Desenvolvimento de um sistema SCADA para automação industrial

Jairo Henrique Wiethan[1], **Reiner Franthesco Perozzo**[1]

[1]Ciência da Computação – Universidade Franciscana (UFN)
Caixa Postal 151 – 97.010-032 – Santa Maria – RS – Brazil

jairo.wiethan@unifra.edu.br, reiner.perozzo@gmail.com

Abstract. *The aim of this work is to develop a SCADA system for use in the automation of industries, along with the making of a PLC program that will interface with electric motors. The system will be written using the programming languages Kotlin for the system's backend, TypeScript for the system's frontend and Structured Text for the PLC's program, along with the frameworks Eclipse Vert.x in the backend and React in the frontend.*

Resumo. *O objetivo do presente trabalho é desenvolver um sistema SCADA para uso na automação de indústrias, juntamente com a criação de um programa de CLP que fará a interface entre o sistema e motores elétricos. Tal sistema será escrito usando as linguagens de programação Kotlin para o backend do sistema, TypeScript para o frontend do sistema e Structured Text para o CLP, além de usar os frameworks Eclipse Vert.x no backend e React no frontend.*

1. Introdução

Com o advento da internet o mundo está cada vez mais conectado, e com essa conectividade veio a capacidade de interligar aparelhos eletrônicos inteligentes, os quais podem coletar dados ou realizar tarefas de acordo com comandos enviados a eles a distância. Com base nesse conceito, a ideia da automação industrial é criar uma rede de aparelhos industriais que possam ser monitorados e controlados remotamente, com o uso de um computador ou telefone celular.

O presente trabalho oferece uma alternativa de baixo custo para a automação de processos industriais que tem como base a operação de motores. Tal sistema pretende ser capaz de permitir o acionamento dos motores a distância e a verificação do estado dos mesmos, de forma segura e cômoda.

1.1. Objetivo Geral

O objetivo deste trabalho é criar um sistema de controle supervisório e aquisição de dados que seja modular e multiplataforma, juntamente com um programa para os CLPs (Controladores Lógicos Programáveis) que o sistema irá controlar.

1.2. Objetivos Específicos

Os objetivos alcançados durante a implementação do projeto são os seguintes:

- Implementação do protocolo de comunicação ModBus, para a comunicação entre os CLPs e o sistema.

- Criação do programa dos CLPs, para que se comuniquem com o sistema.
- Criação da "camada do servidor de dados" do sistema, responsável pelo gerenciamento dos dispositivos conectados e da comunicação e aquisição de dados.
- Criação da "camada do cliente" do sistema, responsável por prover uma interface para que usuários possam operar o sistema.

1.3. Justificativa

O trabalho justifica-se pela necessidade de uma alternativa multiplataforma, modular, de fácil configuração e que tenha uma performance satisfatória mesmo em sistemas embarcados, aos sistemas SCADA (Sistemas de Supervisão e Aquisição de Dados) existentes no mercado. A grande maioria dos sistemas SCADA mais utilizados atualmente necessita de sistemas operacionais Windows, como o WinCC e Elipse E3, o que torna difícil a sua utilização em em sistemas embarcados, que também exigem desempenho superior a soluções como ScadaBR.

Tal sistema será capaz de diminuir os custos de implementação em comparação às soluções existentes, ao usar *hardware* de baixo custo como o Raspberry Pi e não necessitar a compra de licenças de *softwares* proprietários como o Windows. Dessa forma, o sistema poderá se tornar uma boa alternativa para empresas de pequeno porte ou residências que precisem de algum tipo de automação.

1.4. Estrutura do Trabalho

O trabalho é estruturado da seguinte maneira: a Seção 1 apresenta uma introdução ao tema geral do trabalho; a Seção 2 apresenta e define os principais assuntos referentes ao trabalho; a Seção 3 apresenta os trabalhos relacionados ao trabalho em questão; a Seção 4 apresenta os detalhes do desenvolvimento do trabalho; a Seção 5 apresenta a implementação do trabalho; a Seção 6 apresenta a validação do trabalho; e a Seção 7 apresenta a conclusão do trabalho.

2. Referencial Teórico

Este referencial teórico irá abordar os principais assuntos referentes a este trabalho, como IoT (Internet das Coisas), Automação e CLP, servindo como guia para as diferentes nomenclaturas utilizadas no decorrer do trabalho. Além disso serão mostrados sistemas existentes no mercado, relacionados ao trabalho.

2.1. Internet das Coisas (IoT)

A Internet das Coisas (*Internet of Things*, IoT) tem sido uma tendência a muitos anos e hoje em dia está sendo chamada de "terceira revolução industrial" [Guarnieri 2010]. Ela consiste de uma rede de aparelhos eletrônicos, sejam termômetros, carros ou torradeiras, que compartilham dados entre si [ITU 2015].

Este tipo de tecnologia pode ser utilizada para várias aplicações, como por exemplo em automação residencial, automação industrial (IIoT ou Internet Industrial das Coisas) ou coleta de dados como temperatura e pressão atmosférica. No presente trabalho, os CLPs e o sistema estarão conectados formando uma rede IoT.

2.2. Automação

Automação refere-se ao uso de sistemas de controle para operar equipamentos como motores, máquinas ou processos em indústrias, visando reduzir intervenção humana [Groover 2007]. Ela pode ser separada em dois grupos: automação residencial e automação industrial. A primeira refere-se ao uso de sistemas para o controle das funções de uma casa, como por exemplo, ligar e desligar luzes e eletrodomésticos, trazendo mais comodidade. A segunda refere-se ao uso de sistemas para controle de equipamentos e processos em indústrias, fazendo com que seja necessária uma menor quantidade de funcionários para operar as máquinas e diminuir a quantidade de erros humanos [Groover 2007].

Para fazer a interface entre o *software* de controle e os dispositivos físicos são utilizados Controladores Lógicos Programáveis (CLP), os quais podem trabalhar apenas de acordo com sua própria programação ou controlados por um sistema externo.

2.3. Controladores Lógicos Programáveis (CLP)

Controladores Lógicos Programáveis são computadores industriais utilizados para realizar o controle de equipamentos em indústrias, como linhas de montagem e motores. São criados para funcionar mesmo em locais com muita vibração, altas temperaturas e grande quantidade de poeira [Unitronics 2018].

CLPs podem funcionar de forma autônoma, seguindo sua programação e, as vezes, recebendo comandos através de botões físicos, ou podem ser comandados remotamente por meio de Sistemas de Supervisão e Aquisição de Dados. A maioria dos CLPs no mercado seguem a norma IEC 61131-3, a qual define 4 linguagens de programação que devem ser suportadas para a programação do controlador, sendo elas *Ladder Diagram*, *Function Block Diagram*, *Structured Text* e *Sequential Function Charts* [IEC 61131-3 2013]. A linguagem ST, a qual será utilizada no presente trabalho, será explorada abaixo. As outras linguagens são exploradas nos apêndices G, H e I.

2.3.1. Structured Text (ST)

```
2  IF #fp4q > 1 THEN
3      #read4qfppf := #fp4q - 2.0;
4      RETURN;
5  END_IF;
6
7  IF #fp4q < -1 THEN
8      #read4qfppf := -2.0 - #fp4q;
9      RETURN;
10 END_IF;
11
12 IF #fp4q = 1.0 OR #fp4q = -1.0 THEN
13     #read4qfppf := 1.0;
14     RETURN;
15 END_IF;
16
17  #read4qfppf := ABS(#fp4q);
```

Figura 1. Exemplo da linguagem ST

Ao contrário das demais linguagens para CLPs, Structured Text é uma linguagem baseada em texto. Sua sintaxe se assemelha a linguagem de programação Pascal e usa o paradigma de programação estruturado, conforme mostra a Figura 1 [IEC 61131-3 2013].

