Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2024 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Разработка подсистемы синхронизации данных в сети передачи телеметрической информации

090401 Информатика и вычислительная техника

09.04.01.11 «Вычислительные системы и сети»

Руководитель	_____ *подпись*	_____ *дата*	профессор, канд. техн. наук *должность, ученая степень*	О.В. Непомнящий
Выпускник	_____ *подпись*	_____ *дата*		Д.С. Галкина
Рецензент	_____ *подпись*	_____ *дата*	доцент, канд. техн. наук *должность, ученая степень*	К.В. Раевич
Консультант	_____ *подпись*	_____ *дата*	генеральный директор ООО «ПК «Дельта»» *должность, ученая степень*	О.Г. Варыгин
Нормоконтролёр	_____ *подпись*	_____ *дата*	профессор, канд. техн. наук *должность, ученая степень*	О.В. Непомнящий

Красноярск 2024

РЕФЕРАТ

Выпускная квалификационная работа на тему «Разработка подсистемы синхронизации данных в сети передачи телеметрической информации» содержит 88 страниц текстового документа, 20 рисунков, 3 таблицы, 2 приложения, и 21 использованный источник.

СИНХРОНИЗАЦИЯ ДАННЫХ, ТЕЛЕМЕТРИЯ, ПЕРЕДАЧА ДАННЫХ, СЕТЬ, БАЗЫ ДАННЫХ, АЛГОРИТМ, C++, LINUX, MYSQL.

Цель работы: исследование, моделирование и разработка технических решений для создания подсистемы синхронизации данных в сети передачи телеметрической информации.

Задачи, решённые в процессе разработки:

- проведён аналитический обзор предметной области;
- определены критерии сравнения различных методов синхронизации данных;
- предложен метод синхронизации данных, базирующийся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации;
- разработана программная модель системы синхронизации данных;
- составлен набор тестовых данных;
- проведено тестирование разработанной программной модели.

В результате выполнения ВКР предложен метод синхронизации данных в сети передачи телеметрической информации и разработана программная модель модуля управления потоками данных из состава программного обеспечения контроллера сети передачи телеметрических данных.

АННОТАЦИЯ

Магистерская диссертация на тему «Разработка подсистемы синхронизации данных в сети передачи телеметрической информации» выполнена в Институте космических и информационных технологий Сибирского федерального университета.

Проводятся исследования методов синхронизации данных в системах сбора и анализа телеметрической информации. Разрабатывается метод синхронизации данных между устройствами в подобной системе. Разрабатываются программные модели и программные средства из состава подсистемы синхронизации данных в сети передачи телеметрической информации. Проводится тестирование разработанных моделей и программных средств.

Отличительной особенностью разрабатываемой системы синхронизации данных в сети передачи телеметрической информации является предложенный метод синхронизации данных, базирующийся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации. Этот метод позволяет обеспечить синхронизацию данных в системе передачи телеметрической информации в условиях передачи ограниченного объёма данных с учётом приоритетов пользователя.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность проблемы: синхронизация – важный процесс, без которого невозможно нормальное функционирование современных систем сбора и хранения информации. Существует множество видов синхронизации, которые позволяют обеспечить, например, высокую точность и стабильность сигналов электросвязи, поддерживать необходимые временные соотношения протекания процессов во времени, или обеспечить пользователю удобство хранения информации.

Понятие синхронизации многозначно, и, в зависимости от конкретного содержания решаемых задач, может иметь разные значения. Так, различают тактовую и цикловую синхронизацию, синхронизацию пакетов, средств мультимедиа, радиочастот GSM, синхронизацию данных и др. [1].

Системы передачи телеметрических данных предназначены для сбора информации об изменениях и колебаниях значений заданных параметров на подконтрольных объектах. Передача данных при этом осуществляется посредством беспроводных или проводных сетей (оптических, электрических, компьютерных, а также радиоканалов) [2].

Примером подобной системы может служить проект системы передачи телеметрической информации с использованием спутникового канала. Этот проект представляет собой распределённый комплекс спутниковой связи, состоящий, условно, из двух больших подсистем: непосредственно системы спутниковой связи, и наземного программно-аппаратного комплекса передачи телеметрической информации. Наземный комплекс, в свою очередь, представляет из себя совокупность серверов, контроллеров, устройств приёма и передачи телеметрической информации, и клиентских устройств. Все эти устройства обмениваются данными посредством беспроводных сетей.

В связи с использованием спутникового канала связи в разрабатываемой системе появляются ограничения на количество передаваемых по этому каналу данных, ввиду высокой стоимости данного способа передачи информации.

Следовательно, необходимо в первую очередь синхронизировать только те данные, которые находятся в приоритете у пользователя.

Таким образом, актуальной является разработка подсистемы синхронизации данных в сети передачи телеметрической информации с целью отображения на серверах и клиентских устройствах актуальной информации с устройств сбора телеметрической информации.

Цель диссертационной работы: исследование, моделирование и разработка технических решений для создания подсистемы синхронизации данных в сети передачи телеметрической информации.

Для достижения поставленной цели сформулированы следующие задачи исследования:

- исследовать методы синхронизации данных в системах сбора и анализа телеметрической информации;
- разработать метод синхронизации баз данных, и программный модуль управления потоками данных;
- разработать необходимые программные модели и программные средства из состава подсистемы синхронизации данных в сети передачи телеметрической информации;
- провести тестирование разработанных моделей и программных средств.

Предполагаемая научная новизна исследования заключается в предложенном методе синхронизации данных, базирующемся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации, и позволяющим обеспечить синхронизацию данных в системе передачи телеметрической информации в условиях передачи ограниченного объёма данных с учётом приоритетов пользователя.

1 Исследования методов синхронизации данных в системах сбора и анализа телеметрической информации

1.1 Постановка задачи

Системы передачи телеметрической информации позволяют собирать данные, передаваемые от различных устройств или датчиков, и обрабатывать их, а также формировать управляющие сигналы для периферийных устройств, подключенных к системе. Системы, обладающие подобным принципом работы, находят своё применение в самых разных областях – от «умного дома» до автоматизации производства.

В настоящее время в России создаются некоторые проекты в данной области, причём как крупными компаниями, так и менее известными. системы передачи телеметрической информации посредством спутникового канала. Этот проект представляет собой распределённый комплекс спутниковой связи, состоящий, условно, из двух больших подсистем: непосредственно системы спутниковой связи, и наземного программно-аппаратного комплекса передачи телеметрической информации. Данный комплекс представляет из себя совокупность контроллера, сервера, периферийных устройств, и клиентских устройств (Рисунок 1). К периферийным устройствам в разрабатываемой системе относятся датчики для сбора телеметрической информации, и исполнительные устройства, принимающие управляющие сигналы от контроллера. Все устройства комплекса передачи телеметрической информации обмениваются данными посредством беспроводных сетей, но возможно и проводное соединение контроллера с сетью Интернет для доступа к серверу.

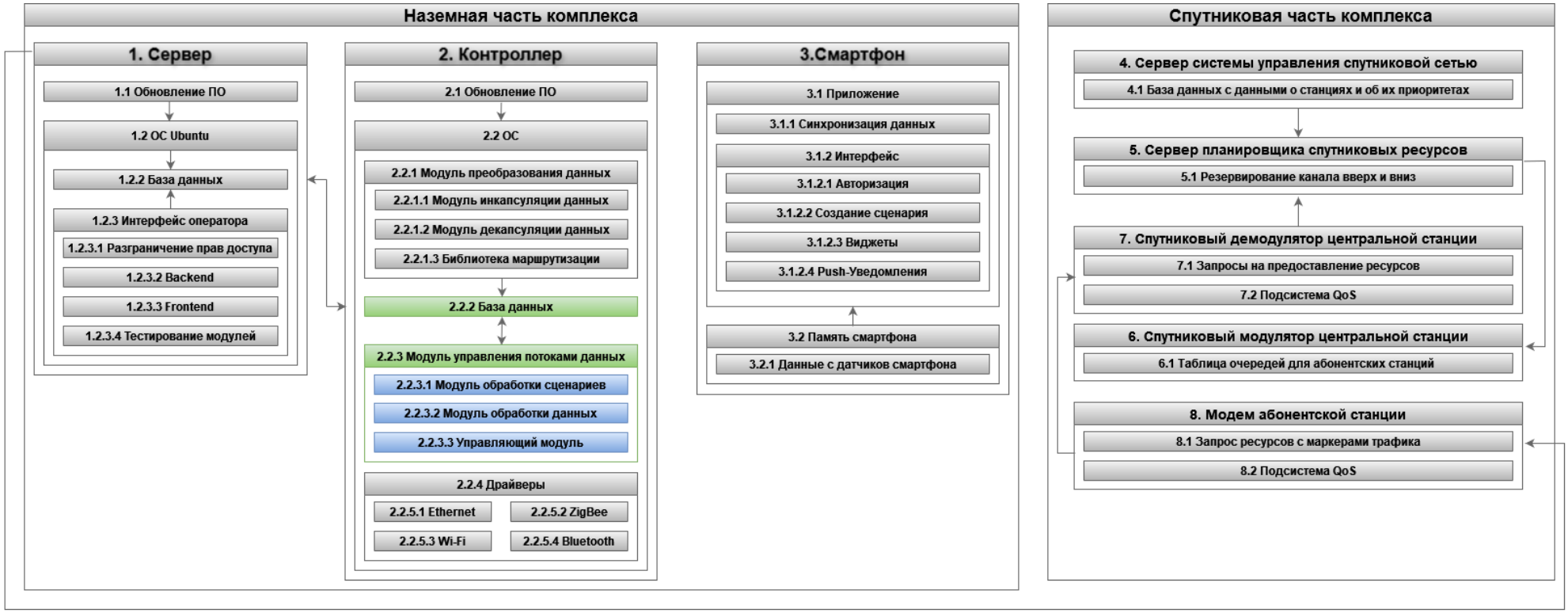


Рисунок 1 – Программная архитектура системы передачи телеметрической информации

Из рисунка 1 можно увидеть, что основными компонентами разрабатываемого программно-аппаратного комплекса являются контроллер, сервер и смартфон пользователя, а также спутниковая сеть передачи данных, служащая каналом связи между контроллером и сервером. Тем не менее, при подключении к сети Интернет через Wi-Fi либо по LAN-соединению, контроллер может обмениваться данными с сервером и без участия спутника. Также пользовательское приложение, установленное на смартфоне, может подключаться к контроллеру как через сервер, так и напрямую по Wi-Fi, при условии, что устройство пользователя и контроллер находятся в одной Wi-Fi сети.

Контроллер, входящий в состав системы, предназначен для сбора телеметрической информации с подключенных к нему датчиков, её обработки и хранения. По запросу пользователя информация передаётся через сервер, либо напрямую в приложение. Также, кроме датчиков, к контроллеру подключены и исполнительные устройства, которыми можно управлять при помощи сигналов, поступающих с контроллера.

Приложение для смартфона предназначено для просмотра поступающей на контроллер информации с датчиков, а также для создания сценариев по сбору данных и управлению исполнительными устройствами.

Сервер необходим для хранения системной информации.

Между контроллером, сервером и пользовательскими устройствами передаются такие данные, как показания датчиков, управляющие сигналы от контроллера, информация о работе системы для клиентских устройств, а также данные, вводимые пользователем на клиентском устройстве. Эти данные нуждаются в синхронизации для того, чтобы корректно отображать и обрабатывать данные пользователя, или данные, пришедшие с контроллера, на всех устройствах. Также синхронизация данных позволит управлять контроллером с любого пользовательского устройства, а людям, обслуживающим систему, позволит исправлять возникающие неполадки.

Однако в связи с использованием спутникового канала связи в разрабатываемой системе появляются ограничения на количество передаваемых по этому каналу данных, ввиду высокой стоимости данного способа передачи информации. Следовательно, необходимо в первую очередь синхронизировать только те данные, которые находятся в приоритете у пользователя.

Таким образом, система синхронизации данных между устройствами программно-аппаратного комплекса передачи телеметрической информации необходима для корректной работы системы, чтобы избегать чрезмерных задержек при передаче данных между устройствами, и всегда предоставлять пользователю и администраторам системы актуальную информацию.

Понятие «синхронизация» может принимать, в зависимости от характера решаемых задач, множество разных значений. К примеру: тактовая синхронизация, синхронизация по циклам, синхронизация пакетов, синхронизация средств мультимедиа, «синхронное» детектирование, и т.д. [3]. В данной работе речь идёт о синхронизации данных, передаваемых между контроллером, сервером и пользовательскими устройствами (Рисунок 1). Таким образом, задача состоит в том, чтобы на контроллере, сервере и устройстве пользователя всегда хранилась актуальная информация о состоянии системы, а также о действиях пользователя, при этом необходимо минимизировать объём передаваемых с контроллера на сервер данных.

В следующих частях данной работы представлен обзор различных способов реализации систем синхронизации данных в беспроводных сетях.

1.2 Обзор предметной области

Согласно определённой цели и поставленным задачам исследования для разработки метода синхронизации данных в системе передачи телеметрической информации следует выполнить обзор предметной области. Для его проведения был выделен ряд стандартов и методов синхронизации, рассматриваемых ниже.

1.2.1 Стандарты OMA DS, SyncML

Для синхронизации данных на мобильных платформах используется стандарт OMA DS, разработанный Open Mobile Alliance (OMA, Открытый мобильный альянс) – международной организацией по стандартизации, которая занимается разработкой открытых стандартов для мобильных платформ [4]. Существует несколько версий протокола OMA DS, последняя из них, 2.0, была опубликована в 2011 году [5].

OMA Data Synchronization (OMA DS) – это спецификация инструментария синхронизации данных и основанного на SyncML формата обмена данными между устройствами, объединенными в сеть. OMA DS разработан для использования на мобильных платформах, для которых подключение к Интернету периодически прерывается.

Протокол OMA DS поддерживает несколько видов синхронизации, в их числе двусторонняя синхронизация, при которой клиент и сервер в равной степени обмениваются информацией друг с другом, и односторонняя синхронизация, при которой только одна сторона (клиент либо сервер) посылает другой сведения. При этом в зависимости от вида синхронизации, сведения, посылаемые другой стороне, могут отличаться – это могут быть только изменённые данные, или же все данные, хранящиеся на клиенте или на сервере. Всего поддерживается семь типов синхронизации данных.

Данные в стандарте OMA DS передаются в формате SyncML. SyncML (Synchronization Markup Language) – это протокол синхронизации данных, на котором основан стандарт OMA DS, а также язык разметки синхронизации, основанный на XML. Каждое сообщение в формате SyncML содержит заголовок (Header) и тело (Body). Заголовок содержит метаинформацию, такую как база-источник <Source>, база-приемник <Target>, информация аутентификации <Cred>, идентификатор сессии <SessionID>, идентификатор сообщения <MsgID>, версия используемого стандарта SyncML. Тело содержит в себе запросы (requests), сигналы (alerts) и данные (data) [4, 6, 7].

Протоколы SyncML и OMA DS работают с протоколами HTTP (HyperText Transfer Protocol, OBEX (OBject EXchange, используется в Bluetooth), WSP (Wireless Session Protocol). Обмен данными между участниками синхронизации данных осуществляется посредством обмена сообщениями формата, описанного выше. При работе этих протоколов ведётся учёт изменений в данных (журналирование изменений). Используются идентификаторы сеансов синхронизации, и проверка их соответствия. Для обеспечения безопасности сеансов применяется проведение аутентификации, а также MD5 на уровне сервера. Оба (клиент и сервер) участника синхронизации данных должны быть в состоянии провести аутентификацию и отправить результаты проведения аутентификации обратно. Кроме того, данные протоколы поддерживают и другие функции.

1.2.2 Синхронизация при помощи базы данных

Ещё один метод, который можно использовать для синхронизации данных в сети – использование баз данных [8]. При таком подходе необходимо разместить базы данных на контроллере, на сервере и на клиентских устройствах. Наборы таблиц и связи между ними должны быть идентичными. При изменении информации в одной из баз данных необходимо копировать изменённую информацию в остальные базы данных. Этот процесс называется синхронизацией баз данных.

При использовании SQL Server можно использовать как встроенные инструменты (например, SQL Server Analysis Services) [9], так и готовые решения, такие как ApexSQL Data Diff [10] и AutosyncDB [11]. Эти решения позволяют выявлять различия между объектами базы данных, анализировать их, создавать собственные настраиваемые скрипты синхронизации.

Изменения в базах данных можно отслеживать при помощи специальных служебных полей в таблицах, в которых записывается некая «контрольная сумма», которая увеличивается при изменении данных в других полях таблицы.

Изменение этого поля можно отслеживать в качестве события для начала синхронизации – в таком случае синхронизация будет начата сразу после записи новых данных в одну из баз, при условии соединения с другими базами данных.

1.2.3 Облачная синхронизация

Рассмотрим ещё один метод синхронизации данных – использование облачного хранилища. Этот метод позволяет сохранять данные, помимо сервера, контроллера и клиентского устройства, в стороннем онлайн-хранилище, что даёт возможность резервного копирования данных.

Облачные хранилища для хранения различных данных и файлов могут предоставляться сторонними компаниями. Рассмотрим несколько примеров подобных сервисов:

Dropbox – облачное хранилище данных. Dropbox позволяет пользователю размещать файлы на удаленных серверах при помощи клиента или с использованием веб-интерфейса через браузер. При установке клиентского программного обеспечения Dropbox на компьютере создается синхронизируемая папка. Существуют мобильные клиенты для мобильных устройств [12, 13].

