

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 серверной части 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  
\_\_\_\_\_ В.Ж. Алексеев  
«\_\_\_»\_\_\_\_\_ 2019 г.

Нормоконтролёр,  
ст. преп. каф. ЭВМ  
\_\_\_\_\_ С.В. Сяськов  
«\_\_\_»\_\_\_\_\_ 2019 г.

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»

УТВЕРЖДАЮ  
Заведующий кафедрой ЭВМ  
\_\_\_\_\_ Г.И. Радченко  
«\_\_\_» \_\_\_\_\_ 2019 г.

### **ЗАДАНИЕ**

**на выпускную квалификационную работу магистра**  
студенту группы КЭ-222  
Алексееву Владимиру Жанновичу  
обучающемуся по направлению  
09.04.01 «Информатика и вычислительная техника»

**Тема работы:** «Разработка серверной части системы расширения функционального взаимодействия компонентов интернета вещей» утверждена приказом по университету от 25 апреля 2019 г. № 899

**Срок сдачи студентом законченной работы:** 1 июня 2019 г.

**Исходные данные к работе:** статьи, книги, техническое задание.

- Блох, Д. Java. Эффективное программирование / Д. Блох. – М.: Лори, 2014. – 310 с.;
- Антти, С. Интернет вещей. Видео, аудио, коммутация / С. Антти. – М.: ДМК Пресс, 2019. – 120 с.;
- Akgul, F. ZeroMQ / F. Akgul. – Бирмингем: Packt Publishing, 2011. – 140 с.

**Перечень подлежащих разработке вопросов:**

- рассмотрение существующих программных комплексов для работы с IoT-устройствами Xiaomi;
- исследование технологии инъектирования кода;
- разработка собственной база для интеграции экосистемы MI Home в расширяемую систему взаимодействия IoT-устройств;
- тестирование и отладка разработанной интеграции;
- написание руководства пользователя.

**Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы \_\_\_\_\_ /И.Л. Кафтанников/

Студент \_\_\_\_\_ /В.Ж. Алексеев/

## КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2019	
Разработка модели, проектирование	01.04.2019	
Реализация системы	01.05.2019	
Тестирование, отладка, эксперименты	15.05.2019	
Компоновка текста работы и сдача на нормоконтроль	24.05.2019	
Подготовка презентации и доклада	30.05.2019	

Руководитель работы \_\_\_\_\_ /И.Л. Кафтанников/

Студент \_\_\_\_\_ /В.Ж. Алексеев/

## Аннотация

В.Ж. Алексеев. Разработка серверной части системы расширения функционального взаимодействия компонентов интернета вещей. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 59 с., 11 ил., библиогр. список – 15 наим.

В рамках выпускной квалификационной работы производится обзор существующих решений для взаимодействия с IoT-устройствами экосистемы Xiaomi. Предлагает универсальный вариант интеграции всех устройств экосистемы Xiaomi в расширяемую систему взаимодействия IoT-устройств. Реализуется база для интеграции Xiaomi устройств методом инъекции кода, производится интеграция физических устройств Xiaomi, имеющихся на базе кафедры электронных вычислительных машин ФГАОУ ВО «ЮУрГУ (НИУ)». Разработанная интеграция протестирована с помощью юнит-тестов, а также на физических устройствах Xiaomi. Написана документация для пользователей интеграцией и разработчиков плагинов новых устройств. Предложены сценарии использования.

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	7
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	9
1.1. ОБЗОР АНАЛОГОВ.....	10
1.3. ВЫВОД.....	14
2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ.....	15
2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ.....	15
2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ.....	16
3. ПРОЕКТИРОВАНИЕ.....	17
3.1. ВЫБОР МЕТОДА ИНТЕГРАЦИИ.....	17
3.2. ФУНКЦИОНАЛ ПРИЛОЖЕНИЯ MI HOME.....	17
3.3. ВЫБОР БИБЛИОТЕКИ ДЛЯ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ.....	18
3.4. ОПИСАНИЕ МЕТОДА ИНЪЕКЦИИ КОДА.....	18
3.5. АРХИТЕКТУРА МОДИФИКАЦИИ.....	20
3.6. РАЗРАБОТКА ОСНОВНОЙ ЧАСТИ МОДИФИКАЦИИ.....	22
3.7. ОПИСАНИЕ УСТРОЙСТВ.....	28
3.8. ПОДДЕРЖИВАЕМЫЕ УСТРОЙСТВА НА ДАННЫЙ МОМЕНТ.....	33
3.9. ИНТЕГРАЦИЯ В РАСШИРЯЕМУЮ СИСТЕМУ ВЗАИМОДЕЙСТВИЯ IOT-УСТРОЙСТВ.....	34
5. ТЕСТИРОВАНИЕ.....	40
5.1. МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ.....	40
5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ.....	40
6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	42
7. ЗАКЛЮЧЕНИЕ.....	44
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	45
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ.....	47

## ВВЕДЕНИЕ

Стремительное развитие технологий позволяет минимизировать участие человека в различных операциях. Одна из таких технологий – Интернет вещей (IoT) [1]. Это концепция вычислительной сети физических предметов, оснащённых встроенными технологиями для взаимодействия друг с другом или с внешней средой, данное явление способно перестроить экономические и общественные процессы, исключив из части действий и операций необходимость участия человека. В рамках конференции разработчиков MIDC Xiaomi IoT 2017 исполнительный директор Xiaomi Лэй Цзюнь объявил, что компания создала самую большую на планете платформу Интернета вещей для умных устройств. На сегодняшний день Xiaomi выпустила 85 млн устройств, имеющих подключение к сети. В платформе компании числятся свыше 800 типов гаджетов, созданных при участии 400 партнеров со всего света [2]. Также Лэй Цзюнь сообщил, что Xiaomi планирует в ближайшие пять лет инвестировать как минимум 10 млрд. юаней (\$1,5 млрд.) в развитие технологий искусственного интеллекта и интернета вещей. Сейчас к платформе Xiaomi суммарно подключено более 132 млн «умных» устройств, более 20 млн из них ежедневно активны в более чем 200 странах и регионах по всему миру. При этом свыше 1,9 млн человек имеют как минимум 5 устройств IoT-платформы Xiaomi.

В связи с ростом количества устройств и популярностью данного производителя, возникают две проблемы. Первая проблема: отсутствие возможности управлять смарт-устройствами Xiaomi сторонними приложениями и интегрировать данные устройства в другие экосистемы. Вторая: обучение

студентов взаимодействию с IoT-устройствами для получения навыков создания сложных автоматизированных систем на основе компонентов Xiaomi.

В настоящее время полный функционал возможностей доступен только в приложении MI Home. Данное приложение работает только на мобильных устройствах, что неудобно для некоторых пользователей. Также приложение предоставляет ограниченные возможности по автоматизации. Существуют различные варианты интеграции к другим экосистемам, но они не поддерживают все устройства, а также их полный функционал. Вышеперечисленные минусы существующего программного обеспечения не позволяют в полной мере использовать его для обучения.

Актуальность и необходимость создания такой системы, которая бы позволила интегрировать устройства производителей Xiaomi в расширяемую систему взаимодействия компонентов интернета вещей, которая была разработана для магистерской диссертация – далее «Расширяемая система взаимодействия компонентов интернета вещей»[3], создавать сложные сценарии, использовать данные устройства в обучении взаимодействия с IoT-устройствами, а также устранила необходимость быть привязанным к приложению MI Home, обусловлена следующими аспектами:

1. Поддержка всех устройств от производителя Xiaomi.
2. Возможность интеграции полного функционала устройств.
3. Управление устройствами не через приложение MI Home.
4. Возможность эксплуатировать систему для популярного направления интернета вещей – создание сложных сценариев для умного дома с использованием устройств от Xiaomi.

Цель работы – интеграция IoT-устройств экосистемы MI Home в расширяемую систему взаимодействия компонентов интернета вещей. Для осуществления поставленной цели необходимо реализовать следующие задачи:



1. Исследовать существующие реализации интеграций, выделить функциональные возможности и проблемы.
2. Исследовать технологии, произвести интеграцию устройств MI Home.
3. Реализовать базу для интеграции экосистемы MI Home в расширяемую систему взаимодействия компонентов интернета вещей.
4. Интегрировать устройства.
5. Написать документацию по работе с расширением.
6. Протестировать разработанное расширение.

## **1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

На сегодняшний день IoT-устройства Xiaomi являются одними из самых доступных и многочисленных. Компания предоставляет широкий выбор «умных устройств» для постройки «умного дома». Xiaomi создала одну из крупнейших экосистем в мире «Интернета вещей». Интернет вещей (IoT) – это сеть, состоящая из уникально идентифицируемых объектов («вещей»), способных взаимодействовать друг с другом без вмешательства со стороны человека [4].

Высокий интерес людей к «умным устройствам» приводит к увеличению количества таких устройств. Большим спросом пользуется фирма Xiaomi – на данный момент к сети подключено более 132 миллионов IoT-устройств. Данное число продолжает расти. Единственным программное обеспечение, позволяющее полноценно взаимодействовать со смарт-устройствами Xiaomi – это их проприетарное приложение MI Home, существующее только для мобильных операционных систем. Задача состоит в том, чтобы интегрировать эти устройства в расширяемую систему взаимодействия, чтобы получить возможность взаимодействовать устройствам от Xiaomi с другими производителями IoT-устройств.

С каждым годом в оборот поступает все больше устройств, оснащенных интеллектуальными программами для сбора и обработки информации. 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 устройств интернета вещей от Xiaomi, существует множество различных программ для управления. Для анализа мы выбрали самые популярные и актуальные системы. Помимо полноценных систем для управления умным домом, существуют библиотеки для взаимодействия с устройствами системы MI Home. Например, python-miio и js-miio [5]. Однако, данные библиотеки поддерживают ограниченный список устройств и отсутствует возможность регистрации событий.

