МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2019 г.

Разработка интерфейса системы расширения функционального взаимодействия компонентов интернета вещей

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____И.Л. Кафтанников
«___»_____2019 г.

Автор работы,
студент группы КЭ-222
_____К.Д. Суханов
«___»_____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____С.В. Сяськов
«___»_____2019 г.

Челябинск-2019

Аннотация

К.Д. Суханов. Разработка интерфейса системы расширения функционального взаимодействия компонентов интернета вещей. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 71 с., 17 ил., библиогр. список – 14 наим.

В рамках выпускной квалификационной работы производится обзор технологии «Интернет вещей», проводится анализ актуальных программных комплексов по управлению IoT-устройствами. Предлагается новый вариант системы для взаимодействия «умных устройств», производится разработка предложенного программного комплекса для управления IoT-устройствами. Разработанная система протестирована с помощью юнит-тестов, а также физических устройств. Написана документация для пользователей системой и разработчиков плагинов новых устройств. Предложены сценарии использования.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день стремительно развивается концепция вычислительной сети физических устройств с технологией взаимодействия друг с другом или внешней средой для выполнения операций или действий, исключающих необходимость участия человека – Интернет вещей. По оценкам компании Ericsson, в 2018 году число датчиков и устройств интернета вещей превысило количество мобильных телефонов, совокупный среднегодовой темп роста данного сегмента в период с 2015 по 2021 год ожидался на уровне 23%, к 2021 году прогнозируется, что из приблизительно 28 млрд подключённых устройств по всему миру, около 16 миллиардов будут так или иначе связаны в рамках концепции интернета вещей.[1] В связи с ростом количества устройств и популярностью данной концепции, возникают две проблемы. Первая проблема: необходимость создания расширяемой системы, которая позволяет взаимодействовать с компонентами интернета вещей различных производителей. Вторая: обучение студентов данной технологии для получения навыков создания сложных автоматизированных систем на основе компонентов интернета вещей.

В настоящее время существуют различные программы для работы со смарт-устройствами, но все они используют достаточно примитивные возможности описания сценариев (собственные скриптовые языки или визуальные блоки). Существующие программные комплексы работают некорректно со столь популярными и доступными смарт-устройствами Xiaomi [2]. Вышеперечисленные минусы существующего программного обеспечения не позволяют в полной мере использовать его для обучения.

Актуальность и необходимость создания такой системы, которая бы позволила настроить взаимодействие смарт-устройств различных

производителей, включая Xiaomi[3], а также использоваться для обучения, обусловлена следующими аспектами:

1. Добавление устройств различных производителей в систему, чтобы расширить возможности.
2. Поддержка языка общего назначения (например, C#), что позволит создавать сложные автоматизированные системы.
3. Использование в обучении создания автоматизированных систем.
4. Возможность эксплуатировать систему для популярного направления интернета вещей – создание умного дома.
5. Доступность для коммерческого использования.

Цель работы – разработка расширяемой системы взаимодействия компонентов интернета вещей, позволяющей создавать сложные сценарии на языке программирования общего назначения. Для осуществления поставленной цели необходимо реализовать следующие задачи:

1. Исследовать существующие программные продукты для работы со смарт-устройствами, выделить функциональные возможности и проблемы.
2. Произвести исследование языков общего назначения для написания системы и использования в качестве языка создания скриптов.
3. Разработать программный комплекс, позволяющий унифицировано управлять различными устройствами от различных производителей, расширяемый с помощью плагинов.
4. Создать API для взаимодействия с разрабатываемым программным комплексом.
5. Написать документацию по работе с программным продуктом.
6. Протестировать разработанный программный продукт.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На сегодняшний день IoT-устройства можно встретить практически во всех сферах, например, в домах, на заводах, объектах нефтедобычи, в больницах, автомобилях и других местах. IoT, или интернет вещей, — это сеть связанных через интернет объектов, способных собирать данные и обмениваться данными, поступающими со встроенных сервисов [4]. Основная задача данной технологии – снизить или полностью исключить участие человека в исполнении некоторых задач.

Стоит отметить высокий рост числа устройств в сфере IoT. Данная ниша интересна для инвесторов и производителей смарт-устройств. Из-за этого рынок насыщен различными устройствами от разных производителей. Каждая фирма пытается создать свою экосистему, чтобы покупатели пользовались только их IoT-устройствами и не могли интегрировать в систему устройства других производителей. Задача состоит в том, чтобы создать унифицированную систему, которая позволила бы использовать IoT-устройства различных производителей.

С каждым годом в оборот поступает все больше устройств, оснащенных интеллектуальными программами для сбора и обработки информации [5]. Но пока «Интернет вещей» еще не до конца сложившаяся система. Одной из главных проблем является отсутствие кадров. Такое положение сформировалось из-за того, что студентам трудно изучать технологии без наличия самих устройств, а в учебных программах редко присутствует предмет, который позволял бы провести подготовку учащихся в данной области. Поэтому задача состоит в том, чтобы подготовить базовую систему, которую можно использовать в обучающих целях, для подготовки будущих кадров.

1.1. ОБЗОР АНАЛОГОВ

В связи с высокой популярности концепции интернета вещей, существует множество программ для управления смарт-устройствами. Для анализа были выбраны самые популярные и актуальные системы.

«Domoticz» [6] - система домашней автоматизации Domoticz представляет собой бесплатную и открытую программную платформу для комплексного управления домашней автоматикой, а также для информационной поддержки жизнедеятельности. Данная система может быть установлена практически на любой персональный компьютер (на платформе Windows и Linux) и совершенно не требовательна к ресурсам. Скриншот приложения представлен на рисунке 1.

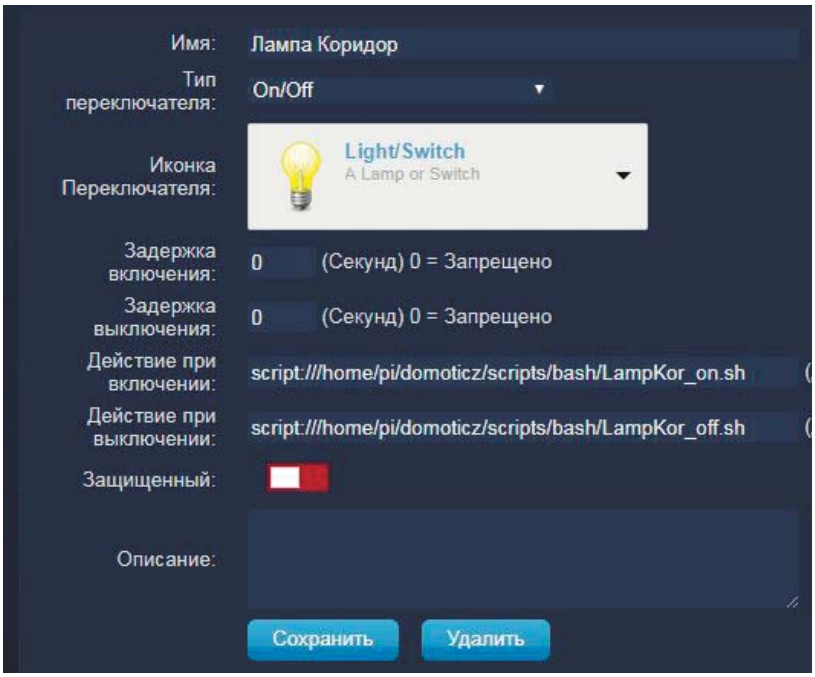

Рисунок 1 – Скриншот системы «Domoticz», настройка параметров для лампы

Достоинства:

– наличие мобильного приложения;
– кроссплатформенность;

– удобный веб-интерфейс.

Недостатки:

– отсутствует полноценная поддержка языка программирования общего назначения;
– плохая поддержка устройств Xiaomi.

«Home Assistant» [7] - open-source платформа для автоматизации, работающая на Python 3. Позволяет отслеживать и контролировать все устройства в доме и автоматизировать действия. Работает на платформах Windows, Linux. Интерфейс построен через браузер, поэтому взаимодействие возможно на любом устройстве Android, iOS. Скриншот приложения представлен на рисунке 2.

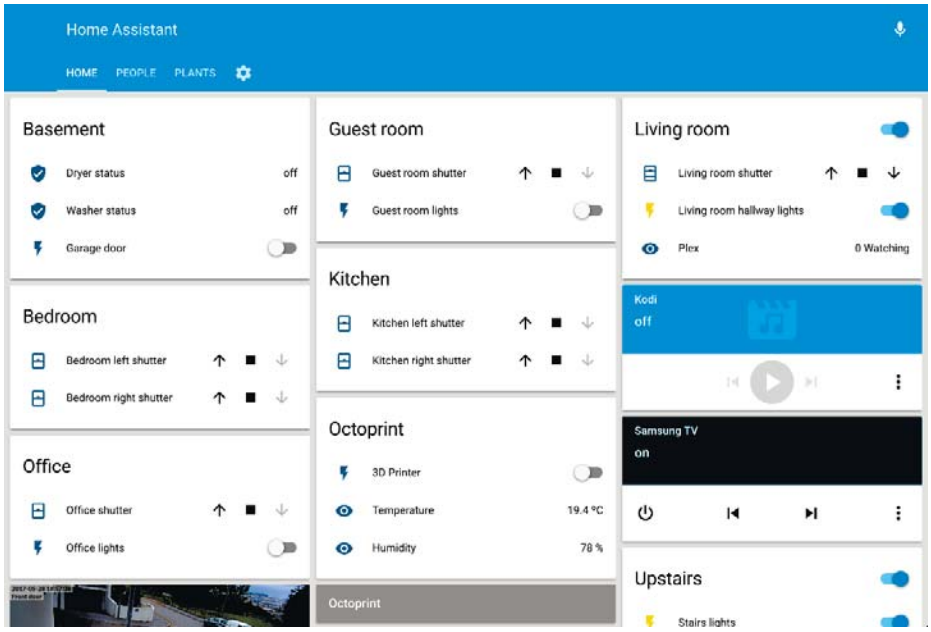

Рисунок 2 – Скриншот системы «Home Assistant»

Достоинства:

– наличие мобильного приложения;
– простой и понятный интерфейс для начинающего пользователя.

Недостатки:

– отсутствует полноценная поддержка языка программирования общего назначения;
– недостаточная стабильность.

«MajorDoMo» [8] - система домашней автоматизации MajorDoMo (Major Domestic Module или Главный Домашний Модуль) представляет собой бесплатную и открытую программную платформу для комплексного управления домашней автоматикой, а также для информационной поддержки жизнедеятельности. Данная система может работать на платформе Windows и Linux, не требовательна к ресурсам. Скриншот приложения представлен на рисунке 3.

Рисунок 3 – скриншот системы «MajorDoMo»

Достоинства:

– кроссплатформенность;
– простой интерфейс.

Недостатки:

– большая часть функционала предназначена для своих устройств;
– редкие обновления;
– отсутствует полноценная поддержка языка программирования общего назначения.

«openHAB» [9] - это платформа домашней автоматизации с открытым исходным кодом, основанная на Java и интегрирующая в себе большое количество систем "умного дома" в единое решение. На верхнем уровне абстракции все подключенные устройства доступны для правил автоматизации и различных пользовательских интерфейсов. Скриншот приложения представлен на рисунке 4.

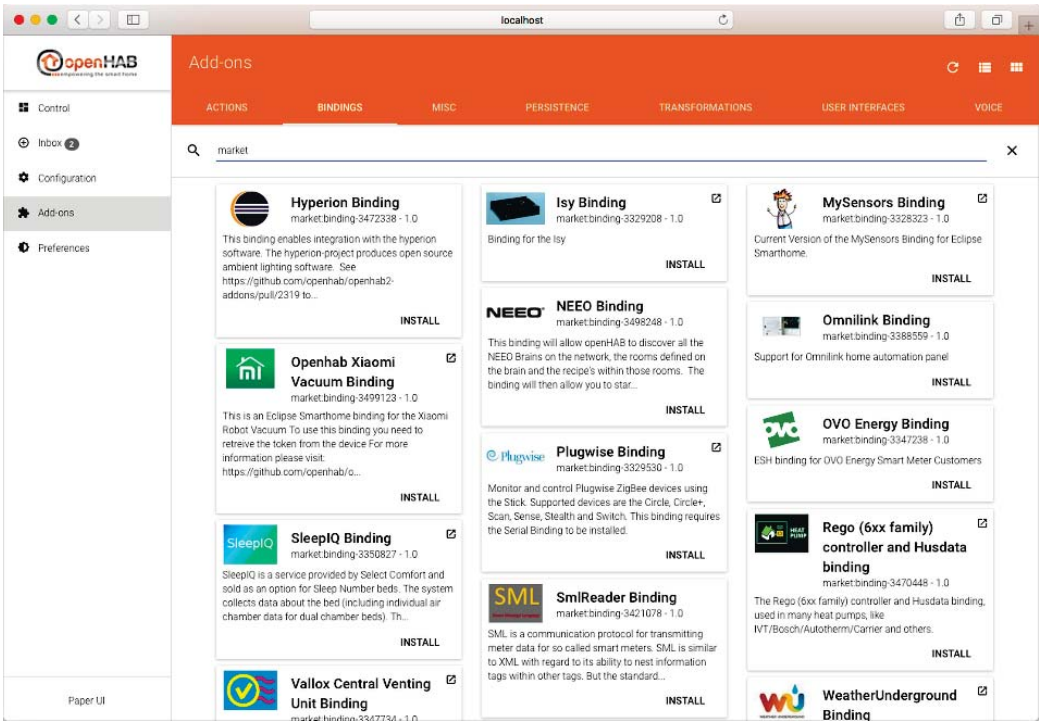

Рисунок 4 – скриншот системы «openHAB»

Достоинства:

– большое сообщество пользователей;
– возможность настройки интерфейса.

