Universidade Federal de Itajubá - UNIFEI
Instituto de Engenharia de Sistemas e Tecnologia da Informação - IESTI

Guia de Desenvolvimento
Supervisório Web

Rafael Coelho Paes - 2019000081
Orientador: Prof. Dr. Jeremias Machado Barbosa

1 Introdução

Este guia tem como objetivo demonstrar uma aplicação de desenvolvimento de um sistema supervisório *web* usando ferramentas muito presentes no mundo da automação, como Python (e suas diversas bibliotecas), banco de dados MySQL, padrão de comunicação OPC UA, etc.

Existem diversas formas de criar uma plataforma web para um supervisório de um processo de automação. Um ponto crucial para o desenvolvimento de um sistema SCADA é conseguir, através de algum método, acesso às variáveis do processo (PV). Uma das ferramentas que torna isso possível é o padrão de comunicação *OPC UA*.

Este guia não irá cobrir a parte de criação de um servidor OPC UA, uma vez que esta etapa pode ser alcançada de diversas maneiras com diferentes ferramentas de automação, tais como CODESYS ou até mesmo o software *AutomationStudio*, que será usado durante o passo a passo.

Se tratando de linguagens de programação, muitas linguagens oferecem bibliotecas com suporte à criação de clientes OPC UA para se obter acesso às variáveis disponíveis no servidor, como por exemplo a biblioteca node-opcua para as linguagens *JavaScript* e *TypeScript*, com integração de ferramentas como *NodeJS*, que juntos são ótimos recursos para criação de páginas web. Também existe a biblioteca python-opcua para a linguagem *Python*, que será usada neste guia, juntamente com a biblioteca *Streamlit*, que fornece ferramentas para criação rápida e fácil de páginas web.

Uma vez que a ideia de capturar as variáveis do processo esteja consolidada, sinta-se à vontade para usar a linguagem de programação de sua preferência e de melhor proveito para sua aplicação para a criação do seu próprio supervisório!

Para preparar o seu ambiente de desenvolvimento, segue a lista das bibliotecas que podem ser necessárias, com uma breve descrição sobre suas aplicações:

- **streamlit** - Criação de páginas Web
- **streamlit_authenticator** - Módulo de autenticação
- **asyncua** - Cliente OPC UA assíncrono para acesso às variáveis
- **asyncio** - Criação de programas assíncronos
- **mysql.connector** - Criação e Manipulação de bancos de dados SQL
- **plotly.express** - Plotagem de gráficos
- **pandas** - Análise de dados
- **datetime** - Ferramentas para tratar variáveis de tempo
- **yaml** - Manipulação de arquivos .yaml

Caso não possua alguma das bibliotecas listadas, instale-a com o simples comando *pip install* [nome da biblioteca] pelo **CMD** de seu computador.

2 Comunicação com servidor OPC UA

3 Cliente OPC UA

Uma vez que se tenha um servidor OPC UA estabelecido, é recomendado utilizar um *software* auxiliar como cliente OPC UA. Uma opção de fácil acesso e gratuita se trata do programa *UaExpert*, que pode ser baixado no site oficial do programa.

Uma vez instalado, basta abrir o programa e adicionar o servidor OPC UA desejado, que no caso será aquele gerado pelo *software* de automação. Para fazer isso, basta clicar na pasta **Servers** com o botão direito e clicar em **add**, como mostra a Figura 1