«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. Данная платформа умеет взаимодействовать с ограниченным пулом устройств из экосистемы MI Home. А поддерживаемые устройства работают с данной

системой нестабильно: медленный отклик или отсутствие отклика на заданные команды.



Рисунок 1 – Скриншот системы «Domoticz», управление устройствами 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. Данная платформа позволяет использовать устройства из экосистемы MI Home, но существуют проблемы при реализации сложных сценариев, а именно: недостаточная стабильность при работе с устройствами. Например, сложно добиться аналогичности действий при взаимодействии со смарт-устройствами используя разные устройства управления. Также в данный программный продукт нет возможности интегрировать любое устройство от Xiaomi.

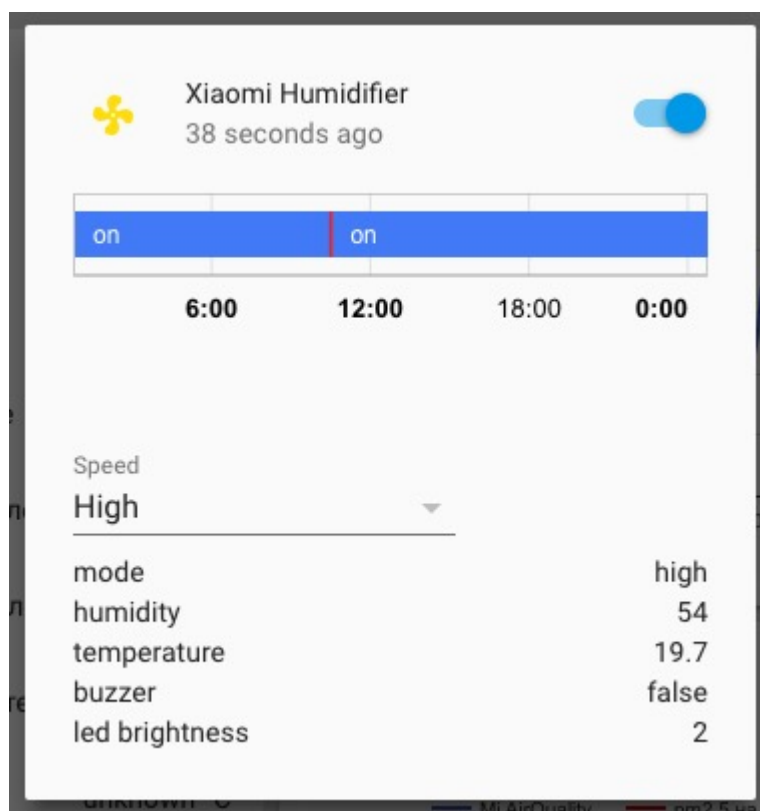


Рисунок 2 – Скриншот системы «Home Assistant», управление увлажнителем воздуха от Xiaomi

«MajorDoMo» [8] - система представляет собой бесплатную и открытую программную платформу для комплексного управления домашней автоматикой, а также для информационной поддержки жизнедеятельности. Русскоязычная платформа для самостоятельного создания Умного дома с открытым исходным кодом, написана на PHP, выпущена под лицензией MIT. Скриншот приложения представлен на рисунке 3. В данном программном комплексе существует поддержка 28 устройств через проприетарный сетевой протокол Xiaomi. Стоит отметить, что поддерживают только самые популярные «умные» устройства, поддержка других устройств их экосистемы отсутствует. Также недостатком является то, что устройства должны быть обязательно в одном сегменте локальной сети, в котором разрешен широковещательный UDP-трафик.

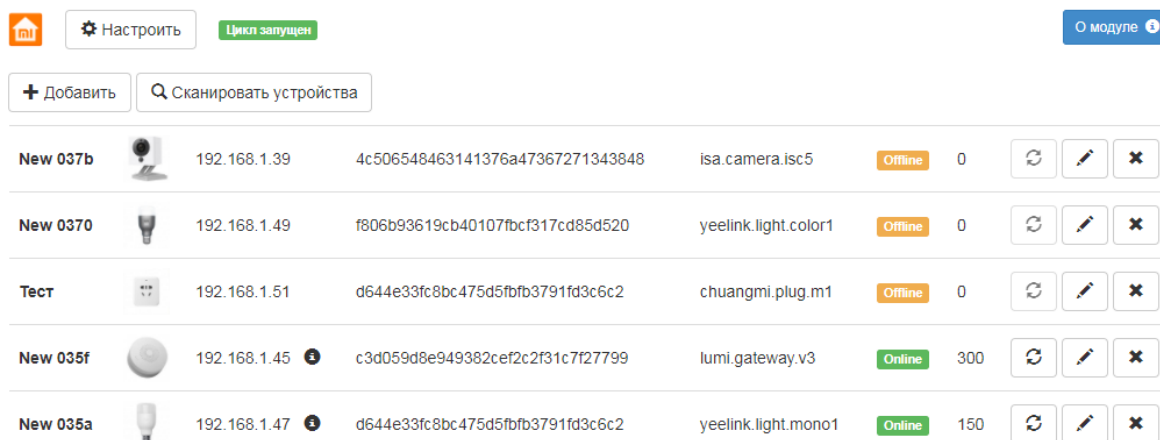


Рисунок 3 – скриншот системы «MajorDoMo» с подключенным модулем Xiaomi miIO Devices

«openHAB» [9] – система с открытым кодом, которая способна объединить системы домашней автоматизации, «умные устройства» и другие технологии в единое решение. Данная платформа обеспечивает единый пользовательский интерфейс и общий подход к правилам автоматизации во всей системе, независимо от количества задействованных производителей и подсистем. Скриншот приложения представлен на рисунке 4. Данный программный комплекс также взаимодействует по проприетарному сетевому протоколу Xiaomi, соответственно, на сегодняшний день поддерживаются всего 28 устройств.

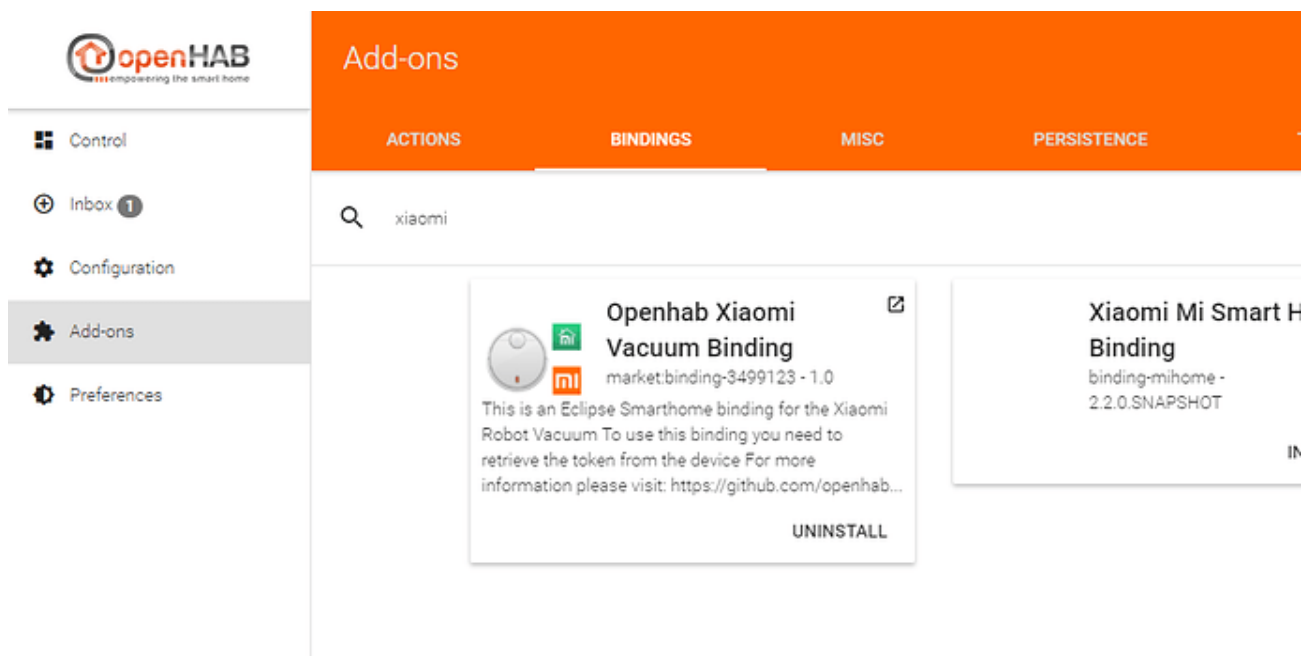


Рисунок 4 – скриншот системы «openHAB», расширение для робота-пылесоса Xiaomi

### 1.3. ВЫВОД

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

1. Возможность поддержки всех устройств от Xiaomi.
2. Интеграция в расширяемую систему взаимодействия компонентов интернета вещей.
3. Возможность работы не только в локальной сети, но по Интернету.

## 2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Для достижения интеграции устройств Xiaomi в расширяемую систему взаимодействия IoT-устройств, необходимо выполнить следующие действия:

1. Исследовать внутреннее строение официального приложения MI Home.
2. Модифицировать приложение таким образом, чтобы появилась возможность внешнего управления.
3. Написать плагин для интеграции модифицированного приложения в расширяемую систему.
4. Реализовать интеграции для физических устройств Xiaomi, установленных на кафедре ЭВМ.

