Complemento OPC-UA: Desenvolvimento e Implementação do protocolo com um módulo OPC-UA externo e Tia Portal

Documento escrito por:

Daniel Jesus Camarneiro Nº84753, daniel.camarneiro@ua.pt

Gonçalo Almeida Vieira de Matos Nº88937, goncalomatos@ua.pt

Índice

Índice de Figuras

Introdução

Este documento representa um complemento ao desenvolvido pelo professor José Paulo Santos relativo à comunicação OPC-UA, onde é apresenta as características deste protocolo, assim como alguns exemplos para desenvolver um cliente OPC-UA em Visual Basic e a respetiva integração do Facon Srv como servidor OPC-UA entre o nosso cliente e o PLC. Assim, nesta fase iremos abordar um outro equipamento onde será executado o nosso servidor OPC-UA (IBH Link UA) [Fig. 1] [1], neste caso um módulo externo desenvolvido especificamente para esta funcionalidade, e como configurar o módulo OPC-UA para aceder a um PLC com o Tia Portal. Também serão realizados três exemplos de clientes OPC-UA em abordagens diferentes: uma em Node-Red (javascript) [2], outra em Python [3] e uma em Node.js (javascript) [4]. Antes de seguir com o desenvolvimento dos clientes OPC-UA anteriormente descritos, será demonstrada a comunicação com o módulo através de um outro cliente OPC-UA, o UaExpert [5].

Fig. 1: Imagem do módulo OPC Server utilizado

A Fig. 2 representa a arquitetura utilizada para a elaboração do vídeo exemplo em anexo. No computador 1 é possível observar a utilização de dois programas para comunicar com o módulo OPC Server: o UaExpert como um cliente OPC UA *ready-to-use* para existir uma referência com os outros programas a desenvolver e o cliente OPC UA que iremos posteriormente desenvolver em Node.js e Electron. No computador 2 encontra-se o Cliente OPC UA desenvolvido em Python. Por fim, no IBH Link UA e no PLC encontram-se os programas desenvolvidos no TIA Portal, tanto para a configuração do Servidor OPC-UA como o programa do PLC.

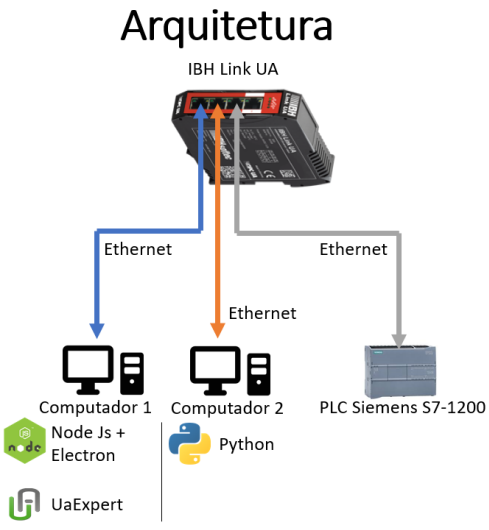


Fig. 2: Arquitetura utilizada no vídeo exemplo

Antes de prosseguir com o início do tutorial, na Fig. 3 e Fig. 4 encontram-se exemplificados os processos para troca de informação relativamente ao pedido de leitura e de escrita de variáveis. Para realizar a leitura de variáveis [Fig. 3], o cliente OPC UA começa por realizar um pedido de leitura, indicando as referências das variáveis a lei (neste caso, com um 'i' e um 'n'). Quando este pedido chega ao IBH Link UA (Servido OPC UA) é convertido de TCP/IP para S7, o protocolo de comunicação utilizado pelo PLC da Siemens. Assim que o PLC recebe uma nova mensagem, processa-a e envia a devida resposta (neste caso contendo informação relativamente às variáveis pedidas) para o IBH Link UA, que posteriormente as reencaminha para o Cliente OPC UA. Comparativamente com as mensagens para escrita, a grande diferença encontra-se na interpretação por parte do PLC, que em vez de realizar o levantamento das variáveis, modifica-as de acordo com os parâmetros da mensagem recebida. No exemplo prático demonstrado no vídeo exemplo utilizam-se pedidos de leitura com 1 segundo de intervalo para monitorizar o estado de algumas saídas do PLC e mensagens de escrita para modificar variáveis de memória que foram programadas para ligar/desligar as saídas monitorizadas.

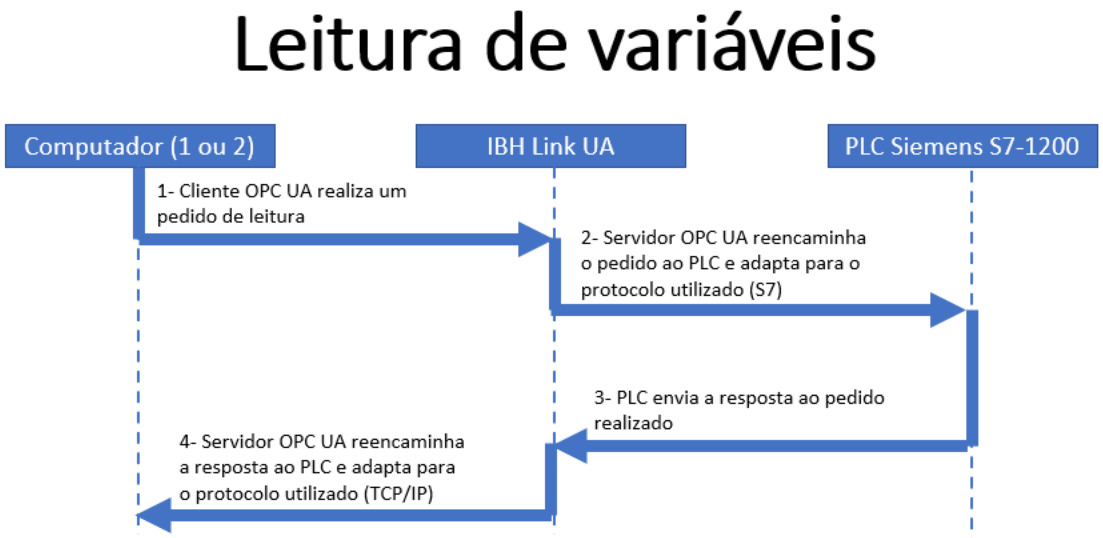


Fig. 3: Diagrama UML relativo à realização de pedidos de leitura de variáveis

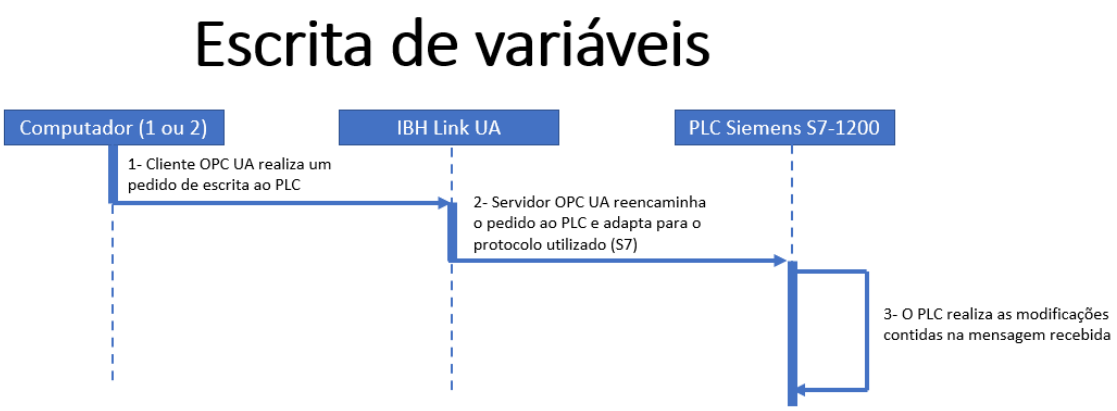


Fig. 4: Diagrama UML relativo à realização de pedidos de escrita de variáveis

Instalação e configuração do módulo OPC IBH Link UA com um PLC

Este capítulo apresenta a metodologia para estabelecer uma comunicação OPC-UA entre um PLC (Siemens S7-1200 [6]) e um *gateway* recorrendo a um módulo OPC-UA server (IBH Link UA). Durante a realização do exemplo utilizaram-se as seguintes versões de *firmware*:

- Siemens S7-1200: V4.1.1;
- IBG Link UA: V5.20.

Se o *PLC Siemens S7-1200* utilizado possuir a versão de *firmware* V4.4 instalada e o *TIA Portal* o *add-on Openness* instalado, é possível utilizar o editor OPC-UA fornecido pela IBH, pois esta versão permite facilmente configurar o servidor OPC-UA do PLC.

Programação do PLC S7-1200

O programa exemplo seguinte apenas pretende demonstrar a utilização de uma comunicação OPC-UA entre um PLC S7-1200 e o seu *gateway*. Assim, o programa apenas permite realizar duas funções:

- Ativar/desativar uma entrada que liga/desliga uma saída;
- Ativar /desativar uma memória que liga/desliga outra saída.

Este exemplo foi realizado em TIA Portal V15.

