POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: INŻYNIERIA INTERNETOWA

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja sterująca urządzeniami wykorzystująca serwisy GATT

Device control application that uses GATT services

AUTOR:
Dominik Hofman

PROWADZĄCY PRACĘ:
dr inż. Marek Bawiec, W4/K9

OCENA PRACY:

WROCŁAW, 2018

Spis treści

Spis rysunków

Spis tabel

Skróty

BLE (ang. *Bluetooth Low Energy*)
ATT (ang. *Attribute Protocol*)
GATT (ang. *Generic Attribute Profile*)
MQTT (ang. *Message Queuing Telemetry Transport*)
M2M (ang. *Machine To Machine*)
WLAN (ang. *Wireless Local Area Network*)
IT (ang. *Information Technology*)
SSH (ang. *Secure Shell*)
SD (ang. *Secure Digital*)
xD Picture Card (ang. *Extreme Digital Picture Card*)
MAC address (ang. *Media Access Control Address*)
RAM (ang. *Random-Access Memory*)
JSON (ang. *JavaScript Object Notation*)
XML (ang. *Extensible Markup Language*)
YAML (ang. *YAML Ain't Markup Language*)
IoT (ang. *Internet of Things*)
UUID (ang. *Universally Unique Identifier*)
GIL (ang. *Global Interpreter Lock*)
D-BUS (ang. *Desktop Bus*)

Rozdział 1

Wstęp

Internet rzeczy w ostatnich czasach staje się coraz popularniejszy za sprawą postępującego rozwoju technologicznego. Często wybieraną technologią komunikacji w takich zastosowaniach jest BLE (ang. *Bluetooth Low Energy*) GATT (ang. *Generic Attribute Profile*) ze względu na zalety, jakie oferuje standard Bluetooth Low Energy. Natłok różnych urządzeń korzystających z serwisów GATT oraz konieczność każdorazowego przygotowania dedykowanych aplikacji do konfiguracji tych urządzeń stwarza potrzebę uniwersalnego rozwiązania.

Niniejsza praca jest odpowiedzią na ten problem poprzez przygotowanie skryptów, które mogą zarządzać dowolnym urządzeniem wykorzystującym serwisy GATT, z poziomu systemu operacyjnego z rodziny Linux. Pozwoli to na obniżenie kosztów testowania oraz wdrażania nowych urządzeń i systemów IoT (eng. *Internet of Things*) opartych o tą technologię.

W kolejnych rozdziałach pracy zostanie opisany cel pracy, stawiane wymagania przed aplikacją, wybrane technologie, przygotowanie oraz konfiguracja systemu operacyjnego, podręcznik użytkowania oraz wybrane szczegóły implementacji.

Rozdział 2

Cel i zakres pracy

Celem niniejszej pracy inżynierskiej jest przygotowanie niezależnej biblioteki w języku Python, która zaimplementowana w postaci skryptów umożliwi komunikację z urządzeniami za pomocą aplikacji działającej w trybie serwisów GATT dzięki wierszowi poleceń z konfigurowalnymi parametrami. Zakres pracy obejmuje:

1. Przegląd oraz wybór dostępnych technologii pozwalających na komunikacje z urządzeniami BLE.
2. Wykonanie testów komunikacji na podstawie dołączonej dokumentacji z przykładowym urządzeniem.
3. Wybranie technologii oraz zaprojektowanie sterowania zdalnego.
4. Zaprojektowanie oraz implementacja skryptów zapewniających komunikację dwustronną z urządzeniami przy pomocy linii komend.
5. Zapewnienie łatwej modyfikacji konfiguracji powyższych skryptów, oraz możliwości przystosowania do dowolnego urządzenia wykorzystującego serwisy GATT.
6. Opracowanie podręcznika użytkowania aplikacji.
7. Testy manualne przygotowanego systemu.

Rozdział 3

Określenie wymagań

3.1. Analiza projektu

Projektowana aplikacja powinna spełniać następujące założenia:

- Skrypty pozwalają na zarządzanie urządzeniami z poziomu linii poleceń.
- Skrypty udostępniają interfejs pozwalający na zdalne zarządzanie urządzeniami.
- Użytkownik musi znać adres MAC urządzenia z którym chce wejść w interakcję.
- Skrypty nie muszą zapewniać możliwości wykrywania pobliskich urządzeń.
- Docelowym systemem na którym będą pracować skrypty będzie Raspbian.
- Użytkownik musi przygotować definicje charakterystyk dopasowaną do urządzenia z których chce wejść w interakcję.

3.2. Wymagania funkcjonalne

Koniecznym elementem jest określenie wymagań dla projektowanej aplikacji. W tym celu sporządzono tabele, które w dokładny sposób definiują funkcjonalność rozwiązania.

Tab. 3.1: Sterowanie za pomocą linii komend

Nazwa wymagania	Sterowanie za pomocą linii komend
Opis	Użytkownik ma możliwość zarządzania urządzeniami za pomocą linii komend.
Ograniczenia i warunki	Użytkownik jest zalogowany do systemu oraz posiada uprawnienia administratora.
Dane wejściowe	-
Wynik	Użytkownikowi udostępniony jest interfejs zarządzania z poziomu linii komend.

Tab. 3.2: Wyświetlenie pomocy

Nazwa wymagania	Wyświetlenie pomocy
Opis	Użytkownik ma możliwość wyświetlenia instrukcji użytkowania skryptów.
Ograniczenia i warunki	-
Dane wejściowe	-
Wynik	Skrypt wyświetla tekstową instrukcję użytkowania.

Tab. 3.3: Połączenie z wybranym urządzeniem

Nazwa wymagania	Połączenie z wybranym urządzeniem
Opis	Skrypt jest w stanie połączyć się z wybranym przez użytkownika urządzeniem.
Ograniczenia i warunki	Użytkownik musi podać adres MAC (ang. *Media Access Control Address*) urządzenia, urządzenie musi być włączone oraz być w odpowiednim zasięgu.
Dane wejściowe	Adres MAC docelowego urządzenia
Wynik	Skrypt łączy się z urządzeniem.