### 2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

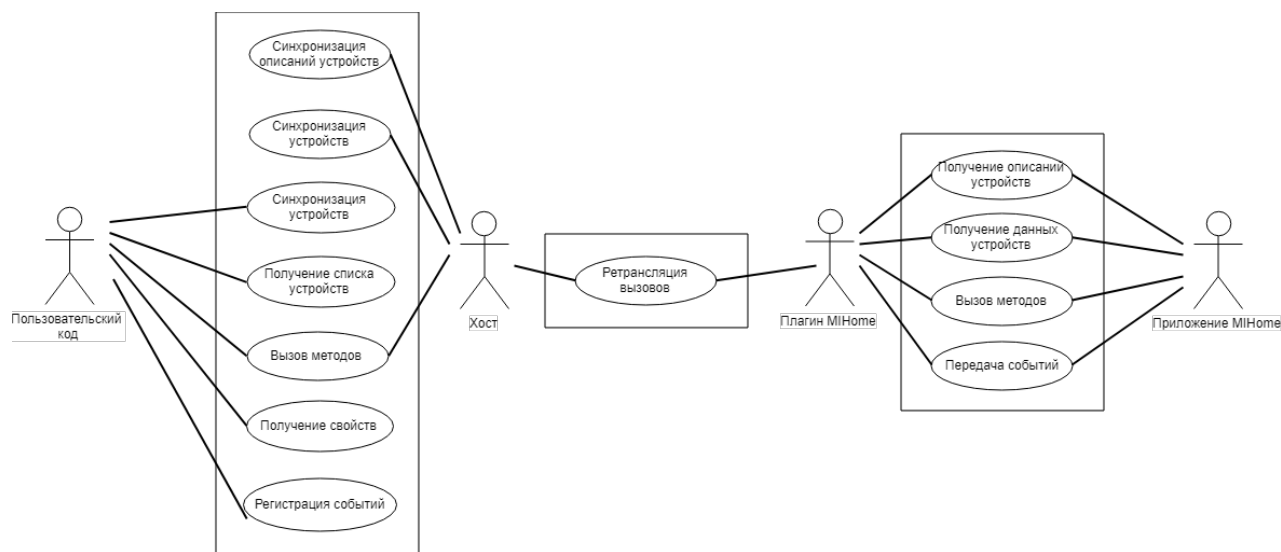


Рисунок 5 – Диаграмма вариантов использования системы включающая плагин расширения и модифицированное приложение

На рисунке 5 показаны 4 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.
3. Плагин MI Home – библиотека, задача которой организовать взаимодействие между расширяемой системой и модифицированным приложением MI Home. Происходит транслирование вызовов методов от хоста к приложению MI Home и обратно.
4. Модифицированное приложение MI Home – программа, запущенная на Android-смартфоне, которая получает запросы от хоста и управляет устройствами.

После загрузки плагина MI Home, происходит синхронизация описания и данных устройств с модифицированным приложением MI Home. После синхронизации хост имеет доступ к устройствам Xiaomi и может передать их пользовательскому коду.

## **2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ**

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

1. Бесперывная работа.
2. Возможность работы как в локальной сети, так и в сети Интернет.
3. Стабильная работа с IoT-устройствами.
4. Возможность поддержки всех «умных устройств» Xiaomi.



## **3. ПРОЕКТИРОВАНИЕ**

### **3.1. ВЫБОР МЕТОДА ИНТЕГРАЦИИ**

Существующие решения реализуют взаимодействие с устройствами Xiaomi, используя проприетарный протокол miIO [10], который управляет устройствами по WiFi. Данный метод не позволяет реализовать управление всеми устройствами, так как многие устройства работают по другим протоколам.

Также данный метод сложен в настройке и не позволяет регистрировать события устройств, например, нажатие на кнопку.

Исходя из этого, необходим новый метод для интеграции устройств, который не имел бы перечисленных недостатков. Для решения этой проблемы, необходимо модифицировать приложение MI Home таким образом, чтобы была возможность управлять им удаленно. Имея модифицированное приложение MI Home, необходимо интегрировать его функционал в расширяемую систему взаимодействия компонентов интернета вещей, путем написания плагина для данной системы.

### **3.2. ФУНКЦИОНАЛ ПРИЛОЖЕНИЯ MI HOME**

MI Home – это официальное приложение от Xiaomi, которое позволяет управлять устройствами умного дома от данного производителя [11]. Данное приложение позволяет:

- подключать устройства умного дома от Xiaomi;
- изменять, переименовывать устройства;
- управлять отдельными устройствами;
- создавать сценарии автоматизации умного дома;

- получать уведомления о срабатывании датчиков или устройств.

После модификации приложение MI Home будет использоваться для добавления новых устройств, а весь остальной функционал будет реализован с помощью внешней системы

### **3.3. ВЫБОР БИБЛИОТЕКИ ДЛЯ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ**

Так как требуется взаимодействовать с Android-приложением, то существует необходимость в сетевом взаимодействии. Для этого была выбрана библиотека ZeroMQ [12]. Данная библиотека обладает следующими преимуществами:

**Производительность.** ZeroMQ работает существенно быстрее, чем большинство реализаций AMQP.

**Удобство использования.** С помощью API ZeroMQ передача сообщения проще, чем при использовании сокетов, где нужно, например, следить за длиной сокетного буфера, а в ZeroMQ - просто инициировать отправку сообщения, а дробление (или агрегация) и отправка делается API в отдельном потоке, асинхронно с выполнением пользовательского кода.

**Платформонезависимость.** Данная библиотека поддерживает различные операционные системы и платформы.

### **3.4. ОПИСАНИЕ МЕТОДА ИНЪЕКЦИИ КОДА**

Модификация оригинального приложения MI Home достигается за счет инъекции кода. Инъекция кода – это ввод фрагмента собственного кода в другую существующую программу или библиотеку с целью изменить поведение. Процесс инъекции кода выглядит следующим образом.

Процесс инъекции начинается с дизассемблирования оригинального APK-файла приложения. Для дизассемблирования используется программа apktool. Чтобы дизассемблировать APK, необходимо выполнить команду “apktool d app.apk”. В результате дизассемблирование с помощью данной программы получается директория, содержащая файлы приложения и программный код приложения в виде smali-код. Smali схож с Java-байт-кодом, но, так как в процессорах ARM-архитектуры много регистров, Google заменили длительные обращения в память (в стек, как в JVM) на быстрые обращения к регистрам [13]. Поэтому основное отличие байт-кода Dalvik от байт-кода JVM — ориентированность на регистры.

#### Листинг 1 – Пример smali-кода

```
.method showToast()V
    .locals 2

    .line 37
    const-string v0, "Test"

    const/4 v1, 0x1

    invoke-static {p0, v0, v1}, Landroid/widget/Toast;
>makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    .line 38
    return-void
.end method
```

Имея в наличии smali-код приложения, можно его модифицировать в любом текстовом редакторе. Однако, вместо того, чтобы вручную писать smali-код, гораздо удобнее будет написать нужную инъекцию на Java, скомпилировать в APK, а после дизассемблировать APK с инъекцией с помощью apktool и скопировать smali-код инъекции в smali-код оригинального приложения [14].

После внедрения нужного кода в smali-код приложения, можно собрать файлы приложения обратно в APK-файл с помощью команды “apktool b app”. После выполнения данной команды, в подпапке dist будет помещен новый APK-файл приложения. Для установки данного файла на смартфон, необходимо подписать этот файл с помощью программы jarsigner.

### 3.5. АРХИТЕКТУРА МОДИФИКАЦИИ

Модификация приложения MI Home состоит из инъекции кода в APK-приложение и динамически подгружаемого кода. Задача инъекции в APK-приложение MI Home состоит в динамическом подключении кода, содержащегося в отдельном APK-файле в контекст приложения MI Home.

Динамическая загрузка кода осуществляется следующим образом: в хранилище смартфона находится файл “injection.apk”, данный файл передается в класс DexClassLoader, который загружает код из файла в JVM исполняемого приложения, вызывается метод loadClass() объекта DexClassLoader, который ищет класс Main. У класса Main находится метод main, который является точкой входа в динамически подгружаемый код. Затем, метод main вызывается и в качестве аргументов ему передаются ссылки на context-приложения MI Home и Activity-объект главного экрана приложения.

Листинг 2 – Функция, осуществляющая динамическую загрузку кода

```
private void loadInjection()
{
    String apkPath = Environment.getExternalStorageDirectory() +
"/mihome_hack/injection.apk";
    String className = "com.mihome_injection.Main";

    File dexOutputDir = getDir("mihome_hack_dexcache", Context.MODE_PRIVATE);
    dexOutputDir.mkdir();
    DexClassLoader dexClassLoader = new DexClassLoader(apkPath,
dexOutputDir.getAbsolutePath(), null, getClassLoader());

    try
    {
        Class<?> mainClass = dexClassLoader.loadClass(className);
```

```

        Method mainMethod;
        mainMethod = mainClass.getMethod("main", Context.class, Activity.class);
        mainMethod.invoke(null, getApplicationContext(), this);
    }
    catch (Exception e)
    {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
        return;
    }
}

```

Помимо динамической загрузки кода в приложение MI Home внедрен класс `DeviceNotificationListenerService`, наследующийся от класса `NotificationListenerService`. Также модифицирован файл `AndroidManifest.xml`, где указан класс `DeviceNotificationListenerService`. Данный класс представляет собой регистратор push-уведомлений. Зарегистрированное уведомление передается объекту `notificationConsumer`, который устанавливается динамически загружающимся кодом и обрабатывает полученное уведомление. Если push-уведомление пришло от приложения MI Home и текст уведомления соответствует заданному формату, то данное уведомление считается триггером события устройства.