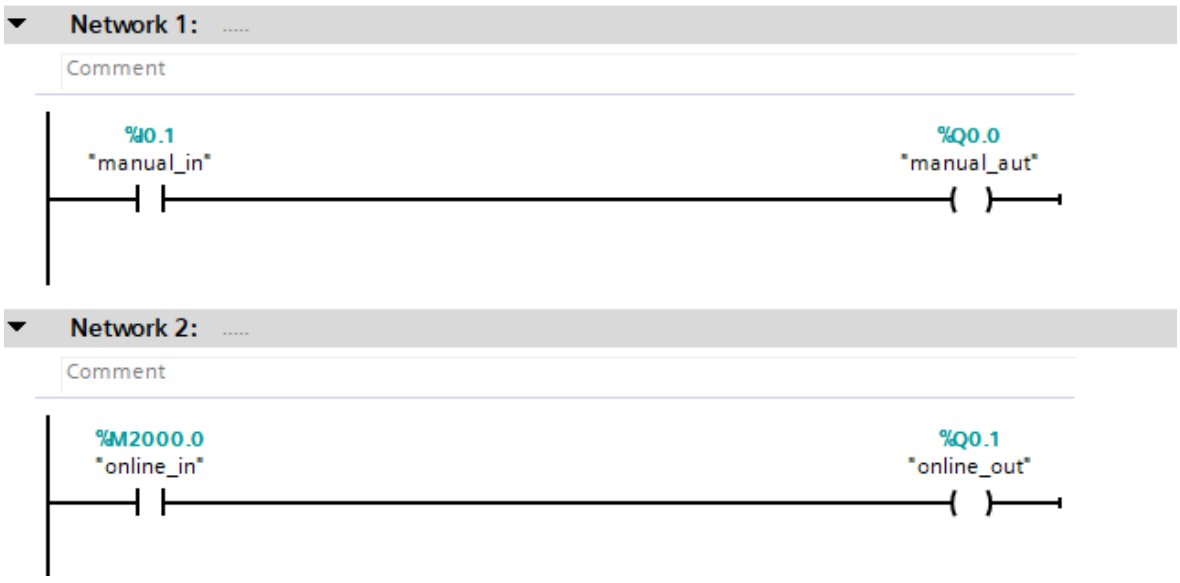


Fig. 5: Programa Ladder utilizado no exemplo

Se a entrada '%I0.1' for ativada, a saída '%Q0.0' é ativada e se a memória '%M2000.0' for ativada, a saída '%Q0.1' é ativada.

Após a criação do programa, segue-se a configuração da porta ethernet do PLC. Um detalhe importante é configurar o PLC, o módulo e o *gateway* na mesma subnet (foi utilizado o IP 192.168.114.xxx para configurar os dispositivos).

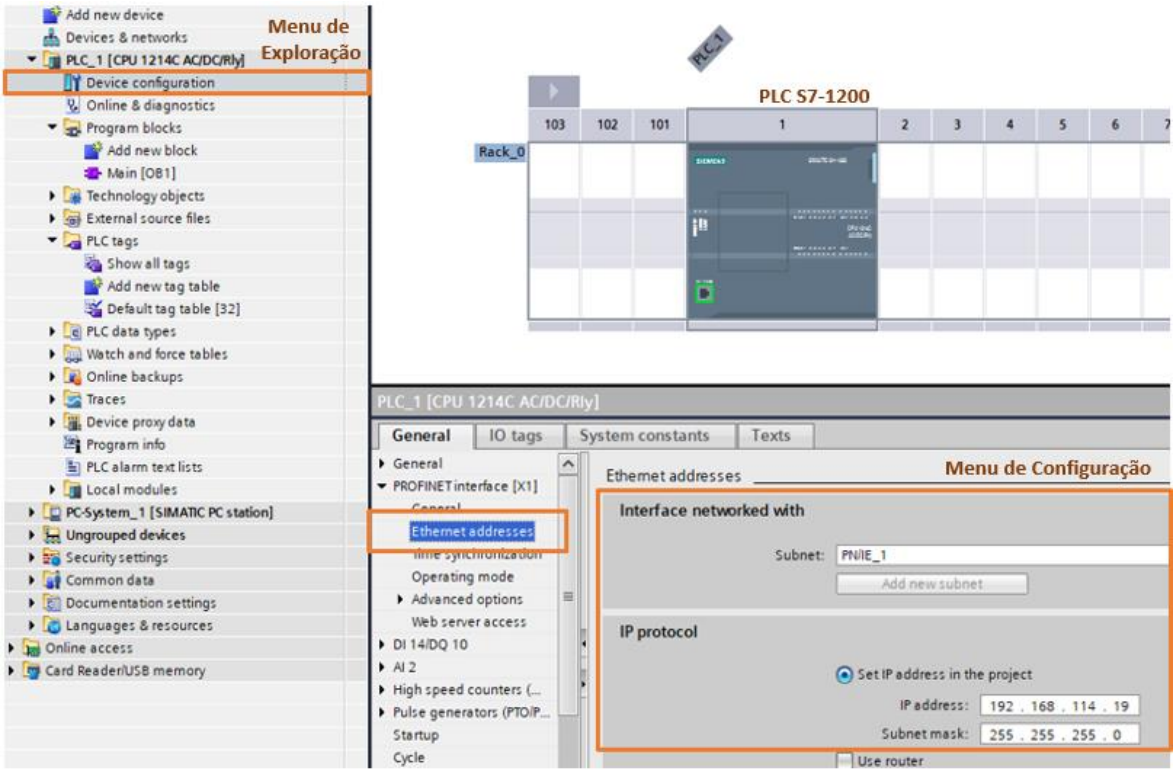


Fig. 6: Configuração do endereço IP do PLC

O IP atribuído ao PLC no exemplo desenvolvido foi 192.168.144.19 na rede local 'PN/IE_1'.

Outra configuração necessária ativar é a permissão por parte de utilizadores externos. Esta permissão permitirá que o módulo OPC interaja com o PLC.

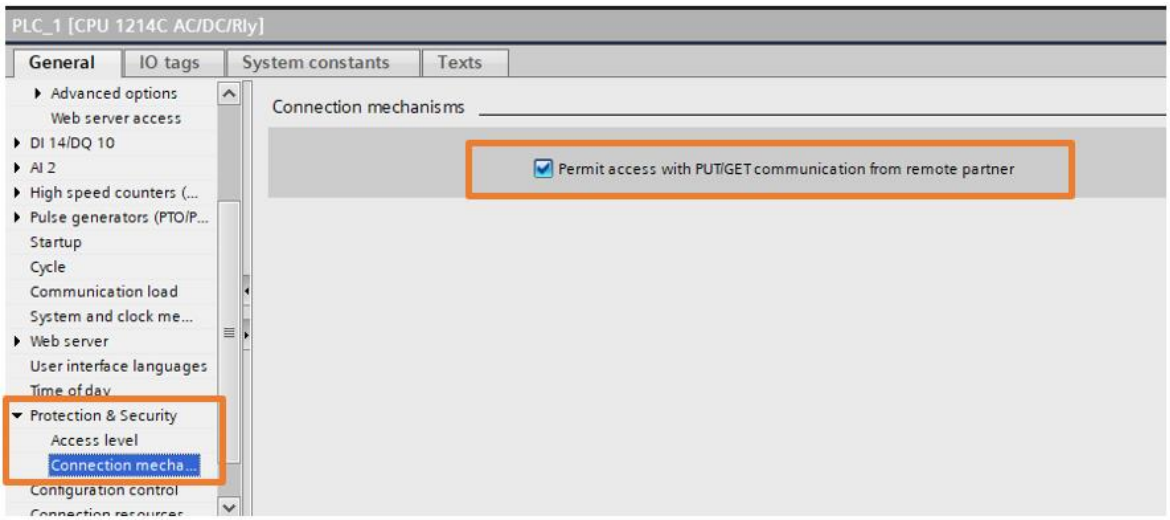


Fig. 7: Habilita a interação de agentes externos

Configuração do IBH Link UA

Neste subcapítulo serão apresentadas as etapas para configurar o módulo IBH Link UA. Se o IP do módulo for desconhecido, ou estiver numa subnet diferente do PLC, utilize um *browser* para editar o IP (é possível utilizar o IP do módulo ou o endereço DHCP 'ibhlinkua_<referencia>' para aceder à página do módulo).

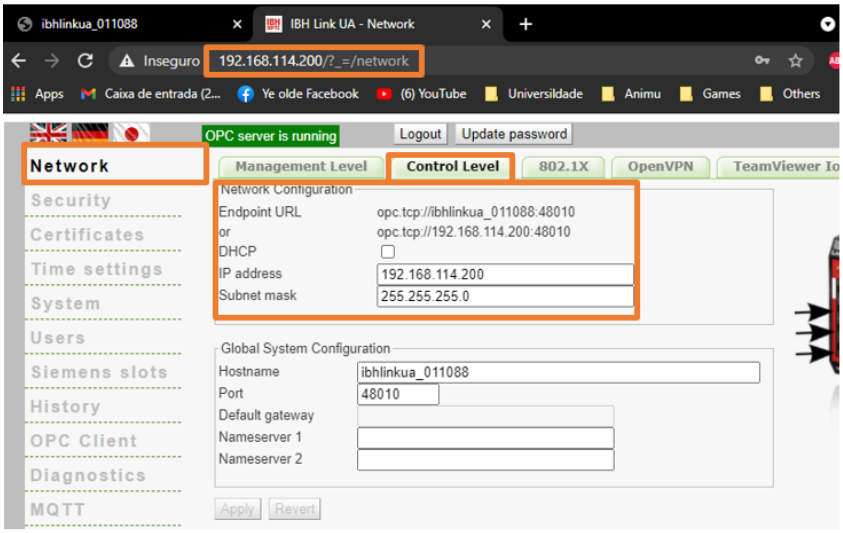


Fig. 8: Configuração do endereço IP do módulo OPC

O IP atribuído ao módulo OPC foi 192.168.114.200.