Por se assemelhar a linguagens de programação tradicionais, é a linguagem mais indicada para pessoas com experiência em linguagens como C, Pascal, ou outras linguagens estruturadas. Tal estrutura permite a fácil representação de cálculos complexos em poucas linhas de código, que ocupariam muito espaço e seriam de difícil leitura nas outras linguagens de programação disponíveis. Por esses motivos é uma linguagem muito utilizada por pessoas da área da computação, ou para criar funções que requerem um grande número de cálculos.

2.4. Sistemas de Supervisão e Aquisição de Dados (SCADA)

Sistemas SCADA são muito utilizados em indústrias para o controle e aquisição de dados de processos industriais [Daneels and Salter 1999]. Tais sistemas podem ser separados em duas camadas: a "camada do cliente" que é responsável a interação entre o usuário e o sistema por meio de uma interface; e a "camada do servidor de dados", responsável pela comunicação entre o sistema de controle e aquisição de dados e os dispositivos periféricos, como sensores e CLPs [Daneels and Salter 1999].

Tais sistemas costumam ser equipados com uma grande variedade de protocolos de comunicação para comunicação com periféricos, sendo o mais comum deles o protocolo ModBus. Tal protocolo será descrito com mais detalhes a seguir.

2.5. ModBus

ModBus é um protocolo de comunicação criado em 1979 para ser usado em CLPs. Tal protocolo é bastante utilizado atualmente para a intercomunicação entre aparelhos eletrônicos industriais [Drury 2001]. Sua utilização é livre de taxas de licenciamento [Modbus Organization 2018] e usa um sistema "mestre - escravo", em que um aparelho "mestre" realiza as requisições e os "escravos" respondem às mesmas.

O protocolo ModBus define um formato simples de mensagem. A mensagem começa com um byte representando o endereço do escravo que deverá receber a mensagem, seguido por um byte representando o código da função a ser executada, um número variável de bytes representando os dados a serem enviados para o escravo, e dois bytes de checagem de erros [Modbus Organization 2012]. O escravo então responde com uma mensagem no mesmo formato.

Os dados que podem ser lidos ou escritos de um escravo modbus são divididos em 4 tabelas: *Discretes Inputs*, *Coils*, *Input Registers* e *Holding Registers*. Cada tabela permite que se tenha até 65536 itens para leitura ou escrita. O funcionamento de tais tabelas está detalhado no apêndice A, juntamente com exemplos de comunicação ModBus.

2.6. Sistemas Existentes

Algumas aplicações SCADA existentes no mercado atualmente podem ser vistas nos apêndices B e C.

3. Trabalhos Relacionados

Alguns trabalhos relacionados ao presente serão discutidos a seguir:

3.1. Desenvolvimento de um sistema SCADA para operação de um Laser de Elétrons Livres

O trabalho proposto por [Pereira et al. 2015] tem como objetivo o desenvolvimento de um sistema SCADA para o controle e supervisão de diversos transdutores responsáveis pelo transporte, monitoramento e centralização do feixe de elétrons e da qualidade da radiação laser. O sistema foi criado usando a linguagem de programação LabVIEW com o intuito de ser uma solução aberta, escalonável e flexível.

Com o trabalho foi concluído que o sistema desenvolvido se mostrou capaz de alcançar seus objetivos. Ele foi utilizado para a caracterização elétrica e magnética dos transdutores eletromagnéticos e circuitos de condicionamento de sinal. Além disso, o sistema se mostrou capaz de ser adaptado para outros projetos de controle de processo ou experimentos científicos. Em comparação com o trabalho em desenvolvimento, tal trabalho relacionado se mostra como um caso de sucesso de implementação de um sistema SCADA e servirá como referência na implementação do presente trabalho.

3.2. Desenvolvimento de um Sistema de Controle e Aquisição de Dados para Testes do Detector de Fibras Cintilantes do Experimento LHCb

O objetivo do trabalho de [de Carvalho and Rodrigues 2016] é a criação de um sistema SCADA para a realização de testes em detectores de fibras cintilantes. Tal sistema realiza testes para aferir se os detectores não apresentam defeitos e trabalham em conformidade com os requisitos do experimento LHCb.

O sistema usa como base o *software* SCADA fornecido pela Siemens, chamado WinCC, com bibliotecas criadas durante este trabalho para a comunicação com os detectores. Uma vez que os primeiros detectores foram construídos, o sistema se mostrou crucial para o teste dos mesmos, sendo a única ferramenta capaz de se comunicar com todos os dispositivos presentes.

3.3. SCADA Pampa - Um Sistema de Supervisão e Aquisição de Dados Livre com fins didáticos

O trabalho de [de Carvalho Scherer et al. 2014] tem como objetivo criar um sistema SCADA genérico, livre e didático, escrito usando a linguagem de programação Java. O sistema controla um CLP através do protocolo ModBus, para realizar a coleta de informações e o controle dos equipamentos conectados ao mesmo.

Com o trabalho foi concluído que sistemas automatizados são fundamentais para o bom funcionamento de processos industriais, além de poderem ser utilizados em laboratórios de pesquisa que possuam equipamentos industriais. Tal sistema, sendo de uso genérico, necessitaria de configurações complexas para seu uso em automação industrial, além de possuir uma interface limitada e não intuitiva.

3.4. Considerações sobre os trabalhos relacionados

As abordagens descritas acima demonstram a grande versatilidade de sistemas SCADA, os quais podem ser utilizados no contexto científico, tanto quanto no industrial. Com relação ao presente trabalho, as grandes diferenças entre este e as abordagens citadas é a capacidade de operação multiplataforma, que é apenas demonstrado pelo *software* SCADA Pampa, e o foco do projeto. O objetivo deste trabalho é de criar uma ferramenta

para automação industrial que seja de fácil configuração, portanto os trabalhos acima não atendem a este critério, pois precisam de grandes modificações para se tornarem aptos para tal função.

4. Metodologia

Esta seção irá abordar a proposta do projeto, as ferramentas utilizadas e a metodologia utilizada no desenvolvimento.

4.1. Proposta

A proposta do presente trabalho é criar um sistema SCADA baseado em tecnologias Web, assim como programar um CLP para que se comunique com o sistema através do protocolo ModBus, sendo comandado pelo mesmo. O sistema deve permitir a configuração e operação de diversos motores, distribuídos em um ou mais CLPs, como ilustrado na Figura 2.

