



TECHNOLOGIE INTERNETU RZECZY
III Rok

INFORMATYKA
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

STEROWANIE URZĄDZENIAMI
VIRTUALCOPERNICUSNG
Z WYKORZYSTANIEM
PROTOKOŁU CoAP
DOKUMENTACJA

KAROL MUSUR
JACEK NITYCHORUK
ALBERT GIERLACH
KAMIL KOCZERA

SEMESTR ZIMOWY
2020/2021

Spis treści

1	Postęp prac po pierwszych konsultacjach	3
1.1	server.py	3
1.2	client.py	4
1.3	Przykład wykonania programów	4
1.4	Dalsze plany	5
2	Postęp prac po drugich konsultacjach	6
2.1	server.py	6
2.2	client.py	7
2.3	Przykład wykonania programów	7
2.4	Klienci CoAP	9
2.4.1	coap-client (libcoap2)	9
2.4.2	gen_coap	10
2.4.3	Copper	11
2.4.4	Californium	14
3	Postęp prac po trzecich konsultacjach	16
3.1	server.py	16
3.2	client.py	18
3.3	Przykład wykonania programów	19
3.4	Przykładowe programy	20
3.4.1	observable_resource	20
3.4.2	lab - Sterownik świateł wykorzystujący protokół CoAP	21
3.5	Uruchomienie na Raspberry Pi	25

1 Postęp prac po pierwszych konsultacjach

Udało nam się przygotować prosty serwer oraz klienta (program testowy) przy wykorzystaniu biblioteki aiocoap oraz VirtualCopernicusa wraz z nakładką dostarczoną w trakcie zajęć laboratoryjnych. Zaimplementowana została także obsługa serwomechanizmu.

1.1 server.py

Program przyjmuje argument 'gpiozero' lub 'virtual' i w zależności od argumentu uruchomiona zostaje funkcja do obsługi serwera na urządzeniu gpiozero (jeszcze nie zaimplementowana) lub na VirtualCopernicucie:

Poniższe funkcje:

```
1  def notify(self):
2      self.updated_state()
3      self.reschedule()
4
5  def reschedule(self):
6      self.handle = asyncio.get_event_loop().call_later(5, self.notify)
7
8  def update_observation_count(self, count):
9      if count and self.handle is None:
10         print("Starting the clock")
11         self.reschedule()
12     if count == 0 and self.handle:
13         print("Stopping the clock")
14         self.handle.cancel()
15         self.handle = None
```

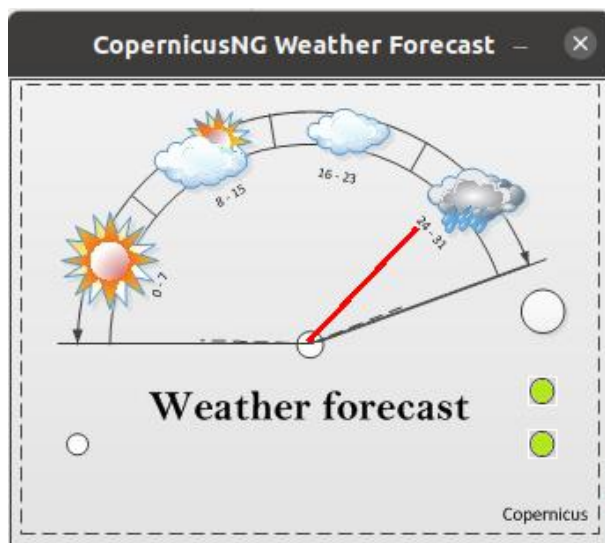
są eksperymentalne i służą do obserwowania zmian. Póki co nie są jeszcze używane w programie i zastanawiamy się nad sensem ich użycia.

1.2 client.py

Program testowy wysyła requesta na adres `coap://localhost/servo` i oczekuje odpowiedzi. W przypadku błędu zwraca stosowną informację o niepowodzeniu wykonania żądanej operacji. Za pomocą tego programu możemy pobrać wartość na serwerze za pomocą komendy 'g', ustawić wartość za pomocą 's v', gdzie v to wartość, która ma zostać ustawiona (jeśli ten proces przebiegnie pomyślnie, to położenie wskazówki (serwomechanizmu) na VirtualCopernicusie powinno ulec zmianie - w zależności od wprowadzonej wartości) lub zakończyć działanie programu za pomocą 'e'.

1.3 Przykład wykonania programów

Po uruchomieniu serwera wyświetlone zostaje następujące okno:



Rysunek 1: Okno pojawiające się po uruchomieniu serwera

Natomiast po uruchomieniu programu testowego dostajemy następujące informacje:

```
e - exit
g - get resource value
s v - set resource value to v
> S
```

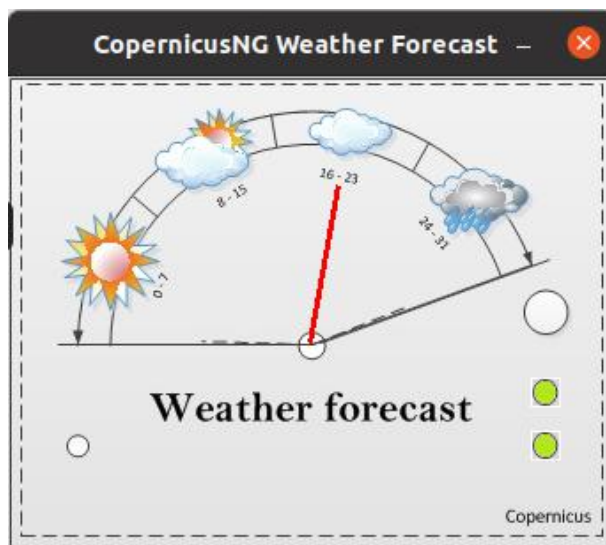
Rysunek 2: Informacje pojawiające się po uruchomieniu programu testowego

Po wprowadzeniu komendy 'g' w programie testowym otrzymujemy następujący komunikat:

```
> g
Result: 2.05 Content
b'44.0'
```

Rysunek 3: Pobranie wartości z serwera

Po wprowadzeniu w programie testowym komendy np. 's 10' możemy zauważyć, że zmieniło się położenie wskazówki (serwomechanizmu):



Rysunek 4: Położenie wskazówki po ustawieniu wartości na '10'

Po ponownym wykonaniu komendy 'g' na programie testowym widzimy, że wartość została prawidłowo ustawiona:

```
> g
Result: 2.05 Content
b'10.0'
```

Rysunek 5: Efekt wykonania komendy 'g' na programie testowym po ustawieniu wartości na '10'

1.4 Dalsze plany

W najbliższym czasie planujemy znaleźć klienta do samego CoAP'a i jeśli wszystko zadziała to spróbujemy odpalić "Eclipse Leshana" w celu sprawdzenia, czy da się wyeksponować zasoby CoAP'owe, tak, aby Leshan je widział. Planujemy także zrobić przegląd pluginów do CoAP'a oraz przygotować kolejne programy testowe.

2 Postęp prac po drugich konsultacjach

Serwer został rozbudowany o dodatkowe zasoby, a klient dostosowany do zmian w serwerze. Dodatkowo do serwera został dodany `"/.well-known/core"` oraz przetestowane zostało funkcjonowanie serwera z innymi (znalezionymi w Internecie) klientami.