Configurado o IP do módulo, segue-se a configuração do servidor OPC no TIA Portal. A sequência para a sua configuração é a seguinte:

Adicionar um OPC Server genérico com as seguintes configurações;

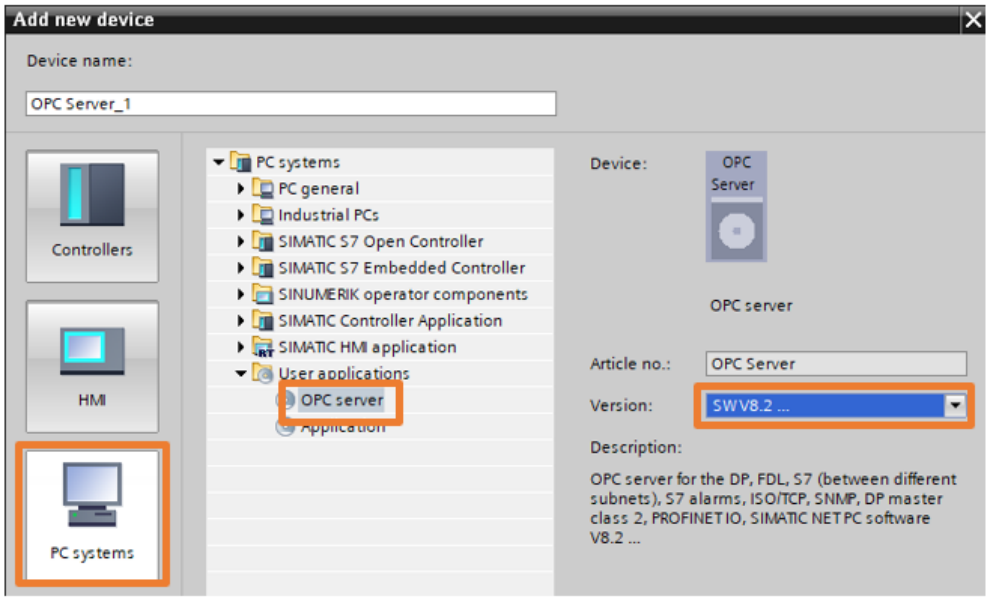


Fig. 9: Etapa 1 na configuração do Servidor OPC

Adicionar um adaptador ethernet;

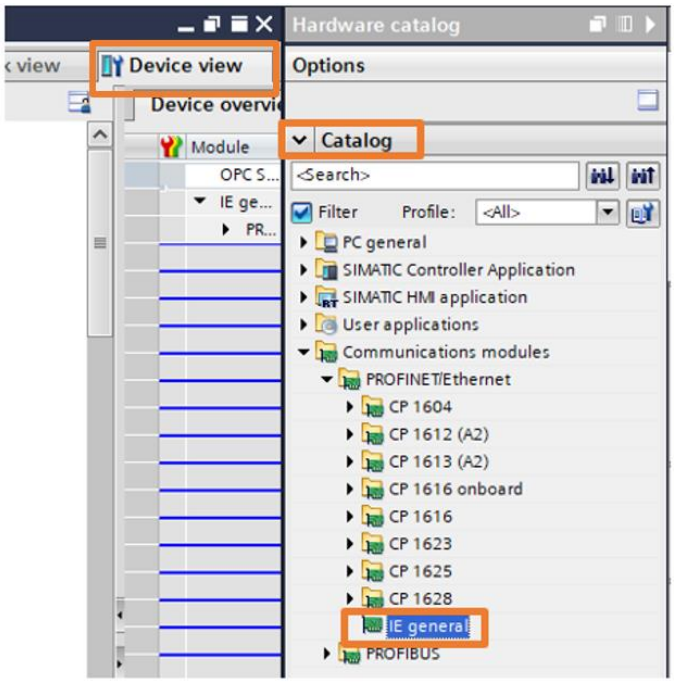


Fig. 10: Etapa 2 na configuração do Servidor OPC

Configurar o IP do OPC Server com o IP do IBH Link UA e ativar o Servidor;

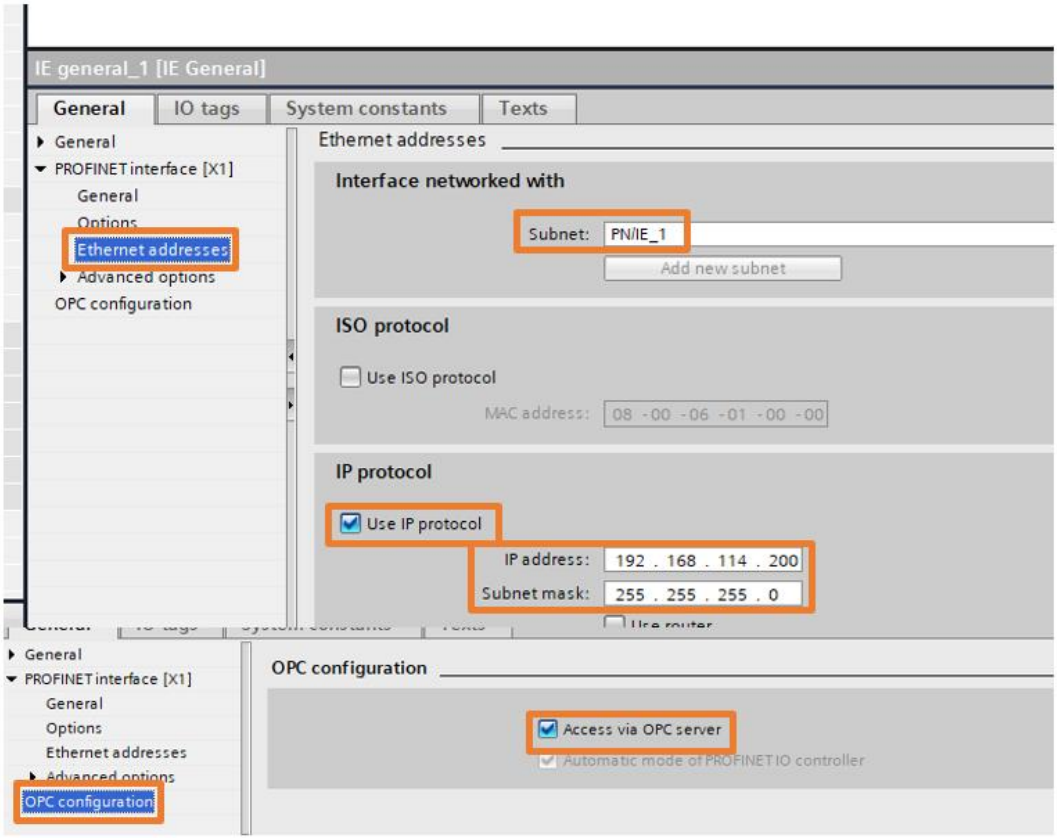


Fig. 11: Etapa 3 na configuração do Servidor OPC

Adicionar uma ligação S7 entre o servidor e o PLC;

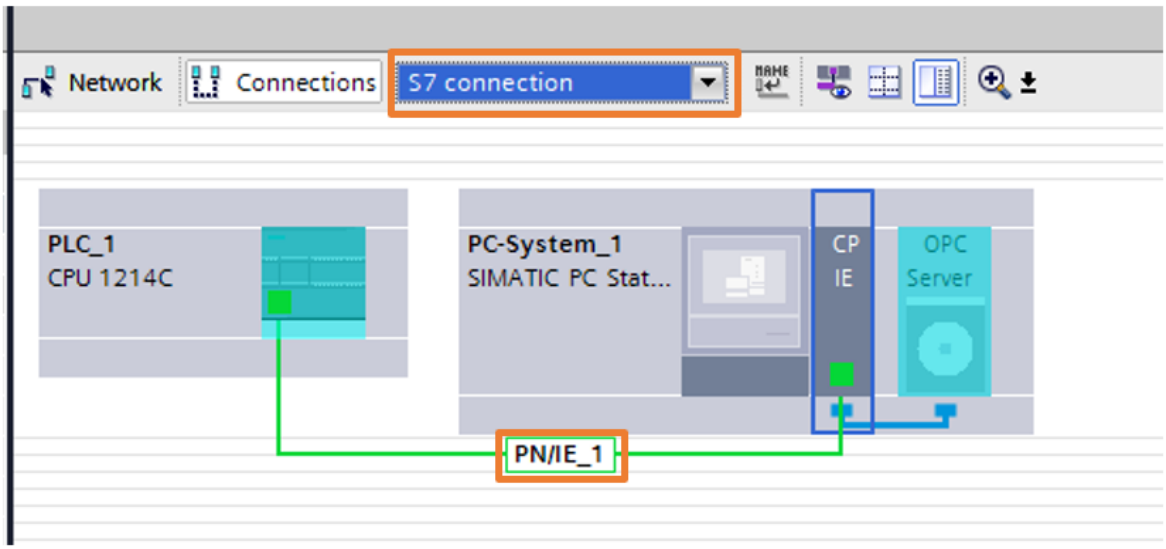
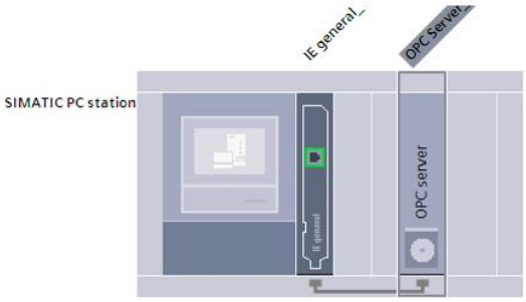


Fig. 12: Etapa 4 na configuração do Servidor OPC

Adicionar as variáveis a ser utilizadas pelo Servidor OPC (se forem todas, selecione 'all');



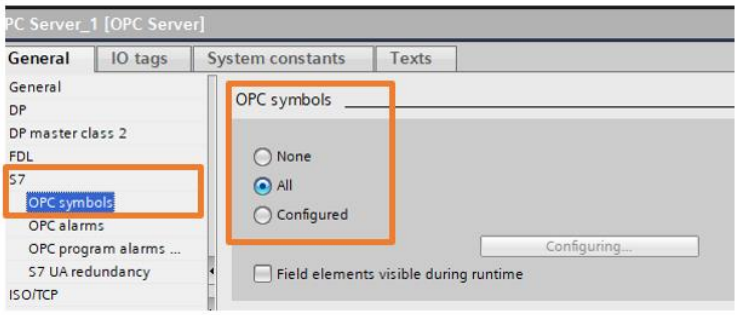


Fig. 13: Etapa 5 na configuração do Servidor OPC

Após realizar a configuração do módulo, segue-se a compilação e a instalação de cada programa para o seu equipamento (programa do PLC para o PLC e o do Servidor OPC para o módulo OPC).