Google Drive (Гугл Диск) – облачное хранилище данных [14]. Сервис хранения, редактирования и синхронизации файлов на различных устройствах, разработанный компанией Google. Его функции включают хранение файлов в Интернете, общий доступ к ним и совместное редактирование. Доступ к хранилищу осуществляется через браузер, или через мобильное приложение.

Облако Mail.ru – облачное хранилище данных, разработанное российской компанией VK [15]. В целом, предоставляет функции, аналогичные функциям Google Drive.

Все представленные выше облачные хранилища имеют встроенную защиту файлов и личных данных пользователя. Однако при хранении в облаке

данных есть риск подвергнуться атаке, поэтому существуют исследования, направленные на дополнительную защиту данных в облачных хранилищах [16].

Также нужно учитывать, что описанные выше облачные хранилища не являются бесплатными для корпоративного и коммерческого использования.

В результате обзора предметной области были рассмотрены некоторые методы синхронизации данных. Полученные результаты обзора позволяют перейти к их анализу.

1.3 Аналитический обзор полученных предварительных результатов

1.3.1 Критерии сравнения различных методов синхронизации данных

Для того, чтобы проанализировать представленные выше методы синхронизации данных, были использованы следующие критерии анализа:

1. Защита передаваемых данных. При синхронизации данных не должно возникнуть утечки, а также нужно минимизировать риск атаки извне, при передаче данных через Интернет.

2. Тип хранимых данных.

3. Режим синхронизации (по событию или по расписанию). Под событием понимается появление файла с новыми данными. Синхронизация по расписанию используется для обновления данных через фиксированные интервалы времени.

4. Ведение журнала событий. Необходимо для администрирования системы.

5. Регулируемый трафик. Выбор из рассмотренных способов синхронизации тех, что предоставляют возможность регулировать количество синхронизируемых данных по желанию пользователя.

1.3.2 Сравнение различных методов синхронизации данных

Используя приведённые выше критерии, составим таблицу, в которой проанализируем различные методы синхронизации данных:

Таблица 1 – Сравнение методов синхронизации данных

Критерий	**SyncML / OMA DS**	**Базы данных**	**Облачное хранилище**
Защита данных	+	+	+
Тип данных	XML	SQL	Файлы
Режим синхронизации	По событию	По событию	По событию
Журнал событий	+	+	+
Регулируемый трафик	-	+	-

Проанализировав полученную таблицу, можно увидеть, что представленные методы синхронизации данных отличаются типом синхронизируемых данных. В случае с протоколами SyncML и OMA DS данные передаются в виде сообщений с разметкой XML. В базах данных данные хранятся в таблицах с полями различных типов. В облаке же есть возможность размещать и загружать файлы. Все рассмотренные методы имеют встроенное шифрование и возможность увеличить защищённость данных с помощью сторонних или собственных разработок. Также все эти методы начинают синхронизацию данных по событию, то есть в случае, когда в данных на разных устройствах наблюдаются различия. Ведение журнала событий также может быть реализовано для всех трёх методов.

Таким образом, все три рассматриваемых метода являются подходящими для синхронизации данных. Однако для разработки системы синхронизации в составе программно-аппаратного комплекса передачи телеметрической информации наиболее подходит вариант использования баз данных, так как в таком случае информация более структурирована, что является важным в

условиях разрабатываемой системы, а также более подходит для передачи через спутниковый канал связи благодаря возможности регулировать объём передаваемого трафика. Однако существует необходимость доработки предложенного способа синхронизации данных, так как, согласно поставленной ранее задаче, необходимо синхронизировать только данные, приоритетные для пользователя.

Полученный результат анализа позволяет разработать задание для дальнейшей работы по теме исследования.

1.4 Выводы по главе

Выполнено исследование методов синхронизации данных в системах сбора и анализа телеметрической информации.

Сформулирована задача дальнейшего исследования и разработки.

Для определения требований к разрабатываемой системе выполнен обзор предметной области. Выделен ряд стандартов и методов синхронизации данных, в том числе Стандарты OMA DS, SyncML, синхронизация при помощи баз данных, облачная синхронизация.

Для выполнения анализа результатов обзора предметной области введены критерии сравнения различных методов синхронизации данных: защита данных, тип данных, режим синхронизации, ведение журнала событий, минимальный трафик.

На основании полученных результатов и введенных критериев выполнено сравнение различных методов синхронизации данных.

В результате проведённого анализа установлено, что все три рассмотренных метода являются подходящими для синхронизации данных. Однако для разработки системы синхронизации в составе программно-аппаратного комплекса передачи телеметрической информации наиболее подходит вариант использования баз данных.

Полученные в ходе проведённой работы результаты позволяют перейти к разработке метода синхронизации баз данных, и программного модуля управления потоками данных.

2 Разработка метода синхронизации баз данных, и программного модуля управления потоками данных

2.1 Задание для разработки подсистемы синхронизации данных в сети передачи телеметрической информации

Синхронизация данных между устройствами может быть реализована в виде программного модуля, расположенного на каждом из устройств. Чтобы говорить о синхронизации, охарактеризуем все устройства и данные, которые предполагается хранить на них.

1. Сервер. Необходим для хранения системной информации, такой как данные аккаунтов пользователей, серийные номера контроллеров, каталог датчиков и исполнительных устройств, ip-адреса других доступных серверов, а также сценарии для контроллера, при помощи которых пользователь может задать контроллеру условия, при исполнении которых нужно предпринять какие-либо действия. Все указанные данные, кроме сценариев, представляющих собой текстовые файлы, можно хранить в базе данных.

2. Контроллер. На нём также хранится информация о пользователе, к аккаунту которого привязан контроллер, а также информация о настройках контроллера, информация о подключенных датчиках, данные, получаемые от датчиков, и сценарии для конкретного контроллера. Для хранения в базе данных также подходят все эти данные, исключая сценарии – для работы с ними необходим отдельный программный модуль.

3. Смартфон с мобильным приложением. В приложении хранятся данные пользователя и привязанных контроллеров, настройки приложения и сценарии, создаваемые пользователем. Также необходимо передавать команды и запросы пользователя из приложения на контроллер и сервер. При разработке задания на разработку было принято решение не использовать базу данных в приложении, а хранить данные иным способом.

На сервере и контроллере имеются две базы данных, между некоторыми полями которых необходима синхронизация. Данные из приложения передаются в виде запросов или файлов. Также немаловажной частью системы являются сценарии, по которым происходит опрос датчиков, и добавление данных с них в базу, а также отправка их на сервер и устройство пользователя. Таким образом, программный модуль управления сценариями одновременно является модулем управления потоками данных. Поэтому для синхронизации данных важно также работать со сценариями.

Таким образом, для создания подсистемы синхронизации данных необходимо разработать:

1. Протокол взаимодействия базы данных контроллера с интерфейсом пользователя (Мобильное приложение). Должен включать в себя такие функции, как:

1.1. При подключении контроллера к первому подключившемуся смартфону блокировка возможности редактирования другими устройствами, только просмотр информации.

1.2. Перенос сценариев, созданных пользователем, на контроллер.

1.3. Обмен данными при авторизации.

2. Протокол взаимодействия базы данных контроллера с сервером.

3. Модуль исполнения сценариев, он же модуль управления потоками данных. Должен обладать следующими функциями:

3.1. Обработка сценариев в текстовом виде.

3.2. Обработка данных, поступающих от датчиков, указанных в сценарии.

3.3. Согласно сценарию, отправка данных на устройство пользователя, на сервер, или в базу данных.

3.4. Отправка управляющих сигналов на исполнительные устройства.

2.2 Архитектура программного обеспечения контроллера

После разработки заданий на разработку различных модулей программного обеспечения контроллера сети передачи телеметрической информации, была разработана общая структурная схема ПО контроллера (Рисунок 2).

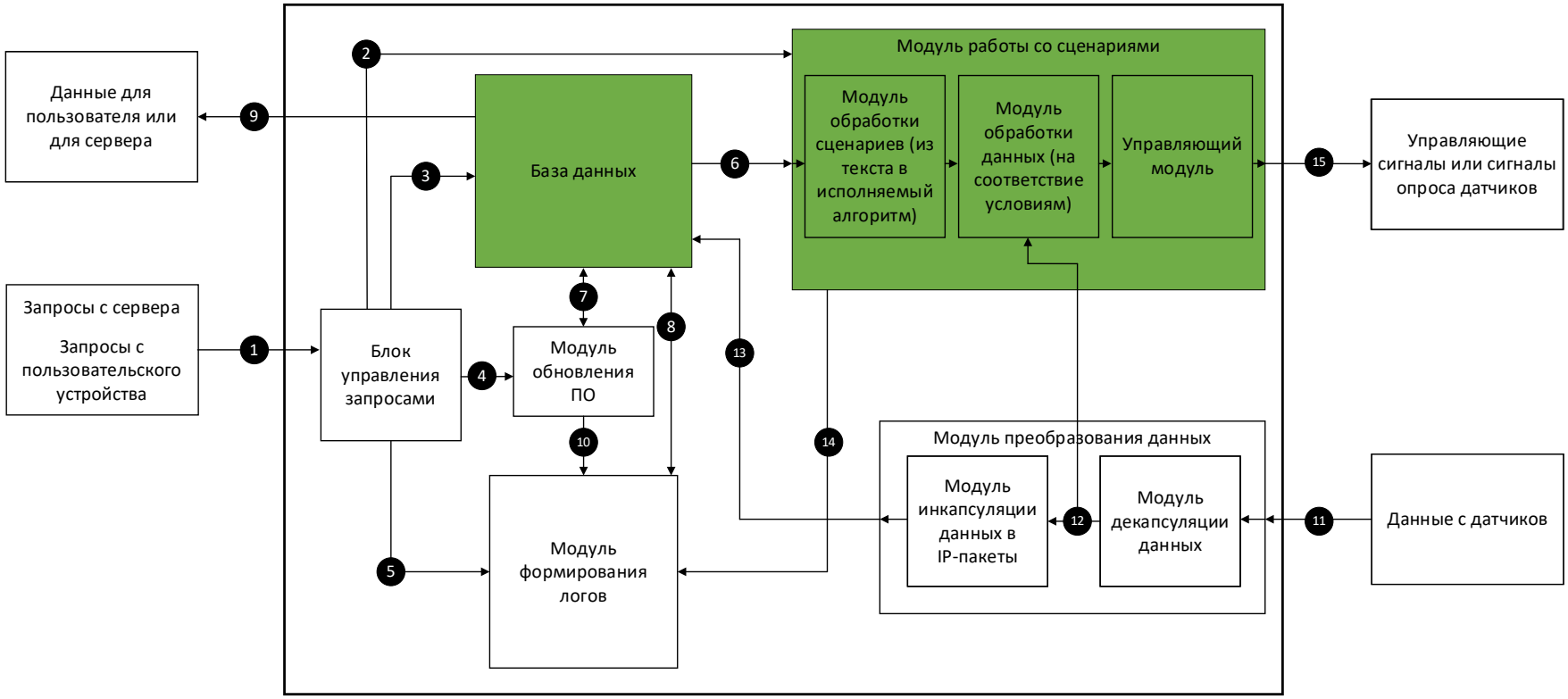
Как можно увидеть из рисунка 2, в состав программного обеспечения контроллера входят такие модули, как:

1. база данных;
2. модуль работы со сценариями;
3. блок управления запросами;
4. модуль обновления ПО;
5. модуль формирования логов;
6. модуль преобразования данных.

Все эти модули тесно связаны между собой, но для создания системы синхронизации данных основными являются база данных и модуль работы со сценариями, выделенные на рисунке 2 зелёным цветом. Рассмотрим подробнее их структуру и принцип работы.

2.2.1 Структура баз данных в сети передачи телеметрической информации

Как было сказано выше, при разработке задания на разработку программно-аппаратного комплекса передачи телеметрической информации было решено организовать базы данных на сервере и на контроллере, так как в приложении на пользовательском устройстве нет необходимости хранить данные, кроме данных учётной записи пользователя, а база данных только усложнит приложение и увеличит объёмом занимаемой им памяти.



Обозначения:

1 – Запросы с сервера или с пользовательского устройства;
2 – Запросы, адресованные модулю работы со сценариями;
3 – Запросы к базе данных;
4 – Запрос на обновление ПО контроллера;
5 – Запрос на отправку логов на сервер или пользователю;
6 – Новый сценарий из базы данных поступает в модуль работы со сценариями;
7 – Обмен данными между базой данных и модулем обновления ПО;
8 – Обмен данными между базой данных и модулем формирования логов;
9 – По запросу данные из базы данных отправляются источнику запроса;
10 – Информация об обновлении системы отправляется в модуль формирования логов;
11 – Данные от датчиков поступают в модуль преобразования данных;
12 – После удаления заголовков передающих протоколов чистые данные отправляются в модуль обработки данных;
13 – Данные, упакованные в IP-пакеты, отправляются в базу данных для хранения;
14 – Информация о работе модуля работы со сценариями отправляется в модуль формирования логов;
15 – В результате работы модуля работы со сценариями формируются управляющие сигналы для исполнительных устройств.

Рисунок 2 – Структурная схема программной архитектуры ПО контроллера

Таким образом, в разрабатываемом комплексе имеется две базы данных. Их структура представлена на рисунках 3 и 4 соответственно.

База данных на сервере (Рисунок 3) является хранилищем данных обо всех пользователях и их устройствах, то есть контроллерах, датчиках и исполнительных устройствах. Для каждого контроллера в данной базе хранится набор сценариев в текстовом виде. Также в базе данных предусмотрена отдельная таблица «Список серверов», где хранятся IP-адреса всех доступных серверов. Эта информация предназначается для обеспечения стабильного соединения между контроллером и серверами.

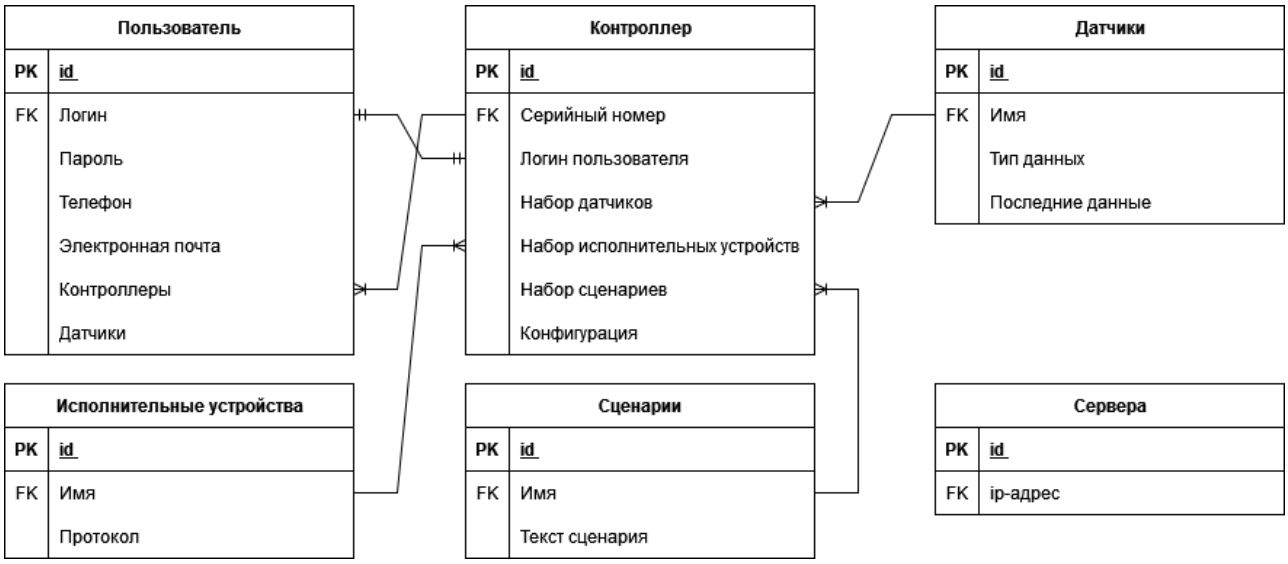


Рисунок 3 – Структура базы данных на сервере

Также на рисунке 3 обозначены требования, предъявляемые к программно-аппаратному комплексу передачи телеметрической информации: для одного абонента (одной учётной записи пользователя) доступны два контроллера, к каждому из которых можно подключить до пяти периферийных устройств и создать до пяти пользовательских сценариев.

База данных контроллера (Рисунок 4) хранит информацию о конкретном пользователе и его устройствах, а также конфигурационные данные контроллера, его состояние, сценарии в текстовом виде, и данные, поступающие с датчиков, подключенных к контроллеру.

Рисунок 4 – Структура базы данных на контроллере

Из рисунков 3 и 4 видно, что, в отличие от структуры базы данных на сервере, база данных контроллера практически не имеет связей между таблицами. Связано это с тем, что база данных на контроллере служит хранилищем данных о конкретной подсистеме внутри распределённой сети передачи телеметрической информации, состоящая из контроллера и подключенных к нему периферийных устройств. Так, в этой подсистеме имеется всего одна учётная запись пользователя, и нет необходимости сопоставлять её с конкретным контроллером, как это происходит в базе данных на сервере, где хранятся данные о множестве подобных подсистем, которые необходимо структурировать. То же относится к настройкам контроллера и к сценариям. Единственные таблицы, где необходима связь – «Датчики и исполнительные устройства» и «Данные с датчиков», так как нужно устанавливать соответствие между данными и устройством, с которого они пришли.