### Листинг 3 – Класс регистратора push-уведомлений

```

public class DeviceNotificationListenerService extends NotificationListenerService {

    private static final String TAG = "minj_DevNotification";
    public static NotificationConsumer notificationConsumer = null;

    public interface NotificationConsumer {
        void consume(NotificationListenerService notificationListenerService,
        StatusBarNotification statusBarNotification);
    }
    @Override
    public IBinder onBind(Intent intent) {
        return super.onBind(intent);
    }

    @Override
    public void onNotificationPosted(StatusBarNotification sbn) {
        if(notificationConsumer != null)
            notificationConsumer.consume(this, sbn);
        else
            Log.d(TAG, "notificationConsumer is null");
    }
}

```

### 3.6. РАЗРАБОТКА ОСНОВНОЙ ЧАСТИ МОДИФИКАЦИИ

Основная часть модификации представляет собой отдельный APK-файл, представляющий из себя обычный Android-проект – далее «Модификация», разработанный в Android Studio. Данный APK-файл загружается динамически при старте приложения MI Home. В задачи модификации входит взаимодействие по сети с плагином MiHomePlatform и взаимодействие с устройствами с помощью вызовов методов из приложения MI Home.

На этапе компиляции модификации нет доступа к классам, объявленным в приложении MI Home, поэтому обычный вызов методов не применим. Для доступа к этим классам и вызова их метода, необходимо использовать Reflection API. Рефлексия – это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов. Сам же механизм рефлексии позволяет обрабатывать типы, отсутствующие при компиляции, но появившиеся во время выполнения программы [15]. Рефлексия и наличие логически целостной модели выдачи информации об ошибках дает возможность создавать корректный динамический код.

Основной список того, что позволяет рефлексия:

1. Узнать/определить класс объекта.
2. Получить информацию о модификаторах класса, полях, методах, константах, конструкторах и суперклассах.
3. Выяснить, какие методы принадлежат реализуемому интерфейсу/интерфейсам.
4. Создать экземпляр класса, причем имя класса неизвестно до момента выполнения программы.
5. Получить и установить значение поля объекта по имени.

## 6. Вызвать метод объекта по имени.

### Листинг 4 – Пример использования рефлексии для нахождения полей классов

```
public ArrayList<Field> findFields(String name, String type, Integer modifiers)
{
    ArrayList<Field> matchingFields = new ArrayList<>();

    if(!checkState())
        return matchingFields;

    for(Field f : _class.getDeclaredFields())
    {
        if(modifiers != null && (f.getModifiers() & modifiers) == 0)
            continue;

        if(name != null && !f.getName().equals(name))
            continue;

        if(type != null && !f.getType().getSimpleName().equals(type))
            continue;

        matchingFields.add(f);
    }

    return matchingFields;
}
```

Так как рефлексия активно используется в модификации, то был написан вспомогательный класс `ClassHelper`, позволяющий искать нужные поля и методы класса, указывая различные параметры: имя, тип возврата, тип аргументов и модификаторы.

Для взаимодействия с приложением MI Home необходимо знать, какие существуют классы, за что они отвечают, и какой функционал их методов. Для понимания внутренней структуры, приложение MI Home было декомпилировано в Java-код с помощью программы `jadx`. Программный код был защищен с помощью программы `ProGuard` с целью усложнения разбора кода. Главная задача `ProGuard` - поменять имена объектов, классов, методов, тем самым затрудняя анализ кода для реверс-инженера. Помимо этого, данная программа и оптимизирует код, удаляя неиспользуемые в программе ресурсы.

Для облегчения работы с защищенным кодом, для каждого необходимого класса приложения MI Home были созданы обертки. Обертка класса – это класс,

наследующийся от базового класса `BaseWrapper` и симулирующий поведение (методы и поля) оригинального класса из приложения `MI Home`. Класс `BaseWrapper` содержит вспомогательные методы для облегчения создания классов-оберток:

- `public ClassLoader getObjectClassLoader()` – возвращает объект `ClassLoader`, к которому был загружен класс оборачиваемого объекта;
- `protected Object method(Method m, Object o, Object ... args)` – вызывает метод объекта с помощью рефлексии;
- `protected static Object staticMethod(Method m, Object ... args)` – вызывает статический метод класса с помощью рефлексии;
- `protected Object field(Field f, Object o)` – возвращает содержимое поля объекта с помощью рефлексии;
- `public Object getObject()` – возвращает оборачиваемый объект;
- `public void setObject(Object object)` – заменяет или устанавливает оборачиваемый объект.

Из-за того, что на этапе компиляции модификации нет доступа к типам, объявленным в приложении `MI Home`, то отсутствует возможность реализации интерфейсов, объявленных в приложении `MI Home`. Для возможности использования этих интерфейсов, необходимо использовать динамические прокси-классы. Прокси-класс создается с помощью вызова метода `Proxy.getProxyClass()`, который принимает объект `ClassLoader` и массив интерфейсов, а возвращает объект класса `java.lang.Class`, который загружен с помощью переданного загрузчика классов и реализует переданный массив интерфейсов. Все вызовы методов реализованных интерфейсов будут переадресованы методу `invoke()`, класса `InvocationHandler`.

На передаваемые параметры есть ряд ограничений:



1. Все объекты в массиве `interfaces` должны быть интерфейсами. Они не могут быть классами или примитивами.
2. В массиве `interfaces` не может быть двух одинаковых объектов.
3. Все интерфейсы в массиве `interfaces` должны быть загружены тем класс-лоадером, который передается в метод `getProxyClass`.
4. Все не публичные интерфейсы должны быть определены в одном и том же пакете, иначе генерируемый прокси-класс не сможет их все реализовать.
5. Ни в каких двух интерфейсах не может быть метода с одинаковым названием и сигнатурой параметров, но с разными типами возвращаемого значения.
6. Длина массива `interfaces` ограничена 65535-ю интерфейсами. Никакой Java-класс не может реализовывать более 65535 интерфейсов.
7. Если какое-либо из вышеперечисленных ограничений нарушено - будет выброшено исключение `IllegalArgumentException`, а если массив интерфейсов `interfaces` равен `null`, то будет выброшено `NullPointerException`.

Для удобного использования интерфейсов из приложения `MI Home`, был создан класс `InterfaceWrapper`. Данный класс использует динамический прокси-объект, для создания которого он формирует ассоциации между методами целевого интерфейса и методами переданного объекта. Таким образом можно создать объект, реализующий нужный интерфейс, и в качестве приемника использовать любой класс, в котором есть соответствующие методы их целевого интерфейса.

За сетевое взаимодействие отвечает класс `AbstractNetworkProcessor`. Для своей работы данный класс использует библиотеку `ZeroMQ` и выполняет свою работу в отдельном потоке. Его работа заключается в приеме и отправке

сообщений. Для синхронизации отправки сообщений он использует класс `CurrentLinkedQueue`, который представляет из себя потокобезопасную очередь.

В данном классе определены следующие методы:

- `protected AbstractNetworkProcessor(String address)` – конструктор, запоминающий переданный адрес и запускающий поток приема-передачи сообщений;
- `public void stop()` – остановка работы;
- `public void sendData(ZMsg message)` – отправка сообщения в формате библиотеки `ZeroMQ`;
- `public void sendData(String data)` – отправка строки;
- `public void sendData(JSONObject data)` – отправка JSON-объекта;
- `public void sendData(JSONArray data)` – отправка JSON-массива;
- `protected abstract void handleMessage(ZMsg message)` – обработка входящего сообщения. Этот метод абстрактный и должен быть реализован в классах-наследниках;
- `protected abstract ZMQ.Socket socketInitialization(ZContext context, String address)` – инициализация сокета для подключения. Этот метод абстрактный и должен быть реализован в классах-наследниках.

Создание абстрактного класса позволяет реализовать общий функционал один раз, но оставить детали подключения и обработки входящих событий конкретным классам, которые наследуются от класса `AbstractNetworkProcessor`. Существуют два класса, наследующихся от класса `AbstractNetworkProcessor`:

`HostNetworkProcessor` – отвечает за коммуникацию с хостом. При приеме входящего сообщения, оно передается классу `CommandDispatcher`.

`DataChannelNetworkProcessor` – отвечает за передачу данных между девайсом и клиентом. Данный класс реализует прием-передачу `ping`-сообщений для отслеживания активности канала.

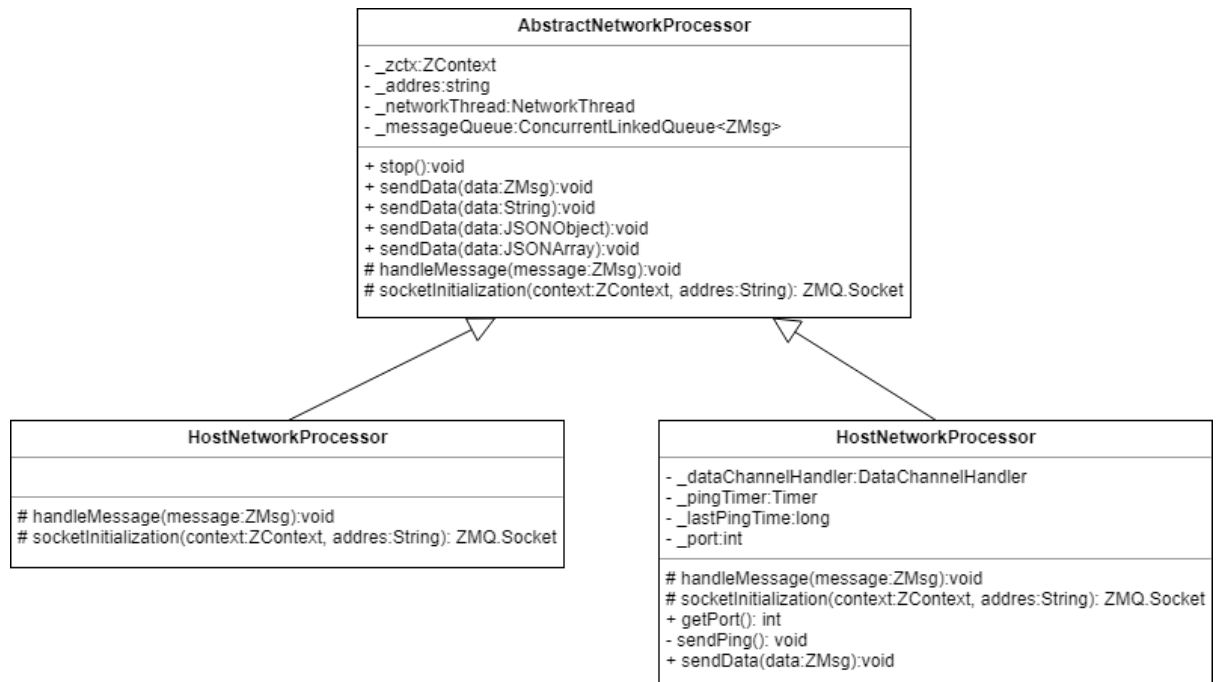


Рисунок 6 – UML-диаграмма AbstractNetworkProcessor класса и его наследников

Обработкой входящих сообщений от хоста занимается класс CommandDispatcher – он получает пакет данных, инспектирует заголовки и выполняет требуемое действие.

