Geração Automática de Modelos de Autômatos Temporizados para Realização de Testes de Sistemas Instrumentados de Segurança

Luiz Paulo de Assis Barbosa

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Engenharia da Computação

Antonio Marcus Nogueira Lima, D.Sc.
Orientador

Campina Grande, Paraíba, Brasil

Geração Automática de Modelos de Autômatos Temporizados para Realização de Testes de Sistemas Instrumentados de Segurança

Luiz Paulo de Assis Barbosa

Dissertação de Mestrado apresentada em Dezembro de 2008

Antonio Marcus Nogueira Lima, D.Sc.
Orientador

Campina Grande, Paraíba, Brasil, Dezembro de 2008

Dedicatória

Dedico este trabalho a minha família, pais, tias, irmãs e principalmente a meus avós a quem tanto amo por suas sábias lições de perseverança, pelo amor incondicional e apoio constante em todos os momentos da minha vida.

Agradecimentos

Ao meu orientador, professor D.Sc. Antônio Marcos Nogueira Lima pelos conhecimentos transmitidos e pela atenção dispensada que possibilitaram a construção desse trabalho.

Ao professor D.Sc. Angelo Perkusich pela sua maneira peculiar de incentivo e pela oportunidade de trabalhar no Lab. Embedded onde pude desfrutar de toda infra-estrutura necessária para a realização da pesquisa e desenvolvimento acadêmico durante o mestrado.

Ao professor D.Sc. Leandro Dias da Silva e a Kyller Costa Gorgônio agradeço pelo incentivo, pela importante colaboração na discussão das matérias técnicas, pelas proveitosas idéias e sugestões que muito contribuíram para a realização desse trabalho.

Aos meus amigos Diego, Larissa, Lorena, Jonas e Estela, agradeço pelo convívio, pelos momentos de descontração, pela solidariedade, presteza e pela amizade compartilhada. A Adrian, Danilo, Olympio, Paulo Ribeiro, Taciana e todas as meninas do Apto. 24, pessoas que me proporcionaram alegria com sua presença em momentos de total divertimento e um valoroso suporte com votos de incentivo.

A CAPES, pelo apoio financeiro.

A todos aqueles que, embora não mencionados, apoiaram e contribuíram para realização deste trabalho, o meu reconhecido muito obrigado!

Resumo

Os Sistemas Instrumentados de Segurança (SIS) são projetados para garantir a operação contínua de sistemas de produção industrial, controlar o comportamento dos equipamentos envolvidos no processo produtivo e prevenir acidentes, portanto, é importante testar a implementação dos SIS contra sua especificação para aumentar a confiança no funcionamento do sistema. Neste trabalho é introduzida uma técnica para melhorar a confiança no funcionamento dos SIS. Um método para obter automaticamente um modelo de autômato temporizado da especificação, diagramas ISA 5.2, e da implementação do SIS, *Function Block Diagram* (FBD), é apresentado. Uma abordagem para realizar teste automático da implementação utilizando os modelos gerados é discutida e finalmente a técnica é aplicada a um estudo de caso provido pela Petrobras. O método introduzido aqui é baseado no uso do verificador de modelos Uppaal e na ferramenta de teste Uppaal-TRON.

Abstract

Safety Instrumented Systems (SIS) are designed to guarantee continuous operation of industrial production systems, control the behavior of the industrial equipments and prevent accidents, therefor, it is importante to be able to test the SIS implementation against its specification in order to increase the dependability of the system. In this work a technique to improve the dependability of SIS is introduced. A method to automatically obtain a timed automaton model of the specification, diagrams ISA 5.2, and the implementation of SIS, *Function Block Diagram* (FBD), is presented. One approach to perform automatic testing of the implementation using the models generated is discussed and finally the the technique is applied to a case study provided by Petrobras. The method introduced here is based on the use of the Uppaal model checker and the Uppaal-TRON testing tool.

Índice

Lista de Figuras

Capítulo 1

Introdução

Todo processo industrial envolve risco. Segurança absoluta nunca pode ser alcançada, o risco pode apenas ser reduzido a um nível aceitável. No projeto de sistemas de automação industrial para determinar o nível de segurança requerido para as diversas partes de uma planta é realizada uma análise detalhada dos riscos. Se o resultado dessa análise indica que apenas o Sistema de Controle (SC) é insuficiente para promover o nível de segurança exigido, pode ser necessário o uso dos Sistemas Instrumentados de Segurança (SIS). Algumas instalações industriais, como as plantas de extração e processamento de petróleo, são críticas com relação à segurança e constituem um típico caso onde o uso de SIS é requerido. O motivo para tanto é que uma falha no sistema de automação e controle desse tipo de instalação pode ocasionar conseqüências muito graves, causar perdas de vidas humanas, danos ambientais irreversíveis e enormes prejuízos econômicos.

Os sistemas de segurança geralmente são mantidos funcionalmente separados do SC, pois o objetivo é proteger o equipamento controlado no caso de falhas operacionais, erros de medição e inclusive quando o SC falhar. A configuração básica de uma planta industrial que utiliza um sistema de proteção é representada na Figura 1.1.

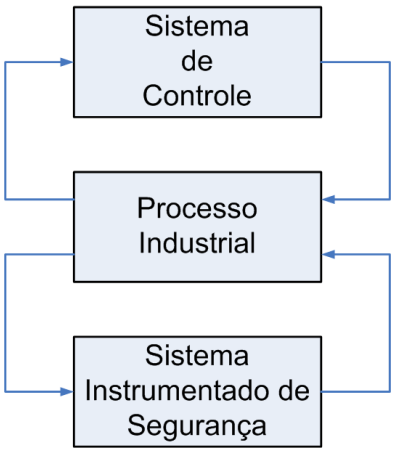

Figura 1.1: Configuração básica de uma planta com SIS.

Nos SIS as informações sobre o estado das partes da planta nas quais o SIS é requerido são processadas em um Controlador Lógico Programável (CLP) de segurança e a presença de redundância é obrigatória em todo o sistema. Quando solicitados, os SIS podem interromper parte do processo ou, em alguns casos, desligar totalmente a planta, por isso, são conhecidos também como sistemas de desligamento ou paralisação de emergência (ESD do inglês, *Emergency Shutdown*) ou sistemas de desligamento de segurança (SSD do inglês, *Safety Shutdown*) [13].

A configuração básica de um SIS está representada na Figura 1.2, na qual os os blocos em tonalidade escura representam os elementos externos ao CLP, tais como: interfaces de programação remota, redes de comunicação para dispositivos industriais; e os sensores e atuadores, os quais interagem diretamente com o processo industrial. Os demais blocos representam a estrutura do CLP, o qual possui alimentação própria e uma interface de comunicação através da qual é implementada a comunicação com outros CLPs e informações sobre o seu funcionamento podem ser acessadas. A interface de entrada permite obter informações do processo industrial através da interação com os sensores e a interface de saída permite enviar informações para o processo industrial através da interação com os atuadores.

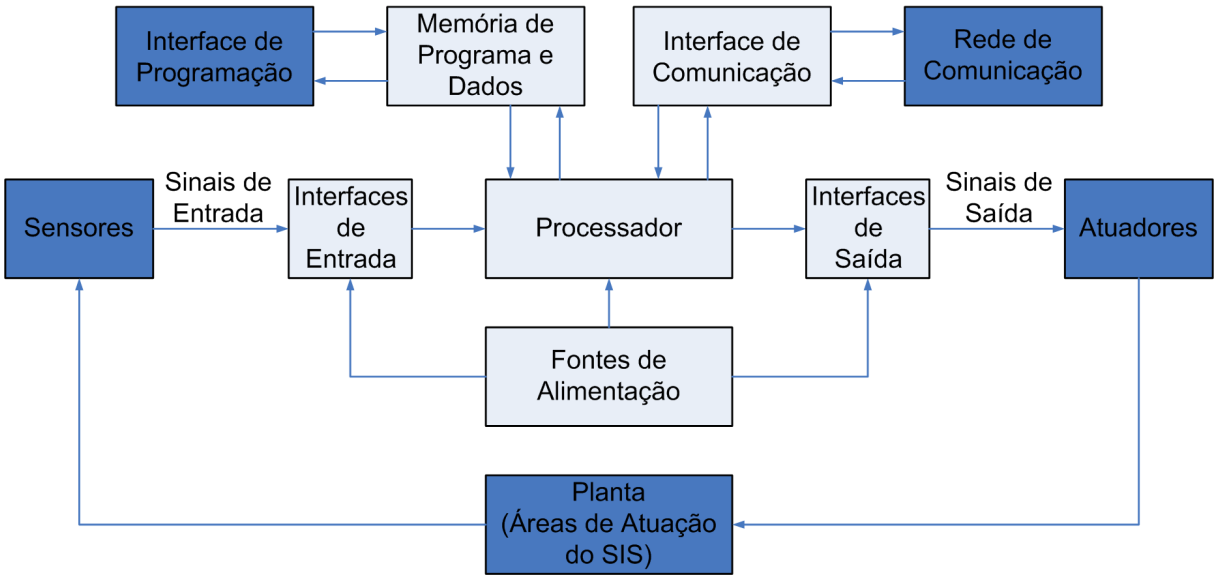

Figura 1.2: Configuração básica de um SIS.

O projeto de sistemas, independentemente de seu tipo, deve seguir algum tipo de metodologia. O uso de uma metodologia objetiva criar um roteiro a ser seguido e estabelecer uma boa comunicação e entendimento entre as pessoas responsáveis pelo desenvolvimento do sistema. Uma boa comunicação evita que erros sejam cometidos devido à falta de informação sobre o funcionamento, implementação e integração dos componentes do sistema. Várias metodologias para realização de projetos foram propostas, tais como: *top-down*,

bottom-up, espiral, refinamentos sucessivos, entre outras. Cada uma dessas metodologias possui características, vantagens e desvantagens que devem ser consideradas de acordo com o tipo de projeto a ser realizado.

Tomando como exemplo a metodologia *top-down*, ilustrada na Figura 1.3, temos que a primeira etapa do projeto de um sistema é o levantamento dos requisitos. Nesta etapa é descrito informalmente o que o sistema faz de forma clara e objetiva, evitando ambigüidades. Um levantamento de requisitos mal feito pode confundir o desenvolvedor induzindo a implementações incorretas.

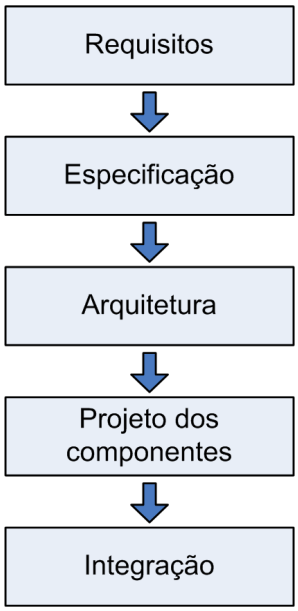

Figura 1.3: Seqüência de etapas da metodologia *top-down*.

A segunda etapa é a especificação. Nela os requisitos são descritos em uma linguagem formal. O motivo para tanto é evitar ambigüidades e permitir o uso das informações durante o processo de desenvolvimento. Ao final dessa etapa, tem que estar claro o que o sistema deve realizar, mas ainda não está definido como ele será implementado.

A terceira etapa é definição da arquitetura do sistema. O sistema é descrito funcionalmente e são usados diagramas de blocos para mostrar como os componentes e módulos do sistema são conectados. As funções de cada componente são descritas e é feita a identificação de quais componentes de software e de hardware serão necessários no projeto do sistema.

O projeto de componentes é necessário quando algum componente requerido pelo sistema precisa ser implementado ou adaptado. O desenvolvimento de um componente deve seguir uma metodologia, a exemplo do projeto do sistema como um todo.

A última etapa é a integração das partes desenvolvidas separadamente e que juntas irão compor o sistema. Nessa etapa normalmente os erros de implementação aparecem,

por isso, é importante testar consistentemente tanto as partes isoladamente como a composição das mesmas de forma a evitar comportamentos indesejados no produto final.

O cenário de desenvolvimento do código de um SIS, representado na Figura 1.4, localiza-se no contexto da metodologia *top-down* nas etapas de projeto de componentes e integração do sistema. O projeto do código do SIS segue uma metodologia próxima à metodologia *top-down* apresentada, sendo constituído das seguintes etapas:

- levantamento dos requisitos do SIS;
- especificação do SIS;
- desenvolvimento e teste do código do SIS por uma equipe especializada em automação industrial;
- e por fim, é realizado o comissionamento o qual entre outras coisas irá determinar se o código será aceito ou se necessita de correções.

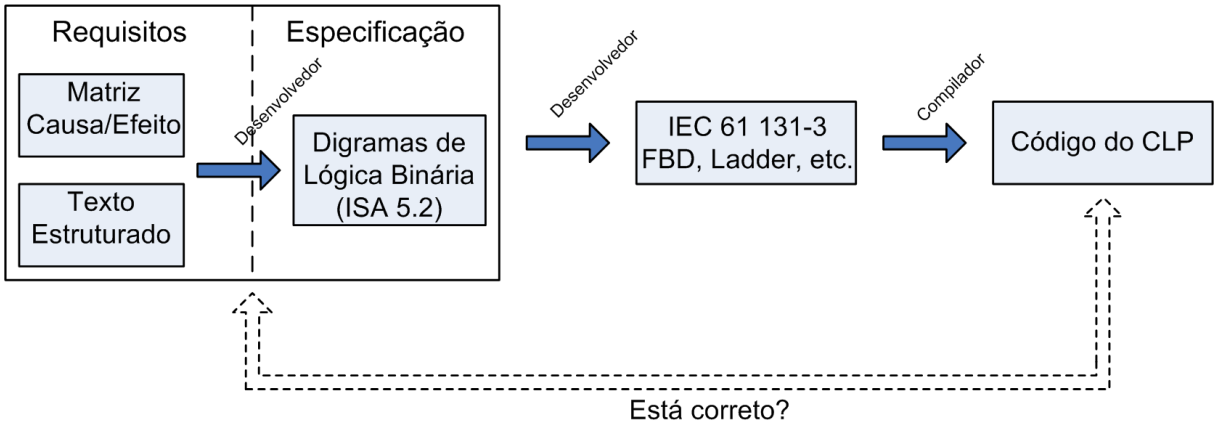

Figura 1.4: Cenário de desenvolvimento do código do SIS.

Na Figura 1.4 os requisitos são representados na forma de uma matriz ou tabela de causa/efeito e de um texto estruturado. Esse tipo de matriz relaciona um conjunto de situações de risco que podem ser observadas durante o funcionamento da planta, as causas, a um conjunto de ações que devem ser realizadas para levar a planta a um estado seguro, os efeitos. No texto são explicadas situações que não estão presentes na matriz e detalhados a lógica e os procedimentos que devem ser realizados para garantir a operação segura do processo industrial. Tomando como base os requisitos, a especificação é produzida na forma de diagramas de lógica binária seguindo o padrão ISA 5.2 [15] e a partir dela, o código do SIS é escrito utilizando uma das linguagens definidas no padrão IEC 61131-3 [16], para ser executado em um CLP. O cenário apresentado corresponde à forma como a empresa Petrobras realiza o desenvolvimento dos seus sistemas de segurança.

Para testar se o código do SIS está em conformidade com o especificado existem duas formas: a primeira caracteriza-se pela utilização de verificação de modelos [7], essa técnica permite que propriedades específicas do SIS sejam definidas e verificadas, utilizando para tanto uma linguagem temporal para especificar as propriedades e um modelo do sistema como base para a verificação. Propriedades de segurança, de vivacidade, de alcançabilidade de estados específicos, de ausência de *deadlocks*, entre outras, podem ser verificadas. A segunda abordagem possível é realizar testes com o código, para aumentar a confiança no funcionamento do SIS.

