Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática Coordenação de Pós-Graduação em Ciência da Computação

Geração Automática de *Script*s de Testes em Ambiente 61850

Alan de Farias Cruz

Campina Grande, Paraíba, Brasil

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Geração Automática de *Script*s de Testes em Ambiente 61850

Alan de Farias Cruz

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Jorge Cesar Abrantes de Figueiredo
(Orientador)
Jacques Philippe Sauvé
(Co-Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

C957g Cruz, Alan de Farias.
Geração automática de scripts de testes em ambiente 61850 / Alan de Farias Cruz. – Campina Grande, 2011.
108 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Prof. Dr. Jorge Cesar Abrantes de Figueiredo, Prof. Dr. Jacques Philippe Sauvé.
Referências.

1. Geração Automática - Testes. 2. Testes. 3. MBT. 4. 61850. I. Título.

CDU 004.415.538(043)

"GERAÇÃO AUTOMÁTICA DE SCRIPTS DE TESTES EM AMBIENTE 61850"

ALAN DE FARIAS CRUZ

DISSERTAÇÃO APROVADA EM 30/11/2011

JORGE CESAR ABRANTES DE FIGUEIREDO, D.Sc
Orientador(a)

JACQUES PHILIPPE SAUVÉ, Ph.D
Orientador(a)

WILKERSON DE LUCENA ANDRADE, D.Sc
Examinador(a)

EDMAR CANDEIA GURJÃO, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

RESUMO

A norma IEC 61850 é o novo protocolo de comunicação para subestações elétricas, padronizando a troca de mensagens entre dispositivos eletrônicos inteligentes (*Intelligent Electronic Device* – IED). O meio de propagação de valores deixou de ser analógico (usando sinais elétricos) para ser digital (usando informações digitais: comandos, mensangens, etc), supervisionadas e controladas por um sistema de automação de subestação (SAS).

A norma definiu um modelo de comunicação entre os IEDs para permitir que equipamentos de diferentes fabricantes possam trocar informação; sem a necessidade de um conversor de protocolos.

Gerar os testes que contemplem a comunicação dos IEDs envolvidos numa subestação não é uma atividade trivial. Sistemas de automação de subestações possuem uma natureza concorrente – quando existem dois ou mais processos executados simultaneamente que disputam recursos do sistema. As diversas situações que podem ocorrer precisam ser testadas para garantir que o desempenho dos IEDs seja satisfatório.

Foi desenvolvida uma estratégia baseada em *Model-Based Testing* de geração de casos de testes, a qual é aplicada em ambientes que utilizam a norma IEC 61850 baseando-se na especificação de uma subestação elétrica e critérios de seleção de casos de testes.

Os casos de testes gerados contemplam ações desempenhadas pelos nós lógicos (*Logical Node - LN*) descritos na topologia de forma integrada. O estudo de caso utilizado para estes experimentos contém os principais nós lógicos (*Logical Node* – LN) envolvidos numa subestação 61850.

O número de casos de testes gerados depende da atual configuração da subestação. Para testes em LN isolados, o número de casos de testes varia de 4 até 30, e em componentes integrados, o número de casos de testes pode chegar a 500.000 ou mais, exigindo uma ferramenta de seleção de casos de testes que permita estabelecer um critério de seleção.

ABSTRACT

The IEC 61850 is the new substation communication protocol; it provides standard input/output messages between intelligent Electronic Device – (IED). The electrical signals from the substation equipments were replaced by digital information, controlled and managed by a substation automation System – SAS.

The 61850 protocol defines a communication model to allow IEDs from different vendors to interact with each other without a protocol conversor.

To generate tests that contemplate the communication involving IEDs is a non trivial task. Substation automation Systems have a concurrent nature – messages can be sent or received at the same time. The many configurations that can occur need to be tested to grant a satisfying performance.

It was created a technique based on Model-Based Testing for test generation used in environments using the 61850 standard, based on substation specification and tests selection criteria.

The generated test cases contain actions executed by the LN described in topology in an integrated approach. The case study used for these experiments contains the main LN involved in a IEC 61850 substation.

The number of generated test cases depends on the actual configuration of the substation. For isolated components, the test case number varies from 4 to 30.000, and using a integrated approach, the number can reach over 500.000, thus needing a tool for test selection criteria.

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado esta oportunidade. A toda minha família, a minha mãe que teve que agüentar todas as minhas horas de estudo, meu pai pela força e pela experiência, minha irmã Sara pela calma.

Agradeço a minha namorada Débora por ter me apoiado nos momentos difíceis. A meus amigos e colegas de trabalho, que estiveram ao meu lado quando precisei: nos momentos de estudo ao fazer reuniões e nos momentos de lazer para recuperar o fôlego.

Agradeço a meus orientadores Jorge Abrantes e Jacques Sauvé pela dedicação e apreço pelo meu trabalho, a professora Patrícia Machado e ao professor Edmar Gurjão pela grande contribuição, e a todos que estiveram envolvidos nesta importante etapa da minha vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Código Fonte 2.1 – Estrutura do formato Aldebaran baseado na Figura 2.4 19
Código Fonte 2.2– Rede de Petri em formato Maude do LN TCTR 35

Código Fonte 5.1 – Especificação do estudo de caso 64
Código Fonte 5.2 – Configuração do SAS 64
Código Fonte 5. 3 – Propósito de Teste utilizado 65
Código Fonte 5.4 – Script de Teste escolhido I 66
Código Fonte 5.5 – Script de Teste escolhido II 67

Lista de Siglas

SAS – Sistemas de automação de subestação

IED – Intelligent Electronic Device

LN – Logical Node

TCTR – Transformador de corrente

XCBR – Disjuntor

PDIF – Proteção Diferencial

CSWI – Chave seccionadora

IHMI – Interface homem-máquina

MBT – Model-Based Testing

TGV – Test Generation and Verification

Capítulo 1

Introdução

1.1 Contextualização e Motivação

Um sistema de transmissão de energia elétrica é composto por um conjunto de equipamentos interligados que desempenham funções de transmissão, conversão, medição e controle de energia elétrica. Esses equipamentos são agrupados em subestações elétricas, funcionando como ponto de apoio, interligadas por linhas de transmissão.

Para permitir que a transmissão de energia elétrica ocorra de forma segura e com confiança, essas subestações elétricas precisam de Sistemas de Automação de Subestação (SAS) e de ligações redundantes. A redundância garante segurança, porém aumenta a complexidade dessas subestações, ao passo que os sistemas de automação realizam funções de controle, medição, proteção, geração de *logs*, controle de processos automáticos, monitoramento, transformação e apresentação por meio de interface gráfica.

As antigas subestações elétricas contavam com a distribuição dessas funções divididas em relés analógicos, cada um desempenhando funções específicas. Com o aparecimento dos dispositivos microprocessados, foi possível substituir os relés analógicos por dispositivos que possam realizar as funções de um SAS agregando em um único *hardware* funções de proteção, controle e monitoramento do sistema (ZANIRATO, 2008).

O termo dispositivo eletrônico inteligente (*Intelligent Electronic Device* – IED) refere-se à dispositivos controladores microprocessados, de sistemas de potência, capazes de receber e enviar sinais de dados ou controle para outros IED, ou equipamentos externos (por exemplo, medidores elétricos de multifunção, relés digitais, controladores) segundo a norma IEC 61850-1 (2003). Os IEDs concentram, num único dispositivo, diferentes funcionalidades: proteção; controle, além de monitoramento e comunicação, de forma

isolada ou distribuída (PAULINO, 2009). Eles são instalados em subestações para coletar dados, permitir a proteção automática de equipamentos da subestação e funcionam de forma integrada para permitir a troca de informação entre os IEDs e a estação de controle segundo (GUPTA, 2008).

A integração entre os IEDs não é trivial, alguns problemas podem ocorrer: os IED podem enviar sinais errados para outros IEDs, ou ainda, pode haver atrasos na comunicação e erros na leitura de sinais. Para contornar esses problemas de comunicação foram criados diversos protocolos para padronizar esse controle de comunicação entre IEDs, mas nenhum garantia a interoperabilidade entre IEDs fornecidos por diferentes fabricantes (GUPTA, 2008).

A norma IEC 61850 foi criada para padronizar a comunicação entre IEDs estabelecendo o conceito de Nós Lógicos (*Logical Node – LN*). Cada LN desempenha uma função específica em um SAS. Na primeira versão da norma IEC 61850 (2003 – 2004) foram criados 91 tipos de LNs que podem ser integrados para desempenhar as funções de um SAS.

A partir da necessidade de padronizar a comunicação entre IEDs e garantir a interoperabilidade entre os fabricantes, foi criado o padrão IEC 61850, o qual veio a avalizar o aproveitamento completo do potencial da automação de subestações. Segundo Dawidczak, Oliveira e Pereira (2007) a norma especifica um conjunto de protocolos e incorpora novas tecnologias de comunicação, promovendo a interoperabilidade de IED de diferentes fabricantes sem a necessidade de conversores de protocolos, usando uma abordagem baseada em *software*: a troca de sinais analógicos por meio de cabos condutores será substituída por um canal de comunicação compartilhado utilizando redes de alta velocidade, onde os IED enviam e recebem dados com mais eficiência.

Desde 2004, a IEC 61850 se tornou a norma internacional para comunicação em SAS de modo que diversas indústrias e concessionárias de energia elétrica passaram a usá-la no projeto e implementação de subestações (DAWIDCZAK; OLIVEIRA; PEREIRA, 2007).

Cada fabricante desenvolve seus IEDs seguindo um padrão de implementação da norma IEC 61850. Para garantir que as funções específicas implementadas nos IEDs estão corretas, são realizados testes de unidade que comprovam as funcionalidades, garantindo que seu comportamento individual funcione como esperado.

Para garantir que um SAS que utilize a norma IEC 61850 esteja dentro dos padrões e que a comunicação de IEDs ocorra corretamente, devem ser realizados testes entre eles. Porém, o número desses componentes (IEDs) e as redundâncias existentes para garantir proteção e confiabilidade, tornam o desenvolvimento de testes uma atividade complexa. Devem-se gerar testes que abordem a integração entre os IEDs envolvidos num SAS para verificar se as funções estão com seu comportamento correto. Outro problema detectado é a inviabilidade de se parar uma subestação inteira para se realizar testes, prejudicando a transmissão de energia elétrica.

Como a norma IEC 61850 é uma solução baseada em *software*, isso permite aplicar técnicas da ciência da computação para resolver o problema de gerar testes relevantes para SAS que implementem esse padrão de comunicação. Para se testar um SAS sem parar uma subestação, os testes devem ser aplicados a um modelo que represente o SAS desejado. Gerar casos de testes a partir de um modelo é uma abordagem chamada *Model Based-Testing – MBT* (LEGEARD; UTTING, 2007).

1.2 Objetivos

O principal objetivo deste trabalho é a definição de uma técnica de geração de *script* de testes baseando-se em MBT aplicada à norma IEC 61850. Para atingir esse objetivo primário, devem-se considerar alguns objetivos secundários:

- Definição de um conjunto de Modelos: MBT exige que seja criado um modelo do sistema a ser testado. O sistema a ser testado é um SAS (formado por um conjunto de LN). Dependendo da topologia da subestação, diferentes LN precisam ser integrados, daí a necessidade da criação de um conjunto de modelos, que serão integrados para representar o SAS completo.
- Implementação de um protótipo: Para validar a técnica proposta será implementado um protótipo que irá gerar *scripts* de testes baseando-se na especificação de um SAS e cenários de teste como prova de conceito

1.3 Resultados

Foi desenvolvida uma técnica, baseada em MBT, que gera automaticamente casos de testes aplicados a um SAS 61850. Essa abordagem permite criar casos de testes baseando-se em especificações de SAS que contemplem as interações entre os LN envolvidos.

A automação desse processo permite gerar *scripts* de testes que descrevem a integração de LN por meio de um protótipo automático que concretiza a técnica com ferramentas de MBT e protótipos de adaptação.

Dependendo da especificação utilizada, o número de casos de testes obtidos é muito grande, isso é uma característica de técnicas de geração baseadas em MBT e exige a utilização de técnicas de seleção de casos de testes. Todos os testes foram realizados usando como estudo de caso um SAS 61850 que contém os principais LN para seu funcionamento.

A técnica mostrou gerar casos de testes que contemplam ações executadas por diferentes LN, mantendo uma ordem de execução e variando possíveis combinações de ações que podem ocorrer, devido à natureza complexa das comunicações entre LN de um SAS.

Para validar os resultados foi definido um conjunto de métricas. O cenário de teste do SAS a ser testado, tempo de execução, e o tamanho do modelo utilizado. O cenário de teste irá descrever a atual configuração do SAS a ser testado, isso inclui, por exemplo, a quantidade de valores lidos por um transformador, o atual estado de disjuntores, etc. O tempo de execução que mostra o tempo gasto para se obter os scripts de testes, quanto menor o tempo mais eficiente é a técnica. O tamanho do modelo indica a complexidade do SAS a ser testado, essa complexidade aumenta com o aparecimento de ações concorrentes ou que dependem da execução de outras ações.

Utilizando um cenário de teste que contém oito LN e um típico cenário de testes que contém eventos concorrentes, foram escolhidos dez casos de testes de um conjunto de 500.000 gerados em 103 segundos de execução utilizando um protótipo de seleção de casos de testes.

1.4 Estrutura da Dissertação

Este documento está dividido da seguinte forma:

Capítulo dois: Fundamentação teórica: Serão apresentados todos os conceitos relativos à geração automática de testes e da norma IEC 61850, bem como para a definição das ferramentas e formalismos adotados, fornecendo embasamento teórico ao leitor;

Capítulo três: Neste capítulo será apresentada uma visão de alto nível da técnica proposta, uma visão arquitetural que mostra os componentes instanciados para concretizá-la e os artefatos gerados;

Capítulo quatro: Neste capítulo será apresentado o protótipo que executa a técnica proposta;

Capítulo cinco: Este capítulo de avaliação mostra, com detalhes, o estudo de caso realizado e apresenta uma análise dos resultados obtidos;

Capítulo seis: Este capítulo mostra os trabalhos relacionados nas áreas pesquisadas: MBT e testes em 61850; e

Capítulo sete: No último capítulo será apresentada a conclusão.

Capítulo 2

Fundamentação Teórica

Os conceitos utilizados neste trabalho foram divididos em três seções: MBT, norma 61850 e Redes de Petri. Essa divisão foi adotada para facilitar a visualização de conteúdos específicos de MBT e da norma IEC 61850; para complementar a fundamentação teórica, serão apresentados os principais conceitos de Redes de Petri abordados pela técnica proposta.

2.1 Testes Baseados em Modelos – MBT

Nesta seção estão definidos alguns conceitos relacionados aos testes baseados em modelos. Serão apresentados os conceitos de testes, testes de caixa-preta, geração manual de testes, o processo de geração automática usando MBT, o formalismo mais adotado nesta área e uma ferramenta de geração de casos de testes.

2.1.1 Testes

Um caso de teste segundo a IEEE (2004) é um conjunto de entradas, condições de execução e resultados esperados desenvolvidos para um objetivo comum, como executar um programa específico ou checar a concordância de requisitos. Um caso de teste contém um identificador, propósito, pré-condições, entradas, saídas esperadas, pós-condições e um histórico de execução (HETZEL; WILLIAM, 1988).

Um teste é o ato de exercitar um software por meio de casos de testes. Um teste possui duas metas distintas: encontrar falhas ou demonstrar uma execução correta segundo Hetzel e William (1988). Dependendo do objetivo a ser alcançado – seja correção de *bugs*, detecção de problemas ou checar a robustez de um *software* – a principal meta em se realizar testes não é necessariamente eliminar todos os problemas que um software pode

ter, e sim abordar e corrigir situações que podem causar um impacto negativo ao consumidor final.

Criar testes baseando-se apenas na especificação do sistema a ser testado (*System Under Test* – SUT) tem uma nomenclatura especial: Teste de Caixa Preta (*Black-box testing)*. Esse tipo de teste trata o sistema todo como uma caixa preta; o interesse em se aplicar este tipo de teste não esta relacionado à sua estrutura interna – como foi implementado – e sim à descrição do comportamento (LEGEARD; UTTING, 2007). Usam-se testes de caixa para criar testes funcionais a partir de modelos baseados em requisitos documentados.

2.1.2 Processo Manual de Criação de Testes

A criação de testes é uma etapa realizada manualmente pelo desenvolvedor de testes, baseando-se em requisitos informais obtidos por documentos do SUT, e um plano de testes que fornece uma visão em alto nível dos objetivos dos testes: quais aspectos serão testados; que tipo de estratégia deverá ser usada; quando os testes deverão ser realizados e quantos testes serão feitos.

A saída desta etapa é um documento, que descreve os casos de testes desejados. A descrição desses casos de testes pode ser em alto nível – detalhes sobre o sistema podem ser deixados de lado de acordo com a visão de quem vai testá-los. Porém, pela natureza manual do processo de criação de testes, não se garante uma cobertura sistemática das funcionalidades do SUT e exige-se muito tempo. Os elementos usados nos testes são mostrados na Figura 2.1 e serão utilizados nas Figura 2.2 e Figura 2.3 (LEGEARD; UTTING, 2007). O processo manual é mostrado por completo na Figura 2.2.

- Documento formal: Artefato produzido contendo informações obtidas usando uma metodologia e um padrão de formatação;
- Documento informal: Artefato obtido por reuniões sem formalismos;
- Ferramenta de Software: Utiliza dados de entrada e fornece uma saída cobrindo uma funcionalidade específica;
- Desenvolvedor de testes: Profissional que irá desenvolver testes com conhecimento sobre o sistema a ser testado;

- Programador: Especialista em desenvolvimento de softwares e manipulação de dados;
- Atividade manual: Atividade realizada manualmente;
- Atividade automatizada: Atividade automatizada por uma metodologia com auxílio ferramental;
- Integração com ferramenta: Requer utilização de uma ferramenta com auxílio de um programador;

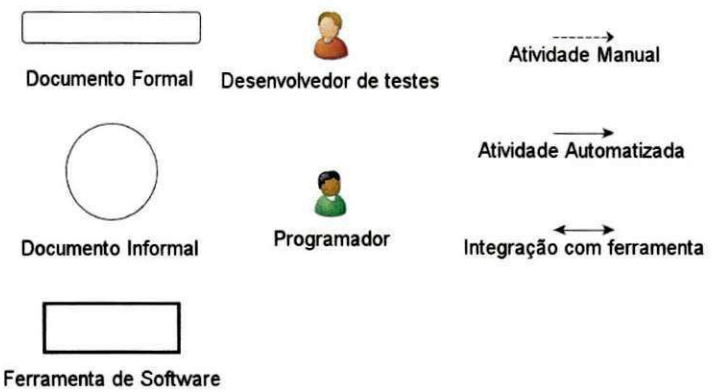


Figura 2.1 – Elementos de descrição dos processos de geração de casos de testes

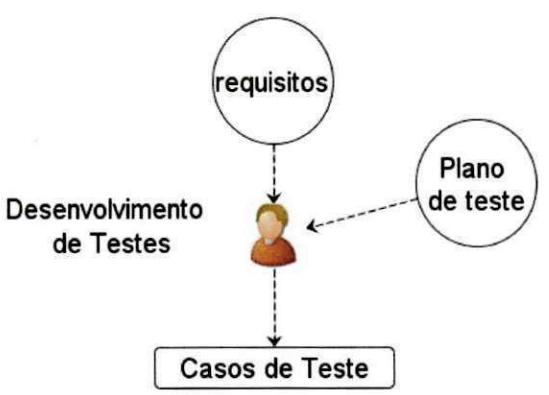


Figura 2.2 – O processo manual de criação de testes

2.1.3 Testes Baseados em Modelos (*Model-Based Testing* – MBT)

Segundo Legeard e Utting (2007) MBT é a automação do desenvolvimento de testes de caixa preta. Normalmente os testes de caixa preta são desenvolvidos manualmente baseando-se em requisitos documentados. Ao usar uma abordagem para automatizar a geração de casos de testes usando MBT, cria-se um modelo comportamental do SUT – detentor dos requisitos desejados – o qual será utilizado por uma ferramenta de geração automática de casos de teste, por exemplo: TGV por Jard e Jéron (2004) e TorX por Tretmans e Vries (2000).

O processo de geração de testes baseado em MBT é dividido em três etapas: Modelagem, Geração e Concretização, os quais possuem como entrada dois parâmetros: os requisitos do sistema e um plano de teste.

A Figura 2.3 mostra o processo de geração de testes usando MBT, indicando as entradas, as saídas e as três etapas básicas para a criação dos casos de testes: Modelagem do sistema real; Geração dos casos de testes – onde os casos de testes são gerados baseando-se no modelo utilizado – e a última etapa, que é concretizar esses casos de testes tornando-os executáveis.

A primeira etapa de MBT é escrever um esquema do sistema que se deseja testar, o qual é denominado "modelo abstrato". Esse modelo deve conter os aspectos mais importantes que se deseja testar e elimina características irrelevantes usando, como base, um documento informal que contém os requisitos do sistema (LEGEARD; UTTING, 2007).

A segunda etapa consiste na geração de casos de testes abstratos do modelo. A ferramenta de geração de casos de testes é guiada por um plano de testes, que define quais aspectos do SUT serão testados. Um exemplo de plano de testes é utilizar propósitos de teste (*Test Purposes* – TP). Um TP é uma descrição abstrata de um subconjunto da especificação do SUT, ou seja, mostra aspectos que devem ser testados.

A saída dessa etapa é um conjunto abstrato de testes, que são seqüências de operações do modelo (LEGEARD; UTTING, 2007). Como o modelo é uma versão simplificada do SUT, os casos de testes gerados não são diretamente executáveis.

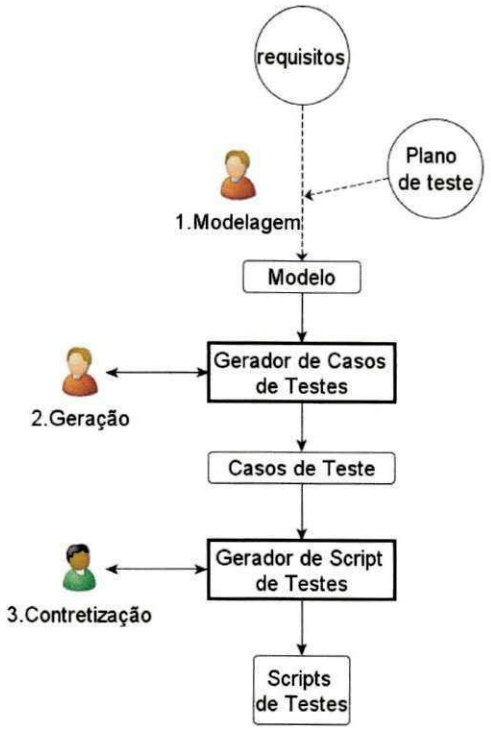


Figura 2.3 – O processo de geração de testes usando MBT

A terceira etapa consiste na transformação desses casos de testes abstratos em casos de testes concretos. Essa etapa pode ser realizada por uma ferramenta que possa preencher o vazio entre os casos de testes abstratos e os casos de testes concretos adicionando detalhes que não foram mencionados no modelo abstrato segundo (LEGEARD; UTTING, 2007)

A principal motivação de se aplicar MBT é a possibilidade de se trabalhar com sistemas que são difíceis de ser testados na realidade, seja pela complexidade dos componentes envolvidos ou pela demanda, a qual impede que uma subestação inteira seja parada para ser testada. Para contornar essa situação são criados modelos que cubram os requisitos obtidos do sistema real e, desses modelos, são criados testes relevantes que cubram as funcionalidades desejadas.

A vantagem de se usar MBT é a automação completa do processo de desenvolvimento de testes. A partir de um modelo representativo do mundo real, podem-se obter seqüências de testes completas que podem ser transformadas em *Script* de testes

(HETZEL; WILLIAM, 1988). A partir desse princípio, e levando-se em consideração o nível de abstração dos modelos, podem-se obter casos de testes usando ferramentas de geração de casos de testes como observado em Abdurazik e Offutt (2000), Cavarra et al. (2000), Abdurazik et al. (2003), Pataricca, Varro e Toth (2003) e Foster et al. (2008). O diferencial da técnica proposta é a utilização de MBT para geração de casos de testes para contemplar testes em ambientes que implementem a norma IEC 61850.

Uma limitação dessa técnica é que MBT apenas considera casos de testes funcionais como dito por Legeard e Utting (2007) e, em alguns casos, com a geração de testes de stress – que avaliam o desempenho de aplicações em situações de sobrecarga.

Outra limitação que pode ser apontada é pertinente ao modelo que representa o SUT. O que pode acontecer é que o modelo não represente o SUT de forma correta, gerando casos de testes irrelevantes. Para evitar modelos não representativos, deve ser realizado um treinamento com o profissional da área do SUT ou que um profissional valide o modelo gerado com o objetivo de representar o sistema em um formalismo que possa ser utilizado por uma ferramenta de MBT.

2.1.4 Sistemas de Transições Rotuladas (*Labelled Transition System – LTS)*

Sistemas de Transições Rotuladas (*Labelled Transitions Systems* – LTS) são grafos dirigidos nos quais os estados são configurações do sistema, ao passo que as arestas representam ações modificadoras da atual configuração do sistema. Um LTS é uma 4-tupla (S, Q, T, S_0) (TRETMANS; VRIES, 2000):

- *S*: Conjunto finito e não vazio de estados
- *Q*: Conjunto de ações
- *T*: Relação de transição
- S_0: Estado inicial

Esse tipo de formalismo é usado para definir semânticas de especificações comportamentais. Seguindo o modelo de grafos, os estados são configurações do sistema, as arestas representam movimentos dessas configurações a partir de ações e o primeiro estado é representado pelo estado que não possui estado anterior. A Figura 2.4 mostra um exemplo de um LTS que possui um estado inicial 0, um conjunto de estados (0,1,2,...,12) , um conjunto de transições (ACTION1, ACTION2,..., ACTION12) e as relações entre estados e transições.

Outro tipo de LTS possui além dos estados, rótulos – que podem ser divididos em entradas e saídas – que representam chegada e envio de dados. Esse LTS é chamado de IOLTS (*input-output Labelled Transition System)*. Os rótulos de entrada são marcados com um prefixo '?' e os rótulos de saída por um prefixo '!', um exemplo é mostrado na Figura 2. 5.