Tab. 3.4: Zmiana wartości wybranej charakterystyki

Nazwa wymagania	Zmiana wartości wybranej charakterystyki
Opis	Skrypt jest w stanie zmienić wartość wybranej przez użytkownika charakterystyki.
Ograniczenia i warunki	Urządzenie musi być włączone oraz być w odpowiednim zasięgu.
Dane wejściowe	Adres MAC docelowego urządzenia oraz UUID (ang. *Universally Unique Identifier*) wybranej charakterystyki.
Wynik	Skrypt zmienia wartość charakterystyki na urządzeniu.

Tab. 3.5: Odczyt wartości charakterystyki

Nazwa wymagania	Odczyt wartości charakterystyki
Opis	Skrypt jest w stanie odczytać wartość wybranej przez użytkownika charakterystyki.
Ograniczenia i warunki	Urządzenie musi być włączone oraz być w odpowiednim zasięgu.
Dane wejściowe	Adres MAC docelowego urządzenia oraz UUID wybranej charakterystyki.
Wynik	Skrypt odczytuje wartość charakterystyki z urządzenia.

Tab. 3.6: Możliwość podania hasła

Nazwa wymagania	Możliwość podania hasła
Opis	Skrypt musi umożliwiać podanie hasła przez użytkownika.
Ograniczenia i warunki	-
Dane wejściowe	Hasło w postaci tekstowej.
Wynik	Skrypt przyjmuje jako parametr hasło.

Tab. 3.7: Zapis do charakterystyki chronionej hasłem

Nazwa wymagania	Zapis do charakterystyki chronionej hasłem
Opis	Skrypt umożliwia zapis wartości do charakterystyki chronionej hasłem.
Ograniczenia i warunki	Urządzenie musi być włączone oraz być w odpowiednim zasięgu.
Dane wejściowe	Adres MAC docelowego urządzenia, UUID wybranej charakterystyki oraz hasło w postaci tekstowej.
Wynik	Skrypt zmienia wartość charakterystyki chronionej hasłem na urządzeniu.

Tab. 3.8: Definiowanie charakterystyk

Nazwa wymagania	Definiowanie charakterystyk
Opis	Użytkownik może zdefiniować listę charakterystyk obsługiwanych przez urządzenie.
Ograniczenia i warunki	Użytkownik musi zastosować się do ściśle zdefiniowanego formatu opisu charakterystyk.
Dane wejściowe	Konfiguracja charakterystyki.
Wynik	Skrypt korzysta z wprowadzonej przez użytkownika definicji charakterystyk.

Tab. 3.9: Interpretacja błędów urządzenia

Nazwa wymagania	Interpretacja błędów urządzenia
Opis	Skrypt potrafi odczytać oraz zinterpretować błędy sygnalizowane przez urządzenie.
Ograniczenia i warunki	Urządzenie musi komunikować błędy poprzez specjalną charakterystykę.
Dane wejściowe	UUID charakterystyki zawierającej dane o błędach.
Wynik	Skrypt wyświetla użytkownikowi w czytelnej formie błędy sygnalizowane przez urządzenie.

Tab. 3.10: Definiowanie błędów

Nazwa wymagania	Definiowanie błędów
Opis	Użytkownik jest w stanie poprzez plik konfiguracyjny definiować możliwe błędy sygnalizowane przez urządzenie.
Ograniczenia i warunki	Użytkownik musi zastosować się do ściśle zdefiniowanego formatu opisu błędów.
Dane wejściowe	Nazwy błędów oraz przesunięcie bitowe dotyczące danej nazwy.
Wynik	Skrypt interpretuje bajt błędu urządzenia na podstawie konfiguracji dostarczonej w pliku konfiguracyjnym przez użytkownika.

3.3. Wymagania niefunkcjonalne

Projekt przyjmuje następujące wymagania niefunkcjonalne:

- Skrypty powinny być łatwe w obsłudze.
- Skrypty będą działać na systemach z rodziny Linux.
- Skrypty powinny działać z dowolnym urządzeniem wykorzystującym serwisy GATT.
- Skrypty umożliwiają zdalne zarządzanie urządzeniem docelowym.
- Skrypty nie obciążają w znaczącym stopniu systemu na którym pracują.

Rozdział 4

Opis wybranych technologii

W tym rozdziale zostały w sposób dokładniejszy opisane technologie wykorzystywane w projekcie oraz stosowne uzasadnienie ich wyboru.

4.1. Platforma sprzętowa

Docelową platformą sprzętową jest Raspbery Pi Zero. Została wybrana ze względu na możliwość instalacji na niej systemu z rodziny Linux (konkretnie system Raspbian, dystrybucja oparta na Debianie). Jest to istotne ze względu na dużą liczbę dostępnych narzędzi developerskich związanych z najnowszym standardem Bluetooth 4.2. Równie ważnymi w docelowym zastosowaniu są cechy takie jak:

- zintegrowany układ Bluetooth obsługujący standard BLE,
- zintegrowany układ łączności bezprzewodowej WLAN,
- małe wymiary,
- niska cena.

4.2. GATT

GATT został zbudowany na podstawie ATT, (ang. *Attribute Protocol*), który używa danych GATT do zdefiniowania, w jaki sposób urządzenia BLE komunikują się ze sobą. GATT definiuje hierarchiczną strukturę danych, która jest udostępniana podłączonemu poprzez technologię BLE urządzeniu. Niniejsza struktura danych określana jest profilem. Protokół ten przewiduje istnienie klienta oraz serwera. Serwer udostępnia swój profil, natomiast klient może odczytywać, zapisywać oraz zażądać powiadomień w razie zmiany wartości charakterystyki.

Na profil składają się serwisy zawierające charakterystyki. Te z kolei składają się z właściwości, wartości oraz opcjonalnych deskryptorów, które dodatkowo opisują daną charakterystykę. W celu lepszego zobrazowania powiązań przedstawiony zostanie konkretny przykład pulsomierza z opcją odczytu pulsu poprzez serwisy GATT.

