



CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA



Universidade Federal
de Campina Grande

ALEX FERNANDES FIGUEIRÊDO

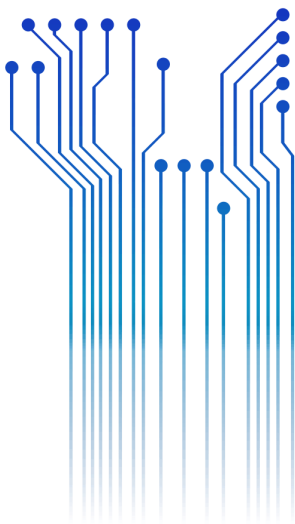


Centro de Engenharia
Elétrica e Informática

TRABALHO DE CONCLUSÃO DE CURSO
ESPECIFICAÇÃO ARQUITETURAL DO SIMULADOR SIMULIHM E
DESENVOLVIMENTO DE MÓDULO DE COMUNICAÇÃO COM SISTEMAS
SUPERVISÓRIOS



Departamento de
Engenharia Elétrica



Campina Grande
2016

ALEX FERNANDES FIGUEIRÊDO

ESPECIFICAÇÃO ARQUITETURAL DO SIMULADOR SIMULIHM E DESENVOLVIMENTO DE
MÓDULO DE COMUNICAÇÃO COM SISTEMAS SUPERVISÓRIOS

*Trabalho de Conclusão de Curso submetido à
Unidade Acadêmica de Engenharia Elétrica da
Universidade Federal de Campina Grande
como parte dos requisitos necessários para a
obtenção do grau de Bacharel em Ciências no
Domínio da Engenharia Elétrica.*

Área de Concentração: Controle e Automação

Orientador:

Professora Maria de Fátima Queiroz Vieira, Ph.D.

Campina Grande

2016

ALEX FERNANDES FIGUEIRÊDO

ESPECIFICAÇÃO ARQUITETURAL DO SIMULADOR SIMULIHM E DESENVOLVIMENTO DE
MÓDULO DE COMUNICAÇÃO COM SISTEMAS SUPERVISÓRIOS

*Trabalho de Conclusão de Curso submetido à
Unidade Acadêmica de Engenharia Elétrica da
Universidade Federal de Campina Grande
como parte dos requisitos necessários para a
obtenção do grau de Bacharel em Ciências no
Domínio da Engenharia Elétrica.*

Área de Concentração: Controle e Automação

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professora Maria de Fátima Queiroz Vieira, Ph.D.
Universidade Federal de Campina Grande
Orientadora, UFCG

Dedico este trabalho a todos que apoiaram e possibilitaram que eu chegasse a esse momento de conclusão de curso: família, amigos e professores.

AGRADECIMENTOS

Agradeço em especial a minha mãe, Edna Fernandes de Brito, e ao meu avô, Joaquim Fernandes de Brito, que são as minhas maiores inspirações como pessoa, passando-me os valores essenciais com os quais eu guio todas as minhas decisões.

Agradeço também a todos os professores com os quais tive oportunidade de absorver algum conhecimento durante toda minha formação, básico e superior. Agradeço em especial aos professores Paulo Barbosa e Maria de Fátima Queiroz Vieira os quais tem me acompanhado nos últimos anos de graduação, sendo a última minha orientadora neste trabalho de conclusão de curso.

Por último, e não menos importante, agradeço a todos os meus amigos e familiares, responsáveis por oferecer todo o apoio e sustentação necessária para enfrentar os incontáveis momentos de dificuldade, e com quem eu pude compartilhar as alegrias dos momentos de conquista.

“Mas, sejam fortes e não desanimem, pois o trabalho de vocês será recompensado”.

2 Crônicas 15:7.

RESUMO

As falhas humanas constituem uma das principais fontes de erro na operação de sistemas elétricos, fazendo com que as empresas do setor elétrico tenham investido cada vez mais na capacitação dos seus operadores. Dentre as diversas técnicas usadas no treinamento de operadores, destaca-se o uso crescente de simuladores que permitem submeter os operadores a cenários críticos e avaliar a capacidade dos mesmos em resolver estas situações. Neste contexto o LIHM (Laboratório de Interface Homem-Máquina) da Universidade Federal de Campina Grande desenvolveu a ferramenta para treinamento de operadores SimuLIHM. Por meio desta ferramenta os operadores podem ser submetidos a cenários de treinamento pré-configurados, e realizar o treinamento em um ambiente virtual com grande semelhança ao ambiente real de operação vivenciado nas empresas. Para atingir tal grau de realismo, o ambiente virtual deve permitir a interação dos operadores com a planta elétrica tanto por meio de painéis como através de telas de sistemas supervisórios. Dada a complexidade atingida pela ferramenta desenvolvida no LIHM, a manutenção e evolução do simulador é dificultada pela ausência de uma especificação arquitetural detalhada. Além disso, a versão atual da ferramenta é restrita à representação do sistema supervisório InTouch. Para minimizar os problemas citados, foram realizadas as seguintes atividades: a especificação em UML (Linguagem de Modelagem Unificada) da arquitetura de software do SimuLIHM e, a implementação de um módulo de comunicação que facilita a integração do simulador a diferentes sistemas supervisórios, além da implementação de um *driver* de comunicação com o protocolo OPC, padrão de comunicação de grande aceitação no meio industrial devido ao alto nível de interoperabilidade proporcionado. Este texto descreve a documentação e os módulos construídos ao longo do trabalho.

Palavras-chave: Treinamento de Operadores, Sistemas Supervisórios, SimuLIHM, Arquitetura de Software, Protocolo OPC.

ABSTRACT

Human errors are a major source of error in electrical systems operation, causing the electricity companies to increasingly invested in the training of its operators. Among the various techniques used in operator training, there is the increasing use of simulators that allow operators to submit to critical scenarios assessing their ability to solve these situations. In this context the LIHM (Laboratory of Human-Machine Interface) of the Federal University of Campina Grande developed a simulator training tool -SimuLIHM. With this tool operators are subject to preconfigured training scenarios, and train in a virtual environment with great similarity to real environment experienced in operating companies. To achieve such a degree of realism, virtual environment should enable the interaction of operators with the power plant both through panels and through supervisory systems screens. Given the complexity achieved by the tool developed in LIHM, its maintenance and evolution is hampered by the absence of a detailed architectural specification. In addition, its current version is restricted to the representation of a single supervisory system, the InTouch. To minimize these problems, the following activities were carried out during this project: the specification in UML (Unified Modeling Language) of SimuLIHM software architecture and implementation of a communication module that facilitates the integration of this simulator to different supervisory systems, as well as implementation of a communication driver with the OPC protocol, a widely accepted communications standard in industry due to the high level of interoperability provided. This text describes the documentation and the modules built throughout the work.

Keywords: Operators Training, Supervisory Systems, SimuLIHM, Software Architecture, OPC Protocol

LISTA DE ILUSTRAÇÕES

Figura 1: Arquitetura de software do SAGE	19
Figura 2: Diagrama unifilar de uma subestação representada no supervisório SAGE	20
Figura 3: Ambiente de desenvolvimento do IWS.....	22
Figura 4: Arquitetura OPC	23
Figura 5: Arquitetura do SimuLIHM.....	28
Figura 6: Ambiente virtual do SimuLIHM.....	29
Figura 7: Interação entre SimuLIHM e sistema supervisório.....	29
Figura 8: Especificação arquitetural da plataforma de treinamento de operadores	32
Figura 9: Diagrama de implantação.....	33
Figura 10: Casos de uso para o Módulo Operador	33
Figura 11: Casos de uso para o Módulo Tutor	35
Figura 12: Diagrama de casos de uso do módulo supervisório	38
Figura 13: Diagrama de atividades para o módulo SupervisorioTutor.....	39
Figura 14: Diagrama de atividades para o módulo SupervisorioTreinando	40
Figura 15: Especificação arquitetural do módulo de comunicação com os sistemas supervisórios	41
Figura 16: Tela de apresentação do supervisório	43
Figura 17: Tela de programas do supervisório	44
Figura 18: Tela de alarmes do supervisório.....	44
Figura 19: Sinótico do supervisório.....	45
Figura 20: Janela de ação para os equipamentos do supervisório	47
Figura 21: Configuração da comunicação OPC	49
Figura 22: MatrikonOPC Explorer 5.0	49
Figura 23: Toolbox OPC Power Server 5.21	50
Figura 24: Interface implementada pelo driver de comunicação OPC	51
Figura 25: Diagrama de implantação para avaliação dos resultados	52
Figura 26: Cenário de teste para o módulo de comunicação com o supervisório	53

LISTA DE TABELAS

Tabela 1: Descrição de componentes do Módulo Operador.....	34
Tabela 2: Descrição dos componentes do Módulo Tutor	36
Tabela 3: Descrição dos sistemas externos.....	37
Tabela 4: Símbolos do sinótico	46
Tabela 5: Atributos da classe Equipamento.....	46
Tabela 6: Atributos da classe Comando	47
Tabela 7: Atributos da classe Login	47

LISTA DE ABREVIATURAS E SIGLAS

CHESF	Companhia Hidroelétrica do São Francisco
LIHM	Laboratório de Interface Homem-Máquina
SimuLIHM	Ambiente de Virtual de Simulação para Treinamento de Operadores
SAGE	Sistema Aberto de Gerenciamento de Energia
CGD II	Campina Grande II
IHM	Interface Homem-Máquina
CLP's	Controladores Lógicos Programáveis
SCADA	<i>Supervisory Control and Data Acquisition</i>
PIMS	<i>Plant Information Management System</i>
MES	<i>Manufacturing Execution Systems</i>
ERP	<i>Enterprise Resource Planning</i>
SOA	Arquitetura Orientada a Serviços
EMS	Energy Management System
Cepel	Centro de Pesquisas de Energia Elétrica
ONS	Operador Nacional do Sistema Elétrico
CAG	Sistema de Controle de Geração
BDH	Sistema de Armazenamento Histórico de Longo Prazo
IWS	<i>InduSoft Web Studio</i>
OEE	Eficiência Total de Equipamentos
BD	Banco de Dados
OLE	<i>Object Linking and Embedding</i>
OPC	OLE para Controle de Processos
OPC DA	OPC Data Access
DCOM	<i>Distributed Component Object Model</i>
OTS	<i>Operator Training Simulator</i>
EPRI	<i>Electric Power Research Institute</i>
ASTRO	Ambiente Simulado para Treinamento de Operadores
TCP/IP	<i>Transmission Control Protocol/ Internet Protocol</i>

DDE	<i>Dynamic Data Exchange</i>
UML	Linguagem de Modelagem Unificada
IDE	Ambiente de Desenvolvimento Integrado
JNI	<i>Java Native Interface</i>
DLL	<i>Dynamic-Link Library</i>
CPU	Unidade Central de Processamento

SUMÁRIO

1	Introdução.....	14
1.1	Motivação	14
1.2	Objetivos	15
1.3	Metodologia	16
1.4	Estrutura do Relatório	16
2	Fundamentação Teórica.....	17
2.1	Sistemas Supervisórios	17
2.2	Padrão OPC.....	22
2.3	Treinamento de Operadores em Subestações Elétricas	24
3	Simulador SimuLIHM.....	27
4	Desenvolvimento.....	31
4.1	Especificação Arquitetural	31
4.2	Implementação de Telas do SAGE no Supervisório Indusoft.....	42
4.3	Implementação do Módulo de Comunicação com o Supervisório.....	48
5	Resultados	52
6	Conclusão	55
	Referências	57
	APÊNDICE A – Código JAVA para o Driver de Comunicação OPC	59

1 INTRODUÇÃO

O trabalho de supervisão e controle de sistemas elétricos exige uma resposta rápida e correta aos eventos que ocorrem no sistema. Essa tarefa é dificultada pela grande quantidade de informações audiovisuais às quais os operadores são submetidos durante as ocorrências.

Devido as constantes mudanças na configuração do sistema, ou ao deslocamento de operadores para diferentes subestações, é comum que estes não apresentem total entendimento sobre o comportamento e características do sistema o qual estão operando.

O erro humano durante a operação de sistemas elétricos pode resultar da complexidade das tarefas realizadas, e pode ocasionar perdas materiais, ambientais, ou mesmo de vidas humanas. Em um estudo de relatórios de falhas do sistema, ocasionadas pelo erro humano, na empresa CHESF – Companhia Hidroelétrica do São Francisco, realizado por (NASCIMENTO NETO, 2010) e citado em (TORRES FILHO, 2011), foi verificado que 20% das falhas do sistema resultam do erro humano.

Para minimizar as causas dos problemas relatados neste contexto de operação, faz-se uso de simuladores para treinamento de operadores em diferentes cenários. Dentre estes, destacam-se os simuladores com representações em realidade virtual, os quais expõem os operadores a situações realistas e geram competências na operação do sistema real de forma rápida e permanente.

1.1 MOTIVAÇÃO

Os centros de supervisão e controle de energia elétrica, assim como ocorreu em outras áreas da indústria na última década, evoluíram da operação através de painéis de controle distribuídos ao longo do sistema de produção para os sistemas supervisórios, que coletam dados do processo, formatam esses dados e os disponibilizam em terminais de computador (MAMEDE FILHO, 2007).

Muitas empresas do setor elétrico adotam simuladores para o propósito de treinamento. Entretanto, nesses simuladores, o supervisório não está associado a uma representação do ambiente da subestação, nem aos painéis de controle. Isso significa que,

numa situação de treinamento, ao atuar apenas no supervisão, o operador não pode perceber os alarmes sonoros e visuais que são disparados nos painéis, nem tão pouco interagir com eles (TORRES FILHO, 2011).

Problemas de comunicação do supervisão com os equipamentos da subestação, ou mesmo a incapacidade de atuação remota em equipamentos mais antigos, pode exigir que em situações reais os operadores precisem atuar diretamente sobre os equipamentos da planta tanto através do sistema supervisão quanto através dos painéis de controle.

Neste contexto, o LIHM (Laboratório de Interface Homem-Máquina) desenvolveu um ambiente de virtual de simulação para treinamento de operadores, SimuLIHM. Neste ambiente de simulação o operador pode se deslocar pela sala de controle da subestação e atuar sobre a planta através de representações virtuais de painéis de controle ou do sistema supervisão integrado ao sistema.

Em (TORRES FILHO, 2011) é detalhada a integração de um sistema supervisão, desenvolvido para integrar o InTouch, ao ambiente de simulação. Devido à complexidade e ao trabalho exigido na implementação das plantas de subestações para o sistema supervisão, foi realizada apenas uma representação parcial da subestação CGD II (Campina Grande II), da CHESF.

Este trabalho é motivado pela necessidade de fornecer uma especificação arquitetural detalhada e expandir os recursos do SimuLIHM para permitir a integração com diferentes sistemas supervisórios e para facilitar a evolução e manutenção do simulador.

1.2 OBJETIVOS

O objetivo principal deste trabalho consiste em realizar uma especificação arquitetural detalhada da ferramenta SimuLIHM, e implementar um módulo de comunicação que permita sua integração a diferentes sistemas supervisórios. Para avaliação da solução proposta, ainda devem ser implementadas telas de um sistema supervisão, com base nas telas no SAGE (Sistema Aberto de Gerenciamento de Energia).