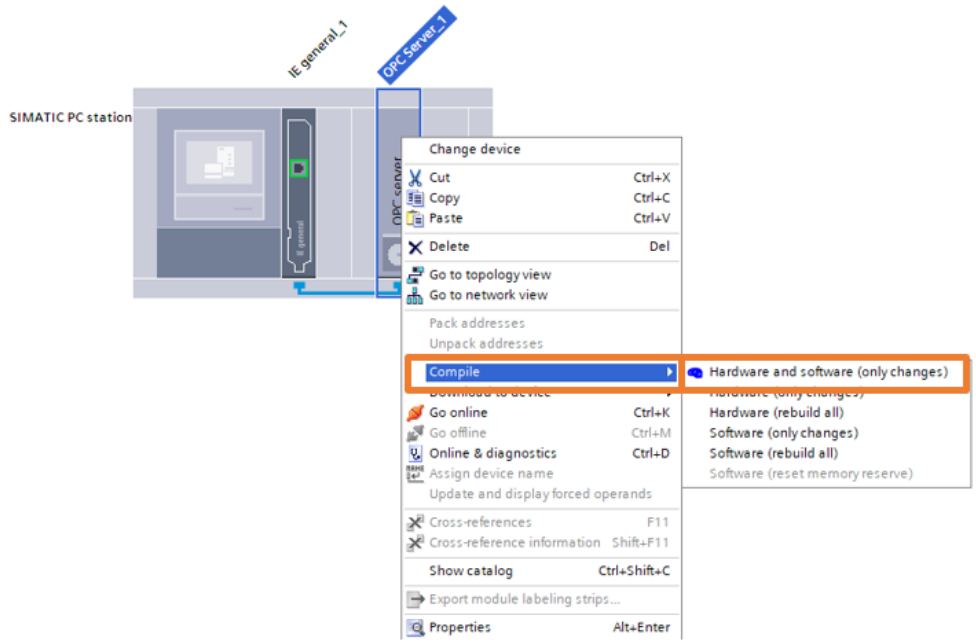


Fig. 14: Compilação do programa desenvolvido

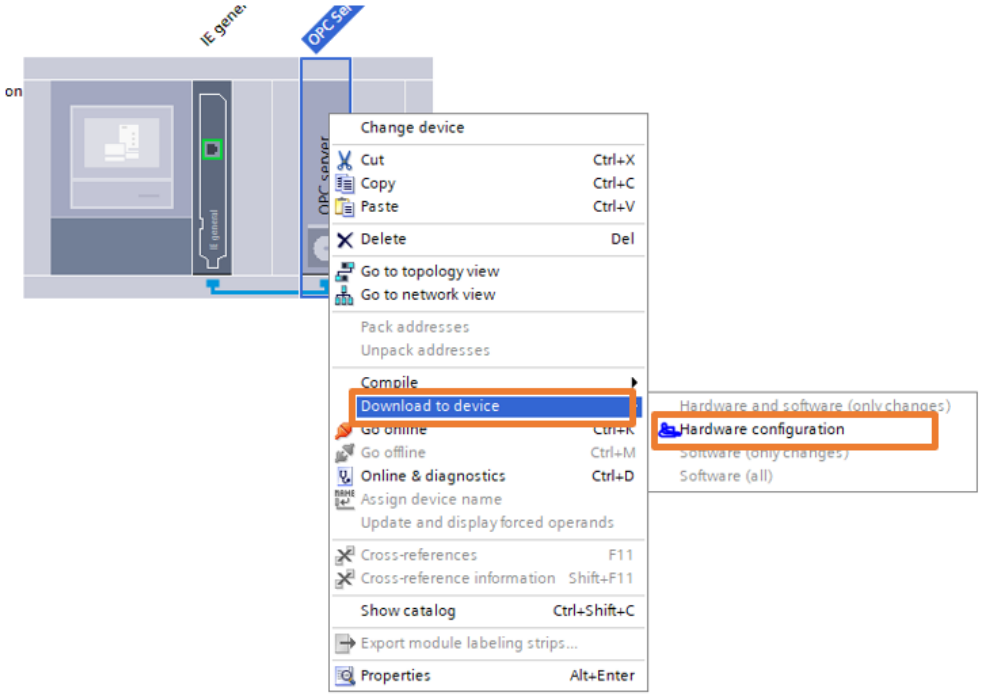


Fig. 15: Envio do programa desenvolvido

Após instalado o programa no módulo é possível visualizar a sua configuração na página de configurações do equipamento.

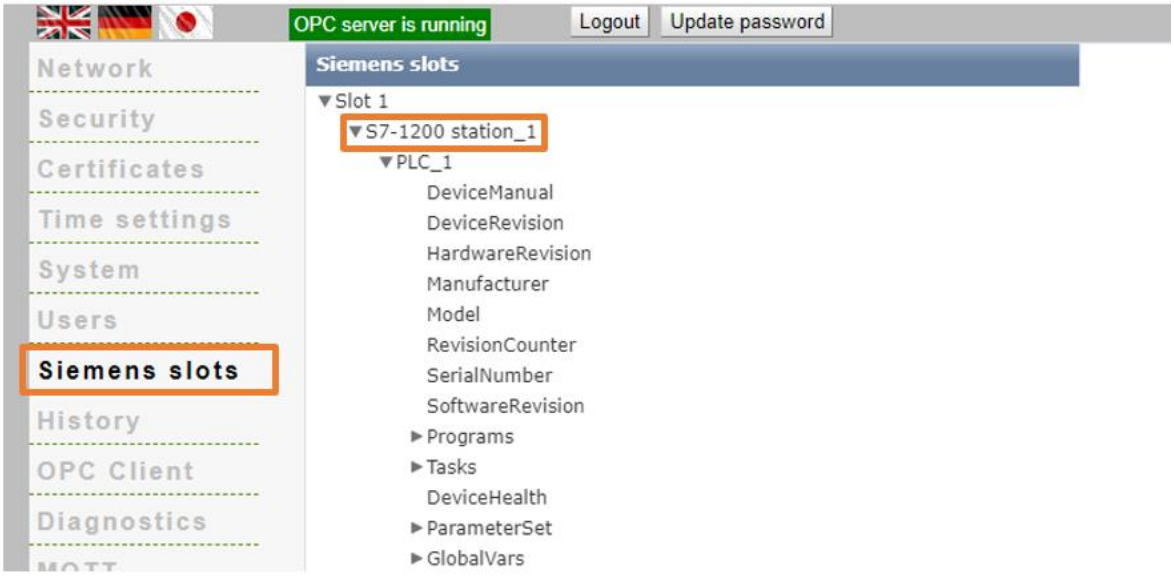


Fig. 16: Visualização do Servidor OPC no módulo OPC

Utilização do cliente OPC-UA "*UaExpert*"

Completa a configuração dos equipamentos, agora segue-se a visualização das variáveis com recurso a um cliente OPC-UA. Para este fim utilizou-se o programa *UaExpert* para estabelecer comunicação com o módulo.

Primeiro adiciona-se o servidor OPC-UA ao *UaExpert*.

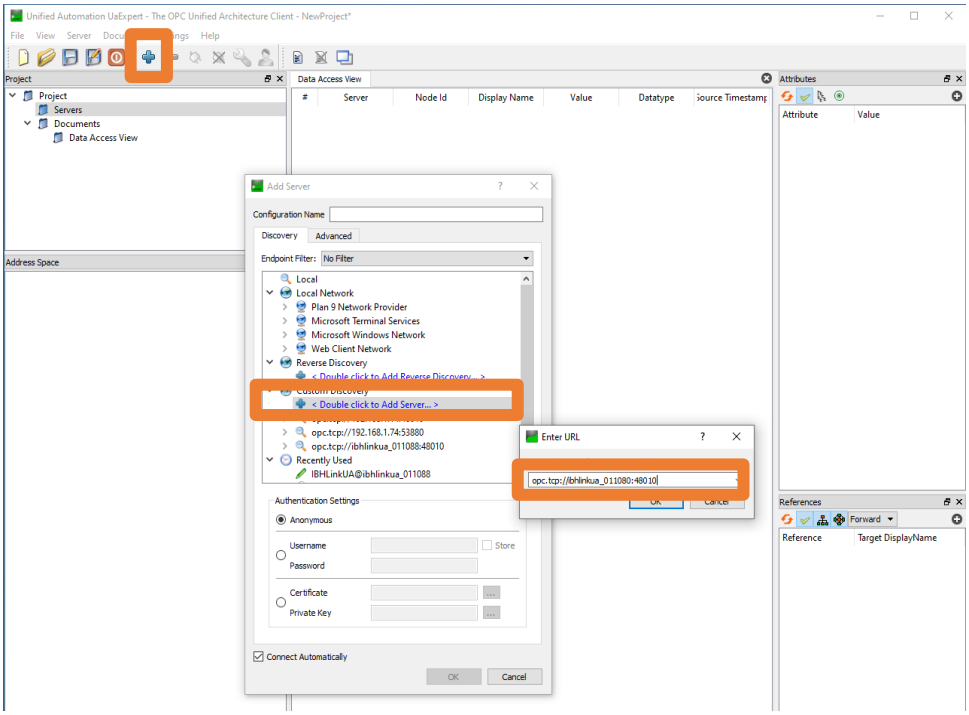


Fig. 17: Criação da ligação OPC com o UaExpert

As configurações do endereço OPC do módulo e os certificados encontram-se na página web deste.

