специальный файл (журнал событий) в структурированном виде, а также дублирует информацию о событиях в консоль сервера.

Согласно представленным результатам разработки приложения, в процессе запуска сервер сбора данных осуществляет успешное взаимодействие с центральным сервером, а именно:

- запрашивает данные на обновление ВБД;
- получает запрашиваемые данные по сети, используя механизм гарантированной доставки;
- заносит полученные данные в ВБД;
- составляет список задач, согласно полученным данным;
- запускает задачи согласно расписанию.

В следствие того, что программное обеспечение для УСПД находится на стадии разработки, в настоящий момент не представляется возможным протестировать работу его сетевого взаимодействия с сервером сбора данных.

В ходе написания дипломного проекта в разделе финансового менеджмента, ресурсоэффективности и ресурсосбережения была проанализирована экономическая эффективность разработки и выявлена ее

примерная стоимость. В разделе социальная ответственность были проанализированы возможные сбои программы последствия этих сбоев, а также меры защиты программы от возникновения ошибок.

Результатом выполнения дипломного проекта является приложение, соответствующее всем требованиям, перечисленным в техническом задании. Некоторые блоки, разработанные в ходе выполнения дипломного проекта (а именно: планировщик задач, блок по взаимодействию с ВБД), будут использованы при разработке других компонентов АСКУЭ.

Внедрение системы автоматического сбора данных с узлов учета энергоресурсов повысит качество жизни населения и упростит процесс сбора информации о потребляемой энергии для управляющих компаний. Так, автоматизированный сбор показаний позволит управляющим компаниям:

- организовать достоверный учет и оперативный контроль за потреблением энергоресурсов;
- исключить хищения электроэнергии недобросовестными потребителями;
- сократить затраты на персонал, контролирующий показания индивидуальных приборов учета электроэнергии;
- сократить задолженности по оплате электроэнергии.

В свою очередь автоматизированный сбор данных позволит упростить потребителям процесс оплаты электроэнергии: каждому потребителю будут поступать уже заполненные квитанции с указанием потребленного объема энергоресурсов и суммы, которую необходимо оплатить.

CONCLUSION

As a result of the graduation project the software of a data-collection server for heterogeneous system accounting of energy resource was developed. Developed software includes following program blocks:

– task processing block;

– task scheduler;

– temp database interaction block;

– network interaction block;

– control block.

Developed software registers the events that occurs during program execution. Events are recorded by program into the special file (event log) in a structured fashion. Developed program duplicates information about program events into the server console.

According the application development result while data-collection server starting occurs successful network interaction with central server:

– data-collection server requests the data for updating temp database;

– data-collection server receives data over the network using mechanism of a guaranteed delivery;

– data-collection server saves received data into the temp database;

– data-collection server forms the task list based on received data;

– data-collection server runs tasks according the time schedule;

Due to the fact that software for the data collection and transmission device is in the developing stage does not seem possible test the network interaction with data-collection server.

During creating the graduation project in section of the financial management, resource efficiency and resource-saving was analyzed the development cost-effectiveness. Also was defined approximate cost of a developed project. In section of the social accountability was analyzed the possible software failures, consequence of software failures and protection measures from program errors.

The result of the graduation project is application corresponding all specifications listed in the requirement for development. Some blocks developing during graduation project (that is task scheduler and temp database interaction block) will be used in development of other parts the automated system of commercial accounting of energy resources.

The implementation of the automatically data collection from energy metering units will enhance the quality of people life. Also it will prune down a process of information collection about energy consumption for Facility Managers. That way, automatically data collection will allow Facility Managers:

- organize a true recording and operating control of energy consumption;
- eliminate electricity theft by dishonest consumers;
- reduce personnel costs which controls the energy meter reading;
- reduce outstanding electricity bills.

In its turn, automated data collection will allow prune down a process of bills payment: the bills for each consumers will be got to filled receipt form where will be shown a volume of consumed energy resources and payment amount.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Постановление Правительства РФ. О функционировании розничных рынков электрической энергии, полном и (или) частичном ограничении режима потребления электрической энергии от 04.05.2012 N 442 (ред. от 22.02.2016).

2) Автоматизированная система коммерческого учета энергоресурсов (АСКУЭ). Схема функциональной структуры. 2015г. – 9 с.

3) Автоматизированная система коммерческого учета энергоресурсов (АСКУЭ). Описание автоматизируемых функций. 2015г. – 13 с.

4) Автоматизированная система коммерческого учета энергоресурсов (АСКУЭ). Пояснительная записка к техническому проекту на программное обеспечение центрального сервера. 2015г. – 39 с.

5) Технические требования к результатам выполнения комплексного проекта по созданию высокотехнологичного производства с участием российского высшего учебного заведения по теме: «Реализация комплексного проекта по созданию высокотехнологичного производства интеллектуальных приборов энергоучета, разработанных и изготовленных на базе отечественных микроэлектронных компонентов, и гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов на их основе». 2014 г. – 19с.

6) Техническое задание на выполнение научно-исследовательских, опытно-конструкторских и технологических работ по теме: «Разработка гетерогенной автоматизированной системы мониторинга потребляемых энергоресурсов, программного обеспечения, а также разработка и реализация проектно-сметной документации на развертывание и проведение натурных испытаний системы на объектах». 2014 г. – 12 с.

7) Git. Documentation [электронный ресурс]. – 2016. – Режим доступа: https://git-scm.com/doc. Загл. с экрана.

8) StarUML 2 [электронный ресурс]. – 2015. – Режим доступа: http://staruml.io/. – Загл.с экрана.

9) Шлее М. Qt4.5. Профессиональное программирование на С. – СПб: БХВ-Петербург, 2010. – 896 с.

10) Бланшетт Ж., Саммерфилд М. Qt 4: программирование GUI на C++. – М: КУДИЦ-ПРЕСС, 2008. – 736с.

11) Cron-выражения [Электронный ресурс]. – 2016. – Режим доступа: https://online.optimabank.kg/ib6/common/ru/help_Cron_Expression.htm. Загл. с экрана.

12) Cron-выражения [Электронный ресурс]. – 2016. – Режим доступа: https://online.optimabank.kg/ib6/common/ru/help_Cron_Expression.htm. Загл. с экрана.

13) Qt Documentation [электронный ресурс]. – 2016. – Режим доступа: doc.qt.io/. – Загл. с экрана.

14) Харбар. Разработка ORM [электронный ресурс]. – 2014. – Режим доступа: https://habrahabr.ru/post/237889/. Загл. с экрана.

15) Catalog of Patterns of Enterprise Application Architecture [электронный ресурс]. – 2015. – Режим доступа: http://martinfowler.com/eaaCatalog/. Загл. с экрана.

16) Хабрахабр. Регулярные выражения. Пособие для новичков [электронный ресурс]. – 2016. – Режим доступа: https://habrahabr.ru/post/115825/. Загл. с экрана.

17) Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с.

18) Прата С. Язык программирования C++. Декции и упражнения. – М: ООО «И.Д. Вильямс», 2012. – 1248с.

19) Введение в JSON [электронный ресурс]. – 2015. – Режим доступа: http://www.json.org/json-ru.html. Загл. с экрана.

20) В.Ю. Конотопский. Методические указания к выполнению раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» магистерских диссертаций для всех специальностей ИК. – Томск: Изд-во Томского политехнического университета, 2015. – 29 с.

21) Постановление Правительства РФ. О классификации основных средств, включенных в амортизационные группы от 01.01.2002 N 1 (ред. от 06.07.2015).

22) Компания «Управа» [электронный ресурс]. 2016. – Режим доступа: http://uprava.tomsk.ru/. Загл. с экрана.

23) Город работ [электронный ресурс]. 2016. – Режим доступа: http://tomsk.gorodrabot.ru/vacancy/. Загл. с экрана.

24) Томскэнергосбыт [электронный ресурс]. 2016. – Режим доступа: http://www.ensb.tomsk.ru/. Загл. с экрана.

25) Хабрахабр. Юнит-тестирование [электронный ресурс]. – 2016. – Режим доступа: https://habrahabr.ru/post/146449/. Загл. с экрана.

ПРИЛОЖЕНИЕ А

(необязательное)

Описание структуры временной базы данных

Сущность device хранит идентификатор устройства, которое является и первичным, и внешним ключом.

Сущность task хранит информацию о задачах, которые необходимо выполнить. Перечень и подробное описание атрибутов сущности представлены в таблице А.1.

Таблица А.1 – Описание атрибутов сущности tasks

Наименование атрибута	Тип данных	Описание атрибута
uid	целочисленное десятичное число	Идентификационный номер задачи. Поле является первичным ключом
cronjob	строка символов	Cron-выражение для выполнения задачи
parameters	строка символов	Параметры опроса УСПД
status	целочисленное десятичное число	Статус задачи

Сущность target связывает сущность device с сущностью task. Перечень и описание атрибутов сущности представлены в таблице А.2.

Таблица А.2 – Описание атрибутов сущности target

Наименование атрибута	Тип данных	Описание атрибута
uid	целочисленное десятичное число	Идентификационный номер. Поле является первичным ключом
task_uid	целочисленное десятичное число	Номер задачи. Поле является внешним ключом
device_uid	строка символов	Номер УСПД. Поле является внешним ключом.

Сущность net_address_type хранит типы сетевых адресов устройств. Перечень и описание атрибутов сущности представлены в таблице А.3.

Таблица А.3 – Описание атрибутов сущности net_address_type

Наименование атрибута	Тип данных	Описание атрибута
net_addres_type_uid	целочисленное десятичное число	Идентификационный номер. Поле является первичным ключом
name_ru	строка символов	Тип сетевого адреса устройства

Сущность net_address хранит сетевые адреса устройств. Перечень и описание атрибутов сущности представлены в таблице А.4.

Таблица А.4 – Описание атрибутов сущности net_address

Наименование атрибута	Тип данных	Описание атрибута
net_addres_uid	целочисленное десятичное число	Идентификационный номер. Поле является первичным ключом
value	строка символов	Значение сетевого адреса
priority	целочисленное десятичное число	Приоритет сетевого адреса
net_address_type_uid	целочисленное десятичное число	Тип сетевого адреса. Поле является внешним ключом
device_uid	строка символов	Идентификатор УСПД. поле является внешним ключом

Сущность income_message хранит в себе информацию о входящих сообщениях, а сущность outcome_message – информацию об исходящих сообщениях. Обе сущности имеют однотипную структуру, которая перечислена в таблице А.5.

Таблица А.5 – Описание атрибутов сущностей income_message и outcome_message

Наименование атрибута	Тип данных	Описание атрибута
uid	целочисленное десятичное число	Идентификационный номер сообщения. Поле является первичным ключом
text	строка символов	Содержимое сообщения

Продолжение таблицы А.5

Наименование атрибута	Тип данных	Описание атрибута
received_status	логический	Статус отправки сообщения (в качестве значения может принимать либо true, либо false)
processed_status	логический	Статус обработки сообщения (в качестве значения может принимать либо true, либо false)
device_uid	строка символов	Идентификационный номер УСПД. поле является внешним ключом
error	строка символов	Сообщение об ошибке. Поле может быть нулевым, если сообщение обработалось без ошибок

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг программы

В настоящем приложении представлено детальное описание классов сервера сбора данных в виде листингов программы.

Б.1 Описание класса Task

```
/*================================================================
=
 Класс Task описывает задачу и предоставляет методы для работы
 в ней.

 Атрибуты класса:
 1) cronjob - строка, содержащая cron-выражение задачи;
 2) parameters - параметры, необходимые для выполнения задачи;
 3) status - статус выполнения задачи;
 4) uid - идентификационный номер задачи в ВБД;
 5) model - модель таблицы задач в ВБД;
 6) query - объект, предназначенный для запросов на выборку из
ВБД;
 7) deviceUid - идентификатор устройства, для которого
                предназначена задача;
 8) file - переменная для работы с файлами.

 Методы класса:
 1) Task() - конструктор по умолчанию;
 2) Task ( Task const & obj) - конструктор копирования;
 3) ~Task() - длеструктор;
 4) void setStatus(int st) - метод устанавливает статус задачи;
 5) void setUid(int i) - метод устанавливает уникальный
                         идентификатор задачи;
 6) void setCronjob(QString cron) - метод устанавливает cron-
                                    выражение для задачи;
 7) void setParameters(QString params) - метод устанавливает
                                         параметры для выполнения
                                         задачи;
 8) void setDeviceUid(QString uid) - метод устанавливает значение
                                     идентификатора УСПД;
 9) int getUid() - метод возвращает уникальный идентификатор
                   задачи из ВБД;
 10) QList <Task> getTaskList() - метод формирует и возвращает
                                  список задач;
 11) QString getCronjob() - метод возвращает текущее
                            cron-выражение задачи;
 12) QString getParameters() - метод возвращает текущие параметры
                               задачи;
 13) QString getDeviceUid() -  метод возвращает идентификатор
УСПД,
```

```
                                    для которого предназначена задача;
 14) bool isEmpty() - метод проверяет, пуста ли таблица задач в
                      ВБД. Если таблица пуста, мтеод возвращает
                      true, в противном случае метод возвращает
                      false;
 15) void save() - метод сохраняет текущую задачу в ВБД;
 16) void remove() - метод удаляет текущую задачу из ВБД;
 17) void edit(int taskStatus) - метод редактирует информацию о
                                 статусе задачи в ВБД;
 18) void print() - метод выводит информацию о текущей
                    задаче в консоль приложения;
 19) Task & operator = (Task const & obj) - метод перегружает
                                            операцию присваивания;
 20) bool operator ==
    (const Task & right) - метод перегружает операцию сравнения.
                           Если один объект данного класса равен
                           другому объекту этого же класса, метод
                           возвращает true, в противном случае
                           метод возвращает false;
 21) bool operator !=
     (const Task & right) - метод перегружает операцию сравнения.
                            Если один объект данного класса не
                            равен другому объекту этого же класса,
                            метод возвращает true, в противном
                            случае метод возвращает false ;
 22) void copy(const Task &obj) - метод копирует данные из объекта
                                  obj в текущий объект;
 23) void toLogFile(QString message - метод записывает информацию
                                      о событиях класса в файл.
 Сигналы класса:
 1) void sig_done(Task) - сигнал, оповещающий о том, что задача
                          выполнена.

==========================================================================*/
class Task : public QObject
{
    Q_OBJECT
public:
    explicit Task(QObject *parent = 0);
    Task ( Task const & obj, QObject *parent = 0 );
    ~Task();

    void            setStatus(int st);
    void            setUid(int i);
    void            setCronjob(QString cron);
    void            setParameters(QString params);
    void            setDeviceUid(QString uid);

    int             getUid();
    QList <Task>    getTaskList();
    QString         getCronjob();
```

```
    QString         getParameters();
    QString         getDeviceUid();

    bool            isEmpty();

    void            save();
    void            remove();
    void            edit(int taskStatus);

    void            print();

    Task &          operator = (Task const & obj);
    bool            operator == (const Task & right);
    bool            operator != (const Task & right);
private:
    QString         cronjob,
                    parameters,
                    deviceUid;
    int             status,
                    uid;
    TaskModel       *model;
    Query           *query;
    QFile           file;

    void            copy(const Task &obj);
    void            toLogFile(QString message);

signals:
    void            sig_done(Task);

public slots:
     };
```