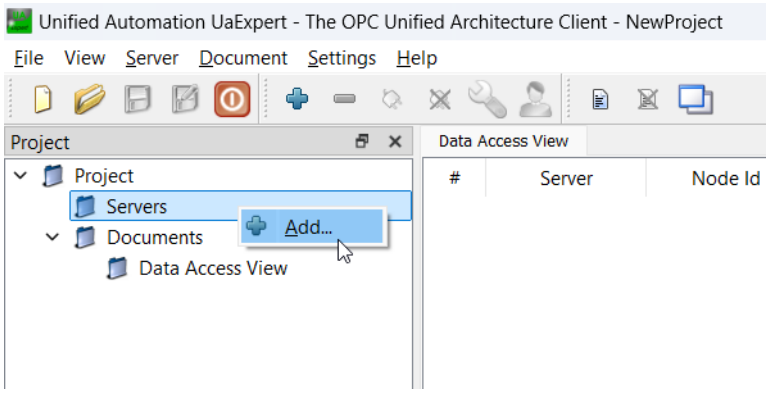

Figura 1

Após isso uma janela irá abrir, oferecendo uma série de opções para se comunicar com um servidor (Figura 2). Na figura, um servidor OPC UA local já está sendo identificado, mas caso isso não ocorra, você terá de procurar o servidor através das opções que o programa oferece.

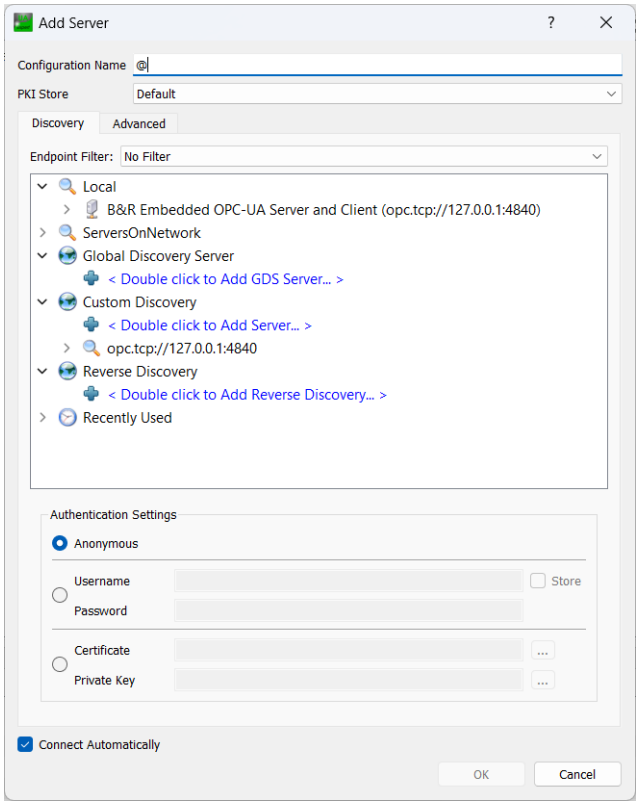

Figura 2

Caso as opções da aba **Discovery** não funcione (o que é possível ocorrer), tente manualmente através da aba **Advanced** (Figura 3)

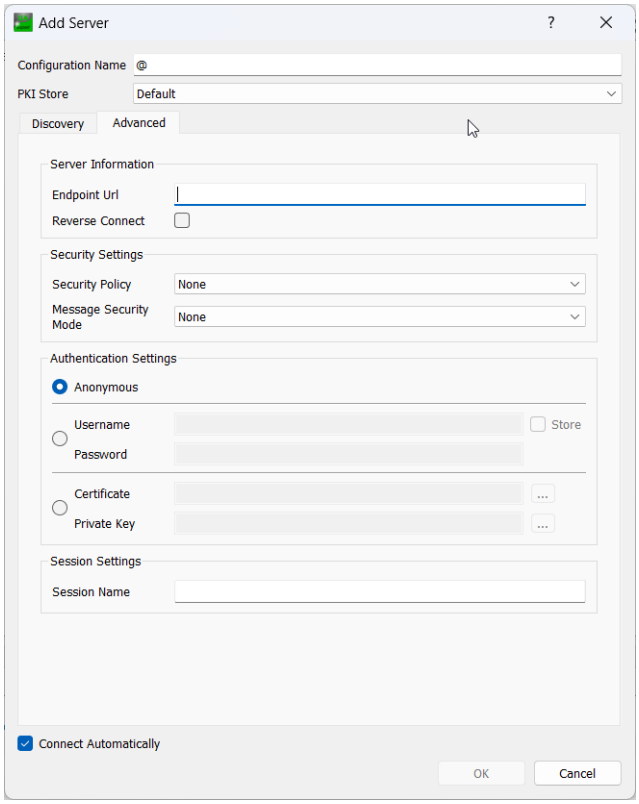

Figura 3

Nesta janela, você deverá preencher manualmente as informações do servidor, tal como o IP do CLP e o port para acesso ao servidor OPC UA. É possível também escolher um método de segurança e criptografia se necessário.