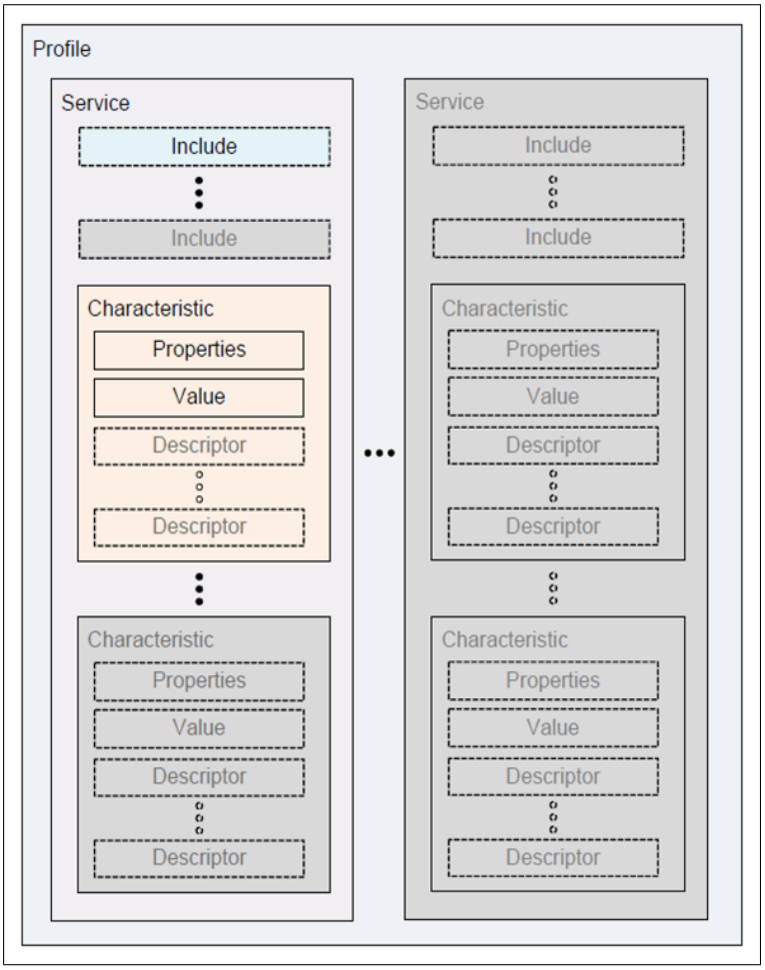
Serwisem w tym przypadku może być serwis zapewniający dane o samym urządzeniu takie jak stan baterii lub wersja wgranego oprogramowania lub też serwis zapewniający informacje o aktualnym tętnie. W tym przypadku charakterystyką byłby odczyt pulsu użytkownika. Właściwości takiej charakterystyki określają prawa dostępu do niej takie jak prawo zapisu, odczytu oraz notyfikacji. W opisywanym przykładzie sensowne są właściwości pozwalające na odczyt oraz notyfikację. Wartością takiej charakterystyki jest sam odczyt pulsu. Natomiast przykładowym deskryptorem może być jednostka, w jakiej został dokonany pomiar.

Dobrym obrazowym porównaniem jest również porównanie profilu do konstrukcji klas w obiektowych językach programowania. W tym przypadku serwis odpowiada klasie, charakterystyka polu a deskryptor rodzajowi pola. Na rysunku 2.1 przedstawiono wizualizację generycznego profilu.

4.3. Python

Zastosowanym w projekcie językiem programowania jest Python w wersji 3.5. Python jest językiem skryptowym ogólnego przeznaczenia, cechuje się dużą przejrzystością napisanego kodu, dużą szybkością pisania w nim oprogramowania, oraz niskim progiem wejścia. Python jest aktualnie popularnym językiem programowania, ma szerokie zastosowania od tworzenia stron internetowych przez aplikacje okienkowe, uczenie maszynowe do prostych skryptów automatyzujących powtarzalne czynności. Python jest najszybciej zdobywającym popularność językiem programowania według StackOverflow Trends[7]. Za wyborem tego języka przemawiały następujące jego cechy:

- wysoka czytelność napisanego w nim kodu,
- wysoka szybkość pisania w nim oprogramowania,
- duża społeczność wokół tego języka,
- duża liczba dostępnych gotowych bibliotek,