- public void receiveData(ZMsg message) – в данный метод поступает входящий пакет данных из сети. Пакет данных разбивается на заголовок и данные сообщения и затем вызывается метод dispatch();
- private void dispatch(String action, JSONObject data) – определяет, какое действие необходимо совершить, исходя из заголовка;
- private void actionRequestDeviceDescriptions(JSONObject data) – отправляет хосту описания всех устройств;
- private void actionRequestDevices(JSONObject data) – отправляет хосту данные устройств;

- `private void actionInvokeDeviceMethod(JSONObject data)` – инициирует вызов метода устройства;
- `public void sendAction(String action, Object data)` – отправка ответа хосту;
- `public void sendDeviceEvent(String deviceID, String event)` – отправка информации о произошедшем событии у устройства;
- `public void sendSuccessMethodInvocationResult(long invocationID, BaseParam result)` – отправка результата вызова метода;
- `public void sendFailureMethodInvocationResult(long invocationID, String error)` – отправка ошибки вызова метода.

### **3.7. ОПИСАНИЕ УСТРОЙСТВ**

Абстрактный класс `Device` служит базой для реализации классов конкретных устройств. Он содержит информацию об устройстве: идентификатор, имя, модель устройства, онлайн-статус и свойства; и объект `DeviceDescription`, описывающий функционал устройств. В нем содержатся следующие методы:

- `public DeviceStat getDeviceStat()` – возвращает объект, содержащий данные устройства;
- `public void setDeviceStat(DeviceStat deviceStat)` – устанавливает новые данные устройства;
- `public String mapNotificationToEvent(String notification)` – транслирует текст push-уведомления в событие устройства;
- `public String getDeviceType()` – возвращает тип;
- `public String getDeviceID()` – возвращает идентификатор;
- `public String getDeviceName()` – возвращает имя;
- `public boolean isDeviceOnline()` – возвращает онлайн-статус;

- `public void invokeMethod(String methodName, long invocationID, JSONArray params)` – выполняет динамический вызов метода устройства путем поиска ссылки на метод в объекте `DeviceDescription` и вызова через рефлексию;
- `public JSONObject toJSON()` – генерирует JSON-представление устройств;
- `public JSONObject getProps()` – возвращает свойства устройства.

В листинге 5 представлен код метода динамически вызывающего методы устройств. Его аргументами являются имя метода, идентификатор вызова и сериализованный массив аргументов метода. Сначала находится описание метода по имени, затем, если описание найдено, происходит проверка количества ожидаемых аргументов с действительным количеством аргументов. Если все проверки пройдены, то создается объект класса `MetaParam`, который содержит идентификатор вызова и `callback` для завершения вызова метода. Затем, аргументы метода десериализуются, записываются в массив и происходит вызов метода через рефлексию.

#### Листинг 5 – Метод динамического вызова методов устройств

```
public void invokeMethod(String methodName, long invocationID, JSONArray params)
throws
    JSONException, IllegalAccessException, InvocationTargetException,
    BaseParam.ParamCastException
{
    DeviceMethodDescription methodDescription =
_deviceDescription.getMethod(methodName);
    if(methodDescription == null)
    {
        Log.e(TAG, "got null method description for method " + methodName + " of
device type " + getDeviceType());
        return;
    }

    if(methodDescription.params.size() != params.length())
    {
        Log.e(TAG, "different params count for method " + methodName + " of
device type " + getDeviceType());
        return;
    }

    Object finalParams[] = new Object[methodDescription.params.size() + 1];
```

```

    finalParams[0] = new MetaParam(invocationID, new
InvocationCallback(invocationID, methodDescription.resultType));

    for (int i = 0; i < methodDescription.params.size(); i++)
    {
        MethodParamDescription paramDescription =
methodDescription.params.get(i);
        Object rawParamObject = params.get(i);

        BaseParam param =
paramDescription.type.newInstanceFromJSONObject(rawParamObject);
        finalParams[i + 1] = param;
    }

    methodDescription.method.invoke(this, finalParams);
}

```

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 определяется строкой из push-уведомления и соответствующим именем события; список методов, где каждый метод имеет имя, тип возвращаемого значения, список именованных параметров и ссылку на метод. Также данный класс имеет метод, позволяющий сгенерировать описание устройства в JSON-формате. Объекты класса `DeviceDescription` генерируется автоматически из классов устройств и позволяют автоматически генерировать интерфейсы к устройствам на стороне клиента.

Для автоматической генерации описания устройств из кода, используются аннотации. Аннотаций в Java являются метки в коде, описывающими метаданные для функции/класса/пакета, также они могут хранить данные, к которым можно получить доступ во время работы программы. Для описания устройств используются следующие аннотации:

- `"DeviceProps"` – является контейнером для описаний свойств устройства;
- `"DeviceProp"` – описание свойства. Содержит имя свойства, тип свойства и документацию для свойства;

- "DeviceEvents" – является контейнером для описания событий устройства;
- "DeviceEvent" – описание события. Содержит имя события, соответствующий текст из push-уведомления и документацию для события;
- "DeviceMethod" – данные аннотации должны быть помечены все методы, которые может вызывать клиент. Содержат имя метода и тип возвращаемого значения;
- "ParamName" – задает имя параметра метода. Это необходимо, так как Java не сохраняет имена параметров методов, соответственно эту информацию нельзя получить во время исполнения программы через рефлексию;
- "Doc" – добавляет документацию к классу устройства и методам.

В листинге 6 показан пример описания устройства на языке Java. Данный код одновременно реализует функционал устройства и служит основой для генерации описания этого устройства.

## Листинг 6 – Пример описания устройства

```
@DeviceProps({
    @DeviceProp(name = "is_pressed", type = BoolParam.class, doc = "True, if
button is pressed")
})
@DeviceEvents({
    @DeviceEvent(name = "on_click", notificationText = "Click", doc = "Emits
when button is clicked"),
    @DeviceEvent(name = "on_double_click", notificationText = "DoubleClick",
doc = "Emits when button is doubleclicked")
})
@Doc("Documentation for class")
public class SomeDevice extends Device
{
    @Doc("Turn devices on\off")
    @DeviceMethod(name = "set_power")
    public void setPower(MetaParam meta, @ParamName("value") BoolParam value)
    {
        // реализация метода
    }
}
```

Автоматической генерацией описания устройств занимается класс DeviceRegistry. Также в его задачи входит создание объекта устройства исходя из модели устройства. В нем присутствуют следующие методы:

- public static Device createDeviceByModel(DeviceStat deviceStat) – создает объект устройства нужной модели;
- public static DeviceDescription getDescriptionForDeviceType(String deviceType) – возвращает описание устройства для конкретной модели;
- public static JSONArray getDeviceDescriptions() – возвращает описание всех устройств в виде JSON-массива;
- private static void generateDeviceDescriptions() – генерирует описания для всех устройств в системе;
- private static DeviceDescription generateDeviceDescription(String type, Class<? extends Device> deviceClass) – генерирует описание устройства из конкретного класса;
- private static List<DeviceMethodDescription> generateDeviceMethodDescriptions(Class<? extends Device> deviceClass) – генерирует описания всех методов для конкретного класса;



- private static DeviceMethodDescription  
generateDeviceMethodDescription(String name, Class<? extends BaseParam>  
resultType, Method method) – генерирует описание конкретного метода.

### 3.8. ПОДДЕРЖИВАЕМЫЕ УСТРОЙСТВА НА ДАННЫЙ МОМЕНТ

На данный момент существуют реализации следующих типов устройств:

- "YeelightLightColor1" – RGB лампа. Поддерживаются методы включения и выключения, установка цвета;
- "ZhimiFanZa1" – умный вентилятор. Поддерживаются методы включения и выключения, установка скорости вращения;
- "LumiSensorSmokeV1" – датчик дыма. Поддерживается единственное возможное событие – определение дыма;
- "LumiSensorSwitchV2" – умная кнопка. Поддерживаются события нажатия, двойного нажатия и долгого нажатия;
- "LumiSensorCubeAQGL01" – умный куб. Поддерживаются события переворачивания на 90 градусов, переворачивание на 180 градусов, поворота, тряски, толкания и двойного постукивания;
- "ChuangmiCameraXiaobai" – камера. Поддерживаются методы подключения к камере, трансляции видеопотока и поворотов камеры.

Существует проблема регистрации событий устройств – информация о событиях устройств не поступает в приложение MI Home. Эта проблема была решена таким образом: для каждого устройства в приложении MI Home была создана автоматизация, которая при наступлении события отправляет push-уведомление на Android-смартфон. Данное push-уведомление регистрируется классом DeviceNotificationListenerService, внедренным в приложение MI Home с

помощью инъекции кода, и передается модификации, где сопоставляется с устройством и затем информация о событии передается хосту.