2.1 server.py

Na podstawie poniższego fragmentu:

```
1 root.add_resource(['.well-known', 'core'],
2                   resource.WKResource(root.get_resources_as_linkheader))
3 root.add_resource(['servo'], ServoResource(17))
4 root.add_resource(['led1'], LEDResource(21))
5 root.add_resource(['led2'], LEDResource(22))
6 root.add_resource(['button1'], ButtonResource(11, lambda: print("Button1 pressed")))
7 root.add_resource(['button2'], ButtonResource(12))
8 root.add_resource(['buzzer'], BuzzerResource(16))
9 root.add_resource(['gpio_buzzer'], GPIOResource(15))
```

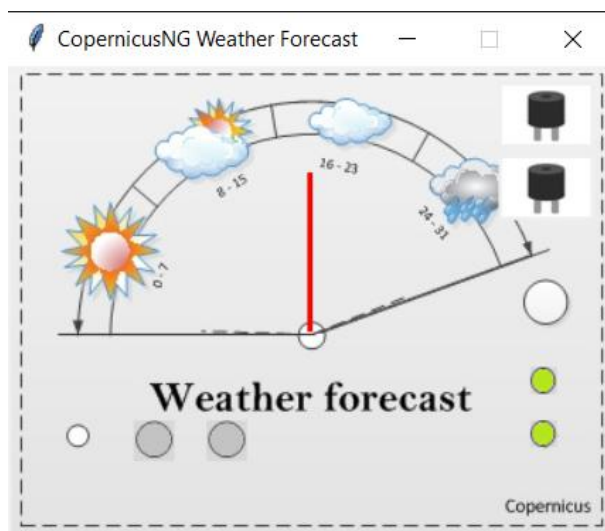
Można zauważyć, że na serwerze obecnie oprócz serwomechanizmu znajdują się takie zasoby jak diody, przyciski, czy głośniki.

2.2 client.py

Za pomocą komendy "g <resource_name>" uzyskamy aktualną wartość interesującego nas zasobu, a za pomocą komendy "s <resource_name> <options>" zmienimy tę wartość.

2.3 Przykład wykonania programów

Po uruchomieniu serwera wyświetlone zostaje następujące okno:



Rysunek 6: Okno pojawiające się po uruchomieniu serwera

Natomiast po uruchomieniu programu testowego dostajemy następujące informacje:

```
e - exit
g name - get resource value
s name v - set resource value to v
```

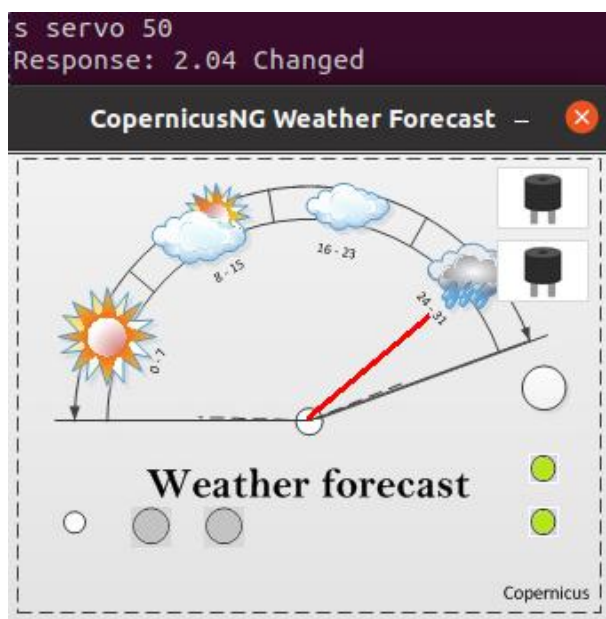
Rysunek 7: Informacje pojawiające się po uruchomieniu programu testowego

Po wykonaniu komendy np. "g servo" otrzymujemy wartość serwomechanizmu (która reprezentuje kąt wychylenia):

```
g servo
Response: 2.05 Content
0.0
```

Rysunek 8: Pobranie wartości serwomechanizmu

Kąt możemy zmienić np. za pomocą komendy "s servo 50":



Rysunek 9: Zmiana wartości serwomechanizmu

Wywołanie komendy "s buzzer beep 1 2 3" spowoduje włączenie głośnika na 1s, wyłączenie go na 2s, a cała operacja wykona się trzykrotnie. Analogiczne miganie diody możemy wykonać za pomocą komendy "s led1 blink 1 2 3".

2.4 Klienci CoAP

Przetestowani zostali następujący klienci:

- <http://manpages.ubuntu.com/manpages/focal/man5/coap-client.5.html>
- https://github.com/gotthardp/gen_coap
- <https://github.com/mkovatsc/Copper4Cr>
- <https://www.eclipse.org/californium/>

2.4.1 coap-client (libcoap2)

Po uruchomieniu serwera na adresie 127.0.0.1 możemy wykonać następujące polecenie w celu pobrania informacji o zasobach:

```
~$ coap-client -m get coap://127.0.0.1:5683/.well-known/core
</.well-known/core>;ct="40 64 504",</servo>;title="Servo Resource - pin: 17",</led1>;title="LED Resource - pin: 21",</led2>;title="LED Resource - pin: 22",</button1>;title="Button Resource - pin: 11",</button2>;title="Button Resource - pin: 12",</buzzer>;title="Buzzer Resource - pin: 16",</gpio_buzzer>;title="GPIO Resource - pin: 15",<https://christian.amsuess.com/tools/aiocoap/#version-0.4b3>;rel="impl-info"
```

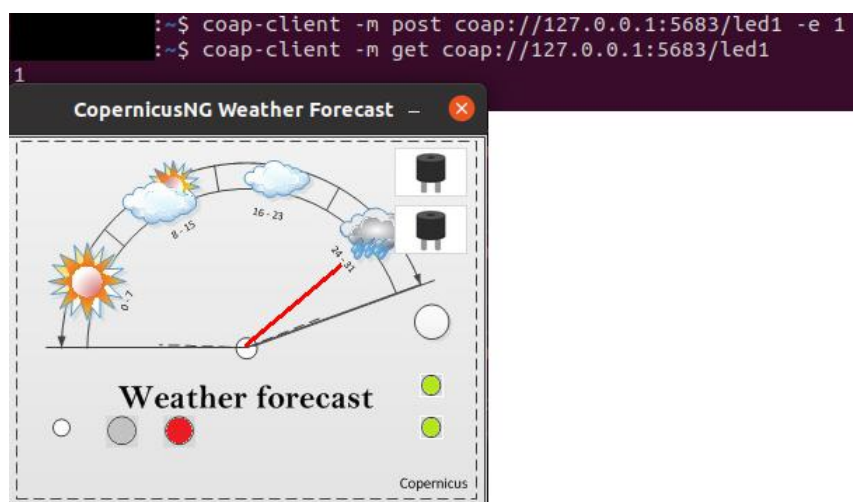
Rysunek 10: Pobranie informacji o zasobach (coap-client)

Informacje o diodzie led możemy uzyskać w następujący sposób:

```
~$ coap-client -m get coap://127.0.0.1:5683/led1
0
```

