Общество с ограниченной ответственностью

"ИНСАЙТ-менеджмент"

УТВЕРЖДАЮ:

Генеральный директор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.К. Грибанова

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

О Т Ч Е Т

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ

И ОПЫТНО-КОНСТРУКТОРСКОЙ РАБОТЕ (НИОКР)

Проектирование экспериментальной локальной информационной сети мониторинга, анализа данных и управления на базе контроллеров IoT. Разработка регламента тестирования и алгоритмов проверки работоспособности информационной системы.

(промежуточный)

Этап № 5

|  |  |  |
| --- | --- | --- |
| Ген. директор  ООО «ИНСАЙТ-Электроникс» |  | С.В. Грибанов |
| Руководитель НИОКР, к.т.н. |  | А.В. Кычкин |
| Исполнитель |  | Р.Р. Гайнанов |

Пермь 2017 СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель работы,

к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Кычкин

подпись, дата

Исполнители темы:

Гайнанов Р.Р.

СОДЕРЖАНИЕ

[Введение 4](#_Toc504323191)

[1. Проектирование модуля для подключения устройств по OPC 5](#_Toc504323192)

1. [1.1. Описание протокола 5](#_Toc504323193)
2. [1.2. Описание транспортного уровня и протоколов обмена данными 8](#_Toc504323194)
3. [1.3. Описание многоуровневой архитектуры модуля OPC UA 9](#_Toc504323195)
4. [1.4. Информационная модель объектов OPC UA 10](#_Toc504323196)
5. [1.5. Концепция и реализация системы на базе OPC UA 12](#_Toc504323197)

[2. Проектирование модуля для подключения устройств по Modbus 14](#_Toc504323198)

1. [2.1. Описание протокола 14](#_Toc504323199)
2. [2.2. Виды команд 15](#_Toc504323200)
3. [2.3. Программы для работы с протоколом 15](#_Toc504323201)
4. [2.4. Пример обмена информацией по протоколу 17](#_Toc504323202)
5. [2.5. Особенности работы протокола по сети Ethernet 19](#_Toc504323203)

[3. Проектирование модуля для подключения устройств по MQTT 21](#_Toc504323204)

1. [3.1. Описание протокола 21](#_Toc504323205)
2. [3.2. Качество обслуживания 24](#_Toc504323206)
3. [3.3. Защита передачи данных 25](#_Toc504323207)
4. [3.4. Описание процедуры первого подключения устройства к системе 25](#_Toc504323208)
5. [3.5. Описание методов передачи параметров устройству 26](#_Toc504323209)
6. [3.6. Сбор, хранение и отправка значений измерений 28](#_Toc504323210)

[4. Проектирование пользовательских интерфейсов 30](#_Toc504323211)

[Заключение 32](#_Toc504323212)

[Список литературы 33](#_Toc504323213)

[Приложение А Реализация базовых функций клиента и сервера OPC UA 34](#_Toc504323214)

[Приложение Б Реализация базовых функций клиента и сервера Modbus TCP 35](#_Toc504323215)

[Приложение В Реализация базовых функций клиента и сервера MQTT 36](#_Toc504323216)

[Приложение Г Проектирование пользовательских интерфейсов 37](#_Toc504323217)

# Введение

Важную роль в приложениях промышленного интернета вещей (IIoT) играет интеграция данных, обеспечить которую возможно бесшовной связью между оборудованием по месту и системами управления верхнего уровня (SCADA, ERP и т. д.). Однако на предприятии, где каждый промышленный контроллер (ПЛК) использует уникальный драйвер связи, осуществить интеграцию данных довольно сложно.

Среди представленных на рынке технологий для обеспечения связи лишь некоторые из них пользуются особой популярностью о производителей оборудования. Нами были выделено несколько протоколов обмена информацией именно о них пойдет речь в данной работе.

# Проектирование модуля для подключения устройств по OPC

## Описание протокола

Для упрощения и удешевления реализации информационных обменов в системах промышленной автоматизации в 1996 году была предложена технология ОРС (OLE for Process Control), в которой единообразие в настройках стыков SCADA-системы с «внешним миром» достигается за счёт использования определённого шлюза, унифицирующего интерфейс взаимодействия с клиентом и скрывающего частный протокол отдельных средств автоматизации. Использование «классической» ОРС ограничено платформой Windows, так как это не протокол передачи данных, а именно программная технология, основанная на механизме удалённого вызова процедур с использованием стека DCOM. Это накладывает свой негативный отпечаток на такие параметры процесса взаимодействия по ОРС, как безопасность, надёжность и резервирование.

Говоря о «классической» ОРС, мы в первую очередь имеем в виду передачу данных согласно спецификациям ОРС DA (Data Access — в масштабе реального времени), ОРС HDA (Historical Data Access — архивов изменений параметров) и ОРС А&Е (Alarm and Events — тревог и событий). Популярность последних двух спецификаций существенно меньше, чем у ОРС DA.

Типовая схема использования ОРС для доступа к данным ПЛК и SCADA-систем показана на рис. 1 [1].

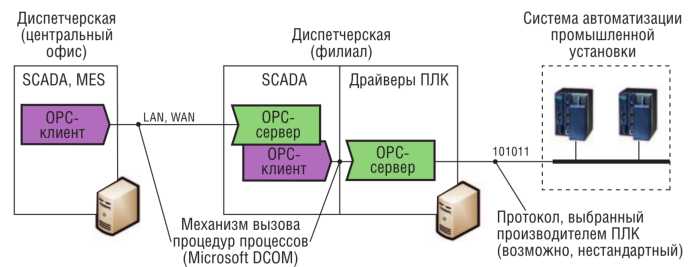


Рисунок 1 – Схема применения «классической» ОРС

Однако сегодня можно смело утверждать, что «классическая» OPC (стандарты OPC DA и OPC HDA) доживает свой век: унификация различных протоколов была с блеском решена на технологиях, которые в настоящий момент уже устарели. Дело в том, что OPC DA и OPC HDA базируются на технологии Microsoft DCOM. Тем самым обеспечивается «жетская» привязка к ОС, на которой возможна реализация компонентов OPC DA и HDA. Кроме того, с приходом интернета добавились новые вызовы: на системы управления ТП начались хакерские атаки, что привело к необходимости шифрования и аутентификации. К новой реальности «классическая» OPC оказалась не готова [2].

Для актуализации технологии OPC организация OPC Foundation инициировала создание нового стандарта, по сути, с нуля. Теперь за основу брались открытые кроссплатформенные технологии без привязки к DCOM. Новая редакция получила название OPC UA — Unified Architecture («унифицированная архитектура»). OPC UA сохранила все достижения «классической» OPC, но при этом лишена ее недостатков.

OPC UA обладает следующими преимуществами [3, 4].

* *Полностью кроссплатформенный стандарт.*

С появлением кроссплатформенности исчезла необходимость в существовании OPC-сервера как отдельного приложения на компьютере, поскольку теперь большинство контроллеров уже имеют встроенную ОС и производитель может установить OPC-сервер непосредственно в контроллер. Для получения тегов из контроллера больше не требуется настраивать отдельное приложение, достаточно задать в SCADA-системе параметры подключения к контроллеру, и весь список переменных (в том числе архивных) будет получен и добавлен в проект. Таким образом, построение проекта существенно ускоряется.

* *Легкость удаленного подключения.*

В классическом OPC настройка удаленного доступа была крайне затруднительна, а в некоторых случаях и невозможна. В OPC UA такой проблемы нет. Все, что нужно, — это открыть разрешение в межсетевом экране на нужный TCP-порт. Если удаленный компьютер находится во внутренней сети, недоступной SCADA, то и это не становится проблемой. Задача решается переадресацией порта. А при работе через Internet обмен можно вести как через VPN, так и через публичный IP адрес.

* *Шифрование и аутентификация.*

К сожалению, к новым вызовам, требующим повышенной безопасности передачи данных, пока не готов ни один промышленный протокол: все они разработаны в конце 1990-х годов (или даже ранее) и не имеют никакой защиты. Технология OPC UA в этом вопросе является исключением: в ней применяются несколько вариантов шифрования и аутентификации. Это позволяет вести передачу данных через Internet, не беспокоясь за их сохранность.

* *Унификация данных.*

В «классической» OPC существуют несколько стандартов для каждого варианта использования: OPC DA — для текущих данных, OPC HDA — для архивных и т. д. В OPC UA все стандарты объединены: текущие данные, архивные данные, сообщения — все это передается через один сервер, по единому интерфейсу.

Использование «классической» ОРС возможно и в ОРС UA-среде: специальная оболочка (wrapper) обеспечивает доступ к обычному ОРС DA-серверу, а рroху-модуль позволяет ОРС DA-клиенту взаимодействовать с новыми ОРС UA-серверами. Таким образом создается туннель благодаря OPC UA протоколам (рис. 2).

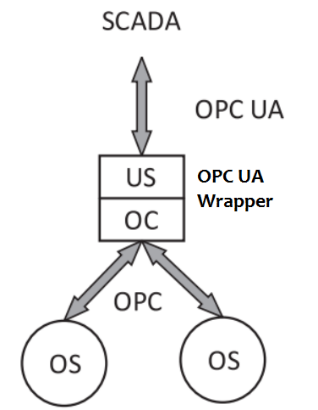


Рисунок 2 – Использование ОРС UA для связи с удаленными OPC DA  
(**US** – OPC UA Server, **OC** – OPC DA Client, **OS** – OPC DA Server)

Использование стандарта OPC UA позволит осуществлять сбор измерительной информации, передачу управляющих команд и изменения состояний выходов со специализированных устройств выступающих в качестве сервера OPC UA. Для выполнения этих действий необходимо реализовать в системе функции OPC UA клиента.

## Описание транспортного уровня и протоколов обмена данными

Для обмена информацией спецификация OPC UA предоставляет три протокола (рис. 3) [5].