Durante o desenvolvimento dos SIS a abordagem baseada em testes é utilizada, porém, as partes do programa são testadas isoladamente à medida que são desenvolvidas, com isso, erros provocados por interações entre as partes previamente testadas podem não ser detectados. Os erros não detectados durante a etapa de desenvolvimento, devido à impossibilidade de realização de testes em maior número e qualidade, possivelmente serão descobertos no comissionamento, provocando o retorno do SIS à fase de desenvolvimento.

No comissionamento é realizado um Teste de Aceitação de Fábrica (TAF). O TAF é realizado da seguinte maneira: as entradas da matriz de causa/efeito são reproduzidas em campo e o comportamento do SIS para essas entradas é comparado com o comportamento especificado. Se o comportamento corresponde ao especificado, o código é aceito, caso contrário, a equipe responsável pelo desenvolvimento do código realiza as correções necessárias. O TAF é um processo que despende muito tempo, geralmente dias ou mesmo semanas, devido às temporizações utilizadas na operação dos processos industriais e a grande quantidade de entradas a serem testadas, tipicamente matrizes de causa/efeito 1000×1000. Não é desejável, portanto, que erros que poderiam ser detectados na fase de desenvolvimento permaneçam no código até o comissionamento.

A problemática abordada neste trabalho é a seguinte: a empresa que contrata o serviço de desenvolvimento do SIS não tem garantia formal de que o código entregue pela empresa contratada está em conformidade com a especificação do SIS. Resta então realizar o comissionamento para determinar a aceitação do código. Uma possível causa para o problema identificado acima é a falta de uma metodologia que proporcione uma forma de testar se o código do SIS está em conformidade com a sua especificação.

1.1 Objetivos

O presente trabalho está inserido no contexto do desenvolvimento de um método para aumentar a confiança no funcionamento dos programas para SIS. O aumento da confiança no funcionamento do SIS é fundamentado no seguinte princípio: quanto maior o número de testes realizado com o SIS maior será a confiança no funcionamento desse sistema. A

criação, desenvolvimento e implantação desse método fazem parte das atividades do Projeto SIS, realizado na UFCG com o apoio da Petrobras. O objetivo do projeto é permitir que testes sejam realizados em todo o SIS implementado até o momento, interferindo o mínimo possível no cenário apresentado na Figura 1.4. A aplicação desse método auxiliará os desenvolvedores na detecção prematura de problemas no código do SIS, o que conseqüentemente reduzirá os erros no produto final.

Uma importante característica do método é a realização de teste automático. No intuito de implementar essa funcionalidade serão utilizadas ferramentas de software desenvolvidas especialmente para realização de testes. Essas ferramentas são capazes de gerar casos de teste, utilizando para tanto um modelo da especificação e uma descrição do ambiente industrial, e aplicá-los a um modelo da implementação. Concluídos os testes é obtido um veredicto indicando se houve ou não conformidade entre o comportamento de um modelo com relação ao outro. Quando são detectadas diferenças de comportamento entre os modelos, além do veredicto negativo, informações sobre o traço de execução corrente bem como os estados do modelo antes da falha são fornecidas para análise e localização da causa do erro. Observe que para viabilizar o uso das ferramentas de teste a construção de modelos formais da especificação e do código do SIS é necessária.

A modelagem de sistemas físicos, por se tratar de uma atividade não trivial, é realizada por equipes especializadas e quanto maior e mais complexo o sistema a ser modelado maior é o tempo gasto para o desenvolvimento do modelo. A implantação do método procura auxiliar no desenvolvimento do SIS, porém, deve atender ao requisito de modificar o mínimo possível o modo como esse desenvolvimento é realizado atualmente. Dessa forma, quanto mais transparentes para os desenvolvedores puderem ser as etapas de construção dos modelos e realização dos testes melhor. Nenhuma dessas etapas deve representar um atraso demasiado no desenvolvimento do SIS, nem a necessidade de contratação de pessoal especializado para sua realização.

Considerando o mencionado anteriormente percebe-se a necessidade de definição de um procedimento para obtenção dos modelos formais da especificação e do código do SIS. Esses modelos devem ser gerados automaticamente a partir das informações presentes nos documentos ISA 5.2 e no código FBD.

O presente trabalho consiste na definição do procedimento de obtenção dos modelos formais necessários para aplicação do método ao desenvolvimento do SIS, bem como o desenvolvimento de uma ferramenta de software que realize a geração dos modelos de forma automática. A integração de ferramentas para geração de modelos e realização de teste automático as atuais ferramentas CAD utilizadas no desenvolvimento dos SIS representaria uma melhoria significativa no processo de desenvolvimento realizado atualmente. Esse cenário é explorado mediante o desenvolvimento de um estudo de caso que

tem como objetivo validar o procedimento de geração dos modelos e demonstrar o uso de teste automático no desenvolvimento de um SIS.

No caso no qual o gerador de casos de teste possa considerar especificidades do ambiente industrial no qual o SIS será instalado, a aplicação do método pode ser utilizada também para auxiliar o comissionamento. Considere que um caso de teste previamente aplicado aos modelos do código e da especificação obteve um veredicto positivo, porém, a mesma combinação de entradas ao ser testada no TAF não passou no teste. Numa situação como essa é provável que o resultado negativo do teste não tenha sido provocado por um erro no código e sim por um erro de medição, inversão de cabos, falha de dispositivos de campo, entre outras causas. Portanto, a utilização de teste automático durante o comissionamento pode ser útil na investigação da origem dos erros observados. Outra possibilidade é utilizar o gerador de casos de teste em campo, gerando as combinações de entrada diretamente para o CLP de segurança, através da tecnologia OPC [1]. Por último, utilizando os modelos, uma quantidade bem maior de casos de teste que aquela realizada no TAF pode ser aplicada o que implica num aumento na confiança no funcionamento do SIS desenvolvido.

1.2 Organização

O presente documento está organizado da seguinte forma: no Capítulo 2, são introduzidas as soluções propostas para a modelagem e verificação de programas escritos utilizando as linguagens do padrão IEC 61131-3 para serem executados em CLPs.

No capítulo 3, é apresentada uma descrição detalhada do método para aumento da confiança dos SIS e realizada uma discussão sobre as semelhanças e diferenças do presente trabalho com as demais abordagens mencionadas no Capítulo 2.

No Capítulo 4, são apresentadas as definições de Sistemas dirigidos a Eventos Discretos (SED), autômatos de estados finitos, Autômatos Temporizados (AT), da semântica de um AT e de uma rede de AT.

No Capítulo 5, todo processo de desenvolvimento dos modelos é apresentado em detalhes. Primeiro o objeto da modelagem é identificado; a seguir são discutidas a concepção e a organização do modelo; na seqüência são descritos os arquétipos dos autômatos, suas funções e o modo como interagem durante a execução do modelo; e por fim é discutido o modo como as informações da especificação e do código do SIS são extraídas e combinadas com os arquétipos dos autômatos, de forma automática, para formar os modelos.

No Capítulo 6, é apresentado um estudo de caso demonstrando o uso do método para o desenvolvimento de um SIS. No Capítulo 7, uma análise dos resultados alcançados com

[1] http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp

o presente trabalho é realizada e sugestões de melhoramentos e trabalhos futuros são apresentadas.

Capítulo 2

Trabalhos Relacionados

Neste capítulo são discutidas algumas soluções propostas para a modelagem e verificação de programas para CLPs descritos utilizando as linguagens do padrão IEC 61131-3, mais especificamente IL que é textual, LD e FBD ambas linguagens gráficas para as quais os programas são representados através de diagramas.

Em Canet *et al.* [5], a verificação automática de programas para CLPs escritos em IL é discutida. É proposta uma semântica formal para um fragmento da linguagem IL, a qual é diretamente codificada para uma ferramenta de verificação automática de modelos. Diversos tipos de propriedades podem ser verificadas como invariantes, propriedades de segurança, de vivacidade e possivelmente propriedades aninhadas e combinadas de modo arbitrário. A restrição imposta para aplicação do método proposto é o tratamento apenas para programas simples, ou seja, programas compostos de um módulo, que possuam apenas variáveis Booleanas ou variáveis inteiras de faixa limitada e sem temporização.

Em Mader & Wupper [17], o objetivo é prover uma base para verificação e teste das propriedades de tempo-real de aplicações para CLPs. É definida uma semântica formal para programas escritos em IL com ênfase no comportamento dos temporizadores, os quais são modelados por AT. Nenhuma verificação ou teste é realizada com os modelos. Os programas a serem modelados devem conter apenas variáveis Booleanas.

Em Heiner & Menzel [14], é definida uma semântica para um subconjunto da linguagem IL, a qual é descrita em um modelo de Redes de Petri (RP) [20]. É proposta a realização de um ciclo de verificação, no qual um modelo de RP do programa em IL e um modelo em RP do ambiente ou planta são combinados para formar o modelo do sistema. Dos requisitos do sistema é extraído um conjunto de comportamentos a serem provados, os quais são expressos através de fórmulas escritas em LTL (do inglês, *Linear Temporal Logic*) [21]. O conjunto de fórmulas LTL pode ser provado através da utilização de um verificador de modelos. Nesse trabalho nenhuma verificação é realizada com os modelos, apenas é proposto um ciclo de validação que prevê o uso de um verificador de modelos.

Os programas a serem modelados devem conter apenas variáveis Booleanas.

Em Willems [24], um conjunto de ferramentas é desenvolvido para converter programas em IL num modelo de AT com o objetivo de facilitar a verificação desses programas. Nesse trabalho o modelo de autômatos é dividido em duas partes, uma temporizada e outra não temporizada. A parte não temporizada, que é tipicamente maior que a parte temporizada, é aplicada um método para redução do espaço de estados através do uso das ferramentas CADP (do inglês, *Caesar/Aldebaran Development Package*). A geração dos modelos para a parte temporizada é realizada automaticamente, em duas etapas: a conversão de IL para um formato intermediário e posteriormente do formato intermediário para AT. Para a parte não-temporizada o procedimento de geração do modelo é diferente, primeiro é realizada a conversão para o formato LOTOS (do inglês, *Language Of Temporal Ordering Specification*), seguida da redução do espaço de estados e finalmente a conversão para AT. Nesse trabalho são considerados apenas os temporizadores TON. Variáveis do tipo Inteiro e Booleanas são suportadas. É utilizado o verificador de modelos Uppaal e não há integração entre as ferramentas desenvolvidas em uma plataforma para o desenvolvimento de sistemas de controle. Observe que foi mencionado o uso de temporizador TON, que é um bloco funcional e faz parte da linguagem FBD, em um programa escrito em IL. Isso é possível porque o uso de blocos funcionais nas linguagens LD, IL e ST é permitido pelo padrão IEC 61131-3. Na linguagem LD o bloco funcional é inserido em um degrau do programa e para as linguagens textuais IL e ST o bloco é utilizado através de uma chamada à função.

Em Rossi & Schnoebelen [22], a verificação formal de programas escritos em LD é discutida. Nesse trabalho é definida uma semântica para LD descrita através de autômatos finitos e o verificador de modelos SMV [18] é utilizado para realizar a verificação formal das propriedades do sistema de forma automática. As propriedades são expressas em LTL e CTL (do inglês, *Computation Tree Logic*) [7] e [11]. O tempo é considerado de forma qualitativa, isso é feito sob a justificativa de simplificação computacional do modelo. É utilizada uma abstração do temporizador TON para simular o repouso, a contagem e o momento de disparo desse elemento. A mudança entre as localidades do autômato que modela o TON é realizada de forma não-determinística sem levar em conta os tempos programados para o temporizador. Os programas devem conter apenas variáveis Booleanas.

Em Moon [19], a verificação de propriedades de segurança e operabilidade dos programas para CLPs escritos em LD é discutida. O método proposto consiste de um modelo para o sistema, ou seja, uma representação baseada em lógica Booleana para o comportamento do CLP. Um conjunto de asserções, que são questões sobre o comportamento do sistema expressas em CTL e um verificador de modelos, o qual, utilizando o modelo do

sistema e o conjunto de asserções como entradas, gera o espaço de estados e verifica a consistência do modelo com relação as asserções. No caso de inconsistência um contra-exemplo é gerado para análise. Nesse trabalho a temporização não é considerada e não são utilizados procedimentos automáticos para a geração do modelo do sistema e das asserções.

Em Baresi *et al.* [2], é apresentado um conjunto de ferramentas chamado PLCTools, o qual fornece um ambiente integrado para projeto e análise de sistemas de controle. O objetivo do trabalho é a geração automática do código fonte para o sistema de controle. O PLCTools utiliza equações diferenciais para modelar a planta, linguagem FBD para o projeto do *software* do controlador e (HLTPN do inglês, *High Level Timed Petri Nets*) [12] para validação do projeto e geração do código de controle. É fornecido um ambiente gráfico baseado na ferramenta *MATLAB/SIMULINK* para o desenvolvimento do código FBD, o qual é automaticamente traduzido para um modelo formal de HLTPN. Durante o desenvolvimento do código é possível realizar simulações utilizando o modelo formal.

Em Younis & Frey [25], o crescente interesse em métodos para realizar a descrição formal dos programas para CLPs é discutido. São identificadas duas razões principais que impulsionam a formalização dos programas para CLPs. A primeira é a necessidade de realizar análise, simulação verificação e validação dos sistemas de controle existentes e em desenvolvimento, visando melhorar sua segurança e qualidade. A segunda é promover a extração de uma descrição formal de código já existente e não documentado, necessária no processo de re-implementação. A re-implementação caracteriza-se pela realização de modificações no código de controle necessárias devido a mudanças no processo produtivo. Ademais, é proposta a classificação das técnicas de formalização segundo os itens: fontes para formalização, níveis de formalização, objetivos a serem atingidos com a formalização e modelo formal usado para a descrição do programa. As fontes para formalização são as linguagens de programação para CLPs e alguma informação adicional sobre o ambiente/planta. Os níveis de formalização compreendem a descrição de parte dos programas de controle, do programa de controle completo ou de uma configuração de controle completa, geralmente envolvendo diversos CLPs. Os objetivos a serem atingidos com a formalização são a realização de engenharia reversa e a verificação e validação do programa de controle. Os modelos formais usados para a descrição do programa são os mais diversos, entre eles o autor destaca autômatos e redes de Petri.

Capítulo 3

O Método

Neste capítulo é apresentada uma descrição detalhada do método para aumento da confiança dos SIS e realizada uma discussão sobre as semelhanças e diferenças do presente trabalho com as demais abordagens mencionadas no Capítulo 2.

O método para o aumento da confiança no funcionamento de SIS pode ser classificado de acordo com o proposto em Younis & Frey [25] da seguinte maneira: as fontes para formalização são a especificação do SIS representadas por diagramas ISA 5.2, os programas para SIS escritos utilizando a linguagem FBD e um ambiente que reproduz o ciclo de varredura do CLP e no qual as variáveis de entrada podem assumir valores de forma não-determinística. O nível de formalização desejado é a descrição completa da especificação, do programa do SIS e do ambiente. O objetivo é aumentar a confiança no funcionamento do SIS através da realização de testes de conformidade e a descrição formal escolhida é a modelagem por AT.

A Figura 3.1 representa a integração do método ao cenário de desenvolvimento de um SIS. A especificação do programa do SIS e a implementação do mesmo são fornecidas como entradas para o método. Segue-se então a geração automática dos modelos de AT de cada uma das entradas fornecidas. Observe que para a geração dos modelos uma descrição do ambiente deve ser considerada. A geração dos autômatos possui um passo intermediário que é a tradução das entradas do método para uma representação em XML. Isso é feito para que a ferramenta responsável pela geração dos modelos possa acessar as informações contidas na especificação e no código FBD. Finalmente, são realizados os testes de conformidade para obtenção de um veredicto sobre a implementação. Existe a possibilidade de utilizar a tecnologia OPC [1] para realizar os testes diretamente na implementação, ou seja, no código executado pelo CLP e não em um modelo. A tecnologia OPC fornece um arcabouço de software para desenvolvimento de clientes OPC, o TRON é um deles, facilitando o acesso as funcionalidades de CLPs de diversos fabricantes. No

[1] http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp

entanto, a abordagem que prevê o uso de OPC não será desenvolvida no presente trabalho.