Таким образом, обе базы данных служит хранилищами информации различного назначения. В то время, как с данными серверной базы данных работает программное обеспечение сервера, на контроллере основную логику работы с данными выполняет модуль работы со сценариями, он же модуль управления потоками данных. Рассмотрим подробнее его принцип работы.

2.2.2 Модуль работы со сценариями

Основной функцией модуля работы со сценариями является обеспечение работы пользовательских сценариев. Также он исполняет функцию управления потоками данных. Из-за ограничений, накладываемых на систему использованием спутникового канала связи, существует необходимость передавать не все данные, а только те, что необходимы пользователю. Пользователь определяет, какие данные в приоритете, в приложении на своём устройстве при настройке и создании сценариев. Таким образом, работа сценариев напрямую затрагивает передачу данных и их синхронизацию, а модуль работы со сценариями становится логическим центром системы синхронизации данных.

Сценарии представляют собой набор действий, выполняющийся при определённых условиях. К примеру, к контроллеру подключен датчик температуры, кондиционер и обогреватель. Сценарий для такой системы, записанный обычным языком, может выглядеть так: «Если температура ниже 20 градусов, включить обогреватель». Или «Если температура выше 21 и ниже 26 градусов, выключить обогреватель и кондиционер». Условием здесь является показание датчика температуры, а выполняемыми действиями – включение или выключение кондиционера и обогревателя.

На рисунке 2 видно, что модуль работы со сценариями состоит из трёх модулей – модуля обработки сценариев, модуля обработки данных и управляющего модуля. Более подробно структура модуля работы со сценариями представлена на рисунке 5.

Сценарии поступают на контроллер с пользовательского устройства или через сервер, в виде текста определённого формата, описываемого ниже. Сценарии в текстовом виде хранятся в базе данных контроллера, откуда поступают в модуль обработки сценариев. Происходит это сразу при добавлении нового сценария. Модуль обработки сценариев ищет в тексте названия устройств, условия для выполнения, и действия, которые должны выполняться

по сценарию. Для этих действий предусмотрены программные функции, реализующие их – таким образом, сценарий из текстового вида преобразуется в исполняемый код, собираемый из готовых функций.

Рисунок 5 – Структура модуля работы со сценариями

С исполняемым сценарием работает модуль обработки данных. Его основная функция – провести проверку на соответствие условиям сценария данных, поступивших от модуля преобразования данных (Рисунок 2).

Если проверка показала выполнение условий, указанных в сценарии, сигнал об этом направляется в управляющий модуль, который запускает необходимые по сценарию действия, то есть вызывает некоторые функции для выполнения этих действий. В результате формируются управляющие сигналы, которые отправляются на нужные периферийные устройства.

Разделение модуля обработки данных и управляющего модуля является достаточно условным – при реализации они могут быть объединены в один модуль.

Таким образом, модуль работы со сценариями реализует одну из основных функций контроллера сети передачи телеметрической информации, а именно обработку телеметрической информации и выполнение сценариев пользователя. Данный программный модуль реализует основную логику работы контроллера.

Чтобы точнее определить требования, предъявляемые к модулю работы со сценариями, рассмотрим формат записи сценариев.

2.3 Формат записи сценариев

Как было сказано выше, сценарии представляют собой набор действий, выполняющийся при определённых условиях. В разрабатываемой системе имеется два вида сценариев: по умолчанию и пользовательский. Сценарии по умолчанию представляют простейшие действия для каждого из периферийных устройств – опрос датчика, включение или выключение исполнительного устройства. Они определяются после первого подключения устройства к контроллеру, и представляют собой функции. Пользовательские сценарии сложнее, в них есть условия, как было показано в примере в главе 2.2.2 данной работы. Их создаёт пользователь в соответствии со своей конфигурацией периферийных устройств и задачами, для которых он использует систему передачи телеметрической информации.

Создаются сценарии в пользовательском приложении, после чего отправляются на контроллер и в базу данных на сервере в виде текста определённого формата. Для того, чтобы модулем обработки сценариев в дальнейшем были правильно определены имена устройств, условия и необходимые действия, был разработан следующий формат сценариев:

1. Сценарий можно условно разделить на две части: блок «Если», где перечислены условия срабатывания сценария, и блок «То», где перечислены действия, которые необходимо выполнить при выполнении условий. Поэтому сценарий записывается в две строки, где первая строка – блок «Если», а вторая – блок «То». Чтобы исключить ошибки при обработке текста, каждая часть начинается с ключевого слова – «ЕСЛИ» и «ТО» соответственно.

2. Блок «ЕСЛИ»:

2.1. Ключевые слова, используемые в блоке «ЕСЛИ»: «И», «ИЛИ». Введено ограничение на количество ключевых слов во избежание избыточности сценария: «И» используется не более трёх раз, «ИЛИ» – не более двух.

2.2. После слова «ЕСЛИ» в круглых скобках записываются условия. В качестве условия может выступать результат сравнения показаний датчика с числом или строкой, или результат сравнения показаний двух датчиков.

2.3. В условиях используются знаки сравнения: <, >, =.

2.4. Имя датчика, чьи показания используются в условии, всегда стоит между открывающей круглой скобкой и знаком сравнения.

2.5. Между знаком сравнения и закрывающей круглой скобкой стоит значение, с которым сравниваются показания датчика, или имя второго датчика.

2.6. После закрывающей круглой скобки может следовать «И», «ИЛИ», либо знак перехода на новую строку. В случае, если после условия стоит «И» или «ИЛИ», далее следует следующее условие в круглых скобках, записанное по тем же правилам.

3. Блок «ТО»:

3.1. Ключевые слова, используемые в блоке «ТО»: «И», «ВКЛЮЧИТЬ», «ВЫКЛЮЧИТЬ», «ПЕРЕСЛАТЬ».

3.2. В блоке «ТО» может быть не более трёх выполняемых действий, соответственно, не более двух использований «И».

3.3. Действия записываются в квадратных скобках следующим образом: после открывающей квадратной скобки записывается название устройства, на которое необходимо подать управляющий сигнал, после чего в круглых скобках записывается действие «ВКЛЮЧИТЬ», «ВЫКЛЮЧИТЬ», или «ПЕРЕСЛАТЬ».

3.4. Действие «ПЕРЕСЛАТЬ» может быть выполнено различными способами: отправка данных на сервер, либо на телефон пользователя через СМС. Способ выполнения действия указывается после него, через запятую пишется ключевое слово «СЕРВЕР», либо номер телефона, куда нужно отправить СМС. Телефон указывается в формате 10 цифр.

4. В сценариях не используются пробелы и символы, кроме указанных выше. Исключение – названия датчиков.

Подобный формат сценария отличает сложная структура с возможностью создавать сценарии с несколькими условиями. Другие системы, которые

используют сценарии, например, системы «умный дом» от различных производителей, имеют возможность создавать простые сценарии, работающие по принципу «Если – То», без использования ветвлений «И», «Или». Таким образом, предложенный формат сценариев является более сложным, но более гибким и подходящим для автоматизации более сложных процессов.

2.4 Алгоритм работы подсистемы синхронизации данных в сети передачи телеметрической информации

В соответствии с заданием на разработку, и архитектурой программного обеспечения контроллера, описанными ранее, был разработан алгоритм работы подсистемы синхронизации данных, в которую входит база данных на контроллере, и программный модуль управления потоками данных, он же модуль работы со сценариями. Блок-схема этого алгоритма представлена на рисунке 6.

Как можно увидеть по рисунку 6, у алгоритма есть начало, но нет конца, так как программы будут работать непрерывно, пока включен контроллер. Работа модуля работы со сценариями и синхронизация баз данных происходит параллельно. Работа модуля работы со сценариями происходит в соответствии с его структурой, описанной выше: после запуска контроллера проводится проверка, существуют ли сохранённые на устройстве сценарии, и, если да, производится их запуск. В модуле обработки данных проверяется время, когда последний раз опрашивался нужный датчик – если это было достаточно давно, проводится новый опрос, в ином случае идёт работа с имеющимися данными. Затем проверяется соответствие данных, поступивших от датчиков, условиям каждого сценария, в результате которой запускаются либо не запускаются необходимые действия. После этого производится запись логов выполнения сценария, и ожидается поступление новых данных от датчиков.

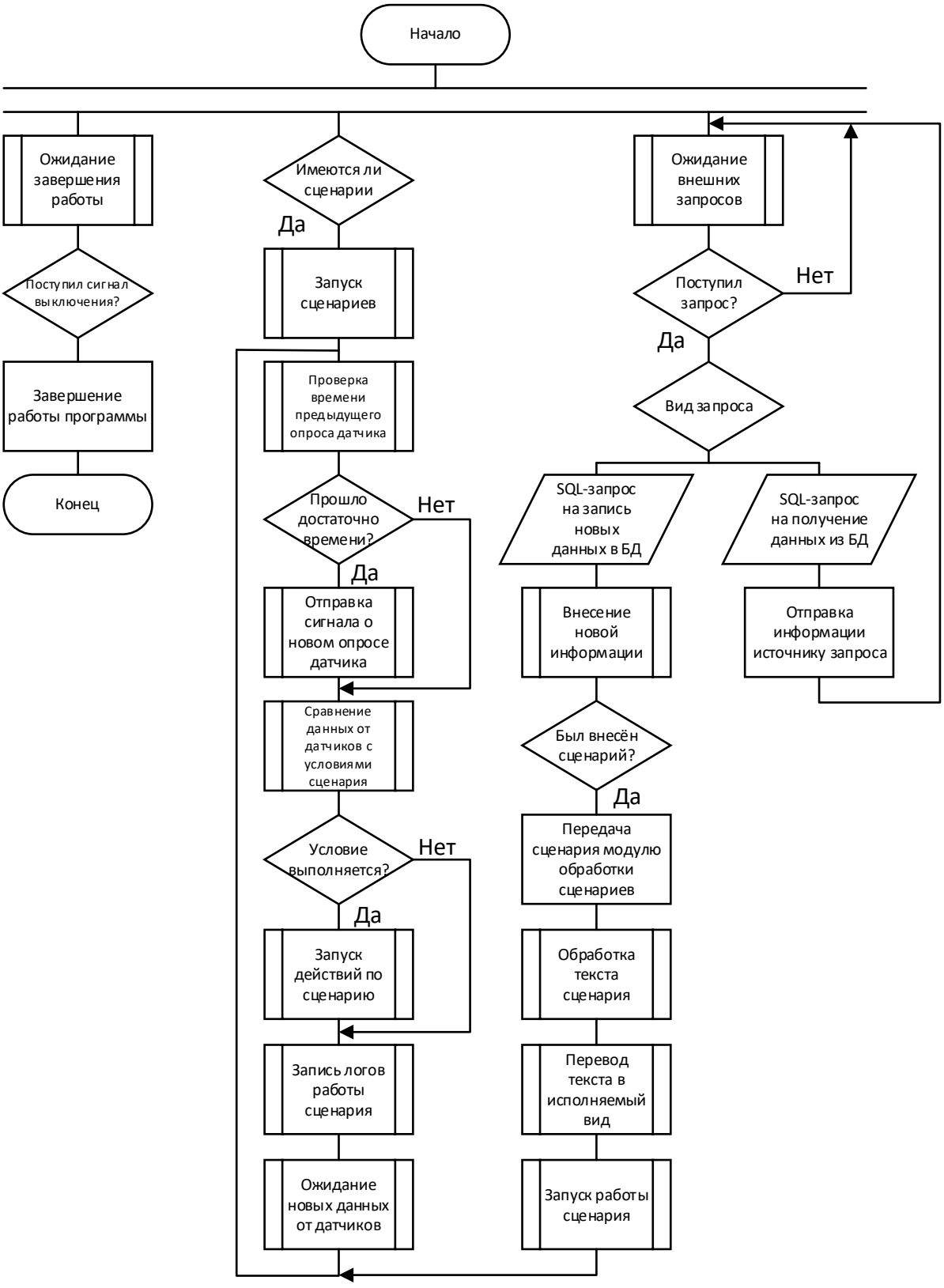


Рисунок 6 – Алгоритм работы подсистемы синхронизации данных

Синхронизация баз данных происходит по внешнему запросу с сервера. Это может быть как запрос на отправку данных на сервер, так и запрос на

внесение новых данных. Также пользователь со своего устройства может запросить информацию из базы данных контроллера. При поступлении одного из таких запросов он выполняется: либо информация из базы данных отправляется источнику запроса, либо заносится новая. При внесении новых записей в базу данных происходит проверка: не сценарий ли это. Если был добавлен новый сценарий, он передаётся модулю обработки сценариев, где преобразуется в исполняемый вид, как было описано выше. После этого сценарий начинает свою работу так же, как и другие.

Разработанный алгоритм позволяет перейти непосредственно к разработке подсистемы синхронизации данных в сети передачи телеметрической информации.

2.5 Выводы по главе

Разработано задание для дальнейшей разработки подсистемы синхронизации данных в сети передачи телеметрической информации, а также составлено задание для программного модуля управления потоками данных в системе передачи телеметрической информации.

В составе рабочей группы проекта разработки системы передачи телеметрической информации разработана архитектура программного обеспечения контроллера, определена структура баз данных, а также структура модуля управления потоками данных, определены его функции.

Разработан формат для записи сценариев в текстовом виде, что позволит структурировать работу с ними как на стороне модуля обработки сценариев на контроллере, так и на стороне пользовательского интерфейса.

На основе задания на ВКР, задания на разработку, и структуры разрабатываемых модулей разработан алгоритм работы подсистемы синхронизации данных.

Предложен метод синхронизации данных, базирующийся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации.

Разработанный метод позволяет обеспечить синхронизацию данных в системе передачи телеметрической информации в условиях передачи ограниченного объёма данных с учётом приоритетов пользователя.

Полученные результаты позволяют перейти к разработке технических решений и программных средств из состава подсистемы синхронизации данных в сети передачи телеметрической информации.

3 Разработка программных моделей и программных средств из состава подсистемы синхронизации данных в сети передачи телеметрической информации

3.1 Выбор инструментальных средств для разработки программной модели подсистемы синхронизации данных в сети передачи телеметрической информации

На основе метода синхронизации данных в системе передаче телеметрической информации, предложенного в предыдущей главе данной работы, была начата разработка программной модели подсистемы синхрониации данных в сети передачи телеметрической информации.

В качестве инструментальных стредств для разработки программной моедели были выбраны:

1. Язык программирования C++. Данный язык был выбран для разработки модулей обработки сценариев и обработки данных. Причины, которыми обусловлен выбор: универсальность и кросплатформенность языка, высокая скорость выполнения кода, имеющийся опыт программирования на данном языке.

2. Виртуальная машина Oracle VM VitualBox [17]. Данная программа позволяет создавать виртуальные машины и устанавливать на них различные операционные системы, что подходит для разработки программной модели системы синхронизации данных. Причиной, по которой была выбрана эта система виртуализации, является её бесплатное распространение и удобство интерфейса.

3. Операционная система Linux Mint 21.3 «Virginia» [18]. Это свободно распространяемый дистрибутив Linux, основанный на Ubuntu и Debian. Данная система была выбрана в качестве ОС для модели системы синхронизации ввиду бесплатного распространения, удобного интерфейса и большей, по сравнению с Ubuntu, стабильности работы на виртуальной машине VirtualBox.

4. Среда разработки Code::Blocks [19]. Свободно распространяемая кросплатформенная среда разработки. Была выбрана для разработки программной модели на языке C++ из-за кросплатформенности, свободного распространения и удобного интерфейса.

5. База данных MySQL [20]. Свободная реляционная система управления базами данных. Причины, по которым была выбрана данная СУБД: работа с реляционными базами данных, свободное распространение, большое количество документации.

После выбора данных инструментальных средств было организовано рабочее место для разработки системы синхронизации данных в сети передачи телеметрической информации, после чего была начата разработка.

3.2 Разработка программной модели

На первом этапе разрабатывается программная модель системы синхронизации данных в сети передачи телеметрической информации. Вместо контроллера используется виртуальная машина с установленной операционной системой Linux Mint.

Программная модель модулей обработки сценариев и обработки данных реализована в виде консольного приложения на языке C++. Также на виртуальной машине развёрнута база данных MySQL.

Для программной модели составлена диаграмма классов, представленная на рисунке 7. Далее подробно рассмотрены все элементы программы и их взаимодействие.

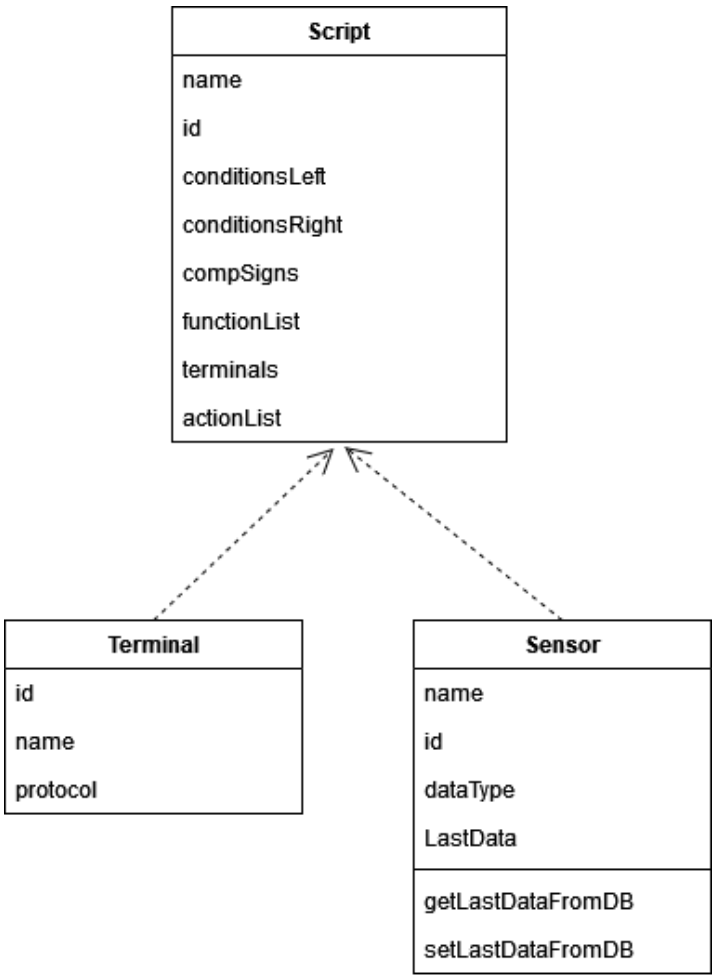


Рисунок 7 – Диаграмма классов разработанной программной модели

3.2.1 Перечень классов

На рисунке 7 видно, что в программной модели имеется три класса – Script, Sensor, Terminal. Рассмотрим подробнее каждый из них.