Rysunek 11: Pobranie informacji o diodzie led (coap-client)

Włączyć diodę (zmienić wartość) możemy w następujący sposób:



Rysunek 12: Włączenie diody led (coap-client)

2.4.2 gen_coap

Informacje o zasobach możemy otrzymać w następujący sposób:

```
$ ./coap-client.sh -m get coap://127.0.0.1/.well-known/core
get "coap://127.0.0.1/.well-known/core"
{ok,content,
  {coap_content,undefined,60,<<"application/link-format">>,
    <<"</.well-known/core>;ct=\"40 64 504\",</servo>;title=\"Servo Resource - pin: 17\",</led1>;t
title=\"LED Resource - pin: 21\",</led2>;title=\"LED Resource - pin: 22\",</button1>;title=\"Button Resource - p
in: 11\",</button2>;title=\"Button Resource - pin: 12\",</buzzer>;title=\"Buzzer Resource - pin: 16\",</gpio_bu
zzer>;title=\"GPIO Resource - pin: 15\",<https://christian.amsuess.com/tools/alocoap/#version-0.4b3>;rel=\"impl
-info\">>}}
```

Rysunek 13: Pobranie informacji o zasobach (gen_coap)

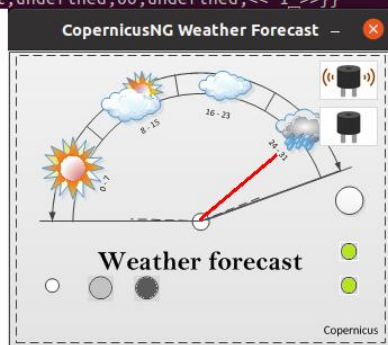
Informacje o głośniku możemy uzyskać w następujący sposób:

```
$ ./coap-client.sh -m get coap://127.0.0.1/buzzer
get "coap://127.0.0.1/buzzer"
{ok,content,{coap_content,undefined,60,undefined,<<"0">>}}
```

Rysunek 14: Pobranie informacji o głośniku (gen_coap)

Włączyć głośnik (zmienić wartość) możemy w następujący sposób:

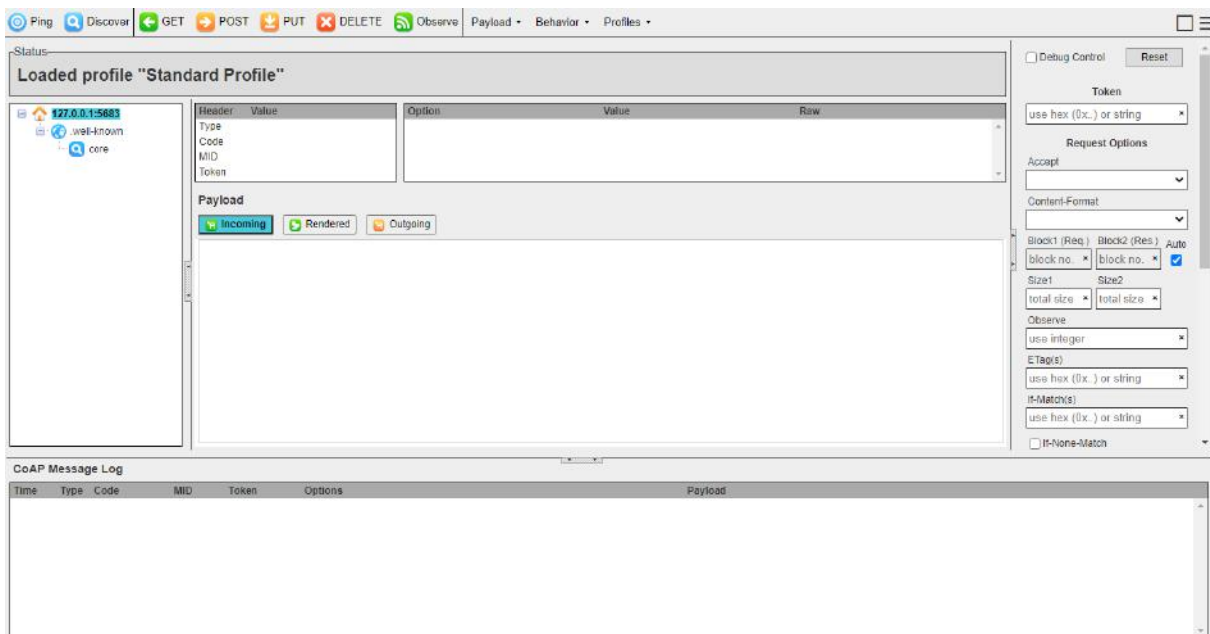
```
$ ./coap-client.sh -m post coap://127.0.0.1/buzzer -e 1
post "coap://127.0.0.1/buzzer"
{ok,changed,{coap_content,undefined,60,undefined,<<">>}}
$ ./coap-client.sh -m get coap://127.0.0.1/buzzer
get "coap://127.0.0.1/buzzer"
{ok,content,{coap_content,undefined,60,undefined,<<"1">>}}
```



Rysunek 15: Włączenie głośnika (gen_coap)

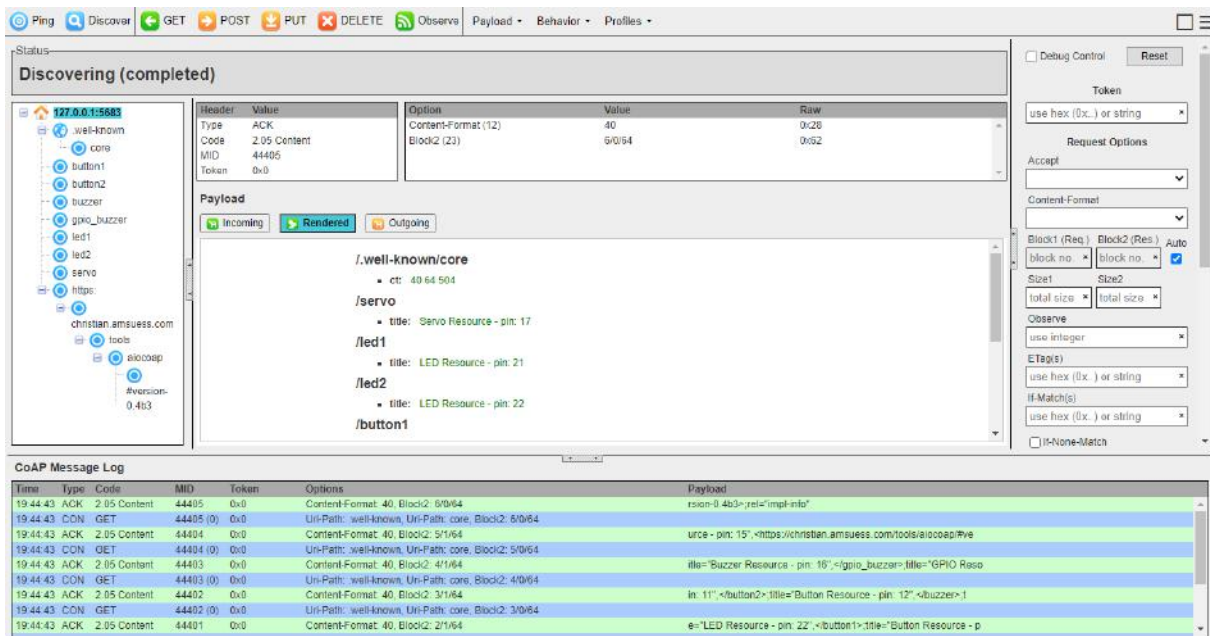
2.4.3 Copper

Po połączeniu z serwerem wyświetlone zostaje następujące okno:



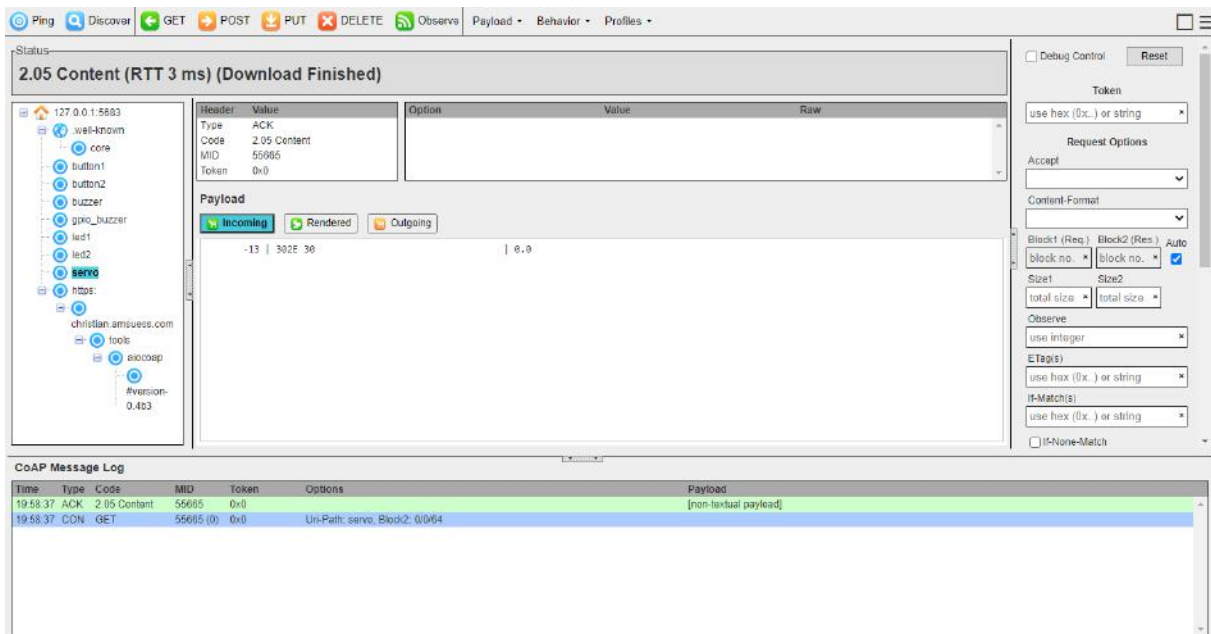
Rysunek 16: Okno pojawiające się po połączeniu z serwerem (Copper)

Po kliknięciu w przycisk "Discover" otrzymujemy informacje o dostępnych zasobach:



Rysunek 17: Uzyskanie informacji o dostępnych zasobach (Copper)

Klikając na interesujący nas zasób, a następnie na przycisk "GET" otrzymujemy aktualną wartość danego zasobu:



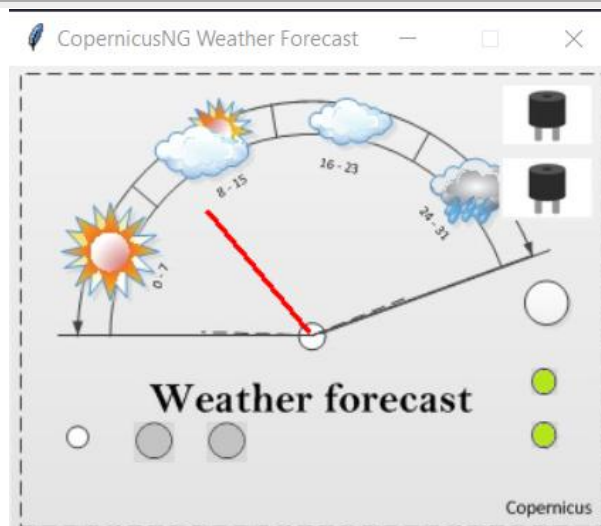
Rysunek 18: Uzyskanie informacji o interesującym nas zasobie (Copper)

Przechodząc na zakładkę "Outgoing", wpisując wartość i klikając przycisk "POST" ustawiamy wartość zasobu:

The screenshot shows the VirtualCopernicusNG web interface. The top navigation bar includes buttons for Ping, Discover, GET, POST, PUT, DELETE, and Observe. The main content area is divided into two sections. The top section, titled "Status", shows a message "2.04 Changed (RTT 3 ms) (Total 3 ms)". Below this, there is a tree view of the device components, including "well-known", "core", "button1", "button2", "buzzer", "gpio_buzzer", "led1", "led2", "servo", "https", "christian.amsuess.com", "tools", "bloccap", and "#version-0.4b3". The bottom section, titled "CoAP Message Log", displays a table of messages:

Time	Type	Code	MID	Token	Options	Payload
20:00:14	ACK	2.04 Changed	55667	0x0		
20:00:14	CON	POST	55667 (0)	0x0	Uri-Path: servo, Content-Format: 0, Block2: 0/0/54	-40
20:00:09	ACK	4.05 Method Not Allowed	55666	0x0		[non-textual payload]
20:00:09	CON	PUT	55665 (0)	0x0	Uri-Path: servo, Content-Format: 0, Block2: 0/0/54	-40
19:58:37	ACK	2.05 Content	55665	0x0		[non-textual payload]
19:58:37	CON	GET	55665 (0)	0x0	Uri-Path: servo, Block2: 0/0/54	