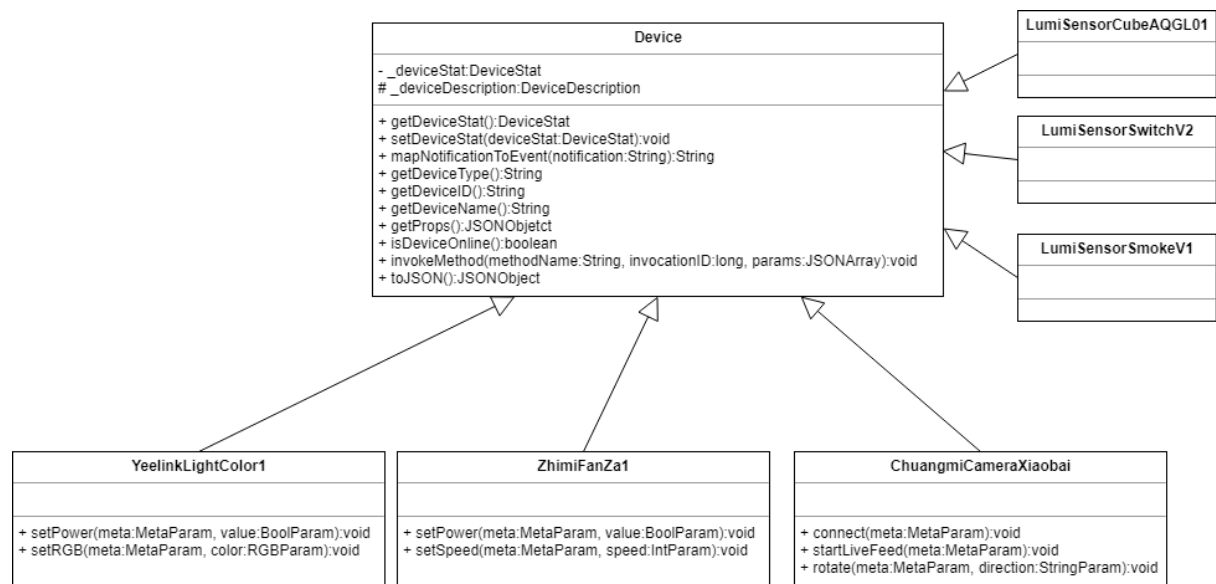


Рисунок 7 – UML-диаграмма классов устройств

### 3.9. ИНТЕГРАЦИЯ В РАСШИРЯЕМУЮ СИСТЕМУ ВЗАИМОДЕЙСТВИЯ ИОТ-УСТРОЙСТВ

Для интеграции в расширяемую систему взаимодействия IoT-устройств [1] был создан плагин MIHomePlatform. Задача данного плагина связать модификацию с хостом для предоставления клиенту доступа к устройствам Xiaomi.

Инициализация плагина начинается с класса MIHomePlatformFactory, реализующего интерфейс IPlatformFactory. Метод Create() данного класса извлекает из переданные для инициализации параметров IP-адрес и порт запущенного приложения MI Home. В случае отсутствия данных параметров, инициализация прерывается, иначе метод Create() возвращает объект классу MIHomePlatform.

Класс MIHomePlatform реализующий интерфейс IPlatform, служит для управления плагином. В начале работы происходит подключение по сети к

приложению MI Home и синхронизация описаний и данных устройств. Класс содержит следующие методы:

- `public void StartPlatform(IHost host)` – запуск платформы;
- `public void StopPlatform()` – остановка платформы;
- `public string GetPlatformName()` – возвращает название платформы;
- `public ICollection<DeviceDescription> GetDeviceDescriptions()` – возвращает коллекцию описаний устройств;
- `public ICollection<DeviceData> GetDevices()` – возвращает коллекцию данных устройств;
- `public DeviceData GetDevice(string id)` – поиск устройства по идентификатору;
- `public void InvokeDeviceMethod(string deviceID, string method, IReadOnlyCollection<JToken> serializedMethodParams, IMethodInvocationInstance invocationInstance)` – вызов метода устройства;
- `public void SendDeviceEvent(string deviceID, string eventName, ICollection<object> eventParams)` – отправка информации о произошедшем событии.

Реализация устройств находится не в плагине, а в модификации приложения MI Home, поэтому запросы вызовов методов необходимо транслировать к приложению MI Home. За это отвечает класс `MIProxyDevice`, наследующийся от класса `AbstractDevice`. Реализация метода `InvokeMethod()` отправляет запрос на вызов приложения MI Home, где он и обрабатывается. В случае, если вызываемый метод имеет тип возврата `DataChannelDescription`, то тогда `invocationInstance` вызова оборачивается в класс `ProxyMethodInvocationInstance`, который в случае успешного выполнения метода, создает прокси для потока данных от устройства до клиента.



Рисунок 8 – Схема работы плагина

## 4. РЕАЛИЗАЦИЯ

После внедрения инъекции в приложение MI Home и сборки проекта модификации, пользователь может установить приложение на телефон и поместить APK-модификации в память телефона. После запуска приложения, при наличии установленного плагина MIHomePlatform на хосте, произойдет синхронизация подключенных к приложению MI Home устройств, и они станут доступными для управления через расширяемую систему взаимодействия IoT-устройствами.



Рисунок 9 – Управление Xiaomi-устройствами

В качестве примера работы модифицированного приложения MI Home, был написан скрипт транслирующий видеопоток с камеры Xiaomi на компьютер под управлением ОС Linux. Скрипт записывает видеоданные в именованный канал, откуда они считываются и выводятся на экран с помощью медиаплеера VLC.



Рисунок 10 – Демонстрация работы камеры

На рисунке 11 показан принцип работы расширяемой системы с разработанной модификацией приложения MI Home. Сигнал от кнопки передается на сервера Xiaomi, откуда поступает в виде push-уведомления на Android-смартфон, на смартфоне уведомление преобразуется в триггер события устройства и информация о событии поступает на хост, затем хост передает информацию о событии в пользовательский код, который принимает необходимое действие в качестве реакции на событие. В данном примере пользовательский код решает включить лампу, и запрос вызова посылается на хост, который, в свою очередь, передает запрос вызова приложению MI Home. Приложение посылает команду включения лампы через сервера Xiaomi.



Рисунок 11 – Принцип работы расширяемой системы с разработанной модификацией приложения MI Home

## 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 Xiaomi кафедры ЭВМ. Было проведено функциональное тестирование.

### 5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Тестирование разработанной модификации приложения MI Home проводилось на смартфоне с ОС Android 5.1. Хост расширяемой системы взаимодействия IoT-устройств с плагином MIHomePlatform запущен на компьютере под управлением ОС Linux.

В таблице 1 описано тестирование действия «Вывод списка устройств».

Таблица 1 – Вывод списка устройств.

Свойство	Значение
Выполняемые действия	Написан скрипт выводящий информацию о подключенных в данный момент устройствах.
Ожидаемые результаты	При запуске скрипта должна вывестись информация об устройствах, подключенных к приложению MI Home.
Полученные результаты	После запуска скрипта, в консоли появляется список всех подключенных Xiaomi устройств.



В таблице 2 описано включение лампы.

Таблица 2 – Включение лампы.

Свойство	Значение
Выполняемые действия	Создан скрипт, включающий лампу Xiaomi Yeelight Color E27 Bulb.
Ожидаемые результаты	После запуска скрипта лампа должна включиться.
Полученные результаты	После запуска написанного скрипта, происходит включение лампы.

В таблице 3 описано тестирование кнопки Xiaomi.

Таблица 3 – тест вызова метода устройства

Свойство	Значение
Выполняемые действия	Написан пользовательский скрипт, который при нажатии на умную кнопку, включает лампу и задает синий цвет.
Ожидаемые результаты	При нажатии на кнопку, включается лампа с синим цветом.
Полученные результаты	После запуска пользовательского скрипта и нажатия на умную кнопку, происходит включение лампы с синим цветом.

В таблице 4 описано тестирование работы камеры.

Таблица 4 – Тест работы камеры.

Свойство	Значение
Выполняемые действия	Написан пользовательский скрипт, который выводит видеотрансляцию с камеры на экран компьютера.
Ожидаемые результаты	Видеопоток с камеры показывается на компьютере.
Полученные результаты	После запуска скрипта, видеопоток с камеры передается на компьютер. В плеере VLC идет трансляция.

Как видно из тестов, разработанное расширение работает корректно и позволяет пользователю взаимодействовать с IoT-устройствами Xiaomi.

## 6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для работы с устройствами Xiaomi, необходимо выполнить следующие шаги:

1. Произвести инъекцию кода в приложение MI Home и установить приложение с инъекцией на телефон.
2. Скомпилировать модификацию и поместить получившийся APK-файл на Android-смартфон по пути /sdcard/mihome\_injection/injection.apk.
3. Установить плагин MIHomePlatform на хост.
4. Запустить хост, передав в качестве параметров настройки для MIHomePlatform: “mihome:app\_ip:IP mihome:app\_port:PORT”.
5. Запустить приложение MI Home, добавить необходимые устройства.
6. Для работы событий устройств:

- a. Переименовать устройство, имя устройства должно соответствовать идентификатору устройства.
  - b. Для всех событий устройства, создать автоматизацию, отправляющую push-уведомление на смартфон.
7. Для взаимодействия с устройствами, приложение должно быть включено постоянно.

## 7. ЗАКЛЮЧЕНИЕ

В ходе выполнения магистерской диссертации был проведен анализ существующих программных решений для взаимодействия IoT-устройствами Xiaomi: выделены их недостатки и был предложен принципиально новый подход к решению проблемы. В отличие от аналогов, разработанное решение может поддерживать все устройства экосистемы Xiaomi, а также реализует работу с событиями устройств. Была произведена интеграция в расширяемую систему взаимодействия IoT-устройствами, что позволяет писать гибкие скрипты автоматизации на языке общего назначения и интегрировать IoT-устройства Xiaomi с устройствами других экосистем.