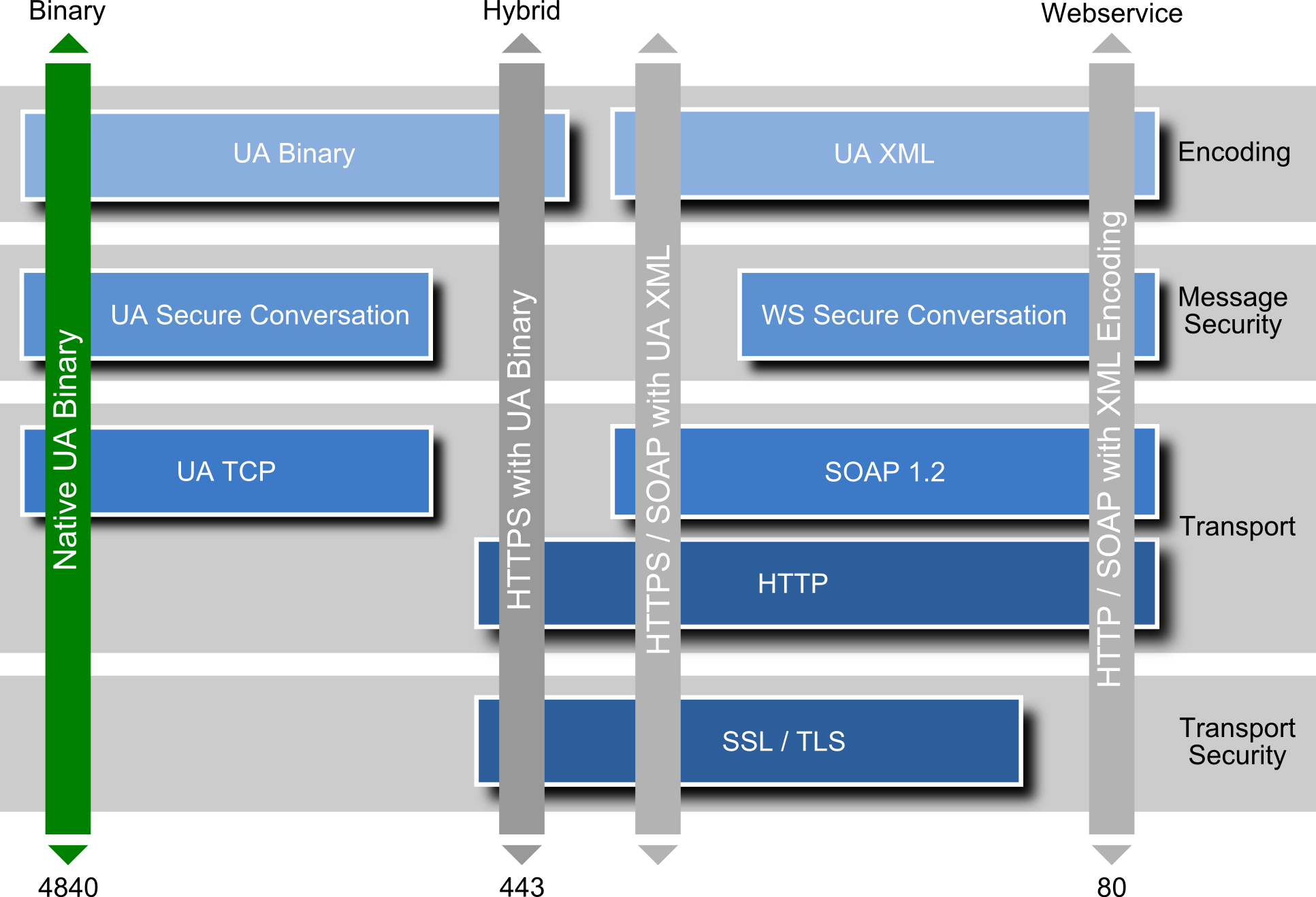


Рисунок 3 – Разновидности протоколов OPC UA

1. Двоичный протокол (UA binary):

* лучшая производительность, минимальные накладные расходы;
* потребляет минимум ресурсов (не требуются обработка XML, SOAP и HTTP);
* наилучшая возможная совместимость (двоичный код определён явно и допускает меньшую степень свободы в процессе исполнения в отличие от XML);
* всего один порт TCP (4840) используется для коммуникации и легко может быть туннелирован или пропущен через межсетевой экран.

2. Веб-службы (XML-SOAP):

* лучшая поддержка из доступных инструментов. Легко может быть использован, например, из окружения Java или .Net;
* применимый с межсетевыми экранами. Порт 80 (http) и 443 (https) обычно будут использоваться без дополнительных настроек.

3. Hybrid (UA-Binary via HTTPS)

* менее перегружен, чем XML-SOAP;
* содержит в себе комбинацию преимуществ обоих типов: двоичные данные предаются через HTTPS соединению;
* не требует дополнительных настроек межсетевого экрана, обычно соединения через 443 порт работают без ручного конфигурирования;

Прикладной программист может распознавать протокол через различие в URL: для двоичного протокола *opc.tcp://server* и *http://server* для веб-служб. Иначе говоря, OPC UA работает полностью прозрачно для API.

## Описание многоуровневой архитектуры модуля OPC UA

Основным отличием ОРС UA от OPC является отказ от технологии СОМ и DCOM фирмы Microsoft и переход к архитектуре SOA (Service Oriented Architecture - "Архитектура, ориентированная на сервисы") с целью обмена информацией и обеспечения совместимости c множеством различных аппаратно-программных платформ. Под сервисом в ОРС UA понимается некоторая функциональность, заключенная в программном компоненте, который может быть транспортирован от сервера к клиенту или обратно и вызван удаленно. Вызов сервиса аналогичен вызову метода в языках объектно-ориентированного программирования. Интерфейс между клиентом OPC UA и сервером определяется как набор сервисов. Основным принципом SOA является независимость от программной технологии, от вычислительной платформы, от языков программирования, от конкретных приложений, а также организация сервисов как слабосвязанных компонентов для построения систем. Сервисы включают в себя средства для обеспечения информационной безопасности (рис. 4).

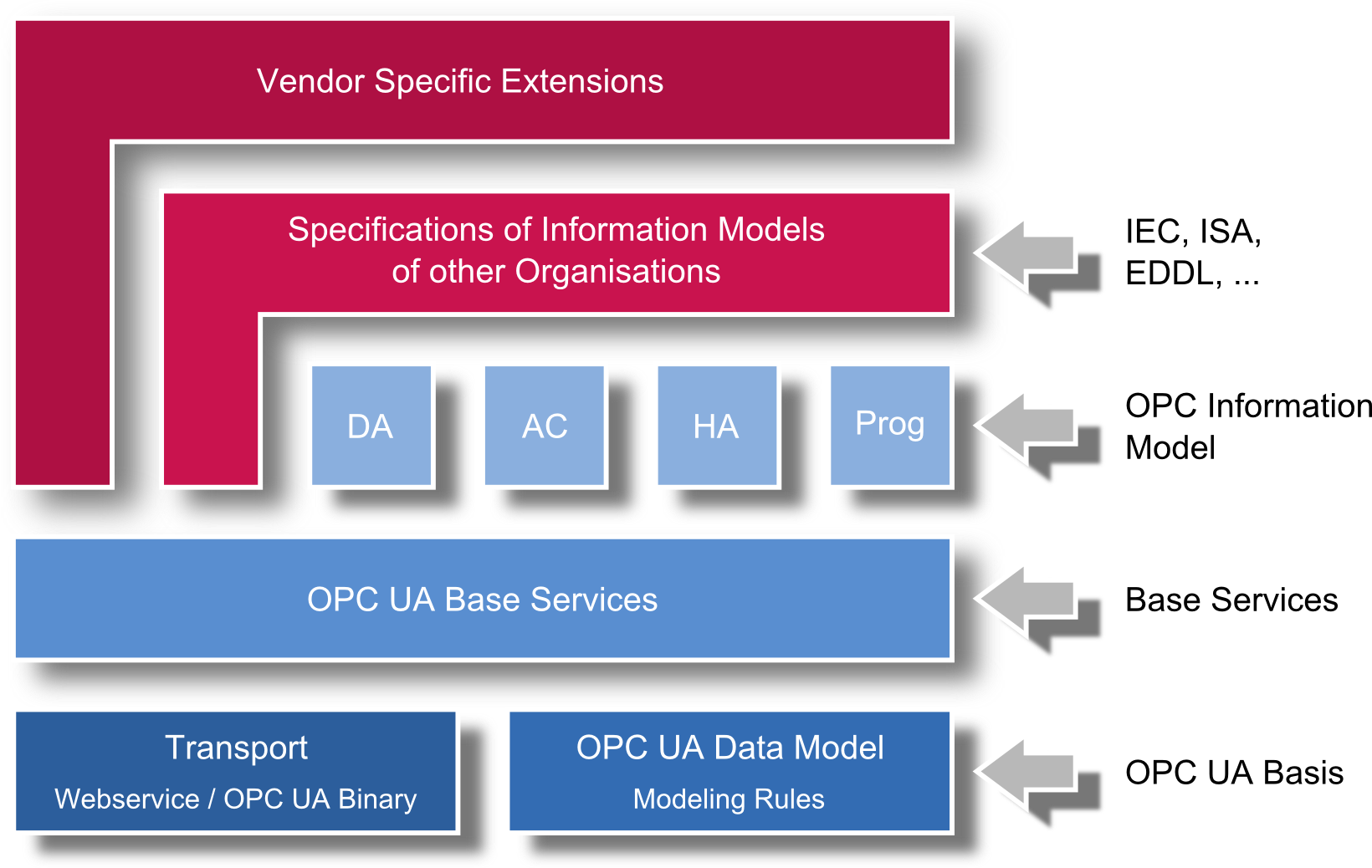


Рисунок 4 – Особенности архитектуры OPC-систем

Информационная модель OPC – это не только иерархия, основанная на группах, элементах и свойствах, но и так называемая сеть Full Mesh, основанная на узлах. Эта сеть узлов может дополнительно передавать все разновидности метаинформации и диагностических данных. Ближайшим изображением узла будет объект, известный из объектно-ориентированного программирования (ООП). Он может иметь атрибуты для доступа к чтению (Data Access (DA), Historical Data Acess (HDA)), методы, которые могут быть вызваны (Commands), и инициированные события, которые могут быть запущены (AE, DA DataChange) для обмена определенной информацией между устройствами. Событие содержит, среди прочего, время уведомления, сообщение и качество. Узлы используются для данных процесса, а также для всех других типов метаданных.