1.3 METODOLOGIA

A realização do trabalho foi dividida nas seguintes etapas, as quais foram projetadas em um cronograma apresentado na proposta do projeto:

- Etapa 1: Revisão bibliográfica sobre os conceitos básicos de sistemas supervisórios, sobre protocolos de comunicação com sistemas supervisórios, sobre o projeto do SimuLIHM, e sobre o ambiente InduSoft;
- Etapa 2: Implementação de telas do sistema supervisório SAGE, para a subestação CGD II, no ambiente Indusoft;
- Etapa 3: Especificação arquitetural do SimuLIHM e implementação do módulo de comunicação entre o SimuLIHM e o sistema supervisório Indusoft;
- Etapa 4: Testes da integração entre o módulo de comunicação desenvolvido e o sistema supervisório Indusoft;
- Etapa 5: Integração e testes do módulo de comunicação desenvolvido ao SimuLIHM;
- Etapa 6: Redação do relatório;
- Etapa 7: Entrega do Relatório.

1.4 ESTRUTURA DO RELATÓRIO

Este relatório foi estruturado da seguinte forma: na Seção 2 será apresentada a fundamentação teórica deste trabalho, de forma a proporcionar ao leitor uma referência sobre os principais assuntos abordados nesse trabalho de TCC; na Seção 3 será apresentado o simulador SimuLIHM desenvolvido no LIHM; na Seção 4 serão apresentados os desenvolvimentos realizados ao longo do trabalho, os quais incluem: a especificação arquitetural; a implementação de telas do supervisório e a implementação do módulo de comunicação com o supervisório; na Seção 5 serão discutidos os critérios a serem adotados na validação dos resultados e os resultados alcançados com os desenvolvimentos realizados nesse TCC; e na Seção 6 são apresentadas as considerações finais e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será apresentada uma revisão bibliográfica com os principais conceitos explorados no decorrer do trabalho, visando proporcionar ao leitor o conhecimento básico necessário ao seu entendimento.

2.1 SISTEMAS SUPERVISÓRIOS

Os sistemas supervisórios começaram a ser utilizados na última metade do século 20, permitindo a visualização e operação de processos industriais de forma remota, por meio de Interface Homem-Máquina (IHM).

Os sistemas supervisórios são constituídos basicamente por telas, base de dados e drivers de comunicação. Por meio dos mais diversos protocolos de comunicação existentes, pode-se realizar a aquisição de dados provenientes de dispositivos de campo, como Controladores Lógicos Programáveis (CLPs), através de drivers específicos. Esses dados são então armazenados na base de dados e usados para animar as telas do supervisório, que mantém várias informações sobre o processo.

Uma outra funcionalidade de grande importância em sistemas supervisórios é a implementação de alarmes. É comum em sistemas supervisórios telas com uma lista dos alarmes gerados durante o período de supervisão e controle. A lista de alarmes é atualizada em tempo real e informa o operador dos eventos ocorridos no sistema elétrico. Cada item da lista possui um conjunto de informações sobre o evento, como a data e a hora da ocorrência, a identificação do equipamento associado ao evento, seu estado atual e uma mensagem de descrição.

Ao tornar a operação de suas plantas mais eficiente e confiável, através da visualização dos dados do processo em tempo real, as indústrias passaram a basear seus relatórios e ferramentas gerenciais nos dados provindos do sistema supervisório. Essa demanda aliada a evolução dos computadores e à acirrada concorrência entre os fabricantes, fez com que a tecnologia SCADA (*Supervisory Control and Data Acquisition*) passasse continuamente por melhorias e modificações em sua estrutura, agregando um vasto conjunto de novas funcionalidades.

Dentre as mais diversas funcionalidades disponíveis nos sistemas supervisórios atuais, algumas são destacadas em (PAIOLA, 2015):

- Identificação da dinâmica do processo, possibilitando análises complexas que auxiliam no gerenciamento da produção;
- Facilidade de integração com os diversos sistemas disponíveis na rede de automação, por exemplo, PIMS (*Plant Information Management System*), MES (*Manufacturing Execution Systems*) e ERP (*Enterprise Resource Planning*). Um dos fatores primordiais para essa integração é a possibilidade de implementação de sistemas supervisórios em Arquiteturas Orientadas a Serviços (SOA);
- Acesso remoto, incluindo visualização, operação, e até mesmo desenvolvimento remoto de aplicativos. Este recurso facilita a manutenção e supervisão de aplicações distribuídas em longas distâncias;
- Uso de tecnologias móveis como *smartphone*, *tablet*, *etc.* possibilitando os responsáveis pelo gerenciamento do processo visualizem e operem o sistema enquanto andam pelo chão de fábrica, em salas de operação, em suas mesas no escritório ou mesmo de sua casa.

2.1.1 SAGE

O SAGE é um sistema SCADA/EMS (*Supervisory Control and Data Acquisition/Energy Management System*) de grande porte e alto desempenho, desenvolvido e constantemente atualizado pelo Cepel (Centro de Pesquisas de Energia Elétrica). É utilizado por dezenas de concessionárias de geração, transmissão e distribuição de energia elétrica no Brasil, em especial pelas empresas fundadoras do Cepel (Chesf, Furnas, Eletronorte e Eletrosul), além do ONS (Operador Nacional do Sistema Elétrico), em todos os seus centros de controle. Sua arquitetura modular permite sua adequada customização para que possa ser utilizado seja como um gateway de comunicação, como um concentrador de dados de um sistema de distribuição, como um supervisório local ou regional, como um centro de operação de sistema ou, ainda, como um sistema “*multi-site*”, composto por vários centros de controle sincronizados e redundantes.

O SAGE implementa um conjunto completo e robusto de funções EMS dedicadas a determinar e monitorar a condição operativa corrente do sistema elétrico em tempo real. Foi desenvolvido com base em conceitos de sistemas distribuídos e expansíveis, sendo constantemente expandido através da adição de novos módulos de software.

Dentre os vários módulos que compõe o SAGE, temos: Sistema de Controle de Geração (CAG), sistema de Armazenamento Histórico de longo prazo (BDH), IHM, suporte a sistemas *multi-site* com facilidades para sincronização de dados e contingenciamento entre centros de controle, interface orientada a serviços (SOA) para conexão com aplicações externas, suporte de múltiplos perfis de usuário, integração com sistemas de segurança eletrônica, entre outros. Na Figura 1, é apresentada a arquitetura de software do SAGE com seus diversos módulos.

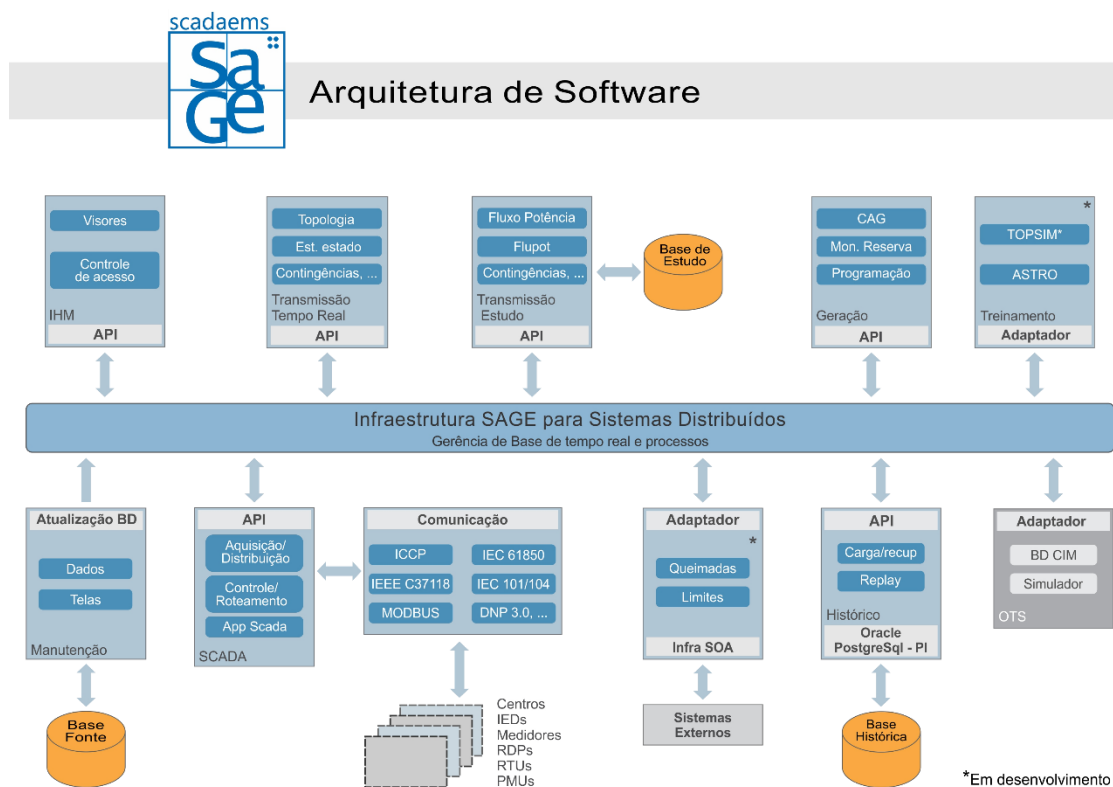


Figura 1: Arquitetura de software do SAGE

Fonte: <http://www.sage.cepel.br/sage/index.php/pt/main-pt-br/arquitetura-de-software>

No contexto de sistemas elétricos, a planta industrial é representada na tela do supervisor através de diagramas unifilares, cujos elementos retratados são equipamentos de controle, proteção e distribuição de energia. Como exemplo, a Figura 2

ilustra o diagrama unifilar de uma subestação da CHESF representado no sistema SAGE (LEITE; RODRIGUES; OLIVEIRA, 2007).

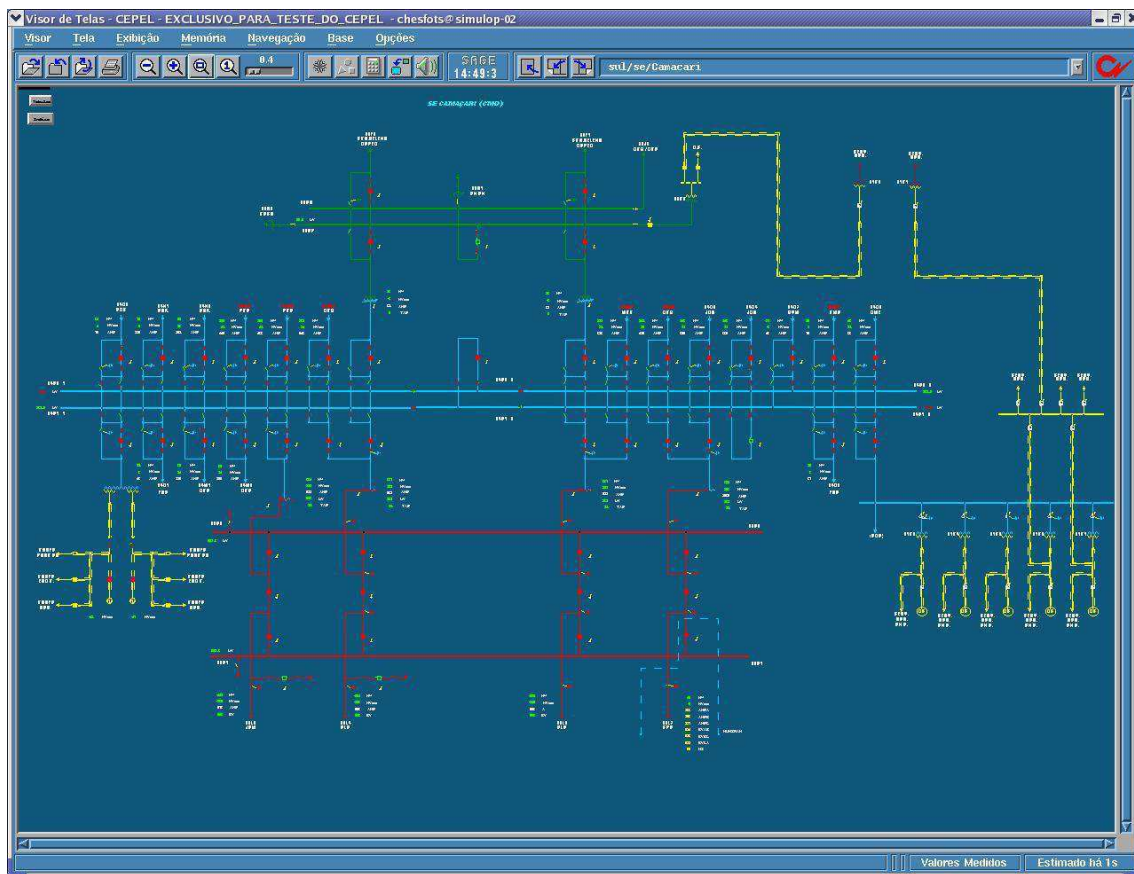


Figura 2: Diagrama unifilar de uma subestação representada no supervisor SAGE

Fonte: CHESF

Durante um turno de trabalho, o operador da subestação pode monitorar através dessa tela o comportamento de variáveis como potência, tensão e corrente nas linhas de transmissão. Além disso, para manter a estabilidade do sistema e o fornecimento de energia elétrica aos clientes, o operador deve reconfigurar o sistema à medida que seu estado evolui. A reconfiguração também pode ser realizada através do comando remoto dos equipamentos representados nessa tela, como a abertura ou fechamento de disjuntores e chaves seccionadoras.

2.1.2 INDUSOFT WEB STUDIO

O *InduSoft Web Studio* (IWS) é uma ferramenta para desenvolvimento de projetos de IHM, de Sistemas SCADA e de monitoramento da Eficiência Total de Equipamentos (OEE), podendo ser aplicado nos mais diversos domínios.

Cada projeto consiste em:

- Um banco de dados de *tags* para gerenciar todos os dados em tempo de execução, incluindo variáveis internas e dados de entrada/saída;
- *Drivers* configuráveis para se comunicar em tempo real com CLP's, dispositivos remotos de entrada/saída, e outros equipamentos de aquisição de dados;
- Tela de IHM e painéis OEE animados;
- Módulos opcionais, tais como alarmes, eventos, tendências, formulários, relatórios, *scripts*, escalonadores, sistema de segurança, interface completa com Banco de Dados (BD).

Os projetos desenvolvidos no IWS podem ser executados localmente ou carregados em um computador remoto. O servidor de tempo de execução do projeto processa dados de entrada/saída dos dispositivos conectados e então reage a estes, mostrando e/ou salvando os dados.

O IWS apresenta uma interface Windows moderna baseada em *ribbon*. Em projetos de interfaces de computador, um *ribbon* é um elemento de controle gráfico na forma de um conjunto de barra de ferramentas alocadas em várias abas. Tais *ribbons*, usam abas para expor diferentes grupos de controle, eliminando a necessidade de várias barras de ferramentas paralelas (RIBBON... 2016). Na Figura 3, é apresentado o ambiente de desenvolvimento IWS.

