Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Unidade Acadêmica de Engenharia Elétrica
Programa de Pós-Graduação em Engenharia Elétrica

Dissertação de Mestrado

Arquitetura para um ambiente de treinamento representado em Realidade Virtual

Ademar Virgolino da Silva Netto

Orientadora: Maria de Fátima Queiroz Vieira, Ph.D.

Campina Grande, Junho de 2010.

Arquitetura para um ambiente de treinamento representado em Realidade Virtual

Ademar Virgolino da Silva Netto

Dissertação de mestrado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências no domínio da Engenharia Elétrica.

Área de Concentração
Engenharia da computação.

Orientadora
Maria de Fátima Queiroz Vieira, PhD.

S586a Silva Netto, Ademar Virgolino da
Arquitetura para um ambiente de treinamento representado em realidade virtual / Ademar Virgolino da Silva Netto. - Campina Grande, 2010.
131 f. : il.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal de Campina Grande, Centro de Engenharia Eletrica e Informatica.

1. Simulacao Utilizando Realidade Virtual 2. Sistema para Treinamento de Operadores 3. Arquitetura de um Simulador 4. Dissertacao I. Vieira, Maria de Fatima Queiroz, Dra. II. Universidade Federal de Campina Grande - Campina Grande (PB) III. Título

CDU 004.5(043)

ARQUITETURA PARA UM AMBIENTE DE TREINAMENTO REPRESENTADO EM REALIDADE VIRTUAL

ADEMAR VIRGOLINO DA SILVA NETTO

Dissertação Aprovada em 14.06.2010

Maria de Fatima Q. Vieira

MARIA DE FÁTIMA QUEIROZ VIEIRA, Ph.D., UFCG
Orientadora

ANGELO PERKUSICH, D.Sc., UFCG
Componente da Banca

LEANDRO DIAS DA SILVA, D.Sc., UFAL
Componente da Banca

CAMPINA GRANDE - PB
JUNHO - 2010

Dedicatória

Aos meus pais e minhas irmãs.

Agradecimentos

Agradeço a Deus. Aos meus Pais pelo incentivo nas minhas decisões, apoio e força nas horas difíceis, e por sempre me aconselharem. As minhas Irmãs pelas conversas, papos em inglês e pelos momentos de grande alegria nas reuniões de família.

Agradeço aos meus avós que sempre me deram suporte quando precisei. Aos meus Tios por me acolherem em Campina Grande e aos, que mesmo distante, sempre torceram por mim.

Agradeço a Fernanda Karoline, minha namorada, que sempre me apoiou, me ajudou, compreendeu e deu carinho nos momentos mais difíceis e a toda sua família por sempre me acolherem, incluindo a turma do Mirante.

Aos meus colegas de Laboratório (Daniel, Alves, Raffael, Flávio, Yuska e não poderia esquecer, da nossa "agregada" Thiciany) que são as pessoas com quem mais convivi nestes dois anos, pelos momentos no café, sugestões, críticas, amizade e pelo ótimo ambiente de trabalho.

Aos meus professores, em especial a professora e orientadora Maria de Fátima por acreditar no meu potencial, compartilhar seus conhecimentos e orientar na execução deste trabalho.

Aos funcionários da COPELE (Ângela e Pedro) que sempre foram muito prestativos e atenciosos.

A todos os meus amigos, Manéricio (José Mauricio), Dona Teresinha, Flávio Brito, Osman, Larissa, Lorena, João André, Taísa, Luiz Paulo, Diego (chuck), Rubem, Taciana e Diego Linhares e a todos que não retratei no momento, que seja de longe ou de perto, mas que de alguma maneira contribuíram na minha formação, no desenvolvimento do trabalho, nos momentos de alegria ou que simplesmente sempre torceram por mim.

Resumo

O uso de ambientes virtuais para treinamento de operadores tem se consolidado devido a fatores como, redução de custo, redução de risco e treinamento personalizado, além de possibilitar o treinamento antecipado em equipamentos ainda não instalados. Porém, a maioria dos simuladores utiliza linguagens de programação, o que dificulta a manutenção e evolução de suas funções. Este trabalho faz parte de um projeto desenvolvido no Laboratório de Interface Homem-Máquina (LIHM) da Universidade Federal de Campina Grande (UFCG). Trata-se de um simulador cujo motor de simulação é implementado a partir de modelos em Redes de Petri Coloridas (CPN) e tem como propósito é apoiar o treinamento de operadores na sala de, e apoiar o estudo do erro humano nestes ambientes. Este trabalho teve como objetivos propor uma arquitetura para o simulador capaz de contemplar um novo conjunto de requisitos (modularidade, interoperabilidade e persistência) e integrar o motor de simulação com os módulos propostos na arquitetura. Este texto apresenta a especificação da arquitetura, sua implementação a partir de ferramentas livres e a integração entre módulos da arquitetura que foi instanciada e validada. O contexto de aplicação do simulador foi o de sistemas elétricos, na operação de uma subestação elétrica. Como resultado deste trabalho, a configuração básica do simulador foi construída e validada.

Palavras chave: Simulação utilizando Realidade Virtual, Sistema para Treinamento de Operadores, Arquitetura de um simulador.

Abstract

The use of virtual environments for operator training has become a routine in many domains. Its advantages include cost reduction and personalized training giving the operator the opportunity to progress in its own rhythm, and the possibility of being trained in the use of new machinery and technologies. The vast majority of those virtual environment simulating tools are program-based which makes difficulty its maintenance and functional evolution. The Human Interface laboratory (LIHM) has developed a simulator prototype that uses as a model based simulator engine. The model is built in Colored Petri Nets (CPN), and the entire simulator was built using freeware software. As a drawback, one of the software tools stopped supporting an important functionality that prevented the prototype from being operational. This work proposes a new architecture for the simulator, which was conceived for supporting the training of operators in a substation control room as well as to support the human error studies performed in the LIHM. This new architecture contemplates new requisites such as persistence and and the appropriate developing tools are proposed as part of this work. As a result of this work, a basic simulator configuration was built validating the proposed architecture. To build this initial prototype it was also developed a driver to interconnect the various modules proposed in the architecture.

Key Words: Simulation based on Virtual Reality, Operator Training System, Simulator architecture.

Lista de abreviaturas

3D: Três dimensões

API: *Application Programming Interface*

AV: Ambiente Virtual

BD: Banco de Dados

CAVE: *Cave Automatic Virtual Environment*

CEPEL: Centro de Pesquisas de Energia Elétrica

CPN: *Colored Petri Net*

DTD: *Document Type Definition*

EAI: *External Authoring Interface*

ESOPE-VR: *Expert System Operations Planning Environment – Virtual Reality*

GIHM: Grupo de Interface Homem Máquina

HTML: *HyperText Markup Language*

HTTP: *Hypertext Transfer Protocol*

IP: *Internet Protocol*

ISO: *International Organization for Standardization*

JDBC: *Java DataBase Connectivity*

LIHM: Laboratório de Interface Home- Máquina

MSC: *Message Sequence Charts*

PDA: *Personal digital assistants*

RV: Realidade Virtual

SAGE: Sistema Aberto de Supervisão e Controle

SAI: *Scene Access Interface*

SCADA: *Supervisory Control And Data Aquisition*

SGBD: Sistema de Gerenciamento de Banco de Dados

SML: *Standard ML*

STPO: Simulador para Treinamento de Proteção e Operação

TCP: *Transmission Control Protocol*

UML: *Unified Modeling Language*

VRML: *Virtual Reality Modeling Language*

X3D: *eXtensible 3D*

XML: *eXtensible Markup Language*

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Sumário

Capítulo 1 Introdução

A representação de ambientes interativos compostos por objetos tridimensionais (3D) tornou-se comum entre os usuários de computadores. Estes ambientes virtuais (AV) podem ser atualizados em tempo real recriando mundos virtuais com os quais um ou mais usuários podem interagir, visualizar e manipular os objetos.

Com o desenvolvimento da tecnologia de suporte à concepção de ambientes virtuais para representar ambientes reais, foi ampliado o campo de pesquisa na área de educação e treinamento utilizando objetos tridimensionais (3D). Dentre os trabalhos pesquisados destacam-se: Marsoli et al (2006) que utilizam Realidade Virtual (RV) no ensino e aprendizagem de conceitos da matemática; Silva & Rogado (2008) no ensino de química; e Hermosilla & Santos (2006) no estudo da estrutura óssea equina. Na área de treinamento, Souza et al. (2006) conceberam um sistema interativo para o treinamento na realização de exame ginecológico (SITEG), enquanto Machado & Zuffo (2002) desenvolveram um simulador de procedimentos médicos invasivos para o treinamento no transplante de Medula Óssea; Grave et al (2001) desenvolveram um simulador para treinamento na montagem de cabos elétricos; e a Universidade de São Paulo em conjunto com a empresa Vale, estão desenvolvendo um simulador de trens e ferrovias para treinamento de maquinistas (AGENCIA USP, 2009). Porém a maioria destes trabalhos não apresenta a arquitetura utilizada no desenvolvimento.

A indústria cada vez mais, demanda ambientes de treinamento para seus operadores, pois o uso dos equipamentos pode colocar em risco a vida dos operadores e em muitos casos os equipamentos não podem ser desligados para execução dos treinamentos. No caso específico dos ambientes de subestação elétrica são cada vez maiores as exigências por qualidade no fornecimento de energia e a cada falta de energia são cobradas multas exorbitantes. Fazendo que os treinamentos fiquem cada vez mais difíceis de ocorrer nos equipamentos.

A partir da evolução do uso dos ambientes em Realidade Virtual e devido à necessidade de treinar os operadores de sistemas industriais, surgiu o projeto de um simulador para estes ambientes no Laboratório de Interfaces Homem-Máquina (LIHM) da Universidade

Federal de Campina Grande (UFCG) (FREITAS, 2006b). Este simulador proporciona às empresas e operadoras de energia uma forma de treinamento realista sem o risco da ocorrência de erros que podem afetar a operação do sistema.

Com o propósito de expandir o projeto do simulador constatou-se a necessidade de propor um novo projeto arquitetural. A proposição de uma arquitetura foi necessária para estruturar o desenvolvimento do sistema em módulos e etapas, com base em metas e requisitos, que representem as funcionalidades desejadas; e de acordo com as tecnologias e ferramentas adequadas ao desenvolvimento do simulador. A modularidade da arquitetura teve como objetivo permitir a fácil adequação a diferentes contextos de aplicação.

Esta arquitetura apóia a equipe de desenvolvimento do simulador para treinamento de operadores, o qual atualmente representa o contexto de uma sala de controle e supervisão de uma subestação elétrica. Com o treinamento apoiado pelo simulador, a qualidade da operação do sistema pode ser melhorada a partir da redução do erro humano. O ambiente simulado apoiará, além do treinamento, a observação do contexto no qual ocorrem as falhas humanas contribuindo assim para identificar os problemas relacionados à ergonomia da interface de operação.

Através do simulador será possível avaliar modos mais eficientes de realizar tarefas, formas de reduzir a sobrecarga cognitiva do operador, treinar a tomada de decisões em condições críticas e situações atípicas cujo tempo de execução é estrito. O simulador, na sua versão inicial, foi construído a partir de diversos trabalhos realizados no Laboratório de Interfaces Homem Máquina (LIHM), dentre os quais: SOUZA (1999); NASCIMENTO et al (2007); SCAICO et al (2001). No entanto, com a evolução dos requisitos originais, os quais serão discutidos mais adiante, surgiu à necessidade de uma nova arquitetura capaz de acomodá-los.

1.1 Objetivos

O objetivo principal deste trabalho é propor uma arquitetura para o projeto do simulador que contemple os requisitos:

- flexibilidade na adequação a diferentes contextos de treinamento, com reuso de código;
- portabilidade;
- ambiente:
    - colaborativo;

- multiusuário;
- distribuído (cliente – servidor).

Embora originalmente o projeto do simulador utilizasse um motor de simulação[1] à base de modelos, os modelos eram bastante simples. Um dos objetivos do trabalho é ampliar e aperfeiçoar esses modelos. Outro problema é que a comunicação entre o motor de simulação construído em Redes de Petri Coloridas (CPN) com a representação em Realidade Virtual (RV) foi perdida, devido à nova versão do visualizador de RV não dar mais suporte a tecnologia utilizada no desenvolvimento. Consequentemente foi construída uma versão do simulador cujo motor de simulação simplificado foi escrito em Java, no entanto o custo de atualizar e adequar esta versão é muito alto, pois a inserção de novos objetos e alterações no funcionamento demanda muito tempo.

Portanto, a principal meta deste trabalho foi recuperar o mecanismo de comunicação entre o motor de simulação (baseado em Modelos) e o Ambiente Virtual (AV), e a partir desta solução propor uma nova arquitetura para o simulador. A nova arquitetura deve também contemplar as novas funcionalidades propostas para o simulador. Para desenvolver este projeto foi necessário estudar as linguagens de representação de Ambientes Virtuais e as ferramentas de construção e visualização, além de migrar e adaptar a biblioteca de modelos em CPN (desenvolvidas para o motor de simulação) para a ferramenta CPN Tools (CPN TOOLS, 2009).

1.2 Motivação

O número de operadores de Subestações Elétricas (SE) experientes tem diminuído, devido à aposentadoria dos profissionais mais experientes e a falta de transmissão sistemática dos conhecimentos por eles armazenados. Por isto, é necessário que o operador tenha treinamentos de aprendizado e reciclagem. Através do projeto do simulador pode-se aproveitar o tempo ocioso (quando a subestação está sobre controle e fora dos horários de picos de consumo), para treinamento dos operadores (SILVA et al, 2009).

O projeto do simulador e suas demandas na área de pesquisa do LIHM (além do interesse despertado na indústria enquanto ferramenta de treinamento) foram os grandes motivadores deste trabalho. Além do enriquecimento acadêmico, pois através do simulador o autor pôde refinar seu conhecimento na modelagem em Redes de Petri Coloridas, a programar

[1] Motor de simulação - neste trabalho, se refere a um conjunto de modelos descritos em redes de Petri Coloridas (CPN) que uma vez executados representam o comportamento dos objetos do ambiente simulado, no caso o ambiente de supervisão de uma subestação elétrica.

na linguagem Java para desenvolver o interpretador de mensagens (*driver*) entre o AV e as CPN, e a modelar Ambiente Virtuais em 3D.

1.3 Metodologia

Para a execução deste trabalho foi adotada a seguinte metodologia:

- Estudo de Realidade Virtual: foram estudadas as linguagens para representação de ambientes virtuais, fazendo um comparativo entre os existentes (X3D, Java3D, VRML, entre outras) e recomendando uma para utilização (apresentado no Anexo A);
- Escolhida a linguagem de representação foi realizado um levantamento de ferramentas para visualizar e construir sistemas em Realidade Virtual, levando em consideração a compatibilidade com Windows/Linux e o suporte à linguagem proposta (ver Anexo A);
- Estudo dos modelos que compõem o motor de simulação: foi estudada a biblioteca de modelos que representa o comportamento dos objetos de uma subestação em Redes de Petri Coloridas (NASCIMENTO et al, 2007) e posteriormente realizada a migração desta biblioteca (construída com a ferramenta Design/CPN) (DESIGN/CPN, 2009) para o ambiente CPN Tools;
- Levantamento dos novos requisitos do simulador;
- Pesquisa de outras aplicações com propostas semelhantes: fazendo um comparativo entre as aplicações estudadas;
- Proposta de uma arquitetura para o projeto do simulador;

Após propor a arquitetura o trabalho objetivou no desenvolvimento do módulo de comunicação da arquitetura que conecta o mundo virtual aos modelos CPN. Para isto foi desenvolvido o *driver* para comunicação entre as CPN que representam o funcionamento dos painéis de comando elétrico e o Ambiente Virtual.

- Para este desenvolvimento as redes CPN foram adaptadas e convertidas para receber e enviar mensagens para o AV. Além disso, foi solucionado o problema de comunicação da ferramenta CPN Tools, pois a função *Accept* não foi desenvolvida na versão 2.2.0 do CPN Tools;
- Propor ferramentas para apoiar o desenvolvimento e a execução do AV, converter de VRML para X3D e aprimorar o AV;
- Apresentar o *driver* de comunicação desenvolvido;
- Apresentar casos de uso do simulador;

- Apresentar sugestões para as próximas etapas do projeto.

1.4 Estrutura da Dissertação

Este documento está organizado da seguinte forma: no capítulo 2 uma revisão de conceitos e trabalhos que concernem às arquiteturas de Ambientes Virtuais com propósitos semelhantes (ensino e treinamento); em seguida são apresentados os requisitos da arquitetura para o simulador; no capítulo 3 é apresentada a arquitetura proposta com base nos requisitos para seu desenvolvimento; no capítulo 4 é apresentado o *driver* de comunicação, desenvolvido neste trabalho para integrar o motor de simulação ao Ambiente Virtual do simulador, e uma validação da arquitetura a partir da nova versão do simulador; seguido no capítulo 5 pelas considerações finais e sugestões de continuidade.

Posteriormente as referências, o glossário e os anexos. Sendo os anexos divididos em quatro partes: Anexo A – Tecnologias para construção de Ambientes Virtuais; Anexo B – Biblioteca Comms/CPN; Anexo C – Detalhes da Biblioteca de modelos; e Anexo D – Códigos gerados para o desenvolvimento do projeto.

Capítulo 2 Realidade Virtual e aplicações

Para o desenvolvimento deste trabalho foi necessário estudar Realidade Virtual, o projeto original do simulador, fazer um levantamento das aplicações desenvolvidas em Realidade Virtual para o contexto de ensino e treinamento, e especificar os requisitos do simulador que embasaram a arquitetura proposta. Este capítulo aborda os tópicos citados.

2.1 Realidade Virtual

O termo Realidade Virtual (RV) surgiu na década de 80 e é creditado a Jaron Lanier, devido à necessidade de diferenciar as simulações tradicionais dos mundos digitais (MACHADO apud MACHOVER, 1997). Porém o termo é abrangente e os desenvolvedores de software e pesquisadores da área definem Realidade Virtual baseados em sua experiência.

Kirner (CARDOSO et al, 2008) definem Realidade Virtual como uma "interface avançada do usuário" para acessar aplicações executadas no computador, propiciando a visualização, movimentação e interação do usuário, em tempo real, em ambientes tridimensionais gerados por computador. Para (GARCIA et al, 2001), RV é uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos. O conceito adotado neste trabalho é que Realidade Virtual é uma interface avançada para aplicações computacionais, que permite ao usuário navegar e interagir, em tempo real, através de um ambiente tridimensional gerado por computador, usando dispositivos multi-sensoriais (mouse, teclado, luva de dados, capacete, entre outros).

Dois fatores importantes devem ser considerados em sistemas de RV são eles a imersão e a interatividade. A imersão pela facilidade de capturar a atenção do usuário, e a interatividade no que diz respeito à comunicação usuário-sistema. Quando se trata de imersão, alguns autores a classificam de acordo com o grau de interação com o ambiente ou com base nos equipamentos que são utilizados na interação. A imersão é classificada da seguinte forma:

a) Realidade Virtual não-imersiva (ou Imersão subjetiva): a simulação do ambiente no plano (bi-dimensional), através de monitores que simulam e exploram as projeções, perspectivas e as transformações geométricas, causando a impressão ao usuário, de estar imerso no Ambiente Virtual.

b) Realidade Virtual projetada (ou Imersão subjetiva com projeção do usuário): o usuário se vê no Ambiente Virtual, com sua imagem projetada em uma, ou em várias telas, a exemplo de CAVE (*Cave Automatic Virtual Environment*) e telas panorâmicas (CARDOSO et al, 2008).

c) Realidade Virtual imersiva (Imersão espacial): neste caso, além da apresentação das imagens, são utilizados sensores de movimento/posição acoplados ao corpo do usuário, possibilitando-lhe uma interação direta com o ambiente. Tipicamente são utilizados: capacetes; luvas e macacões dotados de sensores de movimentos. Aumentando a sensação de imersão já que os movimentos são replicados no Ambiente Virtual.

O uso de capacetes e projeções em paredes (CAVE) vem evoluindo, no entanto a RV cuja interação se dá através de monitores, mouse e teclado permanece relevante, pois exploram recursos acessíveis evitando limitações técnicas e dificuldades no uso de equipamentos como luvas e capacetes. Nos testes realizados no laboratório de usabilidade (LIHM) da UFCG, com a versão original do simulador, constatou-se que o uso de recursos comuns como monitor, teclado e mouse produzem resultados satisfatórios do ponto de vista da imersão do usuário no ambiente simulado e da facilidade na execução das tarefas que lhe foram propostas.

Este trabalho abordará apenas a Realidade Virtual não-imersiva, com o propósito de reduzir o custo do hardware utilizado na interação com o Ambiente Virtual e pela mesma razão adotará o uso de software livre na concepção e execução do simulador.

2.2 Projeto do simulador

Com o propósito de analisar o componente da navegação em projetos de interfaces com usuários, a equipe do LIHM passou a modelar este componente utilizando métodos formais. Os primeiros trabalhos "Avaliação Interativa da Especificação de Interfaces com Ênfase na Navegação" de (SOUZA, 1999) & (TURNELL et al, 2001), contavam apenas com os recursos oferecidos pela ferramenta Design/CPN para analisar os modelos de navegação construídos em Redes de Petri Coloridas (CPN).

Com o propósito de permitir a análise da navegação para projetistas leigos no formalismo, foi então proposta a associação de uma representação gráfica que apresentasse mais facilmente o contexto modelado (FREITAS et al, 2006b). Foi então associado ao modelo CPN da interface uma representação 3D, em Realidade Virtual utilizando a linguagem VRML (V*irtual Reality Modeling Language*). Em seguida foi construída uma biblioteca de modelos

(NASCIMENTO et al, 2007) para representar os objetos de interação tipicamente encontrados em uma subestação; os quais passaram a ser integrados ao modelo de navegação, para representar uma instalação específica.

Na Figura 2-1 é ilustrada a representação 3D de uma Subestação Elétrica (SE), construída na linguagem VRML (FREITAS, 2006a); constituindo assim a primeira versão do simulador. Nesta versão, os painéis de controle e o comportamento dos objetos de interação (chaves e botões) são representados em um modelo CPN. A conexão entre o modelo CPN e a representação visual dos objetos utilizava a biblioteca Comms/CPN e *Java Script Interface* (JSI) (HORSTMAN, 2004). O visualizador de RV utilizado foi o Freewrl (FREEWRL, 2009) para plataforma Linux.

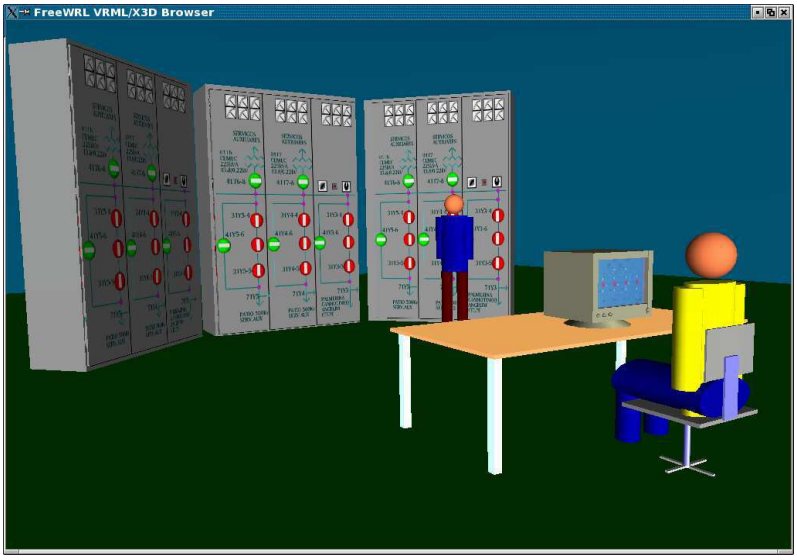

Figura 2-1 – Interfaces elaboradas por FREITAS. (Fonte: FREITAS, 2006a).

Na Figura 2-2 é ilustrada a arquitetura utilizada por (FREITAS, 2006a), o acesso ao Ambiente Virtual era exclusivamente local e o AV construído era simples, representando apenas os objetos de interação sem se preocupar em representar o ambiente físico da sala de comando. Esta versão do simulador veio a tornar-se inoperante em razão do visualizador Freewrl deixar de oferecer suporte ao método JSI, no qual foi desenvolvida a comunicação entre o CPN e o AV. Assim, utilizando a ferramenta Freewrl, ficou como alternativa para comunicação do AV com processos externos a EAI (*External Authoring Interface*).

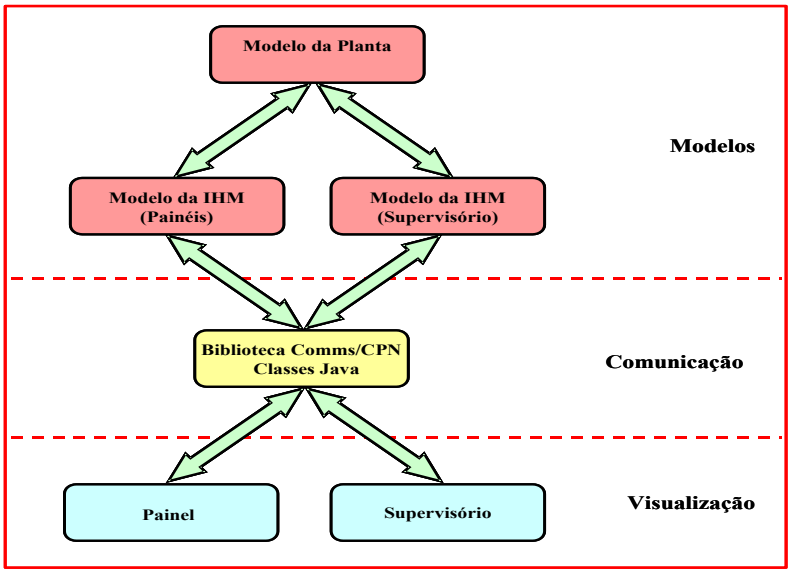

Figura 2-2 – Arquitetura utilizada por Freitas. (Fonte: FREITAS, 2006a).

Devido à quebra no funcionamento dos mecanismos de comunicação; em um trabalho posterior (CASTILHO, 2007), utilizando *Javascript* substituiu o motor de simulação originalmente construído a partir de modelos em CPN, por uma lógica descrita em código Java, a qual era conectada ao AV através da EAI. Surgiu então a versão do simulador (Figura 2-4) com a arquitetura ilustrada na Figura 2-3. O objetivo desta versão foi manter o simulador funcional para realizar uma série de testes de usabilidade dentro de um prazo restrito para investigar outras soluções.

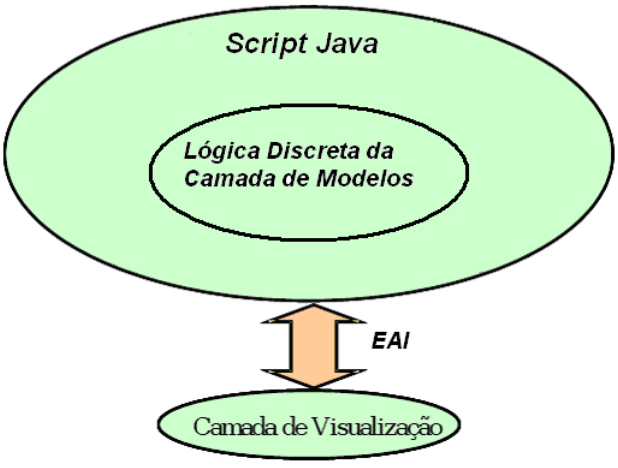

Figura 2-3 – Arquitetura utilizada por CASTILHO (Fonte: CASTILHO, 2007)

Esta solução comprometeu a facilidade de adaptar o simulador a diferentes cenários e instalações, pois o acréscimo de novos objetos implicava na recodificação do comportamento dos dispositivos e da lógica de inter-travamento.

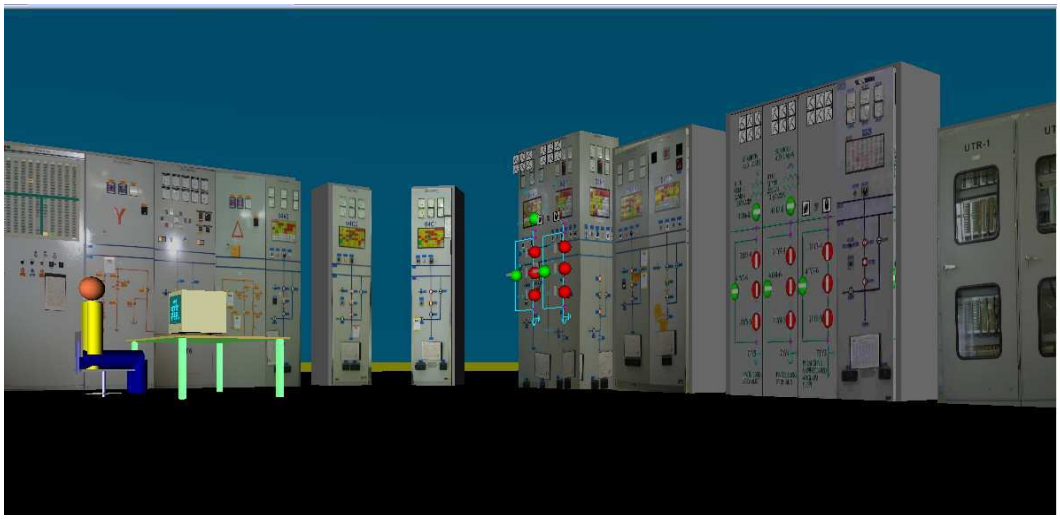

Figura 2-4 – Interface elaborada por CASTILHO. (Fonte: CASTILHO, 2007).

A partir da representação da Realidade Virtual do simulador, mais elaborada (com a representação mais completa dos objetos da subestação e de parte de seu ambiente físico), cujo propósito original era apoiar a pesquisa do erro humano no ambiente da subestação, surgiu o interesse da empresa CHESF em utilizá-lo como ferramenta de treinamento de seu pessoal, no setor de operação de subestações. A partir desta idéia, novos módulos foram concebidos e agregados ao simulador demandando uma reestruturação de sua antiga arquitetura (Figura 2-5) proposta em (VIEIRA et al, 2007), para a nova arquitetura que é o objetivo deste trabalho.

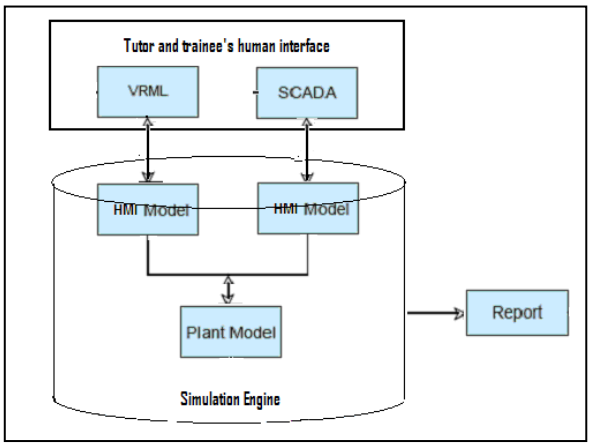

Figura 2-5 – Antiga proposta de arquitetura. (Fonte: VIEIRA et al, 2007).

Para atender o propósito de treinamento, os seguintes critérios ganharam importância:

- aumentar a fidelidade do simulador ao representar o ambiente de trabalho;
- assegurar o tempo de resposta mínimo adequado à imersão do operador neste ambiente;
- investigar ferramentas para a construção e visualização do ambiente;

- restaurar o motor de simulação em CPN e sua comunicação com a representação 3D dos objetos;
- associar um banco de dados contendo dados sobre instalações, treinamentos, tutores, participantes e cenários de treinamento.

Antes de propor a nova arquitetura foi realizada uma pesquisa na bibliografia por trabalhos com proposta semelhantes de simuladores baseados em Realidade Virtual, aplicados ao ensino ou treinamento (pois o simulador tem os dois propósitos) visando analisar suas arquiteturas.

2.3 Aplicações em Realidade Virtual – Arquiteturas

Esta sessão apresenta uma breve revisão de algumas aplicações, em Realidade Virtual encontradas na literatura, com propostas semelhantes à do simulador.

2.3.1 Ambiente de treinamento em Realidade Virtual (ESOPE – VR)

Consiste em um ambiente de treinamento de operadores em sistemas elétricos, desenvolvido pela Universidade de Montreal Quebec – Canadá. O ambiente representa uma subestação de um sistema hidroelétrico do Quebec e a interface desenvolvida possui dois níveis: painéis elétricos (sala de controle) e software supervisório (Fonte: VEH et al, 1996).

Sua arquitetura foi desenvolvida baseada no sistema de simulação e treinamento ESOPE (*Expert System Operations Planning Environment*), onde os operadores interagem através do supervisório (diagrama unifilar). No caso, a simulação do funcionamento dos objetos e do comportamento da planta são descritos neste sistema. Na Figura 2-6 é ilustrada a arquitetura utilizada no desenvolvimento.

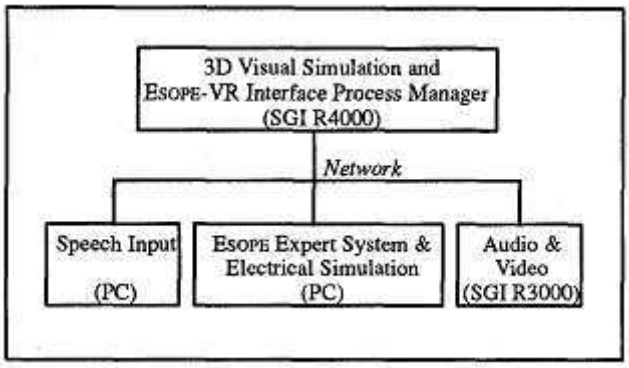

Figura 2-6 – Arquitetura do Sistema ESOPE-VR. (Fonte: VEH et al, 1996)

O Ambiente Virtual foi desenvolvido em AutoCAD e integrado com o ESOPE, tornando-se assim o ESOPE-VR e a comunicação entre os módulos é através de Ethernet.

Uma característica interessante desta arquitetura é um módulo para comunicação (através de microfone) entre o operador em treinamento e o tutor. Porém o sistema não possui o módulo tutor, neste caso o tutor não pode intervir nos cenários ou armazenar as ações executadas, o único modo de intervenção é através do diálogo (fala). Outra limitação é o fato do modelo da planta ser um software proprietário, não permitindo alterações no seu funcionamento.

2.3.2 Ambiente Virtual Colaborativo - Museu Virtual (AVC-MV)

Trata-se de um Ambiente Virtual Colaborativo (AVC) que representa um museu virtual ao qual um ou mais alunos podem ter acesso remoto simultâneo, a partir de diferentes locais, compartilhando os mesmos ambientes virtuais (AV). Esse projeto foi desenvolvido na Universidade Federal de Santa Catarina, para fins educacionais. Neste caso os usuários são crianças e adolescentes (estudantes) e o acesso é através da Internet (KAWAMOTO et al, 2001).

O objetivo do projeto é permitir aos usuários navegar e interagir com um AV que represente lugares reais, que até o momento eram desconhecidos aos seus usuários para que os mesmos possam ter a oportunidade de conhecer esses ambientes sem sair de casa (por exemplo, um museu em outro país). Esse ambiente é disponível através de um navegador de Internet. Na implementação foram utilizadas as linguagens de programação Java e a linguagem VRML 2.0, acopladas por EAI (*External Authoring Interface*).

A arquitetura é do tipo cliente-servidor e é apoiada por um protocolo de comunicação no qual todas as mensagens enviadas possuem um cabeçalho de identificação. A comunicação é bidirecional entre uma *applet* Java e o *plug-in* VRML, e toda a dinâmica da interface é desenvolvida em Java. Na Figura 2-7 é apresentado o funcionamento da comunicação para a aplicação proposta.

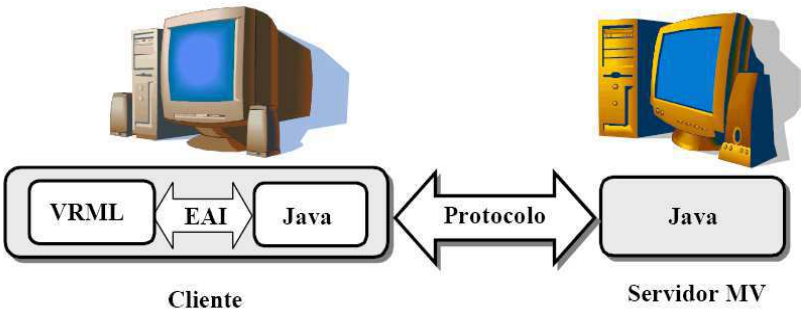

Figura 2-7 – Visão geral do sistema AVC-MV. (Fonte: KAWAMOTO et al, 2001).