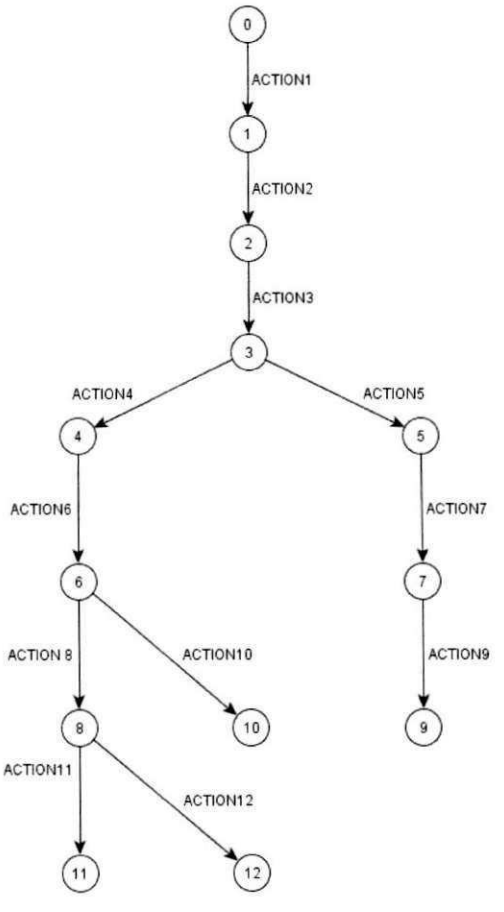


Figura 2.4 – Exemplo de um Sistema de Transição Rotulado (LTS)

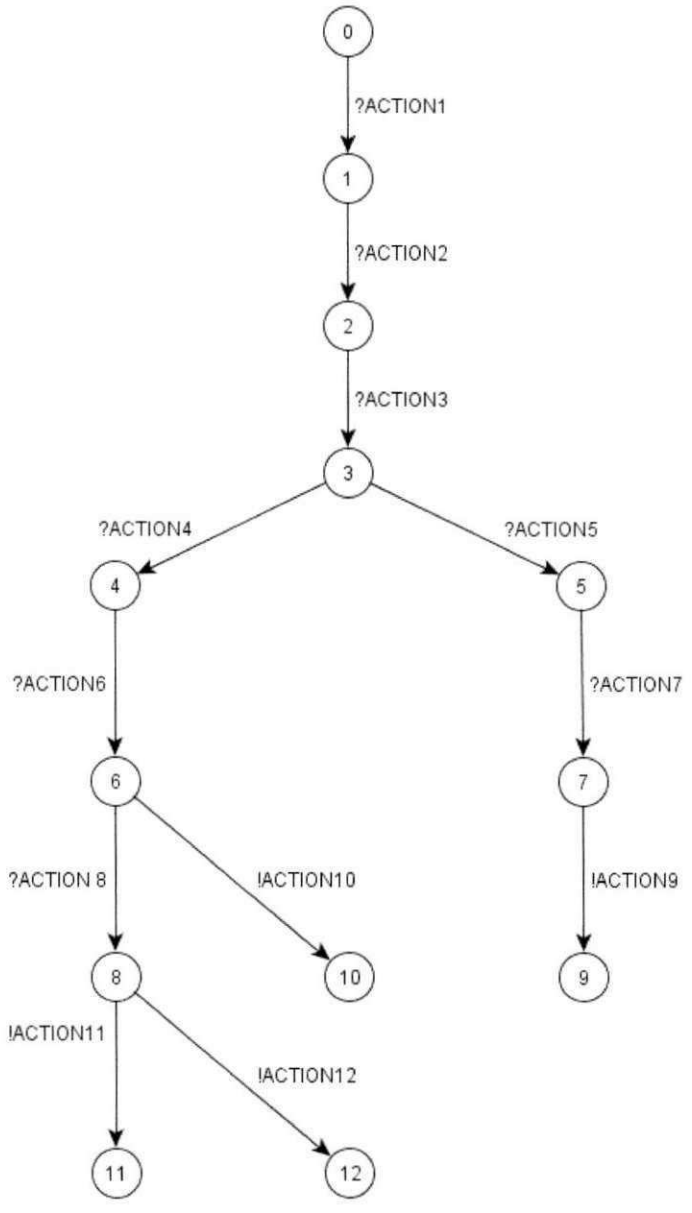


Figura 2. 5 – Exemplo de um IOLTS baseado na Figura 2.4

Em geral, os LTSs fornecem uma descrição global do conjunto de todos os comportamentos possíveis do sistema utilizado. Um caminho em um LTS pode ser considerado como uma seqüência de teste isso faz com que os modelos sejam facilmente testáveis (TRETMANS; VRIES, 2000).

2.1.5 TGV

Segundo Jard e Jéron (2004) TGV (*Test Generation and Verification*) é uma ferramenta que permite gerar automaticamente casos de testes diante da especificação formal de um sistema reativo não determinístico. Essa ferramenta utiliza sistemas de transição rotulados como formalismo base, distinguindo entradas, saídas e ações internas, baseando-se no conceito de relações de conformidade, mantendo as seguintes propriedades:

- *Soundness*: Os casos de testes rejeitam IUT que não possuem conformidade.
- *Exhaustiveness*: Todas as implementações que não possuem conformidade devem ser rejeitadas pela suíte de testes, dado um propósito de teste que satisfaça com que os casos de teste sejam gerados.

A ferramenta gera casos de testes abstratos a partir da especificação do SUT e um critério de seleção de testes – que pode ser, por exemplo: seleção de testes aleatória, seleção orientada a critérios de cobertura, por propósito de testes, ou uma combinação destes, para gerar casos de testes abstratos como mostra a Figura 2.7 que devem ser adaptados para o propósito definitivo.

Os testes gerados são produzidos em formato de grafos (Albebaran – AUT ou BCG – *Bynary Coded Graph*), podendo ser facilmente transpostos em uma linguagem específica para descrição. Esses testes precisam passar por ferramentas específicas, que dependem do SUT para serem convertidos em casos de testes executáveis.

TGV realiza operações de produto síncrono, determinação, suspensão e auxílio do propósito de testes para guiar a seleção de casos de testes obtendo, como resultado final um grafo completo de testes (Complete Test Graph – CTG). Esse grafo, como o próprio nome indica, contém todos os casos de testes contemplados pela especificação e que garantem as propriedades *sound* e *exhaustiveness* – com relação ao propósito (JARD; JÉRON, 2004).

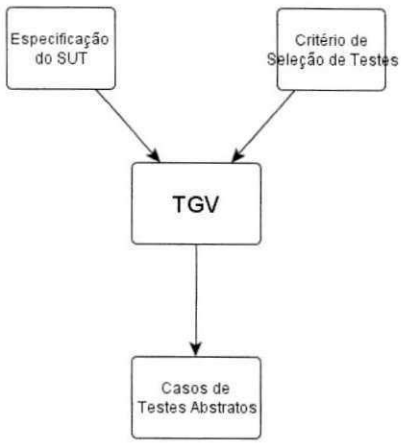


Figura 2.6 – Visão funcional da ferramenta (JARD; JÉRON, 2004)

O modelo de especificação utilizado por TGV é um sistema de transições rotuladas, onde os nós são estados atuais do sistema e as transições são ações que modificam esses estados, um exemplo de especificação é mostrado na Figura 2.7.

Esse exemplo mostra ações internas (que não interferem no sistema representado por 'τ'), as arestas com ações de entradas (que recebem valores do ambiente externo, representado por '? ') e as arestas com ações de saídas (que enviam valores para o ambiente externo, representado por '! ') seguidas por letras ou números. As arestas que não contém os símbolos '?' ou '!' são todas ações de saída.

Os estados são representados pelos nós {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. As arestas de especificações em LTS, que não demonstram ações internas, nem entradas e saídas são consideradas, implicitamente, como arestas de saída.

2.1.5.1 Estado de Parada e Autômato de Suspensão

Os testes possuem três tipos de paradas, podendo ser *deadlock* (quando o sistema não consegue se desenvolver), *outputlock* (quando o sistema fica parado esperando uma entrada) e o chamado *livelock* quando o sistema diverge diante de uma seqüência infinita de ações internas. Ao mostrar esses estados de parada tem-se o chamado autômato de suspensão – o exemplo mostrado na Figura 2.7, por não ter a descrição de entradas e saídas é considerado como o próprio autômato de suspensão.

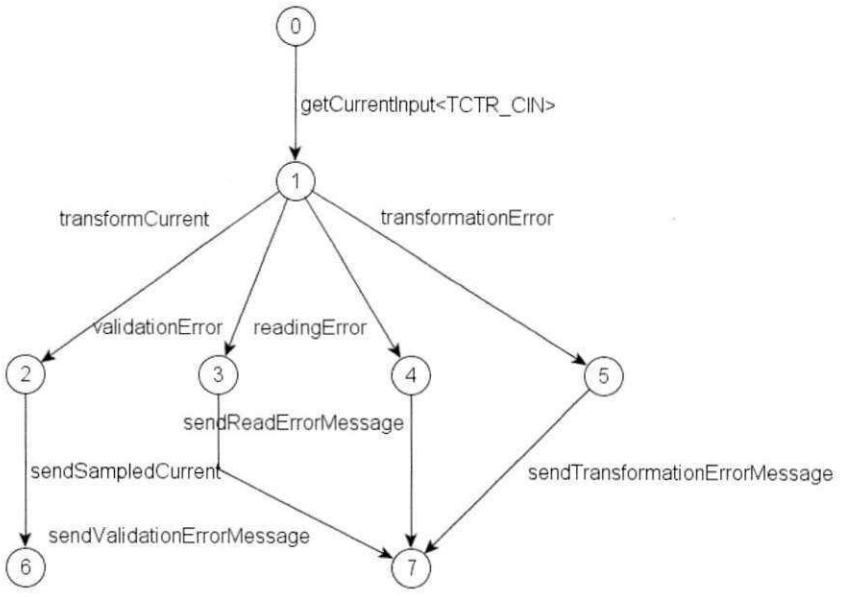


Figura 2.7 – Especificação em LTS para um transformador de corrente

2.1.5.2 Determinação

As arestas do autômato de suspensão representam o comportamento visível da especificação caracterizado pelas seqüências dos autômatos. A idéia é agrupar os meta-estados que levam a outros estados por ações internas num nó comum. A construção acaba quando não existirem meta estados para realizar a determinação. O resultado dessa determinação mostra os comportamentos visíveis da especificação.

2.1.5.3 Propósito de teste

Um propósito de teste (*Test Purpose* – TP) são descrições informais de comportamentos a serem testados, a maioria são sequencias incompletas de ações. Em TGV são modelados como autômatos que aceitam sequencias de ações da especificação (JARD; JÉRON, 2004).

Os propósitos de teste de TGV possuem estados de aceitação e rejeição, permitindo uma seleção de testes eficiente. Os estados de aceitação são usados para os objetivos a serem alcançados enquanto os de rejeição podam a exploração da do espaço de estados da especificação quando ações indesejadas são escolhidas. Um bom uso de estados de rejeição

pode reduzir drasticamente o custo de geração de testes. O exemplo da Figura 2.8 mostra um propósito de teste para a Figura 2.7.

2.1.5.4 Produto Síncrono (PS)

A síntese de testes usa a especificação S e um objetivo de teste TP. O resultado esperado é encontrar os estados de aceitação ou rejeição de S que satisfazem o objetivo do teste.

O produto síncrono irá marcar comportamentos da especificação utilizada com rótulos de aceitação ou rejeição. Precisamente, os comportamentos aceitos pelo produto síncrono são exatamente aqueles comportamentos da especificação que são aceitos baseando-se em um propósito de teste.

No exemplo abaixo o proósito irá aceitar qualquer ação '*' representada pela transição 0 a transição 1, seguido de ações que finalizem nos estados de aceitação 2, 3, 4 e 5.

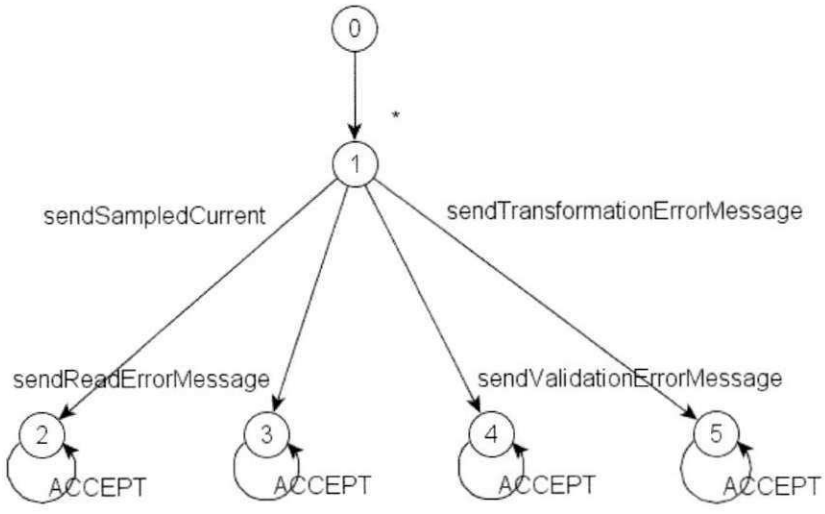


Figura 2.8 – Propósito de testes

2.1.5.5 Seleção de Testes

O IOLTS gerado por determinação e suspensão mostra todos os comportamentos visíveis da especificação utilizada, entre estes, os estados de aceitação ou rejeição pelo objetivo do teste. Para selecionar o caso de teste desejado devem-se escolher os comportamentos de aceitação.

Para isto, deve-se construir um CTG – *Complete Test Graph* (Grafo de testes completo) referente ao IOLTS, removendo caminhos que levam a um veredicto de rejeição e mantendo os caminhos que levam a estados de aceitação ou inconclusivos para a especificação utilizada (JARD; JÉRON, 2004).

TGV faz parte de um pacote de ferramentas do grupo CADP (2010). Como o CTG possui a representação de todos os casos de testes, a seleção é guiada por uma ferramenta do grupo de ferramentas CADP. A ferramenta de seleção realiza uma busca manualmente percorrendo o CTG, o caminho percorrido do nó raiz até um nó folha é considerado um caso de teste. Para a especificação da Figura 2.7 e o propósito de teste da Figura 2.8 podem-se obter quatro diferentes casos de testes, um deles é mostrado na Figura 2.9.

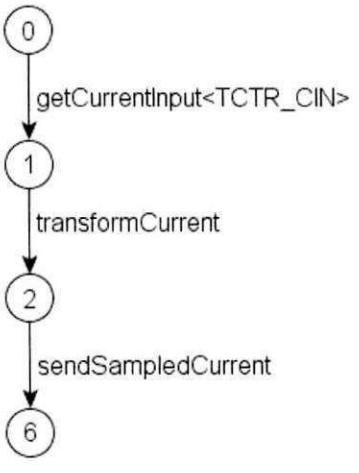


Figura 2.9 – Um possível caso de teste

2.1.6 Formato de Descrição de Grafos - Aldebaran

Um dos formatos utilizados na representação de grafos, que é utilizado na definição de LTS referentes ao espaço de estados e o propósito de testes é o formato de grafo Aldebaran – aut, como mostra o Código Fonte 2.1 'a' e ''b' e 'z' representam os rótulos das transições, X o número de transições, Y o número de estados. A primeira linha é indicada pelo rótulo 'des' (*Description*) que designa os três elementos do grafo (*Si, L, N*):

- S_i : Estado inicial – o estado que se deseja iniciar o propósito de teste
- ***L***: Número de transições – o número de transições existentes no grafo
- ***N***: Número de estados – o número de estados existentes no grafo

Todas as transições são descritas após a linha de descrição, seguindo o seguinte formato: (S_i, l, Sf):

- S_i: Indica o estado fonte da transição
- $l \in \mathbf{L}$: O rótulo da transição
- ***Sf***: Indica o estado destino da transição

```
des(0,12,13)
(0,ACTION1,1)
(1,ACTION2,2)
(2,ACTION3,3)
(3,ACTION4,4)
(3,ACTION5,5)
(4,ACTION6,6)
(5,ACTION7,7)
(6,ACTION8,8)
(6,ACTION10,10)
(7,ACTION9,)
(8,ACTION11,11)
(8,ACTION12,12)
```

Código Fonte 2.1 – Estrutura do formato Aldebaran baseado na Figura 2.4

Como mostra o Código Fonte 2.1, partindo do estado inicial 0, a partir da transição 'a' chega-se ao estado 1. E assim sucessivamente.

Para representar espaço de estados utiliza-se esse formato de descrição de grafos mostrado no Código Fonte 2.1. Para representar propósitos de teste utilizado por TGV esse formato exige dois estados marcados: aceitação – Definição de seqüências que serão incluídas nos casos de testes, marcado por um auto-laço rotulado com 'ACCEPT' – e rejeição – Definição de seqüências de que não serão incluídas nos casos de testes, marcado por um auto-laço rotulado com 'REFUSE' (CARTAXO; MACHADO; NETO, 2007).

2.2 A norma IEC 61850

Esta seção irá apresentar alguns conceitos necessários para o entendimento da norma IEC 61850, utilizados neste trabalho, a saber: subestação elétrica; sistema de automação de subestação; dispositivos eletrônicos inteligentes; nós lógicos, e dispositivos lógicos.

2.2.1 Subestação Elétrica

As subestações elétricas são instalações elétricas de alta potência responsáveis pela transmissão, distribuição, controle e proteção de energia elétrica. As subestações funcionam como centros de controle que são responsáveis pelo direcionamento do fluxo de energia elétrica por meio de linhas de transmissão. As principais funções e características de uma subestação elétrica, segundo Casazza e Delea (2003) são:

- Funcionam como um terminal de comunicação entre sistemas;
- Apresentam segmentação das linhas de transmissão, o que promove um certo grau de redundância no fluxo de transmissão;
- São locais onde as linhas de transmissão podem ser desenergizadas (quando ocorre a supressão de energia elétrica) tanto para a realização de manutenção ou em decorrência de problemas elétricos;
- Fornecem um local para equipamentos de proteção, controle e medição.

Uma subestação é composta por diversos equipamentos destinados à concretização das funções supramencionadas. Os mais comuns, e que são utilizados nos mais variados projetos de subestações, são descritos por Casazza e Delea (2003):

- Barramento: É a estrutura elétrica na qual todas as linhas e transformadores estão conectados.
- Relés de Proteção: São dispositivos que monitoram, continuamente, os potenciais elétricos e correntes associados à linha e seus terminais, visando à detecção de falhas ou problemas na linha ou equipamento. Essas falhas são chamadas de faltas e envolvem o contato entre fases ou entre uma ou mais fases e o aterramento. Os relés de proteção disparam disjuntores.
- Disjuntores: São dispositivos que interrompem o fluxo de eletricidade para isolar uma linha ou transformador. Isso é realizado com a abertura do circuito. Os disjuntores podem ser instalados em série com a linha ou transformador, ou podem ser instalados em ambos os lados da seção do barramento onde a linha se conecta. Permitem que as linhas individuais ou os transformadores possam ser removidos para manutenção do serviço, quando relés de proteção detectam condições adversas.
- Transformadores: São equipamentos que convertem níveis de tensão de corrrente. Como exemplo, um transformador pode receber em um dos seus terminais 138kV e produzir no outro 13kV. No contexto de proteção de sistemas elétricos temos os transformadores de corrente (TC) e transformadores de tensão (TP) que produzem uma versão com níveis reduzidos do sinal (corrente ou tensão) aplicado a um dos seus terminais.
- Chaves seccionadoras: São usadas para abrir um circuito quando é necessário interromper o seu funcionamento. São utilizadas para conectar ou desconectar disjuntores, barramentos ou transformadores que não estejam carregados.
- Pararraios: Protegem transformadores e aparelhagem da subestação dos efeitos de alta tensão, causados por raios ou operações de chaveamento.
- Equipamentos de medição: Fornecem uma linha de medição, carga de transformadores e potencias em barramentos para que os funcionários possam garantir que essas instalações funcionem dentro dos limites aceitáveis.

- SCADA: É um acrônimo para Supervisão de Controle e Aquisição de Dados – *Supervisory Control And Data Acquisition*. Reflete os melhoramentos em medição, telecomunicação e tecnologias de computação que permitem automação nas operações de subestações.

2.2.2 Sistema de Automação de Subestação (SAS)

A automação de subestações é utilizada para controle, proteção e monitoramento de subestações como dito por (BRAND, 2003). Um sistema de automação de subestação (*Substation Automation System* – SAS) possui três níveis de atuação: Subestação; Barramento (também chamado de *Bay* ou baia) e Processo.

Segundo Brand, Lohmann e Wimmer (2003) o nível de estação possui um computador central da subestação, junto com um terminal de interface e um *gateway* para a central de controle de rede; o nível de barramento possui as unidades de controle e proteção; e o nível de processo com as interfaces de conexão para os equipamentos da subestação. Todo o esquema é apresentado na Figura 2.10 (FIGUEIREDO, 2009).

2.2.3 IED

Os dispositivos eletrônicos inteligentes (*Intelligent Electronic Devices* – IED) segundo Gupta (2008) são utilizados na automação e proteção de subestações elétricas para coleta de dados e realizam a proteção automática dos equipamentos da subestação. Recebem as informações a partir de sensores e dos equipamentos da subestação. Realizam diversas funções dentre as quais: abertura de disjuntores em caso de anomalias na rede elétrica; envio de comandos para regular os níveis de tensão.

A comunicação de dados entre IEDs e locais remotos ou entre os próprios IEDs sem o uso da norma IEC 61850 exigia o uso de protocolos de comunicação específicos, exigindo conversores de protocolos para garantir a interoperabilidade, uma vez que os fabricantes desenvolviam seus componentes com diferentes padrões, dificultando a integração de equipamentos de uma subestação.

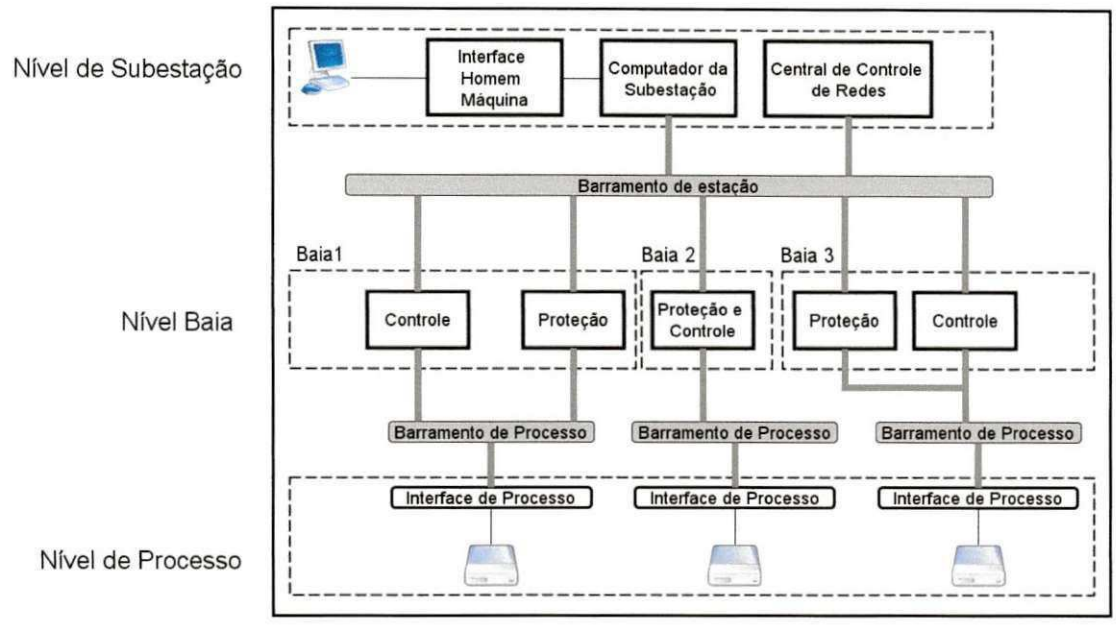


Figura 2.10 – Estrutura de um SAS (FIGUEIREDO, 2009)

Com o aparecimento da norma IEC 61850, foram criados IEDs que dão suporte a um novo padrão de comunicação permitindo a interoperabilidade entre IEDs de diferentes fabricantes.

2.2.4 SAS 61850

As funções desempenhadas pelo sistema de automação de subestação são em geral: controle de posicionamento; monitoramento de dados e proteção. A norma IEC 61850 fraciona essas funções em sub-funções, que são desempenhadas por IEDs instalados nas subestações (GUPTA, 2008). Para realizar uma função do SAS essas sub-funções são integradas e interagem mediante uma rede local de comunicação na subestação, sendo necessárias sintaxe e semântica específicas.

A norma 61850 define entidades chamadas de nós lógicos – a menor parte de uma função onde ocorre a troca de dados como dito em Alencar et al. (2009). Todas as possíveis sub-funções foram padronizadas na norma IEC 61850. Cada nó lógico (LN) contém objetos de dados que são as propriedades e valores que cada LN possui. Segundo IEC 61850-6, (2003), são 91 tipos diferentes de LN que diferem, entre si, por modelos, exemplo: proteção; de medição; controle; genéricos; controle automático e transformadores de

potência. O conjunto de nós lógicos forma um dispositivo lógico (*Logical Device* – LD) que é acessado por um dispositivo físico (IED) possuidor de um endereço de rede associado.

Um nó lógico é definido por uma classe de objeto, chamada de classe de nó lógico. Usando a nomenclatura própria da norma tem-se, por exemplo, que a classe XCBR monitora e opera um disjuntor. Cada classe possui um conjunto de dados que é formado por atributos.

A Figura 2.11 mostra o modelo de dados para dois LN (XCBR e CSWI), formando um dispositivo lógico (Relé1) o qual é armazenado em um IED. Cada LN contém um objeto de dados (ST e CO para XCBR1 e CO para CSWI1) e atributos associados para cada objeto de dados, tendo como referência o nome do LN. Por exemplo: CSWI1 representa uma chave seccionadora, possui um dado rotulado de controles (CO), o qual possui três atributos: pos A, pos B e pos C. utilizados como parâmetros para realização das funções.

Um esquema de comunicação de IED, LD e LN é mostrado na Figura 2.12. F1 e F2, representando funções da subestação, onde: LN0, LN1,..., LN6 são os LN; PD1, PD2 e PD3 são os dispositivos físicos – IED; e PC12, PC13 e PC 23 são as conexões físicas entre os IEDs; e LC12, LC14, LC56 etc são as conexões lógicas entre os LN.

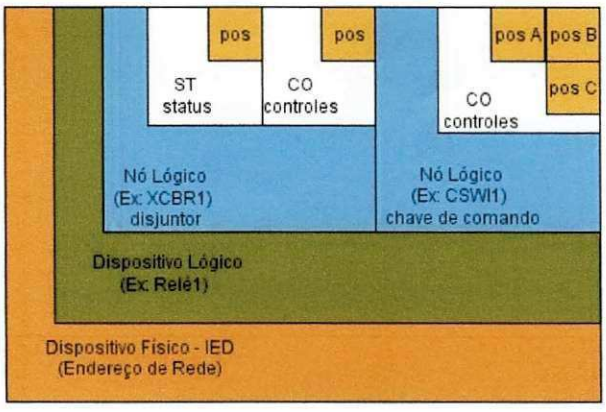


Figura 2.11 – Modelo de dados utilizado pela norma IEC 61850 (CARMO; PAULINO, 2007)

Esse esquema de comunicação é descrito por uma linguagem de configuração de subestação (*Substation Configuration Language* – SCL) definida pela norma IEC 61850 para alcançar a interoperabilidade de configuração e automatização do processo de testes.

Em resumo, é necessária uma especificação formal, em linguagem SCL, para que se desenvolvam testes funcionais, acesso aos pontos de rede e LN envolvidos além da exigência de que haja uma especificação funcional do usuário para fornecer o comportamento funcional desejado para um SAS, (CARMO; PAULINO, 2007).

Um SAS que adota a norma IEC 61850 é formado por um barramento de estação, ligado a um *gateway* conectado à internet, isso permite conexão com um operador remoto, a central de controle ou outras subestações. Nesse mesmo barramento estão conectados: a interface homem máquina (*Human Machine Interface* – HMI) – para monitorar e controlar o SAS –; e os conjuntos de IEDs. Os Relés de proteção e equipamentos de medição são conectados ao barramento de estação e ao barramento de processo. Neste último barramento são conectadas as unidades de unificação (*Merging Units* – MU), as quais unificam os dados de corrente e de potencial elétrico. Todos os barramentos funcionam como um meio de comunicação compartilhado usando redes ethernet; um esquema de um SAS 61850 é mostrado na Figura 2.13.

O que faz um SAS atuar da forma esperada depende: da comunicação; da disposição dos IEDs e dos dispositivos que eles vão gerenciar (disjuntores, transformadores de corrente, etc.). Os IEDs devem realizar as funções específicas para seu funcionamento de forma integrada, enviando e recebendo mensagens de forma que não ocorram conflitos.