Недостатки:

– плохая документация;
– отсутствует полноценная поддержка языка программирования общего назначения.

1.3. ВЫВОД

Проведя исследования существующих программных комплексов в области управления IoT-устройствами, можно сделать вывод о том, что, чтобы система была востребована и уникальна, необходимо реализовать следующий функционал:

1. Возможность расширения функционала системы за счет добавления плагинов, которые увеличивают количество поддерживаемых устройств.
2. Поддержка написания скриптов автоматизации на языке программирования общего назначения.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Для реализации данной системы необходимы следующие компоненты:

1. Хост – программа предоставляющая единый интерфейс к плагинам. Она выступает связующим звеном между пользователем и IoT-устройствами.
2. Интерфейс для взаимодействия клиента с хостом. Необходим, чтобы пользователь смог получать список подключенных устройств, вызывать методы, просматривать свойства и подписываться на события.
3. Генератор интерфейса к устройствам – программа, предназначенная для более удобного написания кода путем генерации интерфейсов к устройствам.
4. Система плагинов – позволяет расширять функционал системы, путем добавления новых устройств.

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

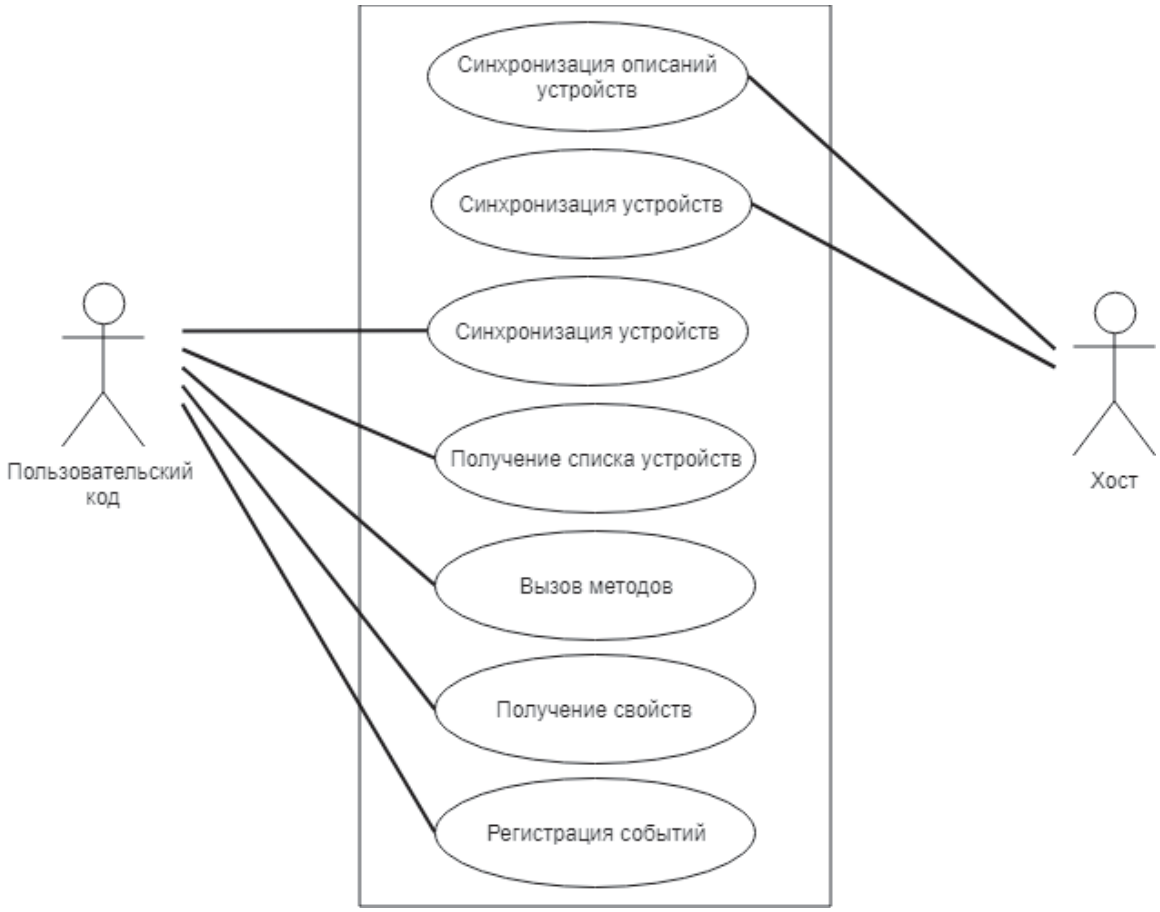

Рисунок 5 – Диаграмма вариантов использования расширяемой системы взаимодействия компонентов интернета вещей

На рисунке 5 показаны 2 ключевых действующих лица:

1. Пользовательский код – скрипт, написанный на языке C#, написанный программистом с целью автоматизации устройств интернета вещей.
2. Хост – программа, запущенная на сервере, получающая команду от пользовательского кода и управляющая устройствами с помощью плагинов.

При старте пользовательского кода, первым делом происходит синхронизация описания и данных устройств, после синхронизации пользовательский код может запрашивать устройства, имея ссылку на

устройство может просматривать его свойства, вызывать методы и подписываться на события.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Для полноценного функционирования системы, должны быть предусмотрены следующие аспекты функционирования:

1. Беспрерывная работа.
2. Возможность запуска на устройствах с низким энергопотреблением – одноплатных компьютерах.
3. Возможность работы как в локальной сети, так и в сети Интернет.

3. ПРОЕКТИРОВАНИЕ

3.1. ВЫБОР ЯЗЫКА ПРОГРАММИРОВАНИЯ

В качестве языка программирования ядра программного комплекса и языка написания пользовательских автоматизаций был выбран C#. C# — простой, современный объектно-ориентированный и типобезопасный язык программирования [10]. Вот лишь несколько функций языка C#, обеспечивающих надежность и устойчивость приложений:

1. Сборка мусора автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами.
2. Обработка исключений предоставляет структурированный и расширяемый способ выявлять и обрабатывать ошибки.
3. Строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы индексируемых массивов или выполнять неконтролируемое приведение типов.
4. Наличие большой стандартной библиотеки классов, которая помогает писать код удобнее и быстрее.
5. Встроенная в язык поддержка асинхронных вызовов.

Данный язык исполняется в среде CLR (Common Language Runtime). CLR — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования [11]. Существуют различные реализации CLR, одна из которых – Mono. Mono – представляется собой открытую реализацию системы .NET Framework, способную работать на разнообразных операционных системах. Использование

языка C# вместе с Mono [12] позволит запускать разрабатываемый программный комплекс на Windows, Linux, Mac OS. Также появляется возможность использовать комплекс на одноплатных компьютерах, например, на Raspberry Pi, что позволит снизить энергозатраты, так как программный комплекс должен постоянно находится в активном состоянии.

Использование языка C# для написания клиентской автоматизации позволяет использовать обширный набор стандартных классов языка, а также бесчисленное количество библиотек, что позволит писать скрипты любой сложности.

3.2. ВЫБОР БИБЛИОТЕКИ ДЛЯ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ

Клиентский код и основной код системы могут находится на разных машинах, поэтому необходимо сетевое взаимодействие. Для этого можно использовать стандартные сокеты системы напрямую, но в таком случае получится достаточно низкоуровневый код, и он не будет являться платформо-независимым. Поэтому необходимо выбрать библиотеку для сетевого взаимодействия, которая поддерживает различные операционные системы и платформы, а также абстрагироваться от использования низкоуровневого кода.

Было принято решение выбрать библиотеку ZeroMQ [13]. ZeroMQ предлагает разработчику некий высокий уровень абстракции при работе с «сокетами». Библиотека берет на себя по буферизацию данных, обслуживание очередей, установление и восстановление соединений. Данная библиотека поддерживает множество языков программирования и операционных систем, что позволяет использовать только одну эту библиотеку на всех платформах.

3.3. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

В Программный продукт состоит из:

1. Хост – программа предоставляющая единый интерфейс к плагинам.
2. Библиотека для взаимодействия хоста и плагина.
3. Клиентская библиотека для взаимодействия с хостом.
4. Генератор интерфейса к IoT-устройствам.
5. Плагины для конкретных IoT-устройств.

Архитектура разрабатываемого комплекса представлена на рисунке 6.

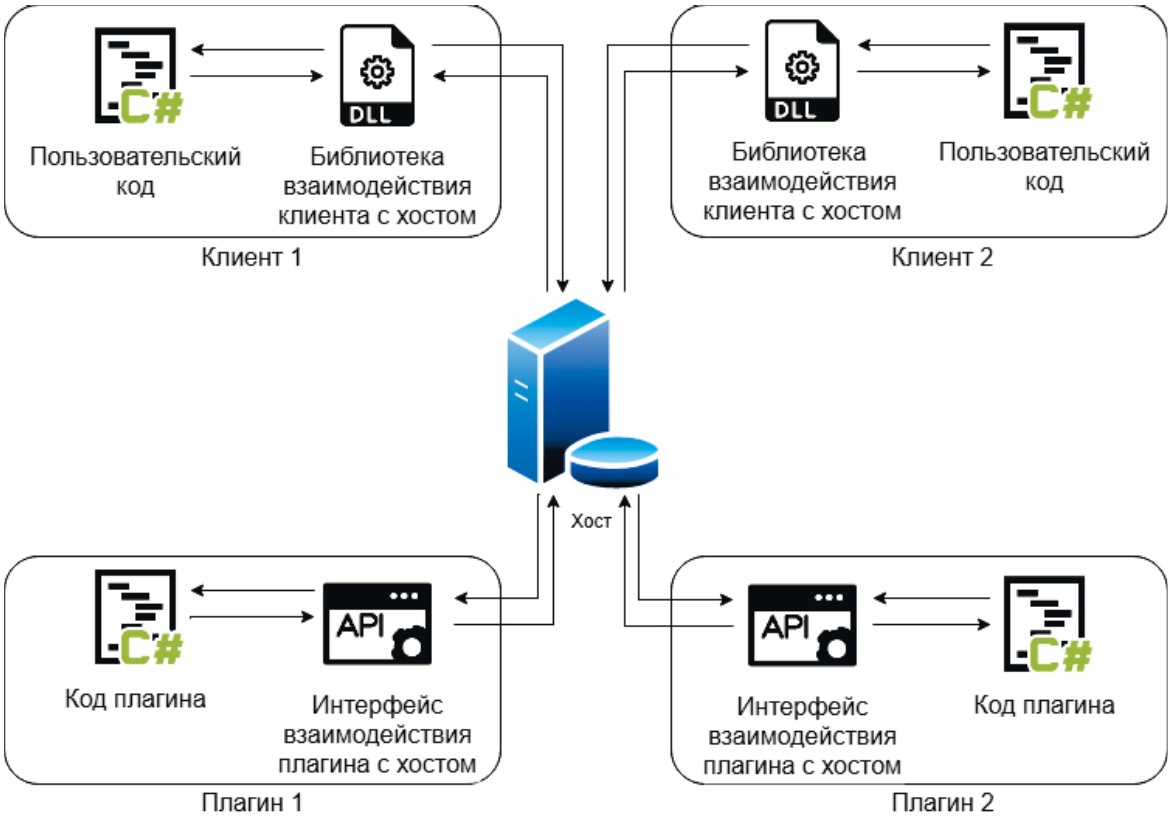

Рисунок 6 – Архитектура разрабатываемого приложения

3.4. ПРОЕКТИРОВАНИЕ ХОСТА

Для создания расширяемой системы, необходимо предусмотреть возможность расширения функционала с помощью подключения плагинов. Плагин — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей [14]. Плагины обычно выполняются в виде библиотек общего пользования.

Функционал для взаимодействия хоста и плагина необходимо выделить в отдельную библиотеку. Данная библиотека должна содержать в себе классы и методы, которые необходимы для коммуникации плагина с хостом и наоборот.

3.5. ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ БИБЛИОТЕКИ

Для получения возможности взаимодействовать с хостом, необходимо разработать библиотеку-интерфейс пользователя к хосту. После подключения библиотеки, пользователь сможет получать список подключенных устройств, вызывать методы, просматривать свойства, подписываться на события.

