UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

THIAGO DE LIMA

DOCUMENTAÇÃO PARA DESENVOLVIMENTO DE UM SISTEMA SUPERVISÓRIO BASEADO EM UMA APLICAÇÃO GENÉRICA

Joinville
2022

THIAGO DE LIMA

DOCUMENTAÇÃO PARA DESENVOLVIMENTO DE UM SISTEMA SUPERVISÓRIO BASEADO EM UMA APLICAÇÃO GENÉRICA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Dalton Luiz Rech Vidor

Joinville
2022

AGRADECIMENTOS

Aos meus pais, Darci e Ivone, por me apoiarem e dar todo o suporte necessário durante toda essa etapa da graduação.
A minha companheira, Dality, pelo incentivo e paciência nos momentos de dificuldades.
Aos meu irmãos e familiares que me apoiaram nas tomadas de decisões e estiveram do meu lado.
A todos os meus amigos aos quais sempre foram grandes parceiros nas horas de descontração.
A todos os professores da UFSC que contribuíram de alguma forma para o meu crescimento intelectual, profissional e pessoal. Ao meu orientador Dalton, pela prontidão em me orientar e apoiar na realização deste trabalho e acabou se tornando um grande amigo que levarei para a vida.

RESUMO

As tecnologias de automação industrial estão passando por grandes mudanças devido ao avanço dos sistemas de monitoramento online, com isso o mercado de trabalho está em busca de profissionais que sejam capacitados para o desenvolvimento desses sistemas. Dentre esses sistemas encontram-se os sistemas supervisórios que têm como função monitorar e controlar outros sistemas a nível de chão de fábrica possibilitando e possibilitar a integração com toda a planta. Levando em consideração este cenário, o presente trabalho apresenta uma abordagem descritiva para o desenvolvimento de toda a estrutura de um sistema supervisório simulando um processo genérico de uma indústria, utilizando elementos básicos desses sistemas. Todas as etapas utilizadas para o desenvolvimento do supervisório foram descritas afim de servir para material de apoio à acadêmicos e profissionais da área. A aplicação foi toda desenvolvida no *software* de desenvolvimento *ElipseE3*. Os recursos desenvolvidos foram todos testados e simulados através da execução da aplicação e da simulação por *software* representando um dispositivo no chão de fábrica.

Palavras-chave: Supervisório. Documentação. Simulação.

ABSTRACT

Industrial automation technologies are undergoing major changes due to the advancement of online monitoring systems, so the job market is looking for professionals who are trained to develop these systems. Among these systems are the supervisory systems whose function is to monitor and control other systems at the factory floor level and enable integration with the entire plant. Taking this scenario into account, this work presents a descriptive approach for the development of the entire structure of a supervisory system simulating a generic process of an industry, using elements commonly used in it. All stages used for the development of the supervisory were described to serve as support material for academics and professionals in the area. The entire application was developed using the ElipseE3 development software. The developed features were all tested and simulated by running the application and software simulation representing a device on the factory floor.

Keywords: Supervisory. Documentation. Simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os sistemas de controle e aquisição de dados estão sendo fortemente impactados com o desenvolvimento de novas tecnologias, uma vez que as plantas estão passando por um processo de virtualização e é importante que estejam preparadas com a sua estrutura de aquisição de informações em pleno funcionamento para se ter dados confiáveis sobre os seus processos.

A disponibilidade e segurança da informação desses sistemas supervisórios mostram a importância deles no âmbito industrial. No início eles eram apenas utilizados como meras ferramentas operacionais, porém atualmente se encontram em um alto nível dentro da cadeia de automação, pois percebeu-se os valores de se monitorar e controlar ao máximo os processos de uma linha de produção (PINHEIRO, 2006).

Os sistemas supervisórios são encontrados em diversas áreas da indústria e sua aplicabilidade é extensa, porém alguns padrões se repetem em uma aplicação supervisória. A animação de objetos em falha ou em funcionamento, a visualização dos dados em tempo real com o dispositivo e os gráficos históricos, são as principais ferramentes de um supervisório.

1.1 OBJETIVO

A demanda de profissionais que dominam os conceitos e o desenvolvimento de sistemas supervisórios vem crescendo, devido a automação e monitoração das plantas industriais. Observou-se que materiais e treinamentos sobre esse tema são escassos, e a fim de contribuir para a difusão de conhecimento a respeito do tema, este trabalho abordará e desenvolverá um sistema supervisório.

1.1.1 Objetivo Geral

Documentar todo o desenvolvimento um sistema supervisório para uma aplicação considerada genérica, pois apresenta elementos frequentemente utilizados nestas aplicações, para fins de aprendizado ou aplicação prática.

1.1.2 Objetivos Específicos

- Apresentar a estrutura de um sistema supervisório;
- Desenvolver layouts padrões utilizando classes e objetos;
- Documentar as configurações e funções do sistema supervisório;
- Validar o desenvolvimento através de uma simulação.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão demonstrados os fundamentos utilizados para a realização do trabalho. Sendo assim, o capítulo 2 abordará de forma breve a história da automação industrial, dos sistemas supervisórios e apresentará os conceitos que englobam a concepção de um sistema supervisório, desde a arquitetura em campo até a estrutura de software.

2.1 HISTÓRIA DA AUTOMAÇÃO INDUSTRIAL

Desde 1940 o conceito de automação vem sendo utilizado para definir sistemas que vinham substituindo processos que necessitavam de esforço manual ou eram repetitivos e passaram a ser automáticos ou com menos intervenção humana (LAMB, 2015). A implementação da automação na indústria iniciou-se com a utilização de dispositivos eletromecânicos, como relés de lógica fixa, temporizadores, botões, posicionadores mecânicos, que realizavam sequências simples de atuação de motores e atuadores (PAREDE, 2011).

Para Lamb (2015), ao se automatizar processos existe um ganho de produtividade e segurança, pois se reduz a frequência de intervenção humana no processo, reduzindo a taxa erros de operação e acidentes de trabalho.

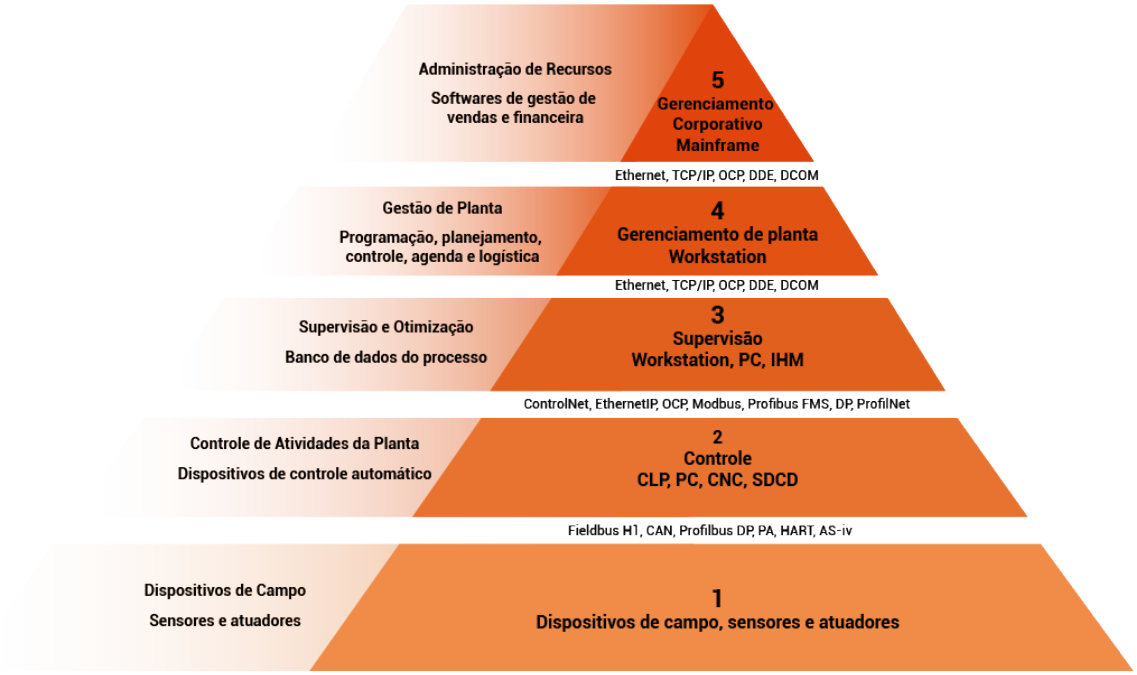
A evolução tecnológica trouxe também diversas mudanças para o mundo da automação, o desenvolvimento de computadores com cada vez mais poder de processamento de dados e as redes de comunicação capazes de transmitir dados a velocidades cada vez maiores foram fundamentais nesse processo de expansão da automação industrial (MAITELLI, 2003).

Os sistemas de automação passaram a envolver toda a planta industrial, uma vez que qualquer sistema, comercial ou industrial geram informações que precisam ser armazenadas e tratadas. Essas informações podem mostrar se determinado pedido pode ser atendido com o estoque de produção atual, quanto tempo as máquinas levam para produzir o pedido e também o custo aproximado. Porém, para que para que essas informações sejam visíveis a todos os níveis da planta é necessário uma integração entre todos esses equipamentos para que exista a troca de informações e assim elas sejam processadas e transformadas em valor para a empresa (MAITELLI, 2003).

Essas topologias de integração de sistemas dentro da industria possibilita uma melhor coordenação e controle de produção, habilita uma visão total do processo de fabricação e pode estabelecer uma melhor gestão de gerenciamento dos negócios, reduzindo o custo de fabricação, aumentando a eficiência e qualidade do processo. Tal integração em diferentes níveis de equipamentos desde do chão de fábrica até

servidores de banco dados pode ser exemplificada como a pirâmide da Figura 1.

Figura 1 – Pirâmide da Automação Industrial



Fonte: ROURE (2017)

- **Nível 1** - *Aquisição de Dados e Dispositivos de Campo*: esse nível é formado basicamente por todos os sinais de campo como sensores, transmissores, atuadores que se situam no chão de fábrica;
- **Nível 2** - *Controle*: composto pelos Controladores Lógicos Programáveis (CLP's), Sistemas Digitais de Controle (SDC's) e reles, que são responsáveis por controlar atividades individuais na planta;
- **Nível 3** - *Supervisão e Otimização do processo*: nível onde os dados do processo são armazenados e tratados, também neste nível pode se encontrar os sistemas de supervisão, controle e aquisição de dados (SCADA), ou supervisórios, esses sistema são centralizadores de informação, e podem atuar em uma ou mais atividades do processo;
- **Nível 4** - *Gestão da Planta*: nível responsável pelo gerenciamento de suprimentos para a planta, matéria ou mão de obra, através de planejamento e programação.
- **Nível 5** - *Administração de Recursos*: nível onde são tomadas decisões que afetam a empresa como um todo, baseada em *softwares* que utilizam dos dados das camadas inferiores, para gerar informações de gestão de venda, financeira e BI *(Business Intelligence)*.

2.1.1 Controladores Lógicos Programáveis

A automação industrial, antes do advento dos microprocessadores, era baseada em relés, chaves de fim de curso e outros dispositivos eletromecânicos que ocupavam grandes painéis de comando. Esses dispositivos eram responsáveis por toda a lógica e funcionamento automático do sistema. A tecnologia evoluiu a ponto desses microprocessadores serem utilizados para substituir esses painéis, assim surgiram os CLP's sigla para *Controlador Lógico Programável*(PESSOA, 2014). Um *CLP* possui *hardware* e *software* compatível com o ambiente industrial, sua memória programável possibilita a aplicação em diferentes tipos de processos.

Atualmente existem diversos fabricantes de *CLP* no mercado, porém todos possuem uma estrutura base para realizar a função de controle: módulo de processamento, módulo de entrada e módulo de saída. O módulo de processamento é responsável pelo armazenamento e execução do *software* carregado na memória do controlador, já o módulo de entrada recebe as informações do chão de fábrica, como valores de variáveis do processo(temperatura, pressão,etc.) e o módulo de saída envia informações para o chão de fábrica. O funcionamento de um *CLP* ocorre de forma cíclica conforme a sequencia de operação abaixo (SILVEIRA L.; LIMA, 2003):

- **Varredura de entrada:** todas as entradas do módulo de entrada são lidas(tipicamente dura poucos milissegundos);
- **Varredura do programa:** percorre todas as instruções do programa carregado no módulo de processamento, levando em consideração as entradas lidas (duração variável com o tamanho e complexidade do programa carregado na memória);
- **Varredura de saída:** escreve no módulo de saída o resultado das instruções após o processamento delas (tipicamente dura poucos milissegundos).

2.1.2 Redes e Protocolos de Comunicação

As interfaces de comunicação entre o controlador e os equipamentos de campo são muitas. Existem diferentes métodos para a transferência de dados entre os pontos de coleta de informação (entrada), processamento (CLP) e atuadores(saídas), tais métodos são chamados de protocolos de comunicação e acontecem nas redes de comunicação(LAMB, 2015). Para YAMAGUCHI (2006) a definição de uma rede de comunicação para automação de um sistema deve se levar em conta alguns princIpais aspectos como:

- Meio físico de transmissão;
- Velocidade de comunicação;
- Método de comunicação.

O meio físico de transmissão se refere por onde e como os dados serão transmitidos entre os equipamentos do sistema, atualmente existem três principais tipos

de meio físico de transmissão dentro da automação, os cabos, mais comum, através de sinais elétricos. O ar também pode ser utilizado como meio físico para transmissão de dados com tecnologia *wireless*, sem fio, através de ondas de rádio. E por fim, sinais de luz, são utilizados para transmissão de dados através das fibras ópticas.

A velocidade de comunicação é a taxa em que dados são transferidos pela rede, normalmente medido em kilobits ou megabits por segundo.

Os métodos de comunicação são a maneira em que será feita a troca de informações entre os dispositivos conectados. O principais métodos são de produtor-consumidor e origem-destino.

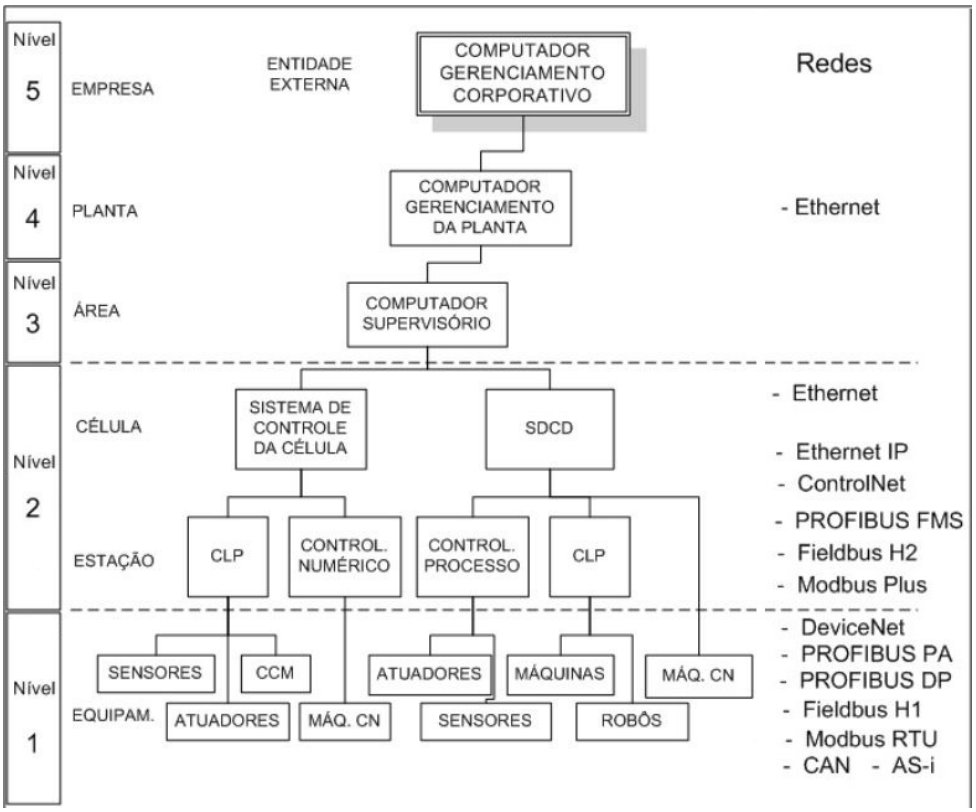
O método produtor-consumidor consiste em um nó que produz informações, o produtor, e um nó que solicita as informações do produtor que ele precisa, o consumidor. Já no método origem-destino, os dados são produzidos e enviados para os outros nós. Esse envio pode ser *Unicast*, onde o nó que produz a informação envia a mesma para apenas um outro nó, *Broadcast* onde o nó que produz envia para todos os nós da rede e *Multicast* onde a mensagem é enviada para nós específicos da rede (YAMAGUCHI, 2006).

Conforme cita Lamb (2015) a maneira em que esses dados são transmitidos através do meio físico podem ser de forma:

- **Serial:** por meio sequências digitais de zeros e uns, enviadas por um único fio;
- **Paralela:** múltiplas linhas paralelas realizam a transmissão de dados;
- **Ethernet:** utiliza um cabeamento em forma de par trançado, coaxial ou fibra óptica, a velocidade via Ethernet é superior à outras formas de transmissão, mas por ser um modelo de rede não determinístico, onde não é possível garantir um tempo máximo de resposta, não é recomendado para controle de sistemas críticos.

Assim como mostrado anteriormente na pirâmide de automação, a Figura 2 mostra que as redes de automação e protocolos de comunicação também possuem um nível hierárquico, e são utilizados em diferentes níveis da camada de automação.

Figura 2 – Nível Hierárquico Redes de Automação



Fonte: Adaptado de YAMAGUCHI (2006)

Os principais protocolos de comunicação que aparecem na Figura 2 serão descritos a seguir, com as informações básicas de seu funcionamento e comportamento.

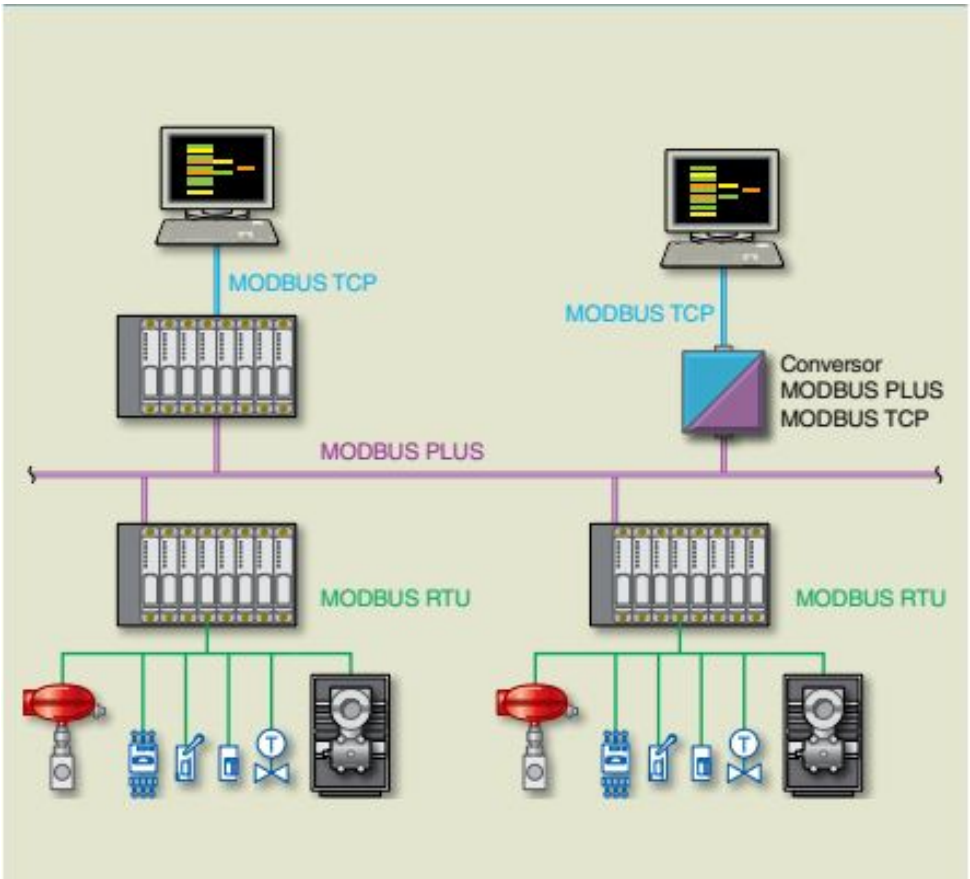
2.1.2.1 MODBUS

Desenvolvido em 1979, sendo um protocolo aberto, de fácil configuração e alta robustez na transferência de dados é utilizado em indústrias pelo mundo todo. Possui três principais versões: ModBus RTU*(Unidade Terminal Remota)*, ModBus Plus e ModBus TCP/IP (PAREDE, 2011). O principal meio físico de transmissão de dados é o serial RS-232 ou RS-485 para as versões RTU e Plus, já a TCP/IP usa redes Ethernet. Um exemplo de rede Modbus é mostrado na Figura 3

A literatura usa a terminologia *Mestre* e *Escravo*, para descrever os níveis dos dispositivos em alguns protocolos de comunicação. Neste trabalho os termos utilizados serão *Consumidor* e *Produtor* respectivamente.

O fluxo de comunicação acontece com o consumidor requisitando as informações para o produtor, tais requisições são previamente programadas no consumidor, já o produtor deve estar preparado com valores de endereços para responder quando solicitado. O pacote de dados enviado pelo consumidor possui o endereço requisitado, o código da função a ser executada, quantidade de registros a

Figura 3 – Exemplo de topologia de rede Modbus



Fonte: Parede (2011)

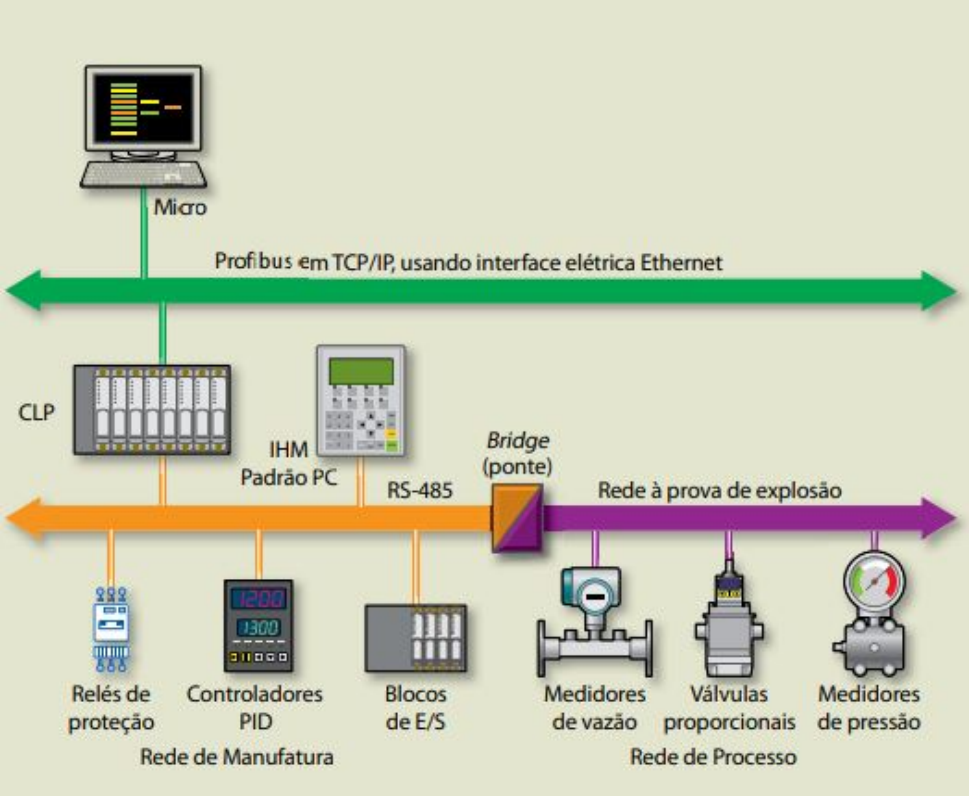
serem lidos e um código para validação de recebimento, já na resposta do produtor é enviado o mesmo código de validação, confirmando o recebimento da mensagem, o endereço solicitado, os dados do endereço e o código da função a ser executada, caso ocorra algum erro na transmissão o código é alterado e é escrito nos dados do endereço código do erro equivalente (PAREDE, 2011).

2.1.2.2 PROFIBUS

O protocolo Profibus foi desenvolvido por empresas Alemãs (LAMB, 2015) e é amplamente utilizado na automação de produção e na automação de processos como um protocolo de campo serial. As variações dele são o PROFIBUS-PA(*Process Automation*), utilizado para monitorar variáveis do processos, e o PROFIBUS DP (*Decentralized Peripherals*), utilizado para controle de dispositivos de campo como atuadores, via controlador central. Um exemplo de arquitetura PROFIBUS é mostrado na Figura 4.

Existem conversores, que possibilitam a transferência de dados de um meio físico à outro, por exemplo um conversor de padrão serial (RS-485) para Ethernet,

Figura 4 – Exemplo de topologia de rede Profibus



Fonte: Parede (2011)

esses dispositivos são chamados de *gateways* (PAREDE, 2011).

Dentro da topologia pode haver múltiplos consumidores, porém apenas tem acesso ao barramento de comunicação o consumidor que possui o *token*. Esse *token* é passado para cada dispositivo consumidor em tempo e sequência pré-determinada, evitando que haja colisões de dados entre consumidores (CLARKE; REYNDERS, 2004).

2.1.2.3 TCP/IP

TCP/IP é o nome dado ao conjunto de protocolos utilizado pela rede Ethernet, esses acrônimos são referentes ao seus principais protocolos *Transmission Control Protocol* e *Internet Protocol* (LAMB, 2015).

Formada por várias camadas, o fluxo dos dados acontece da camada inferior para superior através de *frames*. Esses *frames* possuem os endereços de origem e destino, os dados e a informação de verificação de erro (LAMB, 2015).

2.1.3 Sistemas de supervisão

Como parte fundamental da automação industrial, encontra-se a supervisão e o controle dos processos, mostrado anteriormente na Figura 1, que pode ser feita através

de IHM (interface homem-máquina) e/ou sistemas supervisórios (BRANQUINHO, 2014).

Antes essa integração da percepção humana com o processo industrial era feita através de um painel sinótico. Nele estavam presentes sinaleiros, chaves seletoras, botões para comando, lâmpadas para visualização de estados, entre outros itens para simular o processo para o operador (MORAES; CASTRUCCI, 2007).

O desenvolvimento das interfaces homem-máquina (IHMs) facilitou a visualização e desenvolvimento desse sistema de supervisão, a complexidades dos painéis sinóticos foram substituídos por telas sensíveis ao toque, com teclados alfanuméricos e com telas que representavam de forma fiel o processo (MORAES; CASTRUCCI, 2007).

Segundo Branquinho (2014) os sistemas supervisórios são um pacote de software posicionado no topo da camada de hardware, ao qual se conectam os dispositivos de campo, principalmente os CLPs.

O sistemas supervisórios entram na pirâmide de automação pois automatizam os processos de monitoração, aquisição, coleta e tratamentos dos dados de campo em ambientes complexos e que possam estar geograficamente dispersos (BRANQUINHO, 2014).

Os dados são coletados periodicamente ou através de eventos pré-determinados e armazenados em banco de dados que podem ser exibidos em tempo real ou em pesquisas posteriores para relatórios ou análises do processo (LAMB, 2015).