## Информационная модель объектов OPC UA

OPC UA определяет общую объектную модель, включая связанную систему типов. В дополнение к этой модели данных были определены правила для описания того, как преобразовать каждую физическую систему в модель, соответствующую OPC UA, чтобы представлять ее на сервере OPC UA. Каждый вид устройства, функции и системной информации может быть описан с использованием этой метамодели. Система базового типа поддерживает отношения между объектами, так называемые ссылки, а также множественное наследование. Таким образом, его можно сравнить с современным объектно-ориентированным языком программирования. Базовая модель предоставляет типы объектов и переменных, а также ссылочные и типы данных. На основе этой модели OPC UA может представлять каждый вид данных, включая их метаданные и семантику.

Модель данных OPC UA формирует базу для информационных моделей UA. Будучи специализированными моделями, они расширяют базу, добавляя определенные функции, такие как Data Acess, Alarms and Conditions, Historical Access или Programs (рис. 5).

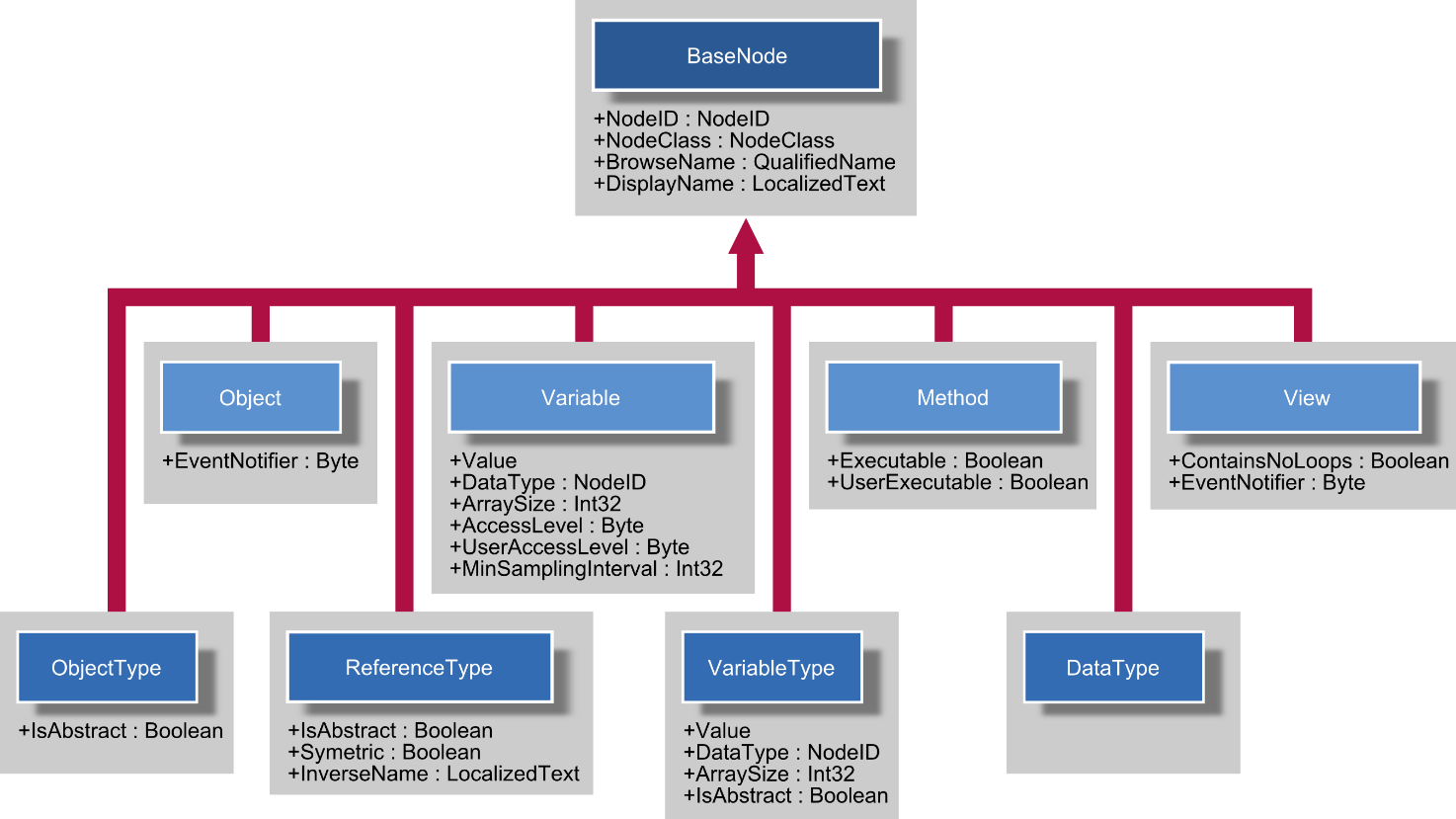


Рисунок 5 – Информационная модель объектов OPC UA

Модель данных и типов произвольно расширяема. Для сервера необходимо предоставить модель своего типа, особенно если он предоставляет типы, которые не являются базовыми типами UA, так называемые «хорошо известные типы». Цель состоит в том, чтобы указать расширения, насколько это возможно, в зависимости от отраслевых моделей. Это разработано в дополнительных спецификациях OPC [6].

## Концепция и реализация системы на базе OPC UA

Система на базе ОРС UA может содержать множество клиентов и серверов. Каждый клиент может работать параллельно с несколькими серверами и каждый сервер может обслуживать нескольких клиентов.

Структура клиента показана на рис. 6. Клиентская программа выполняет запросы сервисов ОРС сервера через внутренний интерфейс, который является изолирующей прослойкой между программой и коммуникационным стеком. Коммуникационный стек конвертирует запросы клиентской прикладной программы в сообщения для вызова необходимого сервиса, которые посылает серверу. После получения ответа на запросы коммуникационный стек передает их в клиентскую программу.

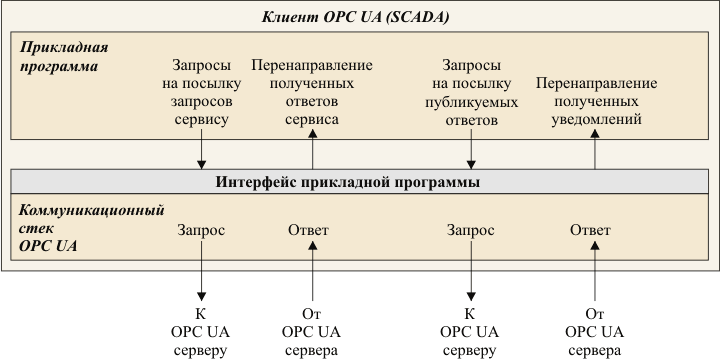


Рисунок 6 – Структура клиентской программы в стандарте ОРС UA

Структура сервера ОРС UA представлена на рис. 7. Модули ввода-вывода, ПЛК, интеллектуальные устройства и программы, которые могут поставлять данные через ОРС сервер, обозначены на как "реальные объекты". Серверное приложение представляет собой программную реализацию функций, которые должен выполнять сервер. Взаимодействие ОРС UA сервера с клиентом выполняется через интерфейс прикладной программы, путем отправления запросов и получения ответов.

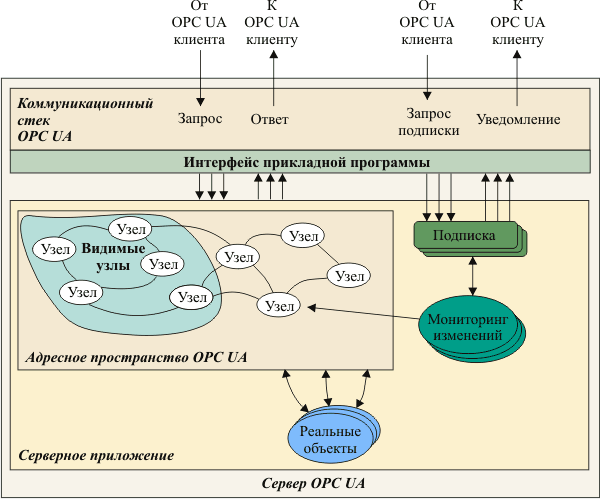


Рисунок 7 – Структура сервера в стандарте ОРС UA

Адресное пространство OPC сервера представляет собой множество узлов, доступных клиентской программе с помощью сервисов ОРС UA. "Узлы" в адресном пространстве используются, чтобы представить реальные объекты, их определения и перекрестные ссылки. В адресном пространстве выделяется подпространство узлов, которые сервер делает "видимыми" для клиента. Видимые узлы организуются в виде иерархической структуры, для удобства навигации их клиентской программой.

Обмен данными между клиентом и сервером может выполняться как путем получения мгновенных ответов на запросы, так и по схеме "издатель-подписчик" как это реализовано в протоколе MQTT.

ОРС UA допускает обмен между двумя серверами. Для этого один из серверов выступает в роли клиента, второй - в роли сервера. Пользовательское приложение (например, SCADA) может создавать комбинированные группы клиентов и серверов для ретрансляции сообщений, которыми оно обменивается с другими клиентами и серверами, как показано на рис. 8. При этом ОРС UA клиент и сервер могут быть скомбинированы в одном приложении для взаимодействия с другими ОРС UA клиентами и серверами.

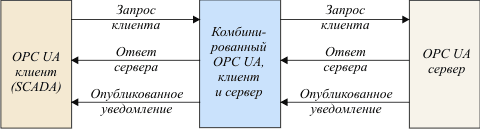


Рисунок 8 – Комбинирование клиента и сервера OPC UA

Реализации функций клиента и сервера OPC UA с использованием языка Python и свободно-распространяемой библиотеки FreeOpcUa [7] представлены в приложении А.

