UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DOS CURSOS DE
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Tese de Doutorado

Aplicação de Métodos Formais no Projeto de Interfaces para Sistemas Industriais Críticos

Alexandre Scaico

Campina Grande – PB
Fevereiro - 2007

Aplicação de Métodos Formais no Projeto de Interfaces para Sistemas Industriais Críticos

Alexandre Scaico

Tese submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Maria de Fátima Q. Vieira Turnell, PhD
Orientadora

Charles Santoni. Dr
Orientador

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S178a 2007

Scaico, Alexandre

Aplicação de métodos formais no projeto de interfaces para sistemas industriais críticos / Alexandre Scaico. ─ Campina Grande, 2007.

150f. : il.

Referências.

Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientadores: Profª. Maria de Fátima Queiros Vieira Turnell, PhD., Prof. Charles Santoni, Dr.

1. Interface homem-máquina. 2. Redes de Petri coloridas 3. Sistemas industriais críticos. 4. Métodos formais. I. Título.

CDU- 004.5

APLICAÇÃO DE MÉTODOS FORMAIS NO PROJETO DE INTERFACES PARA SISTEMAS INDUSTRIAIS CRÍTICOS

ALEXANDRE SCAICO

Tese Aprovada em 30.03.2007

Maria de Fatima Q. Vieira Turnell

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

CHARLES SANTONI, Dr., Univ. Marseille
Orientador

Bittencourt

GUILHERME BITTENCOURT, Dr., UFSC
Componente da Banca

ALEXANDRE CABRAL MOTA, Dr., UFPE
Componente da Banca (Ausência Justificada)

Jorge Cesar Abrantes de Figueiredo

JORGE CESAR ABRANTES DE FIGUEIREDO, D.Sc., UFCG
Componente da Banca

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

CAMPINA GRANDE – PB
MARÇO - 2007

Agradecimentos

Agradeço a profª Maria de Fátima Q. Vieira Turnell pela amizade, por ter acreditado e incentivado este trabalho, e por ter me mostrado o verdadeiro significado da palavra orientação.

Agradeço ao prof. Charles Santoni pelas contribuições e direcionamentos.

Sou grato à minha família (Bárbara, Ademar, Babi, Izabel, Rafael, Ademarzinho, Luciana, Juliano, Jéssica, Alice e Ana) pelo amor, apoio, carinho e compreensão que eles sempre me deram, por entenderem as minhas ausências, e pelos momentos de alegria que eles proporcionam sempre.

Aos amigos e colegas do doutorado (principalmente a Zé Alves, Daniel, Pedrosa, Claudia, Ticiane, Leandro e Kyller) sempre presentes para incentivar e ajudar. Aos professores, que estiveram sempre presentes para apoiar e ajudar a transpor as dificuldades que surgiram no decorrer deste doutoramento.

Agradeço também a todos que de alguma forma contribuíram para a realização deste trabalho.

Resumo

Este trabalho tem como foco o estudo de métodos formais aplicados à modelagem de interfaces homem-máquina de sistemas industriais críticos, a exemplo de sistemas supervisórios industriais utilizados no controle de sistemas elétricos. O principal objetivo desta pesquisa é propor a utilização de um formalismo adequado à modelagem dessa classe de sistemas que possa auxiliar no projeto de interfaces mais ergonômicas. Este trabalho apresenta o método formal escolhido (as redes de Petri Coloridas [49][50]), a metodologia adotada na análise de modelos da interface e os recursos desenvolvidos para apoiar este processo de análise. Também é apresentado um estudo de caso utilizando a metodologia proposta.

Esta pesquisa está inserida no contexto da melhoria da qualidade de interfaces com o usuário de sistemas críticos com o propósito de reduzir a incidência do erro humano, e teve como foco os sistemas supervisórios industriais. Este trabalho é uma das etapas do Método para Concepção de Interfaces Ergonômicas (MCIE) [101], a modelagem formal do componente de navegação na interface. Os resultados deste trabalho comprovaram a relevância da construção de modelos no projeto de interfaces.

Palavras chave: Métodos formais, interface homem-máquina, sistemas industriais críticos, Redes de Petri.

Abstract

This work focuses on the study of formal methods applied to the design of human interfaces for critical industrial systems, such as the supervisory software employed in the automation of electric systems. The main goal of this research is to propose a choice of formalism adequate for modelling this class of system that can support the design of more ergonomic interfaces. This work presents the formal method that we choosed, the methodology employed during the analysis of the interface models and the resources developed along this work in order to support the analysis. It also presents a case study using the proposed methodology.

This research is within the scope of the major theme of quality improvement of user interfaces with industrial critical systems in order to reduce the human error. This research targets industrial supervisory systems. This work is one of the phases to be performed during interface conception when using the Method for the Conception of Ergonomic Interfaces (MCIE), the formal modeling of the navigation component in the interface. The results from this research corroborate with the views that formal modelling is a useful and practical approach to interface design.

Keywords: Formal methods, man-machine interface, industrial critical systems, Petri Nets

Sumário

Lista de Figuras

Capítulo 1 – Introdução

1.1 Definição do Problema

A interface com o operador do sistema, em um ambiente industrial automatizado, é um item considerado crítico. Por ser o meio de atuação no sistema, a interface está diretamente relacionada à operação segura e eficiente. Por sua vez, nos sistemas industriais automatizados não são tolerados erros nem atrasos na execução de tarefas por parte do operador. Apesar do elevado grau de automação de alguns destes sistemas os erros humanos ainda representam uma parcela considerável das causas de acidentes e incidentes industriais. [1]. O volume de informações a que é submetido o operador destes sistemas continua a crescer com os avanços tecnológicos, enquanto a realização da tarefa de supervisão e controle impõe restrições de tempo e de segurança que resultam em elevadas cargas cognitivas. Nestas condições o operador deve reagir aos eventos rotineiros e inesperados e completar tarefas, respeitando restrições de fim de prazo e de segurança prescritas nos procedimentos operacionais da indústria.

Este trabalho se apóia na premissa de que através da análise da interação entre o usuário e o sistema é possível estudar problemas potenciais decorrentes do projeto da interface e propor soluções para sanar os problemas encontrados, bem como estudar alternativas de interação para a realização das tarefas.

Uma vez que muitos incidentes e acidentes na indústria decorrem do erro humano, a análise do contexto de ocorrência contribui para a revisão dos procedimentos de realização da tarefa e particularmente para a revisão do projeto da interface com os operadores. A análise dos fatores relacionados à ocorrência do erro humano é particularmente importante em sistemas industriais cuja segurança na operação é crítica, isto é, ambientes cujas conseqüências dos erros e falhas podem ser catastróficas, do ponto de vista humano ou material. Aqui, o processo de análise do contexto do incidente pode se beneficiar do uso de ferramentas computacionais e particularmente da simulação de modelos do componente de interface com o operador.

Admitindo que não é possível eliminar os erros do usuário é necessário prevê-los e buscar tornar os sistemas mais robustos de modo a evitar a ocorrência de acidentes. Do ponto de vista das interfaces o objetivo é torná-las mais ergonômicas e robustas do ponto de vista de tolerância a erros.

Da literatura consultada [22][23][106][112] constatou-se que muitos estudos são realizados sobre o processo industrial e sobre a construção dos sistemas que os controlam, mas pouco é relatado sobre o projeto da interface com o operador. Os projetistas destes sistemas, em geral, não se aprofundam na escolha do estilo de interação mais adequado ou na melhor forma de apresentar as informações. Dado o baixo investimento no projeto das interfaces os programas supervisórios industriais podem resultar na baixa qualidade da interação. Nestes casos, embora o componente de controle possa ter sido bem estruturado, se a interface for inadequada o usuário não conseguirá explorar os recursos oferecidos pelo sistema. Cite-se como exemplo a escolha de ícones inadequados para representar objetos do processo que resulta em um tempo maior de reconhecimento e conseqüentemente de realização da ação.

O Grupo de Interface Homem-Máquina (GIHM) [36] da Universidade Federal de Campina Grande (UFCG) vem trabalhando na proposição de um método para concepção e avaliação de interfaces ergonômicas (MCIE – Método para a Concepção de Interfaces Ergonômicas [93]). Ao ser aplicado ao contexto de sistemas industriais críticos[1] passou a considerar informações sobre o erro humano visando minimizar sua ocorrência e tornar a interface mais robusta.

Ao longo do desenvolvimento do método, técnicas, modelos e ferramentas vêm sendo integrados no processo de concepção. Um destes modelos representa os mecanismos de navegação e interação disponíveis na interface. Através deste modelo formal é possível verificar a existência de problemas potencias no projeto da interface que podem induzir o erro humano. A construção deste modelo constitui uma etapa do método MCIE e representa formalmente os mecanismos de interação disponíveis ao usuário e os elementos que constituem a interface. O estudo aqui apresentado visa investigar a relação entre o projeto da interface, a ergonomia e o erro humano a partir da construção e verificação de modelos.

[1] Neste trabalho entende-se por **sistemas críticos** aquele nos quais falhas materiais ou erros humanos podem causar acidentes provocando: (i) perda de partes ou de todo o sistema; e/ou (ii) perdas de vidas humanas; (iii) e/ou perdas financeiras.

Acreditamos que o uso de métodos formais, com sua semântica precisa e possibilidade de verificações matemáticas no espaço de estados do modelo, permita garantir alguns aspectos ligados à usabilidade da interface em um momento anterior a prototipação. Mais especificamente, garantir questões de usabilidade ligada à navegação na interface.

Com isso poderemos garantir que o protótipo, concebido a partir das especificações do modelo formal, já possui essas questões de usabilidade, não sendo necessário que os testes de usabilidade foquem essas questões, minimizando assim o esforço de testes do protótipo e a qualidade da interface a ser construída.

Com o uso de modelos formais pretendemos que a etapa de testes de usabilidade seja focada na validação, junto ao usuário do sistema, de versões da interface que contemplem as questões de usabilidade já garantidas pela análise do modelo.

Assim o problema de pesquisa tratado neste trabalho é formulado a seguir:

> "Investigar a aplicabilidade de modelos formais na análise do componente de interação de uma interface com o usuário de sistemas industriais críticos, visando melhorar a usabilidade e minimizar os erros cometidos durante a interação."

1.2 Objetivos do Trabalho

1.2.1 – Objetivo Geral

Este trabalho tem como principal objetivo apoiar o projeto de interfaces para sistemas industriais críticos de modo que apresentem características mais adequadas ao usuário, ao contexto de uso e à tarefa a ser executada (interfaces ergonômicas).

1.2.2 – Objetivo Específicos

Como objetivos específicos deste trabalho podemos destacar:

- Investigar a contribuição oriunda da construção e análise de modelos formais no projeto (ou reprojeto) de interfaces para sistemas críticos;

- Propor uma metodologia para análise dos modelos que facilite sua utilização por projetistas de interface;
- Desenvolver ferramental de apoio ao processo de análise de modelos;
- Apoiar a etapa de modelagem da interação no método MCIE, refinando o projeto da navegação a partir da:
    - Análise da seqüência de passos necessários para a realização de uma tarefa;
    - Garantir da existência de caminhos de interação para a realização das tarefas, verificando quais que demandam menos esforço de interação ou menos tempo para sua execução;
    - Eliminação, e se não for possível a minimização de alternativas de interação disponíveis que são passíveis da ocorrência do erro humano.

1.3 Organização do Texto

Este documento está organizado em seis capítulos e dois anexos. O presente capítulo apresentou a motivação para este trabalho e os objetivos propostos.

No capítulo 2 é apresentada uma visão geral das interfaces homem-máquina, com foco no ambiente industrial, e sobre o projeto de interfaces. Nele é também apresentado o método de concepção de interfaces ergonômicas MCIE [93]. O uso de métodos formais no projeto de interfaces é abordado, e é apresentado o formalismo das redes de Petri coloridas [49][50][60], adotado na construção dos modelos.

No capítulo 3 é apresentada a metodologia proposta para a análise dos modelos da interface com o usuário de modo a investigar propriedades de usabilidade da interface e minimizar o erro humano.

O capítulo 4 trata do estudo de caso, apresentando o ambiente industrial sob estudo e o modelo correspondente. Já o capítulo 5 apresenta as análises no modelo do estudo de caso e discute os resultados.

Finalmente, no capitulo 6 são apresentadas as conclusões e as propostas de continuidade.

No Anexo A estão os códigos das funções desenvolvidas neste trabalho, que visam facilitar o processo de análise de modelos. E, no Anexo B é apresentado o projeto de um simulador para o treinamento de operadores de uma subestação de energia elétrica, cujo

motor de simulação é baseado nos modelos construídos para o estudo de caso relatado neste trabalho.

Capítulo 2 – Projeto de Interfaces com o Usuário

A interface com o usuário de um sistema envolve considerações sobre o sistema propriamente dito, o usuário do sistema, e a maneira na qual o usuário e o sistema interagem. Além dos módulos projetados para a interação, a interface abrange modelos e impressões que são construídos na mente do usuário em resposta a interação com esses módulos. Então o projeto da interface do usuário incorpora elementos que são parte do sistema, características do usuário e os métodos de comunicação de informação entre eles [7].

No contexto de utilização de software, pode-se definir o termo interface homem-máquina como sendo o componente de um aplicativo que traduz uma ação do usuário em uma ou mais solicitações à aplicação propriamente dita e fornece ao usuário uma resposta em conseqüência de suas ações.

Porém, uma constatação no desenvolvimento de aplicativos é que para o usuário de um sistema computacional , a interface "é" o sistema. Assim, o que os usuários desejam é que as aplicações (interfaces) venham ao encontro de suas necessidades e que sejam fáceis de usar.

2.1 A Importância da Interface

Um dos fatores mais significativos na qualidade de um sistema computacional interativo é a IHM (Interface Homem-Máquina) [71]. Este segmento da Ciência da Computação sofreu várias alterações nos últimos anos: o desenvolvimento de novas técnicas, o aparecimento de novas tecnologias para implementar essas técnicas, novas comunidades de usuários com as mais diversas formações e novas tarefas propostas a cada dia. E, com o atual surgimento de dispositivos de interfaceamento mais ergonômicos, dotados de recursos cada vez mais abrangentes, aumenta a possibilidade da especificação e implementação de interfaces mais poderosas [71].

O propósito chave da interação homem-máquina é possibilitar ao usuário o uso dos recursos das aplicações disponíveis no sistema (para a solução de problemas específicos) de forma "transparente", isto é, sem preocupação em como eles são executados. O usuário deve se sentir apto a explorar todos os recursos disponíveis no sistema, mantendo um elevado nível de satisfação durante todo o processo interativo.

A eficiência da interação homem-máquina se traduz na facilidade da comunicação entre o usuário e o conjunto de recursos disponíveis; na segurança que o sistema desperta no usuário ao executar as tarefas por ele solicitadas, permitindo ao usuário concentrar-se em seu trabalho, bem como prever o que ocorre após cada uma de suas ações; no desempenho da execução das tarefas pelo sistema e na integridade das informações fornecidas ao usuário [71].

Porém, mesmo que se defina a funcionalidade adequada, que se assegure à confiabilidade e se otimize o tempo/custo computacional das tarefas, o projeto final da interface poderá ser inadequado para o grupo de usuários ao qual ela se destina. Um dos requisitos essenciais para o sucesso do projeto de uma interface é a familiarização do projetista com as características da comunidade de usuários-alvo e com seus objetivos. Outro requisito é a definição do conjunto de recursos que será oferecido para a execução de tarefas inerentes ao contexto. Assim, o requisito mais importante em todo processo de concepção do modo de interação com os usuários, é a consideração de aspectos ergonômicos [71].

É importante, então, realizar um "Projeto centrado no usuário". Esse tipo de projeto se fundamenta no projeto da interação (e conseqüentemente da interface) do ponto de vista do que é melhor para o usuário, ao invés de fundamentar as decisões de projeto no que é mais rápido e fácil de implementar [106]. Também deve levar em consideração o perfil do usuário-alvo do sistema, projetando a interface baseada no perfil dito "médio" dos usuários do sistema para que as interações não sejam consideradas pelo usuário nem muito simples nem muito complexas, pois em ambos os casos o usuário se sentirá desmotivado para o uso do sistema. E, deve-se dar uma atenção especial a ergonomia da interface, para que o usuário consiga localizar as informações de forma clara, evitando o cansaço do usuário além de reduzir a probabilidade de erros.

Deve-se observar que as diferenças individuais entre usuários e a variabilidade das tarefas por eles realizadas são dois fatores de grande impacto sobre a usabilidade do sistema (e, por conseqüência da interface), com grande repercussão sobre a decisão dos projetos no que diz respeito ao tipo de estilo de interação a ser adotada e no conjunto de

ações que deverão ser implementadas. Facilidade de aprendizado, possibilidade de exploração máxima dos recursos disponíveis após a etapa de aprendizagem, freqüência e gravidade de erros possíveis, possibilidade de adaptação de usuários a outras ferramentas oferecidas no ambiente e, sobretudo, a satisfação subjetiva dos usuários são algumas das regras mais importantes da engenharia de usabilidade. Consistência é outro ponto chave na interação homem-máquina que recebe atenção especial ao se destacar como um dos problemas de usabilidade da atualidade [71].

Então podemos observar que a realização de um projeto de Interface Homem-Máquina é muito mais que apenas um bom projeto visual, deve-se também ter a preocupação com o conforto, a segurança e a eficiência dessa interface [7]:

- Conforto: O conforto para o usuário é mais do que apenas um ambiente físico agradável e ergonômico. Em relação à interface, o conforto tem relação com a apresentação visual das informações. As interfaces não podem ser confusas, dispersas, ou possuírem vários tipos de códigos para operações similares. As interfaces devem ser coerentes, claras e concisas de modo ao usuário se sinta confortável durante o uso.
- Segurança: A segurança do usuário na realização de uma ação é um fator muito importante. A interface deve prover meios para que o usuário saiba exatamente o que está fazendo, além de prover uma realimentação sobre a completa realização de uma tarefa.
- Eficiência: A eficiência em uma interface diz respeito ao projeto funcional, permitindo a realização de uma tarefa da forma mais rápida, melhor e com a menor taxa de erro.

2.2 Os Sistemas Industriais e o Componente de Interface

No ambiente industrial, mais do que em outros contextos, o projeto da interface com o usuário é um processo crítico, uma vez que uma interface confusa pode levar a uma interpretação errada das informações por parte do operador (usuário do sistema) ocasionando erros cujas conseqüências podem ser graves. Por exemplo, um erro pode ocasionar uma parada de produção ou impor risco de vida às pessoas envolvidas no processo.

O operador necessita de acesso rápido à informação devendo responder de modo rápido e inequívoco. A carga cognitiva sobre o usuário deve ser reduzida de modo a minimizar as chances de interpretação errada da informação que resultem no acionamento de "comandos" errados para o sistema (interações equivocadas). Portanto a interface demanda uma estrutura hierárquica, bem definida, com um elevado grau de objetividade e simplicidade na interação. Estas características estão diretamente relacionadas com a natureza da atividade fim: o controle da planta industrial [27].

Outro fator importante nestas interfaces é que o operador tem que reagir a eventos e completar uma tarefa considerando o prazo final e a segurança do processo [26]. As propriedades temporais da interação podem determinar a usabilidade de um sistema interativo [51]. Atrasos na resposta ao usuário fazem aumentar a taxa de erros cometidos pelos usuários [30][51]. E, por outro lado, respostas lentas do usuário também são inadequadas quando o sistema possui restrições temporais para a execução das tarefas [30]. Em sistemas críticos como os de controle industrial, atrasos na resposta do usuário a determinadas ações do sistema (como a ocorrência de um alarme) podem resultar em situações catastróficas.

Segundo [26], apesar da diversidade dos processos que controlam, as interfaces de sistemas críticos industriais compartilham várias características em comum, listadas a seguir:

- Apresentação do sinótico (resumo da planta industrial) da planta monitorada, dando ao operador uma visão geral do sistema e de seu funcionamento;
- Apresentação de tabelas de valores de variáveis do processo, que podem ser modificadas (a exemplo de *set-points*);
- Geração automática de relatórios de produção;
- Geração de gráficos de histórico, que mostram em forma gráfica as variáveis do processo em relação ao tempo ou a outras variáveis;
- Geração de gráficos de tendências;
- Sumário de alarmes, que são reconhecidos pelo operador para tomada de atitudes necessárias;
- Emulação de operação de equipamentos remotos, permitindo a operação de um equipamento remoto como se o mesmo estivesse fisicamente presente;
- Apresentação de informação multimídia para a supervisão do processo industrial.

Os elementos que compõem uma interface industrial são semelhantes àqueles que compõem as interfaces de um modo geral, tais como: menus, botões e caixas de entrada de

dados. A principal diferença consiste na natureza crítica da aplicação. A estrutura de navegação das interfaces industriais em geral é fortemente hierárquica, tipicamente oferecendo apenas uma alternativa de acesso a cada aspecto de funcionalidade do sistema supervisório. Isso se deve à demanda por respostas rápidas do operador do sistema; pois nestas circunstâncias a carga cognitiva sobre o operador deve ser reduzida de modo a minimizar as chances de interpretação errada da informação e o conseqüente equívoco na tomada de decisões [28].

Do estudo das interfaces industriais, podemos destacar as seguintes tarefas típicas realizadas pelo usuário durante a interação [28]:

- Comutação do estado de dispositivos: Consiste nas ações que levam à ativação ou desativação de objetos representados na interface, tal como um botão liga/desliga;
- Ajuste/entrada de valores: Consiste na digitação de dados alfanuméricos ou ainda na manipulação de um objeto gráfico na interface com o propósito de atribuir um valor a uma variável do processo dentro de uma faixa de valores;
- Seleção de opções: Seleção de uma ação para o processo, dentro de um conjunto de comandos disponíveis ao operador;
- Solicitação de informações: Consiste na solicitação da exibição de tabelas, gráficos e relatórios sobre o processo; ou de sua impressão para fins gerenciais;
- Monitoração de variáveis do sistema: Consiste na leitura e atualização de valores de variáveis do sistema industrial. Esses valores podem estar representados na forma gráfica ou alfanumérica;
- Navegação entre janelas: Consiste na seleção de uma nova janela da IHM com a qual se deseja interagir. Essa seleção pode ser através de botões, menus, listas, etc.

2.3 Erro Humano

O erro humano pode ser conceituado de várias maneiras, de acordo com a ótica no qual o mesmo está sendo abordado (ergonomia, psicologia, engenharia, etc.). Neste trabalho podemos conceituar o erro humano como “um disfuncionamento do operador humano (em relação as suas atividades mentais, psicomotrizes, sensoriais ou físicas) traduzindo-se em um desvio anormal em relação a uma norma ou padrão estabelecido” [39].

Na realização de uma tarefa o erro humano está relacionado com as diversas possibilidades de realização de uma dada tarefa. Quanto maior a liberdade na execução, maior a probabilidade da ocorrência de um erro humano.

Em um ambiente industrial, no projeto de uma interface para um programa de supervisão e controle, uma das recomendações é a de ter uma estrutura fortemente hierárquica, não oferecendo muitas alternativas na execução de uma dada tarefa. Isso objetiva evitar problemas de interação por parte do operador, e conseqüentemente reduzir a incidência de erros de operação.

Porém, em alguns casos não existe a possibilidade de se restringir à interação, ficando a cargo do operador a observância de um procedimento técnico na realização da tarefa. Um exemplo de erro humano na operação é a não observância de todos os passos necessários na realização da tarefa, com o operador ignorando algum item e finalizando a tarefa de modo incompleto. Mas, o operador não percebe esse fato (seja por imperícia, fadiga ou descumprimento das normas) e reconhece a tarefa como realizada.

Mas, o erro humano não é fator exclusivo do operador, sendo o sistema também responsável em diversos casos. Especificamente no domínio das interfaces de programas de controle supervisório, uma interface mal elaborada, ambígua ou com projeto visual confuso, pode levar o operador a cometer erros de operação. Isso é mais evidente no caso de operadores novos (inexperientes) e de operadores há muito habituados com a tarefa (a fadiga pode levar a desatenção, que leva ao erro).

Se a interface for bem elaborada, do ponto de vista do projeto visual e da interação, os erros não serão erradicados, mas serão minimizados. E é nesse aspecto que esse trabalho se concentra. Como já é sabido, o estudo do erro humano na operação de sistemas está em foco nas pesquisas do GIHM, como pode ser visto em [39].

Na subseção seguintes será apresentada uma classificação do erro humano.

2.3.1 – Classificação do Erro Humano

Especialistas em erros humanos (psicólogos e ergonomistas) já propuseram diversas formas de classificar o erro humano, dentre as quais destacam-se [39]:

- Rasmussen propõe um modelo seqüencial distinguindo diversos tipos de tratamento da informação (ativação, observação, identificação, interpretação, avaliação, definição da tarefa, escolha do procedimento e execução). Os tipos a serem

considerados dependem do comportamento do operador (baseado em regras, habilidades ou conhecimentos).

- Reason distingue dois tipos de erros:
    - (i) os lapsos são erros de execução nos quais a ação não procede com a intenção;
    - (ii) os equívocos são erros de planejamento nos quais a intenção da ação não é correta.
- Swain propõe uma outra categorização do erros:
    - (i) erro por omissão: quando são omitidas partes da tarefa a ser realizada;
    - (ii) erro de execução: quando é produzido o desenvolvimento de uma tarefa incorreta;
    - (iii) erro de derivação: quando é efetuada uma tarefa a mais, devido a uma distração;
    - (iv) erro de seqüência: quando é realizada uma tarefa fora da seqüência estabelecida;
    - (v) erro de tempo: quando é realizada uma tarefa antes ou depois do tempo planejado.
- Rouse et Rouse propõem um quadro que relaciona os erros com as etapas de tratamento do modelo de Rasmussen (ativação, observação, identificação, interpretação, etc.). Esta forma de categorização permite identificar o nível de atuação do operador (comportamento baseado em habilidades, regras ou conhecimento) para uma determinada tarefa, possibilitando a proposição de medidas de prevenção adequadas.
    - Ação omissa: quando são omitidas partes da tarefa a ser realizada;
    - Ação repetida: quando uma parte da tarefa é realizada mais de uma vez;
    - Acréscimo de uma operação: quando é realizada uma ação a mais na realização da tarefa, mas que não influi no resultado final;
    - Operação fora de seqüência: quando não se realiza a tarefa sem seguir o procedimento de execução, com os passos da tarefa sendo realizados fora da seqüência determinada;

- Intervenção em tempo não apropriado: quando a operação ocorre em um instante (antes ou depois) incorreto, não tendo validade, ou causando problemas na tarefa;
- Posição da operação incorreta: quando a tarefa é realizada, mas o resultado não é o esperado. A tarefa foi realizada, mas de forma incorreta;
- Execução incompleta: quando a tarefa é encerrada antes de ser completada;
- Ação sem relação ou inapropriada: inclusão de uma ação na execução da tarefa, e essa ação influi no resultado final da tarefa.

Através de uma análise dessas diversas classificações de erros humanos é possível fazer uma classificação no contexto desse trabalho. Aqui, estamos interessados na interação do operador com o sistema através da interface de um programa supervisório. E, os erros de interação devem ser passíveis de verificação através de análises em um modelo formal da interação. As classificações de Swain e Rouse & Rouse são as mais diretamente ligadas à interação. Podemos observar que elas são bastante semelhantes em relação aos tipos de erros. A nossa classificação (seguindo a proposta por Rouse & Rouse), com os tipos de erros humanos que pretendemos estudar, é apresentada a seguir:

- Acréscimo de uma ação: ocorre quando o operador realiza a tarefa completa, mas durante a execução ele realiza uma operação sem relação com a tarefa, podendo reverter essa ação ou não. No nosso contexto esse erro engloba, na classificação de Rouse & Rouse, os erros de acréscimo de uma operação e ação sem relação ou inapropriada.
- Operação fora de seqüência: ocorre quando o operador realiza a tarefa completamente, mas a faz fora da ordem definida para a sua execução.
- Omissão de uma ação: ocorre quando o operador realiza uma tarefa sem efetuar uma das ações requeridas na realização da tarefa.
- Execução incompleta: ocorre quando o operador acredita ter finalizado uma tarefa, mas na verdade ele não realizou uma (ou mais de uma) ação que fazia parte da tarefa. Em muitos casos esse tipo de erro ocorrer em conjunto com a operação fora de seqüência.

Os outros tipos de erros não serão abordados devidos a restrições desse trabalho. O erro de intervenção em tempo não apropriado não pode ser verificado nesse modelo devido a não abordarmos as questões temporais da interação, e sim apenas as questões de seqüenciamento. O erro de ação repetida também não pode ser analisado devido ao modelo

não permitir a ocorrência de uma ação repetida. E, o erro da posição de operação incorreta tem o problema de ser uma ação subjetiva do usuário, pois é um erro derivado de uma ação correta no objeto errado.

2.4 O Projeto de Interface com o Usuário

Embora a interface de um sistema computacional interativo seja um item de extrema importância para a aceitação ou não do sistema, apenas recentemente é que se tem dado a atenção necessária para a sua concepção. A interface tinha sua concepção relegada ao segundo plano, sendo a atenção voltada aos aspectos funcionais do sistema. Assim, tipicamente os projetistas não apresentavam conhecimento específico sobre o projeto das interfaces ignorando as diretrizes para uma concepção mais ergonômica.

Atualmente os projetistas percebem que a eficiência de um processo interativo homem-máquina se traduz na funcionalidade da comunicação entre o usuário e o arsenal de facilidades disponíveis; na segurança que o sistema desperta no usuário ao executar as tarefas por ele solicitadas, permitindo-lhe concentrar-se em seu trabalho, bem como prever o que ocorre após cada uma de suas ações; no desempenho da execução das tarefas pelo sistema e na integridade das informações por ele fornecidas ao usuário; enfim, no tempo/custo computacional exigido para a execução das tarefas e, sobretudo, para a comunicação usuário-computador propriamente dita **[**71**]**.

Surgiram então métodos de concepção de interface, como forma de prover um direcionamento no cumprimento dessa tarefa. Existem atualmente diversos métodos de concepção de interfaces disponíveis, mas os métodos baseados em modelos são os mais adotados atualmente [86]. Dentre as metodologias baseadas em modelo podemos citar ADEPT [56], TRIDENT [11], MEDITE [38] e MCIE [93].

As metodologias acima citadas trabalham com modelos semiformais no desenvolvimento da interface, a exceção do MCIE propõe o uso de Redes de Petri Coloridas [49][50] para a modelagem do componente de navegação na interface. Atualmente encontramos na literatura uma tendência a se utilizar métodos formais para a modelagem e análise de aspectos funcionais da interface, a exemplo dos mecanismos de navegação. O uso de métodos formais na modelagem da interface [35][47] é muito útil por se basear no uso de notações com semântica precisa que possibilita a discussão de propriedades e a predição do desempenho do sistema através de verificação matemática

[80]. Caso o modelo não se comporte como o desejado ou não atenda às propriedades desejadas, deve-se retrabalhar o modelo até que todas as falhas sejam resolvidas.

Embora os métodos formais tragam muitas vantagens ao processo de concepção da interface, sua utilização ainda não é amplamente aceita pelos projetistas de interfaces devido à dificuldade inerente ao aprendizado e uso de métodos formais [10][35][97]. Outro objetivo ao explorar o uso de métodos formais no contexto do projeto e desenvolvimento de interfaces, além da modelagem e análise, é prover meios de facilitar a realização dessas tarefas [10][35].

2.5 O Método de Concepção de Interfaces MCIE

A concepção de interfaces ergonômicas com o usuário para aplicações industriais é o foco atual de pesquisa do Grupo de Interface Homem-Máquina (GIHM) [36] na UFCG (Universidade Federal de Campina Grande). O método MCIE - Método para a Concepção de Interfaces Ergonômicas [93], vem sendo desenvolvido pelo grupo e já foi aplicado ao desenvolvimento e avaliação de interfaces industriais [65][86][93]. O método é descrito a seguir.

O processo de projeto do MCIE é centrado na avaliação e é organizado em fases, as quais são suportadas pela construção de modelos. É um processo iterativo onde cada fase deve ser avaliada antes de se prosseguir para a fase seguinte. Ele adota uma abordagem centrada no usuário de modo a alcançar um resultado ergonômico adequado às expectativas, habilidades e limitações dos usuários na execução de tarefas em situações críticas. Na Figura 1 é apresentado um diagrama do MCIE com suas fases e ferramentas relacionadas.

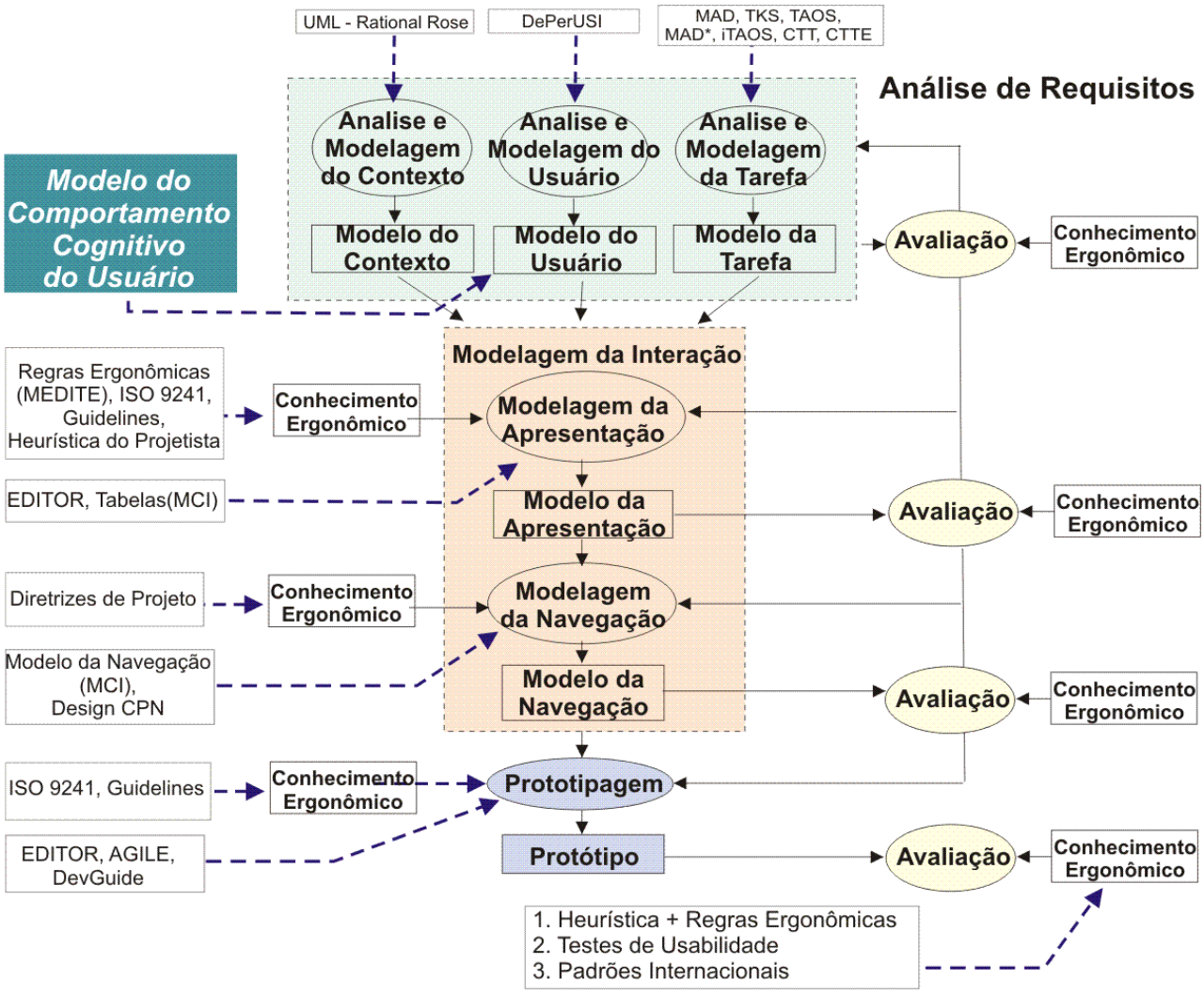


Figura 1: Etapas do MCIE [101]

O MCIE é dividido em quatro etapas [39][86], a saber:

- Análise de requisitos: Define as características desejadas da interface, tendo como base às características dos usuários do sistema, do contexto de execução e da tarefa a ser executada.
- Modelagem da interação: Nesta etapa são identificadas quais funções devem estar disponíveis, qual o seqüenciamento das mesmas, quais os mecanismos possíveis de interação, quais objetos disponíveis para interação, e quais partes da tarefa devem ser realizadas pelo sistema ou pelo usuário.
- Prototipagem: Consiste na geração do código da interface (protótipo) a partir do modelo da interação.
- Avaliação: A avaliação deve ocorrer durante todo o ciclo de desenvolvimento, verificando cada produto gerado em cada etapa.

Como se pode observar, no MCIE uma das etapas de construção do modelo da interação consiste na construção e análise de um modelo formal do componente de navegação do usuário na interface. Este modelo apóia a análise da seqüência de passos realizada pelo usuário durante a execução de uma tarefa (e subtarefas), passos estes que provocam a transição entre estados da interface. Os resultados da simulação e verificação deste modelo são analisados do ponto de vista de um conjunto de princípios ergonômicos descritos em [104], e são realimentados na concepção do modelo de interação.

A modelagem e análise de um modelo formal do componente de navegação já foi objeto de trabalhos anteriores do GIHM. Em [65][84][93] temos a descrição da modelagem utilizando o formalismo das redes de Petri coloridas [49][50]. E em [65][100] o processo de modelagem foi realizado utilizando o formalismo Statechart [41][42].

Atualmente O MCIE está sendo adaptado para o contexto de sistemas cuja segurança na operação é considerada crítica. Para esses sistemas, além da precisão e funcionalidade, é imperativo oferecer aos usuários: segurança na operação, adaptabilidade a diferentes graus de experiência e situações de trabalho, e fácil aprendizado. Dos estudos em andamento está sendo construído um modelo cognitivo do comportamento do usuário durante situações criticas o qual será incorporado ao MCIE objetivando construir interfaces mais ergonômicas.

2.5.1 – A Análise do Erro no MCIE

Foi incorporado ao MCIE uma nova etapa ligada à análise de requisitos, não apresentada na Figura 1, que trata do erro humano na interação [39]. Esta etapa trata da concepção de modelos conceituais de cenários de acidentes causados pelo erro humano. O objetivo principal desta etapa é a elaboração de um modelo que permita analisar e visualizar de forma explícita as principais situações que levam o operador a cometer erros durante a interação com o sistema (erros estes que são os iniciadores dos acidentes) [39].

Observe que essa etapa é válida para a avaliação de interfaces, pois depende do conhecimento prévio dos erros cometidos durante a operação do sistema. Esse tipo de informação nem sempre vai estar disponível na concepção de uma nova interface.

No caso específico do estudo de caso apresentado em [39], o conhecimento sobre os erros de operação foram extraídos a partir de relatórios de falhas de operação disponibilizados pela empresa CHESF.

Depois de construído, o MCCA (Modelo Conceitual de Cenários de Acidentes) [39] é composto por:

- Uma ontologia: um vocabulário comum e coerente de termos e ações relacionadas às situações de acidentes. Ou seja, uma terminologia classificada e relacionada entre si que seja compreendida pelos projetistas e usuários do sistema em questão e que seja representativa das situações de erros do operador com a interface do sistema.
- Uma tipologia de erros: classificação dos diferentes tipos de erro cometidos pelo operador durante a interação com o sistema e sua categorização.
- Uma tipologia de cenários de acidentes: descrição e representação das situações típicas nas quais o operador comete erros de interação com o sistema.

Através da elaboração do MCCA o avaliador pode identificar fatores causadores de erros e propor soluções para corrigi-los, aumentando assim a usabilidade da interface. Maiores informações sobre a construção e aplicação do MCAA podem ser encontradas em [39].

2.6 Métodos Formais na Modelagem da Interface

Na literatura podem ser encontrados vários trabalhos que tratam do uso de métodos formais na modelagem de interfaces homem-máquina de sistemas interativos. Abordaremos esses trabalhos de uma forma crítica nesta seção, justificando o método formal escolhido para este trabalho.

Em [62] e [87] tem-se o enfoque no uso de modelos formais na especificação da interação. Mais especificamente a aplicação do formalismo ICO (*Interactive Cooperative Objects*). O contexto de aplicação é o de interfaces para ambientes virtuais. O formalismo ICO é apresentado de forma sucinta e um exemplo de aplicação na modelagem de um programa de xadrez virtual é apresentado. É mostrada apenas a modelagem, não sendo abordadas as análises ou ferramentas de suporte. Porém, para o formalismo ICO, já existe uma ferramenta desenvolvida chamada *PetShop* [61] para a construção e simulação de modelos ICO. Infelizmente não tivemos acesso nem a detalhes do formalismo, nem a ferramenta, o que inviabilizou um estudo mais aprofundado. Mas, de acordo com os artigos, essa ferramenta só trabalha com simulação, não sendo possível, à época dessa pesquisa bibliográfica, análises formais no modelo.

Em [52] são usadas redes de Petri Coloridas (CPN – Coloured Petri Nets), para a modelagem de questões temporais da modelagem da tarefa. É realizada uma comparação entre um modelo da tarefa na notação semiformal CTT [59][68] e em CPN. Como conclusão os autores afirmam que as CPN se mostram mais adequadas a uma descrição de baixo nível da tarefa, embora saibam que esse formalismo encontra resistência ao seu uso pelas pessoas encarregadas da modelagem e análise da tarefa.

A verificação de modelos (*model checking*) é tratada em [24]. Neste artigo é apresentada a aplicação da verificação de modelos para a análise de modelos de interfaces. Mais especificamente a abordagem trata da verificação de modelos de interfaces escritas em Java utilizando modernos ambientes de trabalho para interfaces, como o Swing. Essa abordagem é interessante, pois mostra a validade da aplicação de *model checking* aplicado à interfaces; mas os teste são realizados em modelos aproximados gerados a partir do código JAVA da interface, e várias etapas são necessárias para a construção do modelo, envolvendo várias linguagens, o que dificulta o processo de modelagem.

Em [35] são apresentadas as vantagens de se utilizar métodos formais e semiformais na concepção de interfaces, bem como as razões do porquê das abordagens formais serem utilizadas de uma forma limitada (falta de ferramentas, notação formal fora do escopo dos projetista de IHM, estudos formais não são ligados às ferramentas de IHM). Neste artigo é utilizado o método B para a formalização e análise da interface (em conjunto com técnicas de verificação de modelos). Este artigo finaliza mostrando direções para incorporar métodos formais nas ferramentas para IHM, especificamente apresenta uma proposta de ferramenta denominada GenBuild.

A especificação formal, projeto, e implementação do componente comportamental de interfaces gráficas com o usuário são abordadas em [10] e [15]. É dada uma visão geral sobre o uso de métodos formais para o projeto de interfaces e é definida a gramática VEG (*Visual Event Grammars*) para a especificação do diálogo em interfaces. A especificação em VEG (convertida na linguagem *Promela*) pode ser verificada automaticamente por verificação de modelos através da ferramenta Spin, e um *toolkit* baseado em VEG gera código XML automaticamente.

Em [72] e [73] é apresentado de forma sucinta um *toolkit* (AMME) para analisar os dados empíricos do comportamento de solução de tarefas interativas descritas em um espaço de estados finito discreto. São utilizadas redes de Petri para a modelagem do comportamento de solução de tarefas. Os autores afirmam que o AMME analisa processos

observados e automaticamente extrai uma descrição em redes de Petri da estrutura lógica da tarefa.

Novas linguagens para a modelagem da tarefa – TaskMODL, e para a modelagem do diálogo – DiaMDL, são apresentadas em [97]. É apresentado como vantagem dessas linguagens o fato de serem gráficas. Mas sua desvantagem é ainda não possuírem ferramental disponível para modelagem e análise, o que inviabiliza seu uso prático.