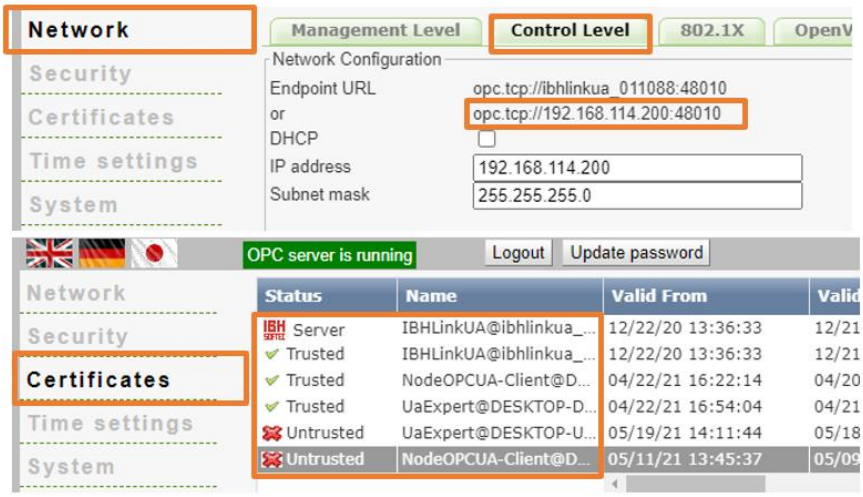


Fig. 18: Configurações e certificados do módulo OPC

Com os parâmetros devidamente configurados, agora é só iniciar a ligação.

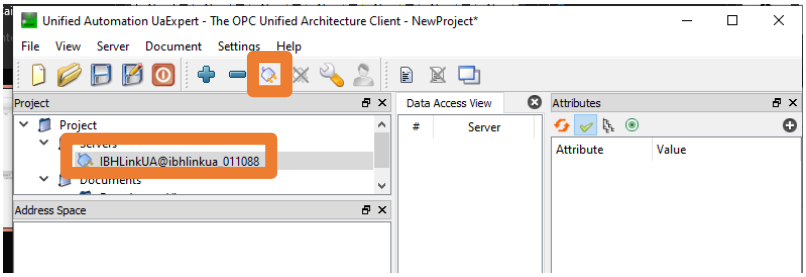


Fig. 19: Iniciar a ligação com o Cliente OPC

Após adicionado o Servidor, é possível explorar as variáveis existentes e visualizar o seu endereço. É importante salientar a importância do *UaExpert* para facilitar a identificação das variáveis a selecionar nos exemplos que se seguem.

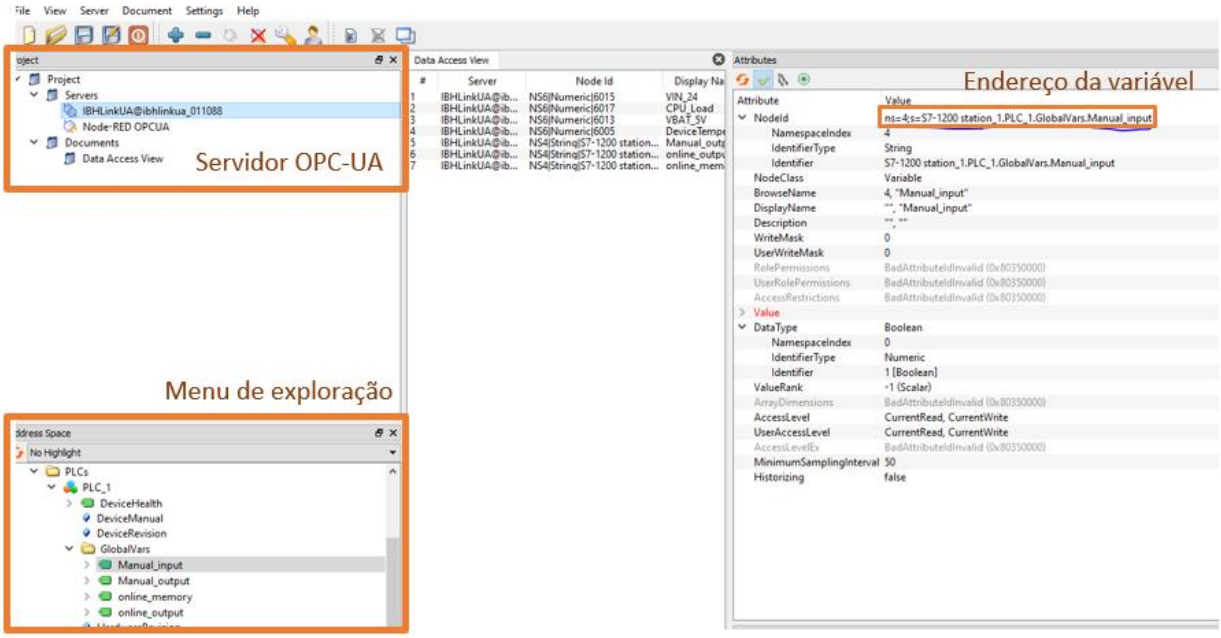


Fig. 20: Visualização das variáveis através de um Cliente OPC

Alcançada esta etapa, é possível visualizar o resultado final de todas as configurações estabelecidas anteriormente, desde os pedidos realizados pelo *UaExpert* até às respostas obtidas do módulo OPC-UA. Deste ponto em diante ser-se-ão realizados os exemplos de clientes OPC-UA indicados anteriormente para desenvolver clientes OPC-UA. É recomendado seguir o documento pela ordem estabelecida, mas é possível apenas desenvolver aquele(s) pretendidos, uma vez que cada exemplo é independente.

Cliente OPC-UA em Node-Red

Neste capítulo será exemplificado o desenvolvimento do cliente OPC-UA em Node-Red. O Node-Red é uma ferramenta de programação gráfica (tipo LabView) com um editor no browser para facilitar o seu desenvolvimento. Este ambiente é executado sobre Node.js, sendo por isso possível desenvolver código personalizado em javascript.

Para a instalação do Node-Red é preciso primeiro instalar o Node.js e depois executar a seguinte linha no terminal (cmd): "*npm install -g –unsafe-perm node-red*" [7]. Concluída a instalação, é só executar "node-red" no terminal e abrir o endereço "localhost:1880" num browser, onde deve aparecer uma janela parecida com a seguinte:

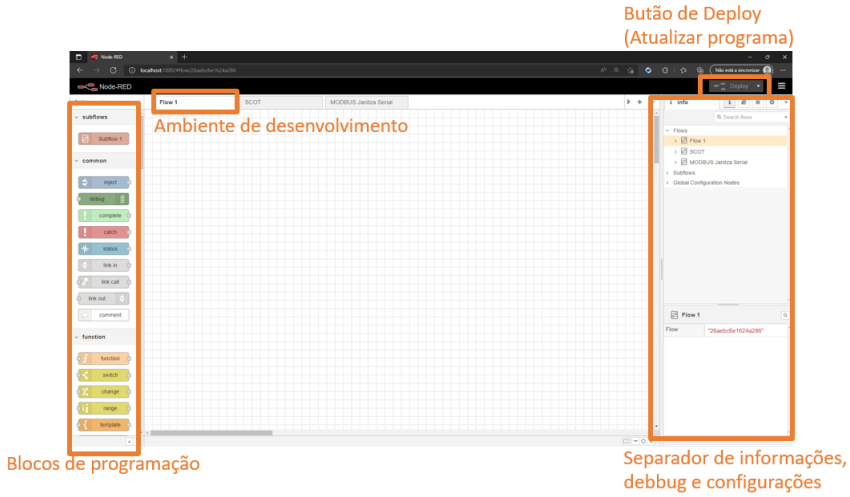


Fig. 21: Ambiente de trabalho do Node-Red

Para desenvolver o programa, primeiro é necessário instalar a biblioteca de OPC-UA, que se encontra no *Manage palette*.

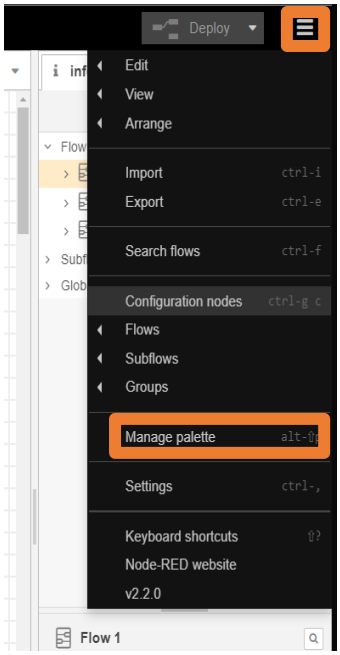


Fig. 22: Localização da 'palette' no Node-Red

Após abrir a nova janela, é só procurar por '*node-red-contrib-opcua*' e instalar.