Б.2 Описание класса TaskManager

```
/*=============================================================
===========
 Класс TaskManager предназначен для управления задачами,
 поступающими из Центрального сервера.
 Основные функции класса:
    -обработка пакета задач, поступающего из ЦС;
    -формирование списка команд для УСПД;
    -управление сформированным списком команд;
    -составление расписания выполнения команд;
    -вызов команд в соответствии с расписанием.

 атрибуты класса:
    taskList - список задач;
    sheduler - объект планировщика задач;
    limit - предельное значение на одновременную
            обработку задач
```

```
    timer - перечень таймеров для списка задач
    file - переменная для работы с файлами

 методы класса:
void addTask(Task &task) - метод добавляет новую задачу в список;
void removeTask(int index)  - метод удаляет задачу из списка по
                              индексу;
void removeTask(Task task) - метод удаляет задачу из списка по
                             содержимому;
void clear() - метод очищает перечень задач;
void print() - метод отображает содержимое списка задач в консоль;
void run() - метод запускает обработчик задач;
void stop() - метод останавливает обработку задач;
void handleTask() - метод запускает задачу в обработку;
void updateTask (int index, int status = 0) - метод изменяет
статус
                                                задачи;
void setConfig() - метод устанавливает параметры обработчика задач
                   из конфигурационного файла;
void startTimer() - метод запускает таймеры для задач;
void toLogFile(QString message) - метод записывает в файл события
                                  класса

сигналы класса:
    sig_callTask(QString deviceUid) - сигнал высылается для
задачи,
                                      которую необходимо
запустить.
слоты класса:
    void slt_Run ( int index) -
    void slt_OnDone (Task task) - метод-слот, реагирущий на сигнал
о
                                  выполнении задачи
    void slt_taskDone(int index) - метод-слот, устанавливающий
задачу с
                                   индексом index в состояние
"DONE".
=================================================================
=========*/

class TaskManager : public QObject
{
    Q_OBJECT
public:
    explicit TaskManager ( QObject *parent = 0 );
    ~TaskManager();

    void                addTask (Task &task);
    void                removeTask (int index);
    void                removeTask (Task task);
    void                clear ();
```

```
    void                print ();
    void                run ();
    void                stop();

private:
    QVector <Timer *>   timer;
    QList <Task>        taskList;
    int                 limit;
    QFile               file;

    void                updateTask (int index, int status = 0);
    void                handleTask();
    void                setConfig();
    void                startTimer();
    void                toLogFile(QString message);

signals:
    void                sig_callTask(QString deviceUid);
public slots:
    void                slt_Run ( int index);
    void                slt_OnDone (Task task);
    void                slt_taskDone(int index);
};
```

Б.3 Описание класса CronParser

```
/*=================================================================
==========================
 Класс CronParser выполняет следующие задачи:
   - обработка строки, содержащей cron-выражение задачи
   - вычисление ближайшей даты вызова текущей задачи

 Атрибуты класса:
 1) cronJob   - строка, содержащая cron-выражение;
 2) cronDate  - дата вызова функции;
 3) minute, hour, dayOfMonth, month,
    year, dayOfWeek - пары значений, содержащие информацию об
отдельной части
                   даты и времени (минута, час, день месяца,
месяц, год,
                   день недели соответственно). Первый элемент
пары -
                   значение части даты и времени, второе
значение - флаг
                   переполнения разряда текущей части даты и
времени;
 4) isCalled   - флаг, показывающий, что объект данного класса уже
был вызван;
 5) file - переменная для работы с файлами.

 Методы класса:
```

```
 1) CronParser(QObject *parent = 0)                                -
конструктор класса по умолчанию;
 2) CronParser ( CronParser & obj, QObject * parent = 0 )  -
конструктор копирования;
 3) ~CronParser() - деструктор класса
 4) QString getCronJob()                                           - метод
возвращает текущее
                                                                     cron-
выражение;
 5) QDateTime getDateTime(QString cron)                            - метод
возвращает время ближайшего

вызова функции;
 6) QPair<int, bool> parse
   (QString cronJob, int var, int minLimit, int maxLimit)  - метод
обрабатывает cron-выражение
                                                                     и
возвращает значение

срабатывания для каждой единицы

времени;
 7) QDateTime calcTaskDate()                                       - метод
парсит cron-выражение и

рассчитывает ближайшую дату

вызова функции;
 8) void isSingle(bool singleShot)                                 -
сигнал, оповещающий о том, что

данная задача является

синглшотом;
 9)void  toLogFile(QString string) - метод предназначен для записи
событий класса в файл.

========================================================================
======================== */
class CronParser : public QObject
{
    Q_OBJECT

public:
    explicit CronParser(QObject *parent = 0);
    CronParser (CronParser & obj, QObject * parent = 0);
    ~CronParser();

    QString             getCronJob();
    QDateTime           getDateTime(QString cron);
    void                setCall(bool var);
```

```
    CronParser &        operator = ( CronParser const & obj );

private:
    QString             cronJob;
    QDateTime           cronDate;
    QPair<int, bool>    minute,
                        hour,
                        dayOfMonth,
                         month,
                        year,
                        dayOfWeek;
    bool                isCalled;
    QFile               file;

    QPair<int, bool>    parse(QString cronJob, int var, int
minLimit, int maxLimit);
    QDateTime           calcTaskDate();
    void                toLogFile(QString string);
signals:
    void                isSingle(bool singleShot);

public slots:
};
```

Б.4 Описание класса Timer

```
/*=============================================================
 Класс Timer выполняет следующие функции:
 -обрабатывает cron-выражение задачи посредством объекта
  класса CronParser и получает время ближайшего выполнения
  задачи;
 -отслеживает время начала выполнения задачи

 Атрибуты класса:
 1) parser - обработчик cron-выражений;
 2) cronJob - cron-выражение;
 3) taskIndex - индекс задачи;
 4) timerStart - время срабатывания таймера;
 5) singShot - флаг сингглшота задачи;
 6) nextExec - время следующего срабатывания;
 7) mtxNextExec - мьтекс для корректного срабатывания таймера;

 Методы класса:
 1) Timer() - конструктор по умолчанию;
 2) Timer (Timer const & obj) - конструктор копирования;
 3) ~Timer() - деструктор;
 4) Timer & operator = (Timer const & obj) - перегрузка
                                             операции
                                             присваивания;
 5) void timerEvent(QTimerEvent *event) - метод по обработке
                                          событий таймера;
```

```
 6) time_t calcDiffTime() - метод осуществляет подсчет
                            разницы времени;
 Сигналы класса:
 1) void timeout(int index) - сигнал о том, что время таймера
                              истекло;
2) void done(int index) - сигнал о том, что задача выполнена

 Слоты класса:
1) void start(QString cron, int index) - метод осуществляет
                                         запуск таймера;
 2) void stop() - метод осуществляет остановку таймера;
 3) void stop() - метод устанавливает переменную singShot в
                  нужное состояние.
 ============================================================*/
class Timer : public QObject
{
    Q_OBJECT

public:
    explicit Timer(QObject *parent = 0);
    Timer ( Timer const & obj, QObject *parent = 0 );
    ~Timer();


    Timer &     operator = (Timer const & obj);

private:
    CronParser parser;
    QString cronJob;
    int taskIndex;
    int timerStart = 0;
    bool singShot = false;
    time_t nextExec = -1;
    std::mutex mtxNextExec;

    void timerEvent(QTimerEvent *event) override;
    time_t calcDiffTime();

signals:
    void timeout(int index);
    void done(int index);

public slots:
    void start(QString cron, int index);
    void stop();
    void setSingleShot(bool singleShot);
};
```

Б.5 Описание класса Model

```
/*============================================================
 Класс Model прендназначен для взаимодействия с ВБД. объект
 класса представляет собой модель одной записи из таблицы.
 Класс выполняет следующие типы запросов:
 - запрос на добавление записи;
 - запрос на изменение записи;
 - запрос на удаление записи.

 Атрибуты класса:
 record - запись таблицы. Атрибут представляет собой карту,
          хранящую имя поля и значение поля записи;
 isNew - флаг, отображающий является ли запись новой или они
         уже существует в таблице;
 database - объект по работе с базами данных;
 schema - схема таблицы, для которой создается модель
 relation - структура, хранящая в себе информацию об отношениях
            с другими таблицами;
 file - переменная для работы с файлами.

 Методы класса:
 Model(QObject *parent = 0) - конструктор по умолчанию;
 Model(Model const & obj, QObject *parent = 0) - конструктор
                                                 копирования;
 ~Model() - деструктор;
 Model & operator = (Model const & obj) - метод перегружает
                                          оператор присваивания
                                          для объекта класса;
 Model * operator = (Model const * obj) - метод перегружает
                                          оператор присваивания
                                          для указателя на объект
                                          класса;
 bool operator == (const Model & right) - метод перегружает
                                          оператор сравнения;
 bool operator != (const Model & right) - метод перегружает
                                          оператор сравнения;
 void setField
(QString fieldName, QVariant value) - метод устанавливает имя
                                      поля записи и значение
                                      этого поля;
 void setSchema
(TableSchema * tableSschema) - метод устанавливает схему
                               таблицы для созданной модели;
 void setRelation
(Model *outModel, QString relationName) - метод устанавливает
                                          отношения с другими
                                          таблицами для текущей
                                          записи;
 void etRelationData
```

```
(QString relationName, QList<Model *> model) - метод устанавливает
                                                перечень значений
из
                                                таблицы, связанной
с
                                                текущей моделью по
                                                конкретному
отношению;
void setIsNew(bool value) - метод устанавливает влаг isNew;
TableSchema * getSchema() - метод возвращает схему таблицы
                            для текущей модели;
bool getIsNew() - метод возвращает значение флага isNew;
QStringList getSelectedFields
(QStringList fieldList) - метод возаращает перечень полей модели,
                          предназначенных для выборки по запросу;
QVariant getRecord(QString fieldName) - метод возвращает значение
                                        поля по его названию;
QMap <QString, QVariant> getRecord() - метод возвращает запись
                                       текущей модели целиком;
QStringList getFields() - метод возвращает перечень полей
                          текущей модели;
QList <Model *> getRelationData
(QString relationName) - метод возвращает данные по конкретному
                         отношению для текущей модели;
bool save(QStringList fields = {}) - метод сохраняет запись
                                     в таблицу;
bool remove() - метод удаляет запись из таблицы;
void print() - метод выводит в консоль данные о текущей модели;
void printRelation() - метод выводит в консоль данные по
                       отношениям текущей модели;
virtual void copy(const Model & obj) - метод копирует данные
                                       из объекта obj в текущий
                                       объект;
void connectToDB() - метод осуществляет соединение с ВБД;
bool insert() - метод добавляет запись в таблицу;
bool update(QStringList fields = {}) - метод изменяет запись
                                       в таблице;
bool execQuery(QString sql) - метод запускает SQL-запросы на
                              выполнение;
String getString(QVariant value) - вспомогательный метод,
                                    предназначенный для
преобразования
                                    строк при генерации SQL-
запросов;
void toLogFile(QString message) - метод сохраняет в файл сведения
                                  о событиях класса.
==========================================================*/
class Model : public QObject
{
    Q_OBJECT
public:
```

```
    explicit Model(QObject *parent = 0);
    Model(Model const & obj, QObject *parent = 0);
    ~Model();

    Model &                         operator = (Model const & obj);
    Model *                         operator = (Model const * obj);
    bool                            operator == (const Model & right);
    bool                            operator != (const Model & right);

    void                            setField(QString fieldName,
QVariant value);
    void                            setSchema(TableSchema *
tableSschema);
    void                            setRelation(Model *outModel,
QString relationName);
    void                            setRelationData(QString
relationName, QList<Model *> model);
    void                            setIsNew(bool value);

    TableSchema *                   getSchema();
    bool                            getIsNew();
    QStringList                     getSelectedFields(QStringList
fieldList);
    QVariant                        getRecord(QString fieldName);
    QMap <QString, QVariant>        getRecord();
    QStringList                     getFields();
    QList <Model *>                 getRelationData(QString
relationName);

    bool                            save(QStringList fields = {});
    bool                            remove();
    void                            print();
    void                            printRelation();

protected:
    QMap <QString, QVariant>        record;
    bool                            isNew;
    QSqlDatabase                    database;
    TableSchema                     *schema;
    relation_s                      relation;
    QFile                           file;
    virtual void                    copy(const Model & obj);
    void                            connectToDB();
    bool                            insert();
    bool                            update(QStringList fields = {});
    bool                            execQuery(QString sql);
    QString                         getString(QVariant value);
    void                            toLogFile(QString message);
signals:
public slots:
};
```

Б.6 Описание класса Query

```
/*=============================================================
 Класс Query предназначен для формирования запросов на выборку:
 как простых, так и сложных.

 Атрибуты класса:
 queryModel - модель таблцы, с которой работает текущий объект;
 where - строка, хранящая в себе набор условий для SQL-запроса;
 order - строка, хранящая в себе информацию об упорядочивании
         выборки по SQL-запросу;
  selectedFields - строка, хранящая в себе информацию о полях,
                   по которым нужно предоставить выборку;
  limit - строка, хранящая данные об ограницении на количество
          записей по выборке;
  countString - строка, хранящая в себе информацию для подсчета
                количества конкретных записей;
  outerJoin - строка, хранящая в себе информацию по запросу типа
              LEFT OUTER JOIN$
  innerJoin - строка, хранящая в себе информацию по запросу типа
              INNER JOIN;
  sqlString - строка, содержащая в себе SQL-запрос на выборку;
  rName - имя отношения;
  schema - схема таблицы для текущего объекта;
  database - объект по работе с базами данных;
  selectedList - перечень полей, предназначенных для выборки;
  file - переменная для работы с файлами.

  Методы класса:
  Query(TableSchema *tableSchema, QObject *parent = 0) -
конструктор
                                                          класса;
  Query(Query const & obj, QObject *parent = 0) - конструктор
копирования;
  ~Query() - деструктор класса;
  void setWhere(QString fieldName, QVariant value ) - метод
устанавливает
                                                    условие для
SQL-запроса;
  void setOrder(QString fieldName) - метод устанавливает
упорядочивание выборки
                                       по SQL-запросу;
  void setSelectedFields(QStringList fields) - метод устанавливает
выборанные
                                              для SQL-запроса
поля;
  void setLimit( QString str ) - метод устанавливает ограничение
на количество
                                    выборанных полей по SQL-запросу;
  void withOuter(QString relationName) - метод устанавливает
параметры для
```

запроса типа LEFT OUTER JOIN;
  void withInner(QString relationName) - метод устанавливает параметры для
запроса типа INNER JOIN;
  void updateSchema(Model* model) - метод обновляет схему модели для текущего
объекта;
  Model * getOne() - метод возвращает одну запись по текущему запросу;
  Model * getEmpty() - метод возвращает пустую модель;
  QList <Model *> getAll() - метод возхвращает все записи по текущему запросу;
  int count(QString fieldName = "*") - метод возвращает количество запрошенных
записей;
  QVariant getlastID() - метод возаращает последний идентификатор, добавленный
в ВБД;
  Query & operator = (Query const & obj) - метод перегружает операцию
присваивания;
  bool operator == (const Query & right) - метод перегружает операцию сравнения;
  bool operator != (const Query & right) - метод перегружает операцию сравнения;
  void clear() - метод очищает все записи объекта;
  void copy(Query const & obj) - метод копирует все данные из объекта obj в
текущий бъект;
  void connectToDB() - метод осуществляет подключение к ВБД;
  QString generateSQL() - метод генерирует строку SQL-запроса;
  matrix_t requestMany(QString sql) - метод возвращает данные обо всех записях
по SQL-запросу;
  list_t requestOne(QString sql) - метод возвращает данные об одной записи по
сгенерированному SQL-запросу;
  Model * appendRecords(list_t values, Model *& model) - метод формирует
модель;
  relation_data_t getRelationModel
 (list_t values, Model * & model) - метод возвращает данные по моделям, связанным
с текущей моделью;
  QString listToString(QStringList list) - метод преобразовывает перечень значений
в строку;
  void toLogFile(QString message) - метод записывает в файл информацию о событиях
класса.

```
==========================================================*/
class Query : public QObject
{
    Q_OBJECT
public:
    explicit Query(TableSchema *tableSchema, QObject *parent = 0);
    Query(Query const & obj, QObject *parent = 0);
    ~Query();

    void                setWhere(QString fieldName, QVariant value
);
    void                setOrder(QString fieldName );
    void                setSelectedFields(QStringList fields );
    void                setLimit( QString str );
    void                withOuter(QString relationName);
    void                withInner(QString relationName);

    void                updateSchema(Model* model);

    Model *             getOne();
    Model *             getEmpty();
    QList <Model *>     getAll();
    int                 count(QString fieldName = "*");
    QVariant            getlastID();

    Query &             operator = (Query const & obj);
    bool                operator == (const Query & right);
    bool                operator != (const Query & right);

    void                clear();

private:
    Model *             queryModel;
    QString             where,
                        order,
                        selectedFields,
                        limit,
                        countString,
                        outerJoin,
                        innerJoin,
                        sqlString,
                        rName;
    TableSchema *       schema;
    QSqlDatabase        database;

    QStringList         selectedList;

    QFile               file;

    void                copy(Query const & obj);
    void                connectToDB();
```

```
    QString             generateSQL();
    matrix_t            requestMany(QString sql);
    list_t              requestOne(QString sql);
    Model *             appendRecords(list_t values, Model *&
model);
    relation_data_t     getRelationModel(list_t values, Model * &
model);
    QString             listToString(QStringList list);
    void                toLogFile(QString message);
signals:
public slots:
};
```