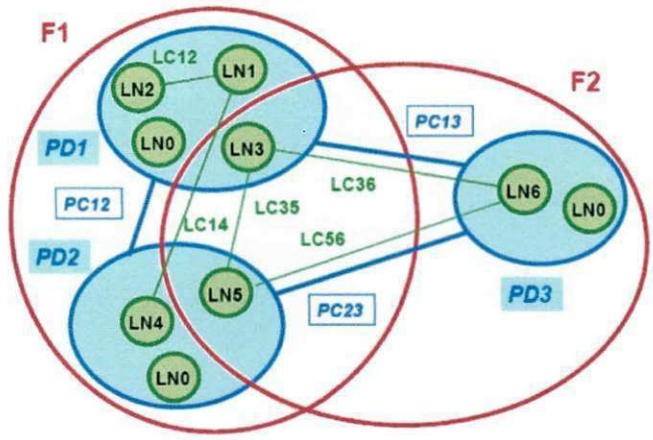


Figura 2.12 – Esquema de comunicação entre IEDs, LNs e LDs (CARMO; PAULINO, 2007)

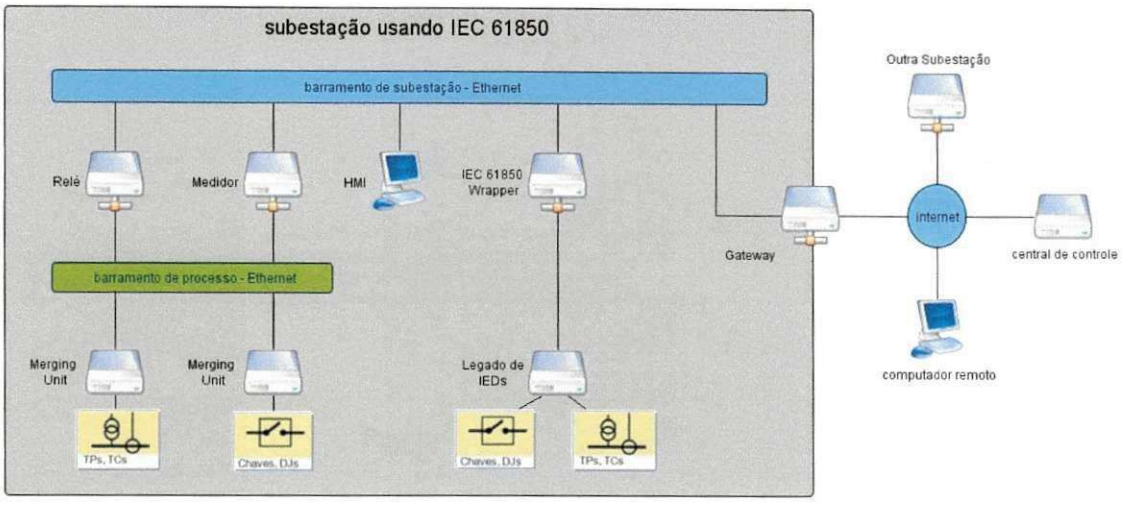


Figura 2.13 – SAS que utiliza a norma IEC 61850 baseada em (CARMO; PAULINO, 2007)

2.3 Redes de Petri

Pela sua ampla representatividade, as Redes de Petri podem ser aplicadas em diversas áreas ou sistemas que possam ser descritos graficamente por meio de uma estrutura que mostre o fluxo da aplicação – como *Flow Charts* – e que possa representar concorrência ou paralelismo nas atividades envolvidas.

Como descrito em Murata (1989) as Redes de Petri são recomendadas em avaliação de desempenho e protocolos de comunicação, e em algumas áreas promissoras como: modelagem e análise de sistemas de software distribuídos, sistemas de banco de dados distribuídos, programas concorrentes ou paralelos, sistemas discretos, sistemas de tolerância à falha, estruturas e circuitos assíncronos e programas lógicos.

2.3.1 Introdução

É uma ferramenta de modelagem gráfica e matemática para a descrição e estudos de sistemas de processamento de informação caracterizados como: concorrentes; distribuídos; paralelos; assíncronos; não-determinísticos e/ou estocásticos (MURATA, 1989).

Uma Rede de Petri é um caso particular de um grafo dirigido com um estado inicial chamado de marcação inicial, M_0, uma marcação (ficha) relaciona, para cada lugar 'p', um inteiro não-negativo 'k'; pode-se dizer, então, que um lugar 'p' é marcado com 'k' fichas (MURATA, 1989).

As principais características de um grafo de uma Rede de Petri é ser dirigido e bipartido, contendo dois tipos de nós, chamados de lugar e transição, cujos arcos partem de um lugar para transição ou de transição para um lugar (MURATA, 1989).

Uma marcação é representada, graficamente, por uma ficha em um estado que representa uma atual configuração do sistema. Os lugares são os estados que o sistema modelado se encontra e as transições representam as ações que ocasionam mudanças de estado. Algumas interpretações típicas para lugares e transições são mostradas na Tabela 2.1 (MURATA, 1989).

Lugar de Entrada	***Transição***	***Lugar de Saída***
Pré-Condições	Evento	Pós-Condições
Dados de entrada	Passo computacional	Dados de saída
Sinais de entrada	Processador de sinal	Sinais de saída
Recursos necessários	Tarefa	Recursos livres
Condições	Cláusula em lógica	Conclusões
Buffers	Processador	*Buffers*

Tabela 2.1 - Interpretações típicas de lugares e transições (MURATA, 1989)

Graficamente, um lugar é representado por um círculo, e as transições, por barras; ambos os componentes são conectados entre si por meio de arcos dirigidos, que podem ser únicos ou múltiplos, como mostra a Figura 2.14.

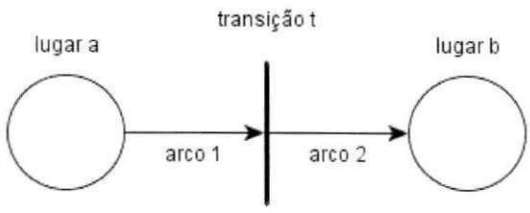


Figura 2.14 – Exemplo de Rede de Petri simples com lugares, transições e arcos

As redes de Petri possuem marcações (fichas) para indicar possíveis situações do sistema, permitindo a simulação de comportamentos dinâmicos, para isso, precisam de uma marcação inicial para ser executadas. Ao ativar uma transição '*t*' a ficha contida no estado origem é consumida e repassada para o estado destino, como mostram as Figura 2.15 e Figura 2.16.

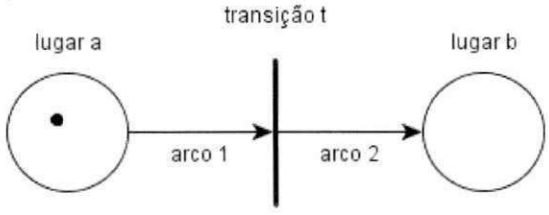


Figura 2.15 – Uma ficha no lugar '*a*' ativa a transição '*t*'

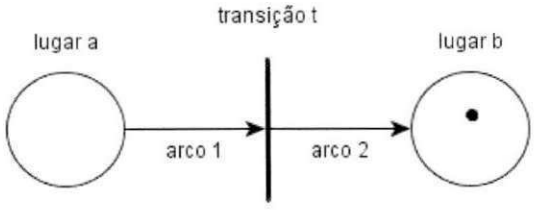


Figura 2.16 – Ao ativar a transição 't', a ficha do lugar 'a' vai para o lugar 'b'

Essa característica dinâmica das Redes de Petri permite visualizar qualquer configuração em que o modelo do sistema se esteja verificando quais transições foram ativadas e o estado atual dos lugares.

Segundo Murata (1989) as Redes de Petri são agrupadas em duas classes: Ordinárias e não-ordinárias, também chamadas de alto nível. Redes ordinárias possuem fichas do tipo inteiro não negativo, ao passo que as redes de alto nível possuem marcas de tipos específicos.As redes ordinárias se subdividem em:

- **Rede binária**: Redes que permitem uma ficha em cada lugar, e arcos com valor unitário;
- **Rede Lugar-Transição**: Tipo de rede que permite o acúmulo de marcas no mesmo lugar.

2.3.2 Redes Básicas

As redes mostradas nesta seção podem ser estendidas ou combinadas, formando redes mais complexas. Serão mostradas redes que representam: seqüenciamento, distribuição, junção e escolha não-determinística.

2.3.2.1 Seqüenciamento

Representa a execução a partir de uma ação, satisfazendo uma determinada condição. Ao término da ação, pode-se ter outra, desde que satisfaça uma condição específica. Um exemplo de rede de seqüenciamento pode ser observando na Figura 2.16. O lugar 'b' só pode ser alcançado se a condição do lugar 'a' for satisfeita, ativando a transição 't'.

2.3.2.2 Distribuição

Quando ocorre a criação de estados paralelos a partir de um local em comum, chamado de pai. A Figura 2.17 mostra os processos filhos (lugares b e c) criados pela distribuição de fichas encontradas no processo pai (lugar a) a partir da transição t.

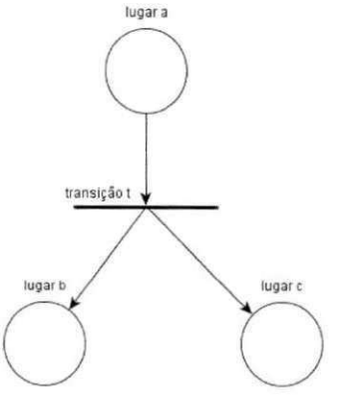


Figura 2.17 – Rede de distribuição

2.3.2.3 Junção

É o que ocorre quando dois processos concorrentes precisam ser sincronizados. A Figura 2.18 mostra dois processos (lugares 'a' e 'b') que são sincronizados pela transição 't', a qual só é ativada se existirem *tokens* nos lugares 'b' e 'c' levando ao lugar 'c', que representa o sincronismo entre os processos.

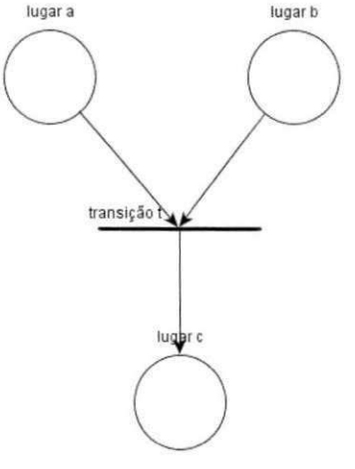


Figura 2.18 – Rede de Junção

2.3.2.4 Escolha não-Determinística

Esse tipo de rede inibe uma transição se outra for disparada, sem possibilidade de escolha. O fator não-determinístico gera uma situação chamada de conflito que está associado ao fato de as duas transições partirem do mesmo lugar. O lugar ' a' ativa uma das transições ('t1' ou 't2') e inibe a outra, podendo levar a diferentes lugares ('b' ou 'c'), como mostrado na Figura 2.19.

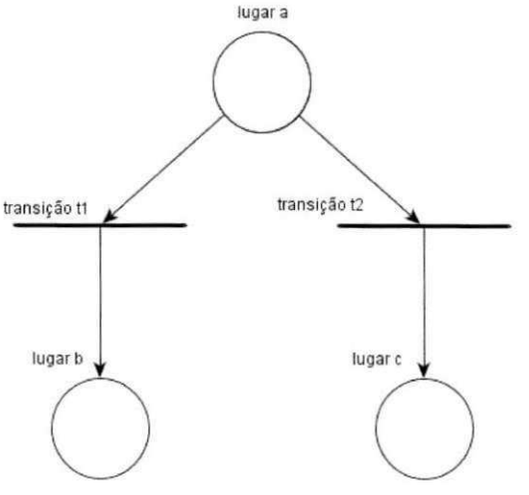


Figura 2.19 – Rede de Escolha não Determinística

2.3.3 Árvore de Cobertura

Em Murata (1989) a partir de uma marcação inicial de uma Rede de Petri, podem-se obter novas marcações a partir das transições que forem sendo ativadas, e, para cada nova marcação, pode-se, novamente, alcançar novas marcações. A restrição a se fazer é que a rede utilizada deve ser limitada para evitar que a árvore cresça indefinidamente.

A árvore é construída usando algoritmos de busca em profundidade ou amplitude. A árvore resultante contém a representação de todas as combinações das marcações alcançáveis da rede, que é chamada também de árvore de alcançabilidade. Os nós representam a marcação inicial e seus sucessores, e cada arco representa uma transição que ativa uma marcação a outra.

A limitação de se usar essa técnica de análise é que, dependendo da quantidade de lugares e transições envolvidos na Rede de Petri e da complexidade da rede trabalhada, pode ocorrer uma explosão de estados. Como dito em Murata (1989) essa técnica é recomendada para redes "pequenas" e para todas as classes de Redes de Petri.

2.3.4 Maude

Segundo Clavel et al. (2000) Maude é uma linguagem de alto nível e um sistema de alto desempenho que suporta lógica equacional e lógica de reescrita podendo ser utilizada em contextos que envolvem computação concorrente orientada a objetos. Essa linguagem descrita por Clavel et al. (2010) possui duas características que facilitam sua utilização em diversos sistemas (a saber: distribuídos, concorrentes e protocolos de comunicação): simplicidade e expressividade.

Simplicidade, por manter uma sintaxe simples e fácil de entender, usando equações e regras. Um programa em Maude é chamado de módulo, que pode ser:

- **Módulo funcional**: Quando o programa descrito é definido por uma ou mais funções por meio de equações utilizadas como regras para simplificação;
- **Módulo de sistema**: Quando o programa contém regras com possíveis equações. Essas regras são interpretadas como regras de transição locais em um possível sistema concorrente.

Expressividade, por expressar naturalmente computações determinísticas e não-determinísticas, representadas respectivamente por equações, por módulos funcionais e por regras e equações em módulos de sistema. Além dessas características citadas, podem-se considerar também: reflexão; suporte a objetos; tipos; subtipos; módulos; tipos genéricos; sintaxe; e dados definidos pelo usuário.

Segundo Clavel et al. (2010) algumas áreas relevantes que utilizaram Maude com sucesso:

- **Modelos computacionais**: Vários modelos computacionais incluindo modelos concorrentes podem ser especificados naturalmente por diversas teorias dentro da lógica de reescrita as quais podem ser executadas e analisadas em Maude;

- **Linguagens de programação**: Lógica de reescrita fornece semânticas formais à linguagens de programação que futuramente se tornam a base de interpretadores, verificadores de modelos e outras ferramentas de análise de software;
- **Sistemas e algoritmos distribuídos**: Pelas características concorrentes e especificação baseada em objetos, sistemas e algoritmos distribuídos, incluindo protocolos de rede e protocolos criptografados.

No contexto da técnica proposta neste trabalho, esta ferramenta foi escolhida pela facilidade de se trabalhar com modelos, sintaxe simples e pelas funções de verificação de modelos utilizadas na modelagem de formalismos.

Uma Rede de Petri no formato Maude é formada por um conjunto de estados que representam os lugares da rede, e um conjunto de regras que representam as transições da rede, o conjunto regras e estados, são armazenados em um módulo.

Um módulo é definido da seguinte forma: 'mod' nome (nome é o nome do módulo) 'is' e termina com a palavra chave 'endm'. Um módulo contém conjuntos 'sort' e subconjuntos 'subsorts' que declaram os diferentes conjuntos de dados manipulados pelo módulo e como eles estão relacionados.

Para Redes de Petri, possuem dois tipos de conjuntos: 'Place' lugar e 'Marking' que é uma marcação. A denomincação 'subsort Place < Marking' indica que < Marking é um subconjunto de Place. Os elementos que formam o conjunto Place são descritos pela sintaxe: 'ops' [a1, a2, a3] : -> Place . Os elementos a1, a2 e a3 são do tipo Place.

Uma marcação é representada como um elemento de um conjunto finito do tipo Marking. A constante 'empty' na declaração 'op empty : -> Marking' representa a marcação vazia e o '_ _" é o operador de união de multiconjuntos, representado por 'op _ _ : Marking Marking -> Marking ' essa declaração é acompanhada por atributos equacionais para declarar que esse operador é associativo 'assoc', comutativo 'comm' tendo como identidade o conjunto vazio 'id : empty'. A seguir são descritas as regras de transição, que indicam que a marcação anterior deve ser trocada pela marcação posterior, essas regras indicam a passagem de um *token* de um lugar para outro.

Essa abordagem é utilizada por Clavel et al. (2000) e por Csaba, Mesenguer, e Stehr (2001) para representação de Redes de Petri, baseando-se em Lógica de reescrita. Um exemplo de Rede de Petri descrita por essa abordagem é uma rede que representa o comportamento de um transformador de corrente apresentada graficamente na Figura 2.20 e apresentada em Maude no Código Fonte 2.2.

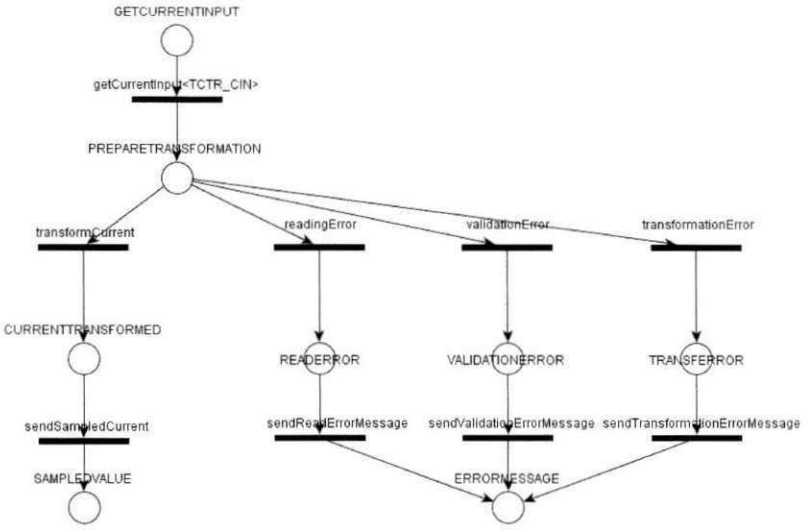


Figura 2.20 – Rede de Petri que descreve o comportamento de um transformador de corrente

Um dos comandos que a ferramenta Maude dispõe é o comando de busca 'search'. Esse comando permite obter o espaço de estados da rede, cobrindo todas as propriedades comportamentais. A sintaxe utilizada para este comando tem o seguint formato:

```
search [a1,a2,a3, ..., na] =>! Any:Marking .
```

A seta '=>!' em Maude indica que serão pesquisados todos os estados finais, considerando os tokens {a1,a2,a3,...,na} que levam a qualquer marcação. Complementando o uso do comando search, usa-se o comando'*show search graph*' para mostrar o espaço de estados obtido em formato textual, para futura manipulação ou visualização. Por exemplo: A Rede de Petri mostrada na Figura 2.20 possui o espaço de estados resultante mostrado na Figura 2.21 realizando-se uma busca considerando duas fichas '*GETCURRENTINPUT*' como entrada.

A ferramenta de busca monta o espaço de estados, da seguinte forma: O lugar inicial da rede é o nó inicial do espaço de estados, a partir da transição disparada, o *token* vai mudar de local, a essa mudança de *token* será atribuído um novo estado no espaço de estados, e ação vai possuir o mesmo rótulo da transição ativada. Maude realiza uma busca em amplitude para varrer todos os estados da Rede que são terminais, e produz o espaço o espaço de estados (CLAVEL et al., 2010).

```
mod TCTR is
sorts Place Marking .
subsort Place < Marking .
ops

GETCURRENTINPUT
CURRENTTRANSFORMED
TCTRERROR
ERRORMESSAGE
SAMPLEDVALUE
PREPARETRANSFORMATION
READERROR
VALIDATIONERROR
TRANSFORMATIONERROR

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl[getCurrentInput<TCTR_CIN>]      : GETCURRENTINPUT          =>
PREPARETRANSFORMATION  .
rl[transformCurrent]               : PREPARETRANSFORMATION    =>
CURRENTTRANSFORMED .
rl[sendSampledCurrent]             : CURRENTTRANSFORMED       =>
SAMPLEDVALUE .
rl[validationError]                : PREPARETRANSFORMATION    =>
VALIDATIONERROR .
rl[readingError]                   : PREPARETRANSFORMATION    =>
READERROR .
rl[.transformationError]            : PREPARETRANSFORMATION    =>
TRANSFORMATIONERROR .
rl[.sendReadErrorMessage]           : READERROR                =>
ERRORMESSAGE .
rl[sendValidationErrorMessage]     : VALIDATIONERROR          =>
ERRORMESSAGE .
rl[sendTransformationErrorMessage]: TRANSFORMATIONERROR      =>
ERRORMESSAGE .
endm
```

Código Fonte 2.2– Rede de Petri em formato Maude do LN TCTR

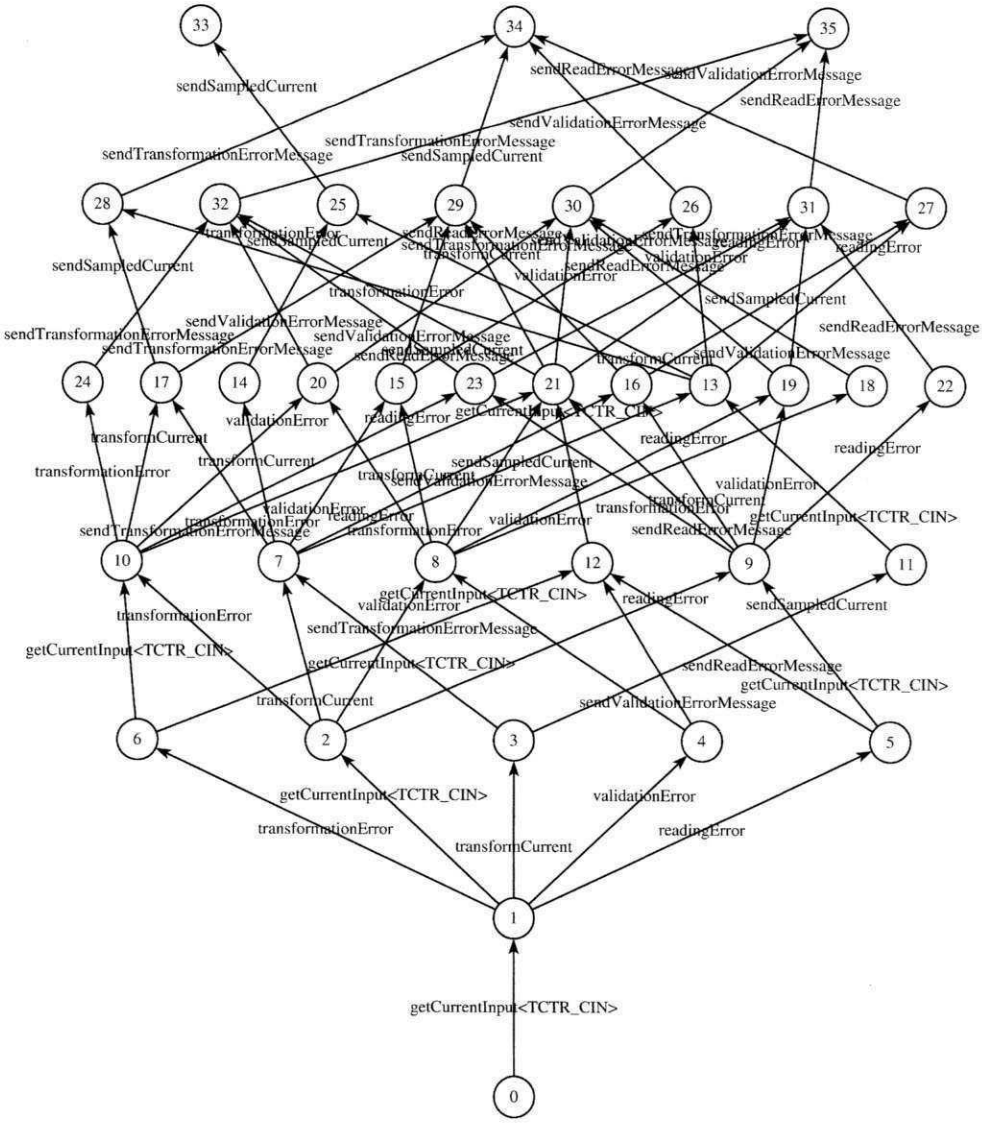


Figura 2.21 - Espaço de estados equivalente a Rede de Petri da Figura 2.20

2.4 Conclusão

Essa seção precedente teve por objetivo apresentar os conceitos abordados neste trabalho: Foi apresentado o conceito de *Model-Based Testing*, explicando, sistematicamente, os passos e conceitos que serviram como base para a construção da técnica de geração de casos de testes proposta. Para aplicar MBT foi realizada uma análise da norma IEC 61850, a qual constitui um dos objetos deste estudo, explicando conceitos como: subestação; sistemas de automação; nós lógicos; dispositivos eletrônicos inteligentes e dispositivos lógicos.

Por fim, foram apresentadas as Redes de Petri para modelagem dos LN e construção do espaço de estados de uma subestação completa.

Capítulo 3

Técnica de Geração Automática de Testes

Este capítulo apresenta a técnica proposta para a geração de casos de testes usando MBT em ambientes 61850. Será apresentada a arquitetura proposta, ferramentas e requisitos utilizados, artefatos gerados e o resultado final.

3.1 Visão Geral da Técnica

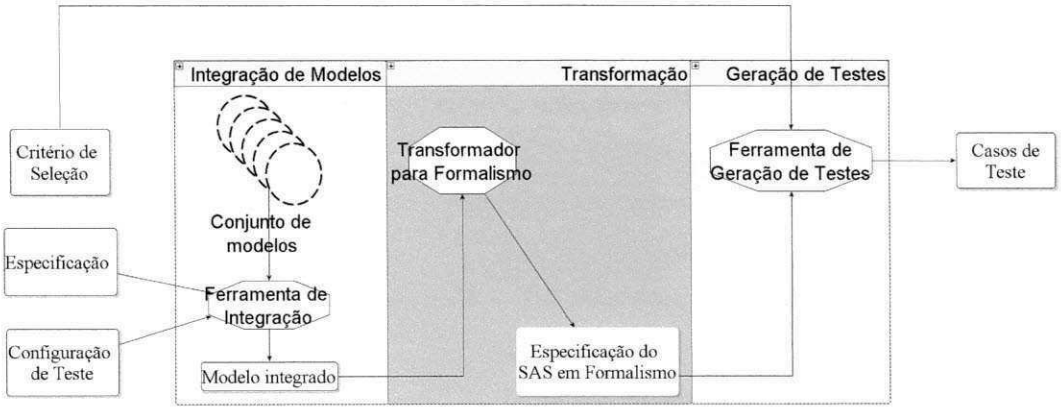


Figura 3.1 – Arquitetura Proposta em alto nível

A Figura 3.1 mostra a arquitetura em alto nível da técnica proposta. Essa arquitetura é dividida em três partes interdependentes: Integração de Modelos; Transformação e Geração de Testes. Essa figura foi obtida baseando-se em um conjunto de requisitos funcionais descritos a seguir:

- Definir a especificação do SAS a ser testado: A técnica deve permitir a geração de script de testes para qualquer especificação de SAS, para isto deve conter modelos individuais dos LN que serão integrados formando um modelo que represente o SAS a ser testado.

- Criação de um algoritmo de integração: Esse algoritmo de integração deve ser formado por um conjunto de regras que estabeleçam a comunicação entre cada LN, padronizando as conexões que possam existir gerando um modelo integrado com os LN definidos pela especificação do SAS a ser testado e de um cenário de teste que contemple situações a serem abordadas pela especificação.
- Definição da ferramenta de geração de casos de testes: Existem ferramentas concretas de geração automática de testes baseadas em MBT, que utilizam especificações de sistemas a serem testados, a técnica escolhida deve gerar casos de testes que contenham a integração de modelos de LN.
- Seleção de script de testes: Ferramentas de geração de casos de testes baseadas em MBT produzem uma grande quantidade de casos de testes levando ao problema conhecido como explosão de estados. Para contornar este problema deve ser utilizada uma técnica de seleção de casos de testes para gerar os scripts de testes.

A partir desses requisitos funcionais, foram definidas algumas decisões arquiteturais para concretizar a técnica proposta:

- Formalismo utilizado para modelagem: As Redes de Petri foram escolhidas por possuir expressividade para representar características concorrentes e interdependentes comuns em SAS 61850.
- Protótipos de adaptação: Para adaptar a especificação do SAS a ser testado no formato de manipulação da ferramenta TGV foram criados protótipos que transformam Espaço de estados de Redes de Petri no formato LTS padrão da ferramenta TGV.
- Geração de casos de testes: A ferramenta TGV produz um grafo que contém todos os casos de testes, os nós são os estados e as arestas são ações realizadas. Um caminho que parte do nó raiz do grafo completo de testes até um nó '*Accept*' corresponde a um caso de teste. Para obter todos os casos de testes foi implementado um protótipo que implementa um algoritmo de busca em profundidade, percorrendo caminhos partindo do nó raiz até nós folha, gerando um conjunto de casos de testes.

- Seleção de casos de testes: Desse conjunto de casos de testes obtidos pela ferramenta TGV com o protótipo de busca em profundidade, foi criado um algoritmo de seleção próprio apenas como prova de conceito, por não existir uma padronização do formato formal de representação do conjunto de casos de testes para ser utilizado com ferramentas existentes para seleção.

Desses requisitos funcionais e não funcionais foi proposta uma arquitetura concreta que instancia cada etapa integrando entradas e saídas, tornando a técnica executável automaticamente.

3.2 Concretização da Técnica

Com base na Figura 3.1 foram pesquisados na literatura ferramentas, formalismos e formatos que permitam instanciar cada etapa proposta visando a geração dos casos de teste a partir de uma especificação, concretizando a arquitetura mostrada na Figura 3.2. A automação da técnica foi possível utilizando um ambiente Linux e um script que executa passo a passo as ferramentas e protótipos implementados.

A técnica proposta possui três parâmetros de entrada: a especificação do SAS a ser testado, um critério de seleção de casos de testes e o arquivo de configuração do SAS.

As ferramentas utilizadas foram Maude – para modelagem e obtenção do LTS da especificação; LNIntegrator.jar que realiza a integração de modelos; TGV – para a geração de casos de testes; e, a fim de adaptar o formalismo utilizado para a ferramenta de casos de testes foram implementados dois protótipos: maude2dot.py e maude2albebaran.jar. Para obter os casos de teste do grafo de teste completo obtido por TGV é realizada uma busca em profundidade utilizando o protótipo DFSonCTG.

O resultado da busca em profundidade é um conjunto de casos de testes que, serão selecionados com base em um algoritmo de seleção para concretizar os casos de testes.

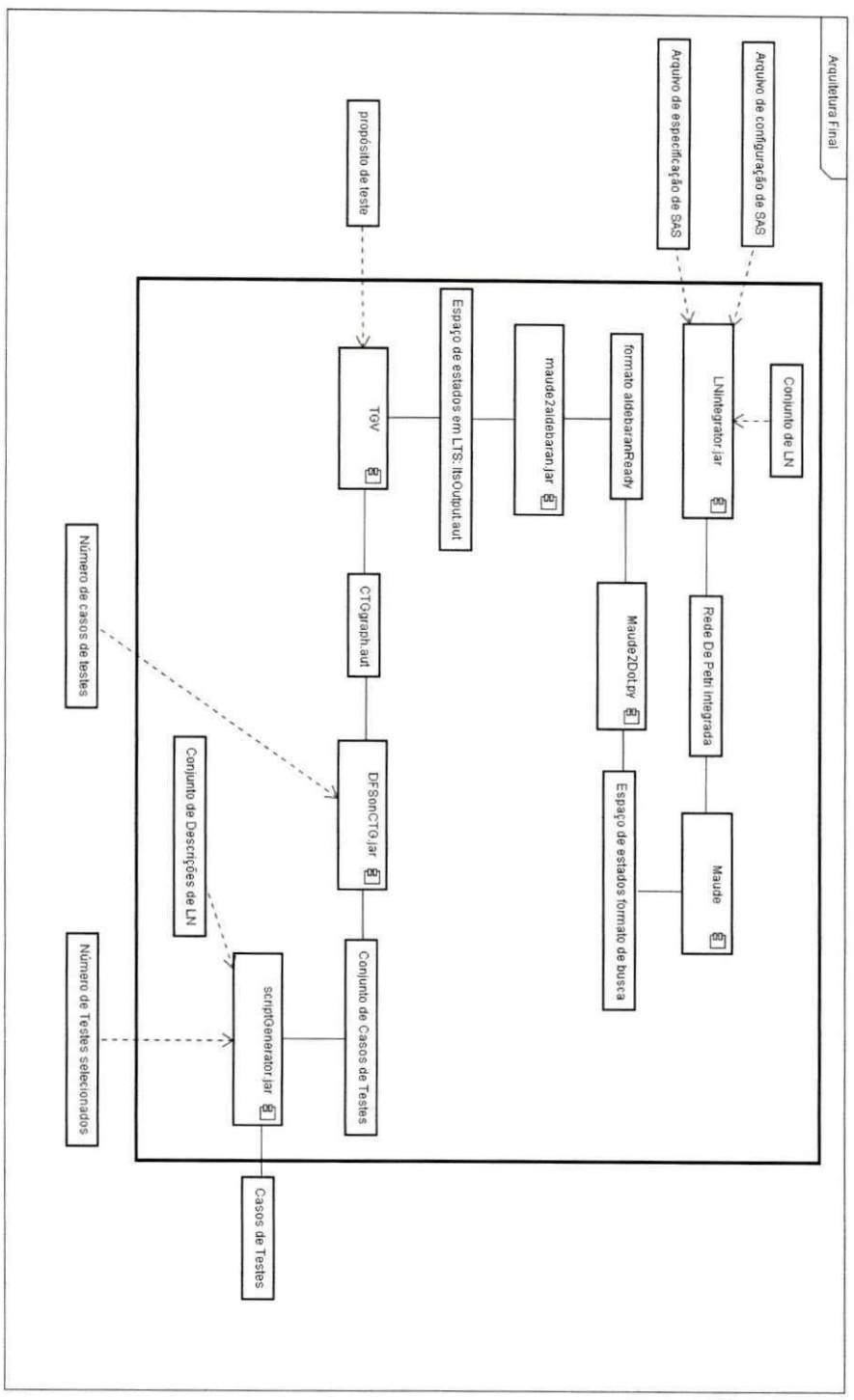


Figura 3.2– Arquitetura Proposta instanciando componentes e artefatos utilizados

3.2.1 Especificação do SAS

Para descrever quais LN compõem um SAS é necessária a utilização de uma especificação que informa a topologia do SAS a ser testado. Pela norma IEC 61850 essa especificação do SAS é informada pelo arquivo SCL. A desvantagem de se usar SCL é que a especificação não está descrita formalmente, e sim na definição de conexões entre LN.

Foi criado manualmente um arquivo de especificação baseado em um diagrama de comunicação, como o da Figura 3.3 obtido a partir de documentação do SAS a ser testado, gerando uma especificação contendo a conexão entre os LN envolvidos. Essa abordagem foi utilizada por dois motivos: primeiro, porque o arquivo de descrição de subestação contém muitos detalhes sobre a implementação do SAS que não são necessários para a geração de testes e, segundo, porque o arquivo de descrição de subestação é voltado para a descrição estrutural dos aspectos do SAS.

Para facilitar a manipulação da especificação foi utilizada como base a topologia da subestação extraída a partir do diagrama de comunicação. A especificação referente ao diagrama baseado na Figura 3.3 de Alencar et al. (2009) é mostrada no Código Fonte 3.1.

No diagrama de comunicação os LN são representados pelo seu nome incluindo o número referente a instância, ex: TCTR1 e TCTR2 são duas instâncias de um LN TCTR. No Código Fonte 3.1, o rótulo antes do '.' indica a classe do LN, e o nome após o '.' indica a referência. Ex.: TCTR.TCTR1, indica que TCTR1 é uma instância da classe de LN TCTR (Transformador de corrente). A nomenclatura X "->" Y indica que um LN X está conectado, diretamente, com o LN Y.

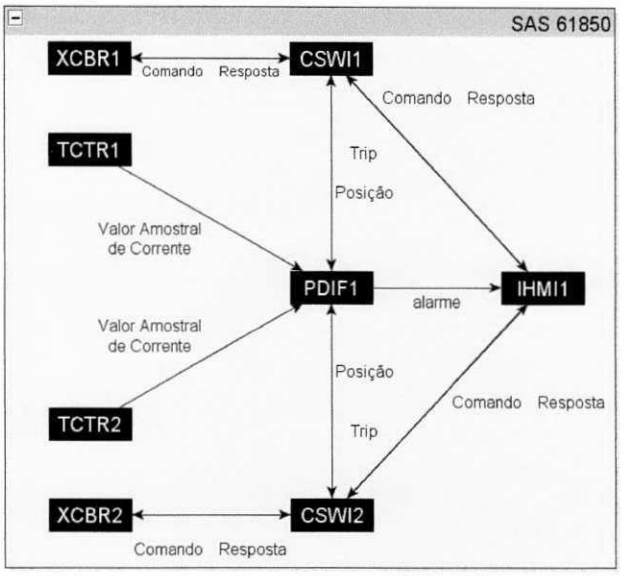


Figura 3.3 – Diagrama de comunicação (ALENCAR et al., 2009)

```
TCTR.TCTR1 -> PDIF.PDIF1
TCTR.TCTR2 -> PDIF.PDIF1
XCBR.XCBR1 -> CSWI.CSWI1
XCBR.XCBR2 -> CSWI.CSWI2
CSWI.CSWI1 -> XCBR.XCBR1
CSWI.CSWI2 -> XCBR.XCBR2
CSWI.CSWI1 -> PDIF.PDFI1
CSWI.CSWI2 -> PDIF.PDIF1
CSWI.CSWI1 -> IHMI1
CSWI.CSWI2 -> IHMI1
PDIF.PDIF1 -> CSWI.CSWI1
PDIF.PDIF1 -> CSWI.CSWI2
PDIF.PDIF1 -> IHMI.IHMI1
```

Código Fonte 3.1 – Especificação do sistema

3.2.2 Critério de Seleção de Casos de Testes

Para o estudo foram utilizados propósitos de teste como requisito para a utilização pela técnica de seleção de casos de testes TGV. No contexto deste trabalho, o propósito de teste é representado por um grafo, onde os nós representam estados que desejam ser contemplados e as transições mostram ações que devem existir no caso de teste. O objetivo é ter um comportamento que é subconjunto do LTS da especificação. Como exemplo da

sintaxe utilizada será mostra um propósito de teste, mostrado no Código Fonte 3.2 especifica o comportamento desejado para a proteção diferencial.

```
des(0,6,6)
(0,"TCTR1.getCurrentInput<TCTR_CIN>",2)
(1,"TCTR2.getCurrentInput<TCTR_CIN>",2)
(2,"PDIF1.tripCommandtoCSWI",3)
(3,"PDIF1.alarmProtectionSuccess",4)
(4,"IHMI1.showMessage",5)
(5,ACCEPT,5)
```

Código Fonte 3.2 – Um possível propósito de testes para a especificação mostrada no Código Fonte 3.1

3.2.3 Configuração do SAS

A especificação do SAS indica quais LN estão conectados, porém essa conexão pode apresentar várias situações que precisam ser testadas, as quais são descritas por um arquivo de configuração que mostra as situações em que o SAS se encontra.

Essa configuração é utilizada para complementar o modelo de Rede de Petri que representa o modelo integrado de LN para descrever os *tokens* necessários para a execução da Rede. Como restrição a configuração do SAS deve conter estados que pertencem ao modelo de Rede de Petri integrado para garantir que os *tokens* sejam instaciados. Essa configuração é de um conjunto de declarações no formato:

```
[InstanciaDoLn].EstadoComToken
```

Onde o 'instanciaDoLn' se refere a instância do LN que se encontrar o estado que define o cenário a ser executado definido pelo rótulo 'EstadoComToken'.

3.2.4 Modelagem de LN

Como os SAS são sistemas compostos por vários LN que se comunicam e interagem entre si, foram estudados dois formalismos para modelagem: Máquinas de estado finito e Redes de Petri.

Os modelos criados em Máquinas de estado permitem analisar a interação entre LN e atuais estados em que o SAS modelo se encontra; porém, a integração dos modelos não é trivial por não ter um bom suporte ferramental e pouca expressividade, quanto às Redes de

Petri, tendo sido esta escolhida por possuir, como diferencial, ferramentas como Maude descrita em Clavel et al. (2000) que facilitam a criação e manipulação dos modelos. As Redes de Petri permitem expressar situações de concorrência, convergência, conflitos e escolhas, situações que ocorrem com freqüência em protocolos de comunicação, como é o caso da norma IEC 61850.

A partir de reuniões com pesquisadores da norma IEC 61850, junto com as descrições dos LN mostrados pela norma IEC 61850-6 (2003) foi possível montar algoritmos que expressem o comportamento desses LN. A partir desses algoritmos comportamentais são construídos os modelos de Redes de Petri para cada um dos LN da norma IEC 61850. O Código Fonte 3.3 mostra o algoritmo em pseudocódigo para o LN TCTR, ao passo que os algoritmos dos demais LN, utilizados na subestação descrita no estudo de caso desse trabalho, serão mostrados no Anexo A.

```
1.    threshold = recebeThreshold()
2.    while(true){
3.          Cin = leCorrenteEntrada()
4.          IF(ocorreErroDeLeitura)
      5.                sendErrorMessage() // envia notificação ao
componente conectado: valor de entrada da corrente possui erro
      6.          ELSE (Cin = ok)
7.          IF(Cin <= threshold)
8.                sendErrorMessage() // envia notificação ao
componente conectado: A transformação só ocorre em Cin > threshold
9.          ELSE // (Cin > threshold)
10.               Cout = realizaTransformacao()
11.         IF(ocorreErroTransformacao)
12.               sendErrorMessage() // envia notificação ao
componente conectado: erro no processo de transformação
13.         ELSE // (Cout = ok)
14.               sendCurrentToComponent(Cout) // envia Cout
componente conectado
15.   }
```

Código Fonte 3.3 – Algoritmo comportamental para LN TCTR

Para este trabalho as Redes de Petri dos LN envolvidos no estudo de caso foram construídas em Maude, seguindo a abordagem mostrada em Mesengues (1992). Essa abordagem mostra como são modeladas cada transição e lugar de uma Rede de Petri no formato Maude. As redes são representadas por módulos, através deles, são implementadas as lógicas de reescrita. A Figura 3.4 mostra um exemplo de um modelo de Rede de Petri

produzida para representar o LN TCTR. Os estados representam configurações do sistema descritos em letras maiúsculas e as ações são representadas pelas transições em letras minúsculas.

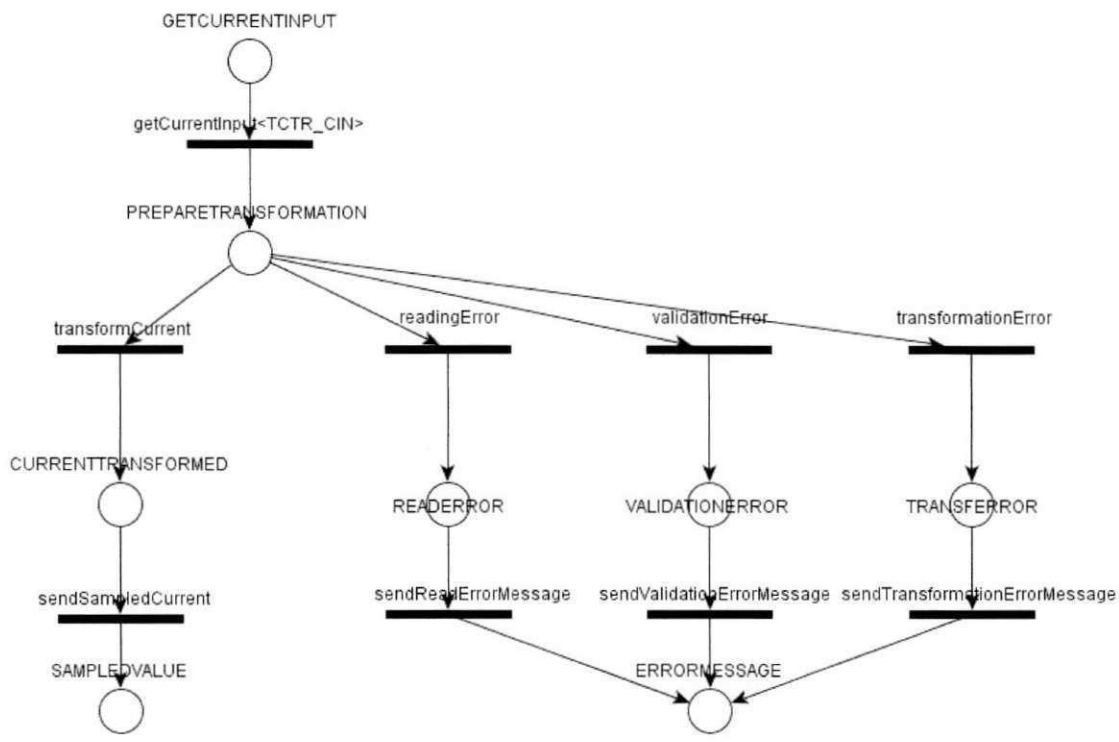


Figura 3.4 – Rede de Petri para o LN TCTR

Os modelos produzidos para representar os LN precisam ser representativos, contendo as principais características que cada LN desempenha e também precisam ser concisos para abstrair informações que não são relevantes ao sistema como um todo. A ação de um transformar corrente, por exemplo, mostrada pela transição 'transformCurrent' da Figura 3.4 não precisa ser detalhada mostrando como é o processo de transformação da corrente, leva-se em consideração que um valor de corrente é recebido e é repassado um valor já transformado.

Todos os modelos em Redes de Petri no formato Maude dos LN da norma IEC 61850 utilizados neste trabalho podem ser visualizados no anexo B e fazem parte de um repositório de arquivos que foi construído especificamente para este trabalho, isso permite

que qualquer SAS que tenha seus LN modelados possa ser utilizado para obtenção de casos de teste.

A utilização de um repositório de vários LN permite que, à medida em que novos LN vão surgindo, sejam adicionados neste repositório. Dependendo da especificação do SAS a ser testado, os LN serão escolhidos e integrados, permitindo que a técnica aborde diferentes topologias de SAS 61850, desde que os LN envolvidos estejam modelados.

3.2.5 Integração de LN

Com base no repositório dos modelos de LN é preciso criar um algoritmo que, baseando-se na especificação, possa integrá-los, compondo o modelo referente a especificação fornecida.

O arquivo de especificação é formado por um conjunto de 2-tuplas (A, B) onde A e B representam redes que possuem conexão. No arquivo de especificação SAS essa nomenclatura é definida, por exemplo: TCTR.TCTR1 -> PDIF.PDIF1. Onde TCTR1 é uma instância da Rede TCTR, que se comunica com uma instancia da rede PDIF, rotulada de PDIF1.

Cada modelo de Rede de Petri de LN possui estados de entrada que indicam estados de iniciação da rede e nós de saída que representam o estado final da rede. Uma rede A, está conectada com uma rede B, de tal forma que o nó final da rede A serve de entrada para o nó de entrada da rede B. A integração de modelos de Redes de Petri no formato Maude é realizada da seguinte forma:

A cada leitura de uma tupla contida na especificação são identificados quais os LN estão sendo utilizados, esses LN são pesquisados no repositório de LN pelo método 'adicionaModuloLN' e são retornados os módulos de cada rede que são adicionados em uma Rede Integrada inicialmente vazia, incluindo os operadores de lugar e as regras de transição de cada rede indexando-os pela instância da rede, essa abordagem permite que sejam utilizadas várias instância de uma mesma classe de LN, se uma instancia for repetida, seu módulo não precisa ser adicionado, já que a Rede Integrada já a contem.

A integração será realizada incrementando ao conjunto de regras de transição, transições entre o lugar de saída de uma Rede e o lugar de entrada de outra, como mostra a Figura 3.5

Ao adicionar essas regras de conexão, a saída de um *token* que seria o último estado de uma rede A será repassado para o estado inicial de uma Rede B. O algoritmo então deve conter duas etapas: a integração dos módulos e a inclusão de transições de comunicação. O arquivo de especificação será lido, linha por linha, concatenando os módulos das duas redes e adicionando a regra de transição entre as redes. O algoritmo é mostrado no Código Fonte 3.4.

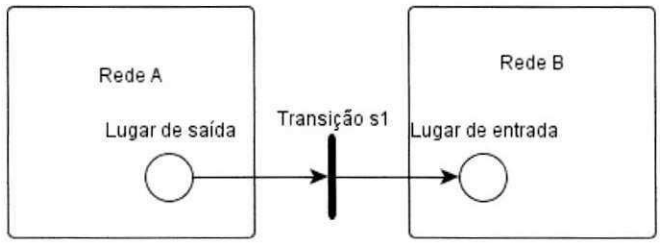


Figura 3.5 – Transição de comunicação entre Redes de Petri

```
1.Integra(arquivo de especificação){
2.ModeloIntegrado = {}.
3.∀ Ligação entre LN descrito em 'arquivo de especificacao' faça
     4.Modelo Integrado adicionaModuloLN(Ligação.LN1) se
Ligação.LN1∉ Modelo Integrado
     5.Modelo Integrado adicionaModuloLN(Ligação.LN2) se
Ligação.LN2∉ Modelo Integrado
     6.ModeloIntegrado.adicionaRegraConexão(Ligação.LN1().regrasCo
municacao(Ligação.LN2)
7.fim laço
8.concatenaConfiguração(arquivo de configuração)
}
```

Código Fonte 3.4 – Algoritmo de Integração de LN

A etapa de inclusão das transições entre LN é definida por um conjunto de regras pré-estabelecidas. Cada LN deve ter regras de conexão que indicam quais outros nós ele pode se conectar, essa regras devem conter todas as possíveis conexões possíveis, para evitar que LN enviem ou recebam mensagens de LN que não estão relacionados. Um dos conjuntos de

regras de conexão é mostrada no Código Fonte 3.5, essas regras são acessadas pela chamada 'adicionaRegraConexão(LN1,LN2), onde LN1 e LN2 são os LN que serão acessados para verificar se é possível a conexão, e que retorna a regra que será adicionada ao modelo integrado. A regra é obtida ao passar pelas condições de conexão, e realizando o algoritmo de definição de regras mostrado no Código Fonte 3.6.

O Código Fonte 3.5 mostra como é conjunto de regras de um LN. Com base nesse conjunto de regras a regra de conexão será inserida no modelo integrado, como mostrada no passo 6 do Código Fonte 3.4.

```
TCTR.regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo PDIS) {
Integratdor.adicionaRegra(this, destinyInstance)}
2. IF(destinyInstance  é tipo MMTR){
Integratdor.adicionaRegra(this, destinyInstance)}
3. IF(destinyInstance  é tipo MMXU){
Integratdor.adicionaRegra(this, destinyInstance)}
4. IF(destinyInstance  é tipo RDRE){
Integratdor.adicionaRegra(this, destinyInstance)}
5. IF(destinyInstance  é tipo PDIF){
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

Código Fonte 3.5 – Regras de conexão LN TCTR

A última etapa do Código Fonte 3.4 é concatenar a configuração do sistema no modelo de Rede de Petri, essa concatenação irá definir o conjunto de *tokens* iniciais, usando pelo comando 'search' e 'show search graph' para realizar a geração de espaço de estados usando Maude. A Figura 3.6 mostra o diagrama de classes que utilizada pelo algoritmo de integração.

A saída desta etapa de modelagem é uma Rede de Petri formada pelos LN integrantes do SAS, em Maude, contendo todos os módulos bem como o conjunto de regras e os *tokens* utilizados para instanciar a rede, como mostrado no Anexo B.

```
adicionaRegra(sourceInstance, destinyInstance)
indiceConexao++
1.Mensagem entradaUtilizada =
destinyInstance.getEntradaNaoMarcada();
2.Mensagem  saídaUtilizada = sourceInstance.getSaidaNaoMarcada();
3.IF(entradaUtilizada é do tipo de saídaUtilizada){
4.      instanciaFonte.marcaSaidaUtilizada(saídaUtilizada)
5.      instanciaDestino.marcaEntradaUtilizada(entradaUtilizada)
6.}
Return 'rl' + [indiceConexao] + : +  'saídaUtilizada' + '=>' +
'entradaUtilizada'
```

Código Fonte 3.6 – Algoritmo de Definição de regras de conexão

3.2.6 Transformação do Espaço de Estados para LTS

A saída da etapa de modelagem é o modelo integrado de LN em Rede de Petri, utilizado como entrada para a etapa de transformação. Nesta etapa, a especificação do SAS deve ser adaptada para ser utilizada pela ferramenta de geração de casos de testes – que no caso foi a ferramenta TGV.

Como requisito de entrada, a ferramenta TGV utiliza uma especificação representada em um sistema de transição rotulado (LTS) descrita em formato Aldebaran. O espaço de estados deve conter todas as combinações dos comportamentos descritos pela rede de Petri. Para transformar o espaço de estados em um LTS foi adotado o seguinte procedimento:

O primeiro passo é transformar a saída de maude, em um formato de grafos, para isso foi implementado o protótipo maude2dot.py. Esse protótipo formaliza a sintaxe da busca do espaço de estados para um formato que possa ser representado em um grafo, como utilizado em Aranha et al. (2011) e armazena o resultado no arquivo' aldebaranReady. A representação de um grafo nesse formato é apresentada no Código Fonte 3.7.

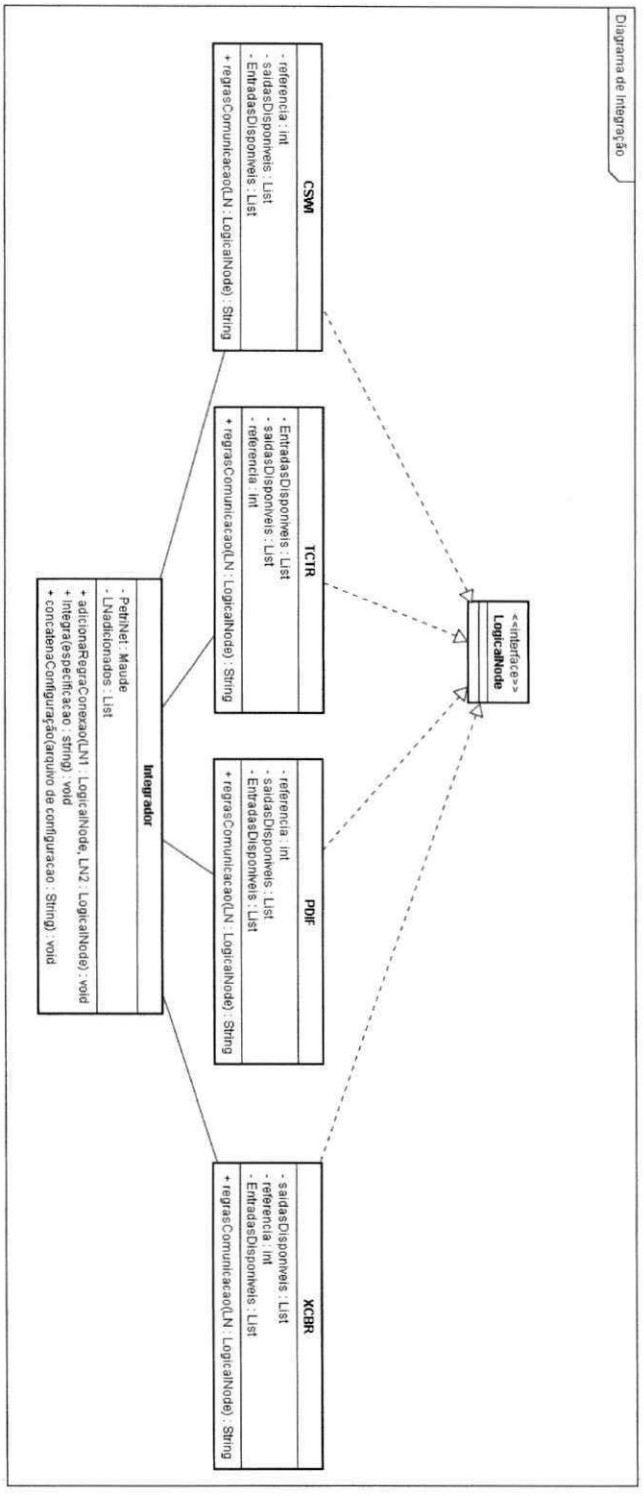


Figura 3.6– Diagrama de classes do algoritmo de integração

```
digraph [nome]
  ranksep=2.0;
  s0 [label="(A B C D)"]
  s0 -> s1 [label="acao1"]
  s0 -> s2 [label="acao2"]
  s1 [label="(A B D E)"]
  s1 -> s3 [label="acao3"]
  s2 [label="(A B E D)"]
  s3 [label="(A B D F)"]
}
```

Código Fonte 3.7 – Representação em formato de grafo do espaço de estados

Para transformar o formato de saída da ferramenta maude2dot para Aldebaran foi preciso manipular os estados e transições. A Tabela 3.1 mostra como é a transformação. A definição de um estado é simplificada para indicar apenas o índice atribuído ao estado e as transições são simplificadas, indicando o índice do estado fonte, transição e o estado destino.

Maude2dot	Aldebaran
s0 [label="(A B C D)"]	0
s1 -> s3 [label="acao3"]	(1,'acao3',3)

Tabela 3.1 Transformação Maude2dot para Aldebaran

Ao aplicar essa transformação a todos os estados e transições produzidos pela ferramenta Maude2dot, o espaço de estados já se encontra no formato Aldebaran. Os protótipos que realizam a simplificação do espaço de estados de Maude e a transformação no formato Aldebaran foram:

- **Maude2dot.py**: Simplifica a saída da ferramenta Maude mostrando as transições entre estados e o rótulo de cada transição. O resultado obtido desse protótipo é o arquivo aldebaranReady;
- **Maude2aldebaran.jar**: É realizada uma leitura do arquivo aldebaranReady, obtido pelo protótipo Maude2dot. Para cada linha do arquivo lido é realizada uma conversão para uma linha no formato Aldebaran. Esse formato é explicado com detalhes no Anexo C. O resultado final é a representação do espaço de estados no formato LTS armazenado no arquivo 'ltsOutput', utilizado como entrada pela ferramenta TGV;

3.2.7 Geração de Casos de Testes

A última etapa utiliza a ferramenta TGV com a especificação do LTS resultante da etapa anterior junto ao propósito de teste utilizado com parâmetro de entrada da técnica. O procedimento utilizado por TGV utiliza dois parâmetros de entrada: um propósito de testes e a especificação do SUT usando operações de determinação e produto síncrono gerando um grafo de teste completo que contém todos os casos de testes.

3.2.7.1 DFSONCTG

O CTG contém todos os casos de testes para a especificação com base no propósito de teste utilizado. Um possível caso de teste desse grafo é o caminho obtido que sai do nó inicial até um nó de aceitação (*pass*). A melhor abordagem para extrair esses casos de testes é utilizar uma busca em profundidade partindo do nó inicial.

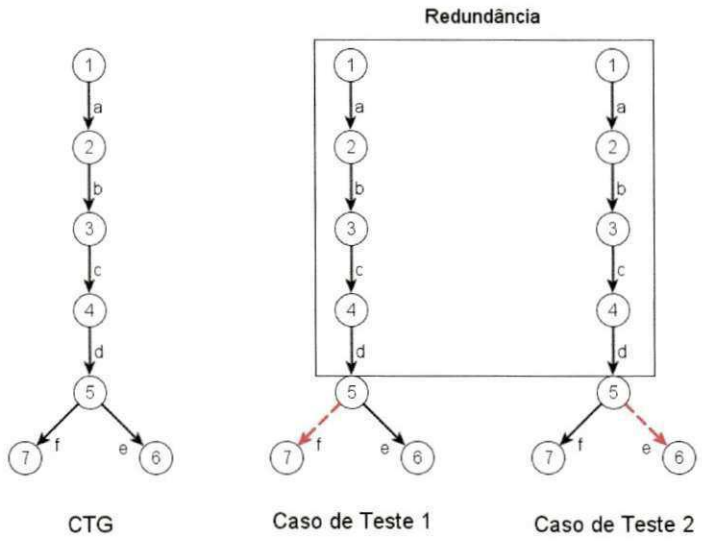


Figura 3.7 – DFS em CTG

O CTG obtido por TGV contém todos os casos de teste para uma dada especificação e um propósito de teste. Uma das formas de selecionar os casos é utilizando uma das ferramentas do pacote que contém TGV e realizar uma busca passo a passo do que se deseja testar, e percorrendo o grafo de teste manualmente. Outra forma de selecionar é realizar uma busca automática no CTG e recuperar os resultados.

Essa busca, parte do princípio que um caminho que parta do nó inicial até encontrar um nó de aceitação, é considerado um caso de teste. Para encontrar todos os casos de testes para uma especificação, é necessário percorrer todos os caminhos do espaço de estados. Essa abordagem automática foi implementada pelo protótipo DFSonCTG.jar, utilizando o algoritmo de busca apresentado em Machado e Sampaio (2007) que realiza uma busca em profundidade visando encontrar caminhos que partem do nó inicial até um nó destino.

Em situações onde o CTG obtido por TGV é pequeno (em torno de 10 ou 20 estados) o algoritmo de busca gera os casos de testes sem nenhuma peculiaridade; mas, para CTGs que possuem mais de 200 estados, o algoritmo de busca exige muito espaço em disco para armazenar os casos de testes gerados. Essa foi uma das desvantagens identificada nessa abordagem. Grande parte dos caminhos percorridos possui várias transições em comum e o número de casos de testes gerados é muito grande, a ponto de interromper execuções por não haver espaço em disco. No exemplo da Figura 3.7, o CTG possui os estados {1, 2, 3, 4, 5, 6, 7}, e as transições {a, b, c, d, e, f}, e pode ter dois casos de teste: {a, b, c, d, e} e {a, b, c, d, f}, que mostram a redundância nos passos: {a, b, c, d}.

Para contornar esse problema de redundância e o grande número de casos de testes, foi utilizado um parâmetro visando finalizar a execução em casos nos quais o limite de armazenamento venha a ser quebrado. Para que os casos de testes possuam maior variedade, é ideal que seja gerada a maior quantidade de casos de teste, utilizando o máximo de hardware para armazenamento.

3.2.7.2 Seleção de Casos de Testes

Um dos problemas detectados foi a grande quantidade de casos de testes obtida utilizando a busca em profundidade. Podendo existir casos de testes irrelevantes – que não caracterize o sistema a ser testado – ou redundantes, sendo recomendada a aplicação de uma técnica de seleção de casos de testes.

Atualmente, existem técnicas de seleção que utilizam *clusterização* ou que se baseiam em critérios de cobertura do conjunto de casos de testes. Qualquer técnica de seleção pode ser adaptada para o conjunto de casos de testes gerado, já que este depende, também, da ferramenta de geração de casos de testes utilizada. Optou-se por fazer uma

seleção implementando o protótipo *scriptGenerator,* por trabalhar com o CTG sem a utilização de um outro software de adaptação para ferramentas existentes.

O algoritmo de escolha utiliza como entrada o conjunto de casos de testes gerados e escolhe um número 'N' (casos de testes), junto com o número total de casos de teste 'T' – quanto maior esse número maior serão as combinações de estados pelo CTG, consequentemente novas combinações de casos de testes. Esse número de casos de testes que serão escolhidos, junto com o número total de casos de teste é utilizado como parâmetro, e os índices dos casos de testes escolhidos são determinados pela seguinte fórmula: I = Nx(T/CT)

- I é o índice referente ao caso de teste escolhido
- N é o índice do caso de teste, vai de 1 até N
- T é o número total de casos de testes
- CT é o número de casos que serão escolhidos

Os casos de teste escolhidos pelo algoritmo de seleção precisam ser adaptados para serem entendidos por um engenheiro; para isso, utiliza-se um algoritmo de descrição de casos de testes, o qual se baseia num conjunto de arquivos de descrição de cada LN, que contém todas as ações e suas respectivas descrições. O algoritmo é mostrado no Código Fonte 3.8.

O algoritmo utiliza um banco de dados de arquivos de descrição de LN. Cada arquivo se refere a apenas um LN, e mostra uma descrição verbal do significado de cada ação realizada, Como exemplo o Código Fonte 3.9 mostra como é o formato de um arquivo de descrição para o LN TCTR, os outros arquivos de descrição dos LN criados neste trabalho estão no Anexo D.

O método montaDescrição lê o arquivo de descrição do LN e recupera a descrição referente à linha lida do caso de teste; essa descrição é escrita no arquivo de teste final e, quando terminar o laço de leitura de caso de teste, é gravado o arquivo de teste final, contendo a ação realiza e uma descrição verbal do seu significado.

```
1. numCasoDeTeste =0;
2. Enquanto não for fim do arquivo de casos de testes
3. Lê linha
4. Se a linha tiver "TC#" {
5.    I++;
6. }
7. Se I == índice{
8.    I ==0
9.    Enquanto não for fim de teste{
10.         se linhaDeTeste não for for conectivo{
11.             LN = recuperaLNDalinha
12.       descrição=montaDescrição(LN.classe,LN.Referencia,LN.ação)
13.             escreveDescriçãoNoArquivoSaída(numCasoDeTeste)
14.             numCasoDeTeste++
15.                 }
16.          }
17. linhaDeTeste = LeLinhaTeste
18. }
```

Código Fonte 3.8 – Algoritmo de descrição de casos de testes

Um *script* de teste é formado por um conjunto de ações descritas por cada LN seguida de uma descrição do que cada ação realiza. Este formato de *script* foi escolhido por poder expressar quais as ações realizadas por cada LN junto a uma descrição textual.

O formato de um script pode ser visto no Código Fonte 3.10. As ações são definidas após o rótulo '*Step*:' e as descrição são mostrada após o rótulo '*Description*:'.

```
1. getCurrentInput<TCTR_CIN> [ Lê valor amostral de corrente e o
armazena ]
2. transformCurrent [ Realiza a transformação da corrente
padronizando-a para manipulação ]
3. sendSampledCurrent [ Com a corrente devidamente  padronizada
ela é enviada para o componente conectado ]
4. validationError [ ocorreu um erro ao receber o valor amostral
de corrente ]
5. readingError [ ocorreu um erro de leitura do valor de corrente
]
6. transformationError [ ocorreu um erro de transformacao da
corrente ]
7. sendReadErrorMessage  [ envia uma mensagem de erro de leitura
ao componente conectado ]
8. sendValidationErrorMessage  [ envia uma mensagem de erro de
validacao ao componente conectado ]
9. sendTransformationErrorMessage [ envia uma mensagem de error de
transformacao ao componente conectado ]
```

Código Fonte 3.9 – Descrição do LN

```
| Step: "LN1.a1" | Description:  decreve a ação a1|
| Step: "LN2.b1" | Description:  decreve a ação b1|
| Step: "LN3.c1" | Description:  decreve a ação c1|
| Step: "LN2.b2" | Description:  decreve a ação b2|
| Step: "LN1.a1" | Description:  decreve a ação a1|
```

Código Fonte 3.10 – Formato de *Script* de testes

3.3 Conclusão

A técnica proposta neste capítulo baseia-se nos conceitos de MBT. A idéia principal é obter casos de teste a partir de um conjunto de modelos individuais mais simples, uma especificação que indique como esses modelos irão se conectar e um modelo integrado que será adaptado para ser utilizado por uma técnica de geração automática de casos de testes.

A vantagem dessa técnica é a criação de uma forma automática para se gerar *script*s de testes usando MBT para SAS que implementem a norma IEC 61850.

Capítulo 4

Protótipo Produzido

Este capítulo irá apresentar o protótipo produzido para concretizar e executar a técnica em um contexto real. As ferramentas escolhidas foram: Maude; TGV e LTS-BT e os protótipos implementados foram: maude2dot.py, maude2aldebaran.jar, DFSonCTG e ScriptGenerator. Todos os artefatos produzidos por essas ferramentas e protótipos são demonstrados no script de execução como pode ser observado no Código Fonte 4.1. Arquivos, configurações e instalação utilizados na experimentação desse trabalho podem ser baixados em: http://www.4shared.com/rar/0iGL2ZQT/test_generation.html?

4. Script de Execução

Para automatizar e reunir todas as ferramentas, protótipos e artefatos em um contexto de execução dependente, foi criado um *script* para execução em ambiente Linux. Esse script é descrito no Código Fonte 4.1. Os rótulos em negrito mostram parâmetros de configuração e entradas que são definidos abaixo:

- **SASSpecFile**: Arquivo de especificação do SAS mostra a descrição dos LN conectados do SAS a ser testado;
- **ConfigFile**: Arquivo que descreve o cenário de teste a ser testado pela topologia descrita no arquivo de especificação;
- **TestPurpose**: Explica um propósito de teste utilizado pela ferramenta de geração de casos de teste TGV;
- **NumTestes**: O número máximo de casos de teste para serem obtidos pelo protótipo DFSonCTG;
- **NumCasosEscolhidos**: O número de casos de teste que será escolhidos pela ferramenta de seleção de casos de teste.

```
#!/bin/bash/
before="$(date +%s)"
 echo 'maude integration start'
   maudeintegration=   java -jar LNIntegrator.jar [SASspecFile]
[ConfigFile] > maudeintegration.maude
   echo 'maude integration end'
   echo 'maude start'
   maudeout= ./maude maudeintegration.maude > maudeout
   echo 'maude end'
   aldebaranready= python maude2dot.py < maudeout > aldebaranready
   echo 'maude to aldebaran start'
   java -jar maude2aldebaran.jar aldebaranready
   echo 'maude to aldebaran end'
   cd /cadp
   bcg_io ltsOutput.aut -unparse bcg_result.bcg
   echo 'executing tgv'
   bcg_open bcg_result.bcg tgv -unparse [TestPurpose]
   bcg_io tgv_result.bcg CTGgraph.aut
   echo 'executing dfs on CTG'
   java -jar DFSonCTGparam.jar CTGgraph.aut [NumTestes]
    echo 'end dfs on CTG'
    echo 'start test Script'
   java -jar scriptGenerator.jar testCases.aut
[NumCasosEscolhidos]
   after="$(date +%s)"
   elapsed_seconds="$(expr $after - $before)"
echo Elapsed time for code block: $elapsed_seconds
```

Código Fonte 4.1 – Script de Execução da técnica

Como a técnica é formada por um conjunto de ferramentas e protótipos, todo o ambiente de execucação deve ser previamente configurado. Máquinas virtuais JAVA e Python para execução dos protótipos implementados. Além das ferramentas Maude e TGV que precisam estar instaladas.

Para execução desses *scripts* os parâmetros em negrito descritos no Código Fonte 4.1 devem ser instanciados, qualquer omissão de parâmetros torna incompleta a execução impossibilitando o processo de geração de casos de testes.

Capítulo 5

Avaliação

Neste capítulo serão apresentados os resultados da aplicação da técnica no contexto de um SAS que utiliza a norma IEC 61850. Na primeira parte é apresentado o estudo de caso, mostrando o contexto do SAS utilizado e a concretização de cada etapa da técnica proposta. Na segunda parte do capítulo serão apresentados os resultados da execução e os casos de testes obtidos pelo processo completo.

5.1 Experimentação

O objetivo do estudo de caso é comprovar a técnica proposta de geração automática de *Script* de testes, gerando um conjunto de scripts para uma dada especificação e um contexto de execução.

Para a escolha de um SAS como estudo de caso foi preciso definir um conjunto dos mais importantes LN. Esses LN devem se comunicar e trocar informações compondo um cenário de teste que contenha o comportamento determinado pelo propósito utilizado como entrada da ferramenta de geração de casos de testes.

Como apresentado na definição de uma subestação no Capítulo 2. Os principais LN que compõem uma subestação elétrica são: disjuntores; transformadores de correntes e chaves seccionadoras. Para demonstrar que a técnica pode ser aplicada neste contexto, foi utilizado, como estudo de caso, uma subestação Baden 220/132kV, descrita em (PATRIOTA, 2009) cuja topologia é apresentada na Figura 5.1, a qual pode ser recuperada com ferramentas adequadas a partir do arquivo de linguagem de configuração de subestação (SCL).

Nesta estação, o transformador T1 possui dois enrolamentos, W1 e W2. W1 está conectado no nível de tensão E2, de 220kV, no barramento Q1, ao passo que W2 está conectado no nível de voltagem E1, de 110kV, no barramento Q2; em cada extremidade do transformador T1 estão conectados transformadores de corrente I1 e I2. O barramento E1Q2, de 132kV, contém um disjuntor QA2 e desconector QB1, e o barramento E2Q1 possui um disjuntor QA1. Esses equipamentos funcionam da seguinte forma:

- **Transformador**: O Transformador T1 irá transferir a energia elétrica de um circuito a outro. De forma geral, a sua função é reduzir as perdas por redução da corrente requerida para transmitir a potência elétrica de 220kV para 132kV (NYNÄS, 2004).
- **Disjuntores**: Os disjuntores (QA1 e QA2) são os principais equipamentos de segurança e mais eficientes dispositivos de manobra nas redes elétricas. Quando estão fechados devem suportar a corrente nominal da linha, respeitando um limite de temperatura e ao receber comando de *trip* eles abrem, isolando o circuito (DUABIBE, 1999).
- **Chaves seccionadoras**: isolam equipamentos ou zonas de barramentos, enviam sinais de comando, recebem resposta de LN conectados e notificam as operações realizadas (IEC 61850-6, 2003).
- **Transformadores de corrente**: Esses transformadores (I1 e I2) permitem que os instrumentos de medição e proteção – no caso da subestação Baden, os disjuntores – funcionem adequadamente repassando o valor amostral padronizado, sem que seja necessário possuírem correntes nominais de acordo com a corrente de carga do circuito que estão conectados (ALENCAR et al., 2009).

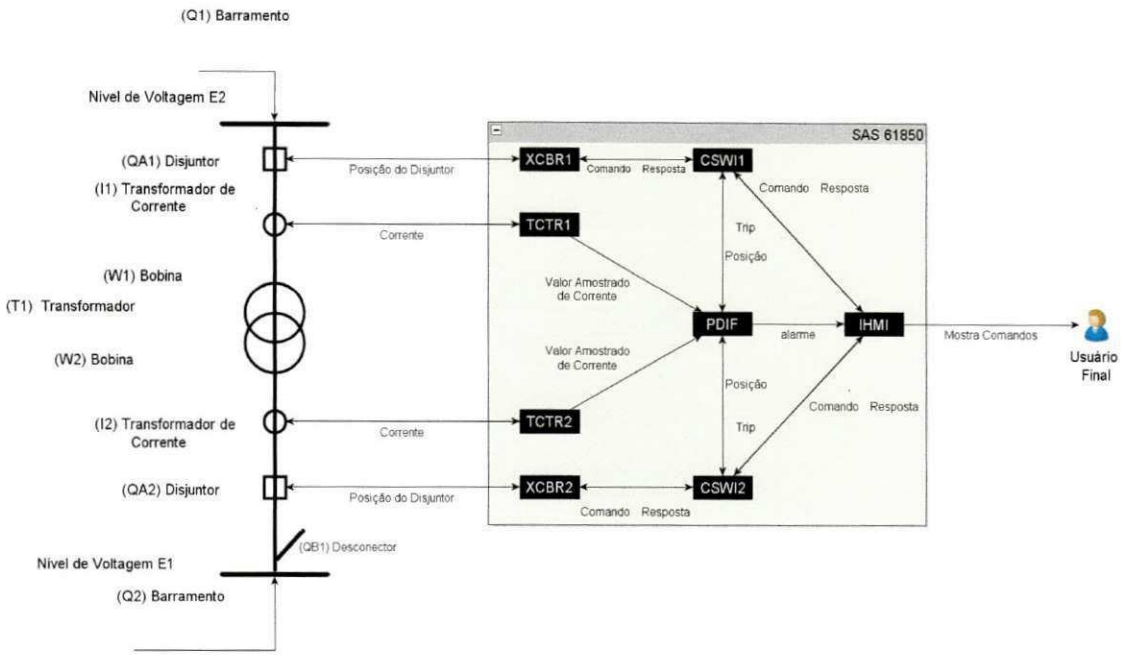


Figura 5.1 – Subestação Baden (Estudo de Caso)

Todos os equipamentos do sistema primário da subestação estão conectados aos IED que contém os LN, que representam os dispositivos reais. Na Figura 5.1, a caixa preta mostra os LN que compõem o SAS para a subestação Baden. Os LN envolvidos e suas nomenclaturas são:

- XCBR1 e XCBR2: Disjuntores;
- PDIF: Proteção diferencial;
- CSWI1 e CSWI2: Chaves seccionadoras;
- TCTR1 e TCTR2: Transformadores de corrente;
- IHMI: Interface homem-máquina.

O objetivo principal deste estudo de caso é obter um conjunto de casos utilizando a técnica proposta. Um possível cenário de teste que requer a integração de todos os LN de uma SAS é um exemplo representativo, já que utiliza todos os nós da especificação. Uma das situações em que ocorre num SAS a utilização de todos os LN é a proteção diferencial.

A proteção diferencial (PDIF) detecta uma mudança nos valores dos transformadores de corrente (TCTR1 e TCTR2) devido a alguma falha no transformador e envia sinais para

os disjuntores conectados (XCBR1 e XCBR2), através das chaves seccionadoras (CSWI1 e CSWI2), visando à manobra de proteção.

Para essa proteção ocorrer o sistema precisa ter as seguintes pré-condições:

- Os disjuntores envolvidos precisam estar fechados;
- Receber dois valores dos transformadores de correntes conectados nas extremidades do transformador de potência.

Ao finalizar a manobra, o sistema estará protegido, os disjuntores conectados estarão abertos e o operador do sistema receberá uma notificação da manobra via IHMI (IEC 61850-7.1, 2003)

5.2 Resultados

Todos os testes foram executados em uma máquina com 220Gb de armazenamento disponível, 4Gb de RAM e processador I5, rodando o Windows 7, como sistema operacional. Como requisito das ferramentas TGV e MAUDE, foi instalado o ambiente cygwin para execução em ambiente LINUX. Os protótipos foram implementados em linguagem JAVA e PYTHON, exigindo que ambos os ambientes precisassem ter sido previamente instalados e configurados.

Utilizando o estudo de caso descrito no tópico anterior, foram realizados experimentos para verificar quantos casos de testes são gerados usando-se a técnica proposta neste trabalho. O número máximo de casos de testes foi limitado pelo tamanho em hardware, ocupando, em disco, um total de 220Gb, gerando 225.881.777 casos de testes durante 10 horas de execução.

O processo de geração de casos de testes será detalhado a seguir. Os dados e parâmetros utilizados foram:

A especificação de um SAS baden mostrada no Código Fonte 5.1 baseando-se na topologia mostrada na Figura 5.1. Para esse SAS usou-se um configuração descrita no Código Fonte 5.2 em que são recebidos os valores de correntes pelos dois transformadores e os disjuntores encontrando-se no estado fechado. Para a geração de casos de teste foi utilizado o propósito de teste mostrado Código Fonte 5. 3 que especifica o comportamento

da proteção diferencial. Foram utilizados como parâmetros: 500.000 como número de casos de testes gerados pelo protótipo DFSonCTG.jar e 10 casos de testes escolhidos pelo protótipo scriptGenerator.jar.

```
TCTR.TCTR1 -> PDIF.PDIF1
TCTR.TCTR2 -> PDIF.PDIF1
XCBR.XCBR1 -> CSWI.CSWI1
XCBR.XCBR2 -> CSWI.CSWI2
CSWI.CSWI1 -> XCBR.XCBR1
CSWI.CSWI2 -> XCBR.XCBR2
CSWI.CSWI1 -> PDIF.PDFI1
CSWI.CSWI2 -> PDIF.PDIF1
CSWI.CSWI1 -> IHMI1
CSWI.CSWI2 -> IHMI1
PDIF.PDIF1 -> CSWI.CSWI1
PDIF.PDIF1 -> CSWI.CSWI2
PDIF.PDIF1 -> IHMI.IHMI1
```

Código Fonte 5.1 – Especificação do estudo de caso

O script de teste foi executado e foi produzido um espaço de estados com 327 estados e 642 transições, depois da transformação e geração de casos de testes o CTG obtido por TGV possui 274 estados e 467 transições. Ao executar a ferramenta de busca para obter o conjunto de casos de testes foram obtidos 500.000 e desse conjunto foram selecionados 10 casos de testes, dois desses casos de testes podem ser observados nos Código Fonte 5.4 e Código Fonte 5.1.

Pelo número de estados e transições do CTG, foi realizada a busca em profundidade utilizando o protótipo DFSonCTG, passando, como parâmetro, 500.000 casos de testes. A partir desse conjunto de casos de testes gerados, foram escolhidos 10 casos de testes – ambos os parâmetros foram escolhidos apenas como prova de concretização da técnica. Como resultado, foram gerados 10 casos de testes durante 97 segundos de execução.