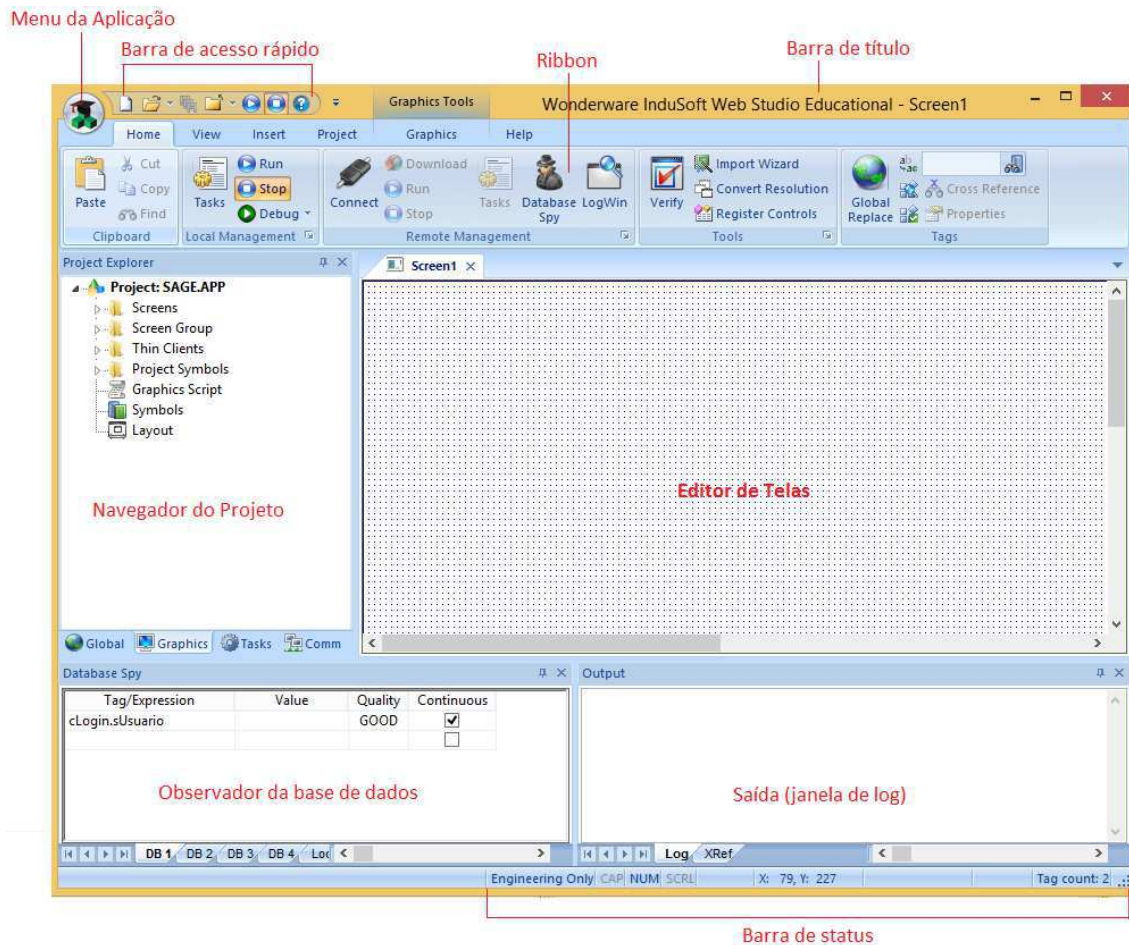


Figura 3: Ambiente de desenvolvimento do IWS

Fonte: O próprio autor

2.2 PADRÃO OPC

Na área de automação industrial, são vários os protocolos de comunicação existentes usados para integrar aplicações, dispositivos e servidores. Nesta seção apresentaremos o padrão OLE (*Object Linking and Embedding*), para Controle de Processos (OPC), o qual detém grande aceitação na indústria de automação.

O padrão OPC foi lançado em 1996 com o objetivo de abstrair protocolos específicos de CLP's (Modibus, Profibus, etc.) e permitir que sistemas IHM/SCADA se comunicassem com uma camada intermediária responsável por transformar solicitações padrão de leitura e escrita OPC em solicitações específicas para cada dispositivo, e vice-versa (OPC FOUNDATION, 2016).

O conjunto de especificações que faz parte do padrão OPC, define a interface entre clientes e servidores, possibilitando o acesso a dados de tempo real, monitoramento de alarmes e eventos, acesso a dados históricos, etc.

Dentre as várias especificações definidas no padrão OPC, neste trabalho será usada a especificação OPC DA (*OPC Data Access*). A OPC DA é uma das especificações do OPC clássico, que tem como base a tecnologia da Microsoft Windows, e faz uso do COM/DCOM (*Distributed Component Object Model*) para troca de dados entre componentes de software locais ou distribuídos.

Na Figura 4 é apresentada uma arquitetura básica para um sistema que faz uso do padrão OPC para comunicação com dispositivos industriais.

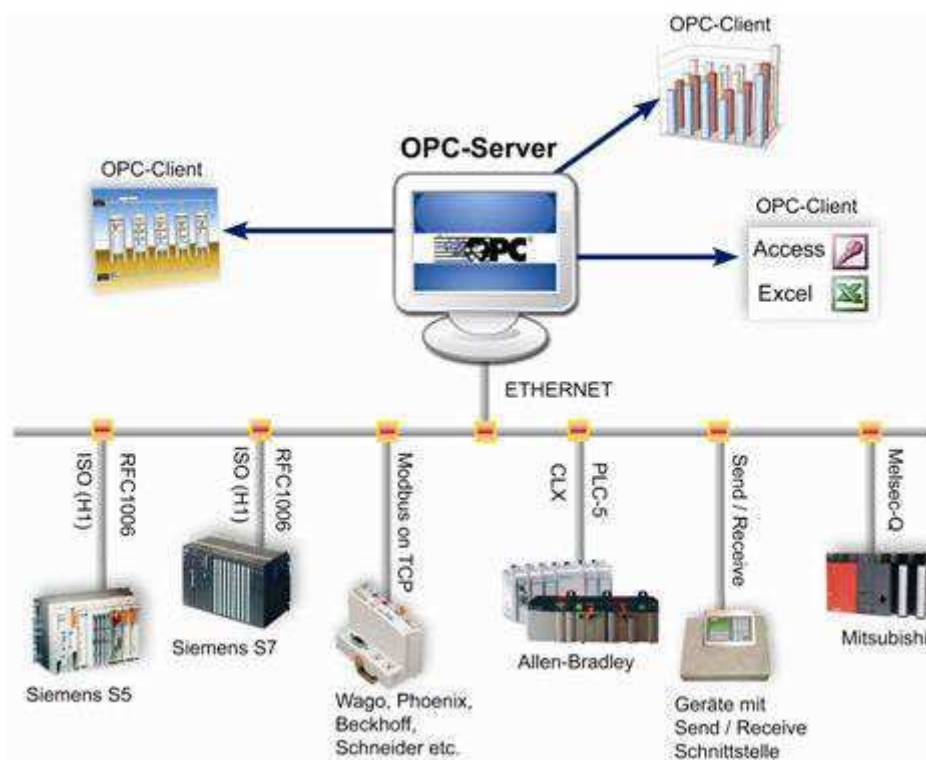


Figura 4: Arquitetura OPC

Fonte: (ELPROCUS, 2016)

Os clientes OPC, tais como sistema SCADA, IHM's, etc., se comunicam com os dispositivos da planta industrial por meio do servidor OPC. Desta forma, elimina-se a necessidade de que cada cliente possuir um driver específico para comunicação, por meio de diferentes protocolos proprietários. Portanto, OPC é um padrão de interoperabilidade que permite dispositivos de diferentes fabricantes realizarem uma troca de dados segura e confiável.

2.3 TREINAMENTO DE OPERADORES EM SUBESTAÇÕES ELÉTRICAS

Nesta seção, apresentam-se os métodos mais comuns na formação, reciclagem e certificação de operadores de sistemas elétricos de potência, destacando-se o uso de software de simulação por empresas do setor elétrico na realização de treinamentos, com exemplos de aplicações e algumas das suas principais características.

2.3.1 MÉTODOS DE TREINAMENTO

Como descrito em (TORRES FILHO, 2011), ao ingressarem na empresa, os novos operadores iniciam um processo de formação com aulas teóricas, aprendizagem dos procedimentos operacionais, visitas técnicas às subestações da empresa, e estudo de ocorrências anteriores, buscando-se transmitir aos profissionais as informações minimamente necessárias para iniciarem suas atividades.

Na etapa seguinte, o operador iniciante passa a vivenciar a rotina da subestação, normalmente supervisionado por um operador mais experiente. A prática é obtida operando o sistema real, a começar pelas manobras mais simples e frequentes. O operador supervisor instrui e acompanha o iniciante na realização dessas manobras. E, na medida em que adquirem experiência e domínio dos procedimentos operacionais, os iniciantes ganham maior autonomia e podem realizar manobras mais complexas (SILVA et al., 2009).

De forma a preencher as lacunas presentes neste método de treinamento, que pode não expor os operadores a situações críticas e de emergência, faz-se uso da dramatização e simulação de cenários reais que expõem os operadores a situações pouco frequentes, apesar de possíveis, e avalia a sua capacidade.

Para manter os operadores sempre atualizados e evitar falhas humanas, também são realizadas reuniões periódicas entre os operadores e engenheiros responsáveis pela subestação para discutir as últimas ocorrências e possíveis modificações nos procedimentos operacionais.

2.3.2 SIMULADORES PARA APOIO AO TREINAMENTO

As empresas do setor elétrico vêm investindo cada vez mais em ferramentas que auxiliem na capacitação dos seus operadores. Dentre estas ferramentas pode-se destacar o uso de simuladores que permitem aos operadores realizar a simulação de cenários diversos, interagindo com um ambiente similar ao encontrado nas situações reais.

A utilização desses softwares de simulação facilita assimilação dos procedimentos de operação e permite que os operadores em treinamento possam praticar com uma réplica do sistema sem a preocupação de interferir no ambiente real. Ou seja, a segurança do operador e do sistema fica resguardada, e o fornecimento de energia elétrica aos consumidores não é interrompido. Ao mesmo tempo, o operador se familiariza com a prática de operação e adquire maior autoconfiança (TORRES FILHO, 2011).

Desta forma, os simuladores representam um importante complemento a metodologia tradicional de treinamento apresentada na Seção 2.3.1, fornecendo uma formação mais completa aos operadores e preparando-os de forma mais eficaz para enfrentar as situações reais presentes no seu cotidiano.

Para que o treinamento por meio de simuladores seja eficaz, é necessário que estes reproduzam com o maior grau de fidelidade possível o ambiente real encontrado pelos operadores durante a operação do sistema elétrico. Desta forma, é desejável que os simuladores possibilitem a interação com a planta elétrica tanto por meio de sistemas supervisórios como pela representação virtual de painéis presentes em salas de operação.

Outros requisitos importantes para os simuladores, como colocado em (TORRES FILHO, 2011), são listados a seguir:

- Possibilidade de realização de treinamentos a distância;
- Supervisão de treinamentos em tempo real por um tutor;
- Geração automática de diferentes cenários de treinamento;
- Geração automática de avaliação de desempenho dos operadores após a realização do treinamento.

A maioria dos simuladores usados pelas empresas do setor elétrico permite a interação dos operadores apenas com o sistema supervisório, o qual pode ser configurado para simular cenários de ocorrências reais. Dois destes simuladores são descritos a seguir:

- SIMULOP – Simulador desenvolvido pelo CEPEL e usado na CHESF para treinamento e certificação de operadores. Este sistema é resultado da integração da ferramenta EMS para supervisão e controle de sistemas elétricos de potência, do SAGE, e do simulador digital de redes elétricas OTS (*Operator Training Simulator*) desenvolvido pelo EPRI (*Electric Power Research Institute*);
- ASTRO (Ambiente Simulado para Treinamento de Operadores) desenvolvido pelo CEPEL, em parceria com a ELETROSUL, para funcionar de forma integrada ao SAGE (SILVA et al., 2009). Dentre os principais recursos fornecidos pelo ASTRO, pode-se destacar a edição de cenários de treinamento, a simulação da planta elétrica, e a análise de desempenho segundo critérios pré-estabelecidos pelo tutor do treinamento.

Tendo apresentado estes conceitos, na Seção 3 será introduzido o simulador para treinamento de operadores desenvolvido pelo LIHM, o SimuLIHM.

3 SIMULADOR SIMULIHM

Nesta seção será apresentado o simulador do ambiente de uma subestação, representado em realidade virtual, o qual foi desenvolvido no LIHM para o treinamento de operadores de sistemas elétricos, o SimuLIHM.

Dentre os principais propósitos do SimuLIHM, encontram-se:

- Desenvolver um ambiente controlado que permita o estudo do comportamento do operador quando submetido a situações críticas ou de emergência;
- Disponibilizar uma infraestrutura para treinamento de operadores de sistemas elétricos.

Além dos operadores que serão treinados na realização de manobras no sistema elétrico, o simulador também permite que usuários tutores possam criar, salvar, selecionar e editar cenários de simulação, assim como acompanhar a execução do cenário em tempo real e disparar eventos durante o treinamento.

A arquitetura inicial do simulador permitia que os operadores interagissem apenas com um ambiente virtual de treinamento contendo a representação de painéis presentes na sala de operação de subestações elétricas.

Em (TORRES FILHO, 2011) essa arquitetura foi ampliada para abranger a interação com sistemas supervisórios, tornando assim o treinamento de operadores por meio da ferramenta SimuLIHM mais eficaz. A arquitetura desenvolvida em (TORRES FILHO, 2011) é apresentada na Figura 5.

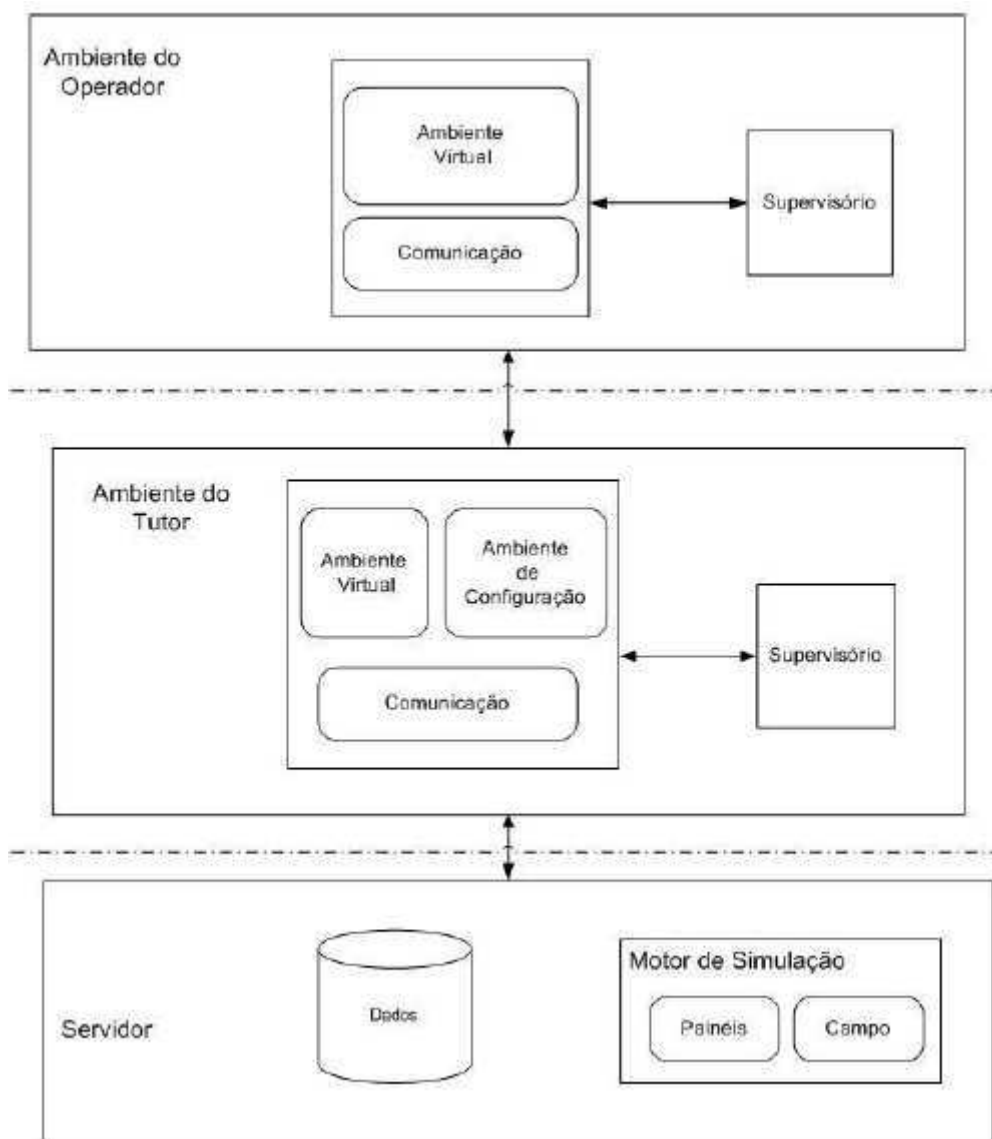


Figura 5: Arquitetura do SimuLIHM

Fonte: (TORRES FILHO, 2011)

Essa arquitetura é composta por três ambientes: Ambiente do Tutor, Ambiente do Operador e Servidor. Tanto o ambiente do Tutor, quanto o Ambiente do Operador, apresentam o módulo Ambiente Virtual. Por meio deste módulo é feita uma representação virtual da sala de operação de uma subestação, como apresentado na Figura 6, proporcionando um alto grau de realismo ao treinamento.