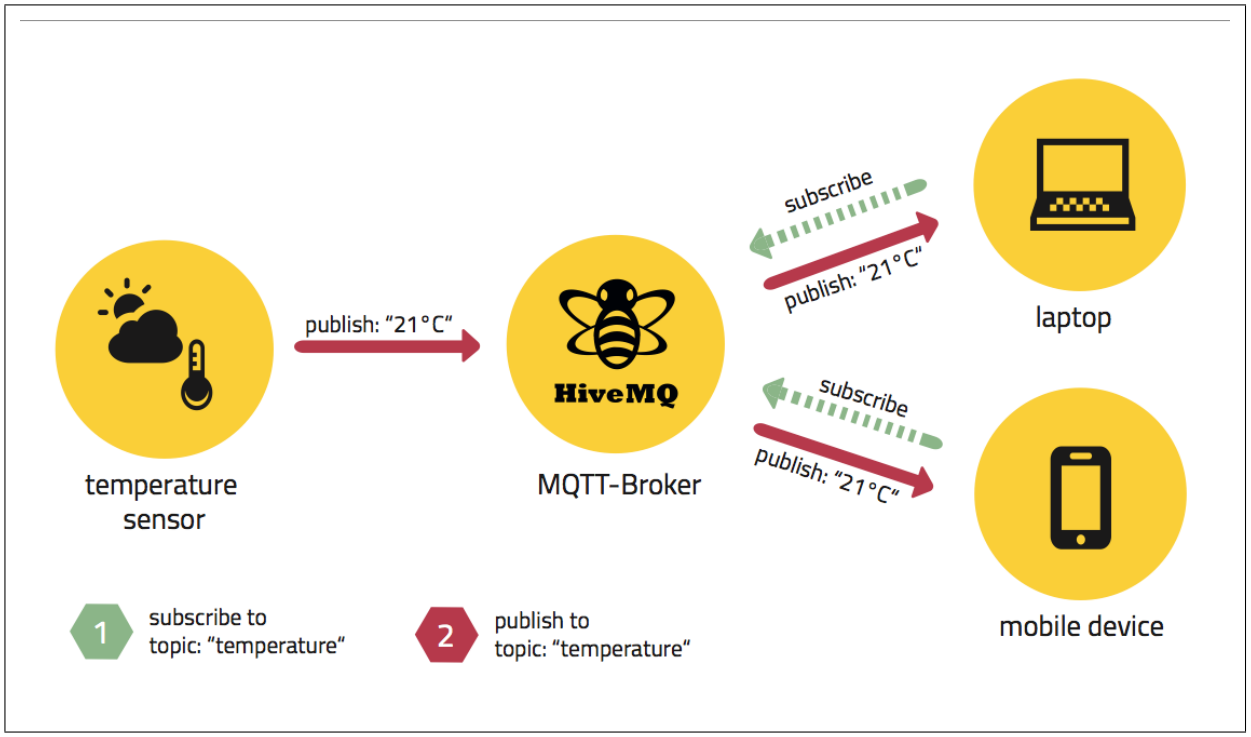
Rys. 4.1: Hierarchia profilu GATT, https://www.bluetooth.com/~/media/images/page-content/gatt%20profile%20hierarchy.ashx?la=en&hash=5809F05343EB927D9E9680E4CA70556B4ED69A67 [dostęp dnia 24.11.2018]

4.4. MQTT

Technologia zdalnego sterowania zastosowana w projekcie musi spełniać szereg wymagań, głównym z nich jest niskie zapotrzebowanie na zasoby. Na docelowej platformie, oprócz modułu sterowania zdalnego, będzie również działać wiele innych programów, które w znaczącym stopniu obciążają platformę. Innym wymaganiem jest zdolność do łatwego zarządzania wieloma platformami w systemie. Biorąc pod uwagę powyższe wymagania została wybrana technologia MQTT (ang. *Message Queuing Telemetry Transport*). MQTT[5] jest protokołem komunikacji MTM (ang. *Machine To Machine*) opartym na modelu publikacji/subskrypcji wiadomości. Model ten opiera się na prostej koncepcji tematu. Klienci mogą opublikować wiadomość na wybranym temacie która to następnie zostanie przez brokera rozesłana do wszystkich klientów którzy subskrybują dany temat. W ten sposób można zorganizować wydajną komunikację wielu urządzeń. Ważną cechą tego protokołu biorąc pod uwagę nasze wymagania jest minimalna liczba bajtów potrzebna do przesłania wiadomości oraz niski narzut wydajnościowy. Protokół ten wymaga dodatkowego serwera, który pośredniczy w wymianie wiadomości pomiędzy zalogowanymi klientami - tak zwany broker. Diagram ukazujący przykładowe zastosowanie oraz komunikację w technologii MQTT został ukazany na rysunku 3.2.

4.5. YAML

Konfiguracja skryptów odbywa się za pomocą plików zgodnych ze składnią YAML (ang. *YAML Ain't Markup Language*). YAML jest to język serializacji danych w przystępnej formie dla człowieka. YAML obsługuje trzy podstawowe typy danych: listy, słowniki oraz skalary (napisy, liczby całkowite oraz zmienno-przecinkowe), pozwala również na zagnieżdżanie się struktur danych. YAML powoli wypiera XML (ang. *Extensible Markup Language*), jeżeli chodzi o pliki konfiguracyjne, głównie za sprawą minimalnej składni oraz formatowania kodu w stylu Pythona,



Rys. 4.2: Diagram MQTT, `https://www.hivemq.com/wp-content/uploads/Screen-Shot-2014-10-22-at-12.21.07-1024x589.png` [dostęp dnia 5.12.2018]

co w efekcie pozwala na wysoką przejrzystość plików konfiguracyjnych. Z powyższych względów został wybrany jako główny format plików konfiguracyjnych w projekcie.

Rozdział 5

Konfiguracja środowiska

W niniejszym rozdziale zostanie opisany proces konfiguracji platformy sprzętowej, aby współpracowała ze skryptami.

5.1. Konfiguracja karty SD

Wybrana platforma sprzętowa wymaga specyficznego przygotowania przed możliwością użycia skryptów. Konfiguracja zostanie opisana na podstawie systemu operacyjnego Linux. Linie rozpoczynające się od $ oznaczają polecenie dla powłoki systemowej bash.

5.1.1. Pobranie Raspbiana

Należy pobrać obraz systemu operacyjnego Raspbian oraz go wypakować.

```
$ wget https://downloads.raspberrypi.org/raspbian_lite_latest -O raspbian.zip
$ unzip raspbian.zip
```

5.1.2. Zapisanie obrazu systemu na karcie SD

Należy zidentyfikować ścieżkę, pod jaką karta SD (ang. *Secure Digital*) pojawiła się w systemie, można tego dokonać poleceniem `mount`, wyświetla ona wszystkie zamontowane partycje, należy odszukać ścieżkę naszej karty pamięci.

```
$ mount | grep /dev/sd
```

Następnie należy odmontować partycje karty.

```
$ umount /dev/sdX*
```

Następnie używając polecenia `dd` zapisujemy obraz systemu na karcie SD. Używamy przy tym flagi `status=progress` która zapewnia nam podgląd na liczbę zapisanych bajtów.

```
$ dd if=./xxx.img of=/dev/sdX status=progress
```