3.6. ПРОЕКТИРОВАНИЕ ГЕНЕРАТОРА ИНТЕРФЕЙСОВ К УСТРОЙСТВАМ

В связи с отсутствием на стороне клиента классов, описывающих устройства, все вызовы методов устройств происходят динамически, то есть названия методов передаются в виде строки, а аргументы в виде нетипизированного массива. Данные метод написания кода неудобен и приводит к ошибкам. Поэтому для удобства написания пользовательского кода будет разработан генератор интерфейса к устройствам, который позволяет

получать помощь от IDE в написании кода и устранять ошибки, связанные, например, с обращением к несуществующим устройствам или методам.

На рисунках представлены два варианта вызова метода устройства. На рисунке 7 представлен вызов метода устройства без использования генератора интерфейсов, а на рисунке 8 с использованием. Наглядно видно, что вызов с использованием генератора позволяет получать подсказки от IDE, что помогает избежать ошибок. Документация также интегрируется в IDE.

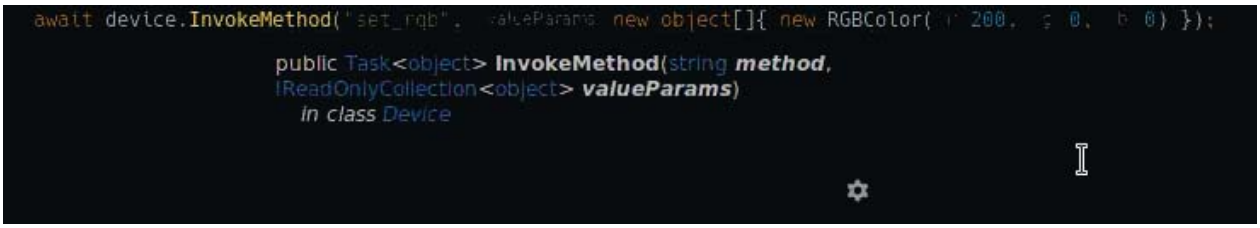

Рисунок 7 – Вызов метода устройства без использования генератора интерфейсов.

Рисунок 8 – Вызов метода устройства с использованием генератора интерфейсов

3.7. РАЗРАБОТКА ОБЩЕЙ ЧАСТИ КОДА

Код, который необходим всем проектам, содержится в сборке Common. Данная сборка содержит следующие классы:

Класс DeviceData – содержит информацию о конкретном устройстве: платформа, тип, идентификатор, имя, свойства и онлайн-статус. Данный класс имеет методы, позволяющие конвертировать JSON-представление устройства в объект класса и наоборот.

Класс DeviceDescription – содержит описание функционала устройств конкретного типа устройств. Описание устройства включает в себя список свойств, где каждое свойство имеет имя и тип значения; список событий, где каждое событие имеет имя и список именованных параметров для этого события; список методов, где каждый метод имеет имя, тип возвращаемого значения и список именованных параметров. Также данный класс имеет методы, позволяющие сериализовать и десериализовать JSON. Объекты класса DeviceDescription генерируется автоматически из классов устройств и позволяют автоматически генерировать интерфейсы к устройствам на стороне клиента.

Интерфейс IMethodInvocationInstance – содержит методы обратного вызова:

- void Completed(object result) – вызывается при успешном завершении вызова;
- void Failed(string error) – вызывается при ошибке вызова;
- void Timedout() – вызывается при истечении времени на вызов.

Класс AbstractNetworkProcessor – является базовым классов, задача которого получать и отправлять сообщения по сети. Отправка и получение

сообщений осуществляются в отдельном потоке. На рисунке 9 представлена UML-диаграмма данного класса и его наследников.

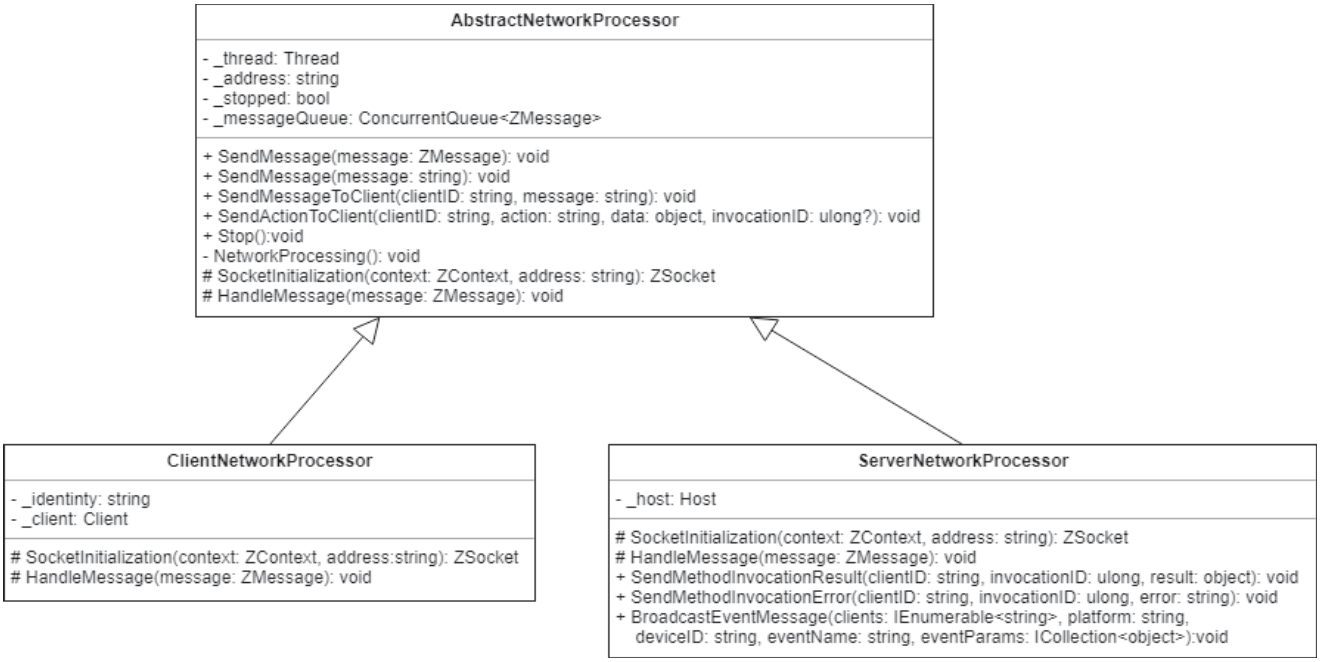

Рисунок 9 – UML-диаграмма класса AbstractNetworkProcessor и его наследников

Класс AbstractMessageProcessor – является базовым классом, задача которого заниматься обработкой полученных сообщений. Обработка сообщений осуществляется в отдельном потоке. На рисунке 10 представлена UML-диаграмма данного класса и его наследников.

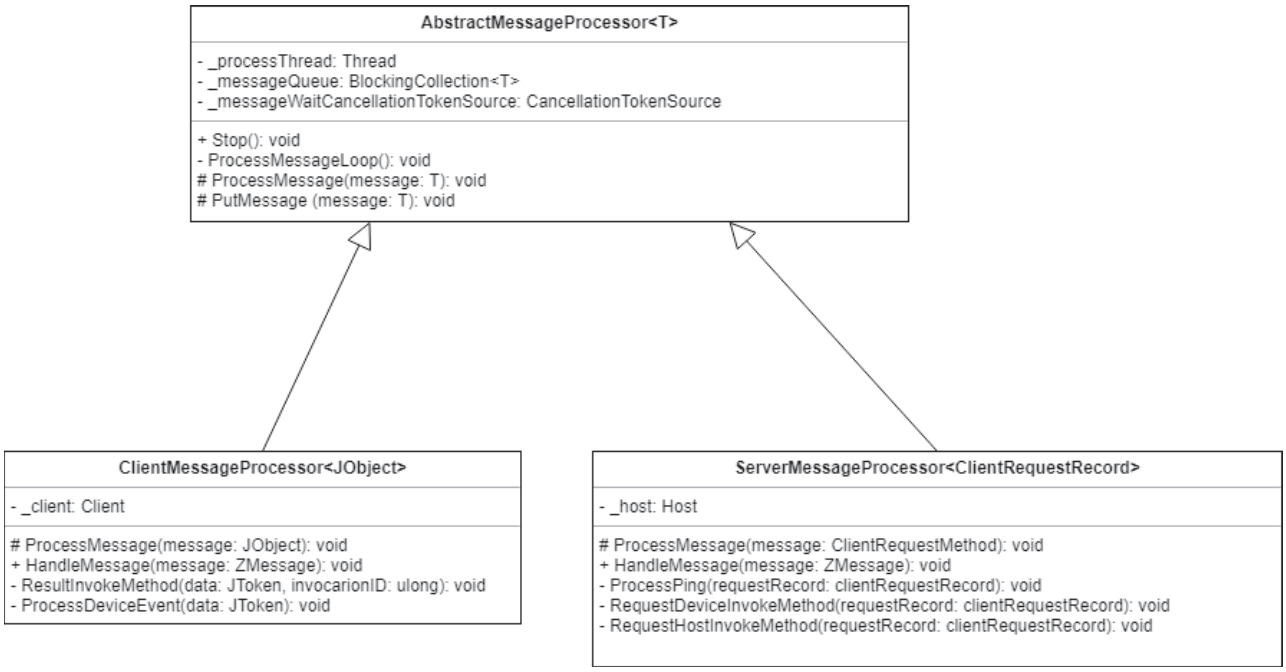

Рисунок 10 – UML-диаграмма AbstractMessageProcessor класса и его наследников

Класс BaseConverter и наследники – отвечают за сериализацию и десериализацию параметров и возвращаемых значений при вызове методов. Класс BaseConverter имеет статический конструктор, который находит все классы в программе, которые наследуются от BaseConverter и регистрируют их как доступные для преобразования типы данных. При необходимости добавления нового типа данных, нужно создать новый класс на следующий BaseConverter и реализовать его методы. На рисунке 11 представлена UML-диаграмма данного класса и его наследников.

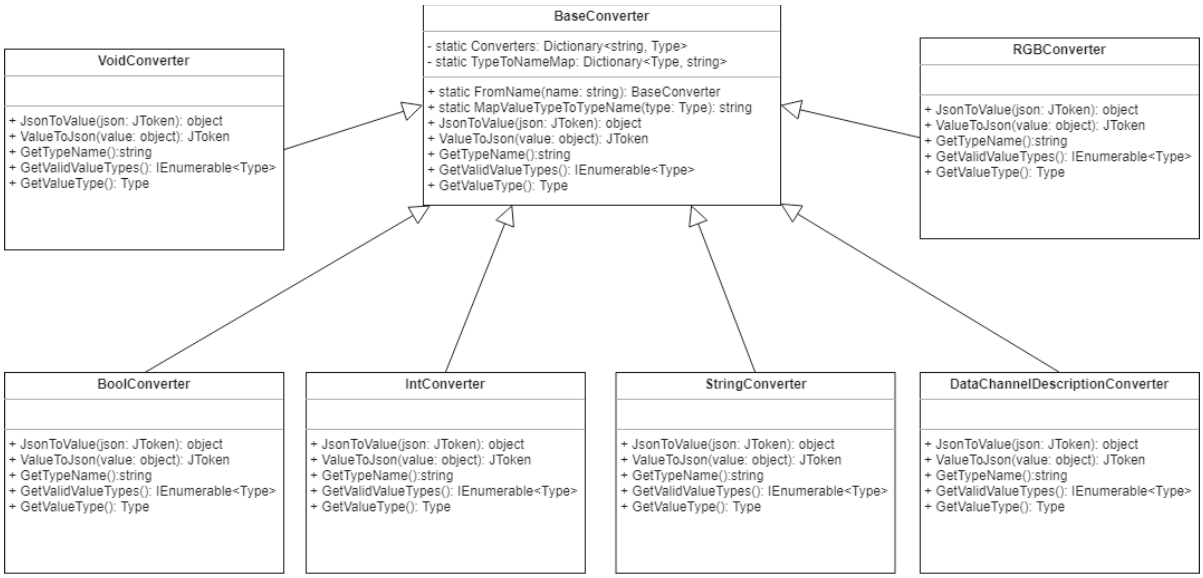

Рисунок 11 – UML-диаграмма класса BaseConverter и его наследников

3.8. ПРОЕКТИРОВАНИЕ ГЕНЕРАТОРА ИНТЕРФЕЙСОВ К УСТРОЙСТВАМ

Библиотека написана на языке C#. PlatformInterface выступает связующим звеном между плагинами и хостом, также в ней содержатся классы, которые необходимы всем плагинам. Данная сборка содержит два основных интерфейса:

Интерфейс IPlatformFactory – инициализирует плагины, содержит в себе два метода:

– string GetPlatformName() – возвращает название платформы;
– public IPlatform Create(out string error, Dictionary<string, string> options) – инициализирует платформу с предоставленными параметрами и, в случае невозможности создания платформы, возвращает сообщение об ошибке.