3.2.1.1 Класс Script

Класс Script является программным представлением сценария. Экземляры этого класса содержат данные, необходимые для работы модуля обработки данных, и создаются при работе модуля обработки сценариев.

Метод, предложенный в главе 2 данной работы, подразумевает преобразование текстового сценария в некий исполняемый вид при проходении через модуль обработки сценариев. Однако на практике такой подход

сталкивается с трудностями в реализации – ведь необходимо не только транслировать текст сценария в код, но и компилировать получаемый код во время работы модуля обработки данных, что значительно усложняет его структуру и повышает затраты памяти, что нежелательно для разрабатываемой системы. Поэтому было принято решение изменить подход к обработке сценариев. При проходждении текста через модуль обработки сценариев, его основные элементы записываются в новый экземпляр класса Script, с которым далее работает модуль обработки данных.

Были выделены основные элементы, общие для всех сценариев формата, описанного в главе 2.3 данной работы. Рассмотрим их на примере сценария Script1 (Рисунок 8).

Как можно увидеть из рисунка, сценарий состоит из трёх строк. В первой записано название, во второй – блок «ЕСЛИ», в третьей – блок «ТО».

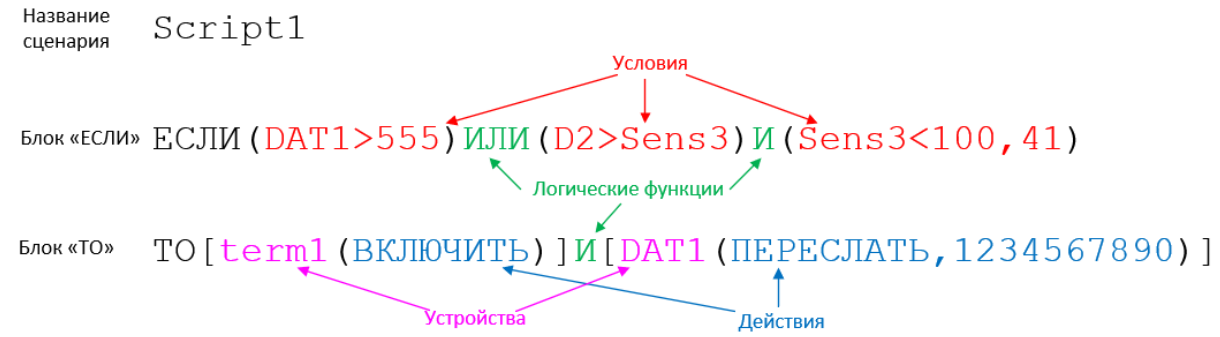


Рисунок 8 – Основные элементы сценария

Основными элементами блока «ЕСЛИ» можно назвать условия, записанные в скобках, и ключевые слова («И», «ИЛИ») между ними. При более подробнос рассмотрении видно, что условия состоят из знака сравнения и значений справа и слева от него. Основными элементами блока «ТО» являются названия устройств, с которыми должны выполняться действия и названия этих действий.

Исходя из вышеизложенного, были определеы следующие поля для класса Script:

- name – название сценария;
- id – идентификатор, или номер сценария;
- conditionsLeft – условия слева: список значений, стоящих слева от знаков сравнения в условиях блока «ЕСЛИ»;
- conditionsRight – условия справа: список значений, стоящих справа от знаков сравнения в условиях блока «ЕСЛИ»;
- compSigns – список знаков сравнения в условиях блока «ЕСЛИ»;
- functionList – список логических функций («И», «ИЛИ») в блоке «ЕСЛИ»;
- terminals – список устройств, с которыми идёт работа в блоке «ТО»;
- actionList – список действий, выполняющихся в блоке «ТО».

Для каждого из указанных полей в классе Script определён метод, позволяющий установить значение этого поля, и метод, возвращающий текущее значение данного поля. Так, метод void setCondRight(list<string> cond) устанавливает значение поля conditionsRight равным передаваемому параметру cond, а метод list<string> getCondRight() возвращает значение этого поля.

3.2.1.2 Классы Sensor и Terminal

Класс Sensor является программным представлением датчика, подключаемого к контроллеру, и необходим для удобства работы с датчиками, каждый из которых представляется в программе, как экземпляр данного класса. Класс Sensor имеет следующие поля:

- name – имя датчика;
- id – идентификатор;
- dataType – тип данных, получаемых от датчика;
- lastData – последние значение, полученное от датчика.

Класс также содержит методы, позволяющие установить значение для каждого поля, и получить эти значения. Также предусмотрены методы для записи данных датчика в базу данных, и чтения данных из БД.

Класс Terminal в целом аналогичен классу Sensor, но служит для представления исполнительных устройств, подключаемых к контроллеру. Каждый экземпляр этого класса является представлением одного исполнительного устройства. Так как исполнительное устройство в данной программной модели не может отправлять данные, а может только получать управляющие сигналы, класс содержит поля name и id, в которых хранятся соответственно имя и идентификатор устройства, а также поле protocol, в котором хранится тип протокола, используемый для подключения конкретного устройства. Этот класс содержит методы, позволяющие установить значение для каждого поля, и методы, возвращающие эти значения.

Рассмотрев все классы, задействованные в разрабатывамой программной модели, перейдём к рассмотрению функций, реализующих работу модулей обработки сценариев и обработки данных.

3.2.2 Алгоритм работы модуля обработки сценариев

Модуль обработки сценариев, как было сказано выше, выполняет одну задачу: создаёт экземпляр класса Script, в поля которого вносит информацию из сценария, приходящего в текстовом виде. В разработанной программной модели данный модуль реализован в виде функции textToScript.

Полностью прототип данной функции выглядит так:

list<Script> textToScript(list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals);

Функция textToScript принимает в качестве аргументов список существующих сценариев, а также списки датчиков и испольнительных устройств. Возвращает она обновлённый список сценариев, с добавлением нового элемента.

Подробнее рассмотрим алгоритм работы данной функции (Рисунок 9).

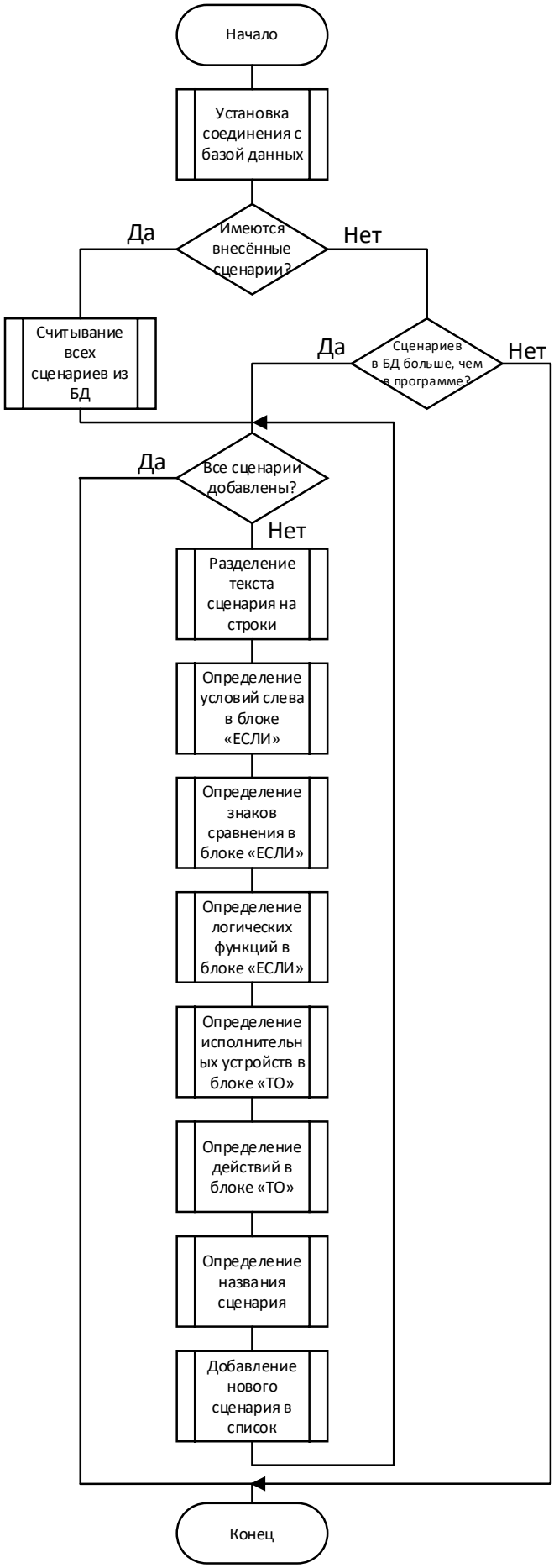


Рисунок 9 – Алгоритм работы функции textToScript

Вначале устанавливается соединение с базой данных. Если установить соединение не удалось, будет показано собщение об ошибке. В случае успеха будет выполнен SQL-запрос к базе данных на предоставление информации о хранящихся сценариях. Выполняется сверка имён существующих сценариев с базой данных. В случае, если найден новый сценарий, его текст, разбитый на строки, начинает обрабатываться.

Каждая строка обрабатывается последовательно. Проверяются ключевые слова в начале строки («ЕСЛИ», «ТО»). Если ключевые слова не найдены, строка считается названием сценария, и заносится в соответствующее поле экземпляра класса Script.

При обнаружении слова «ЕСЛИ» необходимо определить количество условий, которое равно количеству знаков сравнения, сами знаки сравнения, условия слева, условия справа и функции «И», «ИЛИ» между условиями. Их поиск происходит в несколько этапов: сначала опередеяются условия слева и знаки сравнения, потом – условия справа, и на последнем этапе – функции «И», «ИЛИ». При этом условиями слева считаются символы строки, находящиеся между открывающей скобкой «(» и любым из знаков сравнения «>, <, =». Условиями справа считаются символы, находящиеся между закрывающей скобкой «)» и знаком сравнения. Функциями же считаются символы, находящиеся между закрывающей и открывающей скобками.

На этом этапе все данные, получаемые из текста, представлены в виде строк, что позволяет обеспечить возможность преобразования сценария в объект класса Script независимо от содержащихся в нём данных. Их дальнейшая обработка будет производиться в модуле обработки данных.

При обнаружении слова «ТО» необходимо определить названия устройств, с которыми должны выполняться действия в случае выполнения условияйсценарий, и названия этих действий. Как и в случае с блоком «ЕСЛИ», поиск проходит поэтапно. Сначала выявляются названия устройств. Они находятся между открывающей квадратной «[» и открывающей круглой «(» скобками. После определяются действия, которые находятся между

открывающей «(» и закрывающей круглой «)» скобками. Определять функции между различными действиями не имеет смысла, поскольку, согласно формату сценария, в блоке «ТО» может использоваться только ключевое слово «И».

После того, как обработаны все строки, и все значения занесены в нужные поля, созданный экземпляр класса Script добавляется в конец списка существующих сценариев, который поступил в качестве аргумента функции textToScript, и список возвращается в точку вызова функции.

3.2.3 Алгоритм работы модуля обработки данных

Модуль обработки данных выполняет проверку условий сценария на выполнение, и в случае, если условия выпоняются, запускает необходимые по сценарию действия. В разработанной программной модели модуль обработки данных реализован в виде двух функций: ScriptsWorking и functionChecking. Рассмотрим подробнее каждую из них.

Прототип функции ScriptsWorking выглядит следующим образом:

void ScriptsWorking(list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals);

Данная функция принимает в качестве аргументов список существующих сценариев, а также списки подключенных к контроллеру датчиков и испольнительных устройств. Функция работает постоянно, вызываясь на каждой итерации бесконечного цикла, независимо от работы модуля обработки сценариев.

На рисунке 10 предствлен алгоритм работы функции ScriptsWorking.

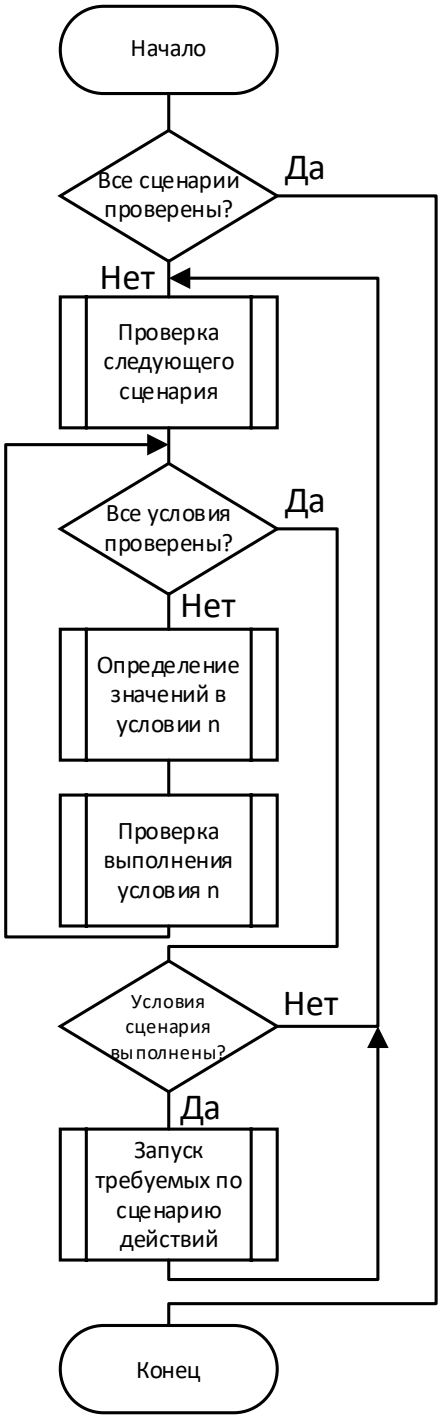


Рисунок 10 – Алгоритм работы функции ScriptsWorking

В этой функции проверяется каждое условие, записанное в сценарии в скобках в блоке «ЕСЛИ». Проверка проводится последовательно для каждого условия. Из текущего объкта класса Script выбирается одно из условий слева и соответствующее ему условие справа. Далее определяются конкретные значения этих условий в числовом виде.

В сценарии условие может быть числом, либо именем устройства. Во втоором случае для сравнения используются последние данные, полученные от этого устройства. При этом хотя бы одно из условий в проверяемой паре обязательно является именем – иначе сравнение двух числовых значений не несёт смысла в рамках данной системы. В функции ScriptsWorking проверяется, какое из условий содержит имя датчика. Поиск проходит по объектам класса Sensor, каждый из которых содержит информацию о последних полученных данных. При нахождении нужного датчика дальнейшая работа идёт с этими данными. Если же условте является числовым значением, число из типа данных string переводится в тип данных float.

Полученные значения сравниваются при помощи оператора выбора switch: происходит выбор знака сравнения, стоящего между условием слева и справа по сценарию. В результате сравнение получается одно из значений типа bool: true (истина) или false (ложь). Это значение помещается в список full_cond.

После проверки всех пар условий слева и справа заполненный список full_cond передаётся в качестве аргумента функции functionChecking. Задача этой функции – найти решение логического выражения, которое получается из значений списка full_cond и функций «И», «ИЛИ», стоящих между условиями по сценарию. Эта функция вызывается из функции ScriptsWorking.

Прототип функции functionChecking выглядит следующим образом:

bool functionChecking(list<bool> full_cond, list<string> functionList);

В качестве аргументов она принимает список значений выполнения условий, и список логических функций в строковом виде.

На рисунке 11 представлен алгоритм работы функции functionChecking.

При разработке формата сценариев определено, что все функции «И», «ИЛИ» в сценариях должны исполняться по порядку, согласно порядку действий в алгебре логики. Таким образом, первым должно выполняться действие конъюнкции (логическое И, соответствует функции «И» в сценарии), а затем действие дизъюнкции (логическое ИЛИ, соответствует функции «ИЛИ» в сценарии).