Definir um formato de arquivo para representar a especificação e o código foi necessário porque os arquivos existentes não podiam ser lidos. Por simplicidade, optou-se por utilizar a mesma formatação para ambas as entradas do processo de geração, bastando identificar se o arquivo refere-se a um diagrama ISA 5.2 ou código FBD. Foi possível utilizar a mesma formatação para ISA 5.2 e FBD porque ambos são interpretados de forma semelhante e para a geração apenas as informações das conexões entre os elementos e do tipo do elemento são relevantes. Cada um dos padrões, ISA 5.2 e FBD, possui um conjunto próprio de símbolos que permite a descrição de diversos sistemas. Para o desenvolvimento da ferramenta de tradução dos diagramas para AT foi definido um subconjunto de símbolos de ambos os padrões, ISA 5.2 e FBD, que seria interpretado pela ferramenta. Os símbolos suportados são os utilizados para representar as operações lógicas E e OU, os temporizadores básicos e os sinais de entrada e saída dos diagramas. A escolha de código escrito em FBD como implementação foi motivada pelo crescente uso dessa linguagem de programação para CLPs.

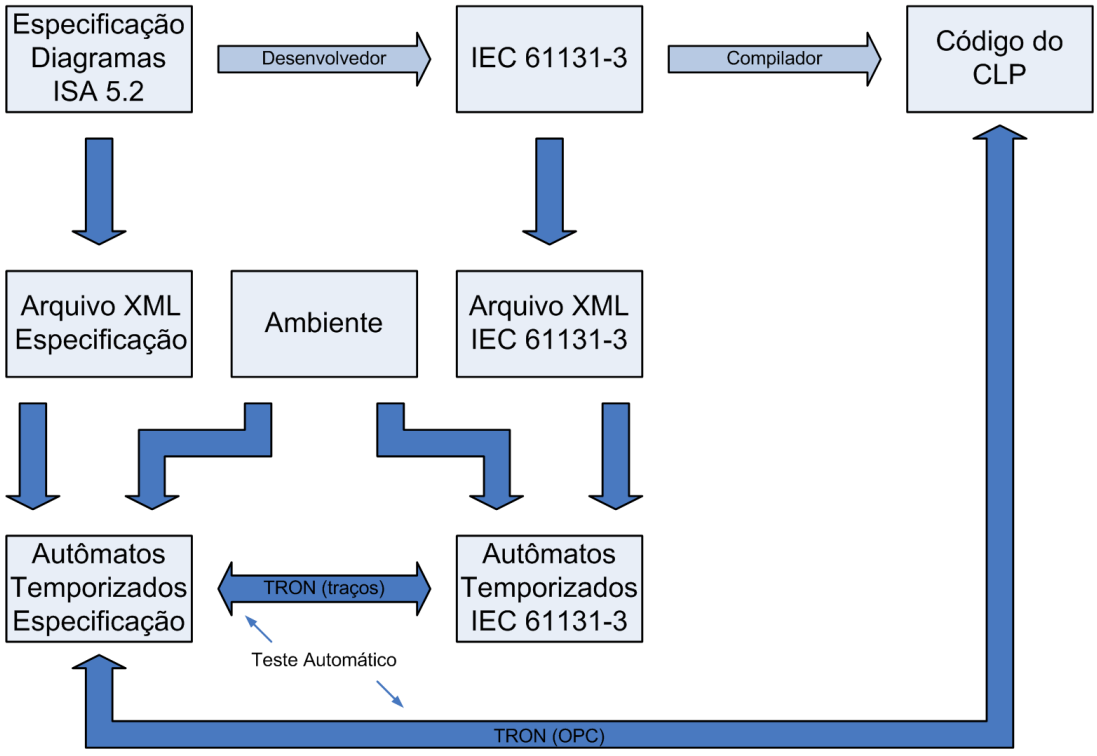

Figura 3.1: Diagrama ilustrando a aplicação da metodologia ao processo de desenvolvimento de um SIS.

Com base na teoria de AT [1], é possível representar formalmente o comportamento de Sistemas à Eventos Discretos (SED) com restrição de tempo-real. A geração automática de modelos permite reduzir o tempo gasto para realizar a modelagem de sistemas de grande porte e garante, por construção, que os modelos gerados estão corretos, uma vez

que as regras de geração estão definidas no algoritmo implementado no software que gera os modelos. A escolha por utilizar testes foi motivada pela complexidade computacional associada aos métodos de verificação formal. O tamanho dos sistemas que possivelmente teremos de lidar é um fator importante, devido ao problema da explosão do espaço de estados, observado quando a verificação de modelos é utilizada. O crescimento exponencial do espaço de estados está diretamente relacionado ao tamanho dos sistemas.

Testar segundo Tretmans [23], consiste em checar a correção ou integridade de uma implementação de determinado sistema através da experimentação da mesma. Casos de testes são aplicados a IUT (do inglês, *Implementation Under Test*) em um ambiente controlado e baseado em observações feitas durante a execução dos testes um veredicto sobre o funcionamento da implementação é dado. O critério utilizado para determinar o funcionamento correto da IUT é a especificação, a qual descreve o que o sistema foi projetado para fazer e o que não deve ser realizado pelo mesmo. Portanto, a especificação serve como base para geração dos casos de testes que devem ser aplicados a IUT. Todo sistema em desenvolvimento deve, em algum momento, passar por uma fase de testes. A medida que a quantidade e a qualidade dos testes realizados aumenta, a confiança no funcionamento do sistema também aumenta. Assim, a realização de testes, utilizando modelos do sistema desenvolvido, serve justamente para aumentar a confiança no funcionamento do sistema.

O processo de testes pode ser dividido em duas fases distintas: a geração dos casos de teste e a execução dos testes. A geração dos casos de teste é realizada mediante uma análise da especificação, que determinará que funcionalidades serão testadas, como estas funcionalidades serão testadas e finalmente, como devem ser escritos os testes para a IUT que se deseja testar. O processo de execução dos testes envolve a criação de um ambiente controlado onde os mesmos possam ser realizados, a realização dos testes e a análise dos resultados para gerar um veredicto sobre o funcionamento da IUT [23].

Dentre os diversos tipos de testes existentes o que melhor aplica-se no contexto desse trabalho são os testes de conformidade formal. Eles têm como objetivo responder a seguinte questão: *o comportamento do sistema está em conformidade com o que foi descrito em sua especificação?*. Entenda-se por teste de conformidade formal o processo de checar o correto funcionamento de um sistema caixa-preta em relação à especificação formal do mesmo, ou seja, uma especificação descrita em uma linguagem com uma semântica formalmente definida.

A principal semelhança do método descrito com os trabalhos mencionados no Capítulo 2 é a realização da descrição formal de uma linguagem de programação para CLPs. A principal diferença é a realização de testes de conformidade, enquanto a maioria dos trabalhos discutidos usa verificação de modelos. As exceções são [17], que considera a possibilidade

de realizar verificação e testes, porém não os realiza, e [2] que utiliza simulação para o desenvolvimento do código de controle. Quanto a consideração da temporização nos modelos, [17] e [24], utilizam AT tratando o tempo de forma quantitativa, abordagem utilizada no presente trabalho. Enquanto [22], trata a temporização de forma qualitativa. A abordagem na forma de um ciclo que inicia-se com a modelagem do sistema e termina com a sua validação, é semelhante à discutida em [14]. O desenvolvimento ou uso de ferramentas de software para auxiliar nas etapas do processo, que na maioria dos trabalhos mencionados encontra-se apenas na etapa de verificação, no presente trabalho é utilizada também para geração dos modelos. A geração automática dos modelos é realizada através do uso de ferramentas de tradução de formatos, semelhantes as utilizadas em [24]. Por fim, a integração das ferramentas de desenvolvimento de código, de geração de modelos e de testes é proposta, de forma semelhante ao discutido em [2], porém, o foco do presente trabalho não é a geração automática do código fonte. O objetivo é auxiliar o processo de desenvolvimento e melhorar a confiança no funcionamento de SIS ao proporcionar que testes de conformidade entre modelos formais da especificação e da implementação de tais sistemas possam ser realizados de forma automática e transparente para o desenvolvedor. Para que a aplicação do método seja possível os sistemas precisam atender os seguintes requisitos: serem descritos em linguagem FBD para executar em um CLP; e utilizar apenas variáveis Booleanas e os símbolos suportados pela ferramenta de tradução. O primeiro requisito deve-se ao fato de que para realizar os testes um modelo do ambiente é utilizado para geração dos casos de teste e para determinar a comunicação com o modelo do SIS. Em nosso caso o ambiente reproduz o modo como um CLP executa o código do SIS e determina como as variáveis de entrada do modelo devem ser atualizadas. O segundo requisito diz respeito ao subconjunto de símbolos escolhidos para serem tratados nessa versão da ferramenta de tradução.

Capítulo 4

Fundamentação Teórica

Neste Capítulo, são apresentadas as definições de Sistemas dirigidos a Eventos Discretos (SED), autômatos de estados finitos, autômatos temporizados (AT), da semântica de um AT e de uma rede de AT.

4.1 Sistemas dirigidos a Eventos Discretos (SED)

Um sistema físico pode ser formalmente modelado de diversas formas: através de equações diferenciais, equações algébricas, linguagens formais, autômatos, redes de Petri, entre outros. A escolha do formalismo a ser usado na modelagem depende do tipo de sistema que deseja-se modelar, se é continuo ou discreto, linear ou não-linear, estático ou dinâmico, etc. No presente trabalho iremos lidar com sistemas dirigidos a eventos discretos. Um sistema é classificado como dirigido a eventos discretos quando o seu espaço de estados é naturalmente descrito por um conjunto discreto de estados e transições entre estados são observadas apenas em instantes discretos do tempo, de forma que é possível associar estas transições a um conjunto de eventos [6]. Na literatura foram propostos vários métodos para para descrever adequadamente o comportamento de SED e permitir o uso de técnicas analíticas para propósito de projeto, controle e análise de performance dessa classe de sistemas, entre eles estão as representações através de linguagens, autômatos e redes de Petri. No presente trabalho será utilizada a modelagem por autômatos temporizados.

4.2 Autômato como Modelo para SED

Um autômato pode ser visualizado como uma máquina de estados que comuta de um estado para outro através da ação de uma transição instantânea [4]. Um autômato é representado graficamente por um grafo dirigido, no qual os nós representam os estados e os arcos etiquetados representam as transições entre os estados. No grafo, indicações

especiais devem ser usadas para identificar o estado inicial e os estados marcados do autômato. A existência de uma representação gráfica é um dos benefícios do formalismo baseado em autômatos.

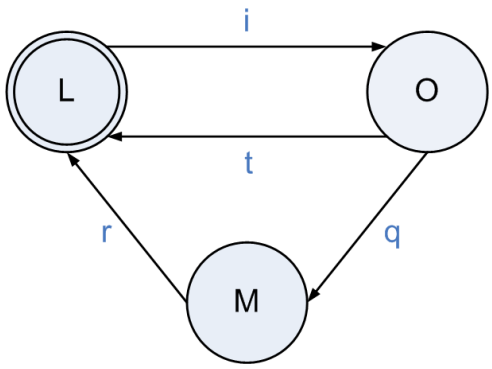

Figura 4.1: Possível representação gráfica para um autômato.

Uma possível representação gráfica para um autômato está ilustrada na Figura 4.1, o autômato representado possui três estados: livre (L) o estado inicial, ocupado (O) e em manutenção (M). As etiquetas associadas com cada arco representam os eventos: início do serviço (i), término do serviço (t), quebra (q) e reparo (r), portanto esse autômato pode ser usado para representar o ciclo de vida de uma máquina-ferramenta, por exemplo. A definição formal de um autômato é dada a seguir:

Definição 1 ***Um Autômato de Estados Finitos****, denotado por A, é uma tupla*

$$A = (Q, E, T, q_0) \tag{4.1}$$

Em que:

- Q *é o conjunto finito de estados;*
- q_0 *é o estado inicial;*
- E *é o conjunto de eventos ou etiquetas associados às transições;*
- $T \subseteq Q \times E \times Q$ *é o conjunto de transições entre os estados.*

No objetivo de modelar sistemas físicos com uma melhor aproximação do comportamento apresentado na realidade, é conveniente permitir que autômatos manipulem variáveis de estado. Um autômato interage com essas variáveis de dois modos distintos, o primeiro é através da operação de atribuição de valores a uma ou mais variáveis, realizada no momento do disparo de uma transição. O segundo é através da criação de guardas nas transições entre as localidades. De agora em diante, os nós do grafo dirigido

que representa graficamente um autômato serão denominados de localidades, uma vez que um estado será composto pela localidade ocupada pelo autômato juntamente com o valor das variáveis de estado. Guardas são condições Boolenas envolvendo as variáveis de estados, de forma que a transição só pode ocorrer se o valor atual das variáveis satisfizer a guarda.

Para ilustrar o uso de variáveis de estado e de guardas considere o autômato da Figura 4.2, no qual a etiqueta (m) representa o evento de manutenção preventiva e a variável (TMP) representa o tempo programado para manutenção preventiva da máquina-ferramenta, os demais estados e etiquetas possuem o mesmo significado do exemplo anterior. Observe que agora dois eventos podem levar à localidade M, o evento de quebra q, que ao ocorrer dispara a transição de O para M e o evento m, que só dispara a transição se o tempo para manutenção preventiva foi atingido ou ultrapassado, conforme estabelecido na guarda presente no arco que leva da localidade L para M.

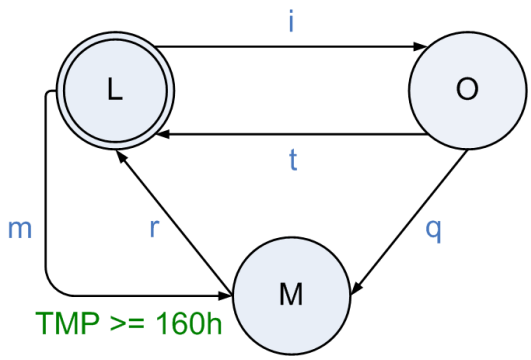

Figura 4.2: Exemplo do uso de guardas nas transições dos autômatos.

No intuito de facilitar a modelagem de um sistema costuma-se dividi-lo em partes menores e modelar cada uma isoladamente, após isso, utiliza-se algum meio para compor as partes de forma a modelar o comportamento do sistema como um todo. No caso de modelos de autômatos é utilizada a operação de composição denominada de produto síncrono, cuja definição formal é apresentada a seguir:

Definição 2 ***Produto Síncrono****, considere um conjunto de n autômatos,* $\forall i \in \{1, \ldots, n\}$ $A_i = \langle Q_i, E_i, T_i, q_0 \rangle$ *e o evento '–' para denotar a ação vazia ou "faz nada" para qualquer autômato que permanecer inativo durante uma transição de estado global do conjunto de componentes. O produto cartesiano* $A_i \times \ldots \times A_n$ *do conjunto de autômatos é um simples autômato* $A = \langle Q, E, T, q_0 \rangle$ *em que:*

- $Q = Q_i \times \ldots \times Q_n$;
- $q_0 = (q_{0,1}, \ldots, q_{0,n})$;

- $E = \prod_{1 \leq i \leq n}(E_i \cup \{-\})$;
- $T = \left\{ \begin{array}{l} ((q_1, \ldots, q_n), (e_1, \ldots, e_n), (q'_1, \ldots, q'_n)) \mid (e_1, \ldots, e_n) \in Sync \\ \text{e } \forall i, e_i = \text{'}-\text{'} \text{ e } q'_i = q_i \text{ ou } e_i \neq \text{'}-\text{'} \text{ e } (q_i, e_i, q'_i) \in T_i \end{array} \right\}$;
- $Sync \subseteq E$ *é um conjunto das transições permitidas em* A.