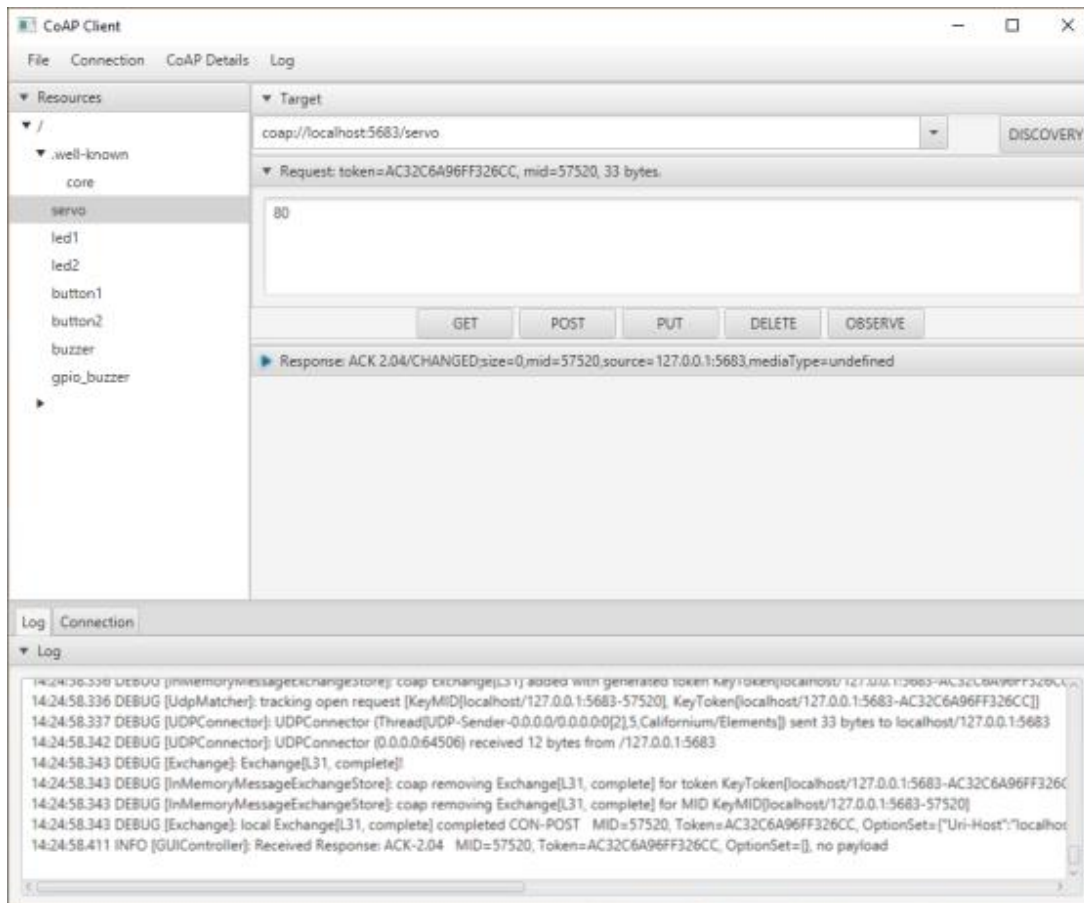
The right sidebar contains various configuration options, including "Debug Control", "Token", "Request Options", "Content-Format", "Block1 (Req.)", "Block2 (Res.)", "Auto", "Size1", "Size2", "total size", "total size", "Observe", "use integer", "ETag(s)", "use hex (0x...) or string", "If-Match(s)", "use hex (0x...) or string", and "If-None-Match".



Rysunek 19: Ustawienie wartości zasobu (Copper)

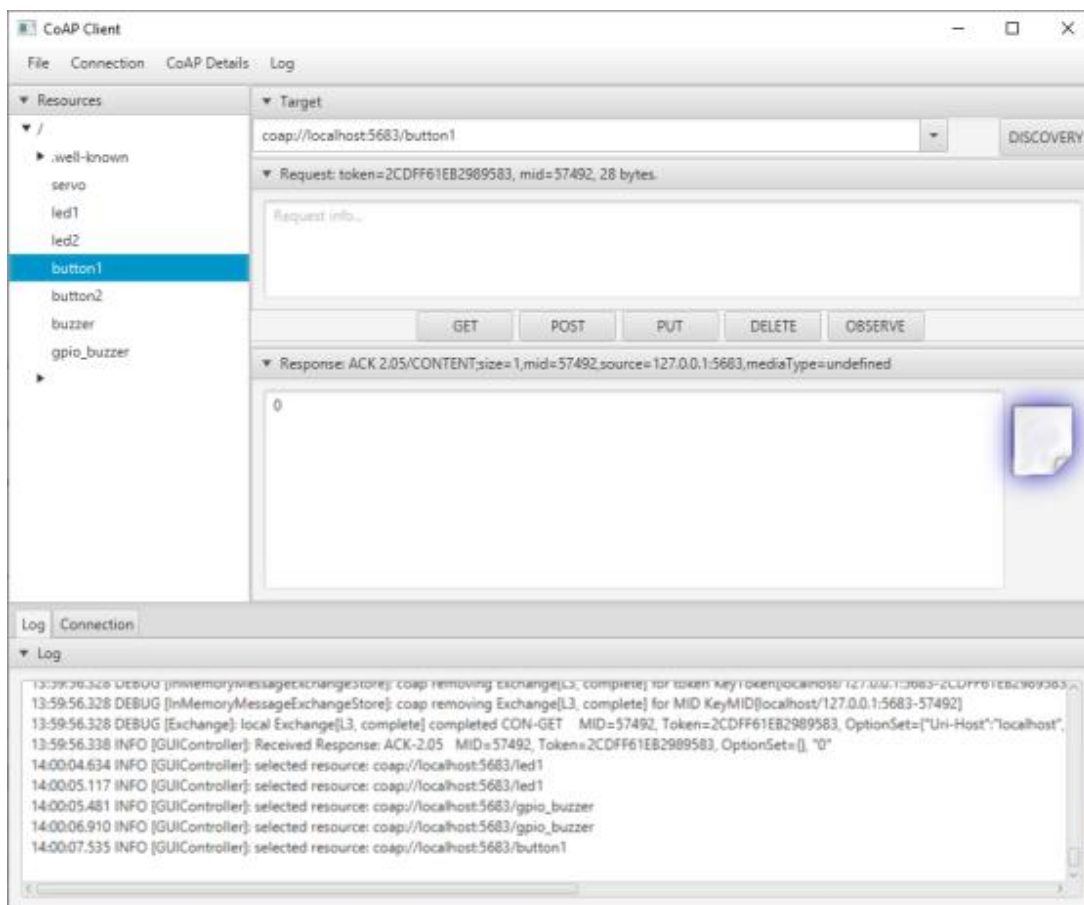
2.4.4 Californium

Zaznaczając interesujący nas zasób i klikając na przycisk "GET" otrzymujemy aktualną wartość tego zasobu:



Rysunek 20: Uzyskanie informacji o interesującym nas zasobie (Californium)

Wpisując wartość i klikając przycisk "POST" ustawiamy wartość zasobu:



Rysunek 21: Ustawienie wartości zasobu (Californium)

3 Postęp prac po trzecich konsultacjach

Udało się dodać obsługę obserwowalnych zasobów oraz uruchmić serwer na Raspberry Pi. Dodany został opis laboratorium wraz z kodem. Dodano możliwość instalacji jako systemd service.