```
TCTR1.GETCURRENTINPUT
TCTR2.GETCURRENTINPUT
XCBR1.CLOSED
XCBR2.CLOSED
```

Código Fonte 5.2 – Configuração do SAS

```
des(0, 6, 6)
(0,"TCTR1.getCurrentInput<TCTR_CIN>",2)
(1,"TCTR2.getCurrentInput<TCTR_CIN>",2)
(2,"PDIF1.tripCommandtoCSWI",3)
(3,"PDIF1.alarmProtectionSuccess",4)
(4,"IHMI1.showMessage",5)
(5,ACCEPT,5)
```

Código Fonte 5. 3 – Propósito de Teste utilizado

```
| Step: "TCTR1.getCurrentInput<TCTR_CIN>" | Description:  Lê valor
amostral de corrente e o armazena |
| Step: "TCTR2.getCurrentInput<TCTR_CIN>" | Description:  Lê valor
amostral de corrente e o armazena |
| Step: "TCTR1.transformCurrent" | Description:  Realiza a
transformação da corrente padronizando-a para manipulação |
| Step: "TCTR2.transformCurrent" | Description:  Realiza a
transformação da corrente padronizando-a para manipulação |
| Step: "TCTR1.sendSampledCurrent" | Description:  Com a corrente
devidamente  padronizada ela é enviada para o componente conectado
|
| Step: "TCTR2.sendSampledCurrent" | Description:  Com a corrente
devidamente  padronizada ela é enviada para o componente conectado
|
| Step: "PDIF1.getCurrentTC2" | Description:  Recebe e armazena o
valor de corrente do tranformador 2 conectado |
| Step: "PDIF1.getCurrentTC1" | Description:  Recebe e armazena o
valor de corrente do tranformador 1 conectado |
| Step: "PDIF1.compareCurrents" | Description:  Compara os dois
valores de correntes lidos |
| Step: "PDIF1.c1minusc2>Thereshold" | Description:  O valor é
maior que o ajuste da proteção diferencial. A proteção é
disparada|
| Step: "PDIF1.tripCommandtoCSWI" | Description:  prepara mensagem
de trip para os Command Switch conectados a PDIF|
| Step: "PDIF1.sendTrip1" | Description:  Envia trip para o
Command Switch 1 conectado |
| Step: "PDIF1.sendTrip2" | Description:  Envia trip para o
Command Switch 2 conectado |
| Step: "CSWI1.getInputCommand" | Description: Recebe comando de
entrada e prepara para enviar para o componente conectado|
| Step: "CSWI2.getInputCommand" | Description: Recebe comando de
entrada e prepara para enviar para o componente conectado|
| Step: "CSWI1.sendCommandToOutputDevice" | Description: Envia o
comando de entrada para o componente conectado e espera resposta|
| Step: "CSWI2.sendCommandToOutputDevice" | Description: Envia o
comando de entrada para o componente conectado e espera resposta|
| Step: "XCBR1.getTripCommand" | Description:  Recebe o comando
para realizar um trip no disjuntor |
```

```
| Step: "XCBR2.getTripCommand" | Description:  Recebe o comando
para realizar um trip no disjuntor |
| Step: "XCBR1.tripXCBR" | Description:  O trip é efetuado com
sucesso |
| Step: "XCBR1.sendTripOpeningSuccess" | Description:  Envia uma
notificação que o trip foi realizado com sucesso |
| Step: "XCBR2.tripXCBR" | Description:  O trip é efetuado com
sucesso |
| Step: "CSWI1.getCommandResponseOK" | Description: Recebe
resposta de sucesso do comando enviado|
| Step: "CSWI1.sendCommandResponseOK" | Description: Envia
mensagem de sucesso para o componente que enviou o comando|
| Step: "XCBR2.sendTripOpeningSuccess" | Description:  Envia uma
notificação que o trip foi realizado com sucesso |
| Step: "CSWI2.getCommandResponseOK" | Description: Recebe
resposta de sucesso do comando enviado|
| Step: "PDIF1.getOkResponse1" | Description:  Recebe notificação
que o trip enviado pelo Command Switch 1 foi realizado com
sucesso|
| Step: "CSWI2.sendCommandResponseOK" | Description: Envia
mensagem de sucesso para o componente que enviou o comando|
| Step: "PDIF1.getOkResponse2" | Description:  Recebe notificação
que o trip enviado pelo Command Switch 1 foi realizado com
sucesso|
| Step: "PDIF1.alarmProtectionSuccess" | Description:  Envia
notificação que a proteção diferencial foi realizada com sucesso|
| Step: "IHMI1.showMessage" | Description: Mostra mensagem no
dispositivo de saída|
```

Código Fonte 5.4 – Script de Teste escolhido I

```
| Step: "TCTR1.getCurrentInput<TCTR_CIN>" | Description:  Lê valor
amostral de corrente e o armazena |
| Step: "TCTR2.getCurrentInput<TCTR_CIN>" | Description:  Lê valor
amostral de corrente e o armazena |
| Step: "TCTR1.transformCurrent" | Description:  Realiza a
transformação da corrente padronizando-a para manipulação |
| Step: "TCTR2.transformCurrent" | Description:  Realiza a
transformação da corrente padronizando-a para manipulação |
| Step: "TCTR1.sendSampledCurrent" | Description:  Com a corrente
devidamente  padronizada ela é enviada para o componente conectado
|
| Step: "TCTR2.sendSampledCurrent" | Description:  Com a corrente
devidamente  padronizada ela é enviada para o componente conectado
|
| Step: "PDIF1.getCurrentTC2" | Description:  Recebe e armazena o
valor de corrente do tranformador 2 conectado |
| Step: "PDIF1.getCurrentTC1" | Description:  Recebe e armazena o
valor de corrente do tranformador 1 conectado |
```

```
| Step: "PDIF1.compareCurrents" | Description:  Compara os dois
valores de correntes lidos |
| Step: "PDIF1.c1minusc2>Thereshold" | Description:  O valor é
maior que o ajuste da proteção diferencial. A proteção é
disparada|
| Step: "PDIF1.tripCommandtoCSWI" | Description:  prepara mensagem
de trip para os Command Switch conectados a PDIF|
| Step: "PDIF1.sendTrip1" | Description:  Envia trip para o
Command Switch 1 conectado |
| Step: "PDIF1.sendTrip2" | Description:  Envia trip para o
Command Switch 2 conectado |
| Step: "CSWI2.getInputCommand" | Description: Recebe comando de
entrada e prepara para enviar para o componente conectado|
| Step: "CSWI1.getInputCommand" | Description: Recebe comando de
entrada e prepara para enviar para o componente conectado|
| Step: "CSWI2.sendCommandToOutputDevice" | Description: Envia o
comando de entrada para o componente conectado e espera resposta|
| Step: "CSWI1.sendCommandToOutputDevice" | Description: Envia o
comando de entrada para o componente conectado e espera resposta|
| Step: "XCBR1.getTripCommand" | Description:  Recebe o comando
para realizar um trip no disjuntor |
| Step: "XCBR1.tripXCBR" | Description:  O trip é efetuado com
sucesso |
| Step: "XCBR1.sendTripOpeningSuccess" | Description:  Envia uma
notificação que o trip foi realizado com sucesso |
| Step: "CSWI1.getCommandResponseOK" | Description: Recebe
resposta de sucesso do comando enviado|
| Step: "XCBR2.getTripCommand" | Description:  Recebe o comando
para realizar um trip no disjuntor |
| Step: "XCBR2.tripXCBR" | Description:  O trip é efetuado com
sucesso |
| Step: "XCBR2.sendTripOpeningSuccess" | Description:  Envia uma
notificação que o trip foi realizado com sucesso |
| Step: "CSWI2.getCommandResponseOK" | Description: Recebe
resposta de sucesso do comando enviado|
| Step: "CSWI2.sendCommandResponseOK" | Description: Envia
mensagem de sucesso para o componente que enviou o comando|
| Step: "CSWI1.sendCommandResponseOK" | Description: Envia
mensagem de sucesso para o componente que enviou o comando|
| Step: "PDIF1.getOkResponse1" | Description:  Recebe notificação
que o trip enviado pelo Command Switch 1 foi realizado com
sucesso|
| Step: "PDIF1.getOkResponse2" | Description:  Recebe notificação
que o trip enviado pelo Command Switch 1 foi realizado com
sucesso|
| Step: "PDIF1.alarmProtectionSuccess" | Description:  Envia
notificação que a proteção diferencial foi realizada com sucesso|
| Step: "IHMI1.showMessage" | Description: Mostra mensagem no
dispositivo de saída|
```

Código Fonte 5.5 – Script de Teste escolhido II

5.3 Análise de Resultados

Foram realizados experimentos com o modelo mais simples LN sob diversas configurações para verificar se a técnica é praticável, observando: o tamanho do espaço de estados, a influência do propósito de teste para guiar os casos de teste obtidos a partir do grafo de teste completo, o tempo gasto para se executar a técnica e o número de casos de testes gerados.

Para esse experimento, foram utilizados, como parâmetro, 500.000 casos de testes gerados na ferramenta DFSonCTG e 10 casos de testes selecionados no protótipo scriptGenerator.jar. Os resultados são mostrados na Tabela 5.1

TCTR1.CIN	CTG states	CTG trans	LTS states	LTS trans	CT gerados	TIME(seg)
1	7	6	8	9	4	40
2	36	45	36	72	30	42
3	122	171	120	324	168	40
4	320	474	330	1080	589	41
5	711	1089	792	2970	1409	43
6	1407	2205	1716	7128	2813	45
7	2556	4074	3432	15444	5017	50
8	4347	7020	6435	30888	8281	62
9	7015	11448	11440	57915	12913	92
10	10846	17853	19448	102960	19273	215
11	16182	26829	31824	175032	27777	627
12	23426	39078	50388	286426	38901	1684

Tabela 5.1 – Analise da técnica para Rede Simples

Os resultados da Tabela 5.1, acima, mostram que, à medida em que a configuração utilizada para uma especificação aumenta, sua complexidade também aumenta, influenciando tanto a geração do espaço de estados, quanto a obtenção dos casos de testes. O propósito de teste utilizado para este exemplo consegue reduzir, consideravelmente, o tamanho do espaço de estados, sendo possível gerar casos de testes que cubram o que foi especificado. A partir do 13° valor de corrente lido pelo transformador TCTR1, ocorreu explosão de estados, e a técnica não pode ser aplicada.

Para analisar a escalabilidade da técnica, foram realizados experimentos com diferentes configurações de entrada para uma dada especificação, mantendo-se a seleção de 10 casos de testes num espaço amostral de 500.000. Os experimentos testaram a variação

de valores amostrais de corrente, simulando a chegada de diferentes valores obtidos pelos transformadores de correntes.

Os resultados dessas execuções são mostrados na Tabela 5.2. TCTR1.CIN indica quantos valores de entrada foram lidos por TCTR1; TCTR2.IN indica quantos valores de entrada foram lidos pelo TCTR2. CTG states e CTG trans indicam o número de estados e transições do CTG respectivamente, LTS states e LTS trans indicam o número de estados e transições do LTS respectivamente e TIME(seg) mostra o tempo gasto para se gerar os casos de testes, em segundos.

TCTR1.CIN	TCTR2.CIN	CTG states	CTG trans	LTS states	LTS trans	TIME(seg)
1	1	274	467	327	642	103
1	2	2759	7567	2992	8774	185
2	2	36065	135489	36776	143256	4502
3	1	15828	57337	16742	64284	1190

Tabela 5.2 – Resultados de execuções com vários *tokens* iniciais

A Tabela 5.2 mostra que, à medida em que se aumenta a complexidade de um SAS, mesmo ao adicionar um valor a mais de corrente, o número de estados e transições do espaço de estados e do grafo de teste completo aumenta consideravelmente, como observado nos valores TCTR1 = 1 e TCTR = 2 comparado com TCTR1 = 1 e TCTR2 = 1.

Para TCTR1 = 3 e TCTR2 =1, o número de estados e transições é menor do que TCT1 = 2 e TCTR2 = 2 porque a proteção diferencial é ativada quando se possuem dois valores de corrente em cada transformador de corrente, logo, será ativada uma vez pelo primeiro valor lido, e os outros dois valores lidos serão descartados; por outro lado, quando os dois transformadores possuem, cada um, dois sinais, a proteção irá atuar duas vezes, aumentando-se o número de combinações no espaço de estados e CTG.

Para o SAS do estudo de caso, podem existir 3 possibilidades de configurações a serem testadas:

- **Único valor nos dois transformadores**: Configuração mais comum de funcionamento. Cada transformador lê um valor de corrente e é realizada a proteção diferencial com os dois disjuntores, se em estado fechado;

- **Mais de um valor em um dos transformadores**: Um dos transformadores lê o valor de corrente mais rápido que o outro transformador de corrente. É comutativa para ambos os transformadores; e
- **Mais de um valor de corrente nos dois transformadores**: Os transformadores recebem mais de um valor amostral de corrente, e é, então, realizada a proteção diferencial.

Pelas tabelas Tabela 5.1 e Tabela 5.1, o tamanho do CTG é sempre menor que o espaço de estados pela utilização do propósito de testes, com o objetivo de gerar casos de testes mais focados em características desejadas.

Durante os experimentos, nas combinações de valores lidos dos transformadores TCTR1 e TCTR2 que recebem mais do que 5 combinações de valores, ocorre explosão de estados, notificado pela ferramenta de geração de espaço de estados Maude.

Técnicas baseadas em MBT, para a geração de casos de testes, geralmente produzem um número muito grande de casos de testes que necessitam de um algoritmo de escolha ou de limitações para reduzir o número de casos de testes obtidos.

Foram estabelecidos dois critérios de avaliação para a ferramenta:

- Número de casos de testes gerados: Dependendo do tamanho da especificação do sistema a ser testado, e da capacidade de armazenamento do ambiente de execução, é necessário limitar o número de casos de testes gerados.
- Tempo de execução: O tamanho do modelo integrado do sistema influencia o tempo gasto para gerar os casos de testes. Quanto maior o modelo integrado, mais tempo leva para a técnica gerar os casos de testes. Para o exemplo TCTR1=2 e TCTR2 =2, na Tabela 5.2, foram gastos 4.502 segundos, ao passo que, nos outros casos, o tempo gasto foi bem menor.

5.4 Conclusão

Um trafo de uma subestação comum contém, no máximo, de 20 a 30 LN. A subestação BADEN, utilizada como estudo de caso, possui 4 tipos de LN com 7 instâncias que se encontram no limiar de número de LN possíveis. Os testes gerados para a subestação utilizada no estudo de caso contêm os LN envolvidos, bem como as ações determinadas pelo propósito de teste. Isso mostra que os resultados condizem com o modelo e o propósito de teste utilizado.

A técnica proposta é baseada em MBT, a qual utiliza o espaço de estados de um modelo integrado dos modelos inicias; dito modelo é utilizado como especificação de entrada para uma ferramenta de geração de casos de testes junto a um propósito de testes para guiar o tipo de testes que serão gerados. A partir do grafo de testes completamente gerado, é feita uma busca em profundidade para encontrar os casos de testes e, a partir desses casos obtidos, é feita uma seleção. Os casos de teste que foram selecionados nesta etapa são representados numa linguagem que o usuário final possa entender.

São gerados casos de testes para modelos de especificações mais simples, contendo poucos estados e transições e também para modelos mais complexos com auxílio de parâmetros fornecidos pelo usuário para guiar os testes gerados. Os casos de testes escolhidos pelo algoritmo de seleção são transformados em um script de teste (escrito com os elementos necessários ao pleno entendimento dos profissionais da área) que contém os passos descritos especificados pelo propósito de teste, o qual define o comportamento a ser testado no caso de teste.

Capítulo 6

Trabalhos Relacionados

A geração de casos de testes já é bem difundida em diversas áreas, porém não existem trabalhos relacionados à geração automática de testes aplicados à norma IEC 61850. O trabalho mostrado em Bouquet et al. (2004) mostra a necessidade de testes de integração de SAS e de elementos funcionais de SAS complexos, que precisam corresponder à hierarquia funcional. Trabalhos relacionados a testes e 61850 podem ser vistos em:

- Testes de SAS baseados em 61850: O trabalho apresentado em Alencar et al. (2009) mostra a integração de IED em um SAS complexo e as necessidades de se testar esse tipo de sistema, onde são discutidos testes de integração, de sistema e de elementos funcionais, apresentando a arquitetura do sistema de teste para dispositivos individuais usando valores amostrais analógicos.
- Testes de desempenho e interoperabilidade utilizando a norma IEC 61850: Em Abreu et al. (2007) são realizados testes de conformidade e desempenho em uma arquitetura montada num bancada, usando IED e *switches* de diversos fabricantes. São apresentados e discutidos os resultados obtidos nos testes e a viabilidade da aplicação da norma IEC 61850.

Além de afeto à engenharia elétrica, a MBT é uma técnica aplicada a diversas áreas. Algumas são citadas abaixo:

- Indústria automotiva: O trabalho apresentado em Pretschner, Prenninger e Wagner (2005) realiza um estudo comparativo que aplica MBT a uma rede de controle de uma rede de informações em veículos. A meta é comparar, sistematicamente, a suíte de testes gerada automaticamente com outra gerada manualmente e medir, com precisão, os benefícios de detecção de falhas de MBT.
- Indústria de cartões inteligentes: Foram utilizadas ferramentas de MBT na indústria de cartões inteligentes. Em Bouquet et al. (2004) observa-se que a

validação destes cartões é um exemplo das principais características, o que torna a MBT facilmente aplicável em processos de testes existentes , obtendo um alto nível de retorno de investimento.

- Aplicações em celulares: O trabalho apresentado em Cartaxo, Machado e Neto (2007) utiliza representações da especificação também no formato LTS e realiza a geração automática de testes utilizando funções de similaridade apresentando a ferramenta LTS-Bt. Foram realizados experimentos com essa ferramenta para obter os casos de testes. Uma das vantagens é a utilização de LTS para representação da especificação, similar à ferramenta TGV; outra vantagem é a utilização de funções de similaridade para realizar a seleção de casos de testes; porém, essa ferramenta não foi adotada por trabalhar apenas com sistemas determinísticos, e a necessidade de instrumentar o LTS com rótulos específicos para a geração de casos de testes.

Capítulo 7

Conclusão

Foi proposta uma técnica de geração automática de casos de testes em SAS que utilizem a norma IEC 61850. Os casos de testes obtidos contemplam o comportamento integrado e mostram o fluxo de execução dos testes, concluindo com o término da execução, representado pela última ação executada no script de teste. O ponto forte deste trabalho é a capacidade de extrair *scripts* de testes a partir de uma especificação e um conjunto de modelos, utilizando tecnologias como prova de conceito.

A técnica, em si, é baseada num conjunto de modelos individuais, que são interconectados baseando-se em um conjunto de regras para se obter um modelo integrado; a partir do qual é obtido o espaço de estados com todas as configurações possíveis. Esse espaço de estado é, então, formatado em um formato padrão para uma ferramenta de geração de casos de testes, e depois os casos de testes são selecionados.

Essa técnica pode ser aproveitada para diversas configurações de SAS que implementem a norma IEC 61850, aproveitando-se a arquitetura, adaptando a especificação, o propósito de testes e a configuração utilizada. A norma IEC pode ter novos LN, e para cada LN criado, devem adicionados: o conjunto de regras de comunicação, o modelo individual de Rede de Petri e o arquivo de descrição.

A geração automática de *scripts* de teste em ambientes 61850 é um trabalho inovador, por mostrar uma abordagem sistemática para obtenção de casos de testes que contemplam o comportamento integrado entre os diversos LN que compõem um SAS. A partir da técnica proposta, pode-se verificar o grande número de possibilidades de casos de testes que podem gerados.

7.1 Limitações

A técnica proposta neste trabalho utiliza várias ferramentas para concretizar a arquitetura. A ferramenta MAUDE, utilizada para obter o espaço de estados, apresenta uma limitação em relação ao número de estados e transições obtidos. Em execuções onde o número de *tokens* de entrada em uma Rede de Petri integrada gera várias combinações de caminhos, o espaço de estados pode ficar muito grande, devido ao número de combinações existentes. Isso acaba gerando um problema de explosão de espaço de estados, impedindo a ferramenta MAUDE de finalizar a execução e notificar uma mensagem de erro.

O grafo de teste completo, obtido pela ferramenta TGV, possui todos os casos de testes para uma especificação guiada por um propósito de testes. O algoritmo de seleção utilizado neste grafo consegue obter um número de caminhos que não pode ser suportado pela máquina virtual JAVA; como consequência, o algoritmo falha por falta de memória.

Para contornar essa situação foram realizados vários experimentos para limitar o número de testes obtidos, com um parâmetro que indica quantos caminhos, ou casos de testes, serão percorridos na busca em profundidade.

A técnica adotada neste trabalho utiliza uma seleção simplificada apenas para visualizar os diferentes casos de testes que podem ser obtidos pela técnica, uma vez que é inviável representar os 500.000 casos de testes gerados devido ao grande número de redundâncias havidas entre alguns casos de testes. Para sanar o problema, deve-se aplicar uma técnica de seleção de casos de testes visando a obter os mais diferentes casos de testes possíveis.

7.2 Trabalhos Futuros

Com o término deste trabalho, existem algumas questões que podem ser abordadas e seguem-se infra:

Seleção de casos de testes: O algoritmo de seleção de casos de testes realiza a seleção com base em proporções do arquivo gerado. A este arquivo gerado podem ser aplicados algoritmos que melhorem o desempenho da seleção dos casos de testes sem utilizar qualquer parâmetro, como, por exemplo, a seleção de casos de testes usando funções de similaridade, clusterização ou cobertura.

Extensão da técnica: A técnica proposta foi aplicada apenas em um SAS que possua os LN mais importantes, que envolvam ações integradas. A mesma abordagem pode ser aplicada a outros SAS que implementem a norma IEC 61850.

Bibliografia

[1] ABDURAZIK, A.; AMMANN, P.; OFFUTT, J.; LIU, S. "Generating Test data from State based Specifications", The Journal of Software Testing, Verification and Reliability, 13(1):25-53 – 2003.

[2] ABDURAZIK A.; OFFUTT J. "Using UML Collaboration diagrams for static checking and test generation", Third International Conference on UML, York, UK – 2000.

[3] ABREU, L.; CARVALHO, S.; GUIEIRO, G.; KIERULFF, J.; SOUTO, A. "Testes de Desempenho e Interoperabilidade Utilizando a Norma IEC 61850". XIII Seminário de Automação de Processos – 2009.

[4] ALENCAR, B.; APOSTOLOV, A.; BABER, G.; CARMO, U.; CARTAXO, R.; CASCAES, A.; GARCIA, A.; HOLSTEIN, D.; JANG, B.; JANG, H.; JANSSEN, M.; MAEDA, T.; PATRIOTA, I.; PAULINO, M.; POSTEC, P.;STEEL, M.; STEINHAUSER, S.; Sun B.; TAN, J.; THOLOMIER, D.; THOMPSON, S.; Vandiver, B.; Yanming, R. "Functional Testing of IEC 61850 Based Systems". Work Group B5.32 – 2009.

[5] ARANHA, A.; BARBOSA, P.; BARROS, J.; COSTA, A.; FIGUEIREDO, J.; GOMES, L.; MOUTINHO, F.; RAMALHO, F. "SysVeritas: A Framework for Verifying IOPT Nets and Execution Semantics within Embedded Systems Design". IFIP international federation for information processing – 2011.

[6] BOUQUET, F.; LEGEARD, B.; PEUREUX, F.; TORREBORRE, E. "Mastering test generation from smart card software formal models". In Proceedings of the International Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices,70–85.Springer- Verlag – 2004.

[7] BRAND, K.; LOHMANN, V.; WIMMER, W. "Substation Automation Handbook" UAC – 2003.

[8] BRAND, K.; JANSSEN, M. "The Specification of IEC 61850 Based Substation Automation Systems". ABB – 2005.

[9] CADP toolbox; CADP (Caesar/Aldebaran Development Package) "A Software Engineering Toolbox for Protocols and Distributed Systems" http://www.inrialpes.fr/vasy/cadp/ – 2010.

[10] CARMO, U.; PAULINO, M. "Solução de Arquitetura para Testes Automatizados a Distância de IED Utilizando Equipamento de Teste". VII Simpase – Simpósio de Automação de Sistemas Elétricos – 2007.

[11] CARTAXO, E; MACHADO, P.; NETO F. "Test Case Generation by means of UML Sequence Diagrams and Labeled Transitions Systems". Universidade Federal de Campina Grande – GMF/DSC – 2007.

[12] CASAZZA, J.; DELEA, F.; "Understanding Electric Power Systems: A overview of the Technology and the Marketplace". IEEE press – 2003.

[13] CAVARRA, A.; CHRICHTON. C.;DAVIES J.; HARTMAN, A.; JERON T.; MOUNIER, L. "Using UML for Automatic Test Generation", Oxford University Computing Laboratory, Tools and Algorithms for the Construction and Analysis of Systems (TACAS) – 2000.

[14] CLAVEL, M.; DURAN, F.; EKER, S.; LINCOLN, P.; MARTÍ-OLIET, N.; MESEGUER, J.; QUESADA, J.; "A Maude Tutorial" – 2000.

[15] CLAVEL, M.; DURAN, F.; EKER, S.; LINCOLN, P.; MARTÍ-OLIET, N.; MESEGUER, J.; TALCOTT, C. "Maude Manual" (Version 2.5) HTTP://maude.cs.uiuc.edu – 2010.

[16] CSABA, P.; MESEGUER. J.; STEHR. M. "Rewriting Logic as a Unifying Framework for Petri Nets" – SRI Internatcional – 2001.

[17] DAWIDCZAK, H.; OLIVEIRA, PEREIRA, A. "A experiência de projetos usando a norma IEC 61850 na Europa e na América". VII SIMPASE, Simpósio de automação de sistemas elétricos – 2007.

[18] DUABIBE, P. "Subestações: Tipos, Equipamentos e Proteção; Centro Federal de Educação Tecnológica Celso Suckow da Fonseca" – 1999.

[19] FIGUEIREDO, J. "Workshop 61850; Departamento de Sistemas e Computação" – 2009.

[20] FOSTER, H.; HARTMANN, J.; RUDER, A.; VIEIRA, M. "UML based Test generation and Execution" – 2008.

[21] GUPTA, R. "Substation Automation Using IEC 61850 Standard". XV National Power Systems Conference (NPSC), IIT – 2008.

[22] HETZEL B.; WILLIAM C. "The Complete Guide to Software Testing, 2nd ed." QED Information Sciences – 1988.

[23] IEC-61850-1. "Introduction and overview", First edition 2003-7 – 2003.

[24] IEC 61850-6. "Communication networks and systems in substations - Part 6: Configuration description language for communication in electrical substations related to IEDs", First edition 2003-7 – 2003.

[25] IEC-61850-7-1. "Basic communication structure for substation and feeder equipments – principle and models", First edition 2003-7 – 2003.

[26] IEEE – Institute of Electrical & Electronics Engineers, Standard 610 (1990), reprinted in IEEE Standards Collection: Software Engineering 1994 Edition.

[27] JARD, C.; JÉRON, T. TGV: Theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems", Software Tools for Technology Transfer (STTT) – 2004.

[28] LEGEARD, B; UTTING, M. "Practical Model-Based Testing: A Tools Approach"– 2007.

[29] MACHADO, P.; SAMPAIO, A. "Automatic Test Case Generation". PSSE – 2007.

[30] MESEGUES, J.; "Conditional rewriting logic as a unified model of concurrency". Theoretical Computer Science 96 – 1992.

[31] MURATA, T.; Petri Nets: Propeties, Analysis and Applications; Proceeding of the IEEE, Vol. 77. No 4 – 1989.

[32] NYNÄS A. "Transformer oil handbook". 1 ed. Sweden: Linderoths in Vingaker – 2004.

[33] PATARICCA, A.; VARRO D.; TOTH, A. "Model Level Automatic Test Generation for UML State-Charts", Sixth IEEE workshop on Design and Diagnostics of Electronic Circuits and System (DDECS) – 2003.

[34] PAULINO, M.; Automação de sistemas elétricos; As novas tecnologias aplicadas a sistemas de automação de subestação. Revista Controle & Instrumentação – Edição nº 146 – 2009.

[35] PRENNINGER, W.; PRETSCHNER, A.; WAGNER, S. "One evaluation of model-based testing and its automation". In Proceedings of the 27th International Conference on Software Engineering,392–401. ACMPress – 2005.

[36] TRETMANS, J; VRIES, D. "On-the-fly conformance testing using Spin". International journal on software tools for technology transfer – 2000.

[37] ZANIRATO, E. "Vantagens Adicionais para a Equipe de Manutenção com a Utilização de Relés Microprocessados". SEL Schweitzer Engineering Laboratories, Comercoal Ltda – 2008.

ANEXOS

Anexo A.

Algoritmos Comportamentais de LN

Foram construídos os algoritmos de funcionamento para os quatro LN utilizados no estudo de caso da subestação BADEN: CSWI, XCBR, PDIF E TCTR. A partir desses algoritmos foram construídos os modelos de Redes de Petri que contemplam essas características

Algoritmo TCTR