Po zapisaniu karty SD warto włączyć usługę SSH (ang. *Secure Shell*) oraz podać systemowi dane dostępowe do lokalnej sieci WLAN

```
# Włączenie SSH
$ touch /boot/ssh

# Konfigurowanie WLAN
# Stworzenie pliku /boot/wpa_supplicant.conf
$ touch /boot/wpa_supplicant.conf
```

```
# Do stworzonego pliku zapisujemy następującą strukturę i podmieniamy
# wartości kluczy "ssid" oraz "psk" na odpowiadające naszej sieci WLAN

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=PL

network={
     ssid="ssid"
     psk="password"
     key_mgmt=WPA-PSK
}
```

5.2. Konfiguracja systemu operacyjnego

Tak skonfigurowaną kartę SD wkładamy w odpowiednie miejsce na płytce RPI oraz podłączamy zasilanie. System Raspbian powinien się załadować z karty SD, podłączyć do skonfigurowanej sieci WLAN oraz udostępnić usługę SSH (ang. *Secure Shell*). Większość poniższych komend wymaga uprawnień administratora, z tego powodu zalecanym jest po zalogowaniu się do urządzenia, przejście na użytkownika `root` przy użyciu komendy `sudo su`, lub poprzedzenie każdego polecenia komendą`sudo`.

5.2.1. Aktualizacja systemu

Pierwszym etapem jest aktualizacja systemu oraz programów zainstalowanych w systemie do najnowszych dostępnych wersji. Poniższe komendy zaktualizują bazę repozytoriów systemowych oraz pobiorą i zaktualizują oprogramowanie zainstalowane w systemie.

```
$ apt-get update
$ apt-get upgrade -y
```

5.2.2. Instalowanie zależności

Następnie należy zainstalować niezbędne programy, które nie są dostarczane razem z obrazem systemu. Wymaganym jest również zainstalowanie zależności biblioteki `bluez` koniecznych do jej skompilowania.

```
$ apt-get install -y python3-pip git vim
$ apt-get install -y pkg-config libboost-python-dev \
                     libboost-thread-dev libbluetooth-dev \
                     libglib2.0-dev python-dev
```

5.2.3. Przygotowanie do instalacji bibliotek

Proces kompilacji oraz instalacji wymaganych bibliotek wymaga znacznych ilości pamięci operacyjnej. Używana platforma sprzętowa posiada 512 MB RAM (ang. *Random-Access Memory*) oraz 100 MB przestrzeni SWAP, co jest niewystarczające do tego celu. Rozwiązaniem tego problemu jest zwiększenie pliku wymiany, po zakończeniu instalacji można powrócić do domyślnych 100 MB w celu zwolnienia miejsca na karcie SD. Dodatkowo podczas kompilacji występowały problemy z wersją Pythona. Biblioteka spodziewa się Pythona w wersji 3.4, natomiast domyślna wersja w systemie operacyjnym Raspbian to 3.5. Najprostszym rozwiązaniem tego problemu jest stworzenie linku symbolicznego do brakującego pliku.

```
# Naprawienie błędu kompilacji modułu GATT
$ ln -s /usr/lib/arm-linux-gnueabihf/libboost_python-py35.so \
        /usr/lib/arm-linux-gnueabihf/libboost_python-py34.so

# Zwiększenie rozmiaru pliku wymiany
$ sed -i 's/CONF_SWAPSIZE=100/CONF_SWAPSIZE=1000/g' /etc/dphys-swapfile
```

Po wykonaniu powyższej komendy konieczne jest ponowne uruchomienie systemu, aby rozmiar pliku wymiany został zaktualizowany.

```
$ reboot
```

5.2.4. Instalacja bibliotek

Ostatnim krokiem jest pobranie plików projektu oraz zainstalowanie modułów Pythona niezbędnych dla pracy skryptów. Proces instalacji może zabrać dużo czasu.

```
$ git clone https://github.com/LazyHater/CowDevice.git
$ cd ./CowDevice
$ pip3 install -r requirements.txt
```

Rozdział 6

Instrukcja użytkowania

Rozdział ten zostanie poświęcony opisaniu procesu konfiguracji skryptów do współpracy pod dowolne urządzenie komunikujące się poprzez serwisy GATT.

6.1. Konfiguracja skryptów

6.1.1. Skrypt `dynamictool.py`

Wprowadzenie do GATT

Aby poprawnie skonfigurować skrypt `dynamictool.py` niezbędna jest wiedza o strukturze serwisów GATT. Urządzenie korzystające z serwisów GATT opakowuje swoje funkcjonalności w logiczne struktury - serwisy. Każdy serwis jest kolekcją charakterystyk które zapewniają informacje oraz pozwalają sterować funkcjonalnością danego urządzenia. Na charakterystykę składa się jej wartość, modyfikatory dostępu oraz opcjonalne deskryptory które zapewniają dodatkowe informacje o zarządzanej przez nią wartości. Modyfikatory dostępu określają w jaki sposób skrypty mogą wejść z nią w interakcję. Modyfikator `READ` oznacza że można odczytać jej wartość, `NOTIFY` zapewnia powiadomienia w razie zmiany wartości, natomiast `WRITE` pozwala na zapisanie własnej wartości. Wykorzystując powyższe mechanizmy można pozyskać informacje na temat interesujących nas funkcjonalności, jak i sterować funkcjami urządzenia poprzez zapis do odpowiadającym im charakterystyk. Odczytywaną wartością charakterystyki są bajty których interpretacja zależy od programu je przetwarzającego. Z tego powodu użytkownik musi zdefiniować w pliku konfiguracyjnym typ danych jaki reprezentuje dana charakterystyka. Każdy serwis, charakterystyka oraz deskryptor musi posiadać unikalny identyfikator (UUID).

Konfiguracja

Konfiguracja skryptu polega na edytowaniu pliku `services.yml` w którym definiuje serwisy oraz charakterystyki w znaczeniu GATT. Plik opisuje charakterystyki, które następnie będą przetwarzane przez skrypty. Plik ma logiczną strukturę słownikową gdzie pod pewnymi kluczami program spodziewa się wartości w konkretnym typie i formacie. W poniższych tabelach kolumna "Typ" określa typ danych akceptowany jako wartość danego klucza. Definicja możliwych typów została zawarta w tabeli 5.1.