Интерфейс IPlatform – управляет плагином и его устройствами, содержит в себе следующие основные методы:

– "void StartPlatform(IHost host)" – запускает платформу и передает ссылке на объект хоста;

– "ICollection<DeviceDescription> GetDeviceDescriptions()" – возвращает коллекцию с описанием всех поддерживаемых устройств;
– "ICollection<DeviceData> GetDevices()" – возвращает коллекцию с информацией о подключенных устройствах;
– "void InvokeDeviceMethod (string deviceID, string method, IReadOnlyCollection<JToken> serializedMethodParams, IMethodInvocationInstance invocationInstance)" – вызов методов устройств.

Абстрактный класс AbstractDevice служит предком для всех классов, описывающих устройства. Данный класс содержит в себе ссылки на объекты классов DeviceDescription и DeviceData и абстрактный метод void InvokeMethod (string method, IReadOnlyCollection<JToken> serializedMethodParams, IMethodInvocationInstance invocationInstance), который вызывает методы у устройств.

Класс BaseDevice является потомком класса AbstractDevice и реализует его метод InvokeMethod. Данная реализация вызывает необходимый метод с помощью интроспекции. При реализации устройств в плагине на C#, необходимо наследоваться именно от этого класса.

Также в данной сборке содержатся классы-атрибуты: DeviceType, DeviceMethod, DeviceProp, DeviceEvent, Doc. Данные классы требуются для автоматического генерирования описания устройств. Генерация устройств осуществляется с помощью класса DeviceDescriptionGenetator, который исследует классы устройств и их содержимое и генерирует объекты класса DeviceDescription.

Листинг 1 – Пример класса устройства с использованием атрибутов для автоматической генерации описания

```
[DeviceType("dummy", "Dummy")]
[Doc("Generated Class docs")]
public class DummyDevice : BaseDevice
{
	[DeviceEvent(Name = "event0")]
	[Doc("Event docs")]
	public delegate void DeviceEvent0Delegate();

	[DeviceMethod(Name = "string_method", ResultType = "string")]
	[Doc("This method returns upper-case variant of string")]
	[return: Doc("Upper-case variant of input string")]
	private string StringMethod([Doc("Input string")] string s)
	{
		return s.ToUpper();
	}
}
```

3.9. РАЗРАБОТКА ХОСТА

Задачей хоста является управление плагинами и работа с клиентами. Хост парсит переданные ему параметры командной строки, из которых получает параметры для настройки плагинов. Далее, сканируется директория с плагинами, где каждый плагин представляет из себя сборку.

Процесс загрузки плагина:

1. В сборке находятся все классы, реализующие интерфейс IPlatfomFactory.
2. Создается объект класса IPlarformFactory.
3. У объекта IPlatformFactory вызывается метод Create, и в него передается параметры для инициализации плагинов. При успешной инициализации плагина, метод Create возвращает объект, реализующий интерфейс IPlatform.
4. У объекта IPlatform вызывается метод StartPlatform и в него передается ссылка на объект, реализующий интерфейс IHost.
5. После загрузки плагинов, хост начинает слушать сообщения от клиентов, и передавать эти сообщения соответствующим плагинам.

Листинг 2 – Функции загрузки плагинов

```
private void LoadPlatforms()
{
    var platformDir = Path.Combine(Directory.GetCurrentDirectory(),
"platforms");
    foreach (var platformAssemblyDir in Directory.GetDirectories(platformDir))
    {
        var platformAssemblyName = Path.GetFileName(platformAssemblyDir);
        if(platformAssemblyName == null)
            continue;

        var platformAssemblyFile = Path.Combine(platformAssemblyDir,
platformAssemblyName);
        Logger.Log(platformAssemblyFile);
        var assembly = Assembly.LoadFrom(platformAssemblyFile);
        var platforms = LoadPlatformsFromAssembly(assembly);

        foreach (var platform in platforms)
        {
            platform.StartPlatform(this);
            Logger.Log($"loaded {platform.GetPlatformName()}");
            _loadedPlatforms.Add(platform.GetPlatformName(), platform);
        }
    }
}

private IEnumerable<IPlatform> LoadPlatformsFromAssembly(Assembly assembly)
{
    var platforms = new List<IPlatform>();
    var definedFactoryList =
        assembly.GetTypes().Where(type =>
type.GetInterfaces().Contains(typeof(IPlatformFactory)));
    foreach (var factoryType in definedFactoryList)
    {
        var factory = (IPlatformFactory)Activator.CreateInstance(factoryType);

        if(!_platformOptions.TryGetValue(factory.GetPlatformName(), out var
options))
            options = new Dictionary<string, string>();

        var platform = factory.Create(out var error, options);
        if (platform == null)
        {
            Logger.Log($"error while instancing platform
'{factory.GetPlatformName()}': {error}");
            continue;
        }

        platforms.Add(platform);
    }

    return platforms;
```

3.10. РАЗРАБОТКА КЛИЕНТСКОЙ БИБЛИОТЕКИ

Основными классами клиентской библиотеки являются классы Client и Device. Класс Client позволяет связываться с хостом, запрашивать список подключенных устройств и находить нужные устройства по его идентификатору. Конкретные устройства представляются в виде объекта класса Device. Объекты данного класса позволяют просматривать информацию об устройстве: статус, свойства; а также вызывать методы и подписываться на события. Все вызовы происходят асинхронно. Процесс вызова метода IoT-устройства представлен на рисунке 12.

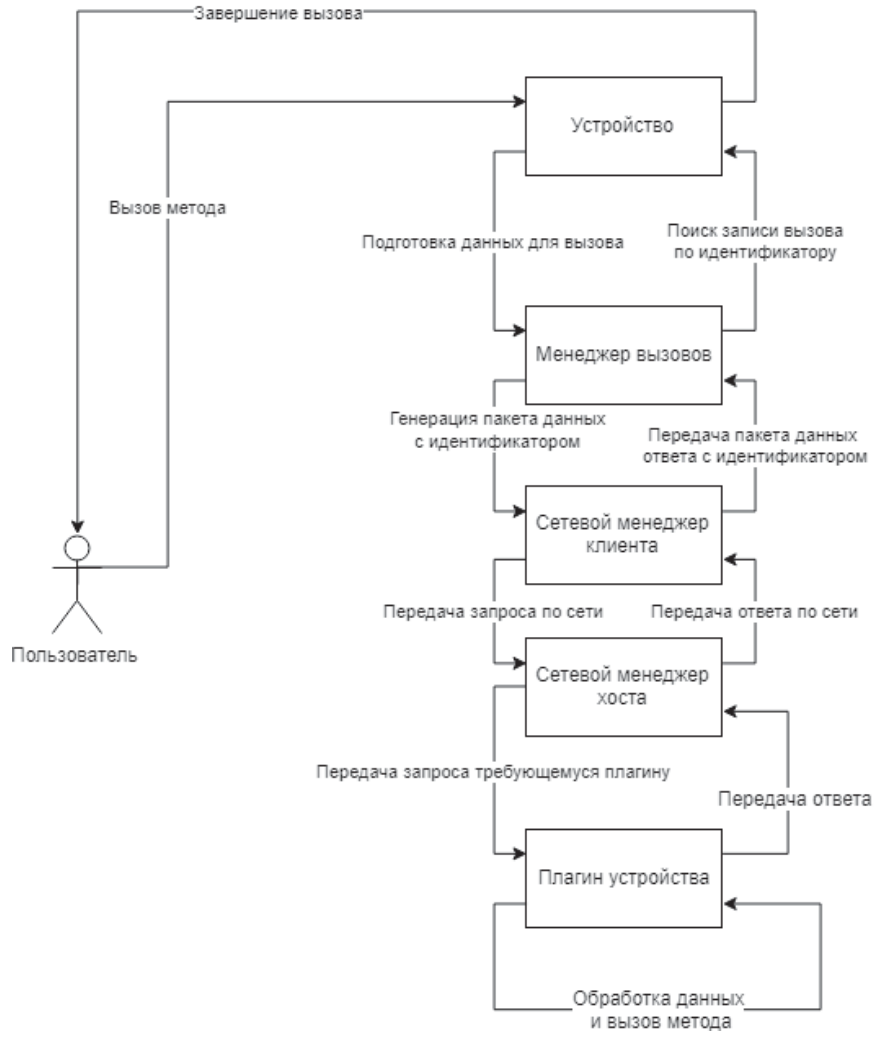

Рисунок 12 – Процесс вызова метода IoT-устройства

На макете изображено главное окно приложения, окно списка друзей (6), окно входа и регистрации (7). На главном окне присутствуют следующие элементы:

1. Список файлов на своем устройстве.
2. Список файлов на удаленном устройстве.
3. Текущий путь и список точек входа на своем устройстве.
4. Текущий путь и список точек входа на удаленном устройстве.
5. Список устройств.

3.11. РАЗРАБОТКА ГЕНЕРАТОРА ИНТЕРФЕЙСОВ

Генерация классов для устройств осуществляется с помощью компонента CodeDom. Данная технология позволяет описывать модель кода в программе в виде графа, а затем превратить это представление кода в исходный код на языке C# или скомпилировать его в сборку.

Для каждой платформы генерируется пространство имен, в которое заключаются классы устройств, принадлежащие данной платформе. Генерация содержимого класса происходит на основании описания устройств из класса DeviceDescription. Для каждого класса генерируется свойства, методы и события. В листинге 3 представлен код, который автоматический сгенерирован на основании описания устройства. Генерация метода заключается в создании метода класса с необходимыми аргументами и типом возвращаемого значения. А аргументы метода собираются в массив, затем передаются динамическому вызову вместе со строковой константой, описывающей название метода. Генерация события состоит из: генерации делегата события, объявления

события (с помощью ключевого слова «event»), генерации прокси-метода и регистрации данного метода у диспетчера событий.

Листинг 3 – автоматически сгенерированный код

```
namespace IOT.Devices.Dummy
{
    using System;
    using System.Threading.Tasks;
    using IOT.Common;
    using IOT.Client;

    public class Dummy
    {
        private IOT.Client.Device _device_controller;

        public Dummy(IOT.Client.Device device)
        {
            this._device_controller = device;
            this._device_controller.SubscribeEvent("event0",
this.@__event_proxy_Event0);
        }

        public event Event0EventDelegate Event0;

        public async Task<System.String> StringMethod(string s)
        {
            object[] @__collected_params_array__ = new object[] {
                    s};
            return ((string)(await
_device_controller.InvokeMethod("string_method", @__collected_params_array__)));
        }

        private void @__event_proxy_Event0(object[] eventParams)
        {
            this.Event0();
        }

        public delegate void Event0EventDelegate();
    }
}
```

3.12. РАЗРАБОТКА ГЕНЕРАТОРА ИНТЕРФЕЙСОВ

Структура проектов расширяемой системы взаимодействия компонентов интернета вещей представлена рисунке 13. Так как в Common содержатся общие классы, то он подключается как зависимость ко всем остальным

проектам. PlatformInterface является связующим звеном между хостом и плагинами, соответственно он подключается как зависимости к хосту и плагину. Для получения описания устройств, DeviceBindingsGenerator использует функционал, реализованный в библиотеке Client.

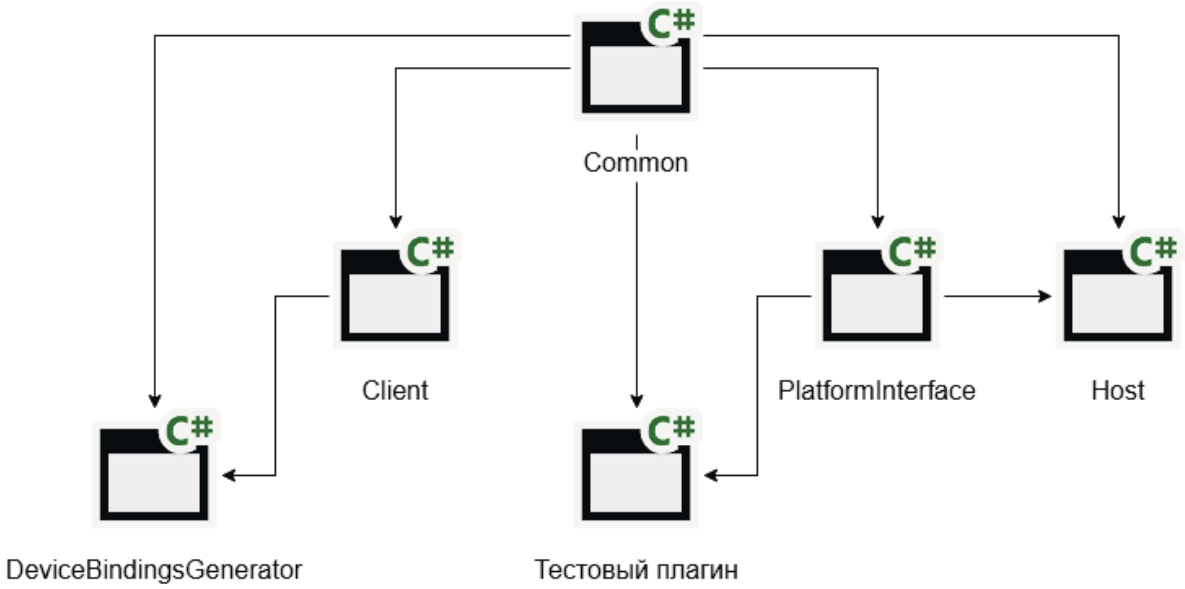

Рисунок 13 – Структура проектов расширяемой системы взаимодействия компонентов интернета вещей

4. РЕАЛИЗАЦИЯ

После сборки проекта пользователь может установить хост на сервер, установить требуемые плагины с устройствами и запустить систему. Пользователь может написать код для взаимодействия устройств с использование клиентской библиотеки. После компиляции пользовательского кода и его запуска, начинается автоматизированное управление устройствами.