Б.7 Описание класса TableSchema

```
      /*=========================================================
==============
       Класс TableSchema представляет собой схему для таблицы БД и
содержит
       методы и атрибуты по работе с ней.

       Атрибуты класса:
       file - переменная для работы с файлами;
       fields - перечень полей таблицы;
       primaryKey - перечень первичных ключей таблицы;
       foreignKey - перечень внешних ключей таблицы;
       tableName - имя таблицы;
       fieldsCount - количество полей тблицы;

       relations - перечень отношений с другими таблицами;
       relationData - данные по отношениям с другими таблицами.

       Методы класса:
       TableSchema(QObject *parent = 0) - конструкто по умолчанию;
       TableSchema(TableSchema const & obj, QObject *parent = 0) -
конструтор

копирования;
       ~TableSchema() - деструктор класса;
       void setFields(QStringList fieldsList) - метод
устанавливает перечень полей
                                                таблицы;
       void setPrimaryKey(QStringList pkList) - метод утснавливает
перечень
                                                первичных ключей
таблицы;
       void setForeignKey(QStringList fkList) - метод
устанавливает перечень
                                                внешних ключей
таблицы;
```

void setTableName(QString tName) - метод устанавливает имя таблицы;
void setRelations(QString name, relation_s relation) - метод устанавливает отношения с другими таблицами;
void setRelationData(relation_data_t data) - метод устанавливает данные по каждому отношению;
QStringList getFields() - метод возвращает перечень полей таблицы;
QStringList getPrimaryKey() - метод возвращает перечень первичных ключей таблицы;
QStringList getForeignKey() - метод возвращает перечень внешних ключей таблицы;
QString getTableName() - метод возвращает имя таблицы;
int getFieldsCount() - метод возвращает количество полей в таблице;
relation_s getRelation(QString key) - метод возвращает отношение с другой таблицей по ключу;
relation_data_t getRelationData() - метод вохвращает данные об отношениях с другими таблицами;
bool checkField(QString fieldName) - метод проверяет наличие поля fieldName в таблице;
TableSchema & operator = (TableSchema const & obj) - метод перегружает операцию присваивания для объекта;
TableSchema * operator = (TableSchema const * obj) - метод перегружает операцию присваивания для указателя на объект;
bool operator == (const TableSchema & right) - метод перегружает операцию сравнения для объекта класса;
bool operator == (const TableSchema * right) - метод перегружает операцию

```
                                                                сравнения
для указателя на объект
                                                                класса;
       bool operator != (const TableSchema & right) - метод
перегружает операцию
                                                                сравнения
для объекта класса;
       bool operator != (const TableSchema * right) - метод
перегружает операцию
                                                                сравнения
для указателя на объект
                                                                класса;
       void copy(TableSchema const & obj) - метод копирует
информацию из объекта obj в
                                                    текущий объект;
       void toLogFile(QString message) - метод сохраняет
информацию о событиях класса
                                                в файл.
      ============================================================
============*/
      class TableSchema : public QObject
      {
          Q_OBJECT
      public:
          explicit TableSchema(QObject *parent = 0);
          TableSchema(TableSchema const & obj, QObject *parent =
0);
          ~TableSchema();

          void            setFields(QStringList fieldsList);
          void            setPrimaryKey(QStringList pkList);
          void            setForeignKey(QStringList fkList);
          void            setTableName(QString tName);
          void            setRelations(QString name, relation_s
relation);

          void            setRelationData(relation_data_t data);

          QStringList     getFields();

          QStringList     getPrimaryKey();
          QStringList     getForeignKey();
          QString         getTableName();
          int             getFieldsCount();
          relation_s      getRelation(QString key);
          relation_data_t getRelationData();

          bool            checkField(QString fieldName);

          TableSchema &   operator = (TableSchema const & obj);
          TableSchema *   operator = (TableSchema const * obj);
```

```
        bool            operator == (const TableSchema & right);
        bool            operator == (const TableSchema * right);
        bool            operator != (const TableSchema & right);
        bool            operator != (const TableSchema * right);

    private:
        QFile           file;
        QStringList     fields,
                        primaryKey,
                        foreignKey;
        QString         tableName;
        int             fieldsCount;

        QMap <QString,
        relation_s>     relations;

        relation_data_t relationData;

        void            copy(TableSchema const & obj);
        void            toLogFile(QString message);
    signals:

    public slots:
    };
```

Б.8 Описание класса Message

```
/*=======================================================
 Класс Message предназначен для работы с сообщениями
 ПО ССД. Класс унаследован от класса Model

 Атрибуты класса:
 received - флаг, сообщающий о факте доставки сообщения;
 processed - флаг, сообщающий о факте обработки сообщения;
 errorText - строка, содержащая соощение об ошибке,
             возникшей при передаче сообщения.

 Методы класса:
 Message(Model *parent = 0) - конструктор по умолчанию;
 ~Message() - деструктор класса;
 void setReceived(bool status) - метод устанавливает
                                 статус отправки сообщения;
 void setProcessed(bool status) - метод устанавливает
                                  статус обработки сообщения;
 void setErrorText(QString text) - метод устанавливает текст
                                   строки об ошибке передачи
                                   сообщения;
 template <class T>
 QJsonDocument getJson
(QList <T*> modelList) - метод возвращает заданную таблицу
                         в фаормате JSON-документа;
```

```
==========================================================*/
class Message : public Model
{
public:
    explicit Message(Model *parent = 0);
    ~Message();
protected:
    bool    received,
            processed;
    QString errorText;

    void    setReceived(bool status);
    void    setProcessed(bool status);
    void    setErrorText(QString text);

    template <class T>
    QJsonDocument   getJson(QList <T*> modelList);
};
```

Б.9 Описание класса Request

```
/*============================================================
=
 Класс Request предназначен для отправки и получения сообщений
 по HTTP-протоколу

 Атрибуты класса:
 tabName - имя таблицы
 addr - адрес устройства
 url - строка, содержащая url-адреса
 part1, part2 - вспомогательные строки для формирования URL-адреса
 errorString - строка, содержащая текст ошибки передачи данных;
 file - переменная для работы с файлами.

 Методы класса:
 Request(QString tableName = "", QObject *parent = 0) -
конструктор;
 ~Request() - деструктор;
 bool post(QJsonDocument command) метод реализует отправку
сообщения
                                    в формате QJsonDocument;
 QJsonDocument get() - метод получает сообщения и возвращает их в
                       формате QJsonDocument;
 void setTablename(QString tableName) - метод устанавливает имя
                                        таблицы ВБД;
 void setAddress(QString address) - метод устанавливает адрес
                                    получателя сообщения;
 QString getErrorString() - метод устанавливает текст ошибки
передачи
                            сообщения;
 QString getTableName(QString tableName) - метод возвращает имя
```

```
 QString getUrl() - метод возвращает URL-адрес получателя
сообщения;
 void toLogFile(QString message) - метод сохраняет в файл события
                                   класса.
=============================================================*
/
class Request : public QObject
{
    Q_OBJECT
public:
    explicit Request(QString tableName = "", QObject *parent = 0);
    ~Request();

    bool            post(QJsonDocument command);
    QJsonDocument   get();
    void            setTablename(QString tableName);
    void            setAddress(QString address);
    QString         getErrorString();
private:
    QString         tabName,
                    addr,
                    url,
                    part1,
                    part2,
                    errorString;
    QFile           file;

    QString         getTableName(QString tableName);
    QString         getUrl();
    void            toLogFile(QString message);
signals:

public slots:
};
```

Б.10 Описание класса HTTPServer

```
/*============================================================
 Класс HTTPServer поддерживает сетевое соединение с ЦС и УСПД

 Атрибуты класса:
 manager - объект для себевого взаимодействия по HTTP-протоколу;
 server - объект, поддерживающий сервер, использующий
          TCP-протокол;
 file - переменная для работы с фалами.

 Методы класса:
 HTTPServer(QObject *parent = 0) - конструктор класса;
 ~HTTPServer() - деструктор класса;
 void start() - метод запускает HTTP-сервер;
 QJsonDocument toJsonDocument(QString command) - метод формирует
```

```
                                                    JSON-документ для
                                                    управляющей
команды;
 Сигналы клкасса:
 void sig_dataSaved() - сигнал, опоыещающий о том, что запись
                        данных в ВБД завершена;

 Слоты класса:
 void slt_ConnectionHandler() - слот вызывается при появлении
                                нового подключения к серверу
============================================================*/
class HTTPServer : public QObject
{
    Q_OBJECT
public:
    explicit HTTPServer(QObject *parent = 0);
    ~HTTPServer();
    void                    start();
    QJsonDocument           toJsonDocument(QString command);
private:
    QNetworkAccessManager*  manager;
    QTcpServer              server;
    QFile                   file;

    void                    toLogFile(QString string);

signals:
    void                    sig_dataSaved();
public slots:
    void                    slt_ConnectionHandler();
};
```

Б.11 Описание класса DCServer

```
/*=========================================================
 Класс DCServer осуществляет управление ПО ССД

 Атрибуты класса:
 taskMan - обработчик задач;
 server - сервер, осуществлящий сетевое взаимодействие
          с другими компонентами системы.

 Методы класса:
 DCServer(QObject *parent = 0) - конструктор класса;
 ~DCServer() - деструктор класса;
 void startServer() - метод запускает сервер сбра данных;

 Слоты класса:
 void startTaskManager() - слот запускает обработчик задач;
 void sendRequest(QString address) - слот отправляет
                                     сообщение на устройство
```

```
                                        с адресом address
                                        посредством HTTP-сервера.
=============================================================*/
class DCServer : public QObject
{
    Q_OBJECT
public:
    explicit DCServer(QObject *parent = 0);
    ~DCServer();
    void            startServer();

private:
    TaskManager         *taskMan;
    HTTPServer          *server;

signals:

public slots:
    void startTaskManager();
    void sendRequest(QString address);
};
```

Б.12 Реализация наиболее важных методов класса TaskManager

```
/*=========================================================
метод запускает обработчик задач
=========================================================*/
void TaskManager::run()
{
    setConfig();
    Task task;
    int size = task.getTaskList().size();
    try
    {
        if(size <= limit)
        {
            if(!task.isEmpty())
            {
                taskList.append(task.getTaskList());
            }
        }
        else
        {
            throw size;
        }
    }
    catch(int)
    {
        QString message = "[!][TASK MANAGER]:\tError! Task list
stack overflow!";
        qDebug() << message;
```

```
        toLogFile(message);
    }

    try
    {
        if ( !taskList.empty() )
        {
            for ( int i = 0; i < taskList.size(); i++ )
            {
                QObject::connect( &taskList[i],
SIGNAL(sig_done(Task)), this, SLOT(slt_OnDone(Task)) );
            }

            handleTask();
        }
        else
        {
            throw taskList;
        }
    }
    catch ( QList <Task> )
    {
        QString message = "[!][TASK MANAGER]:\ttaskList is
empty!";
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
метод загружает параметры обработчика задач из
конфигурационного файла
=========================================================*/
void TaskManager::setConfig()
{
    QSettings set ("config.cfg", QSettings::IniFormat);
    set.beginGroup("task_param");
    limit = set.value("limit").toInt();
    set.endGroup();
}

/*=========================================================
метод удаляет задачу из списка по ее порядковому номеру

параметры метода:
    int index - порядковый номер задачи в списке
=========================================================*/
void TaskManager::removeTask ( int index )
{
    taskList[index].remove();
```

```
    QString message = "[TASK MANAGER]:\ttimer " +
QString::number(index) + " Stopped";
    qDebug() << message;
    toLogFile(message);

    disconnect(timer[index], SIGNAL(timeout(int)), this,
SLOT(slt_Reaction(int)));
    timer.remove(index, 1);
    taskList.removeAt(index);
}

/*=========================================================
метод удаляет задачу из списка по ее содержимому

параметры метода:
    Task task - конкретная задача
=========================================================*/
void TaskManager::removeTask ( Task task )
{
    for ( int i = 0; i < taskList.size(); i++ )
    {
        if ( taskList[i] == task )
        {
            taskList[i].remove();
            disconnect(timer[i], SIGNAL(timeout(int)), this,
SLOT(slt_Reaction(int)));

            QString message = "[TASK MANAGER]:\ttimer " +
QString::number(i) + " Stopped";
            qDebug() << message;
            toLogFile(message);

            timer.remove(i, 1);
            taskList.removeAt(i);
            break;
        }
    }
}

/*=========================================================
метод запускает процесс обработки задачи
=========================================================*/
void TaskManager::handleTask ()
{
    for ( int i = 0; i < taskList.size(); i++ )
    {
        taskList[i].setStatus(PROCESSING);
        updateTask(i, PROCESSING);
    }
    startTimer();
}
```

```
/*=======================================================
метод-слот, запускающий  задачу на выполнение

параметры метода:
    int index - порядковый номер задачи в списке
=======================================================*/
void TaskManager::slt_Run (int index)
{
    QString message = "[TASK MANAGER][";
    message += QDateTime::currentDateTime().toString("dd/MM/yy
hh:mm:ss");
    message += "]:\tCall device " +
taskList[index].getDeviceUid();
    qDebug() << message;
    toLogFile(message);

    emit sig_callTask(taskList[index].getDeviceUid());
}

/*=======================================================
метод запускает таймер для каждой задачи
=======================================================*/
void TaskManager::startTimer()
{
    QString message = "[TASK MANAGER]:\tStarting scheduler...\n";
    qDebug() << message;
    toLogFile(message);

    for(int i=0; i<taskList.size(); i++)
    {
        Timer *t = new Timer();
        timer.append(t);
        connect (timer.at(i), SIGNAL(timeout(int)), this,
SLOT(slt_Run(int)));
        connect (timer.at(i), SIGNAL(done(int)), this,
SLOT(slt_taskDone(int)));
        timer[i]->start(taskList[i].getCronjob(), i);

        message.clear();
        message = "[TASK MANAGER]:\tTmer " + QString::number(i) +
" have cron " + taskList[i].getCronjob();
        qDebug() << message;
        toLogFile(message);
    }
}

/*=======================================================
метод останавливает планировщик задач
=======================================================*/
void TaskManager::stop()
{
```

```
    QString message = "[TASK MANAGER]:\tScheduler stopped\n";
    qDebug() << message;
    toLogFile(message);

    for( int i = 0; i < timer.size(); i++)
    {
        timer[i]->stop();
        disconnect(timer[i], SIGNAL(timeout(int)), this,
SLOT(slt_Run(int)));
    }
}
```