Oprócz charakterystyk możliwych do zdefiniowania przez użytkownika zostały wyodrębnione specjalne pozycje - `password` oraz `error`. Okazało się to konieczne ze względu na specyficzny sposób przetwarzania tych danych. `password` określa hasło, które zostanie zapisane pod

Tab. 6.1: Zestawienie typów danych

Typ	Przykład	Opis
number	542	Liczba
string	”abcd”	Łańcuch znakowy
boolean	true	Wartość logiczna true/false
list	[1, 2, 3]	Lista wartości
dict	{”a”: 2, ”b”: 6}	Słownik

Tab. 6.2: Zestawienie kluczy podstawowych

Klucz	Typ	Opis
description	string	Opis urządzenia którego konfiguracja dotyczy, wyświetlany przez zastosowanie flagi `-help`
services	dict	Słownik serwisów, gdzie kluczem jest nazwa serwisu

Tab. 6.3: Zestawienie kluczy serwisów

Klucz	Typ	Opis
uuid	string	UUID serwisu
characteristics	dict	Słownik charakterystyk gdzie kluczem jest nazwa charakterystyki
special	dict	Słownik specjalnych charakterystyk

Tab. 6.4: Zestawienie kluczy charakterystyk

Klucz	Typ	Opis
uuid	string	UUID charakterystyki
flags	list	Lista flag które będą powiązane z daną charakterystyką
type	string	Typ danej charakterystyki, dostępne wartości: number, string
encoding	string	Typ kodowania który zostanie użyty przy zapisywaniu oraz odczycie wartości charakterystyki, dostępne typy kodowań są wymienione w dokumentacji Pythona [3]
re	string	Wyrażenie regularne zgodne z modułem `re` Pythona. Zostanie użyte do walidacji tekstu wprowadzonego przez użytkownika
help	string	Pomoc wyświetlana dla użytkownika dotycząca tej charakterystyki
choices	list	Lista możliwych wartości, z których użytkownik może wybrać jedną
signed	boolean	Określa czy wprowadzona liczba ma być traktowana jako liczba ze znakiem
order	string	Określa kolejność bajtów w liczbie, możliwe wartości: `"little"` lub `"big"`
length	number	Liczba bajtów przeznaczona na liczbę
password_protected	boolean	Określa czy dana charakterystyka wymaga podania hasła przed zapisem do niej

wskazany UUID przed próbą edycji wartości zabezpieczonych hasłem. Natomiast `error` określa sposób dekodowania danych o błędzie. Wprowadzono założenie, że urządzenie informuje o błędach poprzez jeden bajt, w którym poszczególne bity odpowiadają rodzajom błędów.

Tab. 6.5: Zestawienie kluczy charakterystyk specjalnych

Klucz	Typ	Opis
uuid	string	UUID charakterystyki
errors	list	Lista słowników z możliwymi błędami
bitmask	number	Określa o ile bitów należy przesunąć otrzymany bajt błędu
message	string	Nazwa błędu czytelna dla użytkownika skryptów

Otrzymany z urządzenia bajt błędu zostanie przetworzony za pomocą następującego algorytmu:

```
error_code # otrzymany z urządzenia
errors # lista słowników z pliku konfiguracyjnego
for error in errors:
    if error_code & (1 << error['bitmask']):
        # dany błąd wystąpił
        print(error['message'])
```

6.1.2. Skrypt `remote.py`

Konfiguracja skryptu dostępu zdalnego odbywa się poprzez edycję pliku `remote.yml`. Możliwe typy danych opisuje tabela 6.1.

Tab. 6.6: Zestawienie kluczy konfiguracji skryptu dostępu zdalnego

Klucz	Typ	Opis
host	string	Nazwa hosta brokera
port	number	Numer portu brokera
name	string	Nazwa, która zostanie użyta do subskrypcji tematu, pod którym skrypt będzie oczekiwał poleceń, jeżeli zostanie podany pusty łańcuch znakowy, to nazwa zostanie ustalona na podstawie nazwy hosta urządzenia
auto_reconnect	boolean	Określa czy skrypt ma próbować się ponownie połączyć w przypadku zerwania połączenia z brokerem
reconnect_delay	number	Określa czas w sekundach pomiędzy kolejnymi próbami podłączenia się do brokera
keep_alive	number	Określa czas w sekundach pomiędzy wysyłaniem żądań podtrzymania połączenia z brokerem

Przykładowe pliki konfiguracyjne znajdują się w repozytorium z kodem źródłowym.

6.2. Używanie skryptów

6.2.1. Skrypt `dynamictool.py`

Po odpowiednim skonfigurowaniu skryptów można zacząć administrować docelowym urządzeniem. Odbywa się to poprzez skrypt `dynamictool.py`. Poniższe przykłady będą oparte na konfiguracji skryptów takiej jak w repozytorium z kodem źródłowym. Domyślnie w systemie Raspbian dostęp do modułu Bluetooth wymaga uprawnień administratora, z tego względu wszystkie komendy korzystające z tego modułu wymagają wykonywania z poziomu uprzywilejowanego użytkownika lub poprzedzenia komendą `sudo`. Jako pierwszy argument skryptu należy podać adres MAC urządzenia, z którym chcemy wejść w interakcję. W poniższych przykładach `ca:2c:83:fe:72:48` zostanie przyjęty jako adres MAC.

Aby wyświetlić dostępne flagi, oraz możliwe do przekazania argumenty wraz z opisami, należy użyć polecenia:

```
$ python3 dynamictool.py --help
```

Aby odczytać wartości wszystkich skonfigurowanych charakterystyk, należy użyć polecenia:

```
$ sudo python3 dynamictool.py ca:2c:83:fe:72:48 --read-all
```

Aby dokonać zapisu do charakterystyki, która nie jest chroniona hasłem, należy użyć polecenia:

```
$ sudo python3 dynamictool.py ca:2c:83:fe:72:48 --led 1
```

W powyższym przypadku zapisujemy wartość numeryczną "1" do charakterystyki "led".

