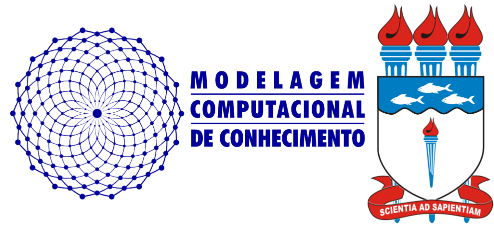

Dissertação de Mestrado

Gungnir - ferramenta para geração e execução automática de testes de conformidade utilizando Autômatos Temporizados

Rodrigo José Sarmento Peixoto
rodrigopex@gmail.com

Orientador:

Leandro Dias da Silva

Maceió, outubro de 2010

Rodrigo José Sarmento Peixoto

Gungnir - ferramenta para geração e execução automática de testes de conformidade utilizando Autômatos Temporizados

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Leandro Dias da Silva

Maceió, outubro de 2010

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Leandro Dias da Silva - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Examinador 1 - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Examinador 2 - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Maceió, outubro de 2010

Resumo

O objetivo deste trabalho é aumentar a confiança e a segurança de sistemas da automação através do uso de uma ferramenta de geração e execução automática de testes de conformidade. A ferramenta desenvolvida chama-se Gungnir e utiliza modelos, cujo padrão utilizado é o formalismo de Autômato Temporizado (AT), para executar suas ações. Os sistemas de controle são constituídos por Controladores Lógicos Programáveis (CLP) e normalmente são desenvolvidos nas linguagens Ladder e *Function Block Diagram* (FBD). A atividade chave da Gungnir é verificar se a implementação do sistema de controle desenvolvida na linguagem Ladder é compatível com a especificação modelada utilizando o padrão ISA 5.2. Para isso são utilizadas ferramentas de tradução de Ladder e ISA 5.2 para modelos AT.Os fatores diferenciais do projeto são o uso de heurísticas que minimizam o tempo e custo de execução dos testes e melhoram a eficácia dos mesmo, pois apenas traces factíveis serão testados e a limitação de pontos de explosão do espaço de estados, permitindo ao usuário controlar a abrangência dos testes em modelos de AT com espaço de estados expansíveis.

Abstract

My Abstract

Agradecimentos

Agradeço à ...

Rodrigo J. S. Peixoto

Conteúdo

1 Introdução **1**
1.1 Descrição do problema . . . 1
1.2 Objetivo do trabalho . . . 2
1.3 Resultados e relevância do trabalho . . . 2
1.4 Estrutura do Documento . . . 2

2 Fundamentação teórica **3**
2.1 Controlador lógico programável - CLP . . . 3
2.1.1 Detalhes da execução de CLPs . . . 5
2.1.2 Linguagens . . . 5
2.2 Autômatos temporizados . . . 6
2.2.1 Guardas . . . 7
2.2.2 Invariantes . . . 8
2.2.3 Eventos . . . 8
2.2.4 Atribuições . . . 8
2.2.5 Sincronizações . . . 8
2.3 Testes baseados em Modelos . . . 8
2.3.1 Vantagens . . . 11
2.3.2 Desvantagens . . . 13
2.4 Trabalho relacionados . . . 13
2.4.1 UPPAAL TRON . . . 14
2.4.2 Método para automatização de testes caixa preta . . . 15

3 Ferramenta Gungnir **19**
3.0.3 Formalismo . . . 21
3.0.4 Testes baseados em modelo . . . 22

4 Estudo de caso **30**
4.1 Controle de engarrafadora . . . 30
4.1.1 Utilizando o Gungnir . . . 30
4.1.2 Geração dos modelos . . . 32
4.1.3 Geração das entradas . . . 32
4.1.4 Execução dos testes . . . 34

5 Conclusões **39**
5.1 Trabalhos futuros . . . 40

A Projeto Gungnir **43**
A.1 Arquitetura geral do Gungnir 43
A.2 Arquivos XML 43

Lista de Figuras

2.1 Diagrama conceitual da aplicação de um CLP. 3
2.2 Arquitetura interna de um CLP. 4
2.3 Representação gráfica do *Scan Cycle*. 5
2.4 Representação gráfica do *Scan Cycle*. 5
2.5 Representação gráfica de uma realimentação. 6
2.6 Modelo do controlador de duplo-clique do mouse. 7
2.7 Modelo do clique do mouse. 7
2.8 Notação do diagrama do método TBM. 10
2.9 Diagrama do método: Testes baseados em modelos. 11
2.10 Diagrama de rastreabilidade. 12
2.11 Tela do construtor de modelos do UPPAAL. 15
2.12 Tela do simulador de modelos do UPPAAL. 16
2.13 Visão geral do método desenvolvido no trabalho de Vasconcelos Oliveira (2009). 17

3.1 Notação do diagrama do método TBM adaptado. 21
3.2 Diagrama do método: Testes baseados em modelos adaptado à aplicação de testes de conformidade para sistema de automação e controle. 22
3.3 (a) estado, (b) estado inicial, (c) estado *committed*, não consome tempo, (d) transição, (e) atribuição e (f) guarda. 23
3.4 Visão interna do Gungnir. 23
3.5 *ReadInputs* - exemplo de especificação ISA 5.2. 23
3.6 *ReadInputs* - exemplo do autômato responsável pela leitura das variáveis de entrada do CLP. 24
3.7 Temporizador Di (TON) - autômato responsável pela gerência dos valores de saída do temporizador timer1 da Figura-3.5. 24
3.8 Temporizador Dt (TOFF) - exemplo do autômato responsável pela gerência dos valores de saída do temporizador Dt genérico. 25
3.9 *WriteOutputs* - exemplo do autômato responsável pela escrita das variáveis de saída do CLP. 25
3.10 (a) *ReadInputs*, (b) *timer_DI*, (c) *WriteOutputs* e (d) *TimerUpdater*. 27
3.11 (a) Heurística para expressões do tipo "e", (b) Heurística para expressões do tipo "ou" 28
3.12 (a) expressão, (b) possíveis valores de A, B, C e D, (c) heurística do fator determinante para o caso, (d) resultado gerado a partir da heurística. 28
3.13 Screenshot do aplicativo Scanion com um exemplo de arquivo **.vcd**. 29

4.1 Engarrafadora - Fonte: Bryan & Bryan (1997*b*), página 485 31
4.2 Autômato *ReadInputs* 32

4.3 Autômato *Timer_Di01* 33
4.4 Autômato *Timer_Di02* 33
4.5 Autômato *WriteOutputs* 34
4.6 Autômato *TimeUpdater* 34
4.7 Tabelas de classes calculadas para as variáveis de saída através da heurística do fator determinante. 35
4.8 Classes selecionadas com a primeira geração de vetor de entrada aleatório. . . 36
4.9 Comportamento da engarrafadora para as entradas geradas. 36
4.10 Comportamento da implementação da engarrafadora com o Erro 1 injetado. . 37
4.11 Comportamento da implementação da engarrafadora com o Erro 2 injetado. . 37
4.12 Comportamento da implementação da engarrafadora com o Erro 3 injetado. . 38

A.1 Arquitetura geral simplificada do Gungnir 49

Lista de Tabelas

Lista de Códigos

A.1 Diagrama ISA 5.2 da Engarrafadora 43
A.2 Programa Ladder da Engarrafadora 45

Capítulo 1

Introdução

A crescente demanda do mercado faz com que as indústrias invistam mais em produtividade e qualidade. Isso é alcançado, normalmente, através de sistemas de automação industrial. Grandes empresas são obrigadas a automatizar a produção, a fim de maximizar lucros e minimizar custos. Um grande exemplo é a Petrobras - uma das grandes empresas no segmento da indústria de óleo, gás e energia no mundo. Hoje, conta uma grande quantidade de plataformas de extração em vários locais do Brasil e do mundo e com grande potencial de crescimento a partir da descoberta de petróleo e gás na região do pré-sal (2010*a*).

Empresas como a Petrobras devem aplicar várias normas internacionais em seus sistemas de automação, com a finalidade de obter um grau mínimo de segurança e confiança. Um dos equipamentos mais importantes da automatização da produção é o Controlador Lógico Programável (CLP) o qual é responsável por caracterizar o sistema automático. Através de sensores a atuadores os CLPs podem controlar a mais vasta gama de processos industriais. Alguns destes processos necessitam de maior cautela por tratarem de produtos perigosos e/ou caros. Sistemas são classificados como sistemas críticos quando falhas no controle produtivo podem gerar catástrofes.

Para evitar falhas em sistemas, várias técnicas são utilizadas que vão desde o cuidado na escolha das peças usadas na linha de produção ao teste exaustivo dos software o qual será executado dentro do CLP que controla a produção. O foco deste trabalho é produzir uma ferramenta para aperfeiçoar o processo de testes de softwares de controle que utilizam Ladder como linguagem de programação e diagramas ISA 5.2 para especificação.

1.1 Descrição do problema

Atualmente, os testes de software de CLPs são feitos por uma empresa especializada após o término do desenvolvimento do sistema de controle. Caso algum problema seja encontrado, o erro é relatado à equipe de desenvolvimento para assim serem feitas as mudanças necessárias.

continuar...

1.2 Objetivo do trabalho

Este trabalho de pesquisa está inserido no contexto do projeto SIS em parceria com a empresa Petrobras. O objetivo central deste trabalho é aumentar a segurança e a confiança de sistemas de controle. Com a finalidade de aproximar-se desse objetivo, será desenvolvida uma ferramenta capaz de, baseado em um modelo de AT, gerar casos de testes, para que possam ser realizados testes de conformidade.

Algumas normas exigem que os sistemas de controle sejam especificados em ISA 5.2. Através do trabalho de Vasconcelos Oliveira (2009) poderemos ter modelos de AT gerados a partir das especificações ISA 5.2 e do código Ladder. Com isso poderemos verificar se a implementação está de acordo com a especificação o que caracteriza testes de conformidade.

1.3 Resultados e relevância do trabalho

blah blah blah

1.4 Estrutura do Documento

No próximo capítulo, o Capítulo-2, serão apresentados todos os conceitos necessários para o entendimento deste trabalho e trabalhos relacionados. No Capítulo-3, será explicitado o conteúdo gerado por este trabalho: a ferramenta gerada e o processo de testes utilizando a ferramenta. Já no Capítulo-4, serão expostos estudos de caso utilizando a ferramenta: o controle de uma engarrafadora e o controle de semáforos. No Capítulo-5, serão feitas as conclusões e listados os trabalhos futuros os quais podem ser gerados a partir deste. Por fim, serão listadas as referências utilizadas neste trabalho.

Capítulo 2

Fundamentação teórica

Nesta seção, serão explicados conceitos básicos necessários para o entendimento desta dissertação. Estes são: controladores lógicos programáveis, autômatos temporizados, testes baseados em modelos e também serão citados alguns trabalhos relacionados ao corrente trabalho.

2.1 Controlador lógico programável - CLP

São dispositivos de controle largamente utilizados na indústria, com a finalidade de aumentar e melhorar a produção. Usam circuitos integrados ao invés de dispositivos eletromecânicos para implementar funções de controle. São capazes de armazenar instruções de: sequenciamento, temporização, contagem, aritmética, manipulação de dados e comunicação; tudo para possibilitar um controle eficiente de máquinas industriais e processos Bryan & Bryan (1997*a*). A Figura-2.2 mostra uma visão geral do uso de CLPs através de um diagrama conceitual.

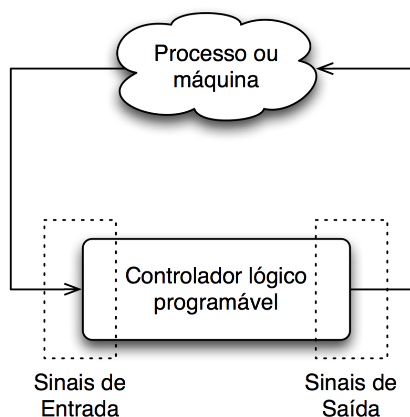

Figura 2.1: Diagrama conceitual da aplicação de um CLP.

Segundo Mader (2000), um CLP é basicamente formado por:

1. Uma CPU, unidade central de processamento, baseada em microprocessador;
2. Memória a qual possui áreas reservadas para dados das entradas, dados da saída e processamento;
3. Pontos de entrada e saída onde os sinais podem ser recebidos e enviados de/para o processo ou máquina respectivamente.

Normalmente, um CLP é equipado com um sistema operacional que permite o carregamento e execução de programas e auto-checagem automática Mader (2000). A Figura-2.2 mostra uma visão da arquitetura interna de um CLP.

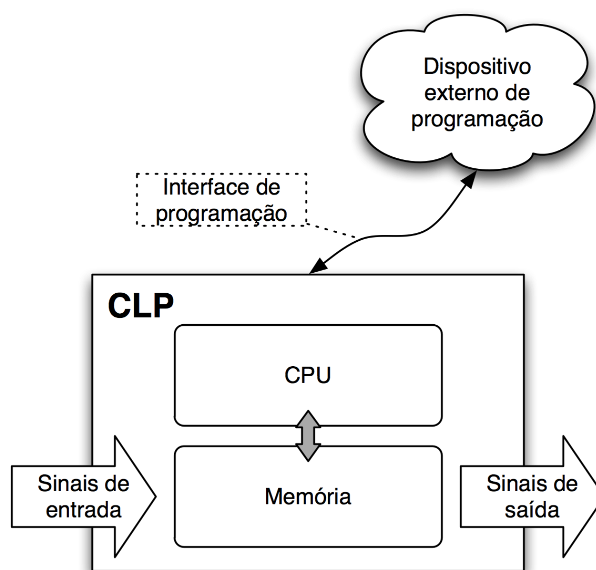

Figura 2.2: Arquitetura interna de um CLP.

Durante a operação de um CLP a CPU executa três tarefas básicas de forma cíclica:

1. Leitura, consiste na obtenção/aceitação das informações pela interface de entrada;
2. Processamento, executa o programa de controle armazenado no sistema de memória;
3. Escrita, atualiza as saídas do CLP via interfaces de saída.

Esse processo cíclico de leitura, processamento e escrita do CLP é chamado *Scan Cycle* Bryan & Bryan (1997*a*). Os ciclos variam de tamanho dependendo da complexidade do programa de controle. É possível observar um tempo máximo por ciclo, normalmente, na ordem de mile segundos Mader (2000). A Figura-2.3 mostrar duas representações do *Scan Cycle*.

Apesar de usualmente não fazer parte do CLP os dispositivos de programação e desenvolvimento são necessários. Não há interfaces gráficas nem meios de interação como teclados ou mouses com CLPs, portanto deve-se desenvolver toda a lógica em estações de trabalho

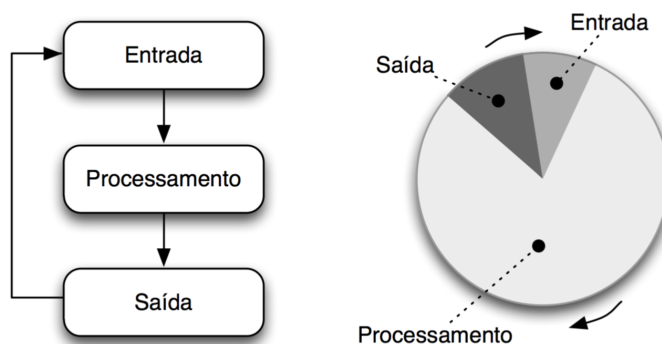

Figura 2.3: Representação gráfica do *Scan Cycle*.

para, no fim, carregar o programa de controle no controlador. Alguns fabricantes fornecem, junto aos CLPs, dispositivos de programação; outros permitem programação direta via porta serial, usb, entre outros meios Bryan & Bryan (1997*a*).

2.1.1 Detalhes da execução de CLPs

Segundo o padrão internacional IEC 61131-3 PLCopen (2004), um CLP lê os valores das placas de entrada, armazena nas variáveis de entrada globais. Processa os dados utilizando os valores previamente lido das placas de entrada, atualiza os valores de saída nas variáveis de saída globais. Por fim, as informações armazenadas nas variáveis de saída global serão escritas nas placas de saída do CLP caso seus valores sejam diferentes dos lá já armazenados.

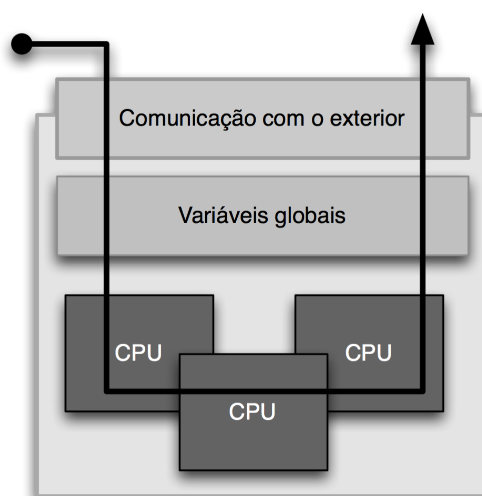

Figura 2.4: Representação gráfica do *Scan Cycle*.

A linhas de realimentação (*feedback*), Figura-2.5, são atualizadas no fim do *Scan Cycle*, ou seja, durante todo o *Scan Cycle* o valor da realimentação é o valor da saída no momento que as entradas são lidas.

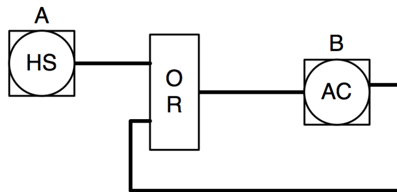

Figura 2.5: Representação gráfica de uma realimentação.

2.1.2 Linguagens

Há um padrão internacional que rege o uso de linguagens de programação para controladores lógicos programáveis, o IEC 61131-3 PLCopen (2004). Linguagens tratadas no padrão:

- Function Block Diagram;
- Ladder Diagram;
- Sequential Function Chart;
- Structured Text;
- Instruction List.

Para este trabalho, duas linguagens serão utilizadas seguindo os padrões e normas da Petrobras. Para especificação, será utilizada a norma ISA 5.2 e, para programação dos CLPs, será utilizada a linguagem Ladder. Ambas largamente utilizada em processos industriais em grandes empresas (Petrobras, por exemplo).

ISA 5.2

A fazer

Ladder

A fazer

2.2 Autômatos temporizados

Autômato temporizado é uma extensão da teoria dos autômatos finitos para modelar sistemas de tempo real Dong et al. (2008). As transições são instantâneas e possuem relógios associados, estes podem ser reiniciados independentemente do resto do sistema durante as transições. Todos os relógios são incrementados ao mesmo tempo e são definidos como números reais possibilitando, assim, a modelagem de sistemas contínuos Jr. et al. (1999).

Os autômatos temporizados utilizados nesta dissertação são do tipo: *Autômato temporizado de Muller* **(ATM)** - fechados sobre as operações booleanas e possuem grande poder expressivo Alur & Dill (1994). Nesta dissertação, quando falarmos de autômatos temporizados, estaremos nos referindo a um ATM. A definição formal dos ATMs é:

Autômato temporizado (ATM) é uma 6-tupla $(\Sigma, S, S_0, E, C, \mathscr{F})$ onde:

1. Σ é um conjunto finito chamado alfabeto de entrada;
2. S é um conjunto finito chamado estados;
3. $S_0 \in S$ é o estado inicial;
4. $E \subseteq S \times S \times \Sigma \times \mathscr{P}(C) \times \Phi(C)$ é o conjunto de arestas, análogo à função de transferência de autômatos finitos acrescido ao conjunto potência do conjunto de relógios $\mathscr{P}(C)$ e o conjunto de restrição de relógio $\Phi(C)$.
5. C é um conjunto finito de relógios;
6. $\mathscr{F}$ é a condição de aceitação $\mathscr{F} \subseteq \mathscr{P}(S)$. Consiste em uma família de estados que podem ser repetidos infinitamente, $\mathscr{P}(S)$ é o conjunto potência de S.

Outros aspectos importantes com relação aos autômatos temporizados os quais não estão na definição formal serão expostos a seguir. Para ilustrar os conceitos serão usadas as Figuras-2.6 e 2.7. O estado com um círculo interno é o estado inicial do autômato.

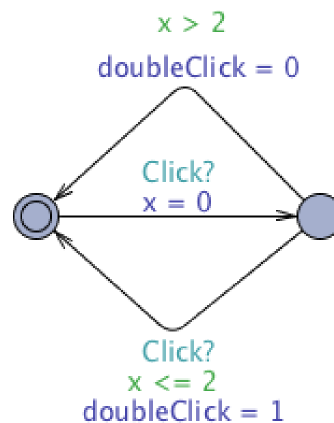

Figura 2.6: Modelo do controlador de duplo-clique do mouse.

2.2.1 Guardas

São restrições, normalmente, conjunções de expressões booleanas aplicadas aos relógios. Em algumas abordagens, há guardas relacionada com variáveis do sistema, como no caso do UPPAAL TRON Hessel et al. (2008) que variáveis de entrada e/ou saída podem ser expressas numa guarda. Como exemplo de guardas podemos citar as condições $x > 2$ e $x <= 2$ na Figura-2.6.

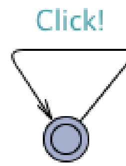

Figura 2.7: Modelo do clique do mouse.

2.2.2 Invariantes

São restrições temporais inerentes aos estados (localidades). Estas limitam o tempo de permanência do sistema em determinado estado. Caso o tempo máximo de permanência em um estado seja alcançado e ele o autômato permaneça no mesmo estado, há caracterização de uma falha.

2.2.3 Eventos

São quaisquer modificações nas variáveis ou canais de sincronização do sistema. Como exemplo podemos citar os seguintes eventos: na Figura-2.6 temos as etiquetas **Click?** e as mudanças dos relógios como eventos. Na Figura-2.7 podemos citar o **Click!** como evento emissor na sincronia, este causará eventos de sincronização no autômato da Figura-2.6. Os canais podem ser simples onde apenas um par de sincronia acontece em um instante de tempo ou ainda *broadcast* onde para um canal emitido podem ser sincronizados n canais.

2.2.4 Atribuições

São modificações que devem ser efetuadas ao fim da transição. Na Figura-2.6 podemos encontrar nas três transições atribuições: o reinício do relógio x e a atribuição dos valores zero $(doubleClick := 0)$ e um $(doubleClick := 1)$ à variável doubleClick.

2.2.5 Sincronizações

São etiquetas de sincronização onde n eventos aguardam um evento x ocorrer. A etiqueta emissora é representada por **Etiqueta!** e a etiqueta receptora por **Etiqueta?**. Os elementos podem esperar por atendimento, porém uma etiqueta não pode esperar um receptor, caso isso ocorra, é caracterizada a ocorrência de uma falha. Nas Figuras-2.6 e 2.7 temos como exemplo as etiquetas **Click?** e **Click!**

2.3 Testes baseados em Modelos

Testes de software consiste na verificação dinâmica do comportamento de um programa sobre um conjunto finito de casos de testes apropriadamente selecionados de um domínio normalmente infinito, contra o comportamento esperado do mesmo IEEE Computer Society (2004). Em resumo, conforme dito por Reid (2005), temos que teste de software consiste na execução de um programa com a intenção de encontrar erros. A execução de testes adiciona valor a um sistema, pois aumenta sua qualidade e confiabilidade. Há muitas aplicações que exigem um alto grau de confiabilidade, porquanto podem causar grandes prejuízos.

Este trabalho está pautado sobre a metodologia de testes chamada **Testes Baseados em Modelos** - **TBM** (do inglês, *Model-Based Testing*) que por definição é a **automação do projeto de testes caixa-preta** Utting & Legeard (2006). Os testes caixa preta são caracterizados pela geração manual de casos de testes baseados na especificação, pois não se tem acesso a detalhes como comportamento ou estrutura internos do sistema em teste (do inglês, *System Under Test* - **SUT**) no momento da geração dos casos de testes. Contudo, em TBM, há uma pequena diferença com relação ao uso de testes caixa-preta, porquanto é criado um modelo do comportamento esperado do SUT o qual detêm parte ou toda especificação do mesmo ao invés de utilizar a especificação diretamente. A partir deste modelo, pode-se, automaticamente, gerar uma vasta gama de casos de testes utilizando ferramentas apropriadas.

Um modelo é uma descrição simplificada de algo. Em TBM os modelos devem ser pequenos com relação ao tamanho do sistema, por conseguinte devem ser detalhados o suficiente para descrever características as quais devem ser testadas Reid (2005). Os modelos descritos pelo engenheiro serão utilizados como base para as ferramentas de geração de casos de testes e podem ser representados nas mais variadas formas. Abaixo temos algumas das formas de representação listadas:

- Diagramas UML enriquecidos com OCL (*Object Constraint Language*) ou máquinas de estado Utting & Legeard (2006);
- Redes de Petri Peterson (1977);
- Autômatos temporizados Alur & Dill (1994);
- EFSM - *Extended Finite State Machine* Utting & Legeard (2006).

Existe também o conceito de Oráculo (do inglês, *Oracle* - termo bastante referenciado no âmbito de testes) o qual é responsável por gerar as saídas esperadas baseado nas entradas. Desta forma fecha-se o ciclo da metodologia TBM, pois têm-se os modelos de descrição do SUT, os casos de testes gerados pelas ferramentas baseando-se nos modelos e os oráculos. A partir de então, deve-se executar os casos testes e avaliar se os dados externados pelo SUT são iguais aos determinados pelo oráculo. Se, para todos os casos de testes, essa premissa

for verdadeira, o veredicto dado deve ser PASSOU; em outro caso FALHOU. Utilizando a notação mostrada na Figura-2.8, a Figura-2.9 mostra uma visão geral do uso da abordagem caixa-preta em TBM.

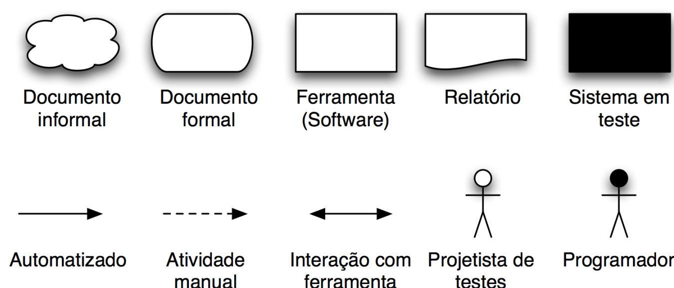

Figura 2.8: Notação do diagrama do método TBM.

A metodologia de testes baseados em modelos tem cinco passos principais:

1. **Criação do modelo** baseado nos requisitos e no modelo de desenvolvimento o projetista de testes deve criar um novo modelo do sistema e/ou do ambiente. Segundo Utting & Legeard (2006) é importante não reutilizar totalmente o modelo criado pela equipe de desenvolvimento, pois em alguns momentos são inadequados com relação ao nível de detalhamento e podem fazer com que erros de especificação perdurem por mais tempo;

2. **Geração dos casos de teste** abstratos. Nesta fase, normalmente, são gerados casos de testes abstratos. Destes deve-se fazer correlação direta com os requisitos para que no fim torne-se possível a construção de uma matriz de rastreabilidade dos requisitos nos testes o que possibilitará uma análise mais completa da cobertura dos testes;

3. **Concretização dos casos de teste** gerados para algo exequível. Após a geração dos casos de teste abstratos deve-se diminuir ainda mais o nível de abstração dos testes para que estes possam ser executados juntos ao SUT. Isso possibilita uma maior modularidade dos testes, por quanto caso haja alguma mudança mais drástica, como por exemplo mudança de linguagem de programação, os testes abstratos ainda servirão, bastando apenas mudar o código do adaptador;

4. **Execução dos casos de teste** concretos. Através de uma ferramenta de execução de testes são obtidos os relatórios dos resultados dos testes. Há, basicamente, duas formas de executar testes:

   online (*on-the-fly*) onde os testes são gerados e executados em sequência, não há registro do casos de testes, contudo registram-se os resultados. Nesta abordagem,

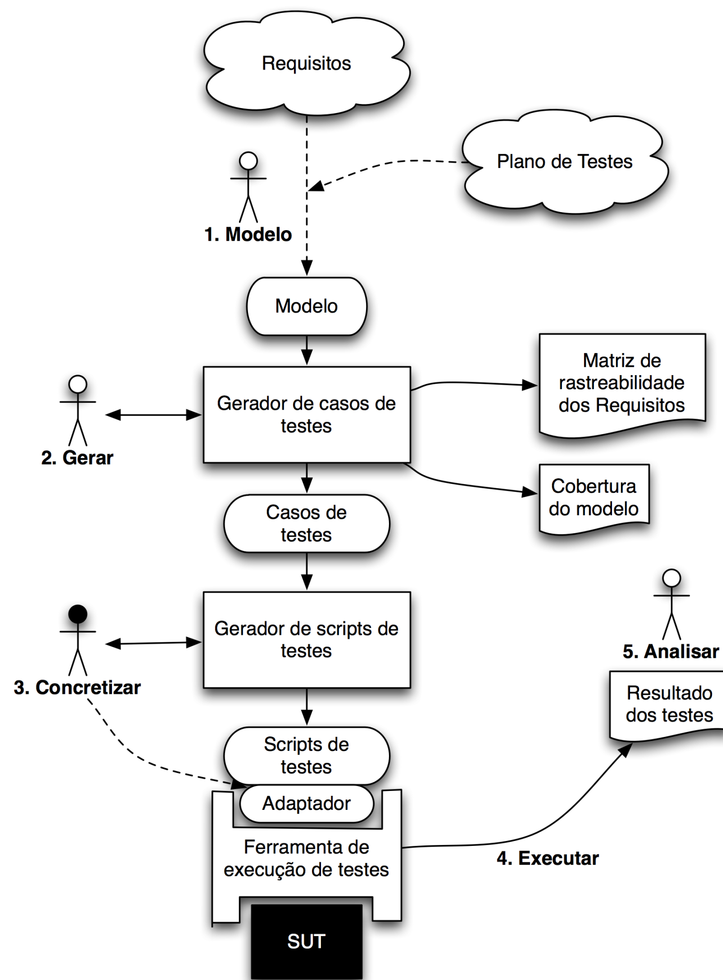

Figura 2.9: Diagrama do método: Testes baseados em modelos.

as fases de **geração de casos** de testes abstratos e **concretização** dos mesmos acabam sendo feitos de forma implícita à fase de **execução** já que não há registro de casos de teste;

offline onde os casos de testes serão previamente armazenados e em seguida concretizados para, enfim, serem executados;

5. **Análise dos resultados**. Deve-se analisar os resultados dos testes (os relatórios), para, então, realizar as ações corretivas necessárias. A partir do momento que surgi um problema, deve-se saber qual sua fonte. Duas novas fontes surgem a partir do uso de TBM: erros no código do adaptador ou no modelo (podem ser erro nos requisitos). Conforme dito em Utting & Legeard (2006), grosseiramente metades dos erros encontrados serão no SUT e a outra metade será encontrada no modelo ou nos requisitos. Achar erros nos requisitos do sistema até mesmo antes da implementação é uma das vantagens do uso de TBM, será detalhado na Seção-2.3.1.

2.3.1 Vantagens

Esta seção está destinada a apontar os pontos positivos do uso da metodologia de testes baseados em modelos. Estes pontos depende diretamente da qualidade dos modelos gerados, das ferramentas de geração e profissionais envolvidos com o projeto. A vantagens de se usar TBM são:

Detecção de erros no SUT: Bom para encontrar erros no sistema em desenvolvimento já que pode-se gerar uma grande quantidade de testes baseados na especificação/requisitos e escolher o critério de testes;

Redução de custo e tempo: Há uma redução considerável de tempo de execução/construção de testes com relação a outras técnicas. Bom para manutenção de testes quando o sistema evolui. Há a possibilidade de testar apenas os testes que foram afetados por uma determinada mudança, o resto dos testes mantêm-se intocados;

Aumento da qualidade dos testes: Com a possibilidade e gerar um número "arbitrário"de testes, escolher os critérios de testes e, ainda, analisar a cobertura dos requisitos através da matriz de rastreabilidade a qualidade dos testes tende a aumentar. Dependendo da equipe de testes a qualidade dos testes pode alcançar níveis excelentes;

Detecção de problemas nos requisitos: Por ser necessário a concepção de um modelo, erros nos requisitos informais ficam expostos, pois os projetistas acabam por se questionar sobre aspectos relevantes do projeto que podem não estar mal incompletos ou incorretos. O comportamento do sistema deve ficar claro a fim de que o modelo seja fiel ao real comportamento desejado. A detecção de erros nos requisitos do sistemas é o principal aspecto positivo do uso de TBM, pois quando erros são encontrados nessa fase, é muito mais barato corrigir o sistema;

Rastreabilidade: É habilidade de relacionar cada caso de teste com o modelo, com o critério de seleção e ainda com os requisitos informais do sistema Utting & Legeard (2006). A Figura-2.10 representa as possibilidades de rastreabilidade entre os casos de testes, o modelo e os requisitos. Pode também ser usada para aumentar a qualidade dos testes, por exemplo: é possível saber qual a porcentagem dos requisitos que foram testados, saber se um requisito mais crítico foi mais ou menos testado que requisitos menos críticos, definir uma quantidade mínima de testes por requisitos e assim por diante.

2.3.2 Desvantagens

Esta seção está destinada a apontar os pontos negativos do uso da metodologia de testes baseados em modelos. O principal deles é que não há nenhuma garantia que serão encon-

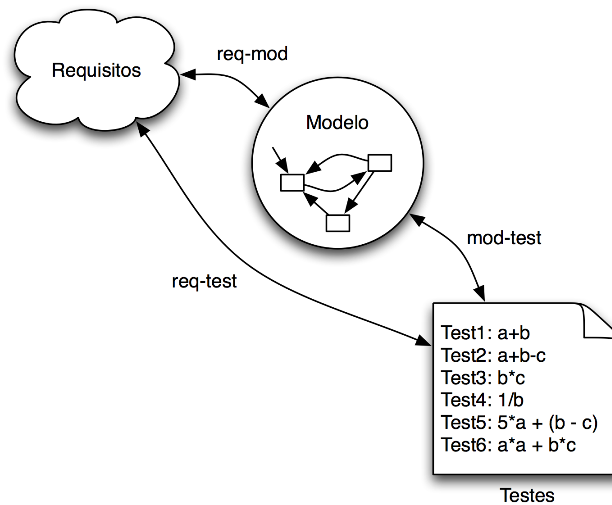

Figura 2.10: Diagrama de rastreabilidade.

tradas todas as diferenças entre o modelo e a implementação. Esse é comum a qualquer tipo de teste Utting & Legeard (2006). Outras desvantagens mais específicas serão listadas abaixo:

Habilidades extras: É imprescindível possuir conhecimentos além dos necessários à execução de testes manuais;

Apenas testes funcionais: Não testa outros aspectos de um sistema. Há trabalhos na linha de testes de desempenho utilizando TBM;

Maturidade de testes: Não deve ser aplicado em equipes cujo nível de maturidade de testes é baixa e é importante um conhecimento prévio de ferramentas de execução automática de testes. Isto normalmente é inerente a um bom nível de maturidade;

Curva de aprendizado: Os projetistas de teste terão de aprender a usar novas ferramentas e a construir modelos apropriados às necessidades do projeto de testes baseados em modelos.

Na Seção-2.4 serão descristas outras ferramentas com os mesmos intuitos do Gungnir com a finalidade de podermos comparar e argumentar sobre pontos de melhorias e novas características que devem ser adicionadas, para que o Gungnir torne-se uma ferramenta útil para uso no método testes baseados em modelos.

2.4 Trabalho relacionados

Esta seção é destinada a dissertar sobre trabalhos relacionados que influenciaram diretamente na concepção deste trabalho. Como base temos os trabalhos gerados em Hessel et al.

(2008) - o UPPAAL TRON - e de Vasconcelos Oliveira (2009) - gerador/tradutor de modelos de ISA 5.2 e Ladder para autômatos temporizados.

O UPPAAL TRON foi escolhido como inspiração baseado em resultados extraídos das pesquisas já realizadas em de Vasconcelos Oliveira (2009). Chegou-se à conclusão que a ferramenta que mais se encaixa às necessidades do projeto SIS é o TRON, por conseguinte algumas de suas características podem ser melhoradas no âmbito de aplicação de testes de conformidade para autômatos temporizados direcionados a aplicações de automação e controle utilizando ISA 5.2 e, a linguagem de programação para CLPs, Ladder. Na Seção-2.4.1 serão descritas as características que necessitam ser melhoradas.

Serão utilizados, também, resultados do trabalho de Vasconcelos Oliveira (2009). O gerador/tradutor de modelos descritos em termos de autômatos temporizados a partir de modelos descritos em ISA 5.2 e programas descritos em linguagem de programação Ladder. Esta ferramenta é de vital importância para a aplicação da metodologia de testes baseados em modelos.

A seguir será dado um maior detalhamento dos trabalhos citados, para que seja possível entender de onde surgiram as características básicas do Gungnir.

2.4.1 UPPAAL TRON

O UPPAAL TRON é uma ferramenta para verificação sistemas de tempo real desenvolvida pela parceria entras as universidades de Uppsala e Aalborg. Sua implementação teve início como parte da tese de mestrado e continuado como parte do projeto da tese de doutorado de Marius Mikucionis supervisionado por Kim G. Larsen e Brian Nielsen Larsen et al. (2009). Possui uma série de estudos de caso aplicados que abrangem de protocolos de comunicação a aplicações multimídia Behrmann, A & Larsen (2004).

Principais características da ferramenta baseado em Larsen et al. (2009):

- **Aplicação de testes de conformidade**: a ferramenta checa se em execuções temporizadas o sistema em teste (SUT) está especificado no modelo do sistema e se algum comportamento indesejado é observado.
- Ênfase nos **testes de propriedades funcionais e temporais**. O tempo é considerado contínuo, eventos podem acontecer a qualquer momento no tempo, porém os *deadlines* são definidos como inteiros. Geração de dados de teste também é possível, por conseguinte tipos de dados e seleção de valores são limitados pela linguagem de modelagem.
- A especificação é uma **rede de autômatos temporizados** UPPAAL Behrmann, David & Larsen (2004) particionada em modelo do sistema e modelo do comportamento do ambiente. Os modelos podem ser **não-determinísticos**, permitindo um bom grau de

liberdade na implementação de sistemas, modelando possíveis/toleráveis desvios de tempo, *soft time deadlines.*

- A primitivas de testes são geradas diretamente do modelo, executadas e as respostas do sistema são checadas ao mesmo tempo, ***online (on-the-fly)***, enquanto conectada com o SUT; isso evita *suites* de testes intermediárias gigantescas.

- Durante os testes a ferramenta segue o modelo do ambiente o qual pode ter várias propostas:

  1. um modelo completo do ambiente o qual será suficiente para executar todos os testes de conformidade;
  2. um ambiente específico que diminui o esforço para realização de testes para um nível realista de conformidade;
  3. um ambiente como um guia de casos de uso para uma funcionalidade de particular interesse;
  4. um modelo do ambiente como execuções de testes pré-gravados usadas para re-executar testes durante a depuração ou testes de regressão.

- Herdou a *engine* de checagem de modelos do projeto UPPAAL a qual permite **explorações rápidas e eficientes de modelos de autômato temporizados**;

- Se o modelo do ambiente for não-determinístico (o que é muito comum de ocorrer), então, **escolhas de valores de entrada ou tempos de espera são aleatórios;**

- Em geral, testes de conformidade para sistemas de tempo real são indecidíveis, contudo em termos numéricos, tem se mostrado suficiente em um espaço limitado de tempo.

O UPPAAL TRON também herdou do projeto UPPAAL Behrmann, David & Larsen (2004) a capacidade de construir graficamente os modelos de autômatos temporizados e simulá-los. A partir dos modelos construídos, são feitos os testes. As Figuras 2.11 e 2.12 mostram telas do programa em uso.

2.4.2 Método para automatização de testes caixa preta

Em de Vasconcelos Oliveira (2009), foi desenvolvido uma ferramenta e um método os quais contribuem para o aumento da confiança de programas para CLPs. O método configura a utilização de testes baseados em modelos que utiliza como ferramenta de automação dos testes o UPPAAL TRON. A linguagem de implementação suportada no referido trabalho, Ladder, está em conformidade com o padrão internacional IEC 61131-3 PLCopen (2004).

A seguir serão melhor detalhados a ferramenta e o método desenvolvido.

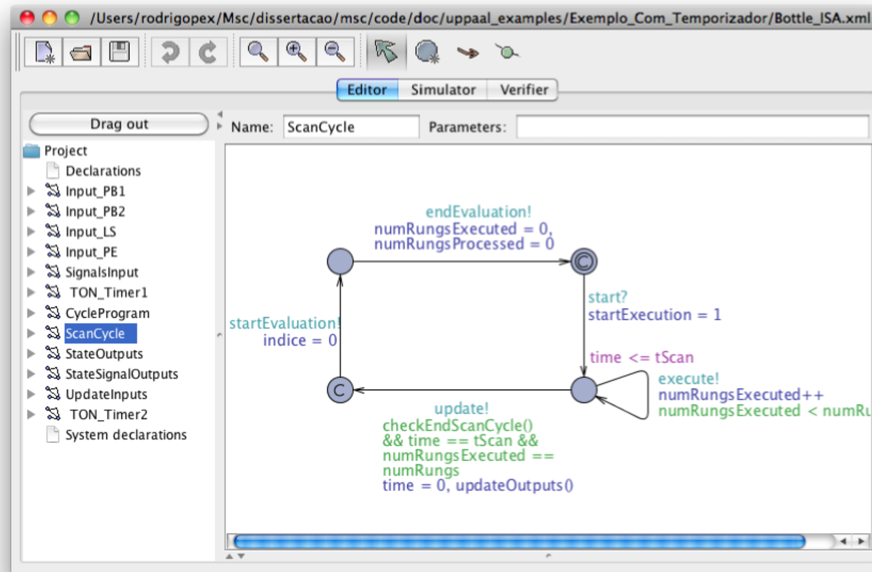

Figura 2.11: Tela do construtor de modelos do UPPAAL.

Ferramenta

É capaz de gerar modelos de autômatos temporizados utilizando o XML de uma representação da especificação em ISA 5.2 ou de um programa em Ladder. O modelo de autômatos temporizados gerado tentam imitar o comportamento de um CLP, de modo que as gerações dos degraus de Ladder serão traduzidos em um autômato onde cada estado representa um degrau. Ações como *Scan Cycle*, leitura e escrita de valores também são representados nos autômatos. Todos os elementos são, igualmente, traduzidos, de forma automática, em ATs o quais funcionam coordenados pelo AT que representa o *Scan Cycle*.

Método

Tem como resultado um veredicto a cerca da corretude da implementação. Trata-se por um sistema correto aquele, cuja implementação está em conformidade com a especificação. Toda a geração e execução dos casos de testes são feitas de forma automática pela ferramenta UPPAAL TRON bastando apenas que os modelos AT estejam descritos no XML padrão do UPPAAL. A fim de obter o veredicto, deve-se seguir alguns passos:

1. Construir a especificação em ISA 5.2 utilizando a ferramenta SIS;

2. Construir a implementação em Ladder utilizando a ferramenta SIS;

3. A partir do XML da especificação, gerar o modelo AT no XML padrão do UPPAAL utilizando a ferramenta descrita na Seção-2.4.2;

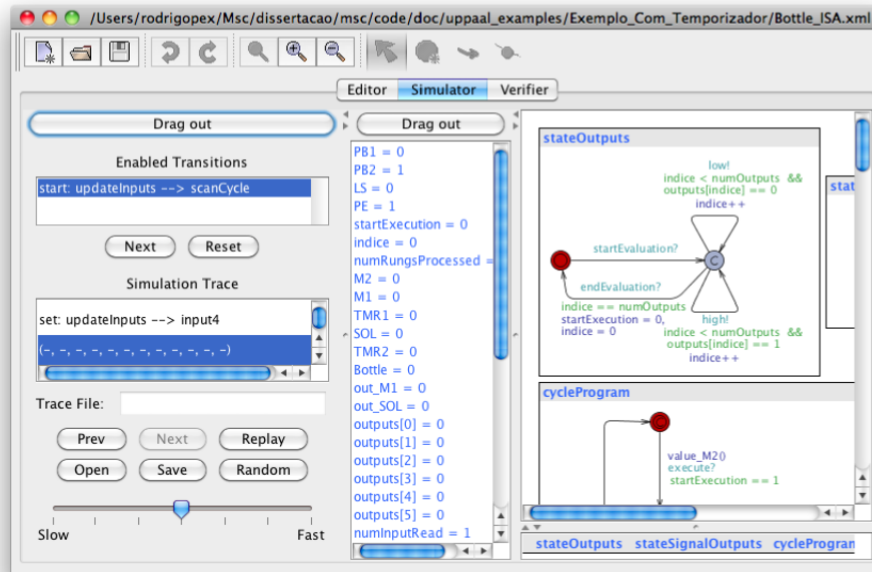

Figura 2.12: Tela do simulador de modelos do UPPAAL.

4. A partir do XML da implementação, gerar o modelo AT no XML padrão do UPPAAL utilizando a ferramenta descrita na Seção-2.4.2;

5. Inserir os dois modelos no UPPAAL TRON e através de um adaptador ligar os dois modelos;

6. Executar os testes *on-the-fly*, etapa responsável por gerar o *log* de veredicto;

7. Por fim, analisar o *log* e obter a principal informação: se os modelos passaram ou não nos testes.

A Figura-2.13 mostra uma visão geral do método descrito. Alguns problemas foram encontrados com o uso do UPPAAL TRON no trabalho de Vasconcelos Oliveira (2009), os quais serão listados abaixo:

Projeto fechado: O código fonte do projeto não está disponível para modificação, porém a ferramenta é livre para uso. Quando modificações são necessárias, deve-se entrar em contato com a equipe de desenvolvimento e sugerir tais modificações. Fica a critério da equipe aceitar ou não as sugestões;

Restrição de uso: Só pode ser usado em pesquisas sem fins lucrativos. Porém o projeto SIS é feito em convênio com a Petrobras para o uso da mesma, o que inviabiliza o trabalho de Vasconcelos Oliveira (2009) a partir do momento que é usada a ferramenta UPPAAL TRON;

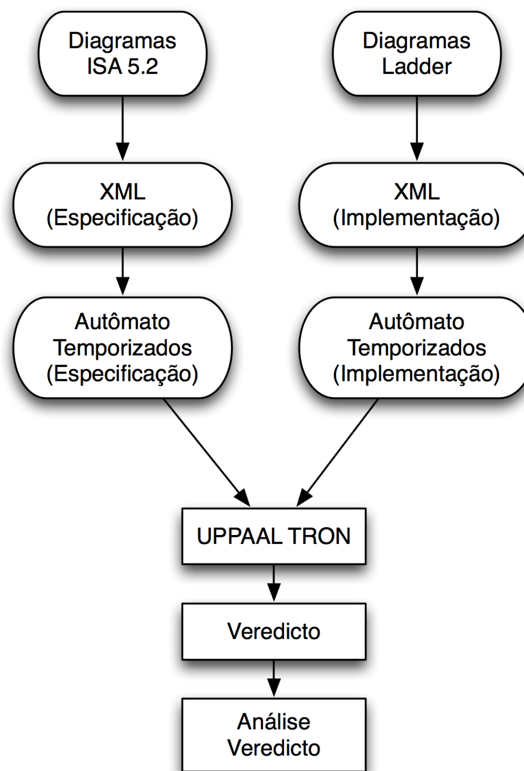

Figura 2.13: Visão geral do método desenvolvido no trabalho de Vasconcelos Oliveira (2009).

Força bruta: A geração dos casos de testes são feitas de forma aleatória, ou seja, utilizando a força bruta [Por a citação]. Não há nenhuma análise prévia dos possíveis valores da entradas para a geração dos casos de testes. Muitos casos de testes são impossíveis de acontecer o que causam desperdícios computacionais;

[Dependência entre relógios:] Atualmente no trabalho de Vasconcelos Oliveira (2009), os modelos gerados no padrão XML do UPPAAL podem possuir apenas um relógio o que torna menos rica a descrição dos autômatos, pois todos os elementos temporizados dependem de apenas um único relógio. Isso é decorrente do problema de dependência entre os relógios. Essa limitação acarreta o problema de divisão, porquanto nem todos os temporizadores são múltiplos e o tempo de todo o modelo é baseado no tempo do *Scan Cycle* pode chegar a números não inteiros de ciclo;

Particionamento rígido: Os modelos utilizados no processo de teste devem estar particionados adequadamente. Variáveis e eventos de saída têm que ser diferenciados dos de entrada. Uma variável de entrada não pode ser modificada em eventos de saída e vice-versa. Assim, surgem redundâncias nos modelos, pois valores devem ser manipulados em variáveis temporárias auxiliares;

Resultados confusos: Os dados gerados pela ferramenta UPPAAL TRON são complicados

de serem analisados, pois são grandes e inlegíveis a "leigos". Não são facilmente ou objetivamente caracterizados a falha ou sucesso dos testes. Há também os testes inconclusivos os quais tornam a análise ainda mais penosa.

O corrente trabalho foi desenvolvido, para resolver/amenizar os problemas citados acima. Desta forma, torna possível o uso do trabalho de Vasconcelos Oliveira (2009) no projeto SIS. Concebe também uma ferramenta objetiva e especializada em testes baseado em modelos para programas para CLPs. O Capítulo-3 explica com detalhes a ferramenta Gungnir e também como ela trata os problemas citados.

Capítulo 3

Ferramenta Gungnir

O Gungnir é uma ferramenta de automação de testes caixa-preta *offline* a qual utiliza autômatos temporizados como formalismo para concepção de modelos. Está inserida no contexto do projeto SIS como substituta da supra citada UPPAAL TRON pelos motivos citados na Seção-2.4.2. Em resumo, os problemas são: o projeto possui o código fechado, há restrições de uso, executa todas as possíveis combinações das entradas (força bruta), tem restrições com relação ao uso de relógios, os relatórios de testes são confusos e os modelos são complexos.[Confirmar a seção anterior.]

Para resolver/amenizar os problemas da ferramenta UPPAAL TRON, foi implementada uma séria de características no Gungnir que vão de encontro com os pontos negativos tentando criar novos caminhos para uma solução mais específica e adequada ao problema. A seguir, temos uma lista com as principais características desta ferramenta:

- Geração de autômatos temporizados que representam os modelos da especificação (ISA 5.2) e da implementação (Ladder);
- Simulação de autômatos temporizados (modelos) com um número arbitrário de estados (com ou sem consumo de tempo) e relógios;
- Número arbitrário de transições entre os estados;
- Permite modelagem de autômatos temporizados em Python;
  - Atribuições limitadas pela sintaxe da linguagem Python;
- Geração automática de casos de testes utilizando a força bruta;
- Execução do testes em modo *offline*, os casos de testes (dados de entrada e gabarito de saída) são gerados a priori, para então, serem executados no modelo SUT;
- Interface *socket* que permite a conexão de sistemas ao testador;

- Veredicto claro e objetivo. Entendível e analisável por indivíduos não especialistas. O relatório dos testes será mais detalhado, demonstrará todas as modificações sofridas pelas placas de memória (entrada e saída) do CLP durante a execução dos testes num arquivo **.vcd** (*Value Change Dump*). No fim, tem-se o resultado da execução de forma objetiva: Passou ou Falhou;

- Código acessível, customizado especificamente para o projeto;

O método formalizado no trabalho de Vasconcelos Oliveira (2009) pode ser visto na Figura-3.2 auxiliada pela Figura-3.1. Algumas mudanças podem ser observadas com relação ao diagrama mostrado na Figura-2.9, mudanças essas causadas por capacidades do Gungnir as quais estão listadas a seguir:

1. O papel do projetista de testes não é mais necessário, todos os testes serão automatizados;

2. O programador entra no início do processo, pois não será mais necessário criar os modelos. Basta desenvolver os códigos da especificação (ISA 5.2) e da implementação (Ladder);

3. Não há mais planos de testes, porquanto todo o processo será automatizado, sem interferência humana;

4. Não há necessidade da geração de casos de testes em dois níveis (bastante utilizado em TBM), as vantagens da geração em dois níveis não se aplicam aqui;

5. Os passos 2 e 3 foram embutidos no Gungnir. Não há mais o conceito de criação de modelos por humanos. O programador não precisará mais modificar o adaptador, não há necessidade de adaptação, porque, agora, os testes poderão ser aplicados ao próprio modelo do SUT;

6. O conceito de SUT poderá ser aplicado ao modelo gerado a partir do programa Ladder ou realmente ser usado via *socket*;

7. A ferramenta de geração de modelo é uma extensão de parte do trabalho de Vasconcelos Oliveira (2009) que foi adaptada ao Gungnir e acrescentada ao mesmo, eliminando, desta forma, o esforço na criação (normalmente manual) do modelo[1]. Os modelos gerados são menos complexos, por conseguinte mantém todas as características necessárias para a geração e execução fiel do comportamento de CLPs;

[1]A criação manual do modelo é um dos pontos fortes de TBM Utting & Legeard (2006), pois os requisitos são de fato analisados, entendidos, revisados e questionados; desta forma, encontrando erros mais prematuramente baixando o custo do sistema com correção de falhas. Eliminando essa etapa por inteiro estaríamos enfraquecendo o método, por conseguinte a especificação em ISA 5.2 e a implementação Ladder assumem o papel dos modelos na verificação de erros nos requisitos.

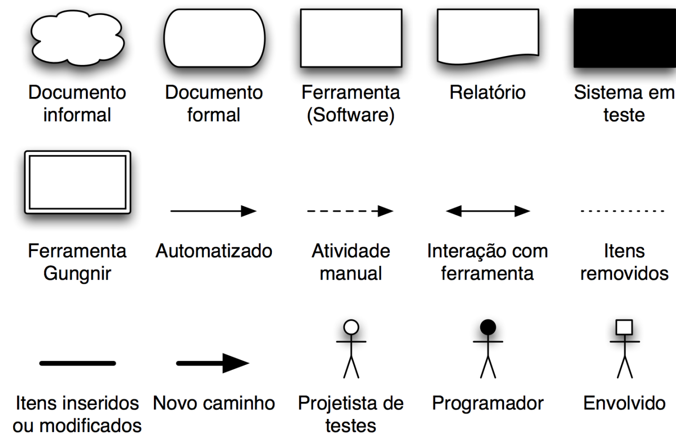

Figura 3.1: Notação do diagrama do método TBM adaptado.

Agora serão descritos os formalismos com relação a autômatos temporizados entendidos pela ferramenta Gungnir.

3.0.3 Formalismo

O Gungnir reconhece e simula redes autômatos temporizados Alur & Dill (1994) não-concorrente que podem contar com número arbitrário de estados os quais podem consumir ou não tempo. As transições são feitas de forma aleatória, onde a transição que tiver guarda válida entra na lista de possíveis sorteadas. Qualquer variável global pode entrar na verificação das guardas inclusive os relógios que também são representados dentro das variáveis do sistema. Na Figura-3.3 estão representadas graficamente as entidades existentes nos autômatos temporizados do Gungnir.

Os relógios são atualizados durante a simulação, o tempo de simulação é atualizado quando uma transição consumidora de tempo é escolhida. Caso não haja nenhuma transição candidata a executar em determinado momento, diz-se que aconteceu um *"Deadlock"* ou simplesmente uma rejeição. Ao fim de cada transição, pode-se atribuir valores às variáveis globais onde estão inclusos os relógios e variáveis de saída.

O conceito de palavra temporizada também faz parte do formalismo compreendido pelo Gungnir. Uma palavra temporizada é uma mudança em alguma das variáveis dos autômatos em determinados pontos no tempo. Na Expressão-3.1 temos um exemplo de palavra temporizada.

$$TimedWWord((10,"v":1)->(12,"v":0)->(15,"v":1)->(20,"v":0)) \tag{3.1}$$

A variável v de um determinado autômato tem seus valores alterados para um nos tempos 10 e 15 e para zero 0 nos tempos 12 e 20 durante a simulação de um autômato. A seguir, serão descritas algumas características do modelos, as fontes de geração dos mesmos e a

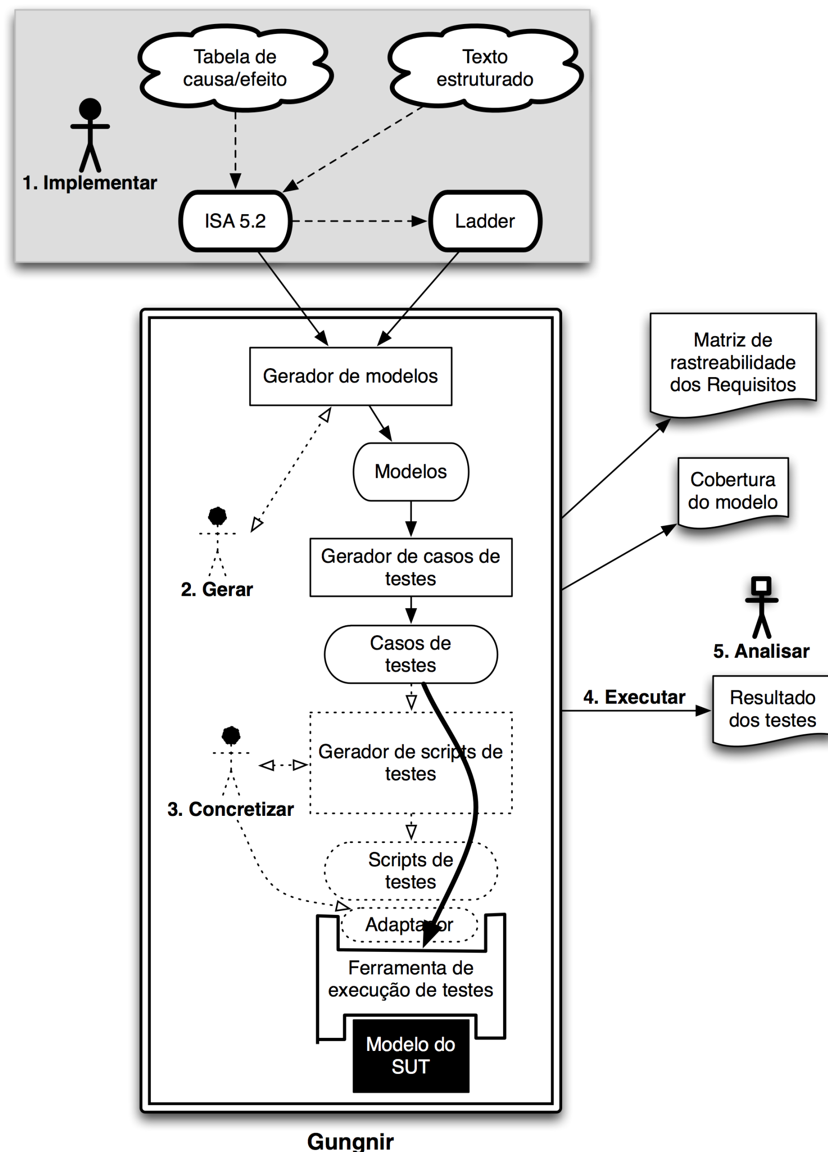

Figura 3.2: Diagrama do método: Testes baseados em modelos adaptado à aplicação de testes de conformidade para sistema de automação e controle.

representação que o Gungnir usar para interpretá-los.

3.0.4 Testes baseados em modelo

O Gungnir foi construído de acordo com os conceitos da metodologia de testes baseados em modelos citada na Seção-2.3. Os modelos, aqui utilizados, são redes de autômatos temporizados conforme será citado na Seção-3.0.4. Uma característica diferente da ferramenta com relação a TBM é fato dos testes serem gerados e aplicados a modelos e não no sistema propriamente dito. A Figura-3.4 representa graficamente o processo interno geração e criação de testes utilizando o Gungnir. A seguir serão detalhados cada parte integrante do Gungnir marcadas na Figura-3.4.

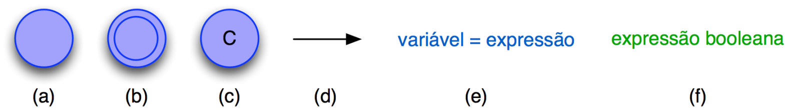

Figura 3.3: (a) estado, (b) estado inicial, (c) estado *committed*, não consome tempo, (d) transição, (e) atribuição e (f) guarda.

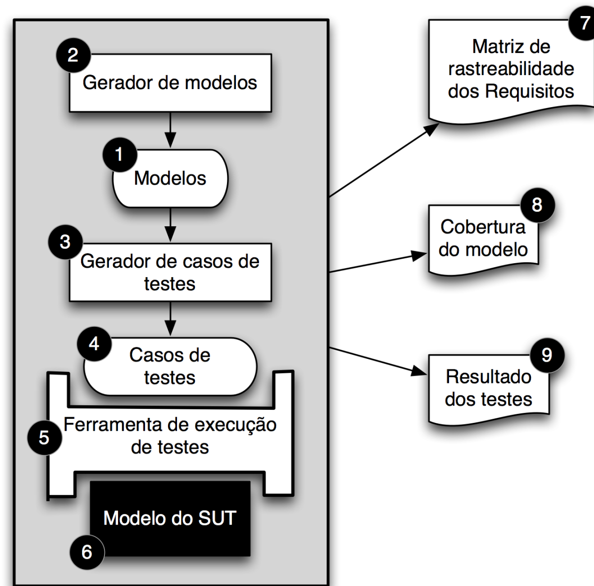

Figura 3.4: Visão interna do Gungnir.

Modelos

Os modelos da especificação ISA 5.2 e do programa Ladder manipulados pelo Gungnir são descritos utilizando autômatos temporizados como formalismo. Conforme descrito na Seção-2.1.1, os CLPs possuem características especiais de execução as quais devem ser mantidas, com a finalidade de gerar modelos o mais próximos possível do comportamento real de um controlador lógico programável. Para ilustrar a ideia da concepção dos modelos iremos utilizar o exemplo da Figura-3.5

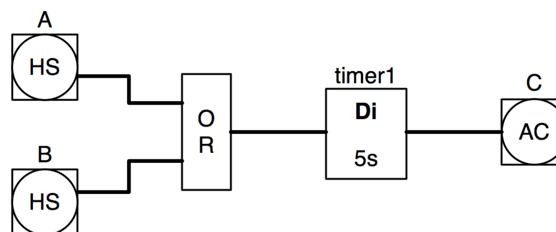

Figura 3.5: *ReadInputs* - exemplo de especificação ISA 5.2.

As fases de trabalho de um CLP são expressas em termos de autômatos, ou seja, para

cada etapa do processamento um ou mais autômatos são gerados. Da primeira fase a qual é a de leitura das placas de entrada e armazenamento dos valores nas devidas variáveis globais extrai-se o autômato (com apenas um estado *committed*) *ReadInputs*. Para cada variável de entrada do CLP duas transições são criadas no *ReadInputs*. A Figura-3.6 mostra o resultado da geração do *ReadInputs* para o exemplo da Figura-3.5.

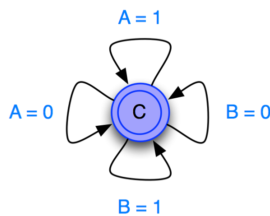

Figura 3.6: *ReadInputs* - exemplo do autômato responsável pela leitura das variáveis de entrada do CLP.

A segunda fase é a de processamento. O único elemento de processamento que pode ser encontrado no conjunto restrito de elementos do ISA 5.2 e Ladder (citado na Seção-2.1.2 e Seção-2.1.2 respectivamente) são os temporizadores. Estes são de dois tipos: Di e Dt no ISA 5.2 e TON e TOFF para o Ladder. O autômato responsável por reger o comportamento do temporizador Di ou TON (equivalentes) da Figura-3.5 está ilustrado na Figura-3.7. Já os Dt e TOFF (equivalentes) estão representados genericamente na Figura-3.8.

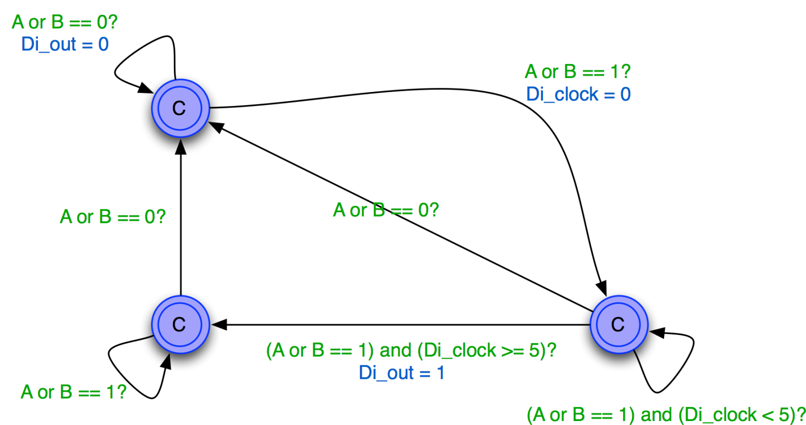

Figura 3.7: Temporizador Di (TON) - autômato responsável pela gerência dos valores de saída do temporizador timer1 da Figura-3.5.

A terceira e última fase de trabalho do CLP é a escrita dos valores das variáveis globais de saída nas placas de saída do controlador. Segue o mesmo raciocínio do *ReadInputs*, é necessário apenas um estado *committed* no autômato para gerar as saídas esperadas. A Figura-3.9 mostra o modelo do autômato de um único estado que atualiza os valores das placas de saída para o exemplo da Figura-3.5.

Observe que todos os estados utilizados até agora são *committed* (não consomem tempo). Para completar o comportamento real de trabalho de um CLP basta atualizar no

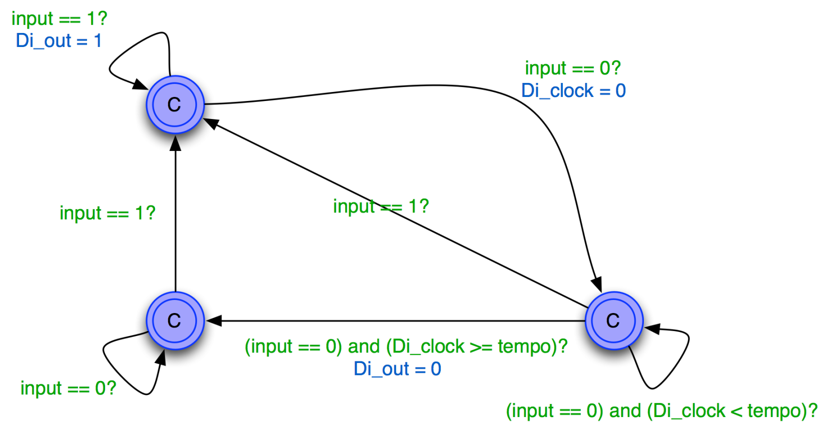

Figura 3.8: Temporizador Dt (TOFF) - exemplo do autômato responsável pela gerência dos valores de saída do temporizador Dt genérico.

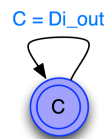

Figura 3.9: *WriteOutputs* - exemplo do autômato responsável pela escrita das variáveis de saída do CLP.

tempo do sistema o tempo de *ScanCycle*, portanto deve-se criar um último autômato chamado *TimeUpdater*, cuja função é apenas atualizar o tempo do sistema como todo. Assim, fecha-se os conceitos sobre os modelos utilizados no Gungnir. A seguir, serão mostrados as ferramentas, estratégias e os algoritmos utilizados para gerar os modelos autômatos temporizados discutido até o momento.

Geração de Modelos de autômatos temporizados

Os modelos são obtidos a partir de arquivos XML(*eXtensible Markup Language*) o quais descrevem especificações ISA 5.2 e Ladder. As *tags* e a hierarquia de *tags* dos arquivos XML são praticamente iguais para os dois casos utilizados. A lista das principais *tags* utilizadas em comum são:

input: Variáveis de entrada;

output: Variáveis de saída;

feedback: Variáveis de realimentação (do inglês, *feedback*);

timer_Di, timer_Dt, timer_TON ou timer_TOFF: Temporizadores;

rung: Ligações de variáveis de entrada e *feedback* e temporizadores às variáveis de saída;

and, or, not: Operações binárias permitidas;

ref: Referência a variáveis globais.

Exemplos de arquivos podem ser vistos no Anexo-A.2. Com a finalidade de gerar os modelos, foram desenvolvidos algoritmos que traduzem arquivos XML de especificação e implementação para os modelos Python que descrevem os autômatos temporizados. O passo a passo executado para se concretizar a tradução é:

1. Cria-se, a partir de um modelo, o AT *ReadInputs*;

2. Para cada *tag input* adiciona-se duas transições no *ReadInputs*: uma com nome da variável igual e 1 e outra igual a zero e ambas com guardas sobre o valor do id. Por exemplo, caso a variável seja a terceira a ser lida, deve conter a guarda $id == 2$. Cadastra a variável nas variáveis globais;

3. Para cada variável *feedback* encontrada deve-se criar uma transição em *ReadInputs* com a guarda sobre o id e uma atribuição da variável de saída referente à realimentação. Por exemplo: suponha que a variável de *feedback* encontrada seja $test$, então a transição deve conter a seguinte atribuição: $feedback_test = output_test$;

4. Para cada relógio encontrado cria-se, a partir de um modelo, o AT do temporizador. A esse modelo deve-se aplicar o tempo de ativação, a expressão (obtida recursivamente sobre a entrada do temporizador) de entrada do temporizador. Deve-se cadastrar a variável de saída e o relógio interno nas variáveis globais. Em cada transição adicionar a guarda com o id do temporizador;

5. Cria-se, a partir de um modelo, o AT *WriteOutputs*;

6. Para cada variável de saída deve-se adicionar uma transição com a guarda do id e uma atribuição referente à expressão de entrada (obtida recursivamente);

7. Por fim, cria-se, a partir de um modelo, o AT *TimeUpdater*;

A Figura-3.10 ilustra a representação gráfica dos autômatos gerados para o exemplo da Figura-3.5. Observe que há uma sequência de ids isso é quem permite a execução cíclica do CLP. Todos os estados não consomem tempo exceto o *TimeUpdater*. E nele que o tempo de *Scan* é atualizado. Como as variáveis só serão vistas no mundo exterior no final do ciclo, pode-se considerar que a leitura, o processamento e a escrita na saída não gastam tempo e no fim o *updater* espera o tempo necessário de um ciclo.

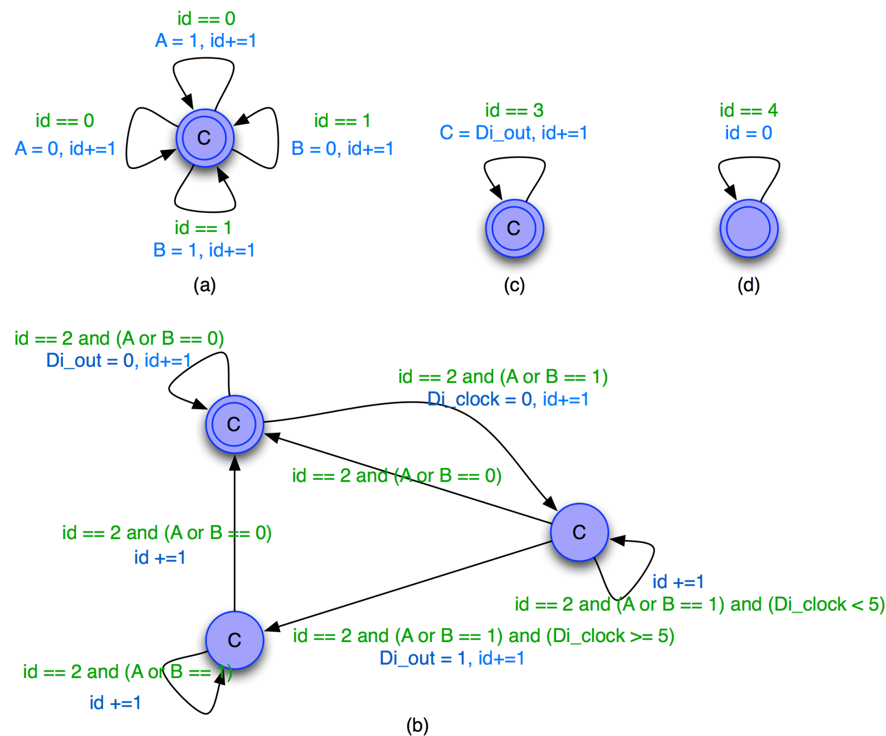

Figura 3.10: (a) *ReadInputs*, (b) *timer_DI*, (c) *WriteOutputs* e (d) *TimerUpdater*.

Geração dos casos de testes

Os casos de testes devem ser entendidos como um vetor de dados de entrada combinado com um vetor de dados de saída e o tempo de ocorrência desses. Para cada *ScanCycle* temos um caso de teste. Se para um determinado vetor de dados de entrada, o modelo da implementação externar um vetor com valores diferentes dos externados pelo modelo da implementação, implica na ocorrência de uma falha.

Com a finalidade de explorar todo o modelo da implementação, o Gungnir gera aleatoriamente os valores de entrada, fato também conhecido como "geração na força bruta". Fato que, em alguns momentos, gera desperdícios computacionais pelo fato de testar combinações de entrada que nunca ocorrerão na realidade. Por exemplo, sensores de Temperatura altíssima e baixíssima nunca terão seus valores 1, se supusermos que os sensores estão em perfeito estado. Para resolver/amenizar esse problema o usuário poderia indicar a relação entre as variáveis, porém uma das intensões desse trabalho é fazer testes 100% automáticos usando TBM fato ainda pouco explorado segundo CITAR O ARTIGO DE TESTES DA CAROL.. O desperdício nesse caso é para evitar a necessidade de um conhecimento prévio do sistema na execução dos testes.

Os erros são capturados por comparação de valores de saída "desejados"gerados pelo modelo da especificação. Erros em degraus puramente lógicos (não possuem elementos temporizados) são capturados de forma simples, porém custosa, basta testar todas as possibilidades dos valores da entrada. Para amenizar o impacto desta geração, foi utilizada a

heurística do "fator determinante"(Figura-3.11) que reduz o conjunto de entrada. Ao invés de injetar todas as possibilidades nos vetores de entrada, a injeção é aleatória e a cada classe de valores encontrado para os *rungs* baseado na heurística é marcado no conjunto de classes que devem executar para cumprir o critério de cobertura.

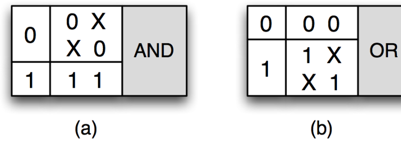

Figura 3.11: (a) Heurística para expressões do tipo "e", (b) Heurística para expressões do tipo "ou"

A Figura-3.12 mostra um exemplo da aplicação recursiva da heurística. Com uso dessa heurística pode-se dizer que foi testado pelo menos um elemento de cada classe de variação nas saídas do CLP. Assim, obtêm-se uma cobertura de dados adequada, pois são testadas todas as variações que mudariam o resultado da expressão.

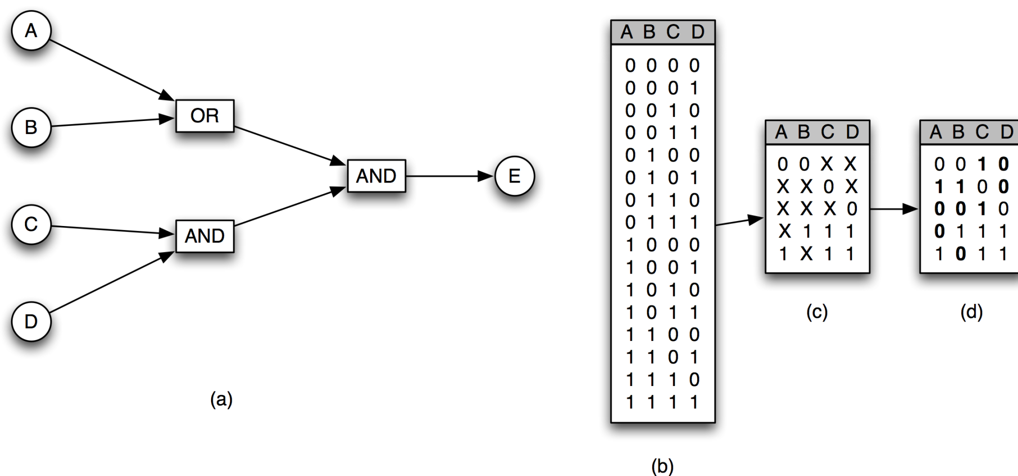

Figura 3.12: (a) expressão, (b) possíveis valores de A, B, C e D, (c) heurística do fator determinante para o caso, (d) resultado gerado a partir da heurística.

Para elementos temporizados, os testes são guiados pela ativação dos temporizadores. Para obter uma cobertura adequada todos os temporizadores tem que ser acionados ao menos uma vez num momento em que sejam determinantes. Para acionar um temporizador deve-se fazer o seguinte cálculo: divide-se o tempo de ativação do temporizador pelo tempo de *Scan* arredondando sempre para cima. O número encontrado é a quantidade de vezes que deve-se aplicar a classe de valores que ative-o nas entradas. As outras entradas que não estão envolvidas na ativação continuarão sendo geradas aleatoriamente.

As gerações aleatórias ocorrerão até que o critério de cobertura seja alcançado. Em alguns momentos, a geração pode levar um certo tempo. O usuário tem a opção de dizer

quantos *ScanCycles* ele deseja executar. Inicialmente, procura-se alcançar a cobertura dos *rungs* temporizadores, para, então, concluir os *rungs* sem temporizadores. Observe que ao passo que são cobertos os temporizadores, os outros *rungs* já estarão sendo cobertos em paralelo, já que os valores continuam sendo injetados aleatoriamente.

Execução dos casos de testes

A execução dos casos de testes é bem direta. Os valores dos vetores de entrada e saída definidos na fase de geração de casos de testes utilizando o modelo da são injetados no modelo da implementação. Caso alguma saída não esteja de acordo com o esperado, a mensagem de erro "FALHA: Implementação incorreta."é mostrada para o usuário e o *rung* o qual contém o erro é indicado para possível correção. Caso os testes sejam executados com sucesso, a mensagem "Implementação correta."é exibida na tela.

Como auxílio opcional, um arquivo **.vcd** será criado com as entradas e saídas executadas no modelo da implementação para possível análise visual. A Figura-3.13 mostra um exemplo de visualização de um arquivo **.vcd** utilizando a ferramenta Scanion (2010*b*).

Figura 3.13: Screenshot do aplicativo Scanion com um exemplo de arquivo **.vcd**.

Aspectos relacionados a detalhes das implementações da ferramenta Gungnir estão disponíveis no Apêndice-A. No próximo capítulo, para completar o entendimento do uso da Gungnir na geração de casos de testes utilizando modelos, serão esmiuçados dois estudos de casos: Controle de engarrafadora e Controle de semáforos.

Capítulo 4

Estudo de caso

Esta capítulo destina-se a concretizar e demonstrar a eficácia do uso do Gungnir como ferramenta para geração e execução automática de testes. Dois serão os estudos de caso, os mesmos do trabalho de Vasconcelos Oliveira (2009): o controle de engarrafadora e o controle de semáforos.

4.1 Controle de engarrafadora

Conforme descrito em de Vasconcelos Oliveira (2009), na Figura 4.1 um sistema que tem a finalidade de encher garrafas ilustrado. Seu funcionamento ocorre da seguinte forma: uma vez que o botão de inicializar (PB1) é pressionado, o motor de auto-realimentação (M2) é ligado. Este motor permanecerá ligado até que o botão de parar (PB2) seja acionado. O motor M1 será ativado assim que o sistema for iniciado (M2 estiver ligado) e irá parar quando o sensor (LS) detectar uma garrafa na posição correta. Quando a garrafa estiver na posição correta e 0.5 segundos tiverem se passado, o solenoide (SOL) irá abrir a válvula para liberar o refrigerante e o enchimento ocorrer até que o fotosensor (PE) detecte um nível adequado de líquido no interior da garrafa. Após ser enchida, a garrafa permanecerá nesta posição durante 0.7 segundos. Em seguida, o motor M1 é inicializado. Este irá permanecer ligado até que o sensor detecte outra garrafa.

As Figuras **??** e **??** ilustram o diagrama ISA 5.2 e o programa Ladder para a Engarrafadora respectivamente.

4.1.1 Utilizando o Gungnir

Inicialmente, deve-se executar o Gungnir na linha de comando utilizando o seguinte comando:

```
#> gungnir [file_isa52] [file_ladder]
```

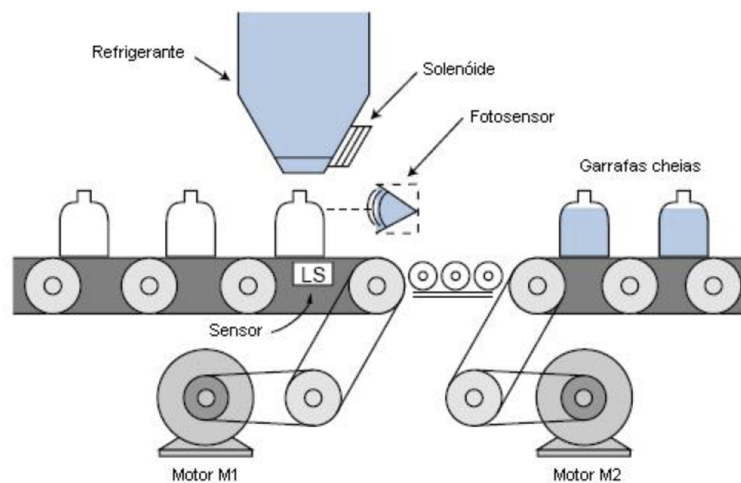

Figura 4.1: Engarrafadora - Fonte: Bryan & Bryan (1997*b*), página 485

Como saída, se a implementação estiver correta, aparecerá as seguinte mensagens:

```
#> gungnir engarrafadora_isa.xml engarrafadora_ladder.xml
==[[ Gungnir 0.4 ]]============================================================

ISA 5.2 file: /Users/plutao/msc/estudo\_de\_caso/engarrafadora_isa.xml
Ladder file: /Users/plutao/msc/estudo\_de\_caso/engarrafadora_ladder.xml

--| Generating models |----------------------------------------------------

* ISA 5.2 .............................................................[ok]
--> Rungs (1 Free, 5 Non free)
--> Variables (4 inputs, 4 feedbacks and 6 outputs)
--> Timers (2 Di)

* Ladder .............................................................[ok]
--> Rungs (1 Free, 5 Non free)
--> Variables (4 inputs, 4 feedbacks and 6 outputs)
--> Timers (2 Di)

--| Generating inputs |----------------------------------------------------

--> Timers [100%]
+ timer_Di01..........[ok]
+ timer_Di02..........[ok]

--> Outputs [100%]
+ M2 .................[ok]
```

```
+ M1 .................[ok]
+ TMR1 ...............[ok]
+ SOL ................[ok]
+ TMR2 ...............[ok]
+ Bottle .............[ok]

Coverage criteria: 100%

--| Test execution |-------------------------------------------------------
[========================================================================>]

--------------------------------------------------------------------------
Result: Implementation is correct!
==========================================================================
```

Para entender e ver o que acontece por trás do comando detalharemos o passo a passo do processo interno da ferramenta.

4.1.2 Geração dos modelos

Inicialmente, os modelos são gerados. Para a especificação e para implementação da engarrafadora os modelos gerados são praticamente os mesmos, para este caso, o diagrama Ladder foi feito sob medida com relação ao ISA 5.2. Basta analisar os código no Apêndice-A.2. Os modelos que representam a engarrafadora estão ilustrados nas Figuras 4.2, 4.3, 4.4, 4.5 e 4.6.

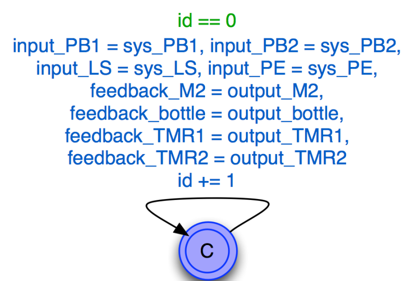

Figura 4.2: Autômato *ReadInputs*

4.1.3 Geração das entradas

A partir das transições do autômato *WriteOutputs*, podemos usar as expressões booleanas para obter o conjunto de valores das entradas necessários para atingir o critério de cobertura utilizando a heurística do fator determinante. Para cada saída devem ser calculadas as

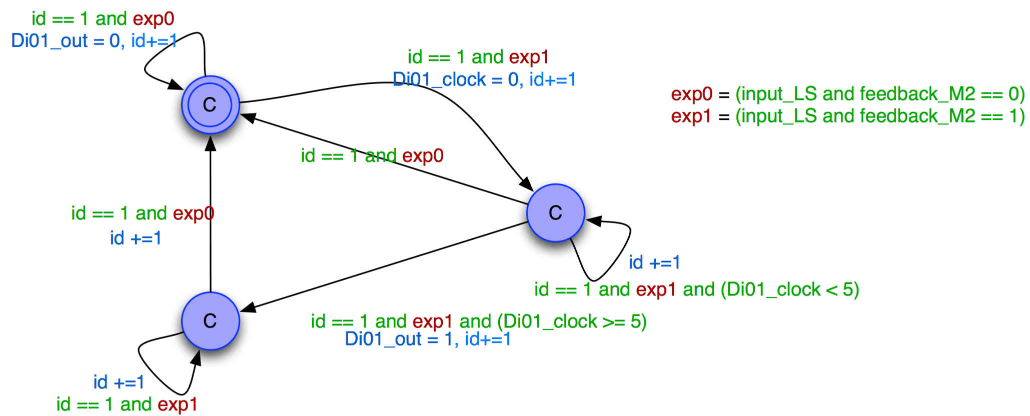

Figura 4.3: Autômato *Timer_Di01*

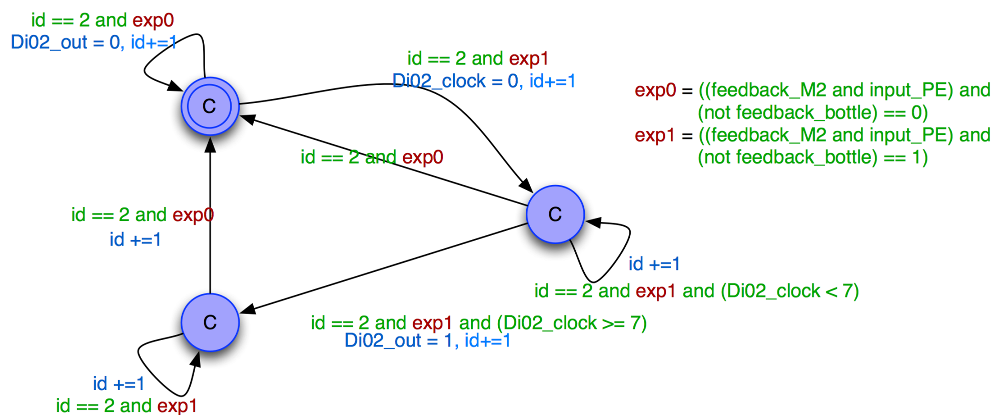

Figura 4.4: Autômato *Timer_Di02*

classes de valores a serem utilizados nas entradas. A Figura-4.7 mostra o resultado da heurística para todos as saídas.

Na primeira geração aleatória podemos observar que algumas classe já serão injetadas. Por exemplo, a seguir temos um exemplo de vetor de entrada aleatório onde as variáveis de *feedback* (marcadas com "*") estão todas em zero, pois ainda não foi executado nenhum ciclo. A Figura-4.8 mostra as classes já executadas com o vetor de entrada citado. Em alguns casos, na mesma saída, podem ser encontradas mais de uma classe para um vetor de entrada, por conseguinte apenas uma por vez pode ser marcada. Marcam-se primeiro as classes que dependem de *feedback*, pois, em vários casos, variam menos, principalmente, as temporizadas.

```
PB1 = 0, PB2 = 1, M2* = 0, LS = 0, Bottle* = 0, TMR1* = 0, PE = 1, TMR2* = 0
```

Exemplificando; nas classes de M2, podemos encontrar duas candidatas: 1 - (PB1 = 0, M2* = 0, PB2 = x) e 2 - (PB1 = x, M2* = x, PB2 = 1). A escolhida como executada, neste caso, foi a 1, pois o valor de M2* (*feedback*) tem prioridade na escolha. Facilmente, em outra geração

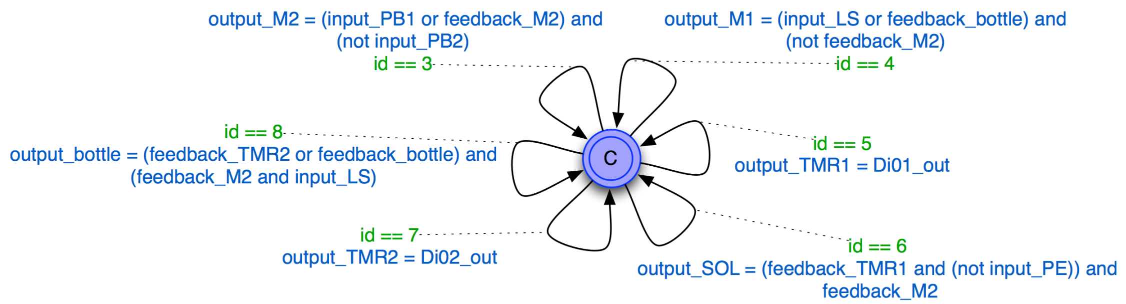

Figura 4.5: Autômato *WriteOutputs*

Figura 4.6: Autômato *TimeUpdater*

aleatória PB1 pode receber o valor 1, porém, como M2* é a saída de um outro *rung*, pode ser que não volte mais a ser zero.

Para cumprir o critério de cobertura com relação aos temporizadores deve-se acioná-los em algum momento durante a execução. Se eles forem os últimos elementos antes da saída, basta fazer com que os as classes sejam satisfeitas; caso contrário o temporizador apenas será contabilizado como coberto quando for fator determinante da saída. Para a engarrafadora temos apenas temporizadores ligados às saídas. Quando todos as saídas e temporizadores estiverem cobertos analisa-se o tempo atual e gera-se 100% aleatoriamente a mesma quantidade de testes.

4.1.4 Execução dos testes

O Gungnir executa automaticamente os testes gerados. O usuário pode, opcionalmente, solicitar que o Gungnir armazene os resultados em um arquivo **.vcd**. Para o caso da engarrafadora executaremos o programa Ladder correto o qual está representado na Figura-**??**. Para esse caso o resultado do teste foi: "Correct implementation.", para obter essa informação o Gungnir executou aproximadamente 1000 *ScanCycles* para cobrir o modelo. A Figura-**??** mostra o arquivo **.vcd** gerado. Os arquivos **.vcd** estão com *ScanCycles* de 500 ms.

Para validar a aplicação da ferramenta, vamos inserir erros e analisar os arquivo modificados. Três tipos de erro foram inserido no arquivo Ladder:

1. Substituição de um contato normalmente aberto por um fechado na entrada LS;

2. Troca de uma operação lógica *and* por uma *or*;

M2

	PB1	M2*	PB2
0	0	0	x
	x	x	1
1	1	x	0
	x	1	0

M1

	LS	Bottle*	M2*
0	1	0	x
	x	x	0
1	x	1	1
	0	x	1

SOL

	TMR1*	PE	M2*
0	0	x	x
	x	1	x
	x	x	0
1	1	0	1

Bottle

	TMR2*	Bottle*	M2*	LS
0	0	0	x	x
	x	x	0	x
	x	x	x	0
1	x	1	1	1
	1	x	1	1

TMR2

T	M2*	PE	Bottle*
0	0	x	x
	x	0	x
	x	x	1
1	1	1	0

TMR1

T	LS	M2*
0	0	x
	x	0
1	1	1

Figura 4.7: Tabelas de classes calculadas para as variáveis de saída através da heurística do fator determinante.

3. Aumento do tempo do temporizador timer_Di02 de 7 para 17 no *rung* de TMR2.

Erro 1

Números de ciclos de *Scan*: 5;

Resultado do teste: "ERROR: Incorrect implementation!";

Arquivo .vcd: Figura-4.10.

Erro 2

Números de ciclos de *Scan*: 11;

Resultado do teste: "ERROR: Incorrect implementation!";

Arquivo .vcd: Figura-4.11.

Erro 3

Números de ciclos de *Scan*: 82;

Resultado do teste: "ERROR: Incorrect implementation!";

Arquivo .vcd: Figura-4.12.

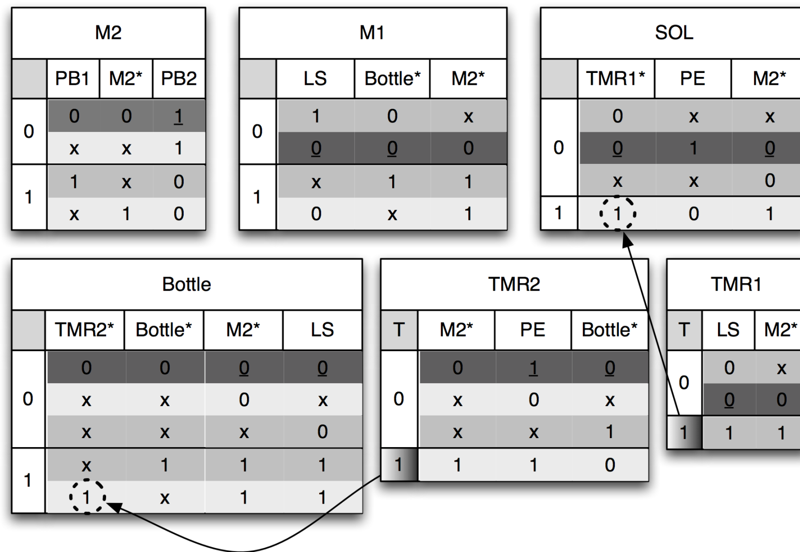

Figura 4.8: Classes selecionadas com a primeira geração de vetor de entrada aleatório.

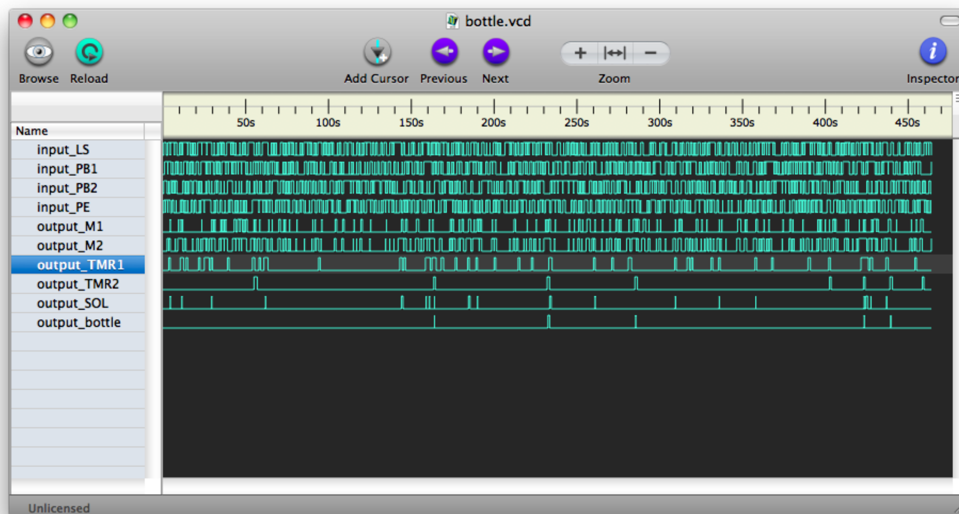

Figura 4.9: Comportamento da engarrafadora para as entradas geradas.

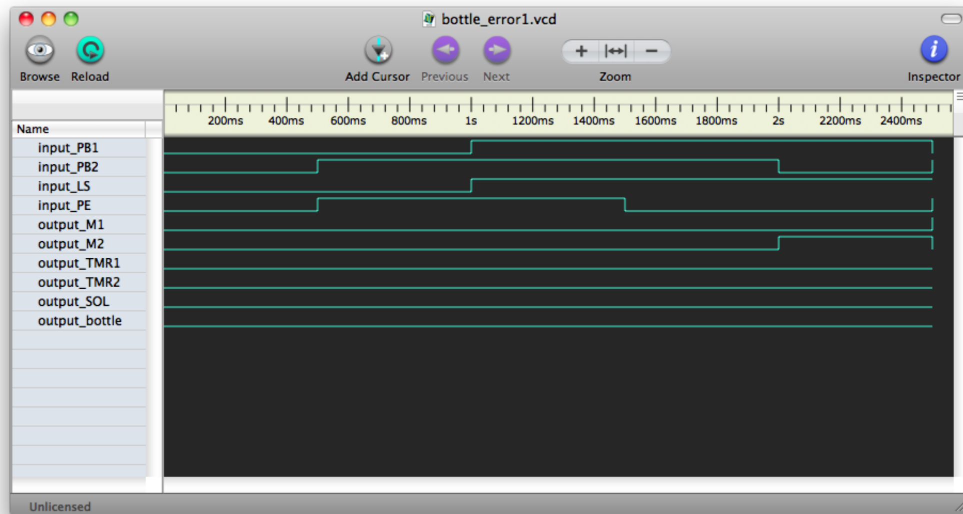

Figura 4.10: Comportamento da implementação da engarrafadora com o Erro 1 injetado.

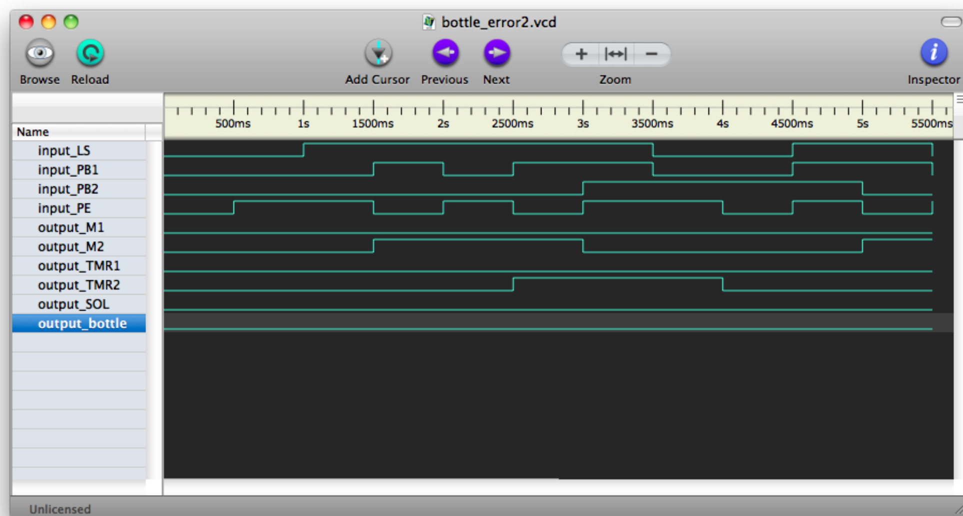

Figura 4.11: Comportamento da implementação da engarrafadora com o Erro 2 injetado.

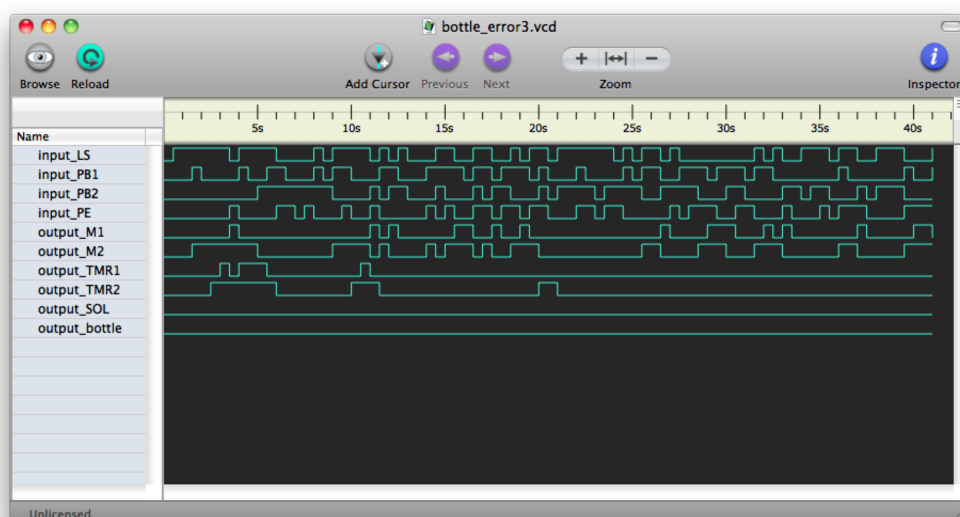

Figura 4.12: Comportamento da implementação da engarrafadora com o Erro 3 injetado.

Capítulo 5

Conclusões

O objetivo principal desse trabalho é aumentar a segurança de sistema de automação e controle fornecendo meios para que desenvolvedores possam executar testes confiáveis de forma 100% automática. Os testes são gerados de forma semi-aleatória, com a finalidade de cumprir os critérios de cobertura de elementos temporizados ou não. O trabalho foi baseado na metodologia de testes baseados em modelos e como resultado foi desenvolvida a ferramenta Gungnir. Algumas adaptações foram feitas com relação à metodologia TBM, a principal é o fato dos casos de testes serem gerados a partir de modelos da especificação e aplicados também a modelos, esses últimos de implementação. Como linguagem de programação de sistemas de controle suportada pelo Gungnir temos o Ladder e como padrão para especificação temos os diagramas ISA 5.2 ambas fazem parte do padrão internacional IEC 61131-3.

Como contribuição deste trabalho ficam destacadas as adaptações feitas nos modelos do trabalho de Vasconcelos Oliveira (2009), a criação de critérios de cobertura junto à heurística do fator determinante que geram um bom nível de segurança dos testes para modelos gerados a partir de um subconjunto reduzido dos diagramas ISA 5.2 e da linguagem Ladder. Também deve ser visto com a devida atenção o simulador de modelos de autômatos temporizados Alur & Dill (1994), parte integrante do Gungnir.

As principais dificuldades encontradas no trabalho foram a implementação do simulador de autômatos temporizados, a obtenção/criação de critérios de cobertura que fornecessem a segurança necessária para os testes gerados, a adaptação do modelo do trabalho de Vasconcelos Oliveira (2009) e a geração direcionada de testes para elementos temporizados.

O trabalho foi todo elaborado utilizando a ideia de *Reproducible Research* todos os arquivos usados e gerados para essa dissertação encontram-se disponíveis para download pelo link: http://bitbucket.org/rodrigopex/msc.

5.1 Trabalhos futuros

Ao fim deste trabalho pode-se concluir que há muitos trabalhos a serem feitos na área de automatização de testes para sistemas de automação e controle. Ficam como trabalhos futuros:

- Desenvolvimento de modelos com baixa complexidade mais aproximados da realidade;
- Expandir o subconjunto de instruções da linguagen Ladder e do padrão ISA 5.2;
- Fazer uma análise estatística mais detalhada sobre a heurística e os critérios de cobertura desenvolvidos neste trabalho;
- Executar estudos de caso mais complexos que possam por realmente a prova o Gungnir no âmbito prático;
- Criação de uma interface socket que implemente algum padrão que possa se comunicar diretamente com o CLP via OPC (*OLE Process Control*) possibilitando, assim, execução online dos testes no CLP;

Bibliografia

(2010*a*).
URL: *http://www.petrobras.com.br*

(2010*b*).
URL: *http://www.logicpoet.com/scansion/*

Alur, R. & Dill, D. L. (1994), 'A theory of timed automata', *Theoretical Computer Science* **126**, 183–235.

Behrmann, G., A, D. & Larsen, P. K. (2004), A tutorial on uppaal, *in* M. Bernardo & F. Corradini, eds, 'LNCS, Formal Methods for the Design of Real-Time Systems (revised lectures)', Vol. 3185, Springer Verlag, pp. 200–237.
URL: *http://doc.utwente.nl/51010/*

Behrmann, G., David, A. & Larsen, K. G. (2004), A tutorial on UPPAAL, *in* M. Bernardo & F. Corradini, eds, 'Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004', number 3185 *in* 'LNCS', Springer–Verlag, pp. 200–236.

Bryan, E. & Bryan, L. (1997*a*), *Programmable controllers theory and implementation*, Pearson Prentice Hall.

Bryan, L. A. & Bryan, E. A. (1997*b*), *Programmable Controllers: Theory and Implementation*, Industrial Text Company. Illustrator-Kory, Gina.

de Vasconcelos Oliveira, K. (2009), Geração automática de testes de conformidade para programas de controladores lógicos programáveis, Master's thesis, Universidade Federal de Campina Grande.

Dong, J. S., Hao, P., Qin, S., Sun, J. & Yi, W. (2008), 'Timed automata patterns', *Software Engineering, IEEE Transactions on* **34**(6), 844 –859.

Hessel, A., Larsen, K., Mikucionis, M., Nielsen, B., Pettersson, P. & Skou, A. (2008), Testing Real-Time systems using UPPAAL, *in* 'Formal Methods and Testing', pp. 77–117.

IEEE Computer Society (2004), *Software Engineering Body of Knowledge (SWEBOK)*, Angela Burgess, EUA.
URL: *http://www.swebok.org/*

Jr., E. M. C., Grumberg, O. & Peled, D. A. (1999), *Model Checking*, The MIT Press.

Larsen, K. G., Mikucionis, M. & Nielsen, B. (2009), *Uppaal Tron User Manual. In http://www.cs.aau.dk/ marius/tron/*.

Mader, A. (2000), 'A classification of plc models and applications'.

Peterson, J. L. (1977), 'Petri nets', *ACM Comput. Surv.* **9**(3), 223–252.

PLCopen (2004), *IEC 61131-3: a Standard Programming Resource. In http://www.plcopen.org/*, PLCopen For Efficiency in Automation.

Reid, S. (2005), 'The art of software testing, second edition. glenford j. myers. revised and updated by tom badgett and todd m. thomas, with corey sandler. john wiley and sons, new jersey, u.s.a., 2004. isbn: 0-471-46912-2, pp 234: Book reviews', *Softw. Test. Verif. Reliab.* **15**(2), 136–137.

Utting, M. & Legeard, B. (2006), *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Apêndice A

Projeto Gungnir

Este apêndice destina-se a registrar aspectos de desenvolvimento do projeto Gungnir.

A.1 Arquitetura geral do Gungnir

A.2 Arquivos XML

Nesta seção estarão todos os códigos XML utilizados nesta dissertação.

Código A.1: Diagrama ISA 5.2 da Engarrafadora

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <isa52>
4      <!-- inputs -->
5      <input>PB1</input>
6      <input>PB2</input>
7      <feedback>M2</feedback>
8      <input>LS</input>
9      <feedback>bottle</feedback>
10     <feedback>TMR1</feedback>
11     <input>PE</input>
12     <feedback>TMR2</feedback>
13
14     <!-- rungs -->
15     <rung>
16         <result>
17             <ref_output name="M2" />
18         </result>
19         <op>
20             <and>
21                 <op>
22                     <or>
23                         <op>
24                             <ref_input name="PB1" />
25                         </op>
26                         <op>
27                             <ref_feedback name="M2" />
28                         </op>
29                     </or>
30                 </op>
31                 <op>
32                     <not>
33                         <op>
34                             <ref_input name="PB2" />
35                         </op>
36                     </not>
37                 </op>
38             </and>
```

```
39          </op>
40      </rung>
41
42      <rung>
43          <result>
44              <ref_output name="M1" />
45          </result>
46          <op>
47              <and>
48                  <op>
49                      <or>
50                          <op>
51                              <not>
52                                  <op>
53                                      <ref_input name="LS" />
54                                  </op>
55                              </not>
56                          </op>
57                          <op>
58                              <ref_feedback name="bottle" />
59                          </op>
60                      </or>
61                  </op>
62                  <op>
63                      <ref_feedback name="M2" />
64                  </op>
65              </and>
66          </op>
67      </rung>
68
69      <rung>
70          <result>
71              <ref_output name="TMR1" />
72          </result>
73          <op>
74              <timer_DI id="01" delay="5">
75                  <op>
76                      <and>
77                          <op>
78                              <ref_input name="LS" />
79                          </op>
80                          <op>
81                              <ref_feedback name="M2" />
82                          </op>
83                      </and>
84                  </op>
85              </timer_DI>
86          </op>
87      </rung>
88
89      <rung>
90          <result>
91              <ref_output name="SOL" />
92          </result>
93          <op>
94              <and>
95                  <op>
96                      <and>
97                          <op>
98                              <ref_feedback name="TMR1" />
99                          </op>
100                         <op>
101                             <not>
102                                 <op>
103                                     <ref_input name="PE" />
104                                 </op>
105                             </not>
106                         </op>
107                     </and>
108                 </op>
109                 <op>
110                     <ref_feedback name="M2" />
111                 </op>
112             </and>
113         </op>
114     </rung>
115
116     <rung>
117         <result>
```

```
118             <ref_output name="TMR2" />
119         </result>
120         <op>
121             <timer_DI id="02" delay="7">
122                 <op>
123                     <and>
124                         <op>
125                             <and>
126                                 <op>
127                                     <ref_feedback name="M2" />
128                                 </op>
129                                 <op>
130                                     <ref_input name="PE" />
131                                 </op>
132                             </and>
133                         </op>
134                         <op>
135                             <not>
136                                 <op>
137                                     <ref_feedback name="bottle" />
138                                 </op>
139                             </not>
140                         </op>
141                     </and>
142                 </op>
143             </timer_DI>
144         </op>
145     </rung>
146
147     <rung>
148         <result>
149             <ref_output name="bottle" />
150         </result>
151         <op>
152             <and>
153                 <op>
154                     <or>
155                         <op>
156                             <ref_feedback name="TMR2" />
157                         </op>
158                         <op>
159                             <ref_feedback name="bottle" />
160                         </op>
161                     </or>
162                 </op>
163                 <op>
164                     <and>
165                         <op>
166                             <ref_feedback name="M2" />
167                         </op>
168                         <op>
169                             <ref_input name="LS" />
170                         </op>
171                     </and>
172                 </op>
173             </and>
174         </op>
175     </rung>
176
177     <!-- outputs -->
178     <output>M2</output>
179     <output>M1</output>
180     <output>TMR1</output>
181     <output>SOL</output>
182     <output>TMR2</output>
183     <output>bottle</output>
184 </isa52>
```

Código A.2: Programa Ladder da Engarrafadora

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <ladder>
4     <!-- inputs -->
5     <input>PB1</input>
6     <input>PB2</input>
7     <feedback>M2</feedback>
```

```
8       <input>LS</input>
9       <feedback>bottle</feedback>
10      <feedback>TMR1</feedback>
11      <input>PE</input>
12      <feedback>TMR2</feedback>
13
14      <!-- rungs -->
15      <rung>
16          <result>
17              <ref_output name="M2" />
18          </result>
19          <op>
20              <and>
21                  <op>
22                      <or>
23                          <op>
24                              <ref_input name="PB1" />
25                          </op>
26                          <op>
27                              <ref_feedback name="M2" />
28                          </op>
29                      </or>
30                  </op>
31                  <op>
32                      <not>
33                          <op>
34                              <ref_input name="PB2" />
35                          </op>
36                      </not>
37                  </op>
38              </and>
39          </op>
40      </rung>
41
42      <rung>
43          <result>
44              <ref_output name="M1" />
45          </result>
46          <op>
47              <and>
48                  <op>
49                      <or>
50                          <op>
51                              <not>
52                                  <op>
53                                      <ref_input name="LS" />
54                                  </op>
55                              </not>
56                          </op>
57                          <op>
58                              <ref_feedback name="bottle" />
59                          </op>
60                      </or>
61                  </op>
62                  <op>
63                      <ref_feedback name="M2" />
64                  </op>
65              </and>
66          </op>
67      </rung>
68
69      <rung>
70          <result>
71              <ref_output name="TMR1" />
72          </result>
73          <op>
74              <timer_TON id="01" delay="5">
75                  <op>
76                      <and>
77                          <op>
78                              <ref_input name="LS" />
79                          </op>
80                          <op>
81                              <ref_feedback name="M2" />
82                          </op>
83                      </and>
84                  </op>
85              </timer_TON>
86          </op>
```

```
87      </rung>
88
89      <rung>
90          <result>
91              <ref_output name="SOL" />
92          </result>
93          <op>
94              <and>
95                  <op>
96                      <and>
97                          <op>
98                              <ref_feedback name="TMR1" />
99                          </op>
100                         <op>
101                             <not>
102                                 <op>
103                                     <ref_input name="PE" />
104                                 </op>
105                             </not>
106                         </op>
107                     </and>
108                 </op>
109                 <op>
110                     <ref_feedback name="M2" />
111                 </op>
112             </and>
113         </op>
114     </rung>
115
116     <rung>
117         <result>
118             <ref_output name="TMR2" />
119         </result>
120         <op>
121             <timer_TON id="02" delay="7">
122                 <op>
123                     <and>
124                         <op>
125                             <and>
126                                 <op>
127                                     <ref_feedback name="M2" />
128                                 </op>
129                                 <op>
130                                     <ref_input name="PE" />
131                                 </op>
132                             </and>
133                         </op>
134                         <op>
135                             <not>
136                                 <op>
137                                     <ref_feedback name="bottle" />
138                                 </op>
139                             </not>
140                         </op>
141                     </and>
142                 </op>
143             </timer_TON>
144         </op>
145     </rung>
146
147     <rung>
148         <result>
149             <ref_output name="bottle" />
150         </result>
151         <op>
152             <and>
153                 <op>
154                     <or>
155                         <op>
156                             <ref_feedback name="TMR2" />
157                         </op>
158                         <op>
159                             <ref_feedback name="bottle" />
160                         </op>
161                     </or>
162                 </op>
163                 <op>
164                     <and>
165                         <op>
```

```
166                             <ref_feedback name="M2" />
167                         </op>
168                         <op>
169                             <ref_input name="LS" />
170                         </op>
171                     </and>
172                 </op>
173             </and>
174         </op>
175     </rung>
176 
177     <!-- outputs -->
178     <output>M2</output>
179     <output>M1</output>
180     <output>TMR1</output>
181     <output>SOL</output>
182     <output>TMR2</output>
183     <output>bottle</output>
184 </ladder>
```

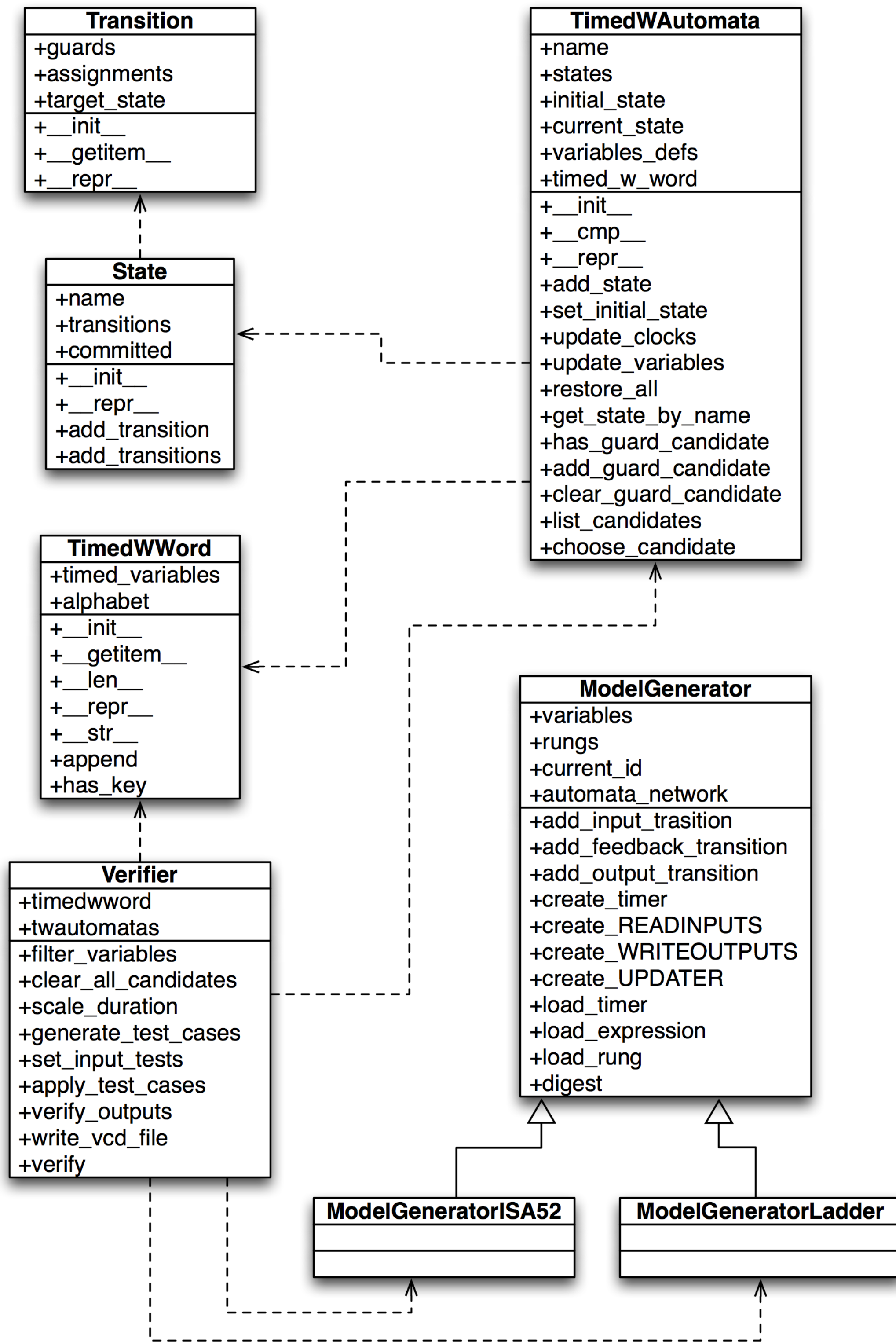

Figura A.1: Arquitetura geral simplificada do Gungnir