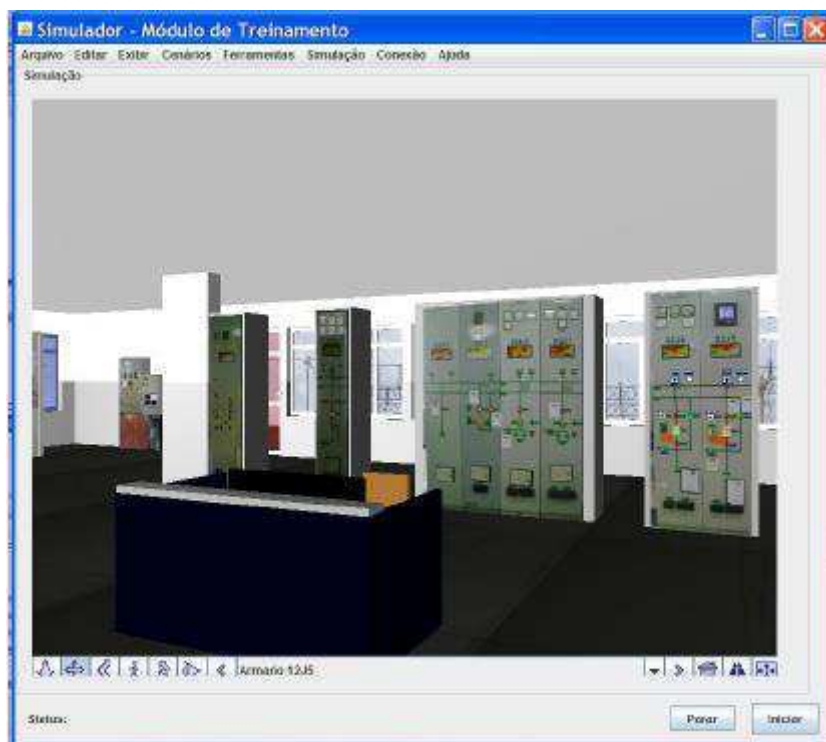


Figura 6: Ambiente virtual do SimuLIHM

Fonte: (TORRES FILHO, 2011)

O Ambiente de Configuração, presente no Ambiente do Tutor, permite a configuração de novos cenários de treinamento, assim como o acesso aos cenários já construídos. Além disso, tanto o Ambiente do Tutor como o Ambiente do Operador podem interagir com uma instância local do sistema supervisório, como apresentado na Figura 7.

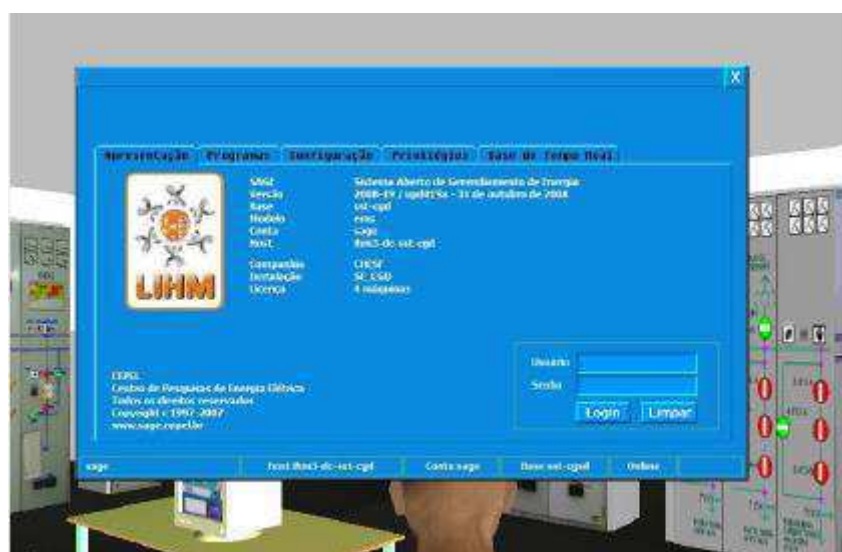


Figura 7: Interação entre SimuLIHM e sistema supervisório

Fonte: (TORRES FILHO, 2011)

O Servidor apresenta um BD com informações sobre treinamentos realizados, histórico de alarmes, estado de objetos da interface, etc. O Servidor ainda apresenta o Motor de Simulação, que possibilita a simulação do comportamento da planta elétrica e dos painéis presentes no ambiente virtual.

Os módulos de comunicação realizam a troca de mensagens entre os diferentes ambientes por meio de *socket*, que faz uso do protocolo TCP/IP (*Transmission Control Protocol/ Internet Protocol*). Já para a comunicação com o sistema supervisorio, em (TORRES FILHO, 2011) foi realizada a especificação e implementação de um driver de comunicação com o protocolo DDE (*Dynamic Data Exchange*).

Observa-se que a arquitetura apresentada não fornece o nível de detalhes necessários para dar suporte ao processo de manutenção e evolução da ferramenta pelos desenvolvedores. Além disso, o módulo de comunicação desenvolvido é específico para o protocolo DDE, o qual que vem caindo em desuso com a expansão de protocolos com maior grau de interoperabilidade, como o OPC.

Sendo assim, nas próximas seções serão apresentadas a especificação arquitetural realizada para SimuLIHM e a implementação do módulo de comunicação OPC para integração desta ferramenta com diferentes sistemas supervisorios.

4 DESENVOLVIMENTO

Nesta seção será apresentado o desenvolvimento realizado durante este projeto, assim como as decisões tomadas e às respectivas justificativas.

4.1 ESPECIFICAÇÃO ARQUITETURAL

A especificação arquitetural do SimuLIHM foi feita em UML (Linguagem de Modelagem Unificada). UML é uma linguagem de modelagem, independente de metodologia, desenvolvida para fornecer uma notação gráfica padronizada que pode ser usada na descrição de modelos orientados a objeto (GOMAA, 2011).

O uso de uma linguagem de modelagem gráfica como UML ajuda no desenvolvimento, entendimento e comunicação de diferentes visões de um sistema. UML é largamente utilizada na descrição arquitetural de aplicações de software que fazem uso de abordagem dirigida a modelos, onde a arquitetura do software é desenvolvida antes de sua implementação.

Para a realização deste TCC, inicialmente foram feitas decisões arquiteturais de forma a especificar as modificações a serem realizadas no SimuLIHM. A seguir, serão detalhadas as especificações arquiteturais tanto no nível geral de sistema, incluindo todos os subsistemas constituintes da plataforma de treinamento de operadores, quanto no nível de comunicação com o sistema supervisor, objetivo principal deste trabalho.

4.1.1 ARQUITETURA GERAL DO SISTEMA

Na Figura 8, é apresentada a especificação arquitetural para a plataforma de treinamento de operadores. Além de apresentar um maior grau de detalhamento, essa arquitetura apresenta modificações importantes se comparada à arquitetura apresentada em (TORRES FILHO, 2011). Nesta seção, serão descritos os diferentes módulos arquiteturais presentes na Figura 8. A comunicação entre os módulos será descrita de forma mais detalhada na Seção 5, onde será apresentado um diagrama de sequência de mensagens para um cenário de uso da plataforma SimuLIHM.

Como já mencionado durante o trabalho, cada atividade de treinamento pode ser composta por dois tipos de usuários, Tutor e Operador, classificados como atores na especificação arquitetural. A interação dos usuários com o sistema pode variar de acordo com o tipo de usuário, apresentando particularidades. Dessa forma, a arquitetura da plataforma de simulação foi dividida em dois subsistemas: Módulo Tutor e Módulo Operador. Além disso, a plataforma interage com uma variedade de sistemas externos: Servidor de Banco de Dados, Motor de Simulação, Servidor de Tags e Sistema Supervisório.

Durante a realização de um treinamento, os subsistemas e sistemas externos podem ser executados em máquinas diferentes, como apresentado no diagrama de implantação da Figura 9. Pelos índices de multiplicidade apresentados neste diagrama, observa-se que para cada treinamento teremos um computador servidor executando alguns dos sistemas externos (Motor de Simulação, Servidor de Tags e Servidor de Banco de Dados), um computador executando o Módulo Tutor e uma instância Sistema Supervisório, e um ou mais computadores, cada um executando o Módulo Operador e uma instância do Sistema Supervisório. O fluxo de informações entre os diferentes módulos obedece ao sentido indicado pelas setas.

Nas Seções 4.1.1.1, 4.1.1.2 e 4.1.1.3 será apresentada a descrição detalhada de cada módulo, assim como dos sistemas externos.

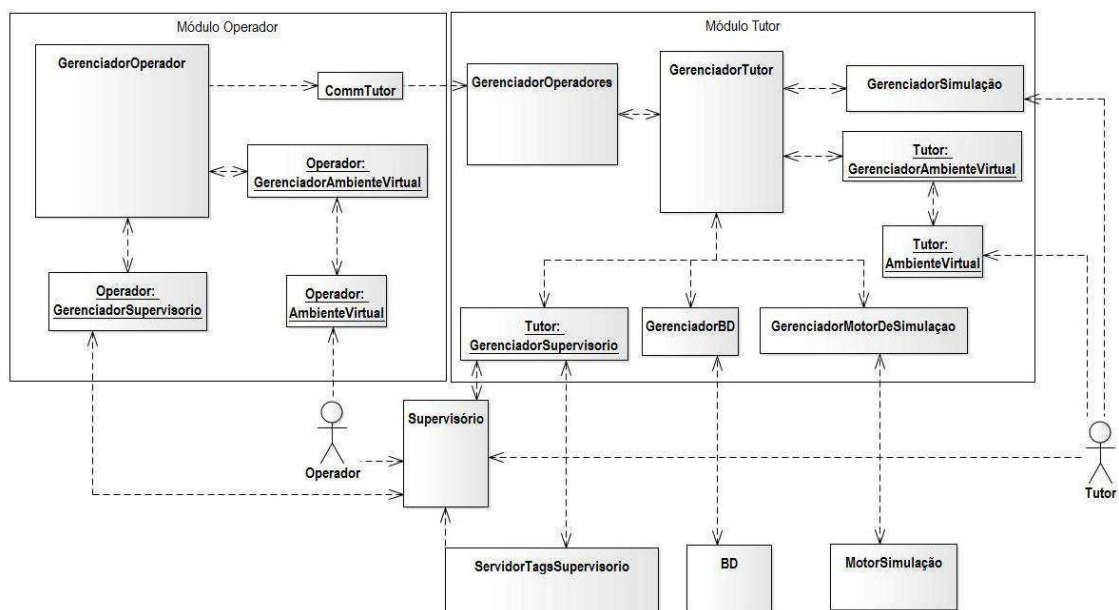


Figura 8: Especificação arquitetural da plataforma de treinamento de operadores

Fonte: O próprio autor

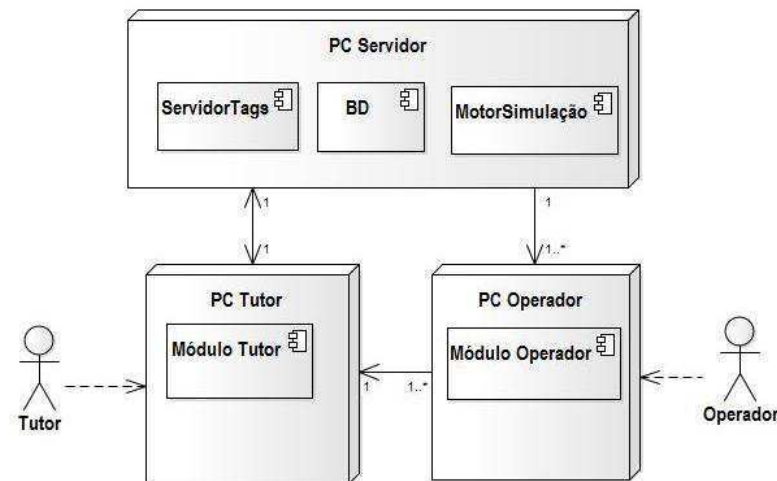


Figura 9: Diagrama de implantação

Fonte: O próprio autor

4.1.1.1 MÓDULO OPERADOR

O Módulo Operador é responsável pelo suporte às funcionalidades permitidas ao usuário Operador. A interação do Operador com o módulo pode ser feita por meio da IHM de um sistema supervisório SCADA ou através de um ambiente virtual representando a sala de controle para a planta elétrica. As principais funcionalidades oferecidas pelo módulo Operador são especificadas no diagrama de Casos de Uso apresentado na Figura 10.

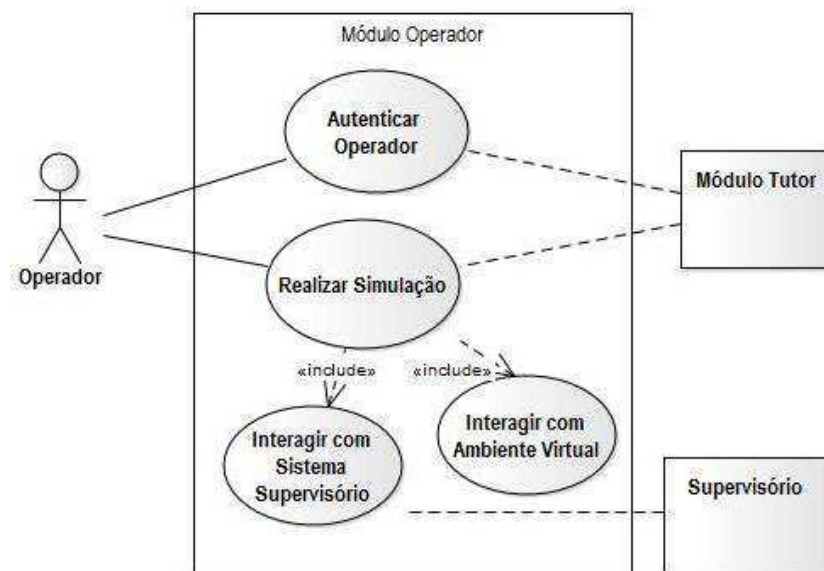


Figura 10: Casos de uso para o Módulo Operador

Fonte: O próprio autor

Com base nas funcionalidades do Módulo Operador, este subsistema foi dividido nos componentes apresentados na especificação arquitetural da Figura 8. A descrição dos subsistemas, assim como da comunicação entre estes, é apresentada na Tabela 1.