Aby dokonać zapisu do charakterystyki, która jest chroniona hasłem, należy użyć polecenia:

```
$ sudo python3 dynamictool.py ca:2c:83:fe:72:48 --date 29/04/1 --password pass123
```

Hasło należy podać jako argument flagi `-password`.

W zależności od konfiguracji skryptów flagi mogą posiadać swoje skrócone wersje. Przykładowo poniższe dwa polecenia są równoważne:

```
$ sudo python3 dynamictool.py ca:2c:83:fe:72:48 --led 1
$ sudo python3 dynamictool.py ca:2c:83:fe:72:48 -l 1
```

Domyślnie komendy zapisujące dane do urządzenia nie generują żadnej informacji zwrotnej dla użytkownika, a komenda odczytująca dane wyświetla jedynie wartości odczytanych charakterystyk. Można ten stan rzeczy zmienić podając dodatkowo jedną z flag `-v -vv -vvv`, powodują one kolejno zwiększenie poziomu tak zwanego `verbosity` na kolejno "Warning", "Info", "Debug". Pozwala to na dokładniejsze przyjrzenie sie procesowi wykonywania komendy. Jest to szczególnie przydatne w przypadku wystąpienia błędów podczas wykonywania polecenia.

6.2.2. Skrypt `remote.py`

Wprowadzenie do MQTT

MQTT wprowadza pojęcia tematu, subskrypcji oraz publikacji. Klienci po połączeniu z brokerem mogą subskrybować tematy oraz publikować na nich wiadomości. Opublikowane wiadomości zostaną rozesłane pomiędzy wszystkimi klientami, którzy dany temat subskrybują. Tematy mają postać łańcuchów znakowych oraz nie trzeba ich tworzyć przed ich subskrybcją lub publikacją na nich wiadomości. Protokół nie definiuje ścisłego sposobu konstruowania tematów, w związku z czym w skrypcie przyjęto konwencję podzielenia tematu na trzy logiczne sekcje oddzielone znakiem "/": `/<inbox/outbox>/<devicename>/<endpoint>`. Pierwsza sekcja określa czy wiadomość jest wyprodukowana przez skrypt (`outbox`), lub czy jest do niego skierowana (`inbox`). Druga sekcja określa dla której platformy w systemie jest przeznaczona, w związku z tym każda platforma w systemie musi mieć unikatowy `devicename`. Trzeci człon określa której funkcjonalności skryptu dotyczy wiadomość. Treścią wiadomości są serializowane obiekty w formacie JSON. Większość popularnych języków programowania posiada implementację MQTT w postaci gotowych do użycia bibliotek, co pozwala na łatwą automatyzację zlecania zadań przez różnego rodzaju programy. Do ręcznego testowania skryptu `remote.py` można użyć graficznego narzędzia MQTT.fx[1]. Pozwala ono między innymi na subskrybowanie tematów, publikację wiadomości oraz odczytanie statusu brokera.

Zasada działania skryptu

Skrypt `remote.py` stanowi nakładkę na skrypt `dynamictool.py`, w związku z czym oferuje identyczną funkcjonalność oraz wymaga wcześniejszej konfiguracji charakterystyk w pliku

services.yml. Argumenty przekazywane w wiadomościach zlecania polecenia są bezpośrednio przekazywane do skryptu dynamictool.py, oznacza to że obowiązuje identyczna składnia jak w przypadku bezpośredniego korzystania z tego skryptu, co zostało opisane w poprzednim rozdziale. UWAGA! W razie nieautoryzowanego dostępu do brokera MQTT do którego podłączony jest skrypt zdalnego sterowania, potencjalny atakujący jest w stanie uzyskać uprawnienia administratora na platformie w której skrypt jest uruchomiony.

Używanie skryptu

Po odpowiednim uzupełnieniu plików remote.yml oraz dynamictool.yml, można użyć skryptu remote.py który udostępni interfejs zarządzania zdalnego urządzeniami docelowymi. Należy do tego użyć komendy:

```
$ sudo python3 remote.py
```

Wartym zaznaczenia jest to że w tym przypadku również jest wymagane użycie tego polecenia z uprawnieniami administratora, aby zapewnić dostęp do modułu Bluetooth.

Po uruchomieniu, program połączy się z brokerem oraz zasubskrybuje temat /inbox/<name>/execute, gdzie "<name>" jest nazwą skonfigurowaną w pliku remote.yml, lub aktualnym hostname systemu.

Aby zlecić wykonanie polecenia, należy opublikować w powyższym temacie wiadomość w formacie JSON (ang. *JavaScript Object Notation*). Opis możliwych kluczy w wiadomości znajduje się w tabeli 6.7.

Tab. 6.7: Zestawienie kluczy wiadomości zlecania wykonania polecenia

Klucz	Typ	Wymagany	Przykład	Opis
id	number	Nie	123	Identyfikator zadania, jeżeli zostanie podany to zostanie użyty w odpowiedzi
mac	string	Tak	"ca:2c:83:fe:72:48"	Adres MAC docelowego urządzenia
args	string	Tak	"-r"	Argumenty które zostaną przekazane skryptowi dynamictool.py
password	string	Nie	"pass123"	Hasło wymagane do edycji charakterystyk zabezpieczonych hasłem

Następnie skrypt doda do kolejki zlecenie wykonania odpowiedniego polecenia dynamictool.py. Program następnie kolejno będzie wywoływał zadania z kolejki. Po zakończeniu polecenia zostanie opublikowana wiadomość w formacie JSON na temacie /outbox/<name>/execute zawierająca rezultat wykonanej komendy skryptu dynamictool.py. Opis kluczy odpowiedzi znajduje się w tabeli 6.8.

