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

Dissertação de Mestrado

Mecanismo para Visualização e Comunicação Bidirecional Entre Modelos em Redes de Petri Coloridas e Modelos em Realidade Virtual

Rodrigo Choji de Freitas

Orientadores: Maria de Fátima Queiroz Vieira Turnell, Ph.D.
Angelo Perkusich, D.Sc.

Campina Grande, Paraíba, Brasil
Julho de 2006

Mecanismo para Visualização e Comunicação Bidirecional Entre Modelos em Redes de Petri Coloridas e Modelos em Realidade Virtual

Rodrigo Choji de Freitas

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Área de Concentração
Processamento da Informação

Orientadores
Maria de Fátima Queiroz Vieira Turnell, Ph.D.
Angelo Perkusich, D.Sc.

F866m Freitas, Rodrigo Choji de
Mecanismo para visualizacao e comunicacao bidirecional entre modelos em redes de petri coloridas e modelos em realidade virtual / Rodrigo Choji de Freitas. - Campina Grande, 2006.
103 f. : il.

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

1. Visualizacao de Modelos CPN 2. Realidade Virtual 3. Simulador para Treinamento de Operadores 4. Dissertacao I. Turnell, Maria de Fatima Queiroz Vieira, Dra. II. Perkusich, Angelo, Dr. III. Universidade Federal de Campina Grande - Campina Grande (PB)

CDU 004.5(043)

MECANISMO PARA VISUALIZAÇÃO E COMUNICAÇÃO BIDIRECIONAL ENTRE MODELOS REDES DE PETRI COLORIDAS E MODELOS EM REALIDADE VIRTUAL

RODRIGO CHOJI DE FREITAS

Dissertação Aprovada em 13.07.2006

Maria de Fatima Q Vieira Turnell

MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFCG
Orientador

ANGELO PERKUSICH, D.Sc., UFCG
Orientador

ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFCG
Componente da Banca

JOSÉ SÉRGIO DA ROCHA NETO, D.Sc. UFCG
Componente da Banca

CAMPINA GRANDE - PB
Julho - 2006

Resumo

Neste trabalho é apresentado um mecanismo para integração entre modelos construídos em redes de Petri coloridas (CPN) e suas respectivas representações construídas em realidade virtual. Na integração são utilizadas as ferramentas Design/CPN (modelagem de redes de Petri) e FreeWRL (modelagem de ambientes virtuais) e, para a comunicação entre os modelos é utilizada a biblioteca Comms/CPN e um conjunto de classes Java, desenvolvido para este fim. Este trabalho se insere em um projeto cujo objetivo é a construção de um simulador para treinamento de operadores de um sistema elétrico. Assim, a validação do mecanismo de integração foi realizada a partir de um estudo de caso no qual o modelo CPN de uma subestação elétrica é integrado com a representação virtual de sua interface homem-máquina (IHM) de modo a reproduzir um cenário de erro ocorrido na subestação Os modelos integrados passaram a constituir uma versão inicial do simulador proposto.

Palavras chave: Mecanismo para integração de modelos, interface homem-máquina (IHM), subestação elétrica, IHM painel, IHM sistema supervisório, modelagem em redes de Petri Colorida, modelagem VRML, biblioteca Comms/CPN.

Abstract

This work presents a mechanism for the integration of colored Petri nets models and their representation in virtual reality. For the integration process the following tools are used; Design/CPN (for modeling coloured Petri nets) and FreeWRL (for modeling the virtual environment). For the communication between the models it is used the Comms/CPN library and a set of Java classes developed for this purpose. This work is part of a project that aims to build a simulator to support the training of electrical systems operators. Thus, in order to validate the proposed integration mechanism it was developed a case study in which the model of an electric substation was integrated to its human interface virtual representation, reproducing a scenario in which a human error has occurred. The integrated models constituted an initial version of the proposed simulator.

Key Words: Model integration mechanism, man-machine interface (MMI), electric substation, MMI panel, MMI supervisory system, Coloured Petri nets, VRML modeling, Comms/CPN library.

Dedicatória

Dedico este trabalho ao meu pai que, no meio desta minha caminhada, precisou voar em outros céus.

Mesmo sem sua presença física, sinto sua companhia, em meus pensamentos e coração.

Agradecimentos

Agradeço a Deus, por ter me dado "uma outra" oportunidade.

Aos meus pais Jatir e Naide por não terem medido esforços em proporcionarem a mim e às minhas irmãs a melhor educação, dentro de seu alcance.

Às minhas irmãs Hana e Yuri, por serem exemplos de irmãs, filhas, profissionais, esposas e mulheres.

Ao meu tio Wilson por ter sido meu segundo pai, sempre me ajudando e orientando.

À Fabiana Loiola que de uma forma ou de outra contribuiu para este trabalho.

À Professora Maria de Fátima Queiroz Vieira Turnell por ter me orientado de forma primorosa no desenvolvimento deste trabalho. Sempre disposta a me ajudar e, sobretudo, a me incentivar, principalmente na fase dos ajustes da dissertação para a "nova" defesa.

Ao Professor Angelo Perkusich pelas orientações e, principalmente, por ter me indicado o tema de pesquisa deste trabalho.

Aos Professores Antonio Marcus Nogueira de Lima e José Sérgio da Rocha Neto por apontarem os problemas da primeira versão desta dissertação e por me motivarem a encontrar suas soluções, apresentadas nesta versão.

Ao Professor Antonio Cauper Filho, da Universidade do Estado do Amazonas, por ter sido muito mais que um professor, mas um amigo que me ofereceu preciosos ensinamentos e lições de vida.

Aos demais Professores da UFCG, que nos honraram com a transmissão de preciosos conhecimentos, ao longo do curso.

Aos Professores Antenor Ferreira Filho, Anízio de Araújo Cavalcante, Alexandre Guedes Guimarães e Odwald Schereder, da Universidade do Estado do Amazonas, por terem sido grandes incentivadores e colaboradores.

Ao grande amigo Christophe Saint-Christie de Lima Xavier, por sua hospitalidade e colaboração para o desenvolvimento desta dissertação e dos artigos dela gerados.

Ao grande amigo Jean Caminha, pelas palavras de ânimo nos momentos difíceis.

Aos amigos Roberto Higino Pereira da Silva e Hernan Santiago Marinho pela colaboração, palavras de incentivo e boas conversas, no convívio do laboratório de telecomunicações da EST-UEA.

Aos amigos do Laboratório de Interface Homem Máquina da UFCG, Alexandre Scaico, Cláudia Verônica Serey Guererro, Daniel Scherer, José Alves do Nascimento Neto e José Pedrosa Barreto Neto que tiveram participação fundamental e decisiva para a realização deste trabalho.

Ao Leandro Dias da Silva, que foi um "ponto de referência" para o desenvolvimento deste trabalho. Sempre disposto a colaborar e tirar minhas dúvidas.

Aos amigos Willen Noel, Iramara Marinho e William Costa, pelas conversas no PCV.

Ao Emerson Ferreira de Araújo Lima, pelas valiosas ajudas em redes de Petri e em VRML.

Ao Patryckson Marinho Santos, pela amizade e gentileza em me hospedar em sua casa, numa de minhas vindas à Campina Grande.

À Dona Graça pela dedicação, atenção e, sobretudo, pelos excelentes lanches que providenciava para os alunos.

A todos os colegas da PRODAM, especialmente aos amigos da Supervisão de Manutenção, Adílio Moreira, Ari Libório, Claudemir Ivan e Pedro "Presidente", que sempre foram compreensíveis e generosos ao assumirem minhas responsabilidades do setor, possibilitando, assim, minhas viagens até Campina Grande.

A todos os colegas da turma do mestrado que, agora, os considero amigos.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

A	- conjunto de arcos
BRITNeY	- *Basic Real-Time Interactive Tools for Net-based Animation*
C	- função de cores
E	- função que mapeia cada arco em uma expressão
F	- conjunto de arcos
F	- função nodal
G	- função guarda
HTML	- *Hypertext Markup Language*
I	- função de inicialização
L	- conjunto de lugares
LIHM	- Laboratório de Interface Homem-Máquina
M_0	- marcação inicial
MVC	- *Model-View-Controller*
P	- conjunto finito de lugares
PNVis	- Petri *Net Kernell Visualization*
RPC	- Rede de Petri Colorida
S	- conjunto de cores
SCADA	- *Supervisory Control and Data Acquisition*
T	- conjunto finito de transições
T	- conjunto de transições
VRML	- *Virtual Reality Modeling Language*
W	- função peso

Capítulo 1 – Introdução

Com o avanço da tecnologia e com a exigência de se baratear custos de projetos e produção de sistemas, surgiu a necessidade do desenvolvimento de protótipos, os quais na fase inicial de projeto se baseiam em modelos. Dessa forma, na atualidade, diversos sistemas (automobilístico, gerenciamento de informações, gerenciamento de documentos, manufatura, gerenciamento de subestações elétricas, tráfego aéreo) são concebidos a partir de modelos.

Segundo [BPF99], um modelo é a representação, geralmente, em termos matemáticos, das principais características de um objeto ou sistema. Por meio dele é possível analisar, controlar e verificar o comportamento de um sistema real. Entretanto, para que esses itens sejam contemplados, é conveniente a adoção de métodos, a fim de garantir a eficiência do modelo e a redução de esforços para sua construção. Para [CW96], a utilização de métodos formais na modelagem de sistemas, tanto de software quanto de hardware, traz comprovadamente, vantagens no seu desenvolvimento tais como: possibilidade de realizar análise, verificação, simulação e validação.

As redes de Petri têm se destacado, amplamente, como ferramenta de modelagem, por consistirem de um formalismo matemático com representação gráfica [Mur89]. Apresenta-se, ainda, como uma de suas principais características, a facilidade para modelar situações de concorrência, paralelismo e sincronização entre processos.

Outro aspecto de um modelo refere-se à possibilidade de visualização do comportamento de um sistema. Todavia, para tanto, esta observação deve ser realizada de forma intuitiva e natural. A modelagem em redes de Petri coloridas possibilita a visualização do comportamento de sistemas, entretanto exige do usuário final o domínio de

suas ferramentas e do formalismo. Este ônus pode ser minimizado através da abstração do formalismo por meio de uma representação mais intuitiva.

A equipe do Laboratório de Interface Homem-Máquina (LIHM) da Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande norteia suas atividades na busca por características mais ergonômicas na interação entre sistemas e o seu usuário. Nos trabalhos de [STP01] e [Nasc04] são construídos modelos da interface com o operador de uma subestação elétrica. Ainda neste contexto, está sendo construído um ambiente para o treinamento de operadores destas subestações, o qual consiste em um simulador cujo motor de simulação é baseado em modelos construídos em redes de Petri coloridas. Os modelos representam o comportamento dos recursos disponíveis para operação da subestação elétrica: (i) painel, onde são modelados dispositivos de interação acessíveis ao operador tais como botões e chaves; (ii) supervisório, onde é modelada a interface com um software do tipo SCADA e; (iii) planta, onde são modelados os dispositivos encontrados no campo a exemplo de disjuntores[1] e chaves seccionadoras[2]. O projeto do simulador exigiu um meio mais natural para a visualização do comportamento dos modelos [FTPX06].

O ambiente de treinamento atende a uma demanda da indústria por um ambiente alternativo para o treinamento de seus operadores além dos equipamentos em operação. Originalmente a atividade de treinamento era realizada diretamente com os equipamentos da operação do sistema, visto que as exigências por qualidade do fornecimento e a normatização do setor assim o permitiam. Por exemplo, uma cidade poderia permanecer sem fornecimento de energia elétrica por alguns minutos, enquanto a subestação era configurada para uma operação de treinamento.

Atualmente, a maior parte do treinamento dos operadores se limita à indicação da seqüência de manobras nos painéis do equipamento, não havendo a interação propriamente dita. Esta restrição resulta do fato de que uma operação errada, executada pelo operador, pode levar a uma interrupção do fornecimento de energia elétrica causando prejuízos

[1] Chave que causa a interrupção da continuidade de atividade um circuito elétrico energizado, quando houver sobrecorrente.
[2] Chave que possibilita o isolamento de uma parte do circuito elétrico.

materiais aos beneficiados do serviço além de multas e perda de credibilidade para o fornecedor.

O simulador proporcionará ao operador uma forma de treinamento alternativa, na qual a ocorrência de erros não afeta a operação do sistema. Com o apoio do simulador, a operação do sistema pode ser melhorada uma vez que este ambiente permite avaliar as falhas humanas contribuindo assim para a ergonomia da interface de operação. Por exemplo: indicar um modo mais eficiente e prático para a realização de determinada tarefa; evitar a sobrecarga cognitiva, pois a atividade de tomada de decisões poderá ser exercitada diversas vezes; treinar os operadores para lidarem com situações anormais; treinar os operadores em manobras, cujo tempo para execução é crítico.

Para que o simulador possa contribuir efetivamente para o treinamento dos operadores, é necessário que a visualização do comportamento das redes de Petri não represente um obstáculo (o evento ocorrido deve ficar evidente ao operador, por exemplo, a abertura ou fechamento de uma chave) e que seja realística (a representação dos dispositivos deve ser a mais fiel possível aos dispositivos reais).

Como solução para este problema, é proposto um mecanismo para visualizar o resultado das operações em um ambiente tridimensional, de forma que o operador possa interagir diretamente a partir de um modelo virtual com o modelo formal. O mundo virtual é criado utilizando-se a linguagem VRML, o comportamento resultante das operações é obtido utilizando modelos de redes de Petri e a conexão entre os modelos é feita a partir de uma biblioteca de comunicação e de um conjunto de funções.

1.1 Objetivos da Dissertação

O objetivo principal deste trabalho é prover um mecanismo de integração que possibilita visualizar o comportamento de um modelo rede de Petri colorida em um modelo virtual tridimensional, a partir do estabelecimento de uma comunicação bidirecional para troca de mensagens entre o modelo formal e o modelo de visualização. Os objetivos específicos são:

- aplicação do mecanismo proposto na construção do simulador para treinamento de operadores de subestações elétricas;
- estudo e desenvolvimento de técnicas para comunicação entre modelos formais e de visualização, bem como a representação de ambientes virtuais.

1.2 Escopo e Relevância

Neste este trabalho é apresentada a visualização em ambiente virtual de um conjunto dos recursos disponíveis ao operador para operação da planta tanto nos painéis de comando quanto através das telas de um programa supervisório.

Os recursos modelados foram selecionados para representar um cenário de erro ocorrido durante uma manobra em uma subestação. Neste cenário um operador foi solicitado a realizar o comando de abertura de uma chave A, mas equivocado efetuou-o sobre uma chave B. Percebendo o erro, imediatamente fechou-a e em seguida acionou a chave C. Percebendo a segunda falha, pediu permissão para comandar novamente a chave C para fechá-la. Após a ocorrência do erro a configuração do sistema e das chaves retornou ao estado precedente.

A versão do simulador construída a partir do trabalho desenvolvido nesta dissertação contempla os dispositivos: disjuntores e chaves seccionadoras; que estiveram envolvidos no referido cenário de erro. O comportamento desses dispositivos corresponde a modelos em redes de Petri coloridas, disponíveis na biblioteca construída por Alves [Nasc04]. Por sua vez, a IHM para painéis [Nasc05]; a IHM para o supervisório [Tur01] e o comportamento da planta [Bar05] foram todos modelados em CPN (*Colored* Petri *Net*) e posteriormente integrados. Para cada um dos modelos da IHM, painéis e supervisório, foi construído um modelo VRML responsável pela animação e visualização dos comportamentos das redes.

A relevância deste trabalho está na apresentação de uma solução que complemente um simulador capaz de possibilitar o treinamento de operadores de subestações elétricas, sem que o conhecimento em redes de Petri seja indispensável. Além disso, é definido um

mecanismo de integração que pode ser utilizado para integração de outros contextos de simulação.

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma:

No Capítulo 2 é abordado o estado da arte, fazendo-se uma revisão bibliográfica, a fim de avaliar outras soluções existentes e obter embasamento teórico para elaboração do mecanismo de integração responsável pela visualização e comunicação de modelos. Ainda neste capítulo, são apresentadas teorias e características sobre redes de Petri e ambientes virtuais.

No Capítulo 3 é definido o mecanismo de visualização e comunicação, a partir da apresentação de suas camadas. Além disso a biblioteca Comms/CPN é detalhada.

No Capítulo 4 é apresentado um estudo de caso que valida cada camada do mecanismo de integração.

Finalmente, no Capítulo 5 é realizada uma discussão acerca das conclusões obtidas, bem como são apresentadas sugestões para trabalhos futuros.

Capítulo 2 – Fundamentação Teórica

Este capítulo tem por objetivo apresentar os conhecimentos necessários para a elaboração deste trabalho. Os assuntos aqui descritos serviram de alicerce para a idealização e implementação do mecanismo proposto. Para tanto, realizou-se uma pesquisa bibliográfica sobre os mesmos, além da identificação de trabalhos correlatos.

2.1 Soluções Existentes para Visualização de Modelos

Os modelos formais provaram sua utilidade na modelagem e entendimento de sistemas complexos [BJ04, KJ04]. A partir deles, é possível observar o comportamento existente ou identificar necessidades exigidas para atingi-lo. Entretanto, com a utilização de um modelo formal, como redes de Petri coloridas, o entendimento pleno do modelo é alcançado apenas por pessoas que possuam conhecimento do formalismo. Assim, erros de interpretação estão propícios a ocorrer, comprometendo a validação.

Na busca de minimizar esta dificuldade de compreensão do modelo formal, soluções foram propostas. Nesta seção, serão discutidas algumas delas.

Em [ExSp1] é apresentada a ferramenta para modelagem de processos discretos, ExSpect. Um modelo criado nesta ferramenta é descrito em termos de um conjunto de subtarefas que se comunicam por meio de mensagens. Adicionalmente, em [ExSp2] é descrito que um modelo concebido na ferramenta ExSpect consiste em uma rede de Petri Colorida Hierárquica [Jen97].

No ExSpect, as transições são representadas por dois tipos de subtarefas: um denominado *Processor*, responsável pelo processamento lógico e outro denominado

System, cuja finalidade é empacotar uma outra rede de Petri. Existe ainda a subtarefa *Channel* responsável por conter as fichas que serão consumidas pelo *Processor*.

Apesar de possuir um ambiente de modelagem bastante prático, os aspectos de formalismo e visualização de comportamento são muito acoplados, ou seja, é necessário conhecer a teoria de redes de Petri para o entendimento pleno da simulação do modelo, que é feita por meio do jogo de fichas. Além disso, os componentes da simulação são os mesmos para qualquer domínio de problema, não havendo a possibilidade de agregação de novos objetos, o que dificulta uma visualização mais realística do comportamento.

Em [RS95], é apresentada a ferramenta Mimic/CPN que possibilita a manipulação de objetos referentes a simulações de modelos redes de Petri, no ambiente do Design/CPN[3]. Nestas simulações, o estado do sistema descrito pela rede pode ser observado, bem como a realização de interação com os objetos pode ser feita pelo usuário, tornando o entendimento do modelo mais intuitivo.

A palavra *mimic* foi herdada dos antigos sistemas de segurança que consistiam de uma maquete (*mimic-board*), cuja finalidade era exibir toda a estrutura física de um prédio, por exemplo, que era supervisionada por equipamentos de segurança. No Mimic/CPN, a *mimic-board* assume a função de meio para o controle da simulação. A partir de uma figura de fundo, regiões desta (sensíveis ao clique do *mouse*) são definidas para que o usuário possa interagir com o modelo.

Um modelo Mimic/CPN é composto dos seguintes objetos:

- *Node* – é a *mimic-board*, ou seja, uma figura de fundo (escaneada);
- *Region* – é o ponto do modelo sensível ao clique do *mouse*;
- *Mimic Name* – é uma palavra única que identifica os objetos *Region* e *Node*.

Embora o Mimic/CPN também seja uma ferramenta que incorpore funcionalidades para a visualização do comportamento de redes de Petri coloridas, alguns pontos devem ser destacados. O acoplamento entre o modelo formal e o modelo de visualização ainda é grande, visto que ambos convivem em um mesmo ambiente (Design/CPN). Outro aspecto,

[3] Ferramenta para análise, modelagem, verificação e simulação de redes de Petri coloridas.

é que a movimentação e animação dos objetos são dadas apenas pelos recursos disponíveis no Design/CPN.

No artigo de Westergaard e Lassen [WL05], é apresentada a ferramenta de visualização BRITNeY (*Basic Real-Time Interactive Tools for Net-based Animation*) que utiliza uma camada de animação para redes de Petri coloridas, cujo objetivo principal é simular seu comportamento.

Esta ferramenta adota o padrão de projeto orientado a objetos, denominado MVC (*Model-View-Controller*). A idéia é utilizar o modelo rede de Petri colorida para modelar o estado e o comportamento do sistema (*Model* e *Controller*), já a visualização (*View*) fica por conta da animação provida por BRITNeY.

Embora BRITNeY promova uma interação natural do usuário para com o modelo formal, possibilitando até mesmo que o primeiro não possua conhecimentos de redes de Petri, algumas limitações são apresentadas. Ainda que haja uma efetiva animação do comportamento da rede, esta é dada por meio de figuras estáticas, fazendo com que a animação se torne deficiente no aspecto de realismo. Para o efeito de animação seja atingido, é necessária a utilização de uma grande quantidade de imagens seqüenciadas. Além disso, não há interação do usuário para com os objetos da simulação, isto é, a comunicação ocorre somente no sentido do modelo rede de Petri para a animação.

Em [KP04], é apresentada a ferramenta PNVis para visualização, em três dimensões, do comportamento de sistemas modelados com redes de Petri. O método consiste na associação de objetos físicos a fichas de lugares da rede.

O objetivo principal desta ferramenta é permitir que usuários possam visualizar o comportamento produzido por uma rede de Petri, a partir de objetos tridimensionais.

PNVis é uma ferramenta que separa o formalismo de uma rede de Petri, da visualização do seu comportamento. Além disso, define um ambiente realístico que permite ao usuário um entendimento privilegiado do sistema modelado. Entretanto, esta ferramenta (versão 0.8.0) é capaz de modelar apenas redes de Petri baixo-nível.

Em [SP03], é apresentada uma técnica para visualizar o comportamento de um modelo rede de Petri colorida em ambiente virtual. A proposta é semelhante à descrita em

[KP04]. Entretanto, alguns pontos adicionais são apresentados nesta. O primeiro é que esta técnica é mais abrangente, pois permite a simulação do comportamento de uma rede de Petri colorida. Além disso, a comunicação entre os modelos é realizada por meio de uma biblioteca de funções.

Embora esta simulação permita que o usuário visualize o comportamento da rede de forma detalhada, a comunicação entre os modelos é feita apenas em uma direção, ou seja, apenas o modelo formal realimenta o modelo virtual, não ocorrendo o contrário.

Nesta dissertação, a partir da proposta de [KP04] combinada com a de [SP03], é definido e desenvolvido um mecanismo para visualização tridimensional do comportamento de uma rede de Petri colorida, que permita a troca de mensagens entre os modelos formal e virtual, ou seja, um evento ocorrido na rede de Petri colorida, terá seu comportamento visualizado no mundo VRML, assim como uma interação do usuário com um objeto do ambiente virtual, será refletida na rede. Na Tabela 2.1 é apresentado um comparativo entre as ferramentas e/ou técnicas pesquisadas bem como a própria solução proposta nesta dissertação, face aos seguintes critérios: (i) **nível de realismo**: indica se o objeto modelado apresenta características (forma, cor, tamanho) que remetam ao correspondente objeto real; (ii) **animação**: indica se a animação do objeto modelado ocorre de maneira compatível com a cinemática do objeto real; (iii) **compatibilidade com redes de Petri coloridas**: indica se a ferramenta ou técnica é compatível com este método formal; (iv) **comunicação bidirecional**: indica se a ferramenta promove comunicação do modelo formal para o modelo de visualização e vice-versa. Na última linha da Tabela 2.1 são apresentados os critérios satisfeitos pela solução proposta neste trabalho de dissertação.