Tabela 1: Descrição de componentes do Módulo Operador

Componente	Descrição
GerenciadorOperador	Realiza a integração de todos os componentes do módulo operador e gerencia a comunicação entre estes.
Operador: GerenciadorSupervisorio	Realiza a integração entre o módulo operador e o sistema supervisorio, gerenciando a comunicação entre estes.
Operador: GerenciadorAmbienteVirtual	Realiza a integração entre o módulo operador e o ambiente virtual de simulação, gerenciando a comunicação entre estes.
Operador: AmbienteVirtual	Representação do ambiente virtual de simulação. Mantém todos os recursos provenientes deste, e monitora os eventos que ocorrem durante o treinamento.
CommTutor	Realiza o envio de mensagens para módulo tutor.

Fonte: O próprio autor

4.1.1.2 MÓDULO TUTOR

O Módulo Tutor é responsável pelo suporte às funcionalidades permitidas ao usuário Tutor. A interação do Tutor com o módulo pode ser feita por meio da IHM de um sistema supervisorio SCADA, através de um ambiente virtual representando a sala de controle para a planta elétrica, ou ainda através de uma IHM para gerência da simulação. As principais funcionalidades oferecidas pelo módulo Tutor são especificadas no diagrama de Casos de Uso apresentado na Figura 11.

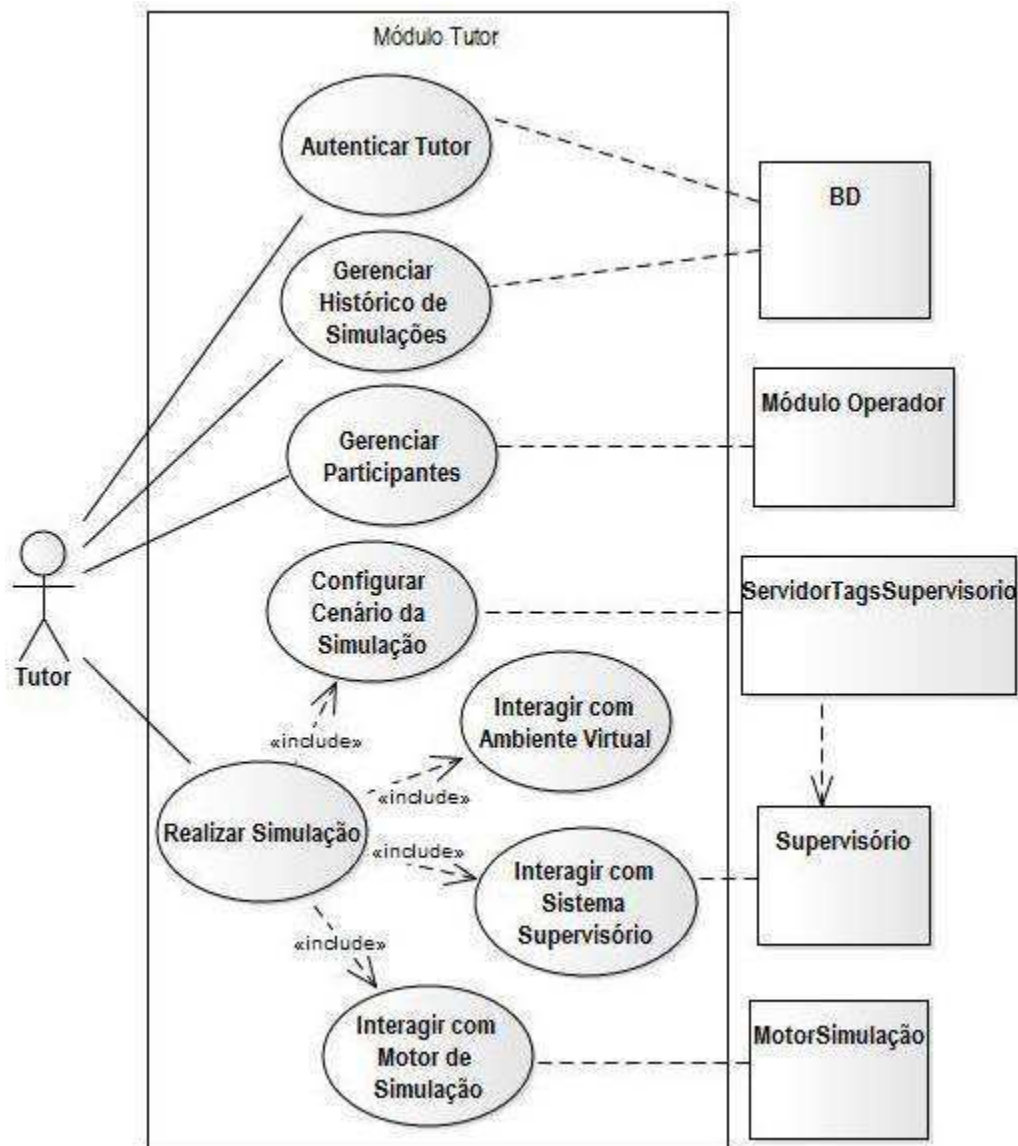


Figura 11: Casos de uso para o Módulo Tutor

Fonte: O próprio autor

Com base nas funcionalidades do Módulo Tutor, este subsistema foi dividido nos componentes apresentados na especificação arquitetural da Figura 8. A descrição dos subsistemas, assim como da comunicação entre estes, é apresentada na Tabela 2.

Tabela 2: Descrição dos componentes do Módulo Tutor

Componente	Descrição
GerenciadorTutor	Realiza a integração de todos os componentes do módulo tutor e gerencia a comunicação entre estes.
GerenciadorOperadores	Gerencia todos os operadores participando de uma simulação, sendo ainda responsável por receber e encaminhar todas as mensagens provenientes dos participantes.
GerenciadorSimulação	Gerencia funcionalidades relacionadas a configuração da simulação, assim como a o acesso a dados provenientes de simulações passadas, geração de relatórios, etc.
Tutor: GerenciadoAmbienteVirtual	Realiza a integração entre o módulo tutor e o ambiente virtual de simulação, gerenciando a comunicação entre estes.
Tutor: AmbienteVirtual	Representação do ambiente virtual de simulação. Mantém todos os recursos provenientes deste, e monitora os eventos que ocorrem durante o treinamento.
Tutor: GerenciadorSupervisório	Realiza a integração entre o módulo operador e o sistema supervisório, gerenciando a comunicação entre estes.
GerenciadorBD	Gerencia a comunicação com o BD da aplicação.
GerenciadorMotorSimulação	Gerencia a comunicação com o motor de simulação, estabelecendo a conexão, enviando e recebendo mensagens do mesmo.

Fonte: O próprio autor

4.1.1.3 SISTEMAS EXTERNOS

Neste relatório, definem-se sistemas externos como sendo aplicações que, apesar da possibilidade de serem implementadas pela equipe de desenvolvimento, estes não são feitos através da linguagem de programação adotada no projeto. O desenvolvimento destes sistemas é realizado em IDE's (Ambiente de Desenvolvimento Integrado) específicas fornecidas pelos fabricantes.

A plataforma de treinamento de operadores SimuLIHM, interage com alguns sistemas externos de forma a realizar suas funcionalidades, como apresentado na especificação arquitetural da Figura 8. A descrição de cada um desses sistemas é apresentada na Tabela 3.

Tabela 3: Descrição dos sistemas externos

Sistema	Descrição
Supervisório	Instância do sistema supervisório SCADA presente em cada uma das máquinas do treinamento, tutor e operadores.
ServidorTagsSupervisorio	Servidor global de <i>tags</i> usadas por todas as aplicações de sistemas supervisórios do treinamento. Facilita a integração entre os módulos da plataforma de treinamento, e a manutenção da coerência entre as várias instâncias de sistemas supervisórios.
ServidorBD	Servidor global de BD onde é mantido logs de treinamentos realizados, informações sobre participantes, etc.
MotorSimulação	Ferramenta global responsável por realizar a simulação do comportamento da planta e dos painéis da sala de controle.

Fonte: O próprio autor

4.1.2 ARQUITETURA DO MÓDULO DE COMUNICAÇÃO COM O SUPERVISÓRIO

Um dos objetivos principais deste projeto corresponde ao desenvolvimento do módulo de comunicação entre a plataforma de treinamento de operadores e diferentes

sistemas supervisórios. Para a especificação da arquitetura deste módulo, os seguintes requisitos não-funcionais foram considerados:

- REQ01: A plataforma de treinamento deve suportar a integração com sistemas supervisórios fornecidos por diferentes fabricantes;
- REQ02: O módulo supervisório deve possibilitar a comunicação através de diferentes protocolos;
- REQ03: O gerenciamento da comunicação com o sistema supervisório deve satisfazer características específicas para os módulos Tutor e Operador;

Os casos de uso apresentados na Figura 12, representam funcionalidades comuns aos módulos de comunicação com os supervisório no ambiente do Tutor e Operador.

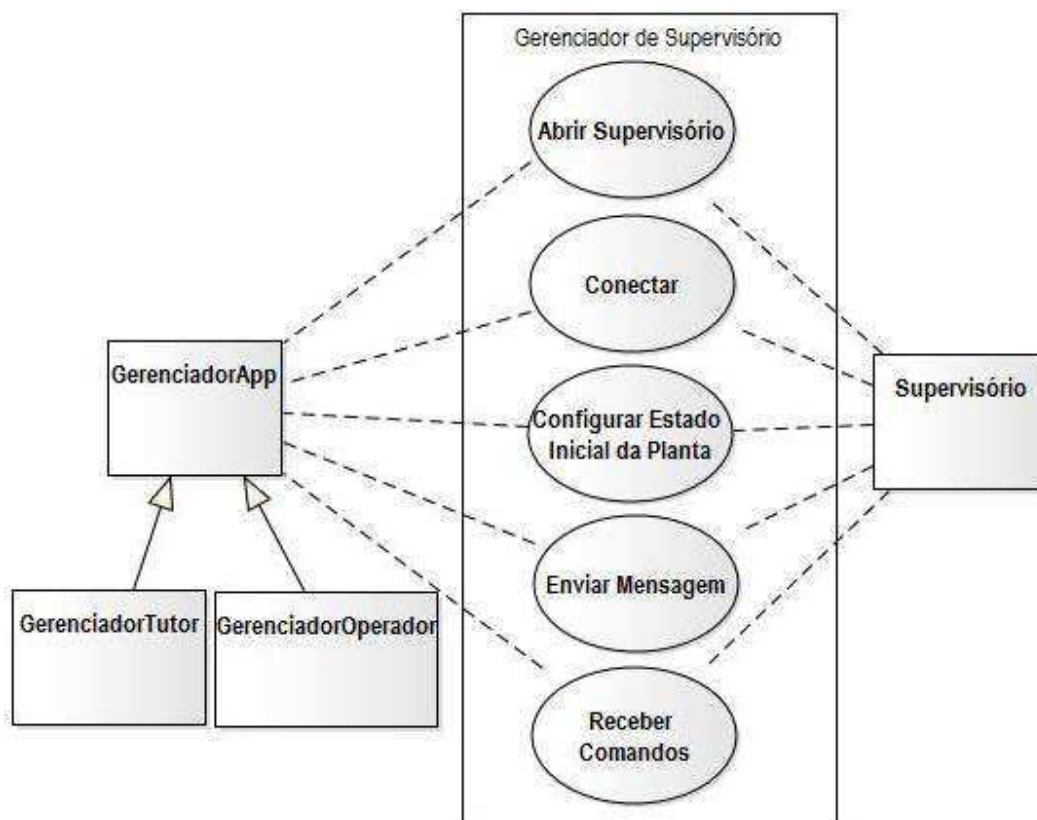


Figura 12: Diagrama de casos de uso do módulo supervisório

Fonte: O próprio autor

A comunicação das aplicações Tutor e Operador com o sistema supervisório apresentam algumas diferenças. O gerenciamento da comunicação com o supervisório do ambiente Tutor é representado no diagrama de atividades da Figura 13.

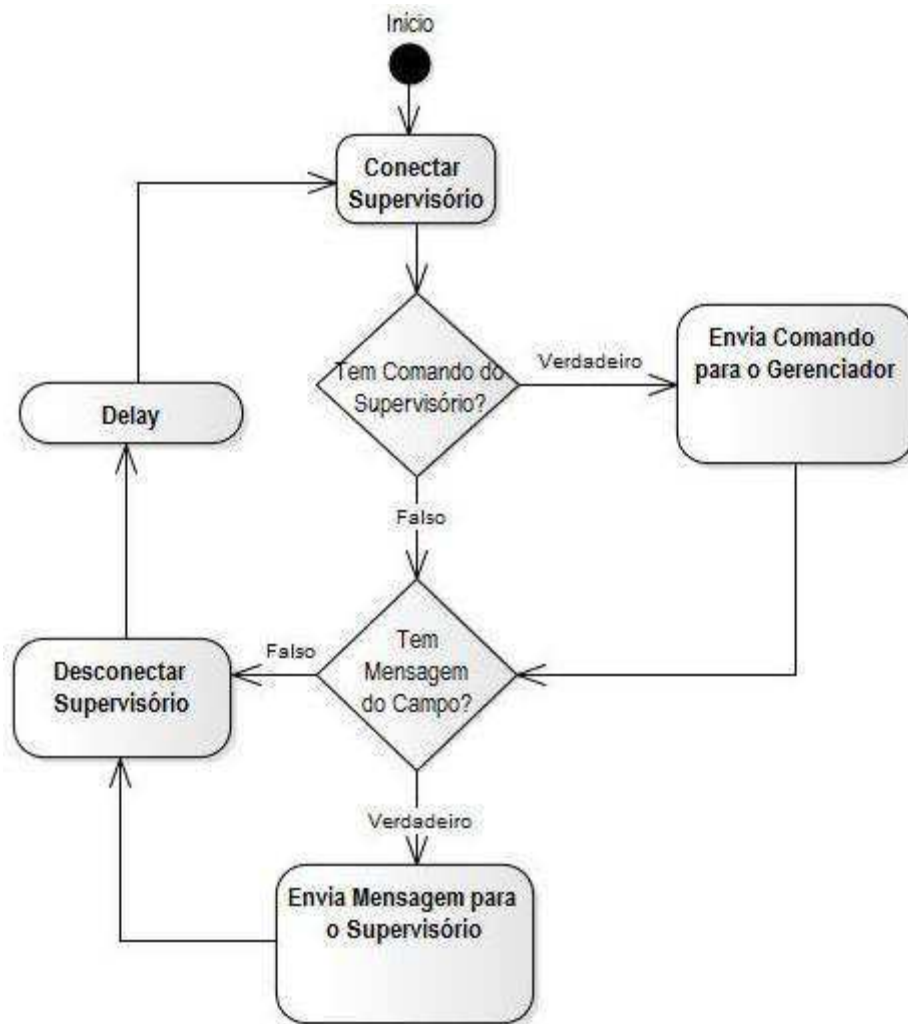


Figura 13: Diagrama de atividades para o módulo SupervisorioTutor

Fonte: O próprio autor

Após estabelecer a conexão com o sistema supervisorio, o módulo SupervisorioTutor verifica se se o Tutor realizou algum comando nos equipamentos representados nas telas do supervisorio. Caso o comando tenha sido realizado, este é enviado para o Gerenciador do Tutor, que deve encaminhá-lo para o motor de simulação. Após estes passos, é verificado se o módulo de comunicação recebeu alguma mensagem proveniente do motor de simulação, a qual deve então ser enviada para o sistema supervisorio, de forma que os estados dos equipamentos sejam atualizados. Finalizando essas atividades, o módulo de comunicação desconecta o sistema supervisorio e espera por um intervalo fixo de tempo, diminuindo a sobrecarga da CPU (Unidade Central de Processamento), após o qual todas as etapas são repetidas.