3.1 server.py

Z głównych zmian wprowadzonych w serwerze to:

- Przycisk stał się teraz obserwowalnym zasobem:

```
1 class ButtonResource(resource.ObservableResource):
2     global event_loop
3
4     def get_link_description(self):
5         # Publish additional data in .well-known/core
6         return dict(**super().get_link_description(),
7                     title=f"Button Resource - pin: {self.pin}")
8
9     def __init__(self, pin, callback_p=None, callback_r=None):
10         super().__init__()
11
12         self.pin = pin
13         self.resource = Button(pin)
14         self.callback_p = callback_p
15         self.callback_r = callback_r
16
17         self.resource.when_pressed = self.on_pressed
18         self.resource.when_released = self.on_released
19
20     def on_pressed(self):
21         event_loop.call_soon_threadsafe(self.on_pressed_callback)
22
23     def on_released(self):
24         event_loop.call_soon_threadsafe(self.on_released_callback)
25
26     def on_pressed_callback(self):
27         self.updated_state()
28         if self.callback_p:
29             self.callback_p()
30
31     def update_observation_count(self, newcount):
32         super().update_observation_count(newcount)
33         print(f"{self}: subscribers num: {newcount}")
34
35     def on_released_callback(self):
36         self.updated_state()
37         if self.callback_r:
38             self.callback_r()
```



```
39
40     async def render_get(self, request):
41         print(f'{self}: GET')
42         payload = f"{self.resource.value}"
43         return Message(payload=payload.encode(), code=Code.CONTENT)
44
45     def __str__(self):
46         return f"BUTTON {self.pin}"
```

- Dodano opcje głośnika
- Dodano funkcję obsługującą fizyczne urządzenia oraz zmodyfikowano odpowiednio funkcję main

3.2 client.py

Kod klienta uległ dość dużej modyfikacji. Przede wszystkim dodana została obsługa obserwowalnych zasobów.

```
1 class ObservedResourcesEntry:
2     def __init__(self, requester, cancel):
3         self.requester = requester
4         self.cancel = cancel
5
6
7 class ObservedResources:
8     def __init__(self):
9         self.resources = {}
10
11     def add(self, name, req, cancel):
12         self.resources[name] = ObservedResourcesEntry(req, cancel)
13
14     def stop_observing(self):
15         for cancel_func in map(lambda x: x.cancel, self.resources.values()):
16             cancel_func()
```

3.3 Przykład wykonania programów

Dodana została możliwość obserwowania zasobów, która zostanie zaprezentowana na przykładzie przycisku:



Rysunek 22: Przykład obserwowania zasobu


Po wpisaniu w kliencie komendy "o button1" przycisk znajdujący się na serwerze zostaje dodany do obserwowanych zasobów. Gdy go wciskamy lub puszczamy to po stronie klienta na konsoli otrzymujemy stosowną informację zwrotną.

3.4 Przykładowe programy

Przykładowe programy znajdują się w folderze "examples".

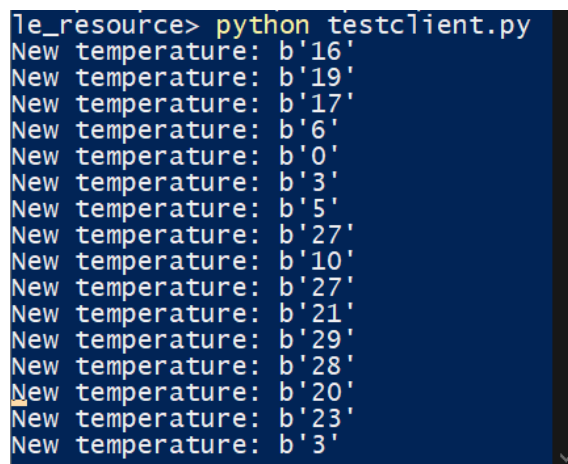
3.4.1 observable_resource

Został napisany program do testowania obserwowalnych zasobów, gdzie klient reaguje na zmiany temperatury na serwerze (gdy dojdzie do zmiany temperatury to po stronie klienta zostaje wyświetlony stosowny komunikat):

A screenshot of a terminal window with a dark blue background. The text is white and shows the execution of a Python script. The first line is a prompt 'le_resource>' followed by the command 'python testServer.py'. The output starts with 'newcount 1' and then lists 15 'Render get requested' messages with varying numerical values: 16, 16, 19, 17, 6, 0, 3, 5, 27, 10, 27, 21, 29, 28, and 20.

```
le_resource> python testServer.py
newcount 1
Render get requested 16
Render get requested 16
Render get requested 19
Render get requested 17
Render get requested 6
Render get requested 0
Render get requested 3
Render get requested 5
Render get requested 27
Render get requested 10
Render get requested 27
Render get requested 21
Render get requested 29
Render get requested 28
Render get requested 20
```

Rysunek 23: Wykonanie testServer.py

A screenshot of a terminal window with a dark blue background. The text is white and shows the execution of a Python script. The first line is a prompt 'le_resource>' followed by the command 'python testclient.py'. The output consists of 15 'New temperature:' messages, each followed by a byte string value in single quotes: b'16', b'19', b'17', b'6', b'0', b'3', b'5', b'27', b'10', b'27', b'21', b'29', b'28', b'20', and b'3'.

```
le_resource> python testclient.py
New temperature: b'16'
New temperature: b'19'
New temperature: b'17'
New temperature: b'6'
New temperature: b'0'
New temperature: b'3'
New temperature: b'5'
New temperature: b'27'
New temperature: b'10'
New temperature: b'27'
New temperature: b'21'
New temperature: b'29'
New temperature: b'28'
New temperature: b'20'
New temperature: b'3'
```

Rysunek 24: Wykonanie testClient.py

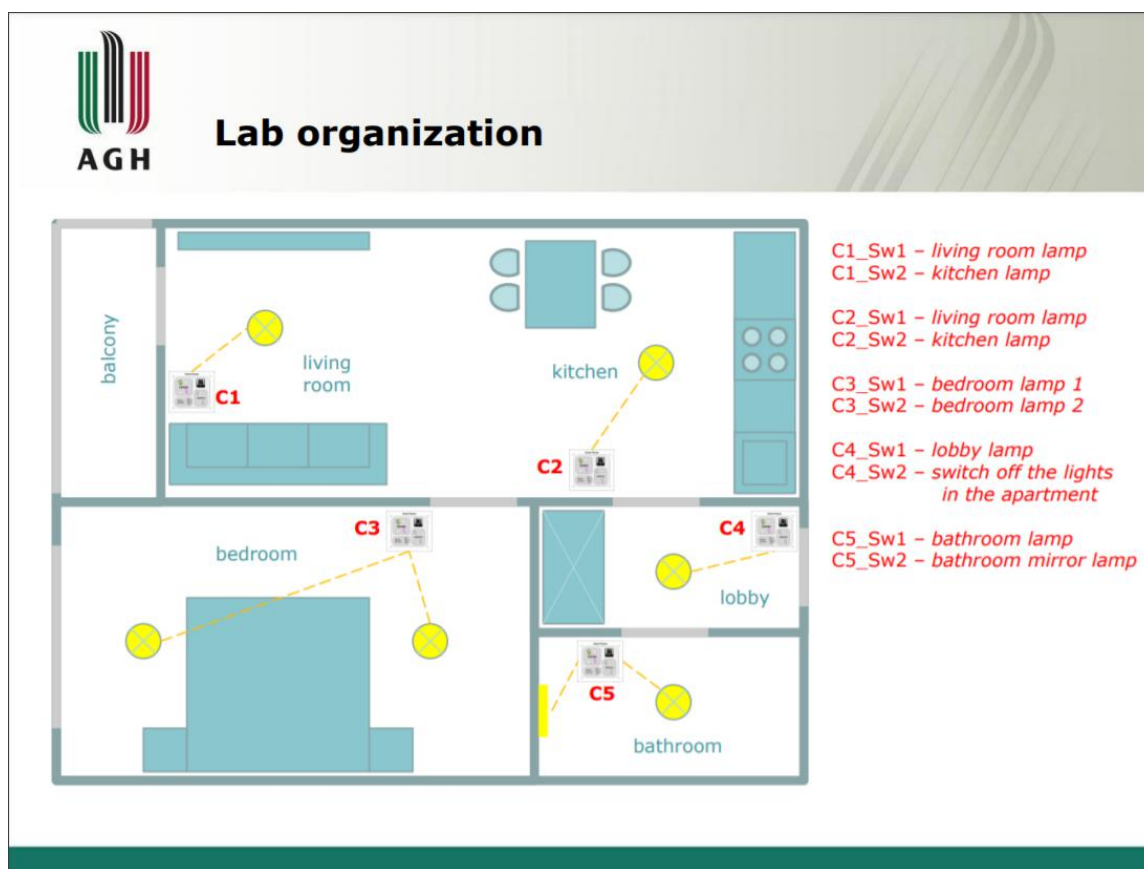
3.4.2 lab - Sterownik świateł wykorzystujący protokół CoAP