Esta arquitetura é interessante por apresentar o padrão mais comum de comunicação entre cliente e servidor, onde o servidor funciona apenas como um repositório de arquivos, o

cliente acessa o AV que está armazenado no servidor diretamente do navegador de Internet. Uma desvantagem nesta arquitetura é que o cliente não pode desenvolver novos ambientes virtuais, podendo acessar apenas as aplicações já disponíveis e, a interação com o AV não é dinâmica (as funcionalidades são vídeos pré-definidos, que são ativados com o clique do mouse).

2.3.3 Realidade Virtual não-imersiva – Ensino da Física

Este trabalho aborda o uso de ferramentas cognitivas e de Realidade Virtual (não-imersiva) no ensino de física. Foi desenvolvido na Universidade Federal de Uberlândia e teve como objetivo disponibilizar um ambiente para o estudo da física. O público alvo são alunos e professores do ensino médio (SILVA et al, 2005).

O ambiente representa experimentos em Realidade Virtual (3D), com os quais os alunos podem interagir. Para possibilitar a escolha do experimento foi desenvolvido um organizador gráfico conceitual (em formato de árvore) e disponibilizado um sistema tutorial baseado em mapas conceituais. Na Figura 2-8 é ilustrada a arquitetura deste ambiente.

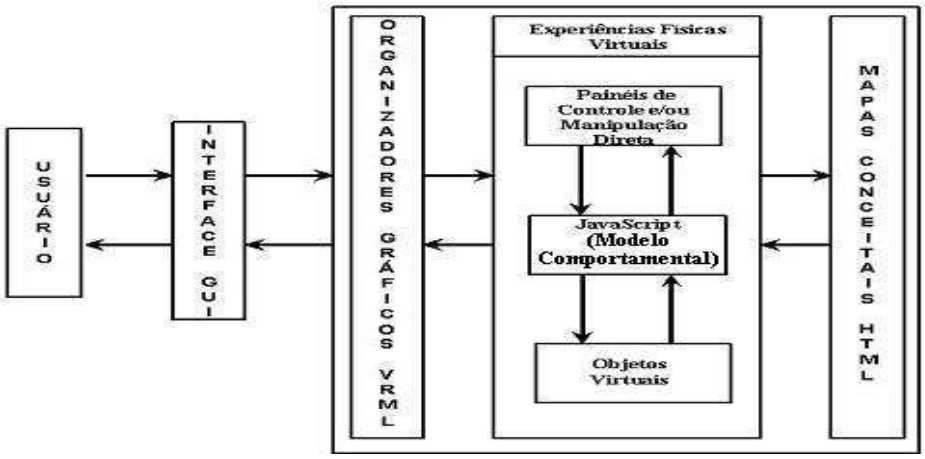

Figura 2-8 – Arquitetura do sistema (Fonte: SILVA et al, 2005).

O acesso aos experimentos é via Internet (Web) e para implementação foi utilizada a linguagem VRML para as imagens em 3D e *Javascript* para modelar o funcionamento dos objetos. A visualização 3D é através do *plug-in* Cosmo Player e o mapa conceitual foi elaborado utilizando a linguagem *Hypertext Transfer Protocol* (HTML). Esse projeto foi dividido em três etapas: um mapa conceitual (tutorial), organizadores gráficos (árvore da grade curricular da física) para o aluno ter uma visualização do todo e os experimentos em 3D.

Na arquitetura apresentada não é detalhada a comunicação com o banco de dados, além de não possibilitar a inserção e/ou alteração dos experimentos, mas apenas utilizar os

que já foram propostos. O que pode se enfatizar nesta arquitetura é a divisão das funcionalidades no servidor, possibilitando ao cliente ter mais de um modo de visualização da aplicação. Outra funcionalidade a ser destacada é o uso de mais de uma linguagem no desenvolvimento (VRML, *Javascript* e HTML).

2.3.4 Ambientes multiusuários utilizando X3D – EVE X3D

Trata-se de um ambiente multiusuário sem aplicação específica. Foi desenvolvido na universidade de Patras, na Grécia. Sua arquitetura, denominada EVE (X3D *enabled networked virtual environment*), suporta um ambiente multiusuário desenvolvido em X3D (BOURAS et al, 2005). A comunicação com o ambiente é via Internet (Web) e do tipo cliente/múltiplo-servidores (Ver Figura 2-9).

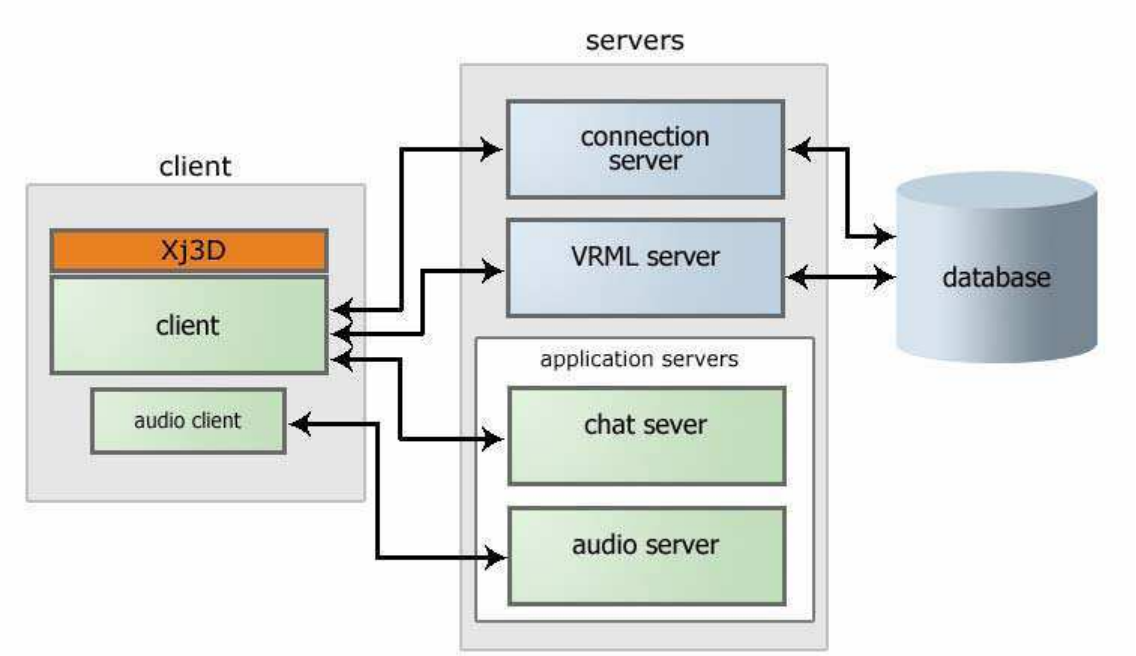

Figura 2-9 – Arquitetura multi-servidor. (Fonte: BOURAS et al, 2005).

Esta aplicação está estruturada em quatro servidores, além do banco de dados. O servidor principal controla o fluxo de informações e direciona o pedido para o servidor requisitado. O servidor VRML controla as telas 3D que são apresentadas no cliente e os outros dois servidores são responsáveis pelas mensagens de áudio e pelo ambiente de bate-papo (escrito). Além do banco de dados, onde são armazenados os dados de comunicação e os códigos dos ambientes virtuais.

O cliente consiste em um *applet* Java com o visualizador Xj3D embutido, uma vez que o *applet* não é capaz de *renderizar* imagens 3D. A comunicação do *applet* com o visualizador é feita através de EAI.

O interessante desta arquitetura é o fato de utilizar o visualizador Xj3D e ter múltiplos servidores, que não necessariamente estão alocados em uma única máquina (computador), porém o texto não apresenta detalhes da implementação dos ambientes virtuais, nem um estudo de caso.

2.3.5 Ferramenta para treinamento em montagens elétricas - FTME

Este trabalho consiste em uma ferramenta para o treinamento na montagem de cabos elétricos, na indústria automotiva. Foi desenvolvido pelo centro de computação gráfica da cidade de Coimbra, Portugal. A ferramenta consiste em um ambiente de montagem de cabos elétricos com imersão na RV através do uso de capacete e luvas (GRAVE et al, 2001). A ferramenta é voltada para dois tipos de usuários: o formador (responsável por elaborar o treinamento) e o formando (operadores que necessitam do treinamento). Tem como objetivo treinar os formandos na memorização da seqüência correta de montagem e na absorção de novas configurações de montagem, com a possibilidade do formador posteriormente acompanhar os passos executados pelos formandos.

Na Figura 2-10 é ilustrada a arquitetura da ferramenta. O aplicativo é estruturado em módulos: o módulo administrador local, situado no servidor central (que contêm as bases de dados: os modelos dos objetos usados; a configuração das montagens; e o estado atual). O servidor também é responsável por manter a consistência do sistema, acoplado a um servidor de páginas web responsável pela conexão via TCP/IP com os módulos: formador e formando.

O módulo formando permite visualizar, interagir e armazenar localmente informações (motor gráfico, controle de interação e tutorial). Esta base de dados local está sincronizada com a base de dados central. Através do módulo do formador pode-se administrar os dados dos formandos, controlar os treinamentos e analisar as estatísticas. A comunicação entre os módulos é via TCP/IP, através do modelo cliente/servidor.

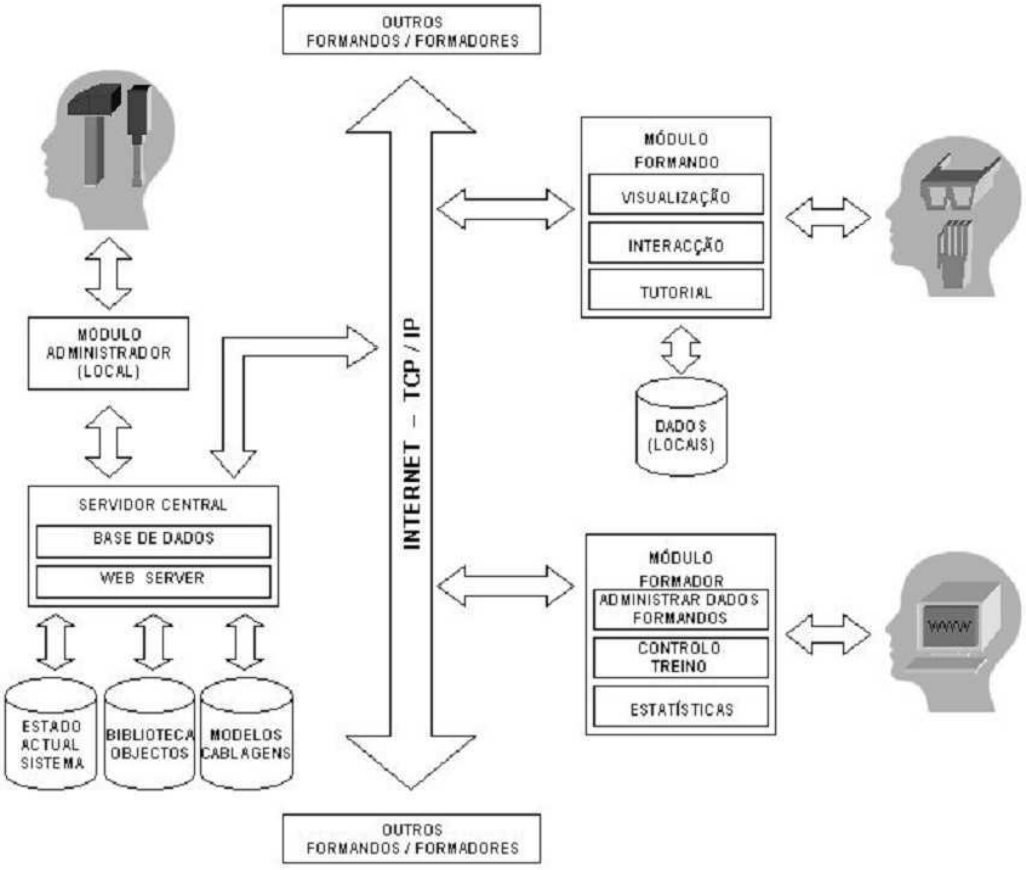

Figura 2-10 – Arquitetura da ferramenta de montagem de cabos elétricos. (Fonte: GRAVE et al, 2001).

Os recursos utilizados no desenvolvimento foram: HTML para as páginas Web, CGI (*Common Gateway Interface*) para ativar remotamente o módulo do formando via formador e uma linguagem *script* para representar o funcionamento dos objetos e acessar base de dados. O interessante desta arquitetura é que proposição apresentada é semelhante à desejada para o simulador. A desvantagem é que o formador não pode visualizar o treinamento em tempo real, seu motor de simulação é através de *scripts* dificultando sua manutenção.

2.3.6 VEPersonal

O VEPersonal (*Personalized Virtual Environment*) consiste em uma infra-estrutura criada para a geração e manutenção de ambientes virtuais adaptativos. Foi desenvolvido na Universidade Federal de Pernambuco, no Centro de Informática, e utilizado em um ambiente de ensino da física. Os usuários são alunos e professores em vários níveis (ensino fundamental, médio e graduação) (AQUINO, 2007).

Este trabalho teve como objetivo desenvolver um sistema para apoiar a criação e manutenção de AV Adaptativos, nos quais os objetos dinâmicos são acessados via Web para navegação e interação. A arquitetura é do tipo cliente/servidor com comunicação TCP/IP;

Na Figura 2-11 é ilustrada a arquitetura adotada, baseada no uso de agentes inteligentes que avaliam o conhecimento adquirido pelo usuário durante sua interação com o sistema e se adaptam de acordo com o modelo do usuário. O visualizador (a interface) é conectado a um arquivo Java onde foi implementado um conjunto de agentes com um sistema de sensores e atuadores. Neste caso possui um agente gerente que recebe as mensagens da interface e repassa aos agentes secundários e vice-versa. Os agentes secundários são: agente pessoal responsável por captar as informações do usuário e levantar o perfil do usuário, para saber em qual o usuário se encaixa; o agente atualizador responsável por atualizar os objetos da interface de acordo com o perfil do usuário; e o agente ambiente, responsável por carregar o ambiente principal comum a todos os usuários. O servidor além dos agentes é composto por um conjunto de banco de dados com todos os registros (modelo do usuário, ambientes virtuais e modelo do ambiente).

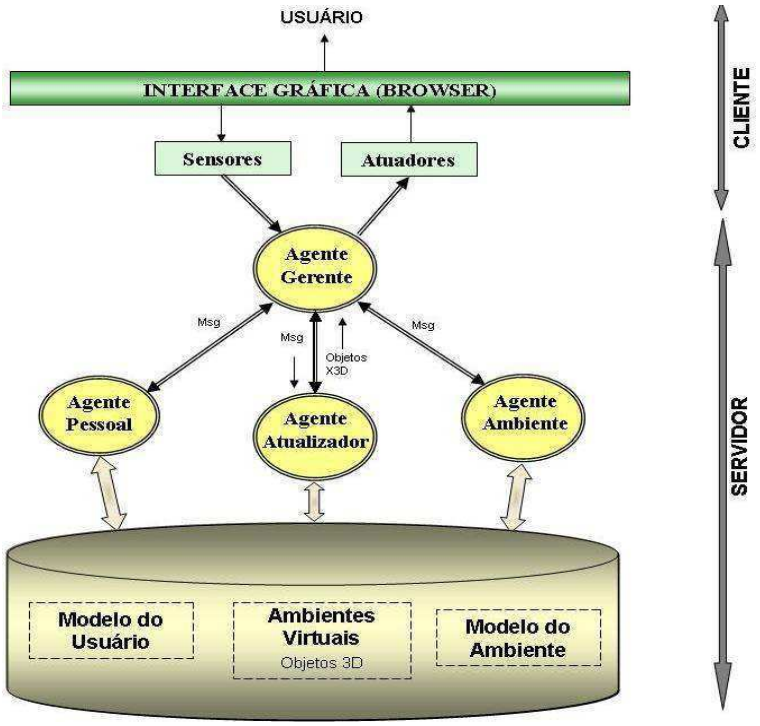

Figura 2-11 – Arquitetura do VEPersonal. (Fonte: AQUINO, 2007)

Na implementação foram usadas as linguagens X3D e Java, e o banco de dados oferece suporte a XML (eXtensible Markup Language). O banco de dados utilizado foi o PostgreSQL (POSTGRESQL, 2009), a ferramenta de desenvolvimento foi JADE (Java Agent DEvelopment framework) (JADE, 2009) que consiste em um framework para desenvolvimento de aplicações utilizando agentes, na plataforma Java.

O interessante desta arquitetura é a adaptabilidade da interface ao usuário, pois o simulador necessita desta característica. Além de apresentar uma estrutura com um agente

principal e subdividir as tarefas. Entretanto a arquitetura não leva em consideração o sistema multiusuário e seu motor de simulação é bastante simples (através de scripts Java), pois a aplicação não demanda uma grande interação com o Ambiente Virtual.

2.3.7 Simulador para treinamento de proteção e operação – STPO

O simulador para treinamento de proteção e operação em sistemas elétricos foi desenvolvido pela Universidade Federal do Ceará e tem como objetivo permitir a capacitação e treinamento de pessoal do setor elétrico através do ensino presencial e à distância, via Internet (BEZERRA et al, 2007). A aplicação foi desenvolvida em DELPHI.

A interface do simulador é composta por um diagrama unifiliar sistêmico contendo os componentes de proteção, onde se pode simular faltas e reconfigurar o sistema. Além disso, o simulador oferece um módulo EmulaScada que consiste em um sistema para simular os alarmes reais que ocorrem em um sistema SCADA (*Supervisory Control And Data Aquisition*), de uma concessionária de energia elétrica.

O STPO foi integrado ao Teleduc que é um ambiente para criação, participação e administração de cursos Web. Onde o usuário pode se cadastrar e ter acesso ao STPO. O sistema não oferece uma representação 3D (Realidade Virtual) e é restrito à simulação do ambiente supervisório.

2.3.8 Ambiente simulado para treinamento de operadores – ASTRO

Desenvolvido pelo CEPEL (Centro de Pesquisas de Energia Elétrica) em parceria com a ELETROSUL, consiste em um ambiente simulado para treinamento de operadores e tem como objetivo simular a ocorrência de falhas no sistema, para que os operadores possam ser treinados em diversos tipos de situações antes que elas efetivamente ocorram. Foi desenvolvido para ser um simulador de operação do sistema e não o comportamento do sistema elétrico (SILVA et at, 2009).

O programa é integrado ao sistema aberto de supervisão e controle (SAGE), operando *off-line*. O ASTRO é composto por três módulos:

- editor de cenários: permite que os supervisores (tutores) criem através de uma interface gráfica, novas simulações para treinamento;
- programa de simulação da operação: funciona de forma integrada e transparente junto ao SAGE;
- analisador de desempenho: permite que o instrutor do treinamento, analise e avalie uma simulação.

Assim como o STPO foi desenvolvido apenas para representar o ambiente do supervisório. Ao contrário deste projeto, não oferece o Ambiente Virtual 3D para simulação da operação nos painéis de controle da subestação.

2.3.9 Casa da eficiência energética - EEHouse

A casa da eficiência energética EEHouse consiste em um jogo educacional para ensino dos conceitos de eficiência energética. Foi desenvolvido na Universidade Federal do Paraná (MARTINS et al, 2009) e é voltada para estudantes do nível médio, oriundos de escolas públicas. O jogo avalia inicialmente o hardware disponível para oferecer o modo de execução mais adequado. Há dois modos de execução: no primeiro, as imagens 3D são *renderizadas* no cliente enquanto no segundo, as imagens são *renderizadas* no servidor (quando o cliente não possui o hardware mínimo) (Figura 2-12).

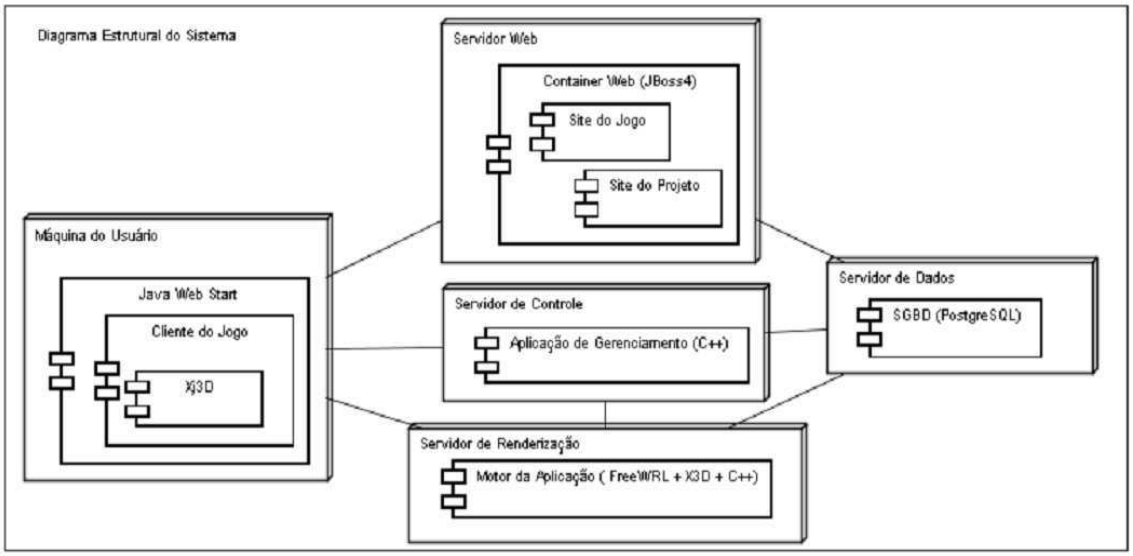

Figura 2-12 – Arquitetura do EEHouse. (Fonte: MARTINS et al, 2009)

A arquitetura é do tipo Cliente/Servidor e a comunicação é via TCP/IP. O ambiente possui dois modelos de interface 2D e 3D. Na Figura 2-12 é ilustrada a arquitetura que é composta por: servidor de dados (SGBD: sistema gerenciador de banco de dados utilizando a tecnologia PostgreSQL); servidor de gerenciamento (uma aplicação desenvolvida para manter a lógica do controle da aplicação, onde serão analisadas as requisições enviadas pelo cliente); servidor web (onde a aplicação será divulgada utilizando JBoss4); servidor de *renderização* (servidor responsável por *renderizar* cenas tridimensionais quando o hardware do usuário não suportar esse processo); e cliente aplicação (o canal de acesso do usuário é desenvolvido em JavaFX e X3D). O visualizador de imagens 3D é o Freewrl (para *renderização* no servidor) e Xj3D (para *renderização* no cliente); ambos integrados com Java.

O interessante desta arquitetura é o fato de existirem dois modos de visualização, um servidor de controle e uma descrição mais detalhada das tecnologias utilizadas na implementação. Porém não é considerada a possibilidade de dois usuários interagirem no mesmo Ambiente Virtual. Cada cliente é independente e, o professor não tem a oportunidade de acompanhar o aprendizado do alunato através do uso do sistema.

2.4 Requisitos do Simulador/Comparativo com as aplicações estudadas

A nova arquitetura do simulador deverá contemplar soluções aos seguintes aspectos: extensão das funcionalidades; variedade de contextos de uso; necessidade de comunicação com processos externos (modelos, outros programas); fluxo e armazenamento dos dados; tempos de resposta estritos; recursos abertos/grátis para executar o simulador (hardware e software). Destes aspectos originaram-se os requisitos da arquitetura do simulador:

- Permitir o acesso remoto (através da Internet): os usuários devem poder acessar o simulador via Web. Isto é importante para oferecer treinamento em uma empresa geograficamente distribuída, na qual o tutor pode estar localizado remotamente em relação ao operador que está sendo treinado, reduzindo os custos para a empresa;
- Ambiente colaborativo – multiusuário: possibilitar que mais de um operador interaja no mesmo ambiente emulando seu trabalho em duplas e permitindo o treinamento simultâneo no mesmo Ambiente Virtual, além de permitir ao tutor acompanhar a execução das manobras;
- Banco de dados: armazenar dados gerenciais dos treinamentos (participantes, instalações, cenários, resultados), as representações virtuais das instalações e seus equipamentos e os modelos que compõem o motor de simulação;
- Restaurar o motor de simulação à base de modelos CPN: restaurando a comunicação entre o mundo virtual e o motor de simulação (modelos CPN);
- Configurar cenários de treinamento: facilitar a configuração dos cenários de treinamento, a partir do reuso de código e de modelos;
- Oferecer múltiplas interfaces: analisar a necessidade de mais de um módulo no cliente (mais de uma interface), já que existe mais de um perfil de usuário. Assegurar usabilidade nos múltiplos níveis de interação, para todos os usuários: tutores, operadores e desenvolvedores responsáveis pela adequação a novos contextos de treinamento;
- Aumentar a fidelidade na representação gráfica: a ferramenta de representação gráfica deve ser robusta o suficiente para modelar o ambiente real simulado;
- Multi-plataforma: compatível com Windows e Linux.

- Interoperabilidade: levar em consideração a interoperabilidade do sistema (propondo uso, por exemplo, de XML);
- Projeto modular: para facilitar a manutenção, ampliação do sistema; e permitir adaptar para outros contextos.

Após a análise das arquiteturas estudadas, foi montada a Tabela 2-1 na qual é apresentado um comparativo entres as arquiteturas, com base nos requisitos do simulador.

Tabela 2-1 – Comparativo entre as arquiteturas estudadas.

	Acessível via Internet	Comunicação com processos externos	Modular	Banco de dados	Permite editar ambiente	Permite inserir novos objetos/ambientes	Utiliza Realidade Virtual	Suporta Múltiplos usuários	Histórico das interações	Multi-plataforma	Possui modelo formal como motor de simulação
ESOPE-VR	■	■	■				■				
AVC-MV	■				■		■	■			
Ens. Física	■	■		■			■		■		
EVE X3D	■	■		■			■	■		■	
FTME	■	■	■	■	■	■	■	■	■		
VEPersonal	■	■	■	■	■		■			■	
STPO	■	■			■				■		
ASTRO	■				■				■		
EEHouse	■	■		■			■	■	■	■	
Proposta do Simulador	■	■	■	■	■	■	■	■	■	■	■

Este capítulo abordou os estudos realizados para embasar este trabalho, fundamentando o conceito de Realidade Virtual, enfatizando que no trabalho foi desenvolvido um ambiente virtual não-imersivo, apresentou os critérios para a arquitetura do simulador, o seu objetivo, as versões anteriores e seus problemas. Outro aspecto abordado foi a apresentação de aplicações em RV com características semelhantes as do simulador que fundamentaram a proposta da nova arquitetura do simulador, que será apresentada no próximo capítulo.

Capítulo 3 Arquitetura proposta

A partir dos requisitos levantados para o projeto do simulador, neste capítulo será abordado à aplicação de um método para concepção de AV, aplicado ao ambiente do simulador. Com base neste método foi levantado o perfil do usuário, elaborada a Interface do AV em 3D, elaborado o motor de simulação e a interconexão entre os módulos da arquitetura.

3.1 Requisitos para o simulador

Retomando os requisitos abordados no capítulo anterior, a arquitetura do simulador deve:

1) permitir o acesso remoto ao simulador (através da Internet);
2) oferecer um ambiente colaborativo (multiusuário);
3) integrar um banco de dados;
4) possuir um motor de simulação baseado em modelos (CPN);
5) permitir a fácil configuração de cenários de treinamento;
6) oferecer múltiplas interfaces;
7) oferecer alta fidelidade na representação gráfica;
8) executar em múltiplas plataformas;
9) oferecer interoperabilidade;
10) ser modular.

A partir destes requisitos foi proposta a arquitetura para a nova versão do simulador ilustrada na Figura 3-1. A arquitetura é do tipo cliente-servidor e a comunicação proposta entre os seus módulos é através de *socket* utilizando protocolo TCP/IP (*Transmission Control Protocol/ Internet Protocol*). Para fazer a conversão das mensagens entre o cliente e o servidor será utilizado um *driver* que, converte as mensagens entre as CPN e o AV e gerencia as mensagens trocadas entre eles.

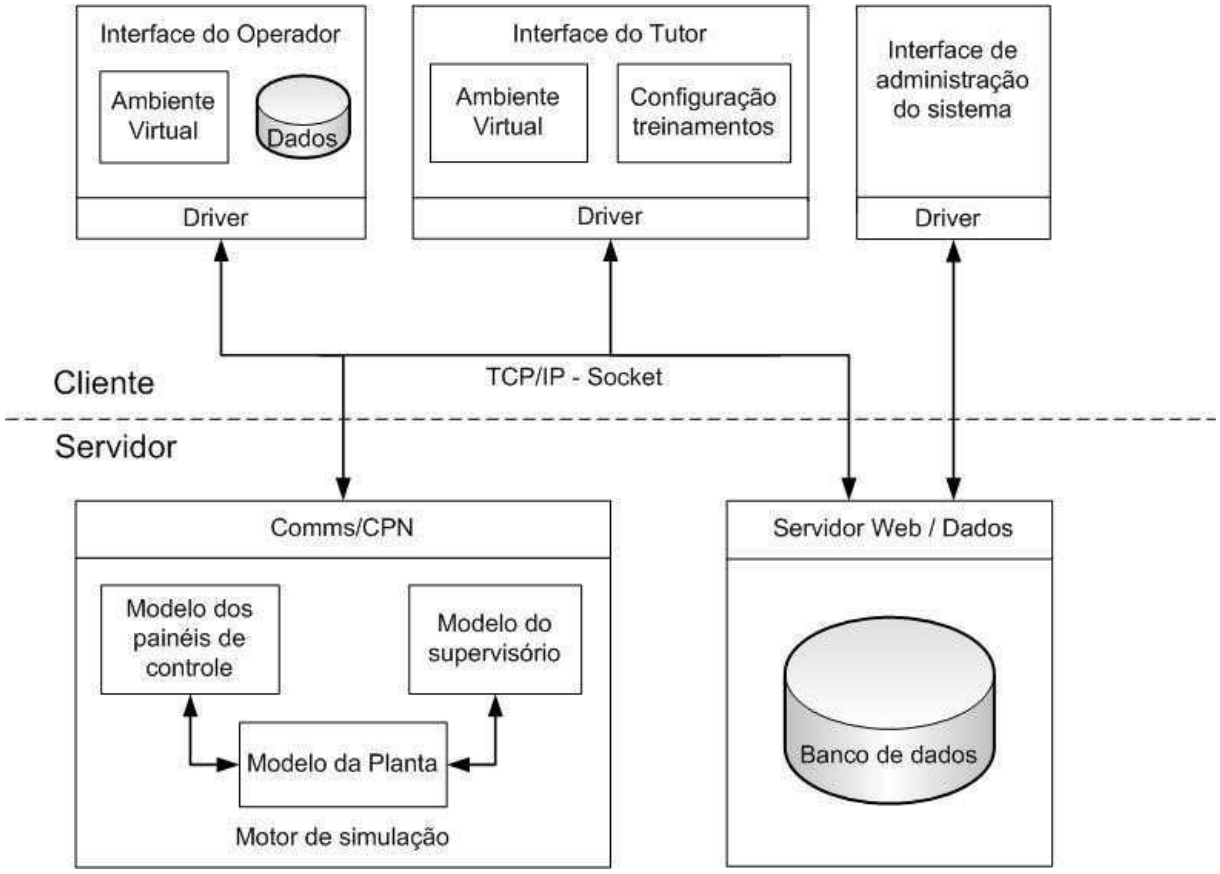

Figura 3-1 – Arquitetura proposta.

Para apoiar o desenvolvimento da arquitetura proposta na Figura 3-1, foi elaborada uma documentação em *Unified Modeling Language* (UML) (BOOCH et al, 2000) que consiste em digramas de classes e outros que detalham os módulos apresentados na arquitetura.

O diagrama de pacotes, ilustrado na Figura 3-2, apresenta os blocos que compõem a arquitetura: ambiente tutor (browser - treinamento); ambiente operador (browser); ambiente do administrador (configuração); servidor (servidor Web e BD) e o Motor de simulação (modelos CPN).

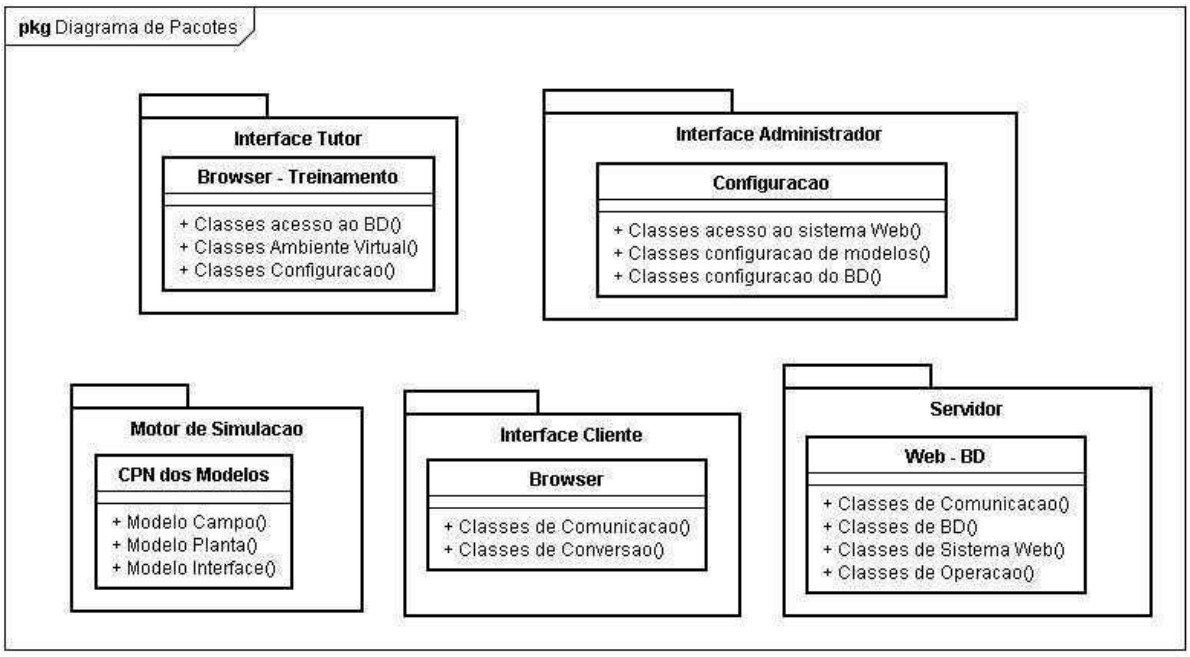

Figura 3-2 – Diagrama de Pacotes

3.2 Cliente

O cliente é composto por três módulos: a interface de administração e configuração do sistema como um todo; a interface do Tutor – responsável por gerir e acompanhar os treinamentos; e a interface do Operador – ambiente no qual o usuário será treinado.

A **interface do operador** consiste no AV, que simula seu ambiente real de trabalho, no qual ele pode visualizar o AV, navegar e interagir com os objetos utilizando o mouse e o teclado. Além disso, é composto por um conjunto de dados locais onde são armazenadas as informações advindas do servidor. Dentre elas o AV que é extraído do servidor e executado diretamente no cliente para diminuir o tráfego de informações durante o treinamento.

A **interface do tutor** é composta pela interface do operador (AV), onde ele pode monitorar as ações do operador e interagir no ambiente, e a interface de configuração de treinamento (que não é necessariamente em 3D), onde conterá funcionalidades como: administrar dados de operadores; configurar treinamentos; consultar treinamentos anteriores; anotar informações durante o treinamento que serão salvos em conjunto com o histórico (*log)* de ações executadas pelo operador, entre outras funcionalidades.

As interfaces do tutor e do operador são acessadas através do mesmo sistema, então o que irá restringir o acesso às interfaces é a página inicial onde o usuário necessita se identificar através de *login* (nome do usuário) e senha de acesso. Neste caso, os usuários deverão ser previamente cadastrados no banco de dados, com os devidos privilégios.

A outra interface, a de administração do sistema, servirá para adicionar novos objetos nas subestações, alterar e adicionar as Redes de Petri Coloridas ao banco de dados para serem utilizadas como motor de simulação, configurar o servidor, além de cadastrar novos usuários (tutores e operadores).

Algumas funcionalidades do sistema são apresentadas na Figura 3-3, através do diagrama de caso de uso. Nele pode-se observar três tipos de usuários que utilizaram o sistema, o operador e tutor que utilizam o ambiente de treinamento e o administrador que é responsável pela manutenção do sistema desenvolvido.