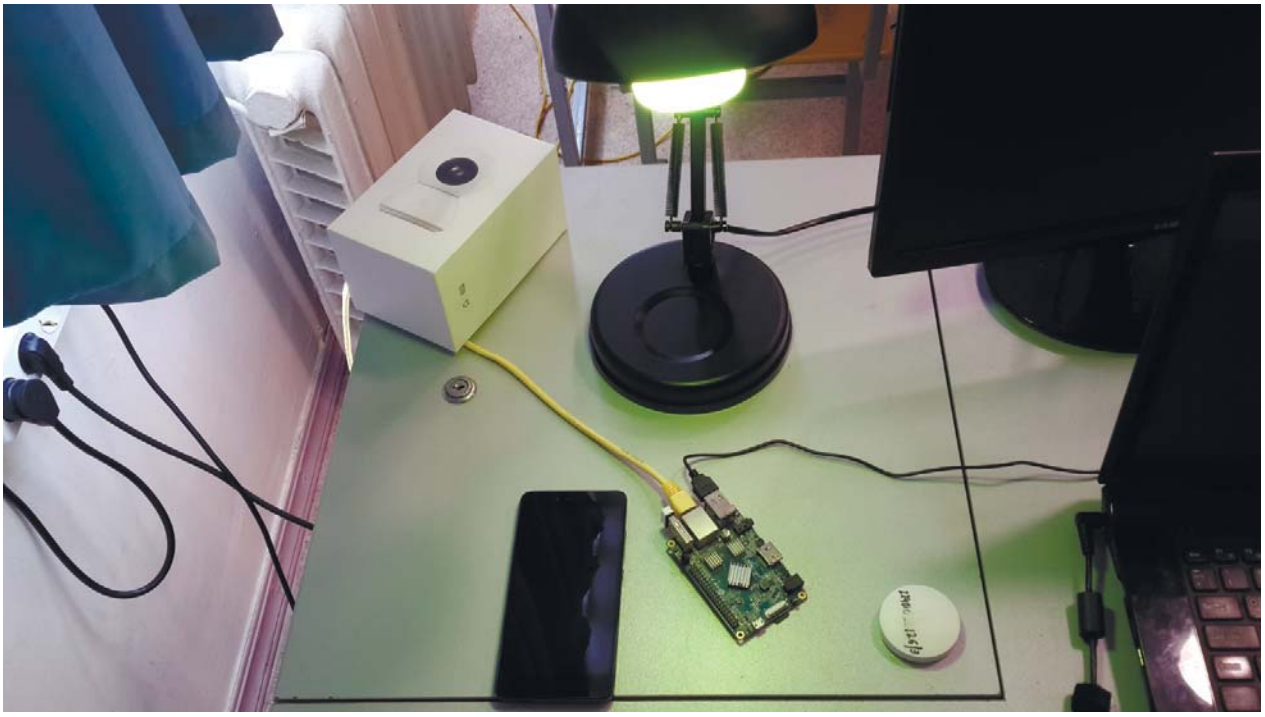

Рисунок 14 – Автоматизированное управление лампой

В качестве примера работы клиентской библиотеки был написан скрипт, подключающийся к хосту и выводящий информацию об устройствах. Выводится такая информация как платформа устройства, тип, идентификатор, имя и онлайн статус. Скриншот показывающий данную информацию представлен на рисунке 15.

```
→  Debug mono Client.exe

'Dummy':'dummy' id:'dev0' name:'Dummy device 0' online:True
'Mihome':'lumi.sensor_smoke.v1' id:'lumi.158d0001fd3b19' name:'lumi.158d0001fd3b19' online:False
'Mihome':'lumi.curtain.v1' id:'lumi.158d0002831730' name:'Aqara Curtain Controller' online:False
'Mihome':'lumi.sensor_switch.v2' id:'lumi.158d0001f3d756' name:'lumi.158d0001f3d756' online:False
'Mihome':'lumi.sensor_cube.aqgl01' id:'lumi.158d000278f002' name:'lumi.158d000278f002' online:False
'Mihome':'yeelink.light.color1' id:'86796240' name:'Yeelight Color Bulb' online:False
'Mihome':'yeelink.light.color1' id:'86779393' name:'Yeelight Color Bulb2' online:False
'Mihome':'zhimi.fan.za1' id:'90964799' name:'Smartmi Inverter Pedestal Fan' online:False
'Mihome':'lumi.curtain.v1' id:'lumi.158d0002883b16' name:'Aqara Curtain Controller' online:False
'Mihome':'lumi.curtain.v1' id:'lumi.158d0002833266' name:'Aqara Curtain Controller' online:False
'Mihome':'lumi.curtain.v1' id:'lumi.158d0002831595' name:'Aqara Curtain Controller' online:False
'Mihome':'chuangmi.camera.xiaobai' id:'79843165' name:'Mi 360 Webcam' online:False
```

Рисунок 15 – Информация об устройствах, полученная с помощью скрипта

Для работы с хостом требуется, чтобы он был постоянно запущен. Хост является кроссплатформенным, а также может быть установлен на одноплатные компьютеры. Данная особенность позволяет держать хост постоянно включенным, а также существенно экономить энергию.

Рисунок 16 – Хост, установленный на одноплатный компьютер Orange Pi

5. ТЕСТИРОВАНИЕ

5.1. МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ

Система успешно проходит юнит-тесты – проверены на корректность отдельные модули исходного кода программы. Разработанный комплекс успешно прошел тестирование на IoT-устройствах кафедры ЭВМ. Было проведено функциональное тестирование.

5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Тестирование разработанного программного комплекса проводилось на системе Linux.

В таблице 1 описано тестирование действия «Установка плагина».

Таблица 1 – Установка плагина.

Свойство	Значение
Выполняемые действия	Помещение сборки плагина в директорию plugins в папке Хоста.
Ожидаемые результаты	При включении системы, должно произойти обнаружение и инициализация плагина. Плагин должен начать функционировать.
Полученные результаты	После включения системы, плагин обнаруживается. Инициализация проходит успешно – плагин начинает работать.

В таблице 2 описано тестирование проверки связи с клиентом.

Таблица 2 – Проверка связи между клиентом и хостом.

Свойство	Значение
Выполняемые действия	Создан проект на C#, подключена клиентская библиотека. Написан код, запрашивающий с хоста список подключенных устройств и выводящий их на экран.
Ожидаемые результаты	Успешное установка связи. Хост передает информацию, информация отображается на экране.
Полученные результаты	После запуска написанной программы, происходит запрос списка подключенных устройств. Список устройств отображается на экране.

В таблице 3 описано тестирование вызова метода устройства.

Таблица 3 – Тест вызова метода устройства.

Свойство	Значение
Выполняемые действия	Написан пользовательский скрипт, который вызывает метод включения лампочки.
Ожидаемые результаты	Пользовательский скрипт управляет включением лампы – происходит включение.
Полученные результаты	После запуска пользовательской программы, лампочка включается.

В таблице 4 описано тестирование событий устройств.

Таблица 4 – Тест событий устройств.

Свойство	Значение
Выполняемые действия	Написан пользовательский скрипт, который связывает сигнал нажатия кнопки и включение лампочки.
Ожидаемые результаты	Пользовательский скрипт связывает действия IoT-устройств. При нажатии на кнопку, лампочка загорается.
Полученные результаты	Пользовательский код запущен. После нажатия на кнопку, лампочка загорается.

Как видно из тестов, разработанная система работает корректно и позволяет пользователю взаимодействовать с IoT-устройствами различных производителей.

6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данный программный комплекс предназначен для создания автоматизированных систем управления IoT-устройствами.

Системные требования для клиента/хоста:

- операционная система: Linux, Windows, MacOS;
- среда исполнения Mono;
- наличие сетевого подключения до хоста;
- оперативная память: 512 МБ.

6.1. НАСТРОЙКИ ХОСТА

1. Установить среду исполнения Mono.

2. Поместить файлы хоста в директорию на компьютере.
3. Установить необходимые плагины:
    a. Создать папку “plugins” в директории хоста.
    b. Для каждого плагина создать папку в папке “plugins”.
    c. Поместить файлы плагина в соответствующую папку.
4. Запуск хоста с передачей параметров плагинов.
    a. Сформировать список параметров для плагинов.
    b. Формат параметров имеет вид: “название_платформы:название_параметра:значение” (например, mihome:app_ip:192.168.0.10).
    c. Запустить хост с помощью команды: mono Host.exe “парметры” (например, mono Host.exe mihome:app_ip:192.168.0.10 mihome:app_port:6666).

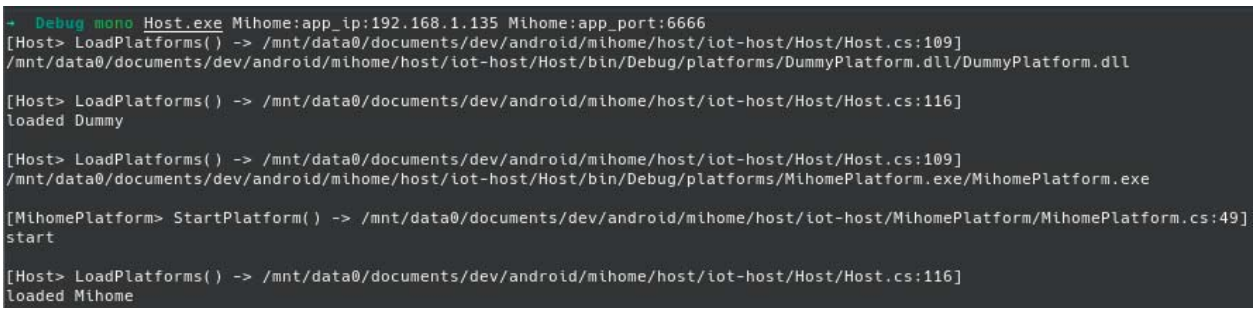

Рисунок 17 – Скриншот запущенного хоста

6.2. ИСПОЛЬЗОВАНИЕ КЛИЕНТСКОЙ БИБЛИОТЕКИ

Клиентская библиотека необходима, чтобы взаимодействовать с программным комплексом.

Процесс подготовки к написанию скрипта для управления IoT-устройства следующий:

1. Создать проект на языке C#.
2. Подключить сборку Client.dll к проекту как зависимость.
3. ОПЦИОНАЛЬНО. Запустить программу DeviceBindingsGenerator.exe для генерации интерфейсов к устройствам.
    - Данная программа сгенерирует сборку IOTDevices.dll, которую необходимо также подключить к проекту.
4. В коде создать экземпляр класса Client, передав ему IP, пароль хоста и идентификатор пользователя.
    - Client client = new Client("196.168.0.10", 6667, "id").
5. Вызвать метод Sync() у объекта Client для синхронизации устройств с хостом.
    - await client.Sync().
6. Получить ссылку на устройство с помощью метода GetDevice().
    - Device device = await client.GetDevice("platform_name", "device_id").
7. Вызвать метод для устройства возможно двумя вариантами:
    - 1 вариант. Напрямую через объект Device (Например, await device.InvokeMethod("set_rgb", new object[]{ new RGBColor(200, 0, 0) }).
    - 2 вариант. С помощью классов, полученных от генератора интерфейсов. (Например, YeelinkLightColor1 lamp = new YeelinkLightColor1(device);
      await lamp.SetRgb(new RGBColor(200, 0, 0));).
8. Получить доступ к свойству устройства возможно двумя вариантами:
    - 1 вариант. Напрямую через объект Device. (Например, RGBColor color = await device.GetProp("rgb");).

- 2 вариант. С помощью классов, полученных от генератора интерфейсов. (Например, YeelinkLightColor1 lamp = new YeelinkLightColor1(device);
  RGBColor color = await lamp.RGB;)

9. Подписаться на события устройств можно двумя вариантами:
   - 1 вариант. Напрямую через объект Device. (Например, device.SubscribeEvent("on_color_change", callback);).
   - 2 вариант. С помощью классов, полученных от генератора интерфейсов. (Например, YeelinkLightColor1 lamp = new YeelinkLightColor1(device);
     lamp.OnColorChange += callback;).

6.3. РУКОВОДСТВО ПО НАПИСАНИЮ ПЛАГИНА

Для добавления нового устройства в расширяемую систему, необходимо реализовать плагин для этого устройства. Каждый плагин представляет из себя сборку, написанную на языке C#.