Cel

Celem laboratorium jest zbudowanie sterownika świateł wykorzystującego protokół CoAP. Sterownik składa się z urządzeń pełniących funkcję zarówno serwera jak i klienta. Urządzenia te zawierają zasoby w postaci diod LED oraz przyciski, które można obserwować. Posiadają również adres IP (zakładamy, że wszystkie są w tej samej sieci). Za pomocą włącznika na jednym urządzeniu jesteśmy w stanie kontrolować diodę LED znajdującą się na innym urządzeniu. Projekt jest zrealizowany za pomocą serwera zasobów CoAP stworzonego w ramach projektu.

Organizacja pomieszczeń

Jest analogiczna do tej prezentowanej na zajęciach laboratoryjnych:



Realizacja w kodzie

W każdym z urządzeń znajduje się następująca funkcja, wysyłająca request do serwera o adresie przekazanym w argumencie "ip":

```
1  async def observe_button(ip, name, callback):
2      context = await Context.create_client_context()
3      await asyncio.sleep(2)
4
5      request = Message(
6          code=Code.GET,
7          uri=f'coap://{ip}/{name}',
8          observe=0
9      )
10
11     requester = context.request(request)
12
13     async for message in requester.observation:
14         if message.payload == b'1': # button pressed
15             callback()
```

Po wywołaniu powyższej funkcji klient przechodzi w stan nasłuchiwania (async for). Reakcję na odpowiednie zdarzenie możemy przypisać następujący sposób (na przykładzie kitchen.py):

```
1  root.add_resource(['button1'], ButtonResource(11,
2      lambda: led1.resource.toggle(), loop=event_loop))
3  root.add_resource(['button2'], ButtonResource(12, loop=event_loop))
4
5  tasks = []
6
7  asyncio.set_event_loop(event_loop)
8  asyncio.Task(Context.create_server_context(root, bind=(SERVER_IP, 5683)))
9  tasks.append(asyncio.ensure_future(observe_button('127.0.0.3', 'shutter',
10      lambda: led1.resource.off()))))
11  tasks.append(asyncio.ensure_future(observe_button('127.0.0.5', 'button2',
12      lambda: led1.resource.toggle()))))
13  event_loop.run_forever()
```

Dzięki powyższemu zapisowi sprawimy, że po wciśnięciu przycisku 'shutter', znajdującemu się na adresie 127.0.0.3 (reprezentującego lobby) zgasimy światło znajdujące się na obecnym urządzeniu (reprezentującego kuchnię), a po wciśnięciu przycisku 'button2' znajdującemu się na obecnym urządzeniu, na adres 127.0.0.5 (reprezentującego salon) zostanie wysłany request o zmianę stanu światła. Cały kod wraz z gotowymi skryptami uruchamiającymi (zarówno na systemach z rodziny Windows jak i Linux) znajdują się w katalogu examples/lab.

Przykład uruchomienia

Po uruchomieniu skryptu ukaże nam się pięć okienek reprezentujących pięć urządzeń oraz konsola:



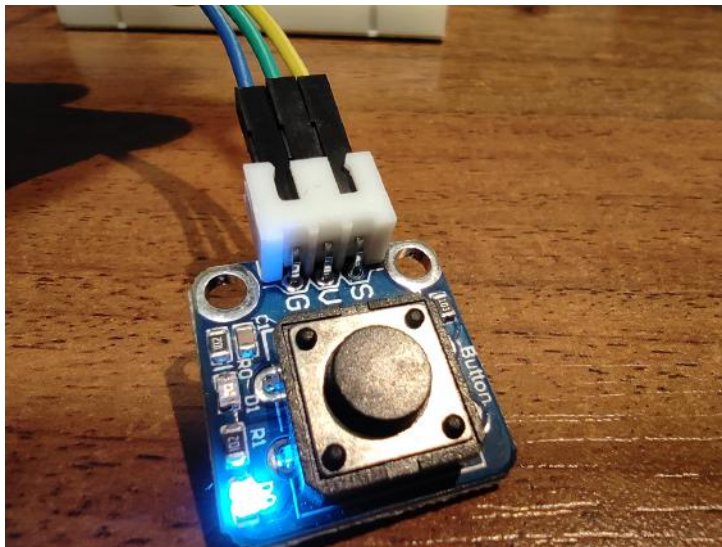
Jak widać na powyższym zrzucie, na konsoli dostajemy informacje o ilości subskrybentów. Odpowiednimi przyciskami możemy odpowiednio zapalać i gasić światła w pomieszczeniach zgodnie z zaprezentowanym wcześniej schematem (np. jeden z przycisków w kuchni steruje światłem w kuchni, a drugi w salonie. Jednym z przycisków znajdujących się w salonie jesteśmy w stanie zgasić światła w całym domu).



3.5 Uruchomienie na Raspberry Pi

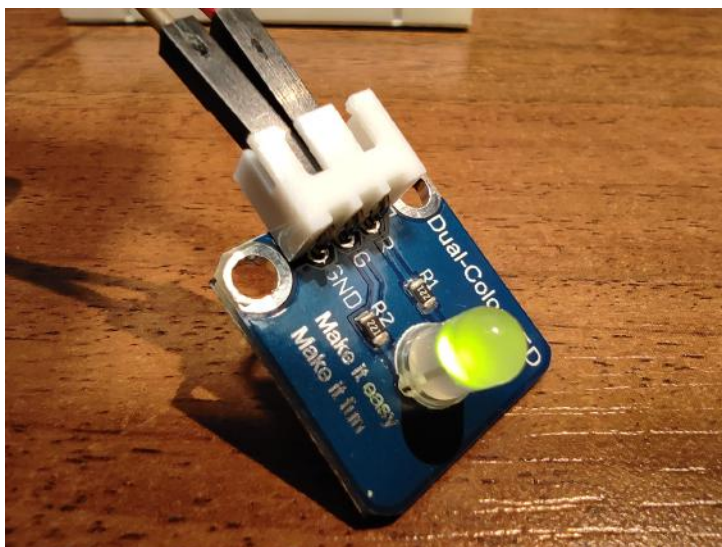
W celu wykonania tego zadania do Raspberry zostały podpięte takie zasoby jak:

- Przycisk:



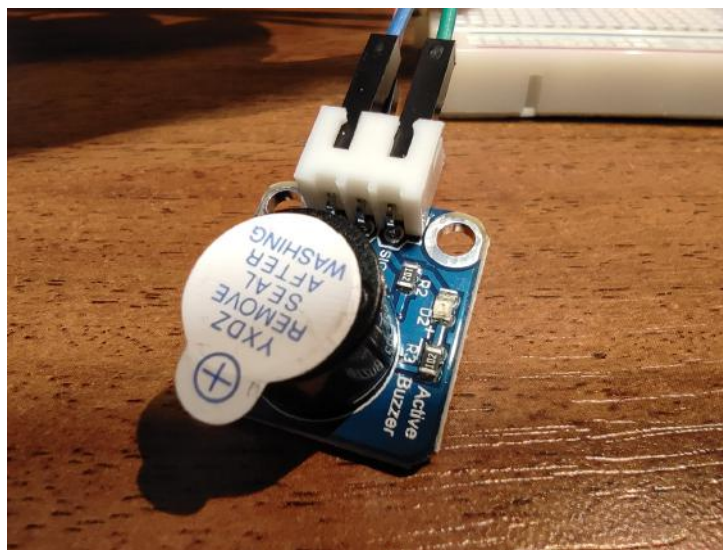
Rysunek 25: Przycisk podpięty do Raspberry Pi

- Dioda LED:



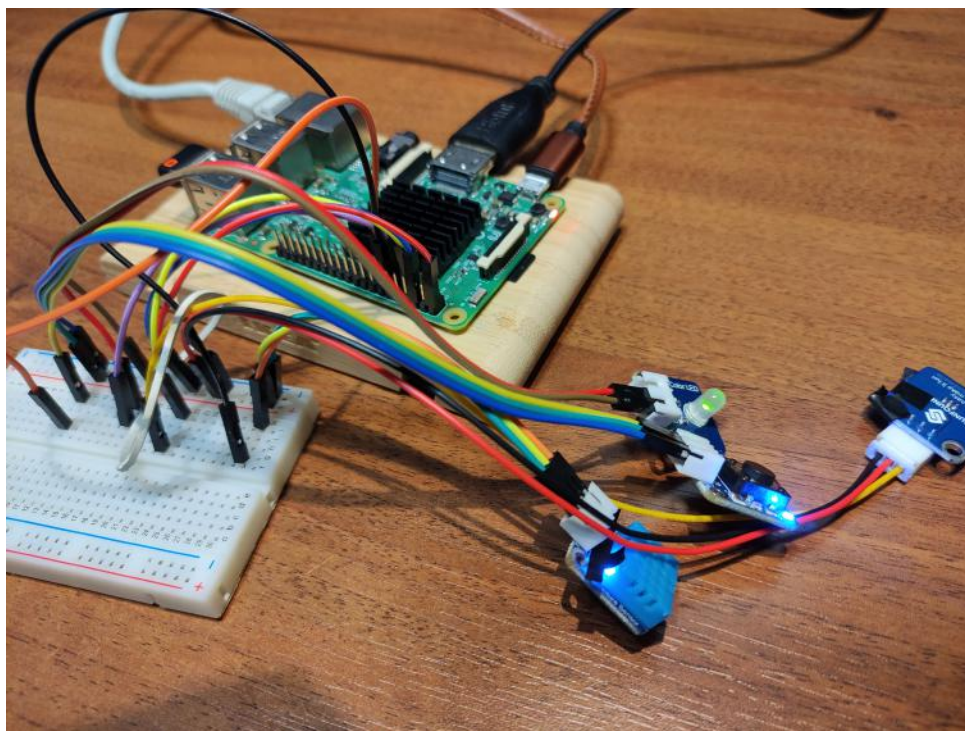
Rysunek 26: Dioda LED podpięta do Raspberry Pi

- Głośnik:

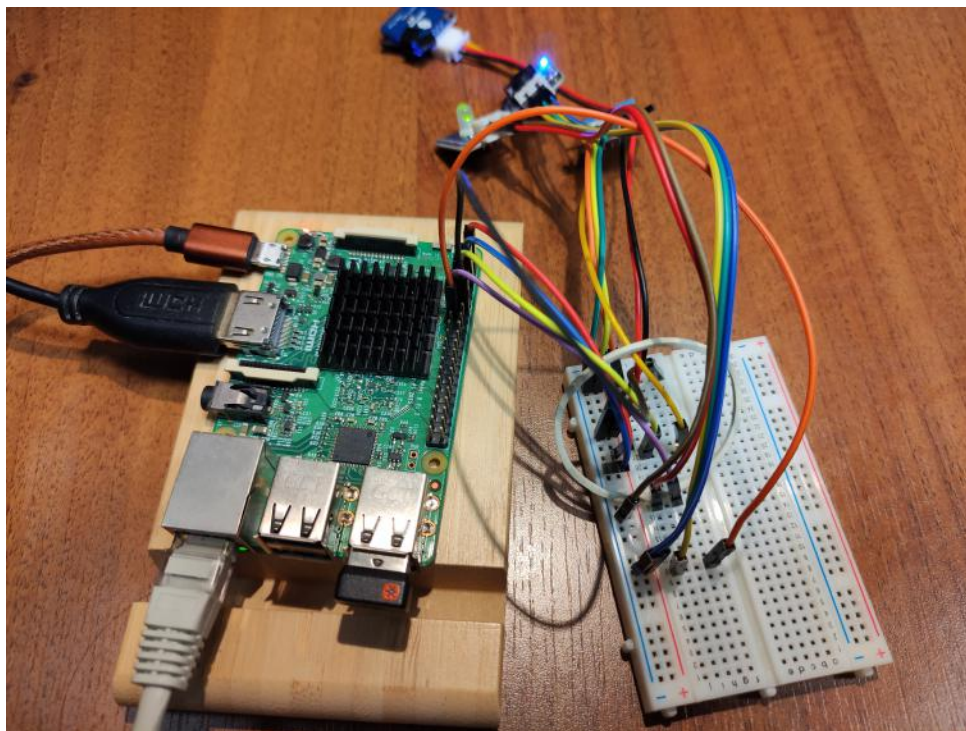


Rysunek 27: Głośnik podpięty do Raspberry Pi

Całość prezentowała się następująco:



Rysunek 28: Raspberry Pi z podłączonymi dodatkowymi komponentami



Rysunek 29: Raspberry Pi z podłączonymi dodatkowymi komponentami

Zadanie zostało wykonane w obie strony. Mianowicie podczas jednych testów Raspberry służyło jako serwer i za pomocą Coperra zostawały uruchamiane, bądź wyłączane poszczególne komponenty. Podczas innych testów natomiast Raspberry służyło jako klient, gdzie za pomocą przycisku podłączonego do płytki włączaliśmy bądź wyłączaliśmy diodę LED znajdującą się na VirtualCopernicusie. Przebieg powyższych testów został pokazany na filmikach znajdujących się pod linkami:

- [Raspberry jako serwer](#)
- [Raspberry jako klient](#)

[Repozytorium github](#)