УТВЕРЖДАЮ

Заместитель управляющего директора по производству и техническому развитию

ООО «НПП «ИТЭЛМА»
Наименование юридического лица

_____ /Лукин Д.Б./
Подпись Фамилия, инициалы

«____» _____ 20____ г.
Дата

Описание протокола обмена сообщениями между ПО РТК «SCADA GSM» и ПО «Программатор»

Москва 2023 г.

Оглавление

1. Цель

Обеспечить бесперебойный процесс производства GSM модемов SIMCOM. Реализовать надежный протокол связи между ПО РТК «GSM SCADA» (в дальнейшем SCADA) и ПО «Программатор» (в дальнейшем Программатор), обеспечивающий минимальное время задержки между отправкой и приемом сообщения.

2. Описание используемых интерфейсов и протоколов

2.1. Описание используемых интерфейсов

Так как SCADA и Программатор будут установлены на отдельных ПК, то обмен сообщениями будет проходить по протоколу TCP/IP через IP адреса ПК, которые будут присваиваться для каждого ПК соответственно.

2.2. Описание протокола связи между SCADA и Программатор

По умолчанию и SCADA и Программатор должны реализовывать на своей стороне и серверный и клиентский протокол TCP/IP, так как возможна инициализация обмена сообщениями как со стороны SCADA, так и со стороны Программатор.

2.2.1. Реализация TCP/IP Server на стороне SCADA

Пример реализации серверного протокола TCP/IP на стороне SCADA приведен в листинге ниже:

```
Ссылок: 1
public ProgrammatorTCPServerService()
{

    IPAddress ipaddress = new IPAddress(new byte[] { 192, 168, 0, 170 }); //Задается локальный адрес ПК
    point = new IPEndPoint(ipaddress, 808); //Задается EndPoint с указанием виртуального порта на котором будет работать SCADA
    Socket tcpServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp); //Создается Socket обрабатывающий входящие TCP подключения
    tcpServer.Bind(point); //Происходит присвоение созданного сокета к виртуальному порту на ПК
    tcpServer.Listen(100); //Запускается процесс прослушивания входящих подключений
    tcpServer.AcceptAsync(); //При появлении нового подключения запускается асинхронный процесс обмена сообщениями с клиентом

}
```

Листинг 2.2.1

Необходимо как на стороне SCADA, так и на стороне Программатора обеспечить возможность настройки локального адреса и портов, используемых для клиентской и серверной части приложения. В примере адреса и порты приведены для ознакомления.

Далее происходит десериализация полученного сообщения в соответствии со структурой пакета JSON описанной в главе СТРУКТУРА ПАКЕТА JSON:

```
Ссылок: 1
public async Task<string> ReceiveMessage(Socket clientSocket)
{
    byte[] data2 = new byte[65536]; //Создается массив буфера для входящего сообщения
    int length = clientSocket.Receive(data2); //Запись полученного сообщения от клиента в буфер
    string msg = Encoding.ASCII.GetString(data2).Trim('\0'); //Обрезка пустых байт, если таковые имеются
    ProgrammatorMessageModel mess = JsonConvert.DeserializeObject<ProgrammatorMessageModel>(msg); //Десериализация полученного сообщения
                                                                                                   //в соответствии со структурой JSON пакета
    return msg;
}
```

Листинг 2.2.2

После обработки сервер обязательно отправляет ответ клиенту с результатом обработки:

```
Ссылок: 1
public async Task<int> SendMessage(Socket clientSocket, string receivedMessage)
{
    Dictionary<int, string> errors = new();
    for (int k = 1; k < 1000; k++)...

    ProgrammatorMessageModel mssg = new ProgrammatorMessageModel()...; //Созданием нового экземпляра класса JSON пакета

    string json = JsonConvert.SerializeObject(mssg); //Сериализация пакета в формат JSON
    byte[] data1 = Encoding.ASCII.GetBytes(json); //Генерация массива байт из сформированного пакета в формате JSON
    int sendLength = clientSocket.Send(data1); //Отправка ответа клиенту

    return sendLength;
}
```

Листинг 2.2.3

2.2.2. Реализация TCP/IP Client на стороне SCADA

Пример реализации клиентского протокола TCP/IP на стороне SCADA приведен в листинге ниже:

```
EndPoint point;
Socket tcpServer;
Ссылок: 0
public TCPClientService()
{
    IPAddress ipaddress = new IPAddress(new byte[] { 192, 168, 1, 71 }); //Создание локального адреса ПК
    point = new IPEndPoint(ipaddress, 808); //Создание EndPoint с указанием порта на котором работает TCP/Ip сервер приложения Программатор
    tcpServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp); //Созданием нового сокета для обмена сообщениями
    tcpServer.Connect(point); //Подключение к TCP/IP серверу приложения Программатор
}
```