Tab. 6.8: Zestawienie kluczy wiadomości odpowiedzi zlecania wykonania polecenia

Klucz	Typ	Opis
id	number	Identyfikator zadania, który zostanie skopiowany ze zlecenia lub wygenerowany losowo w przypadku jego braku
returncode	number	Kod zwrócony ze skryptu dynamictool.py
stdout	string	Standardowy strumień zwrócony ze skryptu dynamictool.py
stderr	string	Standardowy strumień błędu zwrócony ze skryptu dynamictool.py

Rozdział 7

Szczegóły implementacji

W tym rozdziale zostaną omówione wybrane szczegóły związane z implementacją projektu, stanowiące jeden z ważniejszych aspektów niniejszej pracy.

7.1. Biblioteka obsługująca GATT

Jako bibliotekę obsługującą komunikację za pośrednictwem GATT została wybrana biblioteka `gatt-python`[2]. Biblioteka do komunikacji z urządzeniami Bluetooth wykorzystuje oficjalny stos protokołów BlueZ[4] na platformie Linux. Natomiast do komunikacji z samym BlueZ wykorzystuje prosty system komunikacji międzyprocesowej D-Bus[6] (ang. *Desktop Bus*). Bilbioteka udostępnia prosty interfejs komunikacji z urządzeniami wykorzystującymi GATT, zapewniając dodatkową warstwę abstrakcji. Za jej pomocą możliwe są następujące operacje:

1. skanowanie w poszukiwaniu pobliskich urządzeń BLE,
2. podłączanie oraz odłączanie od urządzeń,
3. implementowanie własnego profilu GATT,
4. odczyt dostępnych serwisów urządzenia,
5. odczyt dostępnych charakterystyk urządzenia,
6. odczyt wartości charakterystyki,
7. zapis wartości charakterystyki,
8. powiadomienia o zmianie wartości charakterystyki.

Skrypty korzystają z funkcjonalności 2, 4, 5, 6 oraz 7 tej biblioteki.

Korzystanie z biblioteki polega na stworzeniu klasy dziedziczącej po klasie `gatt.Device` oraz nadpisywaniu funkcji klasy bazowej. Następnie należy stworzyć obiekt klasy `gatt.DeviceManager` który będzie zarządzał połączeniem z urządzeniem. Proces tworzenia wyżej wymienionych obiektów odbywa się w pliku `dynamictool.py`. Definicja obiektu dziedziczącego po `gatt.Device` znajduje się w pliku `cowdevice.py`. Komunikacja z docelowym urządzeniem odbywa się asynchronicznie, poprzez wywoływanie poszczególnych funkcji z klasy definiującej urządzenie przez menedżera urządzenia.

7.2. Moduł przetwarzania konfiguracji

Ważnym elementem projektu było zapewnienie łatwej zmiany konfiguracji obsługiwanych serwisów oraz charakterystyk. Tą częścią aplikacji zajmuje się klasa `CharacteristicsMenager` znajdująca się w pliku `characteristicsmenager.py`. Na podstawie konfiguracji zawartej w pliku `services.yml`, tworzy listę obiektów reprezentujących skonfigurowane charakterystyki.

Udostępnia również wiele metod ułatwiających korzystanie z wygenerowanej listy charakterystyk.

7.3. Moduł obsługi linii poleceń

Przetwarzaniem komend użytkownika na polecenia aplikacji zajmuje się moduł `cmdhandler.py`. Korzysta on z domyślnie zainstalowanego modułu `argparse`. Moduł na podstawie listy charakterystyk wygenerowanych przez klasę `CharacteristicsMenager` tworzy odpowiadające argumenty linii komend oraz rejestruje je w podanej instancji klasy `argparse.ArgumentParser`.

Rozdział 8

Podsumowanie

Zrealizowany projekt inżynierski okazał się użyteczny w pracach rozwojowych nad urządzeniami wykorzystującymi serwisy GATT. Przede wszystkim pozwala zaoszczędzić czas potrzebny do konfiguracji pojedynczego urządzenia, co w efekcie umożliwia wydajną konfigurację masowej liczby urządzeń. Zaimplementowany sposób zmiany konfiguracji charakterystyk pozwala na łatwe dostosowanie skryptów do działania z dowolnym urządzeniem. Co więcej, wybrane technologie pozwalają na łatwe i szybkie rozwijanie oprogramowania pod kątem łatania błędów, oraz dodawania nowych funkcjonalności.

Skrypty zostały wykorzystane do masowej konfiguracji urządzeń w projekcie komercyjnym, czym udowodniły swoją przydatność w praktyce.

Dopracowania natomiast wymagają kwestie bezpieczeństwa. W razie nieautoryzowanego dostępu do brokera MQTT do którego podłączony jest skrypt zdalnego sterowania, potencjalny atakujący jest w stanie uzyskać uprawnienia administratora na platformie w której skrypt jest uruchomiony.

Literatura

[1] Graficzne narzędzie MQTT.fx. https://mqttfx.jensd.de/ [dostęp dnia 14 grudnia 2018].

[2] Repozytorium biblioteki gatt-python. https://github.com/getsenic/gatt-python [dostęp dnia 9 grudnia 2018].

[3] Standardowe kodowania w języku Python. https://docs.python.org/2.4/lib/standard-encodings.html [dostęp dnia 9 grudnia 2018].

[4] Stos protokołów BlueZ. http://www.bluez.org/ [dostęp dnia 14 grudnia 2018].

[5] Strona projektu MQTT. http://mqtt.org/ [dostęp dnia 14 grudnia 2018].

[6] System komunikacji D-Bus. https://www.freedesktop.org/wiki/Software/dbus/ [dostęp dnia 14 grudnia 2018].

[7] D. Robinson. StackOverflow Trends. https://stackoverflow.blog/2017/09/06/incredible-growth-python/ [dostęp dnia 12 grudnia 2018].

Dodatek A

Opis załączonej płyty CD/DVD

Na załączonej płycie znajduje się niniejsza praca oraz repozytorium z kodem źródłowym omawianych skryptów.