Б.13 Реализация наиболее важных методов класса CronParser

```
/*=========================================================
метод обрабатывает cron-выражение и рассчитывает
ближайшую дату вызова функции
=========================================================*/
QDateTime CronParser::calcTaskDate()
{
    QDateTime optimalDate;
    QStringList crons = cronJob.split(" ");
    try
    {
        if(crons.size() != 6)
        {
            throw crons.size();
        }
        else
        {
            int value;
            value = QTime::currentTime().minute();
            if(isCalled != true)
            {
                value += 1;
                setCall(false);
            }
            minute = parse(crons.at(0), value, 0, 59);

            value = QTime::currentTime().hour();
            if(minute.second != false)
            {
                value +=1;
            }
            hour = parse(crons.at(1), value, 0, 23);

            value = QDate::currentDate().day();
            if(hour.second != false)
            {
                value += 1;
            }
```

```
            dayOfMonth = parse(crons.at(2), value, 1,
QDate::currentDate().daysInMonth());

            value = QDate::currentDate().month();
            if(dayOfMonth.second != false)
            {
                value += 1;
            }
            month = parse(crons.at(3), value, 1, 12);

            value = QDate::currentDate().year();
            if(month.second != false)
            {
                value += 1;
            }
            year = parse(crons.at(5), value,
QDate::currentDate().year(), QDate::currentDate().year()+1);

            value = QDate::currentDate().dayOfWeek();
            dayOfWeek = parse(crons.at(4), value, 1, 7);

            if(dayOfWeek.first < value)
            {
                dayOfWeek.first = value - dayOfWeek.first;
            }
            else
            {
                if(dayOfWeek.first == value)
                {
                    dayOfWeek.first = 0;
                }
                else
                {
                    dayOfWeek.first = 7 - dayOfWeek.first + value;
                }
            }

            value = QDate::currentDate().day() + dayOfWeek.first;
            try
            {
                if(crons.at(2) == "*")
                {
                    dayOfMonth.first = value;
                }
                else
                {
                    if(crons.at(4) == "*")
                    {
                        dayOfMonth.first = value;
                    }
                    Else
```

```
                    {
                        if(dayOfMonth.first != value)
                        {
                            throw value;
                        }
                    }
                }
            }
            catch(int)
            {
                QString message =  "[!][TASK MANAGER]:\tError:
Invalid cronjob \"\". Day of week does not correspond with day of
month\n";
                qDebug() << message;
                toLogFile(message);
            }

            optimalDate.setDate(QDate(year.first, month.first,
dayOfMonth.first));
            optimalDate.setTime(QTime(hour.first, minute.first,
0));
        }
    }
    catch(int)
    {
        QString message = "[!][TASK MANAGER]:\tError: Invalid
cronjob \"" + cronJob + "\"\n";
        qDebug() << message;
        toLogFile(message);
    }

    if(optimalDate <= QDateTime::currentDateTime())
    {
        emit isSingle(true);
    }

    return optimalDate;
}

/*=======================================================
метод обрабатывает cron-выражение и возвращает
ближайшую к текущему единицу времени

Параметры метода:
QString cronJob - строка-cron-выражение;
int var - текущая единица времени;
int minLimit - минимальное предельное значение для
               текущей единицы времени;
int maxLimit - максимальное предельное значение для
               текущей единицы времени.
=======================================================*/
```

```
QPair<int, bool> CronParser::parse(QString cronJob, int var, int
minLimit, int maxLimit)
{
    QPair <int, bool> res;
    res.first = var;
    res.second = false;

    int step = 1;
    int start = minLimit;
    int finish = maxLimit;

    QRegExp comma("((\\d+)(,))+(\\d+)");
    QRegExp value("(\\d+)");
    QRegExp star("\\*");
    QRegExp starStep("(\\*)(\\/)(\\d+)");
    QRegExp startStep("(\\d+)(\\/)(\\d+)");
    QRegExp startFinish("(\\d+)(\\-)(\\d+)");
    QRegExp startFinishStep("(\\d+)(\\-)(\\d+)(\\/)(\\d+)");

    if(comma.exactMatch(cronJob))
    {
        QStringList result;
        result = cronJob.split(",");
        int i=0;
        bool f = false;
        while(i < result.size() && result.at(i).toInt() < var)
            i++;
        if(i == result.size())
        {
            i = 0;
            f = true;
        }
        try
        {
            if(result.at(i).toInt() < minLimit &&
result.at(i).toInt() > maxLimit)
            {
                throw result.at(i);
            }
            else
            {
                res.first = result.at(i).toInt();
                res.second = f;
            }
        }
        catch(QString)
        {
            QString message = "[!][TASK MANAGER]:\tError: Invalid
value in a cronjob \""+ cronJob;
            qDebug() << message;
            toLogFile(message);
```

```
            }
        }
        else if(value.exactMatch(cronJob))
        {
            try
            {
                if(cronJob.toInt() < minLimit || cronJob.toInt() >
maxLimit)
                {
                    throw cronJob.toInt();
                }
                else
                {
                    if(cronJob.toInt() < var)
                    {
                        res.second = true;
                    }
                    res.first = cronJob.toInt();
                }
            }
            catch(int)
            {
                QString message = "[!][TASK MANAGER]:\tError: Invalid
value in a cronjob \""+ cronJob;
                qDebug() << message;
                toLogFile(message);
            }
        }
        else
        {
            if(star.exactMatch(cronJob))
            {
                step = 1;
            }
            else
            {
                if(starStep.exactMatch(cronJob))
                {
                    step = cronJob.section("/", 1, 1).toInt();
                }
                else
                {
                    if(startStep.exactMatch(cronJob))
                    {
                        start = cronJob.section("/", 0, 0).toInt();
                        step = cronJob.section("/", 1, 1).toInt();
                    }
                    else
                    {
                        if(startFinish.exactMatch(cronJob))
                        {
```

```
                        start = cronJob.section("-", 0,
0).toInt();
                        finish = cronJob.section("-", 1,
1).toInt();
                    }
                    else
                    {
                        if(startFinishStep.exactMatch(cronJob))
                        {
                            step = cronJob.section("/", 1,
1).toInt();
                            start = cronJob.section("-", 0,
0).toInt();
                            QString tmp = cronJob.section("-", 1,
1);
                            tmp = tmp.section("/", 0, 0);
                            finish = tmp.toInt();
                        }
                        else
                        {
                            start = -1;
                        }
                    }
                }
            }
            try
            {
                if(start < minLimit || start > maxLimit)
                {
                    throw start;
                }
                else
                {
                    if(finish < minLimit || finish > maxLimit)
                    {
                        throw finish;
                    }
                    else
                    {
                        if(step < minLimit || step > maxLimit)
                        {
                            throw step;
                        }
                        else
                        {
                            int i = start;
                            while(i < var && i <= finish)
                            {
                                i += step;
                            }
```

```
                        if(i > finish)
                        {
                            res.first = start;
                            res.second = true;
                        }
                        else
                        {
                            res.first = i;
                        }
                    }
                }
            }
        }
        catch(int)
        {
            QString message = "[!][TASK MANAGER]:\tError: Invalid
value in a cronjob \""+ cronJob;
            qDebug() << message;
            toLogFile(message);
        }
    }
    return res;
}
```

Б.14 Реализация наиболее важных методов класса Timer

```
/*=======================================================
метод запускает таймер

Параметры метода:
QString cron - строка-cron_выражение
int index - порядковый индекс задачи в списке
=======================================================*/
void Timer::start(QString cron, int index)
{
    cronJob = cron;
    taskIndex = index;
    std::lock_guard<std::mutex> lock(mtxNextExec);
    nextExec = parser.getDateTime(cronJob).toTime_t();
    timerStart = startTimer(calcDiffTime());
    assert(timerStart);
}

/*=======================================================
метод останавливает таймер
=======================================================*/
void Timer::stop()
{
    std::lock_guard<std::mutex> lock(mtxNextExec);
    nextExec = -1;
```

```
    killTimer(timerStart);
}

/*=========================================================
метод обрабатывает события таймера

Параметры метода:
QTimerEvent *event - объект события таймера
=========================================================*/
void Timer::timerEvent(QTimerEvent *event)
{
    if(event->timerId() != timerStart)
        return;
    std::lock_guard<std::mutex> lock(mtxNextExec);
    killTimer(timerStart);
    timerStart = 0;
    if(nextExec > QDateTime::currentDateTime().toTime_t())
    {
        timerStart = startTimer(calcDiffTime());
        assert(timerStart);
    }
    else
    {
        emit timeout(taskIndex);
        if(!singShot)
        {
            nextExec = parser.getDateTime(cronJob).toTime_t();
            if(calcDiffTime() == 0)parser.setCall(true);
            timerStart = startTimer(calcDiffTime());
        }
        else
        {
            emit done(taskIndex);
        }
    }
}
/*=========================================================
метод вычисляет время ожидания до следующего вызова
функции
=========================================================*/
time_t Timer::calcDiffTime()
{
    time_t t =
QDateTime::currentDateTime().msecsTo(QDateTime::fromTime_t(0).addS
ecs(nextExec));
    if(t < 0)
        return 0;
    if(t > INT_MAX)
        return INT_MAX;
    return t;
}
```

Б.15 Реализация наиболее важных методов класса Task

```
/*=========================================================
метод сохраняет оъект класса в ВБД
=========================================================*/
void Task::save()
{
    model->setIsNew(true);
    model->setField("status", status);
    model->setField("cronjob", cronjob);
    model->setField("parameters", parameters);
    if(model->save())
    {
        QString message = "[TASK]:\tRecord is changed";
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
метод удаляет объект класса из ВБД
=========================================================*/
void Task::remove()
{
    if(model->remove())
    {
        QString message = "[TASK]:\tRecord deleted";
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
метод изменяет информацию об объекте в ВБД

Параметры метода:
int taskStatus - статус задачи
=========================================================*/
void Task::edit(int taskStatus)
{
    QStringList list = model->getFields();
    list.removeOne("uid");
    if((taskStatus != 0))
    {
        setStatus(taskStatus);
        model->setField("status", status);
        if(status == DONE)
        {
            emit sig_done(*this);
        }
    }
```

```
    if(model->save(list))
    {
        QString message = "[TASK]:\tRecord is changed";
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
метод извлекает из ВБД информацию о невыполненных
задачах, формирует список задач и возвращает его
=========================================================*/
QList<Task> Task::getTaskList()
{
    QList <Task> taskList;
    query->clear();
    QList <TaskModel *> list = TaskModel::toTaskModel(query-
>getAll());

    Query *tarQuery = new Query(TargetModel::getTableSchema());
    TargetModel tar = new TargetModel(tarQuery->getEmpty());
    model->setRelation(&tar, "task_to_target");
    query->updateSchema(model);
    QStringList selLlist =
tar.getSelectedFields(QStringList({"device_uid"}));
    query->setSelectedFields(selLlist);
    query->withInner("task_to_target");
    model = static_cast <TaskModel *> (query->getOne());
    QList <TargetModel *> deviceList =
TargetModel::toTargetModel(model-
>getRelationData("task_to_target"));
    query->clear();

    for(int i = 0; i < list.size(); i++)
    {
        Task t;
        t.setUid(list[i]->getRecord("uid").toInt());
        t.setCronjob(list[i]->getRecord("cronjob").toString());
        t.setStatus(list[i]->getRecord("status").toInt());

        t.query = query;
        t.model = list[i];
        t.setDeviceUid(deviceList[i]-
>getRecord("device_uid").toString());
        taskList.append(t);
    }
    return taskList;
}
```

Б.16 Реализация наиболее важных методов класса Request

```
/*=======================================================
метод отправляет сообщения получателю в формате JSON

Параметры метода:
QJsonDocument document - сообщение для получателя

ПРИМЕЧАНИЕ: в связи с тем, что на ЦС и на УСПД не
реализовано взаимодействия с ССД по приему сообьщений,
все сообщения передаются по localhost (строка 76)
=======================================================*/
bool Request::post(QJsonDocument document)
{
    QByteArray data = document.toJson();
    QNetworkAccessManager *manager = new
QNetworkAccessManager(this);
    QEventLoop eLoop;
    QNetworkRequest request(QUrl("http://localhost:8080/"));
    request.setHeader(QNetworkRequest::ContentLengthHeader,
data.count());
    request.setHeader(QNetworkRequest::ContentTypeHeader,
QString("application/json"));

    QNetworkReply *reply = manager->post(request, data);
    QObject::connect(manager, SIGNAL(finished(QNetworkReply*)),
&eLoop, SLOT(quit()));
    try
    {
        eLoop.exec();
        if(reply->error() != QNetworkReply::NoError)
        {
            throw reply;
        }
        else
        {
            QString message = "[REQUEST]:\tSent request " + data;
            qDebug() << message;
            toLogFile(message);
            return true;
        }
    }
    catch(QNetworkReply *)
    {
        errorString = reply->errorString();
        QString message = "[!][REQUEST]:\tERROR! " + errorString;
        qDebug() << message;
        toLogFile(message);
        return false;
    }
}
```

```
/*=======================================================
метод получает сообщения от отправителя и возвращает их
в JSON-формате для дальнейшей обработки
=======================================================*/
QJsonDocument Request::get()
{
    url = getUrl();
    QNetworkAccessManager *manager = new
QNetworkAccessManager(this);
    QNetworkReply *reply = manager-
>get(QNetworkRequest(QUrl(url)));
    QEventLoop   eLoop;
    QJsonDocument parser;
    QObject::connect(manager, SIGNAL(finished(QNetworkReply*)),
&eLoop, SLOT(quit()));
    try
    {
        eLoop.exec();
        if(reply->error() != QNetworkReply::NoError)
        {
            throw reply;
        }
        else
        {
            QString json = reply->readAll();
            parser = QJsonDocument::fromJson(json.toUtf8());
        }
    }
    catch(QNetworkReply *)
    {
        QString message = "[!][REQUEST]ERROR:\t" + reply-
>errorString();
        qDebug() << message;
        toLogFile(message);
    }
    return parser;
}

/*=======================================================
метод сохраняет в файл информацию о событиях ССД

Параметры метода:
QString message - сообщение о текущем событии
=======================================================*/
void Request::toLogFile(QString message)
{
    QString date = "["+
QDateTime::currentDateTime().toString("dd/MM/yy hh:mm:ss");
    date += "]:\t" + message;
    if(file.open(QIODevice::Append))
    {
```

```
        file.write(message.toUtf8());
    }
    file.close();
}
```

Б.17 Реализация наиболее важных методов класса HTTPServer

```
/*=======================================================
метод запускает HTTP-сервер
=======================================================*/
void HTTPServer::start()
{
    if(server.listen(QHostAddress::Any, 8080))
    {
        QString message = "[HTTPServer TEST:]\tServer is
started\n[HTTPServer TEST:]\tListening...";
        qDebug() << message;
        toLogFile(message);

        connect(&server, SIGNAL(newConnection()), this,
SLOT(slt_ConnectionHandler()));

        Request * r = new Request();
        QJsonDocument data = toJsonDocument("get_data");
        r->post(data);
    }
    else
    {
        QString message = "[!][HTTPServer TEST:]\tError " +
server.errorString();
        qDebug() << message;
        toLogFile(message);
    }
}
/*=======================================================
метод-слот, вызывается при появлении нового подключения
к серверу
=======================================================*/
void HTTPServer::slt_ConnectionHandler()
{
    QString message = "[HTTPServer:]\tHave a new connection\n";
    qDebug() << message;
    toLogFile(message);

    QTcpSocket * socket = server.nextPendingConnection();
    socket->waitForReadyRead();
    QByteArray request(socket->readAll());
    QStringList buf = QString(request).split(' ');

    message.clear();
```

```
    message = "[!][HTTPServer TEST:]:\tData received:\t" +
buf.at(0);
    qDebug() << message;
    toLogFile(message);

    if(buf.at(0) == "POST")
    {
        QString responce = "HTTP/1.1 200 OK\r\n\r\n%1";
        socket->waitForReadyRead();
        QByteArray data(socket->readAll());
        socket->write(responce.toLatin1());
        socket->waitForBytesWritten();
        socket->disconnectFromHost();
        socket->deleteLater();

        QJsonDocument doc = QJsonDocument::fromJson(data);
        Var v(doc);
        if(v.getType() == "map")
        {
            QMap <QString, Var> map = v.getMap();
            Outcome * outcome = new Outcome();
            outcome->send(map.first().toString());
        }
        else
        {
            QList <Var> list = v.getList();
            Income * income = new Income();
            connect(income, SIGNAL(sig_dataSaved()), this,
SIGNAL(sig_dataSaved()));
            if(income->execute(list))
            {
                QJsonDocument doc =
toJsonDocument("is_processed");
                Request * r = new Request();
                r->post(doc);
            }
        }
    }
}

/*=======================================================
метод конвертирует команду в формат JSON для передачи
по сети

Параметры метода:
QString command - строка с управляющей командой
=======================================================*/
QJsonDocument HTTPServer::toJsonDocument(QString command)
{
    QJsonObject json;
    json.insert("request", QJsonValue(command));
```

```
    QJsonDocument document(json);
    return document;
}
```