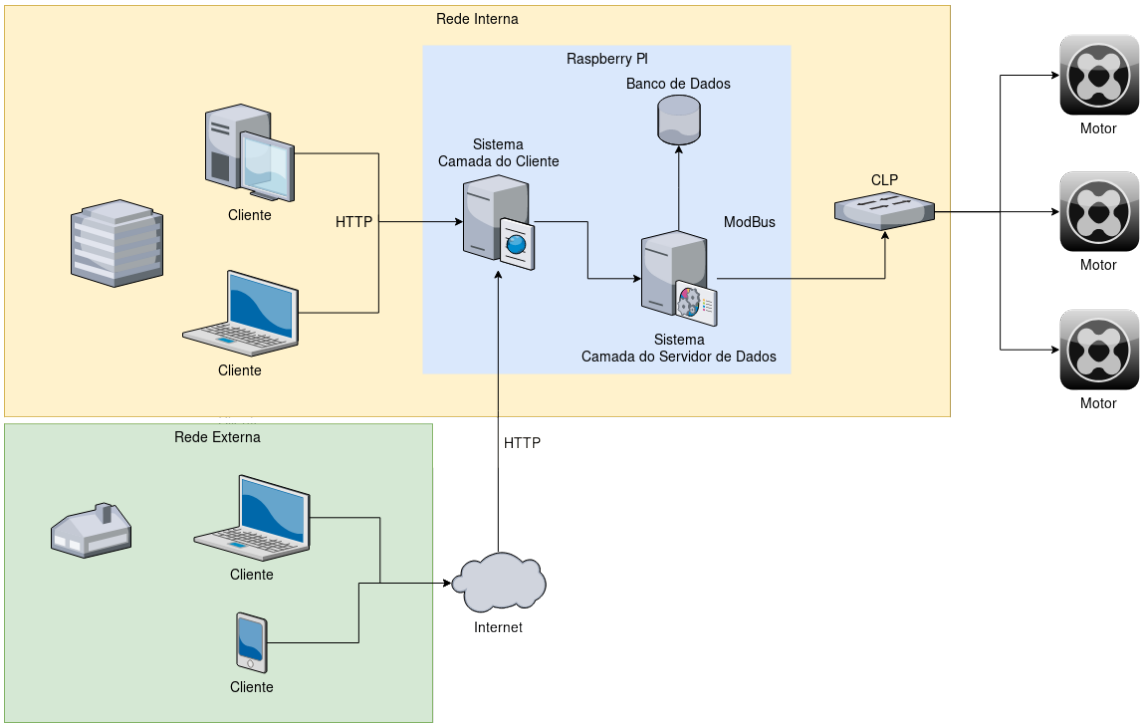

Figura 2. Arquitetura do Projeto

O sistema possui controle de acesso baseado em contas de usuário com senha. A tela de login é exibida ao acessar o sistema através de um navegador Web, e ao fazer login é exibida a tela de operações configurada no sistema. A tela de operações mostra o estado atual de todos os motores cadastrados para essa tela, além de permitir a operação dos mesmos. Podem ser cadastradas uma ou mais telas no sistema, cada uma com diversos motores.

O sistema realiza a leitura periódica dos CLPs configurados e armazena o estado dos mesmos na memória. Caso não consiga realizar a comunicação com algum deles, uma mensagem será exibida a todos os usuários logados, assim como gravada do banco de dados. O CLP tem a função de ser uma interface para os motores através de suas

entradas e saídas digitais. Para cada motor são necessárias uma entrada e uma saída, sendo a saída para controlar o estado do motor (comando do motor) e a entrada para verificar se o mesmo está mesmo ligado, servindo como confirmação (retorno do motor).

O CLP é conectado a rede Ethernet através de um conversor Serial para Ethernet, o qual pode ser configurado com um IP e porta para a interface com o sistema. A porta serial conectada ao conversor será configurada para se comunicar utilizando o protocolo ModBus para que seja possível a comunicação do mesmo com o sistema SCADA.

4.2. Projeto

No intuito de uma melhor organização do projeto, o mesmo está dividido em quatro módulos, sendo eles: Protocolos de comunicação, Programa do CLP, Camada do servidor de dados do sistema e Camada do cliente do sistema. As interações entre os mesmos estão descritas na Figura 2.

4.2.1. Protocolos de comunicação

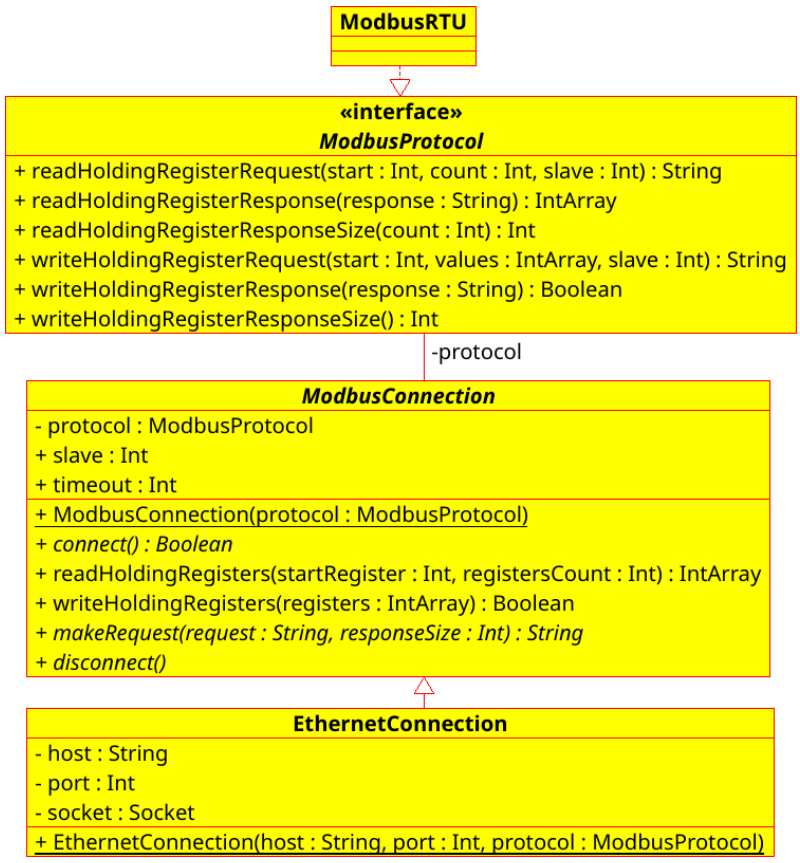

Figura 3. Diagrama de classes da implementação dos drivers de comunicação

A primeira etapa do projeto é a criação de uma interface que permita a interoperabilidade entre o sistema SCADA e o protocolo ModBus. O objetivo de tal interface é abstrair a comunicação de forma que modificações no protocolo ou a adição de novos protocolos semelhantes não acarretem na necessidade de reescrever o sistema SCADA para

comportar tais mudanças. A interface para abstração, assim como suas implementações estão descritas na Figura 3 como um diagrama de classes possuindo métodos para leitura e escrita de *Holding Registers*.

Para o desenvolvimento do protocolo, deve-se implementar a classe *ModbusProtocol* e seus métodos, o qual é responsável por criar as mensagens a serem enviadas para os dispositivos e fazer a compreensão das mensagens recebidas. Já a classe *ModbusConnection* é uma classe abstrata responsável por gerenciar a comunicação com o dispositivo, usando uma instância da interface *ModbusProtocol* para gerar e entender as mensagens. Tal classe deve ser implementada para cada meio de transmissão físico que irá ser usado (Serial, Ethernet, etc), permitindo que uma mesma implementação de protocolo seja utilizada através de vários meios físicos.

Para este projeto foram implementadas as classes *EthernetConnection* a qual implementa a classe *ModbusConnection* responsável pela comunicação através da rede Ethernet, e a classe *ModbusRTU* implementando a interface *ModbusProtocol* com as especificações do protocolo ModBus usado pelo CLP. Tais classes são mostradas no diagrama de classes da Figura 3.

4.2.2. Programa do CLP

O programa do CLP é responsável por receber os comandos do sistema SCADA através do protocolo ModBus, além de controlar e armazenar o estado atual de todos os motores conectados ao CLP. Ele tem seu desenvolvimento baseado na lista de requisitos do apêndice D:

4.2.3. Camada do servidor de dados do sistema SCADA

A Camada do servidor de dados do sistema SCADA faz a interface entre todos os outros módulos do projeto. As principais funcionalidades do sistema, listadas como requisitos, podem ser vistas no apêndice E.

Para melhor compreensão do funcionamento, a Figura 4 apresenta as principais classes usadas no sistema como um diagrama de classes. Tal diagrama serve como base para a implementação da camada do servidor de dados do sistema.

O banco de dados utilizado pelo sistema possui três funções principais: armazenar os dados de login e níveis de acesso dos usuários do sistema, armazenar as configurações do sistema em formato JSON e armazenar um log de eventos do sistema para motivos de auditoria. Na Figura 5 é mostrado o diagrama entidade-relacionamento do banco de dados.

A tabela *users* é responsável por armazenar os dados dos usuários do sistema, sendo a coluna *level* usada para definir o nível de acesso do usuário. A tabela *logs* armazena os eventos que ocorrerem no sistema, assim como o usuário que o causou e um número inteiro para servir como categoria na coluna *type*. Já a tabela *configs* armazena as configurações do sistema, a coluna *data* armazena os dados em formato JSON e o identificador é uma string para a fácil integração com o sistema Key-Value do formato JSON,

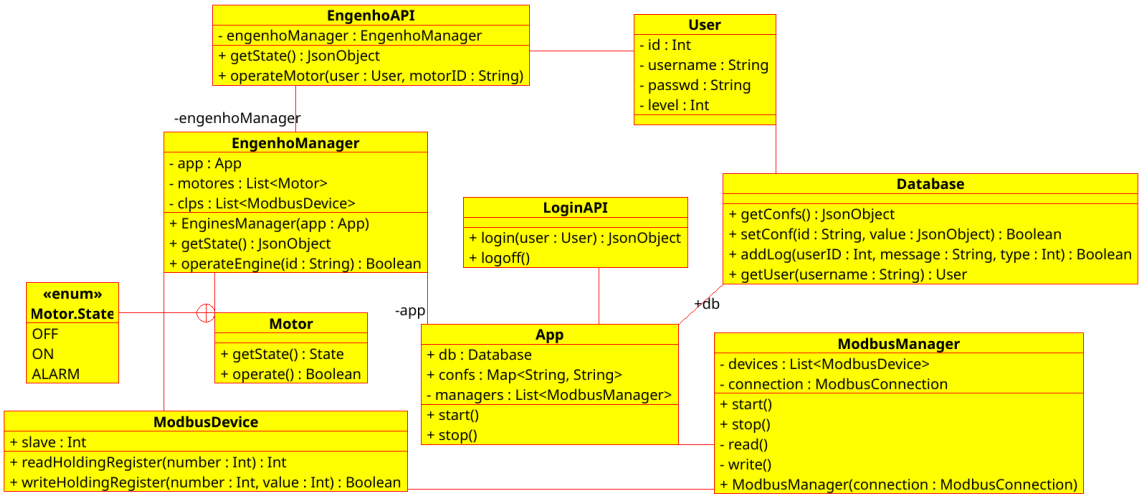

Figura 4. Diagrama de Classes da camada do servidor de dados

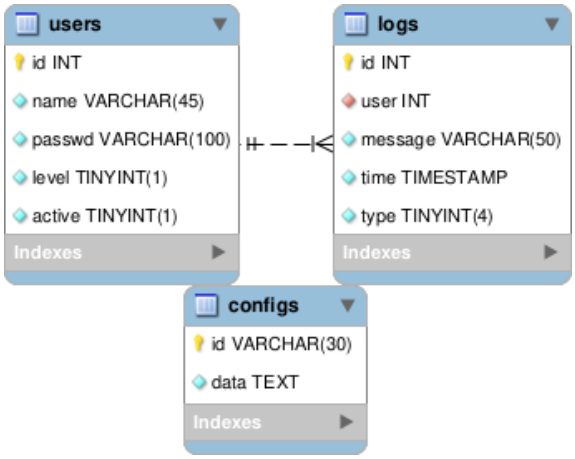

Figura 5. Diagrama Entidade-Relacionamento do Banco de Dados

assim toda a tabela *configs* pode ser serializada como um objeto JSON.

4.2.4. Camada do cliente do sistema SCADA

A camada do cliente serve como a interface gráfica do sistema, a qual é acessada a partir de um navegador web. Um diagrama de casos de uso pode ser encontrado no apêndice F. A interface possui as seguintes telas:

1. Tela de Login: Uma tela onde os usuários escrevem seus nomes de usuário e senha. Uma requisição à API é feita, e caso o login seja bem sucedido, o usuário é redirecionado para a tela dos motores.
2. Tela dos Motores: Tela onde são listados os motores cadastrados e seus estados atuais. Esta tela possui uma ou mais abas, cada uma com um número variável de motores, conforme configurado no banco de dados. Ao clicar em um motor, uma mensagem é exibida para confirmar a operação do motor, e caso confirmada, um comando de alteração de estado é enviado pela API para a camada do servidor de dados. A tela é atualizada em tempo real com os novos estados dos motores.

3. Tela de Configurações: Esta tela só pode ser acessada por usuários com permissão de administrador e permite a alteração das configurações armazenadas no banco de dados, sem a necessidade de serem alteradas acessando diretamente o banco de dados.

4.3. Ferramentas

A seguir serão definidas as ferramentas utilizadas na realização das diferentes partes do projeto.

4.3.1. Camada do servidor de dados do sistema SCADA e drivers de comunicação

Para estas duas partes é feito uso da linguagem de programação Kotlin, juntamente com o framework web reativo Eclipse Vert.x. A linguagem de programação Kotlin é uma das muitas linguagens disponíveis para a plataforma JVM (Java Virtual Machine), sendo utilizada como uma alternativa mais moderna a linguagem de programação Java, e totalmente compatível com bibliotecas escritas em outras linguagens da plataforma JVM [Samuel and Bocutiu 2017]. Já o framework Eclipse Vert.x é uma alternativa de alto desempenho a soluções convencionais de framework web, pois é baseada em eventos, não bloqueante e roda usando um baixo número de threads, características que o tornam viável para uso em sistemas embarcados [Eclipse 2018].

As configurações do sistema são armazenadas em formato JSON em um banco de dados MariaDB. O MariaDB é um sistema gerenciador de banco de dados SQL Open Source criado para substituir o MySQL, após sua compra pela empresa SUN em 2008, sendo totalmente compatível com sua sintaxe e bancos de dados existentes [Bartholomew 2012]. Além das configurações, outros dados como os logs e usuários do sistema são armazenados em suas respectivas tabelas.

O sistema é executado em um Raspberry PI 3 modelo B. O Raspberry é um computador embarcado de baixo custo, capaz de rodar sistemas operacionais Linux, ou outros sistemas baseados em arquitetura ARM. O sistema operacional utilizado é o sistema Arch Linux ARM, sendo uma distribuição linux extremamente leve, pois não vem pré instalado nenhum tipo de interface gráfica, permitindo que o sistema SCADA tire melhor proveito dos recursos limitados do computador.

4.3.2. Camada do cliente do sistema SCADA

Esta camada faz uso da linguagem de programação TypeScript e do framework React. A linguagem de programação TypeScript é uma linguagem criada pela empresa Microsoft, a qual gera programas em JavaScript quando compilada, permitindo sua interpretação por qualquer navegador web moderno [Bierman et al. 2014]. Já o framework React é utilizado para a criação do *frontend*, permitindo a geração de tags HTML dinamicamente através da linguagem de programação JavaScript ou qualquer outra que compile para JavaScript, removendo a necessidade de escrever páginas HTML estáticas [Cechinel et al. 2017].

4.3.3. CLP, conversor e programa do CLP

O CLP utilizado no projeto é o CLP Altus Duo DU351 e a linguagem de programação utilizada é a linguagem ST. O CLP é conectado ao conversor Serial para Ethernet USR-TCP232-410S. O CLP é capaz de se comunicar usando o protocolo ModBus através de uma interface serial RS485 ou RS232, A interface RS485 do CLP é conectada a interface RS485 do conversor USR-TCP232-410S. O conversor é configurado com um IP e porta de comunicação de acordo com a rede local a ser utilizada, para que possa se comunicar com o sistema SCADA.

5. Implementação

Esta seção abordará como o trabalho é implementado, mostrando o uso das bibliotecas e ferramentas em cada módulo do trabalho.

5.1. Protocolos de comunicação

Os protocolos de comunicação são implementados como uma biblioteca responsável pela communicação com os CLPs. As principais classes são *EthernetConnection* e *ModbusRTU*. A classe *EthernetConnection* é responsável pela implementação da comunicação TCP com o conversor serial para ethernet. Ela estende a classe abstrata *ModbusConnection*, implementando os métodos *connect*, *makeRequest* e *disconnect*, como mostra a Figura 6.

```
override fun makeRequest(request: String, resSize: Int): String {
    if (socket == null || !socket!!.isConnected)
        if (!connect()) {
            disconnect()
            return ""
        }
    for (c in request.toCharArray()) {
        out!!.write(c.toInt())
    }
    out!!.flush()
    var resp = ""
    for (i in 0 until resSize) {
        val r = `in`!!.read()
        if (r == -1) {
            System.out.println("Resposta incompleta: $host")
            disconnect()
            return ""
        }
        resp += r.toChar()
    }
    return resp
}
```

```
override fun connect(): Boolean {
    try {
        socket = Socket(host, port)
        socket!!.soTimeout = timeout
        `in` = socket!!.getInputStream().buffered()
        out = socket!!.getOutputStream().buffered()
    } catch (e: Exception) {
        System.out.println("Falha na conexão: $host")
        return false
    }
    return true
}

override fun disconnect() {
    try {
        socket?.close()
    } catch (e: IOException) {
        e.printStackTrace()
    }
    socket = null
}
```

Figura 6. Implementação da conexão TCP

Já a classe *ModbusRTU* é responsável por montar as requisições ModBus de acordo com o tipo de dado requisitado e fazer a tradução da resposta enviada pelo dispositivo para objetos que possam ser usados no programa. A Figura 7 mostra o método *readRequest*, o qual monta a requisição de leitura de registradores para ser enviada para o dispositivo.

5.1.1. Programa do CLP

A implementação do programa do CLP é separada em módulos, sendo o módulo *Motor* o mais importante do programa. Tal módulo é responsável pela lógica de monitoramento e controle das entradas e saídas do CLP. A implementação deste módulo pode ser vista na Figura 8.

```
private fun readRequest(start: Int, count: Int, slave: Int, fcode: Int): String {
    var bytes = IntArray(6)
    bytes[0] = slave
    bytes[1] = fcode                        //Function Code
    bytes[2] = (start shr 8) and 0xFF       //Start register first byte
    bytes[3] = start and 0xFF               //Start register second byte
    bytes[4] = (count shr 8) and 0xFF       //Register count first byte
    bytes[5] = count and 0xFF               //Register count second byte
    val crc = Util.calculateCrc(bytes)
    bytes = Arrays.copyOf(bytes, 8)
    bytes[6] = crc and 0xFF                 //CRC first byte
    bytes[7] = (crc shr 8) and 0xFF         //CRC second byte
    debug(bytes)
    return Util.intsToString(bytes)
}
```

Figura 7. Implementação da requisição de leitura ModBus RTU

```
IF Comando THEN
    IF Alarme THEN
        Alarme := FALSE;
    ELSE
        Saida := NOT Saida;
    END_IF
END_IF

Comando := FALSE;

IF Saida AND Ligado AND NOT Retorno THEN
    Saida  := FALSE;
    Alarme := TRUE;
END_IF

Ligado  := Retorno;

t(IN := Saida AND NOT Retorno, PT := TempoAlarme);

IF t.Q THEN
    Saida  := FALSE;
    Alarme := TRUE;
END_IF
```

Figura 8. Implementação da lógica dos motores no CLP

5.1.2. Camada do servidor de dados do sistema SCADA

O desenvolvimento da camada do servidor tem como base as classes *ModbusDevice* e *ModbusManager*. A classe *ModbusDevice* representa um dispositivo ModBus e expõe métodos para a leitura e escrita de registradores no mesmo. Já a classe *ModbusManager* representa um único meio físico de comunicação com um ou mais dispositivos ModBus, ou seja, cada conversor serial é representado por uma instância desta classe. Ela faz a leitura dos registradores necessários periodicamente, e os armazena em uma árvore binária para serem acessados instantaneamente quando necessários, além de realizar as operações de escrita nos dispositivos, as quais entram em uma fila de espera ao serem requisitadas pelo sistema (Figura 9).

A Figura 10 mostra a classe *Motor*, a qual serve como abstração dos motores físicos. Ela permite controlar e verificar o estado atual dos motores através dos métodos *getState* e *operate*. Cada motor físico é representado por uma instância desta classe no sistema. Outra classe muito importante do sistema é a classe *EngenhoManager*. Ela tem como objetivo reunir todos os motores e CLPs em um único lugar, centralizando o controle e informações dos motores físicos.

A comunicação com a camada do cliente é organizada pela classe *GraphQLAPI*, a qual expõe os dados necessários para a camada do cliente através de um endereço HTTP. O cliente realiza requisições HTTP neste endereço para requisitar o estado dos motores e

```
private fun writeRegs(mb: ModbusDevice) {
    mb.regList.forEach { (reg, vals) ->
        for (`val` in vals) {
            var r: Boolean
            try {
                r = conn.writeHoldingRegisters(reg, intArrayOf(`val`))
            } catch (e: Exception) {
                e.printStackTrace()
                break
            }
            if (!r) {
                Messager.addGlobalMessage("Erro de comunicação. (${mb.id});error")
                break
            }
        }
    }
    mb.regList.clear()
}
```

Figura 9. Implementação da escrita dos registradores na fila de espera

```
abstract class Motor(val numero: Int, val mb: ModbusDevice) {

    enum class Estado {
        ON, OFF, ALARM
    }

    abstract fun ligar(user: Int, obs: String? = null)
    abstract val estado: Estado

}
```

Figura 10. Implementação da classe abstrata *Motor*

requisitar a operação dos motores usando a linguagem de query *GraphQL*.

5.1.3. Camada do cliente do sistema SCADA

No desenvolvimento da camada do cliente são criadas as telas do sistema, as quais serão acessadas pelos usuários através de um navegador web. A tela de login do sistema é uma tela simples, a qual possui um título indicando o nome da unidade (neste caso "Teste"), um campo *username* para a digitação do nome de usuário, um campo *password* para a digitação da senha do usuário e um botão para realizar o login.

A principal tela do sistema é a tela do engenho, onde os motores são mostrados e podem ser operados. A tela do engenho pode ser vista na Figura 11, sendo que ela apresenta um botão para cada motor físico, representando seu estado ao mudar de imagem e podendo ser pressionado para ligar ou desligar o mesmo. Além disso possui um botão para realizar o *logoff* e um botão para abrir o menu de escolha de telas, caso tenha mais de uma tela configurada.

A configuração do sistema é feita a partir de uma tela acessível apenas a usuários administradores chamada tela de configurações. A tela permite visualizar, alterar e adicionar registros na tabela *conf* do banco de dados, os quais são lidos pelo sistema para controlar o comportamento do sistema. Nestes registros podem ser configuradas as telas e seus respectivos motores, os clps que o sistema deve se comunicar e configurações como nome da unidade e as configurações dos CLPs.

6. Validação

Como forma de validação do sistema, o mesmo foi inicialmente montado em um ambiente de testes. O mesmo consiste de um CLP carregado com o programa criado para o

Figura 11. Tela do engenho do sistema

mesmo, um notebook rodando o sistema SCADA e o banco de dados, e um conversor serial RS485. O diagrama do cenário pode ser visto na Figura 12. O sistema foi configurado para se comunicar com o CLP e uma tela com 8 motores foi criada. Após a configuração foi observado se ocorriam erros de comunicação ao observar os logs do sistema enquanto se alterava fisicamente o estado dos motores para verificar se os mesmos apareciam com o estado correto no sistema. Posteriormente foram testados os comandos enviados do sistema ao CLP, para verificar se estavam funcionando apropriadamente. Após os testes foi constatado que o sistema funciona normalmente e pode ser testado em ambiente de produção.

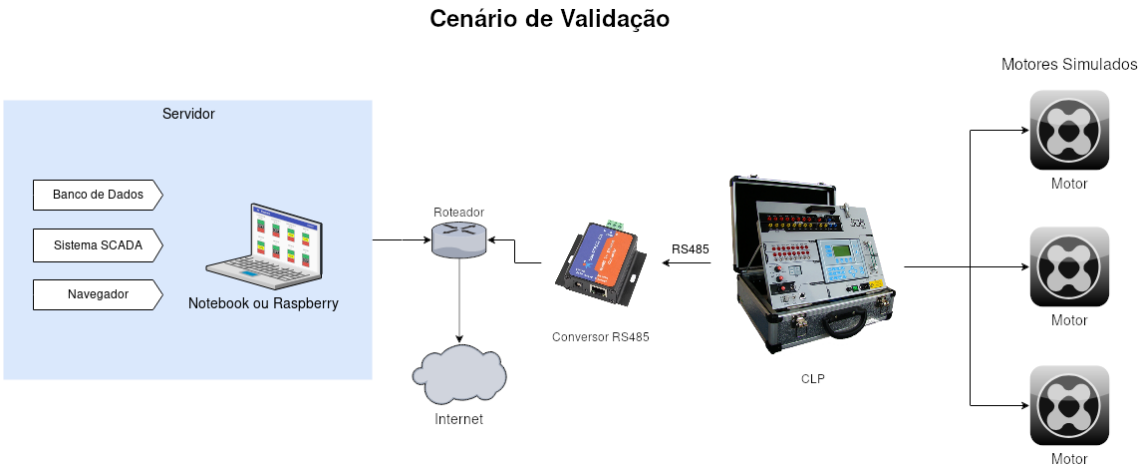

Figura 12. Cenário de validação e testes

Para o uso em produção, o sistema SCADA foi instalado em um Raspberry PI e conversores Serial para Ethernet foram instalados e conectados nos CLPs responsáveis pelo controle dos motores nos quadros de comando. O sistema foi configurado com as informações dos aeradores que precisavam ser acionados pelo sistema e o acesso ao sistema foi liberado para acesso através da rede local ou através da internet. O acesso remoto ao sistema demonstrou melhorar a operação dos motores, pois possibilita fazê-lo de qualquer lugar e de forma mais segura se comparada a operação manual através dos quadros

de comando elétricos. Além disso permite visualizar rapidamente a qualquer momento o estado atual dos motores para verificar o funcionamento e a existência de algum problema.

7. Conclusão

No presente trabalho foi apresentado o desenvolvimento de um sistema SCADA no intuito de ser uma solução de baixo custo e fácil configuração para a automação de indústrias. Tal sistema traz mais segurança e comodidade ao operar os motores de indústrias, permitindo a operação dos mesmos a distância usando um computador ou smartphone, além de permitir verificar os estados dos motores a qualquer momento.

Os sistemas existentes no mercado atualmente são, em sua maioria, sistemas proprietários que funcionam apenas em sistemas operacionais Windows, ou de difícil configuração por serem sistemas de uso geral, requerendo a programação de cada motor individualmente. Já o presente trabalho permite a instalação em qualquer sistema operacional que possua uma implementação da máquina virtual java, além de permitir a adição e remoção de motores através de configurações, dispensando a necessidade de programação para tal. Apesar de possuir um objetivo específico, a implementação do trabalho é modular, permitindo que várias melhorias futuras sejam acrescentadas, como a leitura de sensores de temperatura, controle de fator de potência, ou leitura de medidores de energia.

Referências

Bartholomew, D. (2012). Mariadb vs. mysql. *Dostopano*, 7(10):2014.

Bierman, G., Abadi, M., and Torgersen, M. (2014). Understanding typescript. In *European Conference on Object-Oriented Programming*, pages 257–281. Springer.

Cechinel, A. et al. (2017). Avaliação do framework angular e das bibliotecas react e knockout para o desenvolvimento do frontend de aplicações web.

Daneels, A. and Salter, W. (1999). What is scada?

de Carvalho, M. F. P. R. and Rodrigues, A. M. (2016). Desenvolvimento de um sistema de controle e aquisição de dados para testes do detector de fibras cintilantes do experimento lhcb development of a control system and data acquisition for tests of the scintilating fibers detector at the lhcb experiment. *NOTAS TÉCNICAS*, 6(2).

de Carvalho Scherer, F., da Silva Camargo, S., and Arruda, A. D. (2014). Scada pampa-um sistema de supervisao e aquisiç ao de dados livre com fins didáticos.

Drury, B. (2001). *Control techniques drives and controls handbook*. Number 35. IET.

Eclipse (2018). Eclipse vert.x. Disponível em: <https://vertx.io/>. Acesso em 06 Mai. 2018.

Elipse Software (2015). Elipse e3. Disponível em: <https://www.elipse.com.br/produto/elipse-e3/>. Acesso em 25 Abr. 2018.

Groover, M. P. (2007). *Fundamentals of modern manufacturing: materials processes, and systems*. John Wiley & Sons.

Guarnieri, M. (2010). The roots of automation before mechatronics [historical]. *IEEE Industrial Electronics Magazine*, 4(2):42–43.

IEC 61131-3 (2013). Programmable controllers – part 3: Programming languages. Standard, International Electrotechnical Commission.

ITU (2015). Internet of things global standards initiative. Disponível em: <https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>. Acesso em 05 Mar. 2018.

Modbus Organization, I. (2012). Modbus application protocol specification. *v1. 1b3, April*, 4.

Modbus Organization, I. (2018). Modbus faq. Disponível em: <http://www.modbus.org/faq.php>. Acesso em 24 Abr. 2018.

Pereira, J. M. B., Caetano, R., Galvao, R. M., and Cernicchiaro, G. (2015). Desenvolvimento de um sistema scada para operação de um laser de elétrons livres development of a scada system for a free electron laser operation. *NOTAS TÉCNICAS*, 5(3).

Samuel, S. and Bocutiu, S. (2017). Programming kotlin.

Sensorweb (2017). Scadabr. Disponível em: <http://www.scadabr.com.br/>. Acesso em 25 Abr. 2018.

Unitronics (2018). What is plc? programmable logic controller. Disponível em: <https://unitronicsplc.com/what-is-plc-programmable-logic-controller/>. Acesso em 12 Abr. 2018.

Wikipedia (2018). Basic batch sfc. Disponível em: <https://en.wikipedia.org/wiki/File:Basic_Batch_SFC.jpg>. Acesso em 2 Mai. 2018.

APÊNDICE A. Tabelas do protocolo ModBus

A tabela de *Discretes Inputs* permite operações de leitura somente. Os itens desta tabela possuem apenas 1 bit cada e normalmente representam portas de entrada de sinal digital em CLPs.

A tabela de *Coils* é muito parecida com a de *Discretes Inputs*, mas permite a escrita de dados além da leitura. Também possuem apenas 1 bit de dados por item e costumam representar as saídas digitais em CLPs.

A tabela de *Input Registers* permite apenas a leitura de dados. Os seus itens possuem 16 bits cada e podem ser utilizados para expor dados como leituras de sensores ou dados da memória interna do dispositivo.

A tabela de *Holding Registers* é como a de *Input Registers*, mas com a possibilidade de escrita além de leitura. Possuem 16 bits em cada item e é a tabela mais utilizada por sua grande versatilidade, podendo servir como uma maneira de visualizar e alterar parâmetros do escravo sem a necessidade de reprogramação.

O acesso aos dados é feito através de requisições utilizando códigos de funções. No entanto, diferentes dispositivos suportam um conjunto diferente de funções, sendo as contidas na Tabela 1 as mais utilizadas.

Tabela 1. Códigos de função do protocolo ModBus

Descrição	**Código de função**
Ler *Discrete Inputs*	02
Ler *Coils*	01
Escrever um *Coil*	05
Escrever vários *Coils*	15
Ler *Input Registers*	04
Ler *Holding Registers*	03
Escrever um *Holding Register*	06
Escrever vários *Holding Registers*	16

Exemplo de mensagem ModBus, ler *Holding Registers* 107 a 109 do escravo 17:

Tabela 2. Exemplo de mensagem modbus em hexadecimal

11	03	006B	0003	7687

Sendo o significado dos dados da Tabela 2 os seguintes:

- 11 - Número do escravo, pois 11 em hexadecimal corresponde ao número 17.
- 03 - Código da função para ler *Holding Registers*.
- 006B - Endereço do *Holding Register* inicial, pois 006B em hexadecimal corresponde ao número 107.
- 0003 - Quantidade de registradores a serem lidos, pois 0003 em hexadecimal corresponde ao número 3.
- 7687 - Bytes para checagem de erros.

Exemplo de resposta da mensagem anterior:

Sendo o significado dos dados da Tabela 3 os seguintes:

Tabela 3. Exemplo de resposta modbus em hexadecimal

11	03	06	AE41	5652	4340	47AD

- 11 - Número do escravo.
- 03 - Código da função para ler *Holding Registers*.
- 06 - Número de bytes de dados a seguir.
- AE41 - Dados do registrador 107.
- 5652 - Dados do registrador 108.
- 47AD - Dados do registrador 109.
- 47AD - Bytes para checagem de erros.

APÊNDICE B. Elipse E3

Sistema comercial brasileiro criado pela empresa Elipse Software. Funciona exclusivamente em sistemas operacionais Windows e possui módulos opcionais para acesso através da web [Elipse Software 2015].

O Elipse E3 é uma solução muito completa como um sistema SCADA de uso geral, permitindo a criação de telas gráficas, comunicação com dispositivos usando vários protocolos, incluindo o protocolo ModBus, possibilidade de ter vários servidores redundantes e alteração das configurações em tempo de execução. Sendo assim, Elipse E3 é uma solução muito robusta, no entanto necessita de licenças de uso para seu servidor, módulos web e ambiente de programação [Elipse Software 2015].

O presente trabalho tem como objetivo a criação de um sistema multiplataforma, que possa ser instalado em sistemas embarcados como o Raspberry PI, algo que não é possível no sistema Elipse E3. No entanto ele não contará com funcionalidades como criação de telas gráficas e servidores redundantes.

APÊNDICE C. ScadaBR

Sistema web brasileiro de código aberto escrito em java. Pode rodar em sistemas Windows, Linux ou MacOS e pode ser programado em JavaScript [Sensorweb 2017].

O sistema é feito para ser uma solução SCADA genérica, possuindo vários protocolos de comunicação, incluindo ModBus. Suas principais funcionalidades são o acesso web, criação de telas gráficas, alarmes e eventos parametrizáveis, sistema de permissões por usuário e a possibilidade de criar scripts em JavaScript [Sensorweb 2017].

O presente trabalho se propõe a implementar uma versão mais simples e focada do ScadaBR, que seja de fácil instalação e configuração, além de necessitar de menos poder de processamento. Algumas funcionalidades como os vários protocolos de comunicação, criação de telas gráficas e possibilidade de criar scripts não são implementadas no presente trabalho, pois não necessita ser tão genérico.

APÊNDICE D. Requisitos do programa do CLP

RF 1 Responder a requisições ModBus através de uma porta serial.
RF 2 Enviar sinal elétrico ao motor ao receber comando de ligar motor.
RF 3 Parar de enviar sinal elétrico ao motor caso o mesmo não ligue em um determinado período de tempo (tempo de alarme).

RF 4 Parar de enviar sinal elétrico ao motor ao receber comando de desligar motor.
RF 5 Atualizar o estado dos motores em tempo real de acordo com o estado das entradas digitais do CLP.
RF 6 Registrar o estado do motor como em alarme caso o mesmo não ligue ou seja desligado por meios externos (falta de luz, por exemplo).

RNF 1 A porta serial usada deverá ser uma porta RS485.
RNF 2 A velocidade de comunicação da porta serial será de 115200 bauds.
RNF 3 O tempo de alarme deve ser configurável através da interface ModBus.

APÊNDICE E. Requisitos da camada do servidor de dados

RF 1 Carregar as configurações a partir do banco de dados
RF 2 Periodicamente ler os registradores necessários dos CLPs e armazenar seus valores em memória.
RF 3 Verificar o estado dos motores e armazenar qualquer alteração nos mesmos no banco de dados.
RF 4 Prover uma API HTTP para a comunicação com a camada do cliente do sistema.
RF 5 A API deve gerenciar o login dos usuários, buscando seus dados no banco de dados, e armazenando no mesmo quando algum usuário efetuar o login.
RF 6 A API deve permitir que usuários logados enviem comandos de alteração de estado para os motores, e armazenar no banco de dados quando eles ocorrerem.
RF 7 A API deve permitir que usuários logados verifiquem o estado atual dos motores.
RF 8 A API deve permitir que usuários logados tenham acesso a configurações da interface gráfica armazenadas no banco de dados.
RF 9 Servir a camada do cliente como uma aplicação web estática.

APÊNDICE F. Requisitos da camada do cliente

Na Figura 13 é mostrado o diagrama de casos de uso. Tal diagrama descreve os atores do sistema, assim como seus papéis no mesmo, de modo simplificado.

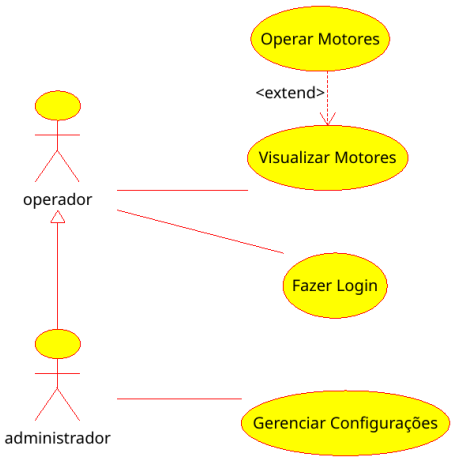

Figura 13. Diagrama de casos de uso da camada do cliente do sistema SCADA

A Figura 14 apresenta o protótipo da tela de motores. Nele é mostrado três motores, o primeiro ligado e os outros dois desligados. Tal protótipo serve como base para o desenvolvimento da tela final.

Figura 14. Protótipo da Tela dos Motores

APÊNDICE G. Ladder Diagram (LD)

A linguagem gráfica Ladder Diagram, também chamada apenas de Ladder, é baseada em circuitos eletrônicos. Sua programação é feita por meio de contatos e bobinas, de forma semelhante a um esquema elétrico [IEC 61131-3 2013]. Por esse motivo, a linguagem se torna fácil de aprender por pessoas acostumadas com diagramas elétricos, como engenheiros elétricos.