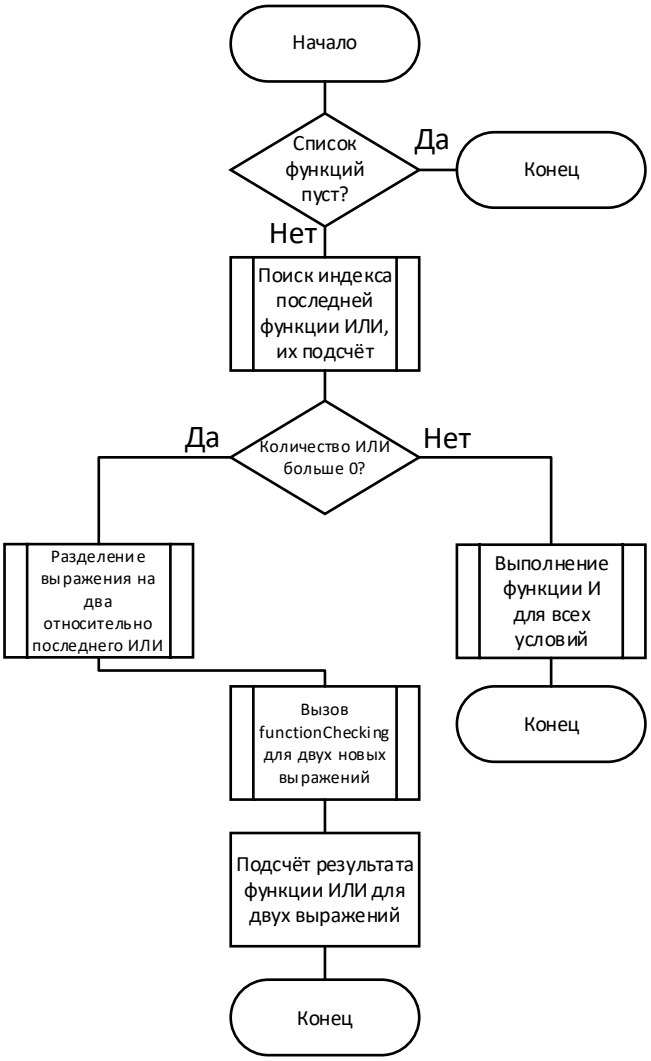


Рисунок 11 – Алгоритм работы функции functionChecking

Функция functionChecking является рекурсивной, и работает по следующему принципу:

1. Определяется количество функций «ИЛИ» и позиция последней из них;

2. Если функций «ИЛИ» в списке нет, выполняется операция «И» для всех значений из списка full_cond;

3. Если «ИЛИ» есть в сценарии, список значений условий и список функций разделяются каждый на два – слева от последнего «ИЛИ» и справа от него. Далее производится рекурсивный вызов функции functionChecking для кадой половины выражения.

4. Если в результате рекурсии осталось только одно значение, оно возвращается в точку вызова функции.

5. Когда найдены ответы для каждой половины выражения, с ними производится операция «ИЛИ», результат которой является искомым значением выражения.

Если функция functionChecking вернула значение true, сценарий выполняется, и нужно переходить к болку «ТО». В функции ScriptsWorking производится проверка списка действий каждого сценария. В зависимости от найденных ключевых слов – «ВКЛЮЧИТЬ», «ВЫКЛЮЧИТЬ», «ПЕРЕСЛАТЬ» – вызываются необходимые функции для нужных устройств, обеспечивая, таким образом, перемещение потоков данных и управляющих сигналов между устройствами в сети передачи телеметрической информации.

После разработки модуля обработки сценариев и модуля обработки данных было выполнено тестирование программной модели.

3.3 Взаимодействие программной модели с базой данных

В разработанной программной модели база данных, установленная локально на виртуальной машине, применяется для хранения информации о датчиках, исполнительных устройствах и сценариях.

В рамках разработки программной модели системы синхронизации данных на виртуальную машину была установлена СУБД MySQL, и развёрнута локальная база данных contr_db, в которой созданы таблицы Sensor, Terminal, Scripts, содержащие данные о датчиках, исполнительных устройствах и сценариях соответственно. На рисунке 12 представлена структура тестовой базы данных.

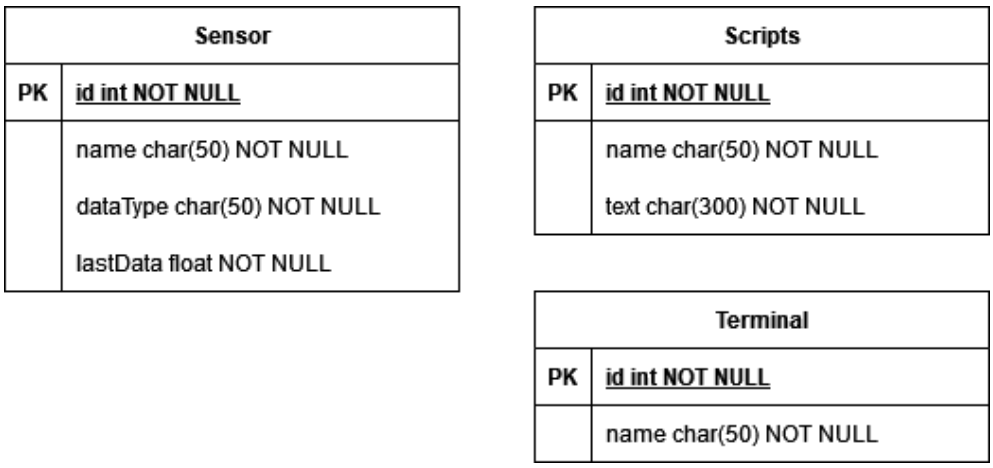


Рисунок 12 – Структура тестовой базы данных

Программная модель модуля управления потоками данных взаимодействует с БД следующим образом:

1. После запуска программной модели происходит инициализация датчиков и исполнительных устройств, данные о которых хранятся в БД. Под инициализацией в данном случае понимается считывание данных из БД и формирование на их основе списков объектов классов Sensor и Terminal для датчиков и исполнительных устройств соответственно.

2. После запуска программной модели происходит формирование списка сценариев, данные о которых также считываются из БД.

3. При добавлении нового сценария в базу данных, он автоматически считывается модулем обработки сценариев, и добавляется в список существующих сценариев.

Таким образом, текущая конфигурация базы данных позволяет осуществлять хранение данных о датчиках, исполнительных устройствах и сценариях, а также обеспечивать обмен этими данными межжду различными программными модулями.

В дальнейшем планируется соединение локальной базы данных с базой данных на сервере, и организация обмена данными между ними в рамках создания системы синхронизации данных в сети передачи телеметрической информации.

3.4 Выводы по главе

Выбраны инструментальные средства для разработки программной модели системы синхронизации данных в сети передачи телеметрической информации, в т.ч. язык программирования C++, среда разработки Code::Blocks, виртуальная машина Oracle VM VitualBox , операционная система Linux Mint 21.3 «Virginia», база данных MySQL.

Разработана программная модель, развёрнутая на виртуальной машине.

Разработаны классы Script, Sensor, Terminal для представления в программе сценариев, датчиков и исполнительных устройств соответственно.

Разработан модуль обработки сценариев и реализован в виде функции textToScript, создающей объект класса Script на основе текста сценария, считываемого из базы данных.

Разработан модуль обработки данных и реализован в виде двух функций: ScriptsWorking и functionChecking. Эти функции выполняют проверку условий сценария, и в случае их выполнения, запускают необходимые действия.

На модели проверен метод взаимодействия разработанной программы с базой данных. База данных успешно применяется для хранения информации о сценариях, датчиках и исполнительных устройствах.

Полученные результаты разработки позволяют перейти к тестированию разработанных программных моделей и программных средств.

4 Тестирование разработанных программных моделей и программных средств

Для тестирования разработанной программной модели был составлен набор тестовых данных, включающий четыре сценария, четыре датчика и два исполнительных устройства. Тексты сценариев приведены в приложении Б, а информация об устройствах – в таблицах 2 и 3.

Таблица 2 – Тестовый набор датчиков

id	Имя	Тип данных	Последние данные
1	DAT1	int	0
2	D2	float	0
3	Sens3	float	0
4	Датчик открытия двери	int	0

Таблица 3 – Тестовый набор исполнительных устройств

№	id	Имя
1	11	term1
2	12	TERM2

Тестирование программной модели системы синхронизации данных проводилось поэтапно, по мере завершения этапов её разработки. Описание всех этапов тестирования приведено ниже.

4.1 Тестирование алгоритмов работы модуля управления потоками данных

На первом этапе проводено тестирование консольного приложения без использования базы данных. Тестовые сценарии хранятся в текстовых файлах. Цель этого этапа тестирования: проверка правильности работы разработанных алгоритмов модуля обработки сценариев и модуля обработки данных. Тестирование проводилось в операционной системе Windows 10.

Согласно заданию на разработку, программное обеспечение для контроллера сети передачи телеметрической информации не предполагает интерфейса пользователя. Но для удобства отладки и тестирования программы было разработано меню консольного приложения (Рисунок 13).

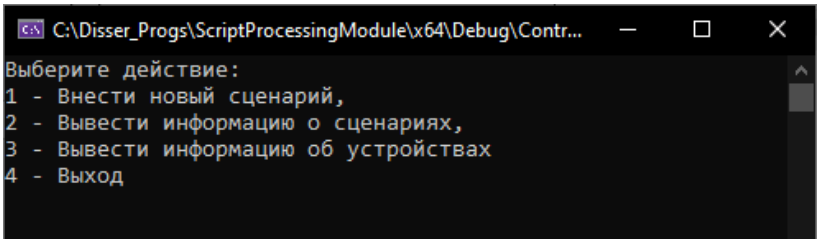


Рисунок 13 – Меню в консольном приложении

В меню имеется четыре пункта:

1. Внести новый сценарий – запускает функцию textToScript для внесения нового сценария из текстового файла;

2. Вывести информацию о сценариях – выводит на экран информацию обо всех существующих объектах класса Script;

3. Вывести информацию об устройствах – выводит на экран информацию обо всех существующих объектах классов Sensor и Terminal;

4. Выход – завершает работу приложения.

После выбора пункта 1 необходимо внести название текстового файла, содержащего сценарий, после чего происходит считывание данных из файла, при условии его существования, и формирование объекта класса Script. На рисунке 14 показано добавление сценария Script1.

При выборе пункта 2 (Рисунок 15) будут выведены на экран все элементы добавленных ранее сценариев.

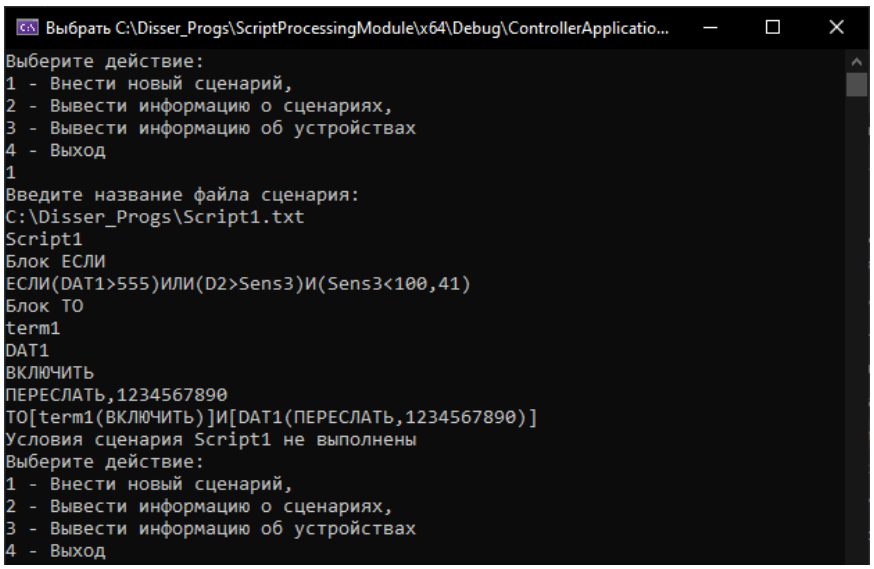


Рисунок 14 – Добавление сценария Script1

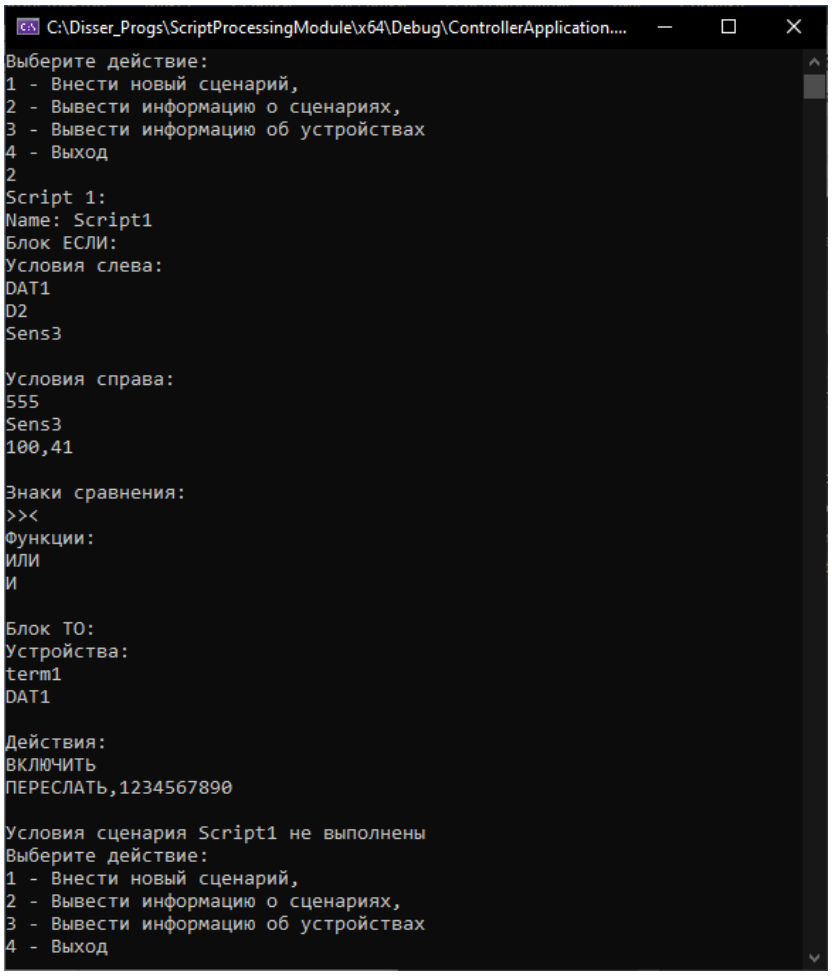


Рисунок 15 – Информация о внесённом сценарии Script1

При выборе пункта 3 на экран выводится вся имеющаяся информация о датчиках и исполнительных устройствах (Рисунок 16).

Из таблицы 2 видно, что значения последних данных для каждого датчика по умолчанию равно 0. На текущем этапе тестирования генерация данных от датчиков отсутствует. Однако проверить работу сценариев возможно, задавая различные данные вручную. На рисунке 17 представлена ситуация, когда в программу внесены два сценария, условия для одного из них не выполняются, а для второго – выполняются. Сообщения об этом выводятся на экран. Также выводится информация о том, какие именно действия должны быть выполнены по сценарию.

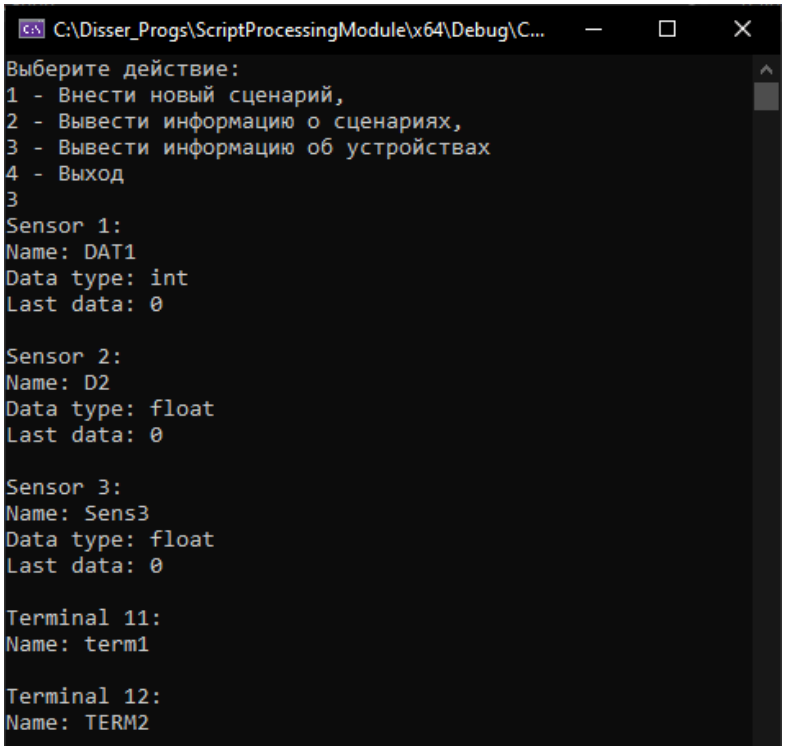


Рисунок 16 – Информация об имеющихся устройствах

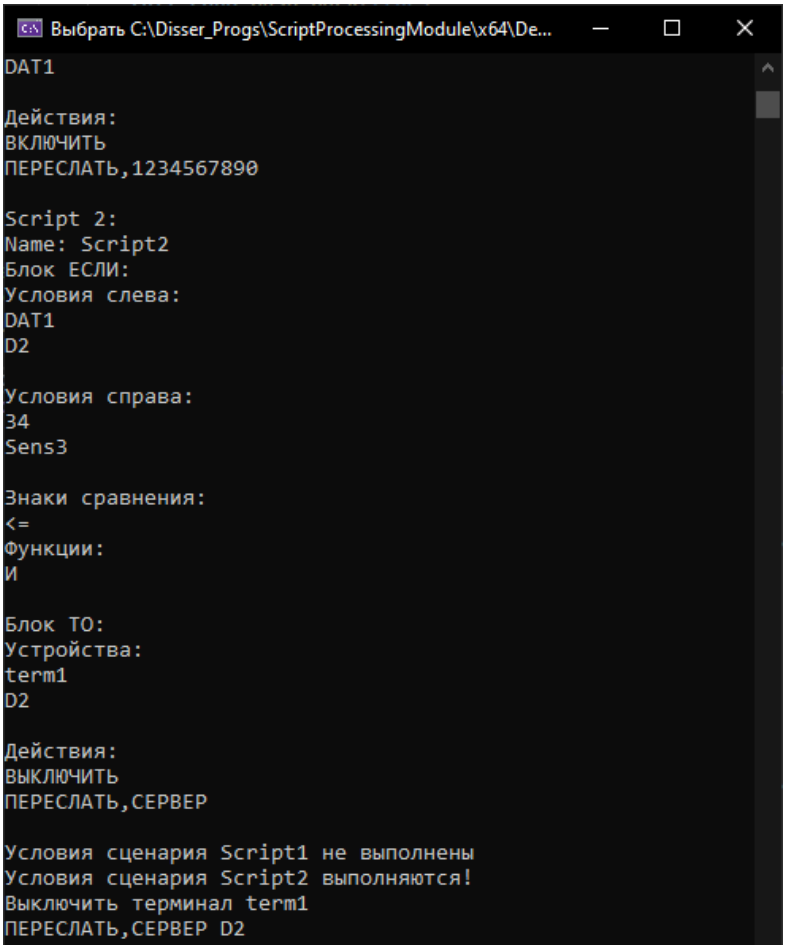


Рисунок 17 – Информация о внесённом сценарии Script2 и сообщение о выполнении условий сценария

Тестирование на данном этапе показало, что все алгоритмы работают исправно: сценарии корректно преобразуются в объекты класса Script модулем обрабоки сценариев, условия выполнения сценариев корректно проверяются модулем обработки данных. Эти результаты позволили перейти ко второму этапу тестирования.