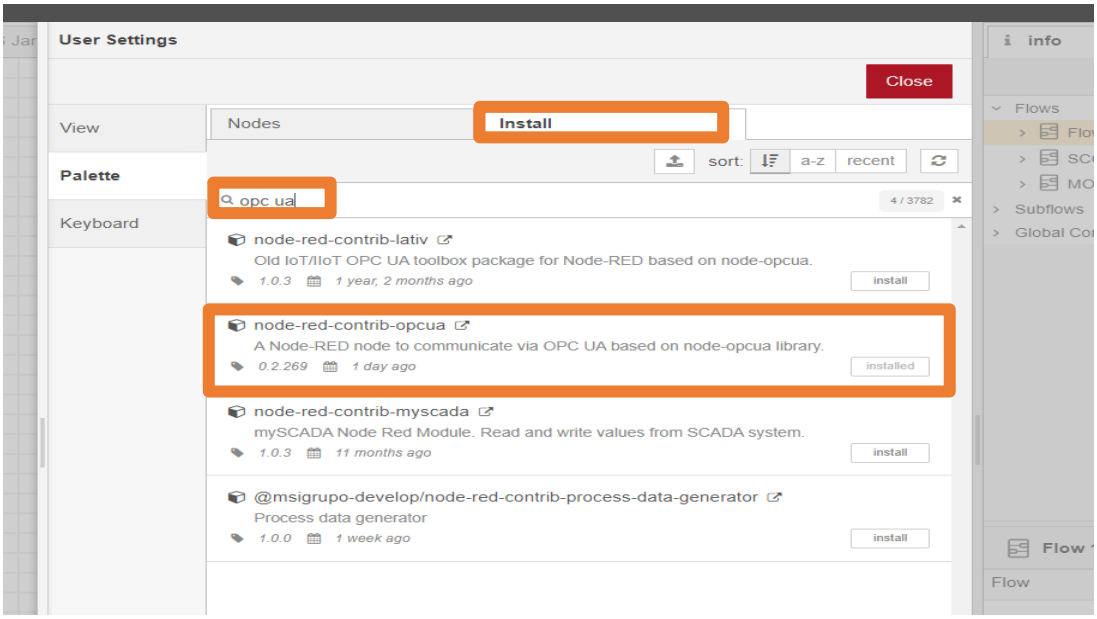


Fig. 23: Seleção do package a instalar

Instalada a biblioteca, agora é só procurar pelos blocos da próxima imagem exemplo no separador dos blocos de programação e seguir com as configurações posteriormente indicadas.

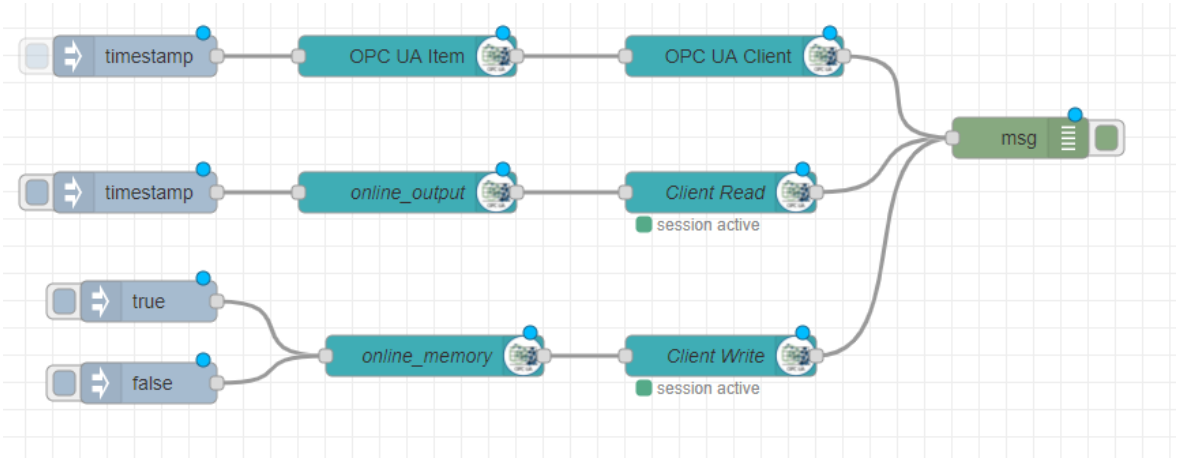


Fig. 24: Nós utilizados para desenvolver o programa

A primeira linha corresponde a uma sequência genérica para identificação dos blocos utilizados, a segunda realiza a leitura da variável *online_output* e a terceira permite escrever *true* ou *false* na variável *online_memory*.

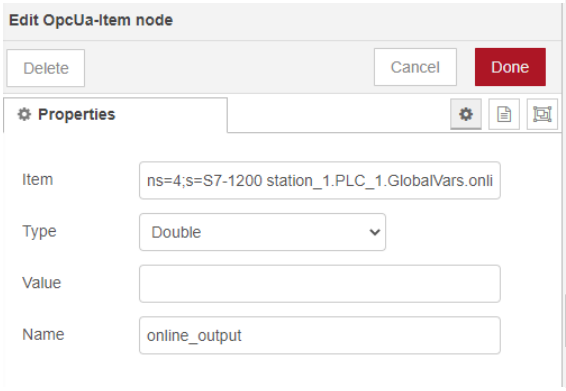


Fig. 28: Configuração da variável de leitura

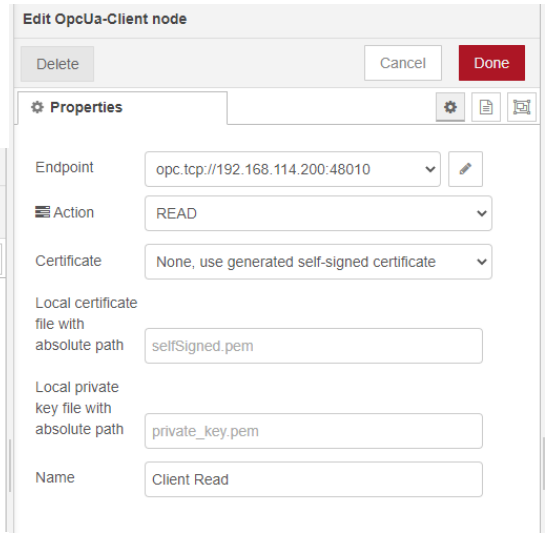


Fig. 26: Configuração do nó destinado à leitura

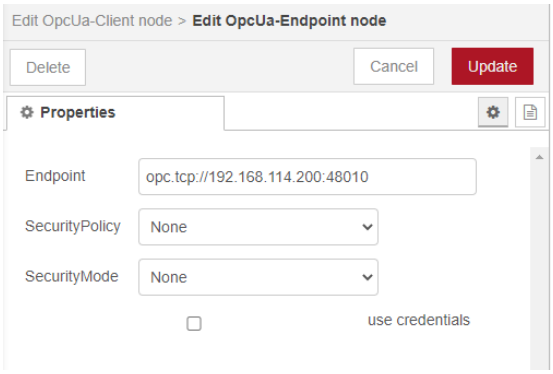


Fig. 27: Configuração do Servidor OPC

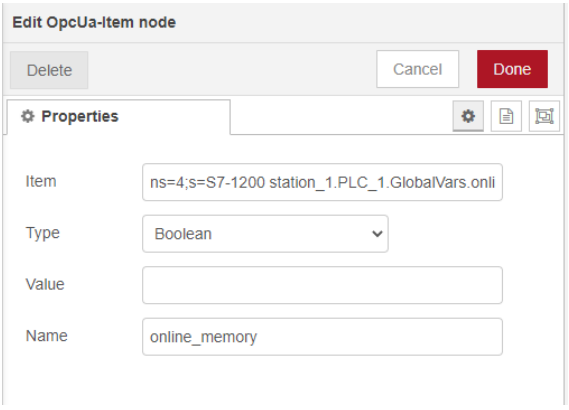


Fig. 25: Configuração da variável de escrita

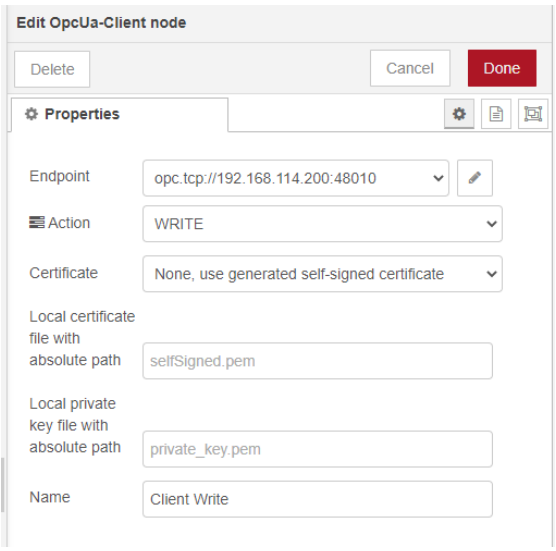


Fig. 29: Configuração do nó destinado à escrita

Concluída a elaboração do cliente OPC-UA em Node-Red, agora é possível observar que todas as interações com o PLC realizadas neste programa são observáveis no *UaExpert*.

Cliente OPC-UA em Python

Neste capítulo será demonstrado como criar um cliente OPC-UA em Python. Serão apresentados dois scripts para leitura de variáveis do servidor OPC-UA e um script para publicação/escrita de variáveis para o servidor. Para melhor compreensão e implementação destes scripts, aconselha-se um conhecimento básico sobre Python, assim como a instalação do mesmo.

Para efeitos de simplicidade, não foram utilizadas funções assíncronas no desenvolvimento dos scripts. No entanto, sugere-se a utilização de funções assíncronas (biblioteca *asyncio*) para conseguir scripts com melhor desempenho que os apresentados e com a mesma funcionalidade.