Процесс написания плагина:

1. Создать новый проект на языке C#.
2. Подключить сборки Common.dll и PlatformInterface.dll к проекту как зависимость.
3. Необходимо создать класс, реализующий интерфейс IPlatform – данный класс будет отвечать за управление плагином. Требуется реализовать следующие методы:
   a. void StartPlatform(IHost host) – данный метод отвечает за запуск платформы. Плагин должен сохранить Объект host, чтобы иметь возможность взаимодействовать с хостом.

b. void StopPlatform() – данный метод отвечает за остановку платформы.
c. string GetPlatformName() – данный метод должен возвращать идентификатор платформы. Идентификатор будет использоваться клиентами для поиска устройств.
d. ICollection<DeviceDescription> GetDeviceDescriptions() – данный метод должен вернуть коллекцию с описаниями всех устройств, поддерживаемых данной платформой. Для генерации описания устройства следует использовать метод DeviceDescription GenerateDescription(Type deviceType, string platform) класса DeviceDescriptionGenerator.
e. ICollection<DeviceData> GetDevices() – данный метод должен вернуть коллекцию всех устройств, которые в данный момент зарегистрированы в платформе.
f. void InvokeDeviceMethod(string deviceID, string method, IReadOnlyCollection<JToken> serializedMethodParams, IMethodInvocationInstance invocationInstance) – данный метод вызывается хостом, когда происходит запрос на вызов метода устройства у данной платформы. В общем случае данному методу необходимо найти среди своих устройств устройство с заданным идентификатором и передать ему запрос вызова (вызвав метод устройства InvokeMethod).

4. Необходимо создать класс, реализующий интерефейс IPlatformFactory, который отвечает за инициализацию платформы. В нем должны быть реализованы следующие методы:
   a. string GetPlatformName() – данный метод должен возвращать идентификатор платформы. Возвращаемое значение этого метода

должно совпадать с возвращаемым значением аналогичного метода у класса, реализующего интерфейс IPlatform.

b. IPlatform Create(out string error, Dictionary<string, string> options) – данный метод инициализирует платформу и возвращает ссылку на класс, реализующий интерфейс IPlatform. В этот метод передается набор параметров, переданных пользователям для данной платформы, которые могут быть использованы при инициализации. В случае ошибки во время инициализации, метод должен вернуть значение null, а в выходной параметр error должна быть записана строка с сообщение об ошибке.

5. В случае, если реализация логики устройств должна находится не в плагине (например, на микропроцессоре устройства), следует создавать классы устройств, наследуясь от класса AbstractDevice. Необходимо реализовать метод InvokeMethod() и в нем транслировать вызов к фактической реализации логики устройства.
6. Во всех остальных случаях, когда реализация логики устройства находится в самом плагине, следует созвать классы устройств, наследуясь от класса BaseDevice.
    a. Для объявления свойства устройства, необходимо объявить в классе поле или свойство и пометить его атрибутом DeviceProp.

    ```
    [DeviceProp(Name ="prop_name")]
    public int prop;
    ```

    b. Для объявления события устройства, необходимо объявить в классе делегат и событие и пометить событие атрибутом DeviceEvent.

    ```
    public delegate void EventDelegate();
    [DeviceEvent(Name ="event_name")]
    public event EventDelegate Event;
    ```

c. Для объявления метода устройства необходимо объявить в классе метод и пометить его атрибутом DeviceMethod.

```
[DeviceMethod(Name = "method_name")]
public void Method(string arg0, int arg1) {/*Реализация метода*/}
```

d. Следует использовать атрибут Doc для внедрения документации в описание устройств. Этот атрибут можно применять к классу, полю, свойству, событию, параметру события, методу, параметру метода и к возвращаемому значению метода.

```
[DeviceMethod(Name = "string_method")]
[Doc("This method returns upper-case variant of string")]
[return: Doc("Upper-case variant of input string")]
private string StringMethod([Doc("Input string")] string s)
{
return s.ToUpper();
}
```

7. ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был проведен анализ существующих программных комплексов для управления IoT-устройствами: выделены их недостатки и предложено усовершенствованное решение. Был спроектирован и реализован расширяемый программный комплекс для управления IoT-устройствами различных производителей. В отличии от аналогичных систем, разработанная система поддерживает написание скриптов на языке общего назначения, что позволяет создавать неограниченно сложные сценарии взаимодействия. Данный программный комплекс можно использовать для обучения студентов работе с современными IoT-технологиями, проводить лабораторные работы. Позволяет писать, как скрипты автоматизации существующих устройств, так и реализовать поддержку для новых устройств. Также расширяемую систему взаимодействия IoT-устройств можно использовать для постройки «умного дома», возможно использование в коммерческих целях. Разработанный комплекс включает в себя: две программы и две библиотеки.

По итогам работы, можно сделать следующие выводы:

- получена расширяемая система управления IoT-устройствами;
- создана база для расширения системы новыми компонентами;
- реализована клиентская библиотека взаимодействия с системой;
- реализован генератор интерфейсов для удобного с взаимодействия с устройством;
- разработанный комплекс будет предложен для внедрения на кафедре электронных вычислительных машин ФГАОУ ВО «ЮУрГУ (НИУ)».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Интернет вещей: прогнозы по развитию рынка. – https://www.likeni.ru/analytics/internet-veshchey-prognozy-po-razvitiyu-rynka. Дата обращения: 10.02.2019.
2. Xiaomi Ecosystem. – https://recon.cx/2018/brussels/resources/slides/RECON-BRX-2018-Reversing-IoT-Xiaomi-ecosystem.pdf. Дата обращения: 24.02.2019.
3. Chou, T. Precision: Principles, Practices and Solutions for the Internet of Things / T. Chou. – Роли: Lulu, 2016. – 314 с.
4. Что такое IoT, или интернет вещей. – https://coinspot.io/beginners/chto-akoe-iot-ili-internet-veshhej. Дата обращения: 10.12.2018.
5. Интернет-вещей. – https://habr.com/ru/post/149593/. Дата обращения: 20.12.2018.
6. Domoticz. – https:// domoticz.com. Дата обращения: 10.02.2019.
7. Home Assistant. – https://www.home-assistant.io. Дата обращения: 15.02.2019.
8. MajorDoMo – Умный дом своими руками. – majordomo.smartliving.ru. Дата обращения: 17.02.2019.
9. openHAB. – https://www.openhab.org. Дата обращения: 19.02.2019.
10. Скит, Д. C#. Программирование для профессионалов / Д. Скит. – М.: Вильямс, 2011. – 544 с.;
11. Обзор среды CLR – https://docs.microsoft.com/ru-ru/dotnet/standard/clr. Дата обращения: 01.03.2019.
12. Mono – https://www.mono-project.com. Дата обращения: 10.03.2019.
13. Akgul, F. ZeroMQ / F. Akgul. – Бирмингем: Packt Publishing, 2011. – 140 с.
14. Плагины | MDN – https://developer.mozilla.org/ru/docs/Plugins. Дата обращения: 20.02.2018.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ХОСТА

Файл ClientManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Timers;
using IOT.Common;
using IOTHost;

namespace Host
{
    public class ClientManager : IDisposable
    {
        private readonly Dictionary<string, Client> _clients = new Dictionary<string,
Client>();
        private readonly Timer _checkActiveClientsTimer = new Timer(2000);


        public class Client
        {
            public readonly string ID;
            public DateTime LastPingTime;

            public Client(string id)
            {
                ID = id;
                LastPingTime = DateTime.UtcNow;
            }
        }

        public ClientManager()
        {
            _checkActiveClientsTimer.Elapsed += OnCheckActiveClientsTimerEvent;
            _checkActiveClientsTimer.AutoReset = true;
            _checkActiveClientsTimer.Enabled = true;
        }

        public void Dispose()
        {
            _checkActiveClientsTimer.Dispose();
        }

        public IEnumerable<string> GetClientIDs()
        {
            return _clients.Keys;
        }

        public void ProcessClientPing(string id)
```

```
        {
            if (_clients.TryGetValue(id, out var client))
                client.LastPingTime = DateTime.UtcNow;
            else
            {
                _clients.Add(id, new Client(id));
                Logger.Log($"new client {id}");
            }
        }

        private void OnCheckActiveClientsTimerEvent(object source, ElapsedEventArgs e)
        {
            var forRemoval = new List<string>();

            foreach (var client in _clients.Values)
            {
                if(DateTime.UtcNow - client.LastPingTime >= TimeSpan.FromSeconds(5))
                    forRemoval.Add(client.ID);
            }

            foreach (var id in forRemoval)
            {
                _clients.Remove(id);
                Logger.Log($"client {id} removed");
            }
        }
    }
}
```

Файл ClientRequestRecord.cs

```
using System;
using Newtonsoft.Json.Linq;

namespace Host
{
    public class ClientRequestRecord
    {
        public readonly string ClientIdentity;
        public readonly string Action;
        public readonly ulong? InvocationID;
        public readonly JToken Data;

        public    ClientRequestRecord(string    clientIdentity,    string    action,    ulong?
invocationID, JToken data)
        {
            ClientIdentity = clientIdentity;
            Action = action;
            InvocationID = invocationID;
            Data = data;
        }
    }
```

Файл Host.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Timers;
using IOT.Common;
using IOTHost;
using IOTHost.PlatformInterface;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace Host
{
    public class Host : IDisposable, IHost
    {
        private const string TAG = "Host";
        private readonly IDictionary<string, IPlatform> _loadedPlatforms = new
Dictionary<string, IPlatform>();
        internal readonly ServerNetworkProcessor ServerNetworkProcessor;
        internal readonly ServerMessageProcessor ServerMessageProcessor;
        internal readonly HostMethods HostMethods;
        internal readonly ClientManager ClientManager = new ClientManager();
        private readonly Dictionary<string, Dictionary<string, string>>
_platformOptions;

        public static void Main(string[] args)
        {
            var host = new Host(args);
            host.LoadPlatforms();
        }

        public Host(string[] args)
        {
            _platformOptions = ParseCommandArguments(args);
            ServerNetworkProcessor = new ServerNetworkProcessor(this, "tcp://*:6667");
            ServerMessageProcessor = new ServerMessageProcessor(this);
            HostMethods = new HostMethods(this);
        }

        public void Start()
        {
            LoadPlatforms();
        }

        public void Dispose()
        {
            ClientManager.Dispose();
            ServerMessageProcessor.Stop();
            ServerNetworkProcessor.Stop();

            foreach (var platform in _loadedPlatforms.Values)
```

```
            {
                platform.Dispose();
            }
        }

        public void ProcessDeviceEvent(string platform, string deviceID, string
eventName, ICollection<object> eventParams)
        {
            ServerNetworkProcessor.BroadcastEventMessage(ClientManager.GetClientIDs(),
platform, deviceID, eventName, eventParams);
        }

        private Dictionary<string, Dictionary<string, string>>
ParseCommandArguments(string[] argStrings)
        {
            var options = new Dictionary<string, Dictionary<string, string>>();

            foreach (var argString in argStrings)
            {
                var firstColonIndex = argString.IndexOf(':');
                if (firstColonIndex == -1)
                    goto printError;

                var secondColonIndex = argString.IndexOf(':', firstColonIndex + 1);
                if(secondColonIndex == -1)
                    goto printError;

                var platform = argString.Substring(0, firstColonIndex);
                var parameter = argString.Substring(firstColonIndex + 1,
secondColonIndex - firstColonIndex - 1);
                var value = argString.Substring(secondColonIndex + 1);

                if(!options.ContainsKey(platform))
                    options.Add(platform, new Dictionary<string, string>());
                if(!options[platform].ContainsKey(parameter))
                    options[platform].Add(parameter, value);
                else
                    Logger.Log($"option {platform}:{parameter} already present,
skipping");

                continue;

                printError:
                Logger.Log($"malformed option: {argString}. format is:
'platform':'option':'value'");
            }

            return options;
        }

        private void LoadPlatforms()
        {
            var platformDir = Path.Combine(Directory.GetCurrentDirectory(),
"platforms");
            foreach (var platformAssemblyDir in Directory.GetDirectories(platformDir))
```

```
            {
                var platformAssemblyName = Path.GetFileName(platformAssemblyDir);
                if(platformAssemblyName == null)
                    continue;

                var     platformAssemblyFile     =     Path.Combine(platformAssemblyDir,
platformAssemblyName);
                Logger.Log(platformAssemblyFile);
                var assembly = Assembly.LoadFrom(platformAssemblyFile);
                var platforms = LoadPlatformsFromAssembly(assembly);

                foreach (var platform in platforms)
                {
                    platform.StartPlatform(this);
                    Logger.Log($"loaded {platform.GetPlatformName()}");
                    _loadedPlatforms.Add(platform.GetPlatformName(), platform);
                }
            }
        }

        private IEnumerable<IPlatform> LoadPlatformsFromAssembly(Assembly assembly)
        {
            var platforms = new List<IPlatform>();
            var definedFactoryList =
                assembly.GetTypes().Where(type                                          =>
type.GetInterfaces().Contains(typeof(IPlatformFactory)));
            foreach (var factoryType in definedFactoryList)
            {
                var factory = (IPlatformFactory)Activator.CreateInstance(factoryType);

                if(!_platformOptions.TryGetValue(factory.GetPlatformName(),    out    var
options))
                    options = new Dictionary<string, string>();

                var platform = factory.Create(out var error, options);
                if (platform == null)
                {
                    Logger.Log($"error         while         instancing         platform
'{factory.GetPlatformName()}': {error}");
                    continue;
                }

                platforms.Add(platform);
            }

            return platforms;
        }

        public IPlatform GetPlatform(string platformName)
        {
            return   _loadedPlatforms.TryGetValue(platformName,   out   var   platform)   ?
platform : null;
        }

        public DeviceData GetDevice(string platformName, string id)
```

```
        {
            return GetPlatform(platformName)?.GetDevice(id);
        }

        public IEnumerable<IPlatform> GetLoadedPlatforms()
        {
            return _loadedPlatforms.Values;
        }
    }
}
```