Assim como nas IHMs o operador também interage com o processo através de interfaces gráficas amigáveis, que representam o processo de forma fiel, para facilitar o entendimento e evitar erros de interpretação humana (MORAES; CASTRUCCI, 2007).

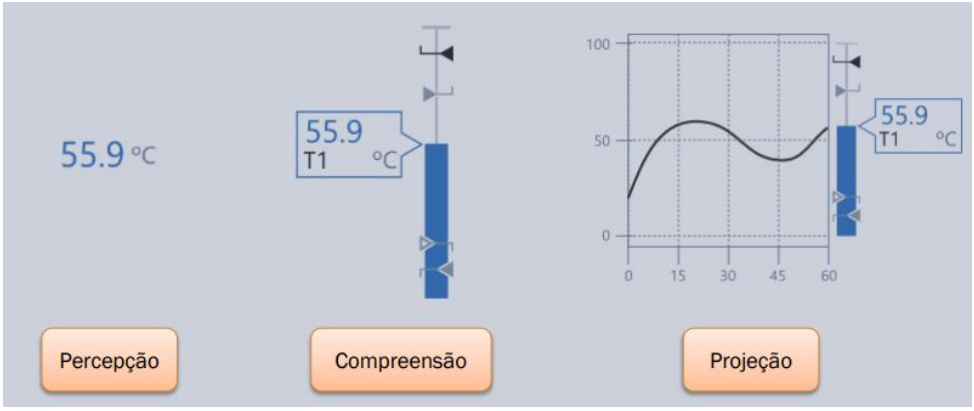
2.1.3.1 IHM's de Alta Performance Visual

A ISA (*International Society of Automation*), atualmente uma das principais instituições de automação do mundo, elaborou um documento de padronização das interfaces homem-máquina a ISA 101. Tal documento é um conjunto de boas práticas no desenvolvimento de telas para IHMs e sistemas supervisórios que visa aumentar a consciência situacional do operador (ISA, 2020).

Para Goetz (2019), o dado bruto (Percepção) muitas vezes não nos dá a condição de saber se determinando valor está dentro da normalidade, para que isso aconteça ele deve ser inserido num contexto com os limites de operação daquela variável (Compreensão), no entanto esse contexto de limite não indica a tendência de variação desse valor, isso só a possível com a projeção no tempo através de gráficos históricos (Projeção). A Figura 5 ilustra uma mesma variável em diferentes níveis de situação.

A ISA 101 tem como objetivo melhorar as interfaces industriais para que o

Figura 5 – Consciência situacional

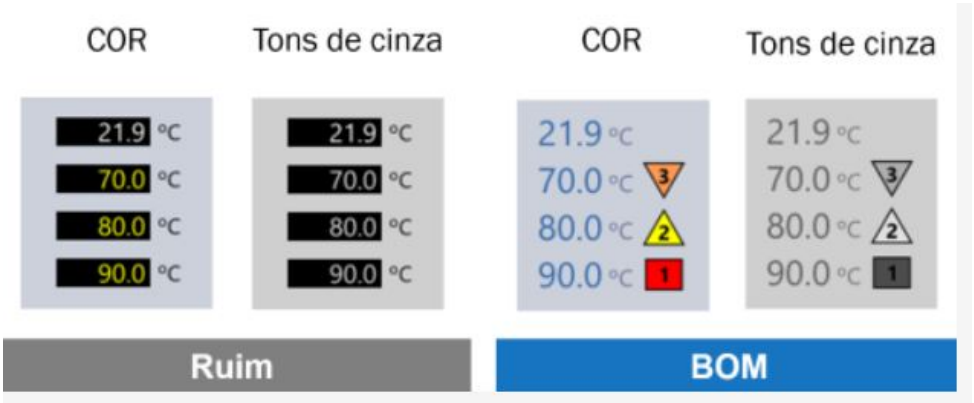


Fonte: ISA (2020)

usuário tenha total entendimento do processo e percepção de falhas e alarmes no processo ou na planta. Sendo assim a interface deve ter cores que não distraiam o operador enquanto existe um alarme sendo sinalizado, excesso de cores e objetos tiram a eficiência informativa da interface, diminuindo sua usabilidade (GOETZ, 2019).

Em relação a cores a ISA 101 também trata da acessibilidade, pois segundo a ISA (2020) o daltonismo, distúrbio visual que impede a diferenciação de algumas cores, atinge cerca de 8% dos homens e 0.5% das mulheres. As principais cores utilizadas para alertar o status de falha, funcionamento e níveis de criticidade de equipamentos são vermelho, amarelo e verde, cores que são afetadas por pessoas com daltonismo, sendo assim além de utilizar apenas cores para níveis de criticidade, deve-se utilizar formas como ilustrado na Figura 6.

Figura 6 – Acessibilidade ISA 101



Fonte: ISA (2020)

3 MATERIAIS E MÉTODO

Neste capítulo será apresentado os *softwares* e a metodologia de desenvolvimento necessária para a elaboração um sistema supervisório base para se utilizar em um ambiente industrial. Primeiramente será feita a descrição dos *softwares* e a configuração dos mesmos para que a aplicação seja funcional e, na sequência, será descrito as metodologias de desenvolvimento da aplicação, linguagem de programação utilizada a arquitetura de uma aplicação supervisória ou da sigla em inglês, aplicação SCADA (*Superviosry Control And Data Acquisition*).

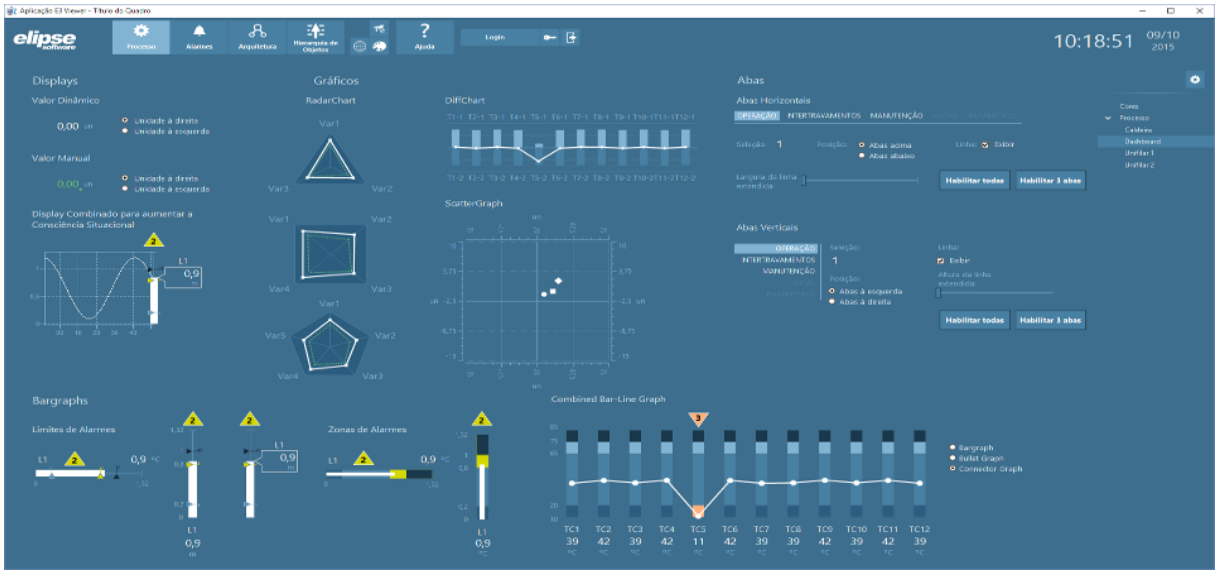
3.1 *SOFTWARES*

Para o desenvolvimento de um sistema supervisório completo é preciso *hardwares* e *softwares* específicos para desempenhar todas as tarefas que um sistema supervisório realiza, desde a coleta de sinais até a visualização desses em interfaces gráficas dentro de uma aplicação SCADA.

3.1.1 Elipse E3

Existem diversos *softwares* no mercado para o desenvolvimento de aplicações supervisórias, no entanto eles costumam ser exclusivamente pagos, ou necessitam da conexão com *hardware* proprietário para o desenvolvimento da aplicação. Como objetivo do trabalho é desenvolver uma aplicação generalista e servir de material de apoio sobre o assunto, será utilizado o *Elipse E3 Studio 6.1.98*, um dos principais *softwares* de desenvolvimento de sistemas supervisórios do mercado, na sua versão gratuita e que possibilita o desenvolvimento de uma aplicação supervisória com até 20 variáveis.

Figura 7 – Aplicação de demonstração Elipse E3



Fonte: Elipse Software (2015)

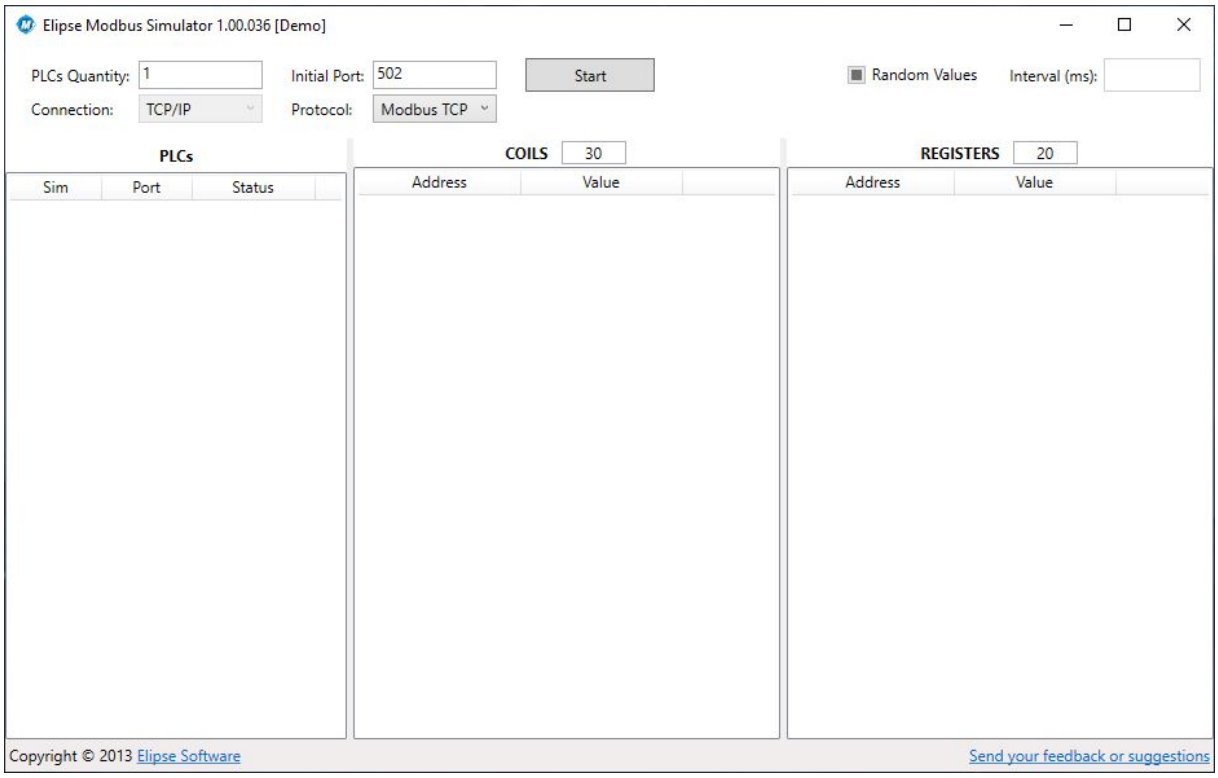
Na Figura 7 é apresentada uma aplicação de demonstração, essa aplicação utiliza conceitos da ISA 101, apresentados na seção 2.1.3.1, esses conceitos serão demonstrados a seguir na metodologia de desenvolvimento.

Os dados que são coletados em um sistema supervisório são armazenados em banco de dados ou lotes de arquivos. O Elipse possui a integração com dois dos principais gerenciadores de banco de dados do mercado, o SQL Server e o Oracle. O *software* também possibilita a gravação dos dados em arquivos do Microsoft Access (extensão MDB). É possível utilizar de forma gratuita o banco de dados neste formato, para arquivos com até 100 *Megabytes* de dados, sendo assim o presente trabalho fará o uso deste método como banco de dados.

3.1.2 Modbus Simulator

Para realizar a simulação de um *hardware* de coleta e processamento de sinais será utilizado um *software* gratuito, o *Elipse Modbus Simulator 1.00.036* (Figura 8), simulando um produtor Modbus, assim como um CLP industrial.

Figura 8 – *Software* para simulação de dispositivo produtor Modubs TCP/IP



Fonte: AUTOR

Nele podem ser simulados 20 valores decimais (*registers*) de 0 à 65535, esse limite se dá pela representação de 16 bits estabelecidas no *software*. Nele também podem ser simulados 30 valores binários para *coils*.

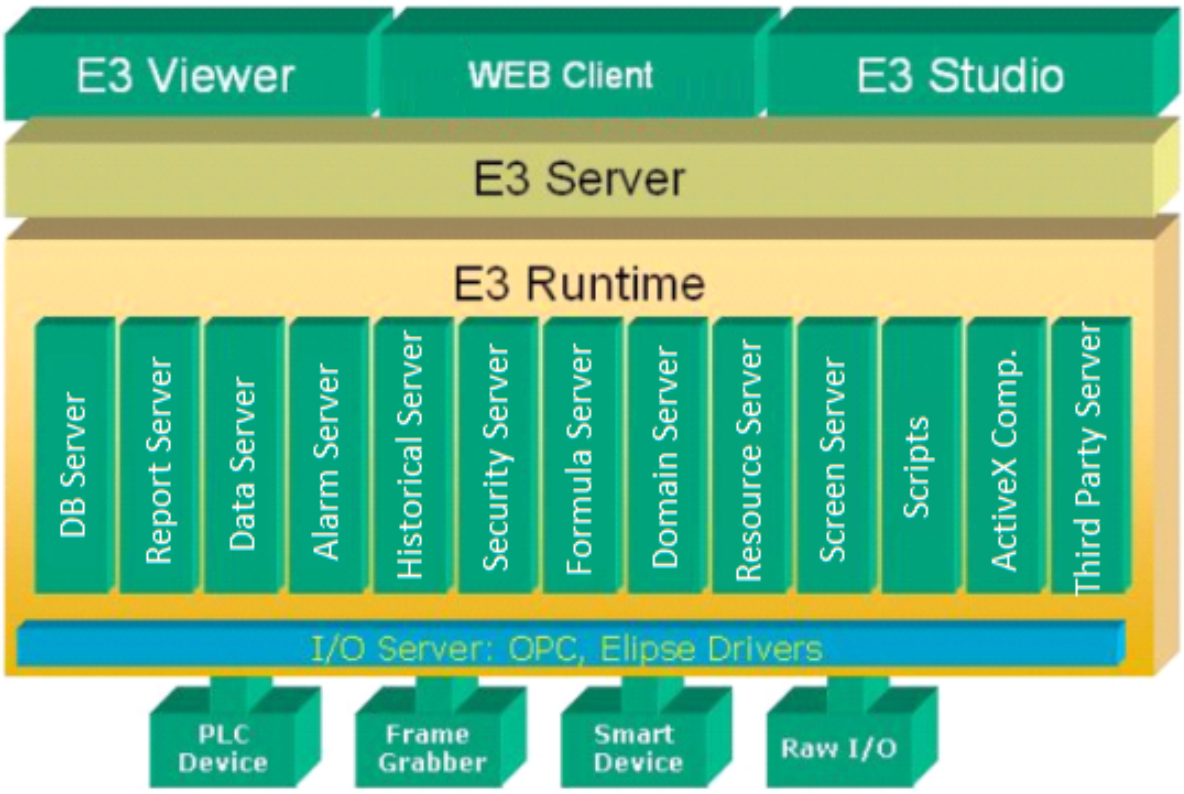
3.2 METODOLOGIAS DE DESENVOLVIMENTO

O Elipse E3 possui uma estrutura como ilustrado na Figura 9, essa estrutura é a base onde são executadas as aplicações, formando um *framework* (ELIPSE SOFTWARE, 2015). No topo do *framework* estão os 3 principais aplicativos do E3:

- **E3 Server**: é onde a aplicação está sendo executada, é realizada o envio e recebimento de informações para clientes remotos e também é o aplicativo que gerencia o processamento dos dados de entrada e saída.
- **E3 Viewer**: é o aplicativo de visualização e comando. É esse aplicativo que se conecta a aplicação que está rodando no E3 Server. É possível essa conexão ser remota através de uma rede interna ou via web pelo E3 Web Client.
- **E3 Studio**: *software* onde é feita toda a configuração e desenvolvimento da aplicação supervisória, desde a interface de comunicação até a interface de visualização gráfica, e que será abordado neste trabalho com mais detalhes.

Na base do *framework* encontram-se as fonte de dados externos, que podem ser variadas como CLP's, dispositivos inteligentes, dados brutos entre outros. A integração dessas diversas fontes é possível e ocorre em tempo de execução através

Figura 9 – Estrutura do aplicativo Elipse E3



Fonte: Elipse Software (2015)

do I/O Server, onde ficam os Drivers de comunicação ou servidores de comunicação, como OPC Server. Essa camada faz a interface entre o dispositivo e o software de desenvolvimento. A Elipse E3 disponibiliza inúmeros drivers, de diferentes protocolos de comunicação e dispositivos. No presente trabalho será utilizado o Driver Modicon Modbus Master para fazer a interface de comunicação do simulador de dispositivo modbus, o Modbus Simulator, com o Elipse E3.

Na Figura 9 é mostrada a estrutura que uma aplicação supervisória desenvolvida no Elipse E3 pode ter:

- DB Server - Servidor de Banco de Dados: onde são criadas as conexões com os banco de dados previamente existentes, como SQL, ou arquivo de dados Access com extensão .MDB;
- Report Server - Servidor de Relatórios: ferramenta que possibilita gerar relatórios com base nos dados atuais, ou a partir do banco de dados. Esses relatórios podem ser personalizados e exportados para diferentes tipos de arquivos (pdf, csv, xmls);
- Data Server - Servidor de Dados: onde se concentra os dados da aplicação. Esses dados podem ser externos, adquiridos por interface de comunicação, ou dados internos, quando a própria aplicação realiza o processamento do dado;

- Alarm Server - Servidor de Alarmes: estrutura onde é centralizado todos os alarmes configurados da aplicação, apenas um servidor de alarme pode ser criado por aplicação. Nele são definidos o banco de dados que serão salvo os alarmes, as informações que serão salvas sobre os alarmes e a definição de variáveis de usuário para os alarmes.
- Historical Server - Servidor de Históricos: onde se encontram os históricos da aplicação, eles funcionam em conjunto com o servidor de banco de dados e o servidor de dados, é onde são configurados quais dados e a frequência que serão armazenados;
- Resources Server - Servidor de Recursos: neste servidor podem ser adicionados mídias com extensões como JPG, MP3, GIF e outras que poderão ser adicionadas na aplicação;
- Screen Server - Servidor de Telas: toda a parte de visualização para o usuário se encontra nesse servidor;
- Scripts - trechos de códigos que são executados através de disparos (*triggers*),possibilitando a aplicação realizar tarefas complexas e automáticas,a linguagem de programação que o E3 utiliza para os scripts é a Visual Basic(VB).

3.2.1 Orientação à objetos no Elipse

O E3 possibilita o a criação de objetos específicos XFolders, XObjects e os XControls. Esse recurso é uma representação de orientação à objetos utilizado no desenvolvimento de aplicações supervisórias no Elipse. A criação desses objetos dentro de uma biblioteca da aplicação trás as vantagens como:

- Reutilização de Objetos: tendo em vista que um supervisório representa equipamentos muitas vezes iguais, como motores, bombas e reservatórios, o desenvolvimento de uma classe com as propriedades e funções já definidas, pode ser replicada para toda a aplicação, bastando apenas instanciá-la;
- Manutenção da Aplicação: muitas vezes tem-se a necessidade de alterar objetos dentro de um supervisório, em casos de *retrofitting* ou troca de equipamentos, com uma classe definida, a alteração na classe principal já se aplica a todos os objetos instanciados na aplicação;
- Padronização e organização: com uma classe bem definida e documentada, o nível de entendimento da aplicação é facilitado para usuário que venham a utilzá-la, seja para desenvolvimento ou aprendizado.

Cada um dos três tipos de objetos que podem ser usados como base para as classes da biblioteca possuem suas definições.

O XFolder é um tipo de objeto *Container*, tem a funcionalidade de guardar outros objeto dentro dele, porém pode ter suas propriedades específicas. Um exemplo

para um XFolder seria uma classe que representa a linha de produção. Uma aplicação supervisória pode englobar uma planta industrial inteira, onde nela existem várias linhas de produção com propriedades e características em comuns. Sendo um *Container* a classe linha de produção pode conter outros objetos de diferentes classes, por exemplo um XControl que representa um motor, ou objetos nativos do Elipse como será mostrado no próximo capítulo.

O XObject pode ser definido como um objeto de dado que pode ou não estar associado a objetos gráficos dentro da aplicação supervisória. Quando criada uma classe do tipo XObject, podem ser atribuídas propriedades que podem ser nativas do Elipse, como Inteiro, Booleano, ou outros XObjects ou XFolders.

Para a criação de classes que poderão ser visualizadas na aplicação, ou seja, estarão no servidor de telas, existe o XControl, esse tipo de classe, assim como as anteriores, permite criar suas propriedades e também métodos, através de funções internas em scripts, que será abordado à frente.

3.2.2 Definições da aplicação

O objetivo de um sistema supervisório é monitorar e controlar variáveis de processos, assim como registrar os valores históricos através de gráficos de linha ou relatórios. A aplicação desenvolvida no presente trabalho simulará as condições de um processo industrial, com variáveis inseridas em um contexto de chão de fábrica obtidas através de um dispositivo com interface de comunicação ModBus. As práticas demonstradas poderão ser aplicadas em outros contextos de supervisão e controle de processos, uma vez que os elementos desenvolvidos poderão ser adaptados conforme a aplicação.

Tabela 1 – Memória *Coil* simulada

Endereço Coil	**Descrição**	**Tipo**	**Função**
1	Emergência	Bool	Entrada digital
2	Seletora automático		
3	Seletora Manual		
4	Sensor 1		
5	Sensor 2		
6	Sinalização Emergência	Bool	Saída digital
7	Sinalização Automático		
8	Sinalização. Manual		
9	Liga motor 1		
10	Liga motor 2		

Fonte: Autor (2022).

A descrição, tipo e endereço das memórias do dispositivo simulado pelo software, são apresentadas nas tabela 1 e 2. O objetivo da aplicação é ter uma visualização e controle de um processo industrial, com as informações que serão

Tabela 2 – Memória *Register* simulada

Endereço Register	Descrição	Tipo	Função
1	Palavra de alarmes	Word	Memória Interna
2	Palavra do Supervisório		
3	Nível reservatório	Word	Entrada analógica
4	Temperatura		
5	Corrente motor 1		
6	Corrente motor 2		
7	Tensão Motor 1		
8	Tensão Motor 2		
9	Válvula proporcional 1	Word	Saída analógica
10	Válvula proporcional 2		

Fonte: Autor (2022).

enviadas a partir do simulador e com as informações que o sistema supervisório irá enviar para o simulador. Portanto serão simulados tanto o fluxo de informações das entradas para o supervisório quanto atuações do supervisório nas saídas simuladas.

As entradas digitais simuladas representam sinais enviados do campo para o dispositivo, portanto são apenas de leitura, assim como as entradas analógicas. Por outro lado as saídas digitais são sinais que o dispositivo envia para o campo. Esses sinais podem ser enviados através de atuações de lógicas internas do dispositivo ou pela escrita nesses endereços de memória através do supervisório, por exemplo.

A memória interna do dispositivo representa as lógicas que podem ser feitas dentro da programação de um CLP por exemplo, uma determinada condição irá atuar um bit na palavra de alarme. Como cada registrador possui 16 bits, a palavra de alarme representa 16 possíveis alarmes, assim como a palavra do supervisório representa 16 bits que podem servir para atuar em alguma condição do dispositivo.

4 DESENVOLVIMENTO DO PROJETO

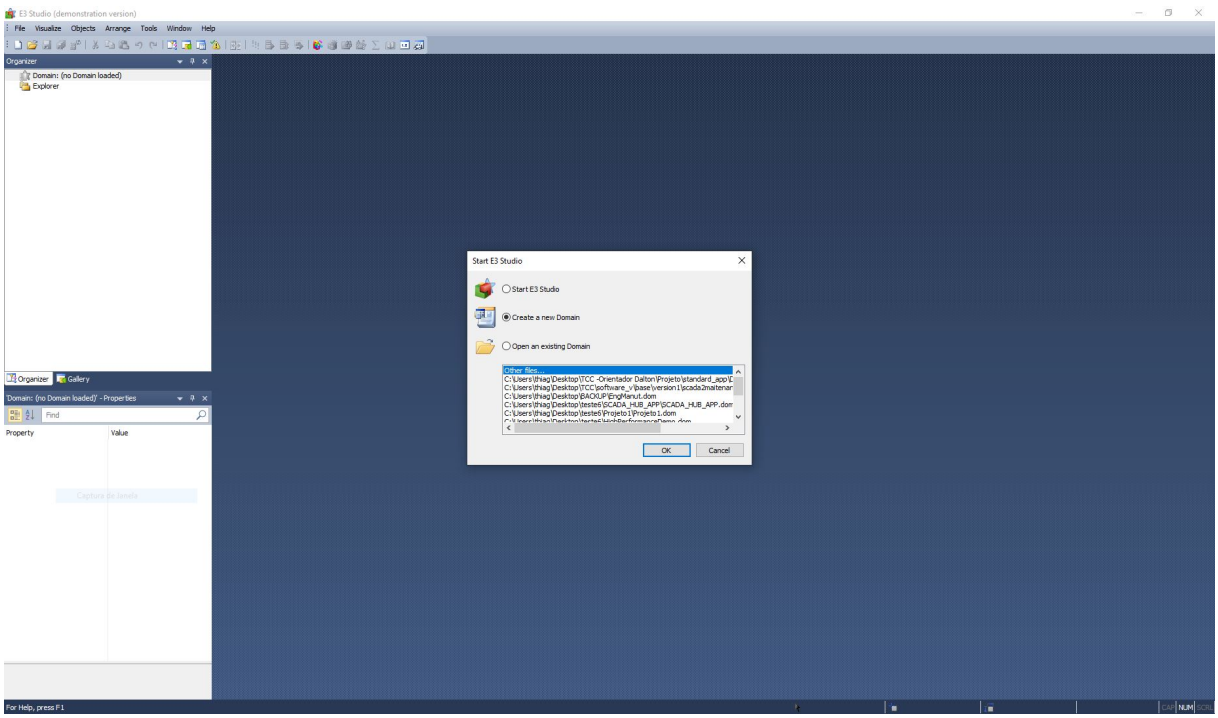
Neste capítulo será descrito os passos para o desenvolvimento do sistema supervisório, utilizando as abordagens e ferramentas apresentadas no capítulo de metodologia e métodos, portanto os passos descritos a seguir poderão ser seguidos no desenvolvimento de outras aplicações desde que o software utilizado seja o Elipse E3.

Será apresentado a seguir, em forma de documentação, desde a configuração para a aquisição de dados, estruturação para armazenamento desses dados, elaboração das interfaces gráficas de alta performance para os usuários e implementação de funcionalidades para sistemas supervisórios.