Б.18 Реализация наиболее важынх методов класса Model

```
/*=======================================================
метод сохраняет информацию о записи в ВБД. Если запись
уже существует, мето выполняет запрос на обновление
записи. Если такой записи нет, то метод выполняет
запрос на добавление новой записи

Параметры метода:
QStringList fields - перечень полей, которые необходимо
сохранить
=======================================================*/
bool Model::save(QStringList fields)
{
    if(getIsNew() == true)
    {
        bool result = insert();
        if(result == true)
        {
            setIsNew(false);
        }
        return result;
    }
    else
    {
        return update(fields);
    }
}

/*=======================================================
метод генерирует SQL-запрос на удаление записи из ВБД
=======================================================*/
bool Model::remove()
{
    QString sql = "DELETE FROM " + schema->getTableName();
    sql += " WHERE ";
    QList <QString> pk = schema->getPrimaryKey();
    QMap <QString, QVariant>::iterator i = record.begin();
    QString temp = "";
    while(i != record.end())
    {
        for(int j = 0; j < pk.size(); j++)
        {
            if(i.key() == pk[j])
            {
                sql += temp + i.key() + " = ";
                sql += getString(i.value());
```

```
                temp = " AND ";
            }
        }
        i++;
    }
    return execQuery(sql);
}
/*=========================================================
метод генерирует запрос на добавление новой записи в ВБД
=========================================================*/
bool Model::insert()
{
    QString sql = "INSERT INTO " + schema->getTableName() + " (";
    QString fields = "";
    QString values = "";
    QMap <QString, QVariant>::iterator i = record.begin();
    QString temp = "";
    while(i != record.end())
    {
        fields += temp + i.key();
        values += temp + getString(i.value());
        temp = ", ";
        i++;
    }
    sql += fields + ") VALUES (";
    sql += values + ")";
    return execQuery(sql);
}

/*=========================================================
метод генерирует SQL-запрос на изменение записи в ВБД

Параметры метода:
QStringList fields - перечень полей, которые
                     подлежат изменению
=========================================================*/
bool Model::update(QStringList fields)
{
    QString sql = "UPDATE " + schema->getTableName();
    sql += " SET ";
    QMap <QString, QVariant>::iterator i = record.begin();
    QString temp = "";
    while(i != record.end())
    {
        if((fields.empty()) || (fields.contains(i.key())))
        {
            sql += temp + i.key() + "=";
            sql += getString(i.value());
            temp = ", ";
        }
        i++;
```

```
    }
    sql += " WHERE ";
    temp = "";
    int j=0;
    while( j != schema->getPrimaryKey().size())
    {
        sql += temp + schema->getPrimaryKey()[j] + " = ";
        sql += record.value(schema-
>getPrimaryKey()[j]).toString();
        temp = " AND ";
        j++;
    }
    return execQuery(sql);
}

/*=========================================================
метод выполняет сгенерированный SQL-запрос
=========================================================*/
bool Model::execQuery(QString sql)
{
    connectToDB();
    QSqlQuery *query = new QSqlQuery(database);
    query->prepare(sql);
    try
    {
        if(query->exec())
        {
            return true;
        }
        else
        {
            throw query;
        }
    }
    catch (QSqlQuery *)
    {
        QString message = "[!][MODEL]:\tQuery Error!";
        message += query->lastError().text();
        qDebug() << message;
        toLogFile(message);
        return false;
    }
}

/*=========================================================
метод осуществляет подключение к ВБД
=========================================================*/
void Model::connectToDB()
{
    if(QSqlDatabase::contains("dbConnection"))
    {
```

```
        database = QSqlDatabase::database("dbConnection");
    }
    else
    {
        database = QSqlDatabase::addDatabase("QSQLITE",
"dbConnection");
        database.setDatabaseName("base.db");
    }
    try
    {
        if(!database.open())
        {
            throw database;
        }
    }
    catch(QSqlDatabase)
    {
        QString message = "[!][MODEL]:\tDatabase connecting error!
" + database.lastError().text();
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
вспомогательный метод, преобразовывающий значение поля
в строку определенного формата для генерации SQL-запроса

Параметры метода:
QVariant value - значение, которое необходимо
                 преобразовать
=========================================================*/
QString Model::getString(QVariant value)
{
    QString res;
    if(value.type() == QVariant::String)
    {
        res = "'" + value.toString() + "'";
    }
    else
    {
        res = value.toString();
    }
    return res;
}

/*=========================================================
Метод устанавливает отношение для текущей записи

Параметры метода:
Model *outModel - модель, с которой устанавливается
```

```
                    отношение;
QString relationName - имя отношения
=========================================================*/
void Model::setRelation(Model *outModel, QString relationName)
{
    relation.model = outModel;
    schema->setRelations(relationName, relation);
}
```

Б.19 Реализация наиболее важных методов класса Query

```
/*=========================================================
метод настраивает строку для формарования запроса типа
LEFT OUTER JOIN
Параметры метода:
QString RelaionName - имя отношения, на основе которого
                      будет формироваться SQL-запрос
=========================================================*/
void Query::withOuter(QString relationName)
{
    rName = relationName;
    relation_s r = schema->getRelation(rName);
    try
    {
        if((r.link.first != "") && (r.link.second != "") &&
(r.type != 0))
        {
            TableSchema *sch = r.model->getSchema();
            outerJoin =  " LEFT OUTER JOIN ";
            outerJoin += sch->getTableName() + " ON ";
            outerJoin += schema->getRelation(rName).link.first;
            outerJoin += " = " + schema-
>getRelation(rName).link.second;
        }
        else
        {
            throw r;
        }
    }
    catch(relation_s)
    {
        QString message = "[!][QUERY]:\tError! Can not get
relation" + rName + "!";
        message += r.link.first + "; " + r.link.second + "; " +
r.type;
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
```

```
метод настраивает строку для формарования запроса типа
INNER JOIN

Параметры метода:
QString RelaionName - имя отношения, на основе которого
                      будет формироваться SQL-запрос
=========================================================*/
void Query::withInner(QString relationName)
{
    rName = relationName;
    relation_s r = schema->getRelation(rName);
    try
    {
        if((r.link.first != "") && (r.link.second != "") &&
(r.type != 0))
        {
            TableSchema *sch = r.model->getSchema();
            innerJoin =  " INNER JOIN ";
            innerJoin += sch->getTableName() + " ON ";
            innerJoin += schema->getRelation(rName).link.first;
            innerJoin += " = " + schema-
>getRelation(rName).link.second;
        }
        else
        {
            throw r;
        }
    }
    catch(relation_s)
    {
        QString message = "[!][QUERY]:\tError! Can not get
relation" + rName + "!";
        message += r.link.first + "; " + r.link.second + "; " +
r.type;
        qDebug() << message;
        toLogFile(message);
    }
}

/*=========================================================
метод генерирует строку с SQL-запросом и возвращает ее
=========================================================*/
QString Query::generateSQL()
{
    QString sql = "SELECT ";
    if(countString != "")
    {
        sql += countString;
    }
    else
    {
```

```
        sql += selectedFields + " ";
    }
    sql += " FROM ";
    sql += schema->getTableName();
    if(outerJoin != "")
    {
        sql += outerJoin;
    }
    else if(innerJoin != "")
    {
        sql += innerJoin;
    }
    sql += where;
    sql += order;
    sql += limit;
    return sql;
}

/*=========================================================
метод генерирует запрос на выборку количества
определенных полей

Параметры метода:
QString fieldName - имя поля, по которому будет
                    выполнен SQL-запрос
=========================================================*/
int Query::count(QString fieldName)
{
    countString = "COUNT(" + fieldName + ")";
    QString sql = generateSQL();

    list_t list = requestOne(sql);
    if(!list.empty())
    {
        return list[0].second.toInt();
    }
    else
    {
        return 0;
    }
}

/*=========================================================
Метод возвращает один объект модели по выборке
=========================================================*/
Model * Query::getOne()
{
    sqlString = generateSQL();

    list_t vector = requestOne(sqlString);
    try
```

```
    {
        if(vector.empty())
        {
            throw vector;
        }
        if((innerJoin == "") && (outerJoin == ""))
        {
            queryModel->setSchema(schema);
            queryModel = appendRecords(vector, queryModel);
        }
        else
        {
            Model * temp = schema->getRelation(rName).model;
            relation_data_t data = getRelationModel(vector, temp);
            schema->setRelationData(data);
            queryModel->setSchema(schema);
            queryModel = appendRecords(vector, queryModel);
        }
    }
    catch(list_t)
    {
        QString message = "[!][QUERY]:\tRequest list is empty!";
        qDebug() << message;
        toLogFile(message);
    }
    return queryModel;
}

/*=======================================================
метод возвращает перечень моделей по выборке
=======================================================*/
QList<Model *> Query::getAll()
{
    sqlString = generateSQL();
    matrix_t matrix = requestMany(sqlString);

    QList <Model *> list;
    int i=0;
    while(i != matrix.size())
    {
        Model * model = new Model();
        model->setSchema(schema);
        model = appendRecords(matrix[i], model);
        list.append(model);
        i++;
    }
    return list;
}

/*=======================================================
Метод формирует модель
```

```
Параметры метода:
list_t values - перечень пар значений "имя поля-значение";
Model *& model - формируемая модель
=====================================================*/
Model * Query::appendRecords(list_t values, Model * & model)
{
    if(!values.empty())
    {
        model->setIsNew(false);
        QStringList fields = model->getSchema()->getFields();
        for(int i = 0; i < values.size(); i++)
        {
            if(fields.contains(values[i].first))
            {
                model->setField(values[i].first,
values[i].second);
            }
        }
    }
    return model;
}

/*=====================================================
метод возвращает перечень данных об отношениях текущей
модели

Параметры метода:
list_t values - перечень пар "имя поля-значение";
Model *& model - текущая модель
=====================================================*/
relation_data_t Query::getRelationModel(list_t values, Model * &
model)
{
    relation_data_t rData;
    QList <Model *> list;
    if(!values.empty())
    {
        QStringList fields = model->getSchema()->getFields();
        int i = 0;
        while(i < values.size())
        {
            int j = i;
            Model * temp = new Model();
            temp->setSchema(model->getSchema());
            temp->setIsNew(false);
            while(values[j] != QPair <QString, QVariant> ("", ""))
            {
                if((fields.contains(values[j].first)) &&
(values[j].second != ""))
                {
```

```
                    temp->setField(values[j].first,
values[j].second);
                }
                j++;
            }
            list.append(temp);
            i = j + 1;
        }
    }
    rData.insert(rName, list);
    return rData;
}

/*=========================================================
метод выполняет SQL-запрос и возвращает данные об одной
записи из полученной выборки

Параметры метода:
QString sql - строка с SQL-запросом
=========================================================*/
list_t Query::requestOne(QString sql)
{
    list_t result;
    QSqlQuery *query = new QSqlQuery(database);
    query->prepare(sql);
    try
    {
        if(query->exec())
        {
            while(query->next())
            {
                for(int i = 0; i < query->record().count(); i++)
                {
                    QPair <QString, QVariant> pair(query-
>record().fieldName(i), query->value(i));
                    if((innerJoin == "") && (outerJoin == ""))
                    {
                        result.append(pair);
                    }
                    else
                    {
                        QSqlRecord records =
database.record(schema->getRelation(rName).model->getSchema()-
>getTableName());
                        if(records.contains(query-
>record().fieldName(i)))
                        {
                            result.append(pair);
                        }
                    }
                }
```

```
                result.append(QPair <QString, QVariant> ("", ""));
            }
        }
        else
        {
            throw query;
        }
    }
    catch (QSqlQuery *)
    {
        QString message = "[!][Query]:/tError!" + query->lastError().text();
        qDebug() << message;
        toLogFile(message);
    }

    return result;
}

/*=========================================================
метод выполняет сгенерированный SQL-запрос и возвращает
все данные по выборке

Параметры метода:
QString sql - строка с SQL-запросом
=========================================================*/
matrix_t Query::requestMany(QString sql)
{
    matrix_t result;
    QSqlQuery *query = new QSqlQuery(database);
    query->prepare(sql);
    try
    {
        if(query->exec())
        {
            while(query->next())
            {
                QList <QPair <QString, QVariant>> list;
                for(int i = 0; i < schema->getFieldsCount(); i++)
                {
                    QPair <QString, QVariant> pair(query->record().fieldName(i), query->value(i));
                    list.append(pair);
                }
                result.append(list);
            }
        }
        else
        {
            throw query;
        }
```

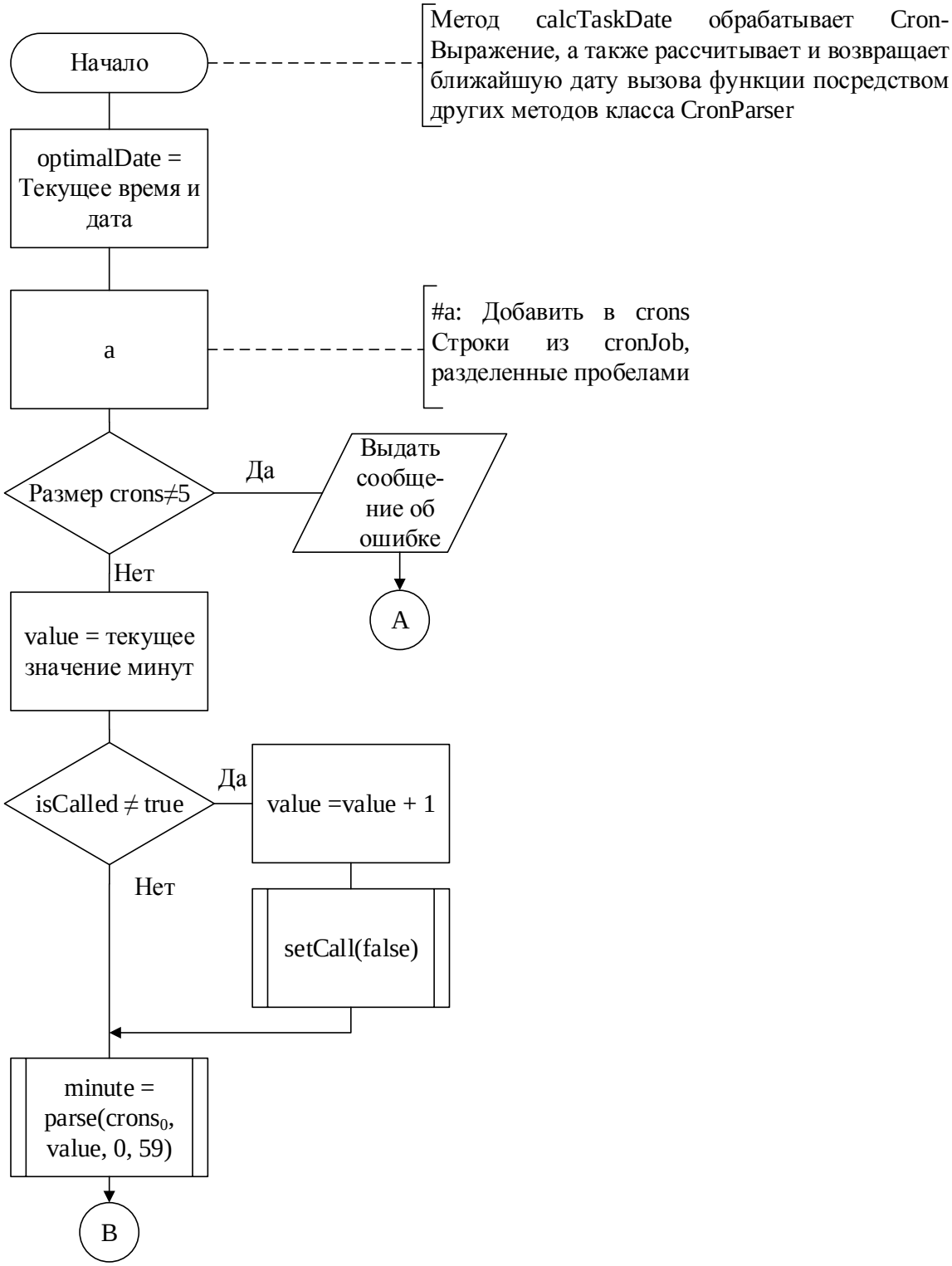
```
    }
    catch (QSqlQuery *)
    {
        QString message = "[!][Query]:/tError!" + query-
>lastError().text();
        qDebug() << message;
        toLogFile(message);
    }
    return result;
}

/*=========================================================
метод возвращает последний идентификатор модели,
добавленный в ВБД
=========================================================*/
QVariant Query::getlastID()
{
    selectedFields = "last_insert_rowid()";
    QString sql = generateSQL();
    list_t list = requestOne(sql);
    if(!list.empty())
    {
        return list[0].second;
    }
    else
    {
        return 0;
    }
}

/*=========================================================
вспомогательный метод, преобразовывающий перечень
значений в строку

Параметры метода:
QStringList list - перечень значений для преобразования
=========================================================*/
QString Query::listToString(QStringList list)
{
    QString result = "";
    QString temp = "";
    int i=0;
    while(i != list.size())
    {
        result += temp + list[i];
        temp = ", ";
        i++;
    }
    return result;
}
```