O campo de IP deve ser preenchido com um valor no formato **opc.tcp://IP:PORT**. O **IP** diz respeito ao IP do CLP na rede, enquanto o PORT é a porta de acesso ao servidor OPC UA sendo gerido pelo controlador. No *AutomationStudio*, é possível conferir estas informações na janela **Configuration** do CLP, como mostra a Figura 4. Atente-se que o IP mostrado nesse caso é para uma aplicação **simulada**. Em uma atividade prática real o IP deverá ser o endereço real do CLP na rede.

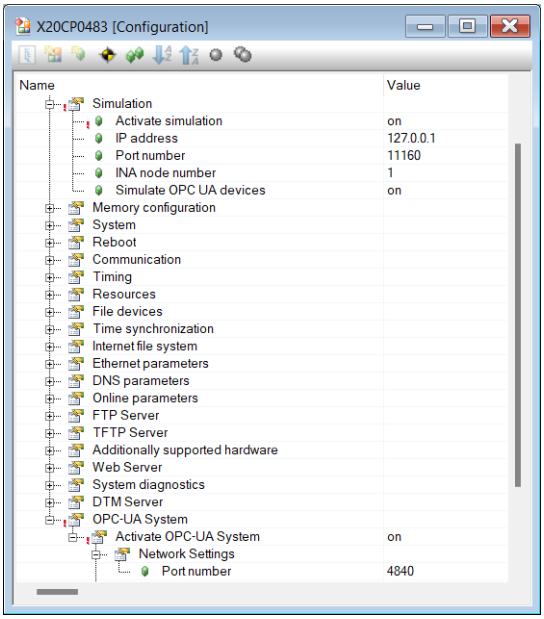

Figura 4

Preencha as informações corretas no cliente **UaExpert** e clique em OK. Para esse caso não será necessário nenhuma configuração de segurança, porém, atente-se ao campo de autenticação **Authentication Settings**: No momento é possível comunicar com o servidor OPC UA no modo **Anônimo**, pois as configurações de acesso e funções de usuários ainda não foram alteradas no *AutomationStudio*, que tem por padrão o usuário **Anonymous**, como mostra a Figura 5. Portanto, para se ter acesso ao servidor após remover ou alterar o nome deste usuário anônimo, será necessário logar com outro usuário colocando manualmente o *nome de usuário* e *senha* (se existir) nos campos de autenticação.