4.2 Тестирование программной модели с использованием базы данных

Второй этап тестирования программной модели: тестирование с использованием базы данных. Этот этап тестирования проводился на

виртуальной машине с программным обеспечением, описанным в разделе 3.1 данной пояснительной записки.

Тестовые данные, приведённые в таблицах 2 и 3, а также в приложении Б, были внесены в базу данных. Программная модель была переработана так, чтобы при её запуске из базы данных считывались данные о датчиках, исполнительных устройствах и сценариях. При помощи функций sensInit() и termInit() данные о датчиках и исполнительных устройствах соответственно считываются из базы данных и формируют объекты классов Sensor и Terminal. Данные о сценариях преобразуются в объект класса Script при помощи функции textToScript. На рисунке 18 показано автоматическое добавление данных о датчиках, исполнительных устройствах, и добавление четырёх сценариев из базы данных сразу после запуска приложения.

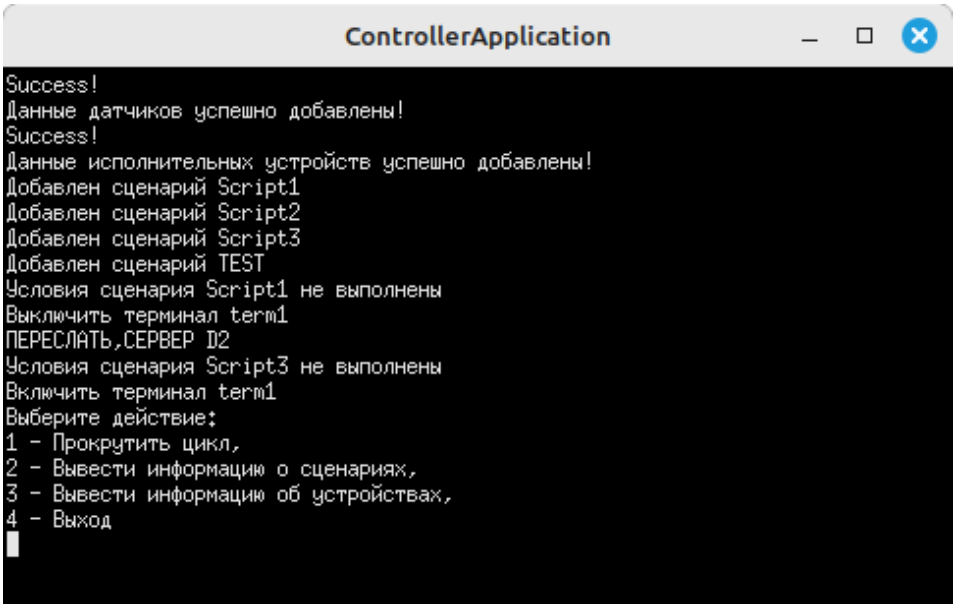


Рисунок 18 – Добавление информации из базы данных

Для удобства тестирования было оставлено меню с выбором вариантов действий, однако вариант 1 был изменён с «Внести новый сценарий» на «Прокрутить цикл». В этом варианте не производится никаких действий, он нужен только для запуска новой итерации бесконечного цикла while, в котором

выполняются все функции: отображение меню, выполнение функций textToScript и ScriptsWorking.

Так как программа работает в бесконечном цикле, на каждой его итерации производится проверка, не добавлен ли в базу данных новый сценарий. Если это так, сценарий автоматически считывается из базы данных и преобразуется в объект класса Script. На рисунке 19 видно, что при запуске программы из базы данных прочитаны и добавлены сценарии Script1 и Script2, а после был добавлен сценарий TEST. Этот сценарий был добавлен в базу данных после запуска программы, но до выбора варианта 1 в меню.

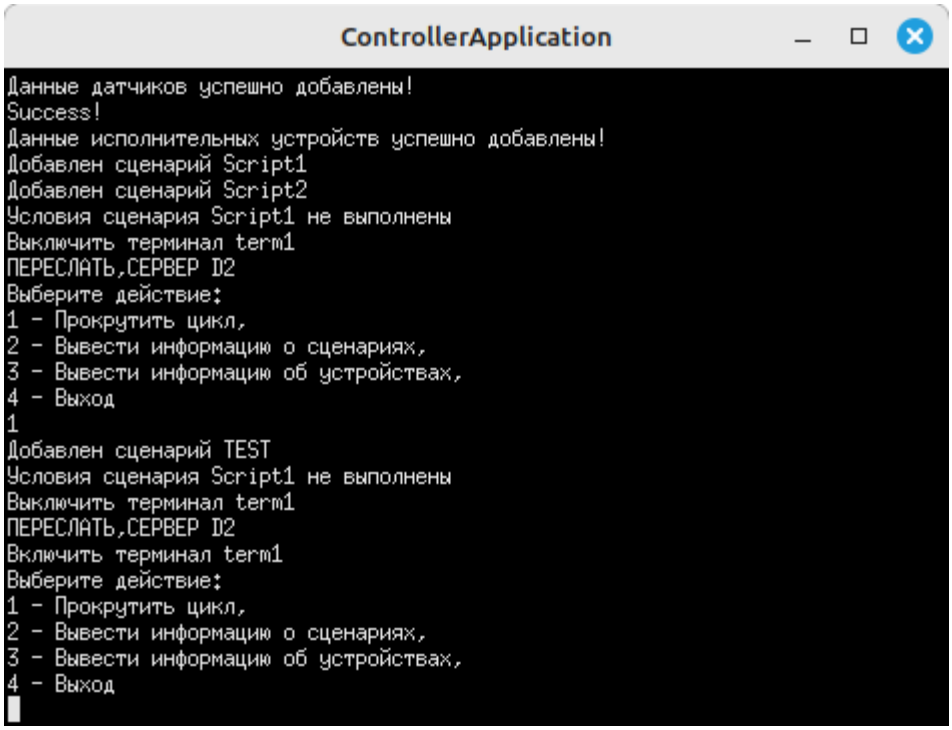


Рисунок 19 – Обновление списка сценариев с обновлением базы данных

Таким образом, было проверено взаимодействие программной модели с локальной базой данных. Полученные результаты позволили перейти к третьему этапу тестирования.

4.3 Тестирование разработанной программной модели совместно с модулем преобразования данных

На третьем этапе тестирования проводилась проверка совместимости разработанной программной модели с программной моделью модуля преобразования данных. Одной из функций этого модуля является получение данных от датчиков, их обработка и сохранение в поле lastData соответствующего объекта класса Sensor, а также обновление информации в базе данных.

На рисунке 20 можно увидеть результат совместной работы двух модулей.

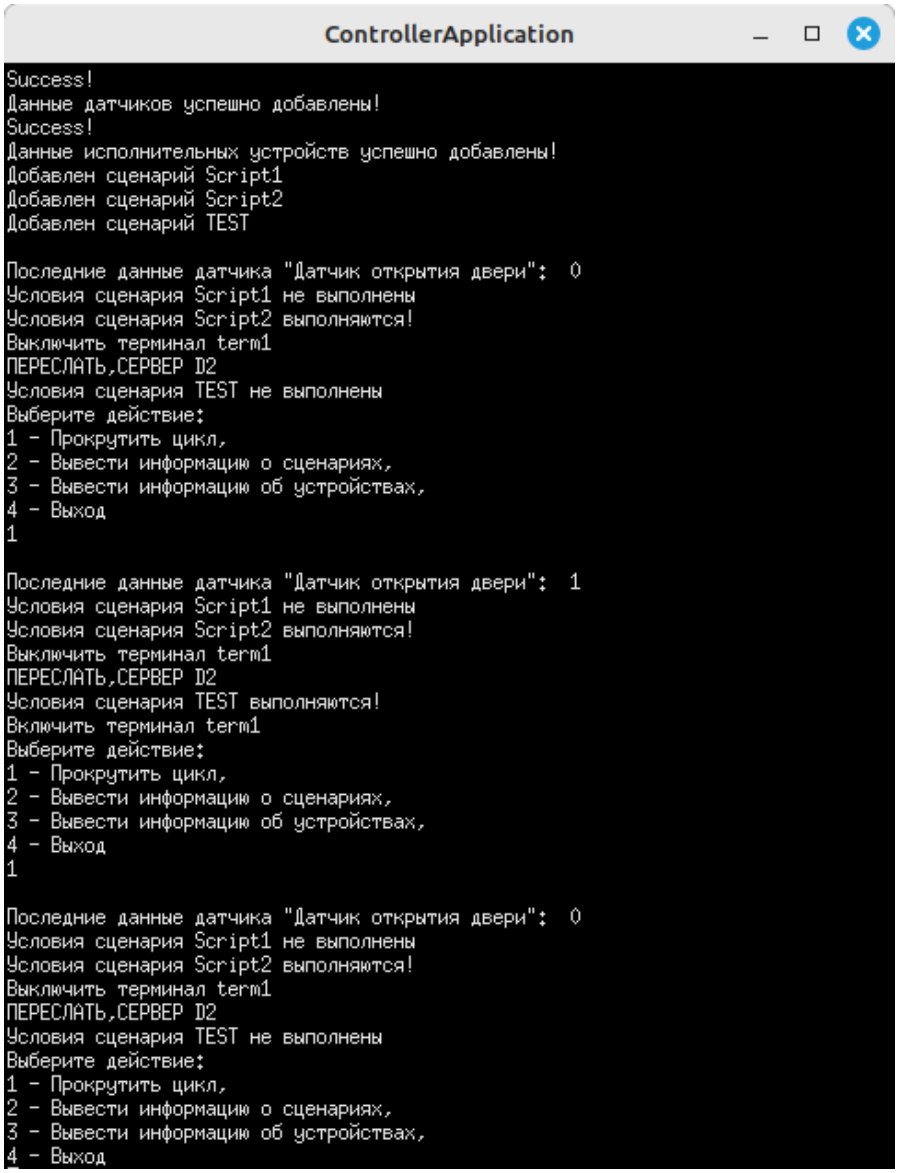


Рисунок 20 – Выполнение сценария в зависимости от данных датчика

Данные с датчика, физически подключенного к компьютеру при помощи беспроводного протокола ZigBee, принимаются модулем преобразования данных в режиме рельного времени, информация о чём выводится на экран. Так как вместе с этим последние данные датчика обновляются, меняется результат работы функции ScriptsWorking, проверяющей условия выполнения сценария. В тесте на рисунке 20 сценарий TEST выполняется, если значение данных датчика открытия двери равняется 1.

Таким образом, тестирование разработанной программной модели показало положительные результаты, и является совместимым с другими модулями из состава программного обеспечения контроллера сети передачи телеметрической информации.

4.4 Выводы по главе

Проведено тестирование разработанной программной модели. Для этого составлен набор тестовых данных, включающих тексты сценариев, описание датчиков и исполнительных устройств.

Тестирование проводилось в несколько этапов, в соответствии с ходом процесса разработки программной модели. Среди этапов можно выделить:

1. тестирование алгоритмов работы модуля управления потоками данных;
2. тестирование программной модели с использованием базы данных;
3. тестирование разработанной программной модели совместно с модулем преобразования данных.

На первом этапе тестирования установлено, что разработанные алгоритмы работы функций, входящих в состав модуля управления потоками данных, работают корректно, в соответствии с заданием на разработку.

На втором этапе тестирования установлено, что разработанная программная модель корректно работает с локальной базой данных, что

позволяет обеспечить управление потоками данных и их синхронизацию в системе передачи телеметрической информации.

На третьем этапе тестирования установлено, что программная модель модуля управления потоками данных совместима с программной моделью модуля преобразования данных: таким образом, при изменении последних данных от датчиков меняется результат проверки условий сценария.

Из вышесказанного следует, что разработанная программная модель соответствует требованиям, предъявляемым к подсистеме синхронизации данных в системе передачи телеметрической информации, и изложенным в главах 1 и 2 данной пояснительной записки.

Тем не менее, в процессе тестирования был выявлен и ряд недостатков разработанной программной модели:

1. отсутствие связи с базой данных на сервере;

2. отсутствие шифрования в локальной базе данных;

3. отсутствие ведения журнала событий для базы данных и модуля работы со сценариями.

Все эти недостатки могут быть исправлены на следующих этапах работы над системой синхронизации данных. Полученные выводы позволяют сформировать перечень основных требований по дальнейшему развитию разрабатываемой системы.

ЗАКЛЮЧЕНИЕ

В процессе реализации проекта поэтапно решались определенные по результатам анализа задания на ВКР задачи. На начальном этапе был проведён аналитический обзор предметной области исследования. Рассмотрены несколько методов и стандартов синхронизации данных, и определены критерии сравнения различных методов синхронизации данных. В результате проведённого анализа установлено, что наиболее подходящим для разработки системы синхронизации в составе программно-аппаратного комплекса передачи телеметрической информации является вариант использования баз данных, а также определены требования к разрабатываемой системе.

На втором этапе работы предложен метод синхронизации данных, базирующийся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации. Разработанный метод позволяет обеспечить синхронизацию данных в системе передачи телеметрической информации в условиях передачи ограниченного объёма данных с учётом приоритетов пользователя.

На третьем этапе работы над ВКР выбраны инструментальные средства для разработки программной модели системы синхронизации данных в сети передачи телеметрической информации. Разработана программная модель, включающая в себя:

- классы Script, Sensor, Terminal для представления в программе сценариев, датчиков и исполнительных устройств соответственно;

- модуль обработки сценариев, представленный в виде функции textToScript;

- модуль обработки данных, представленный в виде двух функций: ScriptsWorking и functionChecking.

- базу данных MySQL, содержащую таблицы Sensor, Terminal, Scripts, содержащие данные о датчиках, испольнительных устройствах, сценариях соответственно.

На модели проверен метод взаимодействия разработанной программы с базой данных.

Проведено три этапа тестирования разработанной программной модели: тестирование алгоритмов работы модуля управления потоками данных, тестирование программной модели с использованием базы данных, тестирование разработанной программной модели совместно с модулем преобразования данных. Каждый этап показал соответствие разработанной программной модели требованиям, определённым заданием на ВКР и заданием на разработку. Полученные результаты тестирования можно использовать при разработке дальнейшего плана модификации системы синхронизации данных.

Таким образом, все задачи, поставленные на первом этапе выполнения ВКР решены, что позволяет сделать вывод о достижении цели работы.

Предполагаемой научной новизной работы является предложенный метод синхронизации данных, базирующийся на разработанном алгоритме работы модуля управления потоками данных в сети передачи телеметрической информации. Этот метод позволяет обеспечить синхронизацию данных в системе передачи телеметрической информации в условиях передачи ограниченного объёма данных с учётом приоритетов пользователя.