Critério / Técnica	Nível de Realismo	Recursos para Animação	Compatível com CPN	Comunicação Bidirecional
ExSpect	Baixo	Não	Sim	Não aplicável
Mimic/CPN	Baixo	Limitados	Sim	Não aplicável
BRITNeY	Baixo	Limitados	Sim	Não
PNVis	Elevado	Sim	Não	Sim
Técnica proposta por Silva e Perkusich	Elevado	Sim	Sim	Não
Solução proposta	**Elevado**	**Sim**	**Sim**	**Sim**

Tabela 2.1 – Critérios de comunicação e visualização satisfeitos por cada ferramenta e/ou técnica

2.2 Redes de Petri

Através da análise do modelo, um sistema real pode ser estudado sem o perigo, custo ou inconveniência da manipulação de seus elementos. Dentre as técnicas formais para modelar e analisar sistemas, citam-se: modelos de fila, álgebra de processos, lógica temporal e redes de Petri.

As redes de Petri são uma ferramenta gráfica e matemática para modelar os mais diversos sistemas. Desde modelos triviais, passando por modelos de sistemas de manufatura, modelos de conexões em rede e chegando aos modelos em que a exigência de paralelismo entre os processos é fator crítico.

Segundo [KP04], as principais vantagens das redes de Petri são sua notação gráfica, sua semântica simples e a rica teoria para análise de comportamento de modelos. Todavia, ao mesmo tempo em que as redes de Petri apresentam todas essas características

facilitadoras para uma boa representação de um modelo, há o surgimento de determinados inconvenientes no que diz respeito à visualização dos comportamentos de modelagens com graus de complexidades maiores. Em vista disso, foram e estão sendo propostas, maneiras para representação dos comportamentos de redes de Petri, como se pôde observar na seção anterior.

2.2.1 Definições Básicas de Redes de Petri Lugar-Transição

De acordo com [Jen97], uma Rede de Petri é uma 5-upla $N=(P,T,F,W, M_0)$, onde:

1. $P = \{p_1, p_2, ..., p_m\}$ *é um conjunto finito de lugares,*
2. $T = \{t_1, t_2, ..., t_n\}$ *é um conjunto finito de transições,*
3. $F \subseteq (P \times T) \cup (T \times P)$ *é um conjunto de arcos,*
4. $W: F \rightarrow \mathbb{N}^*$ *é uma função peso,*
5. $M_0: P \rightarrow \mathbb{N}$ *é a marcação inicial.*

$P \cap T = \phi$ *e* $P \cup T \neq \phi$

Segundo [Mur89], os símbolos utilizados para representar os lugares e transições são círculos e retângulos, respectivamente. Os arcos são rotulados com um peso (inteiros positivos) *k* que pode ser interpretado como um conjunto de *k* arcos paralelos, as fichas são os círculos preenchidos e sua distribuição no instante inicial é chamada de marcação inicial, os arcos podem ligar lugares a transições ou transições a lugares, como ilustra a Figura 2.1.

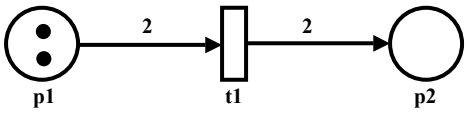

Figura 2.1 – Exemplo de rede de Petri

2.2.2 Dinâmica de uma Rede de Petri Lugar-Transição

Em [BPF99], para simular a dinâmica de um sistema, a marcação (ou estado) de uma Rede de *Petri* evolui de acordo com a regra de disparo a seguir:

i) (Pré-condição): Uma transição *t* está habilitada (ou é *gatilhável*) se cada lugar de entrada *p* de *t* contiver pelo menos *w(p,t)* fichas, onde *w(p,t)* é o peso do arco de *p* para *t*.

ii) Uma transição habilitada pode ou não disparar (ou *gatilhar*).

iii) (Pós-Condição): Ao disparar *t*, *w(p,t)* fichas são removidas de cada lugar de entrada *p* de *t* e *w(t,p)* são acrescentadas a cada saída *p* de *t*, onde *w(t,p)* é o peso do arco de *t* para *p*.

Na Figura 2.2, as fichas do lugar p1 (Figura 2.1) são consumidas de acordo com o peso do arco que liga p1 a t1 e fichas são depositadas em p2, de acordo com o peso do arco que liga t1 a p2.

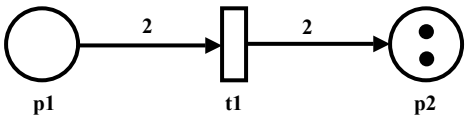

Figura 2.2 – Dinâmica de uma rede de Petri, a partir do disparo da transição t1

2.2.3 Redes de Petri Coloridas

As redes de Petri lugar-transição, como observado, são aquelas cujo significado de suas marcas não são diferenciáveis, a não ser pela estrutura da rede à qual estão associadas. Elas são restritivas do ponto de vista de modelagem, pois apenas permitem a existência de marcas em cada elemento da rede, sem a incorporação de semântica, forçando, desta forma, a representação das nuances de um sistema modelado, por meio dos elementos estruturais da rede [Mar05].

As redes de Petri de alto nível ou coloridas já incorporam às marcas, alguma semântica, viabilizando sua diferenciação, que pode ir desde a atribuição de valores ou cores (tipos das variáveis) às marcas, até a adoção de noções de tipos de dados abstratos, conferindo-lhes um grande poder de expressão [Mar05].

Ainda segundo [Mar05], uma rede de Petri colorida é uma 9-upla *RPC = (S, L, T, A, N, C, G, E, I*) conforme as definições a seguir:

- *S* é o conjunto de cores que determina os tipos, as operações e as funções que podem ser usadas nas inscrições da rede;
- *L* é o conjunto de lugares;
- *T* é o conjunto de transições;
- *A* é o conjunto de arcos, os quais devem ser finitos e disjuntos dois a dois;
- *N* é a função nodal que mapeia cada arco em um par, cujo primeiro elemento é o nó fonte e o segundo elemento é o nó destino;
- *C* é a função de cores que mapeia cada lugar em um conjunto de cores;
- *G* é a função guarda que mapeia cada transição em uma expressão booleana;
- *E* é a função que mapeia cada arco em uma expressão cujo resultado é multiconjunto sobre o conjunto de cores associado ao correspondente;
- *I* é a função de inicialização que mapeia cada lugar em uma expressão, cujo resultado é um multiconjunto sobre o conjunto de cores dos lugares.

Na Figura 2.3, é observada uma rede de Petri colorida em seu estado original com três lugares, duas transições e 4 arcos.

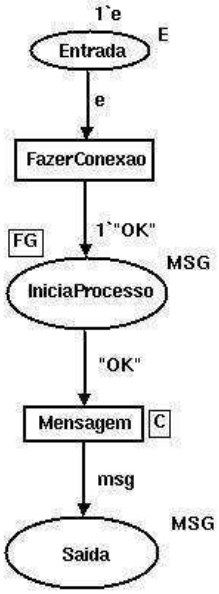

Figura 2.3 – Rede de Petri colorida (marcação inicial)

O lugar por nome “Entrada” exibe a inscrição “E” que identifica o conjunto de cores do mesmo e a inscrição “1’e” que identifica sua marcação inicial. A transição “FazerConexao” é habilitada toda vez que uma ficha *e* é depositada no lugar *Entrada*. O lugar de fusão “IniciaProcesso” possui interconexão com outros lugares. Seguindo o fluxo, encontra-se a transição “Mensagem” para recebimento de uma mensagem. Finalizando o fluxo, tem-se o lugar “Saída” com a cor “MSG” que representa o conjunto mensagens recebidas pela rede.

Na Figura 2.4, é vista a transição “FazerConexao” habilitada, pois há fichas suficientes e de mesmo tipo para que esta operação seja executada.

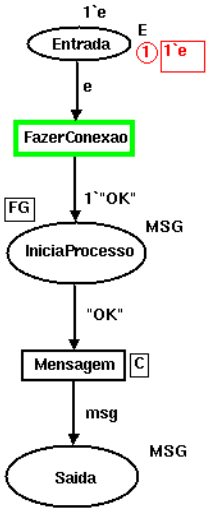

Figura 2.4 – Transição “FazerConexao” habilitada

Assim que a conexão é feita, ou seja, no momento em que a transição “FazerConexao” é disparada, uma ficha “OK” é depositada no lugar “IniciaProcesso”. Imediatamente, a transição “Mensagem” é habilitada, conforme Figura 2.5.

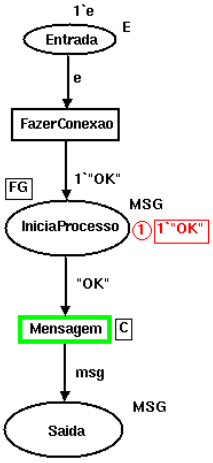

Figura 2.5 – Transição *Mensagem* habilitada

Finalmente, a transição "Mensagem" é disparada, executando a função para receber uma mensagem. Então, uma ficha com a mensagem recebida, é depositada no lugar "Saída", conforme Figura 2.6.

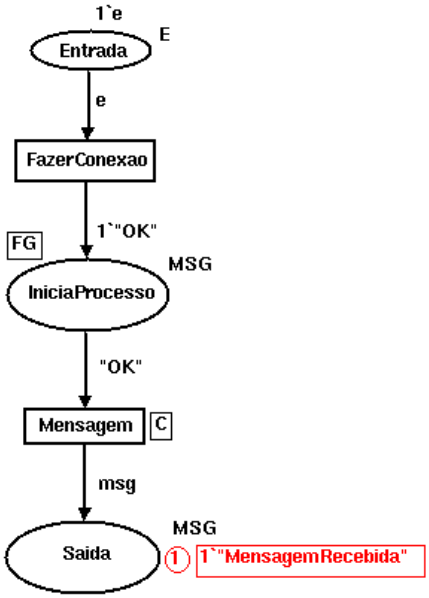

Figura 2.6 – Uma ficha "MensagemRecebida" é depositada no lugar "Saída"

As cores e funções utilizadas nos exemplos acima, precisam ser definidas. Esta definição é chamada de nó declaração, apresentada na Figura 2.7.

```
color E = with e;
var c: E;

color MSG = string;
var msg: MSG;

fun recebe_mensagem() = ConnManagementLayer.receive("Conexao",stringDecode);
```

Figura 2.7 – Nó Declaração

2.2.4 Design/CPN

Como já apresentado, a técnica de redes de Petri coloridas permite que características de sistemas reais sejam modeladas. Com isso, é possível fazer análises, verificações, validações, antes que a execução do sistema proposto seja efetivada, reduzindo-se, assim, o risco de desperdício de recursos.

Entretanto, um modelo pode conter tantos detalhes que seu entendimento ficará dificultado, até mesmo às pessoas que o conceberam. A partir daí, é necessário o suporte de ferramentas computacionais para tratar desta dificuldade.

O Design/CPN é uma ferramenta computacional que tem por objetivo modelar e simular redes de Petri coloridas. Suas características principais são [Des93]:

- Editor para criação e manipulação de Redes de *Petri* Coloridas;
- Verificação de erros de sintaxe para validação dos modelos;
- Simulador para execução dos modelos;
- Monitoramento interativo e depurador;
- Hierarquização de modelos.

Estas facilidades permitem com que os modelos redes de Petri coloridas possam ser criados, modificados, organizados, executados, depurados, examinados e validados de forma clara, rápida e objetiva.

Pelas características citadas, por se tratar de uma ferramenta livre, por já estar consolidada nos meios acadêmico e industrial foi que se elegeu o Design/CPN como ferramenta para concepção dos modelos utilizados neste trabalho.

2.3 Realidade Virtual

O real e o imaginário sempre fizeram parte da vida das pessoas. Até alguns anos atrás, a única maneira de retratar o imaginário era descrevendo-o verbalmente ou, quando possível, desenhando-o ou representando-o de maneira restrita [KT04].

Com o advento da realidade virtual e o avanço dos recursos computacionais, a representação do imaginário e a reprodução do real tornaram-se mais fáceis de serem obtidas. Foram disponibilizadas interfaces mais intuitivas e rompidos os limites existentes, como a barreira da tela do monitor, permitindo a atuação do usuário no espaço tridimensional [KT04]. A atuação do usuário sobre representações da aplicação, como menus e botões, foi substituída por ações diretas sobre elementos tridimensionais conhecidos, como o abrir de uma porta, o acionar de uma alavanca, o puxar de uma gaveta, o girar de um botão, o fechar ou o abrir de um disjuntor. Além disso, no ambiente virtual, os sentidos e as capacidades das pessoas podem ser ampliados em intensidade, no tempo e no espaço. É possível, assim, ser tão grande ou tão pequeno quanto se queira, viajando a velocidades superiores à da luz e aplicando forças descomunais.

O termo realidade virtual refere-se a uma experiência imersiva e interativa baseada em imagens gráficas 3D geradas em tempo-real por computador [PT95], em que a primeira tem por objetivo reter a atenção do usuário e a segunda objetiva uma representação realística e intensa do objeto real.

2.3.1 Categorias de Realidade Virtual

As aplicações de realidade virtual são categorizadas quanto aos seus níveis de imersão e interatividade.

i) Realidade Virtual por Simulação

Foi o primeiro sistema de realidade virtual, originada a partir dos simuladores de vôo desenvolvidos pelos militares americanos, depois da Segunda Guerra Mundial [Jac94].

Uma aplicação deste tipo, basicamente, imita o interior de um carro ou avião, colocando o participante dentro de uma cabine com controles, onde telas de vídeo e monitores apresentam um mundo virtual que reage aos comandos do usuário [Jac94].

ii) Realidade Virtual por Telepresença

Caracteriza uma situação, onde uma pessoa está objetivamente presente num ambiente real que está separado fisicamente da mesma, no espaço. A telepresença, que é implementada por mecanismos de teleoperação, consiste de um usuário, uma interface homem-máquina, um telerobô e um ambiente remoto [Sch95].

iii) Realidade Virtual por Projeção

Também conhecida como realidade artificial, esta categoria foi criada nos anos 70 por Myron Krueger. Nela, o usuário está fora do mundo virtual, mas pode se comunicar com personagens ou objetos dentro dele [Jac94].

iv) Realidade Virtual Aumentada (Realçada)

É a combinação da visão do ambiente real com a do ambiente virtual. Essa categoria é obtida mesclando-se sistemas de telepresença e realidade virtual por projeção [Isd02].

Para aplicações desta categoria, é necessária a utilização de óculos 3D ou capacete com visor semitransparente, de forma que a visão do ambiente real possa ser sobreposta à informação do ambiente virtual.

2.3.2 Virtual Reality Modeling Language (VRML)

VRML é uma linguagem para modelagem de realidade virtual, usada para criar ambientes tridimensionais. Trata-se de um formato de arquivo objeto 3D, análogo ao HTML [MB98]. Além disso, é independente de plataforma e que permite a criação de ambientes virtuais por onde é possível navegar, visualizar objetos por ângulos diferentes e até interagir com eles. Ainda hoje, a VRML é o padrão para desenvolvimento de aplicações de realidade virtual multi-usuário [Net02].

Por ser independente de plataforma, uma aplicação VRML roda em qualquer máquina que tenha um navegador para *Internet*, com o componente para visualização de mundos virtuais ou em navegadores exclusivos para ambientes tridimensionais [MB98].

A linguagem VRML foi escolhida para a modelagem do mundo tridimensional, proposto nesse trabalho, em virtude de características como *script* executável, independência de plataforma, possibilidade de criação de ambiente multi-usuário, transmissão de dados e, sobretudo, por ser considerada uma linguagem padrão para modelagem de mundos virtuais.

2.3.3 FreeWRL

FreeWRL é um visualizador VRML, escrito em Perl[4]. Permite a exibição de mundos virtuais escritos em VRML. É um software livre que utiliza a biblioteca MESA[5] para renderização OpenGL[6], roda sobre o sistema operacional Linux e utliza XFree86 para a resolução dos objetos.

Para composição deste trabalho, foi escolhido o referido visualizador por se tratar de um software livre, com boa navegabilidade, apresentando teclas de atalho, além de rodar em distribuições Linux.

---

[4] *PERL* (*Practical Extraction Report Language*), é uma linguagem de programação interpretada. Não é necessário que a aplicação desenvolvida seja compilada.
[5] É uma biblioteca gráfica 3D.
[6] É uma biblioteca de rotinas gráficas para trabalhar em duas e três dimensões.

2.3.4 Outros Aplicativos para Desenvolvimento de Realidade Virtual

Além do conjunto FreeWRL-VRML, que compõe uma estrutura para criação de ambientes virtuais, há no mercado, outros aplicativos para a criação de mundos virtuais, dos mais variados preços e funcionalidades [Net02]. Na Tabela 2.1, apresentam-se alguns dos principais.

Aplicativo	Descrição
Java 3D	É uma interface para criar programas com gráficos tridimensionais, permitindo a modelagem de ambientes virtuais complexos com personagens 3D que interagem entre si e/ou com o usuário.
X3D	É uma especificação que se estende às potencialidades de VRML 2.0 para permitir a utilização de gráficos 3D. A sintaxe XML foi adotada como padrão para o X3D, ao invés da sintaxe VRML, a fim de resolver vários problemas reais como reusabilidade, integração da página e integração com gerações futuras da *web*.
Virtual Reality Studio	Permite modelar e visualizar cenários 3D e interagir com objetos animados nos cenários. O software suporta placas de som para produção de som interativo e tem baixo custo, entretanto gera imagens de baixa resolução.
Vream	Oferece uma boa interface gráfica com o usuário e permite criar, alterar e apagar objetos tridimensionais, além de suportar som interativo e imagens virtuais em tempo real usando um mouse ou *joystick*, bem como dispositivos de entrada e saída de dados.
3Dwebmaster	É uma ferramenta para projetar ambientes interativos 3D que pode ser utilizada para aplicações comerciais, de

	entretenimento, ou apenas para fins ilustrativos. O software permite integração de páginas *HTML* com uma interface Java, possibilitando duas vias de comunicação para o desenvolvedor entre uma página 3D e um Applet Java ou JavaScript.
VR Jugller	Trata-se de um ambiente de desenvolvimento *Open Source*. Suas principais características são: possuir um sistema de micro-kernel extensivo modular, ser orientado a objetos (C++ e Java) e permitir configuração e reconfiguração em tempo de execução.

Tabela 2.2 – Aplicativos para criação de ambientes virtuais.

2.4 Modelagem de Mundos Virtuais

A modelagem de mundos virtuais é uma importante etapa para o desenvolvimento de uma aplicação em realidade virtual, definindo as características dos objetos como: forma, aparência, movimento, restrições e mapeamento de dispositivos de E/S. Para isto, os sistemas de desenvolvimento de realidade virtual levam em conta os diversos aspectos de modelagem, mapeamento e simulação [BC94].

2.4.1 Modelagem Geométrica

A modelagem geométrica abrange a descrição da forma dos objetos virtuais através de polígonos, triângulos ou vértices e sua aparência, usando textura, reflexão da superfície, cores. A forma poligonal dos objetos pode ser criada, usando-se bibliotecas gráficas ou modelos prontos de bancos de dados ou digitalizadores tridimensionais. Os objetos também

podem ser criados por programas CAD[7], como AutoCAD ou 3-D Studio, ou com o uso de editores de realidade virtual. [Kir05].

A aparência dos objetos está relacionada principalmente com as características de reflexão da superfície e com sua textura. A primeira depende do modelo de iluminação de *Phong*[8] e sombreamentos do tipo facetado, por interpolação de *Gourad*[9], ou interpolação de *Phong* Já a segunda é obtida a partir do mapeamento de um padrão de textura do espaço bidimensional sobre os objetos tridimensionais. É como se um pedaço de plástico, com seu padrão de textura, fosse ajustado e colocado sobre o objeto, fazendo parte integrante dele [WW92]. Com a texturização dos objetos, aumenta-se o nível de detalhe e de realismo de cena, fornece melhor visão de profundidade e permite a redução substancial do número de polígonos da cena, propiciando o aumento da taxa de quadros por segundo [BC94].

2.4.2 Modelagem Cinemática

Segundo [RH92] a modelagem geométrica de um objeto não é suficiente para conseguir uma animação. Para isto, é necessário haver interação, ou seja, deve ser possível agarrar o objeto, alterar sua posição, mudar a escala, detectar colisões e produzir deformações na superfície. Isto é possível por meio da utilização de coordenadas locais dos objetos e de coordenadas gerais, juntamente com matrizes de transformação.

2.4.3 Modelagem Física

Visando a obtenção de realismo nos mundos virtuais, os objetos virtuais, incluindo a imagem do usuário precisam comportar-se como se fossem reais [Kir05]. Por exemplo, um objeto sólido não poderá atravessar por dentro de um outro e deve mover-se de acordo com

[7] Projeto assistido por computador (*Computer-Aided Design*). Inclui um grupo de aplicações que permitem aos escritórios técnicos, designers e engenheiros conceber, testar, avaliar e definir um produto para fabricação no menor tempo possível.
[8] Na modelagem de cenas utilizando-se computação gráfica, os modelos de iluminação e tonalização definem a cor de cada ponto do objeto representado na cena. Esses modelos vão determinar a cor de cada ponto, levando em conta tanto as propriedades dos objetos, quanto a posição do observador e as características da luz que está incidindo sobre o objeto/superfície.
[9] É um método que calcula a iluminação nos vértices da célula e a interpola em seu interior.

o esperado. Além disso, suas características físicas devem ser consideradas (massas, pesos, inércia, deformações).

2.4.4 Comportamento do Objeto

É possível modelar o comportamento de objetos, independentemente da interação do usuário, a partir da utilização de relógios, calendários, termômetros e outros agentes inteligentes acessando, quando necessário, sensores externos [Kir05].

2.5 Requisitos para uma Aplicação de Realidade Virtual

Uma aplicação de Realidade Virtual de grande porte pode ser cara e complexa, em função de todos os recursos envolvidos. Para que o projeto e a elaboração das aplicações sejam bem sucedidos, é necessário que sejam satisfeitos um conjunto de requisitos [Kir05].

2.5.1 Requisitos da Interface do Usuário

Segundo [Sha93], dependendo da complexidade da aplicação, cinco requisitos e propriedades devem ser atendidos para que a mesma se torne utilizável e utilizada:

i) A aplicação deve gerar imagens estereoscópicas[10] animadas e suaves.

ii) A aplicação deve reagir rapidamente às ações do usuário. A resposta do sistema deve apresentar atrasos de imagens iguais ou menores que 100ms;

iii) A aplicação deve suportar sua distribuição em diversos processadores, permitindo, assim, a multiplicidade de usuários e a computação cooperativa;

2.5.2 Requisitos de Engenharia de Software

Para aplicações de Realidade Virtual, é necessária a observação de quatro requisitos, no que tange a Engenharia de Software [Sha93]:

[10] Fornecem ao usuário a sensação de tridimensionalidade. Para atingir esse objetivo, o sistema deve gerar, ao mesmo tempo, duas imagens diferentes, correspondendo às visões de cada um dos olhos.

i) A aplicação deve ser portável, no sentido de que as aplicações deverão ter facilidades para execução em diversas plataformas;

ii) A aplicação deve ser capaz de suportar uma larga faixa de periféricos de entrada e saída.

iii) A aplicação deve possuir independência geográfica. Esta deverá ajustar-se a diferentes configurações de localização física do usuário e de seus periféricos de entrada e saída;

iv) A aplicação deve ser flexível no que diz respeito ao ambiente de desenvolvimento, possibilitando a utilização de ambientes diferentes, bem como a execução de testes com outros dispositivos, com o mínimo de alteração do código.

v) A aplicação distribuída em realidade virtual deve fornecer um mecanismo eficiente de comunicação de dados. A utilização de dados compartilhados deve ser viabilizada com uma comunicação eficiente.

vi) A aplicação deve prover algum mecanismo de avaliação de desempenho. Deve possuir mecanismos de monitoração do tempo real e do desempenho geral da aplicação para garantir o sucesso do conjunto. Dentre estes requisitos, os mais importantes para uma interface de realidade virtual são aqueles relacionados com a taxa de quadros por segundo e com o atraso da resposta do sistema, que garantem a imersão no ambiente.

