POLITECHNIKA ŚWIĘTOKRZYSKA

Wydział Elektrotechniki, Automatyki i Informatyki

<table>
<tr><td colspan="2">Nazwa przedmiotu
Budowa i oprogramowanie komputerowych systemów sterowania</td><td>Grupa
1ED11A</td></tr>
<tr><td colspan="2">Wymiana danych w komputerowych systemach sterowania.
Wzór sprawozdania</td><td rowspan="2">Lista ćwiczących
1. Nazwisko i imię</td></tr>
<tr><td>Data wykonania ćwiczenia
12.12.2024</td><td>Data oddania sprawozdania
15.12.2024</td></tr>
<tr><td colspan="2">Uwagi sprawdzającego</td><td>Ocena i podpis</td></tr>
</table>

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z tworzeniem i konfiguracją serwera OPC UA oraz klienta OPC UA, a także realizacja wymiany danych pomiędzy systemami sterowania za pomocą standardu OPC.

2. Zadania

Ćwiczenie rozpoczyna się od uruchomienia serwera OPC UA za pomocą skryptu w języku Python, z odpowiednią konfiguracją i weryfikacją działania na wskazanym porcie. Następnie należy połączyć się z serwerem przy użyciu aplikacji FreeOpcUA, odczytać wartości zmiennych i przeprowadzić ich modyfikację w celu sprawdzenia poprawności komunikacji. W dalszej części uruchamia się klienta OPC UA w języku Python, aby odczytać i wyświetlić w konsoli wartości zmiennych udostępnianych przez serwer, w tym zmienionych wcześniej w aplikacji FreeOpcUA. Na koniec rozszerza się serwer o nowe zmienne różnych typów danych i testuje ich dostępność oraz możliwość modyfikacji z poziomu klienta OPC UA.

2.1. Zadanie 1: Uruchomienie serwera OPC UA za pomocą skryptu w języku Python

```
import sys
sys. path . insert (0 , "..") import time
from opcua import ua , Server


if __name__ == "__main__": # setup our server server = Server ()
    server.set_endpoint("opc.tcp://127.0.0.1:5840/freeopcua/server/")
    idx = server.register_namespace(uri)
    # get Objects node , this is where we should put our nodes
    objects = server.get_objects_node()

    # populating  our  address  space
    myobj = objects.add_object(idx ,"My Object")
    myvar = myobj.add_variable(idx, "My Variable", 6.7)
    myvar.set_writable() # Set My Variable to be writable by clients
    myvar3 = myobj.add_variable(idx, "My Variable 3", "Hello World")

    myvar3.set_writable() # starting ! server. start ()
    try:
        count = 0
        while True:
        time.sleep(1)
        count += 0.1
```

```
    finally :
        # close connection , remove subscriptions , etc
        server.stop()
```

Listing 1. Kod serwera OPC UA w języku Python.

```
D:\LAB_KW_PM\BIOKS\Lab_7>python opc_server.py
Endpoints other than open requested but private key and certificate are not set.
Listening on 127.0.0.1:5840
```

Rysunek 2.1. Wynik działania kodu Python implementującego serwer OPC UA.

Komentarz do zadania:

Serwer OPC UA został uruchomiony za pomocą skryptu opc_server.py, który zawiera konfigurację serwera. Wcześniej zainstalowano Pythona (wersja 3.11) oraz niezbędną bibliotekę OPC UA za pomocą komendy pip install opcua opcua-client. Następ- nie, po przejściu do katalogu z plikiem skryptu, serwer został uruchomiony komendą python opc_server.py. Poprawne uruchomienie zostało potwierdzone komunikatem w konsoli: Listening on 127.0.0.1:5840, co oznacza, że serwer działa pod adresem opc.tcp://localhost:5840/freeopcua/server/. W razie problemów sprawdzono konfigurację portu oraz zaporę sieciową.