Para o módulo SupervisorioTreinando, o conjunto de atividades realizadas durante a comunicação com o sistema supervisorio é mais simples, como pode ser verificado na Figura 14.

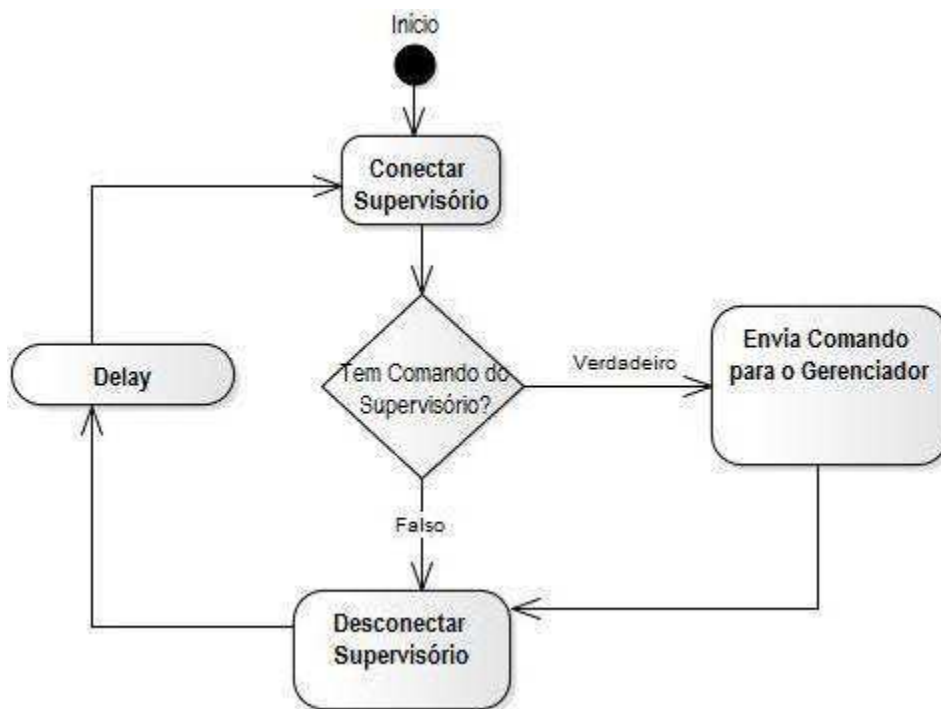


Figura 14: Diagrama de atividades para o módulo SupervisorioTreinando

Fonte: O próprio autor

Após realizar a conexão com o sistema supervisorio, o módulo de comunicação verifica se o Operador realizou algum comando nos equipamentos representados nas telas do supervisorio. Caso o comando tenha sido realizado, este é então enviado para o Gerenciador do Operador, que deve encaminhá-lo para o módulo Tutor, diminuindo a sobrecarga da CPU (Unidade Central de Processamento), após o qual todas as etapas são repetidas

O estado dos equipamentos presentes nas telas do sistema supervisorio do ambiente do operador é atualizado de forma automática quando o Módulo Tutor envia uma mensagem para o servidor global de *tags*. Com isso, não é necessário que o Módulo Tutor envie uma mensagem de volta para o módulo Operador sempre que receber uma mensagem do motor de simulação. Isso representa uma melhoria se comparado a arquitetura apresentada em (TORRES FILHO, 2011), possibilitada pelo uso de um servidor global de *tags*.

Como resultado dos requisitos não-funcionais, expressos nos casos de uso, e na descrição comportamental do módulo de comunicação entre a plataforma de treinamento

e os sistemas supervisórios, foi especificada a arquitetura de software apresentada na Figura 15.

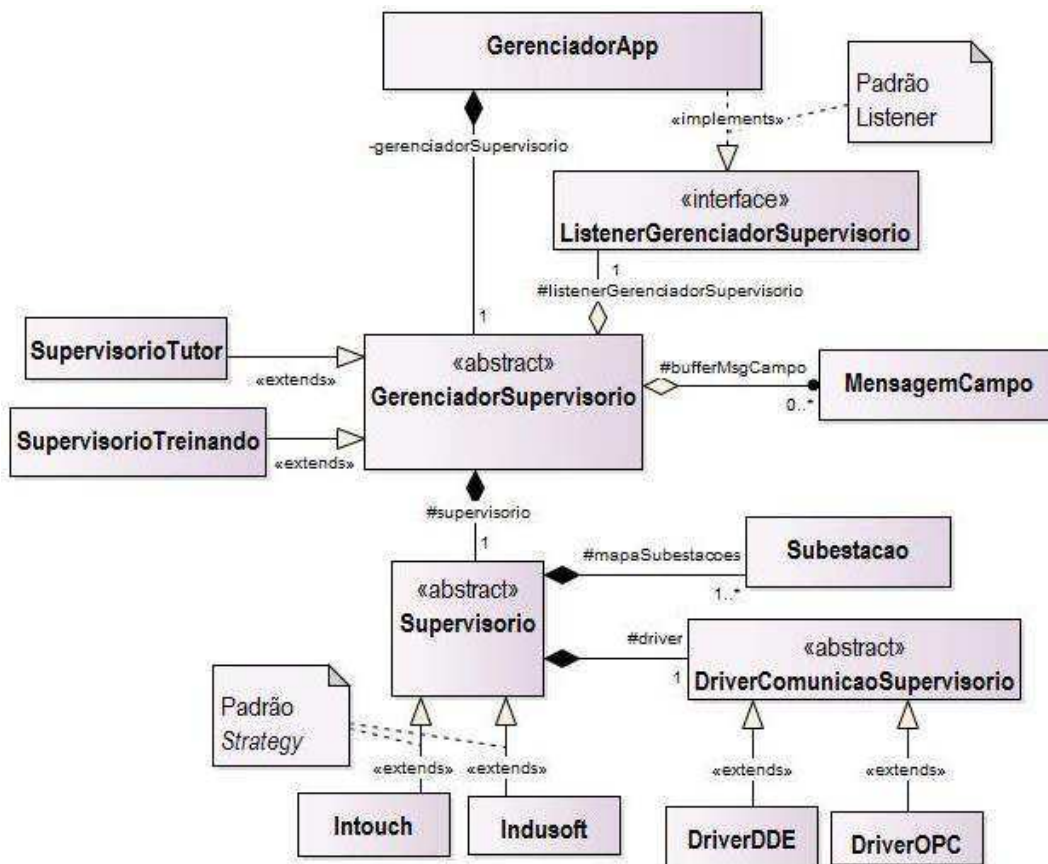


Figura 15: Especificação arquitetural do módulo de comunicação com os sistemas supervisórios

Fonte: O próprio autor

A classe abstrata GerenciadorSupervisorio realiza a integração entre as diferentes classes do módulo de comunicação com o supervisorio, fornecendo ao Gerenciador da Aplicação do Módulo Tutor (GerenciadorTutor) ou Operador (GerenciadorOperador) o conjunto de funcionalidades usadas na comunicação com o sistema supervisorio.

A classe ListenerGerenciadorSupervisorio faz uso do padrão de projeto *Observer* (FREEMAN et al., 2004) para possibilitar que o Gerenciador da aplicação seja notificado sobre comandos executados nas telas do sistema supervisorio, e recebidos pelo módulo de comunicação com o supervisorio.

As classes SupervisorioTutor e SupervisorioTreinando estendem o GerenciadorSupervisorio, herdando todas as funcionalidades e atributos. Desta forma, cada uma dessas classes pode implementar as particularidades da comunicação das aplicações Tutor e Operador com o módulo supervisorio, como apresentado nos diagramas de atividade da Figura 13 e Figura 14.

A classe abstrata *Supervisorio* é uma representação virtual do sistema supervisorio usado pela aplicação. Esta classe define um conjunto de funcionalidades e atributos necessários para que se tenha a comunicação com o sistema supervisorio. Essas funcionalidades devem ser implementadas de forma diferenciada para supervisorios específicos, o que é feito nas classes *Indusoft* e *Intouch*.

Como pode ser observado na Figura 15, cada supervisorio possui os atributos *mapaSubestacoes*, que é um conjunto de objetos do tipo *Subestacao*, e *driver*, que é um objeto do tipo *DriverComunicacaoSupervisorio*. Através do mapa de subestações, são mantidas informações sobre as subestações e equipamentos presentes no supervisorio e usados na simulação. Já o *driver*, é o objeto usado para efetivamente, conectar, enviar e receber mensagens do sistema supervisorio por meio de algum protocolo de comunicação.

A classe abstrata *DriverComunicacaoSupervisorio* define todas as funcionalidades que devem estar presentes nos *drivers*, de forma independente ao protocolo usado. As classes *DriverDDE* e *DriverOPC* implementam essas funcionalidades levando em consideração as particularidades dos protocolos DDE e OPC, respectivamente.

Com a especificação arquitetural apresentada, fazendo-se uso de classes abstratas, pode-se facilmente alternar entre sistemas supervisorios e *drivers* de comunicação usados pela aplicação do simulador. Isso foi possibilitado pelo alto nível de modularidade arquitetural e independência entre classes, alcançados por meio do uso de padrões de projeto de software, como o padrão *strategy* definido em (FREEMAN et al., 2004).

4.2 IMPLEMENTAÇÃO DE TELAS DO SAGE NO SUPERVISÓRIO

INDUSOFT

De forma a validar o módulo de comunicação especificado e implementado neste TCC, foi necessário realizar a implementação das telas de um sistema supervisorio. Durante a realização do projeto, o laboratório LIHM disponibilizou a licença para uso do supervisorio *Indusoft*, no qual foi feita a implementação das telas necessárias. Todas as telas foram implementadas tomando como base a interface com o sistema supervisorio *SAGE*, adotado na *CHESF*, para a subestação *CGD II*, situada em *Campina Grande*.

Na Figura 16 é apresentada a tela inicial do sistema supervisor. Esta tela apresenta informações sobre o supervisor SAGE, campos para autenticação do usuário e abas que dão acesso a outras telas do sistema supervisor. Apenas as abas Apresentação e Programas foram implementadas durante este projeto de TCC.



Figura 16: Tela de apresentação do supervisor

Fonte: O próprio autor

Na Figura 17 é apresentada a aba de Programas do SAGE, possuía qual oferece botões de acesso à diferentes submódulos do supervisor. Destes, apenas os botões Telas e Alarmes foram implementados.

Quando o usuário aciona o botão Alarmes, a tela de alarmes mostrada na Figura 18 é aberta. Apenas a interface gráfica da tela de alarmes foi implementada, restando a implementação das funcionalidades que tornem possível ao usuário realizar a visualização e interação com as diferentes mensagens de alarme que devem ser geradas durante a execução do sistema supervisor. Os botões presentes na parte inferior da tela de alarmes devem ser usados para disponibilizar filtros, de modo que durante a execução do supervisor o usuário possa selecionar quais tipos de alarmes ele deseja visualizar.

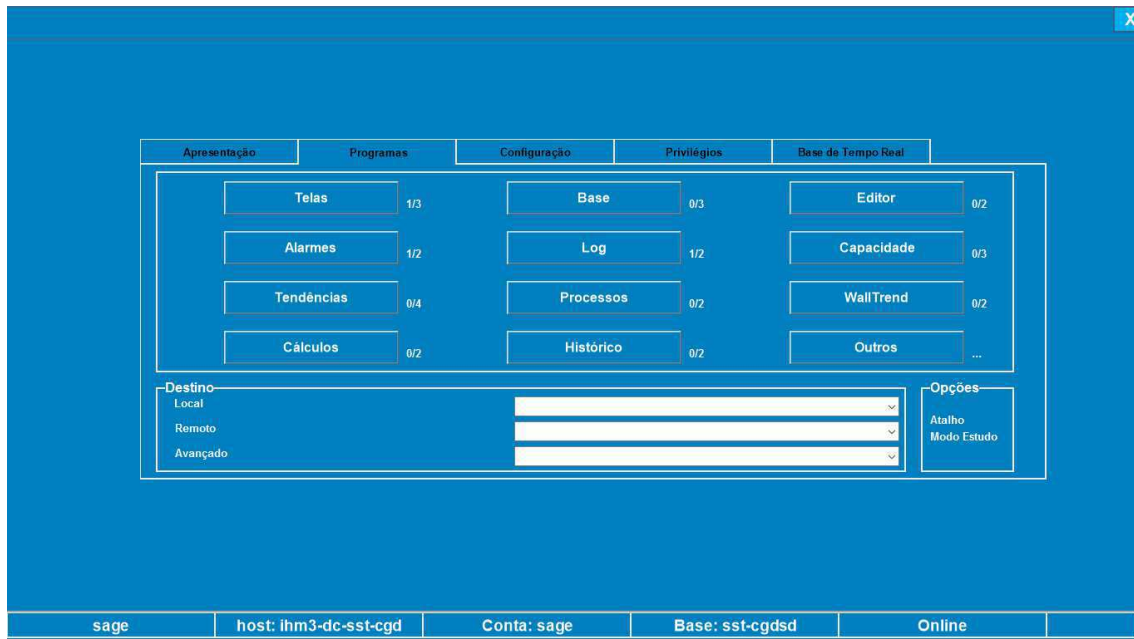


Figura 17: Tela de programas do supervisão

Fonte: O próprio autor



Figura 18: Tela de alarmes do supervisão

Fonte: O próprio autor

A Figura 19 apresenta a tela de sinótico, na qual foi implementada uma parte da subestação CGD II da CHESF. Esta tela é acessada com o acionamento do botão Telas da Figura 17. Para a implementação da tela do sinótico, foi criado um conjunto de

símbolos, apresentados na Tabela 4, que podem ser reaproveitados na implementação de telas de outras subestações.