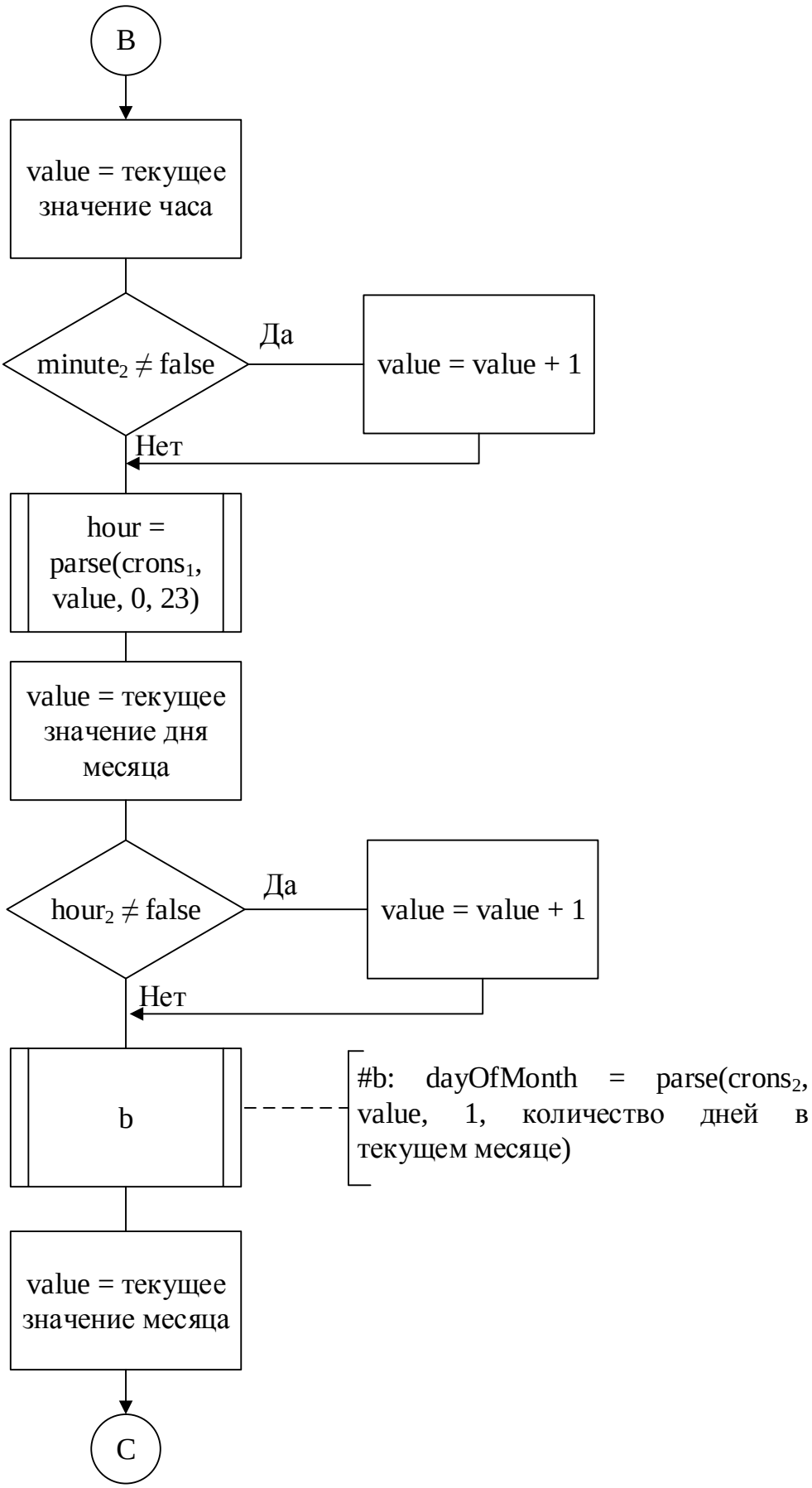
ПРИЛОЖЕНИЕ В

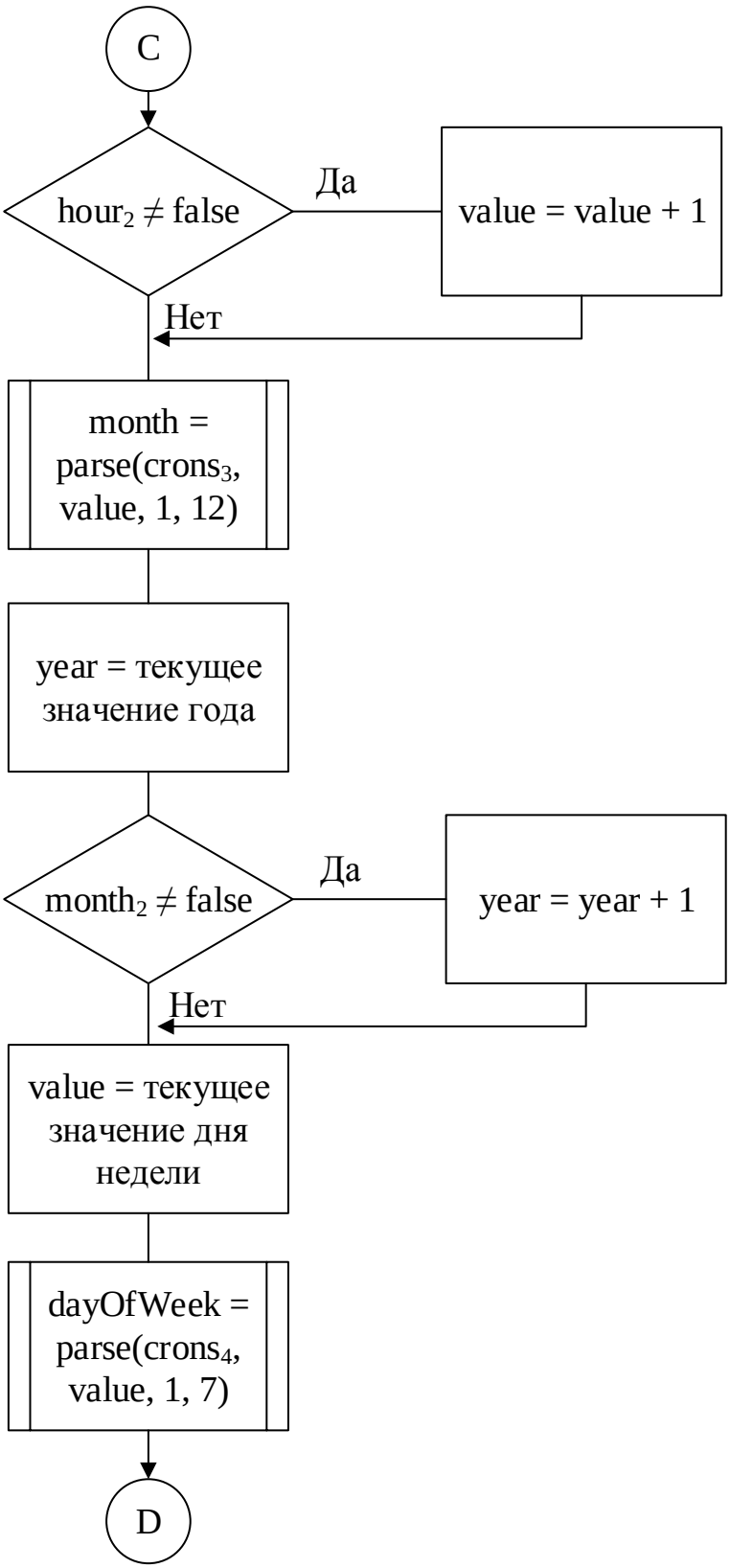
(рекомендуемое)