```
1.    threshold = recebeThreshold()
2.    while(true){
      1.    Cin = leCorrenteEntrada()
      2.    IF(ocorreErroDeLeitura)
            1.    sendErrorMessage() // envia notificao ao
componente conectado (IHMI/PDIF). O valor de entrada da corrente
possui erro
      3.    ELSE (Cin = ok)
            1.    IF(Cin <= threshold)
                  1.    sendErrorMessage() // envia notificao ao
componente conectado (IHMI/PDIF). A transformação só ocorre em Cin
> threshold
            2.    ELSE (Cin > threshold)
                  1.    Cout = realizaTransformacao()
                  2.    IF(ocorreErroTransformacao)
                        1.    sendErrorMessage() // envia notificao ao
componente conectado (IHMI/PDIF)com  erro no processo de
transformação
                  3.    ELSE(Cout = ok)
                        1.    sendCurrentToComponent(Cout) // envia
Cout componente conectado
3.    }
```

Algoritmo XCBR

```
1. While(true){

2.  IF(xcbrState = CLOSED)
     1.   openTrip = waitOpenTripAndWait() //Espera Trip de
abertura
     2.   IF(openTrip = acknowledge)
          1.   authorizeCommand = sendOpenRequestAndWait()
          2.   IF(authorizeCommand = timeout || reject)
               1.   sendErrorMessage() // envia mensagem de erro
aos componentes conectados
          3.   ELSE(authorizeCommand = ok)
               1.   statusOpeningXCBR = xcbrOpenAndWait() //abre
e espera confirmação
               2.   IF(statusOpeningXCBR = sucess)
                    1.   xcbrState = OPENED
                    2.   sendOpeningSucess() // envia mensagem de
abertura realizada aos componentes conectados
               3.   ELSE (statusOpeningXCBR = timeout || error)
                    1.   sendErrorMessage() // envia mensagem de
erro aos componentes conectados
     3. ELSE (openTrip = error | timeout)
          1.   sendErrorMessage() // envia mensagem de erro aos
componentes conectados no caso de timeout ou erro no trip de
abertura

3.   ELSE (xcbrState = OPENED)
     1.   closeTrip = waitCloseTrip()
     2.   IF(closeTrip = acknowledge)
          1.   authorizeCommand = sendCloseRequestAndWait()
          r2.  IF(authorizeCommand = timeout || reject)
               1.   sendErrorMessage() // envia mensagem de erro
aos componentes conectados
          3.   ELSE
               1.   statusClosingXCBR = xcbrCloseAndWait()
               2.   IF(statusClosingXCBR = sucess)
                    1.   xcbrState = CLOSED
                    2.   sendClosingSucess() // envia mensagem de
fechamento realizado aos componentes conectados
               3.   ELSE (statusClosingXCBR = timeout || error)
                    1.   sendErrorMessage() // envia mensagem de
erro aos componentes conectados
     3. ELSE (closeTrip = error | timeout)
               1.   sendErrorMessage() // envia mensagem de erro
aos componentes conectados no caso de timeout ou erro no trip de
fechamento
4.   }
```

Algoritmo CSWI

```
1.    while(true){
      1. commandRecieved = getComandsFromComponent() // recebe os
comandos vindos de PDIF OU XCBR
      2.    IF(commandRecieved = error)
            1.    sendErrorMessage() // erro na leitura do comando,
envia notificação PDIF ou IHMI
      3.    ELSE(commandRecieved = ok)
            1.    statusCommand =
autorizaComandoParaExecucacaoAndWait(commandRecieved)
            2.    IF(statusCommand = ok)
                  1.    sendSucessNotification() //notifica IHMI com
operação realizada com sucesso
            3.    ELSE(statusCommand = error | timeout)
                  1.    sendErrorMessage() // erro na execucao do
comando, envia notificação PDIF ou IHMI
2.    }
```

Algoritmo PDIF

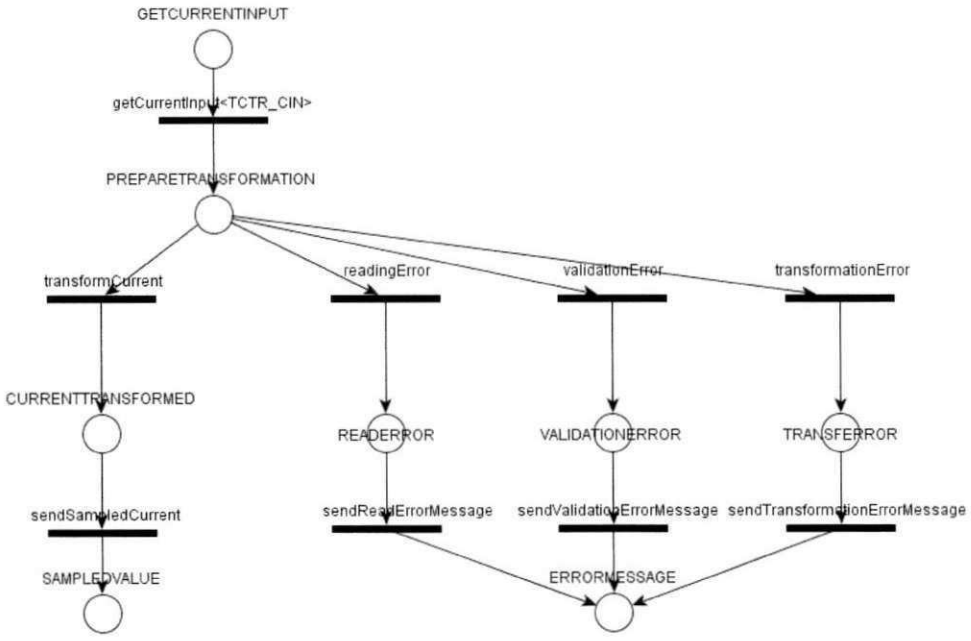
```
1.    threshold = recebeThreshold()
2.    while(true){
      1.    c1 = leCorrente1()
      2.    c2 = leCorrente2()
      3.    erroLeitura = checaCorrentes(c1,c2) //checa erro leitura
      4.    IF(erroLeitura)
            1.  sendErrorMessage()//notifica IHMI com erro (leitura)
      5.    ELSE IF(c1 != c2)
            1.  tripAck = enviaComandoTripAndWait() //envia trip
solicitando autorização e espera sua resposta
            2.  IF(tripAck = timeout | naoAutorizado)
                        1.    sendErrorMessage() // envia notificao
IHMI valor do trip com erro - não autorizado ou timeout
            3.  ELSE (tripAck = ok)
                  1.    response = esperaRespostaComandoRealizado()
                  2.    IF(response = fail | timeout)
                        1.    sendErrorMessage() // envia notificao a
IHMI comando não realizado(IHMI) se ocorreu erro ao processar o
comando ou timeout na operação
                        2.    ativaProtecaoOutroNivel() // ação na
falha da proteção de primeiro nível
                  3.    ELSE (resposnse  = ok)
                        1.    sendSucessNotification() //notifica IHMI
com operação realizada com sucesso
3.    }
```

Anexo B

Modelos de Redes de Petri no formato MAUDE

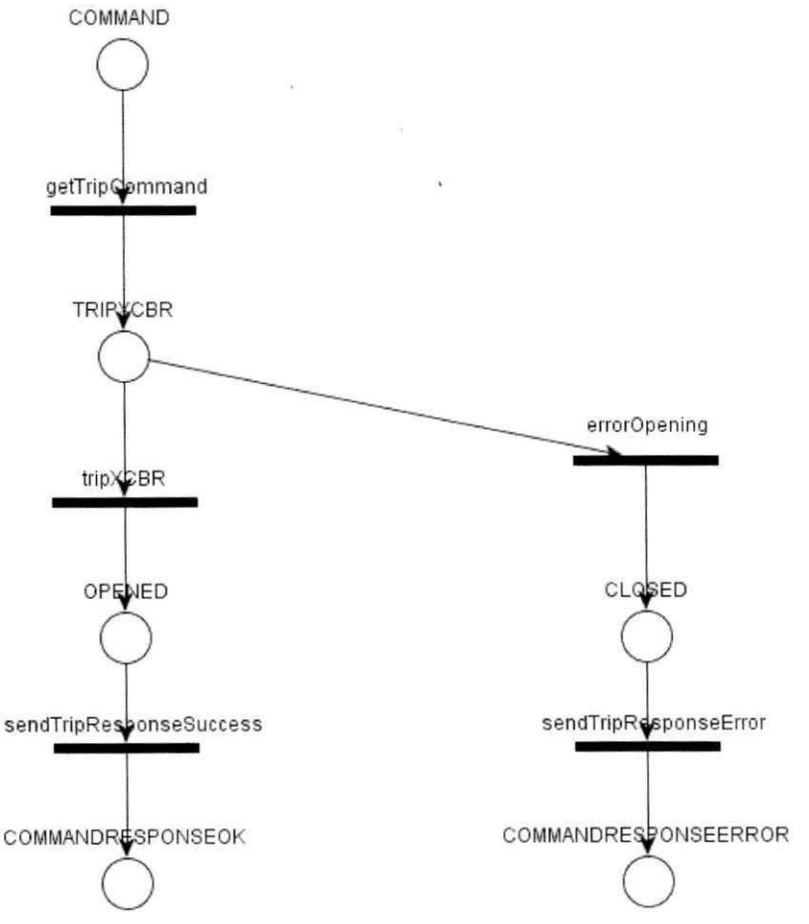
Estes modelos individuais de Redes de Petri representam os comportamentos dos LN descritos pelos algoritmos mostrados no anexo A. Os algoritmos permitiram a criação dos modelos de redes que serão apresentados a seguir no formato de representação em Maude e em formtao gráfico para visualização.

TCTR.maude



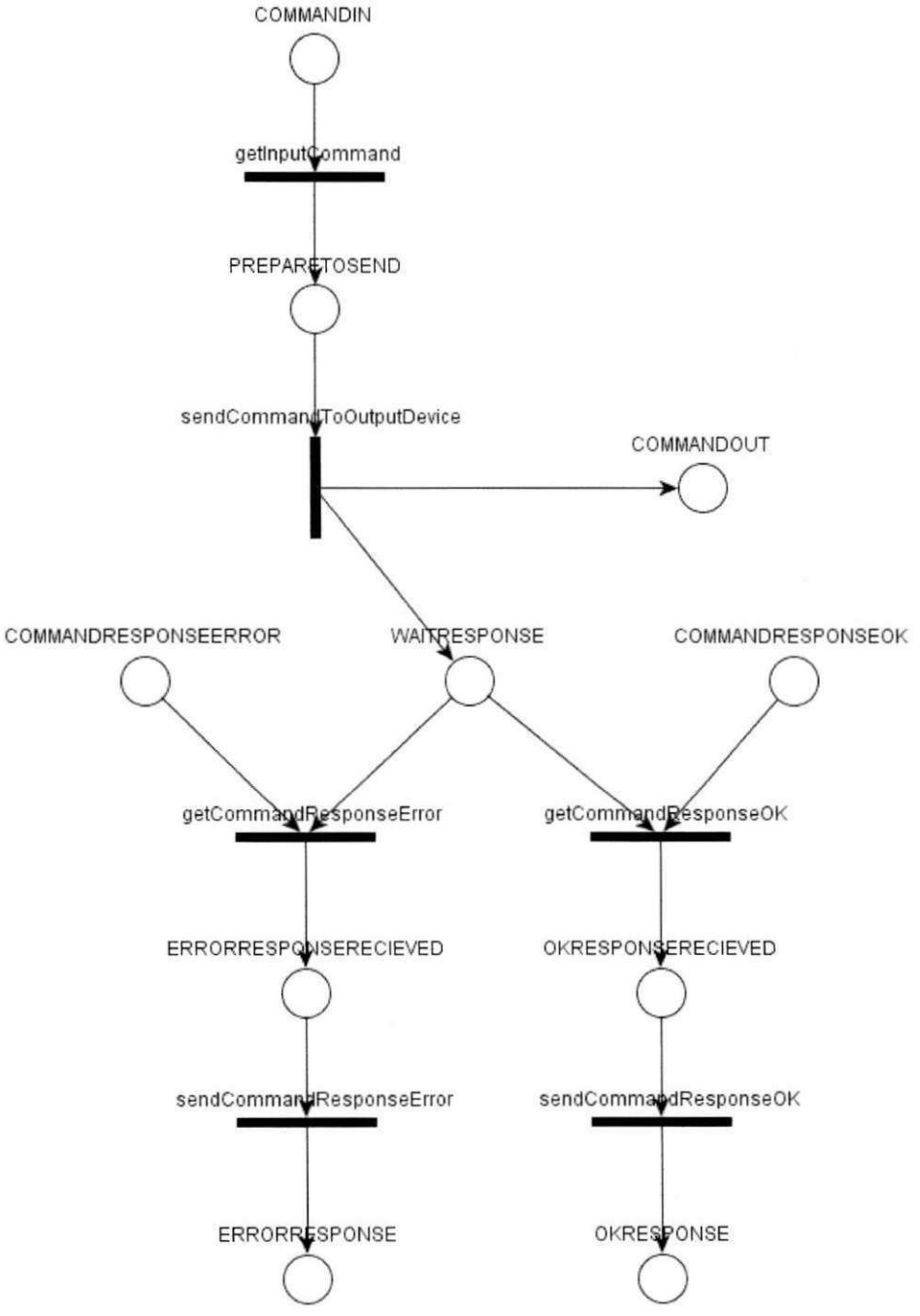
```
mod TCTR is
sorts Place Marking .
subsort Place < Marking .
ops

TCTR.GETCURRENTINPUT
TCTR.CURRENTTRANSFORMED
TCTR.TCTRERROR
TCTR.ERRORMESSAGE
TCTR.SAMPLEDVALUE
TCTR.PREPARETRANSFORMATION
TCTR.READERROR
TCTR.VALIDATIONERROR
TCTR.TRANSFORMATIONERROR

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .



rl[TCTR.getCurrentInput<TCTR_CIN>]      : TCTR.GETCURRENTINPUT
=> TCTR.PREPARETRANSFORMATION  .
rl[TCTR.transformCurrent]               :
TCTR.PREPARETRANSFORMATION   => TCTR.CURRENTTRANSFORMED .
rl[TCTR.sendSampledCurrent]             : TCTR.CURRENTTRANSFORMED
=> TCTR.SAMPLEDVALUE .
rl[TCTR.validationError]                :
TCTR.PREPARETRANSFORMATION   => TCTR.VALIDATIONERROR .
rl[TCTR.readingError]                   :
TCTR.PREPARETRANSFORMATION   => TCTR.READERROR .
rl[TCTR.transformationError]            :
TCTR.PREPARETRANSFORMATION   => TCTR.TRANSFORMATIONERROR .
rl[TCTR.sendReadErrorMessage]           : TCTR.READERROR
=> TCTR.ERRORMESSAGE .
rl[TCTR.sendValidationErrorMessage]     : TCTR.VALIDATIONERROR
=> TCTR.ERRORMESSAGE .
rl[TCTR.sendTransformationErrorMessage]: TCTR.TRANSFORMATIONERROR
=> TCTR.ERRORMESSAGE .

endm
```

XCBR.maude


COMMAND
getTripCommand
TRIPXCBR
errorOpening
tripXCBR
OPENED
CLOSED
sendTripResponseSuccess
sendTripResponseError
COMMANDRESPONSEOK
COMMANDRESPONSEERROR

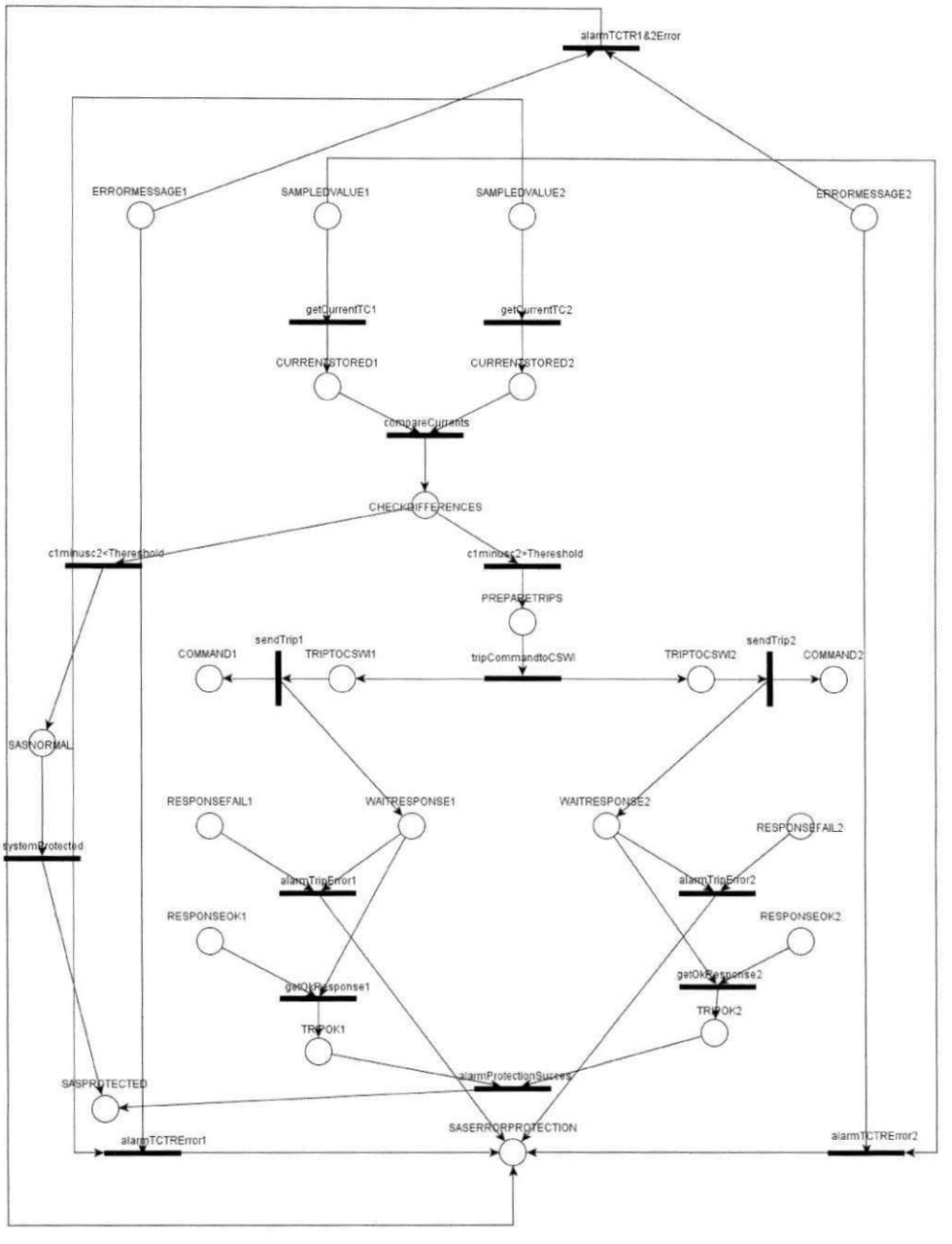
```
mod XCBR is
sorts Place Marking .
subsort Place < Marking .
ops


XCBR.COMMAND
XCBR.TRIPXCBR
XCBR.XCBRTRIPPED
XCBR.TRIPPINGERROR
XCBR.COMMANDRESPONSEOK
XCBR.COMMANDRESPONSEERROR
XCBR.OPENED
XCBR.CLOSED

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl [XCBR.getTripCommand]           : XCBR.COMMAND       =>
XCBR.TRIPXCBR .
rl [XCBR.tripXCBR]                 : XCBR.TRIPXCBR XCBR.CLOSED =>
XCBR.XCBRTRIPPED .
rl [XCBR.errorOpening]             : XCBR.TRIPXCBR XCBR.OPENED =>
XCBR.TRIPPINGERROR .
rl [XCBR.sendTripOpeningSuccess]   : XCBR.XCBRTRIPPED   =>
XCBR.COMMANDRESPONSEOK .
rl [XCBR.sendTripOpeningError]     : XCBR.TRIPPINGERROR =>
XCBR.COMMANDRESPONSEERROR .

endm
```

CSWI.maude


COMMANDIN
getInputCommand
PREPARETOSEND
sendCommandToOutputDevice
COMMANDOUT
COMMANDRESPONSEERROR
WAITRESPONSE
COMMANDRESPONSEOK
getCommandResponseError
getCommandResponseOK
ERRORRESPONSERECIEVED
OKRESPONSERECIEVED
sendCommandResponseError
sendCommandResponseOK
ERRORRESPONSE
OKRESPONSE

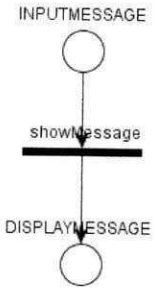
```
mod CSWI is
sorts Place Marking .
subsort Place < Marking .
ops


CSWI.COMMANDIN
CSWI.COMMANDOUT
CSWI.PREPARETOSEND
CSWI.COMMANDRESPONSEERROR
CSWI.COMMANDRESPONSEOK
CSWI.WAITRESPONSE
CSWI.ERRORRESPONSERECIEVED
CSWI.OKRESPONSERECIEVED
CSWI.ERRORRESPONSE
CSWI2.OKRESPONSE

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl [CSWI.getInputCommand]            :  CSWI.COMMANDIN
=>  CSWI.PREPARETOSEND   .
rl [CSWI.sendCommandToOutputDevice]  :  CSWI.PREPARETOSEND     =>
CSWI.COMMANDOUT CSWI.WAITRESPONSE .
rl [CSWI.getCommandResponseError]    :  CSWI.COMMANDRESPONSEERROR
CSWI.WAITRESPONSE =>  CSWI.ERRORRESPONSERECIEVED .
rl [CSWI.getCommandResponseOK]       :  CSWI.COMMANDRESPONSEOK
CSWI.WAITRESPONSE =>  CSWI.OKRESPONSERECIEVED .
rl [CSWI.sendCommandResponseError]   :  CSWI.ERRORRESPONSERECIEVED
=>  CSWI.ERRORRESPONSE .
rl [CSWI.sendCommandResponseOK]      :  CSWI.OKRESPONSERECIEVED
=>  CSWI.OKRESPONSE .


endm
```

PDIF.maude


alarmTCTR1&2Error
ERRORMESSAGE1
SAMPLEDVALUE1
SAMPLEDVALUE2
ERRORMESSAGE2
getCurrentTC1
getCurrentTC2
CURRENTSTORED1
CURRENTSTORED2
compareCurrents
CHECKDIFFERENCES
c1minusc2<Thereshold
c1minusc2>Thereshold
PREPARETRIPS
sendTrip1
COMMAND1
TRIPTOCSWI1
tripCommandtoCSWI
TRIPTOCSWI2
sendTrip2
COMMAND2
SASNORMAL
RESPONSEFAIL1
WAITRESPONSE1
WAITRESPONSE2
RESPONSEFAIL2
systemProtected
alarmTripError1
alarmTripError2
RESPONSEOK1
RESPONSEOK2
getOkResponse1
getOkResponse2
TRIPOK1
TRIPOK2
alarmProtectionSucces
SASPROTECTED
SASERRORPROTECTION
alarmTCTRError1
alarmTCTRError2

```
mod PDIF is
sorts Place Marking .
subsort Place < Marking .
ops

PDIF.ERRORMESSAGE1  PDIF.SAMPLEDVALUE1 PDIF.SAMPLEDVALUE2
PDIF.ERRORMESSAGE2 PDIF.CURRENTSTORED1  PDIF.CURRENTSTORED2
PDIF.CHECKDIFFERENCES  PDIF.PREPARETRIPS PDIF.SASPROTECTED
PDIF.SASERRORPROTECTION  PDIF.TRIPTOCSWI1  PDIF.COMMAND1
PDIF.WAITRESPONSE1  PDIF.RESPONSEFAIL1 PDIF.RESPONSEOK1  PDIF.TRIPOK1
PDIF.TRIPTOCSWI2  PDIF.COMMAND2 PDIF.WAITRESPONSE2  PDIF.RESPONSEFAIL2
PDIF.RESPONSEOK2  PDIF.TRIPOK2 PDIF.SASNORMAL

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl [PDIF.alarmTCTR1&2Error]          : PDIF.ERRORMESSAGE1
PDIF.ERRORMESSAGE2   => PDIF.SASERRORPROTECTION .
rl [PDIF.getCurrentTC1]              : PDIF.SAMPLEDVALUE1
=> PDIF.CURRENTSTORED1 .
rl [PDIF.getCurrentTC2]              : PDIF.SAMPLEDVALUE2
=> PDIF.CURRENTSTORED2 .
rl [PDIF.compareCurrents]            : PDIF.CURRENTSTORED1
PDIF.CURRENTSTORED2 => PDIF.CHECKDIFFERENCES .
rl [PDIF.c1minusc2>Thereshold]       : PDIF.CHECKDIFFERENCES
=> PDIF.PREPARETRIPS .
rl [PDIF.c1minusc2<Thereshold]       : PDIF.CHECKDIFFERENCES
=> PDIF.SASNORMAL .
rl [PDIF.alarmTCTRError1]            : PDIF.ERRORMESSAGE1
PDIF.SAMPLEDVALUE2   => PDIF.SASERRORPROTECTION .
rl [PDIF.alarmTCTRError2]            : PDIF.ERRORMESSAGE2
PDIF.SAMPLEDVALUE1   => PDIF.SASERRORPROTECTION .
rl [PDIF.tripCommandtoCSWI]          : PDIF.PREPARETRIPS
=> PDIF.TRIPTOCSWI1 PDIF.TRIPTOCSWI2 .
rl [PDIF.alarmProtectionSuccess]      : PDIF.TRIPOK1 PDIF.TRIPOK2
=> PDIF.SASPROTECTED .
rl [PDIF.systemProtected]            : PDIF.SASNORMAL
=> PDIF.SASPROTECTED .
rl [PDIF.sendTrip1]                  : PDIF.TRIPTOCSWI1 =>   PDIF.COMMAND1
PDIF.WAITRESPONSE1 .
rl [PDIF.sendTrip2]                  : PDIF.TRIPTOCSWI2 =>   PDIF.COMMAND2
PDIF.WAITRESPONSE2 .
rl [PDIF.alarmTripError1]            : PDIF.WAITRESPONSE1
PDIF.RESPONSEFAIL1 => PDIF.SASERRORPROTECTION .
rl [PDIF.alarmTripError2]            : PDIF.WAITRESPONSE2
PDIF.RESPONSEFAIL2 => PDIF.SASERRORPROTECTION .
rl [PDIF.getOkResponse1]             : PDIF.WAITRESPONSE1 PDIF.RESPONSEOK1
=> PDIF.TRIPOK1 .
rl [PDIF.getOkResponse2]             : PDIF.WAITRESPONSE2 PDIF.RESPONSEOK2
=> PDIF.TRIPOK2 .
endm
```

IHMI.maude




```
mod IHMI is
sorts Place Marking .
subsort Place < Marking .
ops

IHMI1.INPUTMESSAGE
IHMI1.DISPLAYMESSAGE

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl [IHMI1.showMessage]                    : IHMI1.INPUTMESSAGE     =>
IHMI1.DISPLAYMESSAGE .

endm
```

Rede de Petri Integrada – Formato Maude.

Arquivo resultante do algoritmo de integração de Redes de Petri. Esse arquivo contém todas os lugares e transições das Redes individuais bem como as transições que foram criadas para integrar estas redes.