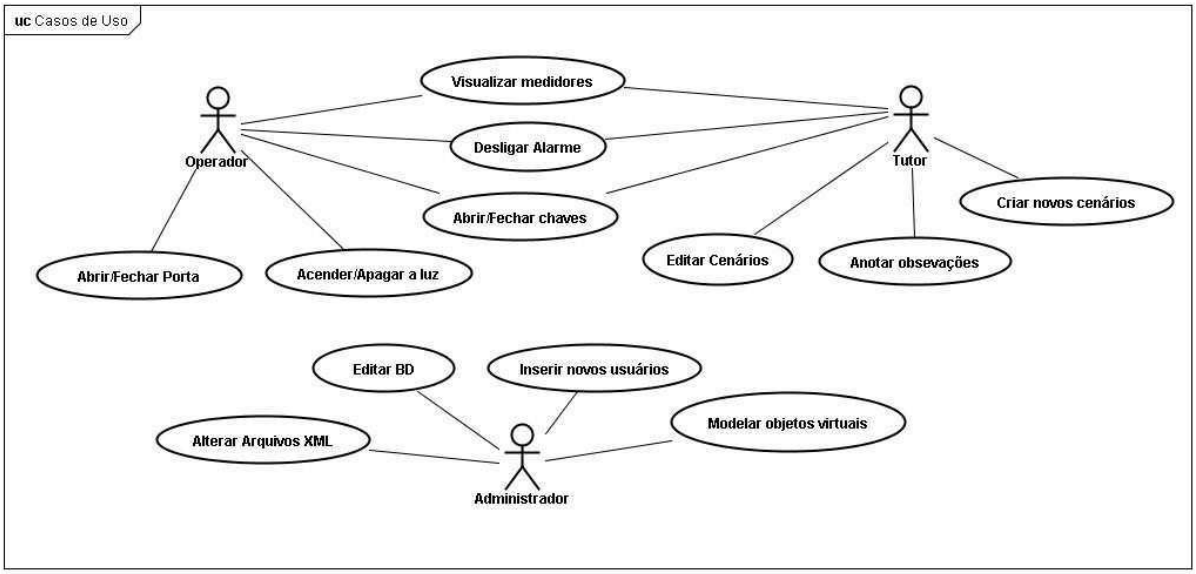

Figura 3-3 – Caso de uso da arquitetura.

3.3 Servidor

O servidor é composto por dois módulos: um módulo é o motor de simulação em CPN e o outro é o módulo de serviço Web e de dados. Os módulos do servidor podem estar presentes em um, ou mais computadores. No caso do uso de dois computadores, um poderá executar o motor de simulação e outro conterá o módulo de serviços Web. O módulo de serviços Web consiste no servidor Web, no servidor de banco de dados e no banco de dados.

O diferencial desta arquitetura é adotar como motor de simulação um modelo formal construído em Redes de Petri Coloridas. Tipicamente tal motor seria desenvolvido em uma linguagem de programação, como Java e C++, porém utilizando linguagens não teria sido possível verificar o comportamento dos objetos modelados.

Além disso, o uso das CPN hierárquicas incentiva a modularidade do projeto facilitando a manutenção do motor de simulação a partir do reuso de modelos e o acréscimo de novos objetos à rede, se comparado à implementação em linguagens que demandam a

escrita de novos códigos para cada novo objeto acrescentado, além da dificuldade de verificar/validar o projeto do motor de simulação.

A escolha das CPN também se deu devido à existência de uma biblioteca de modelos que representam o comportamento dos objetos presentes nos painéis de uma subestação e pelo fato que através das Redes de Petri Coloridas é possível representar sistemas com característica em tempo real, onde o tempo de resposta é finito.

3.4 Banco de dados

O Banco de Dados (BD) conterá os dados dos treinamentos, os cenários de treinamento que consistem na associação de objetos e respectivos estados para representar o cenário de treinamento, além de armazenar a representação dos dispositivos tanto em Redes de Petri Coloridas quanto a respectiva representação gráfica. O BD registrará ainda, o histórico de todas as interações do operador com o Ambiente Virtual durante o treinamento. Isto facilitará uma análise da falhas cometidas pelos operadores durante o treinamento e posteriormente o tutor poderá utilizar esta análise para propor novos cenários de treinamentos. A outra vantagem consiste na possibilidade de salvar o estado atual ao final de uma sessão para que este possa ser recuperado na próxima interação com um mesmo usuário, dando flexibilidade ao treinamento.

O Sistema de Gerenciamento de Banco de Dados (SGBD) será responsável pelo acesso dos múltiplos usuários ao banco de dados. No caso, a comunicação feita entre o cliente e o BD é via *socket* através de uma API – *Java DataBase Connectivity* (JDBC) (JDBC, 2010), que implementa em Java as funcionalidades definidas em pelo padrão SQL (*Structured Query Language*). Esta comunicação deverá ser integrada ao *driver* desenvolvido neste trabalho, então quando as mensagens forem enviadas do AV para o motor de simulação, também devem ser enviadas para o BD fazendo assim o registro de todas as ações do operador (histórico – *log*).

O SGBD escolhido para a versão atual do simulador foi MySQL (MYSQL, 2010) versão *community* (grátis), pois é:

- compatível com sistemas operacionais Windows e Linux;
- um software livre;
- muito utilizado em aplicações Web e bastante popular, possuindo vários tutorias de uso na internet;
- compatível com linguagens como PHP, Java, Python, C#, Ruby e C/C++;

- baixa exigência de processamento (em comparação como outros SGBD);
- oferece vários sistemas de armazenamento de dados (*database engine*), como MyISAM, MySQL Cluster, CSV, Merge, InnoDB, entre outros;
- reconhece instruções em SQL, como indica o nome;

3.5 Sistema Web

Um servidor Web coordena a comunicação (atendendo as requisições dos clientes) e disponibiliza a aplicação na Web, o qual os usuários podem se conectar. Para disponibilizar este sistema é interessante dedicar uma máquina para ser exclusivamente um servidor e instalar um sistema operacional do tipo servidor, pois facilita a configuração do servidor Web, do banco de dados e do sistema de segurança.

No sistema desenvolvido foi utilizado o sistema operacional Windows Server 2008 e o Servidor Web instalado foi o fornecido pela própria Microsoft IIS (*Internet Information Service*) (IIS, 2010). A escolha foi feita devido o ambiente servidor já está implantado no LIHM. Entretanto se for preferível utilizar um ambiente grátis/livre pode ser instalado o sistema operacional Linux Server, por exemplo, UBUNTU *Server* (UBUNTU, 2009) o qual já contém as características de ambiente Web e aplicações de segurança. Ou ainda utilizar o Linux cliente (por exemplo, UBUNTU), com o Tomcat (TOMCAT, 2010) como servidor Web, pois ele foi desenvolvido para aplicações Java Servlet e JavaServerPages e executa tanto em Windows quanto em Linux.

3.6 Módulo de treinamento do Operador

Definida a arquitetura passa-se à fase da implementação dos módulos, um trabalho que foi distribuído entre os participantes do grupo de alunos do LIHM. Neste trabalho o autor ficou responsável essencialmente pela reformulação do Ambiente Virtual e das Redes de Petri Coloridas, além da integração entre módulos da arquitetura a partir do desenvolvimento do *driver* de comunicação entre o Ambiente Virtual e o motor de simulação e pela escolha das ferramentas a serem utilizadas na construção e execução do simulador.

Inicialmente foi realizado um estudo das tecnologias de Realidade Virtual e proposto um método para concepção destes ambientes, o qual se encontra ilustrado na Figura 3-4 (NETTO et al, 2009).

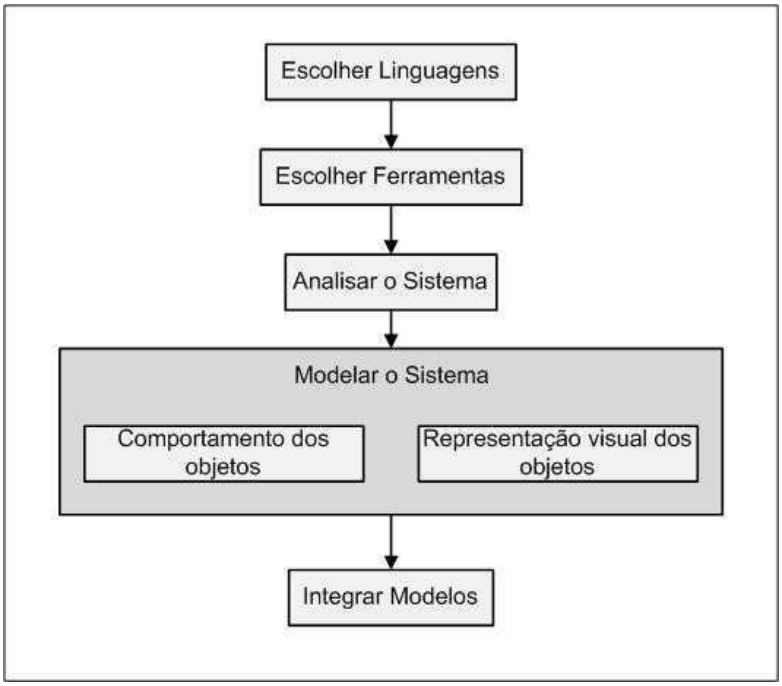

Figura 3-4 – Etapas de implementação do Ambiente Virtual. (Fonte: NETTO et al, 2009)

O procedimento utilizado para a concepção da Interface (Ambiente Virtual) consistiu em:

- escolher o formalismo para representar o comportamento dos objetos;
- escolher da linguagem de representação do AV neste caso é apresentado no Anexo A.1;
- escolher as ferramentas de visualização e edição baseada na linguagem escolhida (vide Anexo A.2);
- analisar o perfil do usuário que utilizará o simulador;
- estudar o sistema que se deseja representar (neste caso a sala de comando de uma subestação)
- modelar os sistema tanto do ponto de vista do comportamento dos objetos (motor de simulação) quanto da representação visual dos objetos (AV) e por fim;
- integrar o sistema.

Os próximos tópicos descrevem a análise do ambiente a ser representado, o motor de simulação e os AV originados de um projeto anterior.

3.6.1 Perfil do usuário

Os usuários do simulador serão os operadores, tutores e desenvolvedores. Os operadores serão treinados na realização de manobras raras e complexas. O treinamento

previsto tanto pode ter um caráter introdutório para os noviços, quanto de reciclagem para os mais experientes. Os tutores são responsáveis por elaborar os treinamentos (editar o ambiente simulado a partir de uma biblioteca de objetos disponível no banco de dados da ferramenta) e por analisar os resultados, os quais serão armazenados em um banco de dados. O desenvolvedor é responsável por atualizar a biblioteca de objetos (modelos) tanto do ponto de vista comportamental quanto de representação visual, editando os modelos existentes e inserindo ou retirando modelos; além de ser responsável pela manutenção do sistema.

O perfil do usuário para o contexto é:

- Gênero: Predominantemente masculino;
- Faixa-etária: 25-60 anos;
- Formação mínima: Curso técnico em eletrotécnica;
- Conhecimento da tarefa de um operador: Médio - Alto (da área de sistemas elétricos);
- Conhecimento da ferramenta: Baixo/Nenhum;
- Conhecimento do uso de computadores: Médio;
- Nível de experiência na função: Médio – Alto;
- Nível de condições físicas, psicológicas e mentais: Bom;

3.6.2 O ambiente representado – sala de comando de uma subestação

Para modelar o ambiente a representar foi realizado um estudo sobre o ambiente, nas subestações. Deste estudo deduziu-se que existem tipicamente dois ambientes: a sala de supervisão e controle, e o campo (planta) onde ficam situados os equipamentos (transformadores, seccionadoras, motores, disjuntores). Na sala de comando (supervisão e controle) os operadores podem interagir com o sistema através de um software supervisório (via SCADA) e/ou através de painéis, ambos ilustrados na Figura 3-5. O painel de supervisão e controle é formado por indicadores luminosos (alarmes), medidores (de corrente, potência e tensão) e botoeiras, as quais os operadores acionam para executar uma ação no campo.

Figura 3-5 – Painel de comando e tela do supervisório (Fonte: Subestação CHESF, Campina grande)

O supervisório é um software para controle e supervisão remota de equipamentos. Sua função é análoga a dos painéis de controle e representa o diagrama sinótico do sistema elétrico, e quadros compostos de alarmes (visual e sonoro), botoeiras/chaves, quadro de eventos e sinalizações que indicam o estado dos equipamentos no campo. Na Figura 3-6 é ilustrado um painel identificando alguns dos seus conteúdos.

Figura 3-6 – Identificação dos objetos no painel

Durante a execução de uma tarefa, os operadores trabalham em duplas, um deles situado na sala de comando executando as manobras e o outro próximo aos equipamentos verificando e validando as ações. As manobras podem ser executadas tanto no campo quanto na sala de controle, porém alguns equipamentos só podem ser operados no campo. Em situação de falta (falha no equipamento, curto circuito) o estado dos equipamentos pode discordar daquele exibido na sala de comando (na tela do supervisório ou no painel). Por

exemplo, um disjuntor recebe um comando da sala de controle para ser aberto, mas o mecanismo de acionamento de abertura não funciona, e o disjuntor continua fechado.

Considerando o comportamento dos equipamentos durante as manobras executadas na subestação, no projeto original do simulador foram concebidas duas redes distintas, porém interligadas, para representar o estado dos equipamentos no campo e na sala de controle; constituindo **o motor de simulação**. Também foram concebidas redes distintas para representar os painéis de controle e o supervisório. Uma vez que os modelos CPN, que compõem o motor de simulação, representam apenas o comportamento dos objetos no simulador; o comportamento de cada objeto deve ser associado a um *script* que representará visualmente este objeto no mundo virtual (tela do simulador).

3.6.3 Ambiente Virtual desenvolvido

Antes de escolher quais tecnologias seriam utilizadas no desenvolvimento da nova versão do simulador, foi feito um estudo das linguagens de descrição de AV. Na Tabela 3-1 é apresentado um comparativo entre os requisitos (critérios de escolha) e as linguagens estudadas, onde o símbolo ✓ indica que a linguagem atende o requisito, vazio (em branco) indica que não atende e, o símbolo * indica que esta característica não foi pesquisada em detalhes uma vez que a linguagem caiu no desuso pela comunidade. Maiores detalhes das linguagens e os requisitos levantados para escolha da linguagem estão presentes no Anexo A.1.

Tabela 3-1 – Requisitos x Linguagens

	VRML	X3D	Java3D	3DML	3DXML	xVRML	3DMLW
Versão/Ano	2.0/1997	3.0/2009	1.4/2007	2005	2006	2003	2009
Aceitação	✓	✓	✓				
Capacidade de representação	✓	✓	✓	*	*	*	*
Documentação/tutoriais	✓	✓	✓				✓
Ferramentas de edição	✓	✓	✓	*	*	*	✓
Ferramentas de visualização	✓	✓	✓	*	*	*	✓
Não proprietária	✓	✓	✓	✓		✓	✓
Formato em XML		✓			✓	✓	
Autor/Grupo	Web3D	Web3D	Sun	Flatland	Dassault	CMEG	R&D

Legenda: ✓ Atende o critério. | | Não atende o critério. | * Não foi pesquisado.

Depois de analisar o quadro e demais fatores, a linguagem escolhida pra representar RV foi o **X3D**, pois preenche todos os requisitos. O formato XML foi decisivo devido às facilidades agregadas com seu uso, como portabilidade, facilidade de armazenamento em banco de dados e diminuição na taxa de transferência dos dados (pois XML são pequenos arquivos).

O **X3D** (*eXtensible* 3D): é um padrão aberto de formato de arquivo de execução, desenvolvido pelo Web3D Consortium para representar cenas e objetos 3D. É reconhecido pela norma ISO/IEC 19776 que provê um sistema para o armazenamento, recuperação e reprodução de conteúdo 3D, embutido em aplicações. Além de ser um padrão baseado em XML (*eXtensible Markup Language*) é considerado como a evolução do VRML (WEB3D, 2009).

Dado que é uma linguagem baseada em XML, o X3D oferece vantagens como, uma melhor interoperabilidade entre aplicações (um dos principais objetivos para a criação do X3D). O projeto anterior foi desenvolvido em VRML e neste trabalho foi realizada a migração para a linguagem X3D através da conversão do ambiente já existente com alguns ajustes.

A partir da escolha da linguagem, foram escolhidas as ferramentas para apoiar o desenvolvimento. Inicialmente foi escolhido o visualizador que melhor se enquadrasse nas necessidades do projeto, para isto foi feito um levantamento de requisitos do projeto e de algumas ferramentas que poderiam ser utilizadas na sua visualização. Na Tabela 3-2 é ilustrada a justificativa para a escolha do Xj3D (XJ3D, 2009) como ferramenta de visualização, pois é uma ferramenta livre, possui tutorias de implementação e uso, foi desenvolvido na linguagem Java e tem uma API para programação em Java, além de oferecer suporte à comunicação (através do SAI – *Scene Access Interface*) com processos externos, neste caso as Redes de Petri Coloridas. Para maiores detalhes sobre o estudo realizado consultar o Anexo A.2.

Para edição não foi definida uma ferramenta específica deixando-se a critério do desenvolvedor (programador), de acordo com sua preferência em trabalhar com uma ferramenta de edição de objetos ou com linha de código. A vantagem de programar através da linha de códigos é que facilita a integração com o Java e a vantagem de trabalhar com uma ferramenta de edição gráfica é a facilidade para criar, alterar e posicionar os objetos.

Na Tabela 3-3 é apresentada a relação entre os requisitos e algumas ferramentas utilizadas para o desenvolvimento dos ambientes virtuais analisados. Porém, fica a critério do

desenvolvedor escolher a ferramenta para o desenvolvimento. Outras ferramentas e maiores detalhes são citadas no Anexo A.2.

Tabela 3-2 – Comparativo entre visualizadores

	Xj3D	Freewrl	OpenVRML	Vivaty	SwirlX3D	Octaga	BS Contact
Representação (completude)	✓	✓		✓	✓	✓	✓
Comunicação externa	✓		✓	✓		✓	✓
Visualização remota	✓	✓	✓	✓	✓	✓	✓
Navegação	✓	✓	✓	✓	✓	✓	✓
Múltiplos usuários	✓	✓		✓	✓	✓	✓
Tutoriais	✓	✓		✓		✓	✓
Grátis/Livre	✓	✓	✓				
Compatibilidade X3D	✓	✓		✓	✓	✓	✓
Dispositivos E/S	✓	✓	✓	✓	✓	✓	✓
Multi-Plataforma	✓	✓				✓	

Tabela 3-3 – Comparativo entre ferramentas de edição

	X3D-Edit	Vivaty	BS editor	Blender	3DS Max	Maya	H3D
Tutoriais	✓	✓	✓	✓	✓	✓	✓
Gerar XML consistente	✓		✓	✓			
Múltiplos Modos	✓	✓	✓				
Suporte a Linguagem X3D	✓	✓	✓	✓	✓	✓	✓
Edição Gráfica		✓	✓	✓	✓	✓	✓
Versão grátis	✓	✓		✓			✓

Na implementação deste trabalho foram utilizadas duas das ferramentas listadas, uma para edição através da descrição do código e outra gráfica. As ferramentas utilizadas foram:

- **X3D-Edit** que é um editor de X3D que facilita a escrita de arquivos. A sua vantagem é que permite um domínio maior sobre o código gerado (tornando a codificação mais clara e eficiente). Além de oferecer (embutido) um validador do XML criado. Sua desvantagem é que alguns parâmetros são de difícil configuração. Por exemplo, posicionar os objetos no mundo virtual (*translation*) e rotacioná-los (*rotation*) (X3D-EDIT, 2009).
- **Vivaty Studio** é uma ferramenta de edição gráfica. Sua vantagem é a facilidade para conceber e posicionar os objetos. Sua desvantagem é gerar um código "cheio de lixo", dificultando a configuração com Java. Para qualquer edição do ambiente é preciso utilizar a ferramenta, dificultando trabalhos futuros. Com isso ele foi utilizado para auxiliar na identificação de parâmetros dos objetos (por exemplo, localizando valores de rotação e translação) (STUDIO, 2009).

Após escolher as ferramentas de trabalho e a linguagem na qual a nova versão do simulador seria desenvolvida, foi estudada a linguagem VRML e a sua evolução – o X3D, para poder entender a codificação já existente (VRML) e aprimorar a nova versão (X3D). Nos Quadro 3-1 e Quadro 3-2 são descritos trechos de código para exemplificar a codificação em cada linguagem e como se pode observar, os códigos são bem semelhantes. Além das vantagens já mencionadas, o X3D refinou a programação e adicionou novas funcionalidades como: *X3DSequencerNode, BooleanFilter, BooleanSequencer*, que consistem em funções para criar animação no mundo virtual.

O código presente nos quadros abaixo descreve uma Cena (*Scene*) em VRML e em X3D, envolvendo uma chave: "Disjuntor_12J5" e contém os parâmetros:

- Ponto de vista na Cena: *viewpoint*;
- Posição: *translation*;
- Rotação: *rotation*;
- Cor: *diffuseColor*;
- Tipo de objeto: *geometry Box*;
- Tamanho do objeto: *size*;
- Sensor de clique: *touchsensor*;

O código do Quadro 3-2 é estruturado no formato de árvore, fazendo a analogia com XML, descrevendo os elementos, sub-elementos e atributos.

```
Viewpoint {
 description "'Disjuntor 12J 5" orientation -1 0 0 0.78  position 0 2.04 2.34
}
Group{
```

```
  children [
        # Chave que gira quando o usuario clica sobre ele
        DEF Disjuntor_12J5 Transform {
              translation -7.5 -1.1 37.5
              rotation 0.0 1.0 0.0 3.14
              center 0.0 0.0 0.0
              children Shape {
                    appearance Appearance {
                          material Material { diffuseColor  0.75 0.75 0.75}
                    }
                    geometry Box  {size 0.2 0.6 0.0}
              }
        },
        # Sensor, detecta quando o usuario clica em um objeto
        DEF Disjuntor_12J5_Sensor TouchSensor { },
  ]
}
```

Quadro 3-1 – Descrição de uma chave em VRML

```
  <Viewpoint description='Disjuntor 12J5' orientation='-1 0 0 0.78' position='0 2.04 2.34'/>
<Group >
   <Transform  DEF="Disjuntor_12J5"  translation="-7.5 -1.1 37.5" rotation="0.0 1.0 0.0  3.14">
     <Shape >
       <Appearance >
         <Material  diffuseColor="0.75 0.75 0.75"/>
       </Appearance>
       <Box  size="0.2 0.6 0.01"/>
     </Shape>
   </Transform>
   <TouchSensor  DEF="Disjuntor_12J5_Sensor" />
 </Group>
```

Quadro 3-2 – Descrição de uma chave em X3D

Como um dos objetivos deste trabalho era reaproveitar o código da descrição do mundo virtual da versão anterior do simulador, além das ferramentas de edição, foi utilizada uma ferramenta para apoiar a conversão entre VRML e X3D. O conversor foi o Vrml97toX3DNist (VUG[2], 2009) o qual foi desenvolvido pelo grupo Web3D.

O conversor está disponível para download em (VUG, 2009). Para utilizar o conversor é necessário extrair o pacote: "Vrml97ToX3dNist.jar" e executá-lo através de linha de comando, indicando a pasta onde está o arquivo que deseja converter. Após utilizar este conversor percebeu-se que alguns parâmetros (como os ROUTES – Rotas para movimentar os objetos no mundo virtual) não foram convertidos automaticamente. Então, para isso foi desenvolvido, neste trabalho, um pacote complementar em Java para converter automaticamente estas ROUTE para o formato XML, facilitando assim o trabalho de conversão.

[2] *Visualization and Usability Group*, grupo que desenvolveu a ferramenta de conversão.

Na Figura 3-7 é ilustrada a classe complementar desenvolvida para fazer a conversão do código VRML, a qual é composta pelos seguintes métodos:

- convertLine( ): conversor das linhas do ROUTE de VRML para X3D (XML);
- writeFile( ): escreve e salva o novo arquivo convertido;
- convertFile( ): lê as linhas convertidas e escreve no arquivo;
- main( ): método principal;
- normalize( ): normaliza as linhas, retirando comentários e linhas em branco.

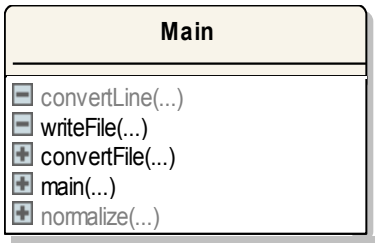

Figura 3-7 – Classe complementar do conversor

Outro problema constatado durante a conversão é que o arquivo gerado possuía diversas falhas de declaração, não executando corretamente no visualizador. Isto ocorre porque quando o arquivo é convertido para XML os parâmetros (formatos) são analisados conforme a norma, possuindo um arquivo DTD[3] (*Document Type Definition*) de validação. Então, para conseguir executar o arquivo foi necessário validar e corrigir todos os parâmetros e nomenclatura, utilizando o sistema de validação de arquivo XML presente na ferramenta X3D-Edit.

Durante a validação da nomenclatura usou-se a oportunidade para modularizá-la padronizando o nome: das chaves; dos objetos; e dos sensores de toque (*touchsensors*). Outra otimização do código consistiu no melhor posicionamento dos objetos no mundo virtual, tornando-o mais fiel ao mundo real. Na Figura 3-8 e na Figura 3-9 é ilustrado um painel e pode-se observar algumas diferenças como o re-posicionamento dos objetos e inserção de medidores digitais.

Outras melhorias também foram acrescentadas ao mundo virtual dentre elas pode-se destacar: a adição de medidores digitais e analógicos; e a modularização do código para facilitar a reprodução e a adaptação para aceitar os parâmetros pela representação CPN do campo, pois a versão anterior só recebia os valores dos medidores através da interface e não se comunicavam com o motor de simulação (modelo CPN da planta).

[3] Documento de validação de arquivos XML contém a descrição dos tipos contidos no arquivo XML.

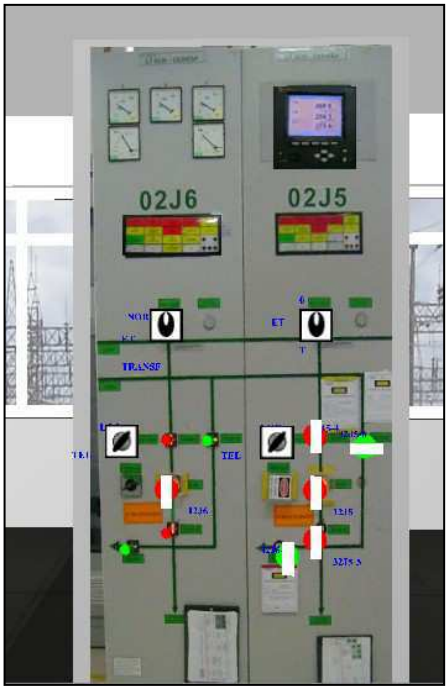

Figura 3-8 – Painel Versão VRML

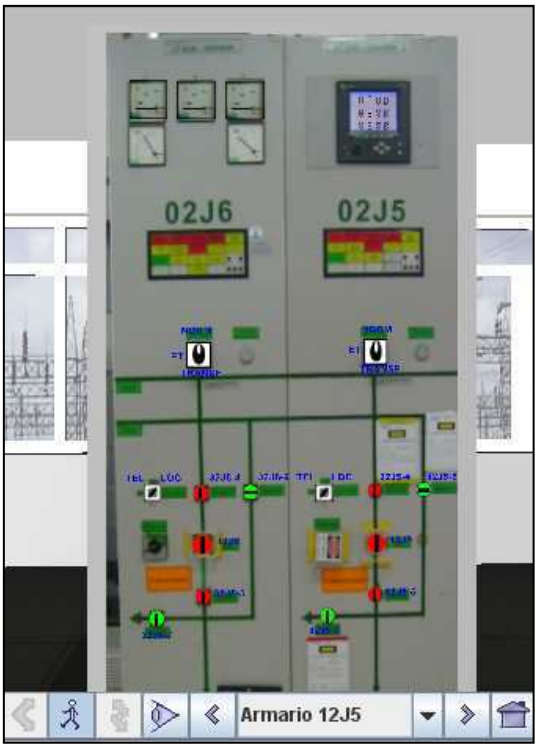

Figura 3-9 – Painel Versão X3D

Após a elaboração do AV foi necessário integrar a representação X3D ao Java. Na Figura 3-10 é ilustrada a arquitetura do visualizador Xj3D (X3D Browser). Nela observa-se que troca de informações entre o visualizador e uma aplicação externa ocorre através do *Scene Access Interface* – SAI definido pela norma ISO/IEC 19775-2:2004, que é a interface de programação (API) usada para fazer esta integração. Existem duas formas de utilizar o SAI que consistem no acesso interno e externo.

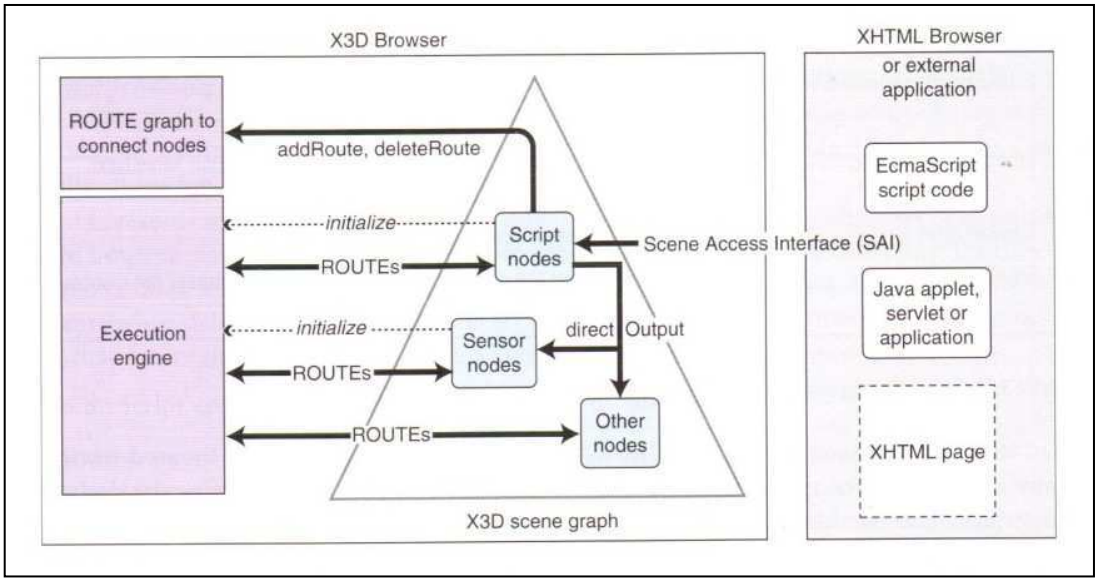

Figura 3-10 – Arquitetura do visualizador Xj3D (Fonte: BRUTZMAN & DALY, 2007)

No acesso interno existe um nó de ***script*** dentro do arquivo X3D que vai fazer a ligação com uma classe Java. Ou seja, é a própria cena X3D que, ao ser carregado em um visualizador de X3D, vai fazer a chamada para classe Java efetuar as alterações pretendidas na cena. Já no acesso externo, a classe Java carrega o arquivo X3D e faz qualquer tipo de alteração na cena correspondente. Neste tipo de acesso, a classe chama uma instância de um visualizador e usa a API para receber as informações ou efetuar alguma ação na cena (por exemplo, inserir ou remover objetos e alterar nós da mesma cena). Este tipo de acesso é útil quando se pretende uma aplicação composta por duas janelas, uma janela referente à cena 3D e outra referente à interface Java. Neste caso será utilizado a acesso externo pois para o mundo do tutor serão necessárias as duas interfaces.

O uso do Xj3D facilitou a integração com Java, pois como a ferramenta foi desenvolvida na mesma linguagem, já existe uma API "org.web3d.x3d.sai" para fazer esta integração utilizando SAI. O código descrito no Quadro 3-3, consiste na classe desenvolvida para inicializar um AV (visualizador Xj3D) através da linguagem Java e é definido o arquivo X3D que se deseja visualizar.

```
import java.awt.BorderLayout;
import javax.swing.*;
import java.util.HashMap;
import org.web3d.x3d.sai.*;

public class CenaX3D extends JFrame {

  public CenaX3D() {
     JFrame frame = new JFrame();

     // Configura parametros do navegador
     HashMap requestedParameters = new HashMap();

     // Cria um componente SAI (Scene Access Interface)
     X3DComponent x3dComp = BrowserFactory.createX3DComponent(requestedParameters);

     // Adiciona um componente na Interface do usuario
     JComponent x3dPanel = (JComponent)x3dComp.getImplementation();
     frame.add(x3dPanel, BorderLayout.CENTER);

     // Cria um Navegador externo
     ExternalBrowser x3dBrowser = x3dComp.getBrowser();

     // Cria uma cena em X3D carregando a partir do arquivo.x3d.
     X3DScene mainScene = x3dBrowser.createX3DFromURL(new String[] { "arquivoX3D.x3d" });

     // Substitui o mundo atual por uma nova cena
     x3dBrowser.replaceWorld(mainScene);

     //Configuracoes da tamanho da tela do navegador
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(1024,768);
     frame.setVisible(true);
```

```
  }

   public static void main(String[] args) {
    //Inicia o programa
     CenaX3D demo = new CenaX3D();
  }
}
```

Quadro 3-3 – Código para inicialização de Ambiente Virtual utilizando Java e Xj3D

Executando o código do Quadro 3-3, o visualizador irá abrir o Ambiente Virtual desenvolvido para o treinamento de operadores na sala de operação e comando de uma subestação elétrica, como ilustrado na Figura 3-11 a qual representa a versão atual do simulador.

Figura 3-11 – Ambiente Virtual do simulador

3.6.4 O motor de simulação

Um modelo é uma representação das principais características de interesse de um objeto ou de um sistema e, pode consistir em uma descrição matemática ou uma representação física. A utilização de métodos formais na modelagem de sistemas se baseia no uso de notações com semântica definida, favorecendo o estabelecimento de métricas de desempenho e possibilitando a discussão de propriedades do sistema modelado (SCAICO et al, 2001).

Para o desenvolvimento deste trabalho motor de simulação consiste nos modelos formais que representam (descrevem) o funcionamento dos objetos presente em uma subestação elétrica. Neste caso foram utilizadas Redes de Petri Coloridas (CPN) como motor de simulação, pois trata-se de um formalismo bastante utilizado no LIHM com diversos trabalhos (AGUIAR et al, 2009; SCAICO et al, 2001). Dentre estes trabalhos, destaca-se a biblioteca de objetos que serviu de base para o motor de simulação (NASCIMENTO et al,

2007). Devido à facilidade de validar o funcionamento dos objetos, diferentemente de um motor de simulação construído em uma linguagem de programação, pode ser verificado e validado usando as ferramentas disponíveis, tais como CPN Tools, e permite a comunicação com processos externos utilizando TCP/IP via *socket*.

Rede de Petri é um formalismo matemático com representação gráfica, aplicável a vários tipos de sistemas (MURATA, 1989). Sua representação gráfica consiste em um conjunto de arcos direcionais, lugares e transições. Já Redes de Petri Coloridas é uma evolução, pois as fichas podem possuir atributos (cores – valores associados às fichas) (JENSEN, 1992). Na Figura 3-12 é ilustrado o grafo de Redes de Petri Coloridas que representa uma chave do tipo local – telecomando. O nó de declaração e a página de hierarquia, que completam a representação desta rede fazem parte da biblioteca de modelos e são apresentados no Anexo C.

Em conjunto com o desenvolvimento do Ambiente Virtual foram elaborados os modelos em CPN para representar o funcionamento dos objetos de interação de uma subestação. Em um trabalho anterior do grupo foi modelada uma biblioteca dos objetos presentes nos painéis de uma sala de controle e operação de uma subestação. Dado o contexto da época, foi utilizada a ferramenta Design/CPN para modelar os objetos, porém o apoio a esta ferramenta foi descontinuado além dela só poder ser executada na plataforma Linux. Os desenvolvedores deixaram de dar suporte, ao lançarem uma nova ferramenta denominada CPN Tools que pode ser executada tanto na plataforma Linux quanto Windows, além de corrigir algumas falhas (*bug's*) da ferramenta anterior.

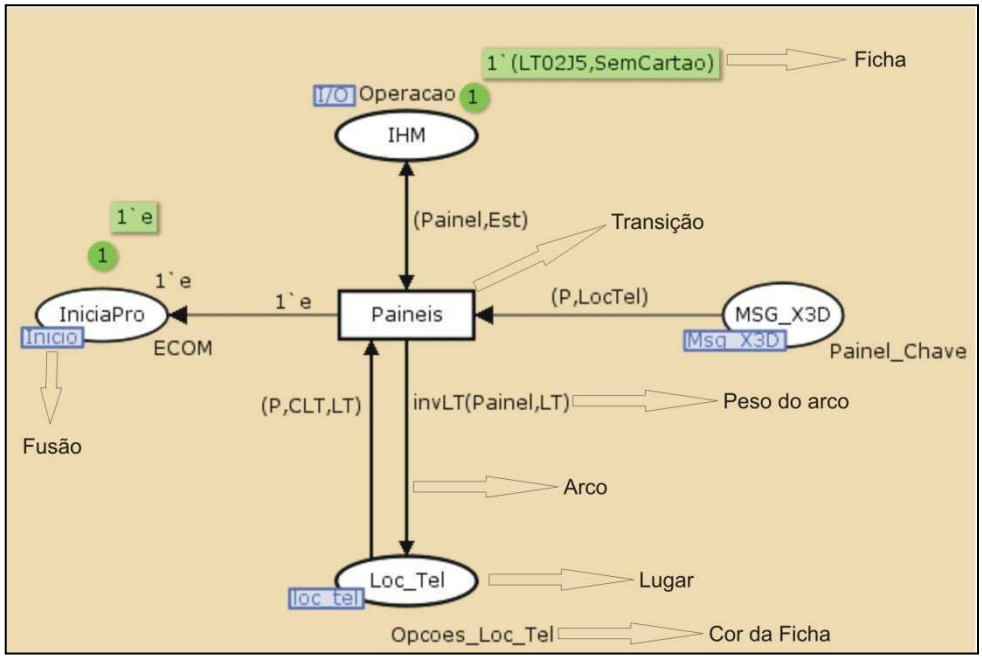

Figura 3-12 – Ilustração do grafo de Redes de Petri Coloridas (CPN)

Visando reutilizar os modelos existentes, estes foram convertidos para a nova ferramenta, no caso CPN Tools. Nas figuras (Figura 3-13 e Figura 3-14) são ilustrados como exemplo o grafo da chave do tipo Giro-Giro nas duas ferramentas. Neste caso a descrição das funções, cores e variáveis não foram alteradas e se encontram no Anexo C.

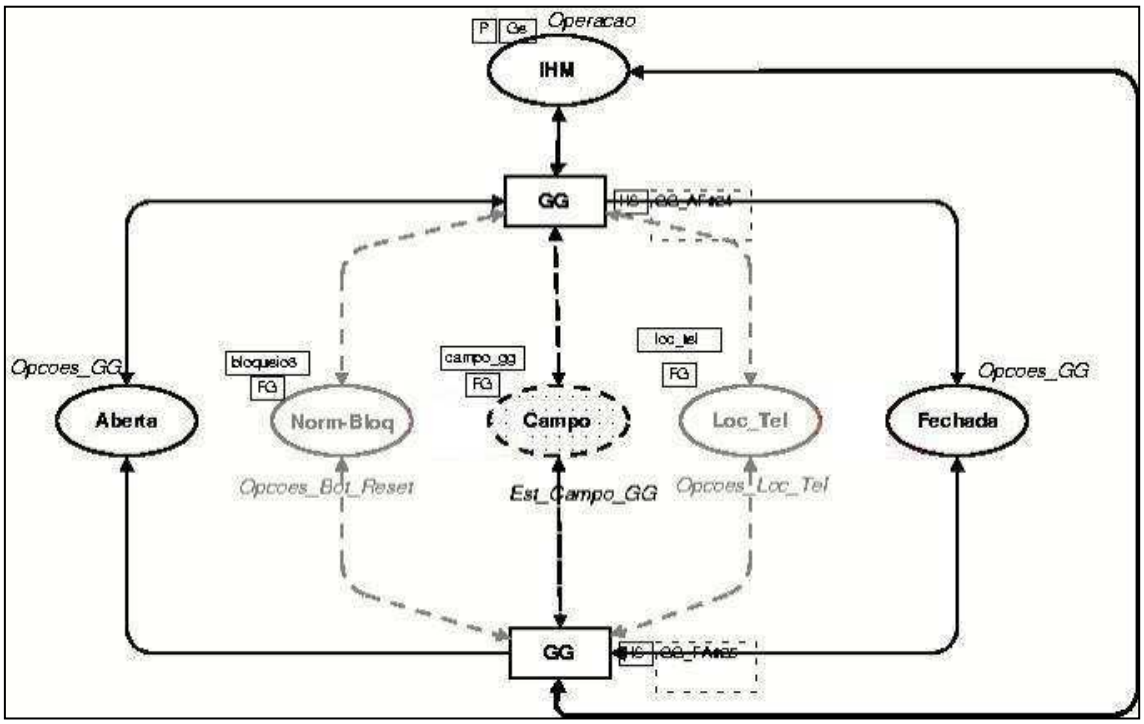

Figura 3-13 – Grafo do Modelo de Chave em Design/CPN (Fonte: NASCIMENTO et al**, 2007)**

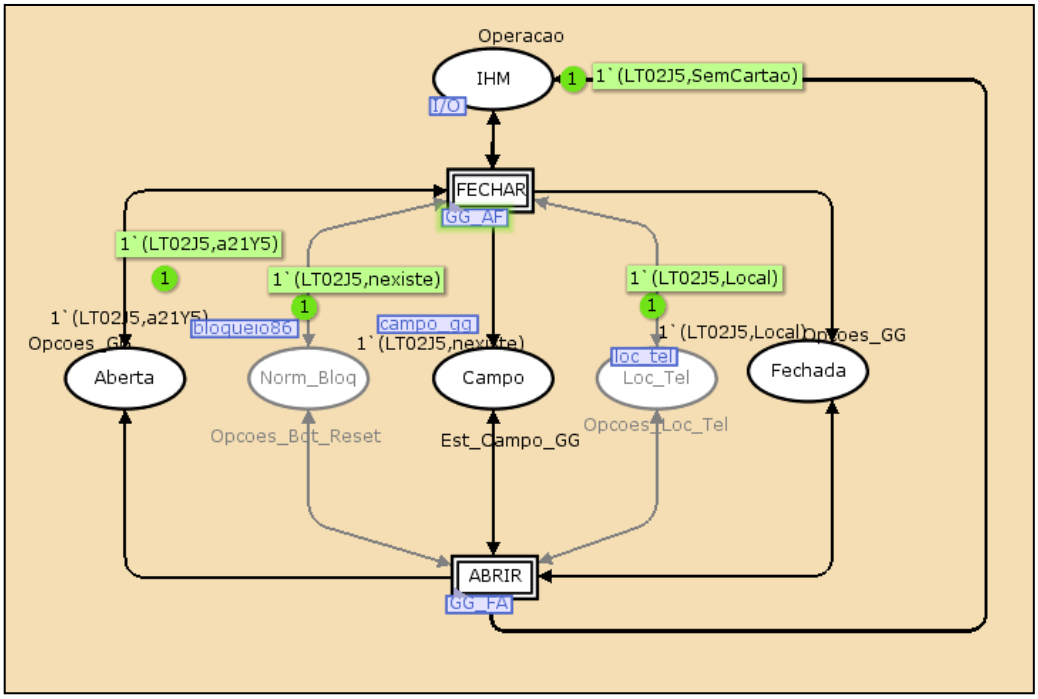

Figura 3-14 – Grafo do Modelo de Chave em CPN Tools

O motor de simulação é composto por um conjunto de modelos em CPN que estão divididos em três grupos e podem ser executados em máquinas distintas. Os grupos são:

- CPN que modelam os comportamentos dos objetos da sala de controle, mais especificamente os painéis;
- CPN que modelam o campo (planta), mais especificamente a área externa da subestação (por exemplo, os disjuntores e seccionadoras);
- CPN que modelam os supervisório, onde é modelada a interface com o software do tipo SCADA. O diagrama ilustrado na Figura 3-15 representa os modelos CPN utilizados e a comunicação entre eles, através de socket e da biblioteca Comms/CPN.

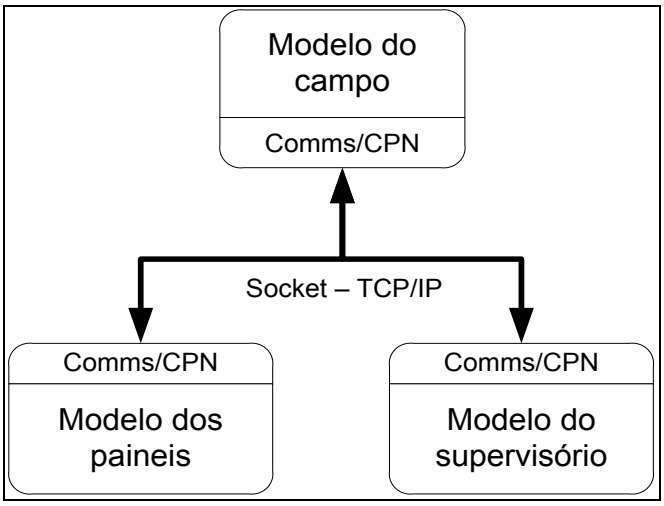

Figura 3-15 – Diagrama de Comunicação entre modelos

Vale salientar que as redes convertidas relativas à biblioteca dos painéis, não estavam adaptadas para serem utilizadas como motor de simulação. Isto é, não estavam integradas segundo a proposta da arquitetura. Esta adaptação, realizada neste trabalho será apresentada no próximo capítulo.

A rede de campo (planta) utilizada neste trabalho já foi desenvolvida diretamente em CPN Tools (TORRES & VIERA, 2010), o qual também converteu os modelos que representam o módulo supervisório do mundo virtual do simulador. No trabalho aqui descrito, o enfoque é voltado para as redes que modelam comportamento dos painéis, pois estas atualmente são a parte funcional do motor de simulação ligada diretamente à interface do mundo virtual do simulador.

3.6.5 Considerações

Este capítulo abordou a proposta de arquitetura para a nova versão do simulador levando em consideração os requisitos levantados. Como citado anteriormente, este trabalho não tem como escopo desenvolver todos os aspectos da arquitetura proposta. O enfoque deste trabalho é assegurar a adaptação dos módulos existentes à nova proposta de arquitetura e o funcionamento do módulo mínimo de treinamento, que consiste na integração do Ambiente Virtual, a partir da troca de mensagens, com o motor de simulação; e apresentar sugestões para o desenvolvimento dos demais módulos com base na proposta de arquitetura.

Além de apresentar a proposta da arquitetura neste capítulo foi apresentado o método adotado na concepção do AV. Além disso, foi apresentado o contexto de uso do simulador e o perfil dos usuários que irão utilizá-lo. Foram escolhidas a linguagem e as ferramentas para o desenvolvimento do Ambiente Virtual, além de apresentar alguns dos modelos CPN que compõem o motor de simulação.

Após modelar os objetos da subestação surgiu a dificuldade de comunicar o motor de simulação com o Ambiente Virtual. Para desenvolver esta comunicação foi utilizado *socket* via TCP/IP, pois é o modo de comunicação apoiado pela ferramenta. Para realizar esta comunicação foi desenvolvido um *driver* que interpreta mensagens enviadas pelo motor de simulação para o AV e vice-versa, o qual será descrito no próximo capítulo.

Capítulo 4 Desenvolvendo a comunicação.

Neste capítulo será abordado o mecanismo de comunicação entre os modelos do motor de simulação e os módulos da arquitetura do simulador. Nele é descrita a integração entre o motor de simulação e o Ambiente Virtual, e como é realizada a troca de mensagens. Para esta integração utiliza-se a linguagem Java e a biblioteca Comms/CPN (detalhada no Anexo B - serve para comunicar as redes CPN com processos externos - neste caso com o visualizador de Ambiente Virtual Xj3D). Porém, para trocar mensagens adequadamente foi desenvolvido um *driver* que converte as informações do Ambiente Virtual para mensagens “que são interpretadas” pelas CPN e vice-versa. Na Figura 4-1 é ilustrada a arquitetura básica do simulador, a qual consiste no Ambiente Virtual, visto através do visualizador (Xj3D) e no

driver desenvolvido em Java utilizando a ferramenta Eclipse (ECLIPSE, 2009), comunicando com o motor de simulação (modelos CPN executados na ferramenta CPN Tools).

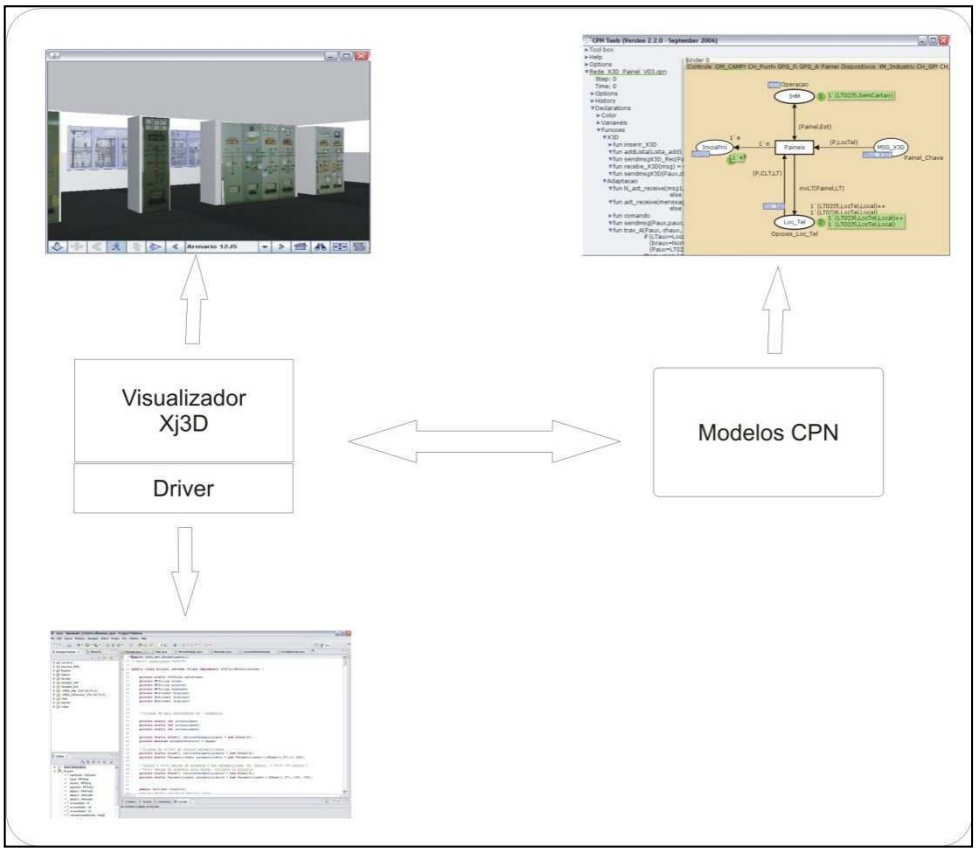

Figura 4-1 – Arquitetura do ambiente do operador

4.1 CPN geradas para comunicação.

Como apresentado no capítulo anterior o motor de simulação é composto por três grupos de redes: as redes que representam o funcionamento dos painéis presentes na sala de controle da subestação; as redes que representação o campo (a subestação); e as redes que representam o funcionamento do módulo supervisório. Neste trabalho, além de converter as redes da biblioteca de modelos, foi necessário desenvolver outras redes para integrar o campo com o AV.

Na Figura 4-2 é ilustrada a nova página de hierarquia para o modelo desenvolvido. No anexo C é abordado à hierarquia antes das alterações realizadas neste trabalho.

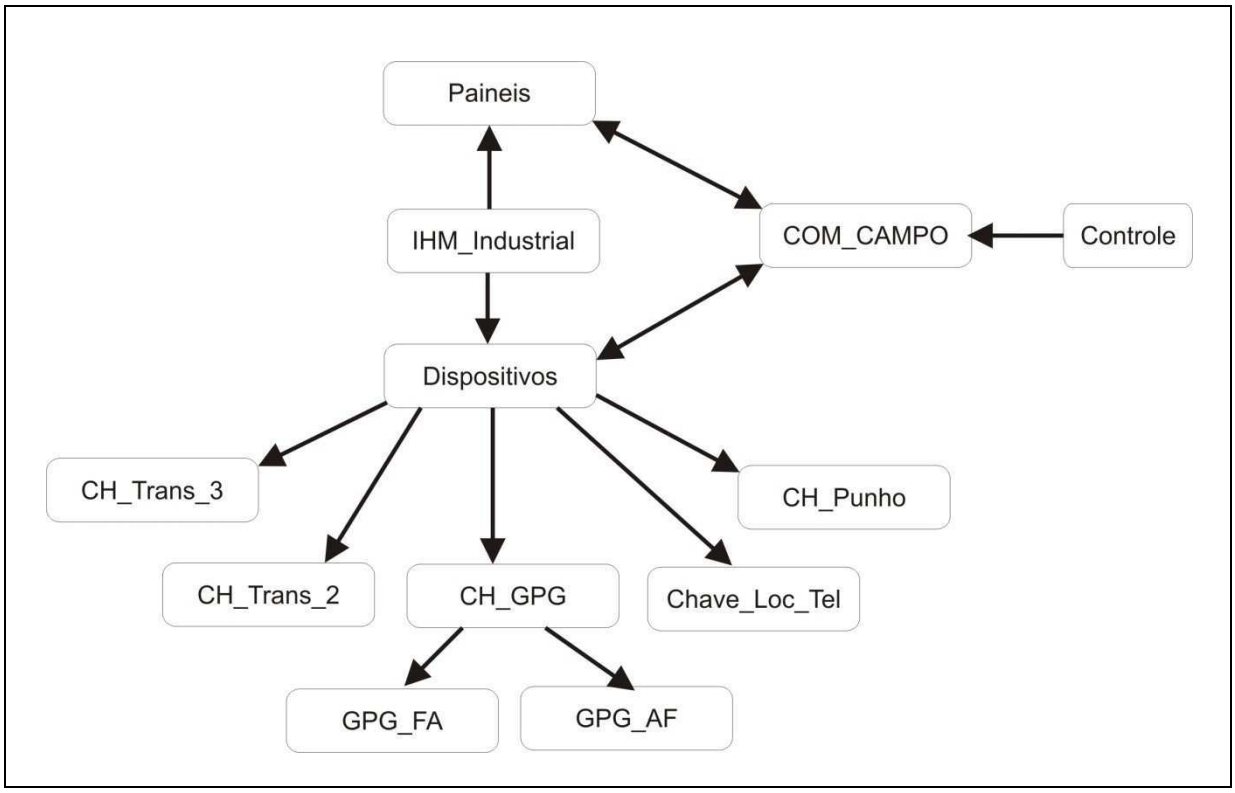

Figura 4-2 – Nova Página Hierárquica.

Para fazer a comunicação (troca de mensagens) da nova Interface (AV) com o motor de simulação foi elaborada a rede ilustrada na Figura 4-3 que representa a **COM_CAMPO**, onde é feita a comunicação entre a rede CPN que representa os painéis e o AV e o campo. Para isto, foi necessário criar cores, variáveis e funções as quais são apresentadas no Quadro 4-1, Quadro 4-2 e Quadro 4-3 respectivamente.

```
colset MSG = string;
colset ECOM = with e;
colset Pcampo = with a;
colset Lista_Pcampo = list Pcampo;
colset Armario = with LT02J6 | LT02J5 | LT02J4 | LT02J3 | LT02J2 | LT02J1;
colset Estado_Campo = with Aberto | Fechado | Nulo;
colset Chaves = with DJ12J1 | DJ12J2 | DJ12J5  | DJ12J6 | SC32J54 | SC32J55 |SC32J56|SC32J57| DJ12D5 |
SC32J64 | SC32J66 | SC32J65 |SC32J67 |LocTel |Transf_3;
colset Est_Campo = product Armario * Chaves * Estado_Campo;
colset Painel_Chave = product Armario*Chaves;
colset Lista_CMD = list Est_Campo;
```

Quadro 4-1 – Declaração das Cores.

No Quadro 4-1 são apresentadas as cores das fichas, onde:

- **MSG:** representa as mensagens recebidas do modelo;

- **ECOM**: modela um evento (ficha genérica);
- **Pcampo**: identifica quando a mensagem é recebida;
- **Lista_Pcampo**: A quantidade identifica se a mensagem recebida é advinda diretamente do campo, ou se é uma resposta de uma interação no Painel;
- **Armario**: representa o nome do armário modelado na Interface (AV);
- **Estado_Campo**: Define os possíveis estados da chave no campo;
- **Chaves**: Representa os dispositivos de interação com os quais o usuário pode interagir através da Interface (AV);
- **Est_campo**: é uma tupla *Armário* x *Chave* x *Estado_Campo*. Relaciona o armário com uma chave e o estado recebido do campo;
- **Painel_Chave**: é uma dupla *Armario* x *Chave*. Relaciona um painel com uma chave a qual foi acionada na Interface (AV);
- **Lista_CMD**: Representa a lista de mensagens recebidas do campo.

```
var ch, chaux, cCampo : Chaves;

var msg, menssage, msg1,msg2: MSG;

var lista, lista_Aux: Lista_CMD;

var Lista_pCampo,Lista_add: Lista_Pcampo;

var pCampo: Pcampo;

var P, Painel, Paux: Armario;

var E, Est, Est1, Eaux: Estado;

var camp, campaux: Estado_Campo;
```

Quadro 4-2 – Declaração das variáveis.

```
fun inserir_X3D(mens)= case mens of  "LT02J6,SC32J66" => 1`(LT02J6,SC32J66) | "LT02J6,SC32J65" =>
1`(LT02J6,SC32J65) | "LT02J6,SC32J64" => 1`(LT02J6,SC32J64) | "LT02J6,SC32J67" =>
1`(LT02J6,SC32J67) | "LT02J6,DJ12J6" => 1`(LT02J6,DJ12J6) | "LT02J6,LocTel" => 1`(LT02J6,LocTel)
|"LT02J5,SC32J56" => 1`(LT02J5,SC32J56) | "LT02J5,SC32J55" => 1`(LT02J5,SC32J55) | "LT02J5,SC32J54"
=> 1`(LT02J5,SC32J54) | "LT02J5,SC32J57" => 1`(LT02J5,SC32J57) | "LT02J5,DJ12J5" =>
1`(LT02J5,DJ12J5) | "LT02J5,LocTel" => 1`(LT02J5,LocTel) | _ => empty;

fun addLista(Lista_add) = union Lista_add [a];

fun sendmsgX3D_Rec(Paux,chaux,campaux) = ConnManagementLayer.send("Con_X3D_Rec",
Armario.mkstr(Paux)^","^Chaves.mkstr(chaux)^","^Estado_Campo.mkstr(campaux),stringEncode);

fun recebe_X3D(msg) = ConnManagementLayer.receive("Con_X3D",stringDecode);

fun sendmsgX3D(Paux,chaux,campaux) = ConnManagementLayer.send("Con_X3D",
Armario.mkstr(Paux)^","^Chaves.mkstr(chaux)^","^Estado_Campo.mkstr(campaux),stringEncode);

fun N_act_receive(msg1,msg2) = if (canreceive(msg1) orelse canreceive(msg2)) then empty else 1`e;

fun act_receive(menssage) = if (canreceive(menssage)) then 1`e else empty;

fun comando(menssage) = case menssage of "LT02J5DJ12J1Aberto" => 1`(LT02J5, DJ12J1, Aberto) |
"LT02J5DJ12J1Fechado" => 1`(LT02J5, DJ12J1, Fechado) | "LT02J5DJ12J5Aberto" => 1`(LT02J5, DJ12J5,
Aberto) | "LT02J5DJ12J5Fechado" => 1`(LT02J5, DJ12J5, Fechado) | "LT02J5SC32J54Fechado" =>
1`(LT02J5, SC32J54, Fechado) | "LT02J5SC32J55Fechado" => 1`(LT02J5, SC32J55, Fechado) |
"LT02J5SC32J54Aberto" => 1`(LT02J5, SC32J54, Aberto) | "LT02J5SC32J55Aberto" => 1`(LT02J5,
SC32J55, Aberto) | "LT02J5SC32J56Aberto" => 1`(LT02J5, SC32J56, Aberto) | "LT02J5SC32J56Fechado"
=> 1`(LT02J5, SC32J56, Fechado) | _ => empty;

fun sendmsg(Paux,paux,campaux) = ConnManagementLayer.send("Conn_Paineis",
```

Armario.mkstr(Paux)^CH_Punho.mkstr(paux)^Estado_Campo.mkstr(campaux),stringEncode);

Quadro 4-3 – Nó de declaração das funções de envio e recepção de mensagens.

No Quadro 4-3 são apresentadas as funções utilizadas para adaptar as redes da biblioteca construídas para realizar a comunicação com a Interface (AV):

- **recebe_X3D**: Recebe as mensagens vindas da Interface;
- **act_receive**: teste para identificar a existências de mensagens a serem recebidas ou da rede de campo ou da Interface;
- **N_act_receive**: caso não exista mensagem a ser recebida, a ficha retorna ao lugar inicial;
- **inserir_X3D e comando**: converte as mensagens recebidas de *string* para fichas, ou advindas da Interface ou da rede campo;
- **sendmsgX3D_Rec e sendmsg**: envia mensagem para o AV, sendo mensagens vindas diretamente da rede de campo ou da rede de campo em resposta a uma interação na Interface;
- **addLista:** Adiciona um valor à lista.

As funções da Tabela 4-1 estão presentes na página **Controle** e foram utilizadas para abrir os canais de comunicação entre o motor de simulação e o visualizador de AV. Pode-se observar que foram utilizados dois canais de comunicação, um para enviar e receber mensagens advindas da rede de Interface e outro para receber mensagens vindas diretamente do campo.

Na comunicação implementada neste trabalho, um canal aguarda as mensagens vindas diretamente do campo e o outro troca mensagens entre a interface e os modelos, evitando o congestionamento (por espera de mensagens) e facilitando o envio e recebimento das mensagens. Este artifício foi utilizado para identificar a origem das mensagens, já que as mensagens vindas diretamente do campo podem chegar a qualquer momento, precisando que um canal permaneça sempre aberto à sua espera, enquanto as mensagens advindas da interface (visualizador 3D), sempre iniciam a comunicação.

Tabela 4-1 – Codificação para conectar AV - CPN

acceptConnection("Con_X3D", 9985);	Espera conexão a conexão com alguma máquina pela porta 9985.
openConnection("Con_X3D_Rec", "IP_Maquina",9986);	Abre conexão com a máquina de IP descrito pela porta 9986.
closeConnection("Con_X3D"); closeConnection("Con_X3D_Rec");	Fecha a conexão.

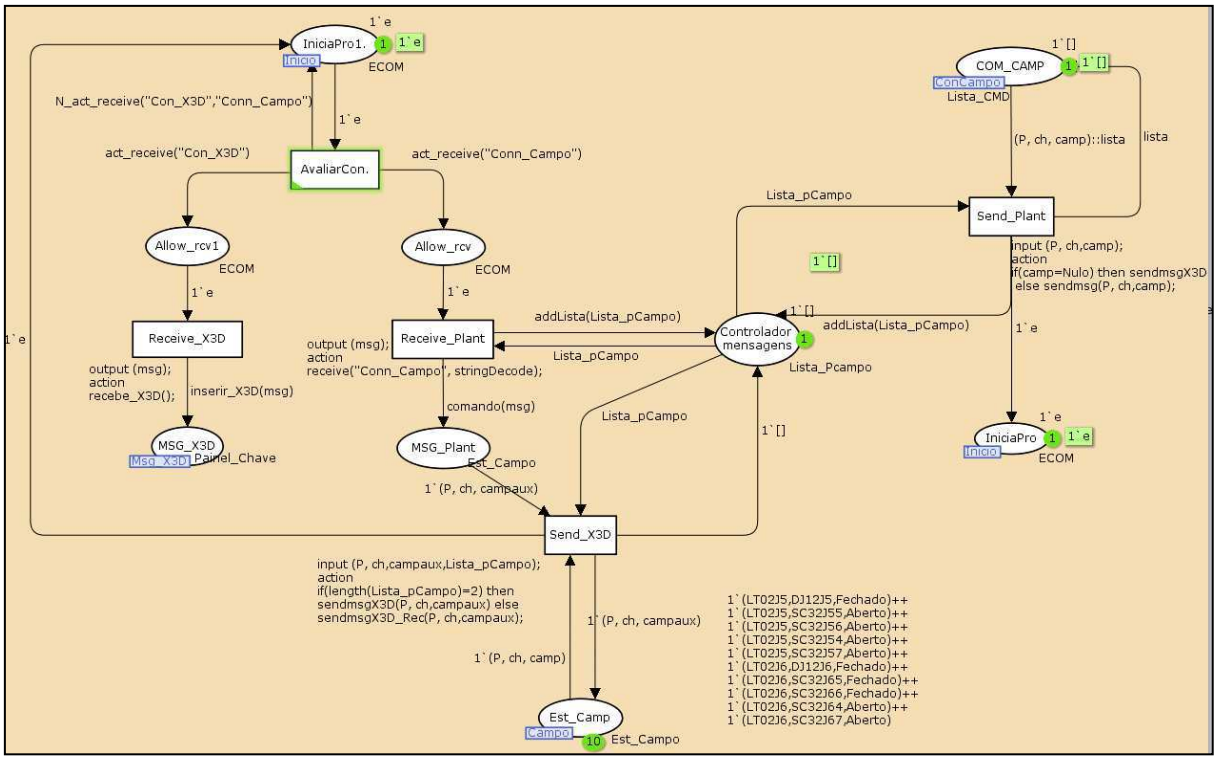

Figura 4-3 – Grafo da Rede de Comunicação no Modelo dos Painéis

Após fazer a conexão entre o *driver* de comunicação e o motor de simulação, que são os comandos apresentados na Tabela 4-1, a execução segue pelo grafo ilustrado na Figura 4-3. Inicialmente a transição "***AvaliarCon.***" é disparada e testada através da função *canreceive* se existe alguma mensagem para ser recebida ou do campo ou da Interface (AV X3D) . Caso seja falso a ficha retorna ao lugar "*IniciaPro1*", caso seja verdadeiro a transição coloca uma ficha no lugar "*Allow_rcv1*" se for uma mensagem vinda da Interface e, no lugar "*Allow_rcv*" se a mensagem vier do campo.

Quando a mensagem recebida é advinda da Interface, a transição "***Receive_X3D***" é disparada e a função *recebe_X3D* converte a mensagem recebida da Interface para uma ficha válida. O lugar "*MSG_X3D*" é um lugar de fusão comum a todas as redes dos painéis. Este lugar contém o nome da chave acionada e o nome do painel ao qual pertence no AV. Então este lugar é quem controla qual a próxima transição que será disparada. Por exemplo, se a chave acionada for do tipo punho, então será a transição da rede do tipo punho, como ilustrado na Figura 4-4 que estará disponível para disparar.

O lugar "*Controlador de mensagem*" foi o artifício utilizado para identificar por qual canal deve ser enviada a mensagem de volta ao AV, pois quando a mensagem é recebida da Interface, enviada ao campo, processada no campo e enviada de volta à Interface; o lugar "*Controlador de mensagem*" fica com dois valores e quando a mensagem vem diretamente do

campo fica com apenas um valor. A transição "***Send_X3D***" retorna a mensagem à Interface, de acordo com o valor deste lugar.

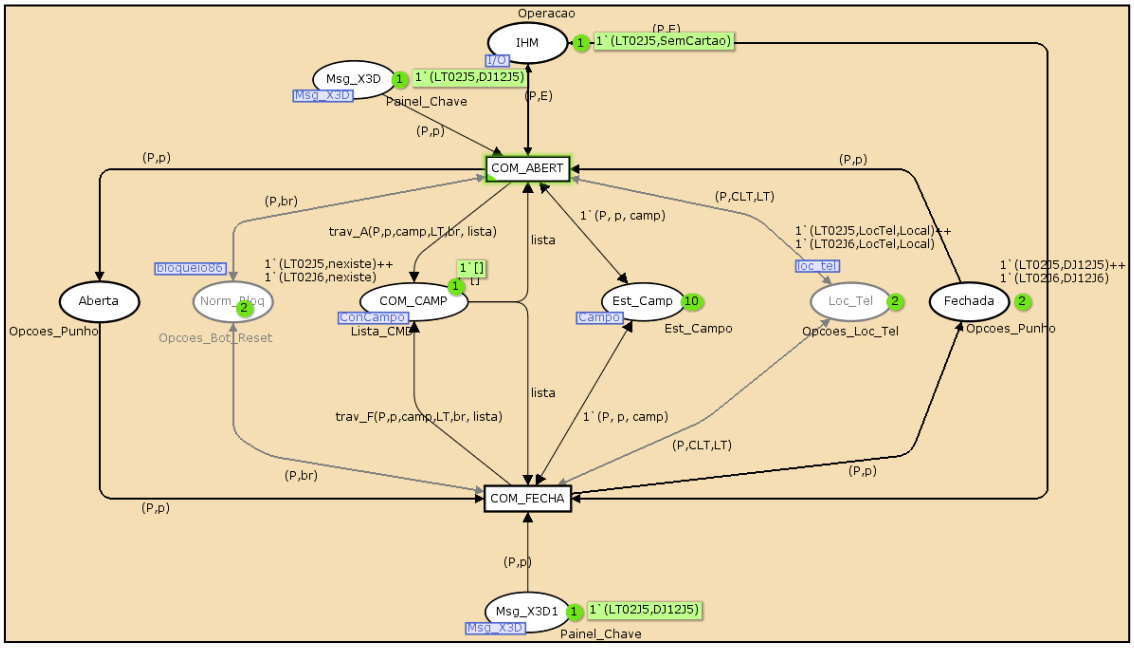

Figura 4-4 – Grafo da Rede da chave tipo Punho Adaptado

Para validar a rede elaborada foi desenvolvido o diagrama de sequência de mensagens (MSC – *Message Sequence Charts*) utilizando a ferramenta BRITNeY (BRITNEY, 2009). Ilustrado na Figura 4-5, o diagrama representa a execução da rede que inicialmente não possuía nenhuma mensagem para receber (Sem Evento). Posteriormente foi recebida uma mensagem advinda da interface do usuário. A mensagem foi processada (neste caso era fechar uma chave), depois a mensagem foi enviada para o campo, que por sua vez retornou a mensagem confirmando a ação e, por fim a mensagem retornou à Interface. Ainda na continuidade da execução foi recebida uma mensagem, de uma ação executada no campo, que foi repassada para a Interface.

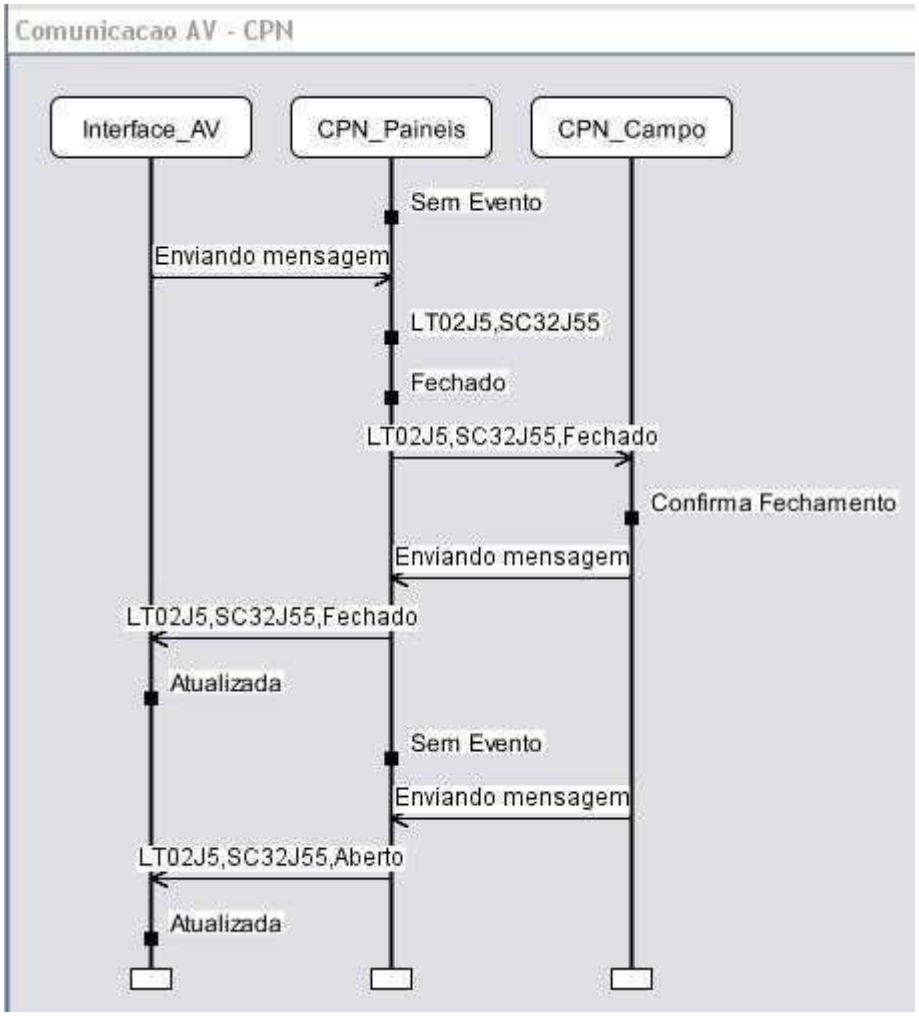

Figura 4-5 – MSC da CPN de comunicação.