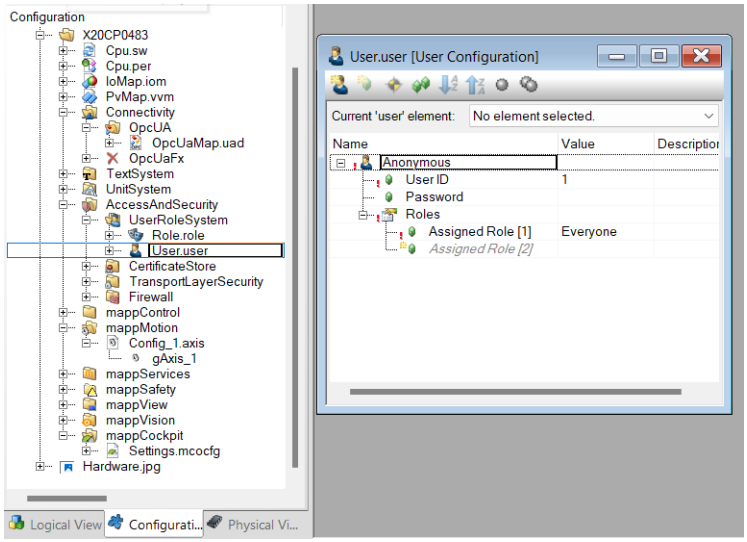

Figura 5

Ao clicar em OK na janela de conexão OPC UA, será solicitado uma validação de certificado (Figura 6). Clique em **Trust Server Certificate** e depois em **Continue**. Após isso, já haverá uma conexão estabelecida entre cliente e servidor (se não for o caso, clique com o botão direito no servidor que apareceu na pasta *Servers* e em seguida clique em *Connect*).

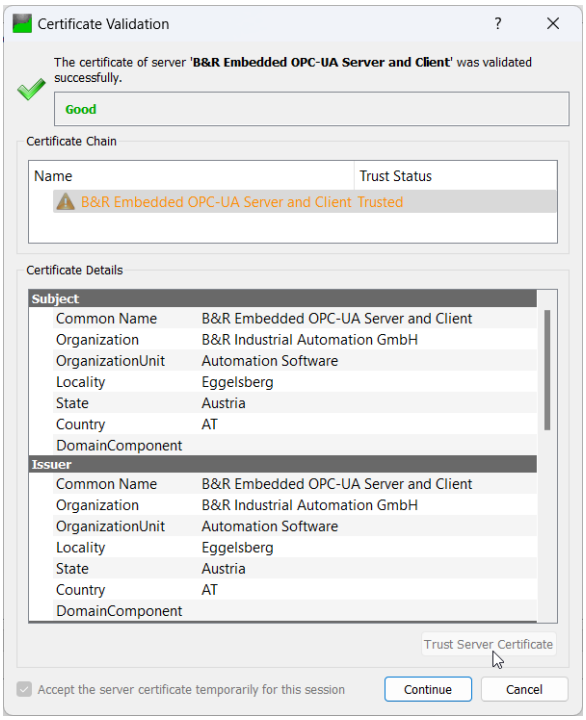

Figura 6

Com uma conexão estabelecida, é possível notar que a janela **Address Space** agora possui uma série de itens. Estes itens são os objetos compartilhados pelo servidor. Nesta árvore é possível encontrar as variáveis do programa rodando no CLP que estão disponíveis no servidor. A Figura 7 mostra o caminho para encontrar algumas variáveis de processo para uma aplicação de *motion*.

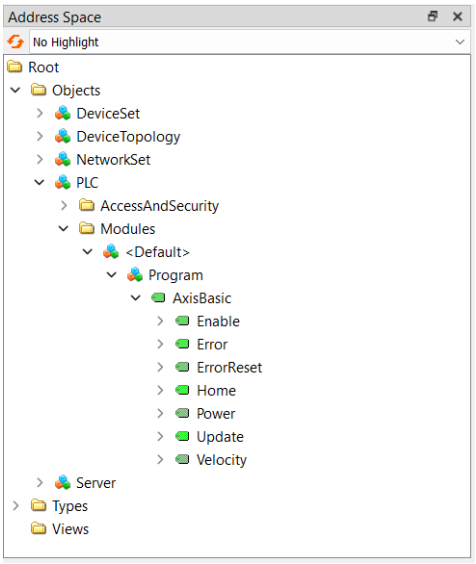

Figura 7

Ao selecionar uma destas variáveis, é possível inspecionar os diversos atributos dela na janela à direita, de nome **Attributes** (Figura 8). Nesta lista conseguimos ter acesso à diversas informações da variável, incluindo **Tipo**, **Valor**, **Nível de Acesso de Leitura e Escrita**, **TimeStamp**, e, talvez o mais importante, o endereço identificador do nó **NodeId**, que é crucial para criação de um cliente OPC UA externo, ideia que será aplicada em scripts *Python*.