# Проектирование модуля для подключения устройств по Modbus

## Описание протокола

**Modbus** – коммуникационный протокол, основан на архитектуре ведущий-ведомый (master-slave). Использует для передачи данных интерфейсы RS-485, RS-422, RS-232, а также Ethernet сети TCP/IP (протокол Modbus TCP).

Сообщение Modbus RTU состоит из адреса устройства SlaveID, кода функции, специальных данных в зависимости от кода функции и CRC контрольной суммы (рис. 9).

|  |  |  |  |
| --- | --- | --- | --- |
| SlaveID | Код функции | Специальные данные | CRC |

Рисунок 9 – Содержимое сообщения Modbus RTU

SlaveID – это адрес устройства, может принимать значение от 0 до 247, адреса с 248 до 255 зарезервированы.

Данные в модуле хранятся в 4 таблицах. Две таблицы доступны только для чтения и две для чтения-записи. В каждой таблице помещается 9999 значений.

В сообщении Modbus используется адрес регистра (табл. 1). Например, **первый** регистр AO Holding Register, имеет **номер** 40001, но его **адрес** равен 0000. Разница между этими двумя величинами есть смещение (offset). Каждая таблица имеет свое смещение, соответственно: 1, 10001, 30001 и 40001.

Таблица 1. Распределение памяти устройства по регистрам Modbus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер регистра | Адрес регистра HEX | Тип | Название | Тип |
| 1-9999 | 0000 до 270E | Чтение-запись | Discrete Output Coils | DO |
| 10001-19999 | 0000 до 270E | Чтение | Discrete Input Contacts | DI |
| 30001-39999 | 0000 до 270E | Чтение | Analog Input Registers | AI |
| 40001-49999 | 0000 до 270E | Чтение-запись | Analog Output Holding Registers | AO |

## Виды команд

Приведем таблицу с кодами функций чтения и записи регистров Modbus (табл. 2).

Таблица 2. Виды команд Modbus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код функции | Описание функции | | Тип значения | Тип доступа |
| 01 (0x01) | Чтение DO | Read Coil Status | Дискретное | Чтение |
| 02 (0x02) | Чтение DI | Read Input Status | Дискретное | Чтение |
| 03 (0x03) | Чтение AO | Read Holding Registers | 16 битное | Чтение |
| 04 (0x04) | Чтение AI | Read Input Registers | 16 битное | Чтение |
| 05 (0x05) | Запись одного DO | Force Single Coil | Дискретное | Запись |
| 06 (0x06) | Запись одного AO | Preset Single Register | 16 битное | Запись |
| 15 (0x0F) | Запись нескольких DO | Force Multiple Coils | Дискретное | Запись |
| 16 (0x10) | Запись нескольких AO | Preset Multiple Registers | 16 битное | Запись |

## Программы для работы с протоколом

Примеры программ, которые позволяют работать с Modbus:

1. **DCON Utility Pro** с поддержкой Modbus RTU, ASCII, DCON (рис. 10).
2. **Modbus Master Tool** с поддержкой Modbus RTU, ASCII, TCP (рис. 11).
3. **Modbus TCP client** с поддержкой Modbus TCP (рис. 12).

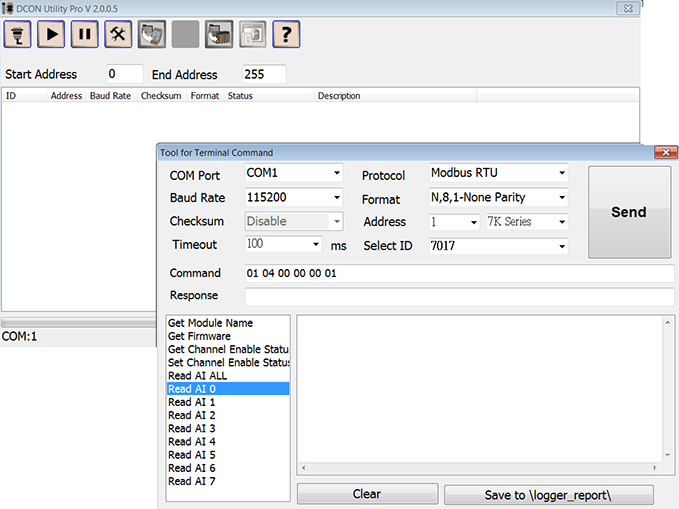


Рисунок 10 – Пример работы с программой DCON Utility Pro

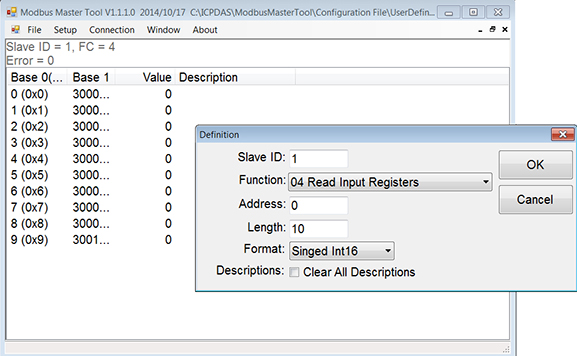


Рисунок 11 – Пример работы с программой Modbus Master Tool

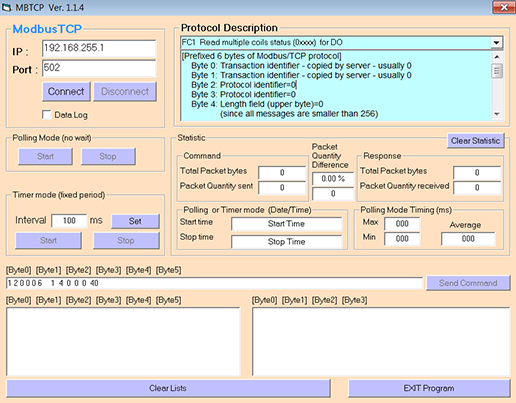


Рисунок 12 – Пример работы с программой Modbus TCP client

## Пример обмена информацией по протоколу

Ниже приведен пример запроса Modbus RTU для получения значения аналогового выхода из регистров (AO Holding Register) от 40108 до 40110 с адресом устройства 17 (рис. 13).

**11 03 006B 0003 7687**

|  |  |
| --- | --- |
| **11** | Адрес устройства SlaveID (17 = 11 hex) |
| **03** | Функциональный код Function Code  (03 – чтение Analog Output Holding Registers) |
| **006B** | Адрес первого регистра (40108-40001 = 107 =6B hex) |
| **0003** | Количество требуемых регистров  (чтение 3-х регистров с 40108 по 40110) |
| **7687** | Контрольная сумма CRC |

Рисунок 13 – Пример запроса

В ответе от Modbus RTU Slave устройства мы получим (рис. 14).

**11 03 06 AE41 5652 4340 49AD**

|  |  |  |
| --- | --- | --- |
| **11** | Адрес устройства (17 = 11 hex) | SlaveID |
| **03** | Функциональный код | Function Code |
| **06** | Количество байт далее (6 байтов идут следом) | Byte Count |
| **AE** | Значение старшего разряда регистра (AE hex) | Register value Hi (AO0) |
| **41** | Значение младшего разряда регистра (41 hex) | Register value Lo (AO0) |
| **56** | Значение старшего разряда регистра (56 hex) | Register value Hi (AO1) |
| **52** | Значение младшего разряда регистра (52 hex) | Register value Lo (AO1) |
| **43** | Значение старшего разряда регистра (43 hex) | Register value Hi (AO2) |
| **40** | Значение младшего разряда регистра (40 hex) | Register value Lo (AO2) |
| **49** | Контрольная сумма | CRC value Hi |
| **AD** | Контрольная сумма | CRC value Lo |

Рисунок 14 – Пример ответа на запрос

Регистр аналогового выхода AO0 имеет значение **AE 41** HEX или 44609 в десятичной системе. Регистр аналогового выхода AO1 имеет значение **56 52** HEX или 22098 в десятичной системе. Регистр аналогового выхода AO2 имеет значение **43 40** HEX или 17216 в десятичной системе.

Значение AE 41 HEX - это 16 бит 1010 1110 0100 0001, может принимать различное значение, в зависимости от типа представления (табл. 3). Значение регистра 40108 при комбинации с регистром 40109 дает 32 бит значение.

Таблица 3. Примеры представлений значений

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип представления** | **Диапазон значений** | **Пример в HEX** | **Будет в десятичной форме** |
| 16-bit unsigned integer | 0 до 65535 | AE41 | 44,609 |
| 16-bit signed integer | -32768 до 32767 | AE41 | -20,927 |
| two character ASCII string | 2 знака | AE41 | ® A |
| discrete on/off value | 0 и 1 | 0001 | 0001 |
| 32-bit unsigned integer | 0 до 4,294,967,295 | AE41 5652 | 2,923,517,522 |
| 32-bit signed integer | -2,147,483,648 до 2,147,483,647 | AE41 5652 | -1,371,449,774 |
| 32-bit single precision IEEE floating point number | 1,2·10−38 до 3,4×10+38 | AE41 5652 | -4.395978 E-11 |
| four character ASCII string | 4 знака | AE41 5652 | ® A V R |

## Особенности работы протокола по сети Ethernet

Modbus RTU не позволяет устройствам обмениваться информацией на дальние расстояния используя широко распространенные сети Ethernet, на которых построена глобальная сеть Интернета. Это ограничение можно преодолеть с использованием Modbus TCP.

Modbus TCP – это сетевой протокол обмена данных, который представляет собой симбиоз RTU спецификации протокола и Ethernet-TCP/IP. На ряду с RTU, Modbus TCP использует тот же прикладной уровень сетевой модели, где и достигается совместимость на уровне обработки данных (рис. 15).