O conjunto *Sync* é usado para determinar as transições que serão permitidas no autômato resultante da operação de produto síncrono, portanto, sua definição determinará as localidades alcançáveis desse autômato. Outros formas de sincronização entre autômatos são possíveis e baseiam-se em permitir a comunicação entre os autômatos a serem compostos, dentre elas podemos destacar as sincronizações por troca de mensagens e por compartilhamento de variáveis de estado.

A sincronização por troca de mensagens consiste em distinguir entre as etiquetas das transições, ou eventos, aqueles associados com o envio de uma mensagem m, denotado por $m!$ e os associados com o recebimento dessa mensagem, denotados por $m?$. A transição etiquetada com $m?$, receptora da mensagem, será executada simultaneamente a transição etiquetada com $m!$, emissora da mensagem.

A sincronização por compartilhamento de variáveis de estados, como o próprio nome sugere, consiste em permitir que uma variável possa ser acessada por mais de um autômato da composição. Dessa forma através da utilização de guardas complementares envolvendo o valor dessa variável é possível realizar intertravamentos entre os autômatos.

O autômato de estados finitos permite modelar seqüências de ações, porém, nenhuma informação quantitativa de temporização entre as ações pode ser considerada. Para incluir quantitativamente a passagem do tempo em modelos usando autômatos e permitir que sistemas que possuam restrições de tempo real sejam modelados utilizando esse formalismo utiliza-se Autômatos Temporizados(AT) [1].

Um AT é construído a partir de dois elementos fundamentais, um autômato de estados finitos e um conjunto de relógios ou cronômetros usados para especificar quatitativamente as restrições de tempo. Relógios são variáveis que assumem valores no conjunto dos números reais não negativos. No estado inicial os valores de todos os relógios são iguais a zero, ao iniciar-se a execução do modelo todos eles evoluem de forma crescente a mesma velocidade e em sincronia com o tempo. Nos AT as transições estão associadas a três itens:

- Uma guarda, também chamada de condição de disparo, a qual pode envolver os valores dos relógios, isto é, restrições envolvendo quantitativamente o tempo.
- Uma etiqueta, incluindo aquelas relacionadas a troca de mensagens entre autômatos.

- E uma atribuição, que no caso de uma transição temporizada, poderá servir para reiniciar um ou mais relógios no momento do disparo da transição.

Como é permitido a atribuição do valor zero aos relógio no momento de disparo de uma transição, o valor atual de um relógio marca sempre o intervalo de tempo desde o último momento no qual ele foi reiniciado. Logo o sistema funciona como se estivesse equipado com um relógio global que mede o tempo e com uma série de cronômetros independentes, sincronizados com o relógio global, medindo o intervalo de tempo a partir do momento em que cada um foi reiniciado.

Um coleção de AT, quando submetida a operação de composição, resulta uma rede de AT. Uma execução de uma rede de AT é uma seqüência de configurações. Uma configuração, ou estado global, consiste de todas as localidades ocupadas por cada autômato da rede mais os valores dos relógios e é denotada pelo par (q, v), no qual q representa as localidades e v os valores dos relógios. Numa rede de AT uma configuração pode mudar de duas maneiras:

- Pela ocorrência de um atraso, denotado por d, que fará com que os valores de todos os relógios sejam incrementados de d;

- Pelo disparo de uma transição em qualquer dos autômatos da rede, ou uma sincronização de várias transições, ocasionando a mudança de localidade para os autômatos afetados pelas transições. Nesse caso, os valores dos relógios evoluem normalmente de forma independente, ou seja, podem ser reiniciados ou permanecerem inalterados.

À definição básica de AT foram incorporadas algumas extensões para facilitar a modelagem e o uso de verificação automática para os modelos. Entre as extensões já mencionadas, como a adição de variáveis discretas, existem duas específicas dos AT, a saber: invariantes e transições urgentes. Invariantes estão relacionadas a propriedade de vivacidade dos sistemas, isto é, existem para assegurar que certas transições irão ocorrer evitando uma espera indefinida em uma certa localidade de um autômato. Para tanto, são associadas a algumas das localidades dos autômatos condições envolvendo os valores dos relógios, chamadas invariantes, que devem ser avaliadas como verdadeiras enquanto o autômato permanecer naquela localidade. Por exemplo, considere x um relógio, a condição $x \leq 5$ indica que a localidade associada a esse invariante só pode ser ocupada enquanto o valor de x for inferior ou igual a 5, quando x assumir um valor fora do especificado no invariante obrigatoriamente será disparada uma das transições que saem da localidade fazendo com que o sistema evolua. Transições urgentes são utilizadas nas situações onde não se pode permitir a ocorrência de nenhum atraso, ou seja, ao se atingir uma localidade que possui uma transição de saída do tipo urgente, não poderá haver incremento dos relógios antes do disparo dessa transição.

No presente trabalho a definição de AT utilizada corresponde a implementação de AT da ferramenta Uppaal, incluindo todas as extensões e facilidades fornecidas por essa ferramenta. Apresentamos aqui a definição formal de um autômato temporizado e as semânticas para um autômato e para uma rede autômatos, conforme apresentado em Behrmann *et al* [3]. A notação utilizada é a seguinte: C é um conjunto de relógios e $B(C)$ é o conjunto de conjunções de condições simples da forma $x \bowtie c$ ou $x - y \bowtie c$, onde $x, y \in C$, $c \in \mathbb{N}$ e $\bowtie \in \{<, \leq, \geq, >\}$.

Definição 3 ***Autômato Temporizado (AT)****, denotado por A_T, é uma tupla*

$$A_T = (L, l_0, C, A, E, I) \tag{4.2}$$

Em que:

- *L é o conjunto de localidades;*
- *l_0 é a localidade inicial;*
- *C é o conjunto de relógios;*
- *A é o conjunto de ações, co-ações e ações internas (τ);*
- *$E \subseteq L \times A \times B(C) \times 2^C \times L$ é o conjunto de arcos entre as localidades com uma ação (a), uma guarda (g) e um conjunto de relógios para serem zerados (r);*
- *$I : L \rightarrow B(C)$ é uma função que atribui invariantes para as localidades.*

Para definir a semântica de um autômato temporizado as seguintes considerações são necessárias. Uma avaliação de relógio é uma função $u : C \rightarrow \mathbb{R}_+$ que mapeia o conjunto dos relógios no conjunto dos reais não negativos. Seja $\mathbb{R}^C$ o conjunto de todas as avaliações de relógios, $u_0(x) = 0 \; \forall \; x \in C$ e $u \in I(l)$ utilizado para denotar u que satisfaz $I(l)$.

Definição 4 ***Semântica de um AT****: Seja (L, l_0, C, A, E, I) um autômato temporizado. Sua semântica é definida como um (LTS do inglês, Labelled Transition System) $\langle S, s_0, \rightarrow \rangle$, onde $S \subseteq L \times \mathbb{R}^C$ é o conjunto de estados, $s_0 = (l_0, u_0)$ é o estado inicial e $\rightarrow \subseteq S \times \{\mathbb{R}_+ \cup A\} \times S$ é a relação de transição tal que:*

- $(l, u) \xrightarrow{d} (l, u + d)$ *se* $\forall \; d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$, *e*

- $(l, u) \xrightarrow{a} (l', u')$ *se existe* $e = (l, a, g, r, l') \in E$, $u \in g$, $u' = [r \mapsto 0]u$ *e* $u' \in I(l)$,

onde para $d \in \mathbb{R}_+$, $u + d$ mapeia cada relógio x em C para o valor $u(x) + d$, e $[r \mapsto 0]u$ denota as avaliações de relógios que mapeiam cada relógio em r para 0 e concorda com u para C/r.

Uma rede de AT consiste de n autômatos $A_{Ti} = (L_i, l_i^0, C, A, E_i, I_i)$, $1 \leq i \leq n$, que compartilham o mesmo conjunto de relógios e ações. Um vetor de localidades é um vetor $\bar{l} = (l_1, \ldots, l_n)$. A composição das funções invariantes em uma função comum do vetor de localidades é definida como $I(l) = \wedge_i I(l_i)_i$. A expressão $\bar{l}[l_i'/l_i]$ denota o vetor onde o i-ésimo elemento l_i de $\bar{l}$ é substituído por l_i'.

Definição 5 ***Semântica de uma rede de AT****: Seja $A_{Ti} = (L_i, l_i^0, C, A, E_i, I_i)$ uma rede de n AT e $\bar{l}_0 = (l_1^0, \ldots, l_n^0)$ o vetor de localidades inicial. A semântica é definida como um LTS $\langle S, s_0, \rightarrow \rangle$, onde $S \subseteq L \times (L_1 \times \cdots \times L_n) \times \mathbb{R}^C$ é o conjunto de estados, $s_0 = (\bar{l}_0, u_0)$ é o estado inicial e $\rightarrow \subseteq S \times S$ é a relação de transição tal que:*

- $(\bar{l}, u) \rightarrow (\bar{l}, u + d)$ *se* $\forall\, d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(\bar{l})$.

- $(\bar{l}, u) \rightarrow (\bar{l}[l_i'/l_i], u')$ *se existe* $l_i \xrightarrow{\tau g r} l_{i'}$, $u \in g$, $u' = [r \mapsto 0]u$ *e* $u' \in I(\bar{l})$.

- $(\bar{l}, u) \rightarrow (\bar{l}[l_j'/l_j, l_i'/l_i], u')$ *se existe* $l_i \xrightarrow{c?g_i r_i} l_{i'}$ *e* $l_j \xrightarrow{c!g_j r_j} l_{j'}$, $u \in (g_i \wedge g_j)$, $u' = [r_i \cup r_j \mapsto 0]u$ *e* $u' \in I(\bar{l})$.

Capítulo 5

A Modelagem

Neste capítulo todo processo de desenvolvimento dos modelos é apresentado em detalhes. Primeiro o objeto da modelagem é identificado; a seguir são discutidas a concepção e a organização do modelo; na seqüencia são descritos os esqueletos dos autômatos, suas funções e o modo como interagem durante a execução do modelo; e por fim é discutido o modo como as informações da especificação e do código do SIS são extraídas e combinadas com os esqueletos dos autômatos, de forma automática, para formar os modelos.

5.1 O Objeto da Modelagem

A modelagem é realizada a partir da descrição da especificação e do código FBD. A especificação e representada através de diagramas construídos utilizando o padrão ISA 5.2. Nesse padrão são definidos símbolos para representação de células de memória, temporizadores e para operações Booleanas. Em um diagrama ISA 5.2 o fluxo de informação entre os elementos se dá apenas através das linhas que conectam os elementos. O sentido do fluxo é da esquerda para a direita e/ou de cima para baixo e qualquer alteração no sentido de fluxo convencional deve ser explicitada através de setas. Um exemplo típico de um diagrama ISA 5.2 é ilustrado na Figura 5.1. Nesse exemplo, o diagrama representa o funcionamento de um alarme composto por dois sensores e uma chave para ligá-lo e desligá-lo. Os sensores ao detectarem algum distúrbio, por cinco segundos consecutivos, ativam o alarme se o mesmo estiver ligado.

Na Figura 5.1 juntamente com os elementos de lógica está representado um dos três temporizadores básicos definidos no padrão, o DI. O funcionamento desses temporizadores é descrito abaixo.

- *Delay Initialization of output* (DI): A existência contínua do sinal de entrada por um tempo t, faz com que o sinal de saída exista. O sinal de saída deixa de existir quando o sinal de entrada deixar de existir;

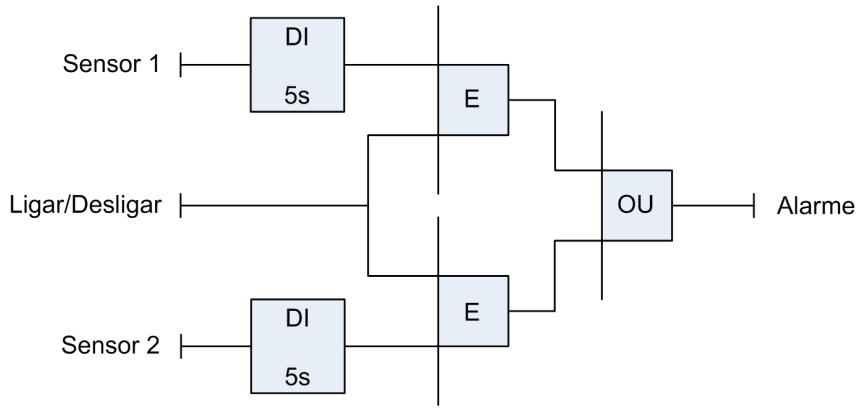

Figura 5.1: Exemplo típico de um diagrama ISA.

- *Delay Termination of output* (DT): A existência do sinal de entrada faz com que o sinal de saída exista imediatamente. O sinal de saída deixará de existir quando o sinal de entrada deixar de existir continuamente por um tempo t;
- *Pulse Output* (PO): A existência do sinal de entrada faz com que o sinal de saída exista imediatamente. Independentemente do estado subseqüente do sinal de entrada, o sinal de saída existirá por um tempo fixo t.

O código deve ser descrito utilizando a linguagem FBD definida no padrão IEC 61131-3. A linguagem FBD usa blocos funcionais para representar as ações que devem ser realizadas. Esses blocos podem ser escolhidos dentro de um conjunto pré-definido pelo fabricante do CLP ou construídos pelo usuário de acordo com suas necessidades. Um bloco funcional FB (do inglês, Function Block), indicado na Figura 5.2, possui um conjunto de parâmetros de entrada e saída, variáveis internas e um algoritmo. Quando o bloco é solicitado o algoritmo processa os parâmetros de entrada, considerando os valores das variáveis internas, para determinar os parâmetros de saída.

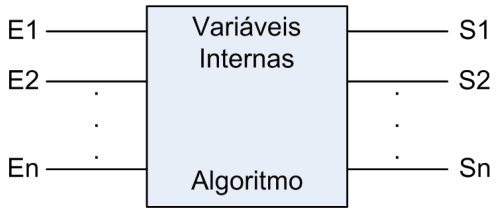

Figura 5.2: Representação de um bloco funcional genérico e seus componentes.

Um diagrama FBD pode ser visto de maneira análoga a um diagrama elétrico, ou seja, as conexões determinam o caminho dos sinais entre os blocos que formam o diagrama. O padrão define que o fluxo dos sinais em um diagrama FBD ocorre da esquerda para a direita, das saídas de funções ou FBs para as entradas de outras funções ou FBs. A negação dos sinais é feita de forma semelhante aos diagramas lógicos, pela presença de

um círculo na entrada ou saída dos FBs, ou pelo uso do bloco NOT. Um exemplo típico de um diagrama FBD é mostrado na Figura 5.3. O diagrama do exemplo reproduz o comportamento do alarme especificado na Figura 5.1, explicado anteriormente. Os símbolos utilizados no FBD para representar os temporizadores são: TON, TOFF e TP, os quais correspondem respectivamente aos elementos DI, DT e PO do padrão ISA 5.2.

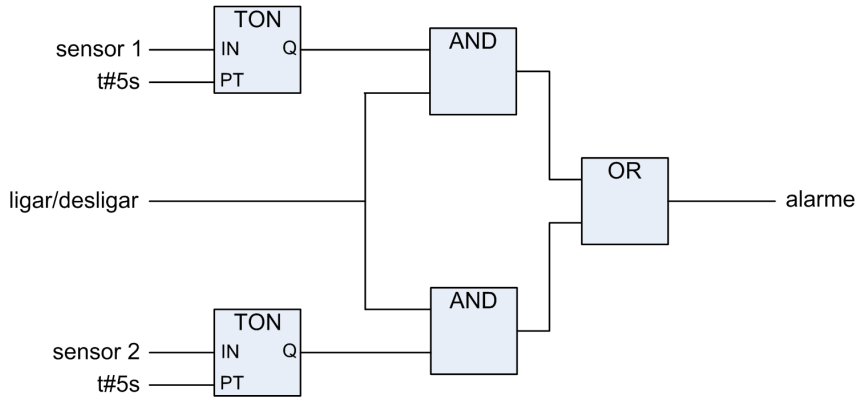

Figura 5.3: Diagrama FBD correspondente ao alarme especificado na Figura 5.1

5.2 A Concepção do Modelo

O objetivo final da modelagem dos diagramas da especificação e do código do SIS é a realização de testes de conformidade e para testar os modelos é preciso saber como interagir com eles. No caso de um SIS o agente com o qual ele interage é o ambiente industrial, portanto modelar o comportamento do ambiente torna-se necessário.