4.1 INICIALIZAÇÃO PROJETO SUPERVISÓRIO

O Elipse E3 possui uma inicialização amigável para novos usuários ao criar uma base para um sistema supervisório ao iniciar um projeto novo. Os principais servidores descritos no capítulo anterior são criados logo na inicialização de um novo projeto:

Figura 10 – Criação novo projeto



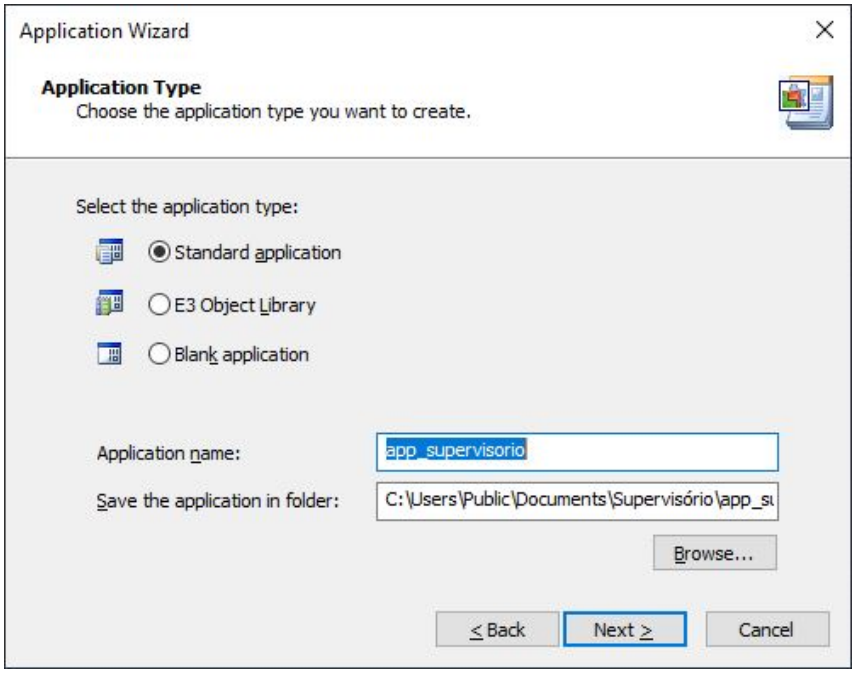
Fonte: Autor (2022)

Na Figura 10 é mostrada a tela inicial ao abrir o *software*, existe a possibilidade de carregar projetos, ou criar um novo **Domínio**(Figura 12), nome dado ao arquivo

executável (*.DOM*) que conta com os projetos (*.PRJ*) e bibliotecas (*.LIB*) relacionados à aplicação. Um domínio pode conter diversos projetos e bibliotecas.

Ao selecionar a criação de um novo projeto é aberto um assistente, como é apresentado na Figura 11,uma aplicação padrão possui os servidores de telas (Figura 13), comunicação (Figura 14), banco de dados (Figura 15) e alarmes(Figura 16), em branco.

Figura 11 – Assistente criação novo projeto



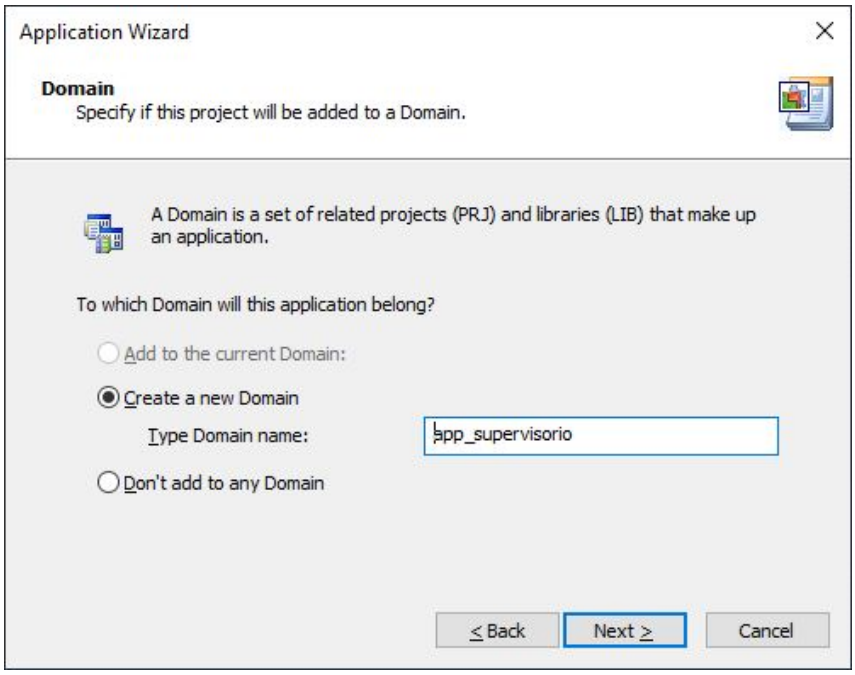
Fonte: Autor (2022)

Os arquivos de uma aplicação supervisória são os com extensão PRJ, que possuem toda a configuração de dados, telas, comunicação e alarmes, com extensão LIB, e é responsável por armazenar os recursos das classes definidas pelo desenvolvedor da aplicação (XFolders, XObjects e XControls) e com extensão DOM, que é responsável pelo conjunto de arquivos PRJ e LIB que a aplicação possui e é criado na tela apresentada na Figura 12.

A configuração de tela de uma aplicação deve ser feita levando em consideração a resolução da tela em que a aplicação rodará e não a tela em que está sendo desenvolvida, pois pode haver distorção de objetos gráficos. Por se tratar de algo difícil de prever a resolução em que será executada a aplicação, pois uma das características de um sistema de supervisão é o acesso remoto, é indicado a utilização de escalas padrões de resolução como 16:9 ou 4:3. Para essa aplicação usaremos a resolução de 1920x1080, escala 16:9, como mostrado na Figura 13.

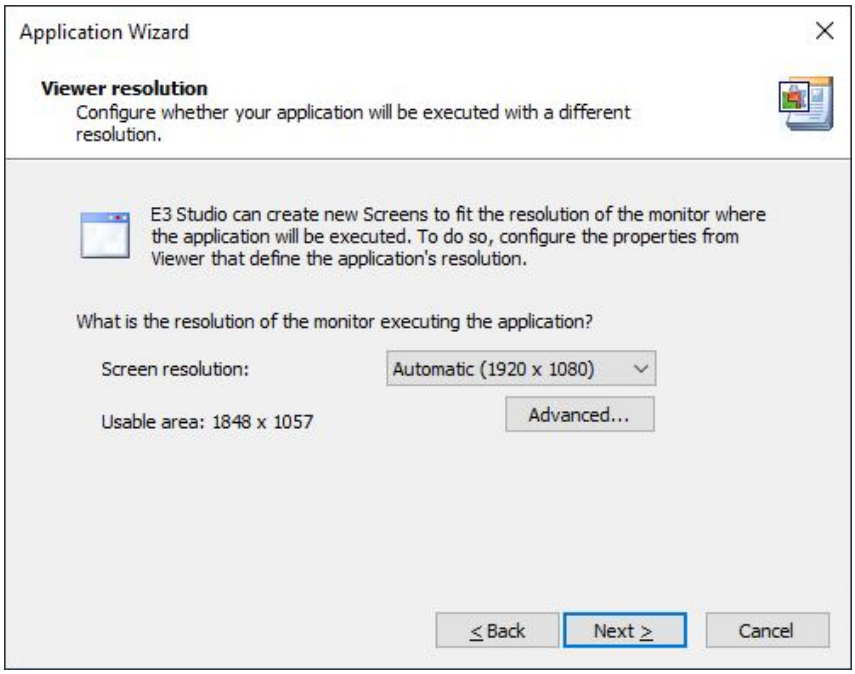
Na inicialização de um novo projeto é possível adicionar um driver de comunicação para que ele seja instanciado. Todos os drivers de comunicação da

Figura 12 – Assistente criação novo domínio



Fonte: Autor (2022)

Figura 13 – Assistente criação servidor de telas

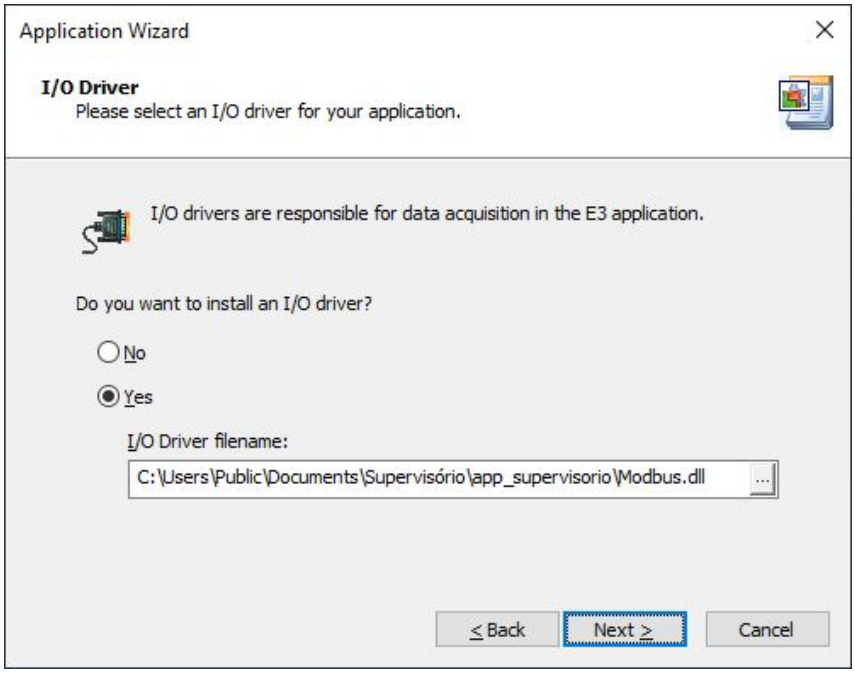


Fonte: Autor (2022)

Elipse se encontram disponíveis para download em seu site. Nessa aplicação como citado anteriormente, será feita a simulação de um CLP em campo, comunicando via MODBUS, logo utilizaremos o driver *Modicon Modbus Master (ASC/RTU/TCP)* da Elipse. Para essa configuração inicial do servidor de comunicação basta apontar para o caminho arquivo de driver baixado da Elipse nessa etapa de configuração como é

apresentado na Figura 14.

Figura 14 – Criação Servidor de Comunicação



Fonte: Autor (2022)

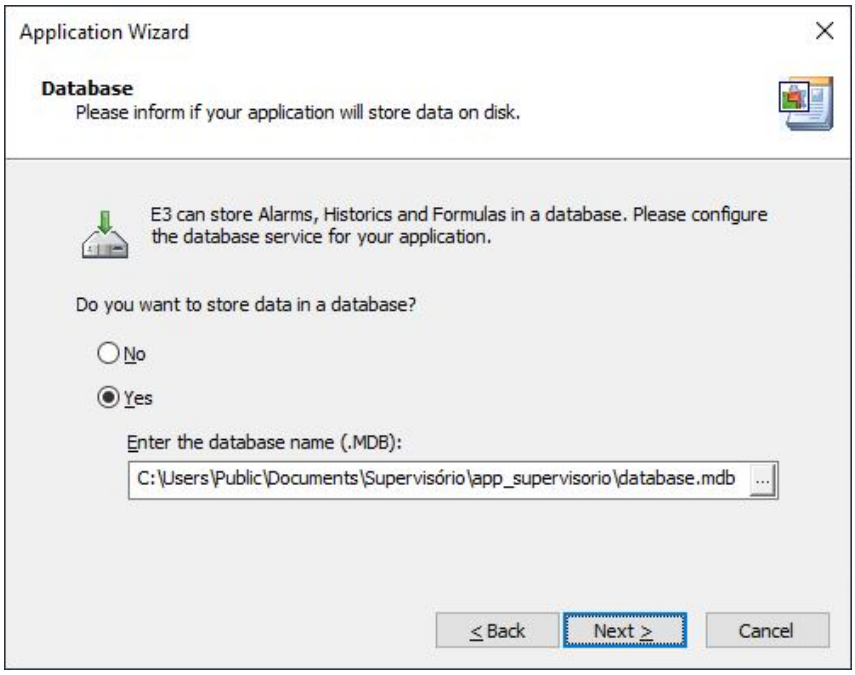
O servidor de banco de dados da aplicação será um arquivo com extensão MDB, o mesmo pode ser criado na mesma pasta em que o projeto principal como é mostrado na Figura 15. Vale ressaltar que o limite de dados em um arquivo MDB não pode ultrapassar 100MB.

A criação do servidor de alarmes conta com a opção de guardar os alarmes no banco de dados criado anteriormente, com a escolha dessa opção, é criado uma tabela com as informações pré configuradas do servidor de alarmes que serão salvas no banco de dados, conforme mostrado na Figura 16.

Todos os servidores criados nesta etapa são os básicos para que um sistema supervisório desempenhe sua função mínima e podem ser alterados durante o desenvolvimento do projeto, que será demonstrado nas próximas seções.

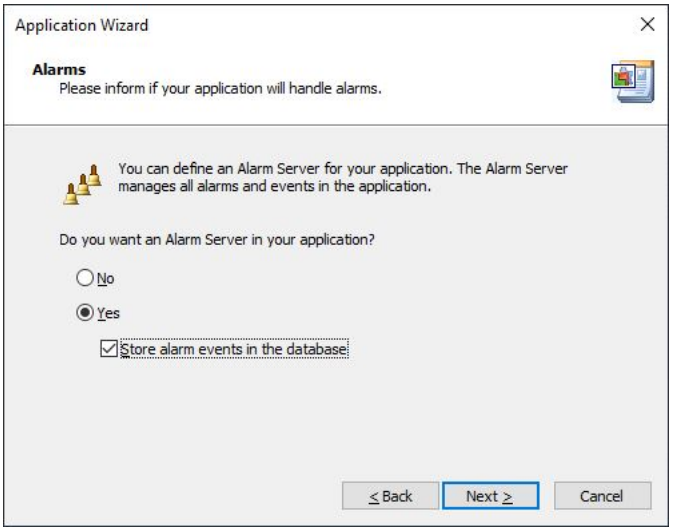
Após a inicialização do novo projeto, a janela para o desenvolvimento da aplicação fica disponível. Na Figura 17 é mostrado o *Organizer*, à esquerda da tela, e nela é apresentado todos os recursos de um sistema supervisório.

Figura 15 – Criação Servidor de Banco de Dados



Fonte: Autor (2022)

Figura 16 – Criação Servidor de Alarmes



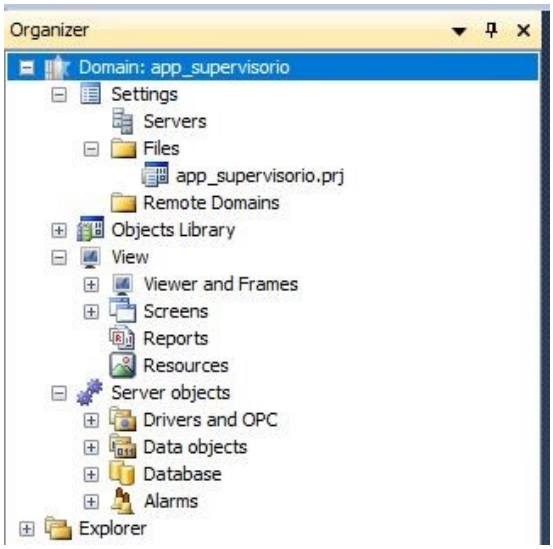
Fonte: Autor (2022)

4.2 ESTABELECER COMUNICAÇÃO MODBUS

O roteiro de desenvolvimento apresentado a seguir leva em conta as boas práticas de desenvolvimento visando facilitar o entendimento e evolução do projeto.

Para iniciar o desenvolvimento da aplicação apresentada neste trabalho, será estabelecida a comunicação do software que irá simular um dispositivo MODBUS, citado no capítulo anterior, com o software de desenvolvimento do sistema supervisório e posteriormente com a aplicação supervisória desenvolvida.

Figura 17 – Organizer Estrutura da Aplicação



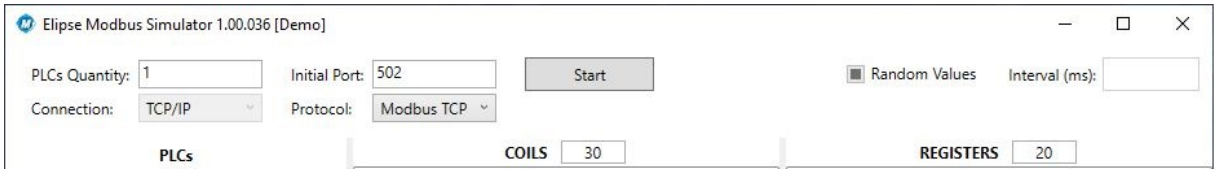
Fonte: Autor (2022)

4.2.1 Configurando driver Modbus no Elipse E3

O software de simulação possibilita a implementação do protocolo MODBUS/TCP e MODBUS/RTU. Nesta aplicação será utilizado o protocolo TCP/IP, além de ser o mais utilizado no mercado, também é possível aplicá-lo com o driver gratuito da Elipse.

Na Figura 18 é mostrada a configuração inicial do *software* que irá simular um dispositivo MODBUS. As configurações utilizadas nesse projeto serão as padrões para a comunicação MODBUS, onde a quantidade de PLC é 1, a porta inicial utilizada é a padrão dos dispositivos MODBUS, a 502, e o protocolo o Modbus TCP.

Figura 18 – Configuração Modbus Simulator

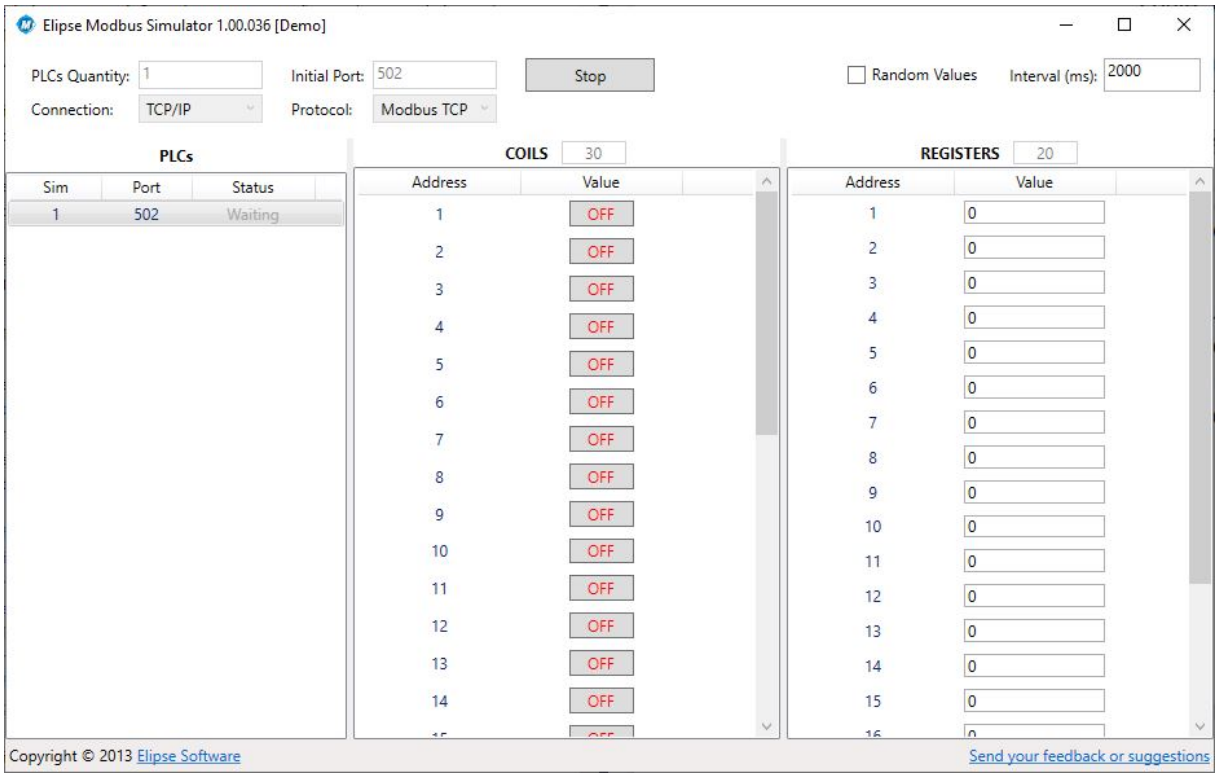


Fonte: Autor (2022)

Ao iniciar clicar em *"Start"*, o simulador está pronto para utilização. Como é mostrado na Figura 19 os valores em suas memórias podem ser editados, ou receber valores aleatórios, caso a caixa de seleção *Random Values* estiver selecionada, esses valores randômicos são alterados conforme o intervalo de tempo em milissegundos, configurado por padrão em 2000ms.

A integração com o sistema supervisório é feita através da configuração do driver dentro do Elipse E3. Na inicialização do projeto feito na seção 4.1, um driver padrão já é adicionado na aplicação, a seguir será apresentado como configurar o

Figura 19 – Modbus Simulator iniciado

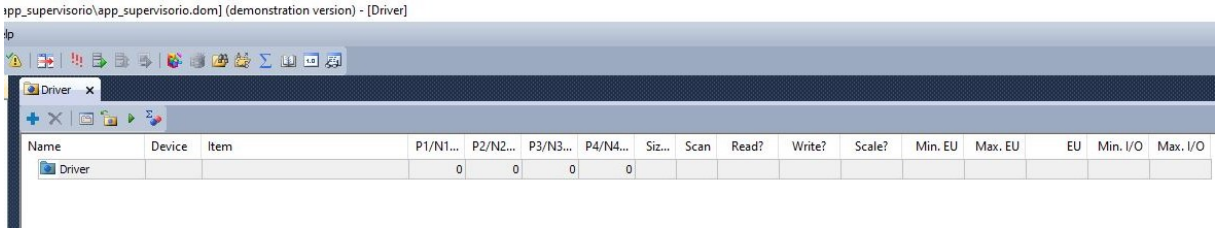


Fonte: Autor (2022)

driver para se comunicar com o software simulador.

A Figura 20 apresenta a tela do driver de comunicação criado na inicialização do projeto vinculado com a DLL do MODBUS e os parâmetros de configuração das tags que serão adicionadas, chamados de parâmetros P/N.

Figura 20 – Configurando Driver Modbus

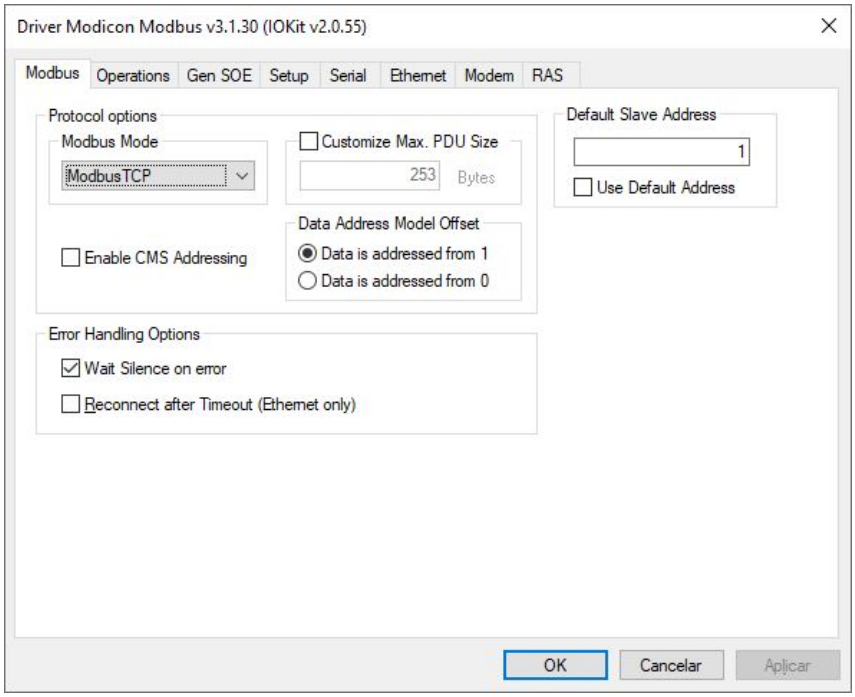


Fonte: Autor (2022)

Na janela de configuração do driver (Figura 21) é realizada a configuração para que o supervisório se conecte com o dispositivo MODBUS, neste projeto o simulador.

Na aba ModBus, a configuração de Protocolo deve ser a mesma configuração em que o simulador foi inciado, *ModBus TCP*.

Figura 21 – Janela de configuração Driver Modbus aba *ModBus*



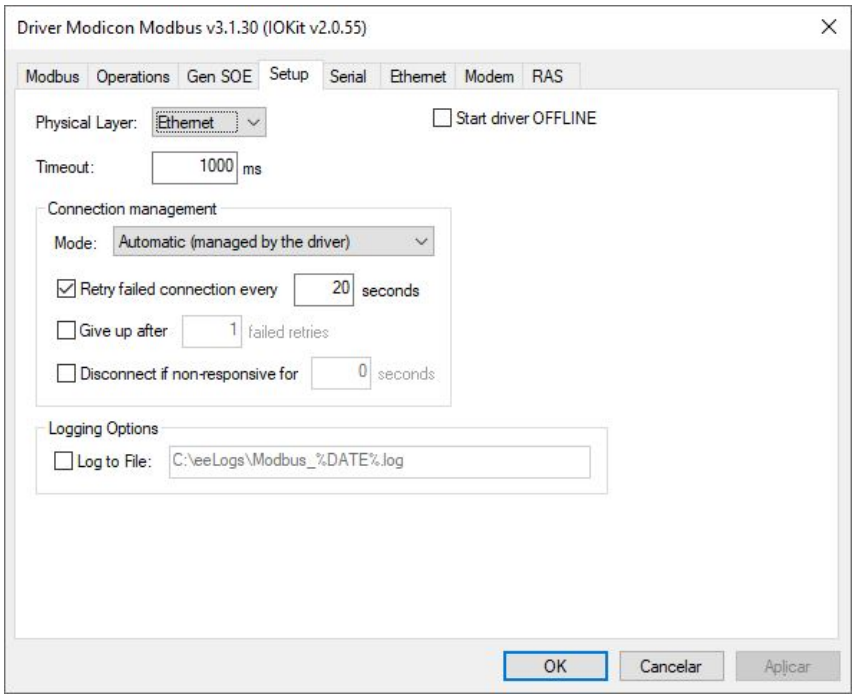
Fonte: Autor (2022)

Para configurar qual meio físico que será utilizado para fazer a interface entre o dispositivo e o supervisório a aba *Setup* possui uma lista, mostrada na Figura 22. Para essa aplicação será utilizado a rede Ethernet, pois o supervisório irá acessar o simulador localmente pela rede da máquina local que estará executando ambos os *softwares*, tanto simulador quanto a aplicação supervisória. Nesta aba também é possível configurar o tempo para desconexão da interface ,*Timeout*, e parâmetros para tentativas de reconexão, caso a mesma seja interrompida.

O meio físico precisa ser parametrizado com as informações do dispositivo que irá ser lido, na aba *Ethernet* (Figura 23, as configurações de protocolo de transferência de dados, endereço do dispositivo e portas devem ser preenchidas para que seja estabelecida a comunicação de maneira correta entre supervisório e dispositivo. As configurações neste projeto serão:

- **Protocolo de transferência:** TCP/IP
- **Endereço:** 127.0.0.1. Este endereço aponta para o próprio computador que está executando a aplicação, pois o simulador do dispositivo também será executado no computador da aplicação.