```
mod PDIF_FINAL is
sorts Place Marking .
subsort Place < Marking .
ops

IHMI1.INPUTMESSAGE
IHMI1.DISPLAYMESSAGE

PDIF1.ERRORMESSAGE1  PDIF1.SAMPLEDVALUE1 PDIF1.SAMPLEDVALUE2
PDIF1.ERRORMESSAGE2 PDIF1.CURRENTSTORED1 PDIF1.CURRENTSTORED2
PDIF1.CHECKDIFFERENCES PDIF1.PREPARETRIPS PDIF1.SASPROTECTED
PDIF1.SASERRORPROTECTION  PDIF1.TRIPTOCSWI1 PDIF1.COMMAND1
PDIF1.WAITRESPONSE1  PDIF1.RESPONSEFAIL1  PDIF1.RESPONSEOK1
PDIF1.TRIPOK1  PDIF1.TRIPTOCSWI2 PDIF1.COMMAND2
PDIF1.WAITRESPONSE2 PDIF1.RESPONSEFAIL2 PDIF1.RESPONSEOK2
PDIF1.TRIPOK2 PDIF1.SASNORMAL


CSWI1.COMMANDIN CSWI1.COMMANDOUT CSWI1.PREPARETOSEND
CSWI1.COMMANDRESPONSEERROR CSWI1.COMMANDRESPONSEOK
CSWI1.WAITRESPONSE CSWI1.ERRORRESPONSERECIEVED
CSWI1.OKRESPONSERECIEVED CSWI1.ERRORRESPONSE
CSWI1.OKRESPONSE


CSWI2.COMMANDIN
CSWI2.COMMANDOUT
CSWI2.PREPARETOSEND
CSWI2.COMMANDRESPONSEERROR
CSWI2.COMMANDRESPONSEOK
CSWI2.WAITRESPONSE
CSWI2.ERRORRESPONSERECIEVED
CSWI2.OKRESPONSERECIEVED
CSWI2.ERRORRESPONSE
CSWI2.OKRESPONSE
```

```
XCBR1.COMMAND
XCBR1.TRIPXCBR
XCBR1.XCBRTRIPPED
XCBR1.TRIPPINGERROR
XCBR1.COMMANDRESPONSEOK
```

```
XCBR1.COMMANDRESPONSEERROR
XCBR1.OPENED
XCBR1.CLOSED


XCBR2.COMMAND
XCBR2.TRIPXCBR
XCBR2.XCBRTRIPPED
XCBR2.TRIPPINGERROR
XCBR2.COMMANDRESPONSEOK
XCBR2.COMMANDRESPONSEERROR
XCBR2.OPENED
XCBR2.CLOSED

TCTR1.GETCURRENTINPUT
TCTR1.CURRENTTRANSFORMED
TCTR1.TCTRERROR
TCTR1.ERRORMESSAGE
TCTR1.SAMPLEDVALUE
TCTR1.PREPARETRANSFORMATION
TCTR1.READERROR
TCTR1.VALIDATIONERROR
TCTR1.TRANSFORMATIONERROR

TCTR2.GETCURRENTINPUT
TCTR2.CURRENTTRANSFORMED
TCTR2.TCTRERROR
TCTR2.ERRORMESSAGE
TCTR2.SAMPLEDVALUE
TCTR2.PREPARETRANSFORMATION
TCTR2.READERROR
TCTR2.VALIDATIONERROR
TCTR2.TRANSFORMATIONERROR

: -> Place .
op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

rl [IHMI1.showMessage]                    : IHMI1.INPUTMESSAGE    =>
IHMI1.DISPLAYMESSAGE .

rl  [PDIF1.alarmTCTR1&2Error]                 :  PDIF1.ERRORMESSAGE1
PDIF1.ERRORMESSAGE2   => PDIF1.SASERRORPROTECTION .
```

```
rl  [PDIF1.getCurrentTC1]                      :  PDIF1.SAMPLEDVALUE1
=> PDIF1.CURRENTSTORED1 .
rl  [PDIF1.getCurrentTC2]                      :  PDIF1.SAMPLEDVALUE2
=> PDIF1.CURRENTSTORED2 .
rl  [PDIF1.compareCurrents]                   :  PDIF1.CURRENTSTORED1
PDIF1.CURRENTSTORED2 => PDIF1.CHECKDIFFERENCES .
rl  [PDIF1.c1minusc2>Thereshold]           :  PDIF1.CHECKDIFFERENCES
=> PDIF1.PREPARETRIPS .
```

```
rl [PDIF1.c1minusc2<Thereshold]          : PDIF1.CHECKDIFFERENCES
=> PDIF1.SASNORMAL .
rl [PDIF1.alarmTCTRError1]                  : PDIF1.ERRORMESSAGE1
PDIF1.SAMPLEDVALUE2   => PDIF1.SASERRORPROTECTION .
rl [PDIF1.alarmTCTRError2]                  : PDIF1.ERRORMESSAGE2
PDIF1.SAMPLEDVALUE1   => PDIF1.SASERRORPROTECTION .
rl [PDIF1.tripCommandtoCSWI]                : PDIF1.PREPARETRIPS
=> PDIF1.TRIPTOCSWI1 PDIF1.TRIPTOCSWI2 .
rl  [PDIF1.alarmProtectionSuccess]              :  PDIF1.TRIPOK1
PDIF1.TRIPOK2              => PDIF1.SASPROTECTED .
rl [PDIF1.systemProtected]                      : PDIF1.SASNORMAL
=> PDIF1.SASPROTECTED .
rl [PDIF1.sendTrip1]                        : PDIF1.TRIPTOCSWI1 =>
PDIF1.COMMAND1 PDIF1.WAITRESPONSE1 .
rl [PDIF1.sendTrip2]                        : PDIF1.TRIPTOCSWI2 =>
PDIF1.COMMAND2 PDIF1.WAITRESPONSE2 .
rl [PDIF1.alarmTripError1]                  : PDIF1.WAITRESPONSE1
PDIF1.RESPONSEFAIL1 => PDIF1.SASERRORPROTECTION .
rl [PDIF1.alarmTripError2]                  : PDIF1.WAITRESPONSE2
PDIF1.RESPONSEFAIL2 => PDIF1.SASERRORPROTECTION .
rl [PDIF1.getOkResponse1]                   : PDIF1.WAITRESPONSE1
PDIF1.RESPONSEOK1 => PDIF1.TRIPOK1 .
rl [PDIF1.getOkResponse2]                   : PDIF1.WAITRESPONSE2
PDIF1.RESPONSEOK2 => PDIF1.TRIPOK2 .


rl [CSWI1.getInputCommand]                      :   CSWI1.COMMANDIN
=>  CSWI1.PREPARETOSEND   .
rl [CSWI1.sendCommandToOutputDevice]   :   CSWI1.PREPARETOSEND =>
CSWI1.COMMANDOUT  CSWI1.WAITRESPONSE .
rl      [CSWI1.getCommandResponseError]                          :
CSWI1.COMMANDRESPONSEERROR          CSWI1.WAITRESPONSE          =>
CSWI1.ERRORRESPONSERECIEVED .
rl [CSWI1.getCommandResponseOK]         :  CSWI1.COMMANDRESPONSEOK
CSWI1.WAITRESPONSE =>  CSWI1.OKRESPONSERECIEVED .
rl       [CSWI1.sendCommandResponseError]                        :
CSWI1.ERRORRESPONSERECIEVED                                      =>
CSWI1.ERRORRESPONSE .
rl [CSWI1.sendCommandResponseOK]       :  CSWI1.OKRESPONSERECIEVED
=> CSWI1.OKRESPONSE .


rl  [CSWI2.getInputCommand]                     :   CSWI2.COMMANDIN
=>  CSWI2.PREPARETOSEND   .
rl  [CSWI2.sendCommandToOutputDevice]    :    CSWI2.PREPARETOSEND
=>  CSWI2.COMMANDOUT CSWI2.WAITRESPONSE .
rl      [CSWI2.getCommandResponseError]                          :
CSWI2.COMMANDRESPONSEERROR          CSWI2.WAITRESPONSE          =>
CSWI2.ERRORRESPONSERECIEVED .
rl [CSWI2.getCommandResponseOK]         :  CSWI2.COMMANDRESPONSEOK
CSWI2.WAITRESPONSE =>  CSWI2.OKRESPONSERECIEVED .
```

```
rl        [CSWI2.sendCommandResponseError]                              :
CSWI2.ERRORRESPONSERECIEVED                                             =>
CSWI2.ERRORRESPONSE .
rl [CSWI2.sendCommandResponseOK]        :   CSWI2.OKRESPONSERECIEVED
=>  CSWI2.OKRESPONSE .

rl [XCBR1.getTripCommand]               : XCBR1.COMMAND          =>
XCBR1.TRIPXCBR .
rl [XCBR1.tripXCBR]                    : XCBR1.TRIPXCBR XCBR1.CLOSED
=> XCBR1.XCBRTRIPPED .
rl [XCBR1.errorOpening]                : XCBR1.TRIPXCBR XCBR1.OPENED
=> XCBR1.TRIPPINGERROR .
rl  [XCBR1.sendTripOpeningSuccess]     :  XCBR1.XCBRTRIPPED       =>
XCBR1.COMMANDRESPONSEOK .
rl  [XCBR1.sendTripOpeningError]         :  XCBR1.TRIPPINGERROR  =>
XCBR1.COMMANDRESPONSEERROR .

rl [XCBR2.getTripCommand]               : XCBR2.COMMAND          =>
XCBR2.TRIPXCBR .
rl [XCBR2.tripXCBR]                    : XCBR2.TRIPXCBR XCBR2.CLOSED
=> XCBR2.XCBRTRIPPED .
rl [XCBR2.errorOpening]                : XCBR2.TRIPXCBR XCBR2.OPENED
=> XCBR2.TRIPPINGERROR .
rl  [XCBR2.sendTripOpeningSuccess]     :  XCBR2.XCBRTRIPPED       =>
XCBR2.COMMANDRESPONSEOK .
rl  [XCBR2.sendTripOpeningError]         :  XCBR2.TRIPPINGERROR  =>
XCBR2.COMMANDRESPONSEERROR .

rl[TCTR1.getCurrentInput<TCTR_CIN>]        : TCTR1.GETCURRENTINPUT
=> TCTR1.PREPARETRANSFORMATION  .
rl[TCTR1.transformCurrent]                                              :
TCTR1.PREPARETRANSFORMATION   => TCTR1.CURRENTTRANSFORMED .
rl[TCTR1.sendSampledCurrent]             : TCTR1.CURRENTTRANSFORMED
=> TCTR1.SAMPLEDVALUE .
rl[TCTR1.validationError]                                               :
TCTR1.PREPARETRANSFORMATION   => TCTR1.VALIDATIONERROR .
rl[TCTR1.readingError]                                                  :
TCTR1.PREPARETRANSFORMATION   => TCTR1.READERROR .
rl[TCTR1.transformationError]                                           :
TCTR1.PREPARETRANSFORMATION   => TCTR1.TRANSFORMATIONERROR .
rl[TCTR1.sendReadErrorMessage]                    :  TCTR1.READERROR
=> TCTR1.ERRORMESSAGE .
rl[TCTR1.sendValidationErrorMessage]       :  TCTR1.VALIDATIONERROR
=> TCTR1.ERRORMESSAGE .
rl[TCTR1.sendTransformationErrorMessage]:
TCTR1.TRANSFORMATIONERROR     => TCTR1.ERRORMESSAGE .

rl[TCTR2.getCurrentInput<TCTR_CIN>]        : TCTR2.GETCURRENTINPUT
=> TCTR2.PREPARETRANSFORMATION  .
rl[TCTR2.transformCurrent]                                              :
TCTR2.PREPARETRANSFORMATION   => TCTR2.CURRENTTRANSFORMED .
```

```
rl[TCTR2.sendSampledCurrent]              : TCTR2.CURRENTTRANSFORMED
=> TCTR2.SAMPLEDVALUE .
rl[TCTR2.validationError]                                          :
TCTR2.PREPARETRANSFORMATION    => TCTR2.VALIDATIONERROR .
rl[TCTR2.readingError]                                             :
TCTR2.PREPARETRANSFORMATION    => TCTR2.READERROR .
rl[TCTR2.transformationError]                                      :
TCTR2.PREPARETRANSFORMATION    => TCTR2.TRANSFORMATIONERROR .
rl[TCTR2.sendReadErrorMessage]                    :  TCTR2.READERROR
=> TCTR2.ERRORMESSAGE .
rl[TCTR2.sendValidationErrorMessage]       :  TCTR2.VALIDATIONERROR
=> TCTR2.ERRORMESSAGE .
rl[TCTR2.sendTransformationErrorMessage]:
TCTR2.TRANSFORMATIONERROR      => TCTR2.ERRORMESSAGE .

rl [s1]                                :             TCTR1.SAMPLEDVALUE
=> PDIF1.SAMPLEDVALUE1 .
rl [s2]                                :             TCTR2.SAMPLEDVALUE
=> PDIF1.SAMPLEDVALUE2 .
rl [s3]                                :             TCTR1.ERRORMESSAGE
=> PDIF1.ERRORMESSAGE1 .
rl [s4]                                :             TCTR2.ERRORMESSAGE
=> PDIF1.ERRORMESSAGE2 .
rl [s5]                                 :              CSWI2.COMMANDOUT
=> XCBR2.COMMAND .
rl [s6]                                 :              CSWI1.COMMANDOUT
=> XCBR1.COMMAND .
rl [s7]                               :             CSWI2.ERRORRESPONSE
=> PDIF1.RESPONSEFAIL2 .
rl [s8]                               :             CSWI1.ERRORRESPONSE
=> PDIF1.RESPONSEFAIL1 .
rl [s9]                                 :              CSWI2.OKRESPONSE
=> PDIF1.RESPONSEOK2  .
rl [s10]                                :              CSWI1.OKRESPONSE
=> PDIF1.RESPONSEOK1  .
rl [s11]                         :          XCBR2.COMMANDRESPONSEERROR
=> CSWI2.COMMANDRESPONSEERROR  .
rl [s12]                         :          XCBR1.COMMANDRESPONSEERROR
=> CSWI1.COMMANDRESPONSEERROR  .
rl [s13]                           :           XCBR2.COMMANDRESPONSEOK
=> CSWI2.COMMANDRESPONSEOK    .
rl [s14]                           :           XCBR1.COMMANDRESPONSEOK
=> CSWI1.COMMANDRESPONSEOK   .
rl [s15]                                :              PDIF1.COMMAND1
=>  CSWI1.COMMANDIN  .
rl [s16]                                :              PDIF1.COMMAND2
=>  CSWI2.COMMANDIN  .
*** rl [s15]                           :             CSWI2.OKRESPONSE
=>  IHMI1.INPUTMESSAGE  .
*** rl [s16]                           :             CSWI1.OKRESPONSE
=>  IHMI1.INPUTMESSAGE  .
```

```
*** rl [s17]                          :            CSWI2.ERRORRESPONSE
=>  IHMI1.INPUTMESSAGE  .
*** rl [s18]                          :            CSWI2.ERRORRESPONSE
=>  IHMI1.INPUTMESSAGE  .
rl [s19]                            :             PDIF1.SASPROTECTED
=>  IHMI1.INPUTMESSAGE  .
rl [s20]                       :           PDIF1.SASERRORPROTECTION
=>  IHMI1.INPUTMESSAGE  .
Any:Place .

endm

search

TCTR1.GETCURRENTINPUT
TCTR2.GETCURRENTINPUT
XCBR1.CLOSED
XCBR2.CLOSED

=>! Any:Marking .

show search graph .
quit
```

Anexo C.

Conjunto de Regras de Comunicação entre LN

Os LN existentes na norma IEC 61850 possuem um padrão de comunicação. Alguns enviam sinais analógicos, digitais, mensagens, e alertas. Para evitar que um LN se comunique com um outro LN que não possuem o envio/recebimento de mensagens padronizadas, foram criados um conjunto de regras de comunicação existentes para cada LN. Essas regras são utilizadas pelo algoritmo de integração para permitir que cada LN possa transmitir suas mensagens corretamente.

1. TCTR

Lê valores amostrais de corrente do dispositivo real, e envia o valor devidamente transformado que pode ser lido pelo componente conectado, ou uma mensagem de erro

Conjunto de regras:

```
TCTR. regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo PDIS) {
Integratdor.adicionaRegra(this, destinyInstance)}
2. IF(destinyInstance  é tipo MMTR){
Integratdor.adicionaRegra(this, destinyInstance)}
3. IF(destinyInstance  é tipo MMXU){
Integratdor.adicionaRegra(this, destinyInstance)}
4. IF(destinyInstance  é tipo RDRE){
Integratdor.adicionaRegra(this, destinyInstance)}
5. IF(destinyInstance  é tipo PDIF){
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

2. XCBR

O disjuntor recebe comandos de trip enviados por LN como PTRC, CPOW ou CSWI. Esses comandos de trip abrem o disjuntor se estiver em estado fechado, isolando o circuito. E respondem se o procedimento ocorreu com sucesso, ou com falha

Conjunto de regras

```
XCBR. regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo CSWI) {
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

3. CSWI

O LN de controle de chaveamento atua nas operações do equipamento do sistema primário que partem dos operadores e de automação relacionada. Checa a autorização, supervisiona a execução de comandos e envia alarmes em caso de fim de comando impróprio. Recebe um comando a ser executado e o repassa ao componente conectado, também recebe a resposta desse comando enviado, se for sucesso ou falha. Qualquer ação é envia a IHMI.

Conjunto de regras:

```
CSWI. regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo XCBR) {
Integratdor.adicionaRegra(this, destinyInstance)}
2. IF(destinyInstance  é tipo PDIF){
Integratdor.adicionaRegra(this, destinyInstance)}
3. IF(destinyInstance  é tipo CPOW){
Integratdor.adicionaRegra(this, destinyInstance)}
4. IF(destinyInstance  é tipo IHMI){
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

4. PDIF

A proteção diferencial atua com base em dois valores lidos pelos transformadores de corrente ligados a um transformador base. Se o valor dos terminais dos transformadores de corrente forem diferentes a proteção diferencial é ativada. Recebe valores de corrente de dois transformadores, e a resposta de cada trip enviado aos disjuntores, pelo LN CSWI e envia qualquer notificação a IMHI

Conjunto de regras:

```
PDIF. regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo IHMI) {
Integratdor.adicionaRegra(this, destinyInstance)}
2. IF(destinyInstance  é tipo CSWI){
Integratdor.adicionaRegra(this, destinyInstance)}
3. IF(destinyInstance  é tipo TCTR){
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

4. IHMI

O LN de interface, repassa mensagens e relatórios recebidos pelos componentes, no dispositivo de interface homem-máquina.

Conjunto de regras:

```
IHMI. regrasComunicacao (destinyInstance){
1. IF(destinyInstance  é tipo PDIF) {
Integratdor.adicionaRegra(this, destinyInstance)}
2. IF(destinyInstance  é tipo CSWI){
Integratdor.adicionaRegra(this, destinyInstance)}
}
```

Anexo D

Descrição de LN

Os arquivos de descrição de LN mostram a descrição de todas as ações que um LN pode efetuar. Alguns dessas ações podem ou não estar presentes no caso de teste final. Por isso é fundamental que o arquivo de descrição contenha todos as arestas definidas. Esses arquivos são utilizados pela ferramenta de seleção de casos de testes para obtenção do *script* de teste.

```
getCurrentInput<TCTR_CIN> [ Lê valor amostral de corrente e o
armazena ]
transformCurrent [ Realiza a transformação da corrente
padronizando-a para manipulação ]
sendSampledCurrent [ Com a corrente devidamente  padronizada ela é
enviada para o componente conectado ]
validationError [ ocorreu um erro ao receber o valor amostral de
corrente ]
readingError [ ocorreu um erro de leitura do valor de corrente ]
transformationError [ ocorreu um erro de transformacao da corrente
]
sendReadErrorMessage  [ envia uma mensagem de erro de leitura ao
componente conectado ]
sendValidationErrorMessage  [ envia uma mensagem de erro de
validacao ao componente conectado ]
sendTransformationErrorMessage  [ envia  uma  mensagem  de  error  de
transformacao ao componente conectado ]
```

Descrição 1 – LN TCTR

```
showMessage [Mostra mensagem no dispositivo de saída]
```

Descrição 2 – LN IHMI

```
getInputCommand        [Recebe comando de entrada e prepara para
enviar para o componente conectado]
sendCommandToOutputDevice    [Envia o comando de entrada para o
componente conectado e espera resposta]
getCommandResponseError       [Recebe resposta de erro do comando
enviado]
getCommandResponseOK           [Recebe resposta de sucesso do
comando enviado]
sendCommandResponseError      [Envia mensagem de erro para o
componente que enviou o comando]
sendCommandResponseOK        [Envia mensagem de sucesso para o
componente que enviou o comando]
```

Descrição 3 – LN CSWI

```
alarmTCTR1&2Error     [ Os dois valores de correntes recebidos
pelos transformadores estão inválidos]
getCurrentTC1     [ Recebe e armazena o valor de corrente do
tranformador 1 conectado ]
getCurrentTC2     [ Recebe e armazena o valor de corrente do
tranformador 2 conectado ]
compareCurrents    [ Compara os dois valores de correntes lidos ]
c1minusc2>Thereshold  [ O valor é maior que o ajuste da proteção
diferencial. A proteção é disparada]
c1minusc2<Thereshold    [ O valor das correntes é normal comparado
com o ajuste. A protação não é disparada]
alarmTCTRError1       [ Ocorre um erro ao obter a corrente no
transformador 1 conectado ]
alarmTCTRError2       [ Ocorre um erro ao obter a corrente no
transformador 2 conectado ]
tripCommandtoCSWI     [ prepara mensagem de trip para os Command
Switch conectados a PDIF]
alarmProtectionSuccess  [ Envia notificação que a proteção
diferencial foi realizada com sucesso]
systemProtected    [  Envia notificação que o sistema está
protegido ]
sendTrip1         [ Envia trip para o Command Switch 1 conectado ]
sendTrip2         [ Envia trip para o Command Switch 2 conectado ]
alarmTripError1    [ Envia notificação que ocorreu um erro no trip
enviado pelo Command Switch 1 ]
alarmTripError2    [ Envia notificação que ocorreu um erro no trip
enviado pelo Command Switch 2 ]
getOkResponse1    [ Recebe notificação que o trip enviado pelo
Command Switch 1 foi realizado com sucesso]
getOkResponse2 [ Recebe notificação que o trip enviado pelo
Command Switch 1 foi realizado com sucesso]
```

Descrição 4 – LN PDIF

```
getTripCommand  [ Recebe o comando para realizar um trip no
disjuntor ]
tripXCBR        [ O trip é efetuado com sucesso ]
errorOpening    [ Ocorreu um erro na manobra de trip ]
sendTripOpeningSuccess [ Envia uma notificação que o trip foi
realizado com sucesso ]
sendTripOpeningError   [ Envia um notificação que ocorreu um erro
durante o trip ]
```

Descrição 5 – LN XCBR

O que esta entre os '[' e ']' é a descrição textual da ação mostrada no início de cada linha. Esses arquivos de descrição permitem facilidade de expandir novos LN. À medida que novos LN são construídos, deve-se ter um arquivo de descrição para cada um.

Anexo E.

Resultados de Execução de Ferramentas

DFSonCTG.jar

O arquivo resultante desta busca em profundidade é um arquivo de texto contendo os casos de testes obtidos pela busca em profundidade do CTG. O início de um caso de teste é indicado por "TC #x" onde x indica o número referente ao caso de teste e "TC" é uma abreviação de TestCase. 'start:' indica o início do caso de teste, 'end' indica o fim do caso de testes e todos as linhas entre esses rótulos indicam as ações que esse caso de teste descreve. As ações que se iniciam por 's' seguindo por um número são ações que ocorrem entre o estado de saída de uma rede e o estado de entrada de uma rede. Os rótulos 'PASS' indicam que o teste chegou a um estado terminal e INCONCLUSIVE, indica que a partir do estado em questão o caso de teste não chega a um estado final previsto.

```
TC #1
start:
"TCTR1.getCurrentInput<TCTR_CIN>; INPUT"
"TCTR2.getCurrentInput<TCTR_CIN>; INPUT"
"TCTR1.transformCurrent; INPUT"
"TCTR2.transformCurrent; INPUT"
"TCTR1.sendSampledCurrent; INPUT"
"TCTR2.sendSampledCurrent; INPUT"
"s2; INPUT"
"PDIF1.getCurrentTC2; INPUT"
"s1; INPUT"
"PDIF1.getCurrentTC1; INPUT"
"PDIF1.compareCurrents; INPUT"
"PDIF1.c1minusc2>Thereshold; INPUT"
"PDIF1.tripCommandtoCSWI; INPUT"
"PDIF1.sendTrip1; INPUT"
"PDIF1.sendTrip2; INPUT"
"s15; INPUT"
"s16; INPUT"
"CSWI1.getInputCommand; INPUT"
"CSWI2.getInputCommand; INPUT"
"CSWI2.sendCommandToOutputDevice; INPUT"
"s5; INPUT"
"CSWI1.sendCommandToOutputDevice; INPUT"
"XCBR2.getTripCommand; INPUT"
"XCBR2.tripXCBR; INPUT"
"s6; INPUT"
"XCBR1.getTripCommand; INPUT"
"XCBR1.tripXCBR; INPUT"
"XCBR2.sendTripOpeningSuccess; INPUT"
"s13; INPUT"
"XCBR1.sendTripOpeningSuccess; INPUT"
"s14; INPUT"
"CSWI2.getCommandResponseOK; INPUT"
"CSWI1.getCommandResponseOK; INPUT"
"CSWI1.sendCommandResponseOK; INPUT"
"CSWI2.sendCommandResponseOK; INPUT"
"s9; INPUT"
"s10; INPUT"
"PDIF1.getOkResponse2; INPUT"
"PDIF1.getOkResponse1; INPUT"
"PDIF1.alarmProtectionSuccess; INPUT"
"s19; INPUT"
"IHMI1.showMessage; INPUT (PASS)"
end
```

Resultado 1 – Caso de teste 1

```
TC #2
start:
"TCTR1.getCurrentInput<TCTR_CIN>; INPUT"
"TCTR2.getCurrentInput<TCTR_CIN>; INPUT"
"TCTR1.transformCurrent; INPUT"
"TCTR2.transformCurrent; INPUT"
"TCTR1.sendSampledCurrent; INPUT"
"TCTR2.sendSampledCurrent; INPUT"
"s2; INPUT"
"PDIF1.getCurrentTC2; INPUT"
"s1; INPUT"
"PDIF1.getCurrentTC1; INPUT"
"PDIF1.compareCurrents; INPUT"
"PDIF1.c1minusc2>Thereshold; INPUT"
"PDIF1.tripCommandtoCSWI; INPUT"
"PDIF1.sendTrip1; INPUT"
"PDIF1.sendTrip2; INPUT"
"s15; INPUT"
"s16; INPUT"
"CSWI2.getInputCommand; INPUT"
"CSWI1.getInputCommand; INPUT"
"CSWI1.sendCommandToOutputDevice; INPUT"
"CSWI2.sendCommandToOutputDevice; INPUT"
"s5; INPUT"
"XCBR2.getTripCommand; INPUT"
"s6; INPUT"
"XCBR1.getTripCommand; INPUT"
"XCBR1.tripXCBR; INPUT"
"XCBR1.sendTripOpeningSuccess; INPUT"
"XCBR2.tripXCBR; INPUT"
"XCBR2.sendTripOpeningSuccess; INPUT"
"s13; INPUT"
"CSWI2.getCommandResponseOK; INPUT"
"s14; INPUT"
"CSWI1.getCommandResponseOK; INPUT"
"CSWI1.sendCommandResponseOK; INPUT"
"s10; INPUT"
"CSWI2.sendCommandResponseOK; INPUT"
"s9; INPUT"
"PDIF1.getOkResponse2; INPUT"
"PDIF1.getOkResponse1; INPUT"
"PDIF1.alarmProtectionSuccess; INPUT"
"s19; INPUT"
"IHMI1.showMessage; INPUT (PASS)"
end
```

Resultado 2 – Caso de teste 2