2.2. Zadanie 2: Uruchomienie klienta OPC UA przy użyciu aplikacji FreeOpcUA.

Rysunek 2.2. Odczytanie wartości zmiennych udostępnianych przez serwer OPC w poziomu aplikacji FreeOpcUA.

Komentarz do zadania:

Aplikacja FreeOpcUA Client została uruchomiona i nawiązano połączenie z serwerem OPC UA pod adresem opc.tcp://localhost:5840/freeopcua/server/. Po popraw- nym połączeniu wyświetlono strukturę węzłów serwera, w tym obiekt MyObject za- wierający zmienne MyVariable, MyVariable2 oraz MyVariable3. Wartość zmiennej MyVariable wynosiła 6.7 i była poprawnie odczytana w sekcji Attributes. Dodatkowo, zmienne były gotowe do monitorowania w sekcji Graph, co potwierdza prawidłową komu- nikację klienta z serwerem. Logi w dolnym panelu aplikacji wskazywały na prawidłowe połączenie z drobnymi ostrzeżeniami, które nie miały wpływu na działanie aplikacji.

2.3. Zadanie 3: Modyfikacja wartości zmiennych z wykorzystaniem aplikacji FreeOpcUA.

Rysunek 2.3: Zmiana wartości z poziomu aplikacji FreeOpcUA (widok cmd).

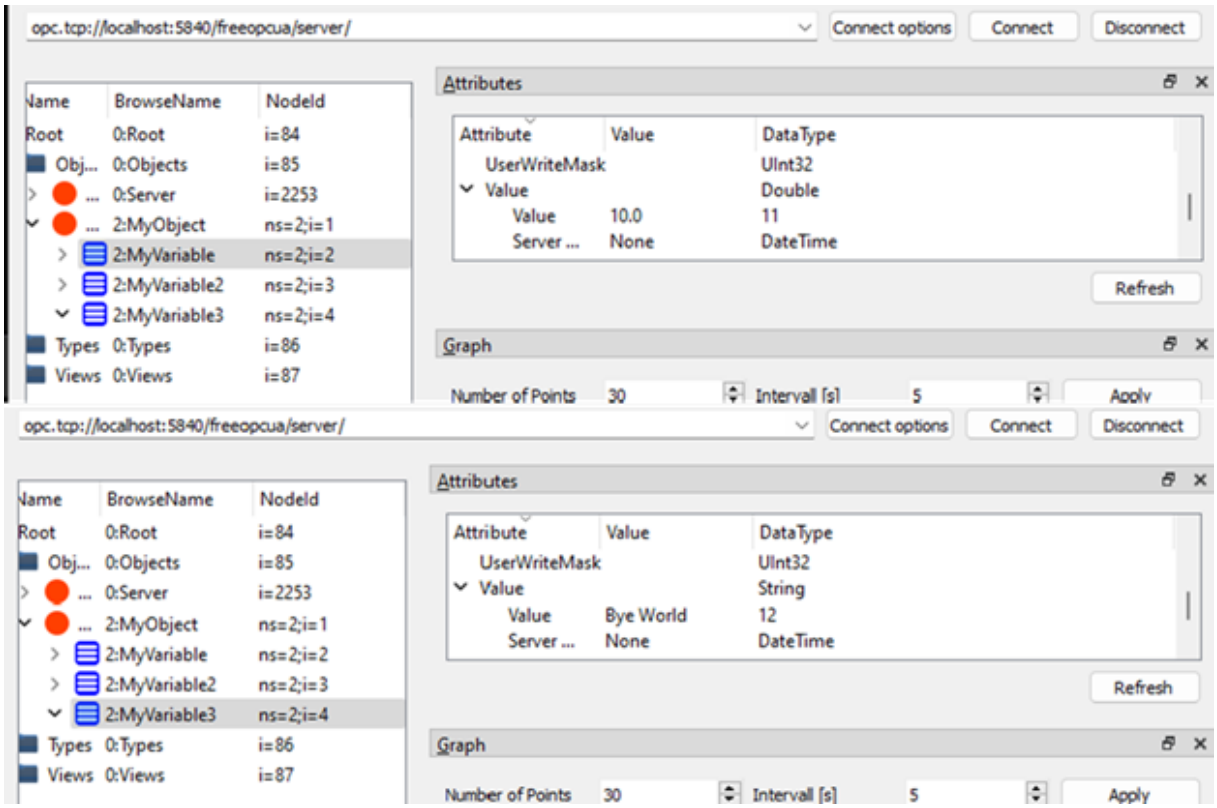
Komentarz do zadania:

W ramach zadania dokonano modyfikacji wartości zmiennych udostępnionych przez ser- wer OPC UA z wykorzystaniem aplikacji FreeOpcUA Client. Po nawiązaniu połączenia z serwerem pod adresem opc.tcp://localhost:5840/freeopcua/server/, użytkow- nik uzyskał dostęp do zmiennych takich jak MyVariable, MyVariable2 i MyVariable3. Następnie zmieniono wartość jednej z tych zmiennych bezpośrednio w aplikacji. Urucho- miony skrypt opc_client.py poprawnie odczytał i wyświetlił zmodyfikowane wartości zmiennych, takie jak 10.0, 25 oraz komunikaty Hello World. Wyniki potwierdziły pra- widłową komunikację klienta z serwerem oraz skuteczne odzwierciedlenie wprowadzonych zmian.

2.4. Zadanie 4: Uruchomienie klienta używając skryptu w języku Python.

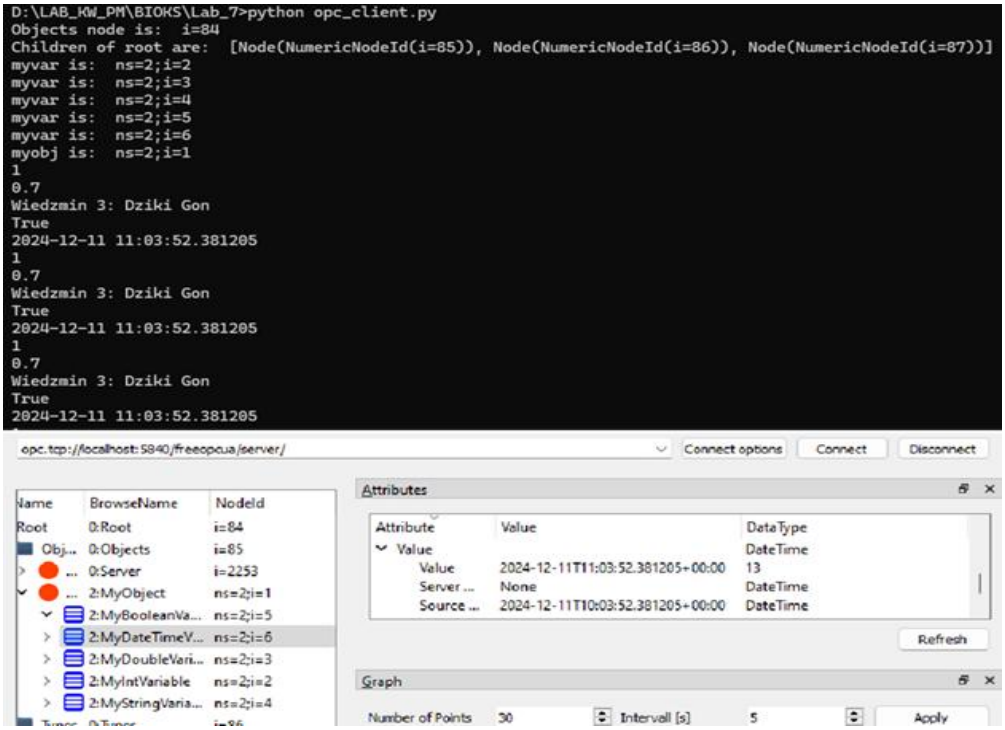
```
import sys
sys.path.insert(0, "..")
from opcua import Client
import time
import DateTime

print(DateTime.DateTime.Date())

if __name__ == "__main__":
    client = Client("opc.tcp://localhost:5840/freeopcua/server/")

    try:
        client.connect()

        # Client has a few methods to get proxy to UA nodes that
        #  should always be in address space such as Root or Objects
        root = client.get_root_node()
        print("Objects node is: ", root)

        # Node objects have methods to read and write node attributes
        #  as well as browse or populate address space
        print("Children of root are: ", root.get_children())

        # Now getting a variable node using its browse path
        myvar = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable"])
        myvar1 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable2"])
        myvar2 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable3"])
        obj = root.get_child(["0:Objects", "2:MyObject"])

        print("myvar is: ", myvar)
        print("myvar is: ", myvar1)
        print("myvar is: ", myvar2)
        print("myobj is: ", obj)
        while True:
            print(myvar.get_value())
            print(myvar1.get_value())
            print(myvar2.get_value())
            time.sleep(1)

    finally:
        client.disconnect()
```