A verificação da rede se dá através da construção do espaço de estados, porém para as redes elaboradas o espaço construído não pode ser completo, pois a cada nova mensagem recebida, seja do AV ou das outras redes, houve a geração de um novo espaço de estado, tornando as possibilidades infinitas.

4.2 Driver de comunicação entre o Ambiente Virtual e o Motor de simulação.

Após configurar as Redes de Petri Coloridas para enviar e receber mensagens de um processo externo foi necessário desenvolver o *driver*. O *driver* fará a comunicação entre o visualizador de AV (Xj3D) e a COMMS/CPN que interfaceia com os modelos do motor de simulação, além de interpretar as mensagens vindas das redes CPN para ações a serem executadas no Ambiente Virtual e, converte as ações oriundas do Ambiente Virtual em mensagens válidas para as redes CPN.

A troca de mensagens se dá através de *socket* e as mensagens adotam o formato de *Strings* (conjunto de caracteres). O protocolo de envio utilizado é o TCP/IP e a conexão entre

os modelos e as redes CPN será persistente. Uma vez que o canal de comunicação é aberto, ele é mantido durante toda a conexão, não abrindo uma nova conexão a cada mensagem e fechando apenas ao término do uso do simulador. O Quadro 4-4 exemplifica algumas das mensagens trocadas. No primeiro caso trata-se de uma *string* enviada pelo *driver* e convertida pela Comms/CPN para uma ficha, e no segundo caso é enviada uma string pela Comms/CPN e convertida em uma ação pelo *driver*.

1. **AV => CPN:** "LT02J6,SC32J66" => 1`(LT02J6,SC32J66)
2. **CPN=> AV :** "LT02J5DJ12J1Aberto" => if(recebido.equalsIgnoreCase(nomeRede+",Aberto")) {color.setValue(GREEN);}

Quadro 4-4 – Exemplos de mensagens

Na Figura 4-6 é ilustrada esta comunicação em que o mundo virtual utiliza o *driver* de comunicação para enviar as mensagens para Comms/CPN, na qual a mensagem é interpretada e enviada para o motor de simulação (rede CPN do campo). A rede de campo processa a informação (executa a rede) e devolve a resposta para as redes CPN dos Painéis e as redes CPN do Supervisório. Finalmente o motor de simulação (a rede dos painéis) envia a resposta para o mundo virtual, que através do *driver* interpreta a mensagem e apresenta (atualiza a representação no mundo virtual).

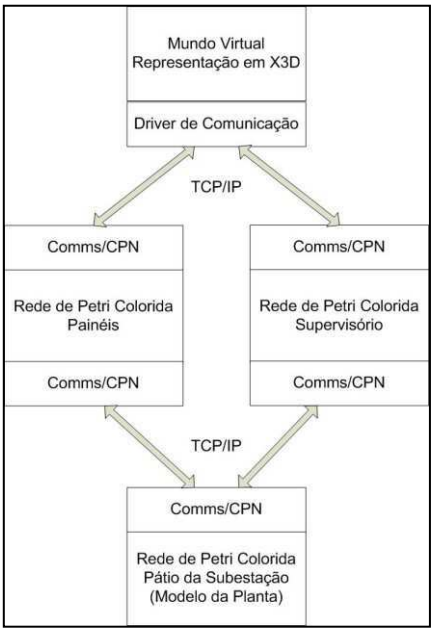

Figura 4-6 – Diagrama de comunicação entre módulos da arquitetura

Na Figura 4-7 é ilustrado o fluxo das mensagens trocadas entre o AV e as CPN, onde o *driver* é representado pelos dois blocos intermediários – Servidor e Browser – que são os pacotes desenvolvidos em Java. Como ilustrado Figura 4-8 o pacote da classe Browser,

consiste na geração (*renderização*) do Ambiente Virtual e na conversão das mensagens (de CPN para ações no mundo virtual).

Figura 4-7 – Diagrama de Fluxo de mensagens

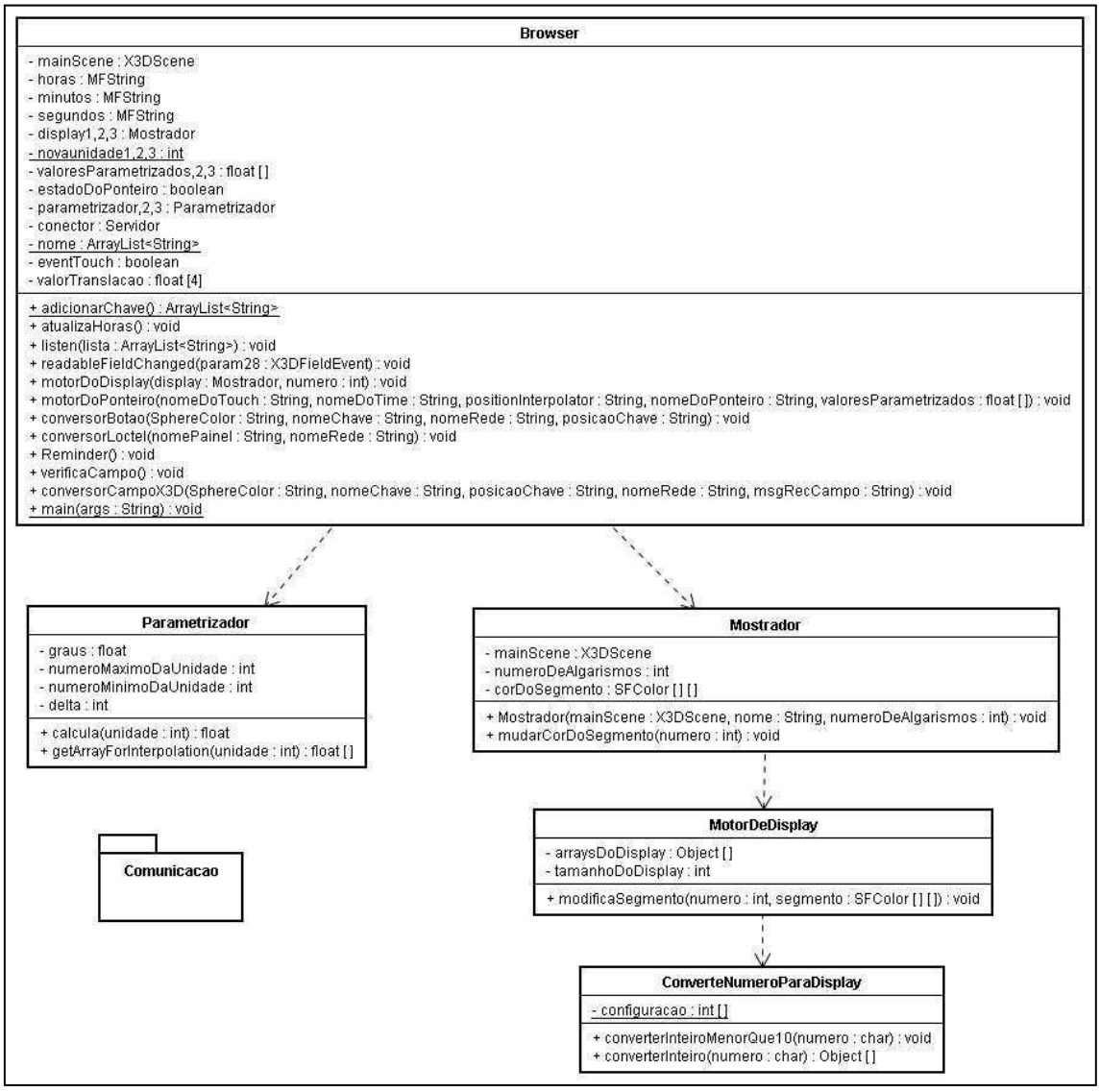

Figura 4-8 – Diagrama de classe do *driver* de comunicação

Para desenvolver está aplicação foram utilizadas as seguintes classes:

i) Browser: esta é a classe principal do programa onde é iniciada a interface de visualização. Esta classe contém as funções de conversão dos valores recebidos e enviados ao motor de simulação. Ela contém também a função que espera as ações do

mundo virtual. Dado que esta classe possui um grande conjunto de métodos, estes estão descritos na Tabela 4-2;

Tabela 4-2 – Descrição dos métodos da classe Browser.

Nome do método	Função
Browser	Inicia o visualizador do Ambiente Virtual. Conecta o motor de simulação com o visualizador e faz a chamada do método *listen*.
adicionarChave	Lista de todas as chaves que foram adicionadas no mundo virtual e que são acompanhadas em tempo real.
listen	Varre a lista de chaves e habilita os sensores de toque (touchsensor) das chaves presentes no Ambiente Virtual. Por fim chama o método de readableFieldChanged quando um sensor é acionado.
readableFieldChanged	Descreve as ações dos sensores que são ativados.
motorDoDisplay	Modifica os valores dos mostradores digitais.
motorDoPonteiro	Modifica os valores dos ponteiros analógicos.
conversorBotao	Converte a ação do clique no objeto em uma ação no Ambiente Virtual para o caso das chaves do tipo giro pressão giro e do tipo punho. Além de gerar as mensagens que serão enviadas para o motor de simulação e converter as mensagens recebidas do motor de simulação em uma ação no Ambiente Virtual.
conversorLoctel	Converte um sinal de toque em uma ação para o caso de chaves do tipo local/telecomando.
verificaCampo	Conecta o canal de comunicação do campo (motor de simulação) com o Ambiente Virtual
reminder	Reabre o canal de comunicação com o campo;
conversorCampoX3D	Converte as mensagens vindas do campo do motor de simulação em uma ação na interface do Ambiente Virtual
main	Inicia a classe e pega os valores dos mostradores.

ii) Mostrador: consiste na classe de representação do funcionamento de um mostrador digital de 7 segmentos que é apresentado no Ambiente Virtual;

iii) MotorDeDisplay: Classe responsável pela atualização de um número para um display;

iv) ConverterNumeroParaDisplay: classe responsável por converter os valores do motor de simulação para mostradores BCD de sete segmentos;

i) Parametrizador: Classe que parametriza dados como tensão, corrente, entre outros, para um ângulo a ser usado no ponteiro analógico;

O pacote de comunicação consiste no diagrama de classe ilustrado na Figura 4-9. Vale salientar que as classes javaCPN, javaInterface e EncodeDecode não foram desenvolvidas, elas são fornecidas pelo grupo que desenvolveu a ferramenta CPN Tools.

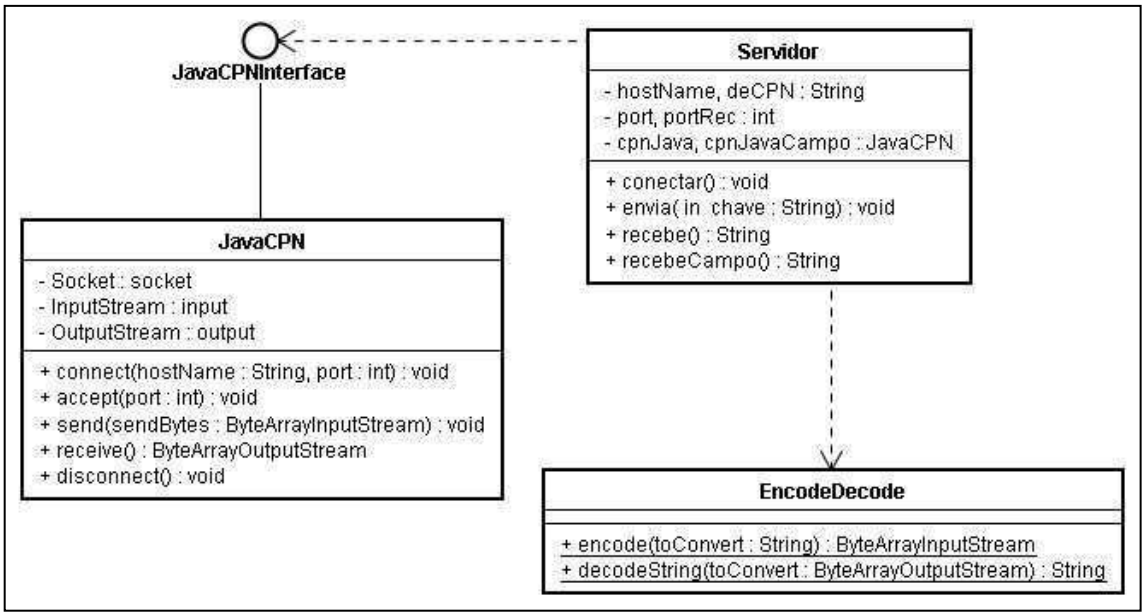

Figura 4-9 – Diagrama de classes/objetos da camada de comunicação

Então sua descrição está no Anexo B. Já a classe "**Servidor**" foi desenvolvida para utilizar as classes fornecidas, nessa classe foram criados os seguintes métodos:

i) **Conecta**: consiste na configuração do IP e das portas de comunicação com os modelos;

ii) **Envia**: Converte a mensagem e envia para o Modelo;

iii) **Recebe**: Recebe a mensagem da Interface e converte para uma String;

iv) **RecebeCampo**: Recebe a mensagem do campo e converte para uma String;

4.3 Validação

O operador poderá realizar ações representadas em vários casos de uso do simulador. Na Figura 4-10 é ilustrado o diagrama de alguns casos de uso, como abrir e fechar chaves (seccionadoras, disjuntores), visualizar medidores, desligar alarmes visuais e sonoros. Para validar o *driver* de comunicação foram elaborados três casos de uso do mundo virtual.

O primeiro caso de uso é fechar uma chave (neste caso o disjuntor 12J5). Para isto é necessário navegar no ambiente do simulador, identificar o painel, a chave e com o uso do mouse fechá-la. Depois de fechar é necessário aguardar a confirmação do campo de que a chave foi realmente fechada.

O segundo caso de uso consiste em tentar abrir uma chave, porém quando existe uma relação de inter-travamento com outras chaves.

O terceiro caso de uso consiste em visualizar no medidor digital, o valor atual da corrente de um Painel (LT12J5).

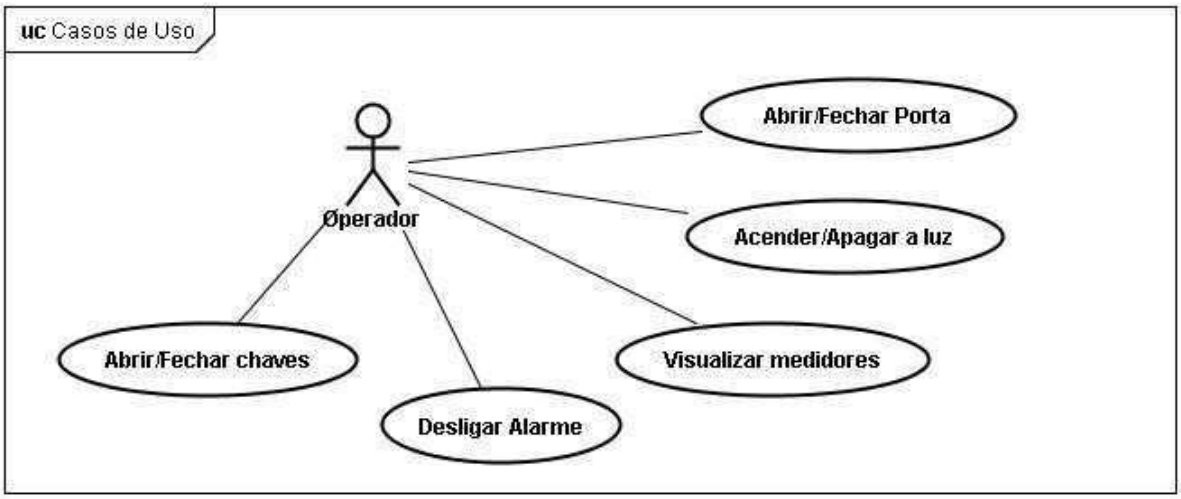

Figura 4-10 – Caso de uso do simulador.

Para compreender as imagens apresentadas nas Figuras 4-11 e 4-12, para os casos de uso de fechar e abrir a chave, é necessário saber que as botoeiras (chaves) dos painéis possuem duas informações: o estado do campo e o estado no painel. As linhas nos painéis indicam as linhas de transmissão. O retângulo com a circunferência, na Figura 4-12, indica a botoeira. Então quando o retângulo (alça) da botoeira está alinhado com a linha do painel indica que a chave está fechada e a luz (representada por um circulo na interface) indica o estado da chave no campo. Quando a luz está verde indica, dispositivo não energizado (aberto), quando está vermelha indica dispositivo energizado (fechado) e, quando está amarela indica que o estado no campo está em discordância com o estado da chave no Painel.

Na Figura 4-11 é exemplificado um painel com diversas botoeiras e a área delimitada foi ampliada na Figura 4-12 para facilitar a visualização da linha de transmissão e ilustrar duas chaves abertas (12J5 e 32J5-6) e duas fechadas (32J5-4 e 32J5-5).

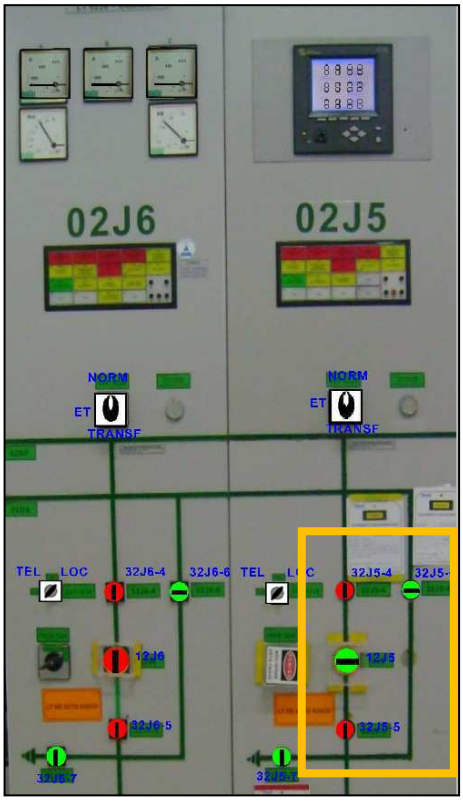

Figura 4-11 – Chave 12J5 Aberta

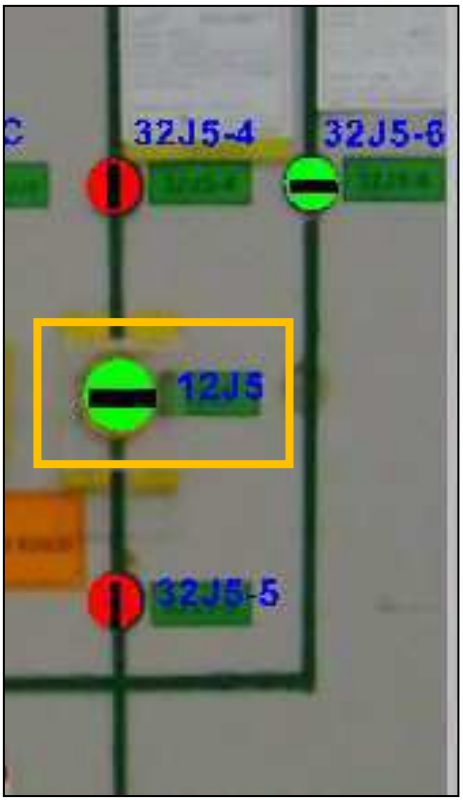

Figura 4-12 – Zoom da área marcada do primeiro Estudo de Caso

O primeiro estudo de caso realizado corresponde à ação do operador de fechar uma chave no Ambiente Virtual e observar a resposta da planta, para saber se ocorreu como o esperado. Para executar esta ação é necessário que o operador se desloque (no simulador) até a frente do painel desejado e clique sobre o objeto que deseja mover (fechar); o qual neste caso consiste no disjuntor 12J5 do painel LT02J5. Na Figura 4-13 é ilustrado o diagrama de seqüência de mensagens (MSC) a partir do momento no qual o usuário atua sobre a botoeira, no AV.

No caso é enviada a mensagem: ‘LT02J5,DJ12J5” para a rede CPN dos painéis; a rede é executada e o disjuntor é fechado. Em seguida é enviada a mensagem: “LT02J5,DJ12J5,Fechada” para a CPN do campo. A rede de campo recebe a informação, fecha o disjuntor no campo e envia de volta a mensagem: “LT02J5,DJ12J5,Fechada” e também manda a mensagem para a CPN do supervisório. O modelo do painel ao receber a mensagem do campo encaminha para a interface, o *driver* em Java recebe e interpreta para uma ação no Ambiente Virtual, que neste caso, como é de concordância, torna a lâmpada indicativa - verde.

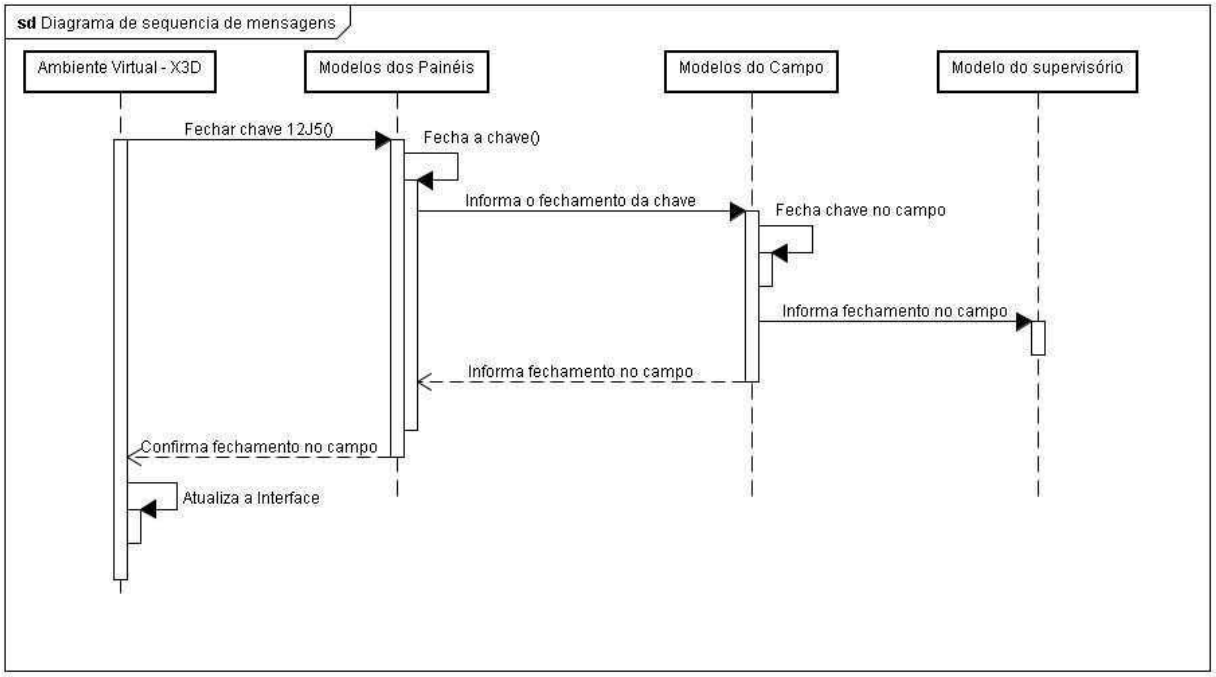

Figura 4-13 – MSC do fechamento de uma chave (com sucesso).

Na Figura 4-14 é ilustrada a ação do usuário "fechar a botoeira do disjuntor 12J5" e na Figura 4-15 é ilustrado a realimentação (a partir dos modelos), indicando que a chave foi fechada corretamente na planta.

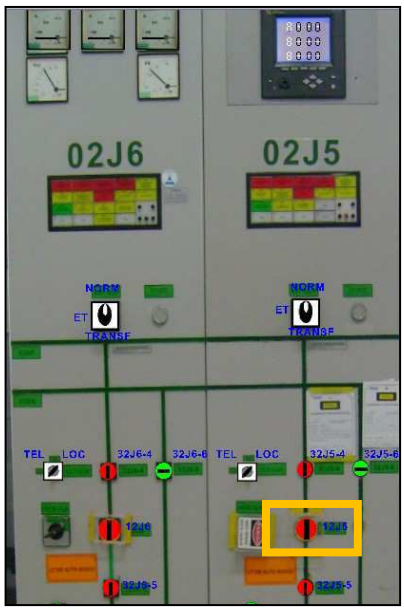

Figura 4-14 – Fechando a Chave

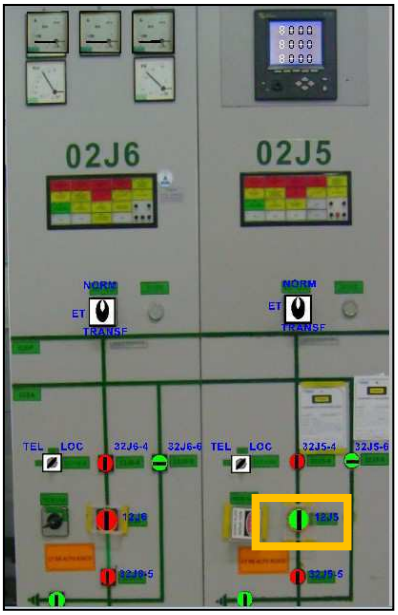

Figura 4-15 – Chave fechada

4.3.1 Segundo Estudo de Caso

No segundo estudo de caso a ação do operador é abrir uma chave no Ambiente Virtual e observar a resposta da planta para saber se ocorreu como o esperado. Para executar esta ação é necessário que o operador se desloque (no simulador) até a frente do painel desejado e clique sobre o objeto que deseja mover (fechar), que neste caso foi escolhido à seccionadora 32J5-4 do painel LT02J5. Na Figura 4-16 é ilustrado o diagrama de seqüência das mensagens (MSC- *Message Sequence Charts*) a partir do momento que o usuário atua sobre a botoeira, no AV.

Neste caso ao selecionar a chave 32J5-4 é enviada à mensagem: "LT02J5,SC32J54" para o modelo dos painéis, a chave é aberta no modelo dos painéis e é enviada a mensagem: "LT02J5,SC32J54,Aberta" para o campo. Porém no campo não é possível abrir essa chave, então é retornada a mensagem: "LT02J5,SC32J54,Fechado" para o modelo dos painéis e do supervisório informando que a chave continua fechada. E, a mensagem é enviada à interface que indica através da lâmpada, que a chave está em discordância, através da cor amarela. A chave seccionadora 32J5-4 não pode ser aberta devido a um inter-travamento, pois depende do estado do disjuntor principal 12J5 que deve estar aberto, para que possa ser aberta. Esta discordância está ilustrada na Figura 4-17.

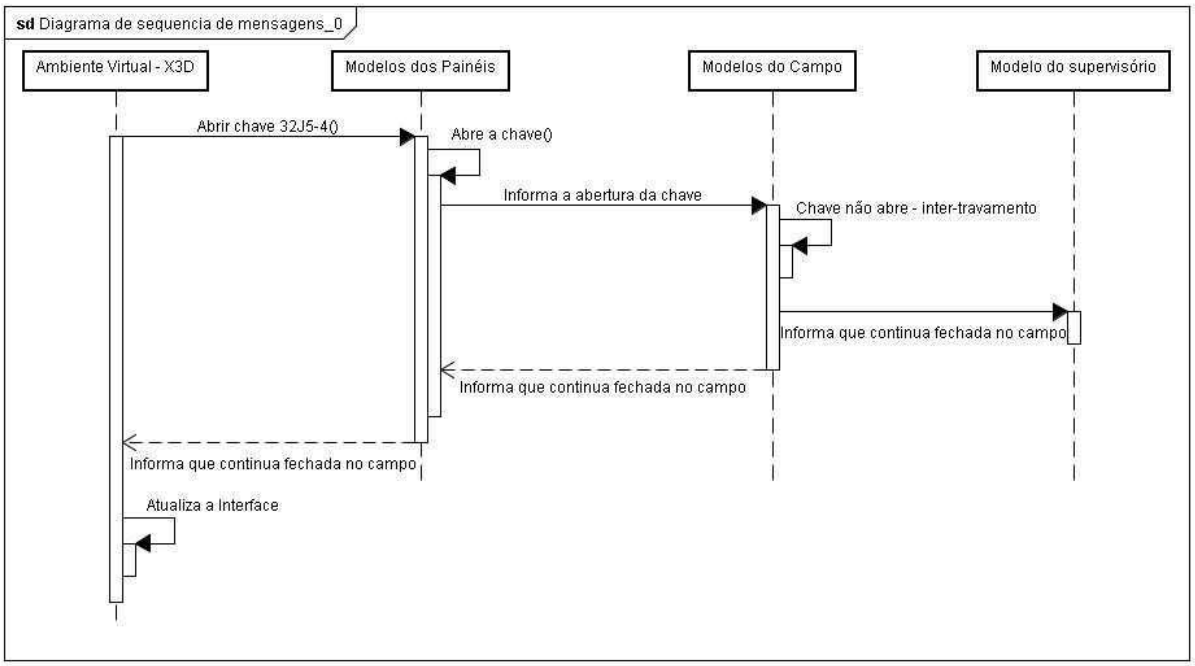

Figura 4-16 – MSC de fechamento da chave (sem sucesso)

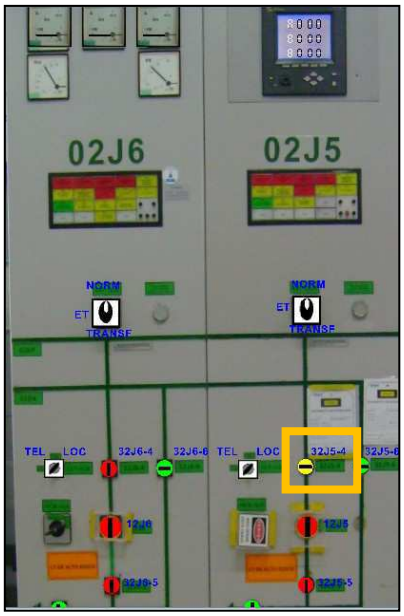

Figura 4-17 – Chave em discordância

4.3.2 Terceiro Estudo de Caso

O terceiro estudo de caso consiste em visualizar no medidor digital o valor atual da corrente no Painel (LT12J5). Existem duas maneiras de fazê-lo: a primeira consiste em utilizar o mouse e teclado, como nos demais casos e, a segunda maneira consiste em utilizar o menu de navegação, no qual é possível escolher uma vista direta do painel ou medidor.

Na Figura 4-18 é ilustrado o menu de navegação e a tela do medidor digital, que neste caso exibe a corrente de 12 A.

Figura 4-18 – Tela do simulador

Neste capítulo foi descrito o *driver* construído como parte deste trabalho assim como seu funcionamento, a partir de exemplos (casos). Para descrever a comunicação e o *driver*

foram utilizados diagramas UML, tais como diagrama de classes, diagrama de blocos, casos de uso e diagrama de Interação (MSC). Com o trabalho aqui apresentado o módulo básico da arquitetura foi completado uma vez que a comunicação entre módulos foi implementada e a interface do AV foi refinada em X3D. Outra contribuição foi a migração dos modelos CPN para o novo ambiente da ferramenta CPN Tools, possibilitando a continuidade do projeto com a abordagem do motor de simulação com base em modelos.

Capítulo 5 Considerações finais

Neste capítulo são apresentadas as conclusões relativas à arquitetura proposta e ao módulo desenvolvido para comunicação, bem como sugestões de trabalhos futuros.

5.1 Considerações finais

Com o desenvolvimento do simulador de ambiente de treinamento, o treinamento do usuário pode ser aprofundando na tentativa de minimizar os erros, e isto é importante para todo processos de automação industrial, pois em muitos casos a intervenção humana é menor, porém suas ações são decisivas e uma falha pode ocasionar grandes prejuízos tanto humanos, quanto materias.

A utilização de regras de concepção de interfaces com princípios ergonômicos leva o projetista ao desenvolvimento de interfaces menos susceptíveis a falhas de projeto. Através do simulador é possível identificar falhas que ainda não foram observadas na concepção, que só serão percebidas durante o uso e isto poderá ser tarde demais.

Através da proposição da arquitetura foi possível organizar o projeto do simulador para treinamento de operadores, a partir de um conjunto de requisitos que levam em conta tanto sua utilização no processo de treinamento, quanto seu uso acadêmico como ambiente de apoio ao estudo do comportamento do operador durante a execução de tarefas críticas. Devido a sua modularidade a arquitetura pode ser adaptada para outros contextos industriais, tais como plantas petroquímicas, sistemas de tratamento de fluídos, entre outros.

O requisito facilidade na configuração de novos contextos e cenários de treinamento (reuso de código e de modelos) foi levado em consideração quando da escolha da linguagem de representação virtual e na elaboração do *driver* de comunicação, facilitando a modularidade e a continuidade do trabalho por outros desenvolvedores. Foi também desenvolvida uma ferramenta para gerar automaticamente os modelos CPN, a partir de uma interface gráfica, onde o usuário monta e configura o painel de uma subestação e depois exporta para um modelo em CPN (NETTO et al, 2010).

A linguagem Java foi escolhida para o desenvolvimento no contexto deste trabalho por diversos fatores dentre os quais: o fato de utilizar a máquina virtual Java (JVM - Java Virtual Machine), pois isto facilita a portabilidade podendo ser executado independente de

plataforma, outro fator foi a facilidade de comunicar Java com o visualizador de Ambiente Virtual (já que também foi desenvolvido em Java), além de ser uma linguagem completa que possui um grande conjunto de API´s para apoiar o desenvolvimento.

Neste trabalho foi resgatado o funcionamento do motor de simulação com base em modelos formais para representar o comportamento dos objetos presentes no ambiente simulado - subestações elétricas. Para isto foi necessário adaptar a biblioteca Comms/CPN para fazer a interligação entre os modelos no ambiente CPN Tools e no AV. A relevância de manter o motor de simulação com base em modelos na nova arquitetura do simulador consiste na modularidade e flexibilidade decorrente, facilitando a adequação de funcionalidades já desenvolvidas e na adaptação para outros contextos.

O uso de modelos na descrição de ambientes industriais propicia representar a inter-relação entre os objetos, e atender a tempos de resposta prescritos e estritos, ao contrário de outras representações. Por exemplo, uma chave seccionadora tem associado um tempo de abertura que pode ser modelado na CPN com maior facilidade do que em uma linguagem de programação.

Para validação da arquitetura foram desenvolvidos três dos pacotes propostos para implementação do simulador, foram eles: construção dos modelos em CPN, na ferramenta CPN Tools, utilizando como base a biblioteca de modelos; a construção do Ambiente Virtual em X3D (aproveitando versões anteriores) e a comunicação entre os dois pacotes anteriores, como ilustrado na Figura 5-1.

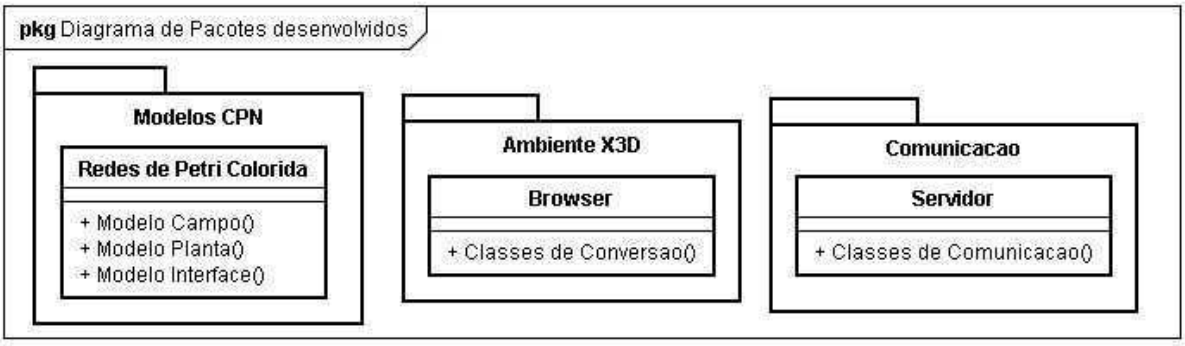

Figura 5-1 – Diagrama dos pacotes desenvolvidos.