Листинг 2.2.4

После успешного подключения происходит формирование и отправка сообщения серверу:

```
Ссылок: 1
public async Task<int> SendMessage(Socket clientSocket, string receivedMessage)
{
    Dictionary<int, string> errors = new();
    for (int k = 1; k < 1000; k++)...

    ProgrammatorMessageModel mssg = new ProgrammatorMessageModel()...; //Созданием нового экземпляра класса JSON пакета

    string json = JsonConvert.SerializeObject(mssg); //Сериализация пакета в формат JSON
    byte[] data1 = Encoding.ASCII.GetBytes(json); //Генерация массива байт из сформированного пакета в формате JSON
    int sendLength = clientSocket.Send(data1); //Отправка сообщения серверу

    return sendLength;
}
```

Листинг 2.2.5

И получение сообщения от сервера с результатом запроса:

```
Ссылок: 1
public async Task<string> ReceiveMessage(Socket clientSocket)
{
    byte[] data2 = new byte[65536]; //Создается массив буфера для входящего сообщения
    int length = clientSocket.Receive(data2); //Запись полученного сообщения от сервера в буфер
    string msg = Encoding.ASCII.GetString(data2).Trim('\0'); //Обрезка пустых байт, если таковые имеются
    ProgrammatorMessageModel mess = JsonConvert.DeserializeObject<ProgrammatorMessageModel>(msg); //Десериализация полученного сообщения
                                                                                                   //в соответствии со структурой JSON пакета
    return msg;
}
```

Листинг 2.2.6

2.2.3. Контроль целостности пакетов

Протокол TCP/IP, является «надежным» протоколом связи и не требует дополнительного контроля целостности пакетов, так как:

1) Обеспечивает надежную доставку данных, так как предусматривает установление логического соединения
2) Нумерует пакеты и подтверждает их прием, а в случае потери организует повторную передачу
3) Делит передаваемый поток байтов на части — сегменты - и передает их нижнему уровню, на приемной стороне снова собирает их в непрерывный поток байтов

3. Описание структуры сообщений между SCADA и Программатор

3.1. Формат пакета сообщения

Ниже приведена модель сообщения для реализации обмена сообщениями между SCADA и Программатор:

```
Ссылок: 4
public class SimModel
{
    //IMEI модема
    Ссылок: 0
    public string IMEI { get; set; } = string.Empty;
    //Номер КУ
    Ссылок: 1
    public int KU { get; set; } = 0;
    //Версия софта
    Ссылок: 1
    public string SWVersion { get; set; } = string.Empty;
    //Время начала программирования, задается скада
    Ссылок: 1
    public DateTime StartProgramming { get; set; } = DateTime.Now;
    //Время окончания программирования, задается скада
    Ссылок: 1
    public DateTime EndProgramming { get; set; } = DateTime.Now;
    //Результат программирования
    Ссылок: 1
    public bool Result { get; set; } = false;
    //Ошибки при программировании
    Ссылок: 0
    public string Error { get; set; } = string.Empty;
    //Команда, подаваемая инициализатором обмена сообщениями
    Ссылок: 1
    public string Command { get; set; } = string.Empty;
    //Ответ на комадну поданную инициализатором обмена сообщениями
    Ссылок: 1
    public string Message { get; set; } = string.Empty;
    //Информация, считанная с баркода модема
    Ссылок: 1
    public string Info { get; set; } = string.Empty;
}
```

Листинг 3.1.1

3.2. Типы данных полей модели сообщения

3.2.1. Поле «IMEI»

В данном поле содержится строковое представления баркода изделия в формате «string» = «@5ITEL»

3.2.2. Поле «KU»

В данном поле содержится порядковым номер КУ для программирования в котором происходит программирование изделия в формате «int» = «1..10»

3.2.3. Поле «SWVersion»

В данном поле содержится версия программного обеспечения которым запрограммировано изделие в формате «string» = «1.2.3»(**формат версии ПО требует согласования с заказчиком**).

3.2.4. Поле «StartProgramming»

В данном поле содержится время начала программирования в формате «DateTime» (**формат представления даты согласуется с заказчиком**). Пример значения данного поля в формате «DateTime» = «2023-01-23T11:49:34.2037224+03:00» Также можно изменить тип данного поля на «string»(**Согласуется с заказчиком**).

3.2.5. Поле «EndProgramming»

В данном поле содержится время окончания программирования в формате «DateTime» (**формат представления даты согласуется с заказчиком**). Пример значения данного поля в формате «DateTime» = «2023-01-23T11:49:34.2037224+03:00» Также можно изменить тип данного поля на «string»(**Согласуется с заказчиком**).