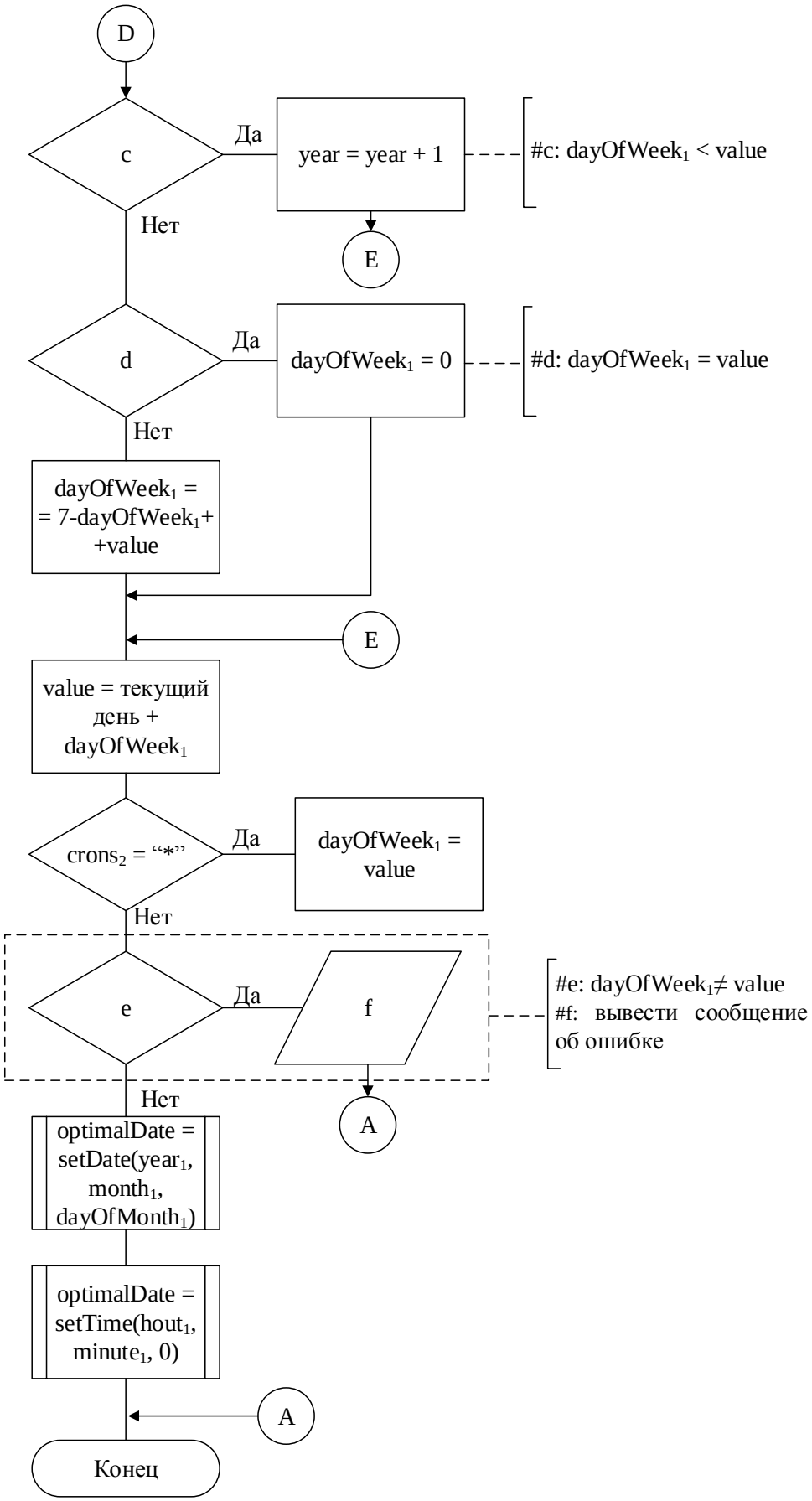
Блок-схемы алгоритмов программы

В.1 Блок-схема алгоритма метода calcTaskDate класса CronParser

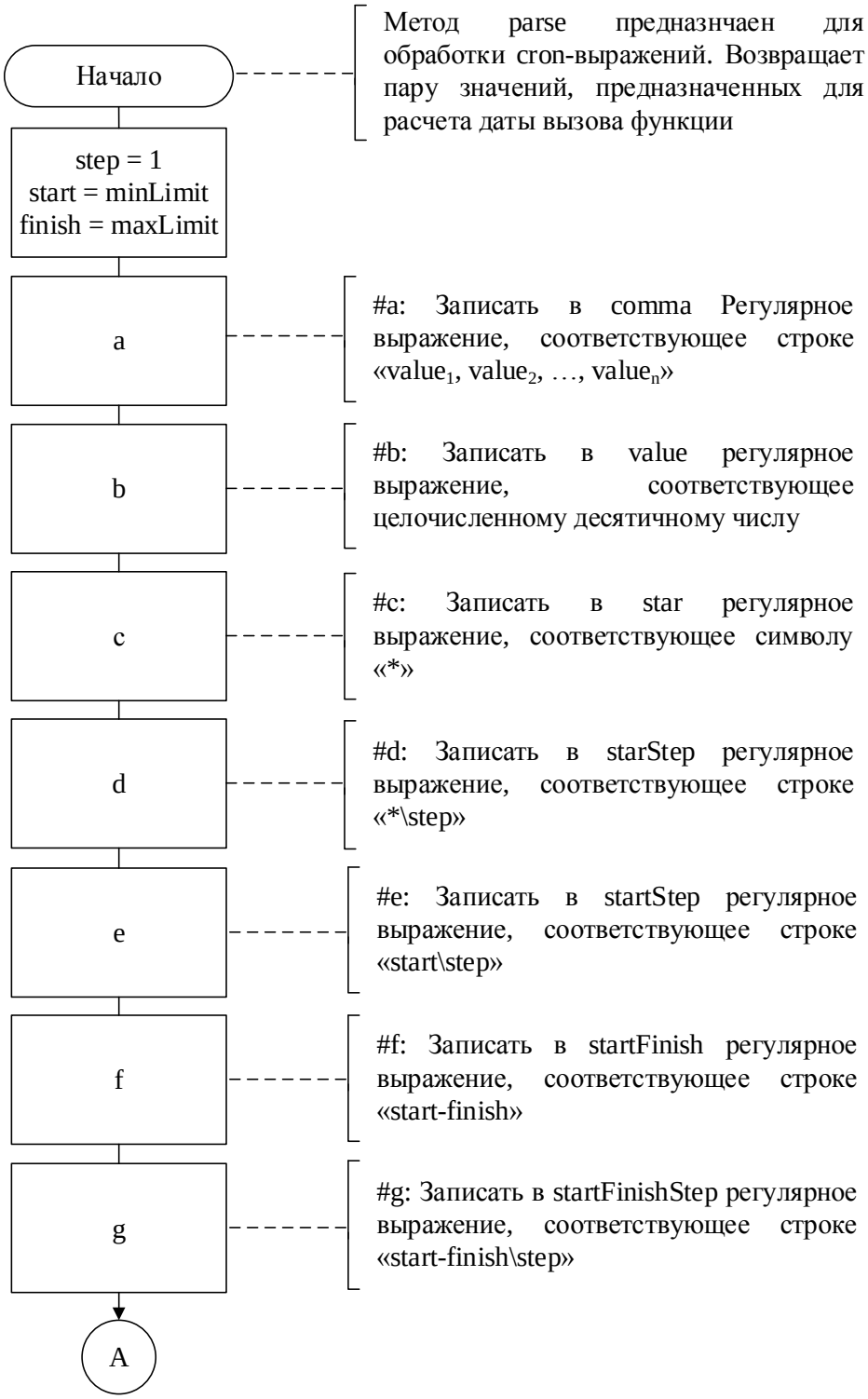


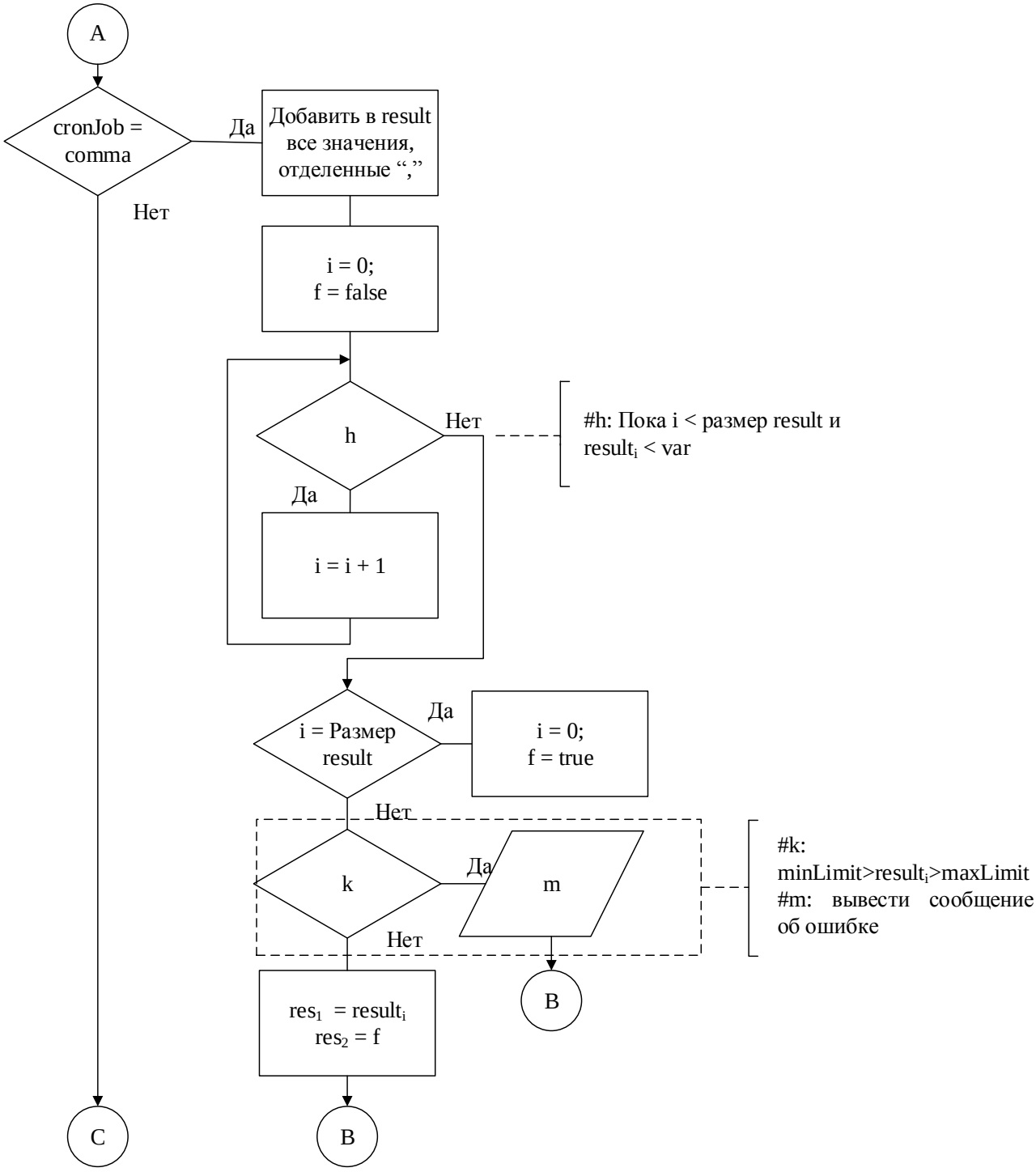

B
value = текущее значение часа
minute$_2$ ≠ false
Да
value = value + 1
Нет
hour = parse(crons$_1$, value, 0, 23)
value = текущее значение дня месяца
hour$_2$ ≠ false
Да
value = value + 1
Нет
b
#b: dayOfMonth = parse(crons$_2$, value, 1, количество дней в текущем месяце)
value = текущее значение месяца
C

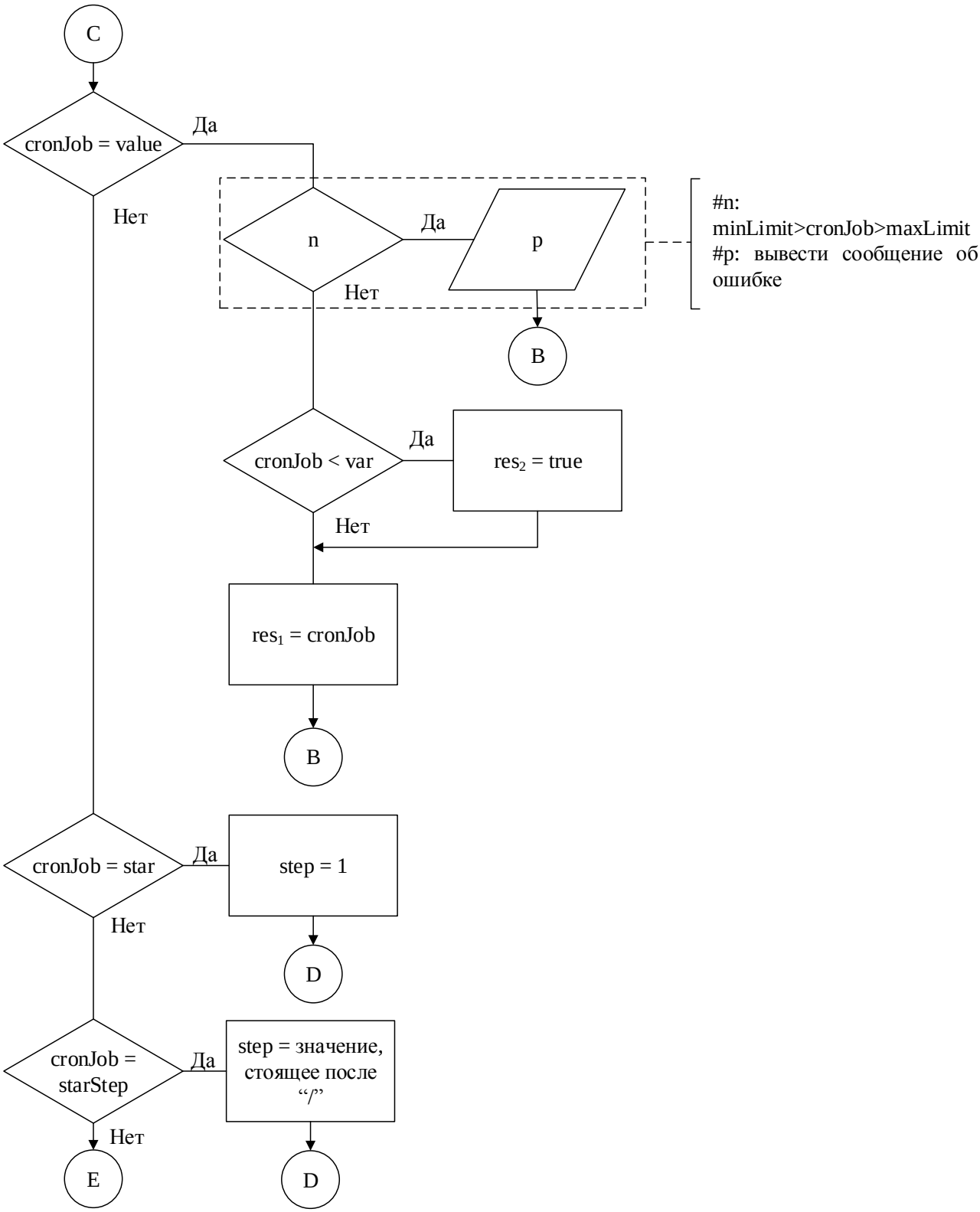

C
hour$_2$ ≠ false
Да
value = value + 1
Нет
month = parse(crons$_3$, value, 1, 12)
year = текущее значение года
month$_2$ ≠ false
Да
year = year + 1
Нет
value = текущее значение дня недели
dayOfWeek = parse(crons$_4$, value, 1, 7)
D

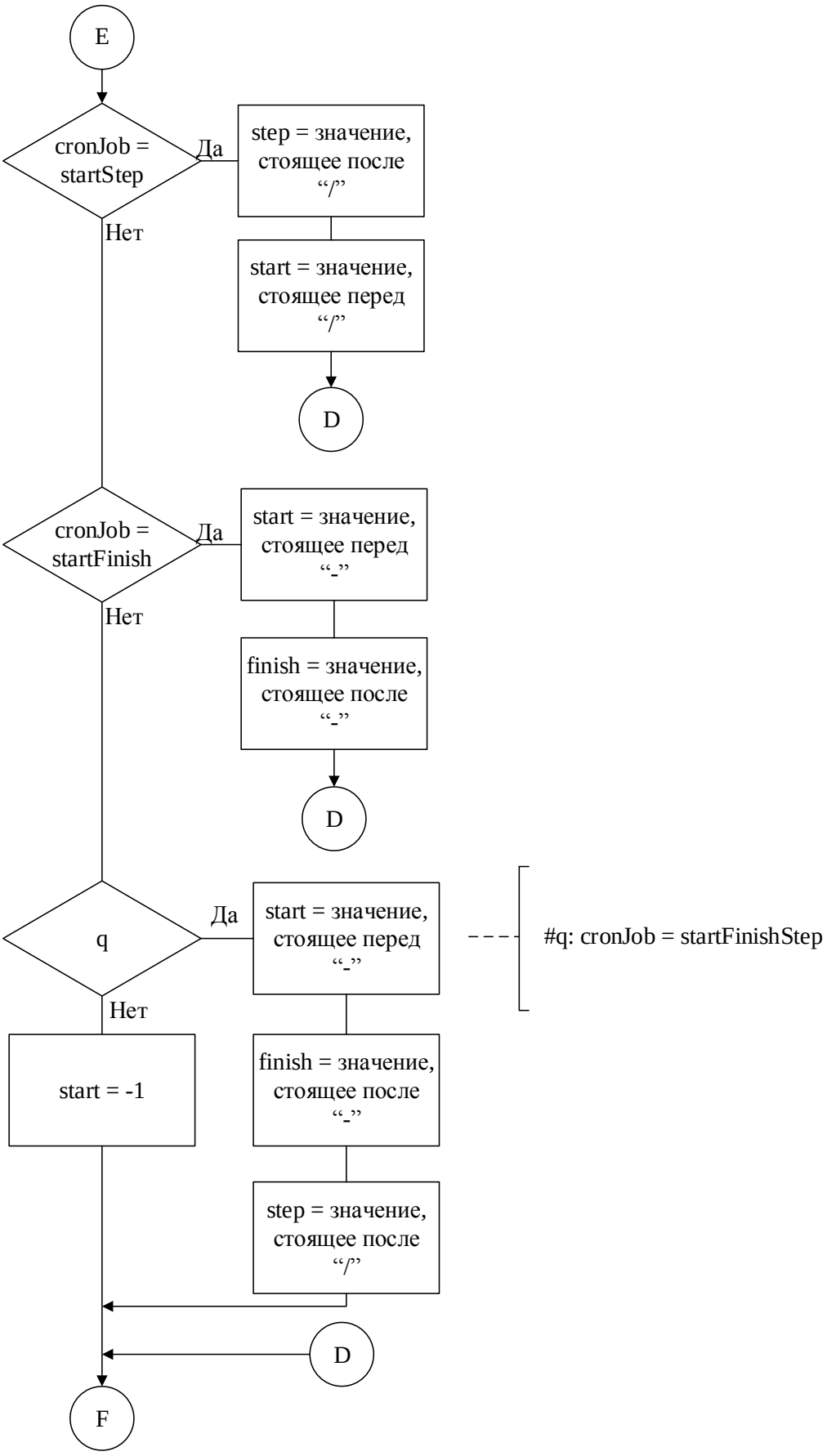

D
c
Да
year = year + 1
#c: dayOfWeek1 < value
Нет
E
d
Да
dayOfWeek1 = 0
#d: dayOfWeek1 = value
Нет
dayOfWeek1 = = 7-dayOfWeek1+ +value
E
value = текущий день + dayOfWeek1
crons2 = “*”
Да
dayOfWeek1 = value
Нет
e
Да
f
#e: dayOfWeek1≠ value
#f: вывести сообщение об ошибке
Нет
A
optimalDate = setDate(year1, month1, dayOfMonth1)
optimalDate = setTime(hout1, minute1, 0)
A
Конец