Em [108] e [109] é apresentada uma extensão do formalismo *Statechart* chamado *StateWebCharts*. Esse formalismo foi desenvolvido para a modelagem de interfaces Web, mais especificamente para a navegação em interfaces web. Porém os autores ainda estão trabalhando na semântica formal para poder usar as ferramentas disponíveis de verificação formal e simulação.

O trabalho apresentado em [88] trata do uso da linguagem funcional Lustre para a modelagem, verificação e validação de interfaces. No contexto desse trabalho uma interface é modelada como um sistema de fluxo de dados e é especificada na linguagem Lustre, e essa especificação pode usar as técnicas de verificação de modelos através do uso da ferramenta Lesar.

A concepção de um modelo orientado à navegação baseado em *statecharts* para a especificação de hiperdocumentos é apresentada em [98]. Este trabalho se detém na definição do modelo, sem apresentar as formas de simulação e análise do mesmo.

Em [76] é apresentada uma metodologia para o desenvolvimento de IHM com o uso de notações semiformais e formais. A metodologia é baseada em 5 etapas: elicitação de requisitos, análise da tarefa (modelado em CTT), projeto semiformal da interação (modelado em UAN), prototipação (modelado em Clock), e especificação formal e prova (modelado em LOTOS). A etapa de prototipação não é realmente a construção de um protótipo, e sim a construção de um modelo detalhado da interação, para em seguida poder ser feita a prototipação propriamente dita.

Um trabalho de modelagem formal da interação é apresentado em [47]. A descrição é feita em uma variante da linguagem VDM (VDM-SL). Os autores definem também uma linguagem de descrição da interface – LSI, onde o modelo da interface em VDM-SL é então convertido em uma descrição LSI utilizada como dados de entrada de um prototipador.

O trabalho contido em [54] trata da modelagem do componente de navegação de um site *Web* por meio do formalismo *statechart*. São definidos os elementos de navegação *Web* que serão modelados (*hiperlink*, navegação entre páginas, navegação dentro da

mesma página, navegação baseada em frames, etc.) e, em seguida são apresentados, de forma superficial, os modelos para os elementos de navegação. Neste trabalho não é abordada a maneira pela qual se efetuaram as análises no modelo.

A modelagem e análise da interação entre o operador e um sistema de controle de tráfego aéreo, de modo a detectar erros de projeto na interação, são apresentadas em [18]. Este trabalho tem o intuito de verificar se uma dada interface é adequada à tarefa a que se propõe. O processo de modelagem é realizado através de autômatos de estados finitos e são construídos dois modelos, do operador e do sistema. Através da composição entre esses modelos é realizada a análise buscando evitar estados de bloqueio da interação e estados de erro (onde o operador acredita estar realizando uma operação correta quando na verdade não está).

Nos trabalhos estudados são citados também outros formalismos utilizados na modelagem de interfaces, a saber: *Dataflow* [58], CSP [87], lógica temporal [35] e linguagem Z [87].

No âmbito do GIHM também existem trabalhos que enfocam o uso de métodos formais para a modelagem de interfaces. Em [80][104] é apresentado um modelo em redes de Petri coloridas para do componente de navegação de uma interface industrial. O modelo pode ser simulado ou verificado formalmente pela ferramenta Design/CPN [21] de modo a garantir um conjunto de critérios de usabilidade [88][93].

Uma variante do modelo apresentado em [80][104], que contempla além da navegação as funcionalidades disponíveis ao operador de uma planta industrial é discutido em [65] e [81].

Em [82] e [100] temos um modelo da interação entre o operador e os elementos de proteção e comando de uma subestação de energia elétrica. O modelo foi construído com o formalismo *Statechart* com o apoio da ferramenta *AnyLogic* [111]. A análise do modelo ocorre através da simulação de cenários pré-estabelecidos.

2.6.1 Outros Formalismos Passíveis de Aplicação

Além dos formalismos citados, que já são utilizados na modelagem da interface de sistemas interativos, existem outros métodos formais para a modelagem que investigamos do ponto de vista de adequação no âmbito do método MCIE.

O formalismo DEVS [1][107] é definido como uma estrutura que descreve os diferentes aspectos do comportamento a eventos discretos de um sistema. Ele permite a

construção de modelos hierárquicos pela composição de modelos. Sua evolução ocorre devido à ocorrência de eventos ou através de uma função de avanço de tempo, e gera eventos de saída que podem servir como eventos de entradas de outros modelos que compõem o modelo global do sistema. Ele possui várias ferramentas desenvolvidas para a simulação de modelos, com o inconveniente de, até a realização dessa pesquisa bibliográfica, essas ferramentas trabalharem apenas com dados na forma textual.

Outro formalismo que se mostra adequado é o RPOO – Redes de Petri Orientadas a Objetos [40][79]. Este formalismo é uma integração dos conceitos de redes de Petri coloridas e orientação a objetos. A proposta é a integração dos conceitos das redes de Petri coloridas e orientação a objetos de uma forma ortogonal, de forma que as características de cada abordagem sejam preservadas, de modo que os dois paradigmas se complementem. Isso permite que as técnicas de análise de cada um dos paradigmas possam ser aplicadas na visão ortogonal do modelo que trata de cada paradigma. Uma ressalva a ser feita é o fato das ferramentas para modelagem, simulação e análise dessa notação ainda serem incipientes. Devido a esse fato não pudemos avaliar seu uso na modelagem de interfaces.

Ressaltamos ainda a notação semiformal CTT [68][69]. O CTT é muito utilizado na etapa de modelagem da tarefa, e possui uma ferramenta – CTTE [59] para a construção, simulação e análise de modelos da tarefa. O interessante de se trabalhar com CTT é o fato da ferramenta CTTE permitir uma tradução automática da especificação em CTT em uma especificação em LOTOS [12], possibilitando o uso de uma ferramenta de verificação de modelos (a exemplo da ferramenta CADP [29][31]) para a realização de verificações formais no mesmo. O CTTE ainda trabalha integrado com a ferramenta TERESA [95] que é um prototipador de interfaces. Com o uso do TERESA várias decisões do projeto visual da interface provêm do modelo da tarefa definido no CTTE, reduzindo assim a carga de trabalho do projetista da interface. O problema aqui é que essa tradução automática não funciona a contento, sendo necessário analisar a especificação em LOTOS para verificar se a mesma realmente está de acordo com o modelo da tarefa em CTT. Não tivemos acesso ao CADP, embora os desenvolvedores ofereçam a ferramenta livremente. Várias tentativas de contato e solicitação foram efetuadas e nenhum retorno obtido, o que inviabilizou a verificação da aplicabilidade dessa ferramenta.

2.6.2 Consideração acerca dos Formalismos

Embora existam vários métodos formais sendo aplicados à modelagem e análise de interfaces, nem todos são adequados no âmbito do MCIE, que é o método de concepção de interface no qual esse trabalho está inserido. O MCIE busca uma metodologia de projeto de interfaces que possa ser utilizada minimizando o esforço dos projetistas. E, como foi observado, é fato corrente que os projetistas não se sentem a vontade em usar um método formal dada a dificuldade inerente ao seu aprendizado.

Então, uma das premissas ao incluir um método formal dentro do processo de concepção da interface é a de que o mesmo não se torne um empecilho ao uso do método. Isso se traduziu no requisito do método formal possuir uma representação gráfica, o que torna mais fácil a aceitação no uso do método ao abstrair os conceitos matemáticos envolvidos.

Outra premissa no uso de métodos formais é a do mesmo possuir ferramental computacional de modelagem, simulação e análise do modelo. Sem uma ferramenta de apoio o processo teria de ser realizado todo manualmente, o que demanda maior tempo e é mais propenso a erros na realização de análises. E, como o método tem o intuito de não onerar o projeto com custo de softwares de apoio, buscam-se preferencialmente ferramentas que sejam gratuitas.

Partindo dessas premissas básicas os formalismos encontrados na literatura foram sendo refinados para determinar quais formalismos seriam adequados para um estudo mais aprofundado.

Inicialmente descartamos os formalismos que não possuem notação gráfica, e com isso foram eliminados os seguintes formalismos: as linguagens B, VEG, Lustre, VDM-SL e Z e o formalismo DEVS.

Em seguida verificamos que formalismos com representação gráfica possuem ferramental. Nessa nova etapa foram descartados os formalismos: ICO, StateWebCharts e RPOO. A ferramenta para uso com o formalismo ICO, denominada PetShop não foi disponibilizada para uso. Já para o StateWebCharts ainda não existe ferramental disponível. E no caso do RPOO as ferramentas disponíveis ainda estavam incipientes quando essa pesquisa foi realizada.

Após essas considerações sobre os formalismos disponíveis restaram dois candidatos possíveis: redes de Petri e Statecharts. As redes de Petri já vinham sendo utilizadas no âmbito do GIHM devido a já existir uma cultura de uso desse método formal

tanto dentro do GIHM como em um grupo específico de trabalho em redes de Petri na UFCG. Devido a isso a escolha pelo uso das redes Petri acabou sendo natural, dada essa cultura de uso da mesma. Dentre os trabalhos realizados em pesquisas do GIHM que envolvem modelagem de aspectos de interface em redes de Petri podemos citar [27], [28], [64], [65], [83], [85], [92], [93], [100] e [104], sendo os trabalhos [64], [83], [85], [100] e [104] relacionados ao presente trabalho de pesquisa.

As redes de Petri são ferramentas matemáticas e gráficas de modelagem aplicáveis a diversos tipos de sistemas [60]. Através das redes de Petri podemos descrever sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e/ou estocásticos. É um formalismo voltado para sistemas a eventos discretos e possui um ferramental disponível para modelagem, simulação e análise de modelos [21]. E, existe ainda uma biblioteca disponível para a realização de verificação de modelos em redes de Petri utilizando lógica temporal.

Os Statecharts, assim como as redes de Petri, possuem uma representação gráfica que permite a abstração de que o mesmo é um formalismo matemático. Segundo [41], os statecharts ampliam os diagramas estado-transição convencionais com elementos que lidam com as noções de hierarquia, concorrência e comunicação. Os *statecharts* estendem a linguagem de diagramas de estados, transformando-a em uma linguagem de descrição altamente estruturada e econômica [41], sendo então apropriados para sistemas complexos que envolvam um grande número de estados concorrentes, sincronização e ações de gatilhamento [54].

Existem várias ferramentas para Statecharts [93], que permitem a modelagem e a simulação de modelos. Porém, apenas uma das ferramentas pesquisadas permite algum tipo de verificação formal, a IarVisualState [43]. E, as ferramentas para Statechart encontradas são todas versões comerciais, não existindo versões gratuitas.

Dentre as ferramentas disponíveis tivemos acesso apenas ao AnyLogic [111]. Essa ferramenta usa conceitos de orientação a objeto, sendo um modelo constituído de componentes, sendo cada componente do sistema um objeto (escrito na linguagem Java [17][48]) com seu comportamento definido por um modelo Statechart. Uma ressalva dessa ferramenta é a de não permitir a análise formal do modelo.

Os Statecharts foram utilizados nesta pesquisa na modelagem e análise da interface de um sistema de proteção e comando de um subestação de energia elétrica [78][100] [102][103].

Em relação à adequação ao MCIE, no quesito facilidade de uso tanto as redes de Petri quanto os Statecharts se mostraram adequados, com seu aprendizado exigindo um esforço menor em relação aos outros formalismos encontrados. Porém, apenas com as redes de Petri são possíveis análises formais no modelo.

Devido a essas considerações e a familiaridade do GIHM com as redes de Petri, essa é a nossa escolha de método formal a ser utilizado no MCIE. E, na seção seguinte as redes de Petri serão apresentadas de forma sucinta.

2.7 Redes de Petri

O conceito das redes de Petri teve origem na tese de Doutorado "*Kommunikation mit Automaten*" de Carl Adam Petri, na Alemanha em 1962 [13]. As redes de Petri são ferramentas matemáticas e gráficas de modelagem aplicáveis a diversos tipos de sistemas [60]. Através das redes de Petri podemos descrever sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e/ou estocásticos.

Uma rede de Petri é composta de um grafo e de um estado inicial (chamado de marcação inicial, M_o). Este grafo é bipartido, direcionado e ponderado. Os nós deste grafo são denominados *lugares* e *transições*. Os nós são ligados através de arcos, observando que um arco deve ligar uma transição a um lugar ou um lugar a uma transição (não sendo permitida a ligação entre nós do mesmo tipo). Aos arcos se associam pesos (números inteiros positivos) como inscrições, onde um arco com peso K pode ser interpretado como um conjunto de K arcos paralelos entre dois nós [60]. Normalmente omite-se a inscrição de arcos de peso unitário.

O grafo define a estrutura da rede de Petri (sua natureza sintática). É necessário ainda expressar a dinâmica do sistema modelado, através de seus estados e evoluções entre estados. Para isso são acrescidas marcas a rede, chamadas de fichas, para definir o estado do sistema [74]. Apenas os lugares da rede podem conter fichas, que pode assumir qualquer número inteiro não-negativo. O número total de fichas em um lugar é denominado marcação do lugar, $M(p)$. A marcação da rede é um vetor M, contendo m elementos, onde m indica o número total de lugares na rede. Os elementos de M são as marcações dos m lugares da rede, onde $M(i)$ indica a marcação do lugar i. M_o denota a marcação inicial da rede. A evolução das fichas (marcação) nos lugares representa os vários estados do sistema [13][60].

Formalmente, as redes de Petri são definidas como uma 5-upla PN = (P,T,F,W,M_o), onde [60]:

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 transições
4. W: $F \rightarrow N^*$ é uma função peso
5. M_0: $P \rightarrow N$ é a marcação inicial
6. $P \cap T = \phi$ e $P \cup T \neq \phi$

Na modelagem, os lugares representam as condições para a ocorrência de um evento ou o estado do sistema, e as transições os eventos do sistema. Para cada transição da rede existe um número de lugares com arcos direcionados dos lugares para a transição (lugares de entrada) e um número de lugares com arcos direcionados da transição para os lugares (lugares de saída). Com isso, para um evento ocorrer (indicando a evolução do sistema através da ocorrência de uma transição) mudando a marcação da rede, a seguinte regra de transição (também chamada de regra de disparo) deve ser seguida [60]:

1. (Pré-condição) Uma transição *t* está habilitada (passível de execução) 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*.
2. Uma transição habilitada pode ou não disparar (ocorrer).
3. (Pós-Condição) Ocorrendo o disparo de *t*, *w(p,t)* fichas são retiradas de cada lugar de entrada *p* de *t*, e são adicionadas *w(t,p)* fichas a cada lugar de saída *p* de *t*, onde *w(t,p)* é o peso do arco de *t* para *p*.

Nem sempre uma transição necessita ter lugares de entrada. Neste caso estas transições são denominadas de transições fonte, pois não necessitam de nenhuma pré-condição para a sua ocorrência, estando sempre habilitadas. O mesmo pode ser afirmado em relação aos lugares de saída, e neste caso temos transições sorvedouro, que em sua ocorrência apenas consomem fichas.

Graficamente os lugares são representados por círculos, as transições por retângulos e a fichas por pontos pretos. É comum, de uma maneira informal, referir-se a representação gráfica de uma rede de Petri como se fosse a própria rede de Petri [13]. Na Figura 2 a seguir temos um exemplo gráfico de uma Rede de Petri e sua evolução:

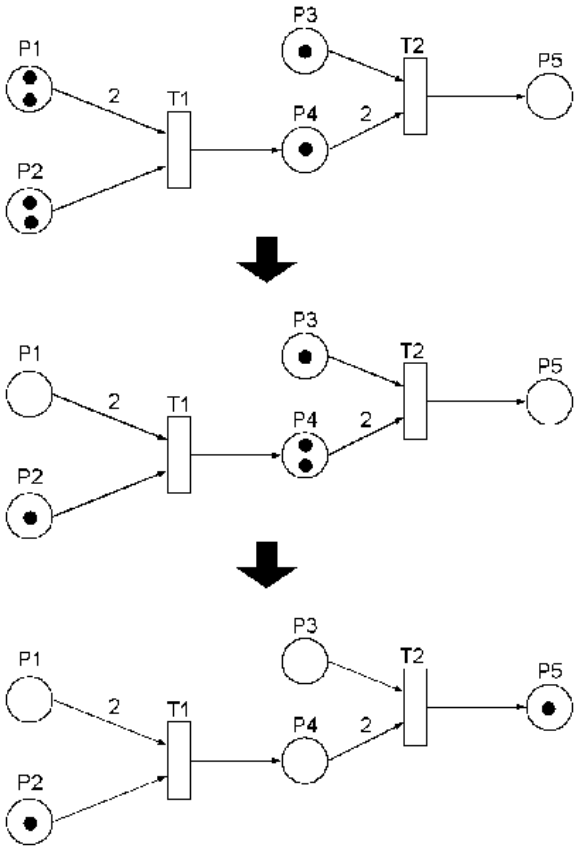


Figura 2: Exemplo de rede de Petri e sua evolução

Na rede da Figura 2 inicialmente apenas T1 está habilitada, pois seus lugares de entrada possuem pelo menos o número de fichas necessárias a sua habilitação. T2 não está habilitada, pois é necessário pelo menos duas fichas em P4 para habilitá-la. Com o disparo de T1 duas fichas são removidas de P1 e uma de P2, e uma ficha é depositada em P4 (como pode ser observado na segunda rede da figura). Com isso T2 se torna habilitada, podendo disparar retirando uma ficha de P3 e duas de P4, e depositando uma ficha em P5, como pode ser observado na figura.

As redes de Petri aqui apresentadas são denominadas redes de Petri Lugar-Transição.

2.7.1 Extensões das Redes de Petri

As redes de Petri definidas na seção anterior não se mostram adequadas para a modelagem de muitos sistemas, devido à complexidade destes sistemas, o que implicaria em redes de Petri lugar-transição complexas e muito extensas (com um número muito grande de nós e arcos); além de não serem capazes de modelar aspectos temporais relacionados ao sistema, impossibilitando uma análise quantitativa [13]. Para contornar estes problemas foram

propostas extensões ao modelo original, das quais as mais relevantes são as redes de Petri de alto nível e as extensões temporais das redes de Petri lugar-transição.

Como este trabalho foi realizado utilizando as redes de Petri Coloridas (CPN) [49][50], vamos detalhar apenas esta extensão das redes de Petri.

2.7.2 Redes de Petri Coloridas (CPN)

As redes de Petri coloridas resultam em uma representação mais compacta em relação as redes lugar-transição devido as fichas carregarem valores de dados, que podem ser complexos (ex: uma lista onde o primeiro elemento é um inteiro, o segundo é uma *string*, o terceiro é um par de inteiros), denominados *cores das fichas*. Cada lugar da rede deve possuir fichas cujas cores pertençam a um tipo específico, sendo este tipo denominado *conjunto de cores* do lugar. É necessário portanto (similarmente ao caso das linguagens de programação) declarar todos os *conjuntos de cores* (tipos) que existam na rede [49]. É permitido definir cores a partir de operações entre cores já existentes (ex: união, produto,etc.).

Aos arcos desta rede, ao invés de números inteiros, são associadas inscrições que correspondem a expressões. Essas expressões determinam, na ocorrência de uma transição, quantas e de que cores são as fichas que devem ser removidas dos lugares de entrada e adicionadas aos lugares de saída [49][74]. Inscrições também podem ser associadas às transições, sendo então denominadas guardas. Uma guarda é uma expressão que tem como resultado um valor booleano, cuja finalidade é restringir a ocorrência de uma dada transição [49].

As declarações de uma CPN (também denominadas nó de declaração) servem para especificar o tipo dos elementos que são suportados pelos lugares da rede, bem como as variáveis e funções utilizadas nas inscrições de arcos e guardas.

As inscrições dos arcos são construídas em função de *variáveis de transição* definidas nas declarações, cuja cor coincide com a cor do lugar de onde as fichas devem ser retiradas/adicionadas. O mesmo é válido para as guardas das transições. Uma *ligação* (*binding*) é a associação das variáveis de transição de um dado arco a um valor possível de cores. Um *elemento de ligação* é um par (t,b) onde t é uma *transição* e b é uma *ligação* para t [49].

Uma transição em uma CPN é dita estar habilitada se possuir em seus lugares um número de fichas suficiente para satisfazer as inscrições dos arcos (existir um *elemento de*

ligação) e se sua guarda for satisfeita. Com isso são retiradas fichas dos lugares de entrada e são adicionadas fichas aos lugares de saída de acordo com as expressões (avaliação das inscrições) dos arcos que ligam a transição aos seus lugares de entrada e de saída.

Uma CPN pode então ser definida como sendo composta de uma estrutura (um grafo bipartido e direcionado), de um conjunto de inscrições e de um conjunto de declarações [49][74]. Em [49] é apresentada a definição formal da CPN.

Na Figura 3 é apresentado um exemplo de uma CPN. O sistema modelado nesta figura é de um sistema ferroviário composto por um trilho de trem circular, com dois trens (trem *a* e trem *b*), que se movem na mesma direção. O trilho é dividido em 7 setores. Para um trem se mover entre setores é necessário que o setor seguinte esteja livre (para que ele possa ocupá-lo) e que o setor subseqüente ao seguinte também esteja livre (para garantir que não existirá a possibilidade de choque entre os trens). Inicialmente o trem *a* está no setor 7 e o trem *b* no setor 2.

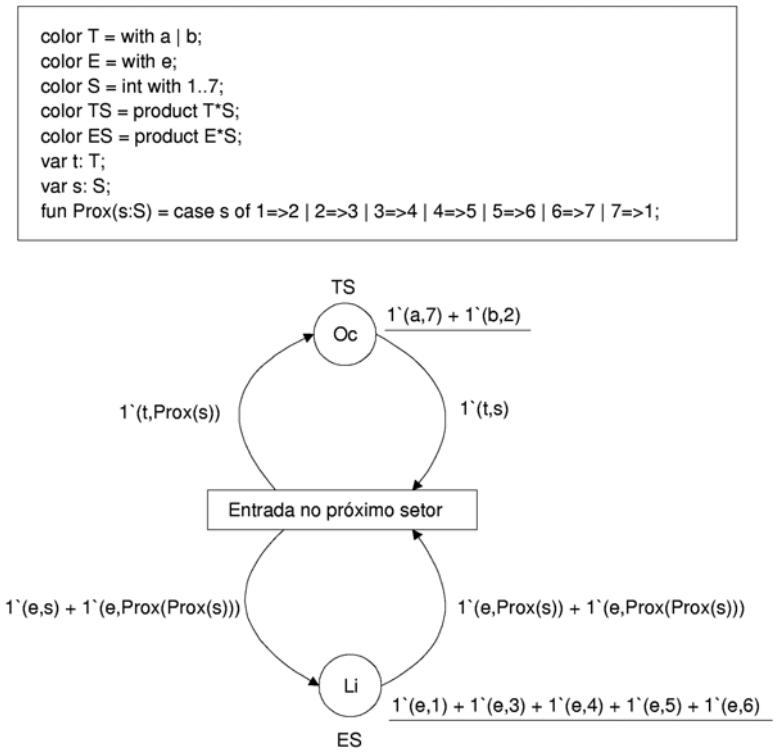


Figura 3: Modelo do sistema ferroviário em CPN

Como exemplo da compactação propiciada pelas CPNs, é apresentada na Figura 4, o modelo construído utilizando redes de Petri lugar-transição.

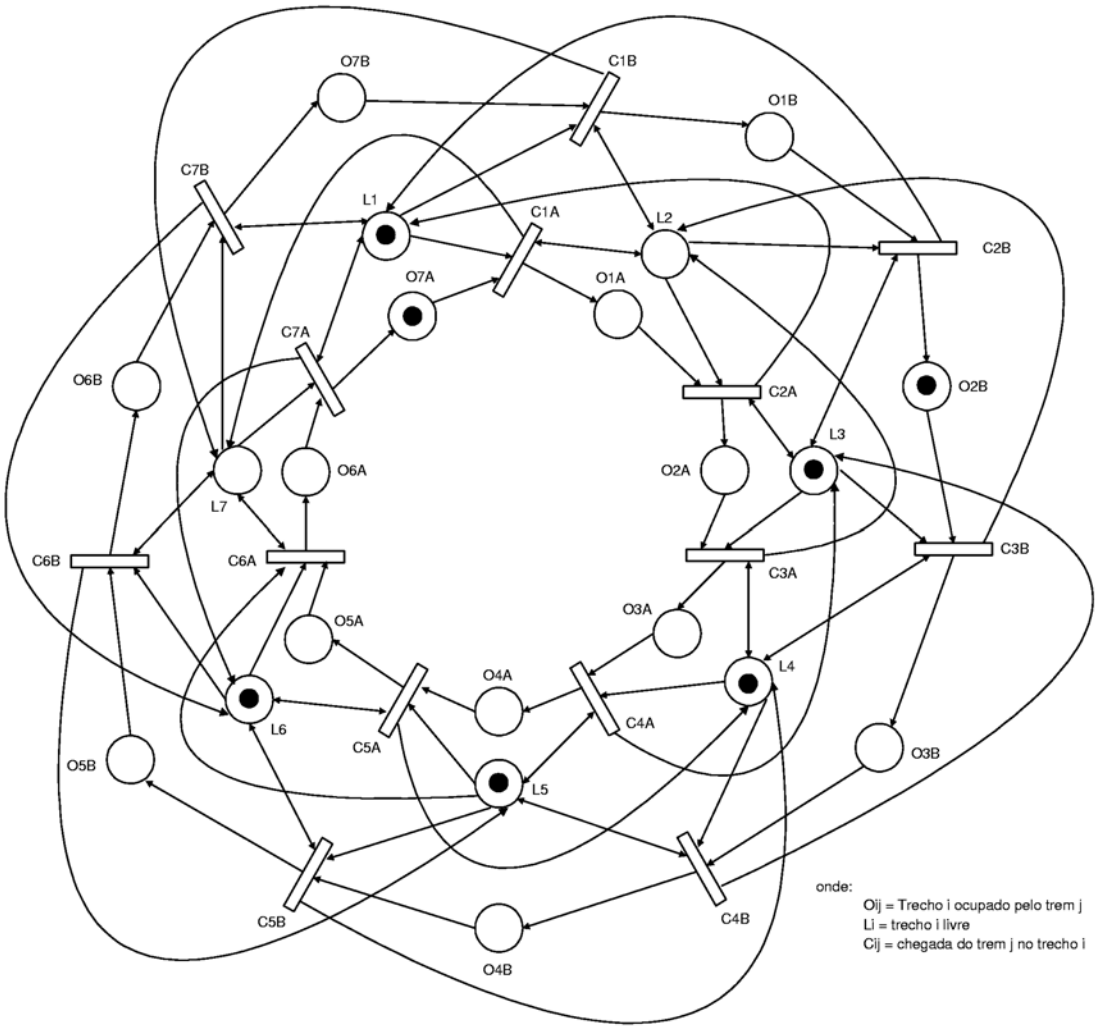


Figura 4: Modelo do sistema ferroviário em redes de Petri lugar-transição

Com as CPNs também é possível realizar a modelagem de forma hierárquica (redes de Petri Coloridas hierárquicas) ou com a inserção de restrições temporais (redes de Petri Coloridas temporizadas). Pode-se modelar de forma hierárquica e temporizada, construindo assim um modelo hierárquico temporizado. Nas seções seguintes serão detalhadas as redes de Petri Coloridas hierárquicas e as redes de Petri Coloridas temporizadas.

2.7.3 Redes de Petri Coloridas Hierárquicas

O objetivo fundamental de uso de redes hierárquicas é possibilitar construir modelos a partir da combinação de outros modelos, similarmente à programação modular, que

permite a construção de um programa mais complexo a partir de um conjunto de módulos e sub-rotinas [49].

Além disso, com o uso de hierarquia também é possível eliminar as redundâncias no modelo, uma vez que as partes redundantes podem ser instâncias de um único sub-modelo do modelo completo.

É importante atentar que com o uso de hierarquia não se obtém um maior poder de modelagem em relação às redes coloridas ou lugar-transição. Esses três tipos de redes possuem o mesmo poder de expressão. A diferença está na complexidade do modelo resultante. Podemos comparar estas três classes de redes em termos de linguagem de programação, onde as redes lugar-transição seriam equivalentes à programação em linguagem de máquina, as redes coloridas equivalentes à programação com o uso de dados estruturados, e as redes hierárquicas comparáveis à programação com o uso de módulos e sub-rotinas [49][74].

Com o uso de hierarquia é possível dividir o modelo em vários níveis de abstração, podendo assim lidar com uma visão mais abstrata de partes do modelo sem se preocupar com os detalhes da implementação dos itens que constituem os níveis mais baixos de abstração.

Para trabalhar com hierarquia, as redes de Petri coloridas hierárquicas (*Hierarchical Coloured Petri Nets – HCPN*) são definidas utilizando os conceitos de *lugar de fusão* e *transição de substituição*. *Lugares de fusão* são estruturas que permitem que vários lugares da rede se comportem como sendo um único lugar. Assim, todos os *lugares* que fazem parte de um *conjunto de fusão* (conjunto de *lugares* definidos como um mesmo *lugar de fusão*) possuem as mesmas fichas e, quando uma ficha é adicionada ou removida de um lugar que faz parte de um *conjunto de fusão*, todos os lugares que fazem parte desse *conjunto de fusão* terão a mesma quantidade de fichas removidas/acrescidas [49].

Transições de substituição são estruturas que permitem relacionar uma transição em um nível mais alto de abstração a uma rede mais complexa que fornece maiores detalhes das atividades que a *transição de substituição* representa [74]. Dessa forma, existe a relação entre redes individuais e nós de outras redes, ou seja, existem relações entre redes não hierárquicas para formar uma rede hierárquica. Para isso é necessário introduzir o conceito de *páginas* [49]. As *páginas* se referem a cada uma das redes individuais que formam a rede hierárquica. Cada *página* deve possuir um nome único.

A rede mais detalhada que a *transição de substituição* representa encontra-se em uma *página* denominada *subpágina*, enquanto a rede que contém a *transição de*

substituição é denominada *superpágina*. Cada *transição de substituição* é denominada *supernó* de sua *subpágina* correspondente.

A relação entre uma *transição de substituição* e sua *subpágina* correspondente se dá através de duas estruturas denominadas *portas* e *sockets*, que descrevem a interface entre a *transição de substituição* e a *subpágina* correspondente. Os *sockets* são atribuições aos lugares da rede contida na *superpágina* que estão conectados às *transições de substituição* enquanto as *portas* são atribuições a lugares da *subpágina*, de tal modo que um par *socket/porta* forma um *conjunto de fusão*. Com isso é definida uma conexão entre uma *superpágina* e uma *subpágina*, pois sempre que uma ficha é adicionada/removida de um *socket* ela também é adicionada/removida da *porta* associada a esse *socket* [49][74].

É interessante observar que uma HCPN pode ser transformada em uma rede não-hierárquica equivalente. Para isso é necessário substituir cada *transição de substituição* por sua respectiva *subpágina*, substituindo cada *socket* por sua respectiva *porta* [49].

Nas figuras a seguir é apresentado um exemplo simples de uma HCPN. Este modelo representa a troca de informações entre um dispositivo mestre e vários escravos, onde um mestre envia uma requisição para todos os escravos, os quais testam a requisição e enviam um cheque ao mestre, o qual define a ação a ser realizada e envia a ordem aos escravos. A página *Sistema* (Figura 5) contém duas transições de substituição, *Mestres* e *Escravos*. A transição de substituição *Mestres* tem como subpágina a rede *Mestres* (Figura 6), e a transição de substituição *Escravos* tem como subpágina a rede *Escravos* (Figura 7). Uma definição formal das HCPNs pode ser encontrada em [49].

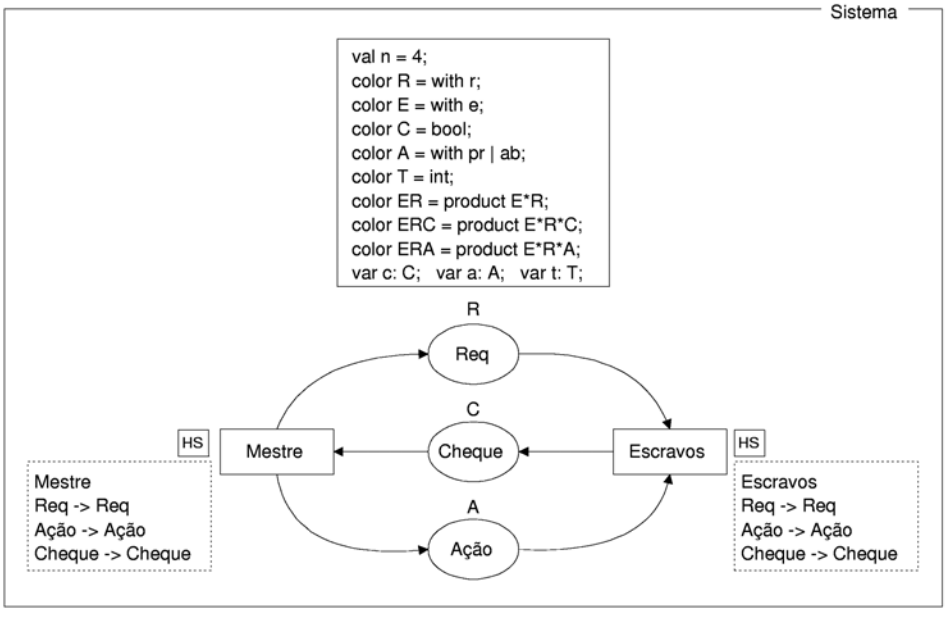


Figura 5: Exemplo de rede hierárquica (superpágina)

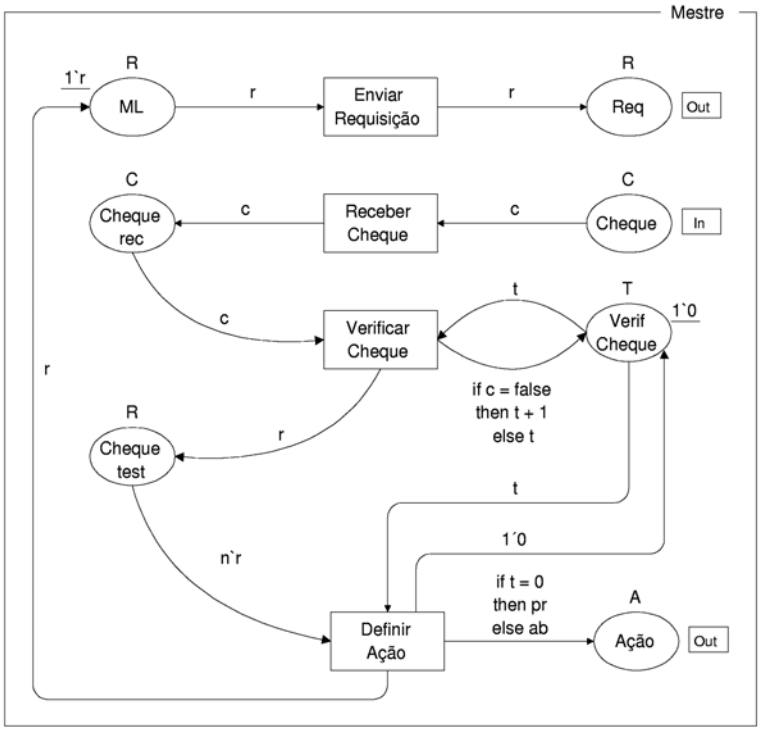


Figura 6: Subpágina correspondente a transição de substituição Mestre

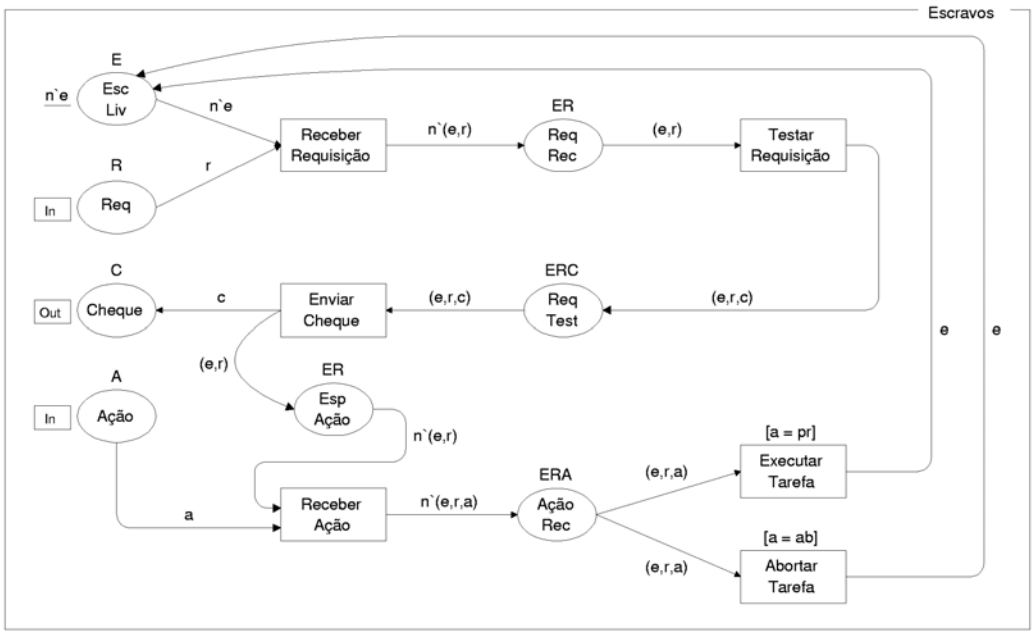


Figura 7: Subpágina correspondente a transição de substituição Escravos

2.7.4 Redes de Petri Coloridas Temporizadas

As redes de Petri coloridas, hierárquicas ou não, não possibilitam a análise de desempenho do sistema modelado. Para contornar essa limitação foram propostas as redes de Petri

coloridas temporizadas, nas quais foi incluído o conceito de tempo. Com o uso dessas redes é possível especificar como as diferentes atividades e estados "consomem" tempo.

A inserção do conceito de tempo em um modelo CPN se dá pela introdução de um *relógio global* (*clock*). O valor do relógio representa o *tempo do modelo*, que pode ser discreto ou contínuo. Com isso, cada ficha agora possui uma *marca de tempo* em adição a sua cor. A *marca de tempo* indica o tempo de espera mínimo a partir do qual a ficha pode ser usada (removida por um elemento de ligação) [50].

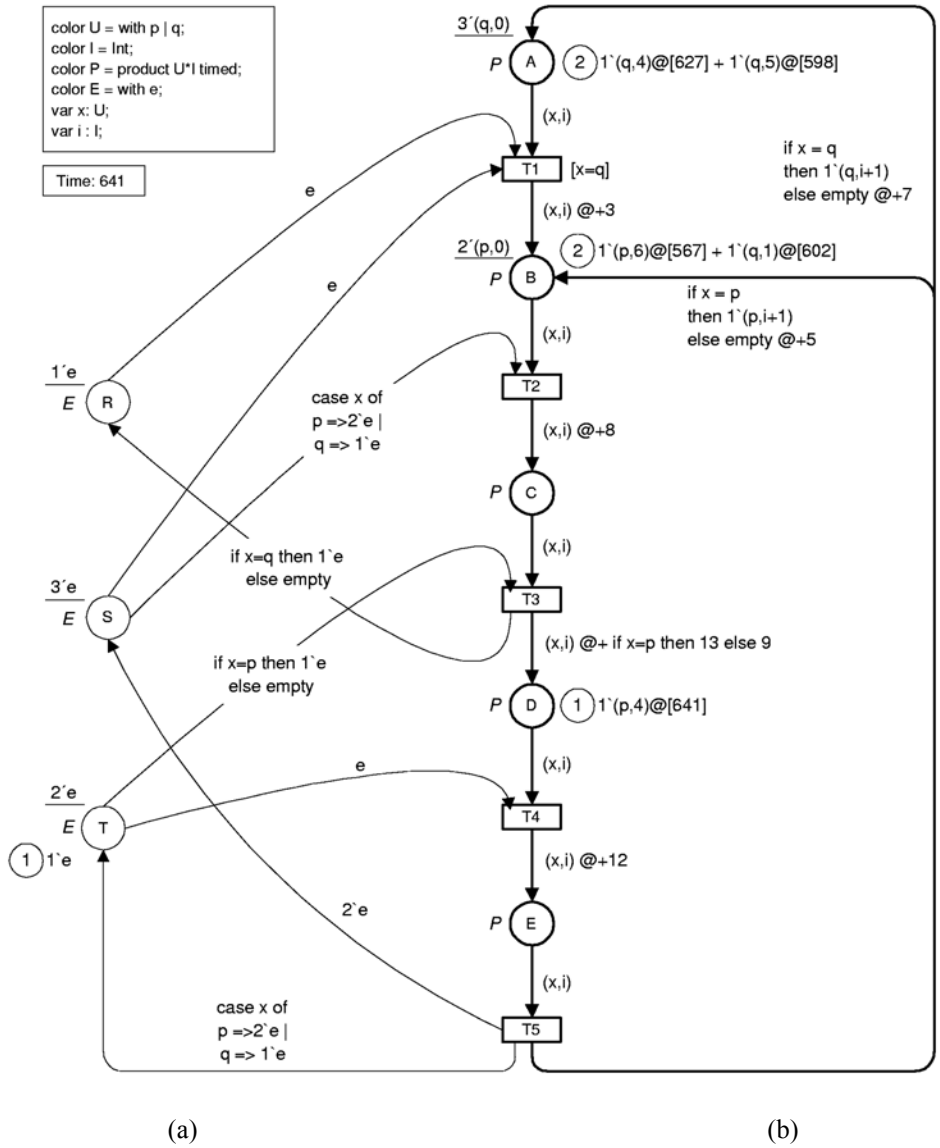
A habilitação de uma transição depende de dois fatores: da existência de um *elemento de ligação* e de sua *marca de tempo*. O conceito de habilitação das CPN em que bastava existir o *elemento de ligação* foi aqui extendido. A existência de um *elemento de ligação* indica que a transição contém as fichas necessárias para a sua habilitação sem levar em conta as restrições temporais (diz que a transição está *habilitada pela cor*). Mas, para a transição estar habilitada, o *elemento de ligação* deve estar também em um estado chamado *pronto*. Isso indica que todas as *marcas de tempo* das fichas removidas devem ser menores ou iguais ao tempo corrente do modelo (relógio do modelo) [50].

A modelagem de atividade que requer um certo tempo *x* para sua execução é obtida fazendo com que a correspondente transição *t* ao ocorrer crie marcas de tempo nas fichas de saída que são *x* unidades de tempo maiores que o valor do relógio na ocorrência de *t*. Com isso tem-se que essas fichas ficam indisponíveis para a habilitação de uma transição por *x* unidades de tempo, modelando assim a execução de um evento que consome tempo.

Como exemplo de redes coloridas temporizadas apresenta-se a rede da figura 8. Na Figura 8 pode-se observar que nas declarações, a cor *P* é temporizada. O retângulo abaixo das declarações mostra o valor corrente do relógio, e as fichas nos lugares de cor *P* possuem uma marca de tempo (valor após o símbolo @). Como é mostrado na figura, uma rede temporizada pode conter elementos não temporizados.

Na Figura 8 apenas a transição *T4* está habilitada, pois ela dispõe da ficha não-temporizada necessária, e a ficha temporizada necessária está no estado *pronto* (marca de tempo 641 alcançada). Com a ocorrência de *T4* essas fichas de entrada são retiradas e é criada uma ficha no lugar *E* com uma marca de tempo 12 unidades maior que o tempo no qual *T4* ocorreu (definido pela inscrição do arco de saída). Com isso *T5* não se torna habilitada pois a ficha no seu lugar de entrada não está no estado pronto (*T5* está apenas habilitada pela cor). *T5* só será habilitada quando o relógio global chegar na marca de tempo de ficha em *E* (653 unidades de tempo), ocasionando a ocorrência de *T5*.