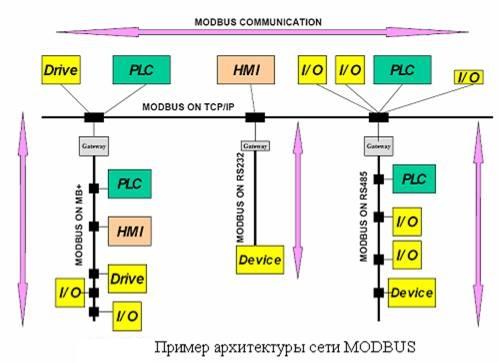
[](http://autoworks.com.ua/wp-content/uploads/2012/03/%D1%80%D0%B8%D1%81.6.27.jpg)

Рисунок 15 – Пример архитектуры Modbus

Modbus TCP считается эффективным сетевым решением для промышленности. На сегодняшний день Modbus TCP является одним из самых распространенных протоколов из семейства Industrial Ethernet.

Используя Modbus TCP можно получить, практически, ничем не ограниченное, количество узлов в сети, скорость передачи от 10 до 1000 Мбит/с, протяженность линий связи до 352000м (при использовании оптоволокна).

Коммуникационная система Modbus TCP позволяет участвовать в обмене и устройствами на последовательных линиях связи (RTU/ASCII) (рис. 16).

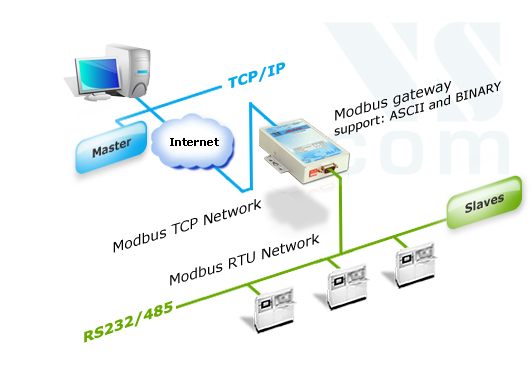


Рисунок 16 – Организация межсетевого взаимодействия между устройствами Modbus RTU и TCP

При такой конфигурации клиент сети TCP (он же мастер сети ModBus) подключается к шлюзу (серверу) и ведёт общение только с ним. Шлюз же переадресует сообщение внутри шины ModBus (RTU или ASCII) тому устройству, адрес которого указан в ModBus-пакете.

Структура сообщения Modbus TCP схожа с RTU спецификацией, однако к ней еще добавляется специализированный MBAP-заголовок (рис. 17).

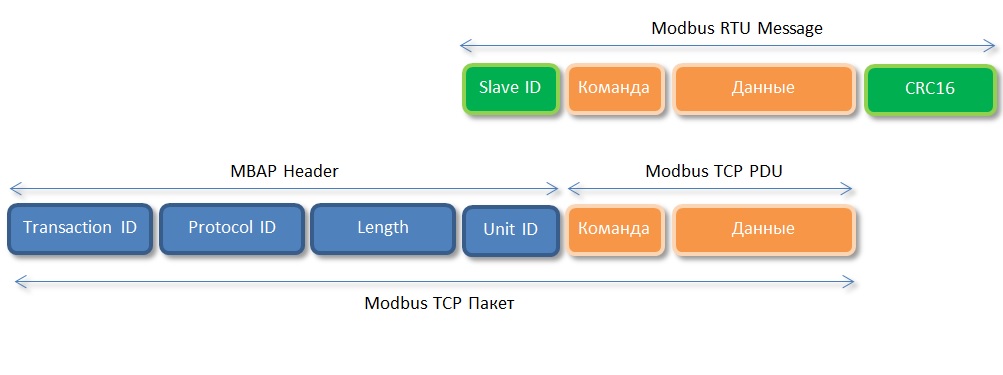


Рисунок 17 – Структура сообщений Modbus TCP и RTU

Поля MBAP заголовка стандарта Modbus TCP и их описание представлены в табл. 4.

Таблица 4. Поля MBAP заголовка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Поле** | **Длина (байт)** | **Пояснение** | **Клиент** | **Сервер** |
| TransactionID | 2 | идентификация транзакций запросов/ответов | инициализирует Клиент | копирует из запроса в сообщение-ответ |
| ProtocolID | 2 | тип протокола | инициализирует Клиент | копирует из запроса в сообщение-ответ |
| Length | 2 | количество следующих байт | инициализирует Клиент | инициализирует Сервер в ответе |
| UnitID | 1 | адрес Ведомого, который подключен к узлу | инициализирует Клиент | копирует из запроса в сообщение-ответ |

Конвертировать запросы между Modbus RTU и Modbus TCP довольно просто, и реализация Modbus RTU через TCP может показаться простым способом для маршрутизации запросов, на самом деле в Modbus TCP есть несколько положительных моментов:

* не нужно вычислять CRC16;
* есть возможность идентифицировать пару ответ/запрос используя TransactionID;
* можно легко добавлять свои версии протоколов, меняя ProtocolID;

Разработка модуля возможна с использованием open-source библиотек на языке Java [8] или Python [9]. Программа для взаимодействия с устройством представлена в приложении Б.

# Проектирование модуля для подключения устройств по MQTT

## Описание протокола

MQTT (Message Queuing Telemetry Transport) – это простой открытый протокол, разработанный специально для IoT и применяемый для обмена данными между устройствами. MQTT-сеть включает в себя MQTT-брокера, который служит посредником во взаимодействии MQTT-агентов – издателей и подписчиков. Издатели публикуют информацию, предназначенную для подписчиков (рис. 18).

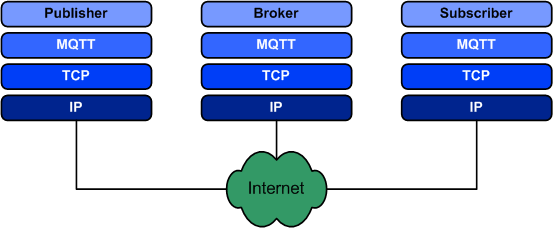


Рисунок 18 – Брокер, издатель и подписчик в MQTT-сети

MQTT разработан в расчёте на маломощные встроенные устройства, поэтому вычислительные требования для его реализации минимальны. В дополнение к очень низкой нагрузке на системы, MQTT отличается высокой эффективности связи даже в сетях с низкой пропускной способностью.

Протокол MQTT работает на прикладном уровне поверх TCP/IP и использует по умолчанию 1883 порт (8883 при подключении через SSL) (рис. 19).

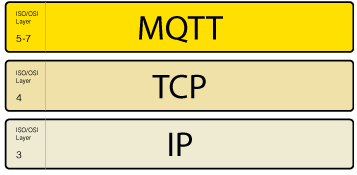


Рисунок 19 – Место протокола MQTT в сетевой модели OSI

MQTT реализует модель «издатель – подписчик», используя минимальное количество методов. Они служат для указания действий, которые нужно выполнять. Эти действия сводятся к взаимодействию с брокером и к работе с темами и сообщениями. Агенты подключаются к брокеру, а затем либо публикуют темы и сообщения в них, либо подписываются на темы и получают сообщения, в этих темах опубликованные. Завершив работу, агент отключается от брокера.

Устройства MQTT используют определенные типы сообщений для взаимодействия с брокером, ниже представлены основные:

* Connect – установить соединение с брокером;
* Disconnect – разорвать соединение с брокером;
* Publish – опубликовать данные в канал на брокере;
* Subscribe – подписаться на канал на брокере;
* Unsubscribe – отписаться от канала.

На рис. 20 представлена упрощённая схема взаимодействия между подписчиком и издателем с использованием MQTT-брокера.

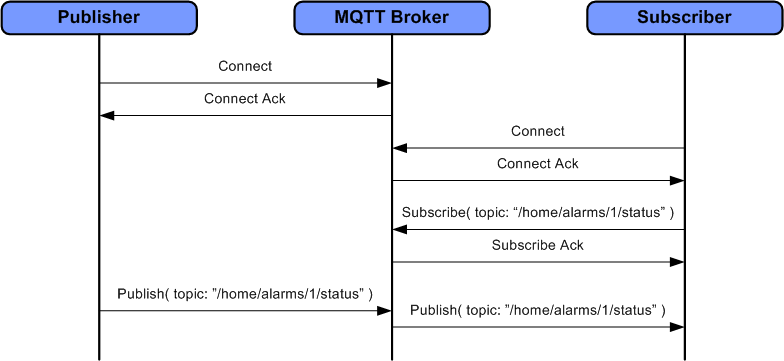


Рисунок 20 – Взаимодействие подписчика и издателя

Издатель, который является источником неких данных, подключается к брокеру. Подписчик, потребитель информации, делает то же самое и подписывается на тему, которая представлена здесь как «/home/alarms/1/status». В данном примере в этой теме публикуются сведения об изменениях состояния домашней сигнализации в некоей «зоне №1». Когда у издателя есть новые данные, он обращается к брокеру и публикует сообщение в этой теме. Брокер же, в свою очередь, передаёт опубликованное сообщения всем, кто на эту тему подписан.

MQTT также позволяет пользоваться подстановочными символами, что упрощает процесс подписки. Если подписчик, например, желает знать о состояниях всех датчиков сигнализации, он может подписаться на тему следующего вида: «/home/alarms/+/status». В результате, он будет уведомлён о срабатываниях сигнализации во всех зонах, а не только в «зоне №1», как в вышеприведённом примере. Подписаться на целое поддерево тем можно с помощью шаблонна со знаком «#». Например, подписка на «/home/#» позволит получать сообщения обо всём, что происходит в темах, находящихся ниже узла «/home».

## Качество обслуживания

MQTT поддерживает указание уровня качества обслуживания (QoS). А именно, существуют три таких уровня:

* QoS 0. Этот уровень задействует стратегию «максимум однократная доставка сообщений». Приёмник сообщения не подтверждает их получение, отправитель, соответственно, передаёт сообщение лишь раз, не предпринимая попыток по их повторной передаче. Это – метод «отправил и забыл» (рис. 21).