Listing 2: Kod klienta OPC UA w języku Python.



Rysunek 2.4: Sprawdzenie, czy wartości zmiennych wyświetlane w konsoli są zgodne z ustawionymi w aplikacji FreeOpcUA.

2.5. Zadnie 5: Rozszerzenie serwera o dodatkowe zmienne.

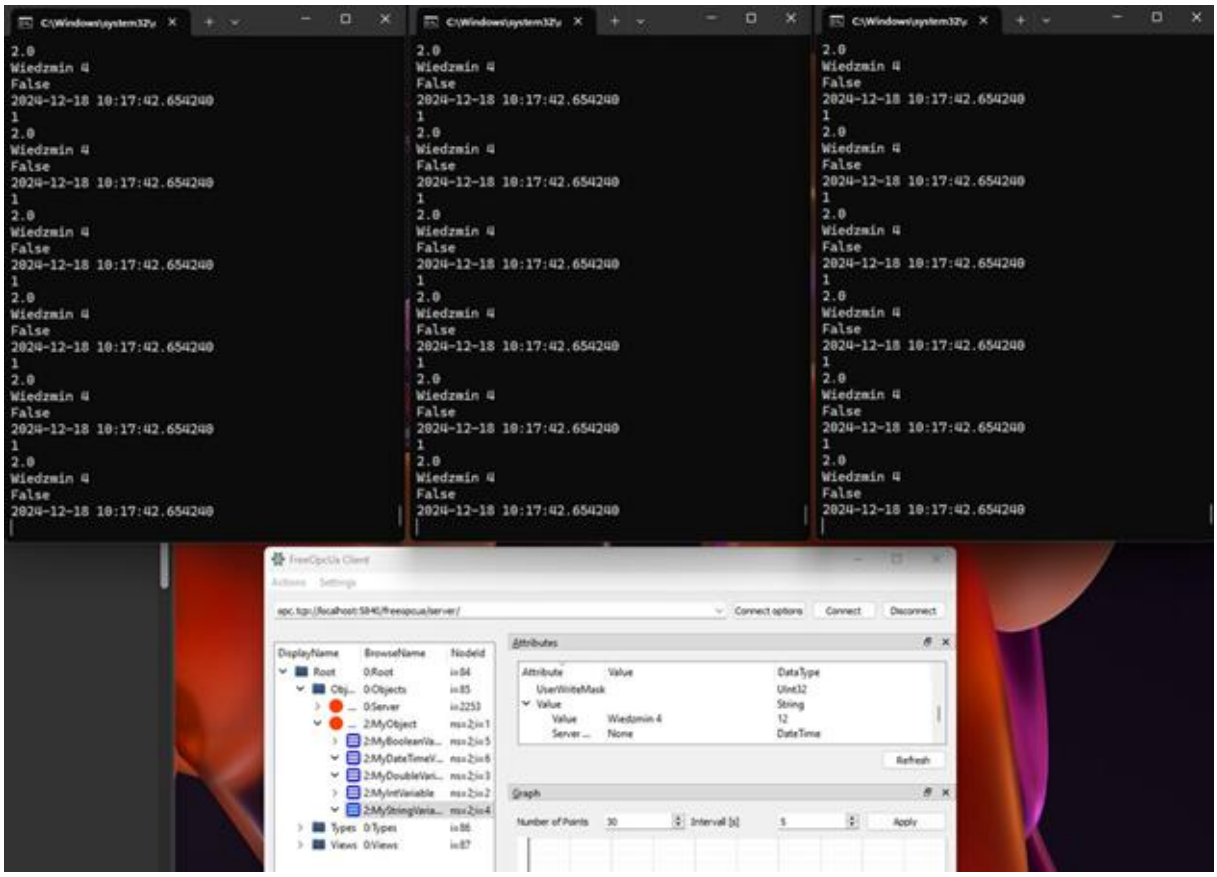


Rysunek 2.5: Dodanie 5 nowych zmiennych o różnych typach danych, oraz uruchomienie serwera i sprawdzenie, czy nowe zmienne są widoczne w aplikacji FreeOpcUA.

Komentarz do zadania:

Serwer OPC UA został rozszerzony o pięć nowych zmiennych o różnych typach danych: MyIntVariable (Integer), MyDoubleVariable (Double), MyStringVariable (String), MyBooleanVariable (Boolean) oraz MyDateTimeVariable (DateTime). Zmiany war- tości tych zmiennych zostały poprawnie odczytane i zapisane, co potwierdziły wyniki wyświetlane w aplikacji FreeOpcUA Client.

2.6. Zadanie 6: Sprawdzenie działania wielu instancji klienta OPC.



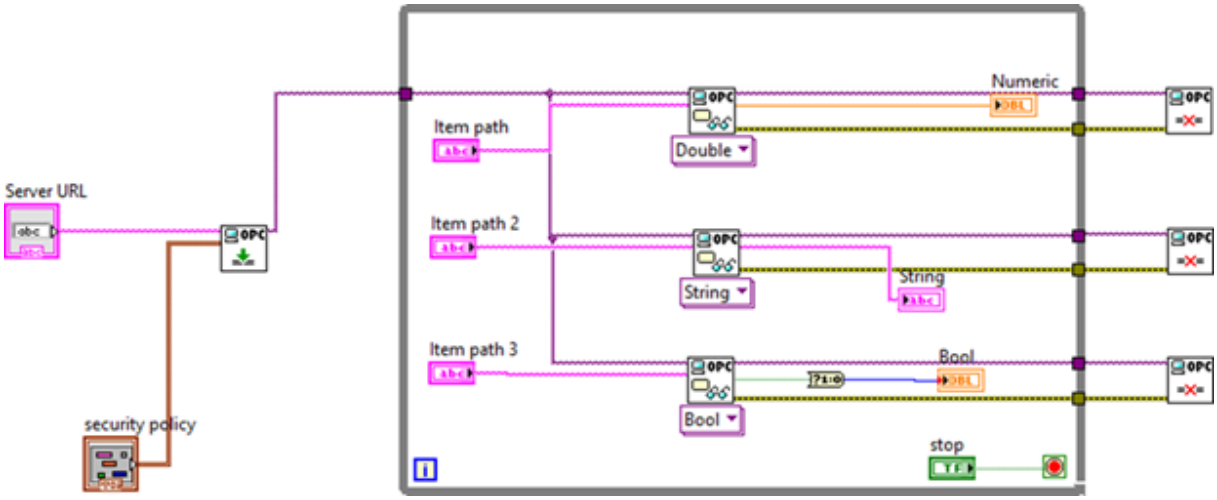
Rysunek 2.6: Sprawdzenie działania trzech instancji klienta OPC.