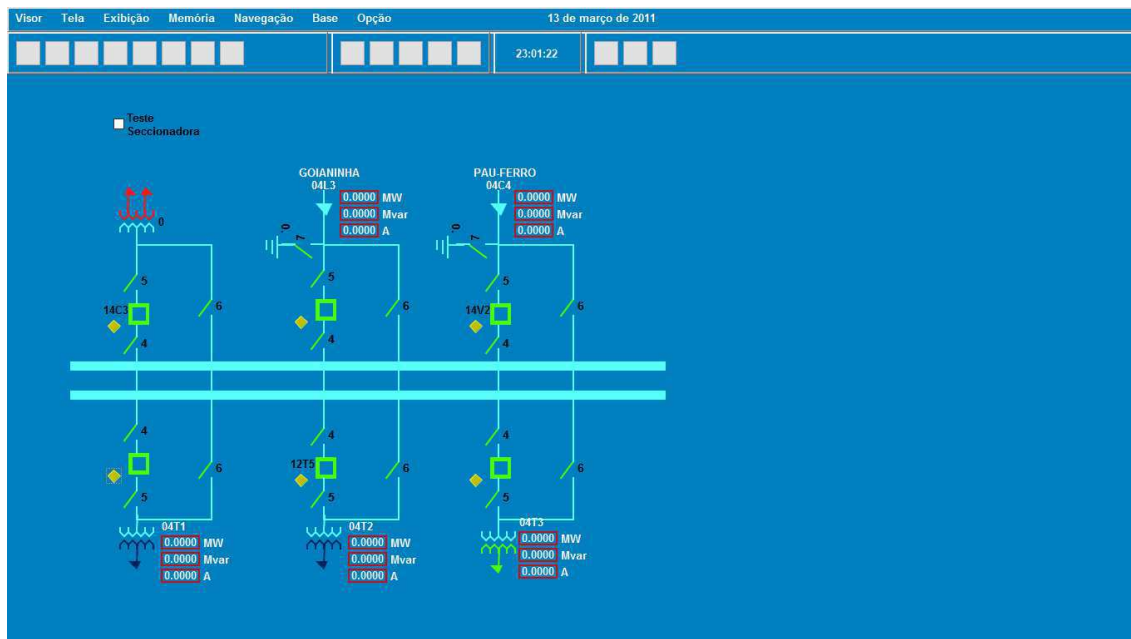




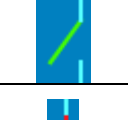
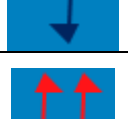
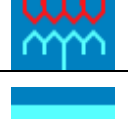

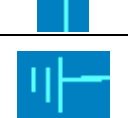
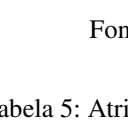
Figura 19: Sinótico do supervisório

Fonte: O próprio autor

Dentre todos os equipamentos implementados na tela do sinótico, apenas os disjuntores e seccionadoras podem ser operados durante a execução do supervisório, já que a versão atual do motor de simulação usado pelo SimuLIHM modela apenas esses dois tipos de equipamento.

Os equipamentos do sinótico foram associados a *tags* que permitem, entre outras coisas, configurar seu estado, definir o código do equipamento, definir o tipo de equipamento, etc. Dado que o Indusoft permite a criação de classes de *tags*, cada uma podendo conter uma quantidade arbitrária de atributos, para a implementação do supervisório apresentado foram definidas três classes: Equipamento, Comando e Login. Na Tabela 5, Tabela 6 e Tabela 7 são apresentadas as listas de atributos para cada classe. Essas classes foram então instanciadas, fazendo parte do banco de *tags* do supervisório representado no simulador.

Tabela 4: Símbolos do sinótico

Símbolo	Descrição
	Disjuntor aberto
	Disjuntor fechado
	Seccionadora aberta
	Seccionadora fechada
	Transformador I
	Transformador II
	Barramento
	Linha de transmissão
	Aterramento

Fonte: O próprio autor

Tabela 5: Atributos da classe Equipamento

Nome	Tipo	Descrição
sCodigo	String	Código do equipamento. Exemplo: 12J5
sId	String	Identificador único do equipamento. Exemplo: CGD:12J5
bEstadoAtual	Inteira	Estado do equipamento (0 – Aberto, 1 – Fechado)
sTipo	String	Tipo do equipamento (Disjuntor, Seccionadora, etc.)
sLinha	String	Código da linha de transmissão. Exemplo: 02J5
sSubestacao	String	Identificador da subestação. Exemplo: CGDII

Fonte: O próprio autor

Tabela 6: Atributos da classe Comando

Nome	Tipo	Descrição
nCodigo	Inteira	Código do comando (0-Abrir; 1-Fechar)
bExecutar	Booleana	Sinalizador de novo comando
sIdEqp	String	Identificador do equipamento acionado
nSelecaoAtual	Inteira	Valor atual selecionado na lista de comandos
sSubestacao	String	Identificador da subestação

Fonte: O próprio autor

Tabela 7: Atributos da classe Login

Nome	Tipo	Descrição
sUsuario	String	Identificador do usuário autenticado
sSenha	String	Senha do usuário autenticado

Fonte: O próprio autor

Para realizar um comando sobre um equipamento durante a execução do supervisor, o usuário deve clicar sobre o mesmo, abrindo assim a janela de comando, como ilustrado na Figura 20.



Figura 20: Janela de ação para os equipamentos do supervisor

Fonte: O próprio autor

Esta janela apresenta informações sobre o equipamento selecionado e permite ao usuário realizar um comando sobre o equipamento. Quando o usuário seleciona o comando a ser realizado e aciona o botão Executar, os atributos presentes na instância da classe Comando são atualizados com as informações do equipamento e a tag bExecutar, sinalizadora de comando, é definida para 1.

Desta forma, o módulo de comunicação com o supervisor do SimuLIHM pode identificar a ocorrência de um novo comando e identificar as informações do equipamento acionado. O supervisor só permitirá que o usuário realize um novo comando quando o último comando realizado tiver sido reconhecido pelo SimuLIHM, que deve definir a *tag* `bExecutar` para 0. O estado do equipamento só será alterado na tela do supervisor quando o SimuLIHM responder ao comando, modificando a *tag* `bEstadoAtual` do equipamento.

4.3 IMPLEMENTAÇÃO DO MÓDULO DE COMUNICAÇÃO COM O SUPERVISÓRIO

A arquitetura do módulo de comunicação com o sistema supervisor especificada na Seção 4.1.2 foi implementada na linguagem de programação orientada a objeto Java, no ambiente de desenvolvimento integrado NetBeans 8.1 para Windows-x64. Nessa seção serão detalhados os principais aspectos relacionados à implementação: configurações realizadas, ferramentas utilizadas e implementação do protocolo OPC.

4.3.1 CONFIGURAÇÃO DO AMBIENTE DE EXECUÇÃO

Para implementação e execução da solução proposta, foi necessário realizar algumas configurações para o Netbeans 8.1 e o sistema operacional Windows 10 x64. A implementação e execução do módulo de comunicação foi realizada usando a plataforma de desenvolvimento Java JDK 1.8 e o ambiente de execução Java JRE 1.8, ambos disponíveis para download no site da ORACLE.

Para possibilitar a comunicação OPC entre aplicações executando em máquinas diferentes, é necessário realizar um conjunto de configurações nos serviços DCOM e no Firewall do Windows. Essas configurações podem ser encontradas em (SOFTWARE TOOLBOX, 2016) e (FARNHAM; PIMENTEL, 2012).

Essas configurações habilitam a comunicação com o computador via DCOM, definindo níveis de autenticação exigidos e permissões para os usuários; definem exceções no Firewall para que os clientes OPC possam acessar, por meio do `OpcEnum`, o registro dos servidores OPC's executando na máquina; e definem permissões de acesso via DCOM ao `OpcEnum` e aos servidores OPC.

Os diferentes níveis de segurança que devem ser configurados para se estabelecer a comunicação OPC são representados na Figura 21.

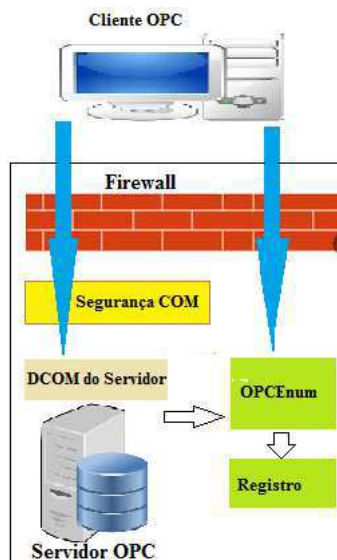


Figura 21: Configuração da comunicação OPC

Fonte: O próprio autor

4.3.2 FERRAMENTAS DE TESTE PARA A COMUNICAÇÃO OPC

Para testar a comunicação OPC podem-se usar ferramentas disponíveis gratuitamente. Neste trabalho de TCC foi usado o cliente MatrikonOPC Explorer 5.0 (Figura 22), disponível para download em (<http://www.matrikonopc.com/products/opc-desktop-tools/opc-explorer.aspx>), e o Toolbox OPC Power Server 5.21 (Figura 23) disponível para download em (<https://www.softwaretoolbox.com/topserver/>).

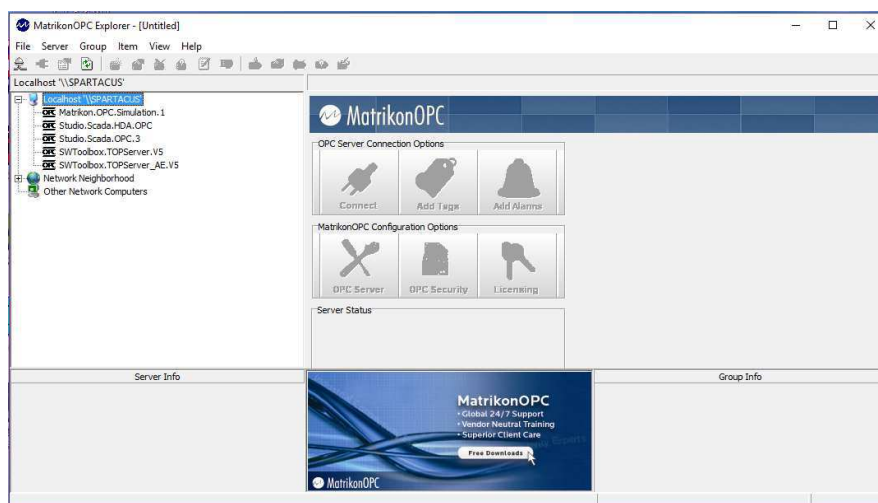


Figura 22: MatrikonOPC Explorer 5.0

Fonte: O próprio autor

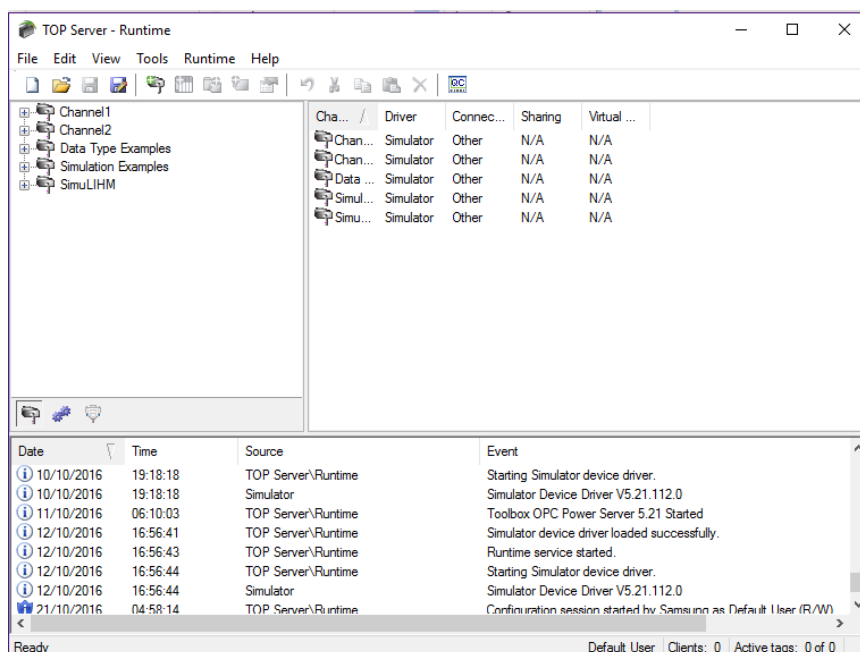


Figura 23: Toolbox OPC Power Server 5.21

Fonte: O próprio autor

4.3.3 DRIVER DE COMUNICAÇÃO OPC

Como apresentado na Seção 4.1.2, o módulo de comunicação com o sistema supervisório permite o uso de diferentes protocolos por meio da implementação de *drivers* de comunicação específicos. Neste trabalho optou-se por realizar a implementação de um driver de comunicação OPC, devido ao elevado nível de interoperabilidade e todas as vantagens oferecidas por este protocolo, como discutido na Seção 2.2.

De modo a realizar a implementação do protocolo em JAVA, realizou-se uma busca por bibliotecas disponíveis. Com isso, as seguintes soluções gratuitas foram encontradas:

- Java Easy OPC Client – Solução em JAVA para comunicação OPC. Faz uso de uma camada JNI (*Java Native Interface*) codificada em Delphi, específico para ambiente Windows. Dentre as funcionalidades disponíveis tem-se o acesso a OpcEnum para a busca de servidores OPC, e navegação nos servidores de dados OPC. O código fonte está disponível em (<https://sourceforge.net/projects/jeasyopc/>);

- Utgard – Solução para cliente OPC completamente implementada em JAVA. O projeto Utgard independe de biblioteca JNI ou DLL (*Dynamic-Link Library*), garantindo assim toda a independência de sistemas operacionais intrínseca a linguagem JAVA. Por meio da biblioteca Utgard pode-se realizar a comunicação com servidores OPC com suporte a OPC DA 2.0. Links para downloads necessários e tutorias para uso dessa biblioteca podem ser encontrados em <http://openscada.org/projects/utgard/>.

A solução escolhida para a implementação do driver de comunicação OPC foi a biblioteca Utgard, por ser uma solução 100% JAVA. As funcionalidades implementadas pelo drive de comunicação são mostradas na Figura 24, e o código em JAVA encontra-se no Apêndice A.

```
public abstract class DriverComunicacaoSupervisorio {  
  
    public abstract void inicializar();  
    public abstract boolean conectar();  
    public abstract void desconectar();  
    public abstract String lerTag(String tagId);  
    public abstract void escreverTag(String tagId, String valor);  
  
}
```

Figura 24: Interface implementada pelo driver de comunicação OPC

Fonte: O próprio autor

Na próxima seção será feita uma avaliação do resultados obtidos para o módulo de comunicação.

5 RESULTADOS

A validação dos resultados foi realizada com base em uma arquitetura física diferente daquela proposta na Seção 4.1.1. Visto que o objetivo principal desta atividade foi a avaliação do módulo de comunicação com o sistema supervisório, usou-se apenas o ambiente tutor juntamente com as ferramentas externas necessárias, ambos executando na mesma máquina, como mostrado na Figura 25.

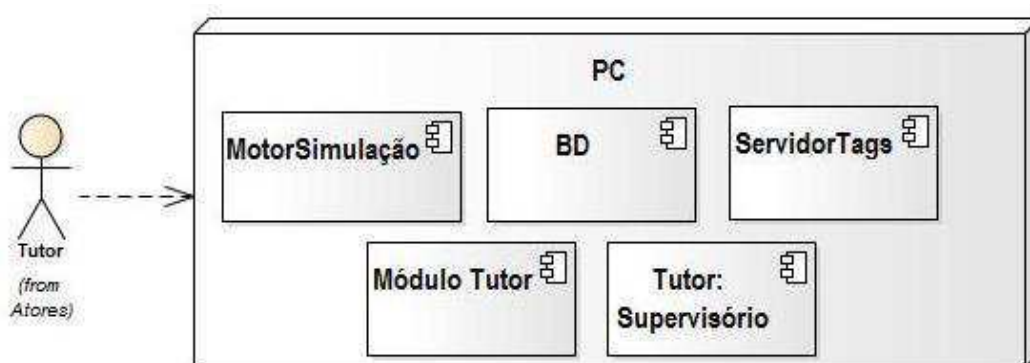


Figura 25: Diagrama de implantação para avaliação dos resultados

Fonte: O próprio autor

Também deve-se ressaltar que durante a avaliação, o módulo de gerenciamento do ambiente virtual foi desativado, devido a problemas encontrados na instalação do ambiente virtual nas máquinas usadas para o desenvolvimento, cuja arquitetura é de 64 bits.