Рисунок 21 – Взаимодействие издателя и брокера по уровню QoS0

* QoS 1. Здесь применяется подход «минимум однократная доставка сообщений». Гарантируется, что приёмник получит сообщение хотя бы один раз. При этом подписчик может получить одно и то же сообщение несколько раз. А отправитель будет предпринимать повторные попытки отправки до тех пор, пока не получит подтверждение в успешной доставке сообщения (рис. 22).

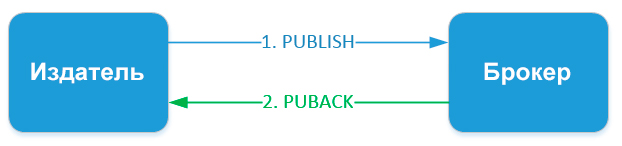


Рисунок 22 – Взаимодействие издателя и брокера по уровню QoS1

* QoS 2. Этому уровню качества обслуживания соответствует самая медленная процедура доставки сообщений, но при этом он – самый надёжный. Его основная особенность – реализация стратегии «однократная доставка сообщений». При его использовании применяется четырёхступенчатая процедура подтверждения доставки сообщений (рис. 23).

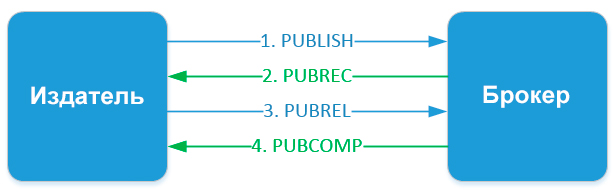


Рисунок 23 – Взаимодействие издателя и брокера по уровню QoS2

Выбор конкретного уровня качества обслуживания зависит от особенностей передаваемых данных и от того, насколько важно, чтобы они были доставлены.

## Защита передачи данных

Для обеспечения безопасности в MQTT протоколе реализованы следующие методы защиты:

* *Аутентификация клиентов.*

Пакет CONNECT может содержать в себе поля USERNAME и PASSWORD. При реализации брокера можно использовать эти поля для аутентификации клиента

* *Контроль доступа клиентов через Client ID.*
* *Подключение к брокеру через TLS/SSL.*

## Описание процедуры первого подключения устройства к системе

Подключение устройства к серверу будет производиться по зашифрованному каналу (SSL/TLS шифрование) используя логин и пароль. Логином будет GUID, пароль будет выдаваться устройству на этапе изготовления при его первом включении. Процесс получения GUID и пароля показан на рис. 24.



Рисунок 24 – Процесс получения GUID и пароля новым устройством

После получения GUID и пароля, устройство переходит в режим 1 – «Авторизовано». Далее устройство может подключаться к серверу по умолчанию для получения настроек и разрешения на сбор и передачу данных.

## Описание методов передачи параметров устройству

Процесс получения настроек происходит следующим образом:

1. Устройство подключается к серверу по умолчанию, используя выданные ранее GUID и пароль.
2. Устройство передает информацию о себе и подключенных входах/выходах.
3. Устройство получает первоначальные конфигурации и индивидуальные настройки, произведенные пользователем.

Данный процесс наглядно представлен на диаграмме (рис. 25).



Рисунок 25 – Процесс настройки устройства

Подключившись к серверу, устройства отправляет свои текущие настройки серверу путем сообщения PUBLISH ATTRIBUTES в JSON-формате (листинг 1).

Листинг 1. Пример сообщения, содержащего информацию об атрибутах устройства

{

"name": "INSYTE EBR-21",

"devid": 123546324,

"typeid": 23454564,

"last\_changed": "2017-12-02T10:01:06Z",

"start\_time": "2017-12-02T13:04:04Z",

"server": "ems.insyte.ru",

"keeping": "disabled",

"sending": "disabled",

"variables": [

{

"name": "var1",

"type": "in"

},

{

"name": "var2",

"type": "out"

},

{

"name": "var3",

"type": "in/out"

}

]

}

Сервер сверяет их с теми, что были найдены в БД для данного устройства, если они новее производит обновление хранимых настроек в БД. Если же БД содержит настройки актуальнее тех, что были получены от устройства, сервер оповещает устройство о необходимости обновить параметры. Для включения возможности сбора и отправки данных, устройство должно быть закреплено за пользователем и включена такая возможность в системе вручную или автоматически. Устройство о включении узнает от сервера при получении новых настроек, прочитав поле **sending** из сообщения. Если оно установлено в "enabled", то устройство имеет право производить отправку собираемых с сенсоров значений. В результате установки данного флага устройство переходит в режим 2 – «Производится сбор и отправка значений»

## Сбор, хранение и отправка значений измерений

После того как устройство было сконфигурировано и получило свои начальные настройки автоматически от сервера (или вручную), оно переходит в режим 2. В этом режиме в зависимости от выставленных флагов устройство может производить сбор и хранение значений изменений в энергонезависимой памяти (флаг **keeping**) или отправку значений на сервер (флаг **sending**). Если установлены оба флага в состояние *enabled*, то при наличии подключения к сети передачи данных и доступности сервера устройство производит отправку текущих измерений и таким образом обеспечивается режим близкий к режиму реального времени, в случае отсутствия доступа к сети устройство сохраняет значения в память для последующей отправки их на сервер.

Отправка производится в виде сообщений в JSON-формате (листинг 2).

Листинг 2. Пример сообщения, содержащий данные измерений

{

"timestamp": "2017-12-02T10:01:06Z",

"value": 22.4

}

В случае если требуется отправить только текущее значение измерение – поле **timestamp** может быть исключено из сообщения. В случае, если требуется отправить несколько значений одного канала измерений, различающихся по времени одной посылкой – их можно упаковать в следующее сообщение (листинг 3).

Листинг 3. Пример сообщения, содержащего несколько измерений

[

{

"timestamp": "2017-12-02T10:01:06Z",

"value": 22.4

},

{

"timestamp": "2017-12-02T10:01:16Z",

"value": 28.4

},

{

"timestamp": "2017-12-02T10:01:26Z",

"value": 31.5

}

]

Также можно отправлять в одном пакете разные измерения, проведенные одновременно (листинг 4).

Листинг 4. Пример сообщения, содержащего одновременное измерение нескольких величин

{

"timestamp": "2017-12-02T10:01:06Z",

"variables": [

{"name": "input1",

"value": 22.4},

{"name": "input2",

"value": 28.4},

{"name": "input3",

"value": 31.5}

]

}

Особенности программной реализации представлены в других отчетах, в данном же отчете укажем, что реализация протокола MQTT может быть осуществлена с помощью библиотек программирования на Java [10] или Python [11]. Для демонстрации возможностей такой реализации в приложении В приведен исходный код, позволяющий выполнить базовый функции обмена информацией.

# Проектирование пользовательских интерфейсов

Один из основных интерфейсов – интерфейс добавления новых устройств. Полностью он представлен в приложении Г. Ниже приведем моменты касательно ввода настроек для подключения устройств к серверу с помощью различных протоколов, рассмотренных ранее.

Для подключения устройства к серверу по протоколу MQTT используется форма ввода, представленная на рис. 26.

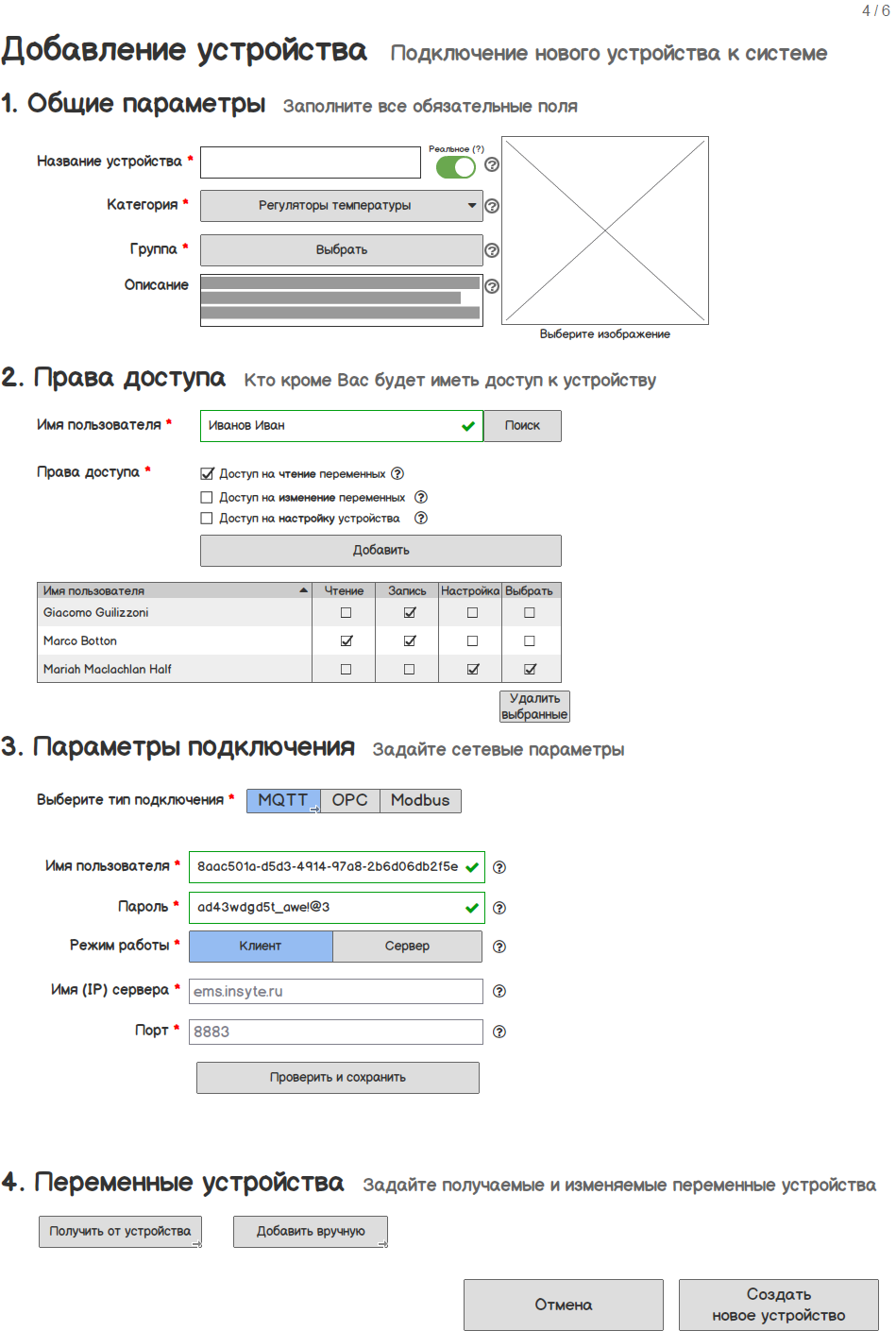


Рисунок 26 – Процесс настройки параметров подключения устройства по MQTT

Подключение устройств с использованием OPC UA возможно после заполнения формы с параметрами подключения на изображенными на рис. 27.