O modelo é constituído por uma rede de autômatos temporizados. As estruturas dos autômatos foram previamente determinadas para reproduzir o comportamento dos elementos do padrão ISA 5.2, do código FBD e do ambiente. Essas estruturas são representadas pelos arquétipos (*templates*) dos autômatos. O modelo é particionado em um modelo para o ambiente e um modelo para o SIS. O particionamento é realizado através de regras de escopo para cada parte do modelo e a sincronização entre os modelos do ambiente e do sistema é realizada através de troca de mensagens, conforme ilustrado na Figura 5.4. Informações sobre as primeiras versões dos modelos para ISA 5.2 podem ser encontradas em [10] e para FBD em [8] e [9].

5.3 O Modelo do Ambiente

Essa parte do modelo deve ser capaz de produzir os valores das variáveis monitoradas pelo SIS e reproduzir o ambiente de execução do programa. A atualização dos valores

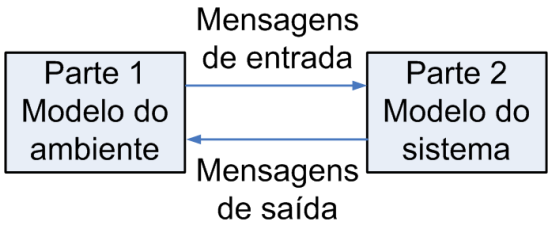

Figura 5.4: Diagrama de blocos ilustrando o particionamento do modelo.

das variáveis de entrada do programa deve ser realizada de forma consistente com aquela observada no ambiente físico do processo a ser monitorado. No presente trabalho são consideradas apenas variáveis Booleanas, portanto, o ambiente é responsável por gerar combinações com os valores das variáveis de entrada de forma não-determinística ao longo do tempo.

Como o código FBD irá executar em um CLP é importante que o modelo do ambiente reproduza o ciclo de varredura executado por esses dispositivos. Para executar o código os CLPs realizam três operações de forma cíclica em um processo denominado ciclo de varredura, ilustrado na Figura 5.5, são elas: a leitura dos valores dos sinais de entrada oriundos do ambiente para os registradores de entrada do CLP, a execução do código para processar os valores das variáveis de entrada; e a escrita dos valores de saída produzidos na execução corrente nos registradores de saída do CLP e conseqüentemente para o ambiente.

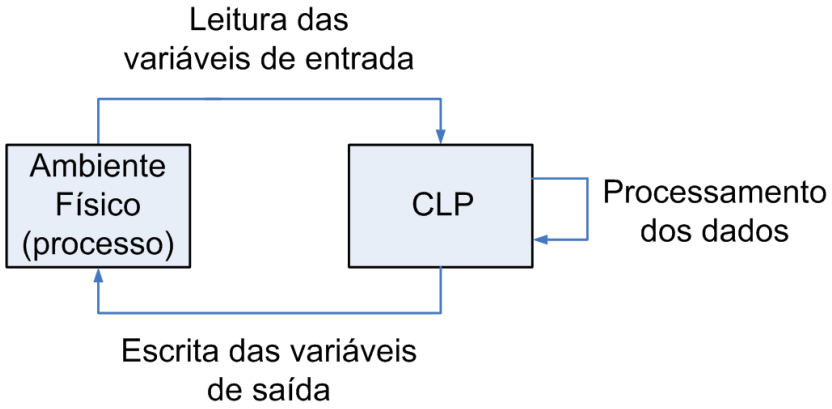

Figura 5.5: Diagrama de blocos do ciclo de varredura.

A Figura 5.6 ilustra a interação entre os autômatos do modelo do ambiente e do modelo do sistema, as setas cheias representam operações de troca de mensagens entre os autômatos e as setas finas operações de leitura e escrita de variáveis do modelo. O modelo do ambiente realiza as seguintes tarefas:

- Atualização das variáveis de entrada e sinalização do início do processamento para o modelo do sistema;

- Controle da execução do modelo do sistema, através do envio de uma mensagem de sincronização e determinação do término do processamento das variáveis de entrada;
- Recepção das mensagens informando os valores das variáveis de saída produzidos na execução corrente do modelo do sistema.

Para realizar essas tarefas são utilizados quatro autômatos: um representando o ciclo de varredura, outro que controla a geração dos valores das variáveis de entrada, um terceiro que representa as variáveis de entrada e o quarto para receber as mensagens informando os valores das variáveis de saída do sistema.

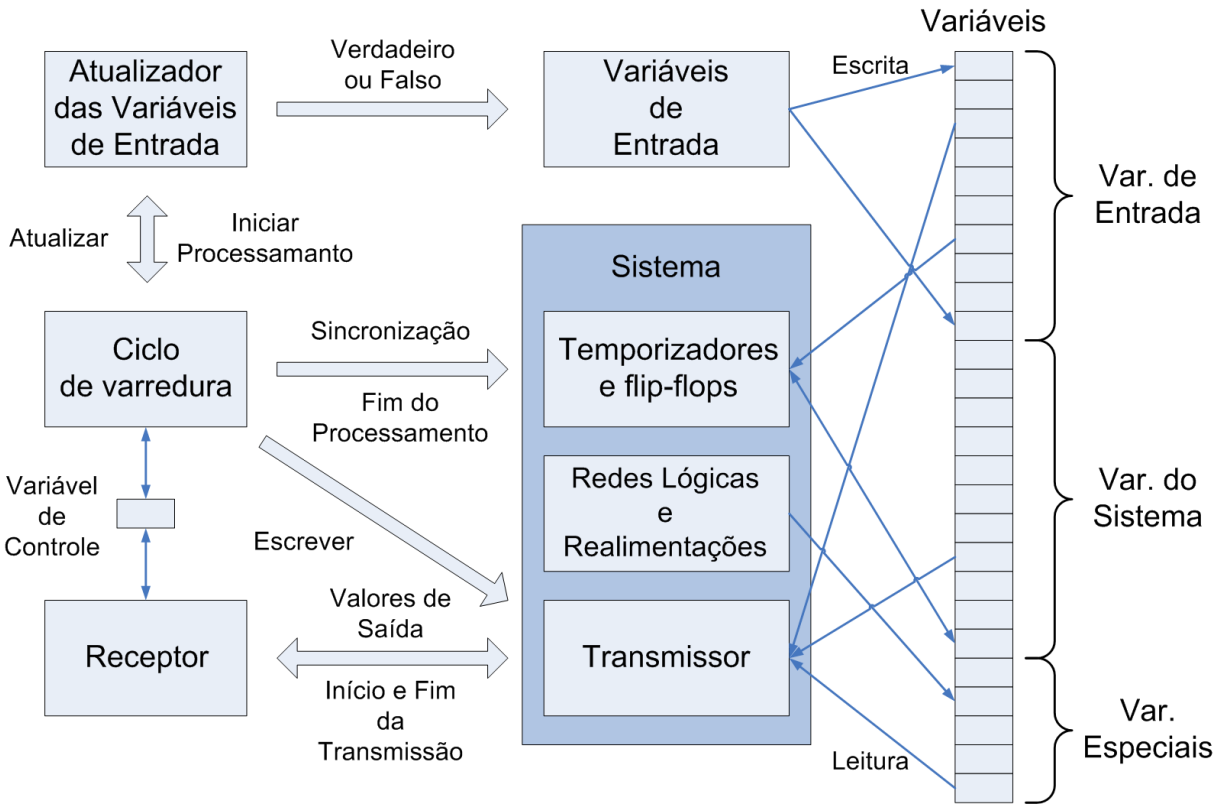

Figura 5.6: Diagrama de blocos ilustrando a interação entre os autômatos do modelo.

5.4 O Modelo do Sistema

O modelo do sistema é formado pelos autômatos dos elementos dos padrões ISA 5.2 e FBD, ou seja, células de memória e temporizadores. Os elementos de lógica Booleana não são modelados como autômatos, mas por expressões Booleanas que formarão as guardas dos demais autômatos dessa parte modelo.

Esse modelo deve reproduzir o comportamento descrito nos diagramas ISA 5.2 e FBD, os quais contêm as regras para o processamento dos valores das variáveis do processo

industrial que são monitoradas. O comportamento do modelo do sistema é descrito a seguir e sua interação com o modelo do sistema está ilustrada na Figura 5.6. Uma vez determinada a combinação de entrada, por parte do modelo do ambiente, o modelo do sistema deve processá-la para produzir os valores das variáveis de saída. O processamento das variáveis de entrada é realizado de forma iterativa e cada iteração do modelo do sistema é iniciada pelo recebimento de uma mensagem de sincronização vinda do modelo do ambiente, mas precisamente do autômato do ciclo de varredura.

Para determinar o momento de parada das iterações da rede de autômatos que compõe o modelo do sistema é usada uma função para computar a convergência do modelo para o resultado final daquele ciclo. Essa função, computada pelo modelo do ambiente, compara os valores das variáveis do modelo antes e depois de cada iteração até que não ocorram mais mudanças nesses valores de uma iteração para outra, desse momento em diante cessa o envio de mensagens de sincronização para o sistema e o ambiente se prepara para receber as mensagens informando os valores das variáveis de saída. O modelo do ambiente também controla o tempo de cada ciclo que é fixo e determinado previamente no momento da programação do SIS. O ciclo reinicia com a determinação de uma nova combinação de entrada por parte do modelo do ambiente e com o envio das mensagens de sincronização para um novo processamento.

Todos os valores das variáveis do modelo encontram-se num *array* que é divido em três regiões: uma para as variáveis de entrada, outra para as variáveis dos elementos do sistema e a última para variáveis especiais. As variáveis especiais são utilizadas em dois casos: no tratamento das realimentações no qual é necessário criar duas variáveis para um mesmo sinal, uma fixa que guarda o valor do sinal produzido no ciclo anterior, V_p, a qual fica na região das variáveis de entrada e outra que muda de valor durante o ciclo de processamento corrente V_r que fica na região de variáveis especiais. Ao final do ciclo corrente o valor de V_r é copiado para a variável V_p para ser usado no ciclo posterior. O segundo caso trata de situação em que uma variável de saída corresponde a saída de uma porta lógica. Nesse caso, após o final da execução da rede de autômatos quando os valores das saídas de todos os elementos do sistema, temporizadores e flip-flops, estão estabilizados os valores dessas variáveis de saída são computados e guardados na região de variáveis especiais. Os autômato do ambiente e do sistema lêem e/ou escrevem no *array* de variáveis.

5.5 Arquétipos para Autômatos

Para exemplificar os arquétipos utilizados para representar a estrutura dos autômatos, utilizaremos o arquétipo do temporizador DI ou TON, ilustrado na Figura 5.7. As lo-

calidades identificadas com um círculo interno são as localidades iniciais e as localidades com uma estrela são localidades *committed*. Esse tipo de localidade serve para modelar operações atômicas, enquanto pelo menos um autômato da rede ocupar uma localidade *committed* o sistema está em um estado *committed*. Nesse estado não é permitida a evolução dos relógios e a próxima transição a ser disparada deve sair de uma localidade *committed*.

Os arcos que conectam as localidades dos arquétipos são constituídos de três expressões: a primeira está relacionada a sincronização daquele arco com um ou mais arcos de outros arquétipos, por exemplo *sinc?*; a segunda determina que condições devem ser satisfeitas para que aquele arco esteja habilitado e possa ser disparado, essas condições são chamadas de guardas, por exemplo $guarda(id) == 1$; e a terceira corresponde a alguma ação que deve ser realizada no momento do disparo do arco, por exemplo a atribuição $var[id] = 1$.

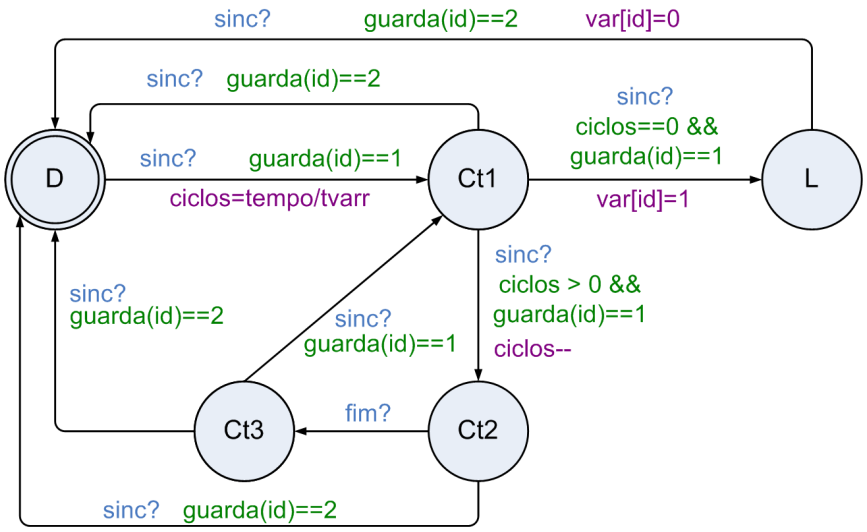

Figura 5.7: Arquétipo para o autômato do elemento DI.

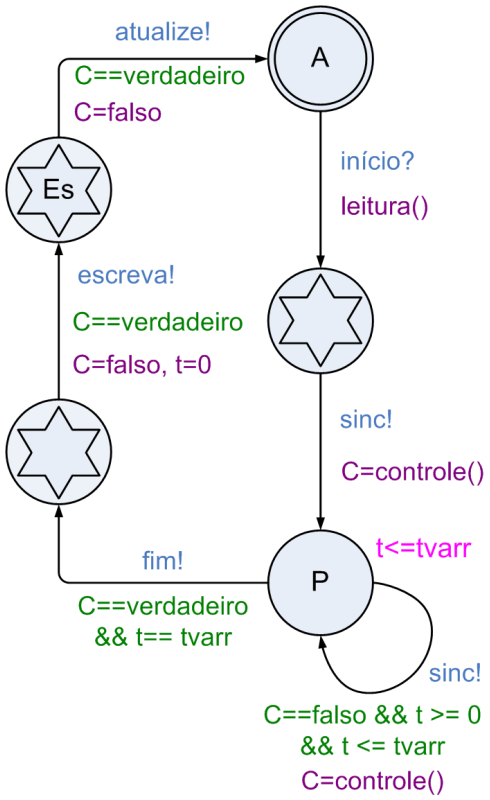

Figura 5.8: Arquétipo para o autômato do ciclo de varredura.

Na modelagem realizada *sinc!* é uma mensagem de sincronização enviada a todos os elementos do modelo pelo autômato que modela o ciclo de varredura do CLP, mostrado na Figura 5.8. O autômato do ciclo de varredura, conforme mencionado anteriormente, controla a seqüência de tarefas realizadas durante a execução do modelo. Todo elemento (autômato) do modelo do sistema possui um identificador, a função *guarda* recebe como parâmetro esse identificador e avalia as expressões que irão determinar o estado dos sinais presentes na entrada desse elemento. O valor retornado por essa função é testado no autômato, juntamente com as demais restrições presentes em suas guardas, para determinar que arco está habilitado e se ele será ou não disparado. No momento do disparo pode ser realizada alguma ação, como a atribuição de um novo valor a uma variável. No modelo, *var* é um array de variáveis Booleanas que guarda os valores das variáveis associadas aos elementos do modelo.