2.6 Considerações sobre o Capítulo

Neste capítulo, foram apresentados o ferramental e conhecimento teórico necessários para a definição do mecanismo de visualização, a ser proposto no capítulo seguinte. Foram vistos conceitos de redes de Petri e a ferramenta Design/CPN, responsável pela criação dos modelos. Apresentaram-se, também, conceitos de realidade virtual, ferramentas para criação de ambientes virtuais e a linguagem VRML.

Capítulo 3 – Mecanismo para Integração entre Modelos Redes de Petri Coloridas e Modelos VRML

Neste capítulo, descreve-se o mecanismo proposto para promover a integração entre um modelo rede de Petri colorida e sua respectiva representação em realidade virtual. Nesta solução são utilizadas as ferramentas Design/CPN e FreeWRL, descritas no capítulo anterior . Para a realização da troca de mensagens entre esses dois tipos de modelos, utiliza-se a biblioteca Comms/CPN, a ser detalhada mais adiante.

Na Figura 3.1, é apresentada a estrutura do mecanismo de integração proposto neste trabalho, composta por três camadas: (1) a camada de modelos, onde estão os modelos redes de Petri que trocam (enviam e recebem) mensagens com o mundo virtual; (2) a camada de comunicação que estabelece a interação entre os modelos formais e a representação VRML; (3) a camada de visualização, que corresponde ao modelo VRML.

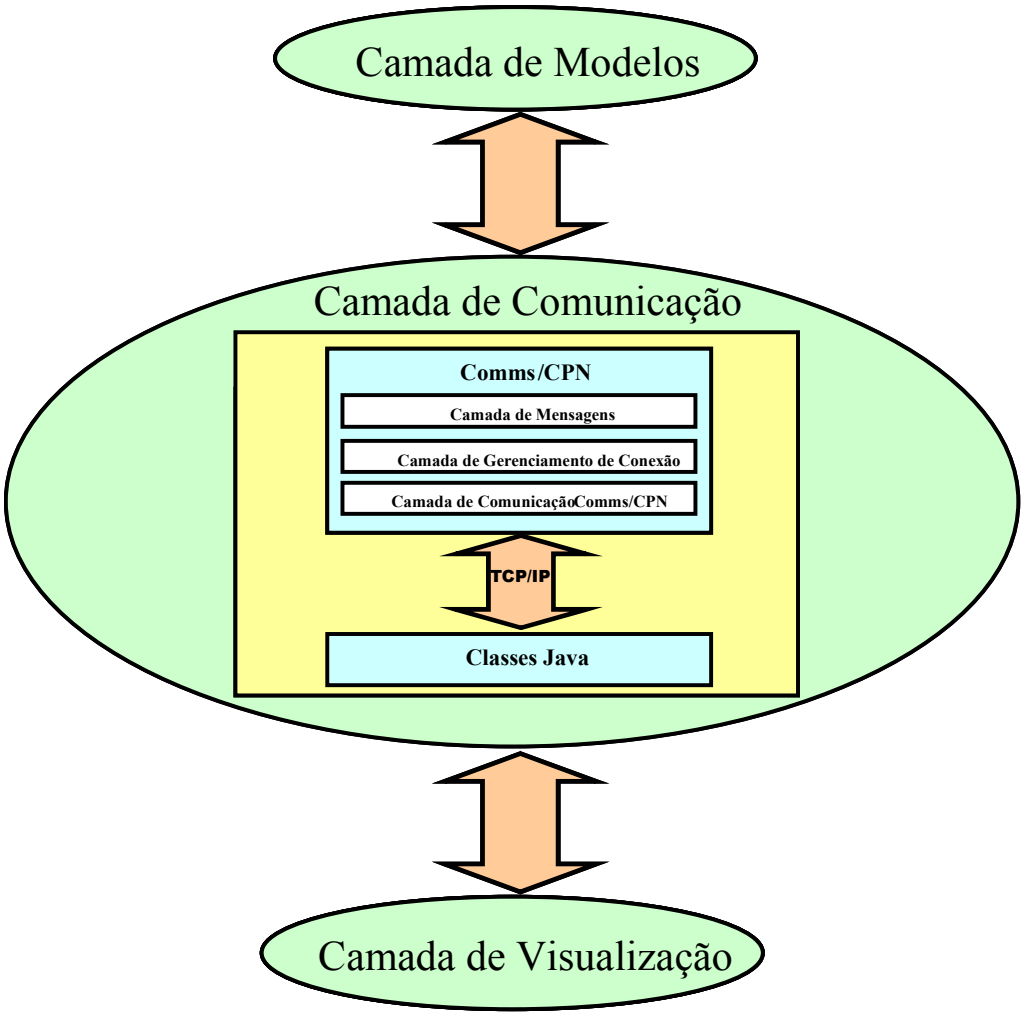

Figura 3.1 – Mecanismo de Integração

3.1 Camada de Comunicação

Esta camada cria o meio de comunicação entre a rede de Petri colorida e o modelo VRML. É nela que as mensagens são tratadas. É constituída, essencialmente, por uma biblioteca de comunicação e por um conjunto de classes java que abstraem as suas funções. Nesta seção, a referida biblioteca será detalhada, a partir do artigo de [GK01].

3.1.1 Biblioteca Comms/CPN

Modelos concebidos por meio da ferramenta Design/CPN, são restritos em sua capacidade de interagir com processos externos. Para suprir essa deficiência existe um conjunto de programas, codificados em SML[11], que provê ao Design/CPN a infra-estrutura necessária para estabelecer uma comunicação entre redes de Petri coloridas (CPN) e processos externos, denominada biblioteca Comms/CPN.

As principais características e/ou vantagens promovidas pela biblioteca Comms/CPN são:

- A visualização dos comportamentos e o controle dos modelos são independentes do Design/CPN;
- A possibilidade de visualização de modelos em máquinas remotas.

3.1.1.1 Estrutura da Biblioteca Comms/CPN

A biblioteca Comms/CPN foi projetada para atuar como interface entre os modelos redes de Petri coloridas e o protocolo TCP/IP. A Figura 3.2 mostra a arquitetura geral da biblioteca Comms/CPN, composta de três módulos principais, organizados em:

i) **Camada de comunicação**, que contém os mecanismos de transportes básicos do protocolo TCP/IP e todas as funções primitivas fundamentais relacionadas a *sockets*;

ii) **Camada de mensagens**, que é responsável pela conversão dos dados em fluxo de bytes para que haja troca de mensagens entre Design/CPN e aplicações externas;

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

[11] Linguagem de Modelagem Padrão.

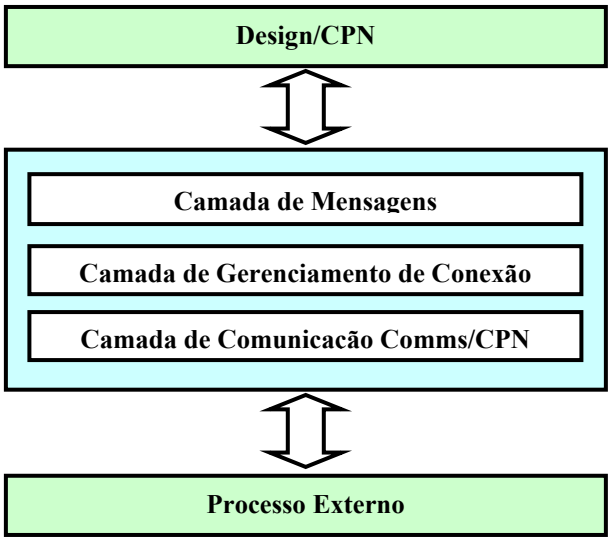

Figura 3.2 – Arquitetura geral da biblioteca Comms/CPN

O projeto das três camadas foi baseado em cinco requisitos funcionais essenciais, representando os serviços esperados pela biblioteca: i) abrir conexões para processos externos; ii) aceitar conexões de processos externos; iii) enviar dados a processos externos; iv) receber dados de processos externos e; v) fechar conexões de processos externos.

3.1.1.2 Camada de Comunicação da Biblioteca Comms/CPN

O protocolo utilizado pela biblioteca Comms/CPN é o TCP/IP, por dois fatores principais: (1) as transmissões de dados deveriam ser livres de erros e ordenadas; (2) 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*.

3.1.1.2.1 Conceito de Protocolo TCP/IP

TCP/IP é um acrônimo para o termo *Transmission Control Protocol* / *Internet Protocol Suite*, ou seja, é um conjunto de protocolos, em que dois dos mais importantes (o IP e o TCP) 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*) [Red04].

Ainda segundo [Red04], a arquitetura TCP/IP realiza a divisão de funções do sistema de comunicação em estruturas de camadas. Em TCP/IP as camadas são:

- **Aplicação;**
- **Transporte;**
- **Inter-Rede;**
- **Rede.**

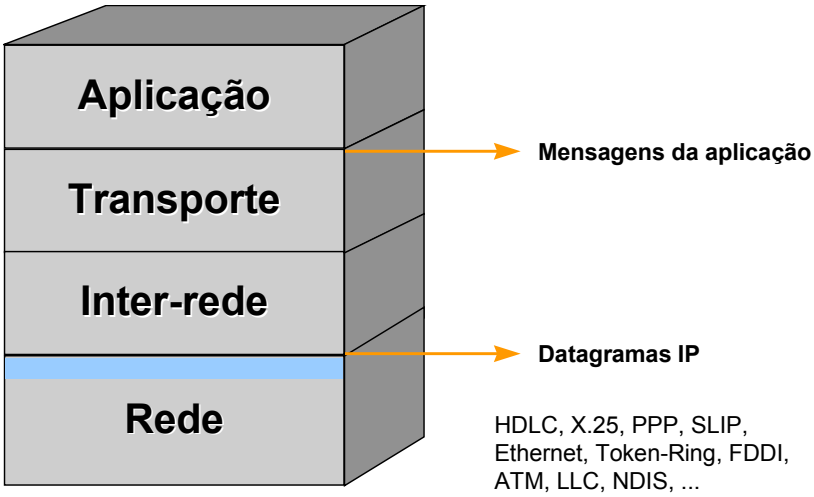

Figura 3.3 – Camadas da Arquitetura TCP/IP

3.1.1.2.2 Camada de rede

A camada de rede é responsável pelo envio de datagramas construídos pela camada Inter-Rede. Esta camada realiza também o mapeamento entre um endereço de identificação de nível Inter-rede para um endereço físico ou lógico do nível de Rede.

3.1.1.2.3 Camada Inter-Rede

Esta camada realiza a comunicação entre máquinas vizinhas através do protocolo IP.

Para identificar cada máquina e a própria rede, onde estas estão situadas, é definido um identificador, chamado endereço IP, que é independente de outras formas de endereçamento que possam existir nos níveis inferiores. No caso de existir endereçamento nos níveis inferiores é realizado um mapeamento para possibilitar a conversão de um endereço IP em um endereço deste nível.

3.1.1.2.4 Camada de Transporte

Esta camada reúne os protocolos que realizam as funções de transporte de dados fim-a-fim, ou seja, considerando apenas a origem e o destino da comunicação, sem considerar elementos intermediários. A camada de transporte possui dois protocolos que são o UDP (User Datagram Protocol) e TCP (Transmission Control Protocol).

O protocolo UDP realiza apenas a multiplexação para que várias aplicações possam acessar o sistema de comunicação de forma coerente.

O protocolo TCP realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. São responsabilidades do protocolo TCP: o controle de fluxo, o controle de erro, a sequenciação e a multiplexação de mensagens.

3.1.1.2.5 Camada de Aplicação

A camada de aplicação reúne os protocolos que fornecem serviços de comunicação ao sistema ou ao usuário. Podem-se separar os protocolos de aplicação em:

- Protocolos de serviços básicos, que fornecem serviços para atender as próprias necessidades do sistema de comunicação.
- Protocolos de serviços para o usuário: FTP, HTTP, Telnet, SMTP, POP3, IMAP, TFTP, NFS, NIS, LPR, LPD, ICQ, RealAudio, Gopher, Archie, Finger, SNMP e outros.

3.1.1.3 Camada de Gerenciamento de Conexões

A conexão representa um canal de comunicação entre o modelo rede de Petri e um processo externo. É a camada de gerenciamento de conexão que faz esse controle, criando, encerrando, estabelecendo dinamicamente e mantendo múltiplas conexões. O projeto desta camada reflete os seguintes requisitos:

i) **Identificação de conexão**. Associa uma *string* única a cada conexão feita. Esta *string* pode ser definida pelo usuário ou, internamente, pela biblioteca.

ii) **Atributos de conexões**. Quando uma conexão é criada, é necessário que informações sobre a mesma sejam armazenadas. Para tanto, armazenam-se sua *string* de conexão (para que haja identificação, propriamente dita) e o *socket* TCP/IP (para o envio e recebimento de mensagens).

3.1.1.4 Camada de Mensagens

Um requisito essencial da biblioteca é que esta deve possibilitar o intercâmbio de diferentes tipos de dados, inclusive os definidos pelos usuários. Isto ocorre por meio do protocolo TCP/IP que trata os dados na forma de seqüência de bytes para a transmissão pela rede e que, portanto, os objetos de dados devam ser convertidos.

Existem duas funções que tratam da conversão de objetos de dados, uma para codificação e outra para decodificação. A função de codificação converte um objeto de dados em fluxo de *bytes* para transmissão, via TCP/IP e a função de decodificação, converte uma seqüência de *bytes* em um objeto de dados.

A seqüência de *bytes* consiste em dados empacotados para transmissão, de tal forma que o receptor possa saber quando todos os pacotes já chegaram, por meio da segmentação do fluxo de bytes, providenciando que cada segmento possua um cabeçalho que o identifique, formando um ou mais pacotes de dados.

O formato do pacote adotado pela biblioteca Comms/CPN consiste de um *byte* cabeçalho e um segmento de 127 *bytes*, conforme visto na Figura 3.4. Sete *bits* do cabeçalho indicam o tamanho do segmento de dados ($2^7 - 1$) e o *bit* restante indica se este é

o último pacote na transmissão dos dados. O tamanho máximo de 127 *bytes* de segmento de dados pode ser alterado e um cabeçalho de tamanho diferente pode ser definido.

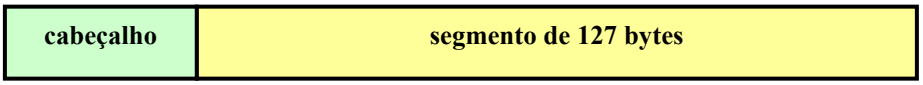

Figura 3.4 – Formato do pacote para transmissão de dados, definido para a biblioteca Comms/CPN

3.1.1.5 Primitivas da Biblioteca Comms/CPN

As primitivas de estabelecimento de comunicação e troca de mensagens da biblioteca Comms/CPN são implementadas em arquivos SML. Nesta seção, serão identificados os referidos arquivos e cada primitiva que os compõe.

i) Camada de Comunicação

As primitivas desta camada são responsáveis pelas funções principais de rede do protocolo TCP/IP. A Tabela 3.1 apresenta as primitivas.

Primitiva	Responsabilidade
connect	Criar uma conexão com um processo externo.
accept	Aguardar por uma conexão na porta especificada. Esta primitiva bloqueia os processos até que uma comunicação com uma aplicação externa seja estabelecida.
send	Enviar uma seqüência de *bytes* a um processo externo.
receive	Receber uma seqüência de *bytes* de um processo externo. Esta primitiva bloqueia os processos até que todos os segmentos de *bytes* sejam recebidos.
disconnect	Fechar conexão.

Tabela 3.1 – Primitivas da camada de comunicação da biblioteca Comms/CPN

ii) Camada de Mensagens

Na Tabela 3.2 estão descritas as primitivas responsáveis por empacotar e desempacotar dados.

Primitiva	Responsabilidade
send	Converter objetos de dados em fluxo de bytes e invocar a primitiva de envio de mensagens da camada de comunicação.
receive	Invocar a primitiva de recebimento de mensagens da camada de comunicação e converter o fluxo de bytes recebido em objetos de dados.

Tabela 3.2 – Primitivas da camada de mensagens.

iii) Camada de Gerenciamento de Conexão

Nesta camada são definidas as primitivas invocadas diretamente pelo modelo rede de Petri concebido pelo Design/CPN.

A Tabela 3.3 apresenta as primitivas que compõem a camada de gerenciamento de conexão:

Primitiva	Responsabilidade
openConnection	Analisar se a string de conexão é única para, então, criar uma conexão para o processo externo, invocando primitivas da camada de comunicação.
acceptConnection	Receber uma *string* de conexão e um número de porta, como parâmetros de entrada para, em seguida, verificar se a *string* passada é única e então, aguardar pela requisição de conexão, bloqueando a rede de Petri até que esta seja recebida. Quando isso acontece, uma conexão é estabelecida com o processo externo.

send	Permitir o envio de qualquer tipo de dados a processos externos. Esta função é polimórfica, de modo que os dados para envio podem ser de qualquer tipo, inclusive tipos definidos pelo usuário. Esta primitiva codifica os dados para envio em uma seqüência de bytes e invoca a função send da camada de mensagens.
receive	Permitir o recebimento de qualquer tipo de dados, vindos de processos externos. Esta primitiva também é polimórfica. Ela recupera a conexão do dado a ser recebido e, em seguida, invoca o receive da camada de mensagens para o recebimento dos dados, que por sua vez, invoca a função de decodificação.
closeConnection	Permitir 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.

Tabela 3.3 – Primitivas da camada de conexões.

3.1.2 Arquivos Java para Abstração da Biblioteca Comms/CPN

Como já mencionado, os arquivos que compõem a biblioteca Comms/CPN são codificados em SML. Entretanto, esta é uma linguagem não tão usual para os desenvolvedores. Portanto, para facilitar a codificação dos serviços necessários à comunicação entre um modelo rede de Petri colorida e um processo externo (modelo VRML), lança-se mão de uma linguagem de domínio público, no caso desta dissertação, a linguagem Java.

A Figura 3.5 ilustra a inter-relação entre as classes Java que compõem a camada de comunicação e estas com os modelos rede de Petri e VRML. Tem-se na parte superior da referida figura, a camada de modelos, compreendida pelos modelos redes de Petri. Estes enviam e recebem mensagens à camada de comunicação, composta pela biblioteca Comms/CPN que possui suas funções de gerenciamento de conexão, comunicação e troca de mensagens abstraídas pelas classes JavaCPNInterface, PollServer e EncodeDecode, respectivamente. Na parte inferior da figura, tem-se a camada de visualização composta pela classe Java AnimaMundo que recebe mensagens da classe PollServer e, a partir dessas provoca a animação do *script* VRML.

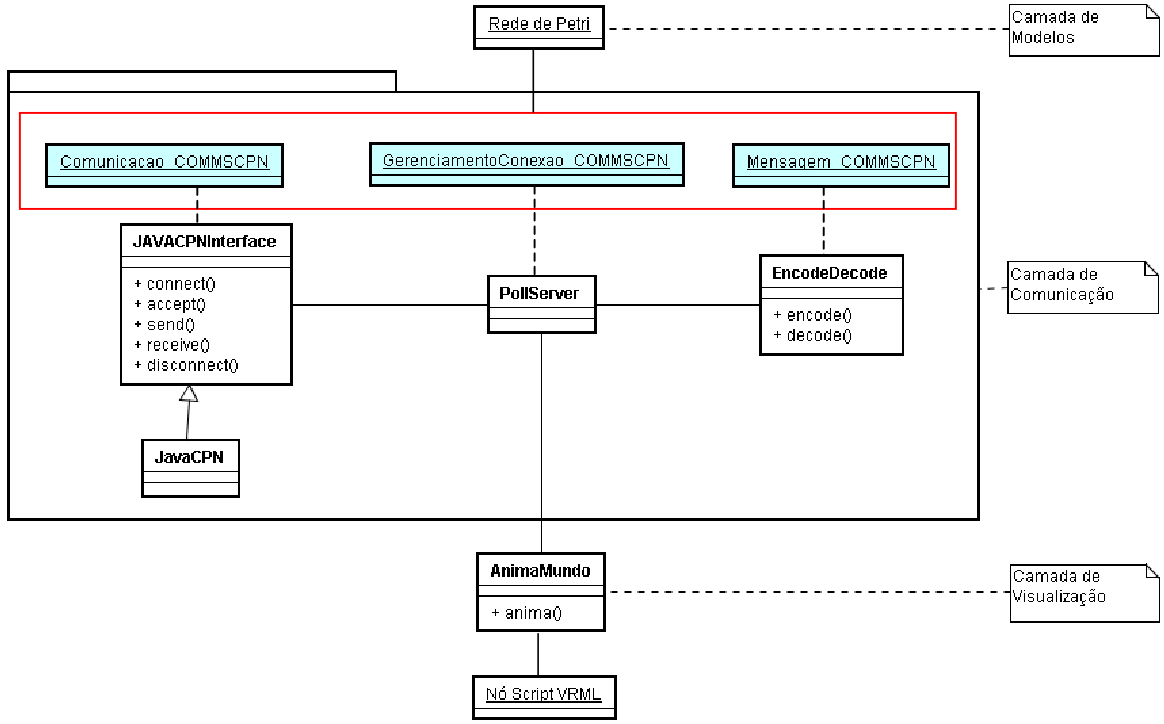

Figura 3.5 – Diagrama de classes/objetos da camada de comunicação

A seguir, serão apresentados os algoritmos das classes que compõem a camada de comunicação.

i) ***JavaCPNInterface.class***

Esta classe é uma abstração da camada de comunicação da biblioteca Comms/CPN. Define a interface para as primitivas connect, accept, send, receive e disconnect. O algoritmo que a define é o seguinte:

1. Definir assinatura do método responsável pelo estabelecimento de conexão (connect);
2. Definir assinatura do método responsável pelo aceite de conexão (accept);
3. Definir assinatura do método responsável pelo envio de mensagem (send);
4. Definir assinatura do método responsável pelo recebimento de mensagem (receive);
5. Definir assinatura do método responsável pelo fechamento de conexão (disconnect).

ii) ***JavaCPN.class***

Esta classe implementa as primitivas connect, accept, send, receive e disconnect, definidas na classe JavaCPNInterface. A seguir, são apresentados os algoritmos de cada primitiva:

- connect
    1. Criar *socket* para estabelecimento de conexão, passando como parâmetros *string* de conexão e 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 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 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) ***PollServer.class***

É uma classe do tipo *thread*, responsável pela abstração da camada de gerenciamento de conexões da biblioteca Comms/CPN. Utiliza as funções primitivas das classes JavaCPN e EncodeDecode.

Para o estabelecimento da conexão entre as ferramentas Design/CPN e o FreeWRL é utilizada uma variável do tipo *socket*, que recebe como parâmetros a string de conexão e a porta. O trecho de código a seguir, mostra esta implementação:

```
try
  {
    cpn_VRML = new JavaCPN();
    cpn_VRML.connect(stringConexao, porta);
    stop=false;
  }
  catch (UnknownHostException e)
    {
      stop=true;
    }
      catch (IOException e)
    {
      stop=true;
    }
```

O envio de mensagens do mundo virtual para o modelo rede de Petri é feito por meio da segmentação da mensagem, em que os dados são convertidos em um fluxo de bytes, conforme é apresentado no trecho de código, a seguir:

```
ByteArrayInputStream paraCPN=EncodeDecode.encode(mensagem);
cpn_VRML.send(paraCPN);
```

Já o recebimento de mensagens vindas da rede de Petri é feito por meio da conversão da mensagem, de uma cadeia de bytes para objetos de dados, observado na linha de código, adiante:

```
deCPN=EncodeDecode.decode(cpn_VRML.receive());
```

3.2 Camada de Modelos

Esta camada representa o modelo rede de Petri colorida. Sua proposta é possibilitar a adição de uma seqüência de lugares e transições a uma rede de Petri principal que servirá de canal de comunicação com um processo externo, no caso deste trabalho, com um modelo virtual VRML correspondente. Esta seqüência é denominada "Fluxo de Conexão", ilustrada pela Figura 3.6. Seu detalhamento é feito a seguir:

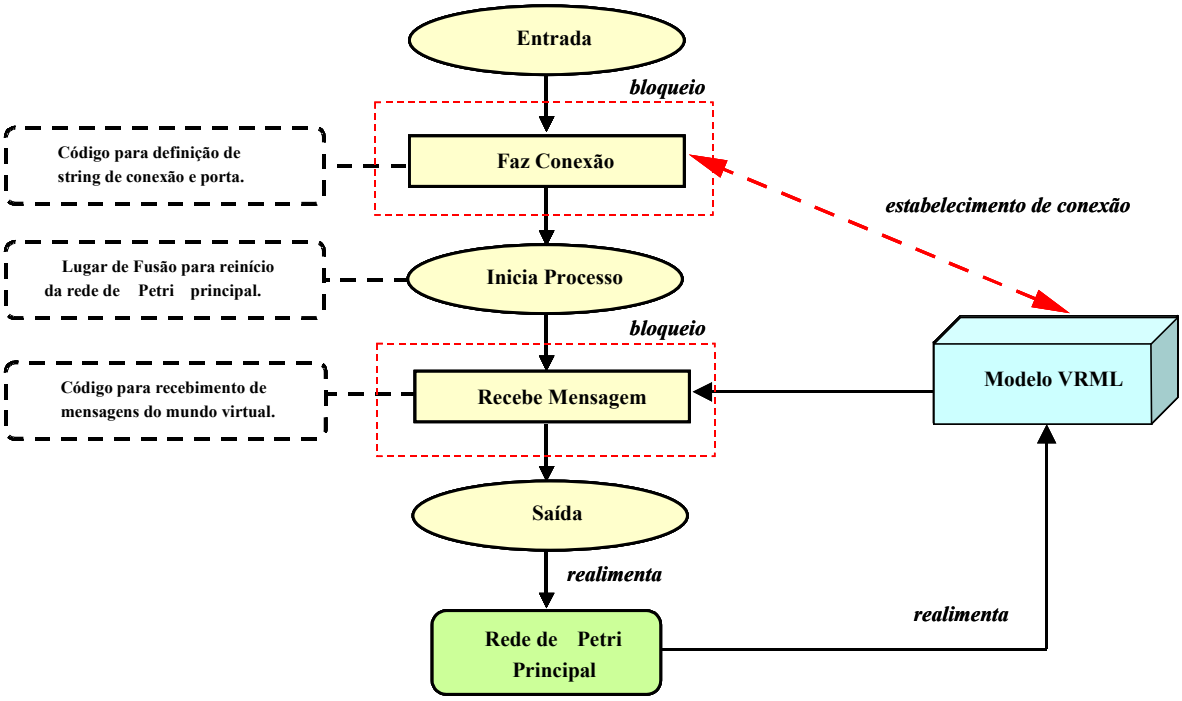

Figura 3.6 – Fluxo de Conexão

1. O fluxo de conexão possui um lugar denominado "Entrada" que habilita, automaticamente, uma transição denominada "Faz Conexão", assim que o modelo inicia sua simulação;

2. No momento em que a transição "Faz Conexão" é disparada, o código associado a ela é acionado e uma função para estabelecer a conexão com o modelo do mundo virtual é executada. Esta função define dois parâmetros, do nome da

conexão (*string* que a identifica) e da porta (número inteiro, referente a um canal livre para comunicação), que são tratados pelas primitivas da biblioteca Comms/CPN. Esta transição é disparada uma única vez;

2.1. Na biblioteca Comms/CPN, os dois parâmetros são avaliados e a tentativa de conexão com o mundo VRML é feita;

3. O canal de comunicação é ocupado;

3.1. A comunicação entre o modelo rede de Petri e seu respectivo mundo virtual é estabelecida;

4. Uma ficha é inserida no lugar denominado “Inicia Processo”;

4.1. A transição denominada “Recebe Mensagem” é habilitada;

4.1.1. Quando a transição “Recebe Mensagem” é disparada, ocorre um bloqueio da rede. Neste instante, é aguardado o recebimento de uma mensagem do mundo VRML, por meio da biblioteca Comms/CPN.

5. Uma interação do usuário com o mundo virtual é realizada, ou seja, uma mensagem é enviada para a rede;

5.1. Uma ficha é inserida no lugar denominado “Saída”, correspondendo à mensagem vinda do mundo virtual;

5.1.1. A rede de Petri principal é realimentada;

6. A partir da execução da função de envio, vinculada à uma transição da rede de Petri principal, uma mensagem é enviada para o mundo virtual, por meio da biblioteca Comms/CPN;

   6.1. O modelo VRML é realimentado.

O diagrama de seqüência apresentado na Figura 3.7 ilustra, a seqüência de passos descrita.

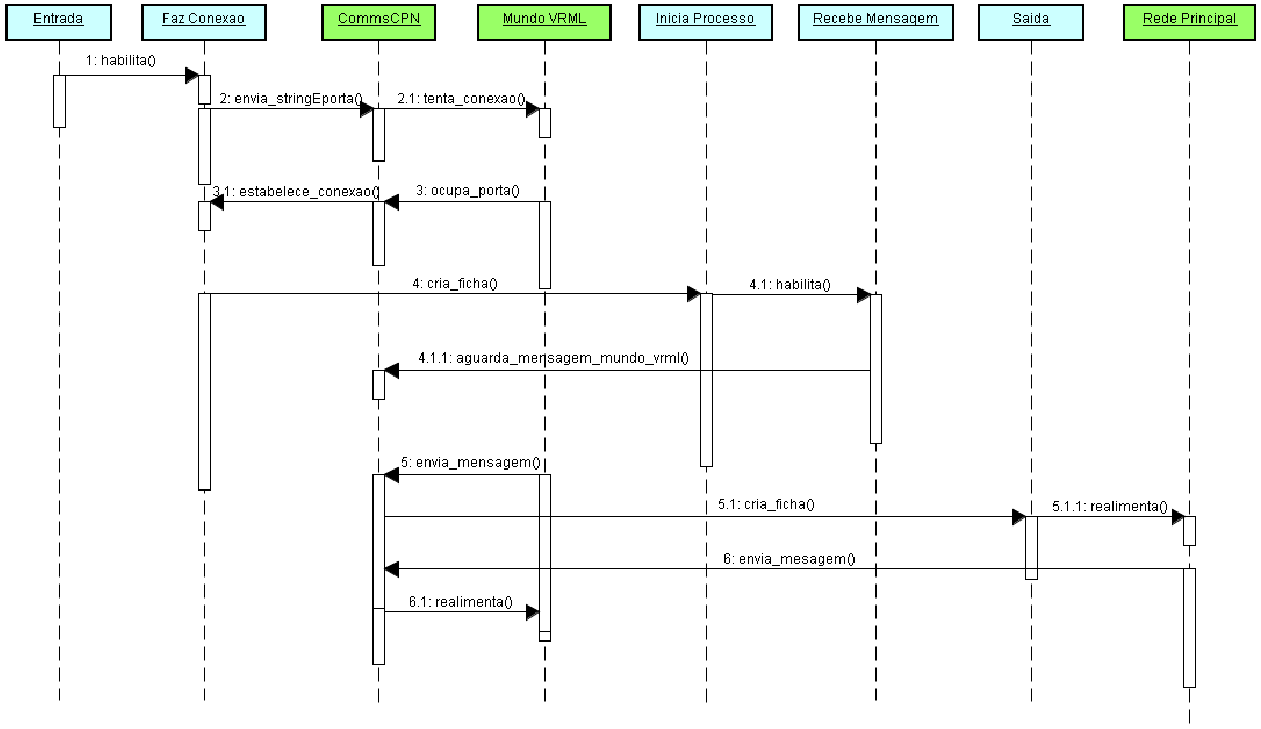

Figura 3.7 – Diagrama de seqüência da execução do "Fluxo de Conexão".

3.2.1 Descrição das Funções para Implementar o "Fluxo de Conexão"

i) Estabelecimento da Conexão

O trecho de código a seguir, define a função responsável pelo estabelecimento da conexão entre a rede de Petri e o mundo virtual.

```
action
acceptConnection("Conexao1",portaX);
```

Onde, “Conexao1” é a string de conexão e “portaX” a porta utilizada como canal de comunicação.

ii) Recebimento de Mensagem

```
output(mensagem)
action
recebe_mensagem();
```

A variável “mensagem” armazena a mensagem vinda do mundo virtual e “recebe_mensagem” a função responsável pelo recebimento da mensagem.

```
fun recebe_mensagem() = ConnManagement.receive(”Conexao”,
stringDecode);
```

A função “recebe_mensagem” utiliza o método ConnManagementLayer.receive da biblioteca Comms/CPN.

iii) Envio de Mensagem

```
action
envia_mensagem(”Comando1”);
```

O parâmetro da função “envia_mensagem” é uma *string* que o mundo virtual entende como um comando para um objeto, fazendo com que seu comportamento seja alterado.

```
fun envia_mensagem(doing) = ConnManagement.send(”Conexao”,
doing, stringEncode);
```

A função “envia_mensagem” utiliza o método ConnManagementLayer.send da biblioteca Comms/CPN.

3.3 Camada de Visualização

Esta camada é responsável pela visualização, em ambiente tridimensional, do comportamento de uma rede de Petri. Nela, é definida a classe Java AnimaMundo e o nó *script* VRML. O primeiro implementa os serviços que provocarão a animação dos objetos definidos no segundo arquivo.

i) ***AnimaMundo.class***

Esta classe não possui um formato padrão, como as demais vistas, pois é nela que são definidos os métodos responsáveis pela animação do mundo VRML, bem como os métodos que tratam das mensagens enviadas do modelo virtual para o modelo rede de Petri. Portanto, esta classe deve receber as devidas alterações (criação de novos serviços) para que estas possam surtir a animação desejada, no contexto do ambiente virtual proposto.

A visualização e animação dos objetos ocorrem por meio da instanciação desta classe pelo *script* que define o mundo VRML.

ii) ***NohScript.wrl***

Este arquivo contém o *script* VRML que define os objetos tridimensionais que terão seus comportamentos modificados, de acordo com as mensagens vindas do Design/CPN, via camada de comunicação e tratadas pelos métodos da classe AnimaMundo.

Este arquivo, também, possibilita que as ações realizadas sobre os objetos, possam ser refletidas no modelo criado no Design/CPN. Essas ações são mensagens enviadas à classe AnimaMundo e tratada por seus métodos.

Capítulo 4 – Estudo de Caso

Neste capítulo, será detalhado um estudo de caso que utiliza o mecanismo de integração proposto.

4.1 Contextualização do Estudo de Caso

Em ambientes industriais a interação com o sistema controlado ou supervisionado pode ser feita através da utilização de terminais de computador ou através da manipulação direta dos equipamentos. No primeiro caso, utilizam-se programas supervisórios, software do tipo SCADA (*Supervisory Control And Data Aquisition*), já no segundo caso a interação ocorre por meio da manipulação direta de botões e chaves, nos painéis dos dispositivos.

Do ponto de vista das funcionalidades, são observadas algumas similaridades entre os dispositivos encontrados em *software* e os dispositivos físicos encontrados nos painéis. Em ambos os casos os dispositivos possuem estados que podem ser percebidos de forma visual. A diferença básica para os dois casos consiste na forma de interação. No caso da modelagem do software a interação se dá através de dispositivos de entrada e saída associados a um terminal de computador, enquanto que para o sistema dito "manual" a interação se dá através da manipulação direta das chaves no painel de operação.

Este estudo de caso está inserido no contexto da construção de um simulador para treinamento de operadores de subestações elétricas. Neste estudo de caso serão tratadas as interações do operador tanto através do supervisório quanto através dos painéis, embora em ambos a interação se de através de um computador. O estudo de caso possibilitou a representação de dispositivos físicos do painel, em uma tela, preservando o realismo da interação.

Na Figura 4.1 é apresentada a idéia geral do simulador, na qual o operador pode interagir com os objetos de dois modelos virtuais tridimensionais, um responsável pela visualização de um ambiente supervisório (composto de disjuntores e seccionadoras) e outro responsável pelo ambiente de painéis (composto de botões do tipo giro-pressão-giro). A cada modelo virtual corresponde um modelo de rede de Petri colorida responsável pela alteração do comportamento dos objetos tridimensionais.

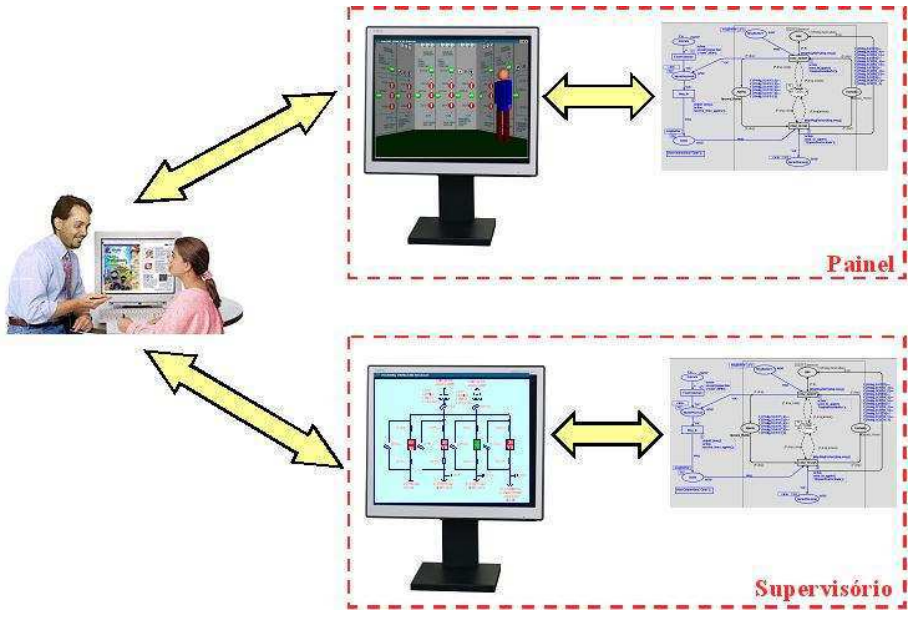

Figura 4.1 Visão geral do simulador

Este simulador utiliza a estrutura de camadas apresentada no capítulo anterior, conforme apresentado na Figura 4.2.

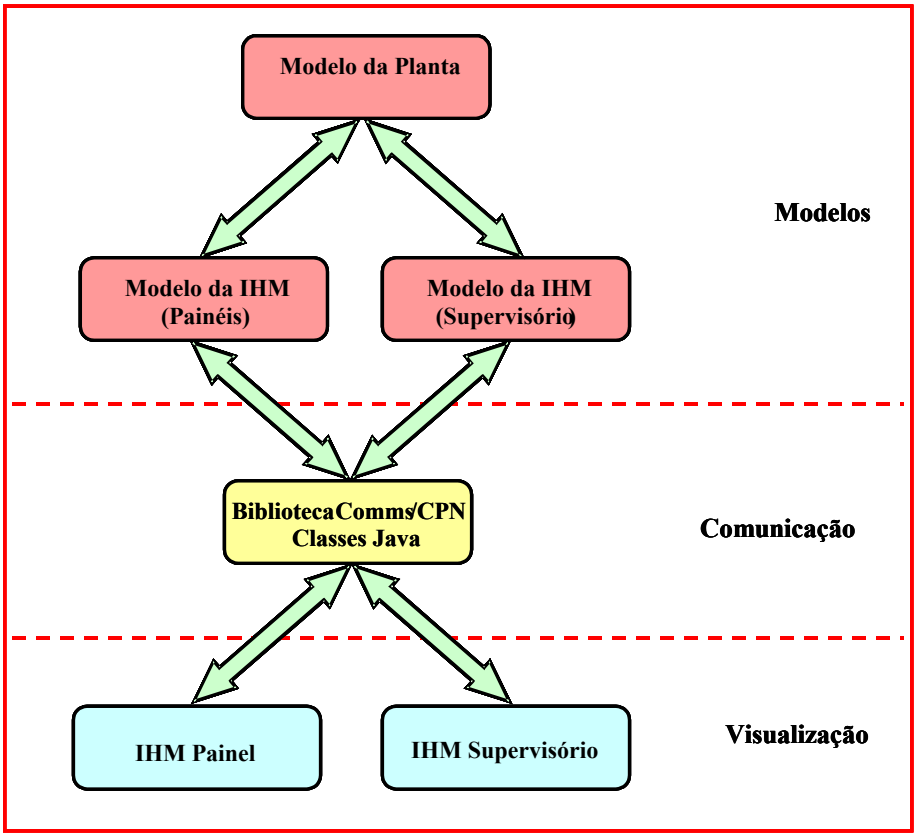

Figura 4.2 Representação do mecanismo de integração, aplicado ao estudo de caso

A camada de modelos é composta por três redes de Petri coloridas: modelo da IHM[12] supervisório, onde é modelada a interface de operação de quatro disjuntores e quatorze chaves seccionadoras, conforme observado no diagrama unifilar da Figura 4.3; modelo da IHM painéis, onde é modelada, também, a interface de operação de quatro disjuntores e as quatorze seccionadoras. Entretanto, neste caso, a abordagem é do ponto de vista dos dispositivos acessíveis ao operador, botões giro-pressão-giro, por exemplo; e por fim o modelo da IHM planta, onde são modelados os dispositivos de campo.

A camada de comunicação é composta pelas classes JavaCPNInterface, JavaCPN, PollServer e EncodeDecode que abstraem as funções da biblioteca Comms/CPN.

[12] Interface Homem-Máquina.

A camada de visualização representa dois contextos. Um que simula o painel de uma subestação elétrica, por meio da classe Java AnimaPainel e do *script* VRML painel.wrl e outro que simula o supervisório de uma subestação elétrica, via classe AnimaSupervisorio e *script* supervisorio.wrl.

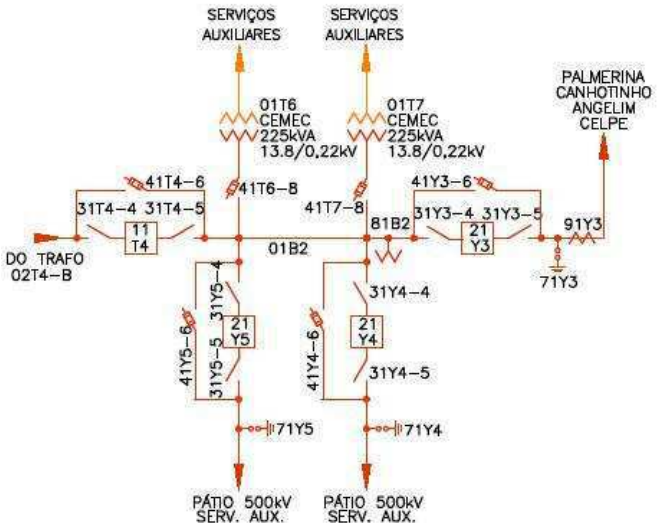

Figura 4.3 – Diagrama unifilar do subconjunto de dispositivos representado no estudo de caso

O diagrama unifilar da Figura 4.3 é detalhado na seção seguinte.

4.2 Camada de Modelos

Nesta seção, serão apresentados os modelos redes de Petri da IHM supervisório e da IHM painel, deste estudo de caso.

4.2.1 Modelo rede de Petri da IHM supervisório

Este modelo representa o comportamento dos disjuntores e das seccionadoras, mais especificamente, o comportamento das operações de abertura e fechamento dos mesmos.

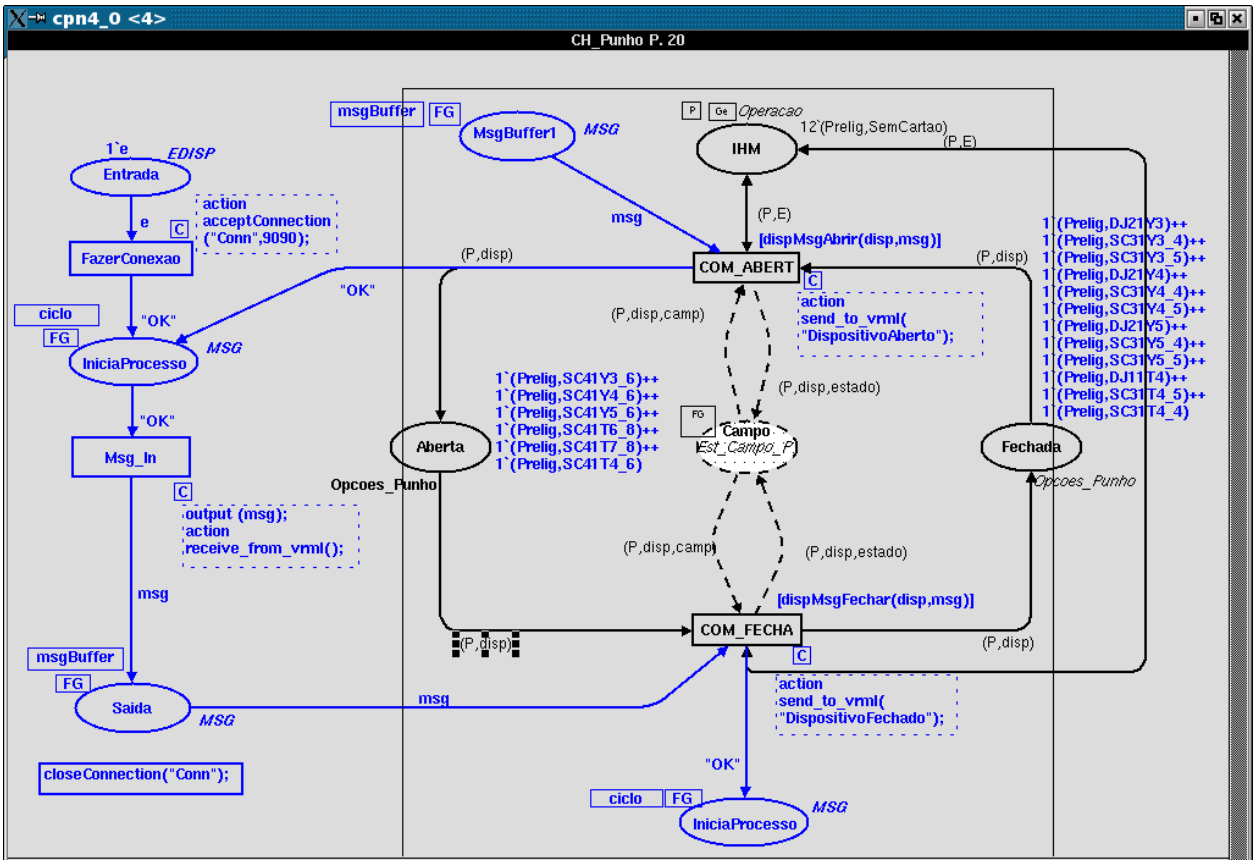

Figura 4.4 – Modelo rede de Petri da IHM Supervisório

Na figura 4.4, observa-se a existência de duas partes que a constituem. O segmento à esquerda (em azul), representado pelo "Fluxo de Conexão", definido nesta dissertação e o segmento à direita (em preto), representado pela rede principal, responsável pelo comportamento (abertura e fechamento) dos dispositivos.

A seguir, será explicado cada item deste modelo.

4.2.1.1 Lugar "Entrada"

Este lugar é responsável por habilitar a transição "FazerConexao", assim que a rede entra em modo de simulação. Para este estudo de caso, utiliza-se a porta 9090.

4.2.1.2 Transição “FazerConexao”

Assim que esta transição é disparada, o código associado a ela é executado para estabelecimento de conexão com um processo externo, neste caso, o modelo VRML do supervisório.

4.2.1.3 Lugar de Fusão “IniciaProcesso”

Este lugar é responsável por manter o modelo sempre disponível para receber mensagens de abertura ou fechamento de um dispositivo do mundo virtual. Toda vez que um processo de abertura ou fechamento é finalizado, uma ficha “OK” é depositada neste lugar.

4.2.1.4 Transição “Msg_In”

Esta transição executa o código responsável por receber uma mensagem de abertura ou de fechamento de um disjuntor ou seccionadora, vinda do mundo virtual. Ao recebê-la, a transição fica habilitada, pronta para o disparo.

4.2.1.5 Lugar de Fusão “Saida”

Assim que a transição “Msg_In” é disparada, o comando de operação vindo do modelo VRML é depositado em forma de ficha, neste lugar, realimentando a rede de Petri principal (ver Figura 4.4, lado direito).

4.2.1.6 Guarda [disjMsgAbrir(disp,msg)]

Esta é a guarda da transição “COM_ABERT”. É responsável por retornar um valor booleano. Se o retorno for verdadeiro, isto indica que houve um par de valores satisfeito. Esse par corresponde a dois parâmetros, o primeiro “disp” que identifica o dispositivo, o segundo “msg” que indica o comando de abertura do dispositivo, vinda do modelo virtual, a partir de uma interação do operador.

A Tabela 4.1 apresenta os dispositivos, seus comandos de abertura e seus estados, após a execução do comando.

disp	msg	Estado (disp x msg)
Disjuntor 21Y3	“AbrirDJ21Y3”	“Aberto”
Seccionadora 41Y3_6	“AbrirSC41Y3_6”	“Aberto”
Seccionadora 31Y3_5	“AbrirSC31Y3_5”	“Aberto”
Seccionadora 31Y3_4	“AbrirSC31Y3_4”	“Aberto”
Disjuntor 21Y4	“AbrirDJ21Y4”	“Aberto”
Seccionadora 41Y4_6	“AbrirSC41Y4_6”	“Aberto”
Seccionadora 31Y4_5	“AbrirSC31Y4_5”	“Aberto”
Seccionadora 31Y4_4	“AbrirSC31Y4_4”	“Aberto”
Disjuntor 21Y5	“AbrirDJ21Y5”	“Aberto”
Seccionadora 41Y5_6	“AbrirSC41Y5_6”	“Aberto”
Seccionadora 31Y5_5	“AbrirSC31Y5_5”	“Aberto”
Seccionadora 31Y5_4	“AbrirSC31Y5_4”	“Aberto”
Disjuntor 11T4	“AbrirDJ11T4”	“Aberto”
Seccionadora 41T4_6	“AbrirSC41T4_6”	“Aberto”
Seccionadora 31T4_5	“AbrirSC31T4_5”	“Aberto”
Seccionadora 31T4_4	“AbrirSC31T4_4”	“Aberto”
Seccionadora 41T6_8	“AbrirSC41T6_8”	“Aberto”
Seccionadora 41T7_8	“AbrirSC41T7_8”	“Aberto”

Tabela 4.1 – Dispositivos x comandos de abertura

4.2.1.7 Guarda [disjMsgFechar(disp,msg)]

Esta é a guarda da transição “COM_FECHA”. É, também, responsável por retornar um valor booleano. Se o retorno for verdadeiro, isto indica que houve um par de valores satisfeito. Esse par corresponde a dois parâmetros, o primeiro “disp” que identifica o dispositivo, o segundo “msg” que indica o comando de fechamento do dispositivo, vinda do modelo virtual, a partir de uma interação do operador.

A Tabela 4.2 apresenta os dispositivos, seus comandos de fechamento e seus estados, após a execução do comando.

disp	*msg*	*Estado (disp x msg)*
Disjuntor 21Y3	“FecharDJ21Y3”	“Fechado”
Seccionadora 41Y3_6	“FecharSC41Y3_6”	“Fechado”
Seccionadora 31Y3_5	“FecharSC31Y3_5”	“Fechado”
Seccionadora 31Y3_4	“FecharSC31Y3_4”	“Fechado”
Disjuntor 21Y4	“FecharDJ21Y4”	“Fechado”
Seccionadora 41Y4_6	“FecharSC41Y4_6”	“Fechado”
Seccionadora 31Y4_5	“FecharSC31Y4_5”	“Fechado”
Seccionadora 31Y4_4	“FecharSC31Y4_4”	“Fechado”
Disjuntor 21Y5	“FecharDJ21Y5”	“Fechado”
Seccionadora 41Y5_6	“FecharSC41Y5_6”	“Fechado”
Seccionadora 31Y5_5	“FecharSC31Y5_5”	“Fechado”
Seccionadora 31Y5_4	“FecharSC31Y5_4”	“Fechado”
Disjuntor 11T4	“FecharDJ11T4”	“Fechado”
Seccionadora 41T4_6	“FecharSC41T4_6”	“Fechado”
Seccionadora 31T4_5	“FecharSC31T4_5”	“Fechado”
Seccionadora 31T4_4	“FecharSC31T4_4”	“Fechado”
Seccionadora 41T6_8	“FecharSC41T6_8”	“Fechado”
Seccionadora 41T7_8	“FecharSC41T7_8”	“Fechado”

Tabela 4.2 – Dispositivos x comandos de fechamento

4.2.1.8 Transição “COM_ABERT”

Assim que esta transição disparar e a guarda “dispMsgAbrir” retornar um valor verdadeiro, a função “send_to_vrml” envia uma mensagem para o mundo virtual, via biblioteca Comms/CPN, fazendo com que a abertura requisitada da seccionadora ou do disjuntor seja efetivada.

***Observação*:** Após seu disparo, uma ficha referente ao dispositivo e seu respectivo estado é depositada no lugar de fusão "Campo". Em seu arco existe a variável "Estado", que é livre. Isto foi feito de forma proposital, para que esta possa assumir um estado (aberto ou fechado), aleatoriamente, possibilitando a simulação de erros ocorridos no campo. Portanto, quando o comportamento do modelo VRML não coincidir com o estado indicado na ficha correspondente, isto evidencia que houve algum problema no campo. Por exemplo, foi enviado um comando para que um disjuntor fosse fechado, mas no campo, esta situação não ocorreu.

4.2.1.9 Transição "COM_FECHA"

Assim que esta transição disparar e a guarda "dispMsgFechar" retornar um valor verdadeiro, a função "send_to_vrml" envia uma mensagem para o mundo virtual, via Comms/CPN, fazendo com que o fechamento requisitado da seccionadora ou do disjuntor seja efetivado.

A observação anterior, também, vale para esta transição.

4.2.1.10 Lugar "Aberta"

Este lugar define a configuração inicial das seccionadoras 41Y3-6, 41Y4-6, 41Y5-6, 41T6_8, 41T7_8 e SC41T4_6, inicialmente em estado aberto.

4.2.1.11 Lugar "Fechada"

Este lugar define a configuração inicial das demais seccionadoras e dos quatro disjuntores, inicialmente fechados.

4.2.1.12 Nó declaração

O nó declaração é a página do modelo onde estão definidas todas as fichas, cores (tipos de dados criados pelo usuário) e funções utilizadas pelo modelo rede de Petri da IHM supervisório. A seguir, serão apresentados estes itens.

i) Implementação da função "send_to_vrml"

A função "send_to_vrml" passa, por meio da string de conexão, a mensagem na forma de fluxo de bytes, para a biblioteca Comms/CPN que, por sua vez, repassa para a classe PollServer, conforme o código a seguir.

Esta mensagem indica se o dispositivo foi aberto ou fechado, ocorrendo assim, a realimentação do mundo virtual.

```
fun send_to_vrml(doing) =
  ConnManagementLayer.send("Conexao",doing,stringEncode);
```

ii) Implementação da função "receive_from_vrml"

A função "receive_from_vrml" recebe, por meio da string de conexão, uma mensagem da biblioteca Comms/CPN que é decodificada para um tipo de dado do Design/CPN, conforme o código a seguir.

A mensagem recebida funciona como comando para abertura ou fechamento do dispositivo. Assim, a rede de Petri principal é realimentada.

```
fun receive_from_vrml()
  ConnManagementLayer.receive("Conexao",stringDecode);
```

iii) Implementação da guarda "dispMsgAbrir"

Neste trecho, a guarda "dispMsgAbrir" avalia se há combinação entre algum dispositivo e mensagem de abertura do mesmo, vinda do mundo virtual. Se houver, então a guarda retorna um valor verdadeiro. O trecho de código a seguir, apresenta a implementação do referido teste.

```
fun dispMsgAbrir(disp,msg) =
   if (((disp=DJ21Y5)    andalso (msg="AbrirDJ21Y5")) orelse
       ((disp=DJ21Y4)    andalso (msg="AbrirDJ21Y4")) orelse
       ((disp=DJ21Y3)    andalso (msg="AbrirDJ21Y3"))orelse
       ((disp=DJ11T4)    andalso (msg="AbrirDJ11T4"))orelse
       ((disp=SC41Y5_6) andalso (msg="AbrirSC41Y56"))orelse
       ((disp=SC31Y5_4) andalso (msg="AbrirSC31Y54"))orelse
       ((disp=SC31Y5_5) andalso (msg="AbrirSC31Y55"))orelse
       ((disp=SC41Y4_6) andalso (msg="AbrirSC41Y46"))orelse
       ((disp=SC31Y4_4) andalso (msg="AbrirSC31Y44"))orelse
       ((disp=SC31Y4_5) andalso (msg="AbrirSC31Y45"))orelse
       ((disp=SC41Y3_6) andalso (msg="AbrirSC41Y36"))orelse
       ((disp=SC31Y3_4) andalso (msg="AbrirSC31Y34"))orelse
       ((disp=SC31Y3_5) andalso (msg="AbrirSC31Y35"))orelse
       ((disp=SC41T4_6) andalso (msg="AbrirSC41T46"))orelse
       ((disp=SC31T4_5) andalso (msg="AbrirSC31T45"))orelse
       ((disp=SC31T4_4) andalso (msg="AbrirSC31T44"))orelse
       ((disp=SC41T6_8) andalso (msg="AbrirSC41T68"))orelse
       ((disp=SC41T7_8) andalso (msg="AbrirSC41T78")))
              then true else false;
```

iv) Implementação da guarda "dispMsgFechar"

Neste trecho, a guarda "dispMsgFechar" avalia se há combinação entre algum dispositivo e mensagem de fechamento do mesmo, vinda do mundo virtual. Se houver, então a guarda retorna um valor verdadeiro. O trecho de código a seguir, apresenta a implementação do referido teste.

```
fun dispMsgFechar(disp,msg) =
   if (((disp=DJ21Y5)    andalso (msg="FecharDJ21Y5")) orelse
       ((disp=DJ21Y4)    andalso (msg="FecharDJ21Y4")) orelse
       ((disp=DJ21Y3)    andalso (msg="FecharDJ21Y3"))orelse
       ((disp=DJ11T4)    andalso (msg="FecharDJ11T4"))orelse
       ((disp=SC41Y5_6) andalso (msg="FecharSC41Y56"))orelse
       ((disp=SC31Y5_4) andalso (msg="FecharSC31Y54"))orelse
       ((disp=SC31Y5_5) andalso (msg="FecharSC31Y55"))orelse
       ((disp=SC41Y4_6) andalso (msg="FecharSC41Y46"))orelse
       ((disp=SC31Y4_4) andalso (msg="FecharSC31Y44"))orelse
       ((disp=SC31Y4_5) andalso (msg="FecharSC31Y45"))orelse
       ((disp=SC41Y3_6) andalso (msg="FecharSC41Y36"))orelse
       ((disp=SC31Y3_4) andalso (msg="FecharSC31Y34"))orelse
       ((disp=SC31Y3_5) andalso (msg="FecharSC31Y35"))orelse
       ((disp=SC41T4_6) andalso (msg="FecharSC41T46"))orelse
       ((disp=SC31T4_5) andalso (msg="FecharSC31T45"))orelse
       ((disp=SC31T4_4) andalso (msg="FecharSC31T44"))orelse
       ((disp=SC41T6_8) andalso (msg="FecharSC41T68"))orelse
       ((disp=SC41T7_8) andalso (msg="FecharSC41T78")))
              then true else false;
```

v) Definição de Variáveis e Cores

Neste trecho do nó declaração, são definidas as variáveis e cores do modelo. A seguir, será explicada cada cor, a partir de [Nasc04].

```
color EDISP = with e;
color MSG = string;
color Dispositivo = with DJ11T4 | DJ21Y3 | SC41Y3_6 | SC31Y3_4 | SC31Y3_5|
                         DJ21Y4 | SC41Y4_6 | SC31Y4_4 | SC31Y4_5|
                         DJ21Y5 | SC41Y5_6 | SC31Y5_4 | SC31Y5_5|
                         SC41T4_6 | SC31T4_5 | SC31T4_4 | SC41T6_8|
                        SC41T7_8 | BT25_3 | BT24_7 | BT23_11;
color Armario = with Prelig | LTP | LT02J6 | LT02J5 | LT02J4 | LT02J3 | LT02J2 | LT02J1;
color Estado  = with ComCartao | SemCartao;
color Operacao = product Armario*Estado;
color Local_Telecomando = with Local | Telecomando | nulo;
color Opcoes_Loc_Tel = product Armario*Local_Telecomando;
color Bot_Comando = with Bot1 | Bot2;
color Opcoes_BCom = product Armario*Bot_Comando;
color Bot_Central = with Bcentral1 | Bcentral2;
color Opcoes_R_C = product Armario*Bot_Central;
color Bot_Reset = with Normal | Bloqueado | nexiste;
color Opcoes_Bot_Reset = product Armario*Bot_Reset;
color CH_GG = with DJGG | b21Y6;
color Opcoes_GG = product Armario*CH_GG;
color Estado_Campo = with Aberto | Fechado;
color Est_Campo_GG = product Armario*CH_GG*Estado_Campo;
color Quadro_Eventos = with Quadro1 | Quadro2;
color Opcoes_Event = product Armario*Quadro_Eventos;
color CH_GPG = with  DJGPG | DJ12J4 | DJ12J5 | DJ12J6 | CS32J5_4 | CS32J5_5 | CS32J5_6 |
CS32J6_4 | CS32J6_5 | CS32J6_6;
color Opcoes_GPG = product Armario*CH_GPG;
color Est_Campo_GPG = product Armario*CH_GPG*Estado_Campo;
color CH_Punho = with CH21Y3 | CH21Y4 | CH21Y5 | DJP | DJ12J1 | DJ12J2 | DJ12J3;
color Opcoes_Punho = product Armario*Dispositivo;
color CH_T_4 = with ch1 | ch2;
color Opcoes_T_4 = product Armario*CH_T_4;
color Est_Campo_P = product Armario*Dispositivo*Estado_Campo;
color CH_V_3 = with Ch_V_3_1 | CH_V_3_2;
color Opcoes_V_3 = product Armario*CH_V_3;
color CH_V_4 = with ch_v_4_1 | ch_v_4_2;
color Opcoes_V_4 = product Armario*CH_V_4;
color CH_mA_4 = with chave_ma_4_1 | chave_ma_4_2;
color Opcoes_A_4 = product Armario*CH_mA_4;
color CH_Transf_3 = with chave_tr_2 | chave_tr_1;
color Opcoes_Tr_3 = product Armario*CH_Transf_3;
color CH_Transf_2 = with chave_tr2_1 | chave_tr2_2;
color Opcoes_Tr_2 = product Armario*CH_Transf_2;
color CH_Sel_mA_2 = with chave_mA_1 | chave_mA_2;
color Opcoes_A_2 = product Armario*CH_Sel_mA_2;
color CH_on_off = with ch_onoff1 | chonoff2;
color Opcoes_onoff = product Armario*CH_on_off;
color CH_Relig_Autom = with Ativado | Desativado | N_consta;
color Opcoes_Rel_Aut = product Armario*CH_Relig_Autom;
var disp: Dispositivo;
var P, Painel, Painel1, Paux: Armario;
var E, Est, Est1, Eaux: Estado;
var LocTelaux, LTaux, LocTel, LT: Local_Telecomando;
var com,comaux: Bot_Comando;
var br, braux: Bot_Reset;
var qe: Quadro_Eventos;
var gg, ggaux: CH_GG;
var camp, estado, campaux: Estado_Campo;
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 c: EDISP;
var msg:MSG;
```

- ***Armario***: representa os armários modelados do ponto de vista da interface
- ***Estado*:** representa os possíveis estados de um dispositivo: aberto ou fechado
- ***Operacao***: é uma dupla *Armario x Estado* que representa o estado de cada dispositivo.
- ***Local_Telecomando***: representa os dispositivos de interação através dos quais os armários podem manobrados de forma local ou telecomandada. Para esta cor há três tipos de fichas: **Local**, quando o painel pode ser manobrado localmente; **Telecomando**, quando o painel é manobrado de forma telecomandada; **Nulo**, quando o painel não apresenta esta chave.
- ***Opcoes_Loc_Tel***: é uma dupla *Armario x Local_Telecomando* que caracteriza o armário onde a chave Loc-Tel está presente, e seu estado.
- ***Bot_Comando***: modela a existência do conjunto de botoeiras, a maioria utilizada para a realização de comandos e uma botoeira comum de confirmação.
- ***Opcoes_BCom***: é uma dupla *Armario x Bot_Comando.* A existência de uma ficha dessa cor implica na presença de um conjunto de botoeiras naquele painel identificado pelo primeiro elemento do par.
- ***Bot_Central***: modela a existência de uma chave de duas posições com referência central.
- ***Opcoes_R_C***: é uma dupla *Armario x Bot_Central* que relaciona o armário à presença da chave de duas posições com referência central.
- ***Bot_Reset***: modela a existência de uma botoeira de reset de proteção.
- ***Opcoes_Bot_Reset***: é uma dupla *Armario x Bot_Reset* que relaciona o armário à presença de uma botoeira utilizada no reset de proteções.
- ***CH_GG***: uma ficha desta cor modela a existência de uma chave do tipo Giro-Giro.
- ***Opcoes_GG***: é uma dupla *Armario x CH_GG* que modela a presença de uma chave do tipo Giro-Giro em no painel representado pelo primeiro elemento do par.

- ***CH_GPG***: uma ficha desta cor modela a existência de uma chave do tipo Giro-Pressão-Giro.
- ***Opcoes_GPG***: é uma dupla *Armario x CH_GPG* que modela a presença de uma chave do tipo Giro-Pressão-Giro em no painel representado pelo primeiro elemento do par.
- ***CH_Punho***: uma ficha desta cor modela a existência de uma chave do tipo punho.
- ***Opcoes_Punho***: é uma dupla *Armario x CH_Punho* que modela a presença de uma chave do tipo punho no painel representado pelo primeiro elemento do par.
- ***Estado_Campo***: as fichas desta cor podem assumir os valores Aberto ou Fechado, modelando assim os estados de dispositivos de disjunção presentes no campo, como disjuntores e seccionadoras.
- ***Est_Campo_GG***: é uma cor formada pela tripla *Armário x CH_GG x Estado_Campo* onde pode-se perceber o estado no campo do dispositivo relacionado à chave e ao painel no qual se encontra o dispositivo de interação.
- ***Est_Campo_GPG***: é uma cor formada pela tripla *Armário x CH_GPG x Estado_Campo* onde o pode-se perceber o estado no campo do dispositivo relacionado à chave e ao painel no qual se encontra o dispositivo de interação.
- ***Est_Campo_P***: é uma cor formada pela tripla *Armário x CH_Punho x Estado_Campo* onde o pode-se perceber o estado no campo do dispositivo relacionado à chave e ao painel no qual se encontra o dispositivo de interação.
- ***Quadro_Eventos***: modela a existência de um quadro anunciador de eventos.
- ***Opcoes_Event***: é uma dupla *Armario x Quadro_Eventos* onde é relacionada a existência de um quadro de eventos naquele painel caracterizado pelo primeiro elemento do par.
- ***CH_V_4***: modela a existência de uma chave seletora de tensão de 4 posições.
- ***Opcoes_V_4***: é uma dupla formada pelo par *Armario x CH_V_4* que modela a presença de uma chave seletora de tensão de 4 posições naquele armário representado pelo primeiro elemento da dupla.

- ***CH_V_3***: modela a existência de uma chave seletora de tensão de 3 posições.
- ***Opcoes_V_3***: é uma dupla formada pelo par *Armario x CH_V_3* que modela a presença de uma chave seletora de tensão de 3 posições no painel representado pelo primeiro elemento da dupla.
- ***CH_T_4***: modela a existência de uma chave de seleção de temperatura de 4 posições.
- ***Opcoes_T_4***: é uma dupla formada pelo par *Armario x CH_T_4* que modela a presença de uma chave seletora de temperatura de 4 posições no painel representado pelo primeiro elemento da dupla.
- ***CH_mA_4***: modela a existência de uma chave de seleção de amperímetro a qual tem 4 posições.
- ***Opcoes_A_4***: é uma dupla formada pelo par *Armario x CH_mA_4* que modela a presença de uma chave seletora de amperímetro de 4 posições no painel representado pelo primeiro elemento da dupla.
- ***CH_mA_2***: modela a existência de uma chave de seleção de amperímetro a qual tem 2 posições.
- ***Opcoes_A_2***: é uma dupla formada pelo par *Armario x CH_mA_4* que modela a presença de uma chave seletora de amperímetro de 2 posições naquele painel representado pelo primeiro elemento da dupla.
- ***CH_Trnsf_2***: modela a existência de uma chave de transferência de proteção de duas posições.
- ***Opcoes_Tr_2***: é uma dupla formada pelo par *Armario x CH_Trasnf_2* que modela a presença de uma chave de transferência de proteção de duas posições no painel representado pelo primeiro elemento da dupla.
- ***CH_Trnsf_3***: modela a existência de uma chave de transferência de proteção de duas posições.

- ***Opcoes_Tr_3***: é uma dupla formada pelo par *Armario x CH_Trasnf_3* que modela a presença de uma chave de transferência de proteção de 3 posições no painel representado pelo primeiro elemento da dupla.
- ***CH_Relig_Autom***: modela a existência de uma chave de religamento automático, tipicamente com duas posições. Pode assumir três estados **Ativado**, indicando que atuará em resposta a uma falta no sistema, **Desativado**, indicando que não irá atuar, e **N_consta** indicando que o armário não dispõe desse dispositivo.
- ***Opcoes_Rel_Aut***: é uma dupla *Armario x CH_Relig_Autom* a qual relaciona cada chave de religamento automático ao painel exibido no primeiro elemento do par.

4.2.2 Modelo Rede de Petri da IHM painel

O modelo CPN, do ponto de vista do painel, é semelhante ao apresentado na Figura 4.5. As definições para lugar "Entrada", transição "FazerConexao", lugar de fusão "IniciaProcesso", transição "Msg_In", lugar de fusão "Saída", transição "COM_ABERT", transição "COM_FECHA" vistos para o modelo IHM supervisório, valem para o modelo da IHM painel.

Entretanto, a este modelo, foram adicionadas novas fichas nos lugares "Aberta" e "Fechada", conforme visto na Figura 4.5, que servirão para controlar o movimento de rotação dos botões do painel, criando-se assim, um nível maior de abstração, visto que nesta representação, a interação é realizada diretamente sobre os dispositivos acessíveis ao operador.

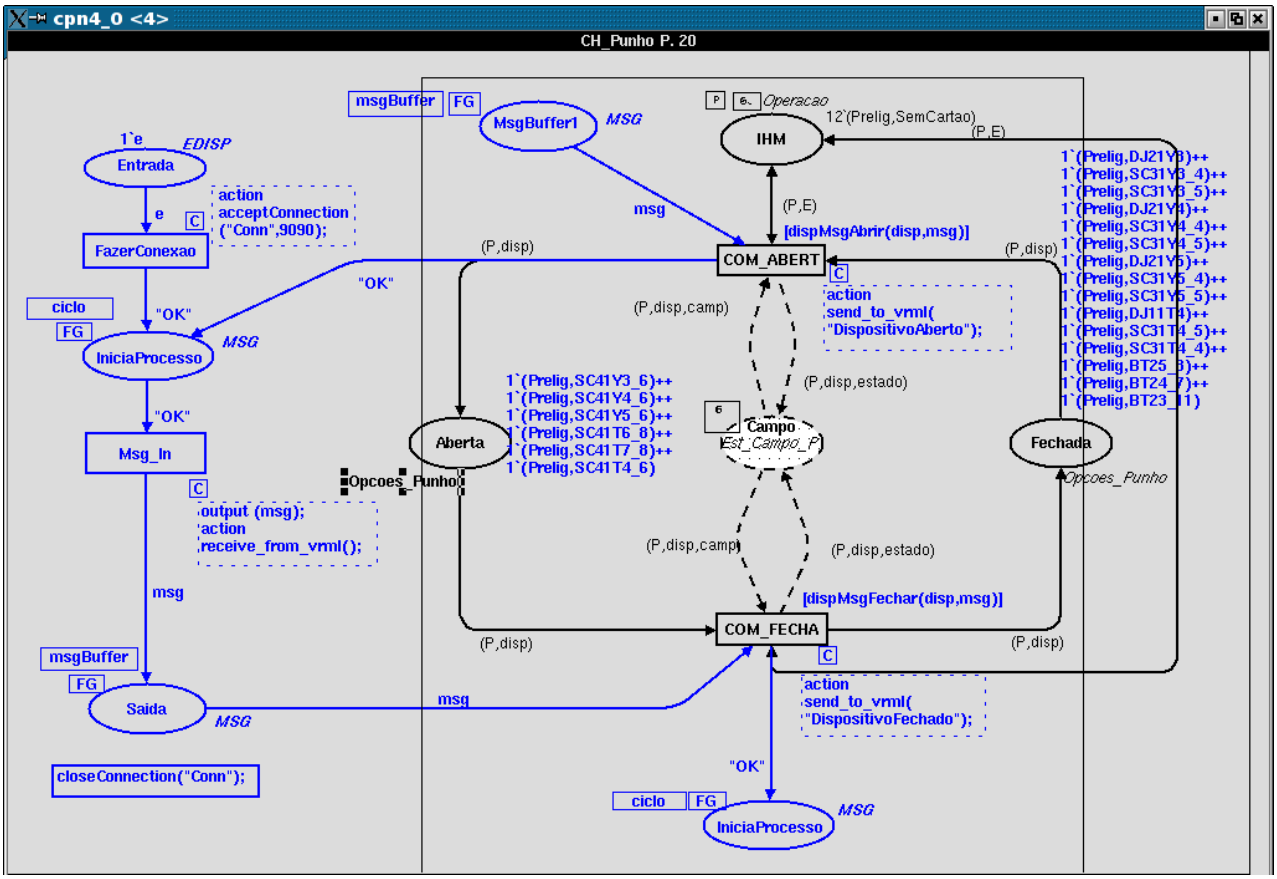

Figura 4.5 – Modelo rede de Petri da IHM painel

4.3 Camada de Comunicação

Nesta seção, será detalhada apenas a classe Java PollServer, visto que somente esta recebeu alterações significativas. As demais classes, JavaCPN, JavaCPNInterface e EncodeDecode que compõem esta camada, não sofreram modificações.

4.3.1 Arquivo *PollServer.java*

Este arquivo é responsável pela comunicação dos modelos redes de Petri da IHM painel e IHM supervisório, com seus respectivos modelos virtuais. Além disso, faz o tratamento das mensagens enviadas e recebidas por ambos.

A seguir é feito um detalhamento deste arquivo no escopo do estudo de caso.

i) Configuração Inicial dos Dispositivos

Este trecho de código apresenta a forma para se definir o estado inicial das seccionados e disjuntores. No trecho de código a seguir, observa-se que o disjuntor 21Y3 está, inicialmente fechado e a seccionadora 41Y3-6, aberta.

```
   this.abertoDJ21Y3 = false;
   this.abertoSC41Y3_6 = true;
```

ii) Estabelecimento da Conexão

Este trecho de código estabelece a conexão entre o visualizados de mundos virtuais FreeWRL e a ferramenta para edição modelos redes de Petri, Design/CPN. A seguir, o referido trecho é comentado detalhadamente:

- A variável cpn_vrml tenta o estabelecimento de conexão entre o Design/CPN e o FreeWRL, a partir da porta 9090;
- Duas exceções são tratadas. A primeira verifica se a *string* de conexão é válida. A segunda avalia se o canal de comunicação está livre.

```
 try
  {
    cpn_vrml = new JavaCPN();
    cpn_vrml.connect(host, 9090);
    System.out.println("Conexao com o Design/CPN, estabelecida com sucesso..." +
newline);
    stop=false;
  }
 catch (UnknownHostException e)
  {
    System.err.println("Host desconhecido " + host);
    stop=true;
  }
 catch (IOException e)
  {
   System.err.println("Erro de E/S ao tentar estabelecer conexao com "+host);
   stop=true;
  }
```

iii) Envio de Mensagem para o Modelo CPN

Neste trecho de código da classe é definido o envio de mensagens do ambiente virtual para o modelo rede de Petri, referentes aos quatro disjuntores e às quatorze chaves seccionadoras.

A seguir, o referido trecho, ilustra a técnica para envio de comando (abertura ou fechamento) do disjuntor 21Y3 ao Design/CPN:

- Verifica se houve clique do disjuntor 21Y3 e se uma mensagem ainda não foi enviada;
    - Se o disjuntor 21Y3 estiver aberto, então o comando a ser enviado será para seu fechamento;
    - Senão o comando a ser enviado será para sua abertura;
- O comando é codificado para um fluxo de bytes;
- O fluxo de bytes é armazenado na variável cpn_vrml;
- O conteúdo da variável "msg_envio_DJ21Y3" é definido como verdadeiro, para indicar que a mensagem foi enviada.

```
//verifica se houve clique do DJ21Y3 e se mensagem não foi enviada
 if (reference.getSensor(1)&&!msg_envio_DJ21Y3)
   {
      String mensagem = "";
      if (abertoDJ21Y3)
         {
             mensagem = "FechaDJ21Y3";
             System.out.println("Fechar Disjuntor DJ21Y3!");
         }
      else
         {
             mensagem = "AbrirDJ21Y3";
             System.out.println("Abrir Disjuntor DJ21Y3");
         }
// chama método da classe EncodeDecode para empacotamento da mensagem
  ByteArrayInputStream paraCPN=EncodeDecode.encode(mensagem);

 // envia mensagem
  cpn_vrml.send(paraCPN);

 // altera cor para verde, ou seja, disjuntor é aberto no mundo virtual
  reference.setcor(reference.green,1);

// indica que a mensagem foi enviada
  msg_envio_DJ21Y3=true;

}
```

iv) Recebimento de Mensagem do Modelo CPN

Nesta parte da classe é definido o recebimento de mensagens do Design/CPN que poderão ser enviadas a partir dos quatro disjuntores e pelas quatorze seccionadoras. Esta mensagem é a confirmação da ação realizada no modelo rede de Petri.

O trecho de código a seguir, exemplifica o tratamento da mensagem para fechamento ou abertura do disjuntor 21Y3.

- Se a variável "msg_envio_DJ21Y3" indicar que o comando para fechamento do disjuntor 21Y3 já foi enviado para o Design/CPN, então,
  - Receber e decodificar a mensagem vinda do Design/CPN;
  - Alterar o conteúdo da variável "msg_envio_DJ21Y3" para falso;
- Se o disjuntor 21Y3 estiver aberto, então o conteúdo da variável "abertoDJ21Y3" é alterado para falso e a cor do disjuntor é alterada para vermelha;
- Senão o conteúdo da variável "abertoDJ21Y3" é alterado para verdadeiro e a cor do disjuntor é alterada para verde.

```
if (msg_envio_DJ21Y3)
  {
   // recebe pacote do modelo CPN e o converte para objeto de dados
   deCPN=EncodeDecode.decodeString(cpn_vrml.receive());

   // msg_envio_DJ21Y3 é setado para falso, possibilitando o início de outro ciclo
   msg_envio_DJ21Y3 = false;

   //verifica se o disjuntor DJ21Y3 está aberto
   if (abertoDJ21Y3)
     {
        reference.setcor(reference.red,1);
        abertoDJ21Y3 = false;
        System.out.println("Disjuntor DJ21Y3 Fechado!");
     }
   else
     {
        reference.setcor(reference.green,1);
        abertoDJ21Y3 = true;
        System.out.println("Disjuntor DJ21Y3 Aberto!");
     }
  }
```

4.4 Camada de Visualização

Nesta seção, será apresentado o *script* para definição dos disjuntores, seccionadoras, botões, armários, dentre outros detalhes que compõem o modelo VRML.

4.4.1 Modelo VRML da IHM supervisório

4.4.1.1 Arquivo *supervisorio.wrl*

Este é o arquivo de *script* VRML responsável pela representação, em ambiente virtual, da IHM supervisório. A seguir, está a estrutura do mesmo.

i) Representação de Circuito

Este trecho define:

- a cor do circuito;
- os pontos nos eixos x, y e z, que alinhados, formam uma reta;
- as coordenadas para a plotagem das retas que formarão os circuitos dos supervisório.

```
Shape {
 appearance Appearance {
  material Material {
   emissiveColor 0.0 1.0 1.0
  }
 }
 geometry IndexedLineSet {
  coord Coordinate {
   point [
       -5.0  0.0  1.0,
        5.0  0.0  1.0,
        5.0  0.0 -1.0,
       -5.0  0.0 -1.0,
       -5.0 -2.0  1.0,
       -3.0 -2.0  1.0,
        5.0 -2.0 -1.0,
        5.0 -2.0 -1.0,
       -5.0  0.5  1.0,
       -5.0  2.5  1.0,
       -5.0  2.5  1.0,
       -3.0  2.5  1.0,
       -3.0  2.0  1.0,
       -3.0  3.0  1.0,
       -3.0 -1.5  1.0,
       -3.0 -2.0  1.0,
       -3.0 -1.0  1.0,
       -3.0 -0.2  1.0,
       -3.0  0.7  1.0,
       -3.0  1.5  1.0,
       -6.0  3.0  1.0,
        5.0  3.0  1.0,
       -3.0 -2.0  1.0,
       -3.0 -2.5  1.0
   ]
  }
  # Plotagem do circuito
  coordIndex [
  0,4,-1,4,5,-1,8,9,-1,10,11,-1,12,13,-1,14,15,-1,16,17,-1,18,19,-1,20,21,-1,22,23,-1,
  24,25,-1,26,27,-1,28,29,-1
  ]
 }
}
```

ii) Representação de Disjuntor

Este trecho de código define:

- a posição inicial do disjuntor 21Y3, nos eixos x, y e z;
- o método *TouchSensor* que torna disjunto 21Y3 sensível ao clique do *mouse*;
- cor inicial (vermelha) do disjuntor 21Y3;
- a forma geométrica (cubo) do disjunto 21Y3.

```
# Defição da forma geométrica de um disjuntor
Transform {

  translation -3.0 0.3 1.0
      children [
        DEF CliqueDJ2Y3 TouchSensor {}
        Shape {
          appearance Appearance {
            material DEF MudaCorDJ21Y3 Material { diffuseColor 1 0 0 }
          }
          geometry Box {size 0.7 1.0 0.0}

        }
      ]
   }
```

iii) Representação de Seccionadora

Este trecho de código define:

- a posição inicial da seccionadora, nos eixos x, y e z;
- o ângulo de rotação inicial da seccionadora;
- o centro de rotação da seccionadora;
- a forma geométrica da seccionadora;
- o método *TouchSensor* que torna a seccionadora sensível ao clique do *mouse*;
- o tempo de duração da animação da seccionadora;
- o movimento da seccionadora (abertura ou fechamento).

```
Group{
 # Definição da forma geométrica da uma seccionadora
 children [
  DEF Cube Transform {
   translation -4.9 0.25 1.1
   rotation 1.0 1.0 8.0 1.0
   center 0.0 -0.30 0.0
   children Shape {
    appearance Appearance {
     material Material {}
    }
    geometry Box  {size 0.2 0.5 0.0}
   }
  },

  # Sensor que detecta quando o usuário clica em um objeto
  DEF Touch TouchSensor { },

  # Relógio para controlar a animação
  DEF Clock1 TimeSensor {
   enabled FALSE
   cycleInterval 4.0
   loop TRUE
  },

  # Caminho da animação (movimento de fechamento)
  DEF CubePath OrientationInterpolator {
   key [0.0, 1.0]
   keyValue [
    0.0 1.0 0.0 0.0,
    1.0 1.0 8.0 1.0
   ]
  }

  # Caminho da animação (movimento de abertura)
  DEF CubePathVolta OrientationInterpolator {
   key [0.1, 0.0]
   keyValue [
    1.0 1.0 8.0 1.0,
    0.0 1.0 0.0 0.0
   ]
  }

 ]
}
```

iv) Definição dos Eventos de Entrada e Saída

Este trecho de *script* possui dois tipos eventos: um de entrada *(EventIn)* que trata dos eventos provocados pelo mundo virtual (um clique sobre um disjuntor ou seccionadora) e outro do tipo saída *(EventOut)* que trata dos eventos provocados pelo modelo rede de Petri (abertura ou fechamento de uma seccionadora e mudança de cor de um disjuntor).

Neste trecho do arquivo supervisorio.wrl, também, é invocada a classe AnimaSupervisorio que implementa as animações no mundo virtual.

A seguir, é apresentada uma exemplificação do que foi explanado:

- A classe AnimaSupervisorio é instanciada;
- Definição das variáveis cliqueDJ21Y3 e cliqueSC41Y3_6 para tratamento dos eventos de entrada do disjuntor 21Y3 e da seccionadora 41Y3-6. Eventos esses, ocorridos após um clique.
- Definição das variáveis abreSC41Y3_6 e fechaSC41Y4_6 para tratamento dos eventos de saída das seccionadoras 41Y3-6 e 41Y4-6. Eventos esses que promoverão a alteração do comportamento do objeto, no ambiente virtual.

```
DEF AnimaMundo Script {
      url "AnimaSupervisorio.class"

      eventIn Time cliqueDJ21Y3
      eventIn Time cliqueSC41Y3_6

      eventOut Animacao abreSC41Y3_6
      eventOut Animacao fechaSC41Y4_6
   }
```

v) Definição das Rotas

Este é o trecho final do *script* e é responsável pelo movimento, propriamente dito, dos objetos. A seguir, é apresentado um detalhamento deste código.

- A partir do clique da seccionadora 21Y3-6 uma mensagem é enviada para o modelo rede de Petri;
- Uma mensagem é enviada da rede de Petri, indicando que a seccionadora 21Y3-6 deve ser rotacionada.

```
# Esta rota trata a mensagem enviada para o Design/CPN

ROUTE TouchSC21Y3_6.touchTime          TO Clock.set_startTime
ROUTE TouchSC21Y3_6.touchTime          TO AnimaMundo.cliqueSC21Y3_6


# Esta rota trata a mensagem vinda do Design/CPN

ROUTE AnimaMundo.fechaSC21Y3_6         TO PathAbreSC21Y3_6.set_fraction
ROUTE PathAbreSC21Y3_6.value_changed   TO RotacionaSC21Y3_6.set_rotation


# Esta rota verifica, a cada instante, se houve algum clique. Se houve, então, uma
ação é enviada para a classe AnimaSupervisorio.
ROUTE Clock.cycleTime TO AnimaMundo.interval
```

4.4.1.2 Arquivo *AnimaSupervisorio.java*

Este arquivo é responsável pela animação dos disjuntores e seccionadoras. Nele são definidas as mudanças de cores dos disjuntores para indicação de abertura ou fechamento dos mesmos, assim como o movimento de rotação das seccionadoras. O arquivo *.class* gerado é invocado pelo arquivo *supervisorio.wrl.*

i) Definição das Variáveis Sensores

Estes sensores indicam se um objeto (disjuntor ou seccionadora) foi clicado, no modelo VRML.

```
boolean sensorDJ21Y3Ativado = false;
boolean sensorSC41Y3_6Ativado = false;
```

ii) Definição das Cores dos Disjuntores

Este trecho define a cor do disjuntor, no padrão RGB.

```
float corDJ21Y3[] = { 1, 0, 0 };
float corDJ21Y4[] = { 1, 0, 0 };
float corDJ21Y5[] = { 1, 0, 0 };
```

iii) Verificação de Ativação dos Sensores

Este trecho identifica se o disjuntor 21Y3 está aberto. Se estiver, então deve ser fechado, caso contrário deve ser aberto.

```
// se o dispositivo for DJ21Y3 e este estiver aberto, então é fechado.
if (dispositivo==1)
  {
   if (sensorDJ21Y3Ativado)
     sensorDJ21Y3Ativado = false;
   else
     sensorDJ21Y3Ativado = true;
  }
```

iv) Retorno de Valor do Sensor

Este método é invocado pela classe PollServer e retorna o estado do disjuntor 21Y3.

```
// Método invocado pela classe PollServer
public boolean getSensor(int disjN)
{
  if (dispositivo==1)
     return sensorDJ21Y3Ativado;
}
```

v) Instanciação de Variáveis de Evento Externo

Estas variáveis tratam as mensagens vindas do Design/CPN e provocam uma alteração no mundo virtual, por meio da abertura, fechamento ou mudança de cor de algum dispositivo. No trecho a seguir, é apresentada a forma para mudança da cor do disjuntor 21Y3 e a forma para abertura e fechamento da seccionadora 41Y3-6.

```
mudaCorDJ21Y3  = (Animacao) getEventOut("mudaCorDJ21Y3");
abreSC41Y3_6   = (Animacao) getEventOut("abreSC41Y3_6");
fechaSC41Y3_6  = (Animacao) getEventOut("fechaSC41Y3_6");
```

vi) Avaliação de Variáveis de Evento Interno

Esta condição avalia as variáveis de evento interno. Ou seja, as variáveis que possuem mensagens vindas do modelo VRML.

```
//verifica se o objeto clicado, no modelo VRML, foi o disjuntor DJ21Y3
if(e.getName().equals("cliqueDJ21Y3") == true)
 {
   this.Sensor(1);
 }
```

vii) Fechamento de Seccionadora

```
if (N==1)
 {
   fechaSC41Y3_6.setValue(dispSec);
 }
```

viii) Abertura de Seccionadora

```
if (N==1)
 {
   abrirSC41Y3_6.setValue(dispSec);
 }
```

ix) Mudança de Cor de Disjuntor

```
if (disjN == 1)
 {
    corDJ21Y3 = ncor;
 }
```

Na figura 4.6 é apresentado o modelo virtual do ambiente virtual da IHM supervisório, de acordo com o diagrama unifilar da Figura 4.3.

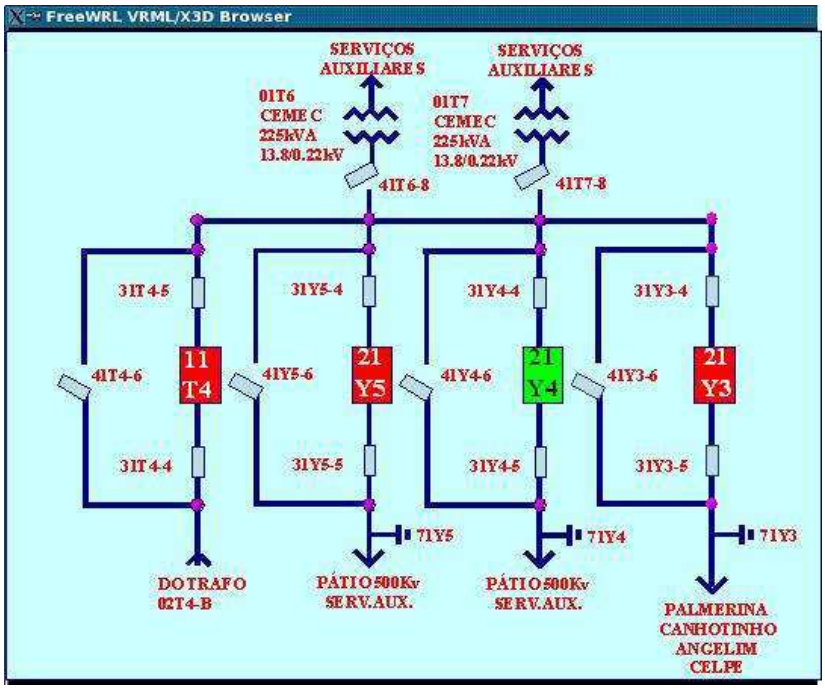

Figura 4.6 – Modelo VRML da IHM supervisório

4.4.2 Modelo VRML da IHM Painel

A representação do modelo da IHM painel é feita de forma similar à representação da IHM supervisório, ou seja, sua composição partiu, também, do diagrama unifilar. Entretanto, esta representação é mais natural e intuitiva ao operador. A Figura 4.7 apresenta três armários, em que cada um é formado de relógios mostradores, textos indicativos e botões do tipo giro-pressão-giro que representam as seccionadoras 41Y5-6, 31Y5-4, 31Y5-

5, 41Y4-6, 31Y4-4, 31Y4-5, 41Y3-6, 31Y3-4, 31Y3-5, 41T6-8, 41T7-9. Os três botões restantes representam os disjuntores.

Para cada botão, há movimento (abertura ou fechamento) e mudança de cor (chave aberta: verde; chave fechada: vermelha; chave sendo pressionada: amarela). A figura 4.8, ilustrar o painel real.

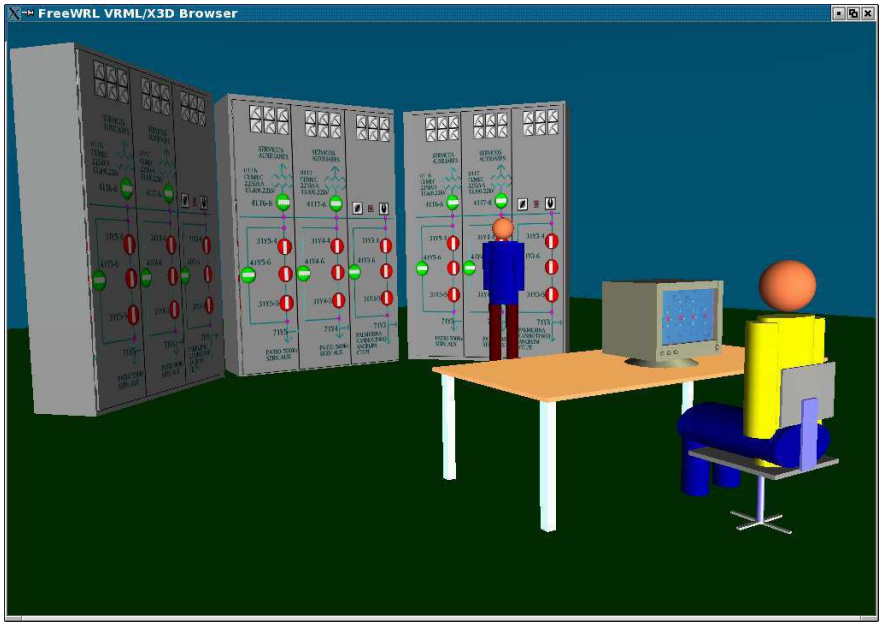

Figura 4.7 – Sala virtual dos painéis.

Figura 4.8 – Sala de painéis da subestação elétrica.

As Figuras 4.9, 4.10, 4.11, 4.12 e 4.13 apresentam outras perspectivas de vistas do modelo VRML da IHML painel.

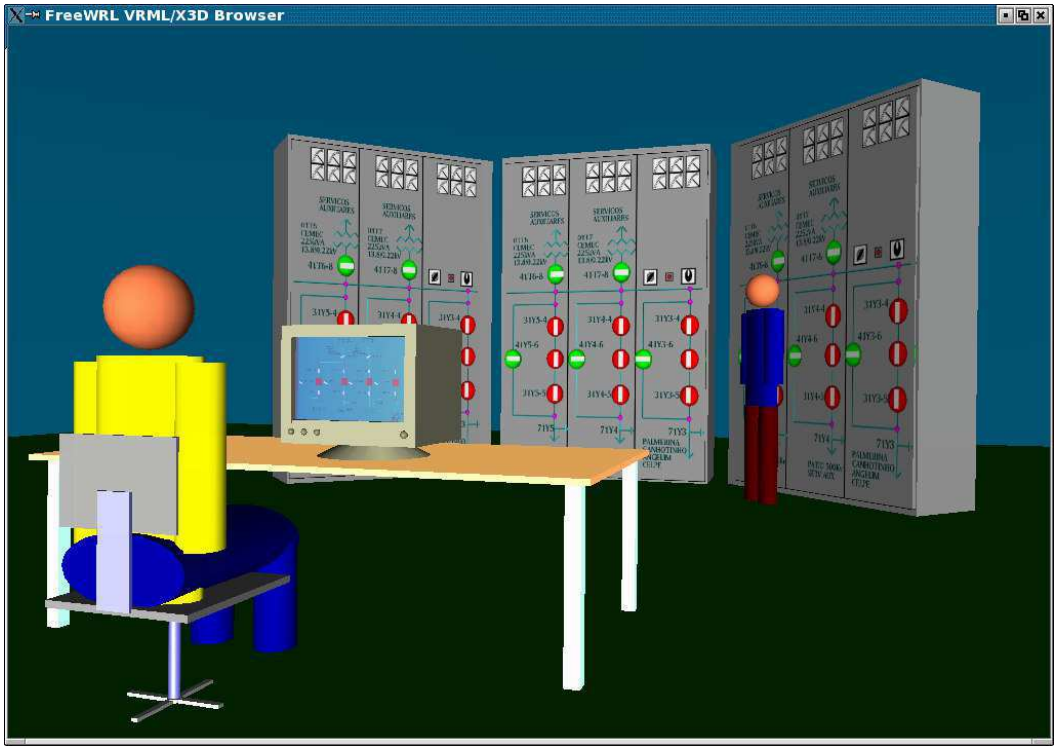

Figura 4.9 – Vista do operador da direita.

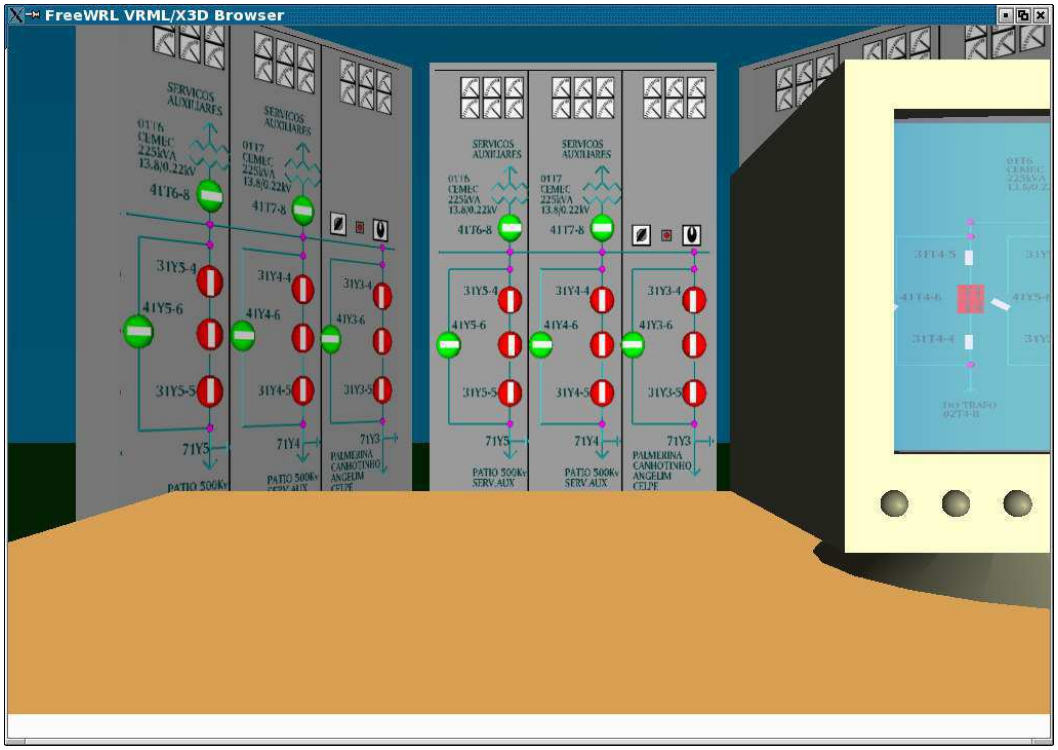

Figura 4.10 – Vista da mesa.

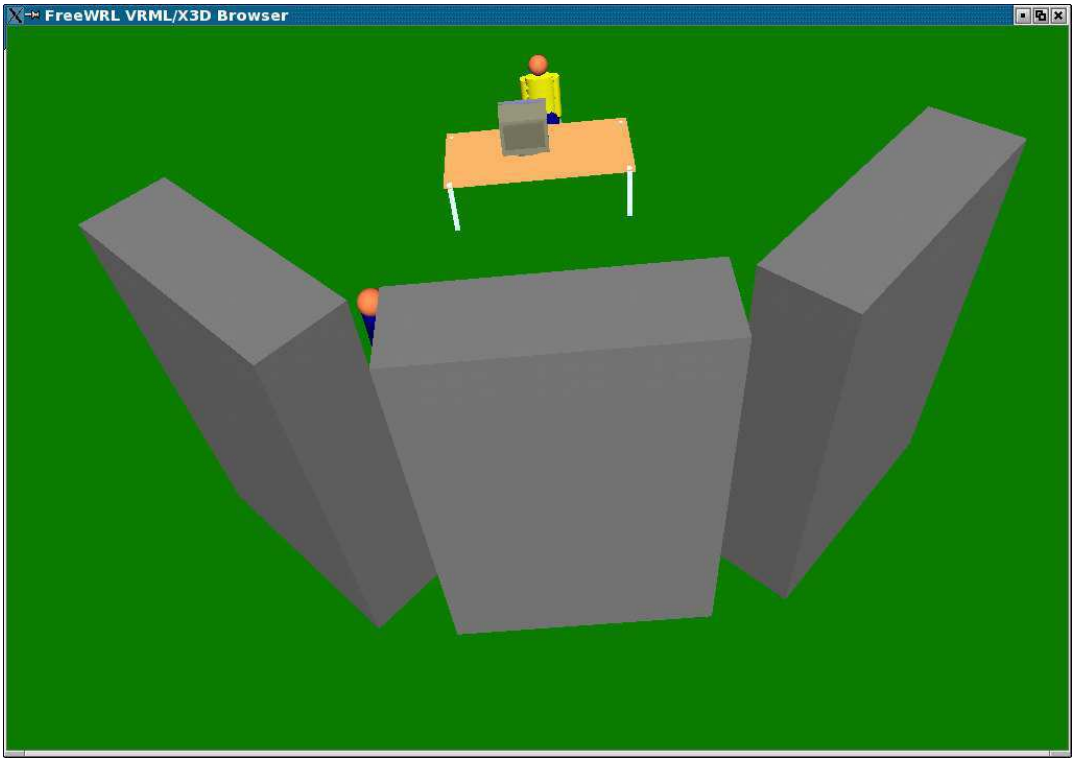

Figura 4.11 – Vista da parte de trás dos armários.

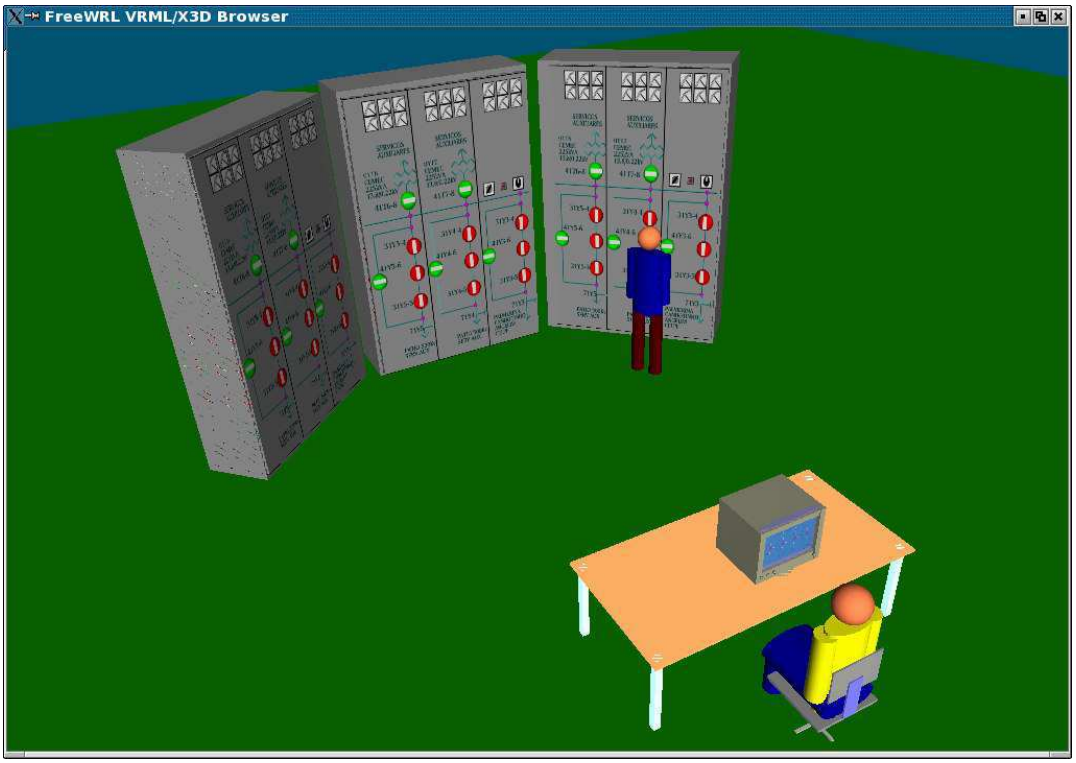

Figura 4.12 – Vista perspectiva da sala.

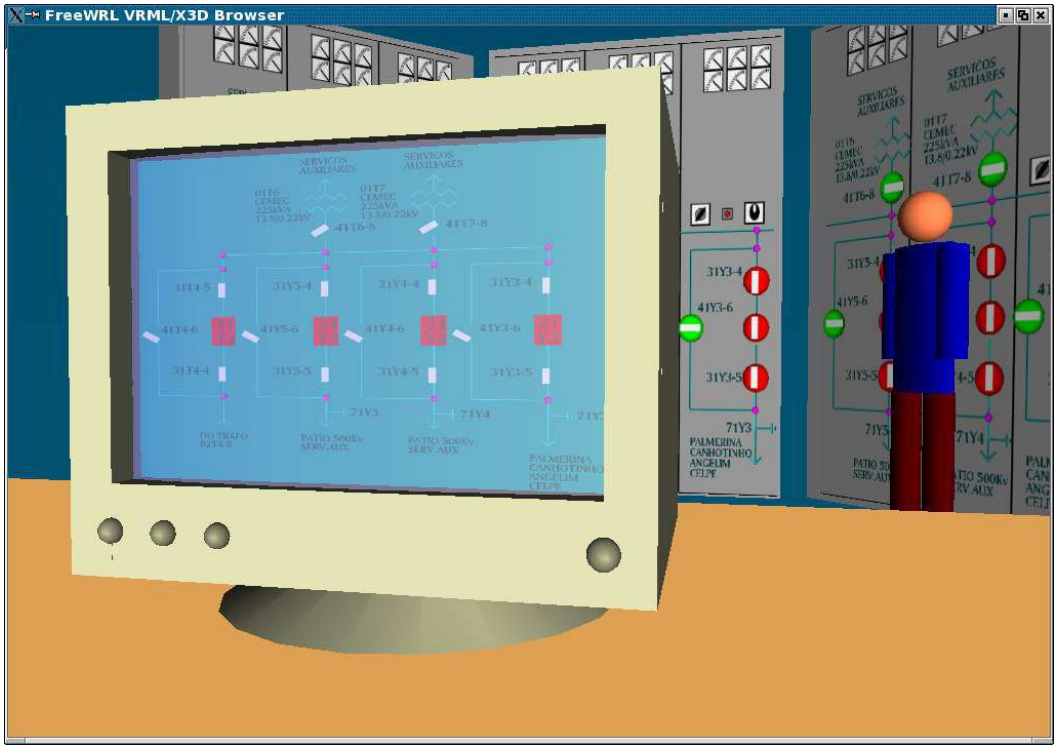

Figura 4.13 – Vista do computador

Capítulo 5 – Considerações Finais

Neste capítulo, será realizada uma discussão sobre os objetivos alcançados, bem como serão relatadas as conclusões à respeito da solução proposta para visualização, em ambiente tridimensional, dos comportamentos de modelos formais. Ao final, serão apresentadas sugestões para trabalhos futuros.

5.1 Conclusões

Este trabalho apresenta um mecanismo para a comunicação entre dois tipos de modelos. Um modelo formal que corresponde à teoria de redes de Petri coloridas e um modelo responsável pela visualização do comportamento.

Para a comunicação entre modelos e visualização do comportamento é definido um mecanismo de integração, dividido em três camadas. Uma camada responsável pelos modelos redes de Petri coloridas, outra camada responsável por modelos virtuais e uma camada intermediária que possibilita a troca de mensagens entre as outras duas.

É definida, também, uma seqüência de lugares e transições, modelada em rede de Petri colorida que possui agregadas, funções utilizadas pela biblioteca Comms/CPN. Esta seqüência, denominada "Fluxo de Conexão", é responsável por estabelecer e manter um canal de comunicação entre o modelo formal e um processo externo, receber mensagem do processo externo e realimentar a rede de Petri principal.

Por estas características, é possível utilizar o "Fluxo de Conexão" em diversos contextos, cujo objetivo seja a comunicação com processos externos. Uma restrição para a sua utilização é a de que o processo externo utiliza o protocolo TCP/IP. E um aspecto a ser

ressaltado é que o envio da mensagem para o processo externo deve ser de responsabilidade de uma ou mais transições da rede principal.

O problema proposto neste trabalho foi visualizar o comportamento de um conjunto de redes de Petri, responsável por modelar o comportamento da interface disponível ao operador de uma subestação elétrica. O objetivo foi disponibilizar aos operadores, em treinamento no simulador, um ambiente intuitivo e realístico, que não requisite conhecimentos sobre os formalismos utilizados na construção do simulador.

O mecanismo de integração proposto, mostrou-se adequado como uma solução a este problema pelos seguintes motivos: (i) o formalismo da rede de Petri é tratado pela camada de modelos, onde não é necessária a interação com o operador; (ii) a visualização do comportamento das redes de Petri é realizada pela camada de visualização, onde estão os modelos VRML que oferecem uma representação realística dos objetos manipulados pelo operador, abstraindo detalhes técnicos e facilitando o seu entendimento; (iii) a camada de comunicação promove troca bidirecional de mensagens entre os dois tipos de modelos; (iv) a camada de comunicação do mecanismo de integração permitiu a comunicação eficaz entre os dois tipos de modelos (VRML e CPN).

5.2 Sugestões para Trabalhos Futuros

No estudo de caso proposto neste trabalho, houve a representação de apenas um subconjunto das operações modeladas pelas redes de Petri do simulador, cuja finalidade foi validar a solução proposta. Todavia, faz-se necessária a sua expansão para todo o conjunto, de forma que os comportamentos de todos os dispositivos modelados sejam refletidos nos modelos VRML.

Com a natural expansão do modelo, é interessante utilizar a abordagem de múltiplas conexões (por exemplo, utilizar um mesmo modelo rede de Petri para estabelecer comunicação com mais de um modelo virtual), o que implica em promover adaptações ao fluxo de conexão.

Outra sugestão é usar os recursos para modelagem acústica e visual (VRML) para agregar características sensoriais aos objetos modelados nos ambientes virtuais,

propiciando uma facilidade adicional ao entendimento do usuário final, lançando-se mão da utilização de acessórios como: luvas e óculos.

Um outro tema para trabalho futuro é adaptar o mecanismo de integração para o ambiente Windows, isto é, transportar os modelos redes de Petri do Design/CPN para o *CPN Tools*. Entretanto, deve ser observado que o *CPN Tools* apenas faz a simulação dos modelos, não possibilitando a análise dos mesmos (geração de gráficos de ocorrência, por exemplo).

Finalmente, um outro tema para trabalho futuro, diz respeito à criação de uma interface visual para as classes PollServer e AnimaMundo, possibilitando a manipulação de seus métodos sem ter que, necessariamente, codificá-los. Por exemplo, uma mensagem enviada para a rede de Petri, cujo comando fosse para abertura de uma chave, poderia ser definida por meio de uma caixa de texto definida numa tela. O *script* VRML também teria uma interface, cuja finalidade seria a definição das rotas dos objetos. As informações informadas (mensagens e definições de rota) seriam armazenadas em um banco de dados.

Espera-se com este trabalho ter contribuído para a integração de modelos e mais especificamente na construção do simulador em desenvolvimento no LIHM.

Referências Bibliográficas

[Bar05] BARRETO NETO, J.P., *Modelagem de uma Subestação Elétrica para Apoio ao Estudo de Situações Críticas*, Relatório de Projeto da Disciplina Modelagem e Validação de Sistemas Usando Redes de Petri, 2005

[BJ04] BOSSEN, C.; JØRGENSEN, J.B., *Context-descriptive prototypes and their application to medicine administration*, In DIS '04: Proc. of the 2004 conference on Designing interactive systems, pages 297-306, Boston, MA, USA, 2004, ACM Press.

[BPF99] BRAGA, J.D.M., PERKUSICH, A., FIGUEIREDO, J.C.A., *Tutorial sobre Redes de Petri*, 1999.

[BC94] BURDEA, G.; COIFFET, P., *Virtual Reality Technology*, John Wiley & Sons. New York, NY, 1994.

[CW96] CLARKE, E.M., WING, J.M., *Formal Methods: State of the art and future directions*, ACM Workshop on Strategic Directions in Computing Research, *ACM Computing Surveys*, vol. 28, no. 4, 1996, pp. 626-643.

[Des93] *Design/CPN* Tutorial for X-Windows – Version 2.0. Meta Software Corporation, Cambridge, MA, http://www.daimi.au.dk/designCPN/man/Tutorial/Tutorial.Contents.pdf , acessado em junho de 2006.

[FTPX06] FREITAS, R.C., TURNELL, M.F.Q.V, PERKUSICH, A., XAVIER, C.S.C.L., *Mecanismo para Visualización y Comunicación Bidireccional*

entre Modelos Redes de Petri Coloreadas y Modelos en Realidad Virtual, Conferencia Internacional Convención FIE 06, Santiago de Cuba, 2006.

[GK01] GALLASCH, G., KRISTENSEN, L.M., *Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN*, In 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01), pp. 75-90, Aarhus University, 2001

[ExSp1] http://www.exspect.com, acessado em junho de 2006.

[ExSp2] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db/exspect.html, acessado em junho de 2006.

[RH92] ROBINETT, W., HOLLOWAY, R., *Implementation of Flying, Scaling, and Grabbing in Virtual Worlds*, Symposium on Interactive 3D Graphics, Cambridge, MA, pp. 189-192, 1992

[Isd02] ISDALE, J., *What is Virtual Reality - A Homebrew Introduction and Information Resource List*, http://www.isdale.com/jerry/VR/WhatIsVR.html, 2002, acessado em fevereiro de 2006.

[Jac94] JACOBSON, L., *Realidade Virtual em Casa*, Rio de Janeiro, Berkeley, 1994.

[Jen97] JENSEN, K., *Coloured Petri Nets – Basic Concepts, Analysis Methods and Pratical Use.* Volume 1. Second Edition.

[KP04] KINDLER, E., PÁLES, C., *3D-Visualization of Petri Net Models: Concept and Realization*, In ICATPN 2004, pp 464-473, 2004.

[KT04] KIRNER, C., TORI, R., *Realidade Virtual – Conceitos e Tendências*, 1ª edição, volume único, editora SBC, São Paulo.

[Kir05] KIRNER, C., *Tutorial de Sistemas de Realidade Virtual. Universidade Federal de São Carlos*, Departamento de Computação, Grupo de Pesquisa em Realidade Virtual, 2005.

[KJ04] KRISTENSEN, L.M., JENSEN, K, *Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad-hoc Networks*. In Integration of

Software Specification Techniques for Application in Engineering, volume 3147 of LNCS, pp 248-269, Springer-Verlag, 2004.

[Mar05] MARRANGHELO, N., *Redes de Petri: Conceitos e Aplicações. Universidade Estadual Paulista*. http://www.dcce.ibilce.unesp.br/~norian/cursos/mds/ApostilaRdP-CA.pdf, acessado em junho de 2006.

[MB98] MARQUES, E. F., BRAMORSKI, M.M., *Apostila de VRML Básico*. Universidade Federal de Santa Catarina. 1998.

[Mur89] MURATA, T., *Petri Nets: Properties, Analysis, And Applications*, In Proceedings of the IEEE, Vol. 77, No. 4, pp. 541-579, 1989.

[Nasc04] NASCIMENTO NETO, J.A., *Modelagem da Interface Homem-Máquina de uma Subestação Elétrica*, Dissertação de Mestrado, Curso de Pós-Graduação em Engenharia Elétrica, UFCG, 2004.

[Nasc05] NASCIMENTO, J. A.N., TURNELL, M. F. Q. V., *Biblioteca de Modelos de Objetos da IHM de Sistemas Eletricos em HCPN*. In: Simpósio Brasileiro de Automação Inteligente, 2005, São Luís. VII Simpósio Brasileiro de Automação Inteligente, 2005

[Net02] NETTO, A. V. et. al., *Realidade Virtual – Fundamentos e Aplicações*. Visual Books, pp 45-65, 2002.

[PT95] PIMENTEL, K.; TEIXEIRA, K, *Virtual reality - through the new looking glass*. 2nd.edition, New York, McGraw-Hill, 1995.

[Red04] Curso Redes de Computadores – Internet e Arquitetura TCP/IP – PUC Rio de Janeiro, http://www.rjunior.com.br/download/tcp2.pdf, 2004, acessado em junho de 2006.

[RS95] RASMUSSEN, J. L., SINGH, M., *Mimic/CPN – A Graphic Animation* Utility for Design/CPN, Computer Science Department, Aarhus University, 1995.

[STP01] SCAICO, A., TURNELL, M. F. Q.V., PERKUSICH, A. *Modelagem da Navegação de Interfaces com o Usuário de Sistemas de Automação Industrial*. V SBAI – Simpósio Brasileiro de Automação Inteligente, novembro de 2001.

[Sch95] SCHLOERB, D.W. - *A Quantitative Measure of Telepresence*, In Presence – Teleoperators and Virtual Environments, pp 64-80, 1995.

[SP03] SILVA, L.D., PERKUSICH, A., *Uso de Realidade Virtual para Validação de Modelos de Sistemas Flexíveis de Manufatura*. VI Simpósio Brasileiro de Automação Inteligente, Bauru, 2003.

[Sha93] SHAW, C. et al. *Decoupled Simulation in Virtual Reality with MR Toolkit*, ACM Transection of Information Systems, pp 287-387, 1993.

[Tur01] Turnell, M. F. Q. V.; Scaico, A.; Sousa, M. R. F., Perkusich, A. *Industrial User Interface Evaluation Based On Coloured Petri Nets Modelling and Analysis. Lecture Notes in Computer Science - Interactive Systems*, LNCS 2220, p. 69-87, Germany, 2001

[WW92] WATT, A., WATT, M. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley/ACM Press, pp 297-298, New York, 1992.

[WL05] WESTERGAARD, M. and LASSEN, K.B., *Building and Deploying Visualizations of Coloured Petri Net Models Using BRITNeY Animation and CPN Tools*. Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2005.

Anexo – Verificação da Execução da Aplicação

Neste anexo, será exemplificado um passo a passo da execução das operações de abertura e fechamento das chaves, apresentando o comportamento de cada uma. Além disso, será vista a configuração do Design/CPN, utilizada exemplo.

1. Configuração Básica do Design/CPN

1.1 Ambiente de Conexão

Esta janela é responsável pela configuração do tipo de conexão (local ou remota), caminho de execução e imagem do ML. As definições apresentadas, na Figura 4.2, são as utilizadas neste estudo de caso.

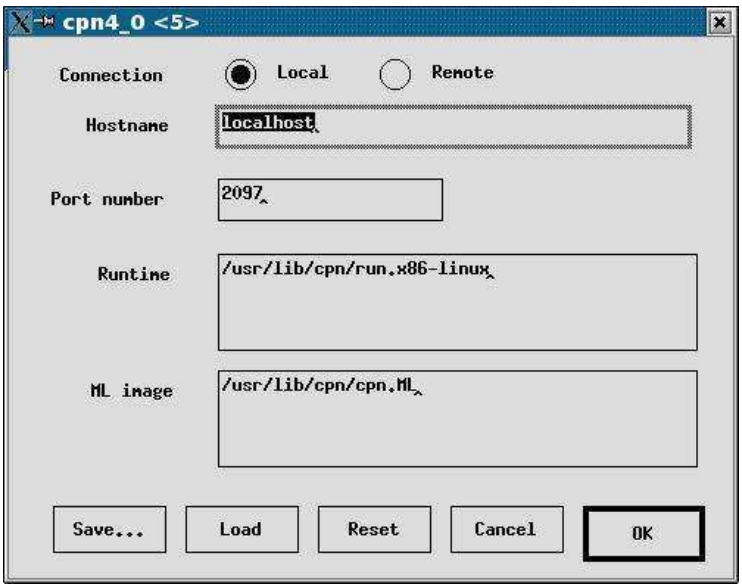

Figura A.1 – Configuração ML *(Set → ML Configuration → Options)*

1.2 Simulação de Código

Para este estudo de caso, o ambiente de simulação foi definido para permitir os modos automático e interativo, sem temporização e com possibilidade de execução de segmentos de código.

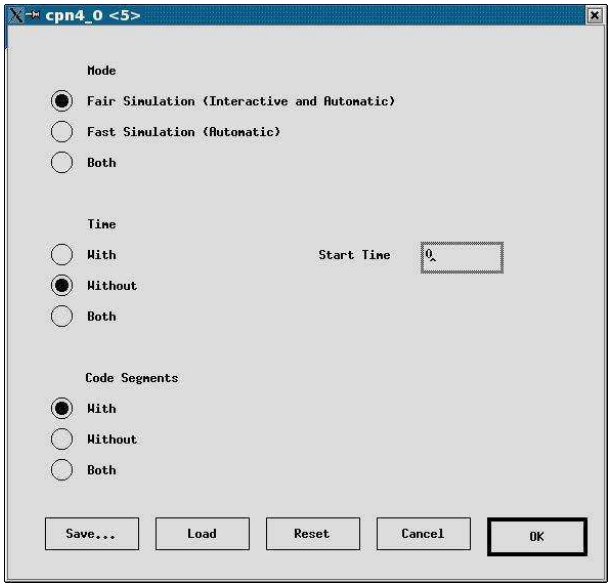

Figura A.2 – Simulação Código *(Set → Simulation Code → Options)*

1.3 Simulação Interativa

Para este exemplo, o ambiente foi configurado de forma que, a cada passo executado, haja atualização da Rede.

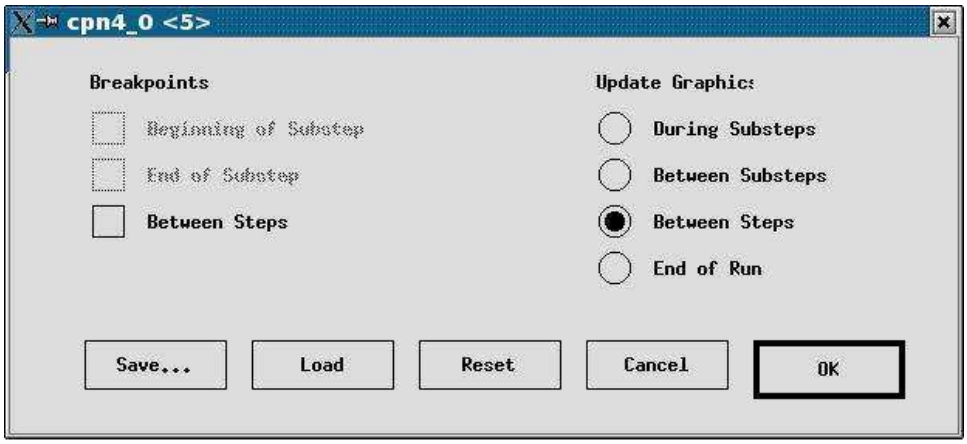

Figura A.3 – Simulação Interativa *(Set → Interactive Simulation → Options)*

2. Abertura de Disjuntor

O disjuntor utilizado para este exemplo, será o DJ21Y3 (localizado no conjunto de armário central, painel da direita, chave vermelha do meio), conforme visto na Figura A.5. Todavia, este roteiro serve, também, para os disjuntores DJ21Y4 e DJ21Y5.

1. **Entrar no diretório onde está o modelo CPN**
2. **Abrir o modelo, no Design/CPN**
3. **Entrar no modo simulador**

File → Enter Simulator

4. Preparar conexão entre Modelo CPN e Modelo VRML

4.1 Clicar sobre transição "FazerConexao" (ver Figura A.4)

4.2 Verificar Bind *(Alt+B)*

4.3 Clicar em *Occur* (neste instante, o modelo Rede de Petri espera conexão com o modelo VRML).

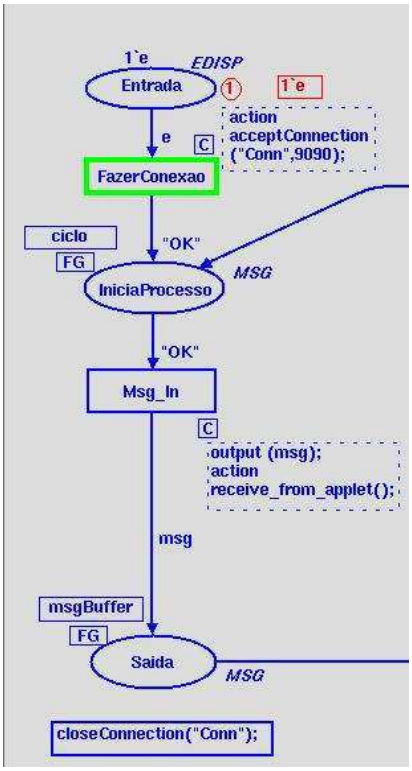

Figura A.4 – Transição "FazerConexao" habilitada

5. Abrir outro terminal (console)

6. Entrar no diretório onde está o *script* do modelo VRML

7. Rodar o *script* VRML da IHM painel

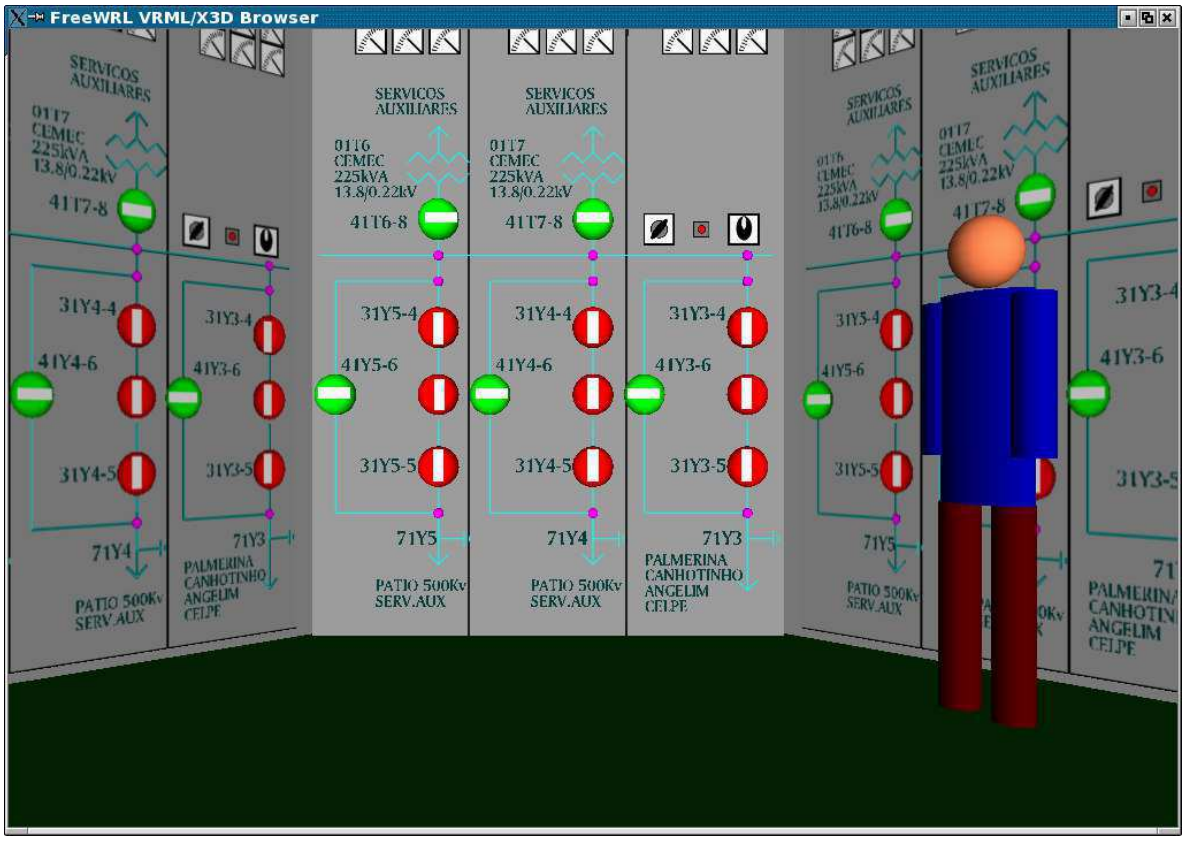

Figura A.5 – Modelo VRML da IHM painel em seu estado inicial

8. Voltar para o Design/CPN

9. Preparar modelo Rede de Petri para receber mensagem do modelo VRML

9.1 Clicar sobre transição “Msg_In” (ver Figura A.6)

9.2 Verificar Bind *(Alt+B)*

9.3 Clicar em *Occur* (neste instante, é esperada que uma interação com a chave DJ21Y3, representada no modelo VRML, seja realizada).

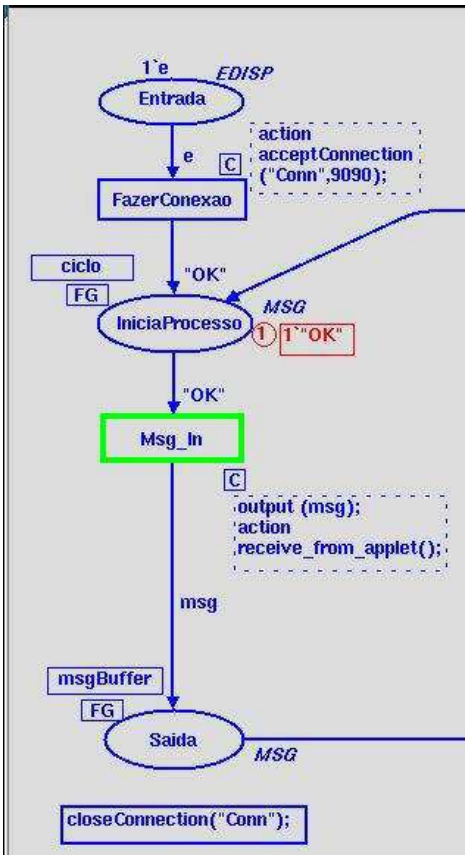

Figura A.6 – Transição "Msg_In" habilitada

10. Voltar para o FreeWRL

11. Clicar sobre a chave DJ21Y3 (neste instante, a chave fica na cor amarela, indicando um estado de transição, conforme apresentado na Figura A.7).

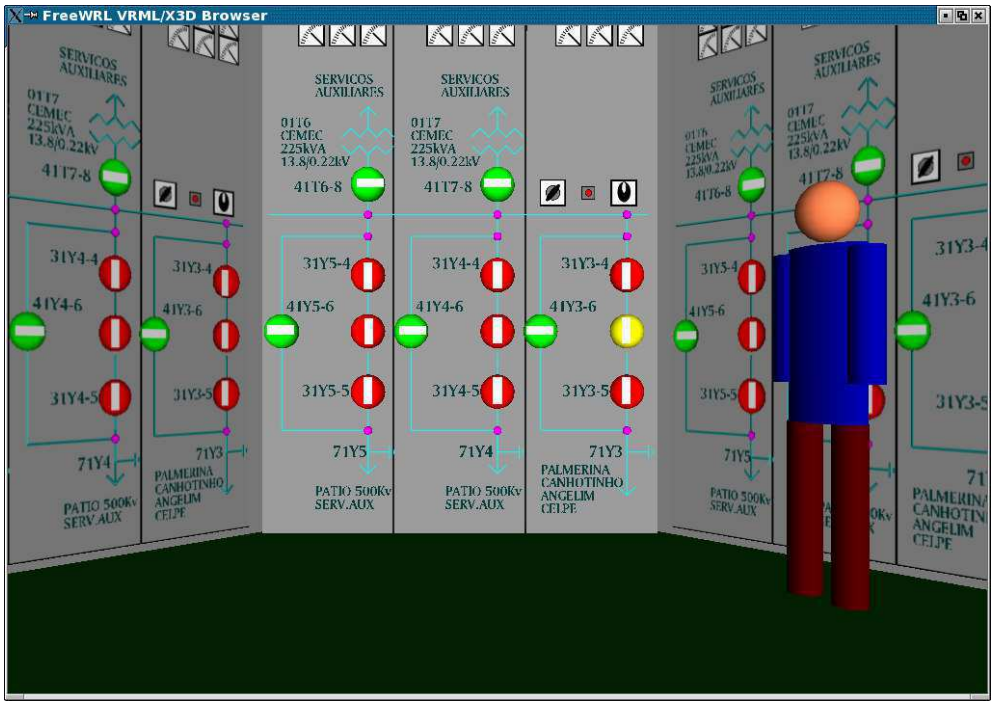

Figura A.7 – DJ21Y3 em estado de transição, de fechado para aberto

12. Voltar para o *Design/CPN*

12.1 É depositada uma ficha no lugar de fusão "Saída", indicando a ação a ser executada, neste caso, a mensagem é "AbrirDJ21Y3".

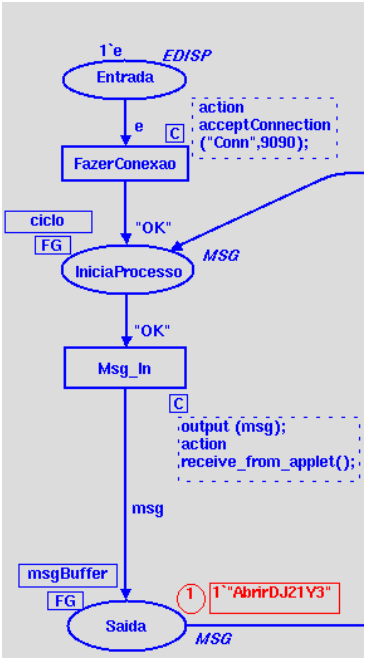

Figura A.8 – Mensagem vinda do mundo virtual

13. Confirmar abertura do disjuntor

13.1 Clicar sobre transição “COM_ABERT” (ver Figura A.9)

13.2 Verificar Bind *(Alt+B)*

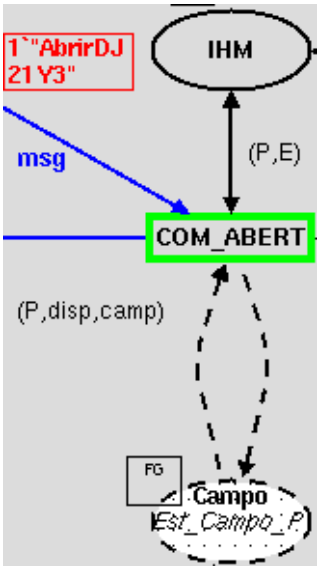

Figura A.9 – Confirmação de abertura do DJ21Y3

13.3 Clicar em *Occur* (neste instante, a chave do disjuntor DJ21Y3, representada no modelo VRML, muda sua cor para verde, conforme visto na Figura A.10)

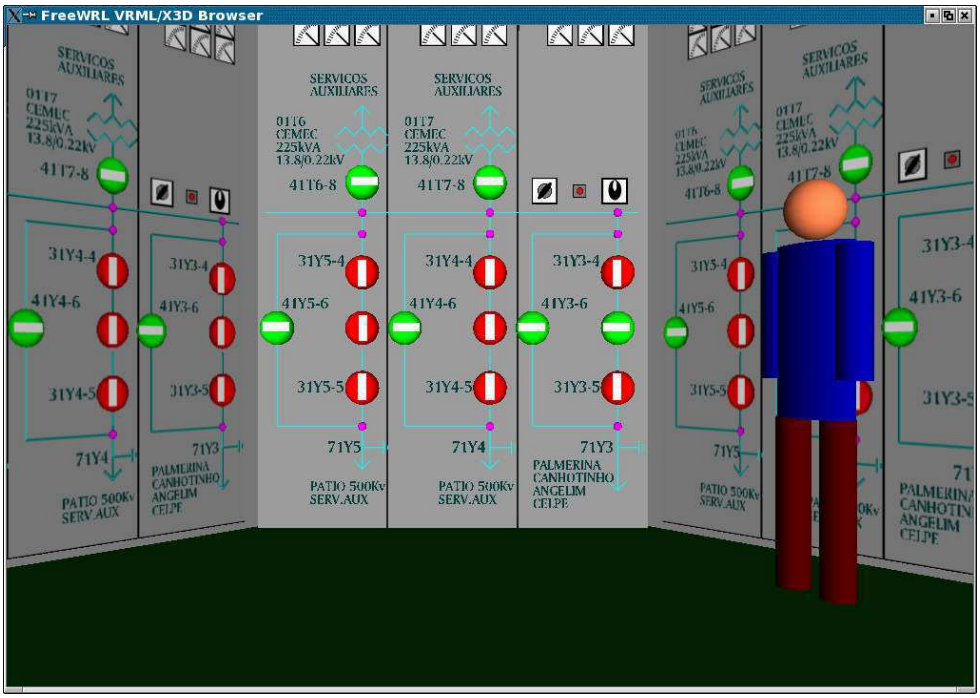

Figura A.10 – Chave DJ21Y3 aberta

14. **É depositada, no lugar de fusão "Campo", uma ficha para cada dispositivo da Rede. Todos continuam em seu estado original, à exceção do disjuntor DJ21Y3 que, agora, possui seu estado aberto.**

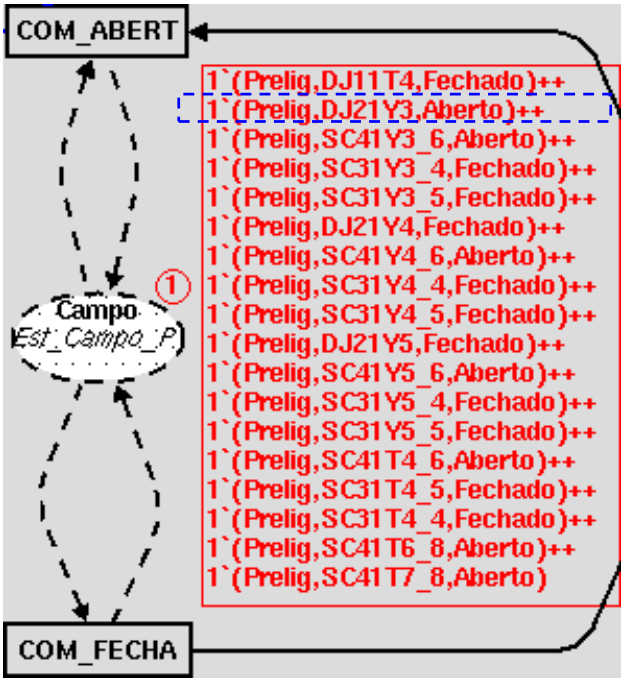

Figura A.11 – Ficha adicionada ao lugar "Campo", indicando estado de DJ21Y3 como aberto

15. Fechamento de Disjuntor

Nesta seção será apresentada a operação de fechamento da chave do disjuntor DJ21Y3. Este roteiro serve, também, para os demais disjuntores.

16. Preparar modelo Rede de Petri para receber outra mensagem do modelo VRML

16.1 Clicar sobre transisção "Msg_In"

16.2 Verificar Bind *(Alt+B)*

16.3 Clicar em *Occur* (neste instante, o modelo VRML da IHM painel aguarda a interação para o fechamento da chave DJ21Y3).

17. Voltar para o FreeWRL

18. Clicar sobre a chave DJ21Y3 (neste instante, o disjuntor fica na cor amarela, indicando um estado de transição, de aberto para fechado).

19. Voltar para o Design/CPN

19.1 É depositada uma ficha no lugar de fusão "Saída", indicando a ação a ser executada ("FecharDJ21Y3"*)*.

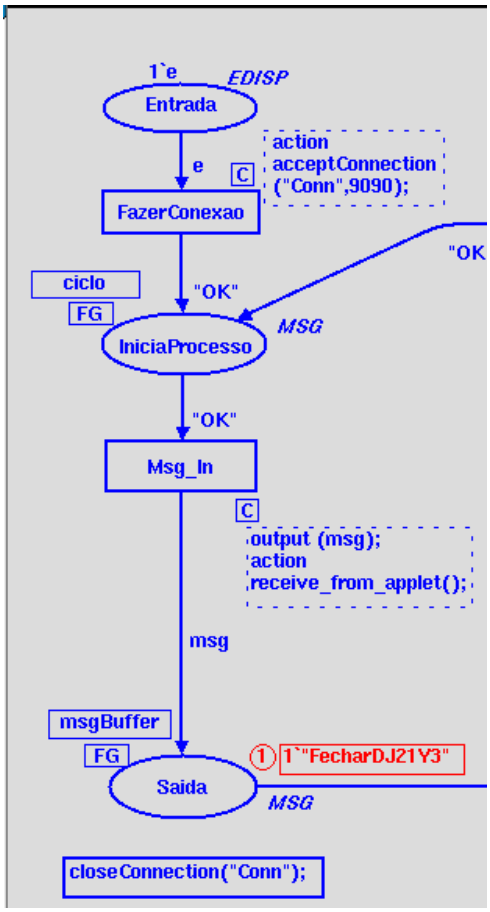

Figura A.12 – Mensagem vinda do mundo virtual para fechamento do DJ21Y3

20. Confirmar fechamento do disjuntor

19.1 Clicar sobre transicao “COM_FECHA” (ver Figura A.13)

19.2 Verificar Bind *(Alt+B)*

19.3 Clicar em *Occur* (neste instante, o disjuntor DJ21Y3, representado no modelo VRML, muda sua cor para vermelho, novamente).

Figura A.13 – Confirmação de fechamento do DJ21Y3

21. **É depositada, no lugar de fusão "Campo", uma ficha para cada dispositivo da Rede. O disjuntor DJ21Y3 volta ao seu estado inicial (fechado), conforme apresentado na Figura A.14.**

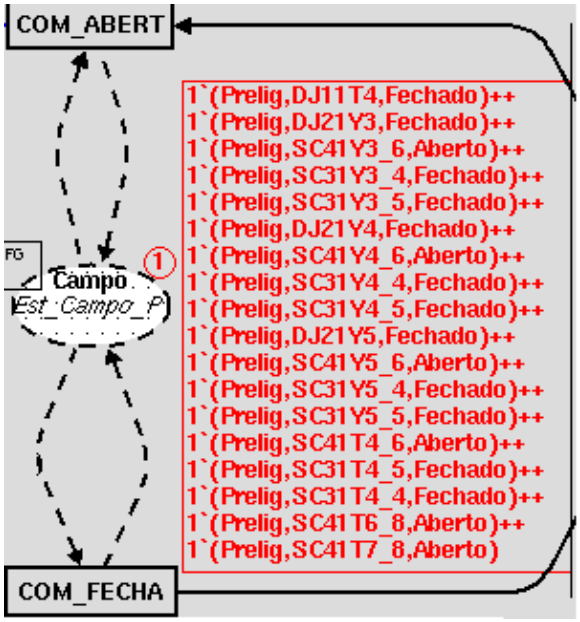

Figura A.14 – Ficha adicionada ao lugar "Campo" indicando o estado de DJ21Y3 como fechado