3.2.6. Поле «Result»

В данном поле содержится булево значение с результатом программирования в формате «bool» = true/false.

3.2.7. Поле «Error»

В данном поле содержится наименования ошибки в формате «string» = «Timeout».

3.2.8. Поле «Command»

Данная команда всегда инициализирует обмен сообщениями между клиентом и сервером со стороны клиента.

В данном поле содержится команда управления (как для Scada, так и для Программатора) в формате «string» и может принимать следующие значения:

1) «Start» - команда на запуск программирования (Scada -> Программатор)
2) «Stop» - команда на остановку программирования по причине превышения времени программирования (Scada -> Программатор)
3) «FinishedOK» - команда положительного результата процесса окончания программирования (Программатор -> Scada)
4) «FinishedNOK» - команда отрицательного результата процесса окончания программирования (Программатор -> Scada)

3.2.9. Поле «Message»

Данное сообщение является результатом обработки полученной команды от клиента на стороне сервера.

В данном поле содержится результат выполнения команды (см. выше) в формате «string» и может принимать следующие значения:

1) «Started» - результат выполнения команды «Start» (Программатор -> Scada)
2) «Stopped» - результат выполнения команды «Stop» (Программатор -> Scada)
3) «Finished» - результат выполнения команды «FinishedOK/FinishedNOK» (Scada -> Программатор)

3.2.10. Поле «Info»

В данном поле содержится дополнительная информация о программирования в формате «string» = «Информация». В данный момент в поле содержится информация, считанная с баркода модема. (**Согласуется с заказчиком**)

4. Описание алгоритма обмена сообщениями

4.1. Алгоритм автоматизации процесса программирования GSM модема SIMCOM

4.1.1. Блочная диаграмма, описывающая процесс взаимодействия между SCADA и Программатор

Ниже приведен алгоритм взаимодействия между SCADA и Программатор:

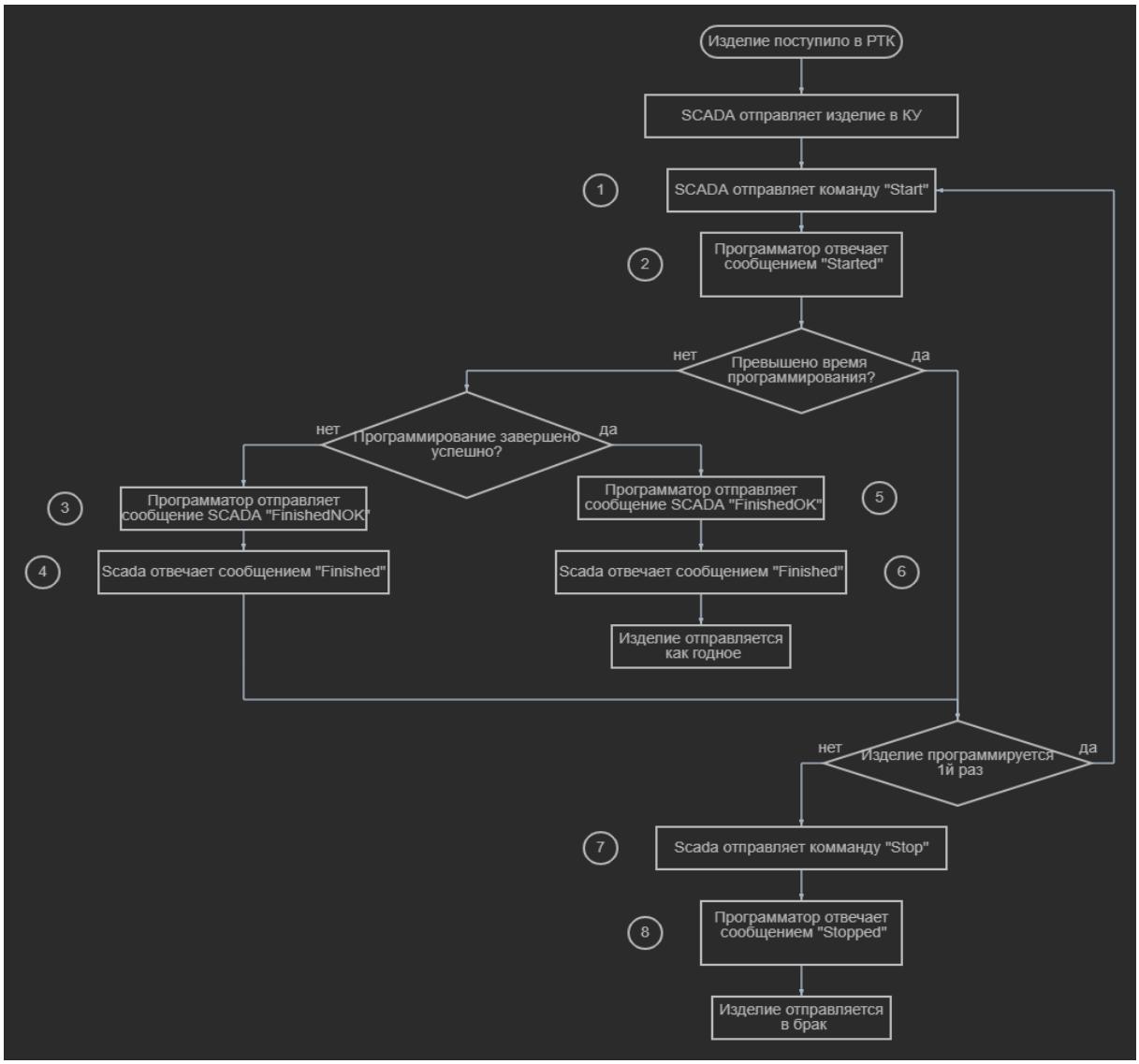

Рисунок 4.1.1

4.1.2. Описание основных моментов алгоритма взаимодействия между SCADA и Программатор.

Пояснения к ссылкам на рисунке выше:

1) Scada отправляет комманду на TCP Server Программатора со следующим содержимым:
    a. IMEI = «860147052220215»
    b. KU = 1
    c. SWVersion = «0»
    d. StartProgramming = «Содержит реальное время начала программирования»
    e. EndProgramming = DateTime.Now
    f. Result = false
    g. Error = «»
    h. Command = «Start»
    i. Message = «»
    j. Info = «Данные с BarCode модема»

2) Программатор отвечает Scada следующим образом (эхо с изменением поля Command и Message):
    a. IMEI = «860147052220215»
    b. KU = 1
    c. SWVersion = «0»
    d. StartProgramming = «Содержит реальное время начала программирования»
    e. EndProgramming = DateTime.Now
    f. Result = false
    g. Error = «»
    h. Command = «»
    i. Message = «Started»
    j. Info = «Данные с BarCode модема»
3) Программатор отправляет на TCP Server Scada команду со следующим содержимым:
    a. IMEI = «860147052220215»
    b. KU = 1
    c. SWVersion = «0»
    d. StartProgramming = «Содержит реальное время начала программирования»
    e. EndProgramming = DateTime.Now
    f. Result = false
    g. Error = «WrongVersion»
    h. Command = «FinishedNOK»
    i. Message = «»
    j. Info = «Данные с BarCode модема»
4) Scada отвечает следующим образом (эхо с изменением поля Command и Message):
    a. IMEI = «860147052220215»
    b. KU = 1
    c. SWVersion = «0»
    d. StartProgramming = «Содержит реальное время начала программирования»
    e. EndProgramming = «Содержит реальное время окончания программирования»
    f. Result = false
    g. Error = «WrongVersion»
    h. Command = «»
    i. Message = «Finished»
    j. Info = «Данные с BarCode модема»
5) Программатор отправляет на TCP Server Scada команду со следующим содержимым:
    a. IMEI = «860147052220215»
    b. KU = 1
    c. SWVersion = «0»
    d. StartProgramming = «Содержит реальное время начала программирования»
    e. EndProgramming = DateTime.Now
    f. Result = true
    g. Error = «»
    h. Command = «FinishedOK»
    i. Message = «»
    j. Info = «Данные с BarCode модема»
6) Scada отвечает следующим образом (эхо с изменением поля Command и Message):

a. IMEI = «860147052220215»
b. KU = 1
c. SWVersion = «0»
d. StartProgramming = «Содержит реальное время начала программирования»
e. EndProgramming = «Содержит реальное время окончания программирования»
f. Result = true
g. Error = «»
h. Command = «»
i. Message = «Finished»
j. Info = «Данные с BarCode модема»

7) Scada отправляет комманду на TCP Server Программатора со следующим содержимым:
   a. IMEI = «860147052220215»
   b. KU = 1
   c. SWVersion = «0»
   d. StartProgramming = «Реальное время начала программирования»
   e. EndProgramming = DateTime.Now
   f. Result = false
   g. Error = «Timeout»
   h. Command = «Stop»
   i. Message = «»
   j. Info = «Данные с BarCode модема»
8) Программатор отвечает Scada следующим образом (эхо с изменением поля Command и Message):
   a. IMEI = «860147052220215»
   b. KU = 1
   c. SWVersion = «0»
   d. StartProgramming = «Реальное время начала программирования»
   e. EndProgramming = «Реальное время окончания программирования»
   f. Result = false
   g. Error = «Timeout»
   h. Command = «»
   i. Message = «Stopped»
   j. Info = «Данные с BarCode модема»