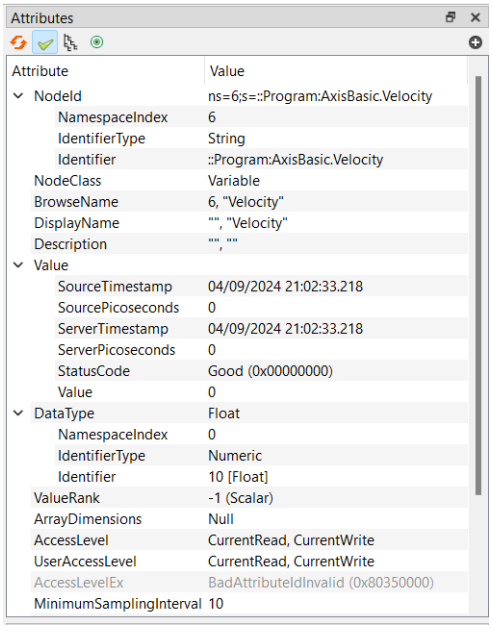

Figura 8

No momento, temos um servidor OPC UA e a garantia de que as variáveis do processo desejadas estão disponíveis neste servidor, além de possuir os endereços de nó destas. Portanto, tudo está preparado para o desenvolvimento de um programa em Python que irá executar a **Leitura** e **Escrita** nestas variáveis.

4 Utilizando bibliotecas Python para criação de um cliente OPC UA

O script abaixo apresenta um exemplo de leitura e escrita em uma variável disponível no servidor OPC UA. Toda a explicação está nos comentários do código.

```python
# Bibliotecas necessárias para comunicação com servidor OPC UA
import asyncio
import asyncua as ua
from asyncua import *

# Endereço do servidor para conexão com o cliente OPC UA
url = "opc.tcp://127.0.0.1:4840/"

# Endereços base OPC UA dos nós utilizados
# Nó base - todas as variáveis vão possuir este prefixo
base_node = "ns=6;s="
# Nó de controle - variáveis do programa Program
# do tipo MpAxisBasic de nome AxisBasic
control_node = "::Program:AxisBasic."
# Nó de parametrização - variáveis do programa Program
# do tipo MpAxisBasicParType de nome AxisParam
param_node = "::Program:AxisParam."

# Variável de potência do motor AxisBasic.Power
cPower = base_node + control_node + "Power"
# Variável de acionamento do movimento AxisBasic.MoveVelocity
cMoveVelocity = base_node + control_node + "MoveVelocity"

# Variável de velocidade AxisBasic.Velocity (para monitoramento - leitura)
cVelocity = base_node + control_node + "Velocity"

# Variável de velocidade AxisParam.Velocity para escrita
pVelocity = base_node + param_node + "Velocity"

async def main():
    print(f"Conectando ao servidor OPC UA {url}")
    print("======================================================================")
    async with Client(url=url) as client:

        # Para se ter acesso ao nó, utilize os seguintes comandos:
        cPower_node = client.get_node(cPower)
        cMoveVelocity_node = client.get_node(cMoveVelocity)
        cVelocity_node = client.get_node(cVelocity)
        pVelocity_node = client.get_node(pVelocity)
        # Isso irá criar um objeto do tipo node da variável desejada
        # no qual podemos utilizar metodos de leitura e escrita

        # Para ler uma variável, basta usar o método read_value() no objeto node criado. Exemplo:
        print("Estado do motor: " + str(await cPower_node.read_value()))   # True = ON | False = OFF
        print("Velocidade do eixo: " + str(await cVelocity_node.read_value()))
        print("======================================================================")
        ''' Para a escrita devemos primeiro transformar a variável
        num objeto compativel com o padrão OPC UA, e para isso
        seguimos a seguinte estrutura:
        ua.DataValue(ua.Variant([VALOR], ua.VariantType.[TIPO]))
```

```
        Atente-se bastante ao tipo que for colocado para evitar erros'''

        # Criando variáveis booleanas que podem ser usadas ao longo do programa:
        dv_True = ua.DataValue(ua.Variant(True, ua.VariantType.Boolean))
        dv_False = ua.DataValue(ua.Variant(False, ua.VariantType.Boolean))

        # Variável do tipo float, para alterar a velocidade na parametrização do motor
        dvVelocity = ua.DataValue(ua.Variant(10.0, ua.VariantType.Float))

        # Para escrever na variável, basta utilizar o método write_value,
        # tendo como argumento o valor com tipo correto
        await pVelocity_node.write_value(dvVelocity)    # Mudando valor da velocidade
        await cPower_node.write_value(dv_True)          # Ligando motor

        print("Ligando motor...")
        print("======================================================================")
        # Ao executar comandos de potência, use um temporizador entre eles para evitar erros
        await asyncio.sleep(0.5)                        # Aguarda 0.5 segundo para proxima escrita
        await cMoveVelocity_node.write_value(dv_True)   # Acionando controle de velocidade

        await asyncio.sleep(0.5)
        print("Estado do motor: " + str(await cPower_node.read_value()))   # True = ON | False = OFF
        print("Velocidade do eixo: " + str(await cVelocity_node.read_value()))
        print("======================================================================")

        print("Motor em movimento por 5s (Confira no seu software de automação!)")
        print("======================================================================")
        await asyncio.sleep(5)

        await cMoveVelocity_node.write_value(dv_False)   # Desligando controle de velocidade
        print("Desligando motor...")
        print("======================================================================")
        await asyncio.sleep(0.5)
        await cPower_node.write_value(dv_False)          # Desligando motor

        await asyncio.sleep(0.5)
        print("Estado do motor: " + str(await cPower_node.read_value()))   # True = ON | False = OFF
        print("Velocidade do eixo: " + str(await cVelocity_node.read_value()))
        print("======================================================================")

if __name__ == "__main__":
  asyncio.run(main())
```