No desenvolvimento deste trabalho foi necessário adaptar os modelos em CPN para, enviar e receber mensagens ao Ambiente Virtual. Uma das dificuldades no desenvolvimento deste trabalho foi a grande diferença na capacidade de representação dos visualizadores de ambientes virtuais, pois enquanto algumas funções e implementações funcionam corretamente em um visualizador, não são executáveis em outros. Outro problema é que a versão do CPN

Tools utilizada para o desenvolvimento das CPN possuía um problema na biblioteca, onde a função *accept* não funcionava, e o problema foi solucionado, fazendo o merge entre duas versões da ferramenta, pois tinha sido observado que na versão 2.0.0 a função funcionava corretamente.

Como apoio à validação da arquitetura outros trabalhos foram desenvolvido, dentre eles pode-se citar: o desenvolvimento do BD do simulador; e a animação dos medidores de grandezas (corrente, tensão) do Ambiente Virtual.

Por fim, possibilitou o desenvolvimento de um simulador que será utilizado no treinamento de operadores contribuindo para a prevenção do erro e diminuição de riscos de acidentes e incidentes industriais os quais podem levar a conseqüências materiais e ameaças à segurança dos que trabalham nestes ambientes.

5.2 Trabalhos futuros

Diante dos resultados obtidos com este trabalho surgem algumas limitações e com elas sugestões de trabalhos futuros:

- Refinar a classe "Browser" para dividir os métodos atuais em subclasses, facilitando o entendimento e a continuidade do trabalho;
- Avaliar as implicações da arquitetura sobre o desenvolvimento do módulo tutor do simulador;
- Refinar a ferramenta para configuração do motor de simulação que apóia o projetista na geração automática das Redes de Petri Coloridas;
- Avaliar o impacto da arquitetura sobre o projeto do módulo supervisório do simulador;
- Analisar o impacto da arquitetura aplicando em outro contexto.

Uma sugestão para o supervisório com a análise prévia realizada é elaborar o ambiente do supervisório dentro do Ambiente Virtual, assim como é feito em um supervisório comercial, dado que a linguagem X3D permite trabalhar com sobreposição de imagens, sugere-se que o módulo supervisório do Ambiente Virtual adote esta solução. No entanto também pode ser investigada a possibilidade de conectar um supervisório comercial, à interface em X3D. Neste caso seria necessário desenvolver um *driver* semelhante ao apresentado neste trabalho para comunicar o supervisório com o AV.

Com base nos requisitos, outro aspecto que não foi contemplado é o desenvolvimento do ambiente multiusuário. Uma proposição de solução é uso de Java RMI (Remote Method

Invocation) para integrar os ambientes virtuais (PERUZZA & ZUFFO, 2004). Pois, através do Java RMI é possível criar um ambiente Java distribuído, onde um método pode ser invocado remotamente por outra máquina virtual Java, a partir de diferentes lugares (JAVA RMI, 2010).

No requisito acesso remoto ao simulador, vale salientar que o acesso será feito diretamente pela web, através de uma *applet* que executará a aplicação através da máquina virtual Java. Foi escolhido applet para o desenvolvimento por ser um programa que pode ser executado dentro de um navegador Web, o código é armazenado na Web e pode ser baixado para o computador do cliente. Para implementar uma *applet* existem duas opções: carregar a *applet* diretamente na sua página HTML (*HiperText Markup Language*), este método é o mais antigo e apresenta problemas com restrição de segurança com os navegadores de Internet. Outro método é através de *Java Network Launch Protocol* (JNLP) atualmente é o mais usado por sua completude. Na SUN (2010) existe um tutorial de como importar para Web uma *applet* utilizando JNLP, o que pode ser utilizado na implementação.

Referências

3DML: **An Introduction to 3DML**: http://www.xml.com/pub/a/1999/01/3dml/3dml-plain.html Publicado em Janeiro 1999. Acesso em Outubro 2009.

3DMLW: **3D Markup Language for Web** - Open Source platform for creating 3D and 2D interactive web content: http://www.3dmlw.com/. Acesso em Outubro 2009.

3DS MAX, Autodesk. Ferramenta de edição gráfica 3D. Disponível em: http://usa.autodesk.com/adsk/servlet/pc/index?id=13567410&siteID=123112. Acesso em Novembro, 2009.

3DXML: **3D XML - a universal, lightweight XML-based format**. Disponível em: http://www.3ds.com/products/3dvia/3d-xml/overview/#vid1. Acesso em Outubro 2009.

AGENCIA USP, Noticia: **Simulador de trens reproduz ferrovia em Realidade Virtual**. Desenvolvido pela USP para treinamento de maquinistas da Vale. Disponível em: http://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=simulador-de-trens-reproduz-ferrovia-em-realidade-virtual, Acesso em Novembro 2009.

AGUIAR, Y. P. C.; NETTO, A. V. S.; SCHERER; QUEIROZ, M. F. V.. **Modelagem do Comportamento de Operadores de Subestação Durante a Realização da Tarefa**. In: IX Simpósio Brasileiro de Automação Inteligente, 2009, Brasília. Anais do IX Simpósio Brasileiro de Automação Inteligente, 2009.

AQUINO, M. S., **VEPersonal - Uma Infra-estrutura para Gerenciamento de Componentes de Ambientes Virtuais Adaptativos**, Tese de doutorado UFPE, 2007.

BEZERRA, H ; BARROSO, G. C. ; SAMPAIO, R. F. ; LEAO, R. P. S.; SOARES, J. M.. Sistema simulador para treinamento de proteção e operação de sistemas elétricos. In: ICECE'2007 - Conferência Internacional em Educação em Engenharia e Computação, Santos. Anais da Conferência Internacional em Educação em Engenharia e Computação, 2007.

BLENDER. Blender - Ferramenta gráfica de edição 3D. Disponível em: http://www.blender.org/download/get-blender/. Acesso em Novembro, 2009.

BOOCH, G; RUMBAUGH, J e JACOBSON, I: **UML, Guia do Usuário: tradução**; Fábio Freitas da Silva, Rio de Janeiro, Campus ,2000.

BOURAS, Ch. PANAGOPOULOS, A. TSIATSOS, Th. **Advances in X3D multi-user virtual environments**. In: Multimedia, Seventh IEEE International Symposium on, page: 7, ISBN: 0-7695-2489-3, 2005.

BRITNEY. BRITNeY - Suite Home, Experimental Test-bed for New Features for CPN Tools. http://wiki.daimi.au.dk/britney/_home.wiki. Acesso em Fevereiro de 2009;

BRUTZMAN, D.; DALY, L.; **X3D: Extensible 3D Graphics for Web Authors**, Ed. Morgan Kaufmann, Abril 2007.

BS CONTACT, Bitmanagement Software Contact. Visualizador de ambientes VRML/X3D. Disponível em: http://bitmanagement.com/download/#BS_Contact_VRML/X3D. Acesso em Novembro, 2009.

BS EDITOR. Bitmanagement Software Editor. Editor gráfico 3D. Disponível em: http://bitmanagement.com/download/#BS_Editor. Acesso em Novembro, 2009.

BS EXPORTER. Bitmanagement Software Exporter, exporta do formato proprietário para X3D. Disponível em: http://bitmanagement.com/download/#BS_Exporter_Blen der. Acesso em Novembro, 2009.

CARDOSO, A.; ZORZAL, E. R.; KIRNER, C.. **Realidade Virtual e Realidade Aumentada Aplicadas à Visualização da Informação**. In: Veronica Teichrieb, Fatima Nunes. (Org.). Realidade Virtual e Aumentada: minicursos SVR2008. 1 ed. Porto Alegre - RS: SBC, 2008, v. 1, p. 147-164.

CASTILHO, E. P. J. Trabalho de conclusão de curso de engenharia elétrica. Titulo: **Integração de Mundos Virtuais relativos ao Ambiente de Operação de uma Subestação Elétrica**, desenvolvido na UFCG, em Dezembro de 2007.

CPN TOOLS, Computer Tool for Colored Petri Nets. Disponível em: http://wiki.daimi.au.dk/cpntools. Acesso em Fevereiro de 2009.

DESIGN/CPN, disponível em: http://www.daimi.au.dk/designCPN/, Acesso em Dezembro de 2009.

ECLIPSE, Eclipse IDE for Java EE Developers. Disponível em: http://www.eclipse.org/. Acesso em Dezembro de 2009.

FREEWRL. Freewrl - Visualizador de VRML E X3D. Disponível em: http://freewrl.sourceforge.net/download.html. Acesso em dezembro de 2009.

FREITAS, R. C.; TURNELL, M. F. Q. V.; PERKUSICH, A. ; c . **Mecanismo para visualización y comunicación bidireccional entre modelos Redes de Petri coloreadas y modelos en realidad virtual**. In: CONFERÊNCIA INTERNACIONAL CONVENCIÓN FIE´06, Santiago de Cuba, 2006a.

FREITAS, R. C.; TURNELL, M. F. Q. V.; PERKUSICH, A.; **Representando a IHM de uma subestação através de modelos formais e Realidade Virtual**. In: Simpósio Brasileiro em Sistemas Elétricos, 2006, Campina Grande. Anais do Simpósio Brasileiro de Sistemas Elétricos, 2006b.

GARCIA, F. L. S.; TISSIANI, G.; PEDROSO, D. E. . **Metodologia para criação de ambientes virtuais tridimensionais**. In: IV International Conference on Graphics Engineering for Arts and Design, 2001, Sao Paulo. Anais do IV International Conference on Graphics Engineering for Arts and Design. São Paulo, 2001. v. 01. p. 156-160.

GRAVE, L.; ESCALEIRA, C.; SILVA, A. F., MARCOS, A. **A Realidade Virtual como Ferramenta de Treino para Montagem de. Cablagens Eléctricas**, 10º Encontro Português de Computação. Gráfica, ISCTE Lisboa, Outubro 2001. 1-3, pp. 147-156.

H3D. Disponível em: http://www.h3dapi.org/. Acesso em Novembro, 2009.

HAROLD, E. R., **XML 1.1 Bible**, 3rd Edition, ISBN: 0-7645-4986-3, 2004.

HERMOSILLA, L. G.; Santos, V.. **Estudo da estrutura óssea equina utilizando técnicas de Realidade Virtual**. In: SEPAVET - Semana da Patologia Veterinária, 2006, Garça. SEPAVET 2006. Garça: EDITORA FAEF, 2006.

HORSTMAN, C; **Big Java**, trad.: Furmankiewiecz, E. – Porto Alegre: Bookman, 2004.

IIS, Internet Information Service, Microsoft. Disponível em: http://www.iis.net. Acesso em Abril de 2010.

JADE. Java Agent DEvelopment Framework. Disponível em: http://jade.tilab.com/, Acesso em Novembro 2009.

JAVA RMI, Java Remote Method Invocation. Disponível em http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp. Acesso em Abril de 2010.

JAVA3D, API Java3D. Disponível em: https://java3d.dev.java.net/. Acesso em Outubro, 2009.

JDBC, Java Database Connectivity. Disponível em http://java.sun.com/javase/technologies/database/index.jsp. Acesso em Abril de 2010.

JENSEN K., **Colored Petri Nets – Basic concepts, analyses methods and practical use** – Vol 1. ed. Spring – Verlag, USA, 1992.

KAWAMOTO, A. L. S. ; CANTÃO, J. ; Pinto, A. ; KIRNER, T. G. ; KIRNER, C. ; WASLAWICK, R. S.. **Aspectos de Comunicação e Funcionalidade de um**

Ambiente Virtual Colaborativo. In: Simpósio Brasileiro de Redes de Computadores, 2001, Florianópolis. Anais do SBRC 2001, 2001.

KUROSE, J F.; ROSS, K. W**. Redes de Computadores e a Internet: Uma Nova Abordagem**. Ed.Pearson Education Brazil, ISBN:85-88639-10-1. 3ed. São Paulo, 2006

MACHADO, L. S.. **A Realidade Virtual em aplicações científicas**. Dissertação de Mestrado pelo INPE. INPE-6389-TDI/605, 1997

MACHADO, L. S.; ZUFFO, M. K. . **Desenvolvimento e Avaliação de um Simulador de Procedimentos Médicos Invasivos Baseado em Realidade Virtual para Treinamento de Transplante de Medula Óssea.** In: V Symposium on Virtual Reality, 2002, Fortaleza. Proceedings of SVR 2002, 2002. p. 416-418.

MARSOLI, A. C.; PINHAT, C. M. V.. **Aplicação da Realidade Virtual no Ensino Aprendizagem de Conceitos de Matemática**. In: 14º Congresso de Iniciação Científica no contexto da 4ª Mostra Acadêmica, 2006, Piracicaba. Aplicação da Realidade Virtual no Ensino Aprendizagem de Conceitos de Matemática, 2006.

MARTINS, F. E.; BATTAIOLA, A. L.; MORONI, L. M.. **Design do Jogo Educacional EEHouse**. In: SBGames, 2009, Rio de Janeiro. Proceedings, 2009.

MURATA, T. Petri **Nets: Properties, Analysis and Applications.** Proceedings of the IEEE, nº77(4), April, 1989. p. 541-580.

MYSQL, Banco de dados MySQL. Disponível em: http://www.mysql.com. Acesso em Abril de 2010.

NASCIMENTO, J. A. N.; TURNELL, M. F. Q. V.; SANTONI, C.. **Modelagem da Rotina Operacional de uma Subestação elétrica em HCPN**. In: VIII SBAI - Simpósio Brasileiro de Automação inteligente, 2007, Florianópolis. VIII SBAI, 2007.

NETTO, A. V. S.; NASCIMENTO, J. A.; VIEIRA, M. F. Q.. **Realidade Virtual aplicada a instalações elétricas.** In: IX Simpósio Brasileiro de Automação Inteligente, 2009, Brasília. Anais do IX Simpósio Brasileiro de Automação Inteligente, 2009.

NETTO, A. V. S.; SCHERER, D.; NASCIMENTO, J. A.; SANTOS, A. M. M.; ANDRADE, J. W.; ALMEIDA, R. P.; VIEIRA, M. F. W.. **Ferramenta para construção de cenários de treinamento em Redes de Petri Coloridas**, In: XVIII Congresso Brasileiro de Automática (CBA, 2010), 2010.

OCTAGA, player. Visualizador de X3D. Disponível em: http://www.octaga.com. Acesso em Novembro, 2009.

OPENVRML. Visualizador de X3D. Disponível em: http://openvrml.org/. Acesso em Novembro, 2009.

PERUZZA, A. ; ZUFFO, M. K. . **ConstruiRV: construting knowledge using the virtual reality**. In: acm siggraph international conference on virtual reality continuum & its applications in industry, 2004, Singapore. Proceedings, 2004.

POSTGRESQL: Postgresql, disponível em: http://www.postgresql.org/, Acesso em Novembro 2009.

VIEIRA, M. F. Q. ; NASCIMENTO NETO, José Alves Do ; SCAICO, A. ; SANTONI, Charles ; MERCANTINI, Jean Marc . **A Model Based Operator Training Simulator to support Human Behavior Studies**. Transactions of the Society for Computer Simulation, v. 86, p. 41-51, 2010.

SCAICO, A.; TURNELL, M. F. Q. V.; PERKUSICH, A. **Modelagem da Navegação de Interfaces com o Usuário de Sistemas de Automação Industrial**. In: V Simpósio Brasileiro de Automação Inteligente, 2001, Canela - RS. Anais do V Simpósio Brasileiro de Automação Inteligente, 2001.

SILVA, J. E.; ROGADO, J.. **A Realidade Virtual no Ensino de Química: O Caso da Estrutura Atômica**. In: XIV ENEQ - Encontro Nacional de Ensino de Química, 2008, Curitiba. Anais do XIV Encontro Nacional de Ensino de Química, 2008.

SILVA, L. F.; CARDOSO, A.; MENDES, E. B; LAMOUNIER, E.. **Associando Ferramentas Cognitivas e Realidade Virtual não-Imersiva para o Ensino de Física**. In: IADIS Conferencia Ibero Americana WWW/Internet 2005, Lisboa. 2005.

SILVA, V. N. A. L. da; LINDEN, R.; RIBEIRO, G. F.; PEREIRA, M. de F. L.; LANNES, R. S.; STANDKE, C. R. **Simuladores para treinamento de operadores de sistema e de instalações do setor elétrico**. XII ERIAC – Encontro Regional Iberoamericano de Cigré, Puerto Iguazú, Argentina, 2009.

SOUZA, D. F. L; VALDEK, M. C. O. ; MORAES, R. M.; MACHADO, L.. **SITEG Sistema Interativo de Treinamento em Exame Ginecológico**. In: Symposium on Virtual Reality, 2006, Belém. Proc. of Symposium on Virtual Reality, 2006. p. 445-456.

SOUZA, M. R. de; **Avaliação Interativa da Especificação de Interfaces com Ênfase na Navegação**. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal da Paraíba, 1999.

STUDIO PLAYER, Vivaty. Visualizador de X3D Disponível em: http://mediamachines.com. Acesso em Novembro, 2009.

STUDIO, Vivaty. Editor gráfico 3D. Disponível em: http://mediamachines.com/. Acesso em Novembro, 2009.

SWIRLX3D. Visualizador de X3D. Disponível em: http://www.pinecoast.com/swirl3d.htm. Acesso em Outubro, 2009.

TORRES, F. F..; VIEIRA, M. F. W.. **Motor de simulação baseado em modelos CPN aplicado a um sistema para treinamento de operadores**, In: XVIII Congresso Brasileiro de Automática (CBA, 2010), 2010.

TURNELL, M. F. Q. V.; SCAICO, A.; SOUZA, M. R. de; PERKUSICH, A. . **Industrial User Interface Evaluation Based on Colored Petri Nets Modelling and Analysis**. Lecture Notes in Computer Science, v. 2220, 2001, p. 69-87.

UBUNTU, Ubuntu Server. Disponível em: http://www.ubuntulinux.org/server. Acesso em Novembro, 2009.

VEH, A. O.; MARCEAU, R. J.;MALOWANY, A.; DESBIENS, P.; DAIGLE, A.; GARANT, E.; GAUTHIER, R.; SHAIKH, A.; RIZZI, J. C.. **Design and Operation of a Virtual Reality Operator-Traninig System**. IEEE Transaction on Powe Systems, Vol 11, No 3, Agosto 1996.

VUG, Visualization and Usability Group. VRML97 to X3D Translation. Disponível em http://ovrt.nist.gov/v2_x3d.html. Acesso em Outubro, 2009.

WEB3D: Web3D Consortium, **Open Standards for Real-Time 3D Communication**. http://www.web3d.org/. Acesso em Novembro, 2009.

X3D-EDIT. Disponível em: https://savage.nps.edu/X3D-Edit/. Acesso em Novembro, 2009.

XJ3D. Disponível em: http://www.web3d.org/x3d/xj3d/. Acesso em Novembro, 2009.

XVRML, **the Extensible Virtual Reality Modelling Language**. Disponível em: http://xvrml.net/#. Acesso em Outubro, 2009.

SUN, **Java Sun tutorial para desenvolvimento de applets**. Disponível em: http://java.sun.com/docs/books/tutorial/deployment/applet/deployingApplet.html. Acesso em Abril, 2010.

Glossário

Applet consiste em um (pequeno) aplicativo cliente, escrito na linguagem de programação Java, que executa na máquina virtual instalada no navegador Web. Todavia, sistemas clientes provavelmente irão precisar do Java *Plug-in* e possivelmente de um arquivo de política de segurança; de modo que a *applet* seja executada com sucesso no navegador de Internet. (HORSTMAN, 2004).

EAI (*External Authoring Interface*) é um método de comunicação que permite a execução de um controle externo da interface em Realidade Virtual. Por exemplo, permite uma applet Java controlar um cenário em RV embarcado em uma página HTML. (BRUTZMAN & DALY, 2007)

JavaScript é uma linguagem interpretada voltada para o desenvolvimento de *scripts* (pequenos programas). Os *scripts* podem ser associados ao código HTML (*HyperText Markup Language*) de uma página Web, proporcionando uma forma acessível para aumentar a interatividade ou funcionalidade; ou ainda serem associados a visualizadores de X3D permitindo alterações dinâmicas dos objetos do mundo virtual modelado. Na associação com X3D é possível utilizar o *JavaScript* para modelar o comportamentos dos objetos, tornando as animações mais complexas (HORSTMAN, 2004).

JSI (*Java Script Interface*): Consiste em um método da linguagem Java para comunicar Freewrl com CPN. (HORSTMAN, 2004)

Mapa conceitual: Ferramentas cognitivas que apresentam uma estrutura formada por um sistema de redes conceituais, que se organizam e se distribuem por diferenciações progressivas e reconciliações integrativas, em um modelo que se relaciona por ligações semânticas (SILVA et al, 2005).

Motor de simulação: neste trabalho, se refere a um conjunto de modelos descritos em redes de Petri Coloridas (CPN) que uma vez executados representam o comportamento dos objetos do ambiente simulado, no caso o ambiente de supervisão de uma subestação elétrica.

Script: Consiste em um roteiro em linhas de código para descrever as funcionalidades dos objetos (HORSTMAN, 2004)

Servidor Web (Web Server) é o conjunto de hardware e software necessário para hospedar uma página na Internet ou fornecer ao computador cliente informações de dados ou serviços necessários, em tempo real. O servidor web é responsável por aceitar pedidos,

através de um navegador, via HTTP (Hypertext Transfer Protocol) para uma determinada aplicação (*applet*, páginas pessoais). Por exemplo, aceitar as requisições e enviar uma resposta para o cliente sobre o acesso a uma página Web. (HORSTMAN, 2004).

Socket: Objeto que encapsula uma conexão TCP/IP. Para se comunicar com a outra ponta da conexão, são usados fluxos de entrada e saída anexados ao sockets (Horstman, 2004).

XML (*eXtensible Markup Language*): linguagem de marcação de dados (meta-markup language) que provê um formato para descrever dados estruturados, formados por um conjunto de regras, definidas por *tags* (marcações), os quais permitem que as especificações da linguagem sejam interpretadas por vários sistemas. O XML é um padrão flexível, aberto, independente de dispositivo que permite múltiplas formas de visualização dos dados estruturados (HAROLD, 2004).

TCP/IP é um acrônimo para o termo *Transmission Control Protocol* / *Internet Protocol Suite*, ou seja, é um conjunto de protocolos, que reúne dois dos mais importantes (o IP e o TCP) os quais deram seus nomes à arquitetura. O protocolo IP, base da estrutura de comunicação da Internet é um protocolo baseado no paradigma de chaveamento de pacotes (*packet-switching*) (KUROSE & ROSS, 2006).

Anexo A: Tecnologias para construção de Ambientes Virtuais

Este trabalho iniciou com um levantamento dos recursos disponíveis à construção de mundos virtuais. Neste anexo são apresentadas as linguagens estudadas e os critérios utilizados para escolha da linguagem a ser adotada no projeto do simulador. Outro aspecto também estudado, e tratado neste anexo foi o conjunto de ferramentas de software necessário à construção e visualização do Ambiente Virtual, considerando que se trata de um Ambiente Virtual não-imersivo.

A.1 Linguagens

Dentre as linguagens citadas na literatura para representar Realidade Virtual destacam-se:

VRML (*Virtual Reality Modeling Language*). Esta foi uma das primeiras linguagens para modelagem tridimensional voltada para Web. VRML surgiu na década de 90 e consiste em uma linguagem de descrição para modelar objetos e ambientes virtuais tridimensionais. É uma linguagem livre (não-proprietária) e de domínio público (WEB3D, 2009).

A linguagem VRML é capaz de representar ambientes estáticos e dinâmicos, além de objetos multimídias com hiperlinks para textos, sons, vídeos e imagens. Porém para a utilização de todos os recursos propostos por esta linguagem é necessária uma ferramenta de visualização que ofereça suporte aos recursos existentes na linguagem. Por exemplo, a representação do comportamento dos objetos no ambiente é feita através da inserção de um *script* e o visualizador escolhido deve ser capaz de executar esse *script* durante a visualização do ambiente para que os objetos sejam representados adequadamente.

A versão analisada do VRML foi à versão 2.0. Sua última atualização foi em 1997, com o acréscimo do padrão ISO (*International Organization for Standardization*) para uso de script. O VRML ainda continua sendo usado, porém não possui mais suporte, pois o consortium responsável pelo seu desenvolvimento - o Web3D, o substituiu com o lançamento do X3D.

X3D (Extensible 3D): é um padrão 3D, internacional e aberto, para Web (X3D, 2007). O X3D é usado na construção de modelos tridimensionais, tanto simples quanto

complexos. O X3D pode mostrar objetos animados de diferentes pontos de vistas, permitindo a imersão do usuário no ambiente e a interação com os objetos do ambiente (BRUTZMAN & DALY, 2007). O X3D é representado no formato XML, permite manipular objetos em tempo real, e facilita o reuso de código VRML, por ser uma evolução deste. O X3D possui as seguintes características:

- **XML Integrado**: Esse é um dos grandes diferencias em relação a outros formalismos, pois pode ser integrado com:
  - Serviços Web (Web Services);
  - Banco de dados;
  - Além de facilitar a transferência de dados.
- É extensível: permite adicionar e estender as funcionalidades da descrição, para aplicações específicas.
- É de fácil adaptação: é possível reaproveitar códigos descritos em VRML.
- Adequado para aplicações embarcadas: já é possível ser utilizados em celulares, tablets e PDA's
- Executa em Tempo real: a representação gráfica é de boa qualidade. Com interação em tempo real e é possível incluir som, vídeos e dados 3D.
- Bem especificado: por ser um padrão ISO, o código é de fácil construção e consistente.

Java 3D: desenvolvida pela Sun, consiste em uma biblioteca orientada a objetos, da linguagem Java, que permite a construção de aplicações que contenham cenários tridimensionais. Esta biblioteca é independente de um navegador, porém necessita de uma *applet* para ser executada dentro de um navegador. (JAVA3D, 2009).

Java 3D oferece suporte para exibição de imagens 3D, áudios e vídeos. Seu nível de desenvolvimento se aproxima de VRML, a ponto dos ambientes definidos em VRML poderem ser utilizados pela biblioteca Java 3D. Esta biblioteca baseia-se em *scene graph* que são coleções de objetos tridimensionais organizados em uma estrutura de árvore. Estes objetos são manipulados pelo desenvolvedor através das construções da linguagem Java.

Uma desvantagem de Java 3D é que o desenvolvedor deve definir os modos de navegação dentro do ambiente, pois Java 3D não utiliza *plug-ins* para serem visualizados. Para a execução de Java 3D é necessário ter instalado na máquina do usuário o JVM (*Java Virtual Machine*), que é a máquina virtual necessária para interpretar os comandos de aplicações desenvolvidas com a linguagem Java.

3DML (3D Markup Language): Foi desenvolvida pela Flatland (3DML, 2009).. É uma linguagem de marcação tridimensional similar a HTML, que foi utilizada na construção de páginas Web mas que está em desuso. Sua última atualização foi em 2005.

xVRML: Baseada na linguagem VRML foi desenvolvida pelo CMEG (*Computer Mediated Experience Group*). O grupo tentou adaptar o VRML para o formato XML, porém não foi dada sequência a esta iniciativa. Sua última atualização foi em 2003 e atualmente está em desuso. (XVRML, 2009)

3DXML: é um formato de arquivo proprietário 3D que foi desenvolvido por Dassault Systemes(3DXML, 2009). É escrito no formato XML e tem um formato proprietário, demandando uma ferramenta exclusiva para seu uso. Não é bem documentada e o seu último release é datado de 2006.

3DMLW: (3D Markup Language for Web): é um formato baseado em XML para representar ambientes tridimensionais e bidimensionais, via Internet. O 3DMLW é um código livre e foi desenvolvido por 3D Technologies R&D(3DMLW, 2004). Possui suporte a script, suporta modelos advindos de outras ferramentas como *Blender* e 3D Max, porém possui apenas um editor. Foram encontradas poucas aplicações e sua documentação é fraca, além dos desenvolvedores serem pouco conhecidos.

Critérios para escolha da linguagem

Para a escolha da linguagem que será utilizada na representação dos ambientes em Realidade Virtual foram aditados os seguintes critérios:

- Aceitação e atualidade: de boa aceitação por usuários e que ainda esteja em uso.
- Potencial de representação: visual (capacidade de reprodução dos objetos a serem representados no simulador), áudio (capacidade de inserir áudio do ambiente a ser representado), vídeo (capacidade de reproduzir vídeos no AV);
- Possibilidade de comunicação com processos externos: oferecer suporte para trocar informações com outras linguagens de programação, modelos formais (ex.: Redes de Petri Coloridas), e bancos de dados;
- Possibilidade de visualização remota: para facilitar o acesso ao AV sem precisar armazenar todos os dados necessários no cliente;
- Acesso a Múltiplos usuários: permitir que mais de um usuário, interaja em um mesmo mundo virtual;
- Disponibilidade de boa documentação/tutoriais/suporte: para facilitar o seu uso;
- Disponibilidade de ferramentas para edição da linguagem;

- Disponibilidade de ferramentas para visualização;
- Apoiada por um Grupo desenvolvedor conhecido e possuir um padrão consolidado;
- Oferecer suporte a diversos dispositivos de entrada e saída: mouse, joystick, teclado, entre outros;
- Distribuição (proprietário): que seja uma linguagem livre e caso seja proprietária que possua distribuição livre (gratuita).
- Formato XML.

Análise comparativa das linguagens

Após pesquisa na literatura foi constatado que o 3DML e xVRML estão em desuso. Já a linguagem 3DXML foi descartada, por ser uma linguagem proprietária e, o 3DMLW possui apenas uma ferramenta de edição enquanto sua documentação é fraca, não permitindo um estudo mais aprofundado. Por isso outros critérios destas linguagens não foram analisados.

As linguagens Java3D e VRML poderiam ser utilizadas não fosse o fato da descrição dos objetos não seguir o formato XML, acarretando algumas desvantagens na portabilidade, no armazenamento em banco de dados e na taxa de transferência dos dados. O VRML ainda possui a desvantagem de ter sido substituído pelo X3D.

A Tabela A-1 apresenta um quadro comparativo das linguagens, no qual o símbolo ✓ indica que a linguagem atende o critério, vazio (em branco) indica que não atende e, * indica que o critério não foi pesquisado dado que a linguagem já não atendia outros requisitos.

Tabela A-1 – Quadro comparativo das linguagens

	VRML	X3D	Java3D	3DML	3DXML	xVRML	3DMLW
Versão/Ano	2.0/1997	3.0/2009	1.4/2007	2005	2006	2003	2009
Aceitação	✓	✓	✓				
Capacidade de representação	✓	✓	✓	*	*	*	*
Documentação/tutoriais	✓	✓	✓				✓
Ferramentas de edição	✓	✓	✓	*	*	*	✓
Ferramentas de visualização	✓	✓	✓	*	*	*	✓
Não proprietária	✓	✓	✓	✓		✓	✓
Formato em XML		✓			✓	✓	
Autor/Grupo	Web3D	Web3D	Sun	Flatland	Dassault	CMEG	R&D

Legenda:	✓	Atende o critério.		Não atende o critério.	*	Não foi pesquisado.

A partir do estudo comparativo sumariado na Tabela A-1, a linguagem escolhida para representar a RV do simulador foi X3D, que atende a todos os requisitos propostos. Após a escolha da linguagem o próximo passo consistiu em investigar as ferramentas necessárias para a concepção e visualização do Ambiente Virtual projetado.

A.2 Ferramentas

Nesta seção serão apresentadas as ferramentas pesquisadas na literatura voltadas para a concepção e visualização de mundos virtuais. Apesar desta pesquisa conclui-se que para trabalhar com X3D é possível utilizar apenas o bloco de notas, ficando a escolha da ferramenta a critério do desenvolvedor.

Requisitos para ferramentas de edição:

- ✓ Disponibilidade de tutoriais para facilitar o uso;
- ✓ Geração de um código XML consistente, o qual o visualizador possa interpretar;
- ✓ Recursos para visualização tanto do código textual quanto de imagens (com geração de código automática);
- ✓ Suporte à versão da linguagem selecionada para o projeto (versão X3D 3.0);
- ✓ Recursos para Edição gráfica: a ferramenta deve possuir um modo de edição gráfico, que facilite o posicionamento das imagens e a concepção da representação dos objetos;
- ✓ Versão *freeware*: deve disponibilizar versão grátis.

Requisitos para a ferramenta de visualização:

- ✓ Capacidade de representação: a ferramenta deve ser compatível com os recursos previstos pela linguagem de descrição, como a visualização de objetos 2D e 3D, inserção de áudio, textura, vídeos, entre outros;
- ✓ Comunicação com processos externos: a ferramenta deve oferecer suporte ao uso de *script* (EAI), ou linguagens de programação, para permitir a comunicação com outros software*s* (por exemplo, via TCP/IP);
- ✓ Deve permitir diferentes modos de navegação no AV;
- ✓ Deve permitir visualização remota, dispensando o armazenamento local do AV;
- ✓ Deve permitir o acesso a múltiplos usuários;

- ✓ Deve disponibilizar tutoriais para facilitar o seu uso;
- ✓ Distribuição: as ferramentas devem ser *freeware* (licença grátis/aberto);
- ✓ Deve oferecer suporte aos dispositivos de entrada e saída (como mouse, teclado, joystick);
- ✓ Deve ser compatível com a linguagem escolhida (X3D);
- ✓ Deve apoiar a execução em Multi-plataforma (ao menos em Windows e Linux).

Ferramentas de edição

- **X3D-Edit**: Ferramenta fornecida pelo desenvolvedor do padrão X3D. (X3D-EDIT, 2009).
- **Vivaty Studio 2.1**: compatível com o ambiente Windows é uma ferramenta de modelagem e animação para ambiente Web em tempo real, que utiliza X3D. (STUDIO, 2009).
- **BS Editor**: é um software para edição de ambientes 2D/3D, oferece suporte para áudio e vídeo, aceita os padrões VRML e X3D, porém não oferece versão gratuita. Foi desenvolvido pela empresa alemã Bitmanagement Software GmbH e é uma evolução da ferramenta Blaxxun. (BS EDITOR, 2009)
- **Blender**: (também conhecido como blender3d): é um editor de código aberto, desenvolvido pela *Blender Foundation*, para modelagem, animação, texturização, composição, "renderização", edição de vídeo e criação de aplicações interativas em 3D. Está ferramenta oferece suporte para importar e exportar arquivos X3D. (BLENDER, 2009)
- **Autodesk 3ds Max**: Ferramenta robusta para edição de imagens em 3D, porém não é *freeware* e só oferece uma versão para teste por 30 dias. (3DS MAX, 2009)
- **Autodesk Maya**: Oferece uma versão grátis, porém atualmente os direitos de uso são da Autodesk, que oferece uma ferramenta robusta para edição de imagens em 3D, mas não é *freeware* (versão de 30 dias para teste). (MAYA, 2009)
- **H3D**: API desenvolvida em C++, *freeware*, para concepção de RV em X3D, oferece suporte a Python e C++. (H3D, 2009)

Tabela A-2 – Quadro Comparativo das ferramentas de edição

	X3D-Edit	Vivaty	BS editor	Blender	3DS Max	Maya	H3D

Tutoriais	✓	✓	✓	✓	✓	✓	✓
Gerar XML consistente	✓		✓	✓			
Múltiplos Modos	✓	✓	✓				
Suporte a Linguagem X3D	✓	✓	✓	✓	✓	✓	✓
Edição Gráfica		✓	✓	✓	✓	✓	✓
Versão grátis	✓	✓		✓			✓

A partir do estudo comparativo das ferramentas de edição, exibido na Tabela A-2, a ferramenta recomendada é X3D-Edit, porém dado que o mundo virtual X3D pode ser desenvolvido utilizando até mesmo em um bloco de notas, cabe ao desenvolvedor escolher a ferramenta de acordo com seu nível de conhecimento em programação e em ferramentas de edição de imagens.

Ferramentas para visualização

- **Xj3D** foi idealizada pela Web3D Consortium e, consiste em uma ferramenta para criar mundos virtuais em VRML e X3D. O Xj3D possui um código aberto e foi desenvolvido em Java. Esta ferramenta foi desenvolvida no intuito de estimular o uso da tecnologia X3D. (XJ3D, 2009).
- **Vivaty Studio Player 2.1**: trata-se de uma evolução da ferramenta Flux e foi desenvolvida pela empresa Vivaty. É compatível com X3D e é uma versão livre, para uso acadêmico e pessoal. Oferece suporte para: mp3, wmv, avi, mpg vídeo (STUDIO PLAYER, 2009).
- **Octaga player**: Desenvolvida pela Octaga, consiste em um visualizador de 3D, em tempo real, cuja representação gráfica é de boa definição, porém o pacote completo do software é pago (OCTAGA, 2009).
- **FreeWRL**: É uma ferramenta grátis, para visualização de VRML e X3D. Na primeira versão elaborada do simulador foi utilizada está ferramenta, porém ela oferece uma versão apenas para Linux (FREEWRL, 2009).
- **OpenVRML**: É uma ferramenta grátis, para visualização de VRML e X3D (OPENVRML, 2009).
- **BS Contact VRML/X3D**: é um software para visualização de ambientes 2D/3D, oferece suporte para áudio e vídeo, e aceita os padrões VRML e X3D. Oferece uma versão para

teste, e foi desenvolvido pela empresa *Bitmanagement Software*. Trata-se da evolução da ferramenta *Blaxxun* (BS CONTACT, 2009).