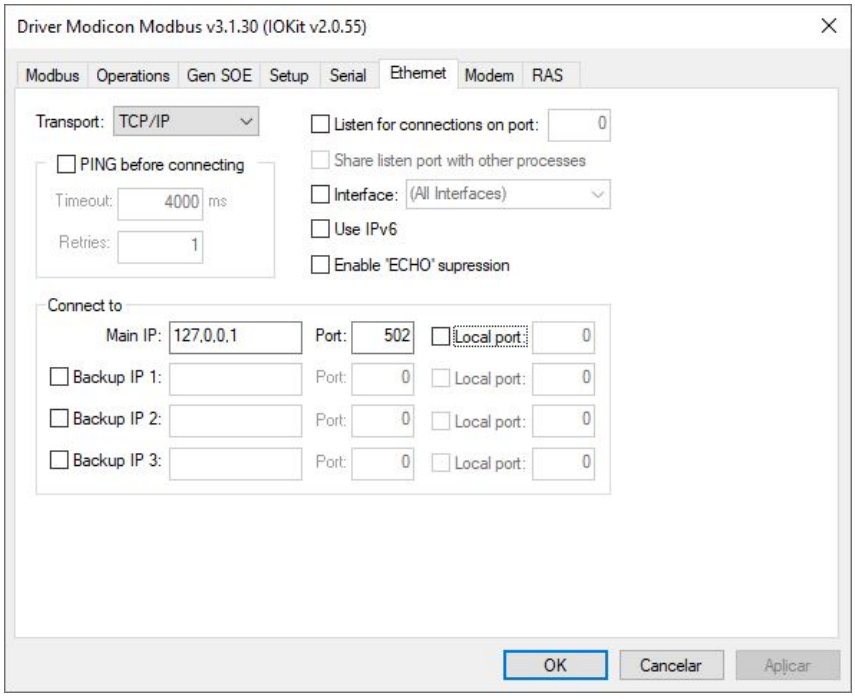
Figura 22 – Janela de configuração Driver Modbus aba *Setup*



Fonte: Autor (2022)

- **Porta:** 502. Porta padrão para os dispositivos ModBus.

Figura 23 – Janela parametrização meio físico aba *Ethernet*

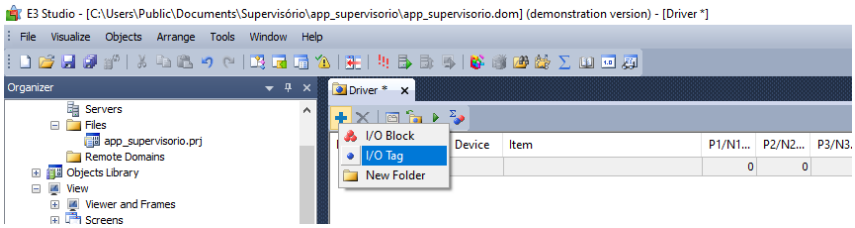


Fonte: Autor (2022)

4.2.2 Lendo informações do dispositivo Modbus

Ao finalizar as configurações do driver mostradas na seção anterior, o Elipse E3 já está habilitado a se conectar com o dispositivo. A comunicação Modbus funciona do modo produtor e consumidor, é preciso que o consumidor faça requisições de qual informação o produtor irá enviar. As informações que serão lidas pelo supervisório são as memórias simuladas do dispositivo ModBus. Dentro do Elipse é necessário adicionar as memórias que serão lidas do dispositivo, as chamadas *Tag's*, conforme mostrado na Figura 24.

Figura 24 – Adição de *Tag* no driver de comunicação



Fonte: Autor (2022)

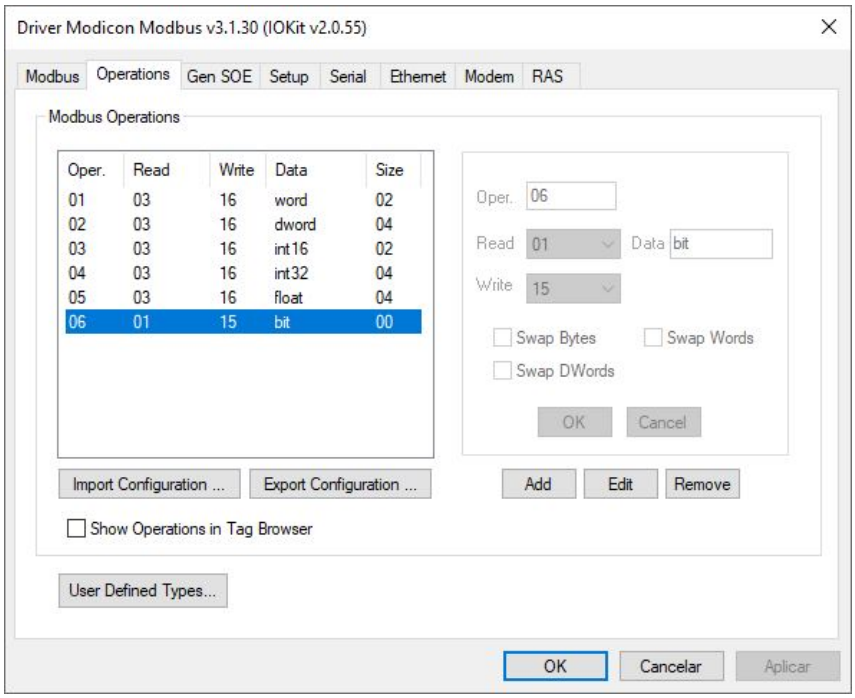
Cada memória no dispositivo ModBus é representado por uma *tag* no driver do supervisório. Para otimizar as requisições do consumidor para o produtor, é aconselhado usar blocos de *tag's*, caso os endereços no dispositivo sejam sequenciais, com o tamanho de memória a ser requisitada para o produtor, número de *tag's*. A configuração dos parâmetros P/N das *Tag's* indicam qual o dispositivo na rede sera lido a tag (P1/N1), qual o numero da operação conforme a configuração do driver (P2/N2), se o dispositivo usa alguma máscara de bit (P3/N3) e qual o endereço da memória no dispositivo (P4/N4).

O parâmetro P2/N2 se refere a operação que será enviada para o dispositivo produtor, com essa informação é interpretado o tipo do dado a ser enviado ou recebido. A aba *Operations* da janela de configuração do driver mostra as operações e os tipos de dados que cada uma é relacionada (Figura 25). Como o simulador oferece dados de 16 bits nos *registers* e 1 bit nos *coils*, as operações para esse projeto serão 1 e 6 respectivamente.

Na Figura 26 é apresentado um exemplo para a receber do dispositivo 1 da rede (P1/N1), uma informação do tipo word (16 bits), ou seja, um *register*(P2/N2), e no primeiro endereço de memória do dispositivo (P4/N4).

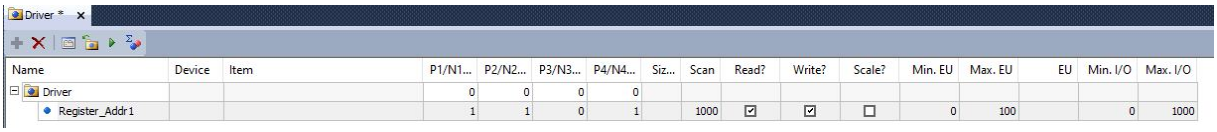
Para leitura de blocos de memórias a deve ser utilizar da configuração conforme a Figura 27, que realiza a leitura dos 10 primeiros endereços de memória de 1 bit (*coil*) do dispositivo. Os elementos são criados automaticamente ao adicionar o tamanho do

Figura 25 – Operações padrões Modbus



Fonte: Autor (2022)

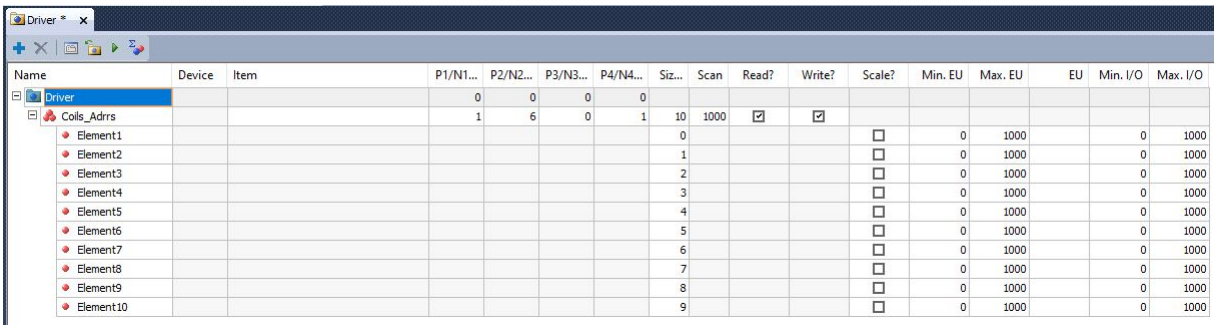
Figura 26 – Exemplo de *tag* individual configurada no driver ModBus



Fonte: Autor (2022)

bloco e representam cada *tag* individualmente.

Figura 27 – Exemplo de bloco de *tag's* configurado no driver ModBus



Fonte: Autor (2022)

A propriedade *Scan* da *tag* se refere a velocidade em que o consumidor requisita àquela *tag* ao produtor, por padrão utiliza-se 1000ms, quanto menor o valor mais rápido a comunicação entre supervisório e dispositivo.

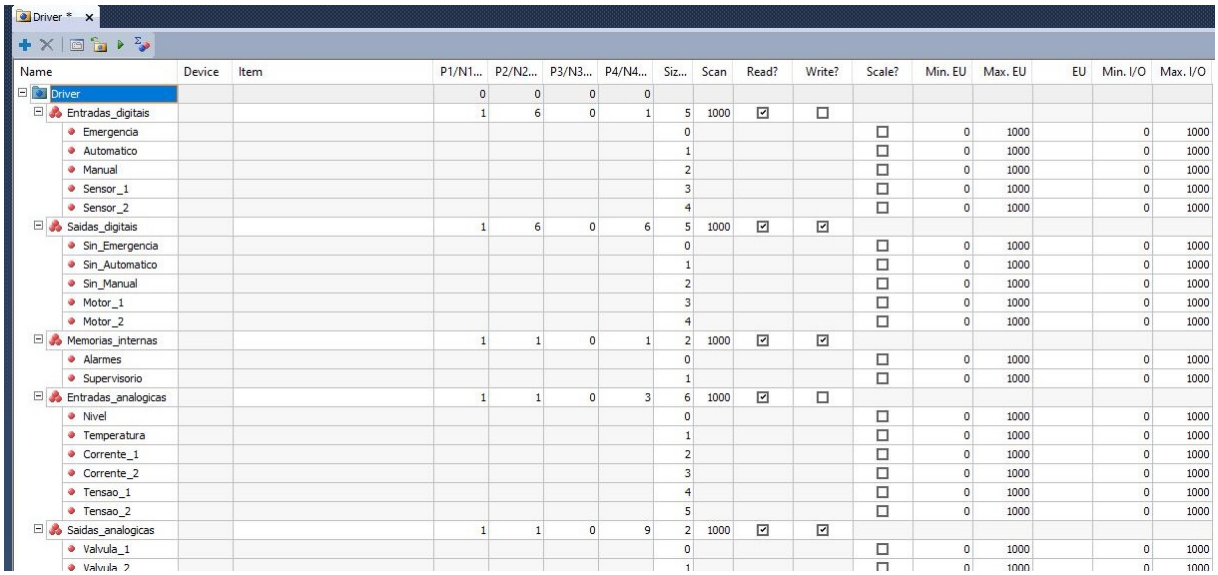
Read e *Write* são as propriedades de leitura e escrita da *tag*, caso o

supervisório não tenha permissão para escrever na *tag*, deve se desabilitar a seleção da propriedade *Write*.

Por fim, é possível utilizar uma escala para a *tag*, pois muitos dispositivos dão valores brutos de corrente (4-20mA) ou tensão(0-10V) e é necessário a conversão para a unidade que a corrente ou tensão está representando. Para isso deve se configurar qual o valor mínimo e máximo real que o dispositivo envia nos parâmetros *Min. I/O* e *Max. I/O* e os valores mínimo e máximo da variável aferida nos campos *Min. EU* e *Max. EU*.

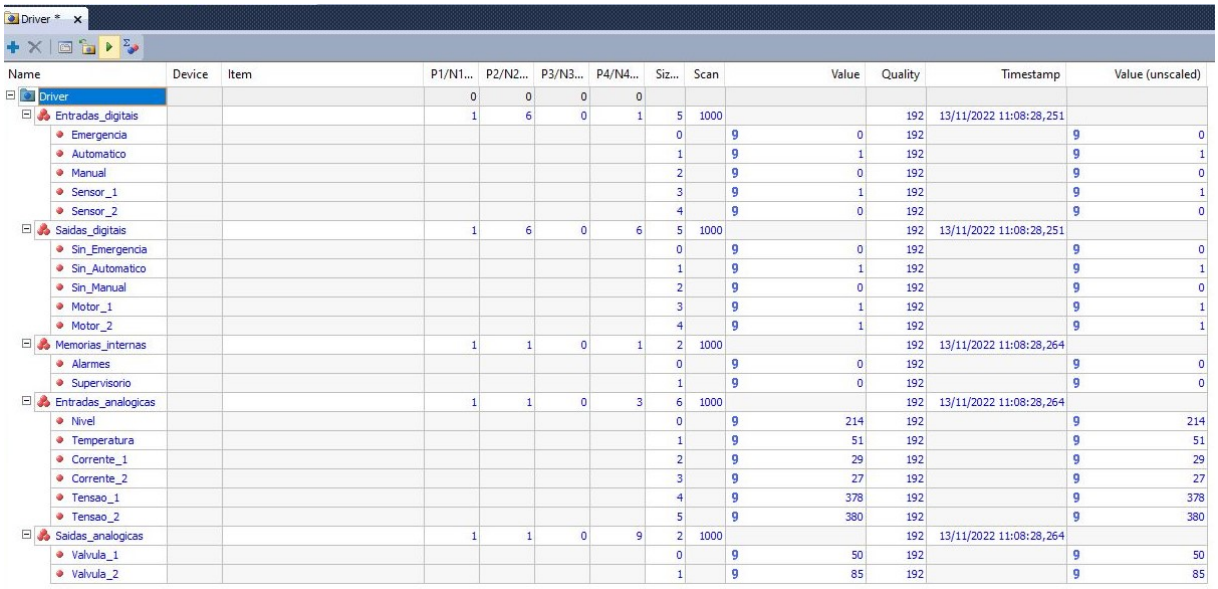
Na Figura 28 é mostrado o driver configurado para o projeto em questão. Com as memórias das Tabelas 1 e 2 mapeadas pelo driver do supervisório. Com o simulador ModBus iniciado, ao ativar a comunicação com o dispositivo (ícone verde na barra de ferramentas do driver), as *tag's* ficarão com a escrita em azul Figura 29 e passarão a mostrar o valor que o simulador está gerando.

Figura 28 – *Tag's* configuradas



Fonte: Autor (2022)

Figura 29 – *Tag's* conectadas com dispositivo



Driver *

Name	Device	Item	P1/N1...	P2/N2...	P3/N3...	P4/N4...	Siz...	Scan	Value	Quality	Timestamp	Value (unscaled)
Driver			0	0	0	0						
Entradas_digitais			1	6	0	1	5	1000		192	13/11/2022 11:08:28,251	
Emergencia							0		0	192		0
Automatico							1		1	192		1
Manual							2		0	192		0
Sensor_1							3		1	192		1
Sensor_2							4		0	192		0
Saidas_digitais			1	6	0	6	5	1000		192	13/11/2022 11:08:28,251	
Sin_Emergencia							0		0	192		0
Sin_Automatico							1		1	192		1
Sin_Manual							2		0	192		0
Motor_1							3		1	192		1
Motor_2							4		1	192		1
Memorias_internas			1	1	0	1	2	1000		192	13/11/2022 11:08:28,264	
Alarmes							0		0	192		0
Supervisorio							1		0	192		0
Entradas_analogicas			1	1	0	3	6	1000		192	13/11/2022 11:08:28,264	
Nivel							0		214	192		214
Temperatura							1		51	192		51
Corrente_1							2		29	192		29
Corrente_2							3		27	192		27
Tensao_1							4		378	192		378
Tensao_2							5		380	192		380
Saidas_analogicas			1	1	0	9	2	1000		192	13/11/2022 11:08:28,264	
Valvula_1							0		50	192		50
Valvula_2							1		85	192		85

Fonte: Autor (2022)

4.3 CONFIGURAÇÃO BANCO DE DADOS

Uma das principais características e funções de um sistema supervisório é a integração com um servidor de banco de dados, possibilitando o armazenamento dos valores das variáveis do processo, para posteriormente gerar relatórios e gráficos personalizados.

Um banco de dados foi instanciado na inicialização do projeto, portanto só é necessário realizar o teste de conexão com o mesmo antes de configurar as tabelas de dados que irão ser registrados nesse banco. O teste de conexão é feito ao acessar as propriedades do banco de dados como é mostrado na Figura 30.

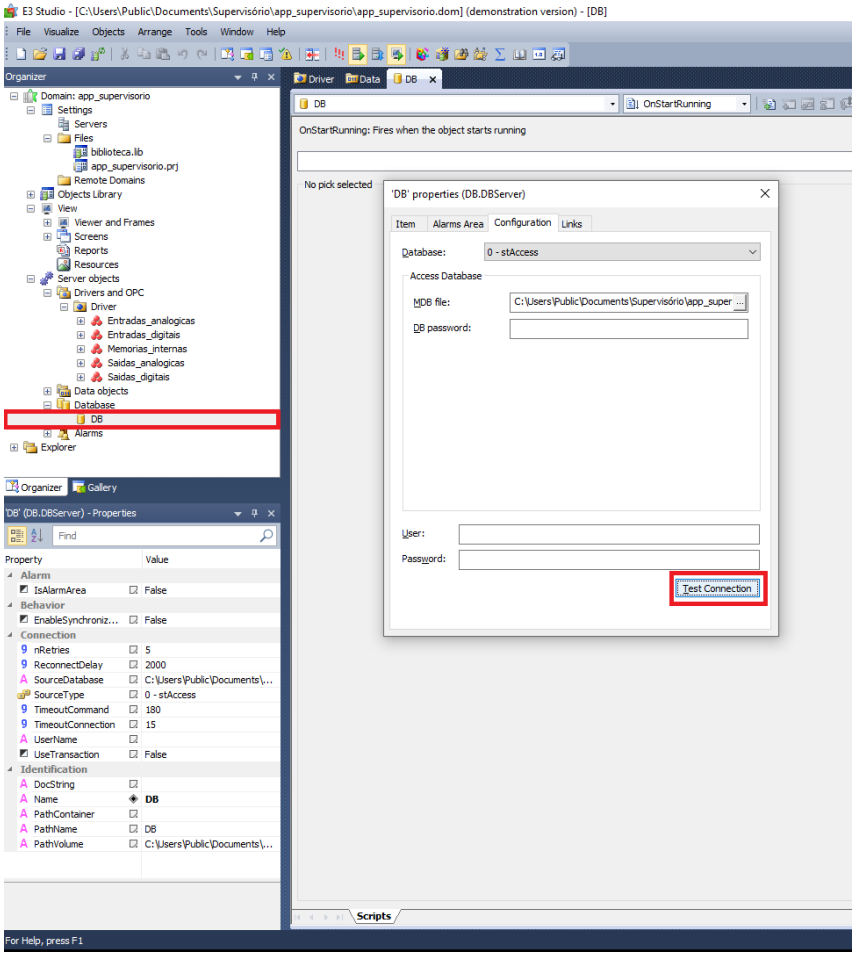
4.3.1 Históricos e tabelas

A configuração das informações que serão salvas no banco de dados são realizadas através do recurso de *Históricos* no Elipse, que podem ser adicionados conforme é mostrado na Figura 31. Este recurso realiza a criação de *tabelas* dentro do banco de dados. As tabelas possuem campos, cada campo corresponde a uma variável que será armazenada.

A estruturação do banco de dados que será utilizado neste projeto contará com duas tabelas, ou seja, dois históricos. Uma tabela irá armazenar os valores das variáveis dos *registers* e a outra os status das *coils*.

Ao adicionar um histórico no projeto o Elipse automaticamente insere um

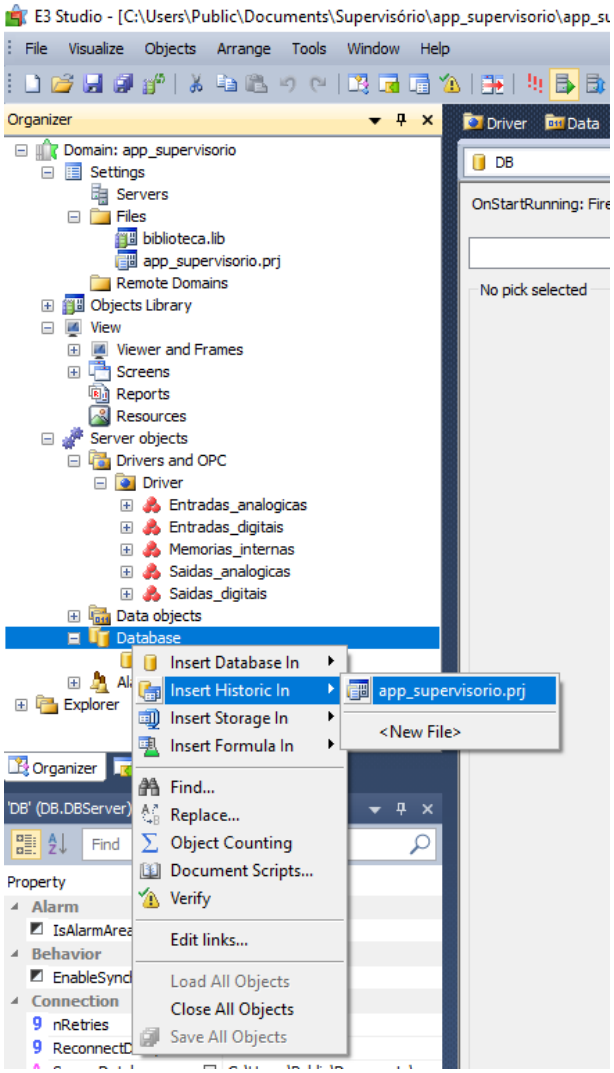
Figura 30 – Teste de conexão com banco de dados



Fonte: Autor (2022)

campo *E3TimeStamp*, este campo representa o momento em que será salvo os valores daquele histórico no banco de dados e usa o horário do computador que está executando a aplicação supervisória. Os demais valores que serão salvos na tabela serão provenientes das *tags* do driver, logo ao adicionar campos no histórico deve-se associar os mesmos às referentes fontes *(Source)*, como é ilustrado na Figura 32, que aponta a propriedade ***Value*** da tag *Corrente_1* do driver ao campo ***corrente_1*** do histórico.

Figura 31 – Inserir histórico no banco de dados

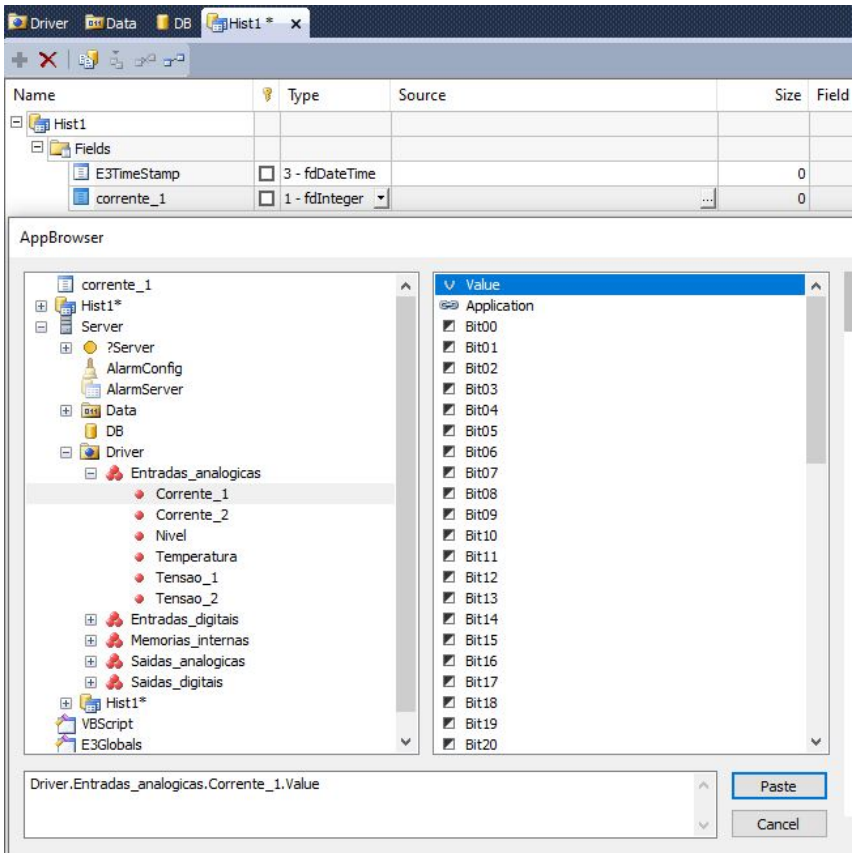


Fonte: Autor (2022)

Nas figuras 33 e 34 são apresentados os campos do histórico dos *registers* e das *coils* respectivamente configurados, que serão utilizados neste projeto.