O comportamento do autômato do temporizador DI é o seguinte: considerando o autômato do ciclo de varredura na localidade processando (P), ou seja, gerando a mensagem de sincronização *sinc!* e o DI na localidade inicial, desligado (D), se a função *guarda* retorna o valor 1, que representa o valor *verdadeiro* na entrada do elemento DI, o único arco de saída da localidade inicial estará habilitado e será disparado levando à localidade Ct1. No momento do disparo, é atribuído à variável *ciclos* o número inteiro de ciclos de varredura que deverão transcorrer para que a saída do temporizador apresente o valor

verdadeiro. A partir desse ponto, a cada ciclo de varredura executado a variável *ciclos* será decrementada e as localidades Ct1, Ct2 e Ct3, serão ocupadas seqüencialmente durante o ciclo de contagem. Em cada uma das localidades que formam o ciclo de contagem existe um arco que permite o retorno ao estado inicial se o valor retornado pela função *guarda* for 2. O valor de retorno 2 representa um valor *falso* na entrada do elemento DI. Esses arcos reiniciam o temporizador no caso do tempo programado para a existência contínua do sinal de entrada não ter sido atingido. Se o sinal de entrada existir continuamente pelo tempo programado, condição testada na guarda do arco que liga as localidades Ct1 e ligado (L), o arco é disparado. No momento do disparo desse arco é atribuído à variável que representa a saída do elemento DI o valor *verdadeiro*. Entre as localidades L e D, existe um arco que reinicia o temporizador quando o valor na entrada do elemento voltar a ser *falso*.

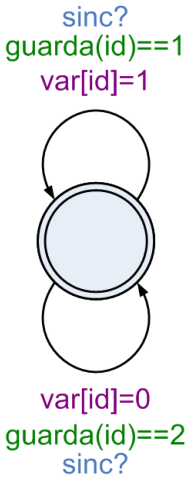

Figura 5.9: Arquétipo para o autômato do *flip-flop* SR.

Juntamente com os temporizadores as células de memória completam o modelo do sistema, a estrutura do autômato utilizado para representar esses elementos é ilustrada na Figura 5.9. Seu funcionamento é o seguinte: de acordo com o valor retornado pela função *guarda* a variável que representa a célula de memória recebe o valor verdadeiro, falso ou mantém seu valor inalterado, valor de retorno igual a zero. Nas células de memória normalmente uma entrada tem prioridade com relação a outra e a função *guarda* é implementada para tratar essa situação.

Para reiniciar um novo ciclo o autômato do ciclo de varredura solicita a atualização dos valores das variáveis de entrada, a qual é realizada pelos autômatos cujas estruturas estão representadas nas Figuras 5.10 e 5.11. O autômato que controla a atualização envia para cada autômato que representa uma variável de entrada uma mensagem informando o valor que deve ser atribuído a esta variável, ao término da atualização é enviada uma mensagem ao autômato do ciclo de varredura informando que o processamento pode começar. Toda

etapa de atualização ocorre sem que haja a evolução do relógio do modelo.

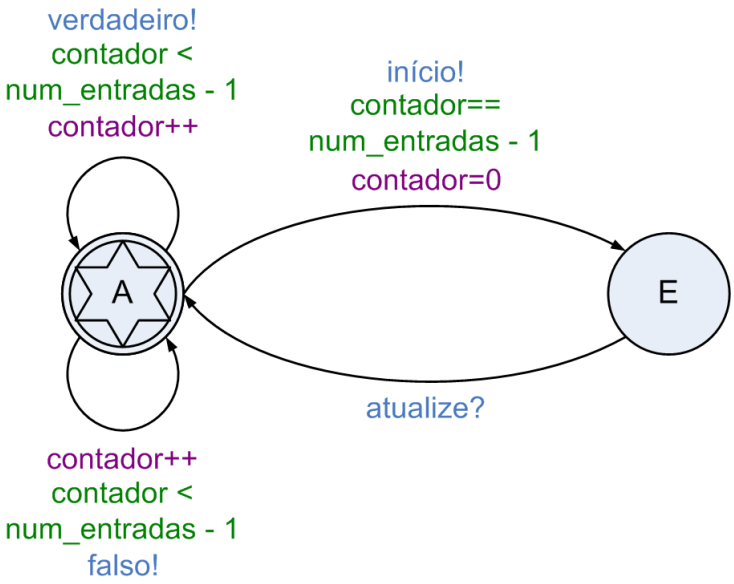

Figura 5.10: Arquétipo para o autômato que atualiza as variáveis de entrada.

A primeira mensagem enviada pelo atualizador é recebida pela primeira variável de entrada a qual escreve em *var* o valor recebido e incrementa a variável identificador, desta forma a segunda mensagem chega apenas para a variável seguinte e o processo se repete.

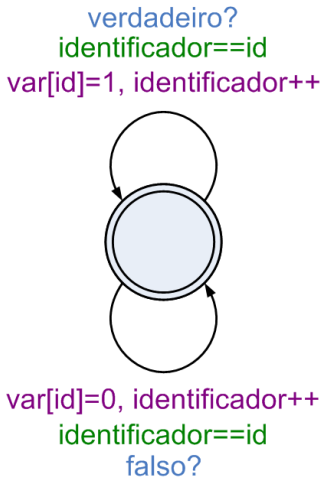

Figura 5.11: Arquétipo para o autômato das variáveis de entrada.

5.6 A Geração dos Modelos

Os modelos são formados pela combinação de informação estrutural, arquétipos dos autômatos e código das funções de controle do modelo, com a informação da lógica de segurança retirada dos diagramas ISA 5.2 e/ou do código FBD. A parte estrutural carrega

apenas informações gerais sobre o modelo a sua composição com a parte lógica é que determina o comportamento final da rede de autômatos do modelo. Conforme ilustrado na Figura 5.12, primeiramente é realizada uma busca no documento fornecido como entrada para a identificação das conexões entre os elementos. Com a informação obtida na busca são formadas as expressões Booleanas que formam as guardas dos autômatos e é formatada a informação para completar o esqueleto de código do modelo. Por fim, é realizada a composição da parte estrutural dos autômatos com a parte lógica gerada a partir das informações dos diagramas para completar o modelo.

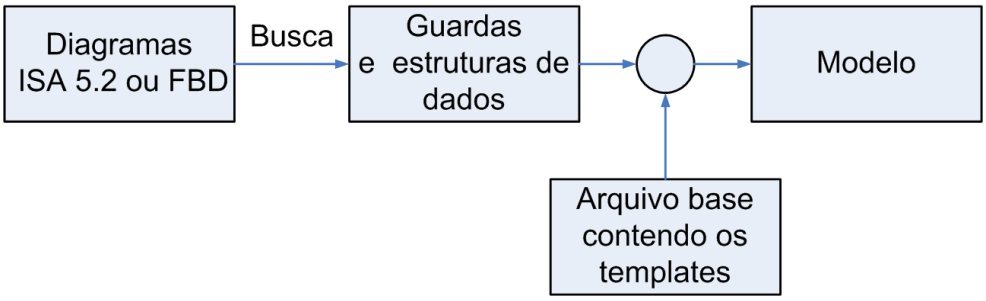

Figura 5.12: Diagrama de blocos ilustrando as etapas de geração do modelo.

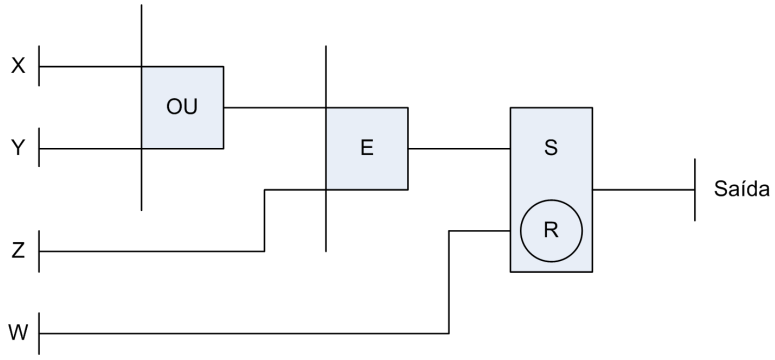

Figura 5.13: Diagrama para explicar o funcionamento do algoritmo.

Para a geração do modelo é utilizado um algoritmo de busca recursivo, alguns métodos para tratar os elementos segundo o seu tipo e para lidar com situações como inversão de sinais nas entradas dos elementos, realimentação de sinais, entre outras. A busca funciona da seguinte forma: para cada elemento do sistema, temporizadores e *flip-flops*, são observados os elementos conectados as suas entradas, caso o elemento encontrada seja do tipo lógico a busca continua para as entradas desse elemento. Caso seja uma variável de entrada, ou outro elemento do sistema, a busca pára e é formada a expressão lógica que determinará o valor da entrada do elemento do sistema pelo qual a busca começou. O processo se repete para o próximo elemento do sistema e assim por diante. Para exemplificar o funcionamento do algoritmo considere o diagrama ilustrado na Figura 5.13.

Nesse exemplo existe apenas um elemento do sistema o *flip-flop* SR, então esse elemento será o ponto de partida do algoritmo.

O primeiro passo é descobrir o elemento conectado a entrada S do elemento SR. Como esse elemento corresponde a uma operação lógica essa informação é guardada e a busca parte agora do elemento E. Esse processo continua até que uma variável de entrada ou outro elemento do sistema seja encontrado. No nosso exemplo encontra-se a variável X conectada a entrada 1 do elemento OU, a informação é guardada e observada-se então a entrada 2 do elemento OU à qual está conectada a variável Y. Como não existem mais entradas a serem observadas para esse elemento, temos que a expressão para a entrada 1 do elemento E está determinada e corresponde a $X+Y$. A busca parte agora da entrada 2 do elemento E, onde se verifica a variável Z, o que implica que a expressão para a entrada S do elemento SR está determinada, $S = ((X + Y) \wedge Z)$. A busca recomeça da entrada R do elemento SR onde encontra-se a variável W, produzindo R = W. A busca termina pois não existem mais elementos do sistema a serem tratados.

Como o elemento do sistema em questão é do tipo SR com prioridade no sinal R, a expressões para S será modificada pra tratar esse caso tornando-se, $S = (((X + Y) \wedge Z) \wedge \overline{W})$, assim no caso de simultaniedade dos sinais de entrada o sinal da entrada R prevalecerá. A função *guarda* retornará o valor 1 se S for avaliado como verdadeiro, 2 para R verdadeiro e 0 para o caso das duas entradas serem avaliadas como falsas fazendo com o que o autômato do SR não execute nenhuma ação mantendo a variável do modelo correspondente a saída do elemento SR inalterada implementando assim a função de memória. Para os elementos temporizados aplica-se o mesmo princípio e não existe nenhum tratamento especial para as expressões lógicas formadas ao final da busca.

Capítulo 6

Estudo de Caso

Neste capítulo será apresentado um estudo de caso mostrando o desenvolvimento do código de um SIS utilizando a metodologia e as ferramentas desenvolvidas durante o presente trabalho.

6.1 A Especificação

A especificação adotada foi o controle de nível de um tanque de uma unidade de geração de hidrogênio da Refinaria Gabriel Passos (REGAP) da Petrobras. O controle do nível do tanque é realizado através do acionamento de um sistema de drenagem composto por uma válvula de escape e uma bomba de sucção. Os diagramas ISA 5.2 da especificação são mostrados nas Figuras 6.1 e 6.2. A lógica utilizada para determinar o acionamento do sistema de drenagem é a seguinte, o sistema tem quatro modos de funcionamento:

Modo 1: Completamente manual;

Modo 2: Completamente automático;

Modo 3: Bomba manual e válvula automática;

Modo 4: Bomba automática e válvula manual.

O modo de funcionamento desejado é escolhido pela combinação dos estados de duas chaves, *AUTOMÁTICO* e *AUTOMÁTICO 2*, que selecionam respectivamente o modo automático ou manual para o acionamento da bomba e da válvula. Com a planta funcionando no modo 1, completamente manual, tanto a válvula quanto a bomba são comandadas independentemente pelo operador. Isto é feito através das chaves *ABRE* e *FECHA*, para a válvula, e *LIGA* e *DESLIGA* para a bomba. No modo 2, completamente automático, o estado dos sensores de nível determinam as ações que serão realizadas.

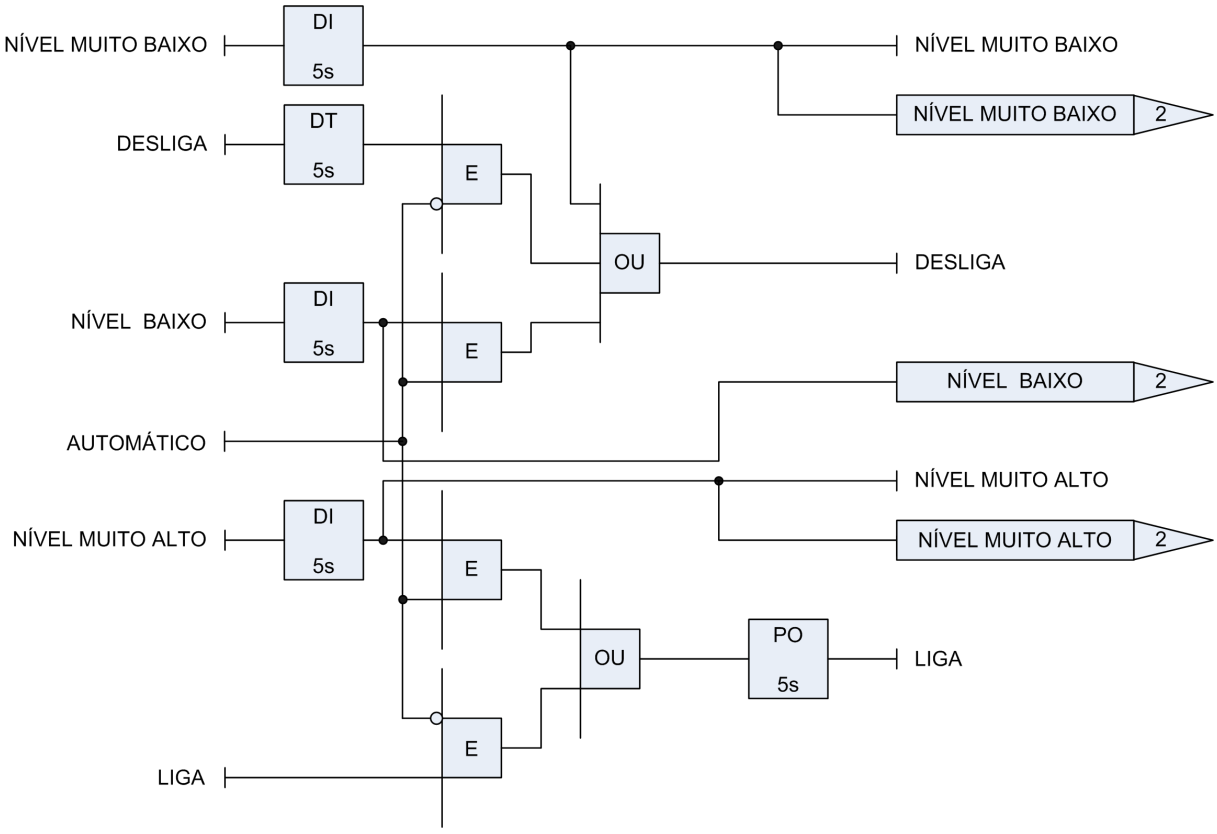

Figura 6.1: Diagrama ISA 5.2 da especificação, página 1.

Os sinais dos sensores são *NÍVEL MUITO BAIXO*, *NÍVEL BAIXO* e *NÍVEL MUITO ALTO*. Nos modos em que a bomba ou a válvula estão no modo manual as ações realizadas são função da combinação do estado dos sensores e dos comandos do operador. Existem também ações de emergência que são tomadas independentemente do modo selecionado e que se combinam com as ações de qualquer modo escolhido para determinar o estado final do sistema. A seguir uma descrição das possíveis combinações para os sinais de entrada e as correspondentes ações que devem ser efetuadas pelo sistema é apresentada.