Файл HostMethods.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using IOT.Common;
using IOTHost;

namespace Host
{
    public class HostMethods
    {
        private delegate void HostMethod(IEnumerable<object> methodParams,
            IMethodInvocationInstance invocationInstance);
        private  Dictionary<string,  HostMethod>  _methods  =  new  Dictionary<string,
HostMethod>();
        private Host _host;

        public HostMethods(Host host)
        {
            _host = host;
            _methods["test_method"] = TestMethod;
            _methods["get_devices"] = GetDevices;
            _methods["get_device_descriptions"] = GetDeviceDescriptions;
        }

        public   void   DispatchMethodInvocation(string   method,   IEnumerable<object>
methodParams, IMethodInvocationInstance invocationInstance)
        {
            if (!_methods.TryGetValue(method, out var hostMethod))
            {
                Logger.Log($"not found host method '{method}'");
                invocationInstance.Failed($"not found host method '{method}'");
                return;
            }

            hostMethod.Invoke(methodParams, invocationInstance);
        }
```

```
        private        void        TestMethod(IEnumerable<object>        methodParams,
IMethodInvocationInstance invocationInstance)
        {
            invocationInstance.Completed("result from test_method!");
        }

        private     void     GetDeviceDescriptions(IEnumerable<object>     methodParams,
IMethodInvocationInstance invocationInstance)
        {
            var descriptions = new List<DeviceDescription>();
            foreach (var platform in _host.GetLoadedPlatforms())
                descriptions.AddRange(platform.GetDeviceDescriptions());
            invocationInstance.Completed(descriptions);
        }

        private        void        GetDevices(IEnumerable<object>        methodParams,
IMethodInvocationInstance invocationInstance)
        {
            var devices = new List<DeviceData>();
            foreach (var platform in _host.GetLoadedPlatforms())
                devices.AddRange(platform.GetDevices());
            var devicesSerialized = devices.Select(dev => dev.ToJson());
            invocationInstance.Completed(devicesSerialized);
        }
    }
}
```

Файл ServerMessageProcessor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using IOT.Common;
using IOTHost;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using ZeroMQ;

namespace Host
{
    public class ServerMessageProcessor : AbstractMessageProcessor<ClientRequestRecord>
    {
        private const string TAG = "ServerMessageProcessor";
        private readonly Host _host;

        public ServerMessageProcessor(Host host)
        {
            _host = host;
        }

        public void HandleMessage(ZMessage message)
```

```
        {
            var identity = message[0].ReadString();
            var messageContent = message[1].ReadString();

            var json = JObject.Parse(messageContent);
            if (json != null)
            {
                var action = (string) json["action"];
                var data = json["data"];
                var invocationID = (ulong?) json["invocation_id"];

                var requestRecord = new ClientRequestRecord(identity, action,
invocationID, data);
                PutMessage(requestRecord);
            }
            else
                Logger.Log($"[ServerDispatcher] cannot parse message as json:
{messageContent}");
        }

        protected override void ProcessMessage(ClientRequestRecord requestRecord)
        {
            switch (requestRecord.Action.ToLower())
            {
                case "ping":
                    ProcessPing(requestRecord);
                    break;
                case "request:device_invoke_method":
                    RequestDeviceInvokeMethod(requestRecord);
                    break;
                case "request:host_invoke_method":
                    RequestHostInvokeMethod(requestRecord);
                    break;
                default:
                    Logger.Log($"unrecognized action '{requestRecord.Action}'");
                    break;
            }
        }

        private void ProcessPing(ClientRequestRecord requestRecord)
        {
            _host.ClientManager.ProcessClientPing(requestRecord.ClientIdentity);
        }

        public class ClientInvocationInstance : IMethodInvocationInstance
        {
            private readonly ServerNetworkProcessor _networkProcessor;
            private readonly string _clientID;
            private readonly ulong _clientInvocationID;

            public ClientInvocationInstance(ServerNetworkProcessor networkProcessor,
string clientID, ulong invocationID)
            {
                _networkProcessor = networkProcessor;
                _clientID = clientID;
```

```
                _clientInvocationID = invocationID;
            }

            public void Completed(object result)
            {
                _networkProcessor.SendMethodInvocationResult(_clientID,
_clientInvocationID, result);
            }

            public void Failed(string error)
            {
                _networkProcessor.SendMethodInvocationError(_clientID,
_clientInvocationID, error);
            }

            public void TimedOut()
            {
                throw new NotImplementedException();
            }
        }

        private void RequestDeviceInvokeMethod(ClientRequestRecord requestRecord)
        {
            if (requestRecord.InvocationID == null)
            {
                Logger.Log("invoke device method requested, but invocation_id not
present");
                return;
            }

            var platformName = (string) requestRecord.Data["platform"];
            var deviceID = (string) requestRecord.Data["device_id"];
            var method = (string) requestRecord.Data["method"];
            var serializedMethodParams = requestRecord.Data["params"].ToList();

            var invocationInstance = new
ClientInvocationInstance(_host.ServerNetworkProcessor, requestRecord.ClientIdentity,
requestRecord.InvocationID.Value);
            var platform = _host.GetPlatform(platformName);
            if (platform == null)
            {
                invocationInstance.Failed($"platform '{platformName}' not loaded");
                return;
            }

            if (platform.GetDevice(deviceID) == null)
            {
                invocationInstance.Failed($"platform '{platform}' doesn't contains
device with id '{deviceID}'");
                return;
            }

            platform.InvokeDeviceMethod(deviceID, method, serializedMethodParams,
invocationInstance);
        }
```

```
        private void RequestHostInvokeMethod(ClientRequestRecord requestRecord)
        {
            if (requestRecord.InvocationID == null)
            {
                Logger.Log("invoke host method requested, but invocation_id not
present");
                return;
            }

            var method = (string) requestRecord.Data["method"];
            var methodParams = requestRecord.Data["params"].ToObject<List<object>>();
            var invocationInstance = new
ClientInvocationInstance(_host.ServerNetworkProcessor, requestRecord.ClientIdentity,
requestRecord.InvocationID.Value);

            _host.HostMethods.DispatchMethodInvocation(method, methodParams,
invocationInstance);
        }
    }
}
```

Файл ServerNetworkProcessor.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Threading;
using IOT.Common;
using IOTHost;
using Newtonsoft.Json.Linq;
using ZeroMQ;

namespace Host
{
    public class ServerNetworkProcessor : AbstractNetworkProcessor
    {
        private Host _host;

        public ServerNetworkProcessor(Host host, string address)
            : base(address)
        {
            _host = host;
        }

        protected override ZSocket SocketInitialization(ZContext context, string
address)
        {
            var socket = new ZSocket(context, ZSocketType.ROUTER);
            socket.ReceiveTimeout = TimeSpan.FromMilliseconds(10);
            socket.Bind(address);
```

```
            return socket;
        }

        public override void SendMessage(string message)
        {
            throw new MethodAccessException("should use SendMessageToClient()");
        }

        public void SendMethodInvocationResult(string clientID, ulong invocationID,
object result)
        {
            var data = new Dictionary<string, object>
            {
                {"status", "success"},
                {"result", result}
            };

            SendActionToClient(clientID, "result:invoke_method", data, invocationID);
        }

        public void SendMethodInvocationError(string clientID, ulong invocationID,
string error)
        {
            var data = new Dictionary<string, object>
            {
                {"status", "failure"},
                {"error", error}
            };

            SendActionToClient(clientID, "result:invoke_method", data, invocationID);
        }

        public void BroadcastEventMessage(IEnumerable<string> clients, string platform,
string deviceID, string eventName, ICollection<object> eventParams)
        {
            var data = new Dictionary<string, object>
            {
                {"platform", platform},
                {"device_id", deviceID},
                {"event", eventName},
                {"params", eventParams}
            };

            foreach (var clientID in clients)
                SendActionToClient(clientID, "device_event", data, null);
        }

        protected override void HandleMessage(ZMessage message)
        {
            _host.ServerMessageProcessor.HandleMessage(message);
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД КЛИЕНТСКОЙ БИБЛИОТЕКИ

Файл Client.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Timers;
using IOT.Common;
using IOT.Common.DataTypes;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace IOT.Client
{
    public class Client : IDisposable
    {
        internal readonly ClientNetworkProcessor NetworkProcessor;
        internal readonly ClientMessageProcessor Dispatcher;
        internal readonly ClientInvocationManager InvocationManager;
        internal readonly EventDispatcher EventDispatcher;
        public readonly List<DeviceDescription> DeviceDescriptions = new
List<DeviceDescription>();
        public readonly List<DeviceData> Devices = new List<DeviceData>();
        internal readonly Timer PingTimer = new Timer(2000);
        public readonly string HostIP;
        public readonly int HostPort;


        public Client(string hostIP, int hostPort, string identity)
        {
            HostIP = hostIP;
            HostPort = hostPort;

            Dispatcher = new ClientMessageProcessor(this);
            NetworkProcessor = new ClientNetworkProcessor(this,
$"tcp://{hostIP}:{hostPort}", identity);
            InvocationManager = new ClientInvocationManager(this);
            EventDispatcher = new EventDispatcher(this);

            PingTimer.Elapsed += OnPingTimerEvent;
            PingTimer.AutoReset = true;
            PingTimer.Enabled = true;
            OnPingTimerEvent(null, null);
        }

        public void Stop()
        {
            Dispatcher.Stop();
```

```
            NetworkProcessor.Stop();
        }

        public void Dispose()
        {
            Stop();
        }

        internal DeviceDescription GetDeviceDescription(string platform, string type)
        {
            return DeviceDescriptions.Find(desc => desc.Platform == platform &&
desc.Type == type);
        }

        public async Task<Device> GetDevice(string platform, string id, bool sync)
        {
            var deviceData = Devices.Find(dev => dev.Platform.Equals(platform) &&
dev.ID.Equals(id));
            if (deviceData == null)
            {
                if (sync)
                {
                    await SyncDeviceDescriptions();
                    await SyncDevices();
                    return await GetDevice(platform, id, false);
                }

                return await Task.FromResult<Device>(null);
            }

            var description =
                DeviceDescriptions.Find(desc => desc.Platform.Equals(platform) &&
desc.Type.Equals(deviceData.Type));
            if (description == null)
                return await Task.FromResult<Device>(null);

            return await Task.FromResult(new Device(this, description, deviceData));
        }

        public async Task Sync()
        {
            await SyncDeviceDescriptions();
            await SyncDevices();
        }

        public async Task SyncDeviceDescriptions()
        {
            var result = await InvokeHostMethod("get_device_descriptions", null);
            if(!(result is JArray))
            {
                Logger.Log($"result of get_device_descriptions should be array, but got
{result.GetType()}");
                return;
            }
```

```
            using (var reader = ((JArray)result).CreateReader())
            {
                JsonSerializer.CreateDefault().Populate(reader, DeviceDescriptions);
            }

            foreach (var description in DeviceDescriptions)
                description.SetupDataTypeConverters();
        }

        public async Task SyncDevices()
        {
            var result = await InvokeHostMethod("get_devices", null);
            if(!(result is JArray))
            {
                Logger.Log($"result of get_devices should be array, but got
{result.GetType()}");
                return;
            }

            var devicesData = (result as JArray).ToList();
            foreach (var json in devicesData)
            {
                if(!DeviceData.ReadPlatformAndType(json, out var platform, out var
type))
                    continue;

                var deviceDescription = GetDeviceDescription(platform, type);
                if(deviceDescription == null)
                {
                    Logger.Log($"no description for device '{platform}:{type}'");
                    continue;
                }

                var newDeviceData = DeviceData.CreateFromJson(json, deviceDescription);
                var existingDeviceData = Devices.Find(d => d.Platform ==
newDeviceData.Platform && d.ID == newDeviceData.ID);

                if (existingDeviceData != null)
                    existingDeviceData.UpdateFromJson(json, deviceDescription);
                else
                    Devices.Add(newDeviceData);
            }
        }

        public async Task<object> InvokeHostMethod(string method, IEnumerable<JToken>
serializedMethodParams)
        {
            return await InvocationManager.InvokeHostMethod(method,
serializedMethodParams);
        }

        public DataChannel OpenDataChannel(DataChannelDescription description)
        {
            return new DataChannel(this, description);
        }
```