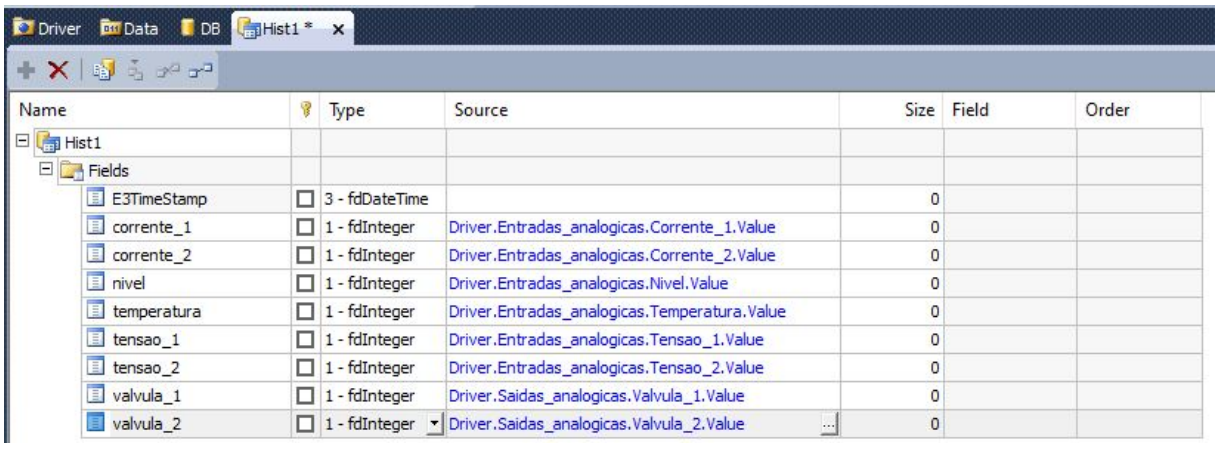
Após adicionar um histórico no projeto e associar todos os seus campos às fontes de dados, o mesmo deve ser configurado e gerado no banco de dados como uma tabela. Para isso ele deve apontar em qual banco de dados do projeto será criada a tabela. Para nomear uma tabela em um banco de dados, assim como um campo, costuma-se utilizar apenas letras minúsculas, fazer o uso do *underline ("_")* em casos de nomes compostos e de uma nomenclatura que explique o que está sendo salvo na tabela. O intervalo de tempo em que será salvo as informações pode ser configurado no campo *Interval between records*, por padrão do Elipse a cada 1000ms grava uma linha na tabela, este valor deve ser maior ou igual ao tempo de scan da variável a ser gravada. É possível verificar as tabelas criadas e suas configurações no supervisório

Figura 32 – Configuração do campo histórico



Fonte: Autor (2022)

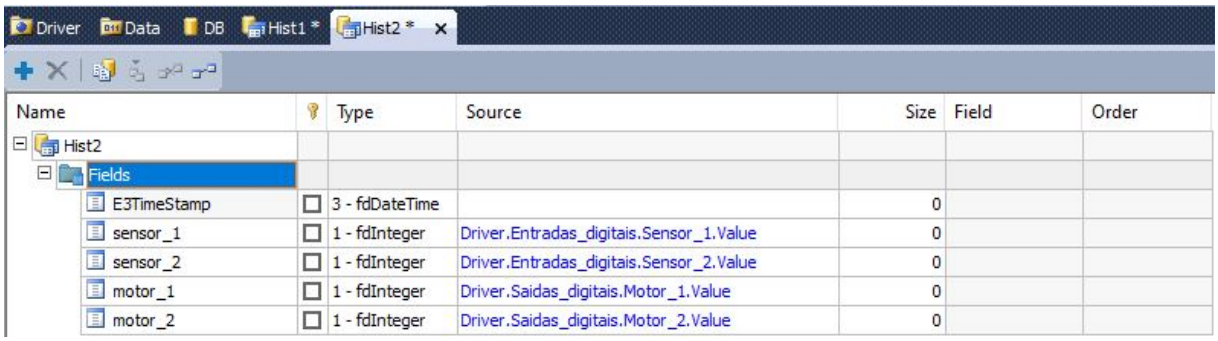
Figura 33 – campos do histórico *registers* configurado



Fonte: Autor (2022)

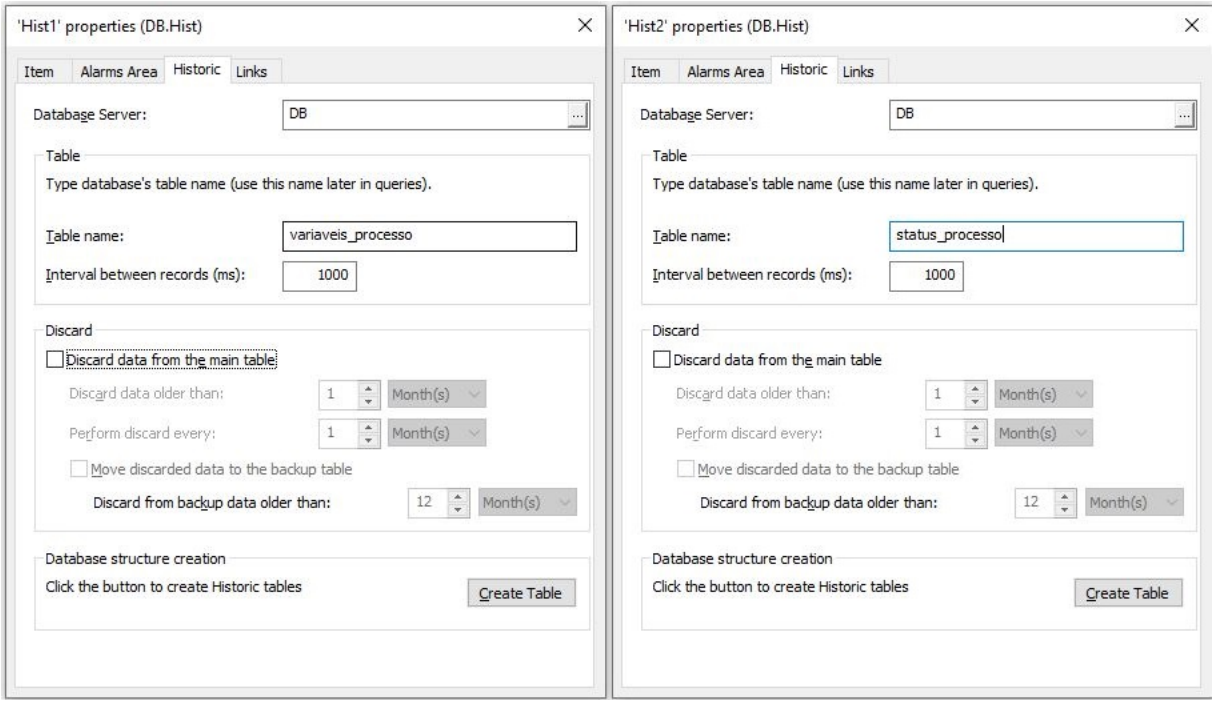
apresentado neste trabalho através da Figura 35. A geração das tabelas se conclui ao selecionar o botão *Create Table* e a partir do momento em que a aplicação supervisória iniciar os registros serão salvos no banco de dados.

Figura 34 – Campos do histórico *coils* configurado



Fonte: Autor (2022)

Figura 35 – Configuração das tabelas *variaveis_processo* e *status_processo*



Fonte: Autor (2022)

4.3.2 Gerenciamento de dados de uma tabela

Há possibilidade de adicionar o descarte automático de dados do banco de dados, para casos em que o banco possui uma capacidade máxima de utilização, mas este projeto não será utilizado tal recurso. Porém para fins didáticos de cálculo de tamanho utilizado por cada linha da tabela, seguem os valores em bytes de cada campo levando em consideração o tipo de dado armazenado:

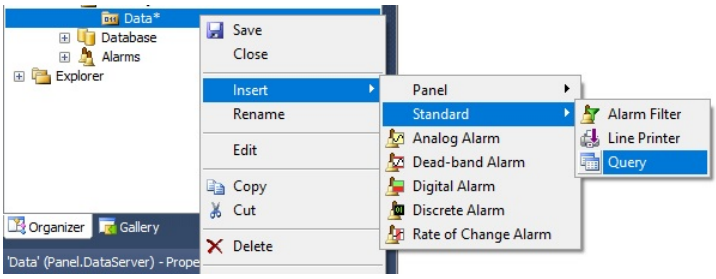
- ***DataTime***: 8 bytes;
- ***Integer***: 4 bytes;
- ***Double***: 8 bytes;
- ***Text***: 1 byte por caractere.

Levando em consideração as tabelas criadas no projeto, a tabela *variaveis_processo* ocupará 40 bytes por linha e a tabela *status_processo* 24 bytes.

4.3.3 Consulta aos dados de uma tabela

Para realizar a consulta aos dados das tabelas criadas anteriormente o Elipse possui um objeto nativo chamado *Query*, ele pode ser adicionado no servidor de dados, telas ou de objetos, através dos menus *Insert>Standard>Query*, como mostrado na Figura 36.

Figura 36 – Adicionando uma *Query* ao projeto



Fonte: Autor (2022)

Após instanciar uma *Query* na aplicação, deve-se clicar com o botão direito do mouse e acessar o menu "*Configure...*". Deve-se apontar a qual banco de dados que será feita a consulta e após isso em qual tabela serão buscados os dados. Para auxiliar na consulta ao banco de dados a janela de configuração de consulta do Elipse é intuitiva (Figura 37), os campos que desejam ser consultados devem ser marcados na aba *Fields* e após isso a consulta deve ser executada na aba *Visualize*.

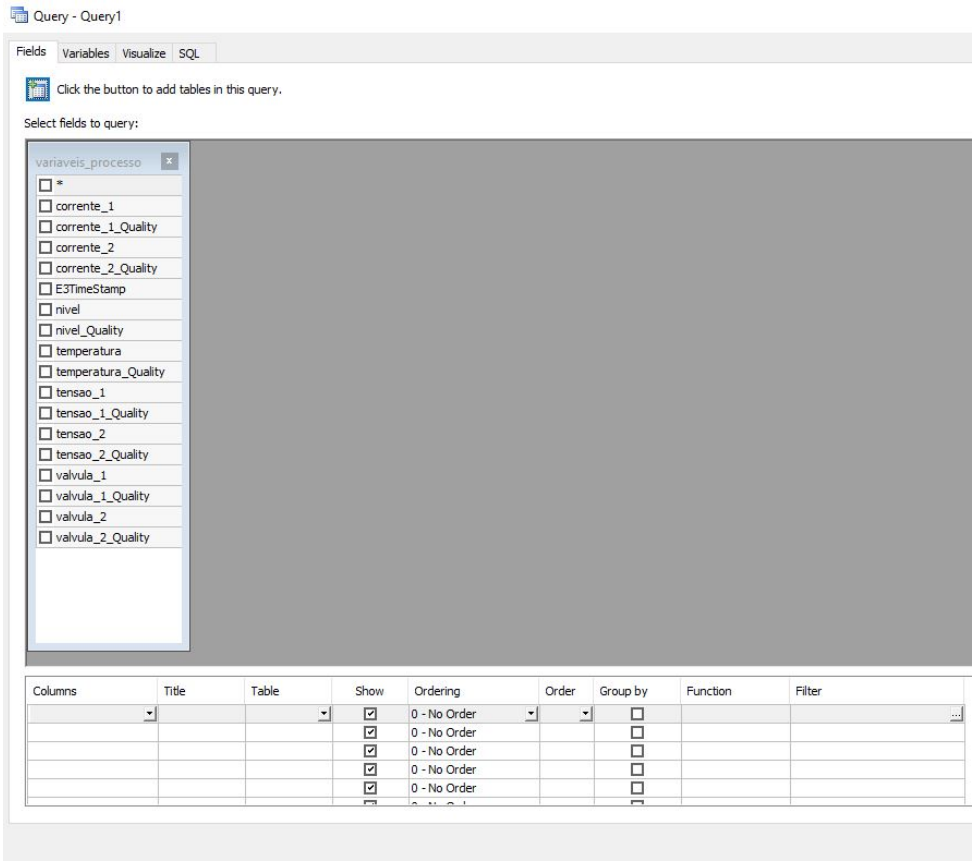
Uma consulta sem filtros irá retornar todos os dados selecionados contidos naquela tabela, assim como é mostrado na Figura 38. Em um supervisório os dados são filtrados por períodos de tempo, por exemplo, os valores da corrente de um motor elétrico nas últimas 12 horas. O retorno desses valores pode ser apresentado como um gráfico ou relatório. Portanto, será utilizado um modelo padrão para os filtros consultas neste supervisório, o filtro para escolha da data inicial e da data final da consulta.

O campo *E3TimeStamp* será filtrado por duas variáveis, ou seja, o campo precisa atender ao valor dessas variáveis, a primeira será a data inicial da consulta, logo o valor campo precisa ser maior ou igual ao valor da data inicial. A segunda variável será a data final, consequentemente o campo precisa ser menor que a data final. Com esse filtro o retorno da consulta será apenas as linhas que se encontram dentro do período das variáveis. A declaração dessas variáveis seguem a sintaxe do SQL onde:

- **<%NomeDaVariável%>** se o valor for numérico.
- **¿%NomeDaVariável%>´** se for texto (string).
- **#<%NomeDaVariável%>#** se o valor for uma data.

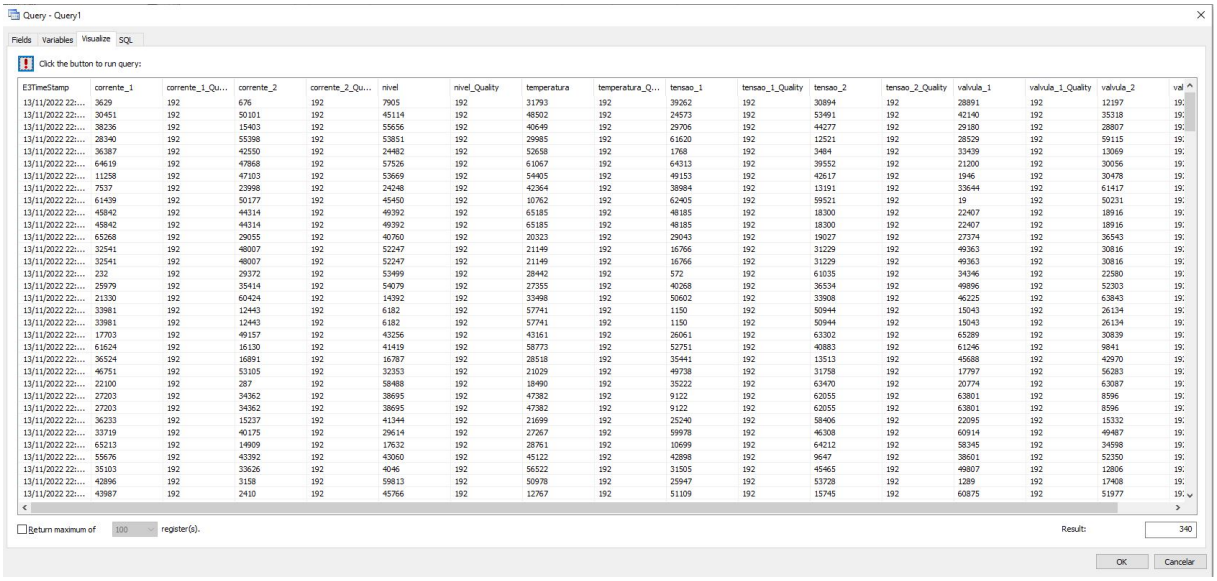
Logo a sintaxe utilizada para as variáveis inicial e final será #<%data_inicial%>#

Figura 37 – Janela configuração de *Query*



Fonte: Autor (2022)

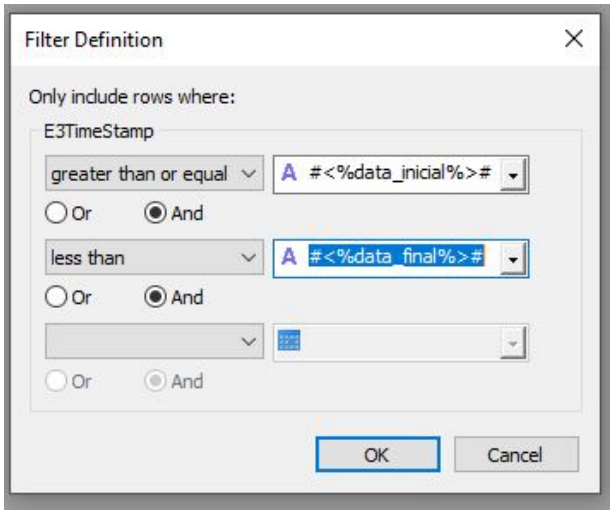
Figura 38 – Consulta à uma tabela sem filtros



Fonte: Autor (2022)

e #<%data_final%>#, como mostrado na Figura 39.

Figura 39 – Filtro de data para consulta



Fonte: Autor (2022)

4.4 SERVIDOR DE ALARMES

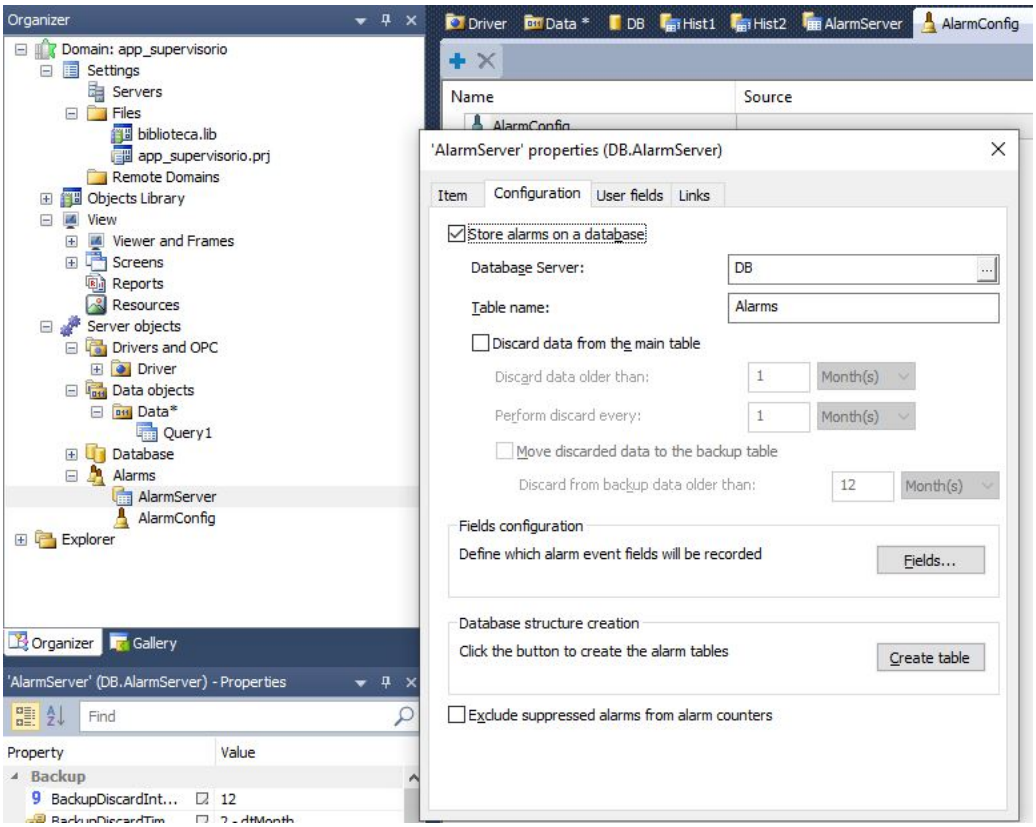
A estrutura de armazenamento dos alarmes é definida pelo servidor de alarmes, já instanciado, assim como os recursos apresentados anteriormente, na inicialização do supervisório. O servidor de alarmes funciona como um histórico e todos os alarmes instanciados dentro do supervisório caso sejam disparados irão ser salvos na tabela criada na inicialização do projeto. Os campos de alarme que irão para o banco de dados podem ser configurados na aba *Configuration*, da janela de configuração do *Alarm Server* através do botão *"Fields..."*, Figura 40.

Os campos de alarmes que serão armazenados no banco de dados para a aplicação serão, o tempo de entrada, tempo de saída, mensagem, severidade e a fonte do alarme.

4.4.1 Criação de Alarmes

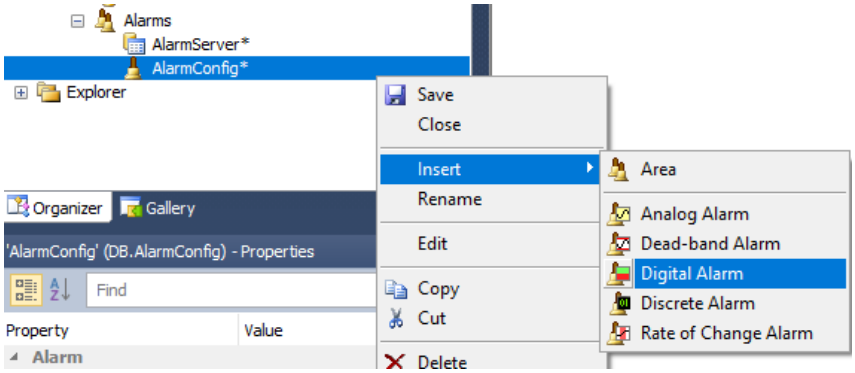
A configuração de alarmes no Elipse podem se através de diferentes tipos de alarmes (Figura 41). Neste projeto serão utilizados alarmes digitais e analógicos. Os alarmes digitais terão como fonte a palavra de alarmes do driver modbus e os alarmes analógicos serão limitados pelas variáveis de processo nível, temperatura, corrente e tensão.

Figura 40 – Configuração *AlarmServer*



Fonte: Autor (2022)

Figura 41 – Tipos de alarme ElipseE3



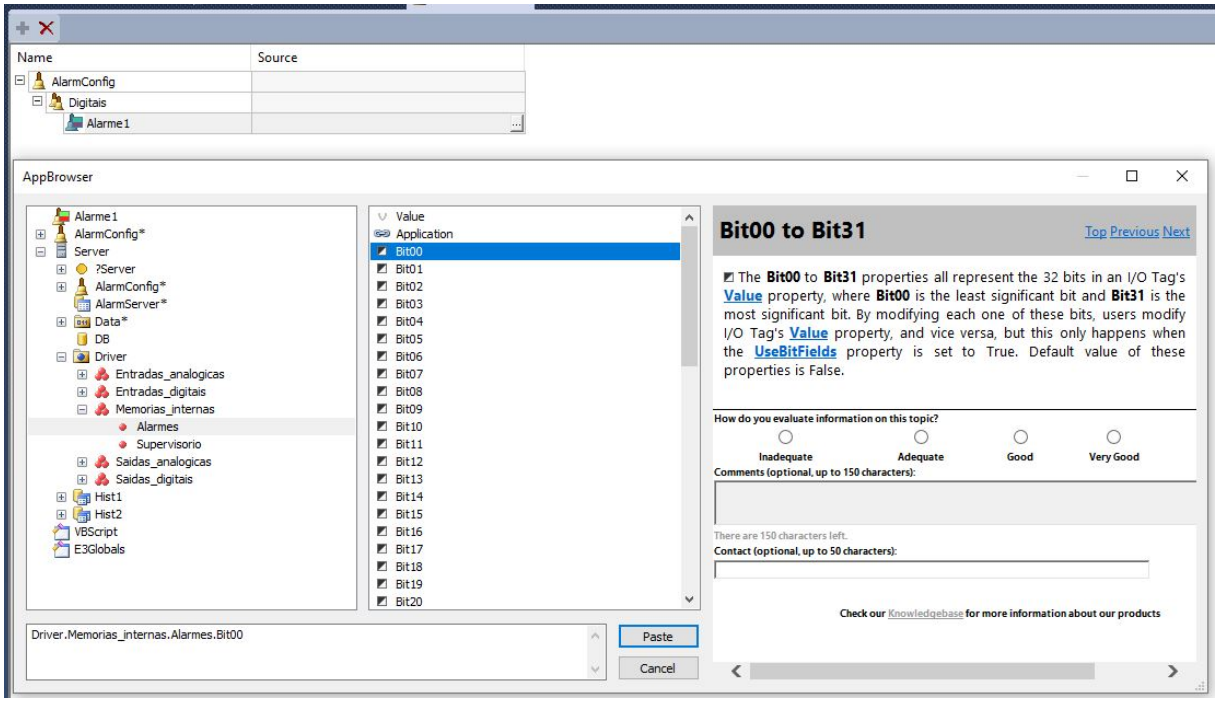
Fonte: Autor (2022)

Existem duas propriedades comuns à todos os tipos de alarmes o *reconhecimento (Ack)* e a *inibição Shelving*. O reconhecimento do alarme é uma condição onde o operador do sistema supervisório necessita realizar alguma operação ou confirmar o alarme, caso isso não seja feito mesmo após a condição do alarme ser falsa ele permanecerá na lista de alarmes. A inibição do alarme é uma condição onde o alarme está ativo, porém o operador inibe fazendo com que o alarme fique em uma condição falsa. Ambas as propriedades devem ser habilitadas ou não conforme as características do alarme.

4.4.2 Alarme Digital

A configuração de um alarme digital se dá principalmente pela associação da sua fonte, que deve ser uma tag que apresente valores booleanos (*ON/OFF*), neste projeto serão utilizado os bits da palavra de alarmes do driver (Figura 42).

Figura 42 – Fonte *bool* alarme digital

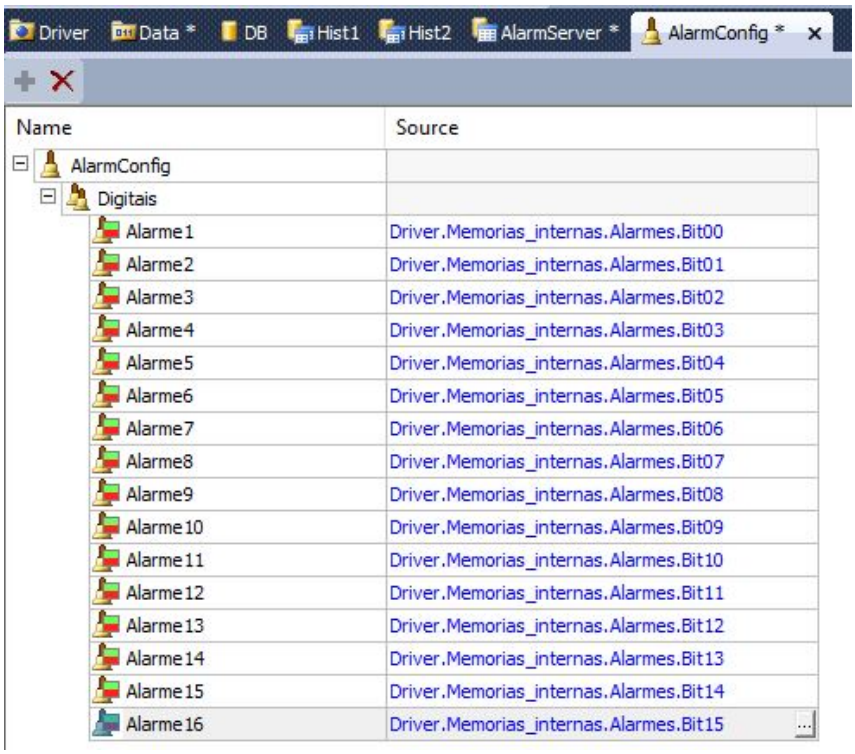


Fonte: Autor (2022)

O método de utilizar bits de palavras é útil para economia de tags, uma vez que, uma tag do tipo Word(16 bits), ou uma tag Bool(1 bit) possuem o custo de uma tag do driver, relembrando que a versão *demo* possibilita apenas 20 tags de driver. Para que seja possível a leitura desses bits a propriedade da tag do driver *UseBitFields* deve estar em ***True***, assim o valor decimal de uma tag é convertido em binário e os bits podem ser associado em até 16 alarmes, mostrado na Figura 43.

As propriedades dos alarmes digitais podem ser editadas através da aba *Digital*. Para um alarme digital ser ativado no sistema a caixa *Enable* deve estar habilitada, após isso suas propriedades podem ser alteradas. A mensagem que o alarme irá mandar no supervisório caso ele atue deve ser inserida na caixa *Message text*, a sua severidade (*Low, Medium, High, Critical*) é configurada na lista *Severity*, a necessidade de reconhecer o alarme é habilitada ao selecionar a caixa *Need ack*, o tempo de atraso é configurado na caixa *Delays(ms)* e indica o tempo que o alarme demora para atuar após a condição de sua fonte ser verdadeira, a mensagem quando a condição do alarme for falsa é definida na caixa *Return message* e a possibilidade de inibir o alarme

Figura 43 – Alarmes digitais do supervisório

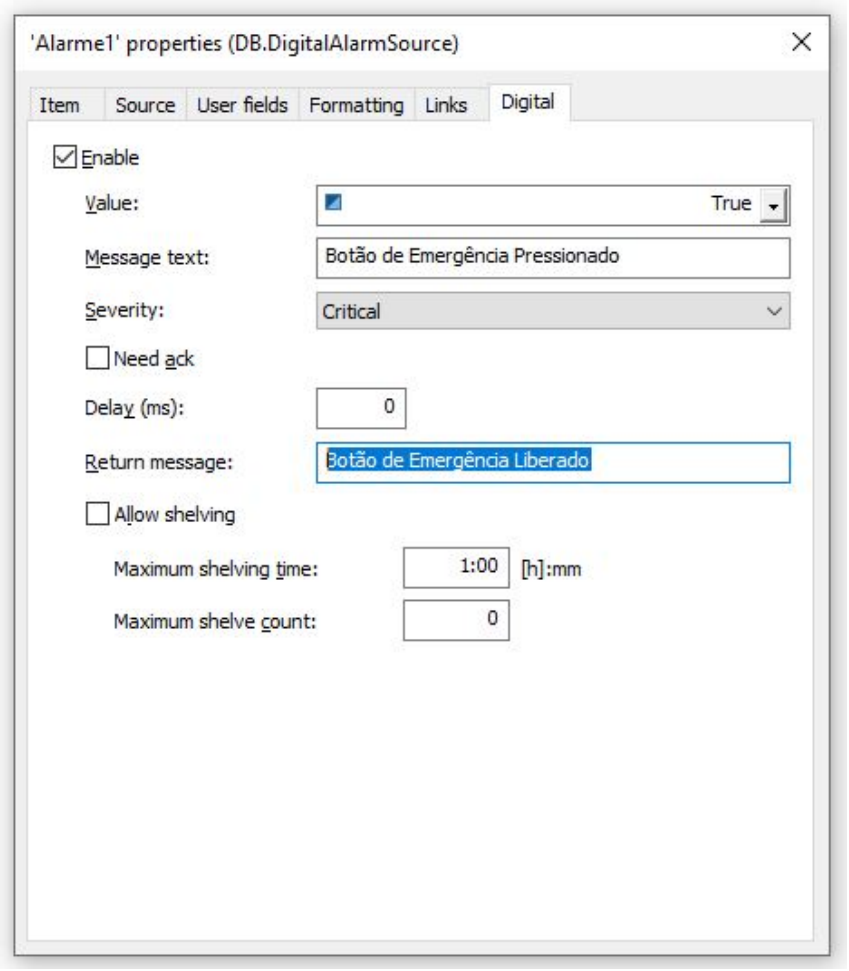


Fonte: Autor (2022)

é habilitada ao selecionar a caixa *Allow shelving*.