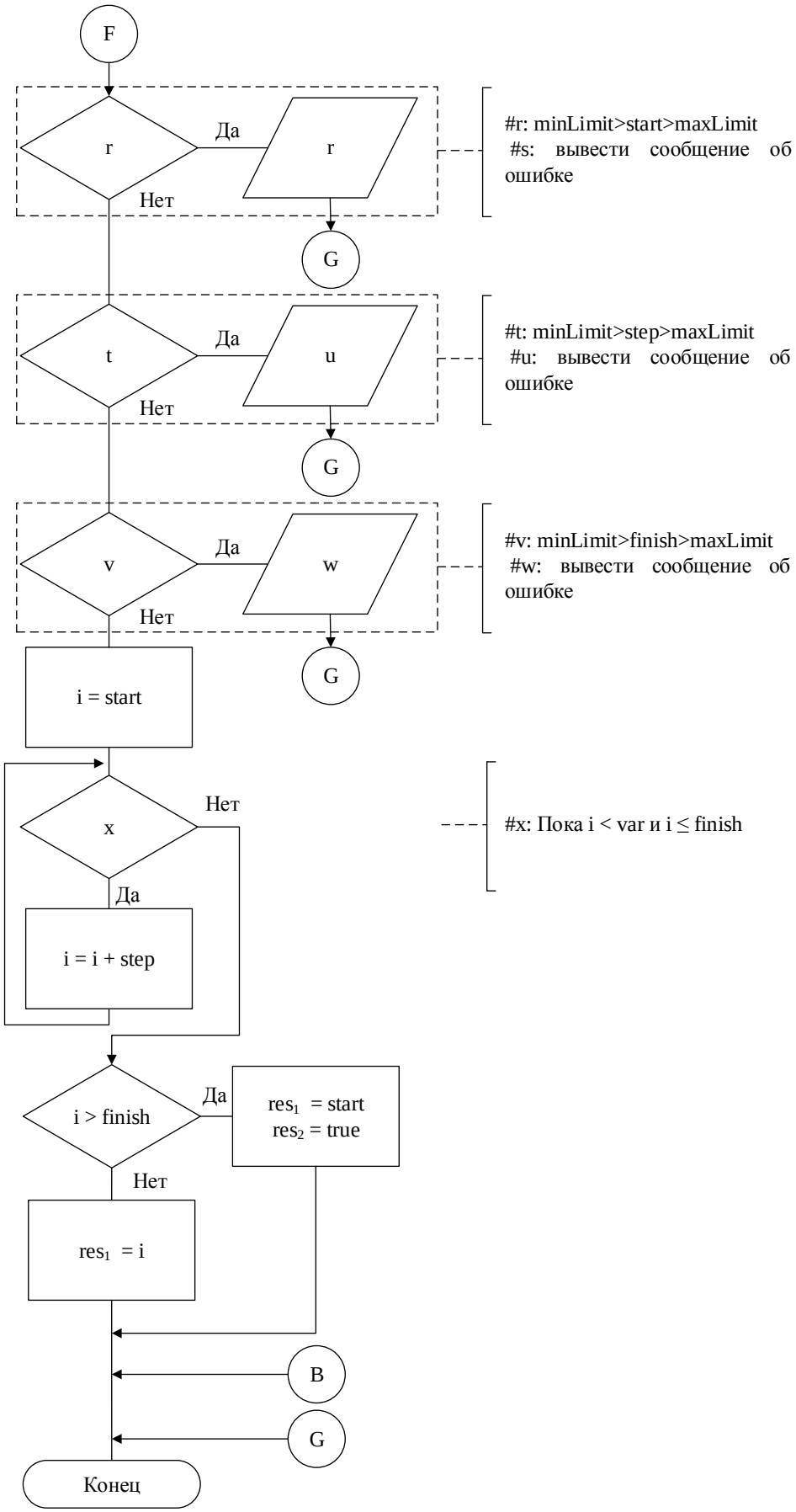
В.2 Блок-схема алгоритма метода calcTaskDate класса CronParser



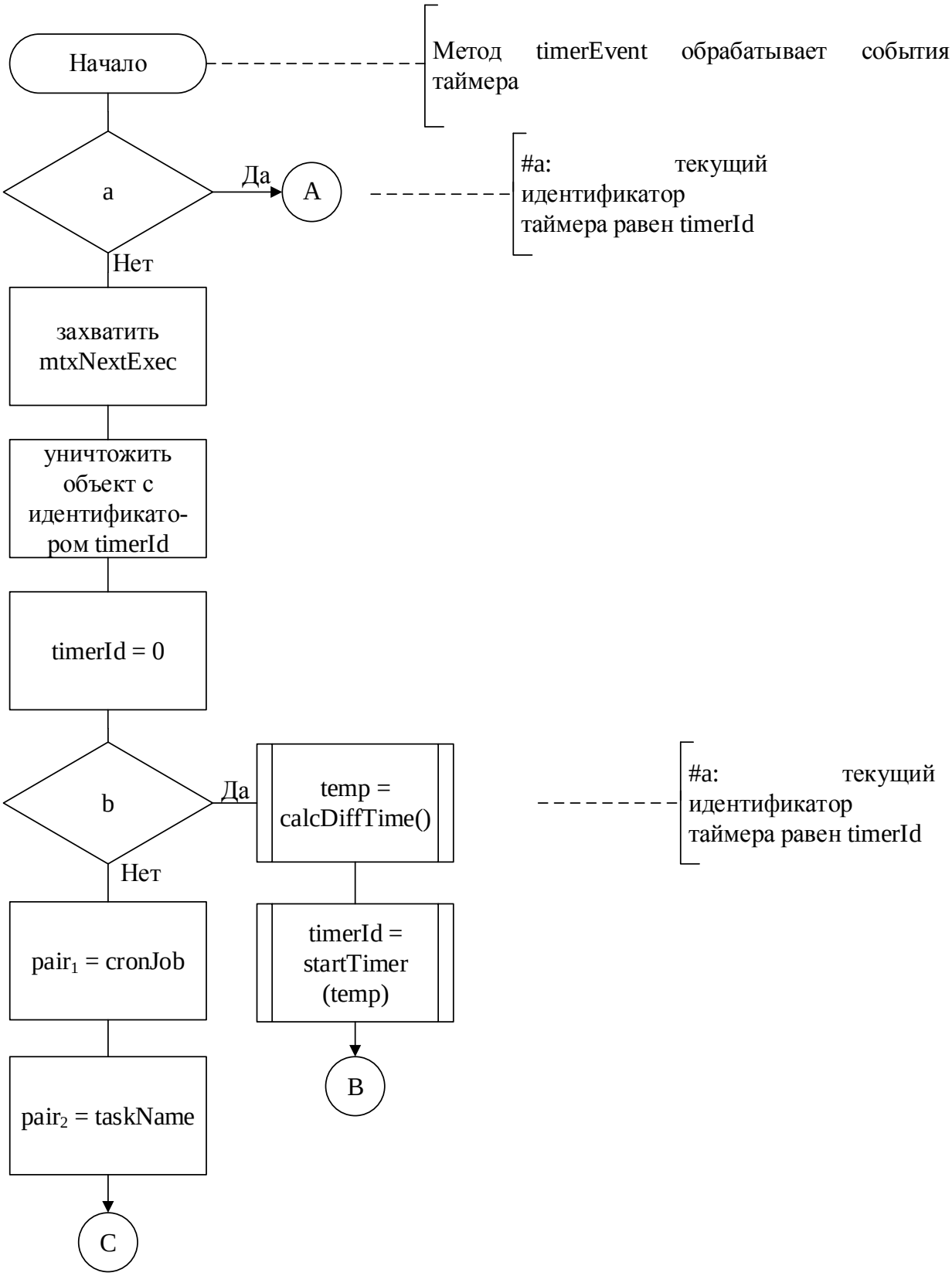

A
cronJob = comma
Да
Добавить в result все значения, отделенные “,”
Нет
i = 0;
f = false
h
Нет
#h: Пока i < размер result и result$_i$ < var
Да
i = i + 1
i = Размер result
Да
i = 0;
f = true
Нет
k
Да
m
Нет
#k: minLimit>result$_i$>maxLimit
#m: вывести сообщение об ошибке
B
res$_1$ = result$_i$
res$_2$ = f
C
B

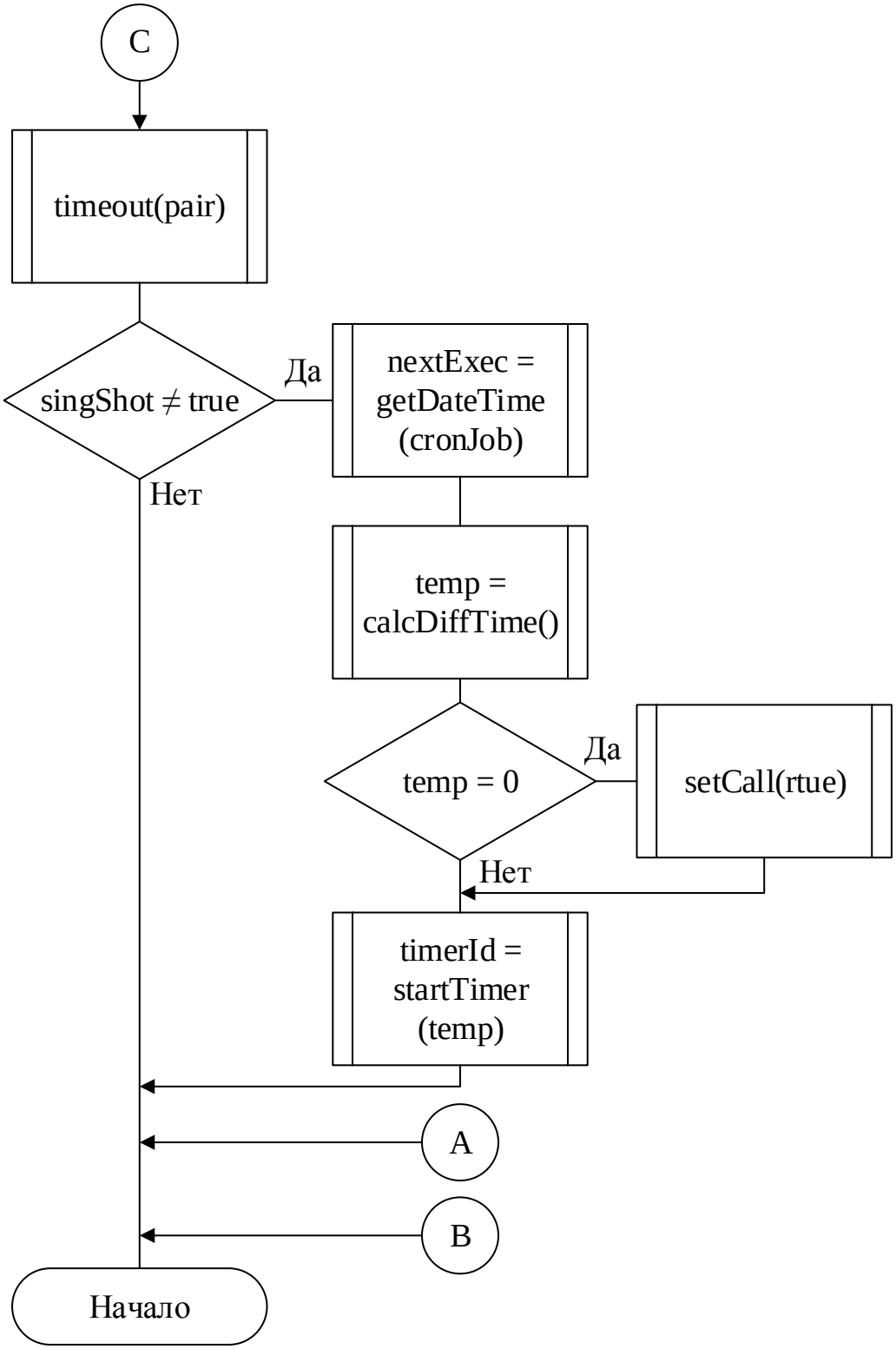

C
cronJob = value
Да
Нет
n
Да
p
Нет
#n:
minLimit>cronJob>maxLimit
#p: вывести сообщение об ошибке
B
cronJob < var
Да
res2 = true
Нет
res1 = cronJob
B
cronJob = star
Да
step = 1
Нет
D
cronJob = starStep
Да
step = значение, стоящее после "/"
Нет
E
D


E
cronJob = startStep
Да
step = значение, стоящее после “/”
Нет
start = значение, стоящее перед “/”
D
cronJob = startFinish
Да
start = значение, стоящее перед “-”
Нет
finish = значение, стоящее после “-”
D
q
Да
start = значение, стоящее перед “-”
#q: cronJob = startFinishStep
Нет
start = -1
finish = значение, стоящее после “-”
step = значение, стоящее после “/”
D
F

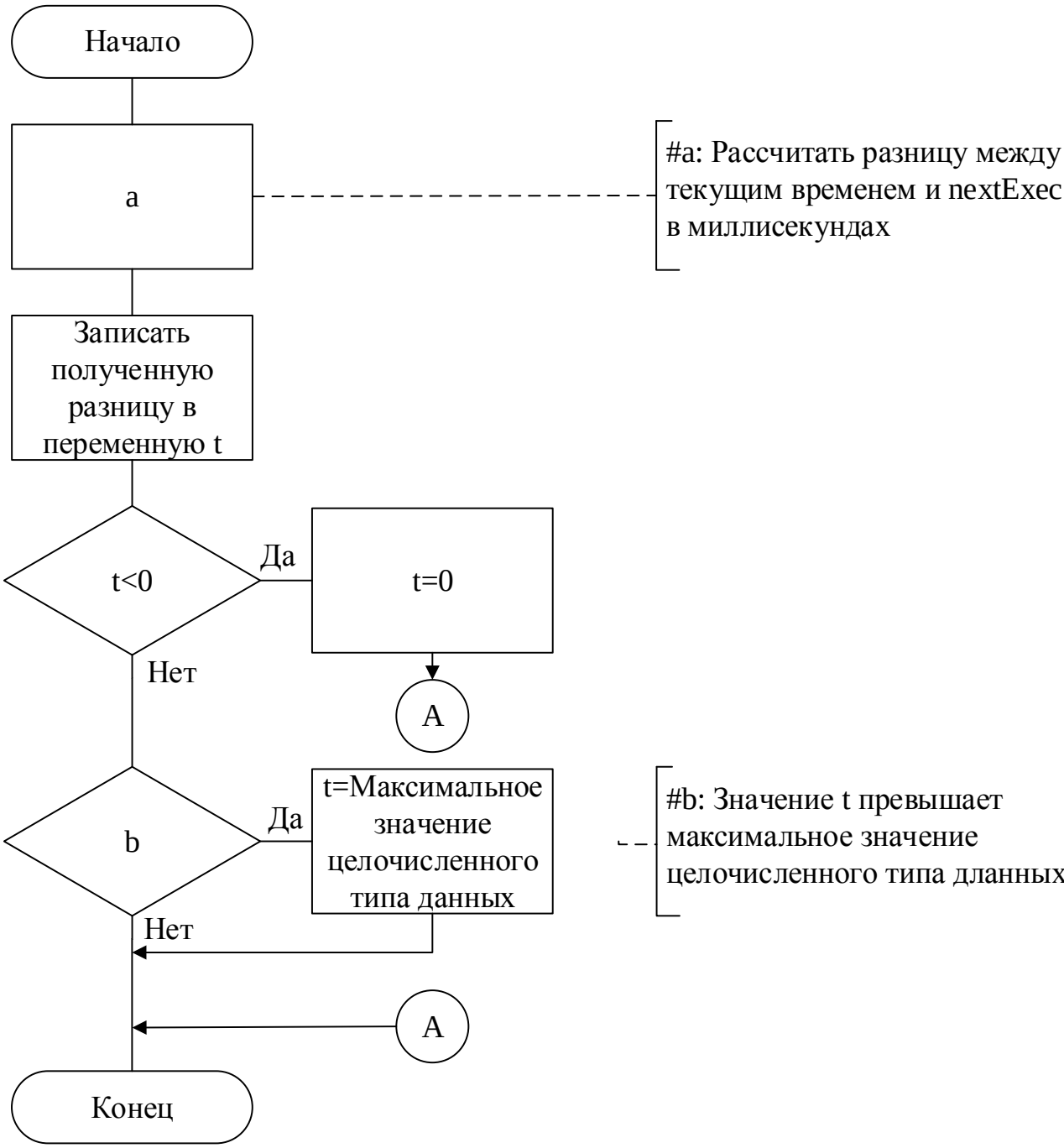

F
r
Да
r
Нет
#r: minLimit>start>maxLimit
#s: вывести сообщение об ошибке
G
t
Да
u
Нет
#t: minLimit>step>maxLimit
#u: вывести сообщение об ошибке
G
v
Да
w
Нет
#v: minLimit>finish>maxLimit
#w: вывести сообщение об ошибке
G
i = start
x
Нет
Да
#x: Пока i < var и i ≤ finish
i = i + step
i > finish
Да
res1 = start
res2 = true
Нет
res1 = i
B
G
Конец

В.3 Блок-схема алгоритма метода timerEvent класса Timer




C
timeout(pair)
singShot ≠ true
Да
nextExec = getDateTime (cronJob)
Нет
temp = calcDiffTime()
temp = 0
Да
setCall(rtue)
Нет
timerId = startTimer (temp)
A
B
Начало

В.4 Блок-схема алгоритма метода calcDiffTime класса Timer



В.5 Блок-схема алгоритма метода start класса Timer