Independente do modo de funcionamento selecionado, se *NÍVEL MUITO BAIXO* apresentar o valor *verdadeiro* por $5s$ consecutivos a ação tomada pelo SIS deve ser, desligar a bomba e fechar a válvula. Se *NÍVEL MUITO ALTO* apresentar o valor *verdadeiro* por $5s$ consecutivos, abrir a válvula. Na descrição dos modos seguintes, os estados apresentados serão combinações entre as ações dos respectivos modos e das ações de emergência. Para diferenciar as ações de emergência elas serão grafadas em itálico.

No Modo 1, completamente manual, o comportamento dos dois dispositivos é determinado pelo operador. Porém, se verificado alguma das combinações de sinais consideradas críticas, o sistema executará as ações de emergência programadas.

No Modo 2, completamente automático, se *NÍVEL BAIXO* apresentar o valor *verdadeiro* por $5s$ consecutivos, desligar a bomba e fechar a válvula. Se *NÍVEL MUITO ALTO*

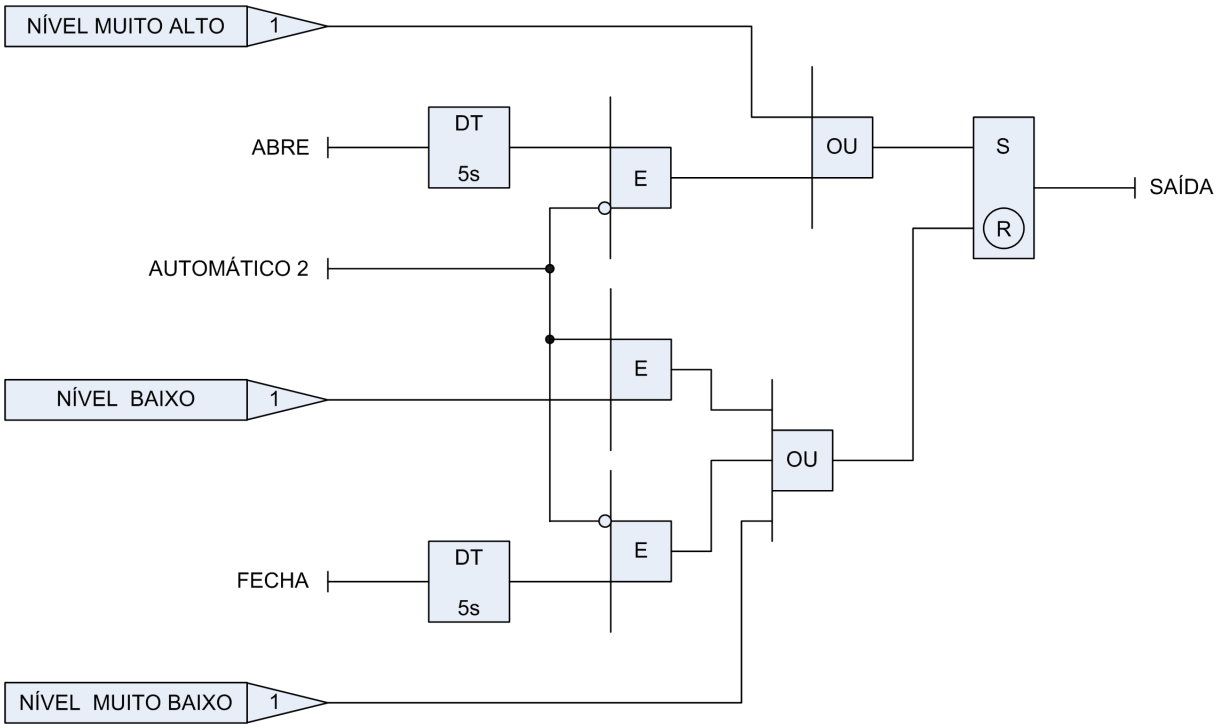

Figura 6.2: Diagrama ISA 5.2 da especificação, página 2.

apresentar o valor *verdadeiro* por 5s consecutivos, ligar a bomba e *abrir a válvula*.

No Modo 3, bomba manual e válvula automática, se *NÍVEL MUITO BAIXO* apresentar o valor *verdadeiro* por 5s consecutivos, *desligar a bomba e fechar a válvula*. Se *NÍVEL BAIXO* apresentar o valor *verdadeiro* por 5s consecutivos, a ação do sistema será fechar a válvula e estado da bomba será determinado pelo operador. Se *NÍVEL MUITO ALTO* apresentar o valor *verdadeiro* por 5s consecutivos, estado da bomba será determinado pelo operador e a ação do sistema será *abrir a válvula*.

No Modo 4, bomba automática e válvula manual, se *NÍVEL MUITO BAIXO* apresentar o valor *verdadeiro* por 5s consecutivos, *desligar a bomba e fechar a válvula*. Se *NÍVEL BAIXO* apresentar o valor *verdadeiro* por 5s consecutivos, a ação do sistema será desligar a bomba e o estado da válvula será determinado pelo operador. Se *NÍVEL MUITO BAIXO* por 5s consecutivos, ligar a bomba e *abrir a válvula*.

6.2 O Código FBD

O código FBD mostrado nas Figuras 6.3 e 6.4 foi produzido a partir dos diagramas ISA 5.2 mostrados nas Figuras 6.1 e 6.2, e reproduz o comportamento descrito na seção anterior. Como mencionado no Capítulo 2 cada um dos padrões, ISA 5.2 e FBD, possui um conjunto próprio de símbolos representando operações que permitem a descrição de diversos sistemas. É possível, no entanto, identificar operações correspondentes entre os dois

padrões, como as realizadas pelos temporizadores DI, DT e PO do ISA 5.2 que correspondem respectivamente as dos elementos TON, TOFF e TP do FBD e as operações lógicas E e OU. Essas operações são representadas por símbolos, tanto para o ISA 5.2 quanto para o FBD, os quais são muito semelhantes graficamente. Isso faz com que o código FBD seja bastante semelhante aos diagramas da especificação. Se considerarmos todos os símbolos definidos em ambos os padrões essa semelhança não se verifica tão fortemente. Os demais símbolos de ambos os padrões correspondem a operações matemáticas e temporizadores especiais e não foram utilizados no presente trabalho por dois motivos: são consideradas apenas variáveis Booleanas e pelo fato de não ocorrerem nos documentos de especificação utilizados no momento de definição do modelo de AT.

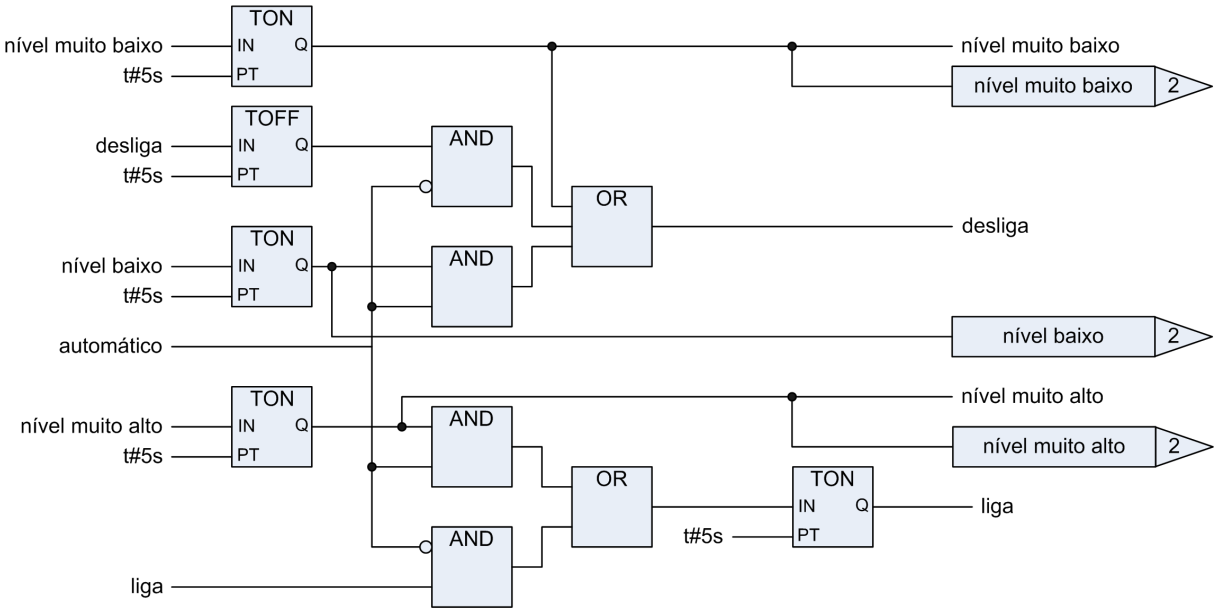

Figura 6.3: Código FBD correspondente à pagina 1 da especificação.

6.3 Ferramentas e Desenvolvimento

O processo de desenvolvimento do nosso estudo de caso é composto das seguintes etapas: escrita da especificação em diagramas ISA 5.2; desenvolvimento do programa em FBD; geração dos modelos da especificação e do código; e finalmente a realização, de forma automática, dos testes de conformidade.

Para escrever a especificação e o programa do SIS, foi utilizado um editor de diagramas. O editor proporciona um ambiente de desenvolvimento para diagramas ISA 5.2 e código FBD. Os diagramas construídos devem utilizar apenas variáveis Booleanas, funções lógicas, temporização e realimentação de sinais. Para ambos os formatos suportados, ISA 5.2 e FBD, essa ferramenta gera um arquivo de saída no formato XML contendo todas

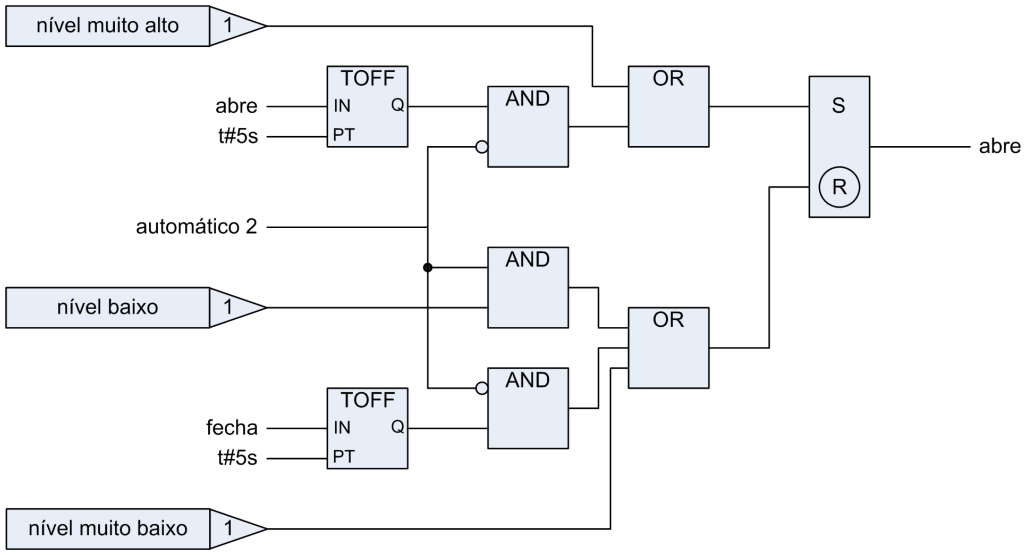

Figura 6.4: Código FBD correspondente à pagina 2 da especificação.

as informações necessárias para a geração dos modelos e visualização gráfica do projeto. Para realizar a geração automática dos modelos da especificação e do código foi utilizada a ferramenta de geração. Essa ferramenta utiliza como entradas para o algoritmo de geração de modelos, as informações presentes nos arquivos gerados pelo editor de diagramas. Ao final da execução do algoritmo é produzido um arquivo, no formato XML, contendo o modelo de autômatos temporizados para o diagrama ISA 5.2 ou código FBD fornecido como entrada. Os arquivos podem ser lidos, executados e editados na ferramenta Uppaal. A ferramenta de geração dos modelos encontra-se integrada ao editor de diagramas e a geração dos modelos é realizada de forma transparente para o usuário, bastando para tanto, informar os diagramas para o qual deseja-se gerar o modelo. O editor foi desenvolvido como parte da etapa de apresentação dos resultados do projeto SIS realizado na UFCG junto à Petrobras, e não como parte integrante do presente trabalho. Ele é utilizado nesse estudo de caso com o objetivo de demonstrar o uso do método para o desenvolvimento de um SIS. A ferramenta de geração dos modelos foi desenvolvida como parte integrante do presente trabalho e do projeto SIS.

Para realizar os testes é utilizada a ferramenta TRON. Para esse exemplo especificamente, as configurações utilizadas para os testes são as seguintes, utilização de tempo lógico, menor atraso possível para a transição escolhida, precisão de uma unidade de tempo e traços que correspondem a duração de trezentas unidades de tempo. A configuração do TRON é realizada através do arquivo *config.trn* e de alguns parâmetros passados na linha de comando.

O TRON é executado e configurado para realização dos testes, através do módulo testador. Esse módulo realiza a interface entre o TRON e o editor. A integração da ferramenta de geração dos modelos e do testador ao editor de diagramas, transforma

esse último em um ambiente de desenvolvimento integrado no qual podem ser realizadas todas as etapas de desenvolvimento do código de um SIS, desde a escrita da especificação, codificação em FBD, geração dos modelos e finalmente os testes automáticos. A interação entre as ferramentas mencionadas acima pode ser visualizada na Figura 6.5. Os blocos brancos representam arquivos e os blocos mais claros as ferramentas que estão integradas. O TRON é mostrado em ton mais escuro por ser uma ferramenta externa. As setas claras e pontilhadas representam as operações de leitura ou escrita de arquivos, e as setas escuras com traço cheio representam a comunicação entre ferramentas.

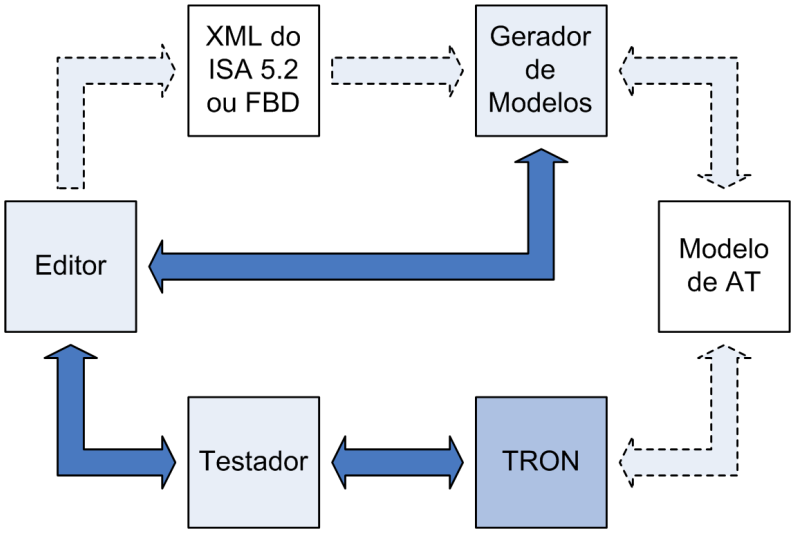

Figura 6.5: Interação entre as ferramentas utilizadas no desenvolvimento do SIS.

6.4 Os Testes

Para ilustrar o processo de teste tanto para o caso de um teste bem sucedido, quanto para o caso de um teste que falhou, foram introduzidos erros no modelo da implementação. Os erros introduzidos são de três tipos: erros de temporização, ou seja, erros na programação dos tempos de contagem dos temporizadores, erros de conexão que correspondem a erros no caminho dos sinais entre os elementos do diagrama FBD e erros por troca de elementos. Os erros inseridos são identificados abaixo:

1. Tempo de contagem do DI conectado ao sinal *nível muito baixo* programado para $3s$, enquanto o correto seria $5s$, ver Figura 6.6.

