INSTITUT SUPERIEUR INFORMATIQUE ISI
المعهد العالي للإعلامية

Sofiatech

Sofiatech Summer Internship Report

Matter controller based on Linux

Prepared by:

Moktar SELLAMI

Professional Supervisor:

Ahmed DAOUD

from 1/7/2024 to 31/8/2024

Academic Year: 2023-2024

Table of Contents

1. Objective: Matter controller based on linux.

1.1. Goal of the Project

Create an application on Linux (Qt cpp, python), that play the role of a matter controller:

- Controls the devices that are locally connected over WiFi.
- Performs device Bluetooth LE commissioning.

1.2. Technologies Involved

Technologies: matter

Programming languages: cpp, Qt

build systems: Ninja, gn makefile

1.3. What i did

Through many approches, ideas, i ended up creating a simple Graphical Application using Qt (cpp) that relyes on the chip-tool CLI to test the linux lighting app example.

- Discover devices over WiFi
- automaticallty detect device category
- See device charachteristics
- Commission/decommession a device (onnetwork)
- Light control (Toggle light)
- Debug Terminal

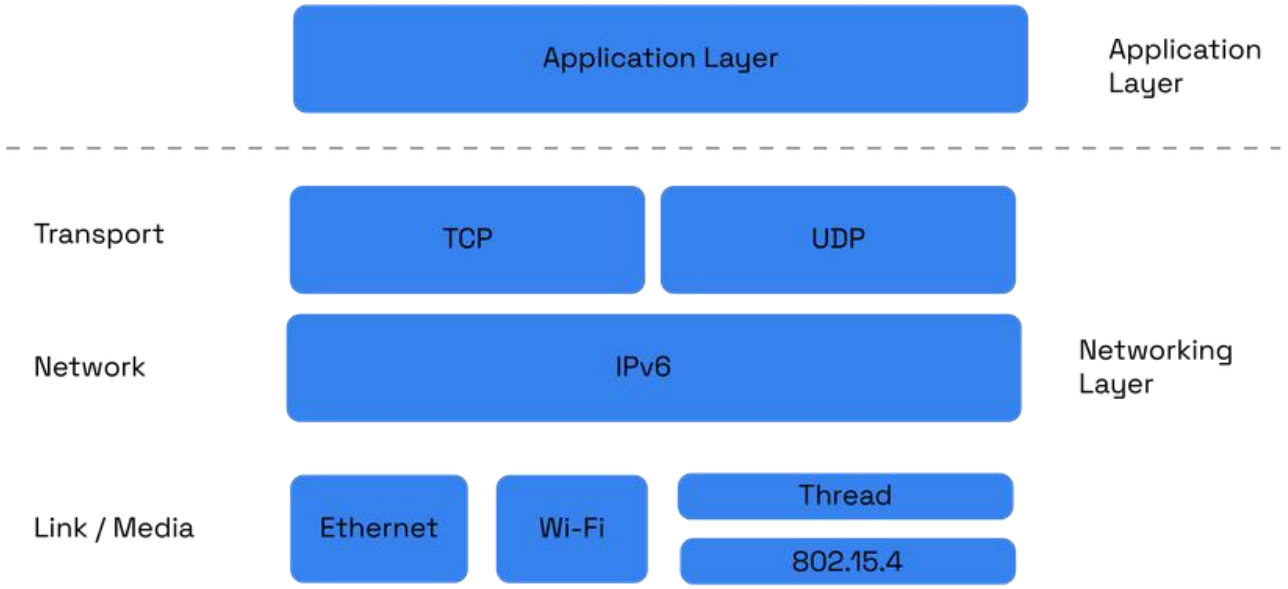
I will explore this in a later section.

2. Matter Overview

2.1. What is a Matter

Matter is a freely available connectivity standard for smart home and IoT devices.
IT aims to improve interoperability and compatibility between different manufacturer and security, and always allowing local control as an option.

Matter originated in December 2019 as the Project Connected Home over IP (or CHIP for short) working group, founded by Amazon, Apple, Google and the Zigbee Alliance, now called the Connectivity Standards Alliance (CSA).

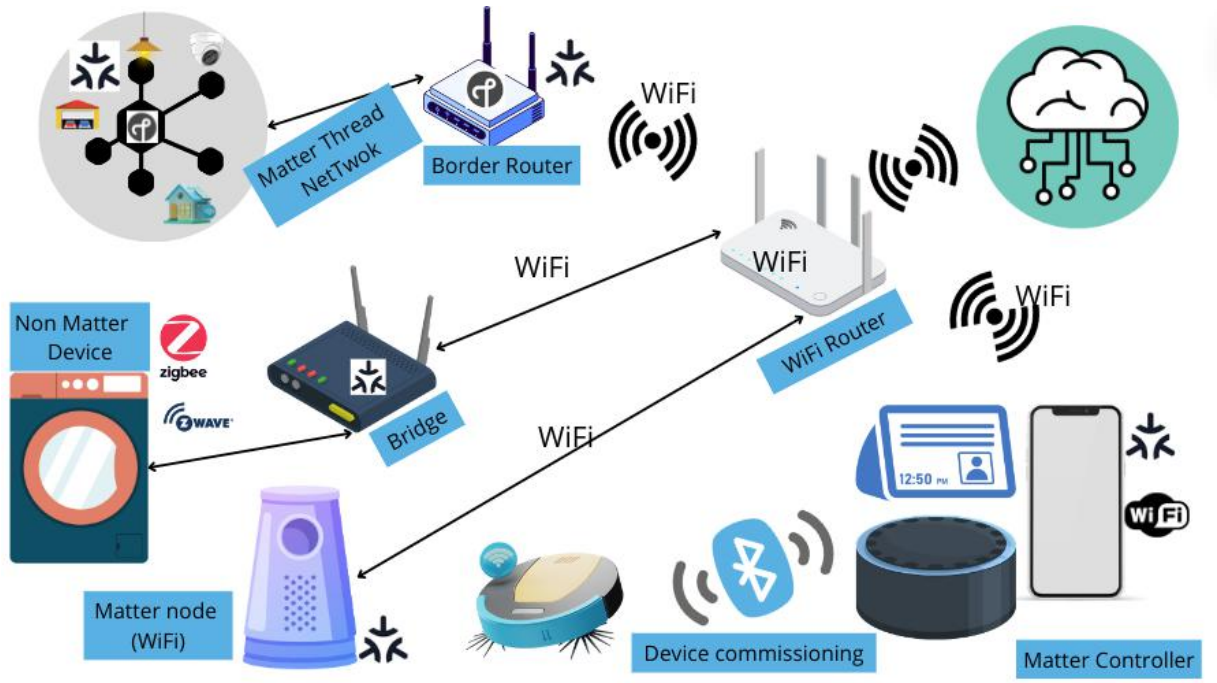


Matter runs on top of:

- Threads: an IPv6-based, low-power mesh networking tech for IoT products
- Wi-Fi
- Ethernet
- Bluetooth low energy for commissioning

The commissioning profile is an example of a proprietary profile that can be used to add nodes to a network using Bluetooth (BLE)
To learn more visit the CSA website.

2.2. Matter architecture



- **Matter Controller**: This is represented by a smartphone and is used to commission (set up) and control all the Matter-enabled devices in the network.
- **WiFi Router**: The devices in the home connect to each other over WiFi using this router, forming the central hub for communication.
- **Matter Node (WiFi)**: Devices like the smart speaker and robot vacuum are directly connected to the Matter network through WiFi.
- **Bridge**: The bridge allows non-Matter devices, like those using Zigbee or Z-Wave (such as the washing machine a **non Matter device**), to communicate with the Matter network. This way, the Matter controller can manage them even though they aren't natively Matter-compatible.
- **Border Router**: This device connects the local Matter network to the internet and external services, allowing remote control. It also handles the **Matter Thread Network**, which is a type of low-power mesh network for connected devices.
- **Device Commissioning**: Bluetooth is used here to pair new devices with the network via the controller (shown by the phone). This is the process of adding a new device to the system.
- **Cloud Connection**: The system is linked to the cloud, enabling control and monitoring of devices even when you're away from home.

2.3. Matter data model

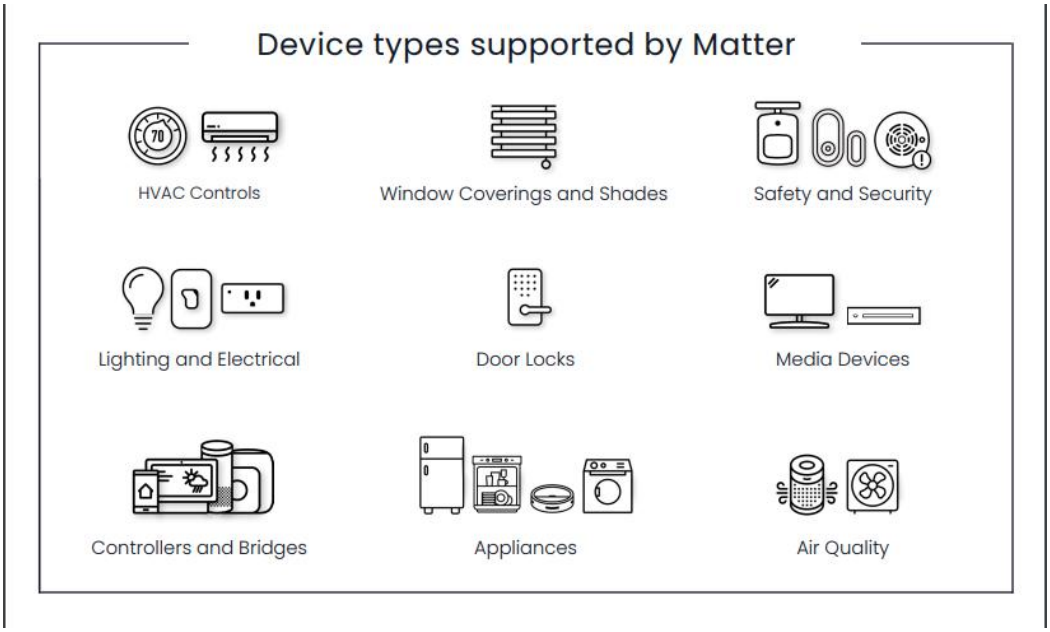
The Matter data model is structured around devices, clusters, attributes, and commands:

- Devices: Represent physical objects (e.g., lights, thermostats) that are identified and controlled within the Matter network.
- Clusters: Group related functionality into logical components. For example, a "Lighting" cluster might control on/off, brightness, and color settings.
- Attributes: Store the state of a device (e.g., the on/off status of a light). Attributes are readable or writable values within clusters.
- Commands: Actions that can be triggered on devices (e.g., turning a light on or off).

2.4. Device types supported by matter

Each IoT device in your home belongs to a certain type/category that is set by the matter specification.

An IoT lamp might fall un der the lighting devices category.



.

You can learn more by reading the in the matter device library specification

3. Matter controller

3.1. Overview Matter controller

Controllers are responsible for controlling Matter devices in a smart Home.
It's the brain of the system
Controller functionality can be built into many devices, it's possible to have many controllers (example: Amazon echo and nested Hub)
A matter controller could also be a smartphone app, dashbpard or desktop app.

3.2. What does a controller do:

- Control a Matter device by sending commands.(onoff, brightness control...)
- Receive Data from matter devices.
- Device commissioning via Bluetooth.
- Perform Over the air updates (add a way to perform update the software on the controller).

4. My Solutions

I spend the first period of the Internship understanding the Matter protocole, testing the chip-tool (on raspberry pi and on my laptop) with multip virtual devices.

The second period is where i started seaching how to develop the Matter controller, I explored several possible solutions. Below is a summary of each approach, the rationale behind it, and the outcomes.

4.1. Approach 1: Searching for an Official Matter API (Outcome: Unsuccessful)

My first approach was to find an official Matter API that I could use directly within my application. The idea was to leverage an existing API to avoid reinventing the wheel and speed up development. However, after researching extensively, I found that:

- There is no standalone API readily available for the Matter protocol that could be easily integrated with Qt or other desktop applications.
- The Matter SDK does not provide a high-level API for controller applications, and most of the available resources were command-line tools. (there is the chip-repl which provides some level of documenation but i will talk about it later) .

Because of this, I realized that I would need to look for other approaches, either by building custom solutions or adapting existing tools.

4.2. Approach 2: Writing My Own API (Outcome: Too Complex)

Realizing that no official API was available, I considered writing my own API for interacting with Matter devices. This would involve:

- Reading the Matter specification and understanding the protocol at a deep level.
- Implementing device commissioning, secure communication, and sending commands manually.

However, the challenges of this approach quickly became clear:

- **Complexity**: The Matter protocol is quite complex, and implementing all necessary components from scratch would take significant time.

- **Time constraints**: Given the duration of the internship, it wasn't feasible to develop and thoroughly test a custom API.
- **Lack of documentation**: The documentation for the low-level Matter protocol is sparse, making the development process difficult.

I was suggested by the Matter community that this sollution is an effort much higher than i can take.

As a result, I determined that building a custom API from scratch was not realistic within the given time-frame.

4.3. Approach 3: Understanding the chip-tool Source Code (Outcome: Infeasible)

Next, I decided to explore **chip-tool**, an official tool provided by the Matter SDK. The chip-tool is a command-line application that can commission devices and send commands, making it an ideal candidate to build upon.

I attempted to understand the inner workings of chip-tool with the goal of either:

- Extracting the relevant portions of the code to build my own API, or
- Modifying the tool to meet my specific requirements.

However, this approach presented its own set of difficulties:

- **No clear documentation**: The chip-tool source code lacked sufficient documentation, making it hard to follow and adapt.
- **High complexity**: The codebase was large, and understanding how different components interacted, espacilly trying to separte the logic of how the CLI work and the actual implementation or usage of Matter controller API was very challenging in the limited time I had.
- **Difficult build system:** the chip-tool uses GN and Ninja (Cmake and make). GN is build system that genrates Ninja files. this was a challenge of its own, i managed through trial and error to set it up.

During my exploration, I was advised to look at the **Espressif Matter Controller**, a simpler alternative that closely resembles chip-tool to understand the latter better. While the Espressif implementation was indeed easier to understand, yet i did't know how long this would take me and at the moment i didn't do anything concrete that is related to my objective and also running out of time, I decided not to pursue this path further .

4.4. Approach 4: Creating a Qt/C++ Wrapper Over chip-tool (Outcome: Chosen Solution)

Given the complexity of developing a custom API and understanding chip-tool in depth, I decided to create a wrapper around chip-tool using Qt and C++. This approach was chosen because it allowed me to:

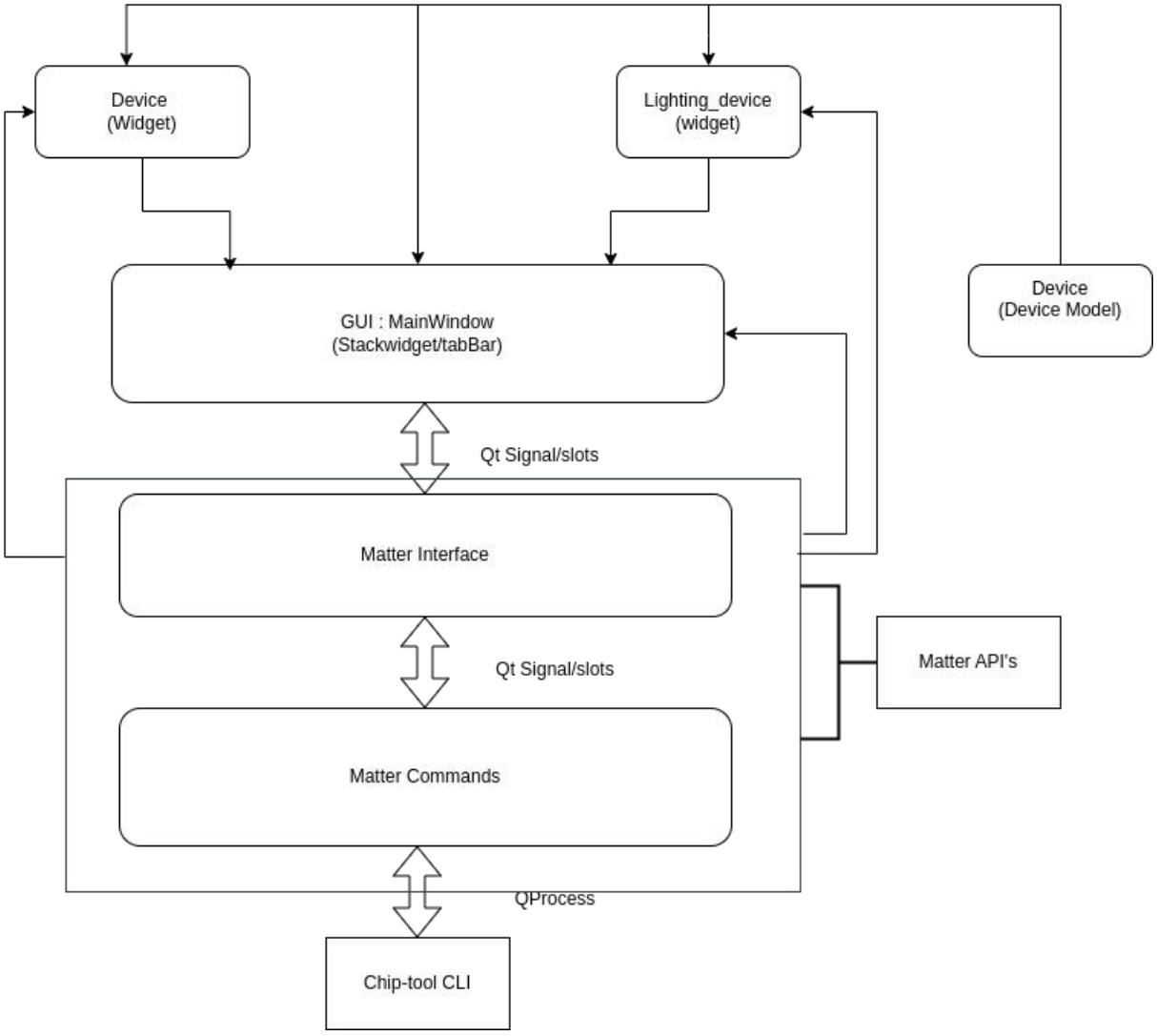
- **Leverage existing functionality**: By using chip-tool, I didn't have to implement the core Matter functionality from scratch.
- **Integrate with Qt**: I could easily wrap the chip-tool commands within a Qt application using QProcess, enabling me to provide a GUI that interacted with Matter devices.

4.1.1. Application key Features

- Discover devices (WiFi)
- automaticallty detect device category
- See device charachteristics
- Commission/decommession a device (onnetwork)
- Light control (Toggle light)
- Debug Terminal

4.1.2. Application Design

I wanted the application to be modular and not dependent from the chip-tool. So we can in future, just remove the layer that is dependent on the chip-tool and replace it with the appropriate Matter controller API.



This picture above demonstrates the design i went through. let's start from bottom of the picture up to the top:

1- First layer is the Matter commands:

It's a class that holds two functions: one to run the chip-tool command, and another to receive the output of the command. I chose this design because whether we modify the chip-tool or create our own API, we still need to

execute key functions (such as on/off, pairing, etc.). This is something I observed while reading the chip-tool source code.

By running the chip-tool command in its own process, I ensure that the main application remains non-blocking and protected from any potential errors that might occur during the execution of the command. This design allows the controller to operate safely and efficiently, without impacting the user experience.

2- Second layer: matter interface:

I wanted to inforce that any class that uses the Matter commands class needs to inherit this class.

To communicate with the process (get the matter command output) or know it’s exit status or code, i used Qt’s signals and slots mecagnisms. This Class enforces the developper to implement the necessary slots to recieve the result from the running process.

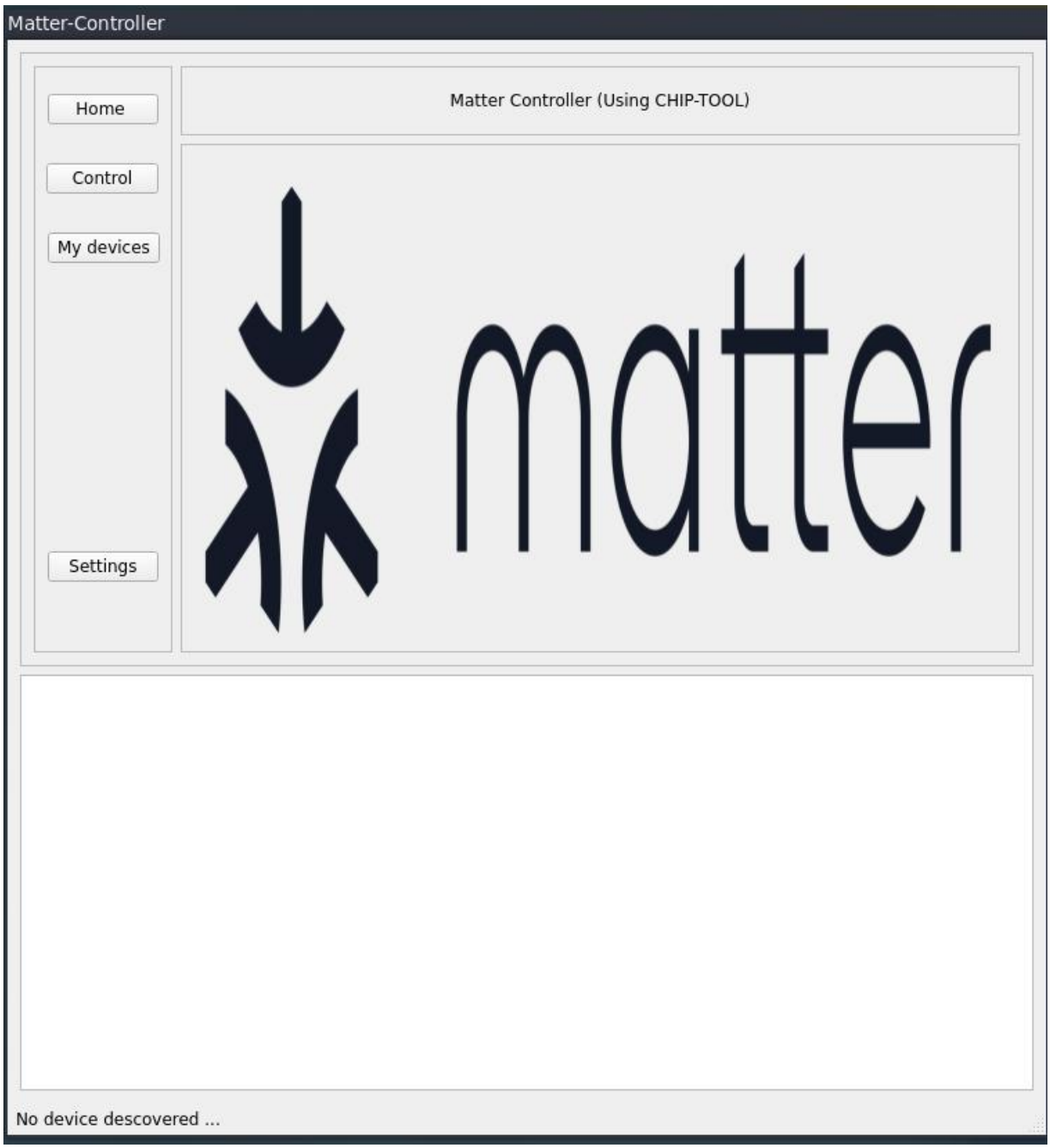
3- Third layer, User Interface:

This layer uses the previous two layers. its a combination of a main widget and two children widget that are used to represent the device object (device descovery) and device control (lighting control).

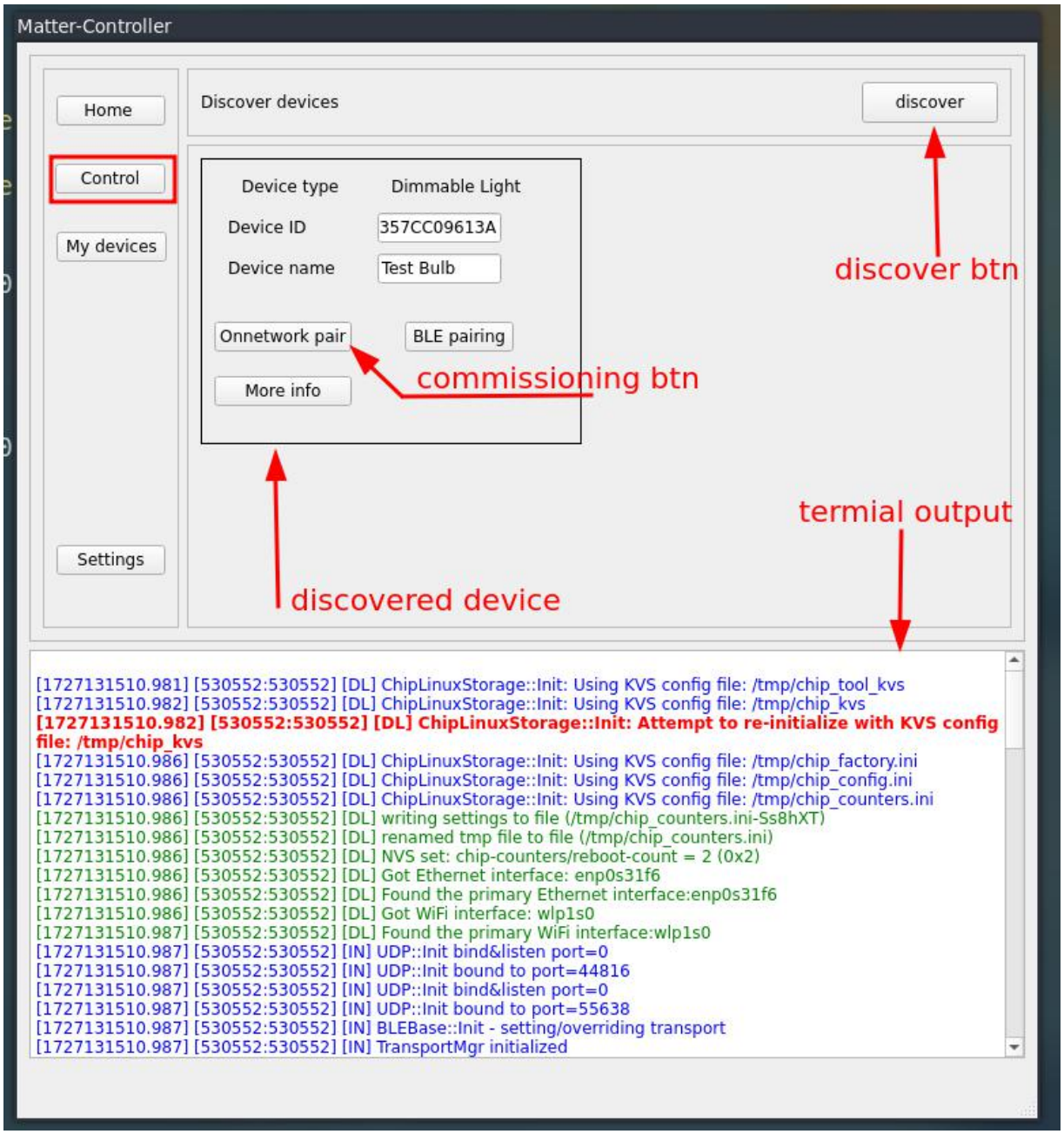
A Class called device is used to hold all the required infomtions about the device thought the process.

4.1.3. The Application

1- Home

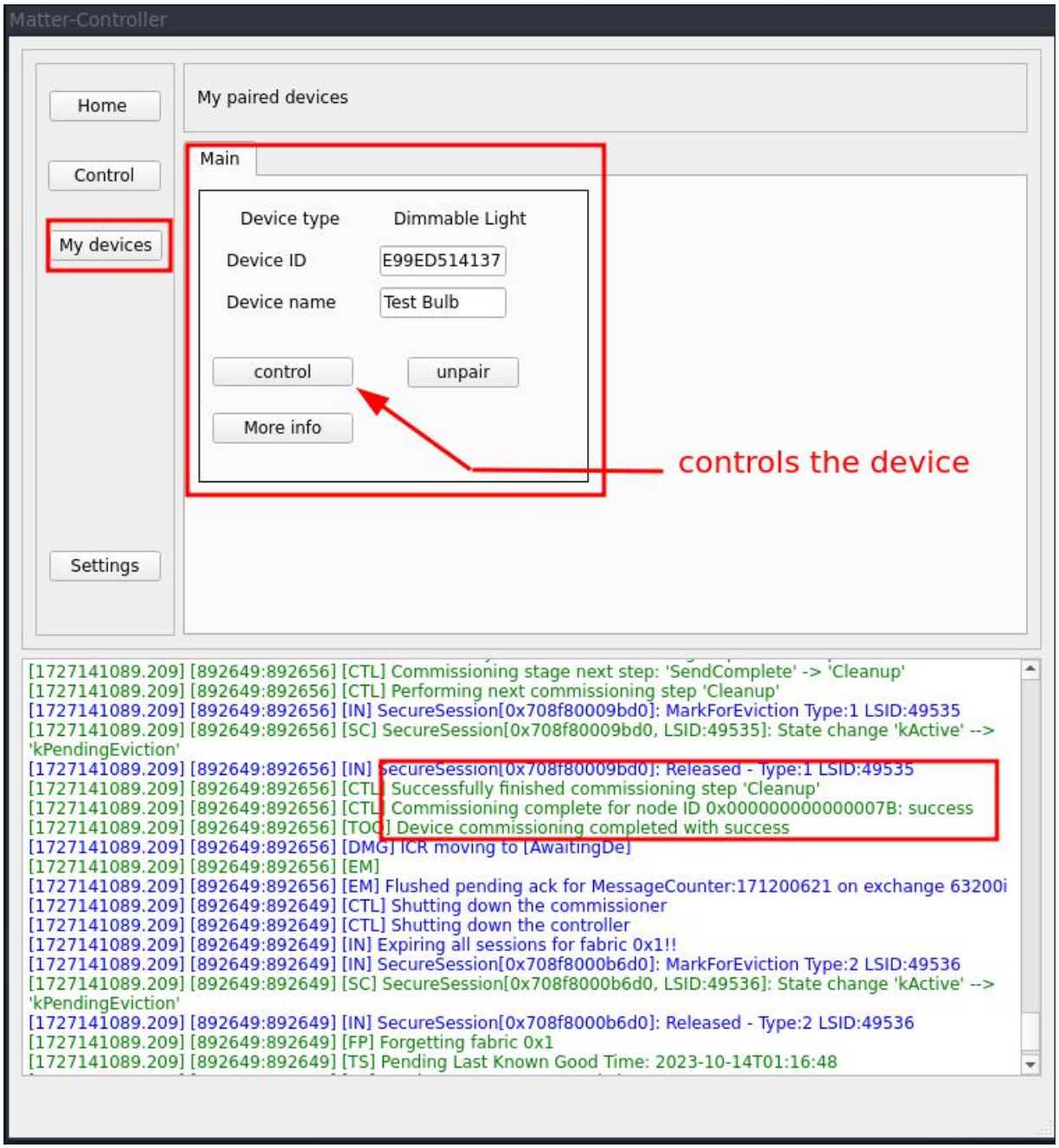


2- Control Ui: discover devices

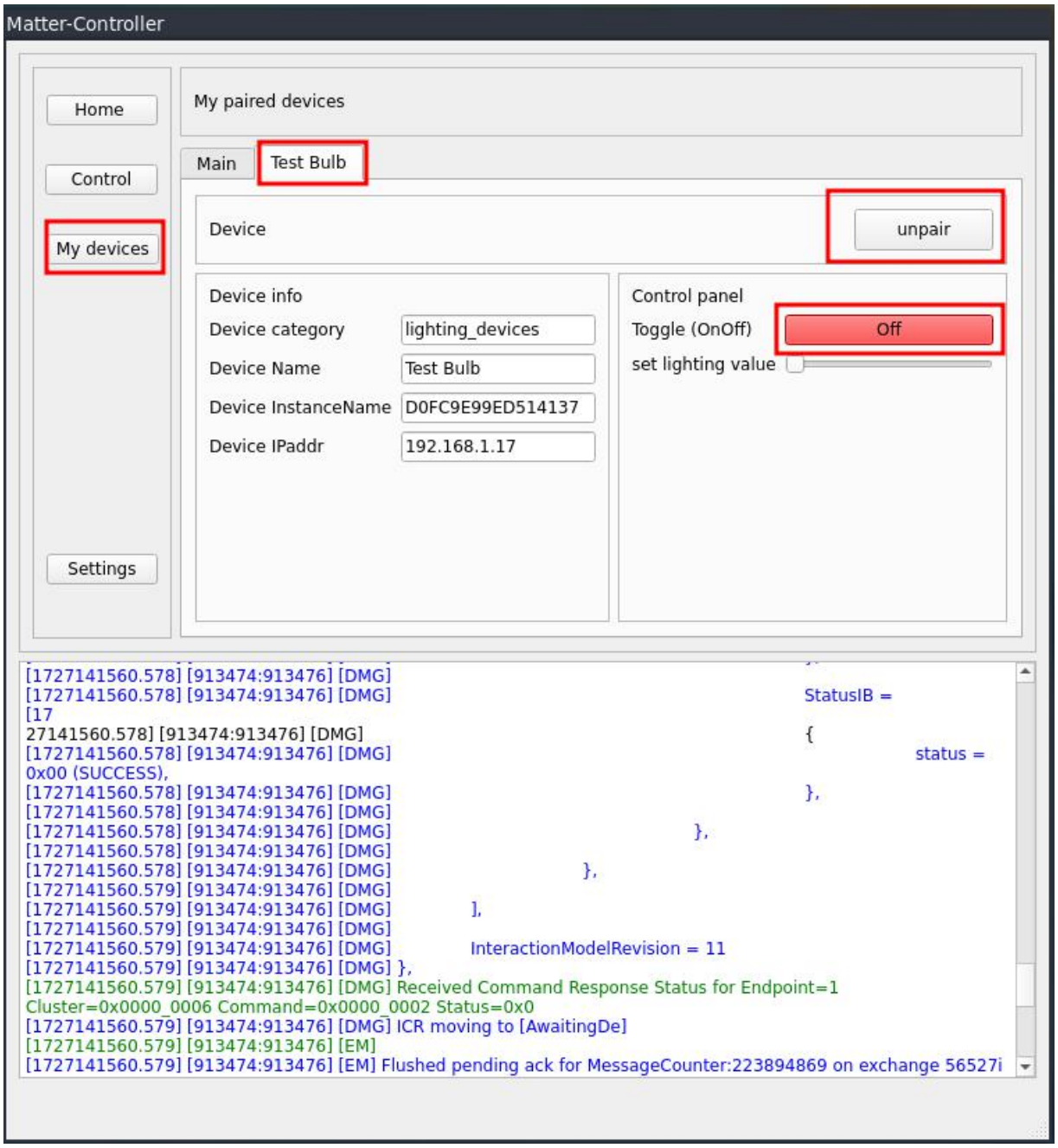


3- Onnetwork Commissionning:

For now, the device ID and the setup-pincode are hard-coded



4- Control commissioned device



4.5. Other Approaches I Considered

This section i go through other approachedi considered.

4.5.1. chip-repl (Python-based):

I was also informed about chip-repl, a Python-based REPL for controlling Matter devices. However, another intern was already working on this, and since I wanted to explore C++/Qt solutions, I opted to pursue other methods.

4.5.2. Matter.js:

I briefly looked into **Matter.js**, a web-based implementation of the Matter protocol. However, since my project was focused on a Linux desktop application, this web-based solution was not suitable for my needs.

5. Conclusion and References

Conclusion

This was a fruitful expereience where i learned a lot

There is still much more to imporve in the application.

- Error handling
- Discover more than on device (chip-tool doesn't support this)
- Support other category devices
- Improve the design modularity
- ...

I want to do much more but here comes the limitation chip-tool and the usage of virtual devices.

Refrences

you can find a more detailed description on matter, the source code in my github accound link.