A configuração de um alarme utilizado neste projeto é ilustrado pela Figura 44, ele está habilitado, sua mensagem quando ativo é *"Botão de Emergência Pressionado"*, sua severidade é a mais elevada *Critical*, não há a necessidade de ser reconhecido, não possui atraso para atuar, sua mensagem quando a condição da sua fonte é retorna para falsa é *"Botão de Emergência Liberado"* e não é permitida sua inibição.

Figura 44 – Configuração de um alarme digital



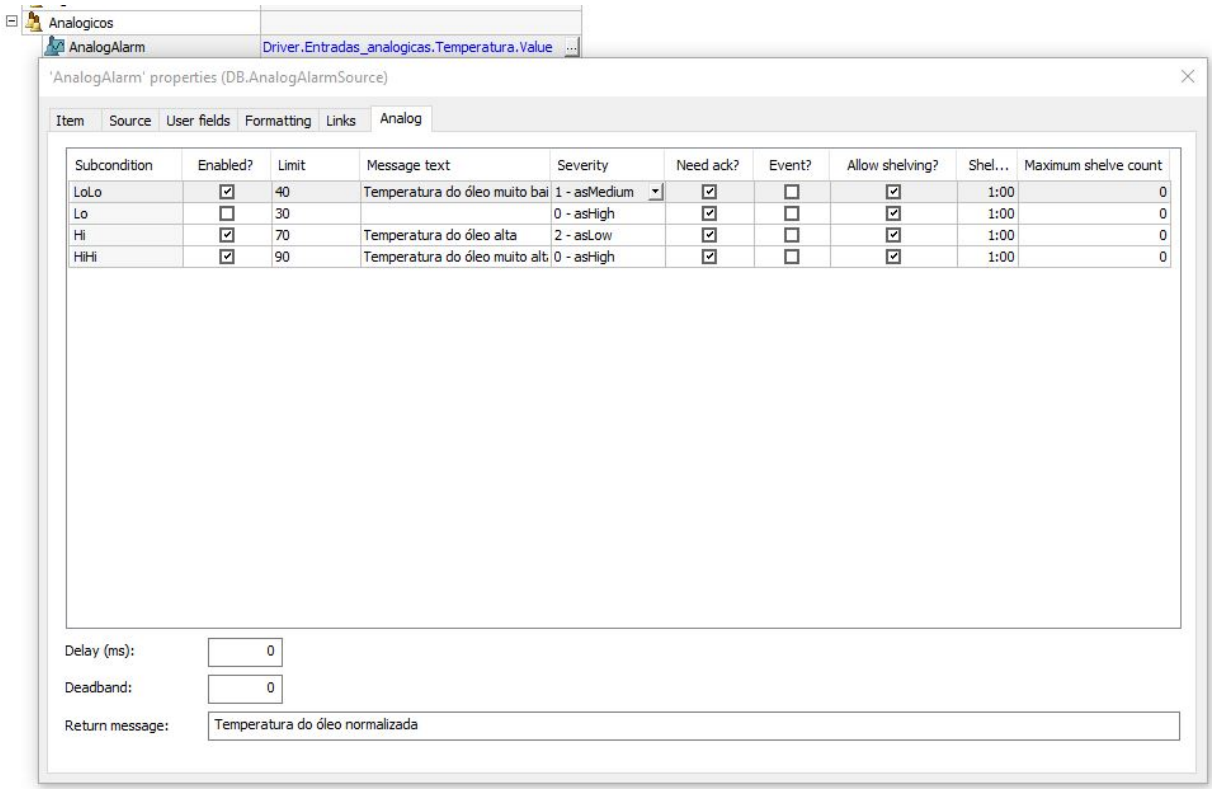
Fonte: Autor (2022)

4.4.3 Alarme Analógico

Assim como no alarme digital, nesta configuração deve ser associada uma fonte ao alarme analógico, porém este tipo de alarme possui 4 diferentes tipos de condições de ativação, *HiHi, Hi, Lo* e *LoLo*. As condições *HiHi* e *Hi* são ativadas se o valor da fonte ultrapassar seus limites, enquanto as condições *LoLo* e *Lo* são ativas quando o valor da fonte for menor que eles. Abaixo é mostrado, na Figura 45, a configuração do alarme da temperatura do óleo que terá como sua fonte o valor da tag Temperatura do driver modbus. Ele possui a condição *LoLo, Hi* e *HiHi* ativas, cada uma com sua mensagem específica, os limites são 40, 70 e 90 respectivamente. A severidade de cada condição é diferente, apenas a condição *LoLo* necessita de reconhecimento e nenhuma condição pode ser inibida.

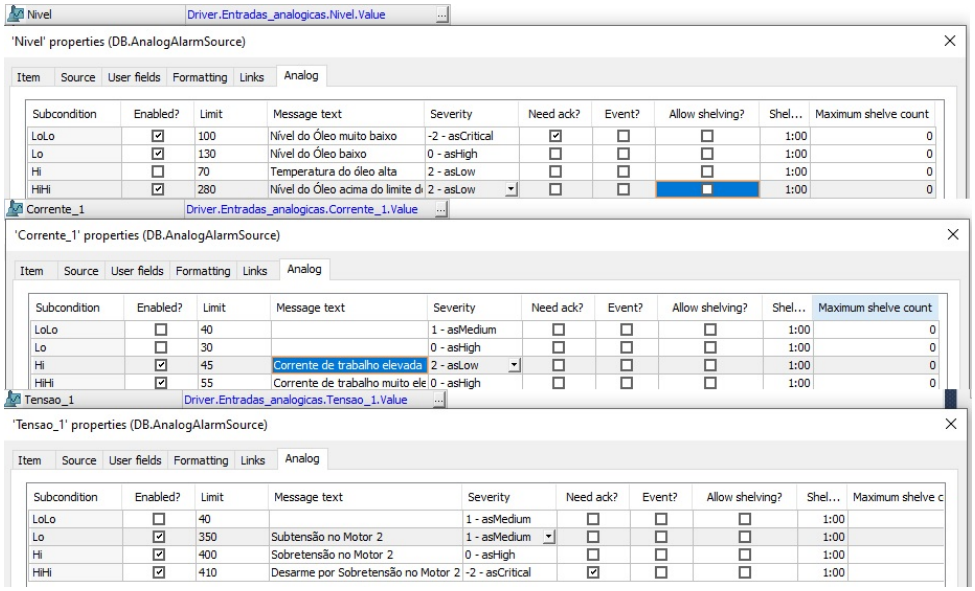
A configuração dos outros alarmes analógicos adicionados no projeto são mostrados abaixo na Figura 46. Os alarmes de Corrente e de Tensão dos motores 1 e 2 possuem os mesmos limites.

Figura 45 – Configuração de um alarme analógico



Fonte: Autor (2022)

Figura 46 – Configuração dos alarmes analógicos do supervisório



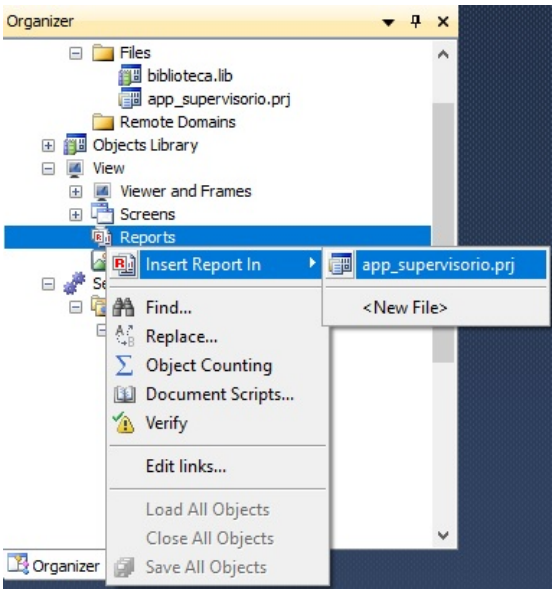
Fonte: Autor (2022)

4.5 CRIAÇÃO DE RELATÓRIOS

Os relatórios funcionam como uma visualização de um histórico, com as informações desejadas e formatadas conforme o desenvolvedor do supervisório deseja

que pode ser exportados para arquivos PDF ou em Excel. Ao adicionar um objeto Relatório , em *Reports> Insert Report In> app_supervisorio.prj* (Figura 47), o mesmo é instanciado com uma consulta em branco em sua estrutura. O relatório que será desenvolvido neste projeto será o relatório dos alarmes do sistema.

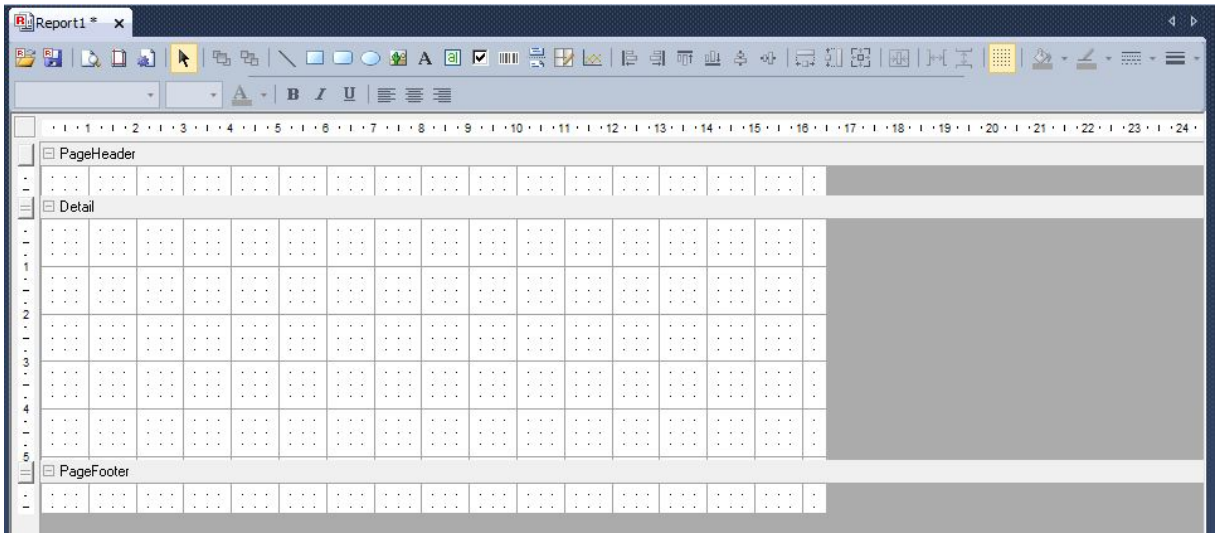
Figura 47 – Adicionando um relatório na aplicação supervisória



Fonte: Autor (2022)

A formatação do relatório é feita pelas ferramentas disponíveis na interface do Elipse mostradas na Figura 48. A interface mostra a área do relatório dividido em três partes, *PageHeader, Detail* e *PageFooter*. As seções *PageHeader* e *PageFooter* são o cabeçalho e rodapé do relatório, respectivamente. A divisão *Detail* é onde as informações do banco de dados serão escritas.

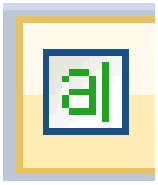
Figura 48 – Interface de formatação do relatório



Fonte: Autor (2022)

A ferramenta de edição que cria uma caixa de texto para a escrita dos dados provenientes do banco de dados é a ferramenta *SetPoint*, ícone mostrado na Figura 49. Ao criar a caixa de texto com a ferramenta *SetPoint* deve se configurar a sua propriedade *DataField* para o mesmo nome que será utilizado na consulta, ou seja, para o relatório de alarmes teremos as possíveis informações que podem ser extraídas da tabela Alarmes: tempo de entrada, tempo de saída, mensagem, severidade e fonte do alarme.

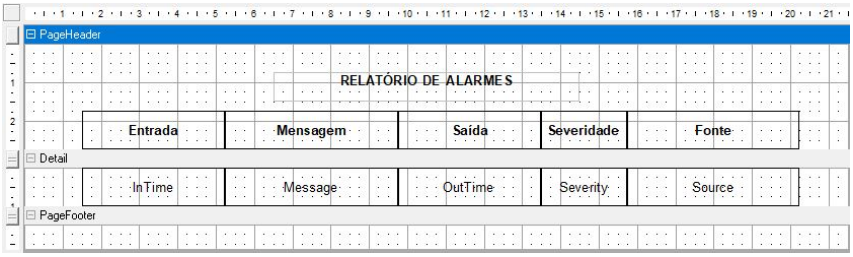
Figura 49 – Ferramenta *SetPoint*



Fonte: Autor (2022)

A configuração da consulta será a mesma apresentada anteriormente, adicionando um filtro de tempo para o campo E3TimeStamp. A formatação do relatório é mostrada a seguir na Figura 50.

Figura 50 – Formatação do relatório de alarme



Fonte: Autor (2022)

A visualização do relatório em tempo de desenvolvimento pode ser executada na ferramenta *Preview Report*, para execução em tempo de execução será apresentado posteriormente. O relatório abaixo (Figura 51) apresenta os alarmes ocorridos em um determinado período de tempo em que os valores enviados pelo simulador, ou seja, a fonte dos alarmes, estavam em modo randômico.

Figura 51 – Visualização do relatório de alarme

RELATÓRIO DE ALARMES

Entrada	Mensagem	Saída	Severidade	Fonte
14/11/2022 22:52:51	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 22:55:13	Temperatura do óleo alta	00:00:00	2	Driver.Entradas_analogicas.Temperatura.Value
14/11/2022 22:55:13	Temperatura do óleo normalizada	14/11/2022 22:55:45	2	Driver.Entradas_analogicas.Temperatura.Value
14/11/2022 22:52:51	Botão de Emergência Liberado	14/11/2022 22:55:56	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:48	Corrente de trabalho muito elevada Motor 2	00:00:00	0	Driver.Entradas_analogicas.Corrente_2.Value
14/11/2022 23:13:48	Desarme por Sobretensão no Motor 2	00:00:00	-2	Driver.Entradas_analogicas.Tensao_1.Value
14/11/2022 23:13:48	Corrente de trabalho muito elevada Motor 1	00:00:00	0	Driver.Entradas_analogicas.Corrente_1.Value
14/11/2022 23:13:48	Temperatura do óleo muito alta	00:00:00	0	Driver.Entradas_analogicas.Temperatura.Value
14/11/2022 23:13:48	Nível do Óleo acima do limite do sensor	00:00:00	2	Driver.Entradas_analogicas.Nivel.Value
14/11/2022 23:13:48	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:48	Botão de Emergência Liberado	14/11/2022 23:13:50	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:52	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:52	Botão de Emergência Liberado	14/11/2022 23:13:54	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:58	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:13:58	Botão de Emergência Liberado	14/11/2022 23:14:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:02	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:02	Botão de Emergência Liberado	14/11/2022 23:14:09	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:11	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:11	Botão de Emergência Liberado	14/11/2022 23:14:13	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:13	Temperatura do óleo alta	00:00:00	2	Driver.Entradas_analogicas.Temperatura.Value
14/11/2022 23:14:15	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:15	Temperatura do óleo muito alta	00:00:00	0	Driver.Entradas_analogicas.Temperatura.Value
14/11/2022 23:14:15	Botão de Emergência Liberado	14/11/2022 23:14:17	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:21	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:21	Botão de Emergência Liberado	14/11/2022 23:14:23	-2	Driver.Memorias_internas.Alarmes.Bit00
14/11/2022 23:14:25	Botão de Emergência Pressionado	00:00:00	-2	Driver.Memorias_internas.Alarmes.Bit00

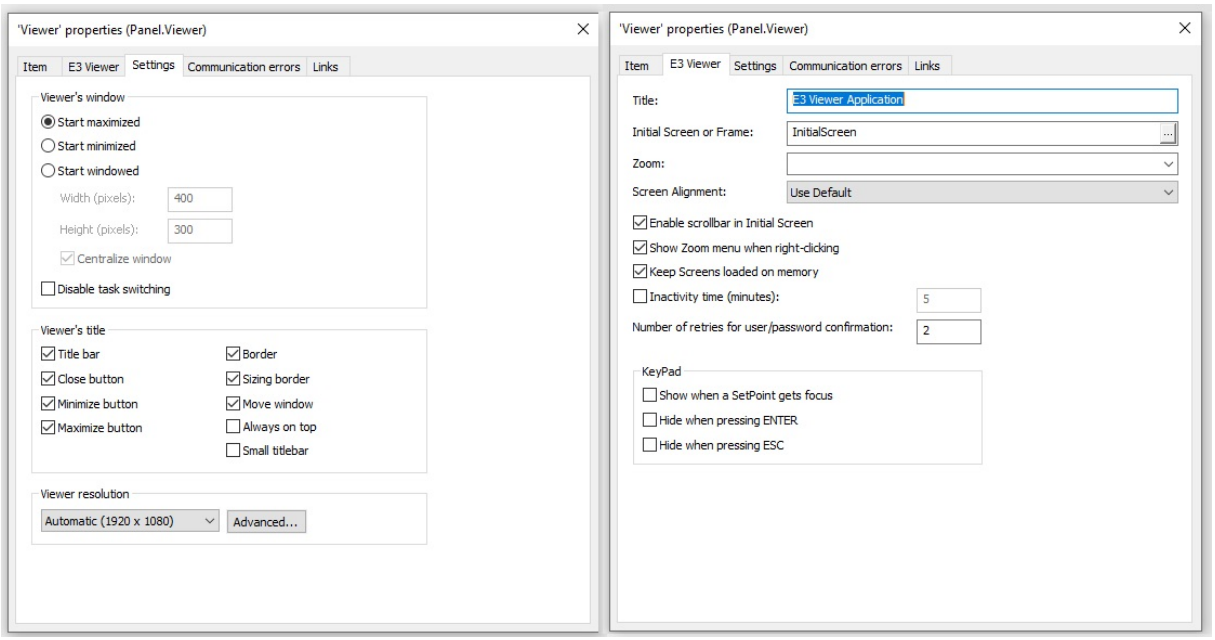
Fonte: Autor (2022)

4.6 CONFIGURAÇÃO E DESENVOLVIMENTO DAS TELAS

O desenvolvimento apresentado até o momento foi realizado para estruturar o *"backend"* da aplicação supervisória. O *"frontend"*, ou seja, o que será mostrado para o usuário final realizar as operações de comando e visualização do sistema supervisório se encontra no servidor de telas e os próximos tópicos abordarão o desenvolvimento dessa etapa.

O *viewer* é o nome dado, dentro do desenvolvimento do supervisório, ao aplicativo onde a aplicação irá ser executada pelo usuário. A configuração do tamanho de display do viewer é feita na inicialização do projeto, é recomendado que a resolução de desenvolvimento seja a mesma em que a aplicação supervisório irá rodar, pois existe a possibilidade de distorção de textos e objetos caso as resolução sejam diferente. O projeto deste trabalho será desenvolvido em um display FullHD, ou seja, com a resolução de 1920x1080 pixels. A configuração do display do projeto pode ser acessada através do servidor de telas pelo *organizer* em *Viewer > Viewer and Frames > Viewer > Settings*. As janelas de configuração do *viewer* são mostradas na Figura 52:

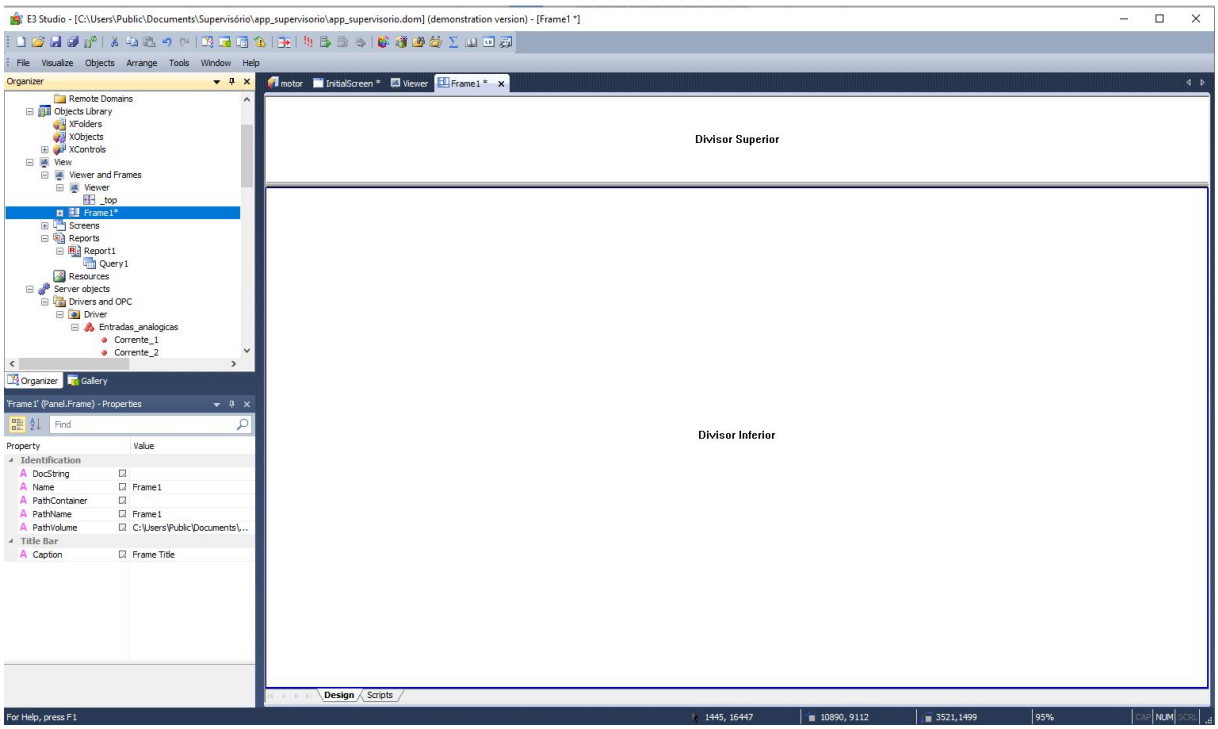
Figura 52 – Janelas de configuração do viewer



Fonte: Autor (2022)

As telas do supervisórios são mostradas em *quadros* do viewer, esses quadros possuem divisores, onde em cada divisor pode ser vinculado à uma tela criada. Um exemplo é mostrado na Figura 53 , onde o quadro possui uma divisão inferior para exibir o conteúdo principal e outra superior destinada para os menus do supervisório. Ao criar a tela de menus deve-se apontar ao divisor superior do quadro e ao criar a tela das informações deve-se associar a tela ao divisor inferior.

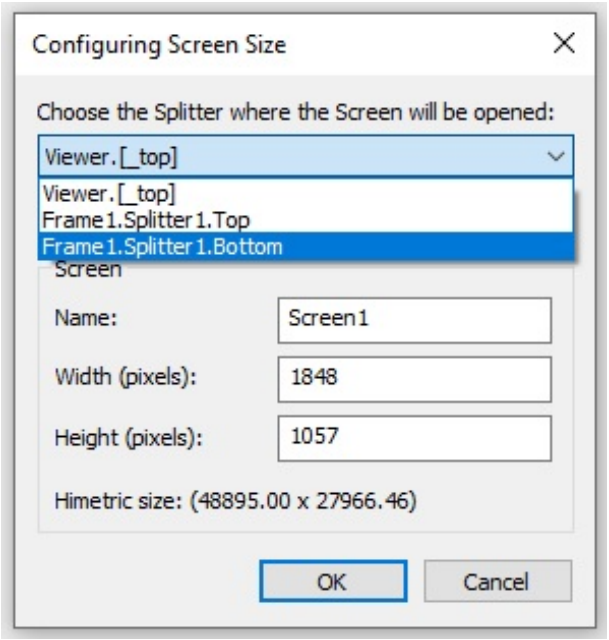
Figura 53 – Configuração do quadro principal



Fonte: Autor (2022)

Ao adicionar uma tela no supervisório um *popup* abre informando em qual divisor a tela será aberta, a fim de dimensionar a tela do mesmo tamanho do divisor evitando problemas de resolução (Figura 54).

Figura 54 – Configuração da tela em relação aos divisores



Fonte: Autor (2022)

Neste projeto serão utilizadas 4 telas no quadro principal, a tela de menus, que será vinculado ao quadro superior e será fixa, a tela de monitoramento do processo, onde será desenvolvido uma simulação do processo com os objetos desenvolvidos, uma tela de gráficos com as variáveis do processo e uma tela de alarmes, essas telas serão mostradas no divisor inferior.

A barra de ferramentas de criação de objetos em tela é mostrado na Figura 55. Os principais objetos que serão utilizados para representação no supervisório não serão desenhados diretamente em tela, eles serão criados como objetos do projeto (XControls) e depois serão adicionados à tela.

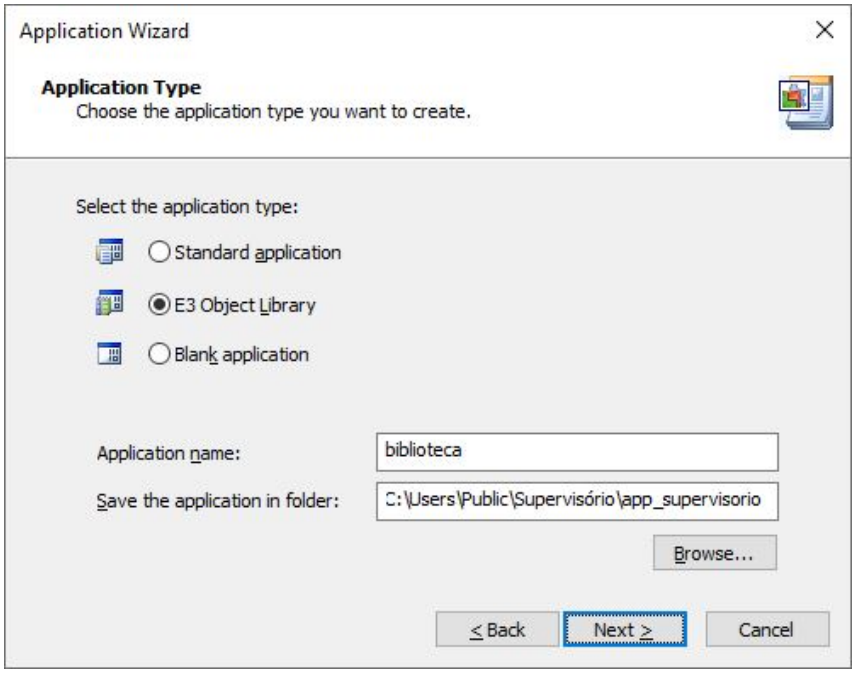
Figura 55 – Barra de ferramentas de desenho em tela supervisório

Fonte: Autor (2022)

4.7 CRIAÇÃO DE BIBLIOTECAS

Para a criação das classes de objetos que serão utilizadas no projeto, é necessário a criação de uma biblioteca (arquivo .lib), nela serão criados os XFolders, XObjects e XControls da aplicação supervisória. Ao clicar em *New* na barra de ferramentas a janela mostra a opção de adicionar uma biblioteca ao domínio atual (Figura 56).