Данную интеграцию можно использовать для обучения студентов работе с одной из самых быстрорастущих систем «Интернета вещей» – Xiaomi. Данным оборудованием оснащена кафедра электронных вычислительных машин, поэтому предложенное решение позволит обучать студентов и проводить лабораторные работы на базе кафедры. Также данную интеграцию можно использовать в постройке «умного дома», так как устройства от Xiaomi стоят недорого относительно конкурентов и очень популярны для данного направления.

По итогам работы, можно сделать следующие выводы:

- произведена модификация стандартного приложения MI Home;
- создано расширение, позволяющее удаленно управлять устройствами Xiaomi;
- реализован плагин для расширяемой системы взаимодействия с IoT-устройствами;
- 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 – крупнейшая платформа интернета вещей. – <https://ichip.ru/xiaomi-sozdala-krupnejjshtuyu-v-mire-platformu-interneta-veshhejj.html>. Дата обращения: 20.02.2019.
3. Суханов, К.Д. Разработка интерфейса системы расширения функционального взаимодействия компонентов интернета вещей: дис. магистра: утв. 01.06.2019 / К.Д. Суханов. – 2019. – 47 с.
4. Интернет вещей. – <https://www.ibs.ru/datalab/works/internet-veshchey-iot/>. Дата обращения: 10.12.2018.
5. Портирование python-miio и js-miio. – <https://connect.smartliving.ru/tasks/40.html>. 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](http://majordomo.smartliving.ru). Дата обращения: 17.02.2019.
9. openHAB. – <https://www.openhab.org>. Дата обращения: 19.02.2019.
10. Реализация взаимодействия с устройствами из экосистемы xiaomi по протоколу miIO - [skysilver-lab/php-miio](https://github.com/skysilver-lab/php-miio). – <https://github.com/skysilver-lab/php-miio>. Дата обращения: 17.02.2019.
11. Работа с Xiaomi Mi Home. – <https://www.ixbt.com/live/smarthome/rabota-s-xiaomi-mi-home-lichnyy-opyt-nastroyka-nyuansy.html>. Дата обращения: 21.02.2019.

12. Akgul, F. ZeroMQ / F. Akgul. – Бирмингем: Packt Publishing, 2011. – 140 с.
13. Кристин, М. Android. Программирование для профессионалов / М. Кристин. – Санкт-Петербург: Питер, 2017. – 688 с.
14. Schildt, H. Java: The Complete Reference / H. Schildt. – Нью-Йорк: McGraw-Hill Education, 2018. – 1248 с.
15. Forman, I. Java Reflection in Action / I. Forman. – Нью-Йорк: Manning Publications, 2014. – 300 с.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл MIHomePlatform.cs

```
using System;
using System.CodeDom;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Threading;
using IOT.Common;
using IOTHost;
using IOTHost.PlatformInterface;
using Newtonsoft.Json.Linq;

namespace MihomePlatform
{
    public class MihomePlatform : IPlatform
    {
        private const string TAG = "Mihome";
        internal const string PlatformName = "Mihome";
        private IHost _host;
        internal MiAppNetworkProcessor AppNetworkProcessor;
        internal MiAppMessageProcessor MessageProcessor;
        internal MihomeApi Api;
        internal MiAppInvocationManager InvocationManager;
        internal DataChannelProxyManager DataChannelProxyManager;
        internal readonly IDictionary<string, DeviceDescription> DeviceDescriptions
            = new Dictionary<string, DeviceDescription>();
        internal readonly IDictionary<string, DeviceData> DeviceData = new
Dictionary<string, DeviceData>();
        internal readonly Dictionary<string, AbstractDevice> Devices = new
Dictionary<string, AbstractDevice>();
        internal string MiAppIP;
        internal int MiAppPort;

        internal MihomePlatform(string appIP, int appPort)
        {
            MiAppIP = appIP;
            MiAppPort = appPort;
        }

        public void StartPlatform(IHost host)
        {
            _host = host;
            MessageProcessor = new MiAppMessageProcessor(this);
            AppNetworkProcessor = new MiAppNetworkProcessor(this,
$"tcp://{MiAppIP}:{MiAppPort}");
            InvocationManager = new MiAppInvocationManager(AppNetworkProcessor);
            Api = new MihomeApi(AppNetworkProcessor, InvocationManager);
        }
    }
}
```

```

        DataChannelProxyManager = new DataChannelProxyManager(this);
        Logger.Log("start");

        Api.RequestDeviceDescriptions();
        Api.RequestDevices();
    }

    public void StopPlatform()
    {
        AppNetworkProcessor.Stop();
        MessageProcessor.Stop();
        DataChannelProxyManager.Dispose();
        Logger.Log("stop");
    }

    public void Dispose()
    {
        StopPlatform();
    }

    public string GetPlatformName()
    {
        return PlatformName;
    }

    public ICollection<DeviceDescription> GetDeviceDescriptions()
    {
        return DeviceDescriptions.Values;
    }

    public ICollection<DeviceData> GetDevices()
    {
        return DeviceData.Values;
    }

    public DeviceData GetDevice(string id)
    {
        return DeviceData.TryGetValue(id, out var device) ? device : null;
    }

    public void InvokeDeviceMethod(string deviceID, string method,
        IReadOnlyCollection<JToken> serializedMethodParams, IMethodInvocationInstance
        invocationInstance)
    {
        if(Devices.TryGetValue(deviceID, out var device))
            device.InvokeMethod(method, serializedMethodParams, invocationInstance);
        else
        {
            Logger.Log($"no device found with id {deviceID}");
            invocationInstance.Failed($"no device found with id {deviceID}");
        }
    }

    public void SendDeviceEvent(string deviceID, string eventName,
        ICollection<object> eventParams)

```



```

        {
            _host.ProcessDeviceEvent(GetPlatformName(), deviceID, eventName,
eventParams);
        }
    }
}

```

## Файл MIHomePlatformFactory.cs

```

using System;
using System.Collections.Generic;
using IOTHost.PlatformInterface;

namespace MihomePlatform
{
    public class MihomePlatformFactory : IPlatformFactory
    {
        public IPlatform Create(out string error, Dictionary<string, string> options)
        {
            if (!options.TryGetValue("app_ip", out var appIP))
            {
                error = "no app_ip option present";
                return null;
            }

            if (!options.TryGetValue("app_port", out var appPortString))
            {
                error = "no app_port option present";
                return null;
            }

            if (!int.TryParse(appPortString, out var appPort))
            {
                error = "app_port must be integer";
                return null;
            }

            error = null;
            return new MihomePlatform(appIP, appPort);
        }

        public string GetPlatformName()
        {
            return MihomePlatform.PlatformName;
        }
    }
}

```

## Файл MIProxyDevice.cs

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

namespace MihomePlatform
{
    public class MiProxyDevice : AbstractDevice
    {
        private readonly MihomePlatform _platform;

        public MiProxyDevice(MihomePlatform platform, DeviceDescription description,
            DeviceData data)
            : base(description, data)
        {
            _platform = platform;
        }

        private class ProxyMethodInvocationInstance : IMethodInvocationInstance
        {
            private readonly IMethodInvocationInstance _originalInvocationInstance;
            private readonly MiProxyDevice _device;

            public ProxyMethodInvocationInstance(IMethodInvocationInstance
            invocationInstance, MiProxyDevice device)
            {
                _originalInvocationInstance = invocationInstance;
                _device = device;
            }

            public void Completed(object result)
            {
                var dataChannelDataType = new DataChannelDescriptionConverter();
                var miAppDataChannel = (DataChannelDescription)dataChannelDataType.JsonToValue((JToken)result);
                _device._platform.DataChannelProxyManager.CreateProxy(miAppDataChannel,
                description =>
                _originalInvocationInstance.Completed(dataChannelDataType.ValueToJson(description)));
            }

            public void Failed(string error)
            {
                _originalInvocationInstance.Failed(error);
            }

            public void TimedOut()
            {
                _originalInvocationInstance.TimedOut();
            }
        }
    }
}
```

```

    }
}

public override void InvokeMethod(string method, IReadOnlyCollection<JToken>
serializedMethodParams, IMethodInvocationInstance invocationInstance)
{
    if (!Description.Methods.TryGetValue(method, out var methodDescription))
    {
        Logger.Log($"no method {method} found");
        invocationInstance.Failed($"no method {method} found");
        return;
    }

    if(methodDescription.ResultConverter is DataChannelDescriptionConverter)
        _platform.Api.InvokeDeviceMethod(Data.ID, method,
serializedMethodParams, new ProxyMethodInvocationInstance(invocationInstance, this));
    else
        _platform.Api.InvokeDeviceMethod(Data.ID, method,
serializedMethodParams, invocationInstance);
}
}
}
}

```

## Файл MIHomeAPI.cs

```

using System.Collections.Generic;
using System.Dynamic;
using IOT.Common;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace MihomePlatform
{
    public class MihomeApi
    {
        private readonly MiAppNetworkProcessor _networkProcessor;
        private readonly MiAppInvocationManager _invocationManager;

        public MihomeApi(MiAppNetworkProcessor networkProcessor, MiAppInvocationManager
invocationManager)
        {
            _networkProcessor = networkProcessor;
            _invocationManager = invocationManager;
        }

        public void RequestDeviceDescriptions()
        {
            _networkProcessor.SendAction("request:get_device_descriptions", null);
        }

        public void RequestDevices()

```

```

        {
            _networkProcessor.SendAction("request:get_devices", null);
        }

        public void InvokeDeviceMethod(string deviceID, string method,
IEnumerable<JToken> serializedMethodParams, IMethodInvocationInstance
invocationInstance)
        {
            _invocationManager.InvokeDeviceMethod(deviceID, method,
serializedMethodParams, invocationInstance);
        }
    }
}

```

## Файл MIAppNetworkProcessor.cs