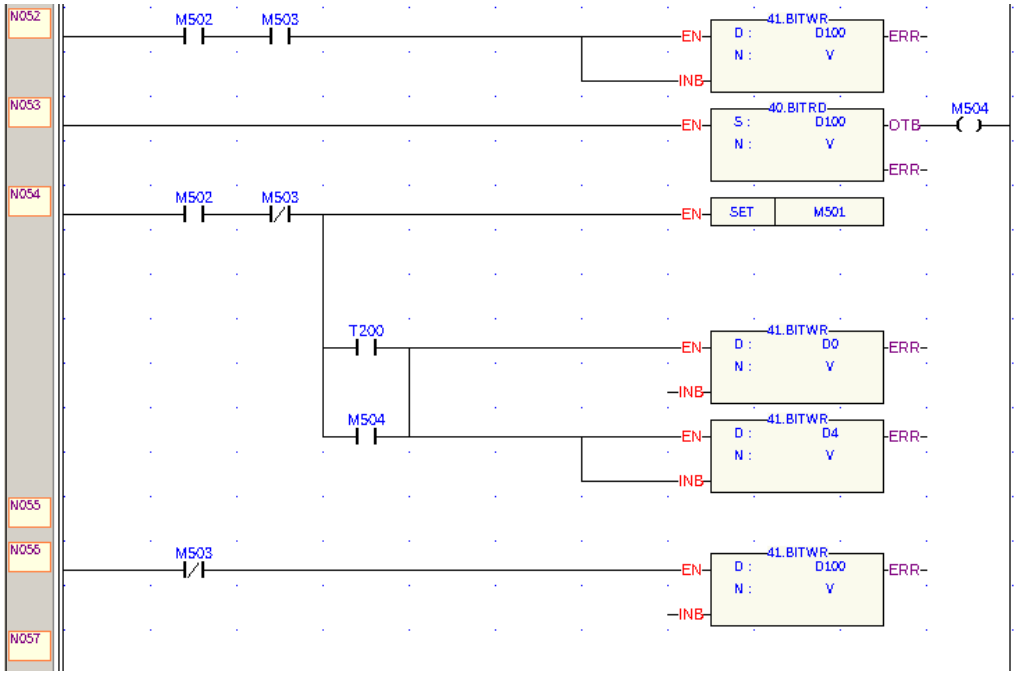

Figura 15. Exemplo da linguagem Ladder

Por ser muito semelhante à esquemas físicos elétricos, a linguagem acaba sendo recomendada para processos que requerem lógica simples, como ligar uma saída de acordo com um sinal de entrada. Tarefas mais complexas, como cálculos e lógicas mais complexas podem ser realizadas na linguagem Ladder com a ajuda de blocos de funções, os quais podem realizar cálculos simples como somas, subtrações e multiplicações, entre outras funções variadas, dependentes da implementação específica da linguagem.

Possuindo conceitos tão simples, a linguagem LD é de fácil aprendizado e permite a rápida construção de sistemas simples. No entanto, sistemas mais complexos acabam

sendo prejudicados pela necessidade de um grande número de contatos auxiliares necessários para fazer a ligação entre as entradas e saídas de diferentes blocos de função. A Figura 15 apresenta um exemplo de programa LD que mostra os vários componentes da linguagem e como podem ser conectados.

APÊNDICE H. Function Block Diagram (FBD)

Assim como a LD, Function Block Diagram se trata de uma linguagem gráfica, no entanto é baseada em blocos de função. Cada bloco possui um número variável de entradas e saídas e a lógica do programa é formada ao ligar as entradas e saídas de diferentes blocos em outros blocos ou em registradores de memória [IEC 61131-3 2013].

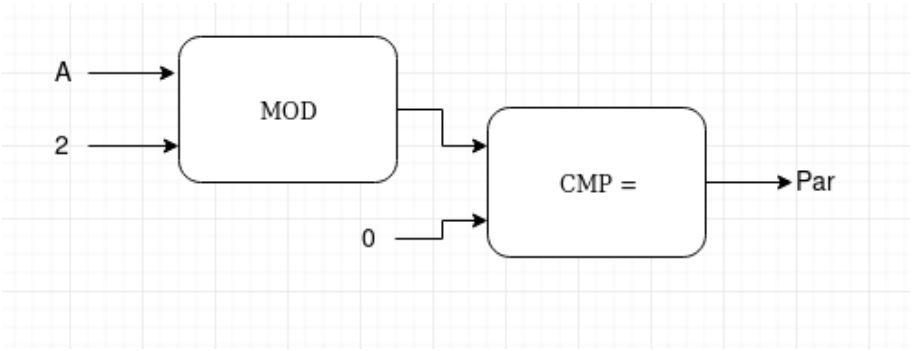

Figura 16. Exemplo da linguagem FBD

Os blocos são conectados entre si com linhas, as quais enviam o sinal da saída a esquerda até uma entrada a direita. A Figura 16 ilustra um programa que compara duas variáveis, e caso a variável LstWert for maior que a MaxSp, copia os dados de LstWert para a variável MaxSp. Os elementos que podem ser conectados são:

- Uma variável e uma entrada de função
- Uma saída de função e uma entrada de outra função
- Uma saída de função e uma variável

APÊNDICE I. Sequential Function Charts (SFC)

A linguagem gráfica Sequential Function Charts consiste de um diagrama semelhante a um diagrama de atividades da UML. Ele é utilizado para escrever programas com controle de concorrência, pois possui elementos nativos que realizam estas funções, permitindo a escrita de códigos concorrentes de maneira mais natural [IEC 61131-3 2013].

Esta linguagem pode ser utilizada para a definição de complexos fluxos paralelos, além de permitir uma boa visualização do fluxo geral da aplicação de forma intuitiva, como pode ser visto na Figura 17, que mostra um exemplo de programa SFC e as partes que o compõem. Cada passo (bloco quadrado na Figura 17) representam blocos de programas, que são criados usando as outras linguagens de programação vistas nesta sessão, por isso, a linguagem SFC é utilizada em conjunto com as outras linguagens da norma para controlar o fluxo do programa de forma macroscópica.

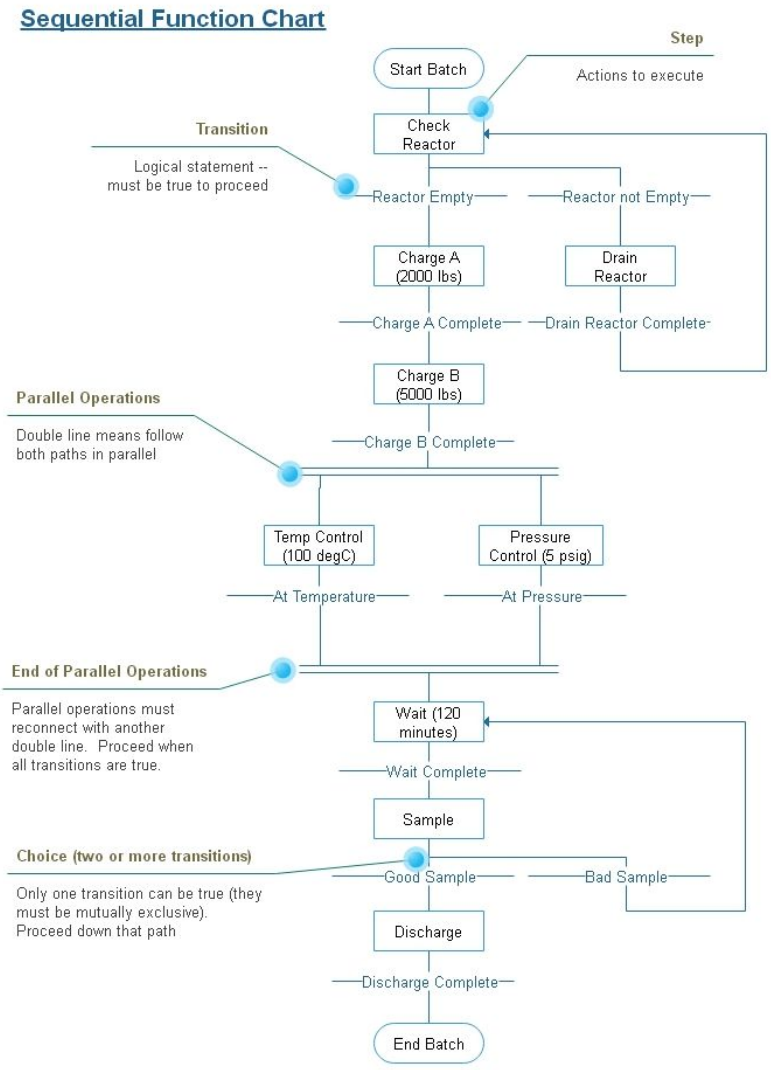

Figura 17. Exemplo da linguagem SFC [Wikipedia 2018]