Komentarz do zadania:

Trzy instancje skryptu opc_client.py zostały uruchomione w osobnych oknach termi- nala, a każda z nich poprawnie odczytuje i wyświetla aktualne wartości zmiennych z serwera OPC UA, takie jak: 2.0 (Double), "Wiedźmin 4" (String), False (Boolean) oraz znaczniki czasu (DateTime). W aplikacji FreeOpcUA Client widoczne są zmienne serwera: MyDoubleVariable, MyStringVariable, MyBooleanVariable i MyDateTimeVariable. Wszystkie klienty poprawnie synchronizują się z serwerem i odczytują te same wartości zmiennych w czasie rzeczywistym, co potwierdza stabilność

serwera podczas obsługi wielu równoczesnych połączeń.

2.7. Zadanie 7: Utworzyć klienta OPC UA w programie LabView, odczytującego dane z serwera.



Rysunek 2.7: Block diagram zadania 7 - klient OPC UA odczytujący dane.

Rysunek 2.8: Front panel zadania 7 - klient OPC UA odczytujący dane.

```
D:\LAB_KW_PM\BIOKS\Lab_7>python opc_client.py
Objects node is:  i=84
Children of root are:  [Node(NumericNodeId(i=85)), Node(NumericNodeId(i=86)), Node(NumericNodeId(i=87))]
myvar is:  ns=2;i=2
myvar is:  ns=2;i=3
myvar is:  ns=2;i=4
myvar is:  ns=2;i=5
myvar is:  ns=2;i=6
myobj is:  ns=2;i=1
1
10.0
Stalker 2
True
2024-12-18 10:17:42.654240
1
10.0
Stalker 2
True
```

Rysunek 2.9: Podgląd danych przez clienta w cmd.

Komentarz do zadania:

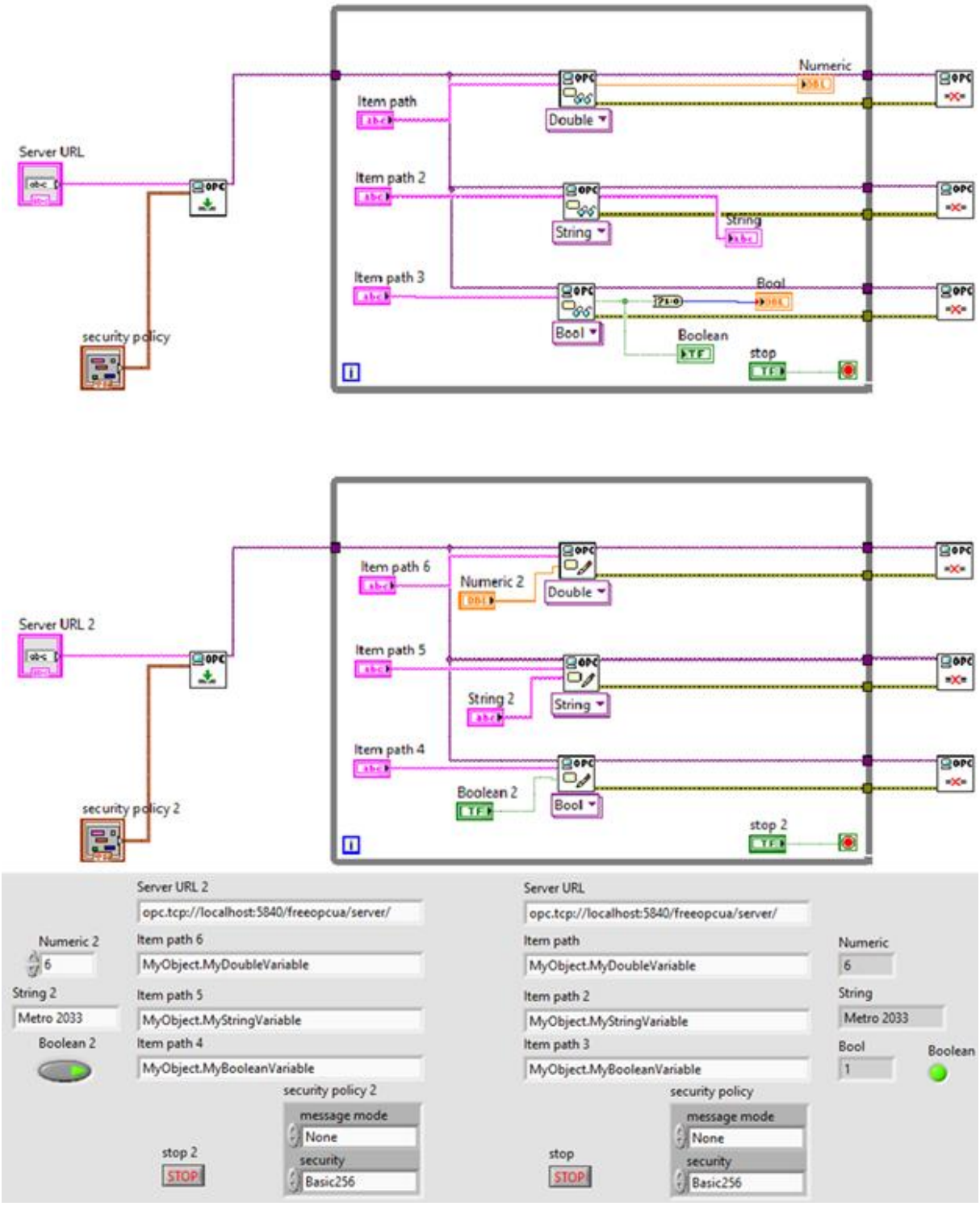
W ramach zadania utworzono klienta OPC UA w środowisku LabVIEW, który odczytuje dane z serwera OPC UA uruchomionego pod adresem opc.tcp://localhost:5840/freeopcua/server/. Klient został skonfigurowany do odczytu zmiennych: MyDoubleVariable, MyStringVariable oraz MyBooleanVariable. Wartości tych zmiennych wyświetlane w interfejsie graficznym wynosiły odpowiednio: 10 (Double), "Stalker 2" (String) oraz 1 (Boolean). Diagram blokowy LabVIEW przed- stawiał poprawne użycie funkcji OPC UA do odczytu danych, co zostało potwierdzone zgodnością wyników z odczytem dokonanym przez skrypt opc_client.py.