```

using System;
using System.Collections.Generic;
using IOT.Common;
using Newtonsoft.Json;
using ZeroMQ;

namespace MihomePlatform
{
    public class MiAppNetworkProcessor : AbstractNetworkProcessor
    {
        private MihomePlatform _platform;

        public MiAppNetworkProcessor(MihomePlatform platform, string address)
            : base(address)
        {
            _platform = platform;
        }

        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)
        {
            _platform.MessageProcessor.HandleMessage(message);
        }
    }
}

```

## Файл MIAppMessageProcessor.cs

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

namespace MihomePlatform
{
    public class MiAppMessageProcessor : AbstractMessageProcessor<JObject>
    {
        private const string TAG = "MiAppMessageProcessor";
        private readonly MihomePlatform _platform;

        public MiAppMessageProcessor(MihomePlatform platform)
        {
            _platform = platform;
        }

        public void HandleMessage(ZMessage message)
        {
            PutMessage(JObject.Parse(message[0].ReadString()));
        }

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

            switch (action.ToLower())
            {
                case "result:get_device_descriptions":
                    UpdateDeviceDescriptions(data);
                    break;
                case "result:get_devices":
                    UpdateDevices(data);
                    break;
                case "result:device_invoke_method":
                    DeviceMethodResult(data);
                    break;
                case "device_event":
                    ProcessDeviceEvent(data);
                    break;
                default:
                    Logger.Log($"unrecognized action '{action}'");
                    break;
            }
        }

        private void UpdateDeviceDescriptions(JToken data)
        {

```

```

        if (!data.Type.Equals(JTokenType.Array))
        {
            Logger.Log("update_device_descriptions should have array data");
            return;
        }

        _platform.DeviceDescriptions.Clear();

        foreach (var json in (JArray)data)
        {
            var deviceDescription = DeviceDescription.LoadFromJson(json);
            if(deviceDescription != null)
                _platform.DeviceDescriptions.Add(deviceDescription.Type,
deviceDescription);
        }

        private void UpdateDevices(JToken data)
        {
            if (!data.Type.Equals(JTokenType.Array))
            {
                Logger.Log("update_devices should have array data");
                return;
            }

            _platform.DeviceData.Clear();
            _platform.Devices.Clear();

            foreach (var json in (JArray)data)
            {
                try
                {
                    if(!DeviceData.ReadPlatformAndType(json, out var platform, out var
type))
                        continue;

                    if(platform != _platform.GetPlatformName())
                        continue;

                    if (!_platform.DeviceDescriptions.TryGetValue(type, out var
description))
                    {
                        Logger.Log($"no description for device type {type}");
                        continue;
                    }

                    var deviceData = DeviceData.CreateFromJson(json, description);

                    _platform.DeviceData.Add(deviceData.ID, deviceData);
                    _platform.Devices.Add(deviceData.ID, new MiProxyDevice(_platform,
description, deviceData));
                }
                catch (Exception e)
                {
                    Logger.Log(e.ToString());
                }
            }
        }

```

```

    }
}

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

    switch (status)
    {
        case "success":
            var result = (object) data["result"];
            _platform.InvocationManager.ProcessInvocationResult(invocationID,
result);

            break;
        case "failure":
            var error = (string) data["error"];
            _platform.InvocationManager.ProcessInvocationError(invocationID,
error);

            break;
        default:
            Logger.Log($"unknown type of invocation result: {status}");
            break;
    }
}

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

    _platform.SendDeviceEvent(deviceID, eventName, new List<object>());
}
}
}
}

```

## Файл MIAppInvocationManager.cs

```

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

namespace MihomePlatform
{
    public class MIAppInvocationManager : AbstractInvocationManager
    {
        private readonly AbstractNetworkProcessor _networkProcessor;
    }
}

```

```

        public MiAppInvocationManager(AbstractNetworkProcessor networkProcessor)
        {
            _networkProcessor = networkProcessor;
        }

        public void InvokeDeviceMethod(string deviceID, string method,
IEnumerable<JToken> serializedMethodParams, IMethodInvocationInstance
methodInvocationInstance)
        {
            var invocationID = NewInvocationID();
            InvocationsInProgress.TryAdd(invocationID, methodInvocationInstance);

            var data = new Dictionary<string, object>
            {
                {"invocation_id", invocationID},
                {"device_id", deviceID},
                {"method", method},
                {"params", serializedMethodParams}
            };

            var message = new Dictionary<string, object>
            {
                {"action", "request:device_invoke_method"},
                {"data", data}
            };

            var zmessage = new ZMessage { new
ZFrame(JsonConvert.SerializeObject(message)) };
            _networkProcessor.SendMessage(zmessage);
        }
    }
}

```

## Файл DataChannelProxyManager.cs

```

using System;
using System.Collections.Generic;
using System.Threading;
using System.Timers;
using IOT.Common;
using IOT.Common.DataTypes;
using IOTHost;
using Timer = System.Timers.Timer;

namespace MihomePlatform
{
    public class DataChannelProxyManager : IDisposable
    {
        private readonly MihomePlatform _platform;
    }
}

```



```

        private readonly List<DataChannelProxy> _activeProxies = new
List<DataChannelProxy>();
        private readonly Timer _destroyInactiveProxiesTimer = new Timer(2000);

        public DataChannelProxyManager(MihomePlatform platform)
        {
            _platform = platform;

            _destroyInactiveProxiesTimer.Elapsed += OnDestroyInactiveProxiesTimerEvent;
            _destroyInactiveProxiesTimer.AutoReset = true;
            _destroyInactiveProxiesTimer.Enabled = true;
        }

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

        public void CreateProxy(DataChannelDescription dataChannelDescription,
Action<DataChannelDescription> proxyReady)
        {
            var proxy = new DataChannelProxy(_platform.MiAppIP,
dataChannelDescription.Port, p =>
            {
                var newDescription = new DataChannelDescription("", p.GetClientPort());
                proxyReady(newDescription);
            });
            lock (_activeProxies)
                _activeProxies.Add(proxy);
        }

        private void OnDestroyInactiveProxiesTimerEvent(object sender, ElapsedEventArgs
args)
        {
            var threshold = TimeSpan.FromSeconds(5);
            int removedCount;
            lock (_activeProxies)
            {
                foreach (var proxy in _activeProxies)
                    if(!proxy.IsActive(threshold))
                        proxy.Stop();
                removedCount = _activeProxies.RemoveAll(proxy =>
!proxy.IsActive(threshold));
            }
            if(removedCount > 0)
                Logger.Log($"removed {removedCount} proxies");
        }
    }
}

```

## Файл DataChannelProxy.cs

```
using System;
using System.Threading;
using IOHost;
using ZeroMQ;

namespace MihomePlatform
{
    public class DataChannelProxy
    {
        private string _miAppIP;
        private int _miAppPort;
        private Thread _proxyThread;
        private int _clientPort = -1;
        private Action<DataChannelProxy> _whenProxyIsReady;
        private volatile bool _stopped;
        private DateTime _lastMiAppPing = DateTime.UtcNow, _lastClientPing =
DateTime.UtcNow;

        public DataChannelProxy(string miAppIp, int miAppPort, Action<DataChannelProxy>
whenProxyIsReady)
        {
            _miAppIP = miAppIp;
            _miAppPort = miAppPort;
            _whenProxyIsReady = whenProxyIsReady;
            _proxyThread = new Thread(ProxyProcessor);
            _proxyThread.IsBackground = true;
            _proxyThread.Start();
        }

        public int GetClientPort()
        {
            return _clientPort;
        }

        public void Stop()
        {
            _stopped = true;
        }

        public bool IsActive(TimeSpan threshold)
        {
            var now = DateTime.UtcNow;
            return !_stopped && now - _lastMiAppPing < threshold && now -
_lastClientPing < threshold;
        }

        private void ProxyProcessor()
        {
            using (var context = new ZContext())
            using (var miAppSocket = new ZSocket(context, ZSocketType.PAIR))
            using (var clientSocket = new ZSocket(context, ZSocketType.PAIR))
            {
```

```

miAppSocket.Connect($"tcp://{_miAppIP}:{_miAppPort}");
clientSocket.Bind("tcp://*:*");

_clientPort = ParsePort(clientSocket.LastEndpoint);
_whenProxyIsReady(this);
_whenProxyIsReady = null;

var sockets = new[] { miAppSocket, clientSocket };
var pollItems = new[] { ZPollItem.Create(ReceiveMessage),
ZPollItem.Create(ReceiveMessage) };

while (!_stopped)
{
    if (!sockets.PollIn(pollItems, out var messages, out var error,
    TimeSpan.FromMilliseconds(100)))
        continue;

    if(messages[0] != null)
        ProxyMessage(messages[0], clientSocket, ref _lastMiAppPing);
    if(messages[1] != null)
        ProxyMessage(messages[1], miAppSocket, ref _lastClientPing);
}
}

private static bool ReceiveMessage(ZSocket socket, out ZMessage message, out
ZError error)
{
    message = socket.ReceiveMessage(ZSocketFlags.DontWait, out error);
    return message != null;
}

private static void ProxyMessage(ZMessage message, ZSocket dst, ref DateTime
lastPing)
{
    var type = message[0].ReadString();
    if(type == "ping")
        lastPing = DateTime.UtcNow;
    dst.SendMessage(message);
}

private static int ParsePort(string endpoint)
{
    var port = endpoint.Substring(endpoint.LastIndexOf(":",
StringComparison.Ordinal) + 1);
    return int.Parse(port);
}
}
}

```