```
        public void DumpDevices()
        {
            Logger.EndLog();
            foreach (var device in Devices)
            {
                Logger.WriteLogLine($"'{device.Platform}':'{device.Type}'
id:'{device.ID}' name:'{device.Name}' online:{device.IsOnline}");
            }
            Logger.EndLog();
        }

        private void OnPingTimerEvent(object source, ElapsedEventArgs e)
        {
            NetworkProcessor.SendAction("ping", null);
        }
    }
}
```

Файл ClientInvocationManager.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using IOT.Common;
using Newtonsoft.Json.Linq;

namespace IOT.Client
{
    internal class ClientInvocationManager : AbstractInvocationManager
    {
        private readonly Client _client;

        public ClientInvocationManager(Client client)
        {
            _client = client;
        }

        private class AsyncMethodInvocationInstance : IMethodInvocationInstance
        {
            private readonly TaskCompletionSource<object> _taskCompletionSource;

            public AsyncMethodInvocationInstance(TaskCompletionSource<object>
taskCompletionSource)
            {
                _taskCompletionSource = taskCompletionSource;
            }

            public void Completed(object result)
            {
                _taskCompletionSource.SetResult(result);
```

```
            }

            public void Failed(string error)
            {
                _taskCompletionSource.SetException(new IOTException(error));
            }

            public void TimedOut()
            {
                _taskCompletionSource.SetException(new Exception("invocation timed
out"));
            }
        }

        private bool CreateInvocation(out ulong invocationID, out Task<object> task)
        {
            var taskCompletionSource = new TaskCompletionSource<object>();
            var invocationInstance = new
AsyncMethodInvocationInstance(taskCompletionSource);

            invocationID = NewInvocationID();
            var invocationAdded = InvocationsInProgress.TryAdd(invocationID,
invocationInstance);

            task = invocationAdded ? taskCompletionSource.Task :
                Task.FromException<object>(new Exception($"cannot create invocation with
id {invocationID}"));
            return invocationAdded;
        }

        public async Task<object> InvokeDeviceMethod(string platform, string deviceID,
string method,
            IEnumerable<JToken> serializedMethodParams)
        {
            var data = new Dictionary<string, object>
            {
                {"platform", platform},
                {"device_id", deviceID},
                {"method", method},
                {"params", serializedMethodParams}
            };

            if(CreateInvocation(out var invocationID, out var task))
                _client.NetworkProcessor.SendAction("request:device_invoke_method",
data, invocationID);

            return await task;
        }

        public async Task<object> InvokeHostMethod(string method, IEnumerable<JToken>
serializedMethodParams)
        {
            var data = new Dictionary<string, object>
            {
                {"method", method},
```

```
                {"params", serializedMethodParams}
            };

            if(CreateInvocation(out var invocationID, out var task))
                _client.NetworkProcessor.SendAction("request:host_invoke_method", data,
invocationID);

            return await task;
        }
    }
}
```

Файл ClientMessageProcessor.cs

```
using System;
using System.Linq;
using IOT.Common;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using ZeroMQ;

namespace IOT.Client
{
    internal class ClientMessageProcessor : AbstractMessageProcessor<JObject>
    {
        private const string TAG = "ClientMessageProcessor";
        private readonly Client _client;

        public ClientMessageProcessor(Client client)
        {
            _client = client;
        }

        public void HandleMessage(ZMessage message)
        {
            var content = message[0].ReadString();
            var json = JObject.Parse(content);
            if (json != null)
                PutMessage(json);
        }

        protected override void ProcessMessage(JObject message)
        {
            var action = (string) message["action"];
            var data = message["data"];
            var invocationID = (ulong?) message["invocation_id"];

            switch (action.ToLower())
            {
                case "result:invoke_method":
                    if(invocationID != null)
```

```
                    ResultInvokeMethod(data, invocationID.Value);
                else
                    Logger.Log("action is result:invoke_method, but invocation_id
not present");
                break;
            case "device_event":
                ProcessDeviceEvent(data);
                break;
            default:
                Logger.Log($"[{TAG}] unrecognized action '{action}'");
                break;
        }
    }

    private void ResultInvokeMethod(JToken data, ulong invocationID)
    {
        var status = (string) data["status"];

        switch (status)
        {
            case "success":
                var result = (object) data["result"];
                _client.InvocationManager.ProcessInvocationResult(invocationID,
result);
                break;
            case "failure":
                var error = (string) data["error"];
                _client.InvocationManager.ProcessInvocationError(invocationID,
error);
                break;
            default:
                Logger.Log($"unknown type of invocation result: {status}");
                break;
        }
    }

    private void ProcessDeviceEvent(JToken data)
    {
        var platform = (string) data["platform"];
        var deviceID = (string) data["device_id"];
        var eventName = (string) data["event"];
        var eventParams = data["params"].ToList();

        _client.EventDispatcher.ProcessEvent(platform, deviceID, eventName,
eventParams);
    }
}
}
```

Файл ClientNetworkProcessor.cs

```
using System;
```

```
using System.Text;
using IOT.Common;
using ZeroMQ;

namespace IOT.Client
{
    internal class ClientNetworkProcessor : AbstractNetworkProcessor
    {
        private readonly string _identity;
        private Client _client;

        public ClientNetworkProcessor(Client client, string address, string identity)
            : base(address)
        {
            _client = client;
            _identity = identity;
        }

        protected override ZSocket SocketInitialization(ZContext context, string
address)
        {
            var socket = new ZSocket(context, ZSocketType.DEALER);
            socket.Identity = Encoding.UTF8.GetBytes(_identity);
            socket.ReceiveTimeout = TimeSpan.FromMilliseconds(10);
            socket.Connect(address);
            return socket;
        }

        protected override void HandleMessage(ZMessage message)
        {
            _client.Dispatcher.HandleMessage(message);
        }
    }
}
```

Файл DataChannel.cs

```
using System;
using System.Collections.Concurrent;
using System.Linq;
using System.Timers;
using IOT.Common;
using IOT.Common.DataTypes;
using ZeroMQ;

namespace IOT.Client
{
    public class DataChannel : AbstractNetworkProcessor
    {
        private readonly BlockingCollection<ZMessage> _incomingMessageQueue = new
BlockingCollection<ZMessage>();
```

```
        private DateTime _lastPingTime = DateTime.UtcNow;
        private Timer _pingTimer = new Timer(2000);

        internal DataChannel(Client client, DataChannelDescription
dataChannelDescription)
            : base($"tcp://{client.HostIP}:{dataChannelDescription.Port}")
        {
            _pingTimer.Elapsed += OnSendPingTimerEvent;
            _pingTimer.AutoReset = true;
            _pingTimer.Enabled = true;
        }

        protected override ZSocket SocketInitialization(ZContext context, string
address)
        {
            var socket = new ZSocket(context, ZSocketType.PAIR);
            socket.ReceiveTimeout = TimeSpan.FromMilliseconds(10);
            socket.Linger = TimeSpan.Zero;
            socket.Connect(address);
            return socket;
        }

        protected override void HandleMessage(ZMessage message)
        {
            if(message.PopString() == "ping")
                _lastPingTime = DateTime.UtcNow;
            else
                _incomingMessageQueue.Add(message.Duplicate());
        }

        public ZMessage TakeMessage()
        {
            var message = _incomingMessageQueue.Take();
            return message;
        }

        public bool IsActive(TimeSpan threshold)
        {
            return DateTime.UtcNow - _lastPingTime < threshold;
        }

        private void OnSendPingTimerEvent(object sender, ElapsedEventArgs args)
        {
            SendMessage("ping");
        }
    }
}
```

Файл Device.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using IOT.Common;
using Newtonsoft.Json.Linq;

namespace IOT.Client
{
    public class Device
    {
        private readonly Client _client;
        private readonly DeviceDescription _description;
        private readonly DeviceData _deviceData;

        public string Platform => _deviceData.Platform;
        public string Type => _deviceData.Type;
        public string ID => _deviceData.ID;
        public string Name => _deviceData.Name;

        internal Device(Client client, DeviceDescription description, DeviceData
deviceData)
        {
            _client = client;
            _description = description;
            _deviceData = deviceData;
        }

        public async Task<object> InvokeMethod(string method,
IReadOnlyCollection<object> valueParams)
        {
            var methodDescription = _description.Methods[method];
            if (valueParams == null && methodDescription.ParamTypes.Count != 0)
                throw new ArgumentException($"params array is null, but expected
{methodDescription.ParamTypes.Count} params");
            if(valueParams != null && methodDescription.ParamTypes.Count !=
valueParams.Count)
                throw  new ArgumentException($"got {valueParams.Count} params, but
expected {methodDescription.ParamTypes.Count} params");

            List<JToken> serializedParams = new List<JToken>();

            if(valueParams != null)
                foreach (var (value, type) in
valueParams.Zip(methodDescription.ParamTypes, (value, type) => (value, type)))
                    serializedParams.Add(type.Converter.ValueToJson(value));

            JToken result = (JToken)await
_client.InvocationManager.InvokeDeviceMethod(_deviceData.Platform, _deviceData.ID,
method,
                serializedParams);
            return methodDescription.ResultConverter.JsonToValue(result);
```

```
        }

        public void SubscribeEvent(string eventName, Action<Device, string, object[]>
callback)
        {
            _client.EventDispatcher.SubscribeEvent(this, eventName, callback);
        }

        public void SubscribeEvent(string eventName, Action<object[]> callback)
        {
            SubscribeEvent(eventName, (d, e, a) => callback.Invoke(a));
        }

        public void SubscribeEvent(string eventName, Action callback)
        {
            SubscribeEvent(eventName, (d, e, a) => { callback.Invoke(); });
        }

        public void SubscribeEvent(string eventName, Func<Device, string, object[],
Task> callback)
        {
            SubscribeEvent(eventName, (d, e, a) => callback.Invoke(d, e, a).Wait());
        }

        public void SubscribeEvent(string eventName, Func<Task> callback)
        {
            SubscribeEvent(eventName, (d, e, a) => callback.Invoke().Wait());
        }

        public object GetProp(string prop)
        {
            var propDescription = _description.Props.First(desc => desc.Name == prop);
            if(propDescription == null)
                throw new IOTException($"no prop '{prop}' in device
'{Platform}:{Type}'");
            return _deviceData.Props.TryGetValue(prop, out var result) ? result : null;
        }
    }
}
```

Файл EventDispatcher.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing.Reader;
using System.Linq;
using System.Threading;
using IOT.Common;
using Newtonsoft.Json.Linq;
```

```
namespace IOT.Client
{
    public class EventDispatcher
    {
        private readonly Client _client;
        private readonly List<EventRecord> _eventRecords = new List<EventRecord>();

        private class EventRecord
        {
            public Device Device;
            public string EventName;
            public Action<Device, string, object[]> Callback;

            public EventRecord(Device device, string eventName, Action<Device, string,
object[]> callback)
            {
                Device = device;
                EventName = eventName;
                Callback = callback;
            }
        }

        public EventDispatcher(Client client)
        {
            _client = client;
        }

        public void SubscribeEvent(Device device, string eventName, Action<Device,
string, object[]> callback)
        {
            var deviceDescription = _client.GetDeviceDescription(device.Platform,
device.Type);
            if(deviceDescription == null)
                throw new IOTException($"description not found for device type
{device.Platform}:{device.Type}");

            var eventDescription = deviceDescription.Events.First(e => e.Name ==
eventName);
            if(eventDescription == null)
                throw new IOTException($"no event {eventName} found in device type
{device.Platform}:{device.Type}");

            lock (_eventRecords)
            {
                if (_eventRecords.Exists(r => r.Device.Platform == device.Platform &&
r.Device.ID == device.ID && r.EventName == eventName && r.Callback == callback))
                    return;

                var eventRecord = new EventRecord(device, eventName, callback);
                _eventRecords.Add(eventRecord);
            }
        }

        public void UnsubscribeEvent(Device device, string eventName, Action<Device,
string, object[]> callback)
```

```
        {
            int removed;
            lock (_eventRecords)
                removed = _eventRecords.RemoveAll(r => r.Device.Platform ==
device.Platform && r.Device.ID == device.ID && r.EventName == eventName
&& r.Callback == callback);
            if(removed > 0)
                Logger.Log($"removed {removed} records");
        }

        public void UnsubscribeAll(Device device)
        {
            lock (_eventRecords)
                _eventRecords.RemoveAll(r => r.Device.Platform == device.Platform &&
r.Device.ID == device.ID);
        }

        internal void ProcessEvent(string platform, string deviceID, string eventName,
ICollection<JToken> serializedEventParams)
        {
            List<EventRecord> matched;
            lock (_eventRecords)
                matched = _eventRecords.FindAll(r =>
                    r.Device.Platform == platform && r.Device.ID == deviceID &&
r.EventName == eventName);
            foreach (var eventRecord in matched)
            {
                ThreadPool.QueueUserWorkItem(state =>
eventRecord.Callback.Invoke(eventRecord.Device, eventName, new object[]{}));
            }
        }
    }
}
```