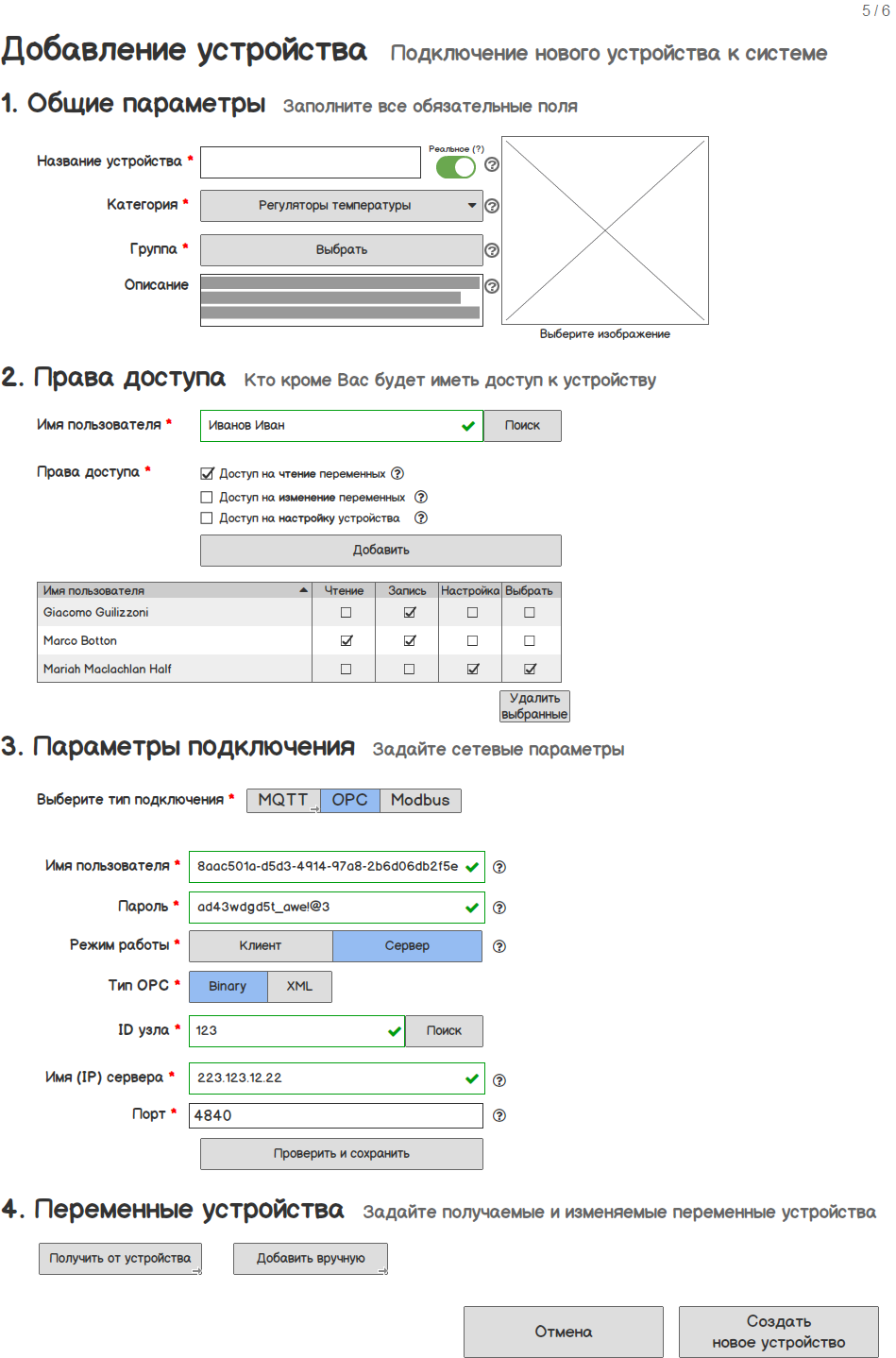


Рисунок 27 – Процесс настройки параметров подключения устройства по OPC

Устройства, предоставляющие доступ к данным по протоколу Modbus, требуют задания следующих параметров для подключения (см. рис. 28).

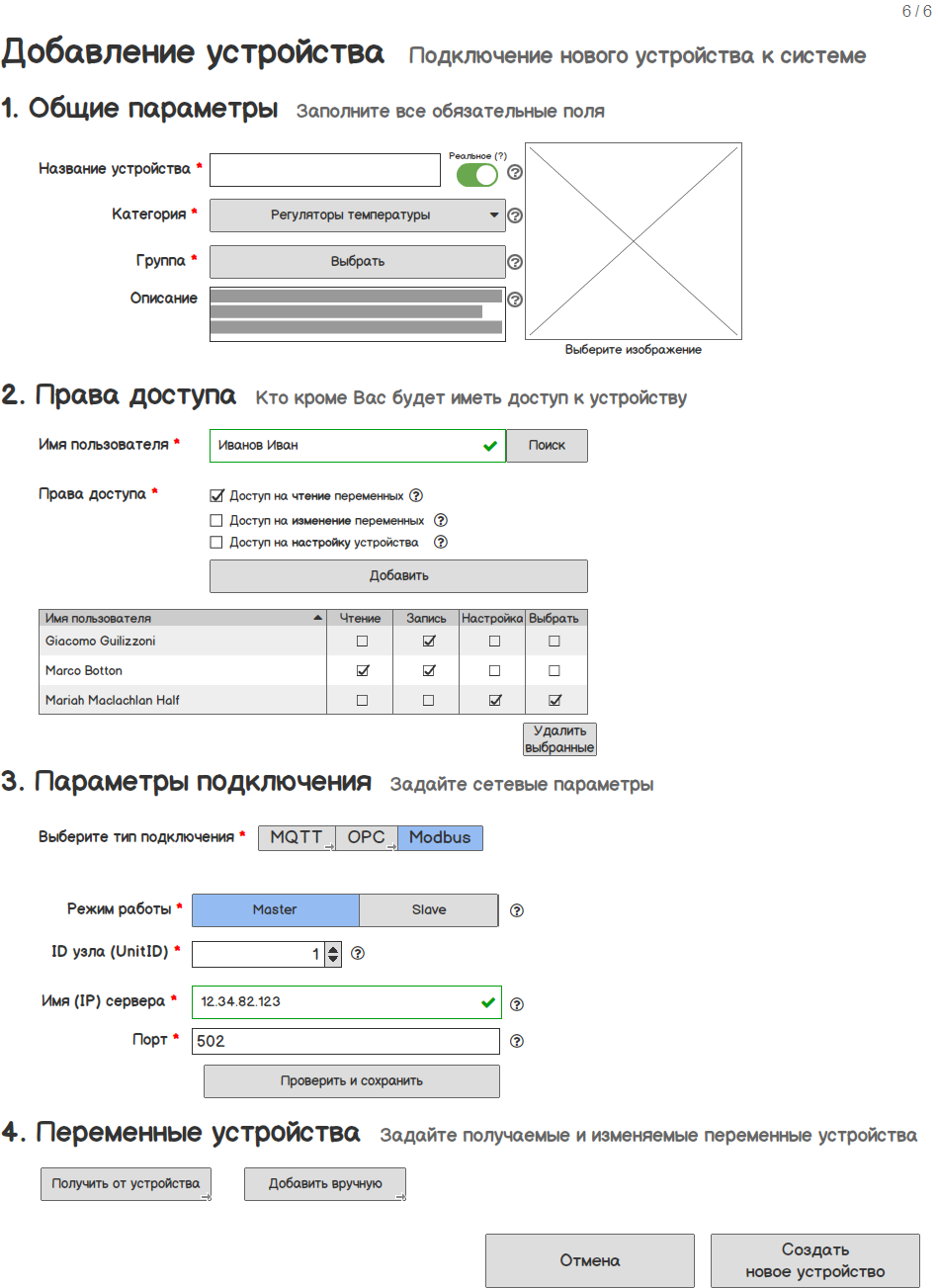


Рисунок 28 – Процесс настройки параметров подключения устройства по Modbus

При этом можно видеть, что в формах также представлены режимы работы, которые могут обеспечивать имитацию системы EMS как устройства (режим Сервер в протоколе OPC, например) и тем самым предоставляется возможность подключения к сторонним системам для передачи (экспорта) параметров как это делает обычное устройство.

# Заключение

MQTT, Modbus, OPC – технологии очень разные, но вместе они помогают строить надёжные IoT-решения. Modbus используется в качестве локального интерфейса для взаимодействия с устройствами, OPC – обеспечивает межсистемное взаимодействие со SCADA системами, роль MQTT – организация глобальных связей между компонентами системы. Каждый из них играет важную роль. При этом шлюзы для интернета вещей использующие все протоколы и позволяющие строить надежные соединения будут в ближайшее время находиться в особенном спросе, и на них будет стоять задача предоставить общую стабильность и безопасность IoT проектов. С помощью таких шлюзов можно быстро создавать надёжные современные решения, которые будут оставаться актуальными все время.