Figura 56 – Criando uma biblioteca no domínio E3



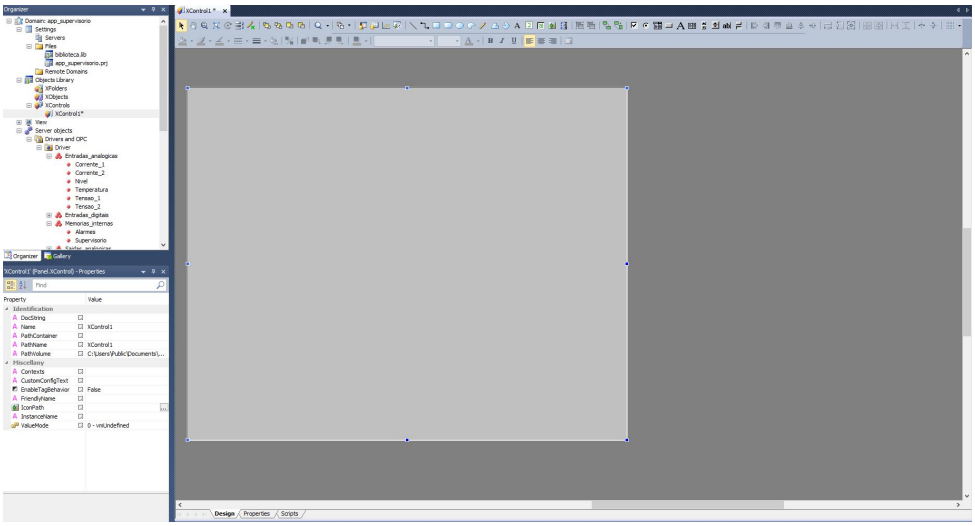
Fonte: Autor (2022)

4.7.1 Criação das classes

Essa seção descreverá a criação das classes utilizadas no projeto supervisório e como configurar e utilizar a mesmas em uma aplicação. A criação de classes tem como objetivo facilitar no desenvolvimento de um projeto trazendo as vantagens da programação orientada à objeto.

Levando em consideração a aplicação atual e as aplicação industriais em um contexto geral, foram listadas as classes que podem ser aplicadas nesta e em outras aplicações supervisórias. Para as classes XControl a biblioteca ira conter:*motor* e *barra_alarme*. A criação de XObject se dá pelo menu *Organizer >XControls >Insert XControl In > biblioteca.lib*. Ao adicionar um novo XControl na biblioteca a interface mostra as ferramentas que podem ser utilizadas para desenhar na aba *Desing*, a opção para configuração das propriedades do XControl na aba *Properties* e os *Scripts* caso seja utilizado na aba Script, assim como é mostrado na Figura 57.

Figura 57 – Interface edição XControl

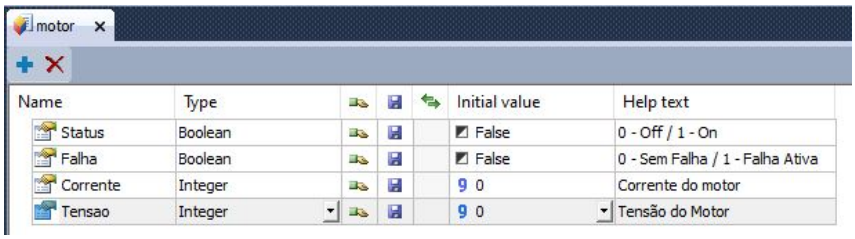


Fonte: Autor (2022)

4.7.1.1 Classe Motor

O XControl motor apresentará as propriedades conforme a Figura 58 abaixo:

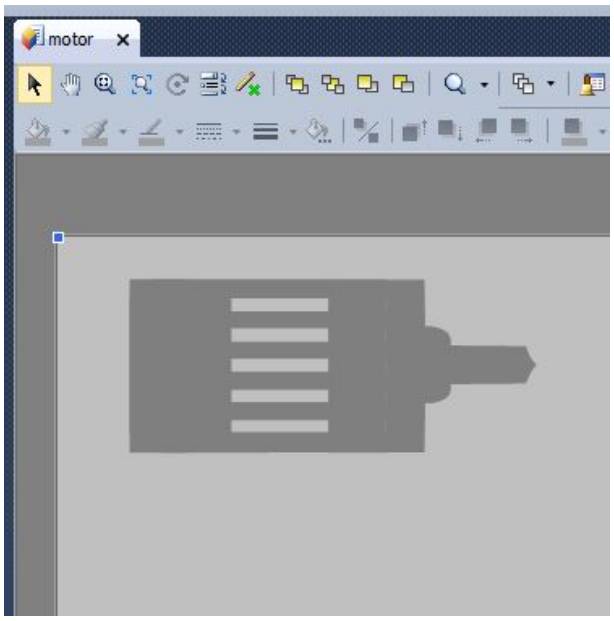
Figura 58 – Propriedades XControl Motor



Fonte: Autor (2022)

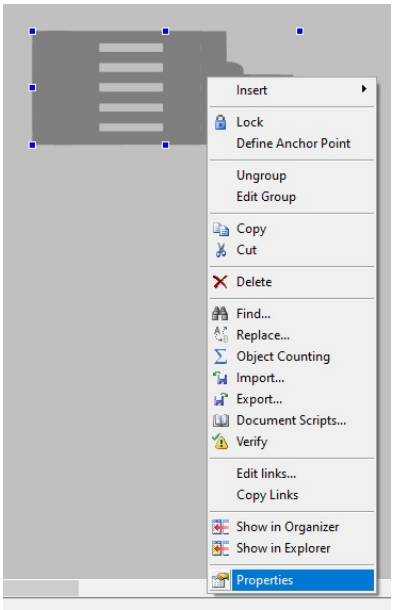
A representação desse objeto graficamente é mostrada na Figura 59, esse modelo apresenta de uma forma simples o motor, visando os conceitos de interface de alta performance. As propriedades desse XControl são vinculadas ao objeto que irá ser representado em tela, logo deve-se associar as propriedade Status e Falha do motor para que elas realizem a animação do objeto quando o mesmo for instanciado, sendo assim é necessário acessar as propriedades do objeto desenhado (Figura 60).

Figura 59 – *Design* do XControl motor



Fonte: Autor (2022)

Figura 60 – Acesso às propriedades de um objeto em tela

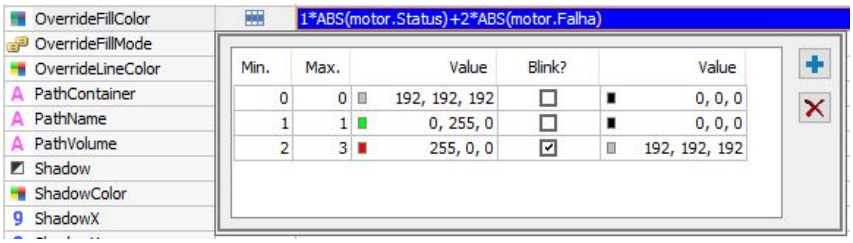


Fonte: Autor (2022)

As associações entre propriedades de objetos e valores ou informações são a principal maneira de animar as telas de um supervisório. Neste caso o que

será animado é a cor desde grupo de formas que representam o motor. Quando a propriedades Status estiver em verdadeira, a cor será Verde, e quando estiver falsa será cinza. A propriedade Falha em verdadeira do motor irá fazer com que a sua cor mude para vermelho e fique alternando com cinza, para chamar a atenção de quem está monitorando esse equipamento via supervisório. A associação para realizar essa animação é mostrada na Figura 61 abaixo.

Figura 61 – Associação para animação do motor



Fonte: Autor (2022)

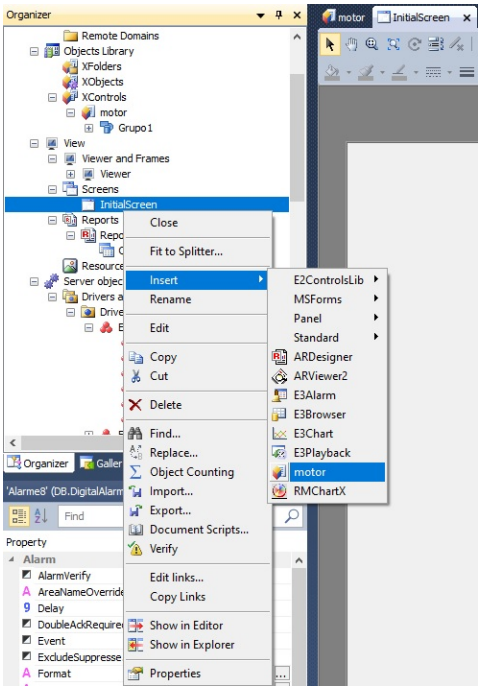
A associação utilizada foi do tipo *tabela*, ela calcula o valor gerado na expressão da fonte, no caso, ***1*ABS(motor.Status)+2*ABS(motor.Falha)***, onde a função ABS() transforma um valor Verdadeiro em 1 e um falso em 0, e faz correlação com os valores inseridos na tabela. Quando o valor da expressão for 0, o motor estará cinza, com valor 1 o motor ficará verde e entre 2 e 3 ele piscará em vermelho alertando uma falha no motor.

4.7.1.1.1 Configurando Motor no supervisório

Com o XControl do motor finalizado é possível adicionar o objeto no supervisório. Na tela inicial é possível adicionar em tela o motor (Figura 62).

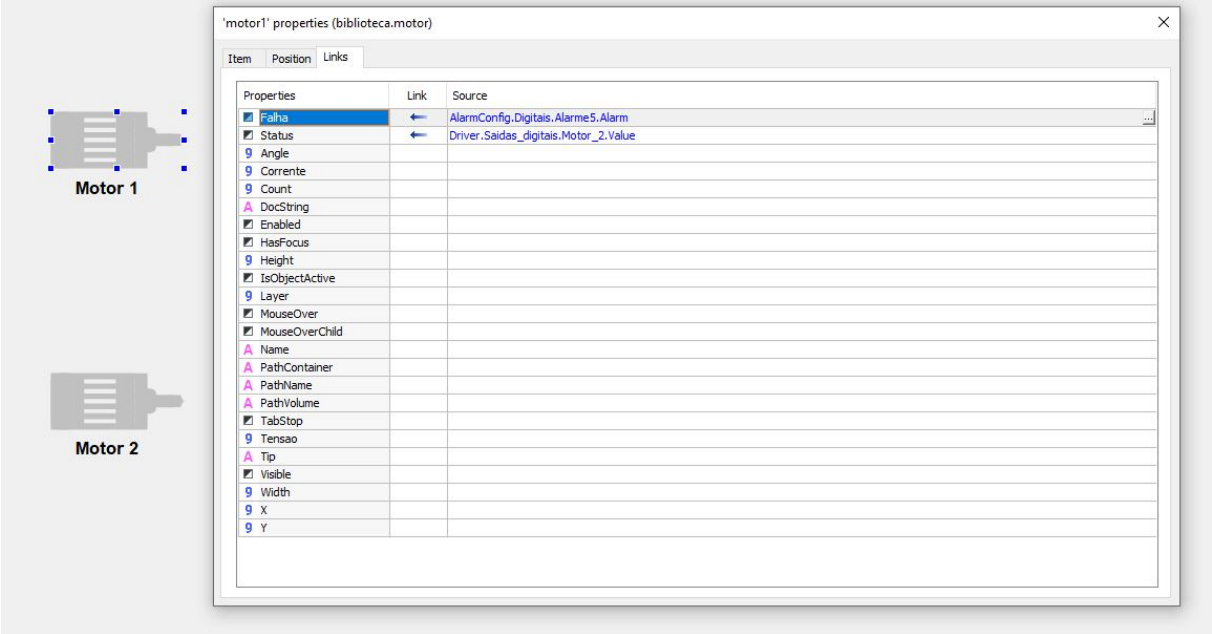
As propriedades criadas no XControl motor precisa ser vinculada aos tags que serão os valores enviados do simulador para o supervisório, sendo assim a falha irá ser o alarme 4 para o motor 1 e o alarme 5 para o motor 2, e o status de cada motor será definido pela saídas digitais do dispositivo (Figura 63).

Figura 62 – Adição motor em tela



Fonte: Autor (2022)

Figura 63 – Associação motor e tags simuladas



Fonte: Autor (2022)

4.7.1.2 Classe Barra de Alarme

Essa classe representará um modo de visualizar uma variável que possui um alarme analógico associado à ela. Ela mostra os limites de cada condição do alarme analógico e define a cor conforme a severidade da condição. O indicador de valor atual

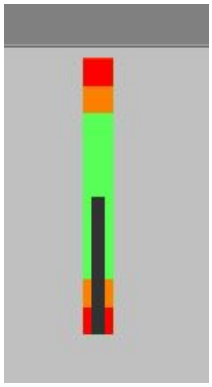
serve além de mostrar em que valor a variável se encontra, mostra o quão próximo a variável está de entrar em alguma condição de alarme. Essa metodologia é baseada na IHM de alta performance, além de mostrar apenas o valor o usuário tem uma percepção se o valor monitorado está dentro da faixa ou próximo dos limites.

As propriedades dessa classe são:

- Alarme - *AnalogAlarmSource*: fonte de alarme analógico instanciado no projeto. Ao associar essa propriedade da classe, ela tem acesso às propriedades do objeto de alarme analógico.
- Maximo - *Double* - valor máximo que a fonte pode assumir, essa propriedade é usada para realizar a proporção que as barras serão preenchidas
- Minimo - *Double* - valor mínimo que a fonte pode assumir, essa propriedade é usada para realizar a proporção que as barras serão preenchidas
- Valor - *Double* - valor associado à variável da fonte do alarme.

Na Figura 64 é apresentado como a classe *barra_alarme* ficou visualmente na biblioteca

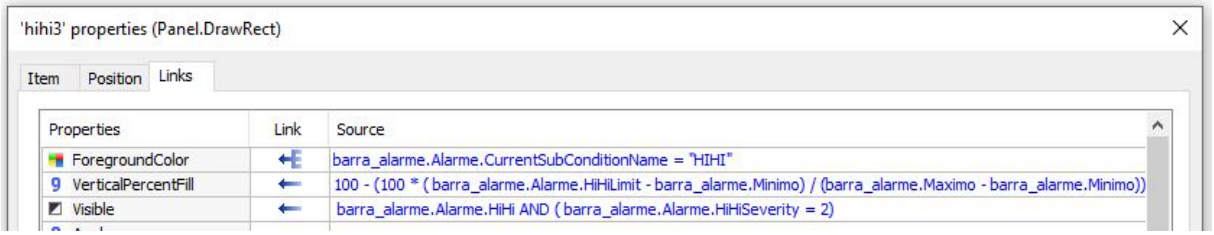
Figura 64 – *Design* do XControl barra_alarme



Fonte: Autor (2022)

A classe possui diversas barras retangulares internas sobrepostas que são configuradas conforme o alarme associado como fonte de informação. As propriedades visuais das barras que são animadas são mostradas na Figura 65 e são realizadas associações para definir o tamanho de cada barra proporcionalmente. Primeiramente a barra verde é a zona onde o alarme não possui nenhuma condição. As extremidades representam as condições HiHi(topo) e LoLo(base) do alarme, e consequentemente só estarão visíveis se forem condições habilitadas no alarme fonte. As barras intermediárias representam as condições Hi e Lo do alarme fonte. As cores que as barras de condições de alarmes assumem leva em conta a severidade da condição em questão. A Figura 65 exemplifica este parágrafo utilizando a barra do alarme HiHi com severidade Low(2).

Figura 65 – Associação da barra com as propriedades do XControl

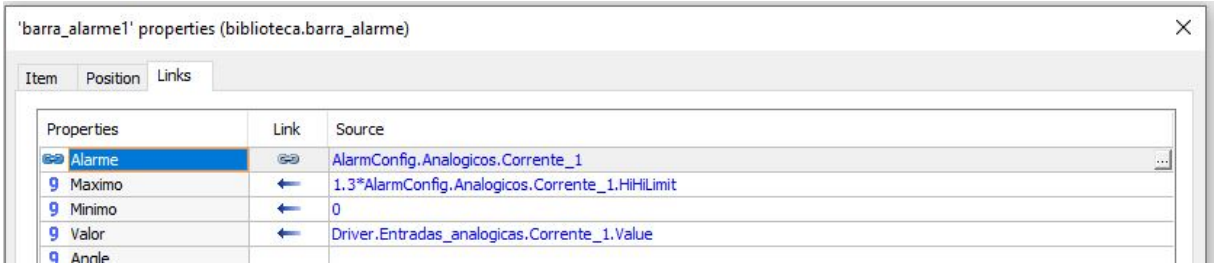


Fonte: Autor (2022)

4.7.1.3 Configurando classe barra_alarme no supervisório

Todos os alarmes analógicos do supervisório irão ser representados por uma barra de alarme na tela principal da aplicação. Como exemplo, abaixo é apresentado na Figura 66 as associações feitas após instanciar a barra de alarme em tela, ela representa o alarme analógico da corrente do motor 1. As demais barras seguem esta configuração utilizando o valor mínimo como 0 e o valor máximo como aproximadamente 30% a mais do que o valor limite da condição mais alta.

Figura 66 – Associação do barra_alarme com as variáveis



Fonte: Autor (2022)

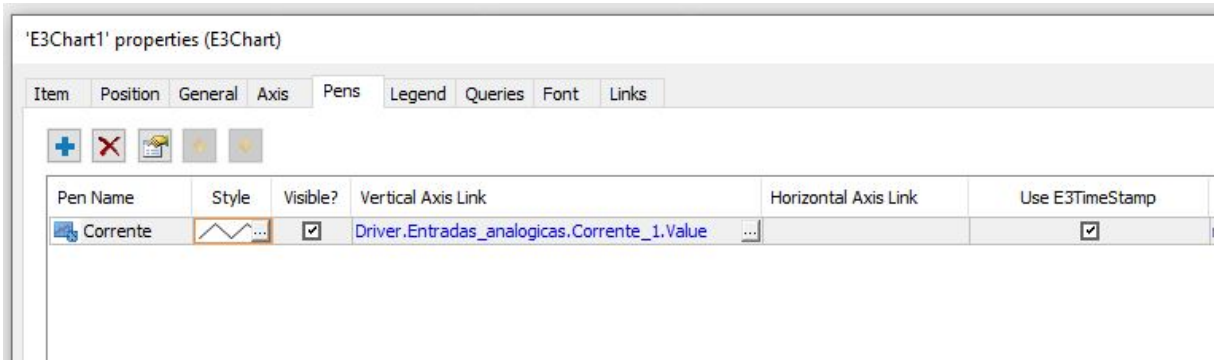
4.8 GRÁFICOS

Os gráficos E3Charts são nativos do Elipse e podem exibir de três maneiras os dados de forma histórica, em tempo real ou mista. Os dados históricos, são dados buscados nos bancos de dados por meio das consultas (*query*). Os dados de tempo real são vinculados diretamente as variáveis e mostrado em tempo real. Os dados mistos apresentam-se no gráfico com a parte relacionada aos registros que já ocorreram (histórico) e os que estão acontecendo no momento (tempo real).

Para a tela principal do supervisório serão utilizados gráficos para compor as barras de alarmes, possibilitando uma melhor visualização do comportamento das variáveis monitoradas, portanto, será um gráfico em tempo real. O gráfico será da altura da barra e os limites verticais, que são as propriedades *VerScaleBegin* e *VerScaleEnd* serão associados diretamente com as propriedades *Maximo* e *Minimo* da barra_alarme. As linhas de valor no gráfico são chamadas de penas no Elipse. A pena de cada

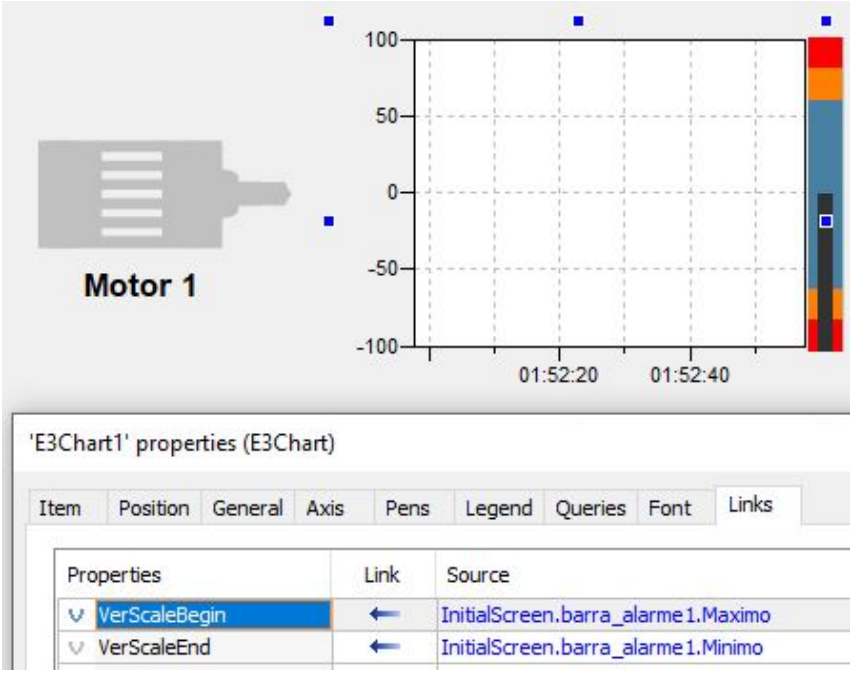
gráfico será, no eixo Y,a variável associada ao alarme e o tempo atual do sistema será associado ao eixo X da pena, como é mostrado na Figura 67. Abaixo, na Figura 68, é apresentada a configuração da associação (*links*) do gráfico em relação a barra de alarme da corrente do motor 1.

Figura 67 – Pena de tempo real



Fonte: Autor (2022)

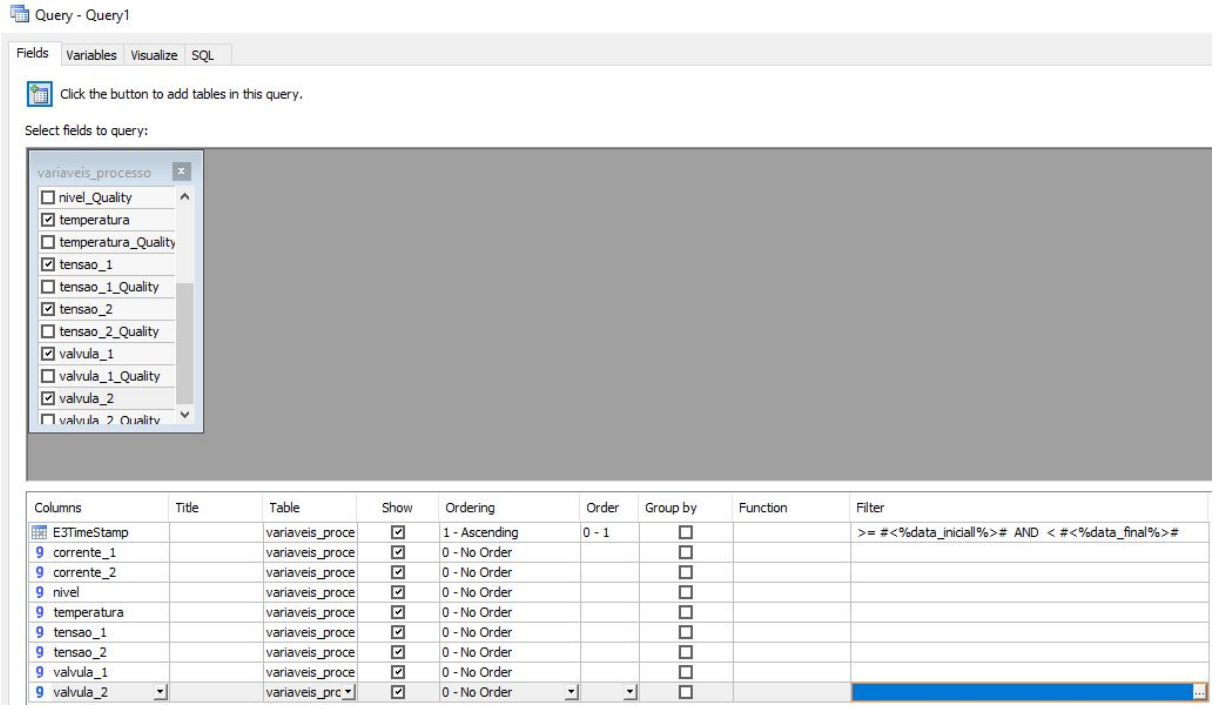
Figura 68 – Associação do gráfico em relação a barra de alarme



Fonte: Autor (2022)

O gráfico histórico será criado em uma tela separado para realizar análise mais detalhada dos valores de cada variável salva no banco de dados. Assim como em um relatório, deve-se configurar uma consulta para realizar a busca dos dados salvos. Será utilizada a tabela criada onde os valores das variáveis do sistema serão salvos com o filtro de data, a configuração da consulta encontra-se na Figura 69.

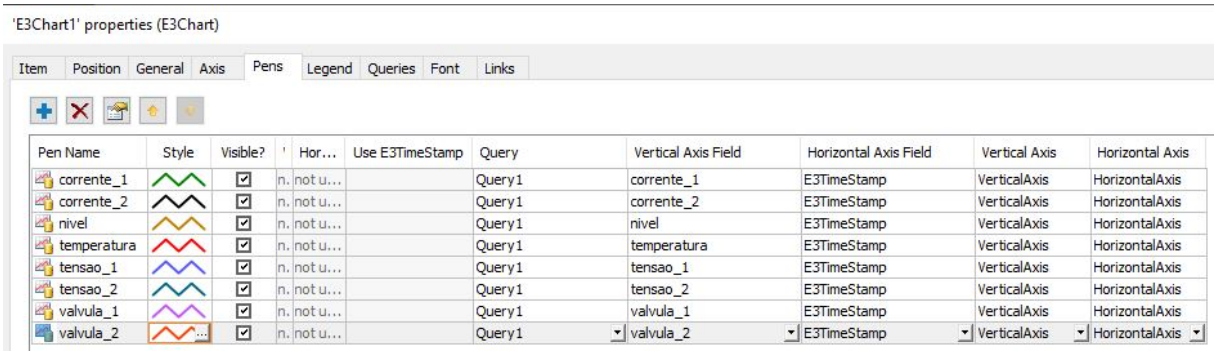
Figura 69 – Configuração da consulta gráfico histórico



Fonte: Autor (2022)

Na configuração de uma pena histórica deve-se associar a consulta em que ela é retornada e a variável para o eixo Y e para o eixo X. As penas históricas terão como valor no eixo X o tempo salvo no banco de dados. A Figura 70 ilustra a configuração de todas as penas utilizadas no gráfico histórico.