СПИСОК СОКРАЩЕНИЙ

БД	– База данных
ПО	– Программное обеспечение
СУБД	– Система управления базами данных
ID	– Identificator
IP	– Internet Protocol
GSM	– Groupe Spécial Mobile
HTTP	– HyperText Transfer Protocol
LAN	– Local area network
MD5	– Message Digest 5
OBEX	– OBject EXchange
OMA	– Open Mobile Alliance
OMA DS	– OMA Data Synchronization
SQL	– Structured Query Language
SyncML	– Synchronization Markup Language
VM	– Virtual machine
WSP	– Wireless Session Protocol
XML	– eXtensible Markup Language

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Бирюков Н.Л., Триска Н.Р. Синхронизация сетей связи с синхронным и асинхронным режимами передачи: опыт и проблемы / Н.Л. Бирюков, Н.Р. Триска // Электросвязь. – 2007. – №10. – URL: http://www.oc.ru/media/el_13_bir/ (дата обращения: 06.07.2023)

2. Системы сбора телеметрических данных. – URL: https://esseo.ru/sistemy-sbora-telemetricheskih-dannyh (дата обращения: 06.07.2023)

3. Бирюков Н.Л., Триска Н.Р. Сети синхронизации: сценарии взаимодействия. / Н.Л. Бирюков, Н.Р. Триска – URL: http://citforum.ru/nets/articles/synchro/ (дата обращения: 08.07.2023)

4. Демиш В.О., Пищик Б.Н. Синхронизация данных на мобильных платформах / В.О. Демиш, Б.Н. Пищик // Вестник НГУ. – Серия: Информационные технологии. – 2013. – №4. – URL: https://cyberleninka.ru/article/n/sinhronizatsiya-dannyh-na-mobilnyh-platformah (дата обращения: 08.07.2023).

5. Open Mobile Alliance Specifications. – URL: https://technical.openmobilealliance.org/index.html (дата обращения: 08.07.2023).

6. DS Protocol. Approved Version 2.0 – 19 Jul 2011. – URL: https://www.openmobilealliance.org/release/DS/V2_0-20110719-A/OMA-TS-DS_Protocol-V2_0-20110719-A.pdf (дата обращения: 08.07.2023).

7. SyncML Sync Protocol, version 1.0. – URL: https://www.openmobilealliance.org/tech/affiliates/syncml/syncml_protocol_v10_20001207.pdf (дата обращения: 08.07.2023).

8. Тюков А.П. Подходы к синхронизации данных при централизованном контроле систем управления микроклиматом в коммерческих зданиях / А.П. Тюков // Современные проблемы науки и образования. – 2012. – № 6. – URL: https://science-education.ru/ru/article/view?id=7954 (дата обращения: 06.07.2023).

9. Синхронизация баз данных служб Analysis Services. – URL: https://learn.microsoft.com/ru-ru/analysis-services/multidimensional-

models/synchronize-analysis-services-databases?view=asallproducts-allversions (дата обращения: 08.07.2023).

10. ApexSQL Data Diff. – URL: https://www.apexsql.com/sql_tools_datadiff.aspx (дата обращения: 08.07.2023).

11. AutosyncDB. Сравнение и синхронизация схем баз данных SQL Server. – URL: https://autosyncdb.com/ru/ (дата обращения: 08.07.2023).

12. Демиш В.О., Пищик Б.Н. Автономная работа android-приложений и алгоритмы синхронизации данных / В.О. Демиш, Б.Н. Пищик // Вестник НГУ. – Серия: Информационные технологии. – 2014. – №3. – URL: https://cyberleninka.ru/article/n/avtonomnaya-rabota-android-prilozheniy-i-algoritmy-sinhronizatsii-dannyh (дата обращения: 08.07.2023).

13. Dropbox : сайт. – URL: http://www.dropbox.com (дата обращения: 08.07.2023).

14. Google Drive : сайт. – URL: https://www.google.com/intl/ru_ru/drive/ (дата обращения: 08.07.2023).

15. Облако Mail.ru : сайт. – URL: https://cloud.mail.ru/home/ (дата обращения: 08.07.2023).

16. Бусенков А.А., Багажков Д.И., Чернов В.В., Панов А.И., Башмуров Н.А. Разработка алгоритма и программная реализация средства защиты персональных данных в облачных хранилищах / А.А. Бусенков, Д.И. Багажков, В.В. Чернов, А.И. Панов, Н.А. Башмуров // Инновации и инвестиции. – 2021. – №12. – URL: https://cyberleninka.ru/article/n/razrabotka-algoritma-i-programmnaya-realizatsiya-sredstva-zaschity-personalnyh-dannyh-v-oblachnyh-hranilischah (дата обращения: 08.07.2023).

17. VirtualBox : сайт. – URL: https://www.virtualbox.org/ (дата обращения: 11.04.2024).

18. Linux Mint : сайт. – URL: https://linuxmint.com/ (дата обращения: 11.04.2024).

19. Code::Blocks : сайт. – URL: https://www.codeblocks.org/ (дата обращения: 11.04.2024).

20. MySQL : сайт. – URL: https://www.mysql.com/ (дата обращения: 20.04.2024).

21. СТУ 7.5-07-2021. Стандарт университета. Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности // Сибирский федеральный университет : официальный сайт. – URL: https://about.sfu-kras.ru/node/8127 (дата обращения: 15.06.2024).

ПРИЛОЖЕНИЕ А

Листинги разработанных программ

Script.h

```
#ifndef SCRIPT_H
#define SCRIPT_H

#include <string>
#include <list>
#include "Sensor.h"
#include "Terminal.h"

using namespace std;

class Script
{
public:
	Script();
	Script(string n, int i, list<string> condLeft, list<string> condRight, list<string> aL, list<string> term);

	string getName();
	int getId();
	list<string> getCondLeft();
	list<string> getCondRight();
	string getCompSigns();
	list<string> getFunctionList();
	list<string> getTerminals();
	list<string> getActionList();

	void setName(string n);
	void setId(int i);
	void setCondLeft(list<string> cond);
	void setCondRight(list<string> cond);
	void setCompSigns(string cS);
	void setFunctionList(list<string> fL);
	void setTerminals(list<string> term);
	void setActionList(list<string> aL);

private:
	string name;
	int id;
	list<string> conditionsLeft;
	list<string> conditionsRight;
	string compSigns;
	list<string> functionList;
	list<string> terminals;
	list<string> actionList;
};
```

```
#endif // SCRIPT_H
```

Script.cpp

```
#include "Script.h"

Script::Script()
{

}

Script::Script(string n, int i, list<string> condLeft, list<string> condRight, list<string> aL, list<string> term)
{
	name = n;
	id = i;
	conditionsLeft = condLeft;
	conditionsRight = condRight;
	actionList = aL;
	terminals = term;
}

string Script::getName()
{
	return name;
}

int Script::getId()
{
	return id;
}

list<string> Script::getCondLeft()
{
	return conditionsLeft;
}

list<string> Script::getCondRight()
{
	return conditionsRight;
}

string Script::getCompSigns()
{
	return compSigns;
}

list<string> Script::getFunctionList()
{
	return functionList;
```

```
}

list<string> Script::getTerminals()
{
	return terminals;
}

list<string> Script::getActionList()
{
	return actionList;
}

void Script::setName(string n)
{
	name = n;
}
void Script::setId(int i)
{
	id = i;
}

void Script::setCondLeft(list<string> cond)
{
	conditionsLeft = cond;
}

void Script::setCondRight(list<string> cond)
{
	conditionsRight = cond;
}

void Script::setCompSigns(string cS)
{
	compSigns = cS;
}

void Script::setFunctionList(list<string> fL)
{
	functionList = fL;
}

void Script::setActionList(list<string> sL)
{
	actionList = sL;
}

void Script::setTerminals(list<string> term)
{
	terminals = term;
}
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

Sensor.h

```
#ifndef SENSOR_H
#define SENSOR_H

#include <string>

using namespace std;

class Sensor {
public:
	Sensor();
	Sensor(string n, int i, string dt, int ld);

	string getName();
	int getId();
	string getDataType();
	float getLastData();

	void setName(string n);
	void setId(int i);
	void setDataType(string dt);
	void setLastData(float ld);

   float getLastDataFromDB();
	void setLastDataToDB(float lD);

private:
	string name;
	int id;
	string dataType;
	float lastData;

};

#endif // SENSOR_H
```

Sensor.cpp

```
#include "Sensor.h"
#include <string>
#include <iostream>
#include <mysql/mysql.h>

Sensor::Sensor()
{

}

Sensor::Sensor(string n, int i, string dt, int ld)
{
```

```
        name = n;
        id = i;
        dataType = dt;
        lastData = ld;
}

string Sensor::getName()
{
        return name;
}

int Sensor::getId()
{
        return id;
}

string Sensor::getDataType()
{
        return dataType;
}

float Sensor::getLastData()
{
        return lastData;
}

void Sensor::setName(string n)
{
        name = n;
}
void Sensor::setId(int i)
{
        id = i;
}
void Sensor::setDataType(string dt)
{
        dataType = dt;
}
void Sensor::setLastData(float ld)
{
        lastData = ld;
}

float Sensor::getLastDataFromDB()
{
   MYSQL *conn;
   MYSQL_RES *res;
   MYSQL_ROW row;

   conn = mysql_init(NULL);
```

```
        if(conn == NULL)
        {
            fprintf(stderr, "Error: can't create MySQL-descriptor\n");
        }

        if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
        {
            fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
        }

        mysql_set_character_set(conn, "utf8");

        char query_[200];
        sprintf(query_, "SELECT Last_data FROM Sensor WHERE Name=\'%s\';", this-
>getName().c_str());

        mysql_query(conn, query_);

        if (res = mysql_store_result(conn))
        {
            while(row = mysql_fetch_row(res))
            {
                for (int i=0 ; i < mysql_num_fields(res); i++)
                {
                    this->setLastData(atof(row[i]));
                }
            }
         }
         else
            fprintf(stderr, "%s\n", mysql_error(conn));

        // Закрываем соединение с сервером базы данных
        mysql_close(conn);

        return lastData;
    }

    void Sensor::setLastDataToDB(float lD)
    {
        lastData = lD;

        MYSQL *conn;
        MYSQL_RES *res;
        MYSQL_ROW row;

        conn = mysql_init(NULL);
        if(conn == NULL)
        {
            fprintf(stderr, "Error: can't create MySQL-descriptor\n");
```

```
        }

        if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
        {
           fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
        }

        mysql_set_character_set(conn, "utf8");

        char query_[200];
        sprintf(query_, "UPDATE Sensor SET Last_data = %f WHERE Name=\'%s\';", lD, this-
>getName().c_str());
        cout << query_ << endl;
        mysql_query(conn, query_);

        if (res = mysql_store_result(conn))
        {
           /*while(row = mysql_fetch_row(res))
           {
              for (int i=0 ; i < mysql_num_fields(res); i++)
              {
                 cout << row[i] <<"\n"; //Выводим все что есть в базе через цикл
              }
           }*/
         }
         else
           fprintf(stderr, "%s\n", mysql_error(conn));

        // Закрываем соединение с сервером базы данных
        mysql_close(conn);
      }
```

Terminal.h

```
#ifndef TERMINAL_H
#define TERMINAL_H

#include <string>

using namespace std;

class Terminal {
public:
        Terminal();
        Terminal(string n, int i);

        string getName();
        int getId();
        string getProtocol();
```

```
        void setName(string n);
        void setId(int i);
        void setProtocol(string p);

private:
        string name;
        int id;
        string protocol;
};

#endif // TERMINAL_H
```

Terminal.cpp

```
#include "Terminal.h"
#include <string>

Terminal::Terminal()
{

}

Terminal::Terminal(string n, int i)
{
        name = n;
        id = i;
}

string Terminal::getName()
{
        return name;
}

int Terminal::getId()
{
        return id;
}

string Terminal::getProtocol()
{
        return protocol;
}

void Terminal::setName(string n)
{
        name = n;
}
void Terminal::setId(int i)
{
        id = i;
```

```
}

void Terminal::setProtocol(string p)
{
	protocol = p;
}
```

ScriptProcessingModule.h

```
#pragma once
#include "stdio.h"
#include <clocale>
#include <fstream>
#include <iostream>
#include <list>
#include "Script.h"
#include "Sensor.h"
#include "Terminal.h"

using namespace std;

list<Sensor> sensInit();
list<Terminal> termInit();
list<Script> textToScript(string name, list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals);
void outputSensData(list<Sensor> dadcheki, list<Terminal> terminals);
void outputScriptData(list<Script> stsenarii);
```

ScriptProcessingModule.cpp

```
// ScriptProcessingModule.cpp - модуль обработки сценариев
//

#include "pch.h"
#include "stdio.h"
#include <clocale>
#include <fstream>
#include <iostream>
#include <list>
#include <mysql/mysql.h>
#include "Script.h"
#include "Sensor.h"
#include "Terminal.h"
#include "ScriptProcessingModule.h"
#include "DataProcessingModule.h"

using namespace std;

//Функция перевода текста сценария в объект класса Script
list<Script> textToScript(list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals)
```

```
{
   list<string> scriptStr;

   MYSQL *conn;
   MYSQL_RES *res;
   MYSQL_ROW row;

   conn = mysql_init(NULL);
   if(conn == NULL)
   {
      fprintf(stderr, "Error: can't create MySQL-descriptor\n");
   }

   if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
   {
      fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
   }

   mysql_set_character_set(conn, "utf8");

   char query_[200];
   sprintf(query_, "SELECT COUNT(*) FROM Scripts;");

   mysql_query(conn, query_);

   if (res = mysql_store_result(conn))
   {
      while(row = mysql_fetch_row(res))
      {
         int idd;
         for (int i=0 ; i < mysql_num_fields(res); i++)
         {
            idd = stoi(string(row[i]));
         }
         if(scripts.size() < idd)
         {
            char query_[200];
            sprintf(query_, "SELECT Text FROM Scripts WHERE id > %d;", scripts.size());
            mysql_query(conn, query_);
            if (res = mysql_store_result(conn))
            {
               while(row = mysql_fetch_row(res))
               {
                  for (int i=0 ; i < mysql_num_fields(res); i++)
                  {
                     scriptStr.push_back(string(row[i]));
                     //cout << "mysql_num_fields(res): " << mysql_num_fields(res) << endl;
                  }
               }
```

```
            }
        }
    }
}
else
    fprintf(stderr, "%s\n", mysql_error(conn));

// Закрываем соединение с сервером базы данных
mysql_close(conn);

for (int i = 0; i < scriptStr.size(); i++)
{
    Script newScript;
    newScript.setId(scripts.size() + 1);
    list<string> conditions;
    string S = get(scriptStr, i);
    string S1, S2, S3;
    int pos1 = S.find('\n');
    int pos2 = S.find('\n', pos1 + 1);
        S3 = S.substr(0, pos1);
        S1 = S.substr(pos1 + 1, pos2 - pos1 - 1);
        S2 = S.substr(pos2 + 1);

    //Работа с блоком ЕСЛИ
    if (S1.find("ЕСЛИ") == 0)
    {
        string compSigns{ "<>=" };
        string comparations;
        //cout << "Блок ЕСЛИ " << endl;

        //Определяем "условия слева" и знаки сравнения
        int ind1 = 0;
        int ind2 = 0;
        while (S1.find_first_of(compSigns, S1.find('(', ind2)) != -1)
        {
            ind1 = S1.find('(', ind2);
            ind2 = S1.find_first_of(compSigns, ind1);
            comparations.push_back(S1[ind2]);
            string datName = S1.substr(ind1 + 1, ind2 - ind1 - 1);
            conditions.push_back(datName);
        }
        newScript.setCondLeft(conditions);
        newScript.setCompSigns(comparations);

        //Определяем "условия справа"
        ind1 = 0;
        ind2 = 0;
        conditions.clear();
        while (S.find(')', ind1 = S1.find_first_of(compSigns, ind2)) != -1)
        {
```

```
        ind1 = S1.find_first_of(compSigns, ind2);
        ind2 = S1.find(')', ind1);
        string datName = S1.substr(ind1 + 1, ind2 - ind1 - 1);
        conditions.push_back(datName);
    }
    newScript.setCondRight(conditions);

    //Определяем функции (И, ИЛИ)
    ind1 = 0;
    ind2 = 0;
    conditions.clear();
    while (S.find('(', S1.find(')', ind2)) != -1)
    {
        ind1 = S1.find(')', ind2);
        ind2 = S1.find('(', ind1);
        string funcName = S1.substr(ind1 + 1, ind2 - ind1 - 1);
        if(funcName == "ИЛИ" || funcName == "И")
        {
            conditions.push_back(funcName);
        }
    }
    newScript.setFunctionList(conditions);
}

//Работа с блоком ТО
if (S2.find("ТО") == 0)
{
    //cout << "Блок ТО" << endl;
    //Определяем терминалы
    int ind1 = 0;
    int ind2 = 0;
    conditions.clear();
    while (S2.find('(', S2.find('[', ind2)) != -1)
    {
        ind1 = S2.find('[', ind2);
        ind2 = S2.find('(', ind1);
        string termName = S2.substr(ind1 + 1, ind2 - ind1 - 1);
        conditions.push_back(termName);
    }
    newScript.setTerminals(conditions);

    //Определяем действия
    ind1 = 0;
    ind2 = 0;
    conditions.clear();
    while (S2.find(')', S2.find('(', ind2)) != -1)
    {
        ind1 = S2.find('(', ind2);
        ind2 = S2.find(')', ind1);
        string funcName = S2.substr(ind1 + 1, ind2 - ind1 - 1);
```

```
                    conditions.push_back(funcName);
                }
                newScript.setActionList(conditions);
            }

            //Название скрипта
            if(!S3.empty())
            {
                //cout << "Ни ЕСЛИ, ни ТО" << endl;
                newScript.setName(S3);
            }

            //cout << S << endl;
            scripts.push_back(newScript);
            cout << "Добавлен сценарий " << newScript.getName() << endl;
        }

        return scripts;
    }

    //Функция для вывода данных о датчиках и терминалах на экран
    void outputSensData(list<Sensor> dadcheki, list<Terminal> terms)
    {
        //cout << "Zdes' byli dannye o dadchekakh!" << endl;

        for (Sensor S : dadcheki)
        {
            cout << "Sensor " << S.getId() << ":" << endl << "Name: " << S.getName() << endl <<
"Data type: " << S.getDataType() << endl << "Last data: " << S.getLastData() << endl << endl;
        }

        for (Terminal T : terms)
        {
            cout << "Terminal " << T.getId() << ":" << endl << "Name: " << T.getName() << endl
<< endl;
        }

    }

    //Функция для вывода данных о сценариях на экран
    void outputScriptData(list<Script> stsenarii)
    {
        //cout << "Tut byli dannye o stsenariyakh!" << endl;

        for (Script S : stsenarii)
        {
            cout << "Script " << S.getId() << ":" << endl << "Name: " << S.getName() << endl;
            cout << "Блок ЕСЛИ: " << endl;
            cout << "Условия слева: " << endl;
            for (string Str : S.getCondLeft())
```

```
        {
            cout << Str << endl;
        }
        cout << endl;
        cout << "Условия справа: " << endl;
        for (string Str : S.getCondRight())
        {
            cout << Str << endl;
        }
        cout << endl;
        cout << "Знаки сравнения: " << endl;
        cout << S.getCompSigns() << endl;
        cout << "Функции: " << endl;
        for (string Str : S.getFunctionList())
        {
            cout << Str << endl;
        }
        cout << endl;


        cout << "Блок ТО: " << endl;
        cout << "Устройства: " << endl;
        for (string Str : S.getTerminals())
        {
            cout << Str << endl;
        }
        cout << endl;
        cout << "Действия: " << endl;
        for (string Str : S.getActionList())
        {
            cout << Str << endl;
        }
        cout << endl;
    }
}
```

DataProcessingModule.h

```
#pragma once

#include "pch.h"
#include "stdio.h"
#include <clocale>
#include <fstream>
#include <iostream>
#include <list>
#include "Script.h"
#include "Sensor.h"
#include "Terminal.h"
```

```
using namespace std;

void newData(list<Sensor> dadcheki);

string get(list<string> _list, int _i);
bool getBool(list<bool> _list, int _i);
Terminal getTerm(list<Terminal> _list, int _i);
Sensor getSens(list<Sensor> _list, int _i);
list<string> divList(list<string> _list, int ind1, int ind2);
list<bool> divBoolList(list<bool> _list, int ind1, int ind2);

bool functionChecking(list<bool> full_cond, list<string> functionList);

void TerminalOn(Terminal T);
void TerminalOff(Terminal T);
void SendTo(string param);
void ScriptsWorking(list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals);

string get(list<string> _list, int _i);
```

DataProcessingModule.cpp

```
// DataProcessingingModule.cpp - модуль обработки данных
//

#include "pch.h"
#include "stdio.h"
#include <clocale>
#include <fstream>
#include <iostream>
#include <list>
#include "Script.h"
#include "Sensor.h"
#include "Terminal.h"

#include <float.h>

using namespace std;

//Функция для получения элемента списка по индексу
string get(list<string> _list, int _i) {
	list<string>::iterator it = _list.begin();
	for (int i = 0; i < _i; i++) {
		++it;
	}
	return *it;
}

bool getBool(list<bool> _list, int _i) {
	list<bool>::iterator it = _list.begin();
```

```
		for (int i = 0; i < _i; i++) {
			++it;
		}
		return *it;
	}

	Terminal getTerm(list<Terminal> _list, int _i) {
		list<Terminal>::iterator it = _list.begin();
		for (int i = 0; i < _i; i++) {
			++it;
		}
		return *it;
	}

	Sensor getSens(list<Sensor> _list, int _i) {
		list<Sensor>::iterator it = _list.begin();
		for (int i = 0; i < _i; i++) {
			++it;
		}
		return *it;
	}

	//Функция для "упаковывания" части списка от одного индекса до другого в другой список
	list<string> divList(list<string> _list, int ind1, int ind2)
	{
		list<string> newList;

		for (int i = ind1; i < ind2 + 1; i++)
		{
			newList.push_back(get(_list, i));
		}
		return newList;
	}

	list<bool> divBoolList(list<bool> _list, int ind1, int ind2)
	{
		list<bool> newList;

		for (int i = ind1; i < ind2 + 1; i++)
		{
			newList.push_back(getBool(_list, i));
		}
		return newList;
	}

	//Функция для определения И и ИЛИ
	bool functionChecking(list<bool> full_cond, list<string> functionList)
	{
		bool result;
```

```
		int ind = -1;
		int count = 0; //кол-во ИЛИ

		//Если список функций пустой (в результате рекурсии), возвращаем одно значение
		if (functionList.empty())
		{
			return getBool(full_cond, 0);
		}

		//Идём от ПОСЛЕДНЕГО ИЛИ, считаем итоговый результат, результаты слева/справа считаем рекурсивно
		for (int i = 0; i < functionList.size(); i++)
		{
			//Ищем индекс последней ИЛИ и считаем их количество
			if (get(functionList, i) == "ИЛИ")
			{
				ind = i;
				count++;
			}
		}

		//Если ИЛИ нет, выполняем И
		if (ind == -1)
		{
			bool res = true;
			for (int i = 0; i < full_cond.size(); i++)
			{
				res = res && getBool(full_cond, i);
			}
			return res;
		}

		else
		{
			//Разделяем выражение на два - слева от ИЛИ и справа от ИЛИ, находим результаты рекурсивно
			list<bool> left_cond = divBoolList(full_cond, 0, ind);
			list<bool> right_cond = divBoolList(full_cond, ind + 1, full_cond.size() - 1);
			list<string> left_func, right_func;
			if(ind > 0)
				left_func = divList(functionList, 0, ind);
			if(ind < functionList.size() - 1)
				right_func = divList(functionList, ind + 1, functionList.size() - 1);

			//Считаем результаты для каждой половины, находим общий результат
			result = functionChecking(left_cond, left_func) || functionChecking(right_cond, right_func);
			return result;
		}
```

```
        }

        //Функции для выполнения сценария
        //Включить
        //Формируем команду для передачи данных в зависимости от протокола
        //Передаём команду в инкапсулятор
        void TerminalOn(Terminal T)
        {
                cout << "Включить терминал " << T.getName() << endl;
                if(T.getProtocol() == "Zigbee")
                {
            string command = "mosquiito_pub -t 'zigbee2mqtt/'" + T.getName() + "' -m '{ \"State\"
: \"On\" }'";
                //cout << command << endl;
                encap_Zigbee(command);
                }
                else if(T.getProtocol() == "IP")
                {
            //передаём данные через сокет
                }
        }

        //Выключить
        //Формируем команду для передачи данных в зависимости от протокола
        //Передаём команду в инкапсулятор
        void TerminalOff(Terminal T)
        {
                cout << "Выключить терминал " << T.getName() << "  " << T.getProtocol() << endl;
                if(T.getProtocol() == "Zigbee")
                {
            string command = "mosquiito_pub -t 'zigbee2mqtt/'" + T.getName() + "' -m '{ \"State\"
: \"Off\" }'";
                encap_Zigbee(command);
                }
                else if(T.getProtocol() == "IP")
                {
            //передаём данные через сокет
                }
        }

        //Переслать
        //Формируем команду для передачи данных в зависимости от протокола
        //Передаём команду в инкапсулятор
        void SendTo(string param, string protocol, string data)
        {
                cout << "Переслать " << data << " " << param << " по " << protocol << endl;
                if(protocol == "Zigbee")
                {
            string command = "mosquiito_pub -t 'zigbee2mqtt/" + param + "' -m '{ " + data + " }'";
            encap_Zigbee(command);
```

```
            }
            else if(protocol == "IP")
            {
        //передаём данные через сокет
            }
    }

    //Функция работы со сценариями
    //Получаем список всех доступных на данный момент сценариев, а также датчиков и терминалов
    //Обрабатываем каждый из них, и, в зависимости от условий и функций, выполняем проверку для каждого по очереди
    void ScriptsWorking(list<Script> scripts, list<Sensor> sensors, list<Terminal> terminals)
    {
        //Для каждого сценария
        for (Script S : scripts)
        {
            list<bool> full_cond; //Список для проверки выполнения условий

            //Проверяем блок ЕСЛИ
            //Берём n-ное условие слева, условие справа и знак. Сравниваем. Если да, то идём на следующее условие (скобку)

            for (int i = 0; i < S.getCompSigns().size(); i++)
            {
                //Определяем, что в условиях значение, а что - имя:
                //В каждом условии обязательно есть одно имя, иначе нет смысла
                //znach1 - слева, znach2 - справа
                float znach1 = FLT_MAX, znach2 = FLT_MAX;

                string condL = get(S.getCondLeft(), i);
                string condR = get(S.getCondRight(), i);

                for (Sensor sen : sensors)
                {
                    if (condL == sen.getName())
                    {
                        znach1 = sen.getLastDataFromDB();
                    }
                    else if (condR == sen.getName())
                    {
                        znach2 = sen.getLastDataFromDB();
                    }
                }

                //заполняем значения
                //Случай 1: условие справа было именем, а условие слева - число:
                if (znach1 == FLT_MAX && znach2 != FLT_MAX)
                {
                    znach1 = stof(condL);
```

```
		}
		//Случай 2: условие справа было числом, а условие слева - имя:
		if (znach1 != FLT_MAX && znach2 == FLT_MAX)
		{
			znach2 = stof(condR);
		}

		//Получили два значения в виде чисел
		//Проводим проверку условия:
		switch (S.getCompSigns()[i])
		{
			case '>':
			{
				if (znach1 > znach2)
				{
					full_cond.push_back(true);
				}
				else
					full_cond.push_back(false);
				break;
			}
			case '<':
			{
				if (znach1 < znach2)
				{
					full_cond.push_back(true);
				}
				else
					full_cond.push_back(false);
				break;
			}
			case '=':
			{
				if (znach1 == znach2)
				{
					full_cond.push_back(true);
				}
				else
					full_cond.push_back(false);
				break;
			}
		}

}

//Проверяем блок ТО
//Если условия выполняются, как надо
if (functionChecking(full_cond, S.getFunctionList()))
{
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
                cout << "Условия сценария " << S.getName() <<  " выполняются!"
<< endl;
                for (int i = 0; i < S.getActionList().size(); i++)
                {
                        if (get(S.getActionList(), i) == "ВКЛЮЧИТЬ")
                        {
                                for (int j = 0; j < terminals.size(); j++) {
                                        if      (getTerm(terminals,      j).getName()      ==
get(S.getTerminals(), i))
                                        {
                                                TerminalOn(getTerm(terminals, j));
                                        }
                                }
                        }
                        if (get(S.getActionList(), i) == "ВЫКЛЮЧИТЬ")
                        {
                                for (int j = 0; j < terminals.size(); j++) {
                                        if      (getTerm(terminals,      j).getName()      ==
get(S.getTerminals(), i))
                                        {
                                                TerminalOff(getTerm(terminals, j));
                                        }
                                }
                        }
                        if      (get(S.getActionList(),      i).find("ПЕРЕСЛАТЬ")      !=
std::string::npos)
                        {
                                for (int j = 0; j < terminals.size(); j++) {
                                        if                              (get(S.getTerminals(),
i).find(getTerm(terminals, j).getName()) != std::string::npos)
                                        {
                                                SendTo(get(S.getTerminals(),            i),
"Zigbee", get(S.getActionList(), i));
                                        }
                                }
                                for (int j = 0; j < sensors.size(); j++) {
                                        if (get(S.getTerminals(), i).find(getSens(sensors,
j).getName()) != std::string::npos)
                                        {
                                                SendTo(get(S.getTerminals(),            i),
"Zigbee", get(S.getActionList(), i));
                                        }
                                }
                        }
                }
        }
        else
        {
                cout << "Условия сценария " << S.getName() <<  " не выполнены"
<< endl;
```

```
                }
            }
        }
```

ControllerApplication.cpp

// ScriptProcessingModule.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.
//

```
#include "pch.h"
#include "stdio.h"
#include <clocale>
#include <fstream>
#include <iostream>
#include <list>
#include <mysql/mysql.h>
#include "Script.h"
#include "Sensor.h"
#include "Terminal.h"
#include "ScriptProcessingModule.h"
#include "DataProcessingModule.h"
#include "DataConvertionModule.h"

using namespace std;

list<Sensor> sensInit();
list<Terminal> termInit();

int main()
{
   setlocale(LC_ALL, "rus");

   string command, filename;
   list<Sensor> sensors;
   list<Terminal> terminals;
   list<Script> scripts;

   sensors = sensInit();
   terminals = termInit();

   while (1)
   {
      //Проверка на новые сценарии и их добавление
      scripts = textToScript(scripts, sensors, terminals);

      //Работа декапсулятора
      decapsulation(sensors, terminals);

      //Запуск функции работы сценариев
```

```
        ScriptsWorking(scripts, sensors, terminals);

        cout << "Выберите действие:"
           << endl << "1 - Прокрутить цикл,"
           << endl << "2 - Вывести информацию о сценариях,"
           << endl << "3 - Вывести информацию об устройствах,"
           << endl << "4 - Выход"
           << endl;
        getline(cin, command);
        switch (stoi(command))
        {
           case 1:
           {
              //scripts = textToScript(scripts, sensors, terminals);
              break;
           }
           case 2:
           {
              outputScriptData(scripts);
              break;
           }
           case 3:
           {
              outputSensData(sensors, terminals);
              break;
           }
           case 4:
           {
              return 0;
           }
           default:
           {
              //cout << "Команда не найдена!" << endl;
              cout << "cho-to proishodit" <<endl;
              break;
           }
        }
     }

     return 0;
}

//Функция для инициализации датчиков
list<Sensor> sensInit()
{
     MYSQL *conn;
     MYSQL_RES *res;
     MYSQL_ROW row;

     list<Sensor> sensList;
```

```
    conn = mysql_init(NULL);
    if(conn == NULL)
    {
       fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    }

    if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
    {
       fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
    }

    else
    {
       fprintf(stdout, "Success!\n");
    }

    mysql_set_character_set(conn, "utf8");

    char query_[200];
    sprintf(query_, "SELECT Name FROM Sensor;");

    mysql_query(conn, query_);

    if (res = mysql_store_result(conn))
    {
       while(row = mysql_fetch_row(res))
       {
          for (int i=0 ; i < mysql_num_fields(res); i++)
          {
             sensList.push_back(Sensor(string(row[i]), i + 1, string("float"), 0));
          }
       }
     }
     else
       fprintf(stderr, "%s\n", mysql_error(conn));

    // Закрываем соединение с сервером базы данных
    mysql_close(conn);
    cout << "Данные датчиков успешно добавлены!" << endl;
    return sensList;
}

//Функция для инициализации терминалов
list<Terminal> termInit()
{
    list<Terminal> termList;

    MYSQL *conn;
    MYSQL_RES *res;
```

```
    MYSQL_ROW row;

    conn = mysql_init(NULL);
    if(conn == NULL)
    {
       fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    }

    if(!mysql_real_connect(conn, "localhost", "root", "admpas", "contr_db", NULL, NULL,
0))
    {
       fprintf(stderr, "Error: can't connect to database %s\n", mysql_error(conn));
    }

    else
    {
       fprintf(stdout, "Success!\n");
    }

    mysql_set_character_set(conn, "utf8");

    char query_[200];
    sprintf(query_, "SELECT Name FROM Terminal;");

    mysql_query(conn, query_);

    if (res = mysql_store_result(conn))
    {
       while(row = mysql_fetch_row(res))
       {
          for (int i=0 ; i < mysql_num_fields(res); i++)
          {
             termList.push_back(Terminal(string(row[i]), i + 1, "Zigbee"));
          }
       }
     }
     else
       fprintf(stderr, "%s\n", mysql_error(conn));

    // Закрываем соединение с сервером базы данных
    mysql_close(conn);
    cout << "Данные исполнительных устройств успешно добавлены!" << endl;
    for(Terminal T : termList)
    {
       T.setProtocol("Zigbee");
    }
    return termList;
  }
```

ПРИЛОЖЕНИЕ Б

Тестовые сценарии

1. Script1

Script1

ЕСЛИ(DAT1>555)ИЛИ(D2>Sens3)И(Sens3<100,41)

ТО[term1(ВКЛЮЧИТЬ)]И[DAT1(ПЕРЕСЛАТЬ,1234567890)]

2. Script2

Script2

ЕСЛИ(DAT1<34)И(D2=Sens3)

ТО[term1(ВЫКЛЮЧИТЬ)]И[D2(ПЕРЕСЛАТЬ,СЕРВЕР)]

3. Script3

Script3

ЕСЛИ(Sens3<68,78)И(D2>145,6)

ТО[term1(ВКЛЮЧИТЬ)]И[TERM2(ВЫКЛЮЧИТЬ)]

4. TEST

TEST

ЕСЛИ(Датчик открытия двери=0)

ТО[term1(ВКЛЮЧИТЬ)]

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«19» 06 2024 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Разработка подсистемы синхронизации данных в сети передачи телеметрической информации

090401 Информатика и вычислительная техника

09.04.01.11 «Вычислительные системы и сети»

Руководитель	подпись	19.06.24 дата	профессор, канд. техн. наук должность, ученая степень	О.В. Непомнящий
Выпускник	подпись	19.06.24 дата		Д.С. Галкина
Рецензент	подпись	19.06.24 дата	доцент, канд. техн. наук должность, ученая степень	К.В. Раевич
Консультант	подпись	19.0624 дата	генеральный директор ООО «ПК «Дельта»» должность, ученая степень	О.Г. Варыгин
Нормоконтролёр	подпись	19.06.24 дата	профессор, канд. техн. наук должность, ученая степень	О.В. Непомнящий

Красноярск 2024