Leitura de Variável

Neste script é feita a leitura cíclica da temperatura interna medida pelo servidor IBH-Link UA. Com auxílio do UaExpert identificou-se o nó "ns=6;i=6005" como o nó que guarda a medida da temperatura.

Tendo isto, em conta, deve iniciar-se o script com a importação do pacote Python necessário para a comunicação com o servidor OPC-UA. De seguida, para criar o objeto cliente OPC-UA, deve identificar-se o URL do servidor. Depois disto, para ler de forma cíclica a temperatura, cria-se um ciclo *While* no qual o cliente faz um pedido de conexão ao servidor, identifica o nó a ler e recebe do servidor o valor do mesmo, finalizando a conexão antes de reiniciar o ciclo.

```
#!/usr/bin python

from opcua import Client # import necessary package


#
============================================================================================
======

# ----------------------------------   MAIN FUNCTION  ----------------------------------------
--------

#
============================================================================================
======

def main():

    print("\n\nStarting\n")

    # ---------------------- OPC UA CLIENT INITIALIZATION ----------------------------

    url = "opc.tcp://ibhlinkua_011088:48010"  # IP Address of IBH Link UA (endpoint url)
    client = Client(url)  # create client

    node_ID = "ns=6;i=6005"

    # ------------------- Loop to continuously read nodes --------------------------
    while True:

        client.connect()  # connect opc ua client to opc ua server

        temp_node = client.get_node(node_ID)  # define node to read

        temp_value = temp_node.get_value()  # read node
```

```
        print("\nCurrent OPC UA server temperature is " + str(temp_value))

        # client.close_session()
        client.disconnect()

if __name__ == '__main__':
    main()


# useful links
# https://github.com/node-opcua/node-
opcua/blob/master/documentation/creating_a_client_typescript.md
# https://python-opcua.readthedocs.io/en/latest/client.html
```

Leitura de múltiplas variáveis

O script seguinte é bastante semelhante ao apresentado na secção anterior, com a diferença de que foi adaptado para ler múltiplas variáveis de uma vez e organizar os valores em formato json, o qual é bastante utilizado para envio de mensagens em ambiente industrial.

Este script é inicializado exatamente da mesma forma que o anterior. No entanto, as diferenças surgem primeiramente antes do ciclo de leitura. Antes do ciclo, listam-se os nomes desejados para os campos a ler e identificam-se os nós correspondentes.

Entrando no ciclo, a cada iteração é inicializado um dicionário vazio ([8]) no qual serão guardados os valores lidos.

De seguida, inicia-se um ciclo *for* onde, para cada elemento de *names,* é lido o correspondente elemento de *ns* e *i* para identificar o nó, é feita a leitura do valor e é adicionado o valor lido ao dicionário no formato *“name” : “valor lido”.* Depois de percorrer todos os nós,

transforma-se o dicionário obtido para o formato json e o ciclo chega ao fim.

```
#!/usr/bin python
import json
from opcua import Client

# ================================================================================================
# ----------------------------------   MAIN FUNCTION  ------------------------------------------
# ================================================================================================

def main():
    print("\n\nStarting\n")
    # ---------------------- OPC UA CLIENT INITIALIZATION ----------------------------

    url = "opc.tcp://ibhlinkua_011088:48010"  # IP Address of IBH Link UA (endpoint url)
    client = Client(url)

    # print("Client Connected to IBH Link UA ")

    # --------------------- OPC UA SERVER Nodes to read --------------------------------
```

```
    names = ['temp', 'cpu_load', 'memory_load']
    ns = [6, 6, 6]
    i = [6005, 6017, 6019]

    # ------------------- Loop to continuously read nodes ---------------------------
    while True:
        msg_dictionary = {}
        client.connect()  # connect opc ua client to opc ua server

        for k in range(len(ns)):  # 'for' cycle to continuously read and process each node

            opcua_node = client.get_node("ns=" + str(ns[k]) + ";i=" + str(i[k]))  # define node to read
            opcua_data = opcua_node.get_value()  # read node

            # add entry to dictionary with format {"name" : "current_value"}
            # example: {"temp":"40.0"}
            msg_dictionary[names[k]] = '"' + str(opcua_data) + '"'

        msg_json = json.dumps(msg_dictionary)  # convert dictionary to json
        print(msg_json)

        # client.close_session()
        client.disconnect()

if __name__ == '__main__':
    main()


# useful links
# https://github.com/node-opcua/node-opcua/blob/master/documentation/creating_a_client_typescript.md [9]
```

Escrita de Variável

Para alterar o valor de variáveis do servidor OPC-UA, vê-se no seguinte script python que, da mesma forma que é feito para ler variáveis, deve começar-se por criar o cliente OPC-UA e iniciar a conexão.

Neste exemplo, pretendeu-se alterar o valor de variáveis do servidor OPC UA (IBH Link UA). Para isso, primeiro faz-se um pedido de input do NodeID da variável a interagir (dica: usar UaExpert para consultar). De seguida, fez-se um ciclo para selecionar o valor a enviar para o servidor (*true* ou *false*) através do terminal.

No entanto, enviar apenas o valor *true* ou *false* não é suficiente para o servidor em questão, pelo que escrever apenas *opcua_node.set_value(true)* resulta num erro ([10]). Isto acontece porque para enviar mensagens para o servidor devem ser enviadas várias informações, ou seja, não serve apenas enviar o valor booleano. De facto, devem ser enviadas informações adicionais tais como Datatype, Timestamp, Statuscode, entre outras. Assim sendo, em vez de enviar apenas o valor *true* ou *false*, converte-se este valor para um DataValue para que as

restantes informações sejam automaticamente guardadas e enviadas (a ter em atenção que deve ser instalada a subpackage *ua,* em adição à subpackage *Client*).

```
#!/usr/bin python

from opcua import Client, ua

#============================================================================================
# ---------------------------------   MAIN FUNCTION  ---------------------------------------
#============================================================================================

def main():

    print("\n\nStarting\n")

    # ---------------------- OPC UA CLIENT INITIALIZATION ----------------------------

    url = "opc.tcp://ibhlinkua_011088:48010"  # IP Address of IBH Link UA (endpoint url)

    client = Client(url)  #

    client.connect()  # connect opc ua client to opc ua server
    print("Client Connected to IBH Link UA \n\n")

    # --------------------- Choose state to write --------------------------------

    node = input("Node id? [no quotation marks (PT: sem aspas)]\n")
    print("Selected node: " + node + "\n")

    # --------------------- Choose state to write --------------------------------

    state_selected = False # True if a valid state is selected
    state = False # True or false after user chooses (default: false)
    press = input("Set True or False? [Type t or f]\n\n")

    # Loop to choose valid state
    while not state_selected:

        if press == 't' or press == 'T':
            state = True
            state_selected = True
        elif press == 'f' or press == 'F':
            state = False
            state_selected = True
        else:
            state_selected = False
```

```
            press = input('Invalid! Type t or f.\n')

    # --------------------- OPC UA SERVER Nodes write --------------------------------

    opcua_node = client.get_node(node)  # define node to read

    print("\nPublishing " + str(state) + " to node: " + node)

    # Some opc ua servers (such as IBH Link UA) don't support normal python datatypes
    # 'state' is a normal python data type (boolean in this case)
    # For that reason "opcua_node.set_value(state)" does not work
    # There is a need to create a DataValue, so opcua_node_dv was created
    # Check link at the end
    opcua_node_dv = ua.DataValue(ua.Variant(state, ua.VariantType.Boolean))

    opcua_node.set_value(opcua_node_dv)

    # client.close_session()
    client.disconnect()


if __name__ == '__main__':
    main()


# useful links
# https://github.com/FreeOpcUa/python-opcua/issues/997
```

Como nota final, resta esclarecer que estes clientes OPC-UA são capazes de ler e alterar os valores do PLC Siemens. Para tal, basta identificar os nós (NodeID) do servidor OPC UA associados às variáveis do PLC, o que pode ser feito através do UaExpert, sendo que o servidor IBH Link UA intermedeia a interação se estiver devidamente conectado com o PLC.

Cliente OPC-UA em Node.js e Electron

Neste capítulo será demonstrado como realizar um cliente OPC-UA em Node.js e com uma interface em Electron [11]. O editor utilizado para a elaboração deste projeto foi o *Visual Studio Code* [12], mas é de liberdade individual utilizar aquele com maior familiarização.

O Node.js [4], que já foi indicado anteriormente, é uma ferramenta baseada na V8 do interpretador de javascript do Google que permite desenvolver aplicações em javascript fora de um browser.

O Electron é uma ferramenta para desenvolver aplicações de desktop em Node.js com uma metodologia semelhante a desenvolver um website, apresentando ferramentas que facilitam a troca de informação entre o programa e a interface gráfica.