2. Sinal *automático* não está sendo invertido na entrada do elemento AND conectado ao sinal *liga*, enquanto o correto seria inverter, ver Figura 6.6.

3. Troca do elemento OR conectado ao sinal *nível muito alto*, na folha 2 do diagrama, por um elemento AND, ver Figura 6.7.

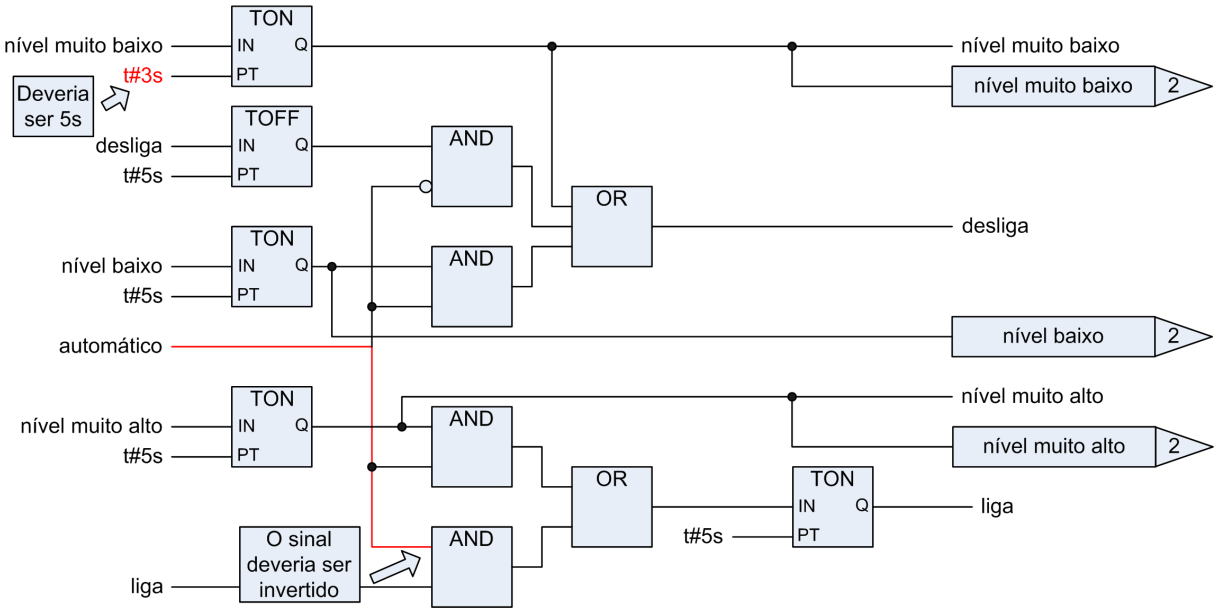

Figura 6.6: Identificação dos erros inseridos no FBD, primeira parte.

É importante destacar que os erros de programação só serão detectados se as seguintes condições verificarem-se. Primeira, o traço de execução utilizado para os testes possui a combinação de sinais necessária para que o erro ocorra. Segunda, o resultado das variáveis de saída é influenciado pela ocorrência do erro dentro do tempo determinado para os testes. Isso ocorre devido ao tipo de teste realizado, o qual compara as saídas do modelo da especificação com as saídas do modelo da implementação para um determinado conjunto de entradas aplicados a ambos os modelos.

Os modelos para a implementação utilizados nos testes foram um modelo correto e outros quatro modelos com erros. Dentre os modelos que contêm erros estão, um modelo com o erro 1, um modelo com o erro 2, um modelo com o erro 3 e outro contendo os erros 1, 2 e 3. A seguir serão apresentados os resultados dos testes para cada um dos modelos mencionados.

Começando com o modelo correto da implementação o resultado obtido foi: o modelo passou no teste, ou seja, não foi detectado dentro do tempo programado para o teste nenhuma diferença de comportamento entre os modelos da especificação e da implementação.

O resultado obtido para o teste executado no modelo com o erro 1 foi: o teste falhou no instante de tempo 140 dos 300 programados, pois foi observada a saída *sinc()* enquanto a saída esperada era *fim()*. A saída observada significa que o modelo da implementação completa o processamento das entradas depois do modelo da especificação. Analisando as informações das localidades ocupadas pelos autômatos e os valores das variáveis do modelo após o erro, verifica-se que os temporizadores DI1 e DT1 conectados respectivamente as

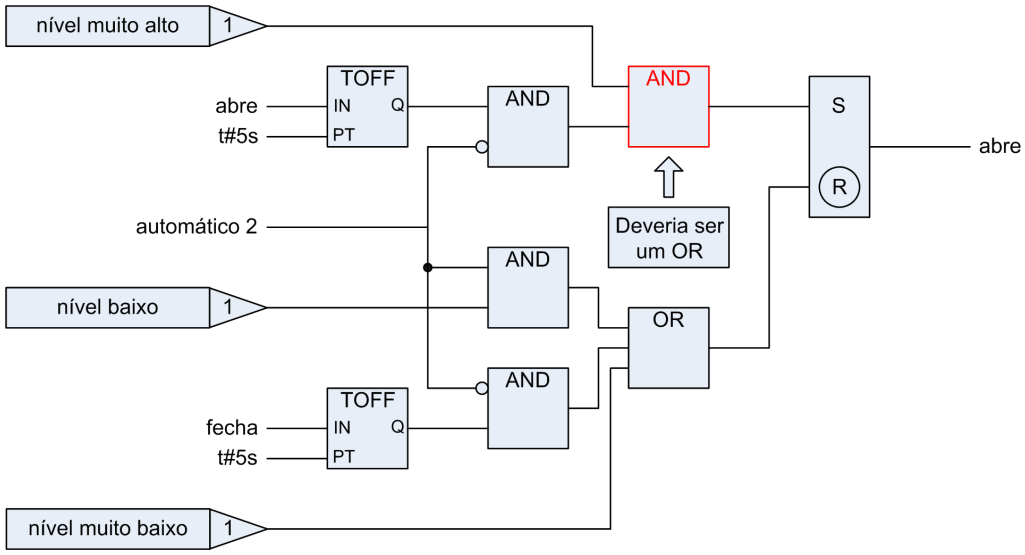

Figura 6.7: Identificação dos erros inseridos no FBD, segunda parte.

entradas físicas *nível muito baixo* e *desliga* já foram disparados, indicando que um desses contadores pode ter causado o erro. Neste caso o erro é uma programação inadequada do tempo de contagem do temporizador DI1, ver Figura 6.6.

O resultado obtido para o teste executado no modelo com o erro 2 foi: o teste falhou no instante de tempo 5 dos 300 programados, pois foi observada a saída *verdadeiro()* enquanto a saída esperada era *falso()*. Observando os diagramas da especificação, Figura 6.1, percebe-se que a saída produzida pelo modelo da implementação considerando o traço que produziu o erro só é possível se o sinal *automático* não sofrer inversão, o que não condiz com o que está previsto na especificação. Neste caso o erro é a não inversão do sinal *automático*, na entrada do elemento AND. Ver Figura 6.6.

O resultado obtido para o teste executado no modelo com o erro 3, é o seguinte: o teste falhou no instante de tempo 37 dos 300 programados, pois foi observada a saída *sinc()*, enquanto a saída esperada era *fim()*. Reconstruindo o traço que produz o erro percebemos que a implementação produziria um valor *falso* para a variável de saída *abre*, diagrama da Figura 6.2, que é o oposto do verificado para a especificação. Com essa informação e observando as entradas para a última iteração do modelo antes do erro ser detectado verifica-se que o temporizador DI1 estava prestes a disparar. Na especificação a saída do DI1 está conectada a um elemento OU que por sua vez está conectada a entrada S do elemento SR, assim, o disparo do DI1 ocasionou o valor *verdadeiro* para a variável *abre*. Podemos deduzir então que ouve uma troca do elemento OU por um elemento AND no caminho dos sinais para a entrada S do elemento SR na implementação. Ver Figura 6.7.

O resultado obtido para o teste executado no modelo contendo os três erros simultaneamente foi o seguinte: os resultados observados para cada um dos erros mencionados acima se repetem um por um, na mesma ordem temporal. Portanto, o primeiro erro

detectado é o erro 2, seguido do erro 3 e por fim o erro 1. As análises para cada um dos erros são as mesmas mencionadas anteriormente.

Capítulo 7

Conclusão e Trabalhos Futuros

Conforme mencionado no Capítulo 1 o problema abordado foi o seguinte: como testar se o software do SIS está em conformidade com a especificação que foi fornecida como base para o desenvolvimento do sistema. Como solução foi proposto o desenvolvimento de um método para aumentar a confiança no funcionamento dos SIS. A abordagem para atingir esse objetivo foi construir modelos formais de AT da especificação e do código do SIS e realizar testes de conformidade entre os modelos, dessa forma quanto maior o número de testes realizados com sucesso no modelo do sistema maior a confiança no SIS desenvolvido. No entanto, para que esse método pudesse ser aplicado durante o processo de desenvolvimento de um SIS era necessário construir modelos de AT para os elementos do padrão ISA 5.2 (especificação) e para código FBD (implementação), além disso, a geração dos modelos e a realização testes deveriam ser realizadas de forma automática. O presente trabalho consistiu em desenvolver os modelos de AT, produzir uma ferramenta de software para gerá-los automaticamente a partir dos digramas da especificação e do código do SIS, utilizar a ferramenta TRON para realização automática dos testes e desenvolver um estudo de caso demonstrando a aplicação do método no desenvolvimento de um SIS.

Como resultado da aplicação desse método esperava-se a detecção prematura de erros, ou seja, ainda na fase de desenvolvimento, e o aumento da confiança no funcionamento do SIS devido a possibilidade de realização de uma grande bateria de testes de forma automática. O resultado dos testes realizados pelo desenvolvedor poderia ser entregue a empresa contratante como um indicativo de uma implementação confiável. Vislumbrou-se também o uso do método em campo como mais uma forma da empresa contratante experimentar o código do SIS desenvolvido em um processo que poderia auxiliar no comissionamento que já é realizado.

No estudo de caso discutido no Capítulo 6 foi explorado o cenário de desenvolvimento de um SIS utilizando a ferramenta de geração de modelos de AT desenvolvida, a ferramenta de testes TRON e uma ferramenta para edição dos digramas da especificação e do

código do SIS. Para realizar os teste o desenvolvedor precisa apenas executar a ferramenta de geração e informar o modelo a ser testado, ambas tarefas que são realizadas na própria ferramenta de edição, para obter um veredicto sobre a implementação em desenvolvimento com respeito a especificação do SIS.

A utilização dessas ferramentas de forma integrada e os resultados obtidos através desse estudo de caso demonstram a viabilidade da aplicação do método para aumento da confiança nos SIS na prática. Como esperado os erros foram detectados através da realização dos testes de conformidade. Os modelos, os casos de teste, a aplicação dos testes e os resultados informando a presença de erros foram todos gerados ou realizados de forma automática. Essa característica indica que não é necessário nenhum conhecimento adicional por parte do desenvolvedor do sistema para que o método possa ser aplicado. Outro resultado importante é a possibilidade de identificar o local do erro através da análise do resultado fornecido pelo TRON, porém a análise dos resultados não está sendo realizada de forma automática.

7.1 Trabalhos Futuros

Os testes provaram sua utilidade ao detectar erros propositadamente inseridos em alguns dos diagramas utilizados no estudo de caso, porém o TRON gera casos de teste aleatórios, sem nenhuma estratégia de seleção ou heurísticas para domínios específicos. Além disso, a saída dos resultados, no caso de falha, é de difícil interpretação para o usuário. Por isso, fica a sugestão como trabalhos futuros o aperfeiçoamento do TRON ou desenvolvimento de um novo testador para melhorar a qualidade dos testes gerados e fornecer os resultados de forma amigável para o usuário, através de uma interface gráfica, inclusive indicando o ponto de falha para facilitar sua correção.

Referências Bibliográficas

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Luciano Baresi, Marco Mauri, Antonello Monti, and Mauro Pezzè. PLCTools: Design, formal validation, and code generation for programmable controllers. In *Proceedings of the IEEE Conference on System, Man, and Cybernetics*, volume 4, pages 2437–2442, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[3] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*.

[4] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *System and Software Verification Model-Checking Techniques and Tools*. 2001.

[5] Gèraud Canet, Sandrine Couffin, Jean-Jacques Lesage, Antoine Petit, and Philippe Schnoebelen. Towards the automatic verification of PLC programs written in Instruction List. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*.

[6] Christos G. Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publisher, 1999.

[7] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[8] Leandro Dias da Silva, Luiz Paulo de Assis Barbosa, Kyller Costa Gorgônio, Angelo Perkusich, and Antonio Marcus Nogueira Lima. On the automatic generation of timed automata models from function block diagrams for safety instrumented systems. In *34th Annual Conference of the IEEE Industrial Electronics Society*.

[9] Leandro Dias da Silva; Luiz Paulo de Assis Barbosa; Kyller Costa Gorgônio; Angelo Perkusich; Antonio Marcus Nogueira Lima. Geração automática de autômatos temporizados para diagramas de blocos funcionais. In *XVII Congresso Brasileiro de Automática*, Juiz de Fora, Brasil, 2008.

[10] Luiz Paulo de Assis Barbosa, Kyller Gorgônio, Antonio Marcus Nogueira Lima, Angelo Perkusich, and Leandro Dias da Silva. On the automatic generation of timed automata models from isa 5.2 diagrams. In *Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation*, Patras, Greece, 2007.

[11] E. A. Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[12] Carlo Ghezzi, Dino Mandrioli, Sandro Morasca, and Mauro Pezzè. A unified high-level petri net formalism for time-critical systems. *IEEE Trans. Software Eng.*, 17(2):160–172, 1991.

[13] Steve Gillespie. Safety instrumented systems. Technical report, Shell Global Solutions UK, Measurement, Instrumentation and Automation Group.

[14] Monika Heiner and Thomas Menzel. A petri net semantics for the PLC language instruction list. In *Proc. IEE Workshop on Discrete Event Systems*.

[15] The Instrumentation, Systems, and Automation Society. *Binary Logic Diagrams for Process Operations*, ISA 5.2-1976 (R1992) edition, July 1992.

[16] Karl-Heinz John and Michael Tiegelkamp. *IEC 61131-03: Programming Industrial Automation Systems*. Springer-Verlag, Berlin, Germany, 2001.

[17] Angelika Mader and Hanno Wupper. Timed automaton models for simple programmable logic controllers. In *Proceedings of Euromicro Conference on Real-Time Systems*, York, UK, June 1999.

[18] Kenneth L. McMillan. *The SMV System: for SMV version 2.5.4*, November 2000. Available at: http://www-2.cs.cmu.edu/~modelcheck/smv/smvmanual.ps.gz.

[19] Il Moon. Modeling programmable logic controllers for logic verification. *IEEE Control Systems Magazine*, 14(2):53–59, 1994.

[20] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[21] Amir Pnueli. The temporal semantics of concurrent programs. In *Proceedings of the International Sympoisum on Semantics of Concurrent Computation*, pages 1–20, London, UK, 1979. Springer-Verlag.

[22] Olivier Rossi and Philippe Schnoebelen. Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs. In *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*.

[23] Jan Tretmans. Testing techniques. Technical report, Formal Methods and Tools Group - Faculty of Computer Science University of Twente - The Netherlands, 2002.

[24] H.X. Willems. Compact timed automata for PLC programs. Technical Report CSI-R9925, University of Nijmegen, nov 1999.

[25] Mohammed Bani Younis and Georg Frey. Formalization of existing plc programs: A survey. In *Proceedings of the IEEE/IMACS Multiconfrence on Computational Engineering in Systems Applications*.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo