UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

Letícia Maria Resende

Pedro Luís Silva Giacomin

Universidade Federal de Minas Gerais – UFMG

TRABALHO DE SISTEMAS DISTRIBUÍDOS PARA AUTOMAÇÃO

DOCUMENTAÇÃO

Belo Horizonte

01 de Agosto de 2024

1- Introdução

O alto-forno é uma estrutura industrial complexa onde ocorrem diversas reações químicas e físicas, resultando na redução do minério de ferro a ferro fundido. Este projeto visa simular via software esse processo industrial, propondo um método de operação distribuído cuja comunicação aconteça por meio dessa estrutura descentralizada.

2- Arquitetura

A arquitetura implementada pode ser visualizada no esquemático a seguir:

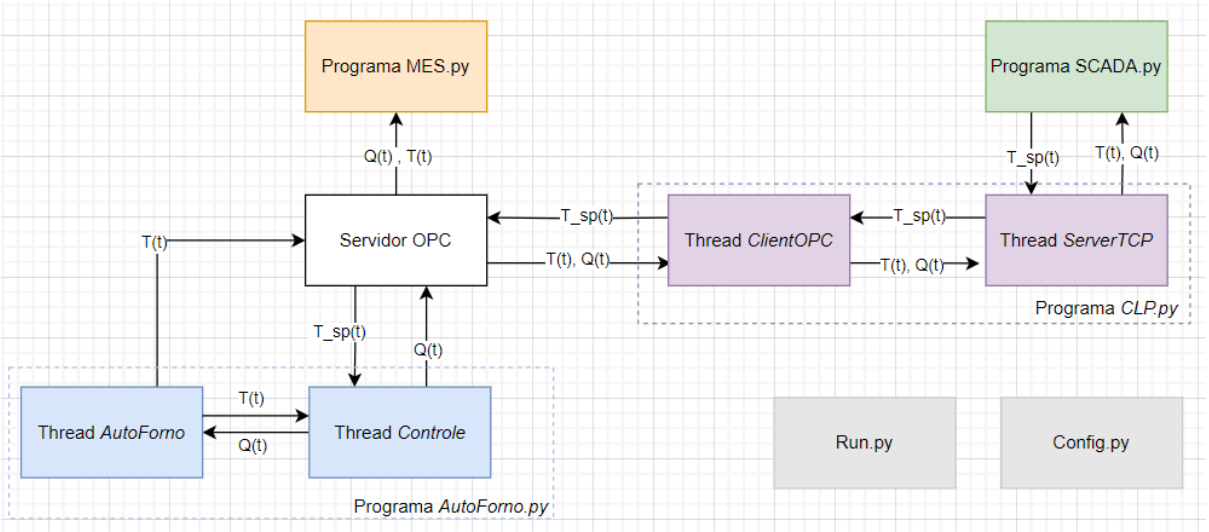

Figura I: Esquemático da arquitetura do projeto

Nesse viés, é perceptível que foram utilizados quatro programas e um servidor OPC para representar a estrutura industrial do alto-forno. Além disso, foram implementados dois programas adicionais para auxiliar o usuário na execução da aplicação que serão detalhados na próxima seção.

2.2. Fluxo de Dados

- **Definição de Valores**: As threads do programa AutoForno.py calculam e atualizam os valores de temperatura (T(t)) e fluxo de calor (Q(t)) no servidor OPC. Além disso, o programa CLP.py é responsável por configurar o valor da temperatura de setpoint(T_sp(t)) nesse servidor.
- **Leitura de Valores pelo MES**: O programa MES.py lê periodicamente os valores armazenados no servidor OPC para monitoramento e análise.
- **Transmissão de Dados pelo CLP**: O programa CLP.py lê os valores do servidor OPC através da Thread ClientOPC e os envia ao servidor TCP via Thread ServerTCP.
- **Interface SCADA**: O programa SCADA.py, como cliente TCP, recebe os dados do servidor TCP, exibindo-os ao operador. Além disso, permite o envio de um novo setpoint de temperatura (T_sp(t)) para o servidor TCP.

- **Atualização de Setpoint**: O setpoint de temperatura (T_sp(t)) é transmitido do programa SCADA para o servidor TCP, de onde é lido pela Thread ClientOPC do programa CLP.py, que então atualiza o servidor OPC. A Thread Controle do programa AutoForno.py lê este novo setpoint e a temperatura atual e ajusta o fluxo de calor (Q(t)) de acordo.

3- Descrição Detalhada do Sistema

3.1. Servidor OPC

O servidor OPC é o núcleo central do sistema, responsável por armazenar os valores de temperatura (T(t)) , fluxo de calor (Q(t)) e temperatura de referência (T_sp(t)) ao longo do tempo. Este servidor permite a comunicação e troca de dados entre os diferentes componentes do sistema, garantindo que todas as partes envolvidas possam acessar e atualizar as informações de processo necessárias.

3.2. Programa AutoForno.py

Este programa contém duas threads principais:

- **Thread autoForno:** Simula a dinâmica do alto-forno, obtendo o fluxo de calor atual e calculando periodicamente a temperatura (T(t)) baseada na equação diferencial do processo. Esses valores são enviados e armazenados no servidor OPC. Essa thread é executada periodicamente a cada 1s.
- **Thread controle:** Responsável por ajustar o fluxo de calor (Q(t)) para manter a temperatura do alto-forno próxima ao setpoint desejado. Ela obtém a temperatura atual e lê o setpoint (T_sp(t)) do servidor OPC. Com esses valores, a thread calcula o novo fluxo de calor necessário, atualizando o servidor OPC com este valor. Essa thread é executada periodicamente a cada 0.5 s.

Além dessas threads, o programa também possui uma função para fazer a integração do fluxo por meio do método Runge-Kutta e uma função para simular o alto-forno, como pode ser visto no código a seguir:

```
def runge_kutta(T, Q, dt):
    k1 = derivada_temperatura(T, Q)
    k2 = derivada_temperatura(T + 0.5 * k1 * dt, Q)
    k3 = derivada_temperatura(T + 0.5 * k2 * dt, Q)
    k4 = derivada_temperatura(T + k3 * dt, Q)
    T_next = T + (k1 + 2*k2 + 2*k3 + k4) * (dt / 6.0)
    return T_next
def simular_alto_forno(T_inicial, tempo_total, dt, fluxo_calor):
    num_pontos = int(tempo_total/dt) + 1
    tempo = np.linspace(0, tempo_total, num_pontos)
    temperatura = np.zeros(num_pontos)
```

```
    # Condição inicial
    temperatura[0] = T_inicial
    # Simulação
    for i in range(1, num_pontos):
        temperatura[i] = runge_kutta(temperatura[i-1], fluxo_calor[i-1],
dt)
    return tempo, temperatura
```

Por fim, é válido enfatizar que esse programa também é um cliente OPC, por esse motivo a thread de controle consegue ler e armazenar valores no servidor.

3.2. Programa MES.py

Esse programa atua como um cliente OPC, lendo os valores de temperatura (T(t)) e fluxo de calor (Q(t)) do servidor OPC. Esses dados são escritos em um arquivo “mes.txt” a cada um minuto pela thread principal. Além disso, também possui uma thread adicional para monitorar o teclado e encerrar o programa quando alguma tecla for pressionada.

3.3. Programa CLP.py

Este programa possui duas threads principais:

- **Thread ClientOPC**: Lê os valores de temperatura (T(t)) e fluxo de calor (Q(t)) do servidor OPC e os transfere para o servidor TCP. Além disso, também envia os valores da temperatura de setpoint recebidos para o servidor OPC. Essa thread é executada a uma taxa que inicialmente foi definida como 1s.
- **Thread ServerTCP**: Funciona como um servidor TCP, recebendo conexões e enviando os dados do processo para o cliente TCP, no caso, o programa SCADA.py. Além disso, também recebe a temperatura de setpoint do SCADA.

3.4. Programa SCADA.py

Atua como um cliente TCP, recebendo os dados do processo do servidor TCP e exibindo essas informações para o operador através de um gráfico atualizado em tempo real. O SCADA também permite que o operador defina o setpoint de temperatura (T_sp(t)), que é enviado de volta ao servidor TCP.

3.5. Programa Config.py

Esse programa tem como objetivo centralizar em apenas um lugar as informações necessárias para fazer a comunicação com o servidor OPC. Desse modo, antes de executar os programas, é necessário inserir a URL do servidor e o nodeId das variáveis.

3.5. Programa Run.py

Esse programa tem como objetivo executar de maneira automática todos os programas. Desse modo, para iniciar a aplicação basta estar na pasta do projeto e executar o seguinte comando:

```
python Run.py
```

Dessa forma, todos os programas serão iniciados e será aberto um terminal para cada.

4 - Execução

Os passos para a execução do programa se encontram no arquivo README.md, na mesma pasta que os códigos do projeto.