Para começar este projeto, primeiro temos de iniciar um package:

1) Abrir o terminal (do Windows ou do editor) numa pasta vazia e escrever "npm init -y" para iniciar o package do programa;
2) Instalar as duas dependências do programa “electron” e “node-opcua” com "npm install --save node-opcua" & "npm install --save-dev electron".
3) Abrir o ficheiro “package.json” a adicionar a seguinte informação

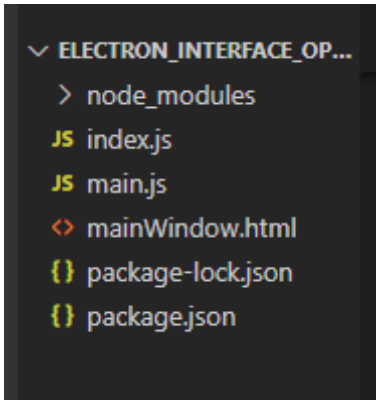
```
“main”:”main.js”,
"scripts": {
    "start": "electron ."
  },
```

No fim o ficheiro “package.json” deve apresentar uma estrutura semelhante à seguinte:

```
{
  "name": "electron_interface_opc_ua",
  "version": "1.0.0",
  "description": "Electron interface to interact with OPC UA",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "author": "Daniel Camarneiro",
  "license": "",
  "devDependencies": {
    "electron": "^17.1.2"
  },
  "dependencies": {
    "node-opcua": "^2.65.1"
  }
}
```

Agora vamos criar três ficheiros:

- “main.js”, que representam o corpo da aplicação em Eletron;
- “index.js”, o renderer onde se encontra o código javascript para interligar o corpo da aplicação e a interface gráfica (este ficheiro não é necessário, mas facilita a organização e o desenvolvimento do programa);
- “mainWindow.html”, que constitui o código HTML para criar a página na interface gráfica.



Vamos primeiro começar por desenvolver o código para o corpo do programa (“main.js”) que estabelecerá a comunicação OPC com o servidor.

```
// Import dos requisitos para o funcionamento do programa
const path = require('path');
const {app, BrowserWindow, ipcMain} = require('electron');
const {
    OPCUAClient,
    MessageSecurityMode, SecurityPolicy,
    AttributeIds,
    DataType
 } = require('node-opcua');

//-------variáveis que vão ser utilizadas------------------
let inter
let var1
let Ent_1

// Criação do OPC cliente (documentação)
const connectionStrategy = {
    initialDelay: 1000,
    maxRetry: 1
}
const options = {
    applicationName: "MyClient",
    connectionStrategy: connectionStrategy,
    securityMode: MessageSecurityMode.None,
    securityPolicy: SecurityPolicy.None,
    endpointMustExist: false,
};
const client = OPCUAClient.create(options);
```

```
const endpointUrl = "opc.tcp://ibhlinkua_011088:48010"; // endereço do
OPC Server

// Loop para atualizar as variáveis num intervalo X (Leitura)
let opc_c= setInterval(async() =>{
    try{
        // Conecta com o servidor
        await client.connect(endpointUrl);
        // Cria uma sessão
        const session = await client.createSession();
        // Executa os pedidos (Leitura)
        var1 = await session.read({nodeId:"ns=6;i=6005",
attributeId:AttributeIds.Value});
        try{var1=var1.value.value}catch{};
        Ent_1 = await
session.read({nodeId:"ns=7;s=.Publish.teste.struct1.u1",
attributeId:AttributeIds.Value});
        try{Ent_1 =Ent_1.value.value}catch{};
        // Termina a sessão
        await session.close();
        // Desconecta com o servidor
        await client.disconnect();
    }
    catch(err){
        console.log(err)
        await client.disconnect();}
},1000) //(1s)

//----Criação da janela para interface gráfica
const createWindow = () => {
    let mainWindow = new BrowserWindow({
        webPreferences: {
            nodeIntegration: true,
            contextIsolation: false,
        }
    });
    mainWindow.loadFile(path.join(__dirname, "./mainWindow.html")); //
Nome do ficheiro HTML com a informação da interface

    //-----------Visualização das variáveis OPC UA-------------
    inter = setInterval(() =>{
        mainWindow.webContents.send('Ent_1', Ent_1);
        mainWindow.webContents.send('Var1',var1);
    },20) // refresh da página em 20 ms
}

// Inicia a aplicação quando estiver pronto
app.on('ready', createWindow);
```

```
app.on('window-all-closed', () =>{
    if(process.platform!=='darwin'){
        app.quit();
        clearInterval(inter);
        clearInterval(opc_c);
    }
})
app.on('activate', () => {
    if(BrowserWindow.getAllWindows().length ===0){
        createWindow()
    }
})

//-----------Adicionar interações (Escrita)-------------
//Entidade 1 ON
// Caso receba o evento 'something' a função é executada
ipcMain.on('ent_1_t', async(event) =>{
    // Envia a ação para o Servidor OPC
    try{
        await client.connect(endpointUrl);
        const session = await client.createSession();
        //nodeId = endereço da variável no UaExpert
        await session.write({nodeId:"ns=7;s=.Publish.teste.struct1.u1",
attributeId:AttributeIds.Value, indexRange:null,
value:{value:{dataType:DataType.Boolean,value: true}}})
        await session.close();
        await client.disconnect();
    }
    catch(err){
        console.log(err)
        await client.disconnect();}
})
//Entidade 1 OFF
ipcMain.on('ent_1_f', async(event) =>{
    try{
        await client.connect(endpointUrl);
        const session = await client.createSession();
        //nodeId = endereço da variável no UaExpert
        await session.write({nodeId:"ns=7;s=.Publish.teste.struct1.u1",
attributeId:AttributeIds.Value, indexRange:null,
value:{value:{dataType:DataType.Boolean,value: false}}})
        await session.close();
        await client.disconnect();
    }
    catch(err){
        console.log(err)
        await client.disconnect();}
})
```

Agora segue-se o código para a definição da interface gráfica (“mainWindow.html”).

```
<!DOCTYPE html>
<html lang="pt">

<head>
    <title>OPC-UA Monitor Interface</title>
</head>

<style>
    /*Cria duas colunas iguais*/
    .column {
        float: left;
        width: 50%;
        }
</style>

<body>
    <form>
         <div class="row">
            <!--Inputs-->
            <div class="column">
                <!--Bloco genérico para adicionar butões switch no
template-->
                <div style="white-space: pre-wrap;">
                    <label>Entity 1</label>
                    <button id="ent_1_t" value="true">Entity 1
ON</button>
                    <button id="ent_1_f" value="false">Entity 1
OFF</button>
                </div>
                <!-----------------------------------------------------
---->
            </div>
            <!--Outputs-->
            <div class="column">
                <div style="white-space: pre-wrap;">
                    <!--Bloco genérico para adicionar Indicadores no
template-->
                    <b>Entity 1:</b>  <span id="Ent_1">-</span>
                    <!-----------------------------------------------
-------->
                    <b>Var1:</b>  <span id="Var1">-</span>
                    <!-----------------------------------------------
-------->
                </div>
            </div>
        </div>
    </form>
```

```
   <!--Mapeamento entre a interface e o main-------------------->
    <script defer src="index.js"></script>
    <!------------------------------------------------------------>
</body>
</html>
```

Por fim poderíamos ter adicionado o render no ficheiro anterior onde diz “<script>”, mas para simplificar o código vamos importámo-lo do ficheiro “index.js”.

```
// Requisitos para o funcionamento do programa
const electron = require('electron');
const ipc = electron.ipcRenderer;
//---------------------------------------

//-----------Render -> Main-------------
//Entidade 1 ON
const ent_1_t = document.getElementById('ent_1_t')
ent_1_t.addEventListener('click', () =>{
    ipc.send('ent_1_t')
})
//Entidade 1 OFF
const ent_1_f = document.getElementById('ent_1_f')
ent_1_f.addEventListener('click', () =>{
    ipc.send('ent_1_f')
})

//-----------Main -> Render-------------
//Entidade 1
ipc.on('Ent_1', (event, message) => {
    document.getElementById('Ent_1').innerHTML = message
});
ipc.on('Var1', (event, message) => {
    document.getElementById('Var1').innerHTML = message
});
```

Completo o programa, agora é só abrir o terminal na pasta do “package” e escrever “npm start” (definido anteriormente no “package.json”) para o Windows executar o programa e abrir a interface gráfica.

Referências

[1] “IBH Link UA - The compact OPC UA server for S5- and S7-PLC - IBHsoftec GmbH.” [Online]. Available: https://www.ibhsoftec.com/IBH-Link-UA-Eng. [Accessed: Apr. 29, 2022]

[2] “Node-RED.” [Online]. Available: https://nodered.org/. [Accessed: Apr. 29, 2022]

[3] “Welcome to Python.org.” [Online]. Available: https://www.python.org/. [Accessed: Apr. 29, 2022]

[4] “Node.js.” [Online]. Available: https://nodejs.org/en/. [Accessed: Apr. 29, 2022]

[5] “UaExpert ‘UA Reference Client’ - Unified Automation.” [Online]. Available: https://www.unified-automation.com/products/development-tools/uaexpert.html. [Accessed: Apr. 29, 2022]

[6] “CLP SIMATIC S7-1200 | Controladores SIMATIC | Siemens Siemens Brasil.” [Online]. Available: https://new.siemens.com/br/pt/produtos/automacao/controladores/s7-1200.html. [Accessed: Apr. 29, 2022]

[7] “Running Node-RED locally : Node-RED.” [Online]. Available: https://nodered.org/docs/getting-started/local. [Accessed: Apr. 29, 2022]

[8] “Python Dictionaries.” [Online]. Available: https://www.w3schools.com/python/python_dictionaries.asp. [Accessed: May 05, 2022]

[9] “node-opcua/creating_a_client_typescript.md at master · node-opcua/node-opcua.” [Online]. Available: https://github.com/node-opcua/node-opcua/blob/master/documentation/creating_a_client_typescript.md. [Accessed: May 05, 2022]

[10] “Can’t Set Value · Issue #997 · FreeOpcUa/python-opcua.” [Online]. Available: https://github.com/FreeOpcUa/python-opcua/issues/997. [Accessed: May 05, 2022]

[11] “Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.” [Online]. Available: https://www.electronjs.org/. [Accessed: Apr. 29, 2022]

[12] “Visual Studio Code - Code Editing. Redefined.” [Online]. Available: https://code.visualstudio.com/. [Accessed: Apr. 29, 2022]