- **SwirlX3D:** é um software para visualização de VRML e X3D, desenvolvido pela empresa *Pinecoast Software*, que está disponível para download na versão de teste, válida por sete dias (SWIRLX3D, 2009).

Tabela A-3 – Quadro comparativo de software visualizador

	Xj3D	Freewrl	OpenVRML	Vivaty	SwirlX3D	Octaga	BS Contact
Representação (completude)	✓	✓		✓	✓	✓	✓
Comunicação externa	✓		✓	✓		✓	✓
Visualização remota	✓	✓	✓	✓	✓	✓	✓
Navegação	✓	✓	✓	✓	✓	✓	✓
Múltiplos usuários	✓	✓		✓	✓	✓	✓
Tutoriais	✓	✓		✓		✓	✓
Grátis/Livre	✓	✓	✓				
Compatibilidade X3D	✓	✓		✓	✓	✓	✓
Dispositivos E/S	✓	✓	✓	✓	✓	✓	✓
Multi-Plataforma	✓	✓				✓	

Dentre os visualizadores analisados, e apresentados na Tabela A-3, o que melhor se adéqua aos critérios propostos pelo projeto do simulador é o Xj3D. A versão escolhida foi a versão 2.0, a qual pode ser encontrada no site: http://www.xj3d.org/snapshots.html. A maior vantagem deste visualizador para o projeto do simulador é a facilidade de integrá-lo com aplicações em Java, apesar de apresentar limitações na reprodução de áudio.

Ferramentas para conversão para X3D

Considerando que já existia uma versão do simulador escrita em VRML, uma vez que foi decidido migrar para X3D, foi necessário buscar meios de converter de forma automática o código existente. Seguem as ferramentas encontradas para este fim:

- **BS Exporter**: desenvolvido pela empresa *Bitmanagement Software GmbH* . De acordo com o fabricante, através dessa ferramenta é possível exportar das ferramentas 3dsmax e Blender para X3D (BS EXPORTER, 2009).
- **Vrml97toX3DNist**: conversor de VRML para X3D, desenvolvido pelo grupo Web3D. A versão disponível foi a versão 1.0, desenvolvida pelo *National Institute of Standards and Technology* (VUG, 2009).

Anexo B: Biblioteca Comms/CPN

Este anexo apresenta a biblioteca Comms/CPN (, 2002b). Esta biblioteca é utilizada para fazer a comunicação dos modelos representados em CPN (JENSEN, 1992) com processos externos (Java e X3D). Na antiga ferramenta Design/CPN (Design/CPN, 2009) esta biblioteca consistia em um conjunto de códigos em SML (*Standard Markup Language*), os quais eram associados às redes construídas na ferramenta visando comunicá-las. Na ferramenta CPNTools (CPN TOOLS, 2009), as funções de comunicação desta biblioteca foram embutidas na própria ferramenta.

B.1 Estrutura da Biblioteca Comms/CPN

A biblioteca Comms/CPN foi projetada para atuar como interface entre os modelos construídos em Redes de Petri Coloridas utilizando o protocolo TCP/IP. A Comms/CPN é composta por três módulos principais, organizados em:

i) **Camada de comunicação:** contém os mecanismos de transporte básicos do protocolo TCP/IP e todas as funções primitivas fundamentais relacionadas à *sockets*;

ii) **Camada de mensagens**: é responsável pela conversão dos dados em fluxo de bytes para que haja a troca de mensagens entre CPN Tools e as aplicações externas;

iii) **Camada de gerenciamento de conexão**: permite que processos externos: abram, fechem, enviem e recebam múltiplas conexões. Esta camada faz a interface direta com os modelos CPN.

O protocolo utilizado pela biblioteca Comms/CPN é o TCP/IP, dado que: (1) as transmissões de dados devem ser confiáveis (garantia de envio e recebimento) e, ordenadas; (2) este é um protocolo padrão, para o qual a maioria dos dispositivos implementa suas funcionalidades e grande parte dos ambientes de programação provê interfaces para seus *sockets*. A biblioteca Comms/CPN contém as funções (*Connection Management Functions*), apresentadas na Tabela B.1-1.

Tabela B.1-1 – Primitivas da camada de conexões.

Primitiva	Responsabilidade
openConnection (“NomeConexao”,”IP”, “portaComunicacao”’)	Abre a comunicação com um servidor (outra rede ou processo externo), devendo para tal: indicar o nome da conexão que será estabelecida, o IP da máquina com a qual deseja conectar e a porta de comunicação.
acceptConnection (“NomeConexao”, “portaComunicacao”)	Ao iniciar fica aguardando uma conexão, através da porta definida. A conexão é identificada por seu nome.
canReceive(“NomeConexao”)	Testa se existe alguma mensagem para ser recebida, evitando assim que a rede trave caso não haja mensagens para serem recebidas. Essa função não foi implementada no Design/CPN.
send()	Permite o envio de qualquer tipo de dados a processos externos. Converte a ficha em um *String* e depois codifica os dados para envio em uma seqüência de bytes.
receive()	Permite o recebimento de qualquer tipo de dados, vindos de processos externos. Codificando os dados recebidos em um *String*.
closeConnection	Permite o fechamento de uma conexão. A *string* de conexão é passada como parâmetro para esta função. Uma busca é feita para garantir que existe uma conexão com o nome da *string* passada. Se houver, a conexão é encerrada e suas informações armazenadas são removidas da lista de

	conexões.

B.2 Codificação Java

Esta sessão apresenta os arquivos Java necessários para comunicar a biblioteca com os processos externos (Xj3D). Na figura B-1 é apresentado o diagrama de classes das funções em Java necessárias para comunicar as Redes de Petri Coloridas com o ambiente do Mundo virtual.

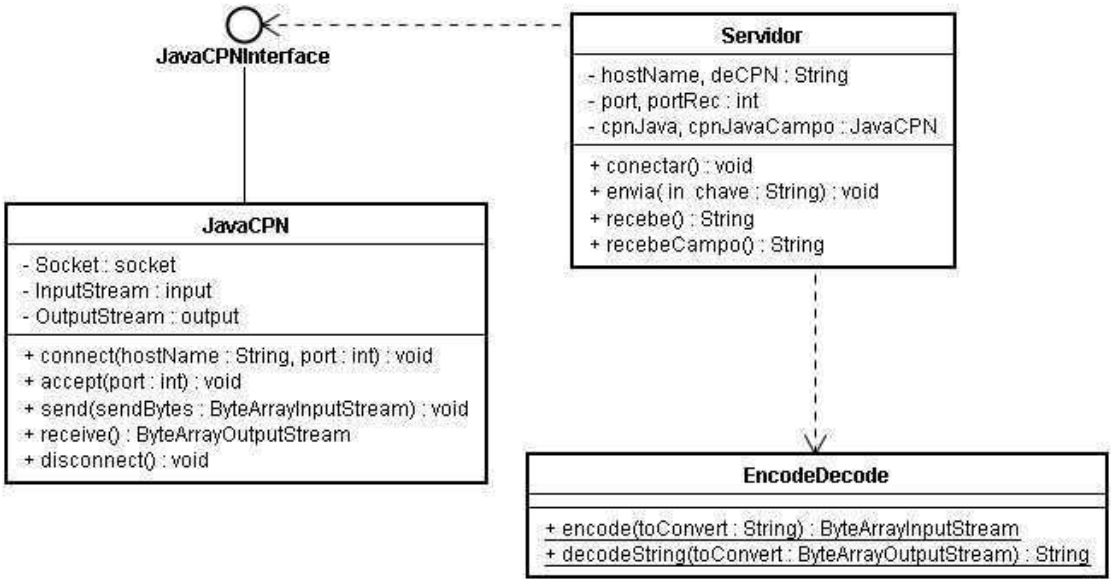

Figura B.2-1 – Diagrama de classes/objetos da camada de comunicação.

As classes JavaCPN, JavaCPNInterface e EncodeDecode são classes fornecidas pelos desenvolvedores do CPN Tools. Por outro lado, a classe Servidor foi desenvolvida neste trabalho para adequar as necessidades da aplicação, de comunicar o Ambiente Virtual com o Modelo que representa o funcionamento da Interface. A seguir serão apresentadas as descrições dessas classes:

i) ***JavaCPNInterface.class***

Esta classe é uma abstração da camada de gerenciamento de conexões da biblioteca Comms/CPN. Define a interface para as primitivas: *connect, accept, send, receive e disconnect*.

ii) ***JavaCPN.class***

Esta classe implementa as primitivas: *connect, accept, send, receive e disconnect*, definidas na classe JavaCPNInterface. A seguir, são apresentadas suas funcionalidades:

- **connect**
  1. Criar *socket* para o estabelecimento da conexão, passando como parâmetros: o *string* de conexão e a porta;

- **accept**
  1. Definir canal para recepção de dados;
  2. Definir canal para envio de dados;

- **send**
  1. Definir vetor de *bytes*, referente ao pacote de dados;
  2. Enquanto o tamanho do pacote for maior que 127 bytes
     2.1 Criar pacote;
     2.2 Ler a seqüência de 127 *bytes* para envio;
     2.3 Se não houve nenhum erro, então,
        2.3.1 Enviar pacote ao processo externo.
  3. Finalizar estrutura de repetição.
  4. Criar pacote para dados remanescentes;
  5. Ler a seqüência de dados remanescentes (menor que 127);
  6. Se não houve nenhum erro, então,
     6.1 Enviar pacote ao processo externo.

- **receive**
  1. Definir variável para armazenar a quantidade total de *bytes* recebida;
  2. Definir variável para armazenar o tamanho de cada pacote (cabeçalho);
  3. Definir variável para armazenar cada pacote do fluxo de *bytes*;
  4. Enquanto houver recebimento de fluxo de *bytes*, fazer:
     4.1 Ler cabeçalho do pacote;
     4.2 Fazer tratamento de exceção;
        4.2.1 Verificar se o *socket* foi fechado pelo processo externo;
     4.3 Se o cabeçalho for > 127 *bytes*
        4.3.1 Criar pacote com 127 *bytes*
     4.4 Caso contrário,
        4.4.1 Criar pacote com tamanho indicado no cabeçalho.

5. Ler pacote do fluxo de *bytes*;
6. Verificar se todos os pacotes foram transmitidos;
7. Se quantidade de *bytes* remanescentes, na variável que armazena a quantidade total de *bytes* recebida, for menor que 127, então,
   7.1 Interromper a estrutura de repetição.
8. Finalizar estrutura de repetição.

- **disconnect**

1. Encerrar canal para recebimento de dados;
2. Encerar canal para envio de dados;
3. Encerrar conexão.

iii) ***EncodeDecode.class***

Esta classe é uma abstração da camada de mensagens da biblioteca Comms/CPN. Implementa duas funções, uma para conversão de objetos de dados para cadeia de bytes (utilizada para mensagens enviadas a processos externos) e outra para conversão de cadeia de bytes para objeto de dados (utilizada para mensagens recebidas de processos externos).

iv) ***Servidor.class***

Esta classe foi desenvolvida, neste trabalho, utilizando as funções primitivas das classes JavaCPN e EncodeDecode para estabelecer a comunicação entre o Xj3D e os modelos em CPN. Lembrando que sempre são abertos dois canais de comunicação: um canal para receber mensagens diretamente do campo e outro canal para enviar e receber mensagens do modelo dos painéis (interface). Esta classe é composta de:

1. conectar
   1.1. Abre as duas conexões (as das mensagens advindas diretamente do campo e as mensagens trocadas (enviadas e recebidas) com o modelo dos painéis);
   1.2. Define o nome das conexões e as portas;

2. Envia
   2.1. Envia as mensagens geradas a partir da Interface X3D.

3. recebe 3.1. Recebe as mensagens advindas do Modelo dos painéis, que constituem a realimentação das mensagens enviadas.
4. recebeCampo 4.1. Recebe as mensagens vindas diretamente do campo.

Anexo C: Detalhes da Biblioteca de modelos

São apresentados neste anexo os detalhes da biblioteca convertida de Design/CPN para CPN Tools dos modelos que representam o comportamento dos objetos na sala de controle de uma subestação elétrica. A versão original da página de hierarquia dos modelos é ilustrada na Figura C-1, que continha os painéis, a IHM_industrial (que relaciona os painéis aos dispositivos) e os dispositivos de interação.

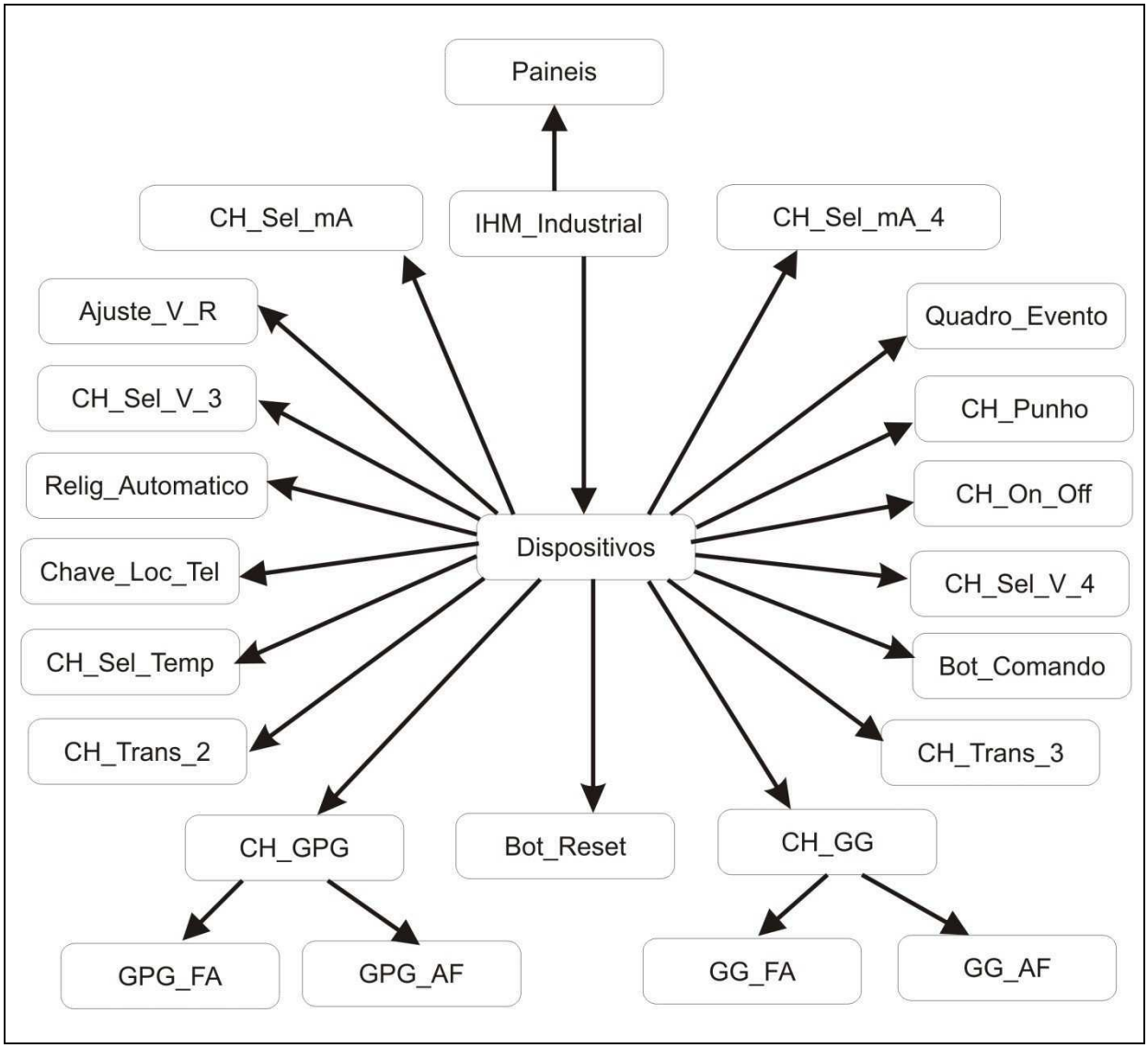

Figura C-1 – Página de Hierarquia dos modelos.

No Quadro C-1 são apresentadas as cores utilizadas para representação do modelo, no Quadro C-2 as variáveis, Quadro C-3 as funções utilizadas para uma descrição mais detalhada vide NASCIMENTO et al (2007).

```
colset Armario = with LT02J6 | LT02J5 | LT02J4 | LT02J3 | LT02J2 | LT02J1;

colset Estado_Campo = with Aberto | Fechado | Nulo;

colset Chaves = with DJ12J1 | DJ12J2 | DJ12J5  | DJ12J6 | SC32J54 | SC32J55 |SC32J56|SC32J57| DJ12D5 |
SC32J64 | SC32J66 | SC32J65 |SC32J67 |LocTel |Transf_3;

colset Est_Campo = product Armario * Chaves * Estado_Campo;

colset Painel_Chave = product Armario*Chaves;

colset Estado  = with ComCartao | SemCartao;

colset Operacao = product Armario*Estado;

colset CH_GPG = subset Chaves with [SC32J54, SC32J55, SC32J64, SC32J66,SC32J65,
SC32J67,SC32J56,SC32J57];

colset CH_Punho = subset Chaves with [DJ12J1, DJ12J2,DJ12D5,DJ12J5,DJ12J6];

colset Bot_Comando = with Bot1 | Bot2;

colset ChaveLoctel = subset Chaves with [LocTel];

colset Est_Loctel = with Local | Telecomando | nulo

colset Opcoes_Loc_Tel = product Armario*ChaveLoctel*Est_Loctel;

colset Opcoes_BCom = product Armario*Bot_Comando;

colset Bot_Reset = with Normal | Bloqueado | nexiste;

colset Opcoes_Bot_Reset = product Armario*Bot_Reset;

colset CH_GG = with a21Y5 | b21Y6;

colset Opcoes_GG = product Armario*CH_GG;

colset Est_Campo_GG = product Armario*CH_GG*Estado_Campo;

colset Quadro_Eventos = with Quadro1 | Quadro2;

colset Opcoes_Event = product Armario*Quadro_Eventos;

colset Opcoes_GPG = product Armario*CH_GPG;

colset Est_Campo_GPG = product Armario*CH_GPG*Estado_Campo;

colset Opcoes_Punho = product Armario*CH_Punho;

colset Est_Campo_P = product Armario*CH_Punho*Estado_Campo;

colset Tip_Bot = with Falta_CC;

colset Est_Bot = with Aceso | Apagado;

colset Opcoes_Bot  = product Armario*Tip_Bot*Est_Bot;

colset CH_Relig_Autom = with Ativado | Desativado | N_consta;

colset Opcoes_Rel_Aut = product Armario*CH_Relig_Autom;

colset CH_Transf_3 = subset Chaves with [Transf_3];

colset Opcoes_Tr_3 = product Armario*CH_Transf_3;

colset CH_T_4 = with ch1 | ch2;

colset Opcoes_T_4 = product Armario*CH_T_4;

colset Bot_Central = with Bcentral1 | Bcentral2;

colset Opcoes_R_C = product Armario*Bot_Central;

colset CH_V_4 = with ch_v_4_1 | ch_v_4_2;
```

```
colset Opcoes_V_4 = product Armario*CH_V_4;
colset CH_V_3 = with Ch_V_3_1 | CH_V_3_2;
colset Opcoes_V_3 = product Armario*CH_V_3;
colset CH_mA_4 = with chave_ma_4_1 | chave_ma_4_2;
colset Opcoes_A_4 = product Armario*CH_mA_4;
colset CH_Transf_2 = with chave_tr2_1 | chave_tr2_2;
colset Opcoes_Tr_2 = product Armario*CH_Transf_2;
colset CH_Sel_mA_2 = with chave_mA_1 | chave_mA_2;
colset Opcoes_A_2 = product Armario*CH_Sel_mA_2;
colset CH_on_off = with ch_onoff1 | chonoff2;
colset Opcoes_onoff = product Armario*CH_on_off;
```

Quadro C-1 – Nó de declaração das Cores

```
var LTaux, LT: Est_Loctel;
var com,comaux: Bot_Comando;
var CLT: ChaveLoctel;</layout>
var br, braux: Bot_Reset;
var qe: Quadro_Eventos;
var gg, ggaux: CH_GG;
var rc: Bot_Central;
var gpg, gpgaux: CH_GPG;
var t4: CH_T_4;
var p, paux: CH_Punho;
var v3: CH_V_3;
var v4: CH_V_4;
var a4: CH_mA_4;
var t3: CH_Transf_3;
var t2: CH_Transf_2;
var a2: CH_Sel_mA_2;
var onf: CH_on_off;
var RAux,ra:CH_Relig_Autom;
var tb: Tip_Bot;
var eb: Est_Bot;
```

Quadro C-2 – Nó de declaração das variáveis

```
fun trav_A(Paux, chaux, campaux, LTaux, braux, lista_Aux) = if (LTaux=Local orelse LTaux=nulo) andalso (braux=Normal orelse braux=nexiste) andalso (Paux=LT02J5 orelse Paux=LT02J6) then union lista_Aux [(Paux, chaux, Aberto)] else union lista_Aux [(Paux, chaux, Nulo)];

fun trav_F(Paux, chaux, campaux, LTaux, braux, lista_Aux) = if (LTaux=Local orelse LTaux=nulo) andalso (braux=Normal orelse braux=nexiste) andalso (Paux=LT02J5 orelse Paux=LT02J6) then union lista_Aux
```

```
[(Paux, chaux, Fechado)] else union lista_Aux [(Paux, chaux, Nulo)];

fun inv(Paux,Eaux)=if Eaux=ComCartao then 1`(Paux,SemCartao) else 1`(Paux,ComCartao);

fun inv_c(camp) = if camp=Aberto then Fechado else Aberto;

fun invLT(Paux,LTaux)=if LTaux=Local then 1`(Paux,LocTel,Telecomando) else 1`(Paux,LocTel,Local);

fun invRA(Paux,RAux)=if RAux=Ativado then 1`(Paux,Desativado)else 1`(Paux,Ativado);

fun atuacao86gg(Paux)=if Paux=LT02J5 then 1`(Paux,a21Y5,Aberto) else empty;

fun atuacao86gpg(Paux)=if Paux=LT02J5 then 1`(Paux,DJ12J5,Aberto) else empty;

fun trav_p(Paux,paux,campaux,LTaux,braux)=if (LTaux=Local orelse LTaux=nulo) andalso (
braux=Normal orelse braux=nexiste) andalso Paux=LT02J5 then 1`(Paux,paux,inv_c(campaux))

else 1`(Paux,paux,campaux);

fun atuacao86p(Paux)=if Paux=LT02J5 then 1`(Paux,DJ12D5,Aberto) else empty;

fun trav_gg (Paux,ggaux,campaux,LTaux,braux)=if (LTaux=Local orelse LTaux=nulo) andalso (
braux=Normal orelse braux=nexiste) andalso Paux=LT02J5 then 1`(Paux,ggaux,inv_c(campaux)) else
1`(Paux,ggaux,campaux);

fun trav_gpg (Paux,gpgaux,campaux,LTaux,braux)=if (LTaux=Local orelse LTaux=nulo) andalso (
braux=Normal orelse braux=nexiste) andalso Paux=LT02J5 then 1`(Paux,gpgaux,inv_c(campaux)) else
1`(Paux,gpgaux,campaux);
```

Quadro C-3 – Nó de declaração das Funções

Anexo D: Códigos gerados no desenvolvimento do projeto

Este anexo apresenta os códigos gerados ao longo deste projeto, visando complementar a documentação necessária à sua continuidade. O código esta apresentado na seguinte sequencia:

- Principal:
  - Browser;
  - Motor Display;
  - Mostrador;
  - Converter número para Display;
  - Parametrizador;
- Pacote comunicação:
  - Servidor;
  - JavaCPN;
  - EncodeDecode.