# Список литературы

1. Богданов Н., Киселёва О. OPC Unified Architecture: изменения в популярной технологии информационных обменов с точки зрения инженера //Современные технологии автоматизации. – 2010. – №. 3. – С. 82.
2. OPC UA - новый виток эволюции / И.В. Иванов // ООО "ИнСАТ". URL: http://insat.ru/articles/?id=49272 (дата обращения: 1.12.2017).
3. Фортин Т., Хокинсон Б. OPC UA и роль стандартов связи в развитии промышленного Internet вещей // Автоматизация в промышленности. 2016. № 8.
4. Mahnke W., Leitner S.H., Damm M. OPC Unified Architecture. Berlin: Springer, 2009.
5. OPC UA Protocols. URL: http://www.ascolab.com/en/technology-unified-architecture/technology-protocols.html (дата обращения: 1.12.2017).
6. OPC Foundation. URL: http://www.opcfoundation.com/ (дата обращения: 1.12.2017)
7. FreeOpcUa: Open Source C++ and Python OPC-UA Server and Client Libraries and Tools. URL: http://freeopcua.github.io/ (дата обращения: 1.12.2017).
8. High-performance, non-blocking, zero-buffer-copying Modbus for Java. URL: https://github.com/digitalpetri/modbus (дата обращения: 1.12.2017).
9. A full modbus protocol written in python. URL: http://riptideio.github.io/pymodbus (дата обращения: 1.12.2017).
10. Vert.x MQTT. URL: https://github.com/vert-x3/vertx-mqtt (дата обращения: 1.12.2017)
11. Eclipse Paho MQTT Python client library. URL: https://github.com/eclipse/paho.mqtt.python (дата обращения: 1.12.2017).

# Приложение А Реализация базовых функций клиента и сервера OPC UA

Листинг 1. Работа в режиме клиента OPC UA

**from** opcua **import** Client

client = Client("opc.tcp://localhost:4840/")

client.connect()

*# Получение корневого узла*

root = client.get\_root\_node()

**print**("Objects node is: ", root)

*# Определение адресного пространства и списка дочерних узлов*

**print**("Children of root are: ", root.get\_children())

*# Получение конкретного узла, зная его ID*

var = client.get\_node(ua.NodeId(1002, 2))

**print**(var)

*# Получение значения заданного узла*

var.get\_value()

*# Установка значения заданного узла с использованием ожидаемого типа данных*

var.set\_value(3.9)

*# Получение переменной используя путь до нее*

myvar = root.get\_child(["0:Objects", "2:MyObject", "2:MyVariable"])

**print**("myvar is: ", myvar)

Листинг 2. Работа в режиме сервера OPC UA

**from** opcua **import** Server

server = Server()

server.set\_endpoint("opc.tcp://0.0.0.0:4840/")

*# Записываем название сервера*

server.set\_server\_name("Server")

*# Настраиваем собственное пространство имен*

uri = "http://server"

idx = server.register\_namespace(uri)

*# Получаем ссылку на объект где будут располагаться наши узлы*

objects = server.get\_objects\_node()

*# Создаем объект и присваиваем ему имя*

Object\_1 =objects.add\_object(idx,'MyFirstObject')

Object\_2 =objects.add\_object(idx,'MySecondObject')

Object\_3 =objects.add\_object(idx,'MyThirdObject')

*# Cоздаем переменные*

Discret\_1 = Object\_1.add\_variable(idx,'Discret\_1',[0,0,0,0,0,0,0,0])

Discret\_2 = Object\_2.add\_variable(idx,'Discret\_2',[0,0,0,0,0,0,0,0])

Analog\_3 = Object\_3.add\_variable(idx,'Analog\_3',[10,20,30,40,50])

*# Запускаем сервер*

server.start()

# Приложение Б Реализация базовых функций клиента и сервера Modbus TCP

Листинг Б.1. Работа в режиме клиента Modbus TCP (режим Master)

from pymodbus.client.sync import ModbusTcpClient

*# Создаем объект клиента с подключением к localhost*

client = ModbusClient('127.0.0.1')

*# Записываем значение `1` в регистр DO по адресу 1*

client.write\_coil(1, True)

*# Читаем 1 значение начиная с 1 регистра DO*

result = client.read\_coils(1,1)

*# Печатаем результат в консоль*

print result.bits[0]

*# Закрываем соединение*

client.close()

Листинг Б.2. Работа в режиме сервера Modbus TCP (режим Slave)

from pymodbus.server.sync import StartTcpServer

*# Создаем объект для хранения значений регистров*

builder = BinaryPayloadBuilder(byteorder=Endian.Little)

*# Записываем в него значения*

builder.add\_bits([0, 1, 0, 1, 1, 0, 1, 0])

*# Создаем на основании регистров блок данных*

block = ModbusSequentialDataBlock(1, builder.to\_registers())

*# Присваиваем этот блок каждой области памяти*

store = ModbusSlaveContext(di=block, co=block, hr=block, ir=block)

*# Создаём сервер с определенным ранее контекстом (данными)*

context = ModbusServerContext(slaves=store, single=True)

*# Запускаем Modbus TCP сервер на порту 5020*

StartTcpServer(context, address=("localhost", 5020))

# Приложение В Реализация базовых функций клиента и сервера MQTT

Листинг В.1. Работа в режиме клиента MQTT (режим Subscriber)

from paho.mqtt import client

# Слушать канал "mqtt/paho/test" в ожидании новых сообщений

def **on\_connect**(client, userdata, rc):

print("Connected with result code: %s" % rc)

client.subscribe("mqtt/paho/test")

# При получении сообщения вывести его в консоль

def **on\_message**(client, userdata, msg):

print("%s: %s" % (msg.topic, msg.payload))

def **main**():

# Подключиться к брокеру по умолчанию на порт 1883

subscriber = client.Client()

subscriber.on\_connect = on\_connect

subscriber.on\_message = on\_message

subscriber.connect("localhost")

subscriber.loop\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Листинг В.2. Работа в режиме сервера MQTT (режим Publisher)

from paho.mqtt import publish

def **main**():

# Сформировать массив из двух сообщений "hello", "world"

msgs = [{'topic': "mqtt/paho/test", 'payload': "hello"},

{'topic': "mqtt/paho/test", 'payload': "world"}]

# Отправить их в канал "mqtt/paho/test" на сервер "localhost"

publish.multiple(msgs, hostname="localhost")

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Приложение Г Проектирование пользовательских интерфейсов

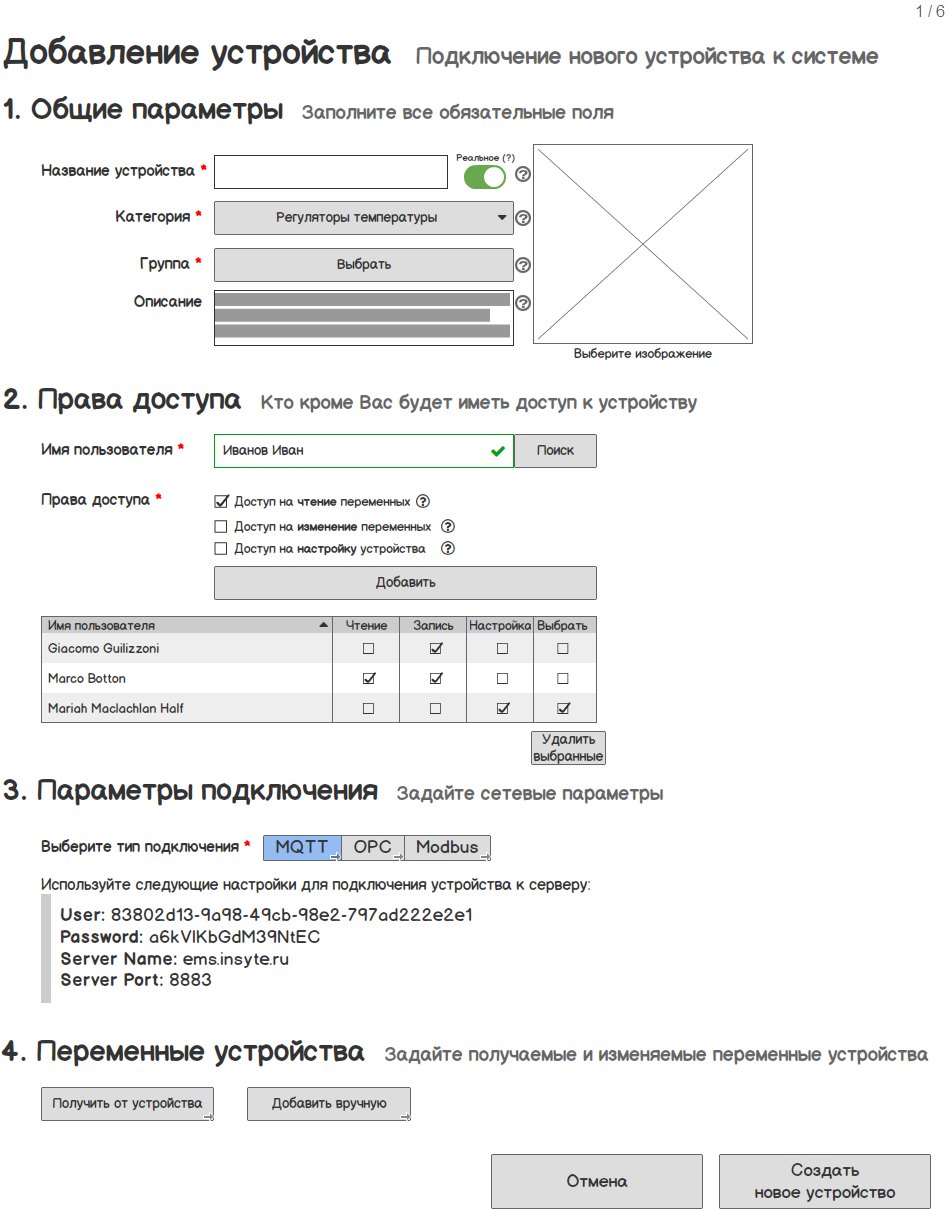


Рисунок Г.1 – Вариант интерфейса пользователя для добавления устройства