5 Criando uma página web utilizando a biblioteca Streamlit

Seguindo o passo a passo do tópico anterior, você conseguirá acesso às variáveis do servidor OPC UA. Para visualizar e utilizar uma página gerada pelo *Streamlit*, primeiro importe a biblioteca adicionando o comando ***import streamlit as st*** (Depois de devidamente instalar a biblioteca em sua máquina), e em seguida rode o seguinte comando no terminal (no *path* onde está localizado o arquivo *.py*):

>streamlit run [nome do arquivo].py

Ao executar esse comando, uma janela no browser irá abrir com a página (provavelmente sem nenhum conteúdo). Agora basta utilizar os recursos da biblioteca para realizar a escrita e leitura das variáveis OPC UA!

6 Conclusão

Este guia de desenvolvimento de um sistema supervisório web para automação industrial foi elaborado com o objetivo de fornecer um caminho claro e prático para os estudantes da área. Para finalizar, aqui vai algumas dicas e possíveis problemas que podem levar um certo tempo para serem resolvidos:

- Para valores que se alteram constantemente, crie uma área *placeholder*;
- Se quiser plotar um gráfico, talvez seja necessário o uso de um banco de dados externo para o armazenamento das variáveis;
- Caso utilize um banco de dados, uma ideia é rodar um script de escrita no banco sendo executado em paralelo (Adquire as variáveis OPC UA e as escreve ciclicamente em um banco com a biblioteca *mysql.connector*) com um script da plataforma web. Dessa forma as variáveis exibidas na plataforma podem ser capturadas pelo banco;
- Para permitir a conexão entre um banco de dados *MySQL* (único testado) e o *streamlit*, é necessário criar/alterar um arquivo denominado *secrets.toml*, localizado em ***C:\Users\[Usuário]\.streamlit***
- Utilize a biblioteca *streamlit_authenticator* para elaborar um sistema de autenticação (Login);
- Streamlit é uma biblioteca versátil, mas é possível utilizar outras bibliotecas para o mesmo fim, assim como é possível integrar outras bibliotecas na solução final, abrindo um mundo de possibilidades.
- Link para a documentação da biblioteca: <https://docs.streamlit.io/>