Código Principal – Browser:

```
/**
 * Descrição: Classe Principal responsável pela conversão das mensagens entre o Ambiente Virtual e os modelos.
 * @author Ademar Virgolino - LIHM
 * @version 1.0
 */

import java.awt.BorderLayout;
import java.awt.Container;
//import java.net.SocketException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Timer;

import javax.swing.JComponent;
import javax.swing.JFrame;
import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.interpolation.OrientationInterpolator;
import org.web3d.x3d.sai.grouping.Transform;

//Pacote interno
//import comunicacao.EncodeDecode;
```

```
//import comunicacao.NovaThread;
import comunicacao.Servidor;
//import comunicacao.JavaCPN;

public class Browser extends JFrame implements X3DFieldEventListener {

	private static X3DScene mainScene;
	private MFString horas;
	private MFString minutos;
	private MFString segundos;
	private Mostrador display1;
	private Mostrador display2;
	private Mostrador display3;


	//Criacao de mais mostradores de 7 segmentos

	private static int novaunidade;
	private static int novaunidade2;
	private static int novaunidade3;

	private static float[] valoresParametrizados = new float[4];
	private boolean estadoDoPonteiro = false;

	//Criacao de arrays de valores parametrizados
	private static float[] valoresParametrizados2 = new float[4];
	private static Parametrizador parametrizador = new Parametrizador(((float)1.57),0, 200);

	//Altera o valor maximo da grandeza a ser parametrizada. Ex: Abaixo, o valor 100 indica o
	//valor maximo da grandeza seja tensao, corrente ou potencia.
	private static float[] valoresParametrizados3 = new float[4];
	private static Parametrizador parametrizador2 = new Parametrizador(((float)1.57),-100, 100);


	public Servidor conector;
	private static ArrayList<String> nome;
	private Boolean eventTouch = false;

	float[] GREEN = {0, 1.0f, 0};
	float[] RED = {1.0f, 0, 0};
	float[] YELLOW = {1, 1, 0};
	float[] FECHADO_V = {0, 1, 0, 0};
	float[] ABERTO_V = {0, 0, 1, 1.57f};
	float[] FECHADO_H = {0, 0, 1, 1.57f};
	float[] ABERTO_H = {0, 1, 0, 0};

	float[] FECHADO_V2 = {0, 1, 0, 0};
	float[] ABERTO_V2 = {1, 0, 0, 1.57f};
	float[] FECHADO_H2 = {1, 0, 0, 1.57f};
	float[] ABERTO_H2 = {0, 1, 0, 0};

	float[] valorTranslacao = new float[4];


	public Browser() {
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container contentPane = getContentPane();
//		Setup browser parameters
		HashMap requestedParameters = new HashMap();
```

```
//                  Create an SAI component
                    X3DComponent x3dComp = BrowserFactory.createX3DComponent(requestedParameters);

//                  Add the component to the UI
                    JComponent x3dPanel = (JComponent)x3dComp.getImplementation();
                    contentPane.add(x3dPanel, BorderLayout.CENTER);

//                  Get an external browser
                    ExternalBrowser x3dBrowser = x3dComp.getBrowser();

                    setSize(500,500);
                    show();


//                  Create an X3D scene by loading a file
                    mainScene = x3dBrowser.createX3DFromURL(new String[] { "Armario 12J5.x3d" });

                    X3DNode horas = mainScene.getNamedNode("HourText");
                    X3DNode minutos = mainScene.getNamedNode("MinuteText");
                    X3DNode segundos = mainScene.getNamedNode("SecondText");

                    this.horas = (MFString)horas.getField("string");
                    this.minutos = (MFString)minutos.getField("string");
                    this.segundos = (MFString)segundos.getField("string");
                    atualizaHoras();
                    X3DNode time = mainScene.getNamedNode("Clock_Sensor");
//                  Replace the current world with the new one
                    x3dBrowser.replaceWorld(mainScene);

//                  Instancia a classe Servidor()
                    conector = new Servidor();
                    conector.conectar();



//                  Cria a lista de sensores touch
                    nome = adicionarChave();

                    listen(nome);

                    verificaCampo();

            }


            /**
             * Define a lista de sensores do tipo touch
             * @return nome uma lista de array com o nome dos sensores
             */
            public static ArrayList<String> adicionarChave(){
                    nome = new ArrayList<String>();

                    nome.add("Chave_32J5_6_Sensor");
                    nome.add("Chave_32J5_5_Sensor");
                    nome.add("Chave_32J5_4_Sensor");
                    nome.add("Chave_32J5_7_Sensor");
                    nome.add("Disjuntor_12J5_Sensor");
                    nome.add("Botao_Loctel_12J5_Sensor");

                    nome.add("Chave_32J6_6_Sensor");
```

```
                nome.add("Chave_32J6_5_Sensor");
                nome.add("Chave_32J6_4_Sensor");
                nome.add("Chave_32J6_7_Sensor");
                nome.add("Disjuntor_12J6_Sensor");
                nome.add("Botao_Loctel_12J6_Sensor");

                nome.add("onofflamp");

                nome.add("Disjuntor_14C3_Sensor");

                return nome;
        }
        /**
         * Cria o monitoramento dos sensores
         * @param lista Contem a lista com o nome dos sensores
         */
        private void atualizaHoras() {

                Date data = new Date(System.currentTimeMillis());
                int auxsegundos = data.getSeconds();
                int auxminutos = data.getMinutes();
                int auxhoras = data.getHours();

                if(auxsegundos < 10) {
                        segundos.set1Value(0, "0" + auxsegundos);
                } else {
                        segundos.set1Value(0, "" + auxsegundos);
                }

                if(auxminutos < 10) {
                        minutos.set1Value(0, "0" + auxminutos);
                } else {
                        minutos.set1Value(0, "" + auxminutos);

                }if(auxhoras < 10) {
                        horas.set1Value(0, "0" + auxhoras);
                } else {
                        horas.set1Value(0, "" + auxhoras);
                }

        }
        /**
         * Varre a lista e habilita touchsensor
         * @param lista
         */
        public void listen(ArrayList<String> lista){

                Iterator<String> it = lista.iterator();
                X3DNode generic;
                SFTime genericTF;
                String nomeSensor;

//              Varre a lista de string para saber o nome dos touchsensors
                while(it.hasNext()){
                        nomeSensor=it.next();
                        generic = mainScene.getNamedNode(nomeSensor);
//                      System.out.println("e  "+generic.toString());
                        genericTF = (SFTime) generic.getField("touchTime");
                        System.out.println("Nome do sensor:  "+nomeSensor);
//                      Envia qual sensor foi ativado
```

```
                        genericTF.setUserData(nomeSensor);
//                      Aguardo um sensor ser ativado
                        genericTF.addX3DEventListener(this);
                }
        }

        /**
         * Função de monitoramento dos touchsensors.
         * É necessario adicionar a condição para ativar a chave.
         * @param evt Recebe o nome do touchsensor ativado
         */
        public void readableFieldChanged(X3DFieldEvent evt) {

                Object touchAtivado;

                if(eventTouch){
                        verificaCampo();
                }
//              Capta o nome do sensor que foi ativado
                touchAtivado = evt.getData();
                System.out.println("O touch ativado foi:  "+touchAtivado);


//              Adicionar aqui a função de comparação para ativar uma chave.

                //Armario 12J5
                if(touchAtivado.equals("Chave_32J5_6_Sensor")){


                        //Chama a funcao de movimentacao dos ponteiros.
                        //Especifica a criacao dos mostradores de 7 segmentos
                        //Ex: seg_ -> nome do mostrador;
                        //    3 -> numeros de mostradores



                        motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR", "PI","TG",
valoresParametrizados3);
                        motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR2", "PI2","TG2",
valoresParametrizados2);
                        motorDoPonteiro("Chave_32J5_6_Sensor", "TIME_SENSOR3", "PI3","TG3",
valoresParametrizados);


                        conversorBotao("32J5_6", "Chave_32J5_6", "LT02J5,SC32J56", "vertical");
                }else
                if(touchAtivado.equals("Chave_32J5_5_Sensor")){
                        conversorBotao("32J5_5", "Chave_32J5_5", "LT02J5,SC32J55", "vertical");
                }else
                if(touchAtivado.equals("Chave_32J5_4_Sensor")){
                        conversorBotao("32J5_4", "Chave_32J5_4", "LT02J5,SC32J54", "vertical");
                }else
                if(touchAtivado.equals("Chave_32J5_7_Sensor")){
                        conversorBotao("32J5_7", "Chave_32J5_7", "LT02J5,SC32J57", "horizontal");
                }else
                if(touchAtivado.equals("Disjuntor_12J5_Sensor")){
                        display1 = new Mostrador(mainScene, "seg_", 4);
                        display2 = new Mostrador(mainScene, "seg2_", 4);
                        display3 = new Mostrador(mainScene, "seg3_", 4);
                        if (!(estadoDoPonteiro)) {
```

```
                                        //Modifica display especifico com a unidade desejada
                                        motorDoDisplay(display1, novaunidade2);
                                        motorDoDisplay(display2, novaunidade);
                                        motorDoDisplay(display3, novaunidade3);

                                        estadoDoPonteiro = true;
                                } else {
                                        motorDoDisplay(display1, 0);
                                        motorDoDisplay(display2, 0);
                                        motorDoDisplay(display3, 0);

                                        estadoDoPonteiro = false;
                                }
                                conversorBotao("12J5", "Disjuntor_12J5", "LT02J5,DJ12J5", "vertical");
                        }else
                        if(touchAtivado.equals("Botao_Loctel_12J5_Sensor")){
                                conversorLoctel("Botao_Loctel_12J5", "LT02J5,LocTel");
                        }else

                        // 12J6
                        if(touchAtivado.equals("Chave_32J6_6_Sensor")){
                                conversorBotao("32J6_6", "Chave_32J6_6", "LT02J6,SC32J66", "vertical");
                        }else
                        if(touchAtivado.equals("Chave_32J6_5_Sensor")){
                                conversorBotao("32J6_5", "Chave_32J6_5", "LT02J6,SC32J65", "vertical");
                        }else
                        if(touchAtivado.equals("Chave_32J6_4_Sensor")){
                                conversorBotao("32J6_4", "Chave_32J6_4", "LT02J6,SC32J64", "vertical");
                        }else
                        if(touchAtivado.equals("Chave_32J6_7_Sensor")){
                                conversorBotao("32J6_7", "Chave_32J6_7", "LT02J6,SC32J67", "horizontal");
                        }else
                        if(touchAtivado.equals("Disjuntor_12J6_Sensor")){
                                conversorBotao("12J6", "Disjuntor_12J6", "LT02J6,DJ12J6", "vertical");
                        }else
                        if(touchAtivado.equals("Botao_Loctel_12J6_Sensor")){
                                conversorLoctel("Botao_Loctel_12J6_3", "LT02J6,LocTel");
                        }else



                        // Armario Extra

                        if(touchAtivado.equals("Disjuntor_14C3_Sensor")){

                                motorDoPonteiro("Disjuntor_14C3_Sensor", "TIME_SENSOR4", "PI4","TG4",
valoresParametrizados2);
                                motorDoPonteiro("Disjuntor_14C3_Sensor", "TIME_SENSOR5", "PI5","TG5",
valoresParametrizados2);
                                motorDoPonteiro("Disjuntor_14C3_Sensor", "TIME_SENSOR6", "PI5","TG6",
valoresParametrizados3);

                                if (!(estadoDoPonteiro)) {


                                        estadoDoPonteiro = true;
                                } else {

                                        estadoDoPonteiro = false;
                                }
```

```
                        conversorBotao("14C3", "Chave_14C3", "14C3,SC14C3", "vertical");
                }


                //Acendimento de lampadas da sala
                else
                        if(touchAtivado.equals("onofflamp")){
                                X3DLightNode lamp = (X3DLightNode)
mainScene.getNamedNode("lamp");
                                boolean aceso = true;
                                boolean apagado = false;
                                boolean estado = lamp.getOn();
                                if(estado == aceso) {
                                        lamp.setOn(apagado);
                                } else {
                                        lamp.setOn(aceso);
                                }

                }


        }
        /**
                 * MOTOR
                 * @param display
                 * @param numero
                 */

        public void motorDoDisplay(Mostrador display, int numero) {
                display.mudarCorDoSegmento(numero);
        }

        /**
         *
         * @param nomeDoTouch
         * @param nomeDoTime
         * @param positionInterpolator
         * @param nomeDoPonteiro
         * @param valoresParametrizados
         */
        //Funcao de movimentacao dos ponteiros
        public void motorDoPonteiro(String nomeDoTouch, String nomeDoTime,
                                        String positionInterpolator, String nomeDoPonteiro, float[]
valoresParametrizados) {
                float[] keyvalues = new float[20];
                X3DNode TCHS = mainScene.getNamedNode(nomeDoTouch);
    X3DNode TS = mainScene.getNamedNode(nomeDoTime);
    X3DNode PI = mainScene.getNamedNode(positionInterpolator);
    ((OrientationInterpolator) PI).getKeyValue(keyvalues);
    //Configura os novos valores para o ponteiro.
    if (!(estadoDoPonteiro)) {
        keyvalues[3] = 0;
        keyvalues[7] = valoresParametrizados[0];
        keyvalues[11] = valoresParametrizados[1];
        keyvalues[15] = valoresParametrizados[2];
        keyvalues[19] = valoresParametrizados[3];
    } else {
         keyvalues[3] = valoresParametrizados[3];
         keyvalues[7] = valoresParametrizados[2];
       keyvalues[11] = valoresParametrizados[1];
```

```
            keyvalues[15] = valoresParametrizados[0];
            keyvalues[19] = 0;
        }
        ((OrientationInterpolator) PI).setKeyValue(keyvalues);
        X3DNode TG = mainScene.getNamedNode(nomeDoPonteiro);
        mainScene.addRoute(TCHS,"touchTime",TS,"startTime");
        mainScene.addRoute(TS,"fraction_changed",PI,"set_fraction");
        mainScene.addRoute(PI,"value_changed",TG,"set_rotation");
            }



            /**
             * Converte o sinal de um touchsensor em uma ação.
             * @param SphereColor Nome do objeto que representa a esfera da chave no X3D Ex.:
SphereColor = "32J5_6".
             * @param nomeChave Nome da Chave representada no X3D que sera manipulada Ex.:nomeChave
= "Chave_32J5_6".
             * @param nomeRede Nome da painel e da chave na rede CPN Ex.: nomeRede =
"LT02J5,SC32J66".
             * @param posicaoChave posicao de Fechamento da chave (vertical/horizontal)
             */
            public void conversorBotao(String SphereColor, String nomeChave, String nomeRede, String
posicaoChave){

                    float[] estadoChave = new float[4];
//                  Recebe mensagem do campo
                    String recebido;

                    X3DNode mat = mainScene.getNamedNode("SphereColor_"+SphereColor);
                    if (mat == null) {
                            System.out.println("Couldn't find material named: SphereColor_"+SphereColor);
                            return;
                    }
                    SFColor color = (SFColor) mat.getField("diffuseColor");


                    X3DNode  tipoChave= mainScene.getNamedNode(nomeChave);
                    if (tipoChave == null) {
                            System.out.println("Nó: "+nomeChave+" não encontrado.");
                            return;
                    }
                    else{
                            SFRotation pos = (SFRotation) tipoChave.getField("rotation");
//                          Salva a posicao de rotacao do objeto e armazena em estadoChave
                            pos.getValue(estadoChave);

                            X3DNode DJ = mainScene.getNamedNode(nomeChave);
                            ((Transform) DJ).getTranslation(valorTranslacao);
                            System.out.println("Nó: "+nomeChave+" Valores: "+valorTranslacao[1]+"
"+valorTranslacao[2]);

                            if(valorTranslacao[1]==0.6f){

                                    System.out.println("Passei por aqui");

//                                  Compara o valor atual de rotacao (estadoChave) com valor da constante
ABERTO/FECHADO

                                    if(posicaoChave.equals("vertical")){
```

```
                                        if(Arrays.equals(estadoChave, ABERTO_V2)){
//                                              Muda o valor de rotacao
                                                pos.setValue(FECHADO_V2);
//                                              Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                conector.envia(nomeRede);
                                                recebido = conector.recebe();
                                                if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                        color.setValue(YELLOW);
                                                }
                                                else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                        color.setValue(RED);
                                                }
                                        }else if(Arrays.equals(estadoChave, FECHADO_V2)){
//                                              Muda o valor de rotacao
                                                pos.setValue(ABERTO_V2);
//                                              Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                conector.envia(nomeRede);
                                                recebido = conector.recebe();
                                                if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                        color.setValue(GREEN);
                                                }
                                                else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                        color.setValue(YELLOW);
                                                }
                                        }

                                }if(posicaoChave.equals("horizontal")){
                                        if(Arrays.equals(estadoChave, ABERTO_H2)){
//                                              Muda o valor de rotacao
                                                pos.setValue(FECHADO_H2);
//                                              Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                conector.envia(nomeRede);
                                                recebido = conector.recebe();
                                                if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                        color.setValue(YELLOW);
                                                }
                                                else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                        color.setValue(RED);
                                                }
                                        }else if(Arrays.equals(estadoChave, FECHADO_H2)){
//                                              Muda o valor de rotacao
                                                pos.setValue(ABERTO_H2);
//                                              Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                conector.envia(nomeRede);
                                                recebido = conector.recebe();
                                                if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                        color.setValue(GREEN);
                                                }
                                                else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                        color.setValue(YELLOW);
                                                }
                                        }
```

```
                                        }

                                }



                                if(valorTranslacao[2]==37.66f){
                                        System.out.println("Passei por aqui");
//                                      Compara o valor atual de rotacao (estadoChave) com valor da constante
ABERTO/FECHADO

                                        if(posicaoChave.equals("vertical")){
                                                if(Arrays.equals(estadoChave, ABERTO_V)){
//                                                      Muda o valor de rotacao
                                                        pos.setValue(FECHADO_V);
//                                                      Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                        conector.envia(nomeRede);
                                                        recebido = conector.recebe();
                                                        if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                                color.setValue(YELLOW);
                                                        }
                                                        else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                                color.setValue(RED);
                                                        }
                                                }else if(Arrays.equals(estadoChave, FECHADO_V)){
//                                                      Muda o valor de rotacao
                                                        pos.setValue(ABERTO_V);
//                                                      Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                        conector.envia(nomeRede);
                                                        recebido = conector.recebe();
                                                        if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                                color.setValue(GREEN);
                                                        }
                                                        else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                                color.setValue(YELLOW);
                                                        }
                                                }

                                        }if(posicaoChave.equals("horizontal")){
                                                if(Arrays.equals(estadoChave, ABERTO_H)){
//                                                      Muda o valor de rotacao
                                                        pos.setValue(FECHADO_H);
//                                                      Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                        conector.envia(nomeRede);
                                                        recebido = conector.recebe();
                                                        if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                                color.setValue(YELLOW);
                                                        }
                                                        else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                                color.setValue(RED);
                                                        }
                                                }else if(Arrays.equals(estadoChave, FECHADO_H)){
//                                                      Muda o valor de rotacao
```

```
                                                pos.setValue(ABERTO_H);
//                                              Envia o nome do painel e chave para a rede de paineis
do CPNTools
                                                conector.envia(nomeRede);
                                                recebido = conector.recebe();
                                                if(recebido.equalsIgnoreCase(nomeRede+",Aberto")){
                                                        color.setValue(GREEN);
                                                }
                                                else
if(recebido.equalsIgnoreCase(nomeRede+",Fechado")){
                                                        color.setValue(YELLOW);
                                                }
                                        }
                                }
                        }
                }
        }
        /**
         * Conversor para a Chave do tipo Local/Telecomando
         * @param nomePainel Nome da chave loctel
         * @param nomeRede Nome da Ficha na redes de petri
         */
        public void conversorLoctel(String nomePainel, String nomeRede){

                float[] estadoChave = new float[4];

                X3DNode  tipoChave= mainScene.getNamedNode(nomePainel);
                if (tipoChave == null) {
                        System.out.println("Nó: "+nomePainel+" não encontrado Loctel.");
                        return;
                }
                else{
                        SFRotation pos = (SFRotation) tipoChave.getField("rotation");
//                      Salva a posicao de rotacao do objeto e armazena em estadoChave
                        pos.getValue(estadoChave);
//                      Compara o valor atual de rotacao (estadoChave) com valor da constante
ABERTO/FECHADO
                        if(Arrays.equals(estadoChave, ABERTO_V)){
//                              Muda o valor de rotacao
                                pos.setValue(FECHADO_V);
//                              Envia o nome do painel e chave para a rede de paineis do CPNTools
                                conector.envia(nomeRede);

                        }
//                      Compara o valor atual de rotation (estadoChave) com valor da constante
ABERTO/FECHADO
                        else if (Arrays.equals(estadoChave, FECHADO_V)){
                                pos.setValue(ABERTO_V);
                                conector.envia(nomeRede);
                        }
                }

        }
        private void Reminder() {
                verificaCampo();
   }


        /**
         * Abre canal de recebimento de fichas vindas do campo
```

```
         */
        private void verificaCampo() {
                String msgRecCampo;
                System.out.println("Esperando nova mensagem: ");
                msgRecCampo = conector.recebeCampo();
                String s[] = msgRecCampo.split(",");
//              conversorCampoX3D(String SphereColor, String nomeChave, String posicaoChave, String
nomeRede,String msgRecCampo)

                // Painel LT02J5
                if(s[1].equals("SC32J56")){

        conversorCampoX3D("32J5_6","Chave_32J5_6","vertical","LT02J05,SC32J56",msgRecCampo);
                }else if(s[1].equals("SC32J55")){

        conversorCampoX3D("32J5_5","Chave_32J5_5","vertical","LT02J05,SC32J55",msgRecCampo);
                }else if(s[1].equals("SC32J54")){

        conversorCampoX3D("32J5_4","Chave_32J5_4","vertical","LT02J05,SC32J54",msgRecCampo);
                }else if(s[1].equals("SC32J57")){

        conversorCampoX3D("32J5_7","Chave_32J5_7","horizontal","LT02J05,SC32J57",msgRecCampo);
                }else if(s[1].equals("DJ12J5")){

        conversorCampoX3D("12J5","Disjuntor_12J5","vertical","LT02J5,DJ12J5",msgRecCampo);
                }

                // Painel LT02J6
                else if(s[1].equals("SC32J66")){

        conversorCampoX3D("32J6_6","Chave_32J6_6","vertical","LT02J06,SC32J66",msgRecCampo);
                }else if(s[1].equals("SC32J65")){

        conversorCampoX3D("32J6_5","Chave_32J6_5","vertical","LT02J06,SC32J65",msgRecCampo);
                }else if(s[1].equals("SC32J64")){

        conversorCampoX3D("32J6_4","Chave_32J6_4","vertical","LT02J06,SC32J64",msgRecCampo);
                }else if(s[1].equals("SC32J67")){

        conversorCampoX3D("32J6_7","Chave_32J6_7","horizontal","LT02J06,SC32J67",msgRecCampo);
                }else if(s[1].equals("DJ12J6")){

        conversorCampoX3D("12J6","Disjuntor_12J6","vertical","LT02J6,DJ12J6",msgRecCampo);
                }else if(s[1].equals("DJ12J6")){

        conversorCampoX3D("12J6","Disjuntor_12J6","vertical","LT02J6,DJ12J6",msgRecCampo);
                }


                eventTouch = true;
                Reminder();
        }
        public void conversorCampoX3D(String SphereColor, String nomeChave,        String
posicaoChave, String nomeRede,String msgRecCampo){
                float[] estadoChave = new float[4];
                X3DNode mat = mainScene.getNamedNode("SphereColor_"+SphereColor);

                if (mat == null) {
                        System.out.println("Couldn't find material named: SphereColor_"+SphereColor);
                        return;
```

```
                }
                SFColor color = (SFColor) mat.getField("diffuseColor");

                X3DNode  tipoChave= mainScene.getNamedNode(nomeChave);
                if (tipoChave == null) {
                        System.out.println("Nó: "+nomeChave+" não encontrado.");
                        return;
                }else{
                        SFRotation pos = (SFRotation) tipoChave.getField("rotation");
                        //Salva a posicao de rotacao do objeto e armazena em estadoChave
                        pos.getValue(estadoChave);
                        if(posicaoChave.equals("vertical")){
                                if(Arrays.equals(estadoChave, ABERTO_V)){
                                        if(msgRecCampo.equalsIgnoreCase(nomeRede+",Aberto")){
                                                color.setValue(GREEN);
                                        }else{
                                                color.setValue(YELLOW);
                                        }
                                }else if(Arrays.equals(estadoChave, FECHADO_V)){
                                        if(msgRecCampo.equalsIgnoreCase(nomeRede+",Fechado")){
                                                color.setValue(RED);
                                        }else{
                                                color.setValue(YELLOW);
                                        }
                                }
                        }else if(posicaoChave.equals("horizontal")){
                                if(Arrays.equals(estadoChave, ABERTO_H)){
                                        if(msgRecCampo.equalsIgnoreCase(nomeRede+",Aberto")){
                                                color.setValue(GREEN);
                                        }else{
                                                color.setValue(YELLOW);
                                        }
                                }else if(Arrays.equals(estadoChave, FECHADO_H)){
                                        if(msgRecCampo.equalsIgnoreCase(nomeRede+",Fechado")){
                                                color.setValue(RED);
                                        }else{
                                                color.setValue(YELLOW);
                                        }
                                }
                        }
                }
        }
        /**
         * Main method.
         *
         * @param args None handled
         */
        public static void main(String[] args) {
                System.out.println("Digite um valor de tensão (1): ");
                Scanner sc = new Scanner(System.in);
                novaunidade = sc.nextInt();
    valoresParametrizados = parametrizador.getArrayForInterpolation(novaunidade);

                System.out.println("Digite um valor de tensão (2): ");
                novaunidade2 = sc.nextInt();
    valoresParametrizados2 = parametrizador.getArrayForInterpolation(novaunidade2);

    System.out.println("Digite um valor de tensão (3): ");
                novaunidade3 = sc.nextInt();
                valoresParametrizados3 = parametrizador2.getArrayForInterpolation(novaunidade3);
```

```
        Browser Inicia = new Browser();

            }
}
```

Código – Motor Display

```
import org.web3d.x3d.sai.*;

/**
 * Descrição: Classe responsável pela atualização de um número para um display.
 * @author Erick - Tutoria Ademar Virgolino - LIHM
 * @version 1.0
 */
public class MotorDeDisplay {
	private Object[] arraysDoDisplay;
	private int tamanhoDoDisplay;
	public MotorDeDisplay(int tamanhoDoDisplay) {
		this.tamanhoDoDisplay = tamanhoDoDisplay;
	}
	/**
	 *
	 * @param numero Número a ser mostrado no display.
	 * @param segmento Array com as o valores do display de 7 segmentos.
	 * @throws Exception Caso o número não seja suportado pelo display.
	 */
	public void modificaSegmento(int numero, SFColor[][] segmento) throws Exception {
		boolean negativo = (numero < 0);
		if(negativo) {
			numero = -1 *numero;
		}
		String auxiliar = Integer.toString(numero);
		/**
		 * Se o número tiver mais algarismos que o display suporta, é lançada uma exceção.
		 * Ex de números suportados para um diplay de 3 algarismos: -99 a 99
		 * O algarismo mais significativo do display é reservado para o sinal.
		 */
		if (((auxiliar.length() >= tamanhoDoDisplay) && (!(negativo))) ||
				((auxiliar.length() > tamanhoDoDisplay-1) && (negativo))) {
			Exception e = new Exception("Número não suportado pelo display.");
			throw e;
		} else {
			/**
			 * Corrige números menores que o suporte do display.
			 * Exemplo, em um display de 3 algarismos, números como 2, -2 serão mostrados
			 * como: 02, -02
			 */
			int diferenca = ((tamanhoDoDisplay) - auxiliar.length());
			for(int i = 0; i < diferenca; i++) {
				auxiliar = "0" + auxiliar;
			}

			if(negativo) {
				auxiliar = "-" + auxiliar.substring(1);
			} else {
				auxiliar = "x" + auxiliar.substring(1);
			}
		}
		// Atualiza a string formatada pro display.
		for(int k=0; k<auxiliar.length(); k++) {
			arraysDoDisplay =
ConverteNumeroParaDisplay.converterInteiro(auxiliar.charAt(k));
			for(int l=0; l < arraysDoDisplay.length; l++) {
				float[] arrayAuxiliar = new float[3];
				arrayAuxiliar = (float[]) arraysDoDisplay[l];
				segmento[k][l].setValue(arrayAuxiliar);
```

```
                        }
                }
        }

}
```

Código – Mostrador

```
import java.awt.*;
import java.util.HashMap;
import java.util.Scanner;

import javax.swing.*;
import java.io.File;

import org.web3d.x3d.sai.*;
import org.web3d.x3d.sai.interpolation.OrientationInterpolator;

/**
 * Classe responsável pelo mostrador de 7 segmentos.
 *
 * @author Erick Lucena – Tutoria Ademar Virgolino - LIHM
 * @version 1.0
 */
public class Mostrador {
   private X3DScene mainScene;
   private int SEGMENTOS = 7;
   private int numeroDeAlgarismos;
   private SFColor[][] corDoSegmento;

   /**
    *
    * @param
    */
   public Mostrador(X3DScene mainScene, String nome, int numeroDeAlgarismos) {
        this.numeroDeAlgarismos = numeroDeAlgarismos;
        this.mainScene = mainScene;
        corDoSegmento = new SFColor[numeroDeAlgarismos][SEGMENTOS];
        X3DNode[][] palitos = new X3DNode[numeroDeAlgarismos][SEGMENTOS];
        for(int k = 0;  k < numeroDeAlgarismos; k++) {
                for(int i = 0; i < SEGMENTOS; i++) {
                        palitos[k][i] = mainScene.getNamedNode(nome + (k+1) + "" + (i+1));
                }
        }
        for(int k = 0;  k < numeroDeAlgarismos; k++) {
                for(int i = 0; i < SEGMENTOS; i++) {
                        corDoSegmento[k][i] = (SFColor) palitos[k][i].getField("diffuseColor");
                }
        }
   }
   public void mudarCorDoSegmento(int numero) {
        MotorDeDisplay motor = new MotorDeDisplay(numeroDeAlgarismos);
        try {
                motor.modificaSegmento(numero,corDoSegmento);
        } catch(Exception e) {
                System.out.println(e.getMessage());
        }
   }
}
```

Código – Converter número para Display

```
public class ConverteNumeroParaDisplay {
        private static float[] ACESO = {0f, 0f, 0f};
        private static float[] APAGADO = {0.65f, 0.7f, 0.6f};
        private static int[] configuracao = new int[7];

        public ConverteNumeroParaDisplay() {
        }

        private static void converterInteiroMenorQue10(char numero) {
                        switch(numero) {
                        case '0': { configuracao[0] = 1;
                                                configuracao[1] = 1;
                                                configuracao[2] = 1;
                                                configuracao[3] = 1;
                                                configuracao[4] = 1;
                                                configuracao[5] = 1;
                                                configuracao[6] = 0;
                                                break;
                                }
                        case '1': {     configuracao[0] = 0;
                                                configuracao[1] = 1;
                                                configuracao[2] = 1;
                                                configuracao[3] = 0;
                                                configuracao[4] = 0;
                                                configuracao[5] = 0;
                                                configuracao[6] = 0;
                                                break;
                                }
                        case '2': {     configuracao[0] = 1;
                                                configuracao[1] = 0;
                                                configuracao[2] = 1;
                                                configuracao[3] = 1;
                                                configuracao[4] = 0;
                                                configuracao[5] = 1;
                                                configuracao[6] = 1;
                                                break;
                                }
                        case '3': {     configuracao[0] = 1;
                                                configuracao[1] = 1;
                                                configuracao[2] = 1;
                                                configuracao[3] = 1;
                                                configuracao[4] = 0;
                                                configuracao[5] = 0;
                                                configuracao[6] = 1;
                                                break;
                                }
                        case '4': { configuracao[0] = 0;
                                                configuracao[1] = 1;
                                                configuracao[2] = 1;
                                                configuracao[3] = 0;
                                                configuracao[4] = 1;
                                                configuracao[5] = 0;
                                                configuracao[6] = 1;
                                                break;
                                }
                        case '5': {     configuracao[0] = 1;
                                                configuracao[1] = 1;
                                                configuracao[2] = 0;
```

```
                                        configuracao[3] = 1;
                                        configuracao[4] = 1;
                                        configuracao[5] = 0;
                                        configuracao[6] = 1;
                                        break;
                        }
        case '6': {             configuracao[0] = 1;
                                        configuracao[1] = 1;
                                        configuracao[2] = 0;
                                        configuracao[3] = 1;
                                        configuracao[4] = 1;
                                        configuracao[5] = 1;
                                        configuracao[6] = 1;
                                        break;
                        }
        case '7': {             configuracao[0] = 0;
                                        configuracao[1] = 1;
                                        configuracao[2] = 1;
                                        configuracao[3] = 1;
                                        configuracao[4] = 0;
                                        configuracao[5] = 0;
                                        configuracao[6] = 0;
                                        break;
                        }
        case '8': {             configuracao[0] = 1;
                                        configuracao[1] = 1;
                                        configuracao[2] = 1;
                                        configuracao[3] = 1;
                                        configuracao[4] = 1;
                                        configuracao[5] = 1;
                                        configuracao[6] = 1;
                                        break;
                        }
        case '9': {             configuracao[0] = 1;
                                        configuracao[1] = 1;
                                        configuracao[2] = 1;
                                        configuracao[3] = 1;
                                        configuracao[4] = 1;
                                        configuracao[5] = 0;
                                        configuracao[6] = 1;
                                        break;
                        }
        case 'x': {             configuracao[0] = 0;
                                        configuracao[1] = 0;
                                        configuracao[2] = 0;
                                        configuracao[3] = 0;
                                        configuracao[4] = 0;
                                        configuracao[5] = 0;
                                        configuracao[6] = 0;
        break;
        }
        case '-': {             configuracao[0] = 0;
                                        configuracao[1] = 0;
                                        configuracao[2] = 0;
                                        configuracao[3] = 0;
                                        configuracao[4] = 0;
                                        configuracao[5] = 0;
                                        configuracao[6] = 1;
        break;
        }
```

```
                }
            }
        public static Object[] converterInteiro(char numero) {
            Object[] resultado = new Object[7];
            converterInteiroMenorQue10(numero);
            for(int k=0; k < configuracao.length; k++) {
                if (configuracao[k] == 1) {
                    resultado[k] = ACESO;
                } else {
                    resultado[k] = APAGADO;
                }
            }
            return resultado;
        }
}
```

Código – Parametrizador

```
/**
 *
 * @author Erick Lucena
 * @version 1.0
 * Descrição: Classe que parametriza dados como tensao, corrente, entre outros
 * para um ângulo a ser usado no ponteiro.
 */
public class Parametrizador {

    private float graus;
    private int numeroMaximoDaUnidade;
    private int numeroMinimoDaUnidade;
    private int delta;

    /**
     *
     * @param graus Número máximo de ângulo que o ponteiro desloca(em radianos).
     * @param numeroMaximoUnidade Número máximo da unidade a ser calculada.
     */
    public Parametrizador(float graus,int numeroMinimoDaUnidade, int numeroMaximoDaUnidade) {
        this.graus = graus;
        this.numeroMinimoDaUnidade = numeroMinimoDaUnidade;
        this.numeroMaximoDaUnidade = numeroMaximoDaUnidade;
        delta = numeroMaximoDaUnidade - numeroMinimoDaUnidade;


    }
    /**
     *
     * @param unidade Unidade em que se dejeja encontrar o equivalente em radianos.
     * @return Retorna o ângulo equivalente ao valor em radianos.
     */
    public float calcula(int unidade) {
            if ( numeroMinimoDaUnidade < 0) {
                return ((((unidade + numeroMaximoDaUnidade)*graus) / delta));
            } else {
                return ((unidade*graus) / delta);
            }
    }
    /**
     *
     * @param unidade Unidade que o ponteiro deve marcar
```

```
     * @return Retorna o array com os valores a serem seguidos pelo ponteiro.
     */
    public float[] getArrayForInterpolation(int unidade) {
        float angulo = calcula(unidade);
        float intervalo = (angulo / 4);
        float[] arrayDeRetorno = new float[4];
        for(int k = 0; k < 4; k++) {
                arrayDeRetorno[3-k] = (angulo - (intervalo * k));
        }
        return arrayDeRetorno;
    }
}
```

Código – Pacote comunicação: Servidor

```
package comunicacao;
import java.io.*;
import java.net.SocketException;

public class Servidor extends Thread {

        JavaCPN cpnJava, cpnJavaCampo;//= new JavaCPN();
        String deCPN;

        private String hostName;// = "127.0.0.1"; //150.165.61.89 IP com o qual deseja comunicar
        private int port, portRec;// = 9985; //Porta de comunicação com CPNTools

        public String msg;

        public Servidor(){
                cpnJava= new JavaCPN();
                cpnJavaCampo= new JavaCPN();
                hostName = "127.0.0.1"; //150.165.61.89 IP com o qual deseja comunicar
                port = 9985; //Porta de comunicação com CPNTools
                portRec = 9986;
        }
        public JavaCPN getJavaCPN(){
                return cpnJava;
        }

        /**
         *Abre comunicacao entre CPNTools e Xj3D de IP: hostName e porta: port
         */
        public void conectar(){

                try{
                        System.out.println("Abrindo comunicao Socket com IP: "+hostName+" pela porta:
"+port);
                        cpnJava.connect(hostName, port);

                        System.out.println("Conexao Aberta com sucesso");

                }
                catch (IOException e)
                {
                        System.err.println("Não pode fazer conexao");
                }
                try{
                        System.out.println("Server Socket com IP: Porta: "+portRec);
```

```
                        cpnJavaCampo.accept(portRec);
                        System.out.println("Conexao Aberta com sucesso - Server");

                }
                catch (IOException e)
                {
                        System.err.println("Não pode fazer conexao");
                }

        }

        /**
         * Envia a mensagem de qual foi a chave acionada
         * @param chave String com o nome do painel e da chave
         */
        public void envia(String chave){

                System.out.println("Enviando mensagem");
                try{
//                      Converte a mensagem e envia
                        cpnJava.send(EncodeDecode.encode(chave));
                        System.out.println("Mensagem enviada: "+ chave);
                }catch (IOException e) {
                        System.out.println("Conexão interrompida...");
                }

        }
        /**
         * Retorna a mensagem recebi convertida em String
         * @return a String com mensagem recebida
         */
        public String recebe(){
                try{
//                      Recebe a mensagem e converte para String
                        deCPN=EncodeDecode.decodeString(cpnJava.receive());
                        System.out.println("Mensagem recebida: "+deCPN);

                }catch (IOException e) {

                        System.out.println("Conexão interrompida...");
                }

                System.out.println("Recebimento concluido");
                return deCPN;
        }
        public String recebeCampo(){
                try{
//                      Recebe a mensagem e converte para String
                        deCPN=EncodeDecode.decodeString(cpnJavaCampo.receive());
                        System.out.println("Mensagem recebida do Campo: "+deCPN);

                }catch (IOException e) {

                        System.out.println("Conexão interrompida...");
                }

                System.out.println("Recebimento concluido");
                return deCPN;
        }
```

```
}
```

Código – JavaCPN

```
package comunicacao;
import java.util.*;
import java.net.*;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.io.*;

/** <p>
 * <h2>JavaCPN</h2> (otherwise known as Comms/Java) - The java
 * implmentation of a
 * peer entity matching the Comms/Java implementation for Design/CPN. (See
 * http://www.daimi.au.dk/designCPN/libs/commscpn/ for more details.)
 * The purpose of Java/CPN is to allow Java processes to communicate with
 * Design/CPN through Comms/CPN.  The current implementation of Java/CPN
 * is the minimal implementation necessary to enable communication.  It
 * incorporates the equivalent functionality of the Messaging and
 * Communication layers from Comms/CPN.  The Communication Layer
 * functionality from Comms/CPN and TCP/IP is already encapsulated
 * in Socket objects provided by Java.
 * </p>
 * <p>
 * No connection management has been implemented within Java/CPN as
 * this is a minimal implementation, however the important thing is that
 * it implements the same protocol at the Messaging Layer as the peer
 * entity.  Generic <CODE>send</CODE> and <CODE>receive</CODE>
 * functions have been provided at
 * the level of the Messaging Layer, meaning that sequences of bytes are
 * passed to the send method and returned from the receive method.  The
 * <CODE>connect</CODE>, <CODE>accept</CODE>, and <CODE>disconnect</CODE>
 * methods have been provided at
 * the level of the Communication Layer from Comms/CPN.  The
 * deliberate attempt was made to make the interface as close to that of
 * Comms/CPN as possible.
 * </p>
 * <p>
 * Methods external to the Java/CPN class must be used to convert
 * from data (i.e. a string) into a ByteArrayInputStream object, and from
 * a ByteArrayOutputStream object back into data.  This is akin to the
 * encoding and decoding functions passed into the send and receive
 * functions of the Connection Management Layer in Comms/CPN.  They are
 * contained within the <a href="EncodeDecode.html">EncodeDecode</a> class.
 * </p>
 * @author Guy Gallasch
 * @version 0.6
 *
 * Change Log:
 *
 * 10th February 2002:
 * - Improved robustness of Receive method - reads multiple times
 *   from the socket in a loop if not all bytes can be read on the first
 *   attempt.
 *
 */


public class JavaCPN implements JavaCPNInterface
{
```

```
/** Internal reference to the socket being used
 */
   private Socket socket;
/** internal reference to the socket input stream
 */
   private InputStream input;
/** internal reference to the socket output stream
 */
   private OutputStream output;

/** Constructor to create a new JavaCPN object. Simply initialises the internal
 * references. In order to establish a connection either the <CODE>connect
 * </CODE> or the <CODE>accept</CODE> methods need to be called.
 */
   public JavaCPN()
   {
      socket = null;
      input = null;
      output = null;
   }

/** Method to actively establish a connection. It takes a
 * host name and port number as arguments, and attempts to establish a
 * connection as a client to the given port on the given host.  Once the
 * connection has been established (i.e. the socket opened) input and
 * output streams are extracted from the socket to enable the transmission
 * and reception of bytes.
 * @param hostName The host to attempt to connect to
 * @param port The port number to attempt to connect to
 * @throws IOException Thrown when there is a communication error
 * @throws UnknownHostException Thrown when the host name provided as the argument
 * cannot be resolved into an IP address
 */
   public void connect(String hostName, int port) throws IOException, UnknownHostException
   {
      socket = new Socket(InetAddress.getByName(hostName), port);
      input = socket.getInputStream();
      output = socket.getOutputStream();
   }

/** Method to passively open a connection. It takes a port number as an
 * argument and, acting as a server, listens on the given port number for
 * an incoming connection request.  When received, it establishes the
 * connection.  Again, once the connection has been established, input
 * and output streams are extracted from the socket to enable the
 * transmission and reception of bytes.   The method will block until a
 * connection is established.
 * @param port The port number to attempt to connect to
 * @throws IOException Thrown when there is a communication error
 */
   public void accept(int port) throws IOException
   {
      ServerSocket serverSocket = new ServerSocket(port);
      socket = serverSocket.accept();
      input = socket.getInputStream();
      output = socket.getOutputStream();
      //serverSocket.close();
   }
```

```
/** Method used to send a ByteArrayInputStream via an established
 * connection. This method takes a ByteArrayInputStream object
 * as the argument.  The segmentation into packets occurs in this method.
 * Bytes are read from the ByteArrayInputStream object, a maximum
 * of 127 at a time, and a single byte header is added indicating the
 * number of payload bytes (header is 1 to 127) or that there is more data
 * in a following packet (header is 255). The data packets formed are then
 * transmitted to the external process through methods acting on the
 * output stream of the socket.
 * @param sendBytes The byte stream to be sent to the receiving end of the connection
 * @throws SocketException Thrown if there is a problem sending the byte stream
 */
   public void send(ByteArrayInputStream sendBytes) throws SocketException
   {
      // A byte array representing a data packet
      byte[] packet;


      // While there are more than 127 bytes still to send ...
      while (sendBytes.available() > 127)
      {
         // ... create a 128 byte packet, ...
         packet = new byte[128];

         // ... set the header to 255, ...
         packet[0] = (byte)255;

         // ... read 127 bytes from the sequence of bytes to send, ...
         sendBytes.read(packet, 1, 127);

         // ... and send the packet to the external process.
         try
         {
            output.write(packet);
            output.flush();
         }
         catch (IOException e)
         {
            e.printStackTrace();
         }
      }

      // Create a packet for any remaining data
      packet = new byte[sendBytes.available() + 1];

      // Set the header appropriately
      packet[0] = (byte)(sendBytes.available());

      // Read the remaining bytes into the packet
      sendBytes.read(packet, 1, sendBytes.available());

      // Send the packet to the external process
      try
      {
         output.write(packet);
         output.flush();
      }
      catch (IOException e)
      {
```

```
            e.printStackTrace();
        }
    }

/** Method used to receive a ByteArrayOutputStream from an established
 * connection. This method has no arguments.  It uses methods that
 * act on the input stream of the socket to firstly receive a header
 * byte, and then receive the number of payload bytes specified in the
 * header, from the external process.  The payload bytes are stored in a
 * ByteArrayOutputStream object as each segment of payload data is
 * received. This process is repeated until all data has been received for
 * the current transmission.
 * @return sendBytes The byte stream received from the other end of the connection
 * @throws SocketException Thrown if there is a problem sending the byte stream
 */
    public ByteArrayOutputStream receive() throws SocketException
    {

            // The complete sequence of bytes received from the external process
        ByteArrayOutputStream receivedBytes = new ByteArrayOutputStream();

        // The header received from the external process
        int header = -1;

        // The number of payload bytes received from the external process for a packet
        int numberRead = 0;

        // The total number of payload bytes received from the external process for
        // a packet, if not all are received immediately.
        int totalNumberRead = 0;

        // The payload received from the external process for each packet
        byte[] payload;

      //Timeout apos 180s - 3 min caso nao chega a mensagem de resposta
//        socket.setSoTimeout(180000);



        while(true)
        {
            // Read a header byte from the input stream
            try
            {
                header = (int)input.read();

            }
            catch (SocketException e)
            {
                throw new SocketException("Socket closed while blocking to receive header.");
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }

            // If the header shows that the socket has closed ...
            if (header == -1)
                // ... throw a SocketException
                throw new SocketException("Socket closed by external process.");
```

```
        // If the header indicates another packet to follow ...
        else if (header >= 127)
        {
           // ... create 127 bytes of payload storage ...
           payload = new byte[127];
        }

        // ... else create storage of the appropriate size
        else
           payload = new byte[header];

        // Read the payload bytes from the input stream

        // Reset the total bytes received to 0 for this iteration
        totalNumberRead = 0;

        // Loop until all data has been read for this packet.
        while(totalNumberRead < payload.length && numberRead != -1)
        {
           try
           {
              // Try to read all bytes in this packet
              numberRead = input.read(payload,totalNumberRead, payload.length - totalNumberRead);

              // If some bytes were read ...
              if(numberRead != -1)

                 // ... record this many bytes as having been read
                 totalNumberRead = totalNumberRead + numberRead;

           }
           catch (SocketException e)
           {
            throw new SocketException("Socket closed while receiving data.");
           }
           catch (IOException e)
           {
              e.printStackTrace();
           }
        }

        // If not all bytes could be read ...
        if ((totalNumberRead < header || numberRead == -1) && header != 255)

           // ... throw a SocketException
           throw new SocketException("Error receiving data.");

        // Write the payload data to the complete sequence of received bytes
        receivedBytes.write(payload,0,payload.length);

        // If no more bytes to follow, break from the loop.
        if (header <= 127)
           break;
     }

     // Return the received bytes
     return receivedBytes;
  }
```

```
/** Method to disconnect the established connection. This method has no
 * arguments, and returns no value.  It closes the input and output
 * streams from the socket before closing the socket itself.
 * @throws IOException if there is a problem closing the connection
 */
   public void disconnect() throws IOException
   {
      input.close();
      output.close();
      socket.close();
   }
}
```

Código – EncodeDecode

```
package comunicacao;
import java.io.*;

/** Class of static methods to convert objects to ByteStream compatible
 * representations and back again
 * @author Guy Gallasch
 * @version 0.6
 */

public class EncodeDecode
{
/** Method to convert a string to a ByteArrayInputStream
 * @param toConvert The string to convert
 * @return A ByteArrayInputStream representing the string
 */
   public static ByteArrayInputStream encode(String toConvert)
   {
      return new ByteArrayInputStream(toConvert.getBytes());
   }

/** Method to convert a ByteArrayOutputStream to a string
 * @param toConvert A ByteArrayOutputStream to convert to string
 * @return String decoded from the ByteArrayOutputStream
 */
   public static String decodeString(ByteArrayOutputStream toConvert)
   {
      return toConvert.toString();
   }
}
```