2.8. Zadanie 8, 9, 10

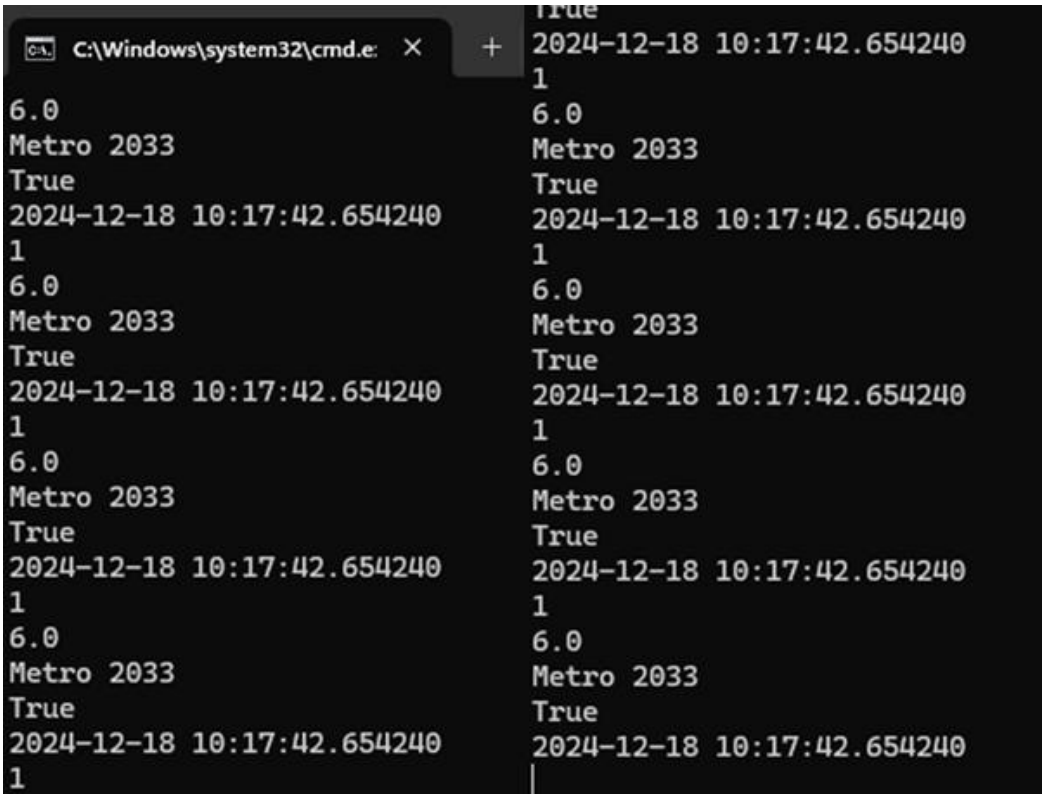
Zadanie 8: Utworzyć klienta OPC UA w programie LabView, zapisującego dane do serwera.

Zadanie 9: Rozszerzyć zadania 7 i 8 o dodatkowy odczyt i zapis zmiennych używając klienta OPC UA.

Zadanie 10: Dokonać wymiany danych między programami.



Rysunek 2.10: Program Labview, realizujący polecenia zadań: 8, 9 i 10.



Rysunek 2.11: Podgląd wyników działania w konsoli cmd.

Komentarz do zadania:

W ramach zadań 8, 9 i 10 utworzono klienta OPC UA w środowisku LabVIEW, który umożliwiał zarówno zapis, jak i odczyt danych z serwera OPC UA. Dwie niezależne instancje klienta zostały skonfigurowane do komunikacji z serwerem pod adresem opc.tcp://localhost:5840/freeopcua/server/, obsługując zmienne: MyDoubleVariable, MyStringVariable oraz MyBooleanVariable. Wartości zmiennych zostały poprawnie odczytane i zapisane, co potwierdzają wartości: 6.0 (Double), "Metro 2033" (String) oraz True (Boolean), widoczne zarówno w interfejsie graficznym LabVIEW, jak i w konsoli klienta Python.

3. Wnioski

Realizacja ćwiczenia pozwoliła na praktyczne zapoznanie się z konfiguracją i uruchomie- niem serwera OPC UA oraz klienta OPC UA, co umożliwiło efektywną wymianę danych w systemach sterowania. Dzięki implementacji serwera w języku Python oraz wykorzystaniu aplikacji FreeOpcUA Client, możliwe było skuteczne odczytywanie i modyfikowanie zmiennych w czasie rzeczywistym.

Rozszerzenie serwera o dodatkowe zmienne różnych typów danych potwierdziło

popraw- ność działania systemu oraz elastyczność protokołu OPC UA w obsłudze różnych typów danych. Testy z wieloma instancjami klienta potwierdziły stabilność i niezawodność serwera w przypadku równoczesnych połączeń.

Integracja klienta OPC UA w środowisku LabVIEW umożliwiła zarówno odczyt, jak i zapis danych, co zostało potwierdzone poprawnym działaniem podczas komunikacji z serwerem OPC UA. Wykonanie zadania z wykorzystaniem programów CODESYS i LabVIEW potwierdziło możliwość wymiany danych między różnymi środowiskami programistycznymi, co może okazać się przydatne przy opracowaniu projektów automatyki przemysłowej.