Uma definição formal das CPN temporizadas pode ser encontrada em [50].



(a) (b)

Figura 8: CPN temporizada

2.7.5 Análise de Redes de Petri

Com o uso de redes de Petri é possível efetuar a verificação formal de propriedades dos modelos construídos. São dois os grupos de propriedades analisáveis nas redes de Petri: as *propriedades comportamentais* ou *dinâmicas*, que dependem da marcação inicial; e as *propriedades estruturais*, que dependem apenas da estrutura da rede [49][50].

Os métodos de análise se baseiam na verificação de propriedades comportamentais do sistema, através da construção de *grafos de ocorrência*; na verificação de propriedades estruturais do sistema, através de *invariantes de lugar e transição*; e, na redução do modelo

tendo em vista a facilitação de sua análise, através de técnicas de *redução e decomposição de redes* [49][50].

O *grafo de ocorrência* (*Occ – Occurence Graph*) é um grafo que contém um nó para cada marcação alcançável na rede (a marcação é descrita em uma inscrição de texto no nó) e um arco para cada elemento de ligação ocorrido na rede, ou seja, é uma representação do espaço de estados da rede [49]. No caso das redes lugar-transição ao invés de elementos de ligação têm-se a indicação da transição que ocorreu [13][60]. Nas redes coloridas temporizadas também há uma diferença no *Occ*. Os nós contêm um estado ao invés de uma marcação. Cada nó contém um valor de tempo e uma marcação temporizada [50].

2.7.6 - Ferramentas para redes de Petri Coloridas

Para as CPN, CPN hierárquicas e CPN temporizadas existe um pacote de ferramentas computacionais de apoio à modelagem e análise das redes, o *Design/CPN* [21]. Ele é constituído de quatro ferramentas computacionais integradas em um único pacote:

- Editor gráfico para construir, modificar e executar análise sintática de modelos CPN;
- Um simulador que pode operar simulação interativa ou automática, possibilitando ainda definir diferentes critérios para parada e observação da evolução da rede;
- Uma ferramenta para gerar e analisar grafos de ocorrência (espaço de estados) de modelos CPN;
- Uma ferramenta para análise de desempenho que possibilita a simulação e observação de dados de desempenho de modelos CPN.

  Maiores detalhes sobre a ferramenta Design/CPN podem ser encontrados em [21].

Existem bibliotecas que permitem ampliar as funcionalidades do Design/CPN. A biblioteca MSC [21] permite a construção de gráficos de seqüência de mensagens [6] a partir da ocorrência das transições do modelo. A biblioteca COMMS/CPN [34] permite a comunicação entre o Design/CPN e uma aplicação externa utilizando TCP/IP [34]. Existe também a biblioteca ASKCTL [19] que permite a realização de verificação de propriedades descritas em lógica temporal no espaço de estados do modelo. A verificação de modelos (*model checking*) em redes de Petri coloridas utilizando lógica temporal será descrita no próximo capítulo.

Outra ferramenta também está disponível para a modelagem e análise de modelos em redes de Petri Coloridas, denominada CPN Tools[2]. Essa ferramenta não foi utilizada devido ao fato de, no momento de escolha da ferramenta a ser utilizada, ela não possuir todas as funcionalidades presentes no Design/CPN, nem as bibliotecas associadas. Na época só era possível a simulação de modelos, não existindo ainda o módulo de verificação formal. Outro problema constatado a época foi o fato do programa exigir requisitos de hardware acima da média dos computadores disponíveis, o que tornava sua execução lenta. E, havia também a necessidade de se adequar os modelos existentes para o formato nativo dessa ferramenta, que difere do formato do Design/CPN. Devido a esses fatos que foi decidido que a ferramenta a ser utilizado seria o Design/CPN.

Hoje em dia a ferramenta CPN Tools foi incrementada e já contornou algumas dessas desvantagens apresentadas, mas ainda possui alguns pontos a serem melhorados, conforme descrito na página web da ferramenta. Vale ainda informar que o Design/CPN é uma ferramenta executável no ambiente Linux e o CPN Tools é uma ferramenta Windows.

[2] Informações sobre a CPN Tools podem ser encontradas em wiki.daimi.au.dk/**cpntools**/

Capítulo 3 – Análise de Modelos da Interface

Este capítulo apresenta a nossa metodologia de análise do modelo formal da interface com o usuário de um sistema de supervisão industrial, construído com o formalismo das redes de Petri Coloridas.

Iniciamos revendo e detalhando os objetivos que pretendemos obter com essas análises. Em seguida, apresentamos um conjunto de propriedades de usabilidade desejadas em uma interface interativa que podem ser mapeadas em propriedades formais do modelo do sistema. Detalhamos então os métodos de análise de acordo com a nossa abordagem de análise.

Para as análises se fez necessário a criação de algumas funções, desenvolvidas na ferramenta Design/CPN, para auxiliar na verificação do modelo. Essas funções auxiliam na verificação das diferentes caminhos possíveis para a realização de uma dada tarefa de interação do operador com o sistema. Essas funções também são descritas neste capítulo.

3.1 Objetivos das Análises

Este trabalho tem como objetivo a melhoria da qualidade das interfaces industriais, com foco no controle e supervisão de sistemas elétricos, do ponto de vista da usabilidade. Visa também verificar as alternativas de interação para as tarefas disponíveis no sistema e as possíveis causas de erros humanos que as interfaces podem permitir, buscando formas ou propostas de tentar evitar esses problemas.

Em relação à usabilidade, enfocamos aspectos ligados a qualidade da interação, focando a navegação na interface. Com base em um conjunto de propriedades de usabilidade que podem ser verificadas no espaço de estados do modelo, e por meio da análise de cenários através de simulação podemos verificar problemas de usabilidade, buscando corrigir os mesmos.

Através da análise de caminhos disponíveis para interação na execução de uma dada tarefa é possível buscar fatores causadores de erros humanos, como a execução incompleta ou incorreta de uma dada ação. E, uma vez descobertos esses fatores buscamos propor, se possível, meios de se evitar esses problemas.

Todas essas análises citadas se baseiam na simulação do modelo ou na verificação de propriedades no espaço de estados do mesmo. Iremos também realizar a verificação de comportamentos desejados no modelo através de verificação de modelos (*model checking*), complementando as análises de usabilidade. Com isso atingimos dois objetivos, a saber: realizar diversos tipos de análises no modelo, e comparar as potencialidades e dificuldades de se utilizar esses meios de análise.

3.2 Propriedades de Usabilidade x Propriedades Formais do Modelo

Esta seção apresenta informalmente, um conjunto de propriedades do modelo que estão associadas à usabilidade da interface, do ponto de vista da navegação. Estas propriedades são utilizadas na análise do modelo de navegação.

Dado que o componente de navegação em uma interface com um sistema supervisório pode ser representada por um modelo CPN, suas características podem ser expressas em termos de um conjunto de propriedades do modelo, como é ilustrado a seguir [91][92].

- *Reversibilidade* – É a possibilidade de retorno na interação, cancelando ações previamente realizadas. Não se aplica a todos os estados da interface. Verifica-se a reversibilidade através da propriedade de alcançabilidade da rede.
- *Existência de caminhos de acesso entre pontos específicos da interação* – Por razões de usabilidade, o acesso a pontos específicos (a exemplo dos mecanismos de ajuda) deve ser assegurado em qualquer ponto da interação. Este aspecto pode ser avaliado analisando as marcações da rede, investigando se a partir de um estado M_k é possível alcançar um estado M_j. A verificação dessa propriedade é realizada através das funções auxiliares de análise desenvolvidas neste trabalho.
- *Reinicialização* – Possibilidade de retorno ao início da interação. Este comportamento é verificado através da análise das marcações da rede. Se o

estado inicial estiver na lista das *home marking*[3] do modelo, esta propriedade é satisfeita. Note que se todas as marcações são *home markings*, conclui-se que há caminhos de conexão entre todos os estados da interface.

- *Acesso a Saída* – Do ponto de vista de usabilidade, os sistemas devem oferecer mais de um acesso à saída do sistema. Para analisar o acesso à saída é necessário verificar se todas as marcações mortas constantes no relatório do Occ correspondem ao estado de saída. A marcação morta deve sempre ser alcançável se existir acesso à saída a partir de qualquer ponto da interação. Nesta situação, a marcação de saída deve estar na lista das marcações mortas e na lista das *home markings*. Esta propriedade não se aplica a qualquer sistema.

Como esse trabalho enfoca o erro humano na operação de sistemas de supervisão, também é importante a verificação de possíveis erros de interação. Esses erros podem ser decorrentes do grau de liberdade que o operador do sistema tem na realização de suas tarefas, podendo realizar ações incorretas ou incompletas, por exemplo.

Com o uso das funções auxiliares desenvolvidas neste trabalho podemos verificar alternativas de execução de tarefas, contemplando a correta execução e os possíveis erros. Ao verificar as alternativas para a execução correta de uma tarefa, estamos verificando os caminhos alternativos disponíveis para o operador do sistema realizar uma dada tarefa, o que implica em flexibilidade na interação. E a verificação das alternativas incorretas de interação nos mostram possíveis problemas de interação na interface, e que devem ser corrigidos para se melhorar a segurança e usabilidade da interface do sistema.

3.3 Métodos de Análise

Para se efetuar as análises seguimos os seguintes passos:

- Simulações interativas e automáticas
- Geração de Gráficos de Seqüência de Mensagens - MSCs [6]
- Geração e análise do espaço de estados do modelo (grafo de ocorrência – Occ)
    - Análise de propriedades
    - Análise de alternativas de interação
- Verificação de modelos (*model checking*) por meio de lógica temporal

[3] Uma *home marking*, ou marcação recorrente, é uma marcação (estado) do modelo para a qual é sempre possível retornar a partir de qualquer outra marcação.

Detalharemos cada um desses passos de análise nas seções seguintes.

3.4 Simulações Interativas e Automáticas

No decorrer do desenvolvimento do modelo as simulações interativas devem ser utilizadas para verificar a corretude das partes do sistema. O uso de simulações interativas em partes do sistema facilita a verificação da evolução dessas partes.

Em seguida devem ser executadas simulações tanto interativas quanto automáticas no sistema como um todo para verificar sua evolução, observando se o mesmo se comporta como desejado.

A idéia nessa proposta de análise de modelos da interface é que o modelo seja desenvolvido a partir de um modelo genérico de navegação e instanciado para o contexto do trabalho. Como o modelo deve ser instanciado ao contexto a ser modelado, as simulações auxiliam na verificação do correto comportamento dessa instanciação. Neste trabalho é apresentado, no capítulo 4, o modelo de navegação instanciado para o caso de sistemas controle e supervisão de energia elétrica. O modelo de navegação apresentado pode vir a ser instanciado para outros contextos diferentes de sistemas elétricos.

As simulações também são úteis nas análises do modelo envolvendo o usuário. Cenários representativos do funcionamento da interface podem ser gerados e apresentados para o usuário verificar se é realmente esse tipo de comportamento que ele espera. Alternativas de interação podem ser discutidas a partir do estado atual do modelo e de sua evolução guiada de acordo com as indicações do usuário. Com isso a ergonomia da interação pode ser discutida com o usuário do sistema, aumentado assim o entendimento das necessidades e expectativas do mesmo, o que faz com que a qualidade da interface final seja incrementada. E isso vai de encontro com a proposta de projeto centrado no usuário.

3.5 Diagrama de Seqüência de Mensagens (MSCs)

Gráficos de Seqüência de Mensagens (MSC – *Message Sequence Charts*) consistem em uma linguagem gráfica e textual para a descrição e especificação de interações entre componentes de um sistema. Eles descrevem a comunicação assíncrona entre instâncias e podem ser utilizados como uma especificação geral do comportamento de comunicação de

sistemas em tempo real [6]. Uma instância é uma entidade abstrata na qual pode-se observar parte da interação com outras instâncias ou com o meio-ambiente. No MSC uma instância é denotada por uma barra vertical. E o tempo ao largo de cada eixo transcorre do topo para a base [6].

Usualmente os projetistas que analisam a especificação de interfaces e os clientes do projeto não estão familiarizados com o formalismos de redes de Petri, assim, a explicação do comportamento destes componentes através de MSCs se torna uma abordagem muito útil. A representação de um evento por meio de uma mensagem é muito mais intuitiva do que a análise das marcações de um modelo CPN. A geração do MSC se dá através da criação de mensagens pré-definidas de acordo com a evolução da simulação do modelo, através da ocorrência das transições do modelo que possuem um código associado com a chamada da função que gera a mensagem.

Essa técnica de análise é complementar as simulações, pois aqui o projetista pode definir cenários representativos da evolução do sistema para mostrar para o usuário, sem ser necessário que o usuário tenha conhecimento sobre a evolução de modelos em redes de Petri coloridas. Neste caso o usuário abstrai o método formal e se concentra apenas na análise do significado das mensagens apresentadas no MSC, ficando a cargo do projetista a geração da simulação no modelo do cenário que irá gerar o MSC.

Na Figura 9 a seguir temos, o exemplo de uma parte de um MSC construído a partir da ferramenta Design/CPN e da biblioteca MSC, disponível em [21].

O exemplo da Figura 9 apresenta três instâncias representando três estados de elementos hipotéticas. Na evolução do modelo correspondente, a ocorrências das transições, que modelam a mudança de estados dos elementos, faz com que as mensagens correspondentes a essas mudanças sejam criadas no MSC, conforme pode ser visto na figura.

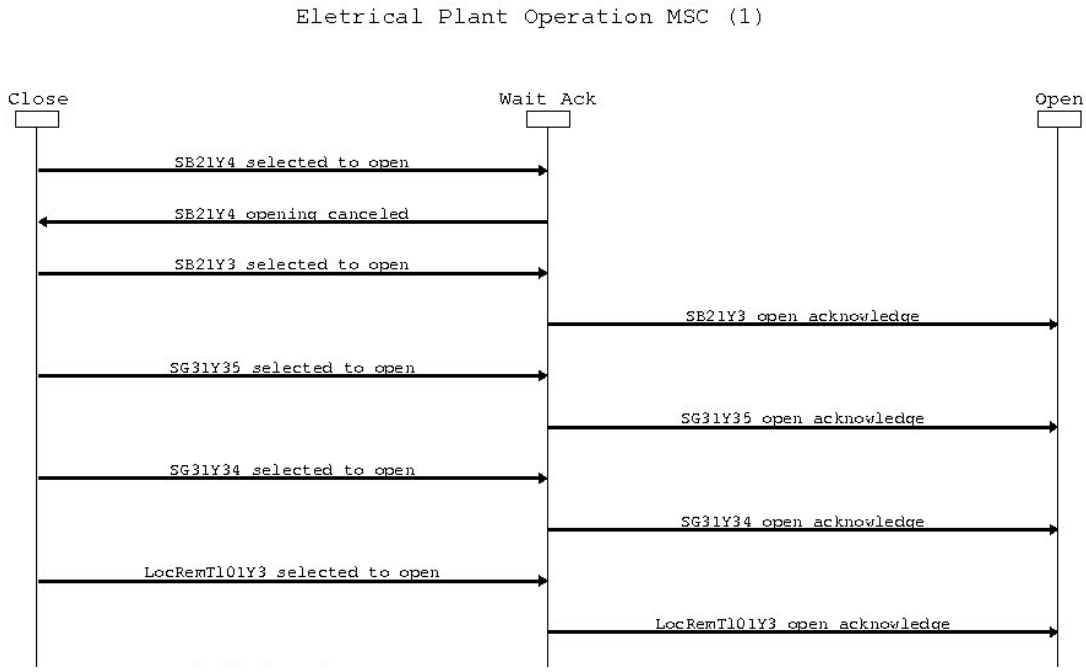


Figura 9: Exemplo de um MSC

3.5.1 – Geração do Occ

O grafo de ocorrência (Occ) consiste em um grafo contendo um nó para cada marcação alcançável e um arco para cada elemento de ligação entre nós [49], ou seja, representa o espaço marcações alcançáveis do modelo CPN. A partir do grafo de ocorrência é possível se realizar verificações no espaço de estados do modelo. E essas verificações podem ser realizadas a partir de funções já presentes na ferramenta de geração do grafo de ocorrência do Design/CPN ou podem ser criadas novas funções (usando as funções existentes como base) para auxiliar nas verificações.

Ao ser gerado o Occ do modelo a ferramenta disponibiliza um relatório padrão contendo informações sobre [20]:

- As estatísticas em relação ao grafo gerado (tamanho do grafo, número de nós e arcos e completude da geração);
- Propriedades de limitação: valores máximos e mínimos de fichas nos lugares do modelo (em termos absolutos ou em função das cores);
- As marcações recorrentes do sistema (*home markings*);
- Propriedades de vivacidade: lista as marcações mortas, as transições que nunca ocorrem, etc.

Em relação às funções de verificação de propriedades disponibilizadas pela ferramenta podemos citar as que verificam propriedades de alcançabilidade, limitação,

recorrência, vivacidade e equidade. Além disso, existem outras funções que são disponibilizadas para facilitar o desenvolvimento de funções customizadas pelo projetista do modelo, dentre as quais podemos citar a função *SearchNodes*, que é uma função que percorre todo o grafo de ocorrência verificando um predicado definido pelo projetista. Maiores informações sobre a ferramenta de geração e análise do grafo de ocorrência do Design/CPN e suas funções de verificação podem ser encontradas em [20].

Então, como pode ser observado, o grafo de ocorrência permite uma ampla verificação do modelo, cabendo ao projetista a tarefa de definir quais propriedades deseja verificar, limitando o escopo da análise, e como interpretar esses resultados de modo a garantir as propriedades que o modelo do sistema deve atender.

A partir da análise do relatório do Occ podemos verificar algumas propriedades do modelo. Inicialmente podemos verificar se os valores, máximo e mínimo, das fichas nos lugares do modelo estão de acordo com o esperado. Se em um dado lugar existirem menos ou mais fichas do que o esperado, é necessário a analise para determinar se esse erro é decorrente de um comportamento não esperado do sistema ou de um erro de modelagem. Se for um comportamento não esperado, deve-se analisar o causa do erro, retrabalhar o modelo e propor as soluções no projeto da interface. Se o erro for de modelagem, o modelo deve ser corrigido e novamente analisado.

Em seguida deve-se observar as marcações recorrentes e os estados de bloqueio do sistema, verificando se estão de acordo com o comportamento esperado do sistema. Caso não estejam, deve-se observar a causa do erro e corrigir, da mesma forma explicada para o caso dos limites máximo e mínimo de fichas.

Ao verificar as marcações mortas do modelo, por exemplo, deve-se analisar o que significam e se são um comportamento esperado do sistema, corrigindo o erro caso contrário. Normalmente as marcações mortas devem estar relacionadas com a saída do aplicativo, não sendo válido outros tipos de impasses no sistema, ou seja, afora o encerramento do aplicativo não devem existir outros tipos de estados de bloqueio do modelo.

A análise do grafo de ocorrência permite também a verificação das propriedades de usabilidade já citadas, a partir do modelo. Na definição das propriedades também estão definidos quais as propriedades do modelo que devem ser utilizadas para a verificação.

Porém, as funções disponibilizadas pela ferramenta não permitem a análise de caminhos alternativos de interação, tendo sido necessário a concepção de um conjunto de funções específicas para essa tarefa, conforme apresentamos na seção a seguir.

3.6 Funções Complementares Utilizadas na Análise da Interação no Modelo CPN

Esta seção apresenta o conjunto de funções desenvolvidas para a análise do modelo CPN baseada na análise dos caminhos de interação permitidos ao usuário do sistema para navegar, ou executar uma ação, na interface. Através da análise dos caminhos é possível verificar as diferentes formas de se realizar uma mesma operação, permitindo a verificação de alguns aspectos de usabilidade da interface como também detectar possíveis problemas de interação. Com problemas de interação queremos dizer ações que não deveriam ser permitidas, ações que podem ficar incompletas, ou ações que podem ser executadas fora da ordem correta. Detectar esses problemas e corrigi-los aumenta a usabilidade e a segurança da interface.

Para se efetuar a análise de caminhos de interação, três etapas são necessárias: a definição dos estados inicial e final da interação, a geração dos caminhos possíveis entre esses estados, e a análise desses caminhos para verificar quais ações foram tomadas pelo operador. Para cada uma dessas fases desenvolvemos funções para auxiliar no processo de análise. Essas funções estão descritas nas subseções a seguir.

Uma ressalva a ser feita é que a função que determina o caminho inicial e final da interação não tem como ser genérica, dado que depende das marcações dos lugares do modelo.

3.6.1 – Funções para Determinar os Estados Inicial e Final da Interação

Dado que se deseja analisar todos os possíveis caminhos entre dois estados da interação, o primeiro passo é definir quais são esses estados. Essa tarefa não é trivial, pois os parâmetros de busca são variáveis dependendo do grau de precisão que se deseja na busca. Podemos querer buscar estados que dependam apenas de alguns elementos presentes na interface a estados que dependam de todos os elementos na interface. Isso se traduz na quantidade de lugares do modelo que devem ter suas fichas testadas de acordo com o estado que se pretende determinar.

Essa função não pode ser desenvolvida com um caráter genérico, dado que depende dos elementos da interface do modelo no qual se está trabalhando, e que podem mudar em uma

nova instanciação do modelo. Mostraremos aqui o desenvolvimento voltado para o modelo de nosso estudo de caso, mas sempre mostrando como generalizar para a adequação para outros contextos.

Foram desenvolvidas algumas que retornam todos os estados que atendam a um certo predicado. Essas funções testam a marcação dos lugares da rede em busca de marcações que modelem os estado que desejamos determinar. Como os tipos das fichas nos lugares pode variar (fichas de um elemento, duplas, listas, etc.), funções para cada tipo de ficha foi criado e depois foram agregadas em uma função geral que busca um estado a partir de todos os elementos presente na interface (que se traduzem nos lugares da rede).

Na criação dessas funções utilizamos duas funções pré-definidas para análises presentes na ferramenta de Occ do Design/CPN [20]: *PredAllNodes* e *SearchNodes*, além de algumas funções auxiliares desenvolvidas para fornecer os parâmetros nos casos em que usamos a função *SearchNodes*.

A função *SearchNodes* possui uma série de parâmetros para descrever como a busca por estados dever ser efetuada, a saber: qual partição do espaço de estados dever ser percorrida, qual o predicado que deve ser satisfeito, quantas vezes o predicado deve ser avaliado limitando o número de iterações, o que deve ser retornado como resposta a execução da função, o valor inicial de busca, e a forma como os diversos resultados devem ser combinados para serem apresentados ao fim da execução da função.

Já a função *PredAllNodes* é uma simplificação da *SearchNodes* aonde alguns argumentos já são predefinidos. Neste caso a função retorna todos os nós a partir do nó inicial que atendam ao predicado que definimos. Uma explicação mais detalhada acerca dessas funções é apresentada em [20].

Para a criação das funções é necessário ter conhecimento da linguagem de programação utilizada no Design/CPN, a linguagem ML. Informações sobre essa linguagem podem ser encontradas em [21].

Apresentaremos a seguir as funções desenvolvidas e indicaremos como a função mais genérica desenvolvida (que busca todos os elementos que desejamos) pode ser alterada para outros casos específicos (casos em que não desejamos testar o estado de todos os elementos da interface). E, ressaltamos que essas funções também podem ser utilizadas para se determinar estados específicos para análise, e não apenas como base para se determinar os estados inicial e final de uma busca. Como exemplo podemos utilizar uma função de busca para achar os estados cuja marcação de um lugar seja definida e com isso

estudar a marcação dos outros lugares da rede para verificarmos todo o entorno desses estados determinados.

A partir do entendimento dessas funções é possível se definir uma função de busca de estados aplicável a qualquer outro modelo, tornando-a assim uma função de estrutura genérica, embora aqui esteja instanciada para o nosso estudo de caso.

3.6.1.1 – Função para determinar nós a partir da marcação de um lugar onde as fichas são de apenas um elemento

Neste caso queremos encontrar todos os nós do Occ (estados do modelo) nos quais a marcação de um determinado lugar de uma dada página tenha valor igual ao parâmetro de busca da função. Essa função é aplicada a lugares cujas fichas sejam de apenas um elemento, sendo o retorno da função todos os estados em que esse lugar tenha uma ficha com o valor do parâmetro, independendo do valor das demais fichas que podem estar presentes. Como no nosso modelo não temos elementos duplicados, não nos interessou inserir essa restrição à função.

Para a criação dessa função foi utilizada a função *PredAllNodes*. Uma dificuldade encontrada foi como definir qual a página e o lugar a ser efetuada a pesquisa, dado que esses parâmetros fazem parte da função. Como não foi encontrada uma solução, é necessário sempre modificar a função para definir qual a página e o lugar de pesquisa. Depois é só executar a função inserindo como parâmetro o tipo da ficha que se quer encontrar no lugar. O retorno dessa função, assim como todas as outras que serão explanadas é uma lista contendo os nós que atendem o predicado definido. No Anexo A desse trabalho temos um exemplo de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e de sua utilização.

3.6.1.2 – Função para determinar nós a partir da marcação de dois lugares onde as fichas são de apenas um elemento

Este caso é muito parecido com o anterior, só que ao invés de buscarmos apenas os estados ligados a um determinado lugar, temos dois lugares. Com isso só obtemos como retorno os estados em que os dois predicados são verdadeiros. Os dois lugares não precisam estar na mesma página do modelo, dado que cada predicado é definido individualmente, mas o teste é realizado nos dois predicados utilizando o operador booleano AND. No Anexo A apresentamos um exemplo da definição de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e dois exemplos de sua utilização.

3.6.1.3 – Função para achar nós cuja marcação de um dado lugar seja uma dupla

A diferença desta função para a primeira função apresentada é o fato de neste caso as fichas no lugar a ser testado serem uma dupla, com o parâmetro de pesquisa sendo os dois elementos da dupla. No Anexo A desse trabalho temos um exemplo de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e de sua utilização.

3.6.1.4 – Função para achar nós cuja marcação seja uma dupla fornecendo como parâmetro apenas o segundo elemento da dupla

Nesta função o parâmetro de pesquisa é apenas o segundo elemento de uma dupla, com a função retornando todos os estados nos quais esse predicado é verdadeiro. No Anexo A desse trabalho temos um exemplo de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e de sua utilização.

3.6.1.5 – Função para achar nós cuja marcação seja uma lista de duplas fornecendo como parâmetro apenas o primeiro elemento da dupla que encabeça a lista

Similar a função anterior, sendo que neste caso temos uma lista de duplas ao invés de apenas duplas. No Anexo A desse trabalho temos um exemplo de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e de sua utilização.

3.6.1.6 – Função geral para achar estados do modelo fornecendo como parâmetro todos os elementos de interesse deste estudo

Esta função engloba as idéias das funções anteriores e é a função base para a definição dos estados inicial e final para os estudos dos caminhos de interação. Aqui são definidos todos os elementos de interesse para a busca dos estados. Como no nosso estudo de caso temos três linha de tensão, é necessário definir o estado de todos os elementos dessas linhas (o disjuntor, as duas seccionadoras e a chave local/telecomando de cada linha) e também a janela ativa da interface. Observe que, embora essa função esteja instanciada para nosso estudo de caso, ela pode ser instanciada para qualquer outro modelo, sendo necessário definir os lugares de interesse pra busca e os elementos dentro desses estados no corpo da função. Não conseguimos uma forma de deixar essa função genérica, mas podemos observar que sua instanciação para outros contextos não é uma tarefa que demande muito esforço por parte do projetista.

Essa função não tem parâmetros de entrada, pois todos os elementos estão definidos no corpo da função. Como não foi possível parametrizar a definição dos lugares

e dos elementos da busca, se faz necessário mudar esses parâmetros quando a busca não for para o estado dos dispositivos da mesma forma que o definido nessa função. Nesta instanciação da função está definido que todos os disjuntores estão fechados, todas as seccionadoras estão fechadas, todas as chaves local/telecomando estão no estado local.

Como o estado de um dispositivo é indicado pelo presença da ficha que modelo o dispositivo no lugar que modela o estado do mesmo, para ajustar a função para um outro estado do dispositivo devemos alterar o nome do lugar na linha da função correspondente ao teste desse elemento.

E, para buscas menos estritas, em que não desejamos testar o estado de todos esses componentes, deve-se alterar a função retirando as linhas de teste dos componentes que não fazem mais parte dos parâmetros de busca. No Anexo A desse trabalho temos um exemplo de uma função com esse comportamento aplicado ao contexto de nosso estudo de caso e de sua utilização.

A seguir temos um exemplo da chamada dessa função.

```
def_est();
```

O retorno dessa função é o seguinte:

```
val def_est = fn : Janela * chave * chave * chave * chave * chave * chave * chave
* chave * chave * chave * chave * chave -> Node list
val it = [8,24] : Node list
```

Temos a validação sintática da função e em seguida uma lista com os estados do Occ que atendem ao predicado de busca. Nesse caso temos que apenas os estado 8 e 24 estão de acordo com o estado da interface definido no corpo da função.

A partir da análise da estrutura das funções apresentadas é possível se criar outras funções para busca de nós do Occ para outros predicados.

3.6.2 – Função *AllPath* (Função para Determinar os Caminhos entre dois Estados)

A função AllPath foi concebida por [91] como uma tentativa de se superar uma limitação de análise da ferramenta Design/CPN, que é a busca por caminhos entre dois estados do espaço de estados do modelo. O Design/CPN possui uma função que retorna apenas um dos possíveis caminhos entre dois estados (normalmente o menor), e, consideramos interessante do ponto de vista de usabilidade em interfaces, a determinação de todos os caminhos possíveis entre dois estados. A função AllPath foi criada para atender exatamente a essa necessidade.

Os parâmetros dessa função são as marcações inicial e final para pesquisa (estados inicial e final da pesquisa), e o tamanho do caminho a ser analisado (número de passos para se alcançar o estado final de pesquisa). O valor zero faz com que sejam retornados os caminhos de qualquer comprimento.

Segundo [91]: "Nesta função, inicialmente são verificados todos os nós de saída adjacentes ao nó correspondente à marcação inicial da pesquisa (função *OutNodes* da ferramenta *Design/CPN*). A partir do exame destes nós de saída, vai sendo formada uma lista com o caminho em direção ao nó correspondente a marcação final da pesquisa, respeitando-se o comprimento especificado. Mesmo apresentando recursividade, este procedimento é finito uma vez que trabalha com um número finito de nós de saída (*OutNodes*), e cada chamada a função considera o próximo nó de saída na seqüência". No Anexo A é apresentado o código da função AllPath e suas funções auxiliares.

Porém com o uso dessa função observamos algumas limitações. Devido à natureza exponencial da análise dos caminhos, essa função só é útil para modelos com poucos estados (100 a 200). Para modelos mais complexos o tempo de resposta da função é muito grande, ou mesmo não há uma resposta (ocorre o estouro da capacidade de memória reservada ao Design/CPN, travando o aplicativo), devido a limitações inerentes ao hardware disponível para execução da função.

O fato disto acontecer é que a função percorre todo o espaço de estados, mesmo quando definimos o tamanho do caminho a ser buscado. Ele testa todos os caminhos possíveis descartando todos os que não forem do tamanho especificado. Mas, toda essa informação fica armazenada na memória do computador utilizado.

Estudando a função e o uso a que a mesma se propõe, verificamos que a mesma não deveria continuar percorrendo caminhos em que o tamanho definido do caminho fosse extrapolado, bem como deveríamos definir um limite para as iterações. Com isso a função foi alterada para contemplar essas características.

Os parâmetros da função agora são: o estado inicial, o estado final, o tamanho do caminho e o número de iterações da função. Agora, o tamanho do caminho diz respeito a todos os caminhos que tenham até o número de passos determinados como parâmetro (antes só eram retornados caminhos com apenas o tamanho determinado). Outra diferença é que agora consideramos que o Occ foi todo percorrido se ele testou todos os caminhos até o tamanho definido, não sendo de interesse testar os caminhos maiores. A função então tem como informação de retorno com os caminhos que satisfazem o parâmetro de tamanho

máximo de passos, o número de iterações realizadas e se o espaço de estados foi todo percorrido (de acordo com o que acabamos de definir para este caso).

Temos no Anexo A o código da nova função AllPath e suas funções auxiliares. Esta função está salva como um arquivo SML para facilitar a portabilidade para trabalho com outros modelos. Um arquivo SML é um arquivo de biblioteca para ser utilizado no Design/CPN. Ao invés de se escrever uma dada função repetidas vezes, cria-se um arquivo texto com a definição da função e o salva com a extensão SML. A partir daí é só indicar o caminho onde este arquivo está armazenado no computador no nó de declaração do modelo, que a função fica disponível para ser executada. A biblioteca para a verificação de modelos utilizando lógica temporal (ASK/CTL) e a biblioteca para a comunicação com processos externos via TCP/IP (COMMS/CPN) são exemplos de arquivos SML existentes.

Um exemplo de uso dessa função é mostrado a seguir.

```
use "/home/scaico/CPN/N6AllPath.sml";
AllPath(8,40,10,5000000);
```

Como pode ser observado, a chamada da função é para todos os caminhos de tamanho máximo de dez passos iniciando no estado 8 e finalizando no estado 40. Também é definido que serão executadas no máximo 5000000 iterações. O retorno dessa função é o seguinte:

```
[opening /home/scaico/CPN/N6AllPath.sml]
val listLen = fn : 'a list -> int
val Member = fn : ''a * ''a list -> bool
val Append = fn : 'a list * 'a list -> 'a list
exception nthElemExcept
val nthElem = fn : 'a list * int -> 'a
exception firstElemExcept
val firstElem = fn : 'a list * int -> 'a list
val printListOfInt = fn : int list -> unit
val AllPathAux = fn
  : int ref * int * int * int * int * int ref * int list ref -> unit
val AllPath = fn : int * int * int * int -> unit
val it = () : unit

List:
[ 8, 27, 40 ]
[ 8, 26, 39, 195, 352, 1281, 40 ]
[ 8, 25, 38, 180, 343, 1250, 40 ]
[ 8, 24, 37, 165, 333, 1198, 40 ]
[ 8, 23, 36, 150, 322, 1137, 40 ]
[ 8, 22, 35, 135, 310, 1061, 40 ]
```

```
[ 8, 21, 34, 120, 297, 970, 40 ]
[ 8, 20, 33, 105, 283, 864, 40 ]
[ 8, 19, 32, 90, 268, 743, 40 ]
[ 8, 16, 30, 63, 239, 529, 40 ]
[ 8, 15, 29, 51, 225, 444, 40 ]

List Size: 11

AllPath finished - Iterations: 388062

val it = () : unit
```

O retorno da função apresenta uma mensagem para a verificação sintática de cada função que compõe a *AllPath* e em seguida lista os caminhos encontrados. Foram encontrados 11 caminhos de no máximo 10 passos. Foram necessárias 388062 iterações para se percorrer todo o espaço de estados do modelo para o caso de caminhos entre o nó 8 e 30.

3.6.3 – Função para Comparar dois Estados

Uma vez com os caminhos determinados pela função AllPath, é necessário analisar esses caminho observando quais as ações realizadas em cada um deles. Com isso pode-se verificar as alternativas de interação, bem como detectar interações que não deveriam ser passíveis de realização.

O problema aqui é exatamente fazer a análise, onde se faz necessário comparar os estados de cada caminho para verificar as ações realizadas. O Design/CPN permite que se verifique a marcação dos lugares do modelo em um determinado estado, utilizando a função *DisplayNodes*. Só que para comparar os estados dos caminhos se faz necessário analisar cada nó e compará-lo com seu nó subseqüente, o que é uma tarefa tediosa, que demanda um tempo razoável dependendo do número de caminhos e do tamanho dos caminhos encontrados, e sujeita a erros devido a quantidade de lugares que devem ser comparado para se determinar as mudanças entre os estados.

Essa tarefa então foi automatizada pela definição de uma função que compara dois nós do espaço de estados e retorna apenas os lugares cuja marcação difere. O retorno dessa função é no mesmo estilo do apresentado na função *DisplayNodes*. O que essa função faz é a partir do número de dois nós, pegar a marcação e transformar numa lista de strings, comparando os elementos de mesma posição das strings e retornando apenas os que

diferem. No Anexo Apresentamos o código dessa função. Essa função também está disponível em um arquivo do tipo SML.

Um exemplo de uso dessa função é mostrado a seguir.

```
use "/home/scaico/CPN/compnode.sml";
cnodes(169,296);
```

Seu retorno é:

```
[opening /home/scaico/CPN/compnode.sml]
/home/scaico/CPN/compnode.sml:20.5-22.44 Warning: match nonexhaustive
          (nil,nil,resp) => ...
          (cab1 :: l1,cab2 :: l2,resp) => ...

val cnodes = fn : Node * Node -> unit
val descr = fn : Node -> unit
val it = () : unit

Difference between node 169 and node 296 is:
Login'Perm_Nav 1: empty --- Login'Perm_Nav 1: 1`nav_perm
Finalizacao'Perm_Nav 2: empty --- Finalizacao'Perm_Nav 2: 1`nav_perm
Finalizacao'Perm_Nav 1: empty --- Finalizacao'Perm_Nav 1: 1`nav_perm
Navegacao'Perm_Nav 1: empty --- Navegacao'Perm_Nav 1: 1`nav_perm
Disjuntor'Perm_Nav1 1: empty --- Disjuntor'Perm_Nav1 1: 1`nav_perm
Disjuntor'Perm_Nav2 1: empty --- Disjuntor'Perm_Nav2 1: 1`nav_perm
Disjuntor'Perm_Nav 1: empty --- Disjuntor'Perm_Nav 1: 1`nav_perm
Disjuntor'Perm_Nav3 1: empty --- Disjuntor'Perm_Nav3 1: 1`nav_perm
Seccionadora'Perm_Nav2 1: empty --- Seccionadora'Perm_Nav2 1: 1`nav_perm
Seccionadora'Sec_Aberta  1:  1`(LT01Y5,SC31Y54)  ---  Seccionadora'Sec_Aberta  1:
1`(LT01Y5,SC31Y54)++ 1`(LT01Y5,SC31Y55)
Seccionadora'Sec_Atua1 1: 1`(LT01Y5,SC31Y55) --- Seccionadora'Sec_Atua1 1: empty
Seccionadora'Perm_Nav 1: empty --- Seccionadora'Perm_Nav 1: 1`nav_perm
Seccionadora'Perm_Nav1 1: empty --- Seccionadora'Perm_Nav1 1: 1`nav_perm
val it = () : unit
```

Como pode ser observado, após a validação sintática da função, são apresentados os lugares cuja marcação foi alterada, mostrando a marcação dos dois nós para cada lugar.

Também foi desenvolvida uma função que retorna as marcações de um estado do modelo sem ter a necessidade de se usar a função *DisplayNodes* presente no menu Occ do Design/CPN. Essa função e um exemplo de sua utilização são apresentados a seguir. Essa função está contida no mesmo arquivo SML da função anterior.

```
fun descr(x) = CPN'Edit.out_val(!st_NodeDescrRef(x));
descr(169);
```

O retorno dessa função é:

```
169
Login'Perm_Nav 1: empty
IHM_Industrial'Login 1: empty
IHM_Industrial'Fim 1: 1`BotFinalizaProg
IHM_Industrial'IHM 1: 1`[(JanSinotico,[BotFechar,BotAjuda])]
Finalizacao'Perm_Nav 2: empty
Finalizacao'Perm_Nav 1: empty
Navegacao'Bot_Close 1: 1`BotAjudaFechar
Navegacao'Jan_Nav2 1: 1`(JanAjuda,[BotAjudaFechar])
Navegacao'Jan_Nav1 1:
       1`(JanLogin,[BotLogout,BotFinalizaProg,BotAjuda,BotSinotico,BotAlarme,
       BotEventos,BotHisto,BotTend])++
       1`(JanAlarme,[BotFechar,BotAjuda])++1`(JanSinotico,[BotFechar,BotAjuda])++
       1`(JanEventos,[BotFechar,BotAjuda])++1`(JanHisto,[BotFechar,BotAjuda])++
       1`(JanTend,[BotFechar,BotAjuda])
Navegacao'Opcoes2 1: 1`(JanAjuda,BotAjuda)
Navegacao'Opcoes1 1: 1`(JanLogin,BotFechar)++1`(JanAlarme,BotAlarme)++
       1`(JanSinotico,BotSinotico)++ 1`(JanEventos,BotEventos)++
       1`(JanHisto,BotHisto)++ 1`(JanTend,BotTend)
Navegacao'Perm_Nav 1: empty
Disjuntor'Disj_Aberto 1: empty
Disjuntor'Esp_Conf_Fech 1: empty
Disjuntor'Disj_Fechado 1: 1`(LT01Y3,DJ21Y3)++ 1`(LT01Y4,DJ21Y4)++
       1`(LT01Y5,DJ21Y5)
Disjuntor'Esp_Conf_Abrir 1: empty
Disjuntor'Local 1: 1`(LT01Y3,LocTelLt01Y3)++ 1`(LT01Y4,LocTelLt01Y4)++
       1`(LT01Y5,LocTelLt01Y5)
Disjuntor'Disj_Atua 1: empty
Disjuntor'Disj_Atua1 1: empty
Disjuntor'Perm_Nav1 1: empty
Disjuntor'Perm_Nav2 1: empty
Disjuntor'Perm_Nav 1: empty
Disjuntor'Perm_Nav3 1: empty
Seccionadora'Perm_Nav2 1: empty
Seccionadora'Sec_Aberta 1: 1`(LT01Y5,SC31Y54)
Seccionadora'Esp_Conf_Fech 1: empty
Seccionadora'Sec_Fechada 1: 1`(LT01Y3,SC31Y34)++ 1`(LT01Y3,SC31Y35)++
       1`(LT01Y4,SC31Y44)++ 1`(LT01Y4,SC31Y45)
Seccionadora'Esp_Conf_Abrir 1: empty
Seccionadora'Local 1: 1`(LT01Y3,LocTelLt01Y3)++ 1`(LT01Y4,LocTelLt01Y4)++
       1`(LT01Y5,LocTelLt01Y5)
Seccionadora'Sec_Atua 1: empty
Seccionadora'Sec_Atua1 1: 1`(LT01Y5,SC31Y55)
Seccionadora'Perm_Nav 1: empty
```

```
Seccionadora'Perm_Nav1 1: empty
val it = () : unit
```

Como pode ser observado pelo retorno dessa função, o número de lugares nesse modelo é grande o suficiente para ocasionar erros de interpretação e demandar bastante tempo caso a comparação entre estados seja realizada de forma manual.

Para a comparação de todo um caminho entre dois estados, utilizamos a função *descr* nos estados inicial e final para termos a representação completa do início e fim do caminho, e utilizamos a função *cnodes* entre cada dois nós adjacentes do caminho para obtermos as mudanças de estado nesse caminho.

3.7 Verificação de Modelos CPN (*Model Checking*)

A verificação de modelos é um método automático onde um conjunto de especificações sobre o comportamento do modelo (descritas em uma lógica temporal proposicional) é confrontado com o espaço de estados do mesmo, verificando se o modelo satisfaz essas especificações. O uso de verificação de modelos tem a vantagem de ser um procedimento totalmente automático e exaustivo no espaço de estados do modelo [89]. E, ao se especificar um comportamento desejado (uma propriedade), caso essa propriedade não seja verdadeira um contra-exemplo é apresentado pelo verificador de modelos, facilitando a análise. Porém, [9] alerta para o fato de que é necessário experiência para se escrever as especificações de comportamento em lógica temporal, e que demanda mais experiência ainda para entender as especificações escritas por outras pessoas.

A lógica temporal é um tipo de lógica especialmente desenvolvida para a definição de especificações comportamentais que envolvam a noção de ordem temporal [9]. Para essa tarefa diversos operadores temporais foram disponibilizados. As propriedades (ou especificações) que se desejam verificar são descritas em lógica temporal por meio de fórmulas construídas por proposições atômicas em lógica proposicional clássica em conjunto com os operadores temporais [89].

A lógica temporal pode ser classificada em Lógica Temporal Linear (LTL – Linear Temporal Logic) e Lógica Temporal Ramificada (CTL – Computational Tree Logic) [9][89]. Na Lógica Temporal Linear só há um futuro possível, enquanto na Lógica Temporal Ramificada vários futuros são admitidos. Uma definição sobre lógica temporal CTL e LTL pode ser obtida em [9][53].

No caso de modelos em CPN as especificações são formuladas em uma extensão da lógica temporal CTL denominada ASK/CTL [19], a qual é disponibilizada como uma biblioteca para uso no Design/CPN. Essa extensão ao CTL faz com que na verificação sejam levados em conta os estado e as transições do espaço de estados do modelo, e não apenas só os estados como ocorre no CTL [26][89].

3.7.1 – ASK/CTL

A biblioteca ASK/CTL pode ser usada, através do Design/CPN, para a verificação de modelos CPN. Essa biblioteca possui tanto a definição da linguagem da lógica do ASK/CTL como também um verificador de modelo [19]. O verificador toma uma fórmula (especificação de propriedade) em ASK/CTL como argumento e verifica se a mesma é válida no espaço de estados do modelo. Uma restrição a essa biblioteca é que não foi implementada a geração de um contra-exemplo caso a fórmula verificada não seja verdadeira.

As fórmulas podem ser escritas no domínio dos estados ou das transições, gerando então duas categorias de fórmulas: de estados ou de transições. Os operadores do ASK/CTL podem ser aplicados tanto a estados quanto a transições, e a semântica do operador vai depender então do domínio no qual estamos trabalhando [19]. A seguir são apresentados de forma sucinta os operadores do ASK/CTL [19] para fórmulas de nós:

1. TT: operador booleano que especifica que a condição é verdadeira
2. FF: operador booleano que especifica que a condição é falsa
3. NOT(A): operador booleano que nega a proposição A
4. AND(A_1,A_2): operador booleano que retorna verdadeiro se as proposições A_1 e A_2 são verdadeiras
5. OR(A_1,A_2): operador booleano que retorna verdadeiro se pelo menos uma das proposições A_1 ou A_2 é verdadeira
6. NF(<expressão de nó>): operador que indica que a proposição deve ser verificada nos nós (estados) do espaço de estados
7. AF(<expressão de nó>): operador que indica que a proposição deve ser verificada nos arcos (transições) do espaço de estados
8. EXIST_UNTIL(A_1,A_2): operador usado para verificar se existe um caminho a partir de nó atual no qual a proposição A_1 é verdadeira em todos os estados do caminho até o último estado no qual A_2 é verdadeiro

9. FORALL_UNTIL(A_1,A_2): operador análogo a EXIST_UNTIL, sendo que ao invés de procurar apenas um caminho, a condição deve ser verdadeira para todos os caminhos
10. POS(A): operador usado para verificar se existe um caminho a partir do estado atual que termine em um estado aonde a proposição A é verdadeira
11. INV(A): análogo a POS(A), só que para todos os caminhos possíveis
12. EV(A): operador usado para verificar se, para todos os caminhos, é sempre possível chegar a um estado no qual a proposição A é verdadeira
13. ALONG(A): operador usado para verificar se existe um caminho no qual a proposição A é sempre verdadeira, podendo este caminho ser infinito ou acabar em um estado terminal (*dead marking*)
14. MODAL(A): operador de mudança de domínio; e no caso de ser uma fórmula de estado a mesma será avaliada como verdadeira se existir uma transição imediata ao estado atual na qual o argumento A é verdadeiro a partir dessa transição
15. EXIST_MODAL(A_1,A_2): também é um operador de mudança de domínio, e no caso de ser uma fórmula de estado, ela será avaliada como verdadeira se existir um estado sucessor ao atual no qual A_2 seja verdadeira e que na transição entre o estado atual e esse sucessor A_1 seja verdadeira
16. FORALL_MODAL(A_1,A_2): similar ao EXIST_MODAL, mas neste caso A_2 deve ser verdadeiro para todos os sucessores imediatos do estado atual e A_1 deve ser verdadeiro para todas as transições entre o estado atual e seus sucessores
17. EXIST_NEXT(A): operador utilizado para verificar se existe um sucessor imediato do estado atual no qual A é verdadeiro
18. FORALL_NEXT(A): similar a EXIST_NEXT, sendo neste caso necessário que A seja verdadeiro para todos os sucessores imediatos

A verificação de uma fórmula é realizada pelo verificador utilizando o comando *eval_node*. Um exemplo é apresentado a seguir.

```
fun PA n = (Mark.Switchgear'SG_Open 1 n = ((1,(TL01Y3,SG31Y34))!!empty));
fun PB n = (Mark.Switch_Breaker'CB_Open 1 n = ((1,(TL01Y3,SB21Y3))!!empty));
val check = POS(AND(NF("Error",PD),NF("Error",PB)));
eval_node checka InitNode;
```

Inicialmente temos as proposições PA e PB que são definições de marcações para os lugares *SG_Open* e *CB_Open* (o significado desses lugares não é relevante para o entendimento do exemplo de verificação). Em seguida temos *check* que é a fórmula que

será avaliada e que verifica se é possível, a partir do nó atual, alcançar um estado no qual tanto PA quanto PB são verdadeiros, ou seja, alcançar um estado em que as marcações dos lugares *SG_Open* e *CB_Open* sejam *(TL01Y3,SG31Y34)* e *(TL01Y3,SB21Y3)*, respectivamente. O passo seguinte é a avaliação dessa fórmula a partir do estado inicial do modelo (*InitNode*). O retorno dessa verificação é mostrado a seguir, indicando que a fórmula foi avaliada como verdadeira, ou seja, que existe pelo menos um caminho a partir do estado inicial do modelo cujo estado final satisfaz tanto PA quanto PB.

```
val it = true : bool
```

Para a verificação de comportamentos no modelo se faz necessário definir quais comportamentos da interface são interessantes de teste, e isso depende das necessidades de análise, e escrever as proposições e fórmulas correspondentes. No capítulo que apresenta as análises aplicadas ao estudo de caso mostramos alguns exemplos de uso de *model checking* aplicado a esse contexto.

3.8 Estratégia Proposta para a Análise de Modelos

Os métodos de análise apresentados nesse capítulo podem ser utilizados em conjunto ou separadamente, de acordo com as necessidades de análise do modelo. Apresentaremos agora algumas propostas de utilização dos métodos.

As simulações podem ser utilizadas para verificar a adequação da instanciação do modelo CPN ao novo contexto de estudo, verificando o comportamento dos objetos de interação e dos mecanismos de navegação.

Se nas análises se faz necessário a integração do usuário final do sistema, a simulação e os MSCs devem ser utilizados de forma a apresentar as idéias de projeto, a evolução do comportamento do modelo a partir de cenários representativos, e estudar as alternativas de interação. O uso de simulações permite a apresentação passo a passo da evolução do modelo, permitindo discussões sobre alternativas de interação disponíveis e comportamento do sistema.

O uso de MSCs é uma maneira de simplificar a apresentação de cenários de evolução do modelo. Isso permite apresentar o comportamento do modelo sem ter que apresentar também o modelo, escondendo assim as dificuldades inerentes ao entendimento do modelo CPN por parte do usuário. Com isso o usuário e o projetista podem discutir o comportamento do sistema e verificar se o mesmo está de acordo com as especificações.

As análises baseadas no espaço de estados do modelo visam verificar a usabilidade e as alternativas (corretas ou não) de interação disponíveis na interface. A proposta de análise neste caso é de iniciar verificando as propriedades de usabilidades.

Em seguida o projetista deve definir quais são os tipos de interações que deseja verificar, e utilizar as funções de testes de caminho desenvolvidas nesse trabalho para gerar todos os possíveis caminhos de realização dessa tarefa. Ele então deve analisar os caminhos, verificando quais são as alternativas corretas e as com incidência de erros. No caso das alternativas corretas ele deve verificar quais devem estar disponíveis de acordo com as restrições de projeto (segurança na operação, tempo, etc). E para as alternativas incorretas ele deve propor soluções para extingui-las ou minimiza-las.

Se existem comportamentos específicos que devem ser garantido em todos os estados alcançáveis do sistema, teste utilizando *model checking* devem ser realizados para verificar se esses comportamentos estão de acordo ou não com as especificações.

Por fim, todas as mudanças no modelo que foram detectadas nas etapas de análise devem ser inseridas no modelo e refeitas as análises para garantir que o modelo da interface está de acordo com as especificações do comportamento da interface a ser projetada/reprojetada, gerando assim uma documentação acerca de como os mecanismos de navegação e interação devem ser implementados para garantir as especificações.

Capítulo 4 – Estudo de Caso - Análise da Usabilidade e do Erro Humano na Interação com a IHM de uma Subestação de um Sistema Elétrico

Este trabalho tem como objetivo o estudo da interação realizada pelo operador de um sistema informatizado de supervisão e controle de uma subestação de energia elétrica, focando na interação com os dispositivos de proteção e controle das linhas de tensão presentes na subestação. A partir da modelagem e análise desse ambiente podemos verificar que aspectos da interface podem ser melhorados de forma a garantir algumas propriedades de usabilidade relevantes, bem como verificar como minimizar os erros de operação por parte dos operadores do sistema. A partir das análises no modelo desse sistema é que pretendemos mostrar que o uso de métodos formais é útil e factível de ser aplicado como parte de uma metodologia de projeto de interfaces ergonômicas que leva em consideração o erro humano envolvido no processo de interação entre o usuário e o sistema.

Inicialmente apresentaremos o ambiente industrial utilizado para o estudo de caso, que é uma parte de uma subestação de transmissão e distribuição de energia elétrica da empresa CHESF – Companhia Hidro-Elétrica do São Francisco [16], definindo de forma sucinta os sistemas de transmissão e distribuição de energia elétrica, e quais os principais dispositivos de proteção e comando existentes. Em seguida apresentaremos o software supervisório empregado para o controle do sistema, o SAGE – Sistema Aberto para Gerenciamento de Energia [75]. Por fim apresentaremos o modelo CPN do sistema.

4.1 Estudo de Caso

4.1.1 – A CHESF

Para a realização do estudo de caso foi escolhida uma empresa de transmissão e distribuição de energia de grande porte que atua na região nordeste do Brasil, a CHESF (Companhia Hidro-Elétrica do São Francisco). A escolha da empresa resultou do fato da mesma possuir um sistema crítico do qual depende um grande número de consumidores de energia elétrica, além de grandes indústrias, e várias concessionárias de transmissão e distribuição de energia.

Outro fator de escolha é o fato da empresa possuir desde sistemas com operação totalmente manual até sistemas totalmente informatizados e operados remotamente, bem como sistemas híbridos onde parte do sistema pode ser operada tanto em modo manual quanto remoto. E, como mais um fator determinante da escolha temos o fato de a empresa ter nos disponibilizado acesso a relatório de acidentes, rotinas operacionais e às instalação da mesma.

A CHESF é uma das maiores e mais importantes empresas do setor elétrico brasileiro, com mais de 5461 empregados, e que garante o provimento de energia elétrica para os estados nordestinos e parte dos estado do Pará e Tocantins [16]. A CHESF possui atualmente quatorze usinas hidrelétricas e duas usinas termoelétricas, e atende a uma área geográfica de 1.219.983 quilômetros quadrados (15% do território brasileiro), com dezoito mil quilômetros de linhas através do estados atendidos pela empresa. A população beneficiada chega a mais de 40 milhões de habitantes (25% da população do Brasil) e tem uma capacidade instalada 10.704 megawatts, sendo 10.271 megawatts de origem hidráulica e 432 megawatts de origem termoelétrica [16].

Via de regra a CHESF atende os grandes consumidores de energia elétrica, tais como indústrias e concessionárias de energia, ficando a cargo das concessionárias o provimento de energia elétrica residencial.

O sistema da CHESF é divido em 6 subsistemas, como segue: Subsistema Norte, Subsistema Sul, Subsistema Leste, Subsistema Oeste, Subsistema Centro e Subsistema Sudoeste. É no subsistema Leste que se encontra a subestação utilizada no estudo de caso, a subestação de Campina Grande II (CGD).

4.1.2 – Subestação de Energia Elétrica

Uma subestação é uma parte do sistema de potência, concentrada em um dado local, compreendendo primordialmente as extremidades de linhas de transmissão ou distribuição, com os respectivos dispositivos de manobras, controle e proteção, podendo incluir transformadores, equipamentos conversores e/ou outros [54]. A subestação é o componente responsável pela interconexão entre as várias partes do sistema elétrico, podendo ser de vários tipos (seccionadoras, elevadoras, abaixadoras, distribuidoras, etc) [54].

Em uma subestação de transmissão típica existem linhas de entrada, cuja tensão pode ser repassada para linhas de saída no mesmo nível ou após rebaixamento. A ligação entre as linhas de entrada e saída se dá através de barramentos. Cada linha de entrada e saída, assim como as entradas do barramento possuem circuitos de proteção próprios, de modo a aumentar a segurança do sistema. Na Figura 10 é apresentado um diagrama simplificado de uma subestação de distribuição de energia elétrica, que serve aos propósitos deste estudo. Nesta figura há uma linha de tensão de entrada, e três linhas de saída, das quais duas recebem uma tensão rebaixada a partir da linha de tensão de entrada.

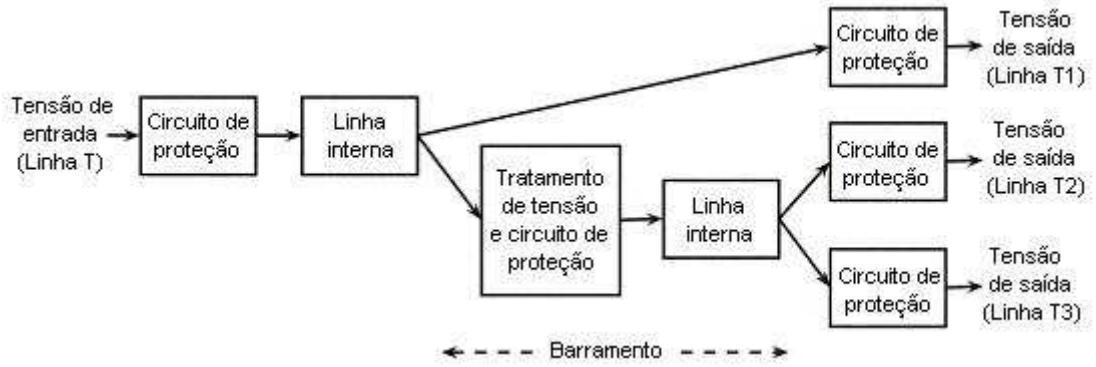


Figura 10: Diagrama de uma Subestação de Energia Elétrica

O sistema de proteção de uma subestação tem como objetivo eliminar uma falta, isolando a parte afetada do restante do sistema no menor tempo possível para garantir e manter a integridade dos equipamentos e assegurar a continuidade do fornecimento de energia elétrica [54].

A proteção de uma linha de transmissão é realizada por um disjuntor em série com duas chaves seccionadoras (uma antes e uma depois do disjuntor). Um disjuntor é um dispositivo de proteção de dois estado (aberto e fechado) cuja função é abrir o circuito na ocorrência de uma falta (essa abertura se dá automaticamente). Ele pode ser religado manualmente (pelo operador local ou remoto) ou através de um circuito de religamento

automático. As seccionadoras também são elementos de dois estados, mas que são acionadas exclusivamente pelos operadores. Sua função é dar uma segurança extra na ocorrência da falta, permitindo ao operador isolar o disjuntor para a realização dos procedimentos de manutenção.

Esse circuito de proteção recebe a tensão do barramento principal (se for uma linha de saída), ou envia a tensão para o barramento principal. Existe também um barramento auxiliar, que é utilizado no processo de transferência de proteção. É quando, por algum motivo (falta ou manutenção no disjuntor da linha), se faz necessário transferir a proteção da linha para o disjuntor de transferência, que é um disjuntor auxiliar para a proteção de uma linha quando da impossibilidade de se utilizar a proteção da própria linha. Para realizar a transferência o circuito de proteção da linha tem em paralelo uma chave seccionadora chamada *bypass* que liga a saída da linha ao barramento auxiliar permitindo assim a ligação do circuito de proteção do disjuntor de transferência e o desligamento do circuito de proteção da linha. Com isso pode-se efetuar manutenções no circuito de proteção da linha mantendo ainda a linha provida de um circuito de proteção. Na Figura 11 temos uma representação do circuito de proteção (disjuntor e seccionadoras) de duas linhas de transmissão e do circuito de proteção utilizando o disjuntor de transferência.

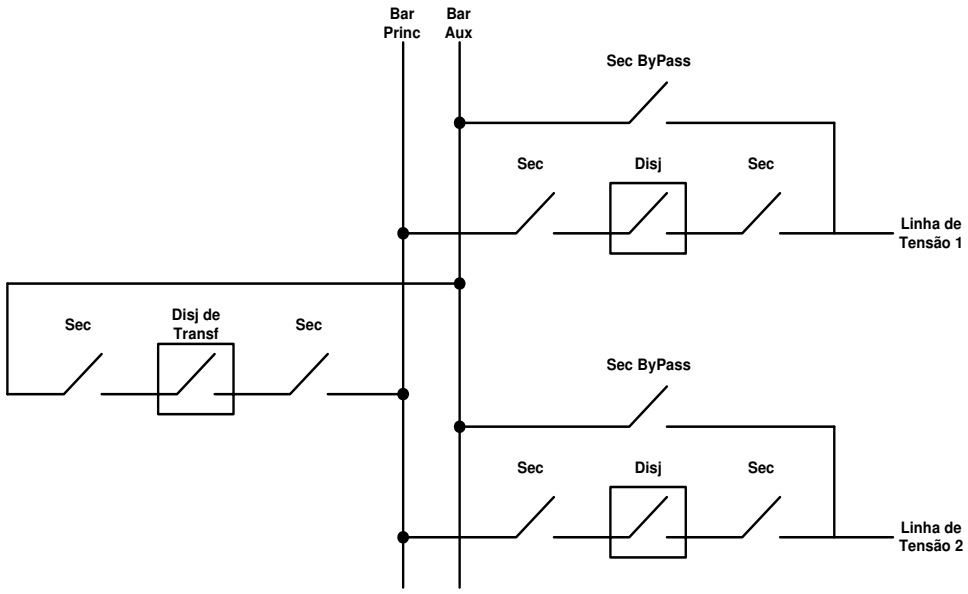


Figura 11: Diagrama do Circuito de Proteção

Esse circuito de proteção possui vários elementos associados para o seu funcionamento: chave local/telecomando (define se a operação dos equipamentos da linha é realizada pelo operador local ou remoto), chave de desbloqueio do disjuntor (em decorrência de uma falta o disjuntor pode bloquear como medida de segurança, sendo necessário desbloqueá-lo), o circuito de detecção de faltas (que envia um comando de

abertura ao disjuntor na ocorrência de uma falta), o circuito de religamento automático do disjuntor (que tenta religar automaticamente o disjuntor após o mesmo ter sido desligado, bloqueando-o após um número pré-definido de tentativas), e a chave de transferência de proteção (que define se a proteção da linha é realizada pelo disjuntor da linha, pelo disjuntor de transferência, ou pelos dois simultaneamente).

4.1.3 – A Subestação CGD

A subestação CGD recebe energia na tensão de 230KV e a transmite para várias linhas de transmissão de saída nas tensões de 128KV, 69KV e 13,8KV, além de possuir uma parte de distribuição de energia para consumidores locais em 220V. Na Figura 12 temos o diagrama unifilar de CGD.

CGD é um sistema composto por 5 linhas de entradas, todas em 230kV oriundas das subestações de Tacaimbó, Pau Ferro e Goianinha. CGD possui 16 linhas de transmissão de saída, onde quatro linhas na tensão de 230KV alimentam o sistema da cidade de Natal; duas linhas em 138kV alimentam a cidade de Santa Cruz; seis linhas em 69KV suprem de energia elétrica o brejo paraibano, alimentando as subestações de Campina Grande I e Bela Vista, as cidades de São João do Carirí, Boqueirão e Esperança, e a empresa Embratex; e quatro linhas em tensão de 13.8kV alimentado as cidades de Pocinhos e Queimadas, e os bairros do Velame e Distrito Industrial da cidade de Campina Grande.

O nosso estudo se concentrou no barramento de 69KV, mais especificamente nos dispositivos de interação de proteção e controle disponíveis ao operador do sistema, buscando alternativa de interação de modo a melhorar a operação do sistema do ponto de vista de segurança na interação com os dispositivos de proteção e controle.

A escolha por esse subsistema (barramento de 69KV) se deve ao fato do mesmo possuir em suas saídas linhas de transmissão tanto totalmente manuais quanto totalmente informatizadas, o que nos oferece um grande campo de estudo para trabalhos futuros. E, vale salientar que dispositivos de interação de proteção e controle são os mesmos não importando a tensão de operação.

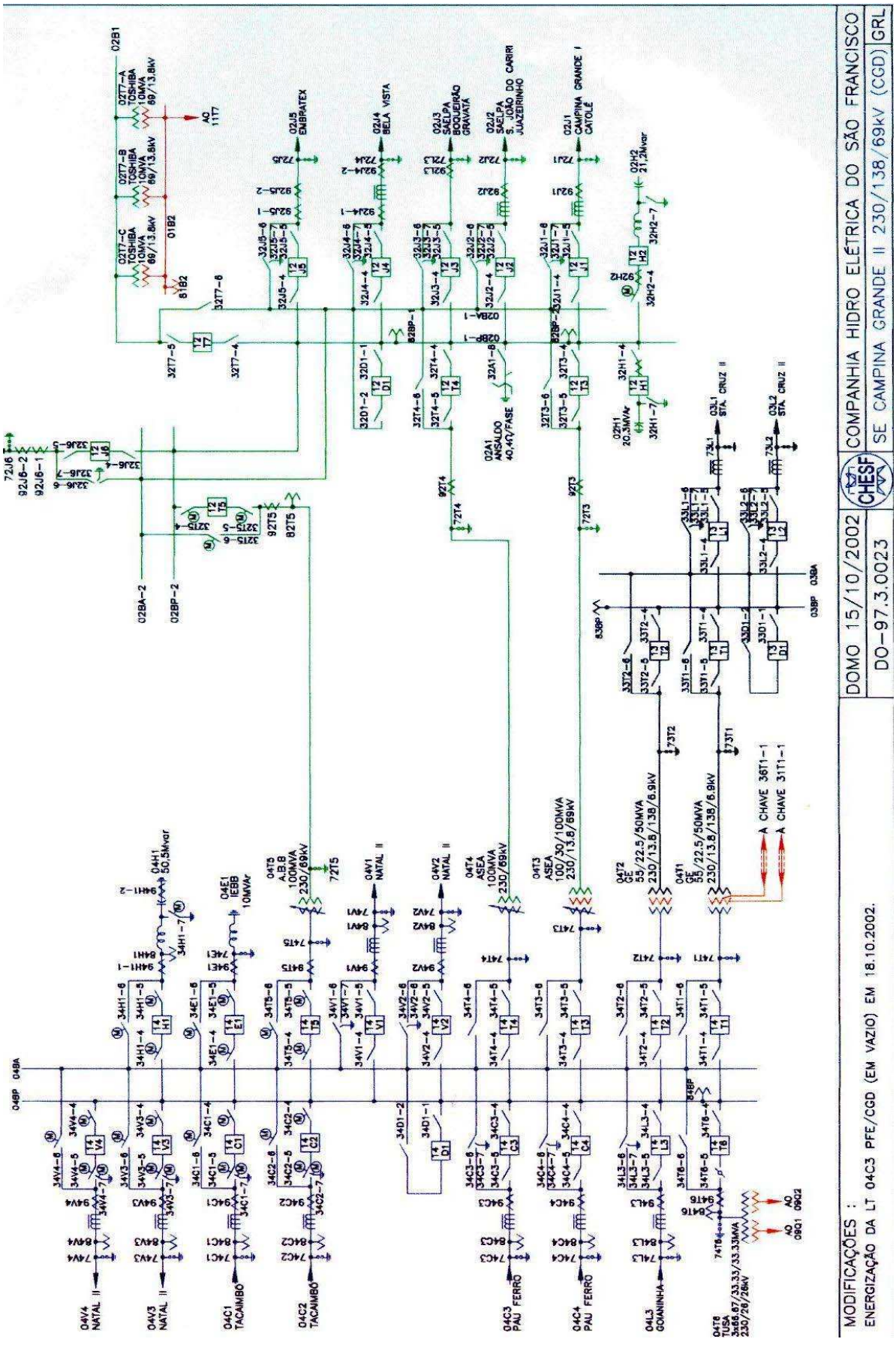


Figura 12: Diagrama Unifilar de CGD

No barramento de 69KV, que serve como base para este trabalho, pode-se destacar três grandes grupos de elementos de proteção e controle, quais sejam: a proteção de entrada do barramento, a proteção de saída do barramento para as linhas de tensão (denominada proteção das linhas de tensão) e o circuito de proteção auxiliar utilizando o disjuntor de transferência (composto pelo disjuntor de transferência e todos os elementos associados ao mesmo no processo de transferência de proteção).

A proteção de entrada e as proteções das linhas de tensão possuem a mesma configuração podendo ser considerados como um único tipo de dispositivo, enquanto o circuito de proteção utilizando o disjuntor de transferência é uma versão um pouco simplificada desses outros dispositivos (possui menos elementos).

No atual estágio deste estudo trabalhamos apenas com os elementos de proteção e controle de três linhas de saída, com cada linha contendo um disjuntor, duas seccionadoras e uma chave local/telecomando.

4.1.4 – O Sistema Supervisório SAGE

O SAGE (Sistema Aberto para Gerenciamento de Energia) é um software de supervisão e controle criado especificamente para centros de controle de energia elétrica [4][70]. Uma característica importante do SAGE é o fato de ser um sistema aberto, ou seja, é uma arquitetura portável, expansível, modular e interconectável [4][70]. Isso significa que ele foi criado de modo a funcionar em diferentes plataformas de *hardware*; que o seu *hardware* ou *software* pode ser expandido na medida que for necessário; que suas funções são implementadas por módulos diferentes, permitindo a fácil adição ou remoção de módulos; e que ele pode conectar diferentes plataformas de *hardware* por meio de uma rede de comunicação de dados.

Além disso o SAGE é um sistema escalável, o que permite seu uso em todos os níveis de supervisão e controle, da operação de uma subestação ao controle de todo o sistema [4]. Isso torna mais fácil a operação do sistema, devido a sua padronização em todos os níveis hierárquicos de controle. E essa padronização também reduz custos de manutenção e treinamento.

Entre os módulos oferecidos pelo SAGE para o operador do sistema podemos destacar [3]:

- Visor de acesso: tela inicial do programa, usada para o login do usuário.
- Visor de alarmes: mostra textualmente os alarmes.
- Visor de logs: apresenta o histórico dos alarmes e eventos. Pode-se definir as datas de início e fim para a apresentação do histórico, bem como o tipo de informação que será exibida (alarme, evento).
- Visor de histórico: é como o visor de logs, só que de forma gráfica.
- Visor de tendências: mostra gráficos de tendências para as variáveis do processo.
- Visor de telas: é o sinótico da planta. É nesta janela que o usuário atua nos elementos da planta, e toda ação com um elemento requer uma confirmação como forma de reduzir os erros de operação.

  Da Figura 13 a Figura 15 mostramos exemplos das interfaces de alguns desses módulos.

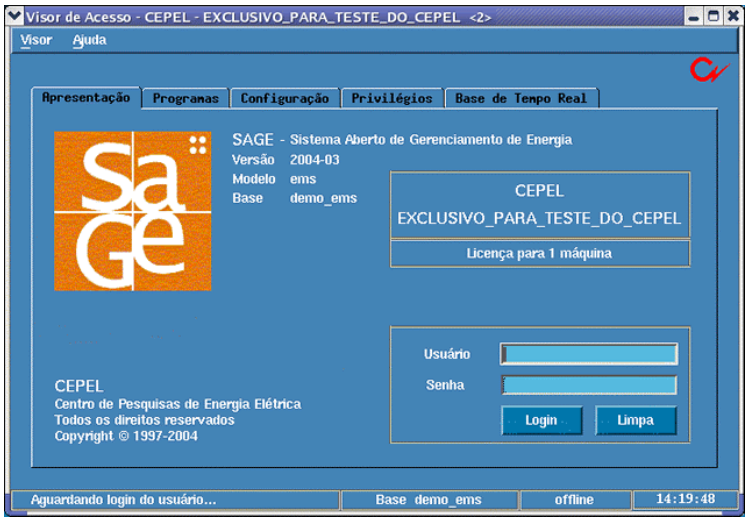


Figura 13: Visor de Acesso

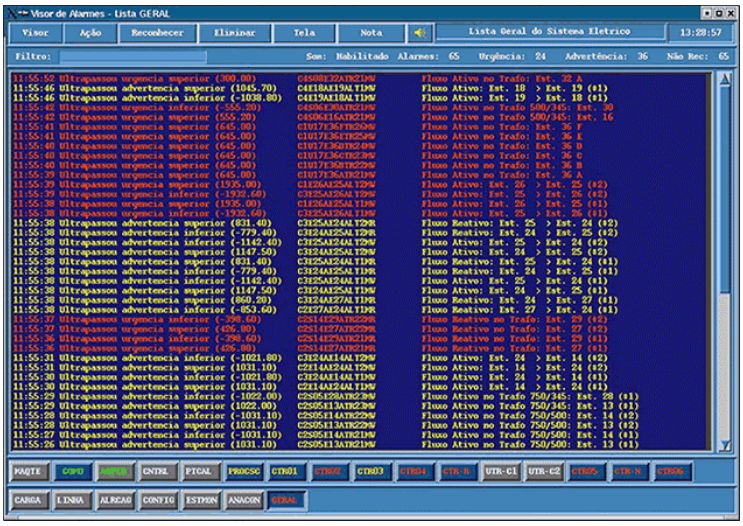


Figura 14: Visor de Alarmes

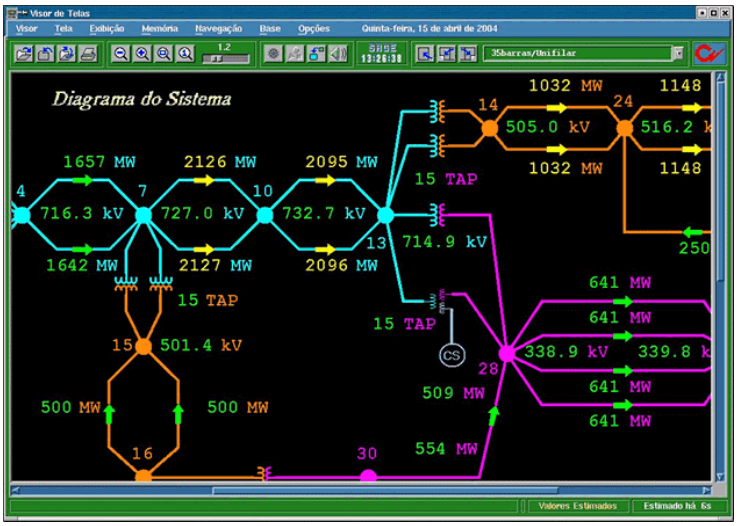


Figura 15: Visor de Telas

Antes do uso do SAGE o processo de supervisão e o controle das subestações era realizado por meio de painéis com sinalizadores luminosos, botoeiras, etc, como mostra a Figura 16.

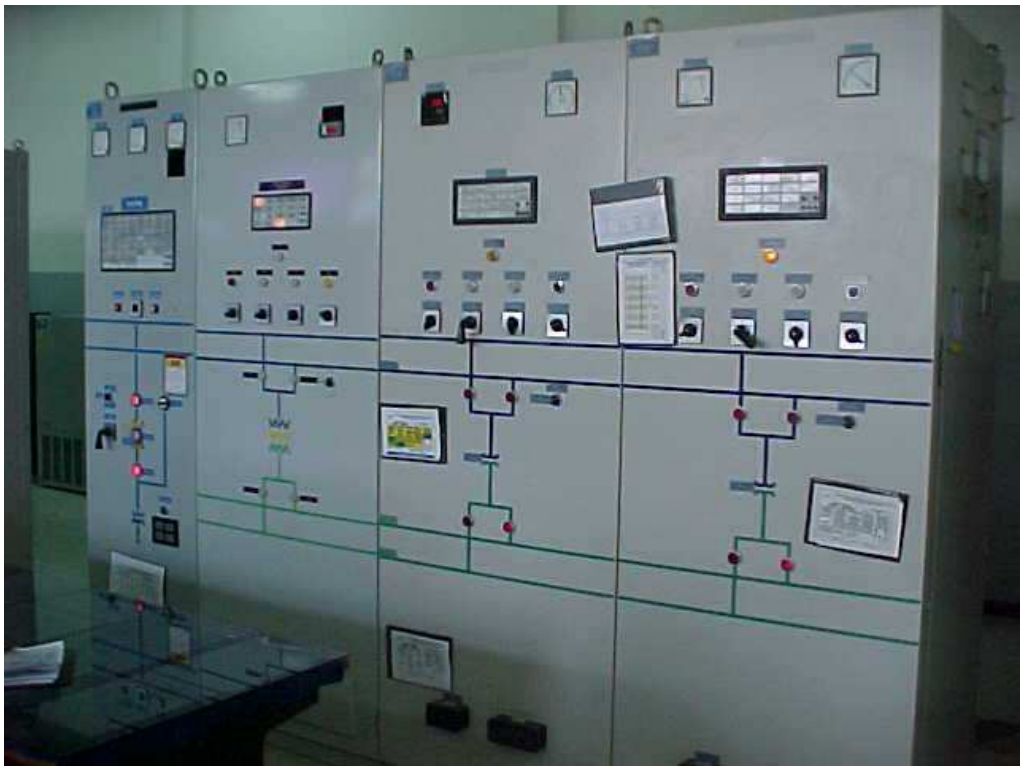


Figura 16: Exemplo de Painel de Operação de uma Subestação

Com a mudança do controle dos painéis para um software supervisório (o SAGE) tivemos uma mudança no ambiente de trabalho dos operadores do sistema, pois onde antes existia um painel com vários elementos a serem fisicamente operados, agora se tem uma interface informática com a operação sendo realizada por meio do mouse e teclado. Em

alguns casos isso acarretou aos operadores a ilusão de não estarem mais operando os dispositivos, devido a não estarem mais operando fisicamente os mesmo. E com isso surgiram novos problemas de interação, provenientes de desconhecimento de uso da nova ferramenta de trabalho e desse aparente distanciamento dos equipamentos da subestação.

Este trabalho se baseia no uso do SAGE para o controle da subestação de CGD, mais especificamente o barramento de 69KV dessa subestação. Após visitas a subestação e leitura de material fornecido pela CHESF foi elaborado o modelo do sistema que será apresentado na próxima seção.

4.1.5 – O Erro Humano na Chesf

O erro humano é uma preocupação constante na CHESF. A empresa documenta todos os acidentes que ocorrem em suas instalações e constatou que o principal causador é o erro humano. Esses relatórios de acidentes possuem o objetivo de documentar os acidentes ocorridos, e a partir daí estudá-los objetivando a diminuição de sua ocorrência e de suas conseqüências [3].

Especificamente abordando os relatórios de acidentes causados por erros humanos (chamados de RFO – Relatório de Falha Operacional), eles traçam um paralelo entre o que foi realizado pelo operador e o que deveria ser executado de acordo com o roteiro de manobra específico[4], delineando em que parte da operação ocorreu o erro humano.

As falhas humanas na CHESF estão relacionadas a diversos fatores, tais como:

- Descumprimento de normativo
- Desconhecimento de configuração
- Sinalização inadequada
- Tempo inadequado
- Desatenção
- Imperícia
- Falta de capacidade técnica
- Falta de motivação (falta de incentivos, treinamento, salário, etc.)
- Estresse (fadiga, pressão, apoio logístico, hora-extra, transporte, etc.)
- Relacionamento deficiente

[4] Um roteiro de manobra é um normativo da empresa contendo todos os passos que devem ser seguidos para a execução de uma dada tarefa de operação do sistema.

E, foi realizado um estudo, apresentado em [39], verificando quais os tipos de erros humanos mais usuais na CHESF. Constatou-se que a maioria dos erros é devido a:

- Acréscimo de uma ação
- Ação correta sobre o objeto errado
- Omissão de uma ação
- Execução incompleta

Observe que os erros humanos mais recorrentes na CHESF são exatamente os mesmos tipos de erros que desejamos verificar e minimizar utilizando o modelo formal da interação.

4.2 O Modelo CPN da IHM do sistema SAGE na SE CGD

Nesta seção iremos apresentar o modelo CPN da interface utilizado neste trabalho. Como o modelo é hierárquico, apresentaremos primeiro a visão hierárquica de mais alto nível do modelo, seguindo com a descrição de cada parte que compõe esse modelo.

O modelo CPN aqui apresentado, conforme já mencionado, é uma representação dos elementos de supervisão e controle de uma parte de uma subestação de energia elétrica, mais especificamente um barramento da subestação CGD da CHESF. A modelagem foi baseada no programa de controle supervisório de subestação SAGE. A partir da interface do SAGE o operador do sistema pode monitorar o estado do sistema e controlar os diversos dispositivos presentes na subestação.

O propósito desse esforço de modelagem é de prover ao projetista um modelo a partir do qual propriedades de usabilidade da interface possam ser garantidas através da análise do modelo; como também permitir a investigação de estratégias de navegação e interação a partir de simulações e análises no espaço de estados do modelo. Devido ao caráter generalista do modelo, o mesmo pode ser utilizado para modelar outros barramento e subestações da Chesf, bem como de outras empresas de transmissão e distribuição de energia elétrica, com um esforço mínimo de adaptação ao novo contexto.

Um outro objetivo desse modelo é de utilizá-lo na criação de um simulador de treinamento de operadores cujo motor de simulação seja baseado em modelos formais do sistema [105]. Como o presente trabalho contempla o modelo da interface via programa supervisório do sistema, se faz necessário a integração do modelo da interface com o modelo do sistema, e com uma representação da interface em um programa supervisório

comercial, sendo com essa representação que o operador vai interagir (do mesmo modo que no SAGE). A integração das partes que compõem o simulador se dará por meio de mensagens TCP/IP [34]. Porém, como o simulador não é o objetivo deste trabalho, e sim um desdobramento do mesmo, sua arquitetura será apresentada nos anexos e sua concepção estará contida nas propostas de trabalhos futuros.

Neste trabalho foram modelados os mecanismos de navegação na interface no supervisório e o comportamento de alguns dos elementos de interação disponíveis ao operador do sistema, a saber: disjuntores, seccionadoras e chaves de comando local/telecomando. Foi dada ênfase nos mecanismos de navegação entre janelas e nos dispositivos de interação mais utilizados para a operação da subestação. Na Figura 17 temos a página de hierarquia do modelo.

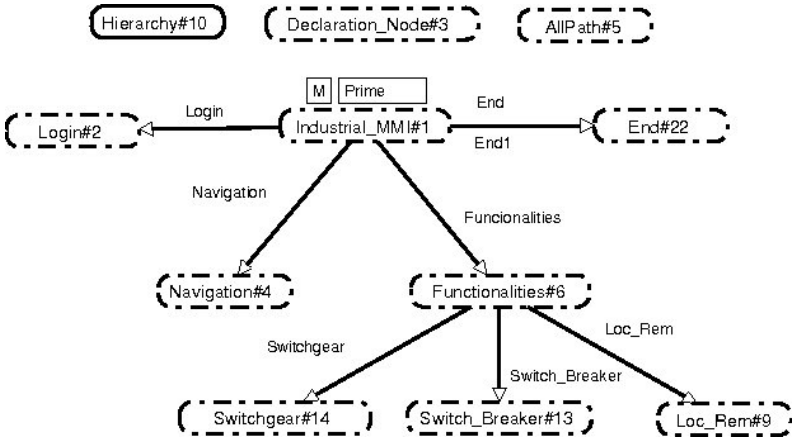


Figura 17: Página de Hierarquia do Modelo

Como pode ser observado na Figura 17, o modelo é composto por vários módulos, a saber: login (submodelo *Login*), navegação (submodelo *Navigation*), encerramento da execução do aplicativo (submodelo *End*) e funcionalidades (submodelo *Functionalities*). As funcionalidades neste caso são os disjuntores (submodelo *Switch_Breaker*), as chaves seccionadoras (submodelo *Switchgear*) e as chaves local/telecomando (submodelo *Loc_Rem*). A função e o modelo de cada um desses módulos serão apresentados nas subseções a seguir.

Antes de apresentar os módulos vamos descrever o nó de declaração (*Declaration_Node)* do modelo com a definição dos tipos e variáveis utilizadas.

```
color Window = with WinLogin | WinAlarm | WinPlant | WinEvents | WinHisto |
               WinHelp | WinTend;
color Buttom = with ButLogin | ButLogout | ButClose | ButPlant | ButAlarm |
               ButEvents | ButHisto | ButTend | ButHelpClose | ButHelp |
               ButEndProg;
color Options = product Window*Buttom;
color List_Buttom = list Buttom;
color Interface = product Window*List_Buttom;
color List_Interface = list Interface;
```

```
color Line1 = with TL01Y3 | TL01Y4 | TL01Y5;
color Switch = with SB21Y3 | SB21Y4 | SB21Y5 | (* Switch Breakers *)
             SG31Y34 | SG31Y35 | SG31Y44 | SG31Y45 | SG31Y54 | SG31Y55 |
             (* Switchgear *) LocRemTl01Y3 | LocRemTl01Y4 | LocRemTl01Y5 |
             (* Switches Local-Remote *) sg1 | sb1 | sg2 | begin1 |begin2;
color Act_Switch = product Line1*Switch;
color Command = with Open | Close;
color Command_Field = product Line1*Switch*Command;
color A_Nav = with allow_nav;

var Win, Win1, Win2: Window;
var But, But1: List_Buttom;
var Action, Action1: Buttom;
var List: List_Interface;
var interface: Interface;
var TL: Line1;
var sw, sw1, sw2: Switch;
var Act: Act_Switch;
var com: Command;
var Act_Field: Command_Field;


(* Function to verify if the element is available *)

fun exist(Action,[]) = false |
      exist(Action,Op::But) =
             if Op=Action then true
             else exist (Action,But);

(* Function to verify the interblock of the switchgear *)

fun interblock(ch,ch1) = if (ch=SG31Y34 andalso ch1=SB21Y3) then true
                  else if (ch=SG31Y35 andalso ch1=SB21Y3) then true
                  else if (ch=SG31Y44 andalso ch1=SB21Y4) then true
                  else if (ch=SG31Y45 andalso ch1=SB21Y4) then true
                  else if (ch=SG31Y54 andalso ch1=SB21Y5) then true
                  else if (ch=SG31Y55 andalso ch1=SB21Y5) then true
                  else false;
```

O nó de declaração possui as seguintes cores (tipos):

- Window: Conjunto de todas as janelas existentes no sistema
- Buttom: Conjunto de todos os elementos de interação para navegação e seleção de opções na janela ativa, excetuando os elementos de interação que atuam em elementos físicos da subestação
- Options*:* É uma dupla que contém o nome de uma janela e o nome de um botão
- List_Buttom*:* É uma lista cujos elementos são do tipo *Buttom*
- Interface*:* É um produto, onde o primeiro elemento é o nome de uma janela e o segundo é uma lista do tipo *List_Buttom*
- List_Interface: É uma lista de itens do tipo *Interface*
- Line1: Conjunto das linhas de tensão presentes na subestação
- Switch: Define os elementos de interação que atuam nos elementos físicos da subestação (disjuntores, seccionadoras e chaves local/telecomando)
- Act_Switch: É um produto, onde o primeiro elemento é uma linha de tensão e o segundo é uma chave (disjuntor, seccionadora ou chave local/telecomando)

- A_Nav: Elemento que permite que ocorra navegação na interface

4.2.1 – Página Principal do Modelo

Na Figura 18 é apresentada a superpágina (página principal) do arcabouço em Redes de Petri. Essa página é o ponto de partida para a execução do modelo. Cada transição dessa página é um submodelo de um dos aspectos da interface.

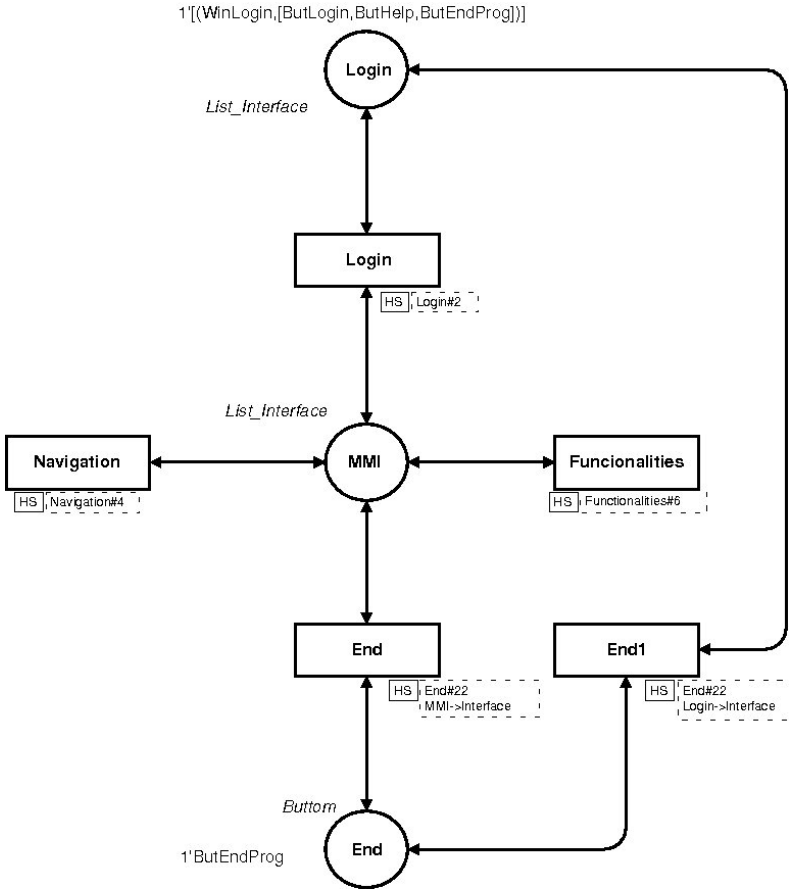


Figura 18: Página Principal do Modelo

O lugar *Login* possui uma ficha quando não existe um operador autenticado no sistema, ou seja, quando não existe um operador atuando na planta via programa supervisório. Esse é o estado inicial do sistema e, com a autenticação do operador temos a criação de uma ficha no lugar *MMI*.

O lugar *MMI* indica qual a janela ativa em um dado instante de tempo durante uma sessão de utilização do programa supervisório por parte de um operador do sistema. Seu valor é uma lista da cor *Interface*, onde o primeiro item da lista indica a janela atualmente ativa com suas opções de navegação (descrição da janela), e os outros itens são os passos utilizados (janelas navegadas) para chegar a essa janela (na ordem inversa de ocorrência).

A partir desse lugar é que o modelo se desenvolve em relação à navegação e as funcionalidades.

Na superpágina do arcabouço temos a transição *Login* que é uma transição de substituição para o modelo de autenticação/desautenticação no sistema, a transição *Navigation* que é uma transição de substituição para o modelo de navegação da interface, e a transição *Funcionalities* é uma transição de substituição para o modelo das funcionalidades de interface.

Temos também nesta página as transições *End* e *End1* e o lugar *End*, que modelam o encerramento da execução do aplicativo. Como o ato de encerrar o aplicativo não causa uma mudança no estado da interface e sim o encerramento do aplicativo, decidimos não inseri-lo nem no modelo de navegação, nem no modelo das funcionalidades.

4.2.2 – Modelo da Autenticação/Desautenticação no Sistema (Submodelo *Login*)

A ação inicial para se poder usar o sistema é se autenticar no mesmo. Só após a autenticação é que as opções de navegação e interação ficam disponíveis ao operador. Para encerrar sua interação com o sistema o operador deve efetuar sua desautenticação. A necessidade disso diz respeito ao controle e auditoria do sistema.

Essa parte do sistema está na subpágina *Login* do modelo do sistema, apresentada na Figura 19.

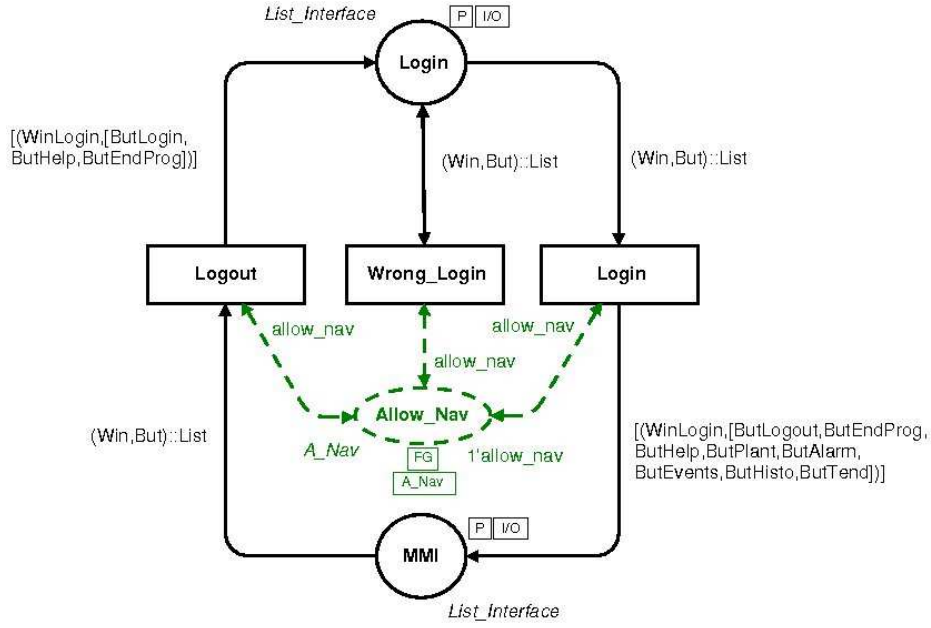


Figura 19: Modelo da Autenticação/Desautenticação no Sistema

Inicialmente existe uma ficha no lugar *Login* indicando que não existe usuário autenticado no sistema. A partir daí podem ocorrer a transição *Login* (modelando a autenticação no sistema) ou *Wrong_Login* (modelando um erro de autenticação). E a transição *Logout* modela o encerramento da sessão ativa no sistema. Como não é intenção desse trabalho verificar o processo de autenticação ou de auditoria do sistema, e sim o de navegação e interação com os elementos de controle do sistema, resolvemos não detalhar aspectos do tipo usuário e senha.

Ao ocorrer a autenticação (disparo da transição *Login*) uma ficha é criada no lugar *MMI* (lugar de fusão presente em todas as páginas do modelo) indicando que o usuário ainda está na página de *Login* (página central no processo de navegação), mas que agora as opções de navegação e interação estão disponíveis.

O lugar *Allow_Nav* modela uma restrição à navegação, pois existem opções de interação que possuem mais de um passo e que devem ser finalizada antes de se poder navegar ou selecionar outra opção. Então, para se iniciar uma ação de navegação ou uma funcionalidade se faz necessário a presença de uma ficha nesse lugar. Esse lugar é um lugar de fusão presente em todas as subpáginas do modelo.

4.2.3 – Modelo de Navegação (Submodelo *Navigation*)

O modelo de navegação trata dos aspectos de interação referentes à mudança da janela ativa. Na Figura 20 esse modelo é apresentado. Nele estão representados os tipos de navegação mais comumente encontrados em interfaces industriais, que são:

- Navegação entre janelas com o fechamento da janela ativa e abertura da nova janela.
- Navegação entre janelas onde a janela ativa é sobreposta por uma nova janela, que se torna então a janela ativa.
- Fechamento da janela ativa que está sobreposta a outra janela.

Temos a seguir um detalhamento de cada um desses tipos de navegação. Nesse modelo, como pode ser observado, temos a presença do lugar de fusão *Allow_Nav*, cuja função já foi explicada.

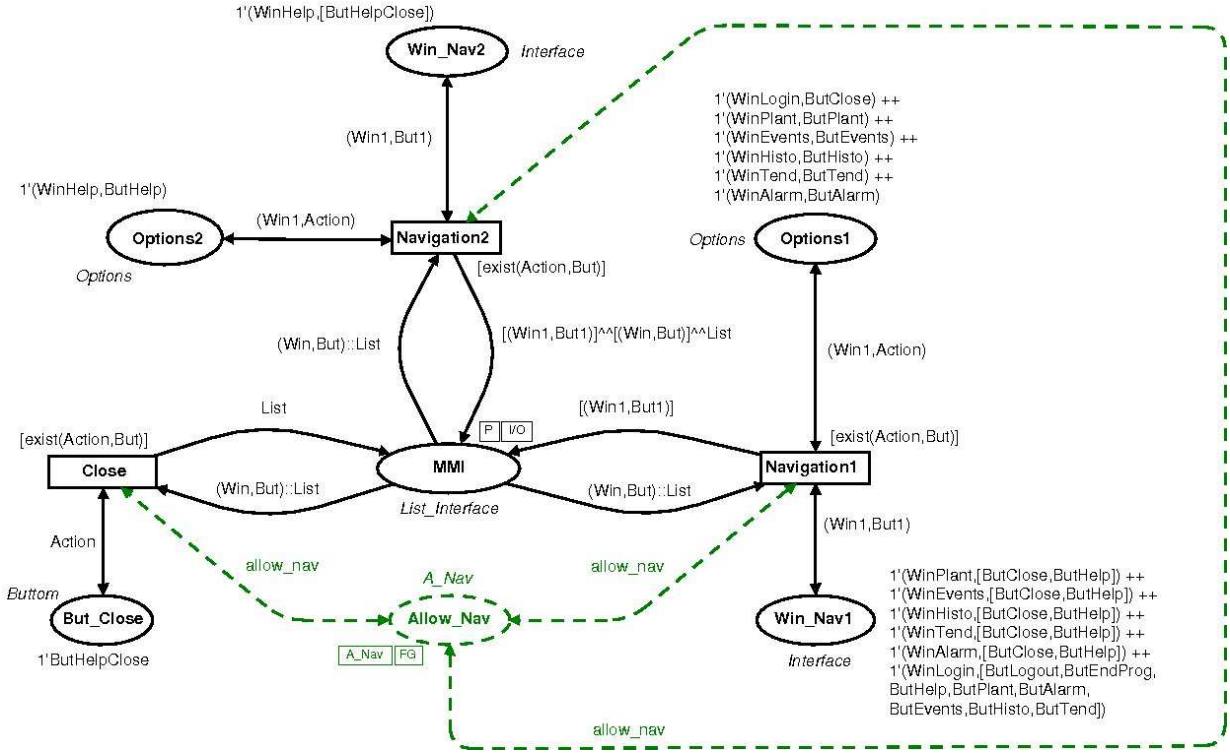


Figura 20: Modelo de Navegação

4.2.3.1 – Navegação entre Janelas com o Fechamento da Janela Ativa

Consiste na navegação entre janelas, com o fechamento da janela ativa e das janelas que estiverem superimpostas a ela, e a abertura da janela solicitada. É modelada pela transição *Navigation1* e pelos lugares *Win_Nav1* e *Options1*. A guarda (restrição à ocorrência) desta transição é a função "*exist*", que verifica se a ação de navegação selecionada está disponível na janela ativa. Ou, em outras palavras, se a ação é permitida nesta janela.

O lugar *Win_Nav1* contém a descrição das janelas da interface, enquanto o lugar *Options1* contém duplas que relacionam uma ação de navegação a uma janela. Para *Navigation1* ocorrer deve existir na janela ativa (primeiro elemento do lugar *MMI*) uma opção de navegação que exista também em *Options1*. A opção em *Options1* define qual é a janela que deve ser aberta (através da ficha no lugar *Win_Nav1*). Com isso a ficha selecionada em *Win_Nav1* vai ser colocada na lista presente em *MMI*, sendo o único elemento da lista.

4.2.3.2 – Navegação entre Janelas com Sobreposição de Janela

Consiste na navegação entre janelas, com a janela requisitada se sobrepondo à janela ativa, tornando-se a nova janela ativa. A nova janela ativa é então adicionada à lista de interface.

A transição *Navigation2* e os lugares *Win_Nav2* e *Options2* modelam esta navegação. A guarda desta transição é a função *"exist"*.

A ocorrência de *Navigation2* é análoga a de *Navvigation1*. A única diferença aqui é que a lista do lugar *IHM* vai ser acrescida da nova janela (ao invés da nova janela ser o único elemento da lista).

4.2.3.3 – Fechamento da Janela Ativa

Consiste no fechamento da janela atualmente ativa, e sobreposta a uma outra janela. É modelada pela transição *Close* e pelo lugar *But_Close*. Em *But_Close* estão fichas que relacionam uma ação de navegação a uma janela indicando qual janela deve ser fechada se esta opção de navegação for selecionada. A guarda desta transição e á função *"exist"*.

A transição *Close* tem como parâmetros de entrada a cabeça da lista que está no lugar *MMI* e uma das fichas do lugar *But_Close* (que indica a ação a ser tomada na janela ativa para fecha-la). Sendo verdadeira a verificação feita pela execução da função *exist*, a janela ativa é fechada (é retirada da cabeça da lista do lugar), retornando à janela a partir da qual essa janela tinha sido requisitada (nova cabeça da lista na janela *MMI*).

4.2.4 – Modelo do Encerramento do Aplicativo (Submodelo *End*)

Este modelo representa o ato de se encerrar o aplicativo e na Figura 21 o mesmo é apresentado.

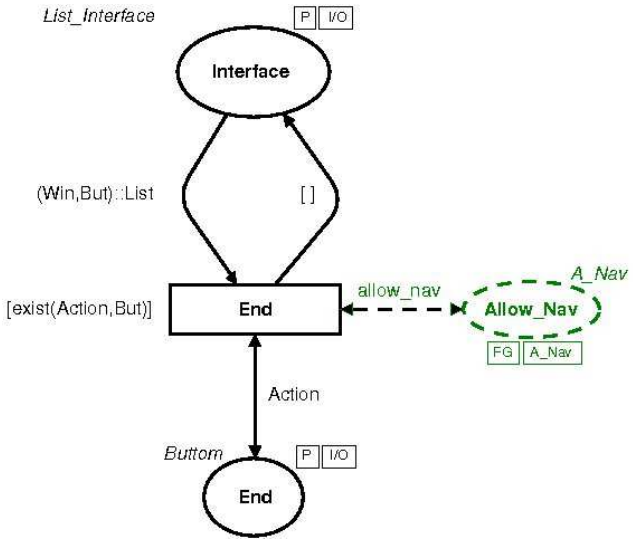


Figura 21: Modelo do Encerramento de Execução do Aplicativo

Este modelo é composto apenas pela transição *End* e pelos lugares *End* e *Interface*, que modelam o encerramento da execução do aplicativo. A guarda da transição *End* é a

função *exist* (explicitada no nó de declaração), cuja finalidade é verificar se a ação de navegação (opção de navegação) pretendida existe na janela ativa. A ocorrência dessa transição retira a ficha do lugar *MMI* se o usuário fechar o aplicativo sem se desautenticar do sistema, ou do lugar *Login* caso não esteja autenticado (esse modelo tem duas instâncias como pode ser observado na página principal do modelo), devolvendo uma lista vazia impedindo qualquer nova evolução do modelo. Existe também o lugar *Allow_Nav*, cuja função já foi explicitada.

4.2.5 – Modelo das Funcionalidades (Submodelo *Functionalities*)

Este modelo agrupa os modelos dos elementos de proteção e controle do sistema. Para o reuso do modelo não necessitar de grande esforço em sua adaptação por parte do projetista, decidimos modelar cada uma das funcionalidades típicas separadamente, ligando-as ao modelo de funcionalidades através de transições de substituição. Acreditamos que com isso a adaptação ocorrerá de modo mais simples, uma vez que o projetista se preocupará com apenas um aspecto de funcionalidade por vez. Na Figura 22 temos a subpágina do modelo com relação às funcionalidades do sistema. E, nas subseções seguintes detalharemos cada uma dessas funcionalidades.

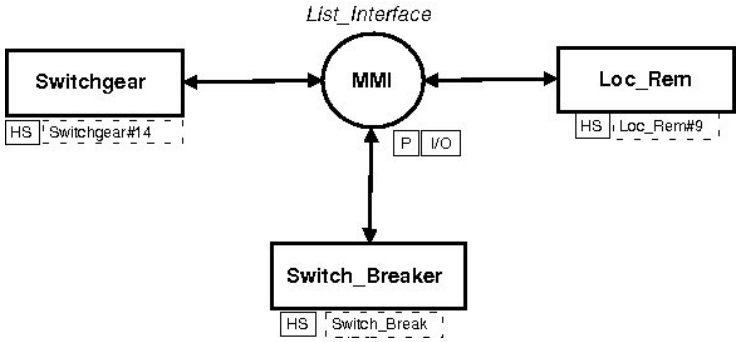


Figura 22: Modelo das Funcionalidades

Conforme já mencionado, neste trabalho modelamos apenas os disjuntores, as chaves seccionadoras e as chaves de comando local/telecomando.

4.2.5.1 – Chaves Local/Telecomando (Submodelo Loc_Rem)

O elemento de interação do tipo chave local/telecomando é utilizado pelo operador para definir o tipo de controle que uma determinada linha tensão vai ter, que pode ser local (interação realizada pelo operador do sistema) ou telecomandada (o operador não pode interagir com os elementos da linha tensão).

Quem define o estado das linhas de tensão é o operador, sendo assim a operação das chaves local/telecomando está sempre disponível. O modelo desse tipo de elemento de interação é apresentado na Figura 23.

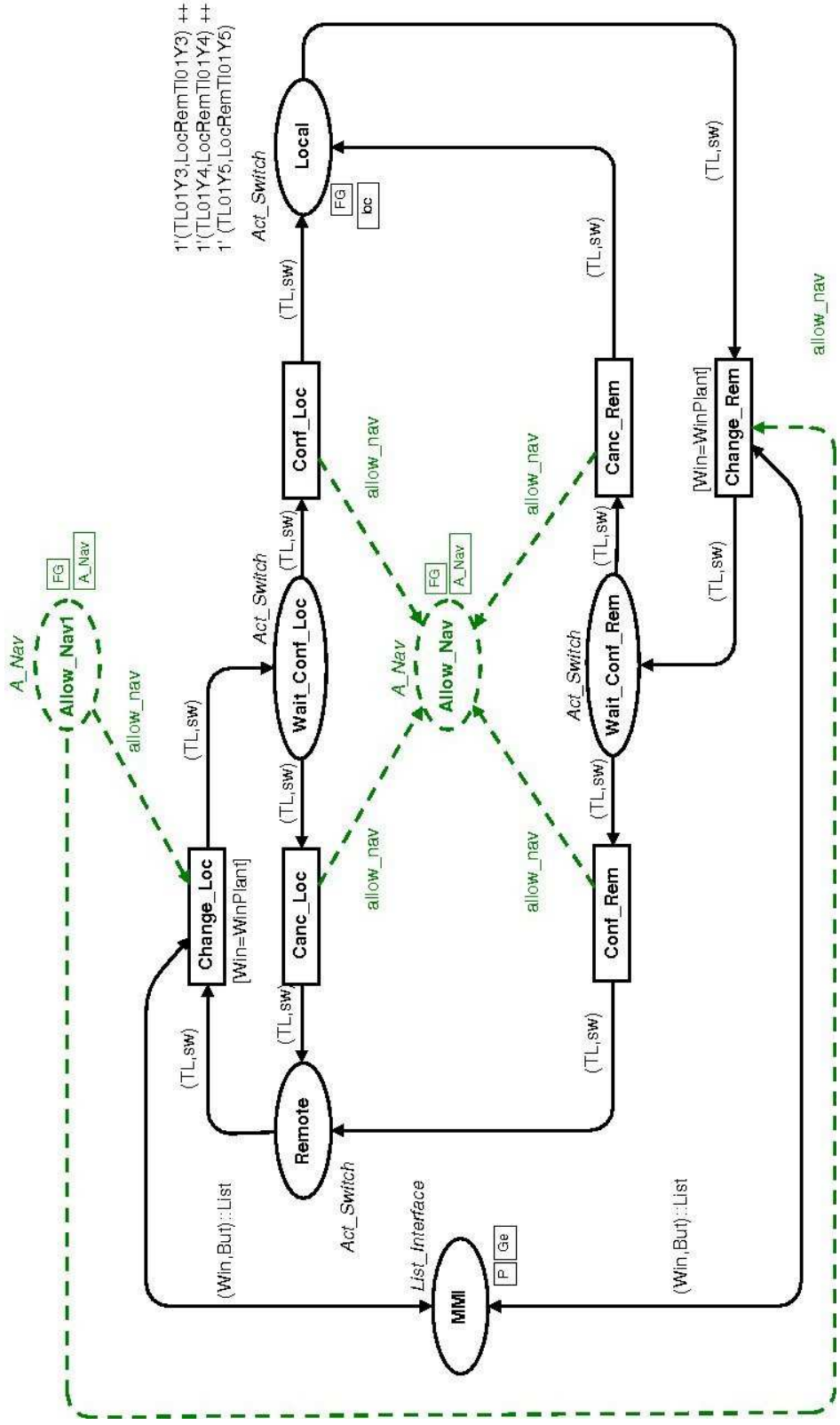


Figura 23: Modelo da Chave Local/Telecomando

O estado das chaves local/telecomando é definido de acordo com o lugar no qual a ficha que representa a chave está: *Local* indicando que o controle da linha de tensão é local, e *Remote* indicando que o controle é remoto. Cada ficha que representa uma chave de tensão é uma dupla contendo a linha de tensão a qual a chave se refere, e a própria chave.

A mudança do controle de local para remoto começa com o disparo da transição *Change_Rem*. A ocorrência dessa transição está condicionada à janela ativa ser a janela do sinótico do sistema (guarda da transição) e ao ocorrer qualquer operação de navegação fica impedida (retirada da ficha do lugar *Allow_Nav*). Isso se dá devido ao fato de que uma vez que se selecionou a chave deve-se finalizar a operação antes de se iniciar qualquer outra atividade.

Uma ficha no lugar *Wait_Conf_Rem* indica que uma dada chave local/telecomando foi selecionada para mudança para controle remoto mas o operador ainda não confirmou a ação. Nesse ponto o operador poder confirmar a ação (disparo da transição *Conf_Rem*) ou cancelar (disparo da transição *Canc_Rem*). Cancelando a ficha volta para o lugar *Local* e confirmando vai para o lugar *Remote* indicando que essa linha de tensão está com o comando remoto.

Para a mudança do controle de controle remoto para local, o procedimento é o mesmo descrito acima só que agoras a transição que seleciona a chave é *Change_Loc*, a que cancela a ação é *Canc_Loc* e a que confirma a ação é *Conf_Loc*.

O lugar *Local* é um lugar de fusão que está presente nos submodelos das outras funcionalidades para a restrição da operação das mesmas. A funcionalidade de uma determinada linha de tensão só está disponível para interação se a ficha que descreve a chave local/telecomando dessa linha estiver no lugar *Local*, indicando que o controle dessa linha está no modo local.

Neste modelo temos as chaves local/telecomando de três linhas de tensão, a saber: a chave *LocRemTl01Y3* referente a linha de tensão *TL01Y4*, a chave *LocRemTl01Y4* referente a linha de tensão *TL01Y4*, e a chave *LocRemTl01Y4* referente a linha de tensão *TL01Y4*. Apenas exemplificando, *LocRemTl01Y3* significa a chave local/telecomando (*LocRem*) da linha de tensão *TL01Y3*.

4.2.5.2 – Disjuntores (Submodelo Switch_Breaker)

Os disjuntores são elementos de proteção do sistema a partir dos quais o operador pode atuar para ativar/desativar o fornecimento de energia. Os disjuntores possuem dois estados:

aberto e fechado. O operador do sistema pode selecionar um disjuntor para ser atuado (aberto ou fechado). Uma ver selecionado é pedida uma confirmação da ação, podendo a mesma ser confirmada ou cancelada. O modelo dos disjuntores é apresentado na Figura 24.

Como pode ser observado, o modelo dos disjuntores é muito parecido com o modelo das chaves local/telecomando, com a inclusão do lugar *Local* que indica quais linhas de tensão estão no modo de comando local (conforme já explicado) indicando quais disjuntores podem ser operados.

A abertura de um disjuntor envolve as transições *Open_CB*, *Canc_Open* e *Conf_Open*, da mesma forma que no caso do modelo das chaves local/telecomando. O fechamento envolve as transições *Close_CB*, *Canc_Close* e *Conf_Close*.

O lugar *CB_Open* contém fichas que indicam quais disjuntores de quais linhas de tensão estão no estado aberto (a ficha é uma dupla indicando a linha de tensão e o disjuntor da mesma). E o lugar *CB_Close* contém fichas que indicam quais disjuntores de quais linhas de tensão estão no estado fechado. Neste modelo inicialmente os três disjuntores (*SB21Y3*, *SB21Y4* e *SB21Y5*) disponíveis estão no estado fechado.

O lugar *CB_Open* é um lugar de fusão que é utilizado no intertravamento das seccionadoras, só permitindo que uma dada seccionadora seja aberta se o disjuntor correspondente da sua linha de tensão também esteja. O modelo das seccionadoras é apresentado na subseção a seguir.

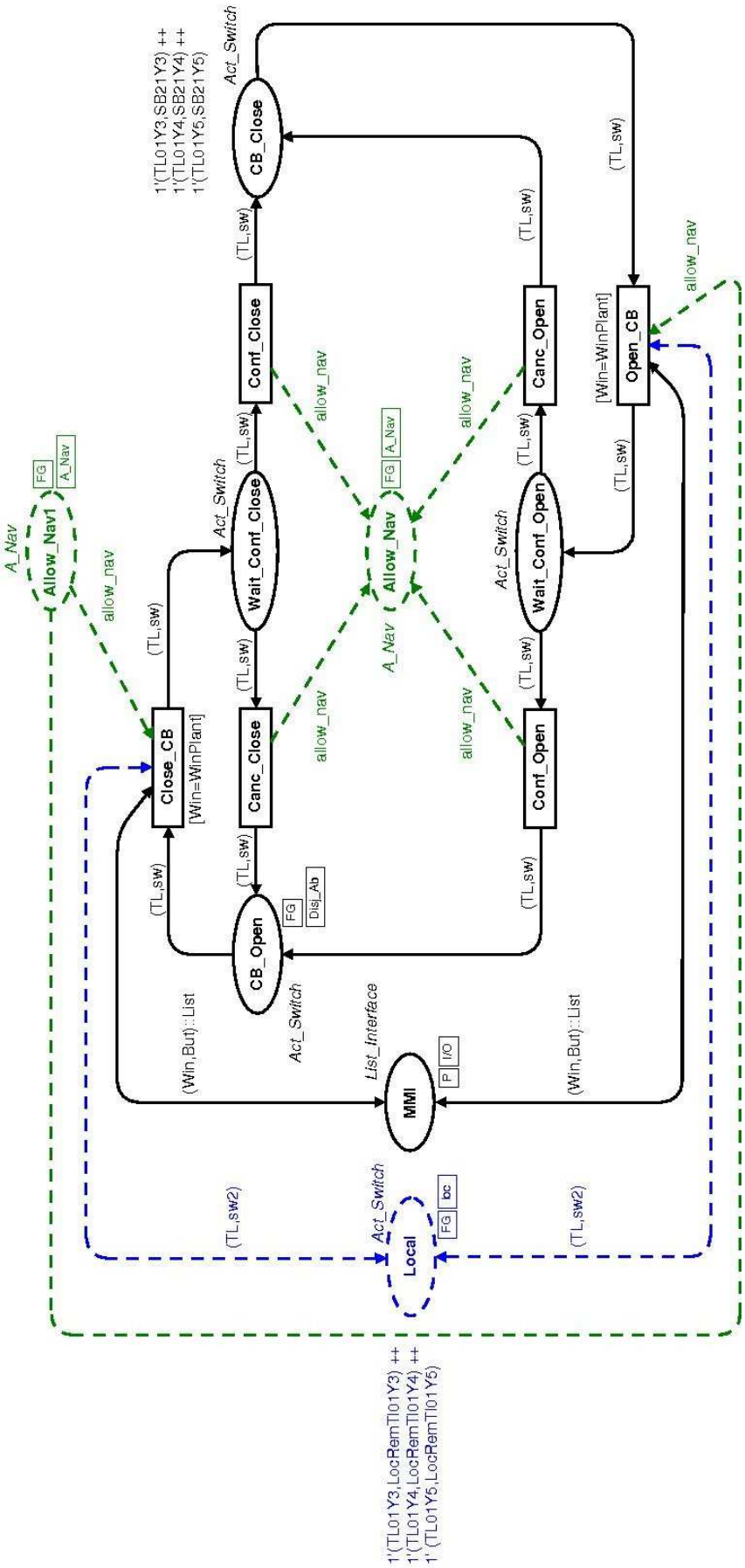


Figura 24: Modelo dos Disjuntores

4.2.5.3 – Chaves Seccionadoras (Submodelo Switchgear)

As seccionadoras são elementos de proteção do sistema que trabalham junto com os disjuntores, a partir dos quais o operador pode atuar para ativar/desativar o fornecimento de energia. Assim como os disjuntores, as seccionadoras possuem dois estados: aberto e fechado. O operador do sistema pode selecionar uma seccionadora para ser atuada (aberta ou fechada), sendo que se for para abrir, se faz necessário que o disjuntor associado a ela também esteja aberto, não sendo possível caso contrário. Uma ver selecionada é pedida uma confirmação da ação, podendo a mesma ser confirmada ou cancelada.

O modelo das seccionadoras é apresentado na Figura 25. Como pode ser observado, o modelo das seccionadoras é muito parecido com o modelo dos disjuntores, com a inclusão do lugar de fusão *SB_Open* que indica quais disjuntores estão aberto, permitindo que as seccionadoras relacionadas a esses disjuntores (mesma linha de tensão) sejam abertas (cada linha de tensão possui duas seccionadoras).

A abertura de uma seccionadora envolve as transições *Open_SG*, *Canc_Open* e *Conf_Open*, da mesma forma que no caso do modelo do disjuntor, mas com a restrição do disjuntor da mesma linha da seccionadora estar aberto (ficha que representa esse disjuntor presente no lugar *CB_Open*). O fechamento envolve as transições *Close_SG*, *Canc_Close* e *Conf_Close*.

O lugar *SG_Open* contém fichas que indicam quais seccionadoras de quais linhas de tensão estão no estado aberto (a ficha é uma dupla indicando a linha de tensão e a seccionadora da mesma). E o lugar *SG_Close* contém fichas que indicam quais seccionadoras de quais linhas de tensão estão no estado fechado. Inicialmente as seis chaves seccionadoras desse modelo (*SG31Y34* e *SG31Y35* da *TL01Y3*, *SG31Y44* e *SG31Y45* da *TL01Y4*, e *SG31Y54* e *SG31Y55* da *TL01Y5*) estão no estado fechado.

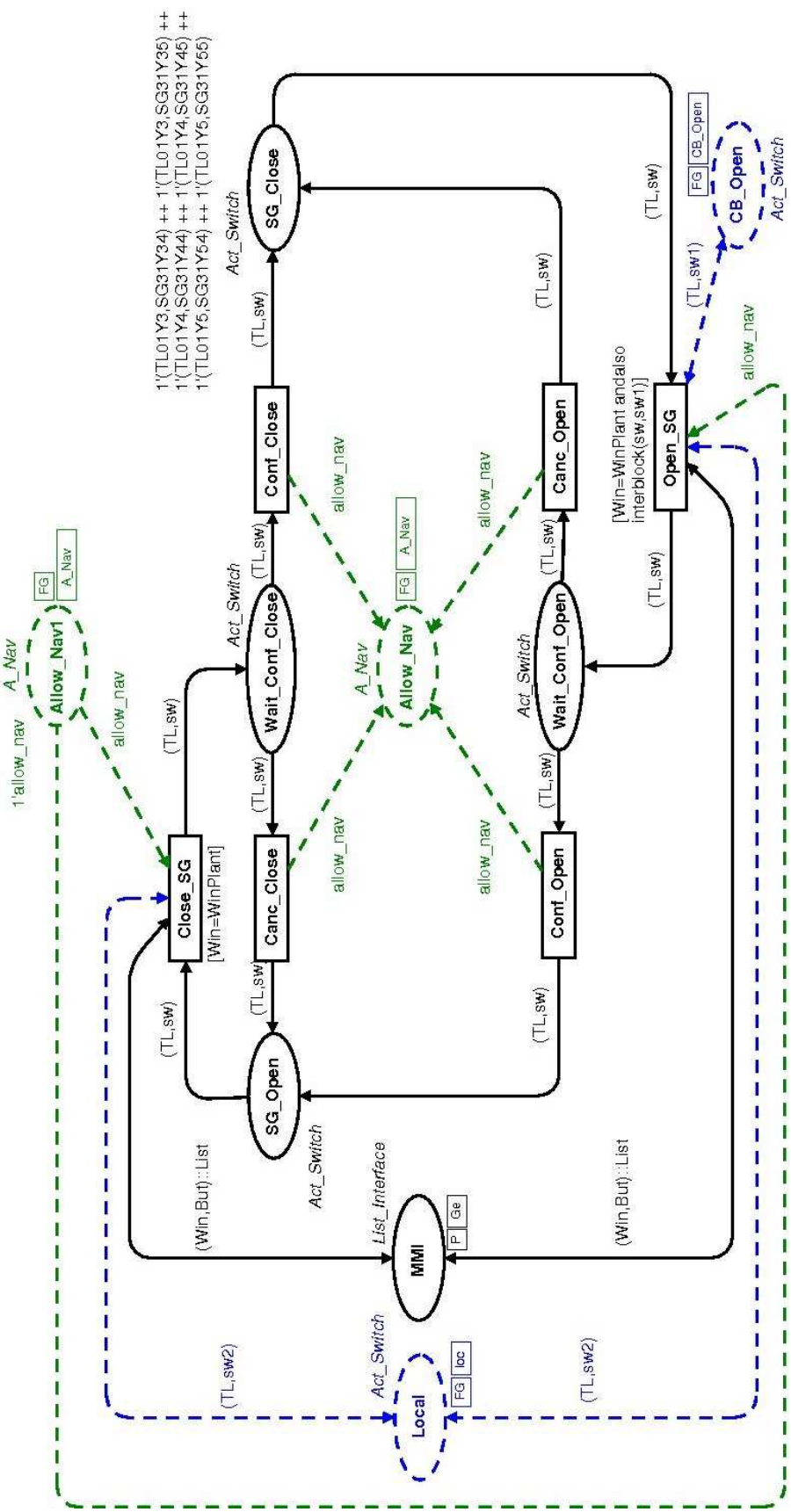


Figura 25: Modelo das Seccionadoras

Capítulo 5 – Análises no Modelo

Neste capítulo serão apresentadas as diversas análises realizadas no modelo e como as mesmas podem ajudar a melhorar a qualidade das interfaces e a detectar e propor soluções para questões relacionadas ao erro humano na interação. Em seguida apresentaremos as análises propriamente ditas, com uma visão crítica sobre os resultados obtidos.

5.1 Simulações Interativas e Automáticas

No decorrer do desenvolvimento do arcabouço as simulações interativas foram utilizadas para verificar a corretude das partes do sistema. Como o modelo é hierárquico e modular, ao finalizar a modelagem de cada uma dessas partes simulações foram realizadas de modo a testar a evolução das mesmas e verificar se se comportam de acordo com o esperado. As partes testadas isoladamente foram referentes à navegação e às funcionalidades do sistema.

O uso de simulações interativas em partes do sistema facilitou a análise da evolução dessas partes. Para realizar essa análise as marcações iniciais dos lugares do sistema foram alteradas para que o sistema evoluísse apenas na parte que se desejava analisar.

Em seguida foram executadas simulações tanto interativas quanto automáticas do sistema como um todo para analisar sua evolução, observando os possíveis tipos de navegação e funcionalidades de forma integrada.

Aqui, o uso das simulações não teve como foco a análise das questões de usabilidade e do erro humano na operação. A função dessa etapa de simulação foi apenas de verificar o comportamento das partes do modelo, e do modelo como um todo, para então realizar as análises.

5.2 Geração de MSCs

Conforme já mencionado, a visualização do comportamento do sistema através do acompanhamento da evolução do modelo durante uma simulação é uma tarefa que requer uma certa familiaridade com as CPNs. E, um modo de facilitar essa visualização é a utilização de gráficos de seqüência de mensagens (MSCs).

Como forma de mostrar as facilidades do uso dos MSCs, dois exemplos gerados a partir do estudo de caso são apresentados na Figura 26 e Figura 27, a seguir.

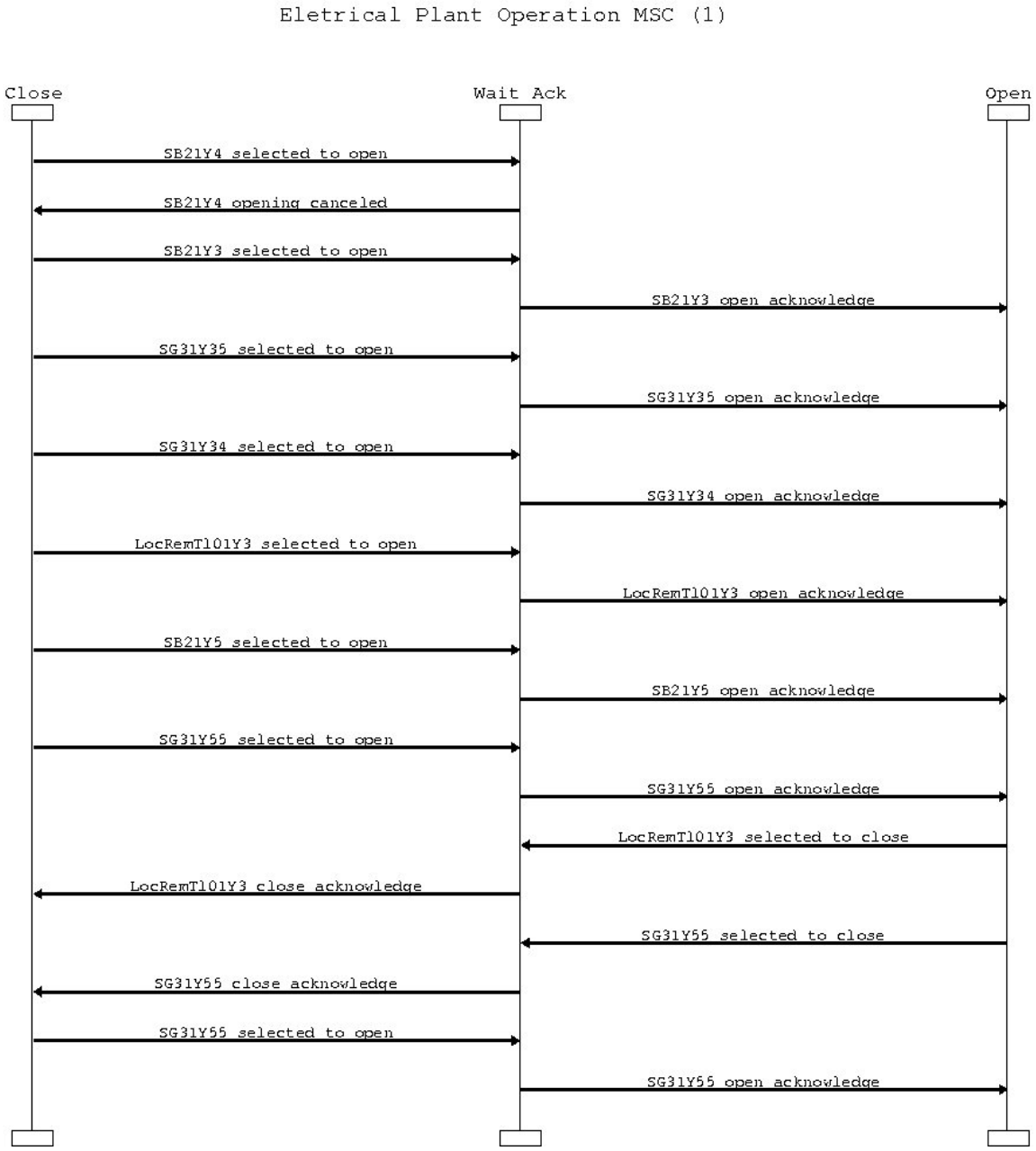


Figura 26: MSC da Operação do Sistema (Parte 1)

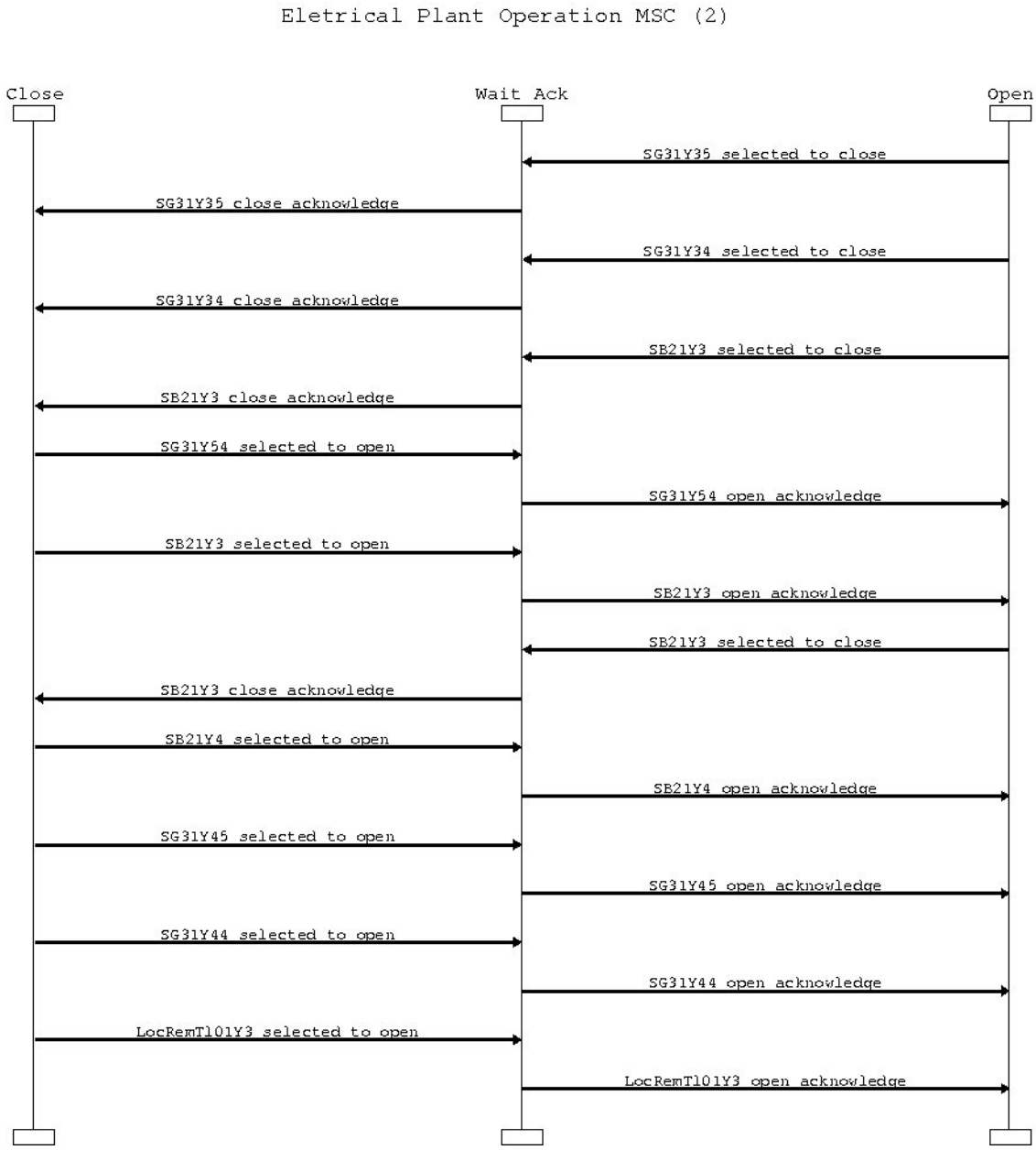


Figura 27: MSC da Operação do Sistema (Parte 2)

Os MSCs possuem três eixos, indicando respectivamente o estado fechado (ou comando local no caso das chaves local/telecomando), em espera de confirmação e aberto (ou comando remoto no caso das chaves local/telecomando) dos dispositivos. Salientamos o fato dos MSCs contemplarem neste exemplo apenas as interações do operador do sistema com os dispositivos presentes na planta industrial (disjuntores, seccionadoras e chaves local/telecomando), não havendo navegação entre janelas.

Analisando os MSCs podemos observar diversas etapas de interação ocorridas. Inicialmente o disjuntor *SB21Y4* é selecionado para abertura, mas essa ação é cancelada logo em seguida. Isso pode ser interpretado como o início de uma ação incorreta por parte do operador, que ao perceber o erro cancelou a ação.

Temos então a seleção e confirmação de abertura do disjuntor *SB21Y3* e em seguida ocorre a seleção e confirmação de abertura das duas chaves seccionadoras dessa mesma linha (*SG31Y35* e *SG31Y34*). Toda essa ação de abertura dessas três chaves configura a abertura, sem a ocorrência de erros de operação, da linha *Tl01Y3*. E, temos a mudança (seleção e confirmação) da chave local/telecomando *LocRemTl01Y3* para o estado telecomando.

Temos então a seleção e confirmação de abertura do disjuntor *SB21Y5* e isso configura que o operador realizará a ação de abertura dessa linha. Ele em seguida procede com a abertura da seccionadora *SG31Y55*. Mas em seguida, ao invés de abrir a seccionadora *SG31Y34* o que o operador faz é retornar o estado da chave local/telecomando *LocRemTl01Y3* para o estado de comando local. Isso configura um erro de operação, pois o correto seria terminar de abrir a linha de tensão *Tl01Y5* para então realizar essa última operação. Em seguida temos o fechamento e abertura da seccionadora *SG31Y55*, mostrando que o fato do operador ter inserido uma ação não prevista na tarefa de abertura da linha de tensão *Tl01Y5* o fez cometer erro de operação. Conforme já mencionado o ideal é realizar cada tarefa completamente antes de iniciar uma nova. Deve-se lembrar que esses MSCs são baseados em relatos de falhas ocorridos na CHESF.

Com essa tarefa incompleta o operador passa então ao fechamento da linha *Tl01Y3*, com o fechamento das duas seccionadoras (*SG31Y35* e *SG31Y34*) e do disjuntor da linha (*SB21Y3*).

Nesse instante é que o operador percebeu (ou foi alertado) que a tarefa de abertura da linha *Tl01Y5* estava incompleta, abrindo então a seccionadora da linha que ainda estava fechada (*SG31Y54*).

Ocorre então a abertura e posterior fechamento do disjuntor *SB21Y3*, que foi aberto devido a um erro. O correto, e que é realizado em seguida, é a abertura da linha de tensão *Tl01Y4*, com a abertura do disjuntor (*SB21Y4*) e das duas seccionadoras (*SG31Y45* e *SG31Y44*) da linha.

Por fim temos a mudança de estado da chave local/telecomando *LocRemTl01Y3* para o estado de comando remoto.

Podemos então observar que o uso de MSCs ajuda na análise de cenários de execução do modelo, podendo ser utilizados para inferir se existem problemas ou não no sistema, sem a necessidade de analisar toda a mudança de estados direto no modelo através da verificação de que transição ocorreu e quais lugares tiveram suas fichas alteradas.

Porém o uso de MSCs, por ser baseado na simulação do modelo, não é um método exaustivo. É bastante complicado, principalmente para modelos com muitos estados possíveis, garantir que todas as possibilidades de interação foram verificadas. Devido a isso é que utilizamos a ferramenta de geração e análise do espaço de estados do modelo, como mostramos na seção a seguir.

5.3 Geração do Occ

O Grafo de Ocorrência (Occ) do modelo foi gerado para permitir a verificação das propriedades lógicas do sistema. Inicialmente foi gerado o relatório padrão da ferramenta para análise. Esse relatório é apresentado a seguir.

```
Statistics
------------------------------------------------------------------------

  Occurrence Graph
    Nodes:  89089
    Arcs:   238595
    Secs:   68
    Status: Full

  Scc Graph
    Nodes:  8194
    Arcs:   8194
    Secs:   616

 Boundedness Properties
------------------------------------------------------------------------
  Best Integers Bounds                  Upper      Lower
  End'Allow_Nav 1                       1           0
  End'Allow_Nav 2                       1           0
  Industrial_MMI'End 1                  1           1
  Industrial_MMI'Login 1                1           0
  Industrial_MMI'MMI 1                  1           0
  Loc_Rem'Allow_Nav 1                   1           0
  Loc_Rem'Allow_Nav1 1                  1           0
  Loc_Rem'Local 1                       3           0
  Loc_Rem'Remote 1                      3           0
  Loc_Rem'Wait_Conf_Loc 1               1           0
  Loc_Rem'Wait_Conf_Rem 1               1           0
  Login'Allow_Nav 1                     1           0
  Navigation'Allow_Nav 1                1           0
  Navigation'But_Close 1                1           1
```

```
  Navigation'Options1 1                6           6
  Navigation'Options2 1                1           1
  Navigation'Win_Nav1 1                6           6
  Navigation'Win_Nav2 1                1           1
  Switch_Breaker'Allow_Nav 1           1           0
  Switch_Breaker'Allow_Nav1 1          1           0
  Switch_Breaker'CB_Close 1            3           0
  Switch_Breaker'CB_Open 1             3           0
  Switch_Breaker'Local 1               3           0
  Switch_Breaker'Wait_Conf_Close 1     1           0
  Switch_Breaker'Wait_Conf_Open 1      1           0
  Switchgear'Allow_Nav 1               1           0
  Switchgear'Allow_Nav1 1              1           0
  Switchgear'CB_Open 1                 3           0
  Switchgear'Local 1                   3           0
  Switchgear'SG_Close 1                6           0
  Switchgear'SG_Open 1                 6           0
  Switchgear'Wait_Conf_Close 1         1           0
  Switchgear'Wait_Conf_Open 1          1           0

 Home Properties
------------------------------------------------------------------------

  Home Markings:  None

 Liveness Properties
------------------------------------------------------------------------

  Dead Markings:  8192 [9996,9995,9989,9988,9981,...]
  Dead Transitions Instances: None
  Live Transitions Instances: None
```

Analisando o relatório padrão é possível tirar algumas conclusões sobre o modelo do sistema. Como pode ser observado, a geração do grafo de ocorrência foi completa, o que significa que podemos fazer verificações em todo o espaço de estados do modelo.

Inicialmente são apresentados os valores máximo e mínimo de fichas em cada lugar do modelo. Analisando esses dados já obtemos um indicador se existe algum erro no sistema. Pode existir um erro se em algum lugar do modelo existirem mais ou menos fichas do que o esperado, sendo necessário a análise para determinar se o erro é decorrente do comportamento do sistema ou do processo de modelagem. Se o erro for do comportamento do sistema deve-se verificar como modificar o sistema para eliminar o erro. Para esse estudo, todos os lugares estão com seus limites máximo e mínimo dentro dos valores esperados.

Em seguida o relatório apresenta as marcações recorrentes, que são os estados do modelos sempre alcançáveis a partir de qualquer outro estado. Observamos que o relatório apresenta que não existem marcações recorrentes no sistema, embora fosse esperado que na interface todos os estados fossem recorrentes. Analisando essa propriedade observamos que não existem estados recorrentes devido à existência de diversos estados terminais, que são os estados de encerramento do aplicativo. Ao se recalcular o espaço de estados do modelo retirando do modelo a parte referente ao encerramento do aplicativo observamos que todos os estados são recorrentes e não existem estados de bloqueio no modelo. Isso também é esperado pois na interface do estudo de caso o operador do sistema sempre possui algum caminho que permita que ele chegue a qualquer estado desejado do sistema. E também não pode haver nenhum estado de bloqueio do sistema (a exceção dos estados ligados ao encerramento do aplicativo), pois isso indicaria a presença de um ou mais estados nos quais o operador não poderia mais interagir com o sistema, não podendo mais exercer as funções de comando e proteção.

Analisando os estados de bloqueio do sistema, a partir do relatório completo, verificamos que existem muitos estados terminais. Embora uma das propriedades de usabilidade que queríamos verificar indicava que o acesso à saída deveria estar sempre disponível, devemos pensar melhor nessa propriedade nesta classe de interfaces. Propomos que o acesso à saída nesse sistema só seja disponibilizado ao operador após a sua devida desautenticação. Propomos também que exista alguma forma do supervisor do sistema interromper uma sessão do operador em caso de falha do aplicativo, tendo ele o acesso à saída a partir de qualquer ponto para uso em uma eventual emergência. Não recomendamos essa característica aos operadores de modo a tornar mais seguro o uso do aplicativo, eliminando possíveis problemas de encerramento do aplicativo por um erro ou descuido do operador.

5.3.1 – Verificação das Propriedades de Usabilidade

Nesta subseção apresentaremos o resultado da verificação das propriedades de usabilidade a partir da verificação das propriedades do espaço de estados do modelo. Elencaremos a seguir as propriedades e como ela foram, ou não, verificadas.

- *Reinicialização* – Essa propriedade é verificada analisando as marcações recorrentes do modelo (*home markings*). Se a marcação inicial do modelo fizer parte das marcações recorrentes então a reinicialização é possível. Como

mostramos que se tirarmos a parte do modelo referente ao encerramento do aplicativo todas as marcações do sistema são recorrentes, e atendo para o fato de que em operação normal o aplicativo não deve ser encerrado, concluímos que essa propriedade é atendida.

- *Reversibilidade* – Essa propriedade é passível de verificação utilizando a função *Reachable* presente na ferramenta Design/CPN. Deve-se definir qual o caminho direto e verificar se existe um caminho partindo do nó final até o nó inicial do caminho direto. Aqui cabe uma ressalva, nessa classe de interface não basta apenas que exista a possibilidade de reversibilidade, mas também devemos verificar quais as possibilidades disso ocorrer, de modo a verificar quais os casos corretos e quais os que implicam na ocorrência de erro de interação. Para se verificar esse fato devemos nos valer das funções auxiliares desenvolvidas para a análise de caminhos alternativos, que serão apresentadas na próxima seção. Mas, analisando apenas o fato de existir reversibilidade, como no modelo todos os estados são recorrentes (retirando-se a parte do modelo que trata do encerramento do aplicativo), podemos afirmar que existe reversibilidade entre todos os estados do sistema.
- *Existência de caminhos de acesso entre pontos específicos da interação* – Para estados específicos da interface utiliza-se a função *Reachable* para garantir essa propriedade. Mas para verificar se existe mais de um caminho entre estados específicos e para analisar quais os mais indicados, deve-se utilizar as funções auxiliares desenvolvidas para esse propósito. Mas, como podemos afirmar que todos os estados do modelo são recorrentes, podemos afirmar que existem caminhos entre todos os pontos da interface, atendendo essa propriedade para qualquer que sejam os estados escolhidos.
- *Acesso à Saída* – Conforme já mencionado, embora essa seja uma propriedade esperada para todos os estados do sistema, não consideramos aplicável a essa classe de sistemas. Na seção anterior já discorremos acerca dessa propriedade e quais as recomendações que propomos.

5.4 Análise de Situações de Erro Utilizando as Funções Auxiliares

Esta seção apresenta o uso das funções auxiliares desenvolvidas para a análise de caminhos alternativos entre dois estados da interface. O uso dessas funções pode ser para a determinação e análise das alternativas de interação disponíveis como também para a verificação da existência da possibilidade de erros na interação do operador com a interface do sistema. A seguir apresentaremos alguns exemplos do uso das funções.

5.4.1 – Verificação 1: Abertura de uma Linha de Tensão no Menor Número de Passos Possível

5.4.1.1 – Passo 1 – Encontrar os estados inicias e finais da interação

Neste caso vamos testar a linha de tensão TL01Y3, podendo ter sido escolhida qualquer outra linha de tensão presente no modelo.

Para essa etapa se faz necessário utilizar a função *def_est* com todos seus parâmetros, definindo os estados do disjuntor e das seccionadoras da linha TL01Y3 primeiro no estado fechado e em seguida no estado aberto. Foram definidos todos os parâmetros para reduzir o número de estados de retorno, dado que para a análise de uma dada linha pode ser desconsiderado o estado das outras linhas, e com isso foi definido que as outras linhas estariam com seus elementos no estado fechado. O retorno da função é apresentado a seguir:

```
Linha TL01Y3 fechada - 1
Linha TL01Y3 aberta - 267
```

5.4.1.2 – Passo 2 – Encontrar os caminhos entre esse dois estados

Nesta etapa utilizamos a função *AllPath* para obtermos todos os caminhos possíveis entre os dois estados em estudo. Sabendo que no melhor caso essa interação requer 7 estados (o estado inicial mais dois estados para a interação com cada elemento), definimos esse como o tamanho máximo dos caminhos a serem retornados. Essa escolha se deu devido a primeiro desejarmos verificar as interações possíveis utilizando o menor número de passos possíveis. E, definimos que seriam realizadas 5000000 iterações da função como forma de limitar o tempo de processamento da mesma. O comando de chamada da função e a resposta obtida foram:

```
AllPath(1,267,7,5000000);
```

```
List:
[ 1, 4, 10, 31, 67, 183, 267 ]
[ 1, 4, 10, 30, 66, 175, 267 ]

List Size: 2

AllPath finished - Iterations: 1944

val it = () : unit
```

O retorno da busca teve como resultado dois caminhos possíveis e foi completo para o tamanho de caminho especificado (sete passos). Esses são os caminhos de interação corretos na abertura da linha de tensão. Mas, se temos três elementos de interação (um disjuntor e duas seccionadoras), por que existem apenas dois caminhos possíveis? Isso se deve ao fato de existir a restrição de que as seccionadoras só podem ser abertas se o disjuntor correspondente já estiver aberto. A verificação de que esses caminhos realmente significam isso é apresentado no próximo passo.

5.4.1.3 – Passo 3 – Verificar o significado de cada caminho encontrado

Nesta etapa o objetivo é verificar o que significa cada caminho encontrado de modo a compreender quais os passos de interação que levaram o modelo (a interface) do estado inicial ao estado final definido (neste caso a abertura da linha de tensão).

Para esta tarefa não ser tediosa e propensa a erros com a comparação entre os estado sendo feita pela análise de cada estado do modelo e sua comparação com o próximo estado a fim de descobrir quais as diferenças entre eles, utilizamos a função *cnodes*, cuja função é fazer a comparação entre dois estados retornando apenas os lugares em que as marcações diferem. Essa função tem que ser executada para cada dois estados do caminho em análise devido ao fato da função ainda não contemplar a comparação de uma seqüência de estados. A seguir é apresentada a chamada da função e seu retorno para o primeiro caminho encontrado no passo 2.

```
cnodes(1,4);
cnodes(4,10);
cnodes(10,31);
cnodes(31,67);
cnodes(67,183);
cnodes(183,267);

Difference between node 1 and node 4 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
```

```
Switch_Breaker'CB_Close 1: 1`(TL01Y3,SB21Y3)++ 1`(TL01Y4,SB21Y4)++
       1`(TL01Y5,SB21Y5) --- Switch_Breaker'CB_Close 1: 1`(TL01Y4,SB21Y4)++
       1`(TL01Y5,SB21Y5)
Switch_Breaker'Wait_Conf_Open 1: empty --- Switch_Breaker'Wait_Conf_Open 1:
       1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 4 and node 10 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switch_Breaker'CB_Open 1: empty --- Switch_Breaker'CB_Open 1: 1`(TL01Y3,SB21Y3)
Switch_Breaker'Wait_Conf_Open 1: 1`(TL01Y3,SB21Y3) ---
       Switch_Breaker'Wait_Conf_Open 1: empty
Switchgear'CB_Open 1: empty --- Switchgear'CB_Open 1: 1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 10 and node 31 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)++
       1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
       1`(TL01Y5,SG31Y55) --- Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y35)++
       1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
       1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open 1: empty --- Switchgear'Wait_Conf_Open 1:
       1`(TL01Y3,SG31Y34)
val it = () : unit

Difference between node 31 and node 67 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: empty --- Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y34)
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y34) --- Switchgear'Wait_Conf_Open 1:
       empty
val it = () : unit

Difference between node 67 and node 183 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y35)++ 1`(TL01Y4,SG31Y44)++
       1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55) ---
       Switchgear'SG_Close 1: 1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++
       1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open 1: empty --- Switchgear'Wait_Conf_Open 1:
       1`(TL01Y3,SG31Y35)
val it = () : unit

Difference between node 183 and node 267 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y34) --- Switchgear'SG_Open 1:
```

```
      1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y35) --- Switchgear'Wait_Conf_Open 1:
      empty
val it = () : unit
```

Como pode ser observado, foi feita a comparação entre cada dois pares de estados adjacentes de modo a verificar as diferenças entre os mesmos. Na resposta para este caso também é apresentada a mudança do lugar *Allow_Nav*, presente em todas as páginas. Esse lugar modela a restrição à navegação ou ao início de uma nova ação sem o término da ação atual (abertura ou fechamento de um dispositivo). Como esse lugar é um lugar de fusão presente em todas as páginas do modelo, na resposta acima apresentadas deixamos apenas uma instância desse lugar.

Analisando esses resultados observamos que na primeira comparação (nós 1 e 4) o disjuntor SB21Y3 é selecionado para abertura e vai para o estado em que aguarda confirmação ou cancelamento (ficha 1`(TL01Y3,SB21Y3) que estava no lugar *CB_Close* e que está agora no lugar *Wait_Conf_Open* da página *Switch_Breaker*). Na comparação seguinte a abertura é confirmada (ficha 1`(TL01Y3,SB21Y3) que estava no lugar *Wait_Conf_Open* e que está agora no lugar *CB_Open* da página *Switch_Breaker*). Neste caso temos também a criação de uma ficha no lugar *CB_Open* da página *Switchgear*, que é usada para verificar o intertravamento das seccionadoras.

As duas comparações seguintes (nós 10 e 31, e nós 31 e 67) tratam da abertura da seccionadora SG31Y34. E as comparações entre os nós 67 e 183, e os nós 183 e 267 dizem respeito a abertura da seccionadora SG31Y35, finalizando assim a ação de abertura da linha de tensão.

Temos a seguir a chamada da função e seu retorno para o segundo caminho encontrado no passo 2. A diferença aqui em relação ao outro caminho é apenas a seqüência em que as seccionadoras são abertas, que nesse caso é primeiro a SG31Y35 e em seguida a SG31Y34.

```
cnodes(1,4);
cnodes(4,10);
cnodes(10,30);
cnodes(30,66);
cnodes(66,175);
cnodes(175,267);

Difference between node 1 and node 4 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switch_Breaker'CB_Close 1: 1`(TL01Y3,SB21Y3)++ 1`(TL01Y4,SB21Y4)++
```

```
      1`(TL01Y5,SB21Y5) --- Switch_Breaker'CB_Close 1: 1`(TL01Y4,SB21Y4)++
      1`(TL01Y5,SB21Y5)
Switch_Breaker'Wait_Conf_Open 1: empty --- Switch_Breaker'Wait_Conf_Open 1:
      1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 4 and node 10 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switch_Breaker'CB_Open 1: empty --- Switch_Breaker'CB_Open 1: 1`(TL01Y3,SB21Y3)
Switch_Breaker'Wait_Conf_Open 1: 1`(TL01Y3,SB21Y3) ---
      Switch_Breaker'Wait_Conf_Open 1: empty
Switchgear'CB_Open 1: empty --- Switchgear'CB_Open 1: 1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 10 and node 30 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)++
      1`(TL01Y4,SG31Y44)++       1`(TL01Y4,SG31Y45)++       1`(TL01Y5,SG31Y54)++
      1`(TL01Y5,SG31Y55)   ---   Switchgear'SG_Close   1:   1`(TL01Y3,SG31Y34)++
      1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
      1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open 1: empty --- Switchgear'Wait_Conf_Open 1:
      1`(TL01Y3,SG31Y35)
val it = () : unit

Difference between node 30 and node 66 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: empty --- Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y35)
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y35) --- Switchgear'Wait_Conf_Open 1:
      empty
val it = () : unit

Difference between node 66 and node 175 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y4,SG31Y44)++
      1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55) ---
       Switchgear'SG_Close 1: 1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++
      1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open   1:   empty   ---   Switchgear'Wait_Conf_Open   1:
1`(TL01Y3,SG31Y34)
val it = () : unit

Difference between node 175 and node 267 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y35) --- Switchgear'SG_Open 1:
      1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)
```

```
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y34) --- Switchgear'Wait_Conf_Open 1:
      empty
val it = () : unit
```

Analisando as mudanças de estados nesses caminhos podemos verificar que o operador, realizando a operação de abertura de uma linha de tensão no menor número de passos, só tem liberdade de escolher qual das duas seccionadoras da linha de tensão ele ira atuar primeiro, já que existe a restrição de que as seccionadoras só podem ser abertas se o disjuntor correspondente já estiver aberto. Então, o operador tem que abrir o disjuntor da linha e depois escolher a seqüência de abertura das seccionadoras.

Essa é uma situação de operação normal e não contempla nenhum erro, mostrando que na operação de abertura de uma linha não é possível se executar erros de operação se a tarefa for realizada no menor número de passos possíveis. Porém isso não é verdade para o caso de fechamento de uma linha, como veremos mais adiante nesse trabalho.

5.4.2 – Verificação 2: Abertura de uma Linha de Tensão com Incidência de Erro Humano

Neste teste buscamos o mesmo resultado anterior, porém com mais passos de interação o que permite verificar erros de interação. Ao aumentarmos o número de passos podemos verificar se existem caminhos em que operador pode cometer algum erro de operação.

Como o estado inicial e final são bem definidos (apenas a linha TL01Y3 aberta) temos como resposta casos em que alguma ação incorreta é realizada (no contexto dessa operação) e revertida, tendo como resultado final apenas a linha escolhida aberta.

5.4.2.1 – Passo 1 – Encontrar os estados inicias e finais da interação

Esse caso é o mesmo do Teste 1, cujos resultados são:

```
Linha TL01Y3 fechada - 1
Linha TL01Y3 aberta - 267
```

5.4.2.2 – Passo 2 – Encontrar os caminhos entre esse dois estados

Neste caso aumentamos o tamanho máximo dos caminhos possíveis para 11, e obtivemos como resposta:

```
List:
[ 1, 6, 12, 44, 69, 201, 10, 31, 67, 183, 267 ]
[ 1, 6, 12, 44, 69, 201, 10, 30, 66, 175, 267 ]
```

[1, 6, 12, 44, 69, 198, 274, 681, 67, 183, 267]
[1, 6, 12, 44, 69, 198, 274, 678, 826, 1755, 267]
[1, 6, 12, 44, 69, 197, 270, 655, 66, 175, 267]
[1, 6, 12, 44, 69, 197, 270, 652, 826, 1755, 267]
[1, 5, 11, 39, 68, 194, 10, 31, 67, 183, 267]
[1, 5, 11, 39, 68, 194, 10, 30, 66, 175, 267]
[1, 5, 11, 39, 68, 191, 273, 674, 67, 183, 267]
[1, 5, 11, 39, 68, 191, 273, 671, 825, 1748, 267]
[1, 5, 11, 39, 68, 190, 269, 648, 66, 175, 267]
[1, 5, 11, 39, 68, 190, 269, 645, 825, 1748, 267]
[1, 4, 10, 36, 69, 198, 274, 681, 67, 183, 267]
[1, 4, 10, 36, 69, 198, 274, 678, 826, 1755, 267]
[1, 4, 10, 36, 69, 197, 270, 655, 66, 175, 267]
[1, 4, 10, 36, 69, 197, 270, 652, 826, 1755, 267]
[1, 4, 10, 35, 68, 191, 273, 674, 67, 183, 267]
[1, 4, 10, 35, 68, 191, 273, 671, 825, 1748, 267]
[1, 4, 10, 35, 68, 190, 269, 648, 66, 175, 267]
[1, 4, 10, 35, 68, 190, 269, 645, 825, 1748, 267]
[1, 4, 10, 34, 62, 148, 260, 586, 67, 183, 267]
[1, 4, 10, 34, 62, 148, 260, 585, 811, 1647, 267]
[1, 4, 10, 34, 62, 147, 259, 576, 66, 175, 267]
[1, 4, 10, 34, 62, 147, 259, 575, 811, 1647, 267]
[1, 4, 10, 33, 56, 103, 241, 449, 67, 183, 267]
[1, 4, 10, 33, 56, 103, 241, 448, 769, 1351, 267]
[1, 4, 10, 33, 56, 102, 240, 439, 66, 175, 267]
[1, 4, 10, 33, 56, 102, 240, 438, 769, 1351, 267]
[1, 4, 10, 31, 67, 188, 274, 678, 826, 1755, 267]
[1, 4, 10, 31, 67, 187, 273, 671, 825, 1748, 267]
[1, 4, 10, 31, 67, 186, 260, 585, 811, 1647, 267]
[1, 4, 10, 31, 67, 185, 241, 448, 769, 1351, 267]
[1, 4, 10, 31, 67, 183, 267]
[1, 4, 10, 30, 66, 180, 270, 652, 826, 1755, 267]
[1, 4, 10, 30, 66, 179, 269, 645, 825, 1748, 267]
[1, 4, 10, 30, 66, 178, 259, 575, 811, 1647, 267]
[1, 4, 10, 30, 66, 177, 240, 438, 769, 1351, 267]
[1, 4, 10, 30, 66, 175, 267]
[1, 3, 9, 26, 62, 149, 10, 31, 67, 183, 267]
[1, 3, 9, 26, 62, 149, 10, 30, 66, 175, 267]
[1, 3, 9, 26, 62, 148, 260, 586, 67, 183, 267]
[1, 3, 9, 26, 62, 148, 260, 585, 811, 1647, 267]
[1, 3, 9, 26, 62, 147, 259, 576, 66, 175, 267]
[1, 3, 9, 26, 62, 147, 259, 575, 811, 1647, 267]
[1, 2, 8, 18, 56, 104, 10, 31, 67, 183, 267]
[1, 2, 8, 18, 56, 104, 10, 30, 66, 175, 267]
[1, 2, 8, 18, 56, 103, 241, 449, 67, 183, 267]
[1, 2, 8, 18, 56, 103, 241, 448, 769, 1351, 267]

```
[ 1, 2, 8, 18, 56, 102, 240, 439, 66, 175, 267 ]
[ 1, 2, 8, 18, 56, 102, 240, 438, 769, 1351, 267 ]

List Size: 50

AllPath finished - Iterations: 79608

val it = () : unit
```

Podemos observar que a busca para esse tamanho de caminhos foi completa e que obtivemos 50 resultados, sendo dois caminhos de 7 passos e quarenta e oito com 11 passos. A escolha por tamanho 11 se deu devido ao fato de que cada ação nos elementos da interface ocorrer em dois passos (seleção e confirmação/cancelamento). Então, para o operador executar uma ação incorreta e reverter a mesma, são necessários quatro passos adicionais aos sete passos da ação correta, totalizando 11 passos.

5.4.2.3 – Passo 3 – Verificar o significado de cada caminho encontrado

Analisando os caminhos de 11 passos (dado que os com 7 passos correspondem aos encontrados no Teste 1) com o auxílio da função *cnodes* observamos que temos casos em que o operador coloca uma outra linha no modo de telecomando e reverte essa situação durante o procedimento de abertura da linha de tensão TL01Y3 (caminhos nos quais o segundo elemento é o estado 6 ou 5). Temos também caminhos em que o operador abre o disjuntor de outra linha e volta a fechá-lo durante o processo de abertura da linha TL01Y3 (caminhos nos quais o segundo elemento é o estado 3 ou 2). E, temos os casos em que após o início do procedimento de abertura da linha TL01Y3 o operador abre e em um passo posterior fecha um disjuntor de outra linha de tensão.

O que podemos observar é que nos casos em que o operador inicia a abertura de uma linha, através da abertura do disjuntor da linha, não há nenhum tipo de restrição quanto à navegação do mesmo, com ele podendo partir para outra tarefa sem ter terminado de abrir a linha.

Para simplificar as análises, toda a parte de navegação entre janelas, login e finalização do aplicativo foram omitidos. A presença desses elementos aumenta muito o grau de liberdade do operador, e por conseqüência a probabilidade de incidência de algum tipo de erro. Porém, para os fins de demonstração do uso das funções isso só aumentaria o número de caminhos retornados pela função, não agregando novos conhecimentos ou conclusões além dos que já foram apresentados.

O que mostramos nessa verificação é a possibilidade da ocorrência de erro de interação e também a possibilidade de reversão desse erro. Porém, nem sempre o operador do sistema se dá conta do erro, e em alguns casos ele percebe como correto o erro, devido a problemas de apresentação da interface.

De acordo com os relatórios de falha humana da CHESF, a inclusão de ações sem relação a operação é o tipo de erro de operação mais comum, o qual pode ser causado por descuido, fadiga ou imperícia no desempenho de suas atribuições.

Uma forma de se minimizar esse tipo de erro é o uso de restrição de interação ou o uso de mensagens para o operador. A restrição de interação ocorreria com o sistema percebendo que o operador está atuando em uma dada linha, o que acarretaria no bloqueio dos elementos das outras linhas até que todos os elementos da linha em trabalhos estivem em concordância (todos abertos ou todos fechados). Já o uso de mensagens seria uma forma de avisar o operador de que ele estaria operando em um elemento que não faz parte da linha atual de trabalho, como forma de alertá-lo para o fato de que a interação com a linha ainda está incompleta.

Vamos a seguir mostrar um exemplo da possibilidade de operação errada em algum ponto da interação, com essa ação sendo desfeita posteriormente.

```
cnodes(1,4);
cnodes(4,10);
cnodes(10,36);
cnodes(36,69);
cnodes(69,198);
cnodes(198,274);
cnodes(274,681);
cnodes(681,67);
cnodes(67,183);
cnodes(183,267);

Difference between node 1 and node 4 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switch_Breaker'CB_Close 1: 1`(TL01Y3,SB21Y3)++ 1`(TL01Y4,SB21Y4)++
      1`(TL01Y5,SB21Y5) --- Switch_Breaker'CB_Close 1: 1`(TL01Y4,SB21Y4)++
      1`(TL01Y5,SB21Y5)
Switch_Breaker'Wait_Conf_Open 1: empty --- Switch_Breaker'Wait_Conf_Open 1:
      1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 4 and node 10 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switch_Breaker'CB_Open 1: empty --- Switch_Breaker'CB_Open 1: 1`(TL01Y3,SB21Y3)
```

```
Switch_Breaker'Wait_Conf_Open 1: 1`(TL01Y3,SB21Y3) ---
       Switch_Breaker'Wait_Conf_Open 1: empty
Switchgear'CB_Open 1: empty --- Switchgear'CB_Open 1: 1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 10 and node 36 is:
Loc_Rem'Wait_Conf_Rem 1: empty --- Loc_Rem'Wait_Conf_Rem 1:
       1`(TL01Y4,LocRemTl01Y4)
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Loc_Rem'Local 1: 1`(TL01Y3,LocRemTl01Y3)++ 1`(TL01Y4,LocRemTl01Y4)++
       1`(TL01Y5,LocRemTl01Y5) --- Loc_Rem'Local 1: 1`(TL01Y3,LocRemTl01Y3)++
       1`(TL01Y5,LocRemTl01Y5)
val it = () : unit

Difference between node 36 and node 69 is:
Loc_Rem'Wait_Conf_Rem 1: 1`(TL01Y4,LocRemTl01Y4) --- Loc_Rem'Wait_Conf_Rem 1:
       empty
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Loc_Rem'Remote 1: empty --- Loc_Rem'Remote 1: 1`(TL01Y4,LocRemTl01Y4)
val it = () : unit

Difference between node 69 and node 198 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)++
       1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
       1`(TL01Y5,SG31Y55) --- Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y35)++
       1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
       1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open 1: empty --- Switchgear'Wait_Conf_Open 1:
       1`(TL01Y3,SG31Y34)
val it = () : unit

Difference between node 198 and node 274 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: empty --- Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y34)
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y34) --- Switchgear'Wait_Conf_Open 1:
       empty
val it = () : unit

Difference between node 274 and node 681 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Loc_Rem'Remote 1: 1`(TL01Y4,LocRemTl01Y4) --- Loc_Rem'Remote 1: empty
Loc_Rem'Wait_Conf_Loc 1: empty --- Loc_Rem'Wait_Conf_Loc 1:
       1`(TL01Y4,LocRemTl01Y4)
val it = () : unit
```

```
Difference between node 681 and node 67 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Loc_Rem'Wait_Conf_Loc 1: 1`(TL01Y4,LocRemTl01Y4) --- Loc_Rem'Wait_Conf_Loc 1:
       empty
Loc_Rem'Local    1:    1`(TL01Y3,LocRemTl01Y3)++    1`(TL01Y5,LocRemTl01Y5)    ---
Loc_Rem'Local 1: 1`(TL01Y3,LocRemTl01Y3)++ 1`(TL01Y4,LocRemTl01Y4)++
       1`(TL01Y5,LocRemTl01Y5)
val it = () : unit

Difference between node 67 and node 183 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y35)++ 1`(TL01Y4,SG31Y44)++
       1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55) ---
       Switchgear'SG_Close     1:     1`(TL01Y4,SG31Y44)++     1`(TL01Y4,SG31Y45)++
       1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55)
Switchgear'Wait_Conf_Open 1: empty --- Switchgear'Wait_Conf_Open 1:
       1`(TL01Y3,SG31Y35)
val it = () : unit

Difference between node 183 and node 267 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y34) --- Switchgear'SG_Open 1:
       1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)
Switchgear'Wait_Conf_Open 1: 1`(TL01Y3,SG31Y35) --- Switchgear'Wait_Conf_Open 1:
       empty
val it = () : unit
```

Neste exemplo apresentamos uma interação em que verificamos um erro de interação, com a posterior correção do mesmo. Ressaltamos o fato de que omitimos nas respostas os valores dos lugares de fusão, com apenas um deles sendo apresentado quando necessário (dado que a marcação dos outros lugares que compõem esse lugar de fusão são iguais).

O que percebemos aqui é que o operador pode começar a abrir a linha de tensão (abrindo o disjuntor da linha) e em seguida pode parar essa tarefa para colocar uma outra linha de tensão no modo de telecomando. Após isso ele retorna a sua tarefa para em seguida interrompe-la novamente para colocar a linha de volta no comando local.

Neste caso, essa operação pode até não ser um erro de operação, mas foi realizada durante a abertura da linha sem que nenhum mecanismo de segurança verificasse ou alertasse o operador para o fato de estar atuando em um dispositivo relacionado a outra linha. Podemos citar a partir de relatórios da CHESF, e que comprovamos pela análise do modelo, que o operador pode se equivocar durante uma operação e abrir uma chave errada

e posteriormente perceber o erro e desfazer a ação.

O problema é que um erro de operação desse pode causar prejuízos para a empresa, pois pode acarretar uma parada do fornecimento de energia em uma linha que deveria estar operando. Por outro lado, imagine se o operador atuar incorretamente em um dispositivo de uma linha que está desenergizada e com pessoal trabalhando na mesma. Isso pode devolver a tensão para a linha, com risco de vida ao pessoal envolvido na manutenção da mesma.

Ressaltamos que nesse caso testamos apenas os caminhos com 11 passos e cujo estado final é o de apenas a linha de tensão TL01Y3 estar aberta no fim da interação. Ao aumentarmos o número de passos na busca dos caminhos, ou se não definirmos completamente o estado final da busca (por exemplo se definirmos apenas o estado dos elementos da linha TL01Y3, deixando os outros indeterminados) verificamos que o operador tem realmente um grau de liberdade grande na operação, podendo inclusive atuar incorretamente em alguns elementos não percebendo o erro e não o desfazendo.

Além de verificar a ocorrência de erros de interação do tipo de execução de uma ação sem relação ou inapropriada, também é possível verificar a omissão de ações durante a interação e as diversas possibilidades de realização de uma tarefa. Nesse último ponto temos que nem sempre a realização da tarefa no menor número de passos e com a operação nos elementos corretos pode ser considerada correta, pois a seqüência de operação dos elementos também deve ser observada. Um exemplo disso é apresentado a seguir.

5.4.3 – Verificação 2: Fechamento de uma Linha de Tensão

Neste teste queremos mostrar que nem sempre a interação no menor número de passos é a correta, pois a seqüência das ações também deve ser observada. O que apresentaremos é o exemplo inverso do que mostramos no teste um, com o fechamento da linha TL01Y3, ao invés de sua abertura.

5.4.3.1 – Passo 1 – Encontrar os estados inicias e finais da interação

Para encontrar os estados inicial e final dessa busca é só utilizarmos os estados dos testes anteriores, apenas fazendo a busca no sentido inverso. Assim sendo, temos:

```
Linha LT01Y3 aberta - 267
Linha LT01Y3 fechada - 1
```

5.4.2.2 – Passo 2 – Encontrar os caminhos entre esse dois estados

Utilizando os mesmos parâmetros de tamanho (7 passos) e iteração (5000000) dos casos anteriores obtivemos:

```
AllPath(267,1,7,5000000);

List:
[ 267, 631, 824, 1737, 268, 637, 1 ]
[ 267, 631, 824, 1736, 272, 663, 1 ]
[ 267, 630, 66, 176, 268, 637, 1 ]
[ 267, 630, 66, 174, 10, 32, 1 ]
[ 267, 629, 67, 184, 272, 663, 1 ]
[ 267, 629, 67, 182, 10, 32, 1 ]

List Size: 6

AllPath finished - Iterations: 3924

val it = () : unit
```

Podemos observar que nesse caso temos 6 caminhos possíveis.

5.4.1.3 – Passo 3 – Verificar o significado de cada caminho encontrado

Analisando esses caminhos percebemos que isso se deve ao fato de não existir no fechamento de uma linha o intertravamento das seccionadoras e o disjuntor da linha. Mas, observamos que é recomendável se fechar as seccionadoras antes de fechar o disjuntor, como forma de proteger as mesmas contra um pico de tensão. Embora essa restrição não exista, achamos que a mesma deveria ser aplicada, portanto consideramos incorretos os caminhos nos quais o disjuntor não é o último elemento a ser operado.

Como exemplo temos um desses casos que consideramos incorreto.

```
cnodes(267,630);
cnodes(630,66);
cnodes(66,176);
cnodes(176,268);
cnodes(268,637);
cnodes(637,1);

Difference between node 267 and node 630 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35) ---
       Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y35)
```

```
Switchgear'Wait_Conf_Close 1: empty --- Switchgear'Wait_Conf_Close 1:
       1`(TL01Y3,SG31Y34)
val it = () : unit

Difference between node 630 and node 66 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'Wait_Conf_Close 1: 1`(TL01Y3,SG31Y34) --- Switchgear'Wait_Conf_Close
       1: empty
Switchgear'SG_Close 1: 1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++
       1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55) --- Switchgear'SG_Close 1:
       1`(TL01Y3,SG31Y34)++ 1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++
       1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55)
val it = () : unit

Difference between node 66 and node 176 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switch_Breaker'CB_Open 1: 1`(TL01Y3,SB21Y3) --- Switch_Breaker'CB_Open 1: empty
Switch_Breaker'Wait_Conf_Close 1: empty --- Switch_Breaker'Wait_Conf_Close 1:
       1`(TL01Y3,SB21Y3)
val it = () : unit

Difference between node 176 and node 268 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switch_Breaker'Wait_Conf_Close 1: 1`(TL01Y3,SB21Y3) ---
       Switch_Breaker'Wait_Conf_Close 1: empty
Switch_Breaker'CB_Close 1: 1`(TL01Y4,SB21Y4)++ 1`(TL01Y5,SB21Y5) ---
       Switch_Breaker'CB_Close 1: 1`(TL01Y3,SB21Y3)++ 1`(TL01Y4,SB21Y4)++
       1`(TL01Y5,SB21Y5)
val it = () : unit

Difference between node 268 and node 637 is:
Loc_Rem'Allow_Nav 1: 1`allow_nav --- Loc_Rem'Allow_Nav 1: empty
Switchgear'SG_Open 1: 1`(TL01Y3,SG31Y35) --- Switchgear'SG_Open 1: empty
Switchgear'Wait_Conf_Close 1: empty --- Switchgear'Wait_Conf_Close 1:
       1`(TL01Y3,SG31Y35)
val it = () : unit

Difference between node 637 and node 1 is:
Loc_Rem'Allow_Nav 1: empty --- Loc_Rem'Allow_Nav 1: 1`allow_nav
Switchgear'Wait_Conf_Close 1: 1`(TL01Y3,SG31Y35) --- Switchgear'Wait_Conf_Close
       1: empty
Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y4,SG31Y44)++
       1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++ 1`(TL01Y5,SG31Y55) ---
       Switchgear'SG_Close 1: 1`(TL01Y3,SG31Y34)++ 1`(TL01Y3,SG31Y35)++
       1`(TL01Y4,SG31Y44)++ 1`(TL01Y4,SG31Y45)++ 1`(TL01Y5,SG31Y54)++
       1`(TL01Y5,SG31Y55)
```

```
val it = () : unit
```

Como pode ser observado, neste exemplo temos o fechamento da seccionadora SG31Y34, seguida do fechamento do disjuntor da linha, e finalizando com o fechamento da segunda seccionadora da linha.

E, se aumentarmos o tamanho dos caminhos a serem encontrados ou a definição dos estados de busca, podemos verificar os mesmos erros que verificamos no caso da abertura de uma linha (ação inapropriada, ação omissa, etc).

Concluímos então que para esse sistema ser mais seguro do ponto de vista das alternativas de interação se faz necessário adotar algum tipo de mecanismos de restrição de navegação ou de alertas ao se interromper uma ação de interação antes de todos os passos da mesma serem completados.

5.4.4 – Considerações sobre o Uso das Funções

O uso das funções auxiliares facilita o processo de análise dos caminhos alternativos entre pontos específicos da interação em comparação à forma como este tipo de análise vinha sendo efetuada no GIHM. Anteriormente o uso da função *AllPath* só era factível em modelos com espaço de estados reduzidos (até uns 200 estados) e só retornava caminhos com um número específico de passos. Agora definimos que a busca deve ser por todos os caminho até um tamanho máximo e também definimos o número máximo de iterações, limitando o tempo de processamento da função. No retorno da função é mostrado se, para aquele número máximo de passos, o espaço de estado foi totalmente percorrido.

A função de comparação de estados também facilita bastante o processo de análise por apresentar apenas os lugares do modelo cuja marcação foi alterada. Com isso reduzimos bastante o tempo de análise e também os possíveis erros de interpretação, dado que a comparação manual das marcações dos lugares de dois estados é um processo tedioso e pode ser propenso a erros.

Já a função para definir os estados inicial e final nada mais é que uma aplicação da função *SearchNodes*, mas sua estrutura serve como base para a definição dessa função para outros contextos, facilitando também o processo de análise.

Em conjunto, o uso das três funções facilita o processo de análise de possibilidades de interação, permitindo a verificação de alternativas de interação e também a verificação de possíveis erros na interação. Sabemos que, embora o uso dessas funções tenha facilitado o processo de análise, ainda é requerido um certo esforço facilmente usadas por pessoas

sem muito conhecimento em redes de Petri coloridas. Mas também observamos o quanto o uso delas facilita o processo atual de análise.

A idéia na construção dessas funções é a de definir funções com as quais o projetista possa se valer para facilitar o processo de análise do modelo sem ter que se preocupar em aprender detalhes da linguagem de programação ML, tornando assim mais fácil o processo de análise do modelo.

5.5 Verificação de Modelos (*Model Checking)*

Apresentaremos agora a utilização da técnica de verificação de modelos utilizando lógica temporal. No decorrer da utilização dessa técnica percebemos que sua aplicação não é trivial, mas acreditamos que o uso dessa técnica complementa as análises apresentadas anteriormente. Apresentaremos a seguir alguns exemplos de proposições que desejamos verificar, sua definição na forma de uma fórmula em lógica temporal e a análise do resultado obtido. Não pretendemos aqui exaurir todas as possibilidades de verificação e sim exemplificar o uso da técnica aplicada ao estudo de caso.

5.5.1 – Verificação 1: É sempre possível realizar o logout do sistema?

Nessa verificação testamos se a partir de qualquer estado da interação é possível se realizar a desautenticação do sistema. Essa é uma propriedade desejada, que garante que ao final do seu turno de trabalho o operador pode se desautenticar do sistema, encerrando assim suas atividade. Esse controle é interessante por questão de controle de uso do sistema supervisório e para auditorias nos sistema.

A seguir temos a definição da proposição em lógica temporal que verifica essa propriedade.

```
fun M1 n = Mark.Industrial_MMI'Login 1 n = empty;
val logout = FORALL_UNTIL(NF("Error",M1),NOT(NF("Error",M1)));
val logout1 = EXIST_UNTIL(NF("Error",M1),NOT(NF("Error",M1)));
eval_node logout 3;
eval_node logout1 3;
```

Temos para essa verificação a definição da fórmula M1, que especifica a marcação do lugar *Login* da página *Industrial_MMI*. Nesse caso especificamos que esse lugar não possui ficha nenhuma. Essa proposição é utilizada na proposição *logout* que verifica se a

partir do estado atual de verificação sempre é possível alcançar um estado no qual o lugar *logout* não esteja vazio, indicando uma desautenticação do sistema.

Também definimos a proposição *logout1* que verifica se existe pelo menos um caminho a partir do estado atual em que o usuário se desautentica do sistema. Essa segunda proposição foi criada como forma de testar o uso da técnica de verificação de modelos, dado que sabemos de antemão que existe pelo menos um caminho de interação na qual a desautenticação é possível.

Em seguida temos a avaliação das proposições. Em ambos os casos definimos o estado inicial como sendo o estado 3, que é o primeiro estado no qual o usuário está autenticado no sistema, seguindo a evolução do modelo (esse estado foi determinado utilizando o função auxiliar *def_est*). Temos a seguir o retorno da avaliação das proposições.

```
val M1 = fn : Node -> bool
val logout = FORALL_UNTIL (NF ("Error",fn),NOT (NF (#,#))) : A
val logout1 = EXIST_UNTIL (NF ("Error",fn),NOT (NF (#,#))) : A
val it = false : bool
val it = true : bool
```

Como podemos observar inicialmente são avaliadas sintaticamente a função *M1* e as proposições *logout* e *logout1*. Em seguida é apresentado o retorno para a avaliação das duas proposições.

A proposição *logout* é avaliada como falso. Analisando esse caso concluímos que isso é devido à ação de encerramento do aplicativo, que pode ser efetuada mesmo sem a desautenticação do operador do sistema. Esse é um comportamento não desejado, que deve ser corrigido na interface.

A segunda proposição é avaliada como verdadeira. Como já discutido, esse resultado era esperado, e esse teste foi apenas uma forma de familiarização com a técnica de verificação de modelos usando lógica temporal.

5.5.2 – Verificação 2: É sempre possível encerrar a execução do aplicativo?

Essa verificação tem o intuito de testar um comportamento não desejado, conforme já mencionado no teste anterior e na verificação do modelo utilizando o grafo de ocorrência. Para aumentar a segurança na operação do sistema e o controle do mesmo, o aplicativo só deveria poder ser encerrado após a desautenticação do sistema. Só em casos extremos é que essa regra poderia se quebrada, conforme já discutido.

A seguir temos a definição da proposição em lógica temporal que verifica essa propridade.

```
fun M2 n = Mark.End'Interface 1 n = empty;
val close = EV(NF("Error",M2));
eval_node close InitNode;
```

Como no caso anterior, primeiro temos a definição da marcação de um lugar da rede, que neste caso é o lugar *Interface* da página *End*. Esse lugar está ligado aos lugares *MMI* da página *Industrial_MMI* e *Login* da página *Login* na duas instanciações da subpágina *End*. A presença de uma ficha em um desses lugares modela o fato do aplicativo estar sendo executado, com a retirada da ficha no encerramento do mesmo.

E é exatamente essa a proposição *close*, que verifica se a partir do estado atual de verificação eventualmente será sempre possível chegar a um estado no qual o lugar *Interface* possui marcação vazia. E neste caso iniciamos a verificação a partir do nó inicial, para com isso percorrer todo o espaço de estados testando a proposição *close*. O retorno da avaliação da proposição é apresentado a seguir.

```
val M2 = fn : Node -> bool
val close = FORALL_UNTIL (TT,NF ("Error",fn)) : A
val it = true : bool
```

Assim como na verificação anterior, primeiro *M2* e *close* são verificadas sintaticamente, e em seguida é apresentado o retorno da avaliação da proposição. O retorno nesse caso é verdadeiro, o que caracteriza um problema nessa interface, conforme já discutido.

5.5.3 – Verificação 3: É possível abrir um seccionadora se o disjuntor da mesma linha de tensão está fechado?

Esta verificação testa o intertavamento existente na abertura de uma linha de tensão, no qual as seccionadoras da linha só podem ser abertas se o disjuntor dessa linha já estiver aberto. A seguir temos a definição da proposição em lógica temporal que verifica essa propridade.

```
fun M3 n = Mark.Switchgear'Wait_Conf_Open 1 n = ((1,(TL01Y3,SG31Y34))!!empty);
fun M4 n = Mark.Switchgear'Wait_Conf_Open 1 n = ((1,(TL01Y3,SG31Y35))!!empty);
fun M5 n = Mark.Switch_Breaker'CB_Close 1 n = ((1,(TL01Y3,SB21Y3))!!empty);
val interblock = POS(AND(NF("Error",M5),EXIST_NEXT(OR(NF("Error",M3),
            NF("Error",M4)))));
eval_node interblock InitNode;
```

Para essa verificação foi necessária a definição de três marcações de lugares do modelo. *M3* define que a seccionadora SG31Y34 foi selecionada para abertura (presença da ficha que a define no lugar *Wait_Conf_Open* da página *Switchgear*). *M4* tem o mesmo significado, só que para a seccionadora SG31Y35. E *M5* define que o disjuntor SB21Y3 está fechado (presença da ficha que o define no lugar *CB_Close* da página *Switch_Breaker*). Observe que na definição de *M5* não definimos o estado dos outros disjuntores do sistema, pois os mesmos não tem influência no intertravamento dessa linha. Mas, para garantir que o intertravamento está correto com o disjuntor da linha e que os outros não funcionam, é necessário verificar essa propriedade para eles também. O teste é similar ao apresentado aqui, e para não estender muito o texto eles não são apresentados.

A proposição *interblock* verifica se é possível alcançar um estado no qual o disjuntor SB21Y3 está fechado e uma das seccionadoras da linha foi selecionada para abertura, estando no estado de espera de confirmação de abertura. O resultado dessa verificação é apresentado a seguir.

```
val M3 = fn : Node -> bool
val M4 = fn : Node -> bool
val M5 = fn : Node -> bool
val interblock = EXIST_UNTIL (TT,NOT (OR (#,#))) : A
val it = false : bool
```

Como pode ser observado a verificação tem como retorno que o proposição é falsa, garantindo assim que o intertravamento funciona como esperado.

5.5.4 – Verificação 4: É possível atuar em um dispositivo de uma linha de tensão sem a mesma se encontrar em comando remoto?

Este último teste verifica a indisponibilidade de interação com os elementos de uma linha de tensão no caso do comando da mesma estar no modo remoto (chave local/telecomando da linha no estado telecomando). Essa é uma ação não desejada no sistema, e que deve ser garantido que a mesma não ocorra.

Para essa verificação ser completa se faz necessário o teste para os elementos de cada linha em relação à chave local/telecomando da mesma. E, para aumentar a confiabilidade do sistema, é também interessante realizar o teste com as chaves local/telecomando das outras linhas, sendo que neste caso o retorno deve ser positivo, já que as outras chaves local/telecomando não devem interferir na operação da linha em teste.

Caso algum dos testes com as outras chaves local/telecomando retornar falso, deve-se verificar o modelo/sistema para verificar qual o fato que está causando esse comportamento anormal

Não iremos mostrar aqui todos esses testes. Apresentaremos apenas uma das verificações, que é a da tentativa de atuação em um disjuntor de um linha em comando remoto. A seguir temos a definição da proposição em lógica temporal que verifica essa propridade.

```
fun M6 n = Mark.Switch_Breaker'CB_Close 1 n = ((1,(TL01Y3,SB21Y3))!!empty);
fun M7 n = Mark.Switch_Breaker'Wait_Conf_Open 1 n = ((1,(TL01Y3,SB21Y3))!!empty);
fun M8 n = Mark.Loc_Rem'Remote 1 n = ((1,(TL01Y3,LocRemTl01Y3))!!empty);
val tel = POS(AND(AND(NF("Error",M8),NF("Error",M6)),
      EXIST_NEXT(NF("Error",M7))));
eval_node tel InitNode;
```

As funções *M6*, *M7* e *M8* definem o estado de certos lugares do modelo de interesse para essa verificação. *M6* indica que o disjuntor SB21Y3 está fechado, *M7* define que este mesmo disjuntor foi selecionado para abertura e está esperando por uma confirmação ou cancelamento de abertura. E *M8* define que a chave local/telecomando LocRemTl01Y3 está no modo de comando remoto.

A proposição *tel* verifica se é possível alcançar um estado no qual *M8* e *M6* são verdadeiros (disjuntor fechado e chave local/telecomando em comando remoto) e se em algum estado seguinte a esse *M7* é verdadeiro (sendo o disjuntor selecionado para abertura). O retorna da verificação é apresentado a seguir.

```
val M6 = fn : Node -> bool
val M7 = fn : Node -> bool
val M8 = fn : Node -> bool
val tel = EXIST_UNTIL (TT,NOT (OR (#,#))) : A
val it = false : bool
```

Como pode ser observado o retorno foi falso indicando que não é possível que essa situação ocorra, conforme era desejado para o comportamento do sistema.

5.5.5 – Considerações sobre o uso de Verificação de Modelos (*Model Checking*)

A utilização de verificação de modelos se mostra uma técnica bastante poderosa na análise de propriedades do modelo. Comportamentos desejados para o sistema podem ser testados garantindo assim a corretude do sistema.

Porém, o uso dessa técnica não se mostrou trivial, requerendo um considerável esforço de aprendizado para sua utilização. A definição das proposições mais complexas

envolvendo combinação de operadores deve ser efetuada com cuidado, de modo que o projetista não interprete errado o significado correto da proposição em questão.

Além de ser necessário o conhecimento de lógica temporal e da dificuldade de se escrever as proposições, existe também a restrição da ferramenta não apresentar um contra-exemplo nos casos de proposições avaliadas como falsas. Isso ajudaria bastante o projetista a verificar qual o fator que levou a essa avaliação falsa, que pode ser um problema de comportamento do sistema, problema de modelagem, ou mesmo problema da definição e interpretação da proposição criada.

Um problema encontrado foi a falta de documentação a respeito do uso da ferramenta e seus operadores, o que causou várias interpretações erradas durante o processo de estudo da mesma, que se baseou bastante na chamada tentativa e erro. A existência de um material de apoio introdutório e com caráter de tutorial seria uma ajuda realmente grande para a utilização do método.

Ponderando todos esses fatores pode-se concluir que a aceitação dessa técnica de análise pelos projetistas pode vir a ser restrita, devido às dificuldades inerentes supracitadas. Faz-se necessária alguma forma de facilitar o processo de definição das proposições em lógica temporal.

5.6 Resultados das Análises

Após o processo de modelagem e análise efetuado nesse trabalho podemos concluir que o uso de métodos formais, mais especificamente as redes de Petri coloridas, pode trazer grandes benefícios no projeto ou avaliação de uma interface.

As análises do modelo permitem garantir aspectos de usabilidade desejados do sistema, como também verificar possibilidades alternativas de realização de tarefas e encontrar possíveis problemas de interação ocasionados por problemas da interface ou decorrentes de erros humanos.

Mas, o uso de métodos formais traz consigo um custo associado, que é o de conhecer o método para sua aplicação. Alguns esforços foram efetuados para facilitar o processo de análise, como a definição de funções auxiliares para facilitar o trabalho.

Acreditamos que o uso de métodos formais pode ser agregado ao projeto de interfaces, trazendo mais benefícios do que desabores. Mas também sabemos que ainda se faz necessário facilitar ainda mais esse processo, tanto para aumentar sua aceitação quanto

para diminuir o tempo necessário para a realização do projeto, pois agora ele conta com o acréscimo do tempo modelagem e análise do modelo.

Capítulo 6 – Conclusões

Este trabalho apresentou uma proposta da aplicação de métodos formais no processo de projeto e avaliação de alguns aspectos ligados a usabilidade de uma interface com o usuário de sistemas de supervisão industrial, utilizando como estudo de caso o projeto da interface com o operador de uma subestação de um sistema elétrico. Buscou-se nesse trabalho averiguar as potencialidades e limitações no uso de métodos formais nesse contexto. Mais especificamente este trabalho buscou contribuir com o método de concepção de interfaces ergonômicas MCIE, dado que uma das etapas do método trata justamente da aplicação de métodos formais na modelagem da interação entre o usuário do sistema e a interface disponível para tal fim.

Da questão de pesquisa proposta inicialmente, ou seja de "investigar a aplicabilidade de modelos formais na análise do componente de interação de uma interface com o usuário de sistemas industriais críticos, visando melhorar a usabilidade e minimizar os erros cometidos durante a interação", e com base na análise dos resultados obtidos pode-se afirmar que o uso da estratégia de construção e análise de modelos da interface traz resultados que ajudam a melhorar a qualidade das interfaces em estudo. Foi apresentado um estudo de caso no qual as análises mostram que o uso de métodos formais, e mais especificamente as redes de Petri coloridas, traz benefícios para o projeto da interface com a análise do componente de interação do sistema. Embora neste trabalho seja apresentado apenas um estudo de caso, outros trabalhos realizados no GIHM também se utilizam, com resultado relevantes, de métodos formais para a análise do componente de navegação na interface, reafirmando nossa tese de que o uso de métodos formais é útil na análise de interfaces.

Além da resposta à questão central proposta neste trabalho foi proposta uma metodologia para análise de modelos e um ferramental de apoio, contribuindo assim com a etapa de modelagem e análise da interação do método MCIE.

A escolha do estudo de caso resultou, não apenas de sua adequação à classe de sistemas a que esse estudo se propõe, uma vez que a operação de sistemas elétricos se encaixa na categoria de sistemas críticos, mas principalmente pela disponibilidade da empresa CHESF ao longo da realização desta pesquisa, permitindo acesso às suas instalações, a dados sobre incidentes, sobre o funcionamento do processo, e aos operadores de uma de suas instalações.

6.1 Discussão dos Resultados

Como já foi mencionado, a escolha das redes de Petri coloridas se deveu ao fato do formalismo possuir uma representação gráfica, e ao ferramental disponível, que permite a construção de modelos, simulação e análise formal. Mas, também não podemos deixar de citar que o GIHM já vem trabalhando com redes de Petri coloridas há algum tempo. Salientamos não apenas o estudo de outros métodos disponíveis, como também o esforço de modelagem realizado com o formalismo Statechart.

Para facilitar o uso das redes de Petri coloridas o GIHM vem realizando esforços no sentido de criar arcabouços de modelagem para vários contextos, facilitando assim o processo de criação de modelos. Ao modelo apresentado nesse trabalho ainda precisam ser incorporados outros elementos presentes na interface do programa supervisório SAGE, bem como refinar os modelos dos elementos presentes de modo a contemplar todos os aspectos do comportamento dos mesmos. Com isso pretendemos que esse modelo se torne um arcabouço genérico para a modelagem de interfaces de programas de controle de subestações de energia elétrica.

Uma dificuldade no uso de modelos se dá no processo de análise. A simulação é um processo que demanda conhecimento da evolução do modelo e o acompanhamento do chamado “jogo de fichas” do mesmo. Para um modelo com muitos elementos, acompanhar essa evolução pode ser complicado. Mas essa limitação pode ser contornada ao se utilizar MSCs para gerar cenários para análise, conforme apresentado no capítulo que trata das análises do estudo de caso.

Na verificação do modelo temos uma ferramenta que possibilita vários tipos de análises de propriedades. Como foi mostrado, isso nos permite garantir questões de usabilidade que foram mapeadas em propriedades do modelo. Um projetista familiarizado com as redes de Petri coloridas não encontrará dificuldades substanciais para realizar

verificações de propriedades no espaço de estados do modelo, sejam elas as já apresentadas neste trabalho ou novas propriedades que venham a ser definidas como importantes do ponto de vista da usabilidade.

Uma limitação que existia consistia na verificação de alternativas de interação na realização de uma dada tarefa. Por simulação não era possível garantir que tínhamos verificado todas as possibilidades, e o ferramental de apoio disponível para as análises no espaço de estados do modelo tinha aplicação limitada. A função de busca de caminhos alternativos só funcionava para modelos com espaço de estados muito pequenos (100 a 200 estados) e quando os caminhos alternativos eram definidos se fazia necessária a análise manual por parte do operador das mudanças entre os estados para se analisar quais ações foram realizadas em cada caminho possível.

Para contornar essa limitação e reduzir esse esforço foram definidas funções auxiliares de análise de caminhos. Uma dessas funções é uma melhora da função que já dispúnhamos para a definição dos caminhos, conforme apresentado no capítulo 3. E a função definida para comparar dois estados facilita a análise dos caminhos e elimina o problema de interpretação errônea que pode ser causado pela comparação manual, conforme já explanado. No capítulo de análises do estudo de caso apresentamos exemplos de como o uso dessas funções pode ajudar na análise dos caminhos de interação e na predição da possibilidade dos erros de operação.

É de nosso conhecimento que essas funções ainda não são ideais para nossos propósitos, mas elas já auxiliam bastante no processo de análise de caminhos. Melhorias dessas funções deverão ser realizadas, e novas funções também deverão vir a ser definidas para facilitar ainda mais o processo de análise. Dentre as melhorias podemos citar: a comparação de estados ser feita em todo o caminho definido, e não apenas entre dois estados; fazer com que a função *AllPath* contemple a reversibilidade de ações; etc.

Também foi experimentado o uso de verificação de modelos (*model checking*) nesse trabalho. Embora tenhamos observado a grande potencialidade que essa técnica permite, através de exemplos de algumas proposições apresentadas, não a consideramos de fácil aplicação dentro do contexto desse trabalho. O uso dessa técnica requer conhecimento razoável de redes de Petri coloridas e da lógica temporal ASK-CTL. A criação de proposição em lógica temporal deve ser realizada com cuidado, para não incorrer no erro de definir uma proposição cujo significado não é o pretendido. A combinação dos operadores do ASK-CTL para se definir proposições mais complexas se mostra propenso a

erros de interpretação se o projetista não estiver bem familiarizado com a técnica. É necessário prover meios facilitadores de se escrever as proposições.

De um modo geral concluímos que a aplicação de métodos formais à modelagem e análise de modelos da interação traz benefícios para o método MCIE. O trabalho aqui apresentado mostrou como combinar diferentes técnicas de análise para avaliar a usabilidade de interfaces e qual a contribuição de cada uma dessas técnicas. E, para facilitar o processo de análise, um ferramental de apoio (funções) foi desenvolvido para facilitar o processo de análise, minimizando o esforço necessário para tal tarefa.

Porém, existem algumas limitações aos resultados apresentados nesse trabalho. Após a realização das análises e da proposta das melhorias a mesma se faz necessário testes de usabilidade com os usuários como forma de validar essas melhorias. Um protótipo deveria ter sido construído a partir das propostas de melhoria e os resultados dos testes com esse protótipo deveriam ser confrontados com os resultados observados na interface antes das análises.

Outra limitação é que, nesse estágio do trabalho, as questões ligadas à duração das tarefas não foram consideradas. E, o estudo se deu em apenas um estudo de caso, embora já existam outros trabalhos no GIHM que se valem da proposta de modelagem e análise apresentadas nesse trabalho.

Finalizando, podemos afirmar que este trabalho é relevante. Apresentamos uma consolidação do uso de análise formal no processo de projeto de interfaces em contrapartida aos trabalhos encontrados na literatura que se valem essencialmente de simulação.

As propriedades ergonômicas das interfaces são essencialmente voltadas para aspectos não-comportamentais (não funcionais), dificultando o mapeamento para propriedades de um modelo formal. Nesse trabalho temos essa formalização a partir das propriedades propostas em [91][92].

6.2 Propostas de Continuidade

Como propostas de continuidade desse trabalho destacam-se:

- Aprimorar as funções auxiliares de análise, a exemplo da função *AllPath*, que não contempla a reversibilidade de ações;

- Verificar se é possível desenvolver algum modo de facilitar o uso da verificação de modelos em ASK-CTL;
- Expandir o modelo CPN da interação de modo a contemplar outros objetos de interação presentes na interface do estudo de caso, a exemplo dos elementos de interação com o circuito de religamento automático das linhas de tensão;
- Incorporar aos objetos do modelo outras características de seu comportamento não presentes neste trabalho, a exemplo de seu comportamento em em face de uma falha do dispositivo que ele opera;
- Parametrizar os elementos do modelo de modo a torná-lo um arcabouço genérico para representar a interface de programas de supervisão do sistema elétrico;
- Realizar testes de usabilidade de modo a validar os resultados das análises aplicados a uma implementação da interface;
- Desenvolver o simulador de treinamento descrito no Anexo A;
- Aplicar a estratégia de análise de modelos a outros contextos para validar sua aplicabilidade nesses contextos;

Referências Bibliográficas

[1] "Introduction to DEVS Modeling and Simulation". July, 2003, http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/DevsJavaBookv1.0.zip

[2] "Relatório de Acompanhamento de Falhas Humanas", CHESF, 2003.

[3] "SAGE – Manual do Usuário", CEPEL, Rio de Janeiro, Brasil, 2001.

[4] Aguiar, H.; Silveira, H. J. R.; Azevedo, G. P.; Filho, E. R. G. "Centros de Controle Abertos: A Experiência do SAGE na CHESF", Revista Controle & Instrumentação, Edição 96, Valete Editora Técnica Comercial LTDA., São Paulo, SP, Setembro de 2004.

[5] Amalberti, R. "La Conduite de Systèmes à Risques". Collection Le Travail Humain, França, 1996.

[6] Andersson, M. and Bergstrand, J., "Formalizing Use Cases with Message Sequence Charts", Department of Communication Systems at Lund Institute of Technology, Maio, 1995.

[7] Barfield, L. "The User Interface – Concepts and Design". Ed. Addison-Wesley, Inglaterra, 1993.

[8] Bartak, J.; Chaumès, P.; Gissinger, S.; Houard, J.; Van Houte, U. "Operator Training Tools for the Competitive Market". IEEE Computer Applicarions in Power, pp. 25-31, Julho, 2000.

[9] Bérard, B.; Bidoit, M.; et all. "Systems ans Software Verification – Model-Checking Techniques and Tolls", Ed. Springer, Alemanha, 2001.

[10] Berstel, J. ; Reghizzi, S. C. ; Roussel, G. ; San Pietro, P. "A Scalable Formal Method for Design and Automatic Checking of User Interfaces". Proceedings of ICSE2001 – 23trd International Conference on Software Engineering, Toronto, Ontario, Canada, Maio 12-19, 2001

[11] Bodart, F.; Hennebert, A.-M.; Leheureux, J.-M.; Vanderdonckt, J.. "Towards a Dynamic Strategy for Computer-Aided Visual Placement". in Proc. of 2nd ACM Workshop on Advanced Visual Interfaces AVI'94 (Bari, 1-4 juin 1994), T. Catarci, M.F. Costabile, S. Levialdi & G. Santucci (éds.), ACM Press, pp. 78-87. Nova York, 1994.

[12] Bolognesi, T.; Brinskma, E. "Introduction to the ISO Specification Language LOTUS". Computer Network ISDN Systems , Vol 14, n° 1, pp. 25-59, 1987.

[13] Braga, J. D. M.; Figueiredo, J. C. A. "Curso de Redes de Petri". I Seminário Regional dos Estudantes de Engenharia do Nordestes, UFPB, Campina Grande, PB, Abril, 1997.

[14] Brenier, H. "Les Specifications Functionelles – Automatisme Industriels et Temps Réel". Ed Dunod, Paris, 2001.

[15] Campi, A.; Martinez, E.; San Pietro, P. "Experiences with a Formal Method for Design and Automatic Checking of User Interfaces". IUI 2004 – International Conference on Intelligent User Interfaces, Madeira, Portugal, Janeiro, 2004.

[16] CHESF Home Page. url: www.chesf.com.br.

[17] December, J. Presenting Java. Sams.Net Publishing, USA, 1994

[18] Degani, A; Heymann, M. "Formal Verification of Human-Automation Interaction". Hum Factors. 2002 Spring;44(1):28-43.

[19] Design/CPN ASK-CTL Manual. url: www.daimi.au.dk/designCPN/. 2006.

[20] Design/CPN Occurrence Graph Manual. url: www.daimi.au.dk/designCPN/, 2006.

[21] Design/CPN Online. url: www.daimi.au.dk/designCPN/, 2006.

[22] Desroches, A. A. "Modelling and Control of Automatic Manufacturing Systems". Ed. IEEE Computer Society Press, Washington, USA, 1990.

[23] Dicesare, F.; Harhalakis, G.; Proth, J. M.; Silva, M.; Vernadat, F. B. "Practice of Petri Nets in Manufacturing". Ed. Chapman & Hall, UK, 1993.

[24] Dwyer, M. B.; Robby, Tkachuk, O.; Visser, W. "Analyzing Interaction Orderings with Model Checking". 19th IEEE International Conference on Automated Software Engineering (ASE'04), Setembro 20-24, Lins, Austria.

[25] Elipse Software Home Page. url: www.elipse.com.br, 2006.

[26] Farias, G. F., "Diretrizes para Projeto de Interfaces Homem-Máquina Aplicadas a Sistemas de Supervisão de Processos Industriais", Dissertação de Mestrado, Curso de Pós-Graduação em Engenharia Elétrica, UFPB, 1994.

[27] Farias, G. F.; Turnell, M. F. V. Q. "Modelagem em Redes de Petri da Interface Homem-Máquina de Sistemas Industriais". Anais do XII Congresso Brasileiro de Automática, Uberlândia, MG, 1998.

[28] Farias, G. F.; Turnell, M. F. V. Q., "Modelagem de Aspectos Funcionais de Interfaces Homem-Máquina de Sistemas Industriais Em Redes de Petri Coloridas Hierárquicas", Anais do XIII Congresso Brasileiro de Automática, 2000.

[29] Fernandez, J. C.; Garavel, H.; Kerbrat, A.; Mounier, L.; Mateescu, R.; Sighireanu, M. "CADP: A Protocol Validation and Verification Toolbox". Anais do 8º Conference on Computer-Aided Verification, pages 437-440, New Brunswick, New Jersey, USA, Agosto, 1996

[30] Fields, B.; Wright, P.; Harrison, M. "Time, Tasks and Errors". SIGCHI Bulletin, Vol. 28, n° 2, p. 53-56, Abril, 1996.

[31] Fraikin, B.; Tchoumtchoua, J. "Présentation de la Boîte à Outils CADP: Application au Protocole POTS". Département de Mathématiques et d'Informatique, Université de Sherbrooke, Québec, Canada,Abril, 2001
ftp://ftp.inrialpes.fr/pub/vasy/publications/others/Fraikin-Tchoumtchoua-01.pdf

[32] Freitas, R. C. "Arquitetura para Integração entre Modelos Redes de Petri Coloridas e Modelos em Realidade Virtual: Uma Abordagem para Subestações Elétricas". Dissertação de Mestrado, Copele, UFCG, Campina Grande, Brasil, Julho, 2006.

[33] Freitas, R. C.; Turnell, M. F. Q. V.; Perkusich, A.; Xavier, C. S. L. "Representando a IHM de uma Subestação através de Modelos Formais e Realidade Virtual", SBSE 2006, Campina Grande, Brasil, 2006.

[34] G. Gasllasch, L. M. Kristensen, "Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN", Anais do 3º Workshop on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01), pp 79–93, 2001.

[35] Girard, p.; Baron, m.; Jambon, F. "Integrating Formal Approaches in Human-Computer Interaction Methods and Tools: an Experience". Anais do INTERACT 2003 Workshop, Sep 1-2, 2003, Zürich, Switzerland.

[36] Grupo de Interfaces Homem-Máquina Home Page. www.dee.ufcg.edu.br/~GIHM.html

[37] Gallasch, G.; Kristensen, L. M. "Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN", Anais do 3º Workshop on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01), pp 79–93, 2001.

[38] Guerrero, C. V. S. "MEDITE - Uma Metodologia Orientada a Modelos para Concepção de Interfaces Ergonômicas". Dissertação de Mestrado, Pós-Graduação em Ciência da Computação da UFCG, Campina Grande, PB, 2002.

[39] Guerrero, C. V. S. "Modelo Conceitual de Cenários de Acidentes causados pelo Erro Humano em Sistemas Industriais Críticos com o Foco na Concepção de Interfaces Ergonômicas". Tese de Doutorado, Pós-Graduação em Engenharia Elétrica, Universidade Federal de Campina Grande (UFGC), Campina Grande, PB, 2006.

[40] Guerrero, D. D. S. "Redes de Petri Orientadas a Objetos". Tese de Doutorado, Pós-Graduação em Engenharia Elétrica, Universidade Federal da Paraíba (UFPB) - Campus II, Campina Grande, PB, 2002.

[41] Harel, D. "Statecharts: A Visual Formalism for Complex Systems". Science of Computer Programming, pp: 231 - 274, North-Holland, 1987.

[42] Harel, D.; Pnueli, A.; Schimidt, J. P.; Sherman, R. "On the Formal Semantics of Statecharts". Anais do 2º IEEE Symp. On Logic in Computer Science, pp: 54 – 64, IEEE Press, 1987.

[43] IAR Sistems Home Page, http://www.iar.com/Products/VS/

[44] I-Logix – Rhapsody in C Home Page, http://www.ilogix.com/products/rhapsody/rhap_inc.cfm

[45] I-Logix – StateMate Home Page, http://www.ilogix.com/products/magnum/index.cfm

[46] Indusoft – Tools for Automation. url: www.indusoft.com.br, 2006.

[47] Jacquot, J.-P. ; Quesnot, D. "Early Specification of User-Interfaces: Toward a Formal Approach". Anais do 19º International Conferece on Software Engeering, Boston, Massachusetts, May 18-27, 1997.

[48] Java Home Page, www.java.sun.com

[49] Jensen, K. "Coloured Petri Nets – Basic Concepts, Analysis Methods and Pratical Use – Vol. 1". Ed. Spring-Verlag, USA, 1992.

[50] Jensen, K. "Coloured Petri Nets – Basic Concepts, Analysis Methods and Pratical Use – Vol. 2". Ed. Spring-Verlag, Germany, 1995.

[51] Johnson, C.; Gray, P. "Temporal Aspects of Usability". SIGCHI Bulletin, Vol. 28, n° 2, p. 32, April, 1996.

[52] Lacaze, X.; Palanque, P. "Comprehensive Handling of Temporal Issues in Tasks Models: What is needed and How to Support it?". Workshop of The Temporal Aspects Of Work For HCI, CHI 2004; Abril 24-29, 2004. Vienna, Austria.

[53] Lemos, A. J. P. "Reuso de Modelos em Redes de Petri Coloridas", Dissertação de Mestrado, Pós-Graduação em Informática da UFPB, Campina Grande, PB, 2001.

[54] Leung, K. R. P. H.; Yiu, L. C. K.; Tang, R. W. M. "Modelling Web Navigation by Statechart". Anais do 24º Annual Internacional Computer Software and Aplications Conference (COMPSAC'2000), Taipei, Taiwan, Outubro, 2000.

[55] Lukka, T. J. "FreeWRL – VRML/Browser". Harvard Society Fellows. http://www.perl.org/tpc/1998/User_Applications/FreeWRL/. 1998.

[56] Markopoulos, P.; Pycock, J.; Wilson, S., Johnson, P. "ADEPT - A Task Based Design Environment". Anais do 25º Hawaii International Conference on System Sciences, Conference, Vol. II, IEEE Computer Society Press (California), pp. 587-596, 1992.

[57] Marques , E. F.; Bramorski, M. M. "Apostila de VRML Básico", Universidade Federal de Santa Catarina, 1998.

[58] Massó, J. P. M.; López, P. G. "Model-Based Design and New User Interfaces Current Practices and Opportunities". IUI 2004 – International Conference on Intelligent User Interfaces, Madeira, Portugal , Janeiro, 2004

[59] Mori, G.; Paternò, F.; Santoro, C. "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design". IEEE Transactions on Software Engineering, Vol 28, nº 8, pp.797-813, Agosto 2002.

[60] Murata, T. "Petri Nets: Properties, Analysis and Applications". Proceedings of the IEEE, nº 77 (4), p. 541-580, Abril, 1989.

[61] Navarre, D.; Palanque, P.; Paternò, F.; Santoro, C.; Bastide, R. "A Tool Suite for Integrating Task and System Models through Scenarios". Revised Papers of 8th Internacional Wordshop, DSV-IS 2001. Glasgow, Scotland, UK. Junho de 2001.

[62] Nedel, L. P.; Freitas, C. M. D. S.; Schyn, A.; Navarre, D.; Palanque, P. "Usando Modelagem Formal para Especificar Interação em Ambientes Virtuais: Por que?". SVR 2003 - SBC Symposium on Virtual Reality, 2003, Ribeirão Preto. Proceedings of SVR 2003 - VI Symposium on Virtual Reality. Porto Alegre: SBC - Brazilian Computer Society, 2003. v. 6, p. 81-92.

[63] Neto, J. A. N. "Modelagem da Interface Homem-Máquina de uma Subestação Elétrica", Dissertação de Mestrado, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 2004.

[64] Neto, J. A. N.; Scaico, A. ; Neto, J. P. B.; Turnell, M. F. Q. V. "Avaliação de Estratégias para Integração de Modelos HCPN de Sistemas Industriais Críticos". Anais do CBA 2006 - Congresso Brasileiro de Automatica, Salvador, BA, 2006.

[65] Neto, J. A. N.; Turnell, M. F. Q. V. "Modelando Componentes da IHM de Sistemas Industriais em Redes de Petri Coloridas". Anais do VI SBAI – Simpósio Brasileiro de Automação Inteligente, Bauru, SP, 2003.

[66] Netto, A. V. et. al. "Realidade Virtual – Fundamentos e Aplicações", Ed. Visual Books, pp 45-65, 2002.

[67] Neuman, P.; Pokorný, M.; Varcop, L.; Weiglhofer, W. "Engineering and Operator Training Simulator of Coal-Fired Steam Boiler". Anais do 10º International Conference MATLAB02, Praga, Republica Tcheca, 2002.

[68] Paternò, F. "ConcurTaskTrees: An Engineered Approach to Model-Based Design of Interactive Systems". CApítulo 24, em Diaper, D., Stanton, N. (Eds.), The Handbook of Task Analysis for Human-Computer Interaction, pp.483-503, Lawrence Erlbaum Associates, Mahwah, 2003.

[69] Paternò, F; Mancini, C.; Meniconi, S. "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models". Anais do Interact'97, Ed. Chapman&Hall, pp.362-369, Jullho, 1997.

[70] Pereira, L. A. C.; Lima, L. C. et all. "SAGE – Um Sistema Aberto para Evolução", www.chesf .gov.br

[71] Queiroz, J. E. R. "Validação de uma Metodologia de Projeto de Interfaces Usuário-Computador". Dissertação de Mestrado, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 1994.

[72] Rauterberg, M.; Fjeld, M. "An Analysing and Modelling Tool Kit for Human-Computer Interaction". Anais do 7º International Conference on Human-Computer Interaction, (HCI International '97), San Francisco, Califormia, USA, 1997.

[73] Rauterberg, M.; Schluep, S.; Fjeld, M. "How to model behavioural and cognitive complexity in human-computer interaction with Petri nets". Anais do 6º IEEE International Workshop on Robot and Human Communication, Sendai, 1997.

[74] Ribeiro, Z. B. S. "Supervisores Distribuídos para Sistemas Flexíveis de Manufatura utilizando Redes de Petri". Dissertação de Mestrado, Departamento de Engenharia Elétrica, UFPB, Campina Grande, PB, agosto, 1999.

[75] SAGE Home Page. url: www.sage.com.br.

[76] Sage, M.; Johnson, C. "A Case Study in Integrating Formal Methods into the HCI Development Cycle". DSVIS'98: 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems, Cosener's House, Abingdon, UK, Junho, 1998

[77] Sampaio, R. F. "Sistema de Diagnóstico de Faltas para Subestações Baseado em Redes de Petri Coloridas". Dissertação de Mestrado, Coordenação do Curso de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Ceará, Fortaleza, CE, 2002.

[78] Santoni, Charles ; Turnell, M. F. Q. V. ; Scaico, A. ; Vieira F. A. Q. ; Pereira M. R. B. . Reviewing Human Errors in Industrial Incidents for Safety Design. In: Agostino G. Bruzzone; Edward Williams. (Org.). Anais do 2004 Summer Computer Simulation Conference. San Diego CA: The society for Modelling and Simulation Intrenational, 2004, v. , p. 69-75.

[79] Santos, J. A. M. "Suporte a Análise e Verificação de Modelos RPOO". Dissertação de Mestrado, Pós-Graduação em Informática, Universidade Federal de Campina Grande, Campina Grande, PB, fevereiro, 2003.

[80] Scaico, A. "Aplicação de um Modelo Genérico de Navegação de IHM ao Contexto de Sistemas Industriais", Dissertação de Mestrado, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 2001.

[81] Scaico, A. "Arcabouço para a Modelagem de Interfaces com Usuário de Sistemas Supervisórios Industriais". Relatório de Projeto e Pesquisa, Coordenação de Pós-Graduação em Engenharia Elétrica, UFCG, 2003.

[82] Scaico, A. "Estudo de Formalismos para a Modelagem do Componente de Interface com o Usuário com Aplicação no Contexto de uma Subestação de Energia Elétrica". Relatório de Projeto e Pesquisa, Coordenação de Pós-Graduação em Engenharia Elétrica, UFCG, 2004.

[83] Scaico, A. "Modelagem e Análise de Características Temporais de Interfaces com Usuário". Relatório de Projeto de Pesquisa, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 2002.

[84] Scaico, A.; Turnell, M. F. Q. V. "Análise do Tempo na Interação do Usuário com Sistemas de Automação Industrial". Anais do CBA 2002 – Congresso Brasileiro de Automática, Natal, RN, 2002.

[85] Scaico, A.; Turnell, M. F. Q. V. "Modelagem da Navegação de Interfaces com o Usuário de Sistemas de Automação". Anais do V SBAI – Simpósio Brasileiro de Automação Inteligente, Canela, RS, 2001.

[86] Scherer, D. "Proposta de Suporte Computacional ao MCI". Dissertação de Mestrado, Pós-Graduação em Ciência da Computação da UFCG, Campina Grande, PB, 2004.

[87] Schyn, A. ; Navarre, D.; Palanque, P.; Nedel, L. P.; "Description Formelle d'une Technique d'Interaction Multimodale dans une Application de Réalité Virtuelle Immersive ". 15th French-Speaking Conference on Human-Computer Interaction, 2003, Caen, Anais do IHM 2003, International Conference Proceedings Series, Nova York, ACM, 2003, p. 150-157

[88] Sessitskaia, I. "Verifying Lustre HCI Models by Abstraction". EUROCAST 01 Functional Programming Workshop, Las Palmas, Grande Canarie, Fevereiro, 2001.

[89] Silva, L. D. "Modelagem Sistemática de Sistemas Flexíveis de Manufatura Baseada em Reuso de Modelos de Redes de Petri Coloridas", Dissertação de Mestrado, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 2002.

[90] Silva, L. D.; Perkusich, A. "Modelagem Sistemática de Sistemas Flexíveis de Manufatura". Anais do XIV CBA – Congresso Brasileiro de Automática, , pp 227-232, Natal, Brasil, 2002.

[91] Sousa, M. R. F. "Avaliação Iterativa de Especificação de Interfaces com Ênfase na Navegação". Tese de Doutorado, Pós-Graduação em Engenharia Elétrica da UFPB, Campina Grande, PB, 1999.

[92] Sousa, M. R. F.; Turnell, M. F. Q. V. "User Interface Based on Coloured Petri Nets Modelling and Analysis". Anais do 1998 IEEE International Conference on Systems Man and Cybernetics, San Diego, USA, 1998.

[93] Sousa, M. R. F.; Turnell, M. F. Q. V. "User Interface Design Based on Coloured Petri Nets Modelling and Analysis", Anais do 1998 IEEE International Conference on Systems Man and Cybernetics, San Diego, USA, 1998.

[94] Spanel, U.; Kreutz, M.; Roggatz, C. "Simulator Based Operator Training − Ensuring Quality of Power System Operation". DUtrain GmbH, Alemanha. 2006. url: www.dutrain.de/en_publikationen.html

[95] TERESA Home Page. http://giove.cnuce.cnr.it/teresa.html

[96] The MathWorks – StateFlow Home Page, http://www.mathworks.com/products/stateflow/

[97] Trætteberg, H. "Model-based User Interface Design". Tese de doutorado; Department of Computer and Information Sciences; Faculty of Information Technology, Mathematics and Electrical Engineering; Norwegian University of Science and Technology; Noruega; 2002

[98] Turine, M. A. S.; Oliveira, M. C. F.; Masiero, P. C. "A Navigation-Oriented Hypertext Model Based on Statecharts**".** Anais do 8º ACM Conference on Hypertext, pages 102-111, Southampton, United Kingdom, 1997.

[99] Turnell, M. F. Q. V. ; Neto, J. A. N. ; Scaico, A. ; Santoni, C. ; Mercantini, J. M. ; Chouraqui, E. ; Guerrero, C. V. S. "O Erro Humano e o Projeto das IHM Industriais". Revista Controle & Instrumentação, São Paulo, p. 71 - 76, 28 fev. 2005.

[100] Turnell, M. F. Q. V.; Scaico, A;. Santoni, C.; Vieira, F. A. Q.; Pereira, M. R. B. "Análise de Incidentes Industriais Baseada em Modelos". Anais do CBA 2004 - XV Congresso Brasileiro de Automática, 21 a 24 de setembro de 2004, Gramado - Brasil

[101] Turnell, M. F. Q. V.; Guerrero, C. V. S.; Mercantini, J. M.; Chouraqui, E.; Vieira F. A. Q.; Pereira M. R. B. "Modelling Incident Scenarios to enrich User Interface Development". Em: Human Error, Safety, and Systems Development ed.: Kluwer Acad. Publ., 2004.

[102] Turnell, M. F. Q. V.; Santoni, C.; Scaico, A.; Vieira, F. A. Q.; Pereoira, M. R. B. "Interaction Modelling and Safety in the Electricity Industry", Anais do I3M'04 - International Mediterranean Modeling MultiConference, Genva, Italia, October 28-31, 2004.

[103] Turnell, M. F. Q. V.; Santoni, C.; Scaico, A.; Vieira, F. A. Q.; Pereira, M. R. B. "Reviewing Human Errors in Industrial Incidents for Safety Design", Anais do SCSC'04 - 2004 Summer Computer Simulation Conference , San Jose, California, USA, Julho, 2004.

[104] 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, Alemanha, 2001.

[105] Turnell, M. F. Q. V.; Scaico, A;. Neto, J. A. N.; Santoni, C.; Mercantini, J. M. "A Model Based Operator Training Simulator for Electric Systems". Anais do International Modeling and Simulation Multiconference 2007 (IMSM07), Buenos Aires, Argentina, 2007.

[106] Turnell, M. F. V. Q., "Conceitos e Projeto de Interfaces Usuário-Computador", Notas de Aula da Disciplina Projeto de Interface Homem-Máquina, Curso de Pós-Graduação em Engenharia Elétrica, UFPB, 2000.

[107] Vangheluwe, H. "The Discret EVent System specification (DEVS) formalism". Modelling and Simulation Course Notes. School of Computer Science, McGill University, Setembro, 2002.

[108] Winckler, M.; Farenc, C.; Palanque, P.; Bastide, R. "Designing Navigation for Web Interfaces". IHM-HCI 2001, 10-14 Setembro, Lille, França, 2001.

[109] Winckler, M.; Farenc, C.; Palanque. P. "Une démarche structurée pour la conception et l'évaluation d'applications Web par l'exploitation synergique des modèles de tâche et de navigation" IHM 2002 - 14e Conférence Francophone sur l'Interaction Homme-Machine, Faculté des Sciences, Poitiers, Novembro, 2002.

[110] Wind River - Better State Home Page, http://www.windriver.com/products/betterstate/index.html

[111] Xjtek Home Page, http://www.xjtek.com/products/anylogic/

[112] Zhou, M.; Dicesare, F. "Petri Net Synthesis for Discret Event Control of Manufacturing Systems". Ed. Kluwer Academic Publisher, Massachusettes, USA, 1993.

Anexo A – Funções Desenvolvidas para a Análise de Modelos CPN

1. Funções para Determinar os Estados Inicial e Final da Interação

1.1 Função para determinar nós a partir da marcação de um lugar onde as fichas são de apenas um elemento

Temos a seguir o código da função para o exemplo da página ser *IHM_Industrial* e o lugar ser *Fim*. Essa função vai retornar todos os estados nos quais a marcação do lugar *Fim* da página *IHM_Industrial* vai ser igual ao parâmetro *a* fornecido para a execução da função. Esse parâmetro deve ser o nome de um elemento presente nessa partição do modelo (página) da interface. E, a seguir apresentamos um exemplo de chamada dessa função para o caso do parâmetro ser *BotFinalizaProg*.

```
fun nos_simples(a) =
      PredAllNodes (
      fn n => cf(a,Mark.IHM_Industrial'Fim 1 n) = 1);

nos_simples(BotFinalizaProg);
```

O retorno dessa função é o seguinte:

```
val nos_simples = fn : Botao -> Node list
val it = [99,98,97,96,95,94,93,92,91,90,9,89,...] : Node list
```

No retorno da função temos que a função é válida sintaticamente e quais são os tipos com os quais ela trabalha, e em seguida temos uma lista com os nós do Occ (estados do modelo) que atendem o predicado definido.

Ressaltamos que o problema de parametrização da página do modelo e do lugar de pesquisa está presente em todas as funções subseqüentes, de modo que sempre que o lugar de busca mudar deve-se mudar esses parâmetros no corpo da função.

1.2 Função para determinar nós a partir da marcação de dois lugares onde as fichas são de apenas um elemento

Temos a seguir a função e dois exemplos de sua execução. Na primeira chamada à função os elementos de busca fora intencionalmente trocados para o retorno ser uma lista vazia, e na segunda chamada a ordem dos elementos está correta. Isso é apenas para mostrar que a ordem dos elementos deve estar de acordo com a definição dos parâmetros em relação ao lugar de busca na função.

```
fun nos_simples1(a,b) =
       PredAllNodes (
       fn n => (cf(a,Mark.IHM_Industrial'Fim 1 n) = 1
       andalso cf(b,Mark.Navegacao'Bot_Close 1 n) = 1));

nos_simples1(BotAjudaFechar,BotFinalizaProg);
nos_simples1(BotFinalizaProg,BotAjudaFechar);
```

O retorno dessa função é o seguinte:

```
val nos_simples1 = fn : Botao * Botao -> Node list
val it = [] : Node list
val it = [99,98,97,96,95,94,93,92,91,90,9,89,...] : Node list
```

1.3 Função para achar nós cuja marcação de um dado lugar seja uma dupla

Temos a seguir a função e um exemplo de sua execução.

```
fun nos_dupla(a,b) =
       PredAllNodes (
       fn n => (cf((a,b),Mark.Disjuntor'Disj_Aberto 1 n) = 1));

nos_dupla(LT01Y4,DJ21Y4);
```

O retorno dessa função é o seguinte:

```
val nos_dupla = fn : Linha1 * chave -> Node list
val it = [82,81,80,79,78,77,76,75,74,599,598,597,...] : Node list
```

1.4 Função para achar nós cuja marcação seja uma dupla fornecendo como parâmetro apenas o segundo elemento da dupla

Temos a seguir a função, um exemplo de sua execução e a verificação do conteúdo de alguns dos estados retornados pela execução da função, através de sua transformação em

uma string, comprovando que os estados retornados possuem o parâmetro de pesquisa como segundo elemento da dupla.

```
fun nos_dupla_elemen(x) =
       let
              fun elementos(x,[]) = false |
                    elementos(x,(a,b)::lista) = if b=x then true
                                                else elementos(x,lista)
   in
              SearchNodes (
              EntireGraph,
              fn n => (elementos(x,ms_to_list(Mark.Disjuntor'Disj_Aberto 1 n))),
              NoLimit,
              fn n => n,
              [],
              op :: )
   end;

nos_dupla_elemen(DJ21Y3);

ms_to_list(Mark.Disjuntor'Disj_Aberto 1 92);
ms_to_list(Mark.Disjuntor'Disj_Aberto 1 60);
ms_to_list(Mark.Disjuntor'Disj_Aberto 1 85);
```

O retorno dessa função é o seguinte:

```
val nos_dupla_elemen = fn : chave -> Node list
val it = [92,91,90,89,88,87,86,85,84,62,61,60,...] : Node list
val it = [(LT01Y3,DJ21Y3)] : Atua_Chave list
val it = [(LT01Y3,DJ21Y3)] : Atua_Chave list
val it = [(LT01Y3,DJ21Y3)] : Atua_Chave list
```

1.5 Função para achar nós cuja marcação seja uma lista de duplas fornecendo como parâmetro apenas o primeiro elemento da dupla que encabeça a lista

A seguir apresentamos a função e um exemplo de sua execução.

```
fun nos_lista(x) =
       let
          fun elementosJ1(x,[]) = false |
              elementosJ1(x,(z,w)::lista) = if x=z then true
                                            else elementosJ1(x,lista);
          fun elementosJ(x,[]) = false |
                  elementosJ(x,c::lista) = elementosJ1(x,c)
   in
              SearchNodes (
```

```
            EntireGraph,
            fn n => (elementosJ(x,ms_to_list(Mark.Navegacao'IHM 1 n))),
            NoLimit,
            fn n => n,
            [],
            op :: )
   end;

nos_lista(JanAjuda);

ms_to_list(Mark.Disjuntor'Disj_Aberto 1 92);
ms_to_list(Mark.Disjuntor'Disj_Aberto 1 60);
ms_to_list(Mark.Disjuntor'Disj_Aberto 1 85);
```

O retorno dessa função é o seguinte:

```
val nos_lista = fn : Janela -> Node list
val it = [99,98,97,96,86,76,66,60,599,598,597,596,...] : Node list
```

1.6 Função geral para achar estados do modelo fornecendo como parâmetro todos os elementos de interesse deste estudo

A seguir apresentamos a função e um exemplo de sua execução. Observe que todos os parâmetros de busca estão inseridos no corpo da função, não existindo assim parâmetros de entrada na função.

```
(* Funcao generica para definir todos os estados dos elementos da interface *)

fun def_est() =
   let
       fun elementosJ1(x,[]) = false |
            elementosJ1(x,(z,w)::lista) = if x=z then true
                                          else elementosJ1(x,lista);
       fun elementosJ(y,[]) = false |
           elementosJ(y,c::lista) = elementosJ1(y,c);
        fun elementos(t,[]) = false |
          elementos(t,(a,b)::lista) = if b=t then true
                                      else elementos(t,lista)
   in
     SearchNodes (
     EntireGraph,
     fn n => (elementosJ(JanSinotico,ms_to_list(Mark.Navegacao'IHM 1 n))
       andalso elementos(DJ21Y3,ms_to_list(Mark.Disjuntor'Disj_Fechado 1 n))
       andalso elementos(DJ21Y4,ms_to_list(Mark.Disjuntor'Disj_Fechado 1 n))
       andalso elementos(DJ21Y5,ms_to_list(Mark.Disjuntor'Disj_Fechado 1 n))
```

```
       andalso elementos(SC31Y34,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(SC31Y35,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(SC31Y44,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(SC31Y45,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(SC31Y54,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(SC31Y55,ms_to_list(Mark.Seccionadora'Sec_Fechada 1 n))
       andalso elementos(LocTelLt01Y3,ms_to_list(Mark.Loc_Tel'Local 1 n))
       andalso elementos(LocTelLt01Y4,ms_to_list(Mark.Loc_Tel'Local 1 n))
       andalso elementos(LocTelLt01Y5,ms_to_list(Mark.Loc_Tel'Local 1 n))
),
   NoLimit,
   fn n => n,
   [],
   op :: )
end;

def_est();
```

O retorno dessa função é o seguinte:

```
val def_est = fn : Janela * chave * chave * chave * chave * chave * chave * chave
* chave * chave * chave * chave * chave -> Node list
val it = [8,24] : Node list
```

2. Função *AllPath* Original

```
fun member (x, [])   = false
  | member (x, h::t) = x = h orelse member (x,t);

fun AllPathAux (aBegin, aEnd, aOrdem, aList, aAux) =
  let val n = ref 1 and k = ref 0 and
      mOutNodes = ref [] and mSavPath = ref [] and
      mList = ref aList and mAux = ref aAux
  in
    ( mSavPath := !mAux;
      mAux := !mAux @ (aBegin :: nil);

      if aBegin = aEnd andalso (aOrdem = 1 orelse length(!mAux) = aOrdem) then
        ( mList := !mList @ (!mAux :: nil) )
      else
        ( mOutNodes := OutNodes(aBegin);
          n := length(!mOutNodes);
          while !k < !n do
            if member( nth(!mOutNodes, !k), !mAux ) then
              ( k := !k + 1 )
```

```
            else
              ( mList := AllPathAux( nth(!mOutNodes,
!k),aEnd,aOrdem,!mList,!mAux);
                k := !k + 1 )
        );

      mAux := !mSavPath
    );

    !mList
  end;

fun AllPath(nB, nE, nOrd) =
  let val mList = ref [] and mAux = ref []
  in
    ( mList := AllPathAux(nB, nE, nOrd+1, !mList, !mAux) );
    !mList
end;
```

3. Função *AllPath* Modificada

```
(* length of List *)
fun listLen [] = 0
  | listLen (_::t) = 1 + listLen t;

(* Member(x,L) determines whether x is on list L *)
fun Member (x, [])   = false
  | Member (x, h::t) = if x=h then true else Member(x,t);

(* contatenation L@M of the lists L and M *)
fun Append([],M) = M
  | Append(h::t,M) = h::Append(t,M);

(* nth element in List *)
exception nthElemExcept;
fun nthElem(h::_,0) = h
  | nthElem(h::t,n) = if n>0 then nthElem(t,n-1)
                      else raise nthElemExcept
  | nthElem _ = raise nthElemExcept;

(* nth first elements in List *)
exception firstElemExcept;
fun firstElem(L,0) = []
  | firstElem(h::t,n) = if n>0 then h::firstElem(t,n-1)
```

```
                        else raise firstElemExcept
  | firstElem _ = raise firstElemExcept;

(* print a List *)
fun printListOfInt([]) = ()
  | printListOfInt(h::t) = (
    print(Int.toString(h));
    if t = nil then
      print("")
    else
      print(", ");
    printListOfInt(t)
  );


(* depth-first search = DFS *)
fun AllPathAux (ik, iIt, iBegin, iEnd, iDepth, iLS, pAux) =
  let
    val iOutNodesLen = ref 0;
    val i = ref 0;
    val ipAuxLen = ref 0;
    val pOutNodes = ref [];
  in
    (
      ik := !ik + 1;
      ipAuxLen := listLen(!pAux);

      if !ipAuxLen <= iDepth orelse iDepth = 0 then
      (
        pAux := Append(!pAux, (iBegin :: nil));

        if iBegin = iEnd andalso (iDepth = 0 orelse !ipAuxLen+1 <= iDepth) then
        (
          iLS := !iLS + 1;
          print("[ ");
          printListOfInt(!pAux);
          print(" ]\n")
        )
        else
        (
          pOutNodes := OutNodes(iBegin);
          iOutNodesLen := listLen(!pOutNodes);
          while !i < !iOutNodesLen do
            if Member( nthElem(!pOutNodes, !i), !pAux ) then
              i  := !i + 1
            else
```

```
            (
              if !ik < iIt then
              (
                AllPathAux(ik, iIt, nthElem(!pOutNodes, !i), iEnd, iDepth, iLS,
pAux);
                i := !i + 1
              )
              else
                i := !i + 1
            )
        );

        pAux := firstElem(!pAux, !ipAuxLen)
      )
      else
        ik := !ik - 1
    )
  end;

(* All Paths from node iB to iE, with iD Depth. Program executes in the maximum
iT *)
fun AllPath(iB, iE, iD, iT) =
  let
    val lAux   = [];
    val pAux   = ref lAux;
    val ikont  = ref 0;
    val iLSize = ref 0;
  in
    if iB = iE then
      print("\nInput Error: Begin Node equal End Node")
    else
      if iD < 0 then
        print("\nInput Error: Depth less than 0")
      else
        if (iT < 0) then
          print("\nInput Error: Iterations less than 0")
        else
        (
          print("\nList:\n");

          AllPathAux(ikont, iT, iB, iE, iD, iLSize, pAux);

          print("\nList Size: ");
          print(Int.toString(!iLSize));
          print("\n\n");
```

```
          if !ikont < iT then
            print("AllPath finished - Iterations: ")
          else
            print("AllPath incomplete - Iterations: ");
            print(Int.toString(!ikont))
    );
    print("\n\n")
  end;
```

4. Função para Comparar dois Estados

```
fun cnodes(initn,endn) =
   let
     fun parts(x) =
        let
          val create_list = String.tokens (fn x => not (Char.isPrint  x));
        in
          create_list(!st_NodeDescrRef(x))
       end;
    val node1 = parts(initn)
    val node2 = parts(endn)
    val node1a = tl(node1)
    val node2a = tl(node2)
    val resp = "\nDifference between node "^hd(node1)^" and node "
^hd(node2)^" is:\n"
    fun state([],[],resp) = resp |
         state(cab1::l1,cab2::l2,resp) = if cab1<>cab2  then
state(l1,l2,resp^cab1^" --- "^cab2^"\n")
                                        else state(l1,l2,resp);
   in
     CPN'Edit.out_val(state(node1a,node2a,resp))
   end;
```

Anexo B – Proposta de um Simulador de Treinamento Baseado em Modelos

Nos dias atuais uma questão chave na qualidade na operação dos sistemas de potência é a habilidade dos operadores para analisar o estado do sistema, inferir conclusões corretas e agir apropriadamente de acordo com o estado atual do sistema [94]. Mas, com a abertura do mercado de energia elétrica, a concorrência entre as diferentes empresas, a necessidade de interação entre fornecedores diversos e o fato dos sistemas trabalharem praticamente em seu limite operacional fazem com que os problemas de operação sejam rapidamente percebidos pelos usuários do sistema, causando insatisfação [94].

Deve-se prover os operadores do sistema de toda a capacidade técnica e cognitiva para lidar com a operação do sistema, principalmente em casos de distúrbios. Mas, devido aos fatores citados, é bastante complicado lidar com distúrbios devido ao tempo de resposta demandado em face da necessidade de um sistema confiável e com garantia de serviço aos usuários [94].

A solução é o treinamento dos operadores, tanto em situação normal de operação (visando a efetiva apreensão dos procedimentos operacionais das tarefas cotidianas) como em situações de distúrbio ou falhas no sistema (visando prover o operador com o discernimento e a rapidez necessária no processo de detecção da falha, eliminação da mesma, e recuperação do sistema) [94][8] [67].

O uso de simuladores permite aumentar a competência do operador, bem como o treinamento em tarefas que não poderiam ser efetuadas no ambiente de trabalho (devido a riscos, custos operacionais, problemas com consumidores, etc). Segundo [67], Os simuladores de treinamento de operação podem ser divididos em três categorias:

- Estimulação completa: neste caso o treinamento ocorre na planta real, sendo necessário deixar a planta inoperante durante a tarefa de treinamento, e, pondo em risco o próprio sistema devido a possíveis erros do operador

- Estimulação parcial: neste caso os sistemas de processo e controle são emulados (é construído um motor de simulação em software representando a planta), mas a IHM é a que é usada no sistema real
- Emulação completa: aqui todo o sistema (planta, controle e IHM) é emulado com base em um software SCADA para prover o realismo necessário fazendo o operador abstrair o fato de estar lidando com uma situação hipotética.

A nossa proposta é desenvolver um simulador por emulação completa com o motor de simulação a partir do modelo formal do sistema. Podemos citar como vantagem do uso de modelos o fato dos mesmos serem utilizados tanto para a simulação do comportamento do sistema e para análise, quanto como motor de simulação para o treinamento dos operadores, não sendo necessário o desenvolvimento de dois componentes separados (um para análise do sistema e outro para treinamento), o que aumenta a confiança do motor de simulação e reduz o tempo de desenvolvimento. Para o caso do simulador o que deve ser feito é a IHM (em um programa supervisório comercial), a comunicação entre a IHM e o modelo do sistema e um módulo que permita inserir distúrbios no modelo do sistema de forma a emular distúrbios.

1. Arquitetura do Simulador

O simulador proposto é composto por três modelos em redes de Petri coloridas que formam o motor de simulação e da representação da interface disponível ao operador do sistema. Na Figura 28 temos um esquema da arquitetura do simulador. Os modelos em redes de Petri representam a planta elétrica, a interface de operação via controle supervisório e a interface de operação via painéis de comandos. E na Figura 29 é apresentada a sala de operação existente na CHESF, onde podemos perceber as estações de trabalho executando o programa supervisório (SAGE) e os painéis de comando.

A representação da interface pode ser realizada de duas formas: por meio de um programa comercial de controle supervisório (emulando o controle supervisório do sistema) ou pelo uso de realidade virtual utilizando a linguagem VRML [57] (emulando o controle via painéis).

A comunicação entre esses componentes é realizada por meio de troca de mensagens TCP/IP. Nas subseções a seguir apresentaremos cada um desses componentes.

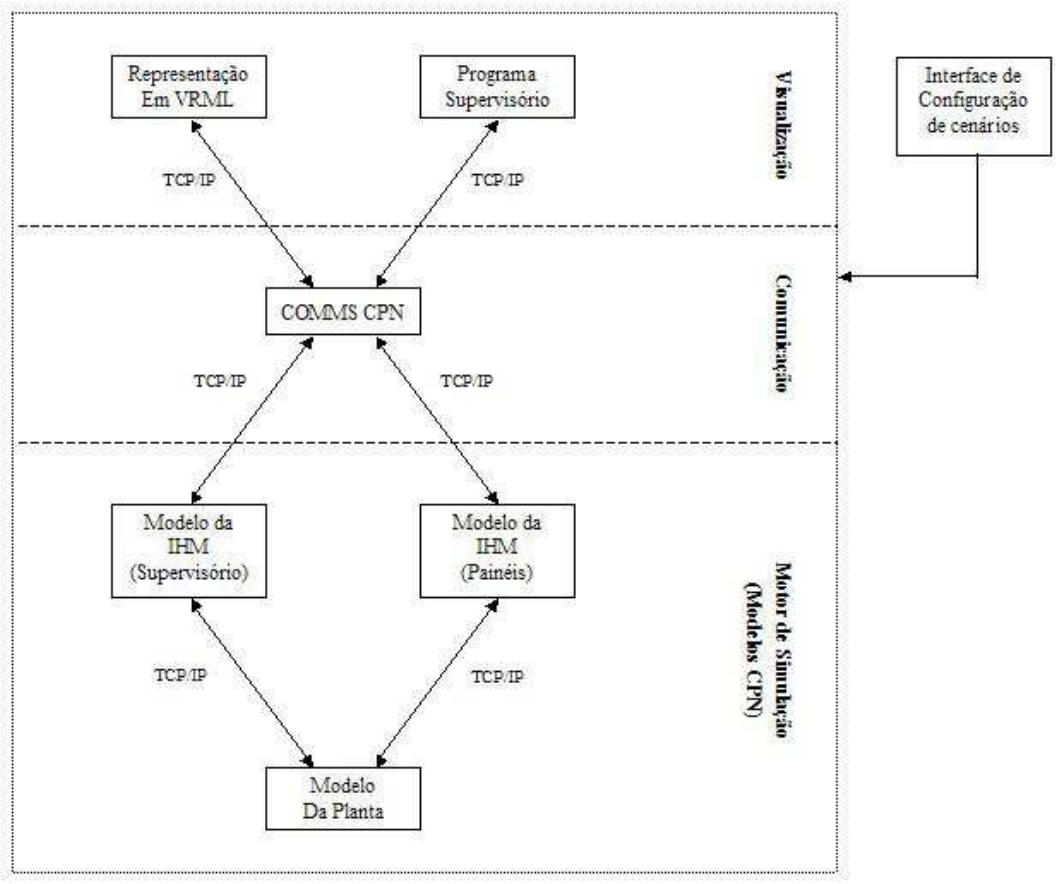


Figura 28: Arquitetura do Simulador

Figura 29: Sala de Operação de uma Subestação

2. Motor de Simulação (Modelos do Sistema)

O motor de simulação do simulador é baseado em três modelos em CPN que representam os três componentes do sistema: a planta industrial, a interface via programa supervisório e a interface via painéis de comando.

A idéia de se utilizar esse modelos é o de se minimizar o esforço de codificação do motor de simulação dado que já temos modelos verificados do comportamento do sistema, bem como eliminar possíveis erros de codificação.

A comunicação entre esses modelos e entre os mesmos e as representações em VRML e com o programa supervisório ocorre da seguinte forma:

- O operador seleciona uma opção em uma das representações (VRML ou programa supervisório) que acarreta o envio de uma mensagem para o modelo CPN correspondente (a representação em VRML se comunica com o modelo dos painéis e a representação no programa supervisório com o modelo do sinótico)
- A evolução do modelo da representação envia uma solicitação ao modelo da planta que irá executar o comando (ou não em caso de simulação de uma falha) e irá enviar uma mensagem para os modelos da interface
- Ambos modelos da interface irão evoluir de acordo com a mensagem recebida e enviarão uma mensagem para a representação da interface correspondente
- As representações irão ser atualizadas de acordo com as mensagens recebidas, finalizando assim a operação.

  A seguir temos uma descrição sucinta de cada um desses modelos.

2.1 – Modelo da Interface via Programa Supervisório

Este modelo é o modelo CPN apresentado neste trabalho de doutoramento, com o mesmo descrito no capítulo 3 deste texto.

2.2 – Modelo da Interface via Painéis de Comando

Na Figura 30 temos a página de hierarquia do modelo da interface via painéis de comando. Os elementos de interação neste modelo não são identificados pelo seu tipo genérico (disjuntor, seccionadora, etc), mas sim pelo seu tipo físico de operação (chave tipo punho, chave tipo GPG, etc). E, esses elementos são relacionados com os dispositivos genéricos

que controlam através da definição dos elementos das cores (tipos) associados a cada um desses elementos.

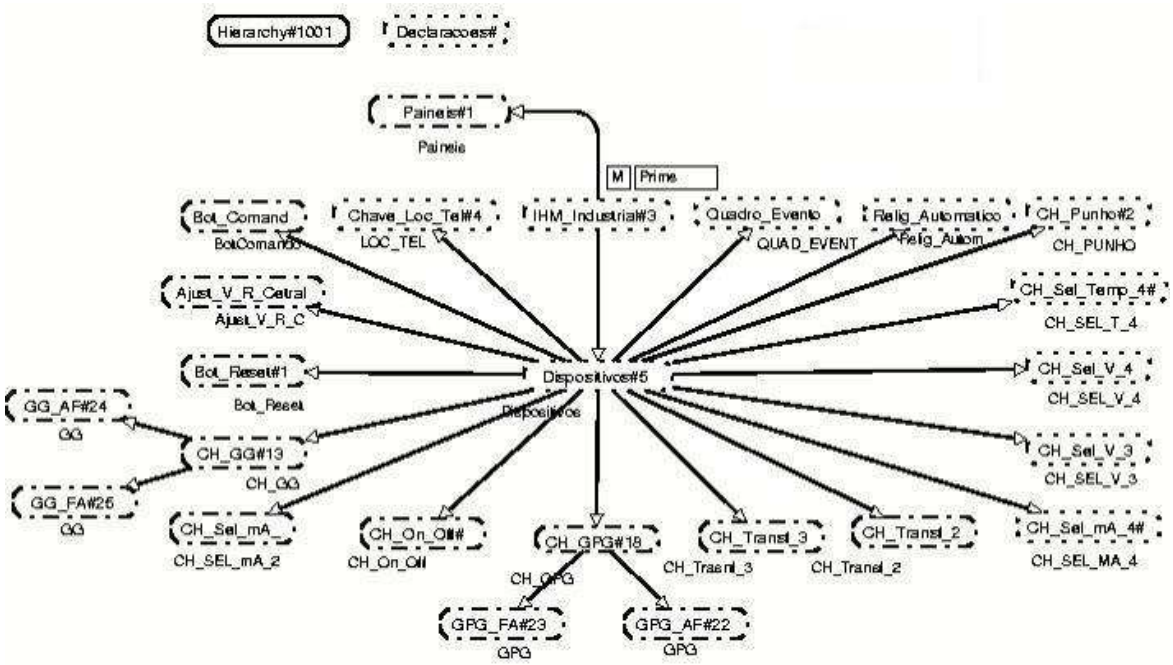


Figura 30: Interface de Supervisão de uma Subestação de Energia Elétrica

Como pode ser observado, temos uma grande quantidade de dispositivos de interação diferentes e não temos submodelos de autenticação de usuário ou de navegação, dado que aqui a operação é realizada diretamente nos painéis de comando através da abertura e fechamento manual dos dispositivos de interação.

Exemplificando os elementos de interação, na Figura 31 temos o modelo de uma chave tipo punho, que é uma chave de duas posições que pode ser utilizada para comandar disjuntores ou seccionadoras. E a descrição completa do modelo da planta está descrito em [63][65].

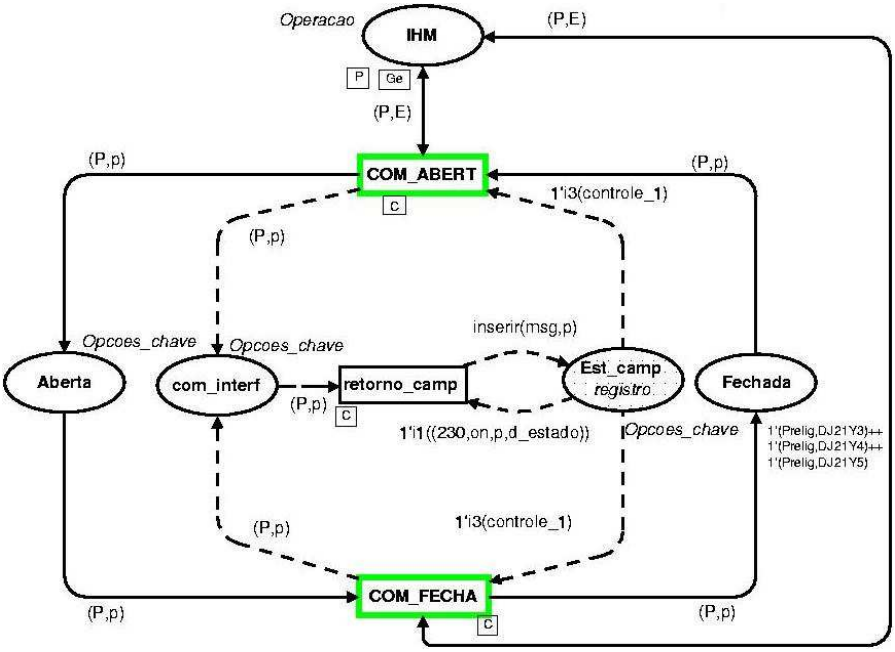


Figura 31: Modelo CPN de uma Chave do Tipo Punho

2.3 – Modelo da Planta Industrial

O modelo da planta representa os elementos físicos do sistema. Esses elementos são operados pela interface via supervisório ou painéis, e diretamente no caso de falhas do comando via interface. Na Figura 32 é apresentado o modelo dos dispositivos da planta. No estágio atual do projeto foram representados apenas os disjuntores e seccionadoras de um barramento de tensão da subestação.

Este modelo evolui sempre que recebe uma nova mensagem de um dos dois modelos da interface, retornando seu novo estado para os dois modelos, de modo aos dois ficarem em concordância com o estado do sistema, e os modelo da interface indicam seu novo estado para as representações do sistema de modo a realimentar o usuário com o novo estado do sistema.

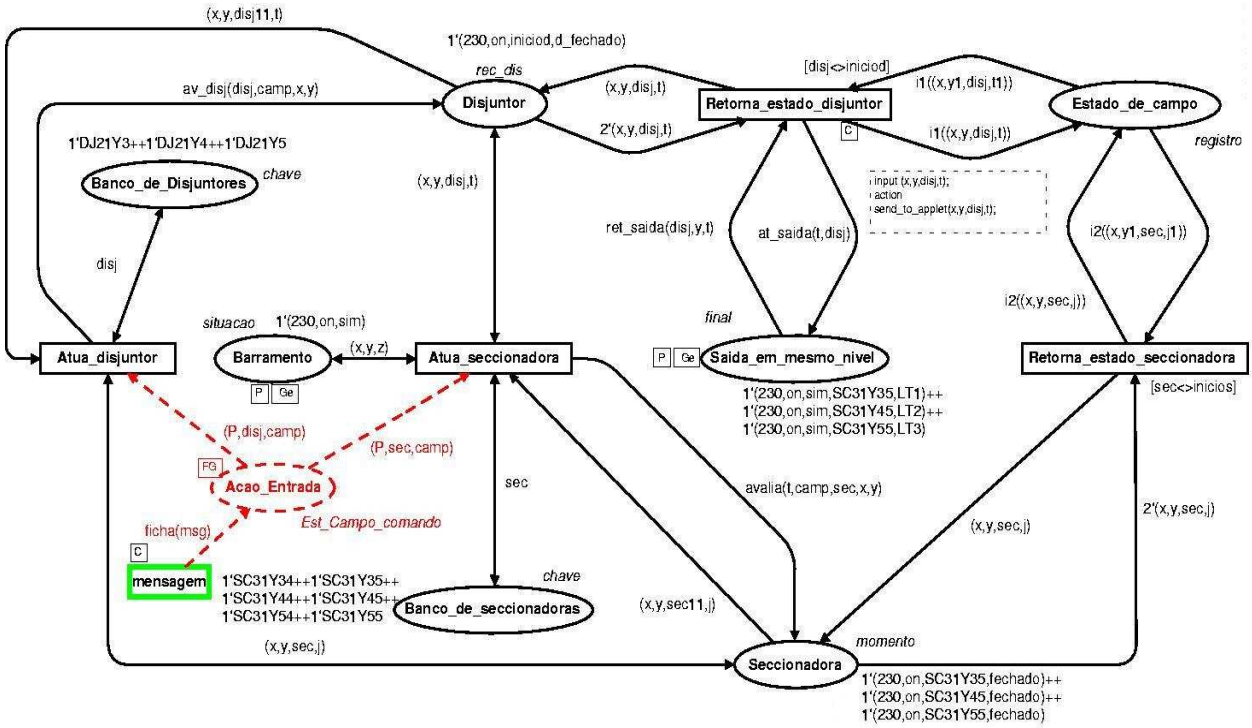


Figura 32: Modelo CPN dos Elementos da Planta Industrial

No estágio atual o modelo da planta não contempla as falhas dos equipamentos, com o modelo da planta sempre respondendo as ações dos usuários. Porém, os aspectos de falhas já estão sendo incorporados a esses modelos de modo a torna-los mais realístico.

3. Comunicação entre os Componentes

A comunicação entre os componentes ocorre através da troca de mensagens utilizando a família de protocolos TCP/IP. O uso do TCP/IP nos permite uma maior flexibilidade na construção dos componentes do simulador, que podem estar em máquinas diversas e não necessariamente sendo executados sob o mesmo sistema operacional.

A biblioteca COMMS/CPN [37] permite que um modelo CPN se comunique com um processo externo por meio de mensagens no padrão TCP/IP. O processo externo pode ser qualquer processo capaz de receber e enviar mensagens TCP/IP, o que inclui outros modelos CPN. A Figura 33 ilustra a arquitetura da biblioteca e sua relação com o Design/CPN e o protocolo TCP/IP. A COMMS/CPN é composta de três módulos, organizados em camadas [37]:

- Camada de Comunicação contém os mecanismos básicos de transporte do TCP/IP e todas as funções primitivas fundamentais relacionadas a sockets;

- Camada de Mensagens responsável pela conversão dos dados em fluxo de bytes para que haja a troca de mensagens entre Design/CPN e aplicações externas;
- 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.

Cada conexão possui um identificador único, a definição da porta TCP a ser utilizada na conexão, e a identificação do endereço IP do processo destino. Maiores informações acerca da biblioteca estão disponíveis em [34][37].

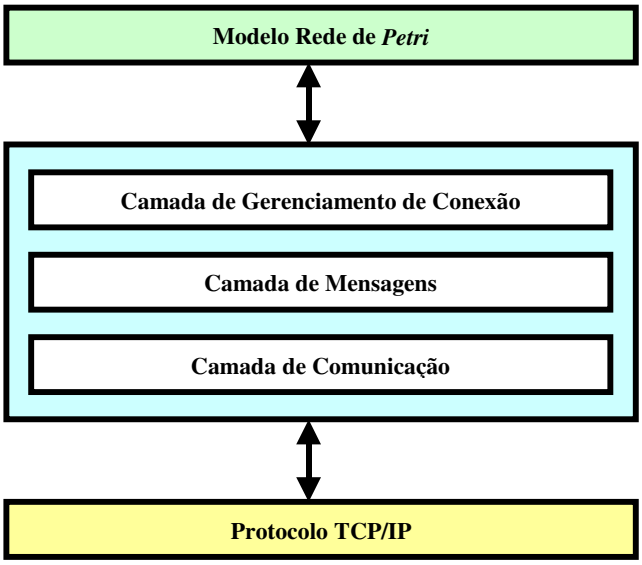


Figura 33: Arquitetura da Biblioteca COMMS/CPN

4. Interface do Simulador

Apresentamos agora nossa proposta para a interface do simulador baseado em modelos formais. Como pretendemos representar as duas forma de interação disponíveis aos operadores do sistema (painéis de comando e programa supervisório), se faz necessário definir a interface para cada um desses casos.

4.1 – Modelagem de Realidade Virtual Utilizando VRML

VRML é uma linguagem para modelagem de realidade virtual, voltada para a construção de ambientes tridimensionais. Trata-se de um formato de arquivo objeto 3D, análogo ao HTML [57]. Além disso, é independente de plataforma e permite a criação de ambientes virtuais através dos quais é possível navegar e visualizar objetos de ângulos diferentes e até

interagir com eles. Atualmente VRML é o padrão para o desenvolvimento de aplicações de realidade virtual multi-usuário [66].

Em virtude das características dessa linguagem: script executável, independência de plataforma, possibilidade de criação de ambiente multi-usuário, transmissão de dados e, sobretudo, por ser padrão para a modelagem de mundos virtuais, VRML foi escolhida como a linguagem para a modelagem da interface via painéis neste trabalho. O visualizador utilizado é o FreeWRL [55], um software de código aberto executado em sistema Linux.

Para a comunicação entre o modelo VRML e a biblioteca COMMS/CPN são utilizadas algumas classes JAVA que tratam as mensagens que são recebidas da COMMS/CPN, bem como das mensagens que são enviadas do modelo VRML para a COMMS/CPN. A descrição de como as classes JAVA e a COMMS/CPN se comunicam é apresentado em [32][33]. Na Figura 34 temos um exemplo da interface por meio de realidade virtual desenvolvida para o estudo de caso da subestação da Chesf.

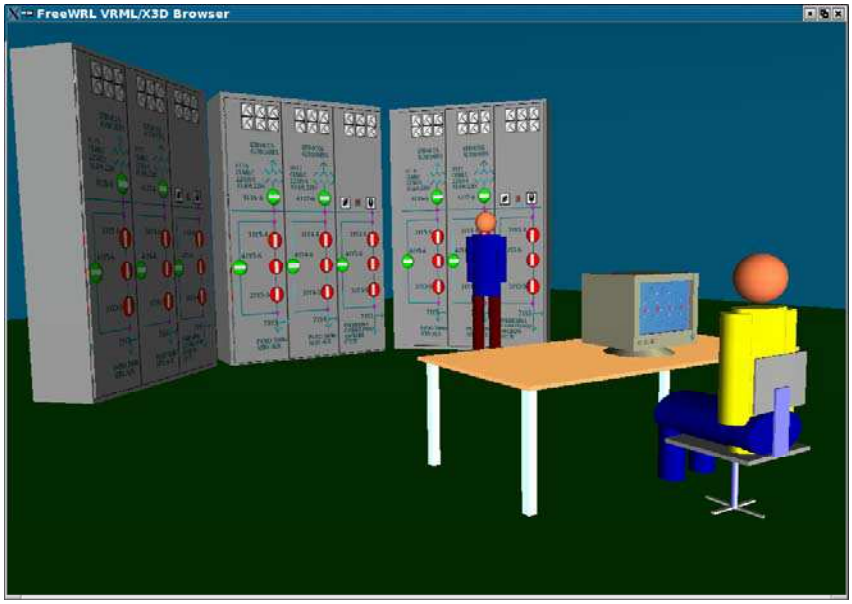


Figura 34: Visualização em VRML do sistema de operação de uma subestação

4.2 – Interface Utilizando Programa Supervisório

Para a interface do simulador ser o mais fiel possível com a realidade, um programa supervisório comercial será utilizado. Nele a interface será criada de acordo com a interface do programa supervisório utilizado no ambiente do estudo de caso, de modo a tornar o ambiente o mais fiel possível ao qual o operador está inserido. A Figura 35 apresenta um exemplo dessa classe de interface (interface do SAGE).

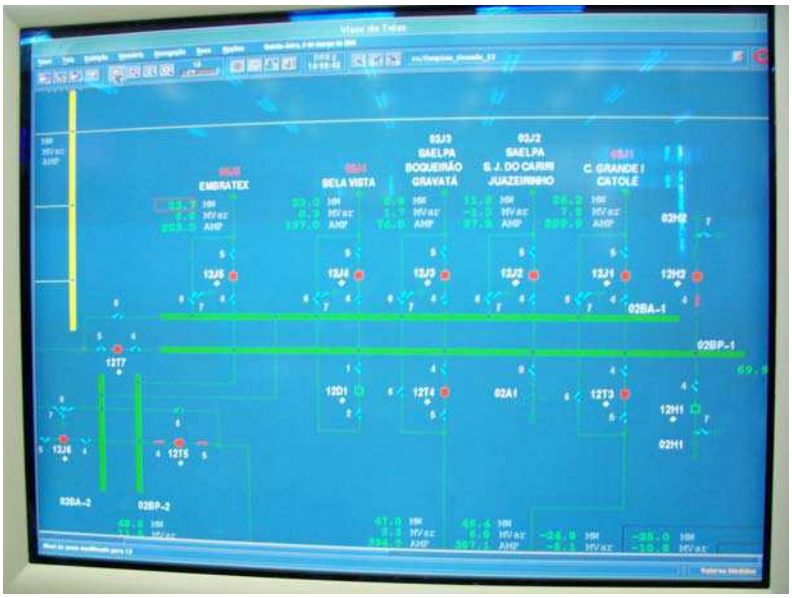


Figura 35: Interface de Supervisão de uma Subestação de Energia Elétrica

A comunicação com o modelo da interface em CPN se dá por meio da troca de mensagens utilizando o driver TCP/IP nativo do programa supervisório. A cada objeto de interação é associado uma chamada de envio de mensagem para o modelo que irá evoluir e retornar a execução (ou não) da ação, atualizando o estado do elemento de interação.

Vários programas supervisórios podem ser utilizados para a construção da interface. Neste trabalho os softwares escolhidos foram o Elipse Scada [25] e o Indusoft [46], os quais já possuem suporte nativo ao TCP/IP. A escolha pelo uso desses softwares é devido à disponibilidade dos mesmos para uso no GIHM.

5. Considerações sobre a Proposta do Simulador

Este anexo apresentou uma proposta de simulador de treinamento de operadores do sistema elétrico baseado em modelos formais. O uso de modelos formais nesse trabalho tem dois objetivos, o de modelar formalmente e verificar os componentes do sistema, e de utilizar esses componentes de modelagem já verificados como motor de simulação de um simulador de treinamento.

Podemos observar várias vantagens dessa abordagem, a saber:

- Menor tempo de desenvolvimento do simulador, dado que o motor de simulação já está pronto
- Menor probabilidade de erros na codificação do simulador, devido a codificação ser apenas na parte de visualização do sistema

- Modularidade do simulador, dado que cada modelo é independente, podendo um modelo ser modificado sem que seja necessário modificar os outros (desde que se mantenha os mesmos formatos de mensagens)
- Facilidade de comunicação, já que toda a comunicação é realizada via TCP/IP

E, para que esse trabalho atinja seus objetivos, temos ainda algumas questões a resolver:

- Incorporação dos elementos do sistema ainda não contemplados pelos três modelos
- Incluir nas visualizações do sistema os elementos não contemplados
- Realizar testes para verificar como o sistema se comporta em relação ao tempo de resposta se os modelos e as representações estiverem em hosts diferentes
- Incluir no modelo da planta as falhas que podem ocorrer nos dispositivos
- Padronizar o modo de comunicação (formato das mensagens) entre os diversos componentes para reduzir o processo de criação de mensagens e eliminar possíveis erros de elementos iguais com mensagens diferentes
- Ter a visualização da representação do supervisório e dos painéis no mesmo computador, dado que devido a cada uma das representações ser executada em um sistema operacional diferente se faz necessário dois computadores.

É necessária também uma interface para a definição do estado inicial do simulador, eliminando assim a necessidade de se editar os modelos CPN diretamente. Com isso, não se faz necessário que o avaliador da simulação tenha conhecimento do formalismo CPN, e sim apenas dos elementos do sistema e seus possíveis estados.

E, outra melhoria nesse trabalho seria modularizar os componentes dos modelos do sistema e da representação, criando assim um repositório de elementos do simulador. Isso tornará mais fácil o uso desses modelos como um arcabouço para a construção de simuladores para outras instalações de controle e supervisão do sistema elétrico. Para a construção do simulador para um outro contexto seria necessário utilizar uma técnica de recuperação de modelos, como a apresentada em [89][90], facilitando a composição do mesmo.