Figura 70 – Penas do gráfico histórico



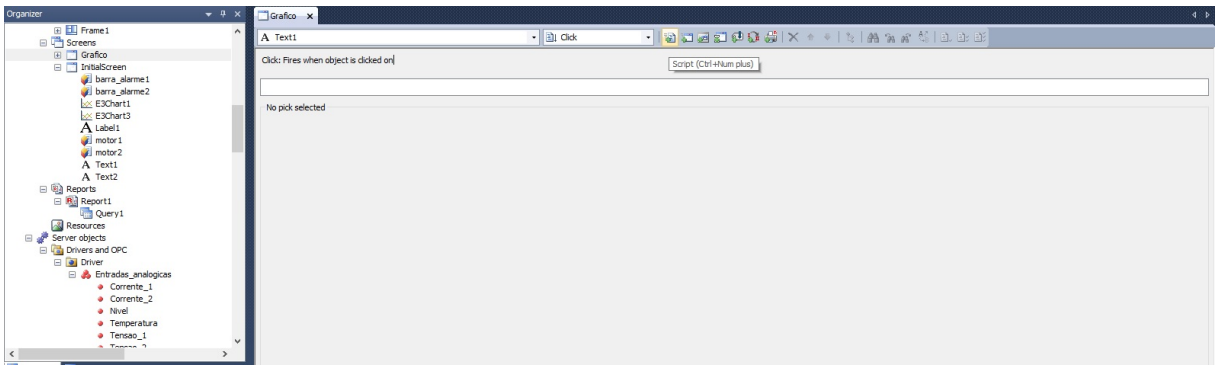
Fonte: Autor (2022)

4.8.1 Configuração da consulta do gráfico

O gráfico histórico deve ser interativo, ou seja, o usuário deve conseguir manipular os valores dos filtros de data. Nessa seção será mostrado como desenvolver objeto para alterar as consultas do gráfico em tempo de execução, utilizando o recurso de *Scripts* do Elipse.

O objeto *SetPoint* deve ser inserido na tela do gráfico histórico para que seja a entrada das datas que o usuário irá pesquisar na consulta. Ao clicar duas vezes sobre o objeto sua página de *scripts* é aberta (Figura 71).

Figura 71 – Tela de *script* do objeto *SetPoint*

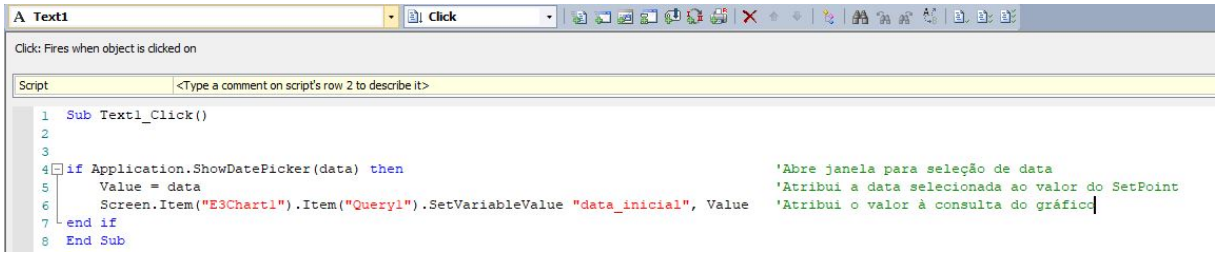


Fonte: Autor (2022)

Todos os objetos podem ter *scripts*, porém alguns objetos possuem *triggers* ou gatilhos específicos para os *scripts* serem executados, neste caso o objeto *SetPoint* possui o trigger *Click*, que executa o *script* no momento em que o objeto for clicado.

O *script* executado ao clicar no *SetPoint* da data inicial é mostrado na figura abaixo (Figura 72). Ao ser executado ele abre uma janela para o usuário selecionar a data em que deseja realizar a pesquisa dos dados e após isso ele insere a data no filtro da consulta. O mesmo script deve ser adicionado no *SetPoint* referente a data final de consulta, porém a função que realiza o *input* na consulta deve ter sua variável alterada para *data_final*.

Figura 72 – *Script* para configurar consulta do gráfico histórico

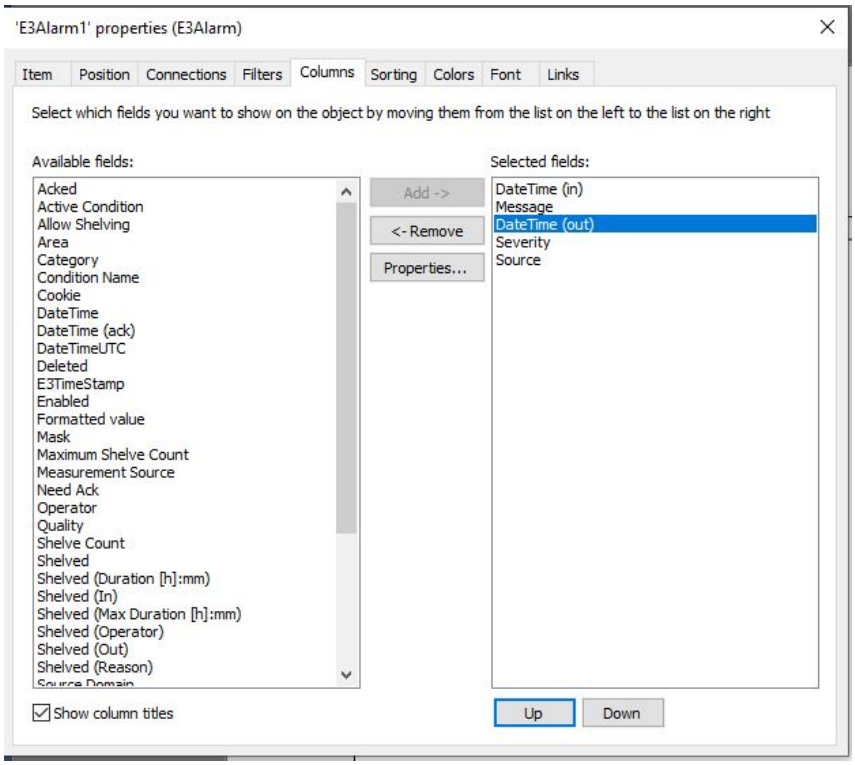


Fonte: Autor (2022)

4.9 TELA DE ALARMES

O software da Elipse conta com um objeto nativo para listagem de alarmes ativos do sistema, o *E3Alarm*. Ele é configurável assim como o servidor de alarmes e pode ser configurado para exibir os mesmos atributos utilizados para o relatório de alarmes. A configuração utilizada neste projeto é mostrado na Figura 73

Figura 73 – Configuração *E3Alarm*



Fonte: Autor (2022)

Na tela de alarmes também será adicionado função para impressão do relatório de alarmes criado anteriormente. Do mesmo modo em que é filtrado os valores de um gráfico será filtrado os alarmes do sistema, a única diferença será a adição de um botão para executar a função de imprimir o relatório abaixo o *script* utilizado para realizar essa função é mostrado na Figura 74

Figura 74 – *Script* para impressão de relatório

Fonte: Autor (2022)

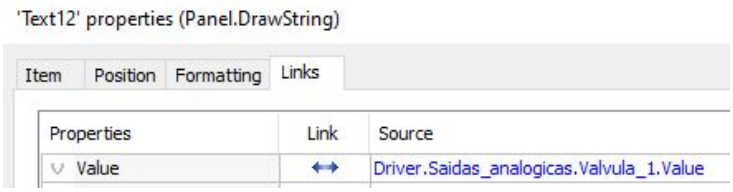
4.10 OBJETOS PADRÃO DO SUPERVISÓRIO

Os demais objetos utilizados nessa aplicação são nativos do Elipse, e estão presentes em todos os softwares de desenvolvimentos de supervisório por se tratarem de objetos padrões para visualização de variáveis ou objetos para realização de comandos.

O objeto *display* é utilizado para mostrar o valor da variável associada à ele, e pode ser formatado para exibir unidades de medida, configurando sua propriedade *Format*, por exemplo, exibir uma variável inteira seguida da unidade de medida Ampère, o campo *Format* deve está configurado como **0 "A"**. Para configurações com duas casas decimais é utilizado **0.00 "A"**. A variável é lida como inteira sem vírgula, logo neste projeto será utilizado o primeiro formato.

A realização de comandos através do supervisório se dá por meio de escritas nas tags do driver, sendo assim deve-se associar o objeto que fará escrita nos tags. A variável da válvula proporicional foi definida como uma saída e o supervisório que irá comandar a abertura dessa válvula através do objeto SetPoint. Para isso deve se vincular a propriedade *Value* do *setpoint* para que ela tenha um associação bidirecional com a tag que representa o valor da válvula proporcional, como é mostrado na Figura 75

Figura 75 – Associação bidirecional para escrita em tag

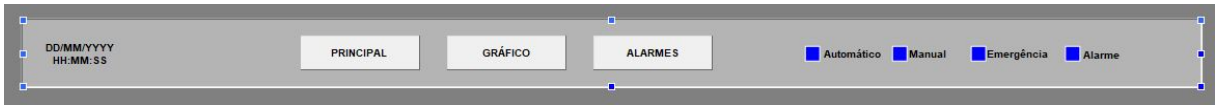


Fonte: Autor (2022)

4.11 NAVEGAÇÃO ENTRE TELAS

O acesso às telas do supervisório se dará por meio de um menu fixo no divisor superior da aplicação. A tela de menus contará com o botões para o acesso a cada tela desenvolvida na aplicação, além de conter um relógio do sistema e algumas informações de status do sistema. A disposição da tela do menu é apresentada na Figura 76.

Figura 76 – *Layout* tela de menus

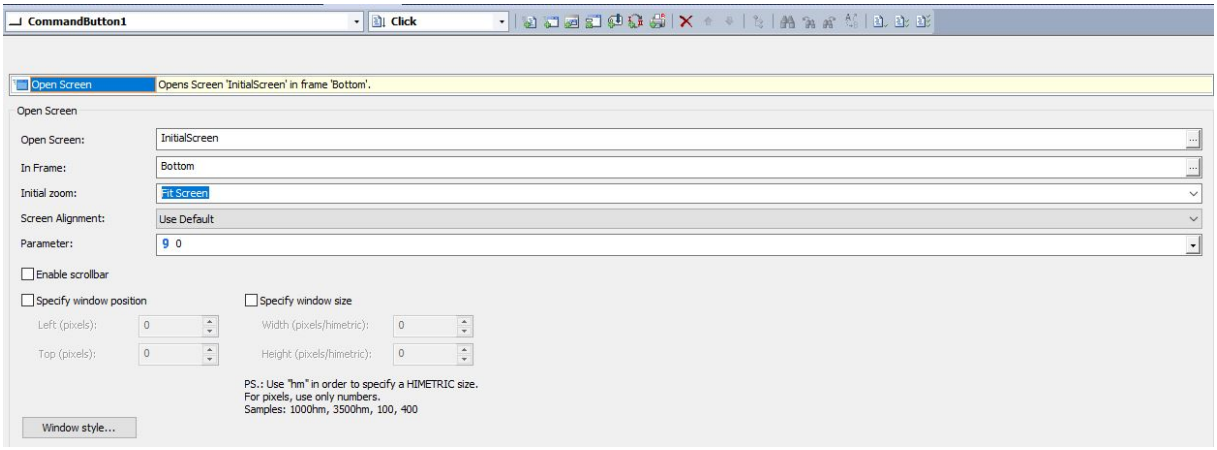


Fonte: Autor (2022)

Cada botão executará um *script* para abrir a tela no divisor inferior, o *script* é

mostrando abaixo na Figura 77.

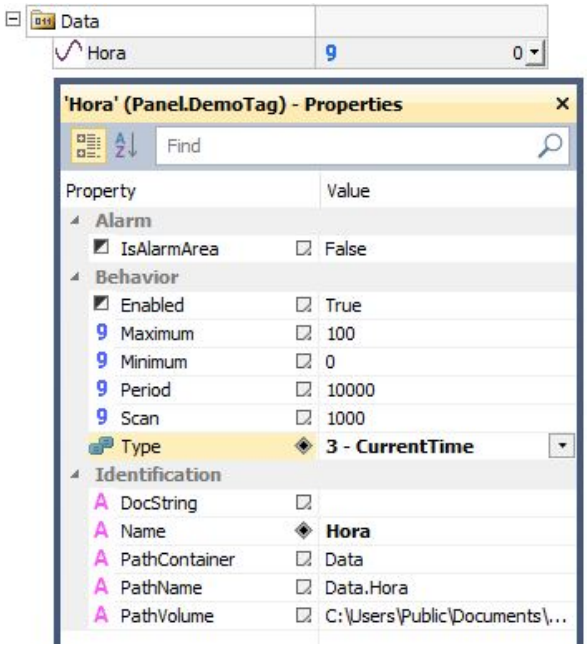
Figura 77 – *Script* em botão para abertura de tela



Fonte: Autor (2022)

A criação do relógio do sistema é feita através da adição de uma tag dentro do servidor de dados. O tipo da tag a ser adicionado é *Demo Tag* e ela deve ter sua propriedade *Type* alterada para *3 - Current Time* (Figura 78). Assim ela assume em seu valor o tempo da máquina que está executando o supervisório. Por fim, associa-se a propriedade *Value* do texto em tela com a propriedade *Value* da *Demo Tag* criada para representa o relógio da aplicação.

Figura 78 – Configuração *Demo Tag* para relógio do sistema

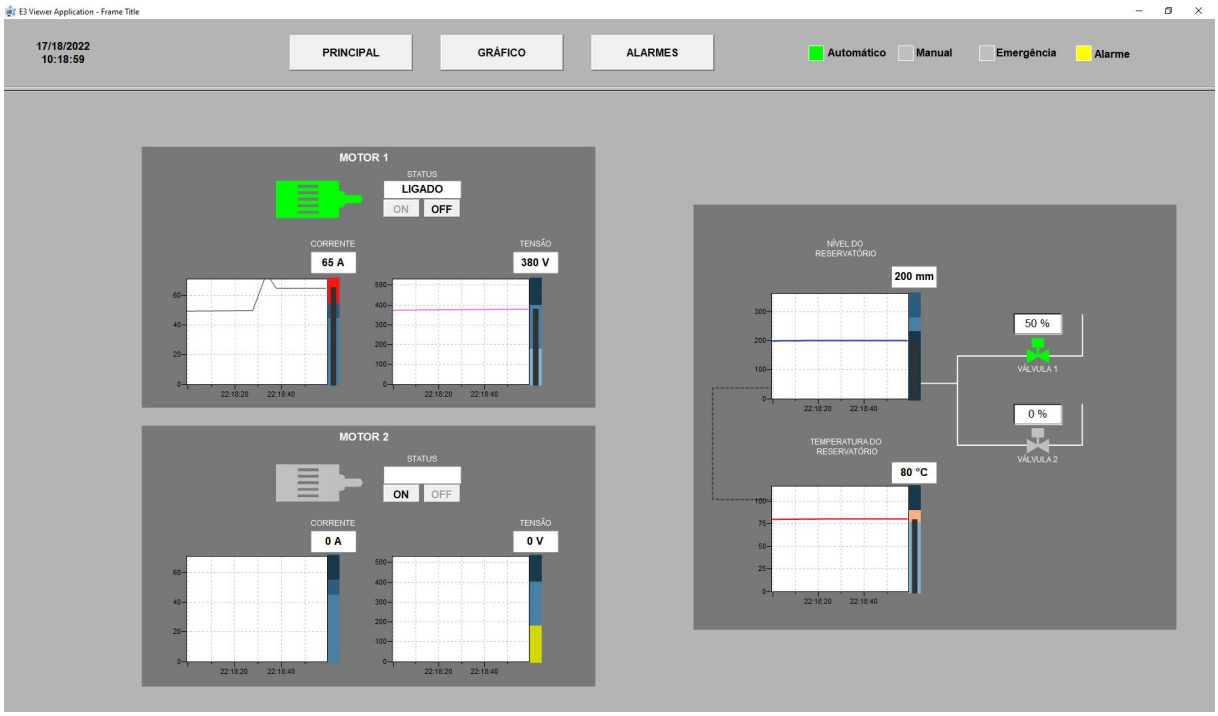


Fonte: Autor (2022)

4.12 EXECUÇÃO DO SUPERVISÓRIO

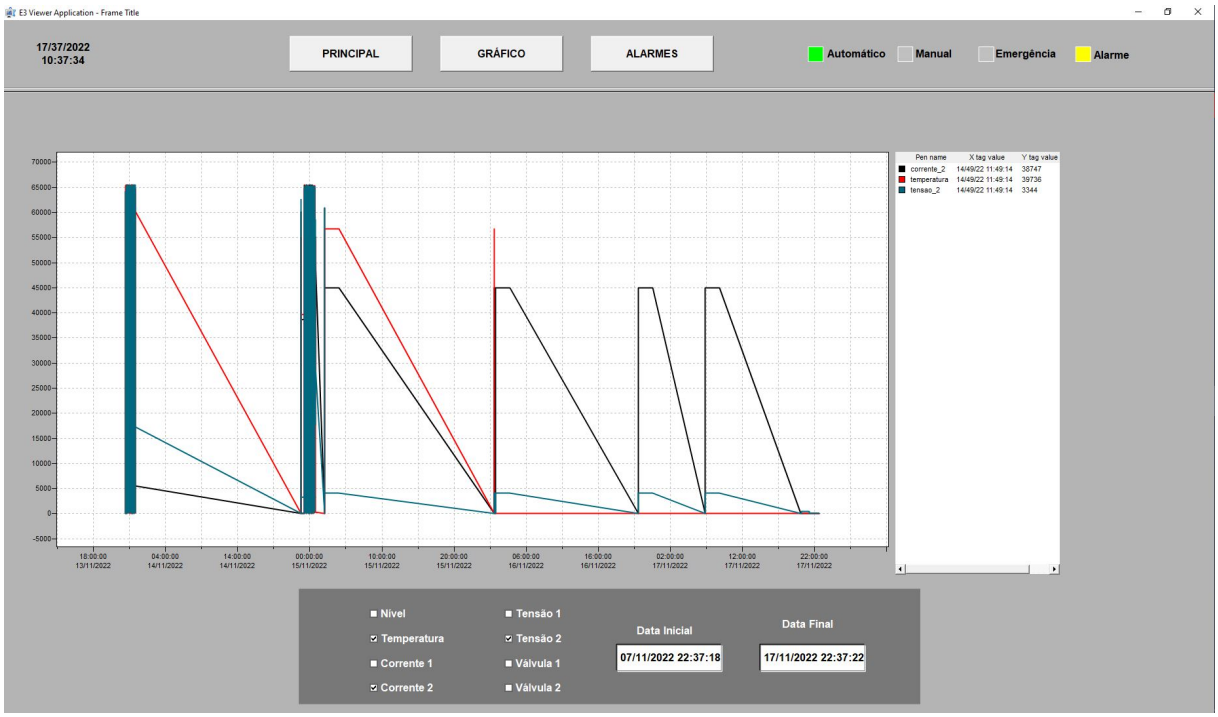
Para inicializar a aplicação supervisório deve clicar no botão *"Saves and Runs the Domain"*, ao executar a tela principal irá abri e poderá ser feito o monitoramento e controle da aplicação. O simulador deve estar rodando e conectado para que as variáveis seja atualizadas. Na Figura 79 é apresentada a tela principal do supervisório com valores sendo simulados pelo software, na Figura 80 é apresentada a tela do gráfico onde é feita a leitura dos vales do banco de dados e na Figura 81 é mostrada a tela de alarmes com os alarmes que estão sendo simulados no sistema.

Figura 79 – Tela principal em tempo de execução



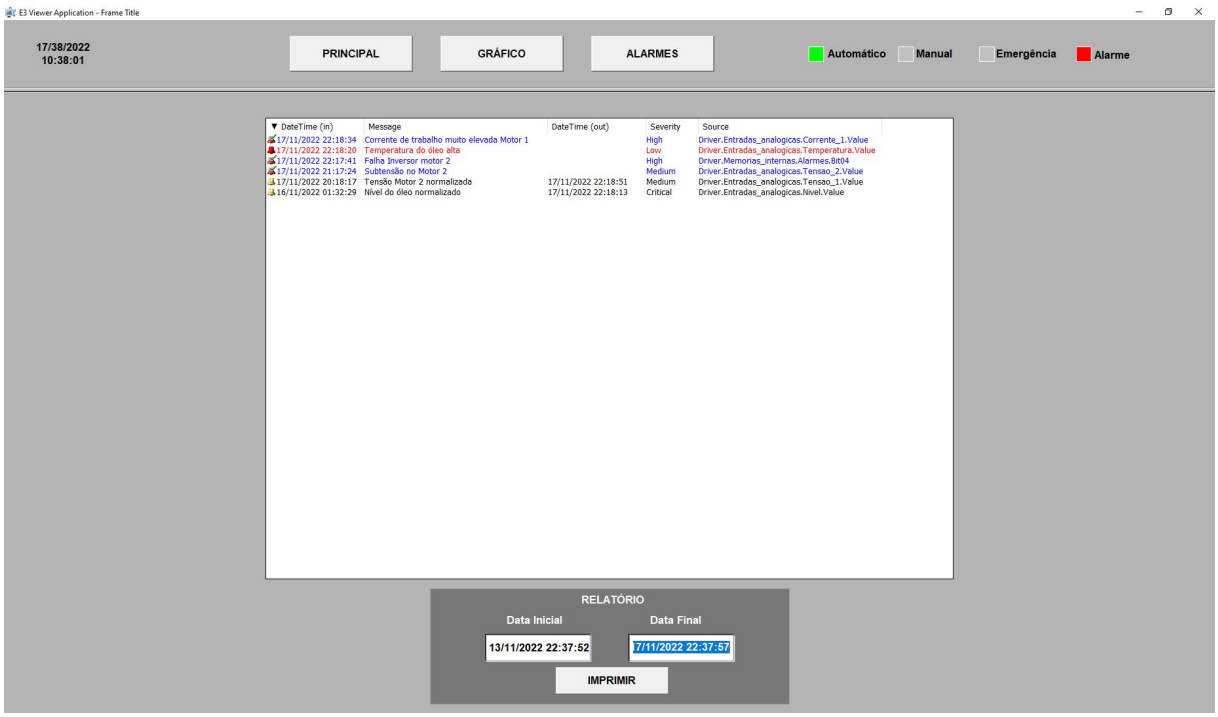
Fonte: Autor (2022)

Figura 80 – Tela do gráfico histórico em tempo de execução



Fonte: Autor (2022)

Figura 81 – Tela de alarmes em tempo de execução



Fonte: Autor (2022)

5 CONCLUSÕES

Este trabalho teve como proposta gerar uma documentação para o desenvolvimento de uma aplicação supervisória genérica, para que possa servir de apoio à acadêmicos ou profissionais da indústria.

No capítulo 2 foram descritas as possíveis arquiteturas de automação industrial e onde os sistemas supervisórios se encontravam nela, mostrando a importância desses sistemas em linhas automatizadas. A apresentação dos recursos que um sistema supervisório possui também foram mostrados, descrevendo a utilidade de cada um.

O *software* utilizado o desenvolvimento da aplicação supervisória foi o ElipseE3, que conta com uma interface intuitiva e possibilita executar a aplicação em modo *demo* além de ser um dos supervisórios mais utilizados comercialmente.

O trabalho apresentou todas as etapas de desenvolvimento de uma aplicação supervisória. Iniciando com a configuração de um dispositivo para realizar a simulação dos dados para controle e monitoramento. A configuração de uma interface de comunicação dentro do supervisório com este dispositivo de simulação, aplicando os conceitos de protocolo de comunicação industrial ModBus. A criação de um banco de dados e a definição dos dados que iriam ser armazenados para posteriores consultas, essas que também foram desenvolvidas e exemplificadas com filtros de busca. Por fim, o desenvolvimento do ambiente para o usuário final do supervisório as telas e objetos da aplicação que são monitorados e controlados nesses sistemas. Todos os passos foram descritos de forma que possam ser utilizados como material de consulta para acadêmicos ou profissionais da área.

As ferramentas de criação de formas e objetos *Elipse* são limitadas, por se tratar de sua versão gratuita, gerando uma certa dificuldade em criar desenhos com boa aparência. A habilidade em *softwares* de desenho como *InkScape* pode facilitar no desenvolvimento de aplicações, uma vez que é possível importar imagens vetoriais para dentro do supervisório.

A Validação do supervisório foi dada com a execução da aplicação e a integração com o simulador ModBus enviando valores randômicos e também com o sistema supervisório escrevendo os valores nas memórias do simulador. O banco de dados criado para o armazenamento dos dados também foi validado utilizando a tela de gráfico. Os alarmes também foram validados pela tela de alarmes, assim como o relatório de alarmes desenvolvido foi validado. As animações também foram verificadas realizando todas a variações de entradas para cada objeto.

Para trabalhos futuros considera-se utilizar os objetos desenvolvidos na

biblioteca e realizar a integração com um dispositivo com lógica de funcionamento, obtendo valores mais próximos de uma aplicação real, uma vez que não é possível configurar limite para os valores randômicos do simulador utilizado.

REFERÊNCIAS

BRANQUINHO, M. A. **Segurança de automação industrial e SCADA**. [S.l.]: Elsevier, 2014.

CLARKE, G.; REYNDERS, D. **Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems**. 1. ed. Burlington, Massachussets: Elsevier, 2004.

ELIPSE SOFTWARE. **ELIPSE E3**. 2015. Disponível em: https://www.elipse.com.br/produto/elipse-e3/. Acesso em: 24 set. 2022.

GOETZ, H. F. Metodologia para desenvolvimento de ihms de alta performance visual. 2019. Disponível em: https://kb.elipse.com.br/metodologia-para-desenvolvimento-de-ihms-de-alta-performance-visual/. Acesso em: 18 set. 2022.

ISA. Introdução à norma isa-101: Interfaces homem-máquina. São Paulo, 2020. Disponível em: http://isasp.org.br/wp-content/uploads/2020/01/ISA-101-III-SimpÃşsio-ISA-SÃčo-Paulo-Sabesp-Nov2016.pdf. Acesso em: 19 set. 2022.

LAMB, F. **Automação Industrial na Prática**. São Paulo: AMGH, 2015.

MAITELLI, A. L. **Apostila Controladores Lógicos Programáveis**. Rio Grande do Norte, 2003.

MORAES, C. C.; CASTRUCCI, P. L. **Engenharia de automação industrial**. Rio de Janeiro: LTC, 2007.

PAREDE, I. M. **Eletrônica: automação industrial**. São Paulo: Fundação Padre Anchieta, 2011. (6).

PESSOA, M. S. P. **Introdução à Automação: para cursos de engenharia e gestão**. 1. ed. Rio de Janeiro: Elsevier, 2014.

PINHEIRO, J. Introdução às redes de supervisão e controle. 2006. Disponível em: https://www.projetoderedes.com.br/artigos/artigo_redes_de_supervisao_e_controle.php. Acesso em: 09 set. 2022.

ROURE, M. Pirâmide da automação industrial. 2017. Disponível em: https://instrumentacaoecontrole.com.br/piramide-da-automacao-industrial/. Acesso em: 16 set. 2022.

SILVEIRA L.; LIMA, W. Q. Um breve históriooco conceitual da automação industrial e rede para automação industrial. Natal, 2003. Disponível em: https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho1/trabalho1_13.pdf. Acesso em: 14 set. 2022.

YAMAGUCHI, M. Y. **Sincronização das bases de tempos de clps distribuídos numa rede de automação de processo industrial**. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2006.