A validação seguiu uma abordagem qualitativa, segundo os critérios a seguir:

- Tempo de resposta: Atraso entre a realização de um comando no sistema supervisório e a visualização da mudança de estado do equipamento na tela. De forma a assemelhar-se ao que ocorre em situações reais de operação do sistema supervisório, este intervalo deve ser de aproximadamente 10 segundos;
- Perda de mensagens: comandos que não resultam em mudança de estado do equipamento no sistema supervisório. Isso pode ocorrer devido a falha na comunicação com o módulo supervisório, falha na comunicação com o motor de simulação, etc.

Na Figura 26 é apresentado o cenário simulado para a avaliação do módulo de comunicação com o supervisor, onde é representada a troca de mensagens entre os módulos do sistema.

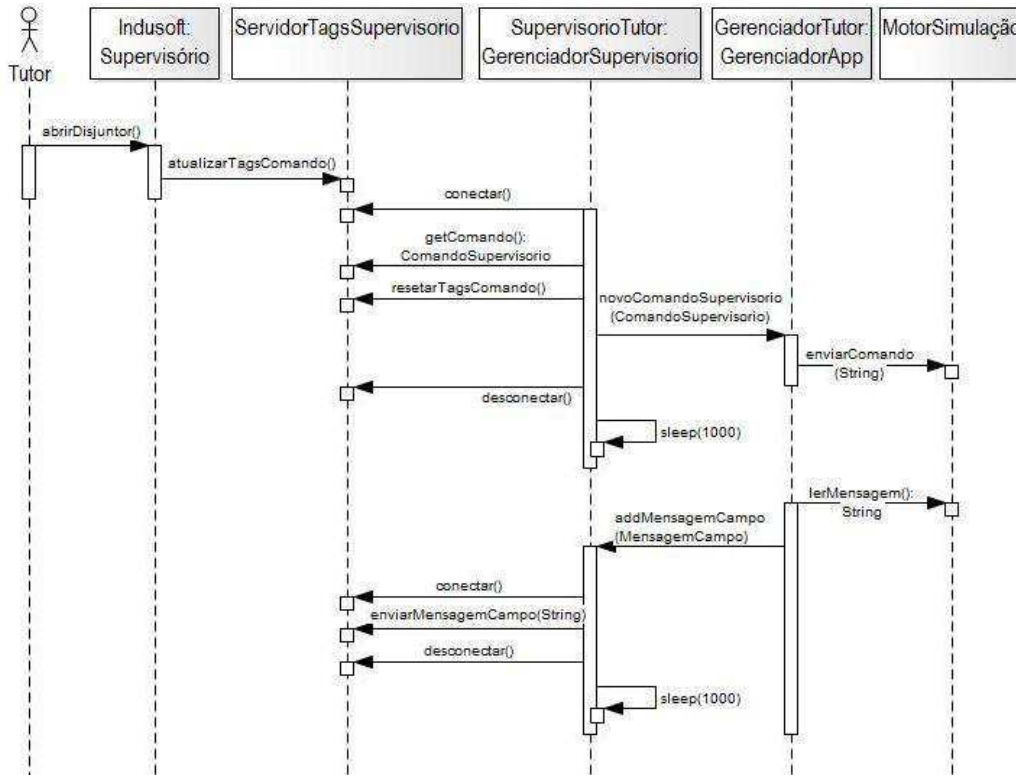


Figura 26: Cenário de teste para o módulo de comunicação com o supervisorio

Fonte: O próprio autor

Neste cenário o Tutor realiza o comando de abertura de um disjuntor na tela do supervisorio Indusoft, fazendo com que as *tags* da classe Comando, no servidor de *tags*, sejam atualizadas com as informações sobre o comando (ID do equipamento, tipo de comando, etc.). Como apresentado no diagrama de atividades da Figura 13, a classe SupervisorioTutor verifica periodicamente a existência de um novo comando no sistema supervisorio. Sendo identificado o comando de abertura do disjuntor, o SupervisorioTutor lê as informações do comando, reinicia as tags de comando do supervisorio, permitindo a chegada de um novo comando, e envia o comando para o gerenciador da aplicação (GerenciadorTutor). O gerenciador da aplicação envia esse comando para o motor de simulação, que de forma assíncrona deve responder com uma mensagem de confirmação. Esta será adicionada a uma lista de mensagens do SupervisorioTutor e enviada para o servidor de *tags*, implicando na atualização do estado do disjuntor na tela do supervisorio.

A avaliação do módulo de comunicação com o supervisor resultou em um tempo de resposta de aproximadamente 10 segundos, considerado satisfatório. Além disso verificou-se a perda de mensagens apenas no caso de comando efetuados sobre a seccionadora CGD:32J67, a qual não estava registrada no motor de simulação. Portanto, para o cenário avaliado, os resultados obtidos validaram o módulo de comunicação especificado.

6 CONCLUSÃO

O desenvolvimento realizado durante este projeto de TCC alcançou todos os objetivos propostos. Por meio da especificação arquitetural gerada, a manutenção e evolução da plataforma SimuLIHM pode ser realizada de forma estruturada e robusta. O módulo de comunicação desenvolvido e integrado à plataforma proporciona um elevado grau de interoperabilidade e possibilita a fácil extensão da ferramenta para integração com diferentes sistemas supervisórios e protocolos de comunicação. Além disso, o *driver* OPC implementado pode ser reutilizado em diferentes aplicações JAVA que precisem exercer o papel de cliente OPC.

Este trabalho abre a oportunidade para desenvolvimento de diversos trabalhos futuros. Visto que a avaliação da ferramenta foi realizada em uma arquitetura local, como apresentado na Seção 5, pode-se expandir essa avaliação para uma arquitetura distribuída, como proposto na Seção 4.1.1, e verificar o comportamento do sistema durante a realização de um treinamento com diversos operadores.

Como o desenvolvimento original do simulador SimuLIHM não foi realizado com base em uma arquitetura bem definida e detalhada, tal como aquela especificada na Seção 4.1, o código da ferramenta precisa ser completamente refatorado e adequado a arquitetura proposta neste TCC, com exceção do módulo de comunicação com o supervisório o qual foi implementado durante a realização deste trabalho.

A implementação das telas no supervisório Indusoft teve como principal objetivo possibilitar a validação do módulo de comunicação implementado, uma vez que a implementação de telas no Intouch realizada por (TORRES FILHO,2011) não encontra-se mais disponível no LIHM. Desta forma, várias telas presentes no supervisório SAGE não foram implementadas, como a tela de alarmes, o que pode ser realizado em desenvolvimentos futuros. Embora deva-se destacar que estas telas não eram indispensáveis aos objetivos deste trabalho.

Neste TCC foi feita a implementação do protocolo de comunicação OPC, que apresenta um alto grau de interoperabilidade e aceitação no meio industrial. As funcionalidades implementadas pelo *driver* OPC foram as exigidas pela interface do *driver* de comunicação. Outras funcionalidades envolvidas em comunicações OPC

podem ser adicionadas, como a manipulação de grupos de *tags*, a navegação na árvores de *tags* do servidor, etc.

Existem vários outros protocolos de comunicação que são bastante utilizados em automação industrial (Modibus, Profibus, etc.). A arquitetura especificada para o módulo de comunicação permite que estes protocolos sejam implementados de forma isolada, como drivers de comunicação, e facilmente integrados à plataforma de treinamento SimuLIHM.

Por fim, um trabalho de grande relevância que pode dar sequência à integração do SimuLIHM com diferentes sistemas supervisórios, é a integração da plataforma ao supervisório SAGE. Por meio dessa integração, evita-se a necessidade de implementação das telas desse supervisório, como realizado neste trabalho, uma vez que estas já são disponíveis nas empresas do setor elétrico que utilizam o SAGE para a operação do sistema elétrico, tal como a CHESF.

REFERÊNCIAS

- NASCIMENTO NETO, José Alves. **Processo para concepção de estratégias para prevenção do erro na operação de sistemas elétricos**. 2010. Tese (Doutorado) - Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, Campina Grande, 2010.
- TORRES FILHO, Flávio. **Desenvolvimento de um módulo supervisor para um ambiente de treinamento de operadores**. 2011. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, Campina Grande, 2011.
- MAMEDE FILHO, João. **Instalações Elétricas Industriais**. 7. ed. Rio de Janeiro: LTC, 2007.
- PAIOLA, Carlos E. G. **O Papel do Supervisor no Atual Contexto Tecnológico**. Intech, p.6-18. Disponível em: <http://www.aquarius.com.br/Boletim/InTech132_artigo.pdf>. Acesso em: 25 out. 2016.
- LEITE, Carlos Roberto R.; RODRIGUES, João José; OLIVEIRA, Jaldemir Gomes de. O Uso de Simuladores no Treinamento de Operadores da CHESF como Ferramenta para Disseminação de Conhecimentos na Operação do Sistema Elétrico. In: Seminário internacional: reestruturação e regulação do setor de energia elétrica e gás natural, 2, 2007, Rio de Janeiro. **Anais Eletrônicos do II Seminário Internacional: Reestruturação e Regulação do Setor de Energia Elétrica e Gás Natural**. Rio de Janeiro: UFRJ, 2007. p. 1-27. Disponível em: <http://www.nuca.ie.ufrj.br/gesel/eventos/seminariointernacional/2007/artigos/pdf/carlosrobertoleite_ousodesimuladores.pdf>. Acesso em: 10 jan. 2011.
- RIBBON (computing). 2016. Disponível em: <[https://en.wikipedia.org/wiki/Ribbon_\(computing\)](https://en.wikipedia.org/wiki/Ribbon_(computing))>. Acesso em: 25 out. 2016.
- OPC FOUNDATION. **What is OPC?** Disponível em: <<https://opcfoundation.org/about/what-is-opc/>>. Acesso em: 25 out. 2016.
- ELPROCUS. **Optimum Idea about an OPC Server in Industrial Control Systems**. Disponível em: <<https://www.elprocus.com/why-is-opc-server-needed-for-industrial-control-systems/>>. Acesso em: 25 out. 2016.
- SILVA, Victor Navarro A. L. Da. et al. Simuladores para treinamento de operadores de sistema e de instalações do setor elétrico. In: Encuentro Regional Iberoamericano De CIGRÉ, 13, 2009, Puerto Iguazú. **Anales del décimo tercer encuentro regional iberoamericano de CIGRÉ**. Puerto Iguazú: XIII Eriac, 2009. p. XIII/PI-C2-04/01-10.
- GOMAA, H. **Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures**. Fairfax: Cambri, 2011.
- FREEMAN, Eric et al. **Head First Design Patterns**. Sebastopol: O'reilly Media, Inc., 2004.
- SOFTWARE TOOLBOX (Matthews). **DCOM Tutorial**. Disponível em: <https://www.softwaretoolbox.com/dcom/html/dcom_for_windows_7-8-_-server_2008.html>. Acesso em: 25 out. 2016.

FARNHAM, Benjamin; PIMENTEL, Marco. **Setting DCOM for OPC under Windows 7 (32/64 bit)**. 0.1 Genève: European Laboratory For Particle Physics, 2012. Disponível em: <ftp://ftp.nist.gov/pub/mel/michalos/Software/Github/MTCconnectSolutions/MtcOpcAgent/doc/DCOM_Config_Step_by_Step.pdf>. Acesso em: 25 out. 2016.

APÊNDICE A – CÓDIGO JAVA PARA O DRIVER DE COMUNICAÇÃO OPC

```

package br.edu.ufcg.dee.lihm.supervisorio;

import java.net.UnknownHostException;
import java.util.concurrent.Executors;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.jinterop.dcom.common.JIException;
import org.jinterop.dcom.core.JIVariant;
import org.openscada.opc.lib.common.*;
import org.openscada.opc.lib.da.*;

public class DriverOPC extends DriverComunicacaoSupervisorio {

    private ConnectionInformation ci;
    private Server server;
    private AutoReconnectController autoReconnectController;
    private String host;
    private String dominio;
    private String usuario;
    private String senha;
    private String clsid;

    public DriverOPC() {

        // Informar os dados da conexão
        this.host = "localhost";
        this.dominio = "default";
        this.usuario = "Samsung";
        this.senha = "Kondera9";
        this.clsid = "6AA77893-6DEF-412A-B910-38B89804E06D";

        this.ci = new ConnectionInformation();
        inicializar();
    }

    /**
     * Inicializa o driver OPC, configurando o servidor que será conectado
     *
     */
    public void inicializar() {

```

```

// create connection information
this.ci.setHost(this.host);
this.ci.setDomain(this.dominio);
this.ci.setUser(this.usuario);
this.ci.setPassword(this.senha);
this.ci.setClsid(this.clsid);// Studio Scada OPC

// create a new server
this.server = new Server(ci, Executors.newSingleThreadScheduledExecutor());
this.autoReconnectController = new AutoReconnectController(server);
}

/**
 * Realiza a conexão com o servidor OPC
 */
@Override
public boolean conectar() {
    try {
        autoReconnectController.connect();
        Thread.sleep(400);
    } catch (Exception e) {
        return false;
    }

    return true;
}

/**
 * Desconecta o servidor OPC
 */
@Override
public void desconectar() {
    try {
        autoReconnectController.disconnect();
        Thread.sleep(100);
    } catch (InterruptedException ex) {
        Logger.getLogger(DriverOPC.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * Ler o valor de uma tag no servidor OPC
 */
@Override
public String lerTag(String tagId) {
    final AccessBase access;
    final Group group;
    final Item item;
    JIVariant valorJIVariant;
    String valor = "";

```

```

try {
    // Add sync access, poll every 500 ms
    access = new SyncAccess(server, 500);

    // Add a new group
    group = server.addGroup("temp");

    // Add a new item to the group
    item = group.addItem(tagId);

    //Leitura do servidor
    valorJIVariant = item.read(false).getValue();

    switch (valorJIVariant.getType()) {
        case JIVariant.VT_BSTR: //Tipo String
            valor = valorJIVariant.getObjectAsString2();
            break;
        case JIVariant.VT_BOOL:
            valor = "" + valorJIVariant.getObjectAsBoolean();
            break;
        case JIVariant.VT_I4: //Tipo inteiro
            valor = "" + valorJIVariant.getObjectAsInt();
            break;

        default:
            break;
    }

    Thread.sleep(100);

    //Remove group
    server.removeGroup(group, true);

} catch (IllegalArgumentException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (NotConnectedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (JIException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (DuplicateGroupException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (AddFailedException e) {

```

```

        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return valor;

}

/**
 * Altera o valor de uma tag no servidor OPC
 */
@Override
public void escreverTag(String tagId, String valor) {
    final AccessBase access;
    final Group group;
    final Item item;

    try {
        // Add sync access, poll every 500 ms
        access = new SyncAccess(server, 500);
        // Add a new group
        group = server.addGroup("temp");
        // Add a new item to the group
        item = group.addItem(tagId);

        //Escrita no servidor OPC
        item.write(new JIVariant(valor));
        Thread.sleep(100);

        //Remove group
        server.removeGroup(group, true);

    } catch (IllegalArgumentException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NotConnectedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (JIException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (DuplicateGroupException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

```
    } catch (AddFailedException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    } catch (InterruptedException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}  
}
```