UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA – PPGEEL

ALEXANDRE JUNKES PINOTTI

UM MÉTODO PARA PROJETO DE SISTEMAS EMBARCADOS BASEADO NO CONTROLE SUPERVISÓRIO MODULAR LOCAL

JOINVILLE - SC
2012

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA – PPGEEL

ALEXANDRE JUNKES PINOTTI

UM MÉTODO PARA PROJETO DE SISTEMAS EMBARCADOS BASEADO NO CONTROLE SUPERVISÓRIO MODULAR LOCAL

Dissertação submetida à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica

Orientador: Prof. Dr. André Bittencourt Leal

JOINVILLE - SC
2012

ALEXANDRE JUNKES PINOTTI

UM MÉTODO PARA PROJETO DE SISTEMAS EMBARCADOS BASEADO NO CONTROLE SUPERVISÓRIO MODULAR LOCAL

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica área de concentração em Automação de Sistemas e aprovado em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica (PPGEEL) da UDESC.

Banca Examinadora:

Orientador:

Prof. Dr. André Bittencourt Leal
Universidade do Estado de Santa Catarina - UDESC

Membro:

Prof. Dr. Antonio Eduardo Carrilho da Cunha
Instituto Militar de Engenharia - IME

Membro:

Prof. Dr. Marco Aurélio Wehrmeister
Universidade do Estado de Santa Catarina - UDESC

JOINVILLE - SC, 11/12/2012

P657u

Pinotti, Alexandre.

Um Método para Projeto de Sistemas Embarcados Baseado no Controle Supervisório Modular Local / Alexandre Junkes Pinotti;
orientador: Dr. André Bittencourt Leal. – Joinville, 2012.

119 f. : il ; 30 cm.

Incluem referências.

Dissertação (mestrado) – Universidade do Estado Santa Catarina, Centro de Ciências Tecnológicas, Mestrado em Engenharia Elétrica, Joinville, 2012.

1. Automação de Sistemas. 2. Teoria de Controle Supervisório. I. Leal, André Bittencourt.

CDD 629.8

AGRADECIMENTOS

Muito obrigado *Deus* pelo dom da Vida e por ter me propiciado o tempo e a motivação necessários para a dedicação aos meus estudos e à conclusão de meu trabalho.

Muito obrigado minha mãe, *Vilma Sueli Junkes*, por sua paciência e compreensão nos meus momentos de maior dificuldade e obrigado por seu Amor eterno.

Muito obrigado meu pai, *Edson Luiz Pinotti*, por ser o meu exemplo a ser seguido e meu maior apoiador.

Muito obrigado minha família, *Erica Rech Pinotti*, quem carinhosamente apelidei de tia, meus irmãos, *Rodrigo Pinotti* e *Lucas Pinotti*, meus avós, tios e primos por todo seu carinho, pela sua fé e por sempre acreditarem em mim.

Muito obrigado meu orientador e amigo, professor *André Bittencourt Leal* por tantos anos de convívio e amizade, por me apresentar oportunidades que me propiciaram aprender, desenvolver e crescer.

Muito obrigado aos professores membros da banca examinadora, professor *Antonio Eduardo Carrilho da Cunha* e professor *Marco Aurélio Wehrmeister* por seu tempo dedicado e suas enriquecedoras contribuições.

Muito obrigado meus amigos, *Douglas Baumer*, *Guilherme Burg Winter*, *Gustavo Gadotti*, *Ricardo de Oliveira Lima*, *Rodrigo Bordin Trindade*, *Thiago Bordin Trindade*, família *Blackstar Basquete*, por me propiciar tantos momentos de alegria quando mais precisei. Aos amigos e também colegas de profissão, *Fernanda Mendes de Morais*, *Raphael Jorge Millnitz dos Santos*, *Luiz Ricardo Lima* e *Horácio Beckert Polli*, por além disso, me incentivarem a continuar.

Muito obrigado aos colegas de trabalho *Valmor Adami Junior*, *Daniel Gumiero Noronha Maas* e *Lucas Giovanni Locatelli* por todas as trocas de ideias e discussões que geraram diversas contribuições. Em nome desses meu obrigado à *Whirlpool* por permitir que desenvolvesse meu trabalho.

A todos meu Muito Obrigado.

Aprender é a única de que a mente nunca se cansa, nunca tem medo e nunca se arrepende.
– Leonardo da Vinci.

RESUMO

PINOTTI, Alexandre Junkes. **Um Método para Projeto de Sistemas Embarcados Baseado no Controle Supervisório Modular Local**. 2012. 119f. Dissertação (Mestrado Profissional em Engenharia Elétrica – Área: Automação de Sistemas) – Universidade do Estado de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica, Joinville, 2012.

Neste trabalho é proposto um método de projeto para sistemas embarcados concebido com base na Teoria de Controle Supervisório de Sistemas a Eventos Discretos. O método é composto por etapas que vão desde a especificação até a aprovação do sistema de controle. A etapa de concepção do método utiliza da abordagem modular local para síntese de supervisores minimamente restritivos que limitam o comportamento da planta através da desabilitação de eventos controláveis e são não bloqueantes em relação ao conjunto de estados marcados. O método ainda inclui uma arquitetura de implementação do controle supervisório direcionada para microcontroladores, visando solucionar os problemas da causalidade, escolha e sincronização inexata. Uma interface é concebida para concentrar a geração de eventos controláveis e não controláveis. Apresenta-se uma solução para o problema da escolha quando são implementados supervisores reduzidos sendo tal escolha realizada de modo *online* definindo um evento controlável, entre os possíveis, de forma aleatória. Apresenta-se uma ferramenta desenvolvida para a geração de código baseada na estrutura de implementação proposta. O código do módulo principal é gerado de forma a ser independente do número de supervisores e modelos de planta envolvidos, sendo o mesmo para qualquer aplicação de controle. Utiliza-se a ferramenta para obtenção do código da lógica de controle em um estudo de caso e para a regulação de temperatura em um refrigerador comercial. A validação do controle é realizada utilizando-se de ferramentas aplicadas à validação de eletrodomésticos da linha branca na Whirlpool Eletrodomésticos.

Palavras-chave: Sistemas a Eventos Discretos. Teoria de Controle Supervisório. Abordagem Modular Local. Métodos Formais. Sistemas Embarcados. Implementação. Microcontrolador.

ABSTRACT

PINOTTI, Alexandre Junkes. **Embedded System Design Method Based on Local Modular Supervisory Control**. 2012. 119p. Dissertation (Masters in Electrical Engineering – Area: Automation Systems) – Santa Catarina State University. Electrical Engineering Post Graduation Program, Joinville, 2012.

This work presents a method for embedded systems design based on the Supervisory Control Theory for Discrete-Event Systems. The method is composed of steps comprising the specification phase until the approval of the control system. The conception phase is based on the local modular approach for the synthesis of minimally restrictive supervisors that constrains the plant behavior by disabling controllable events and are nonblocking with respect to the set of marked states. The method also includes an implementation architecture for the supervisory control applied to microcontrollers to solve problems such as causality, choice and inexact synchronization. An interface concentrates all controllable and uncontrollable events generation. It deals with the choice problem when reduced supervisors are implemented, randomly choosing one controllable event among the possible ones. This dissertation also presents a tool developed for the automatic code generation of the proposed implementation structure. The main module code is generated to be independent of the number of supervisors and plant models, that is, is the same for any control application. The tool is used to obtain the control applied to a case study and also for the temperature control of a commercial refrigerator. The control system has been validated using the same tools applied for the validation of household appliances at Whirlpool Corporation.

Keywords: Discrete-Event Systems. Supervisory Control Theory. Local Modular Approach. Formal Methods. Embedded Systems. Implementation. Microcontroller.

LISTA DE FIGURAS

2.1 Representação de um autômato por um grafo dirigido 24

2.2 Autômato bloqueante 25

2.3 Um autômato como modelo na Teoria de Controle Supervisório 26

2.4 Ação de controle de um Supervisor Monolítico 27

2.5 Abordagem modular local: (a) Subplantas (b) Representação por Sistema Produto (c) Plantas locais 29

2.6 Ações de controle de Supervisores Modulares Locais 30

3.1 Dinâmica de processamento do controle: (a) ciclo de execução (b) monitoramento periódico de sinais de resposta e tradução em eventos não controláveis . 33

3.2 Conversões entre sinais e eventos 34

3.3 Supervisor com problema de escolha e implementação bloqueante 37

3.4 Atuação do *dispatcher* proposto por Basile e Chiacchio (2007) 38

3.5 Atrasos entre Controlador e Planta Física presentes na estrutura lógica do controle 39

3.6 Autômatos com linguagens atendendo a diferentes propriedades 41

3.7 Atraso na conversão do sinal de resposta em evento 43

3.8 Estrutura de Implementação proposta por Queiroz (2004) 45

3.9 Estrutura de Implementação de Teixeira (2008) 48

4.1 Etapas do Método de Projeto baseado no Controle Supervisório 52

4.2 Estruturas (a) Física e (b) Lógica do controle supervisório em Sistemas Embarcados 54

4.3 Arquitetura de Implementação 54

4.4 Ação conjunta de supervisores que elimina a condição do problema de escolha 59

4.5 (a) Autômato cuja linguagem atende à Definição 3.4 (b) Atrasos na conversão de sinais em eventos 61

5.1 Diagrama do Refrigerador adaptado de Teixeira (2008) 67

5.2 (a) Comportamento autônomo de acordo com a variação da temperatura ambiente (b) Histerese para a temperatura ambiente definindo os *setpoints* de funcionamento do compressor 68

5.3 Diagrama do comando PWM do conversor CC-CC para acionamento do ventilador 69
5.4 G_1 Modelo do compressor 71
5.5 G_2 Modelo para a seleção dos *setpoints* de funcionamento do compressor . . . 71
5.6 G_3 Modelo do *Sensor A* posicionado no evaporador 71
5.7 G_4 Modelo do *Sensor B* para medição da temperatura ambiente 72
5.8 G_5 Modelo do *damper* eletrônico 72
5.9 G_6 Modelo do ventilador de velocidade variável 72
5.10 E_1 Modelo da especificação para a relação de *T_Int* com o compressor e *damper* 73
5.11 E_2 Modelo da especificação de controle para a seleção do *setpoint* com a variação de *T_Env* 73
5.12 E_3 Modelo da especificação da velocidade do ventilador de acordo com o estado do *damper* 74
5.13 Supervisores obtidos para o controle do Refrigerador Autônomo 75
5.14 Supervisores reduzidos com o uso do TCT (FENG; WONHAM, 2006) 75
5.15 Diagrama de Classes em UML da Estrutura dos Arquivos 77
5.16 Função principal do módulo *Supervisory Control* 81
5.17 Supervisores na Camada *Controle* 82
5.18 Código ANSI C da Função de Atualização de Estados Supervisores 83
5.19 Código ANSI C da Função de Obtenção das Desabilitações dos Supervisores . 83
5.20 Implementacao do Supervisor SL_1 84
5.21 Código ANSI C das Listas e variável de estado atual de *Supervisors.h* 85
5.22 Verificação da Escolha na camada *Controle* 85
5.23 Trecho do código em ANSI C da função principal para obtenção da ação de controle determinística 86
5.24 Código ANSI C para execução das Funções de Leitura da *Interface* 87
5.25 Código ANSI C da Interface de Leitura para o evento *e2* 87
5.26 Diagrama da bancada para testes de validação de *software* 89
5.27 Diferentes escolhas realizadas pelo controle 90
5.28 Diferentes ciclos do compressor para mudanças na temperatura ambiente . . . 92

6.1 Refrigerador comercial 95
6.2 Modelos para os subsistemas da planta 97
6.3 Modelos das especificações de controle 98

6.4 *Setup* para monitoramento do produto 100
6.5 Ciclos de operação do refrigerador comercial 101
6.6 Ciclo de operação do produto com abertura de porta do refrigerador 102

A.1 Utilização do *plugin* para geração de código 119

LISTA DE TABELAS

3.1 Comparativo entre Estruturas para Microcontrolador 49

5.1 Relação dos Eventos Não Controláveis do Refrigerador Autônomo 70
5.2 Relação dos Eventos Controláveis do Refrigerador Autônomo 70

6.1 Supervisores para controle do refrigerador comercial 99
6.2 Comparação de Consumo de Memória 104
6.3 Variáveis das equações de estimativa de consumo de memória 106
6.4 Consumo base dos tipos de variáveis em C para plataforma 8 *bits* 107

LISTA DE ABREVIATURAS

AD	Analógico-Digital
ANSI	*American National Standards Institute*
CA	Corrente Alternada
CAN	*Controller Area Network*
CC	Corrente Contínua
CLP	Controlador Lógico Programável
FIFO	*First In First Out*
GASR	Grupo de Automação de Sistemas e Robótica
I2C	*Inter-Integrated Circuit*
IDES	*Integrated Discrete-Event Systems*
NTC	*Negative Temperature Coefficient*
PC	Computador Pessoal
PWM	*Pulse-Width Modulation*
RAM	*Random Access Memory*
ROM	*Read-Only Memory*
RSP	Representação por Sistema Produto
SED	Sistema a Eventos Discretos
TCS	Teoria de Controle Supervisório
UART	*Universal Asynchronous Receiver Transmitter*
UL	*Underwriters Laboratories*
VDE	*Association for Electrical, Electronic and Information Technologies e.V.*

LISTA DE SÍMBOLOS

Linguagens e Autômatos como Modelos para SED

Σ	Alfabeto
σ	Símbolo
L,L_i	Linguagens
t,s,s_i	Cadeias, palavras, sequências
ε	Palavra vazia
Σ^*	Conjunto de todas as cadeias finitas de Σ
$L \subseteq \Sigma^*$	Linguagem sobre Σ
st	Concatenação das palavras s e t
$s\|\|\|t$	Conjunto de todos os entrelaçamentos entre s e t
$t \leq s$	t é prefixo de s
$\overline{L}$	Prefixo-fechamento de L
$L_1\|\|L_2$	Produto síncrono entre L_1 e L_2.
A	Autômato determinístico de estados finitos
Q	Conjunto de estados
q_0	Estado inicial
Q_m	Conjunto de estados marcados
$\delta : Q \times \Sigma \to Q$	Função de transição
$\widehat{\delta} : Q \times \Sigma^* \to Q$	Função de transição estendida para uma sequência
$\Sigma(q)$	Conjunto de estados ativos no estado q
$\Phi(q)$	Conjunto de estados desabilitados no estado q
$\setminus$	Operador de remoção de eventos de um conjunto
$\Sigma(q_0) \setminus \Phi(q_1)$	Conjunto de eventos ativos após a remoção de eventos desabilitados

Teoria de Controle Supervisório

G	Modelo de planta a ser controlada
E	Especificação de controle
K	Linguagem alvo para uma planta G e especificação E
S	Supervisor
Σ	Conjunto de eventos
Σ_u	Conjunto de eventos não controláveis em Σ
Σ_c	Conjunto de eventos controláveis em Σ
$L(G)$	Linguagem gerada pela planta G
$L_m(G)$	Linguagem marcada da planta G
S/G	Planta G sob supervisão de S
$C(K,L(G))$	Linguagens controláveis contidas na linguagem alvo K
$SupC(K,L(G))$	Suprema linguagem em C, máxima linguagem controlável contida em K

G_i	Modelo de subplanta componente de G
AG_j	Subplantas assíncronas para composição da RSP
E_i	Especificação de controle modular local
GL_i	Planta local associada à especificação E_i
S_i	Supervisor modular local
SL_i	Supervisor modular local reduzido

Arquitetura de Implementação

SL	Supervisores modulares locais
SP	Sistema produto
Φ_{SL}	Conjunto de eventos controláveis desabilitados pelos supervisores locais
Σ_i	Conjunto de todos os eventos controláveis ativos pelos subsistemas da planta
$\Sigma_i \setminus \Phi_{SL}$	Conjunto de eventos controláveis sob ação de controle dos supervisores locais
Φ_{Esc}	Conjunto de eventos controláveis desabilitados pela escolha
$\Sigma_{Controle}$	Conjunto de eventos controláveis sob ação determinística

SUMÁRIO

1 INTRODUÇÃO 18
1.1 OBJETIVOS GERAIS E ESPECÍFICOS 19
1.2 ORGANIZAÇÃO DO DOCUMENTO 20
2 REVISÃO BIBLIOGRÁFICA DA TEORIA DE CONTROLE SUPERVISÓRIO . 21
2.1 SISTEMAS A EVENTOS DISCRETOS 21
2.2 LINGUAGENS E OPERAÇÕES SOBRE LINGUAGENS 22
2.2.1 Concatenação 22
2.2.2 Prefixo-Fechamento 23
2.2.3 Projeção 23
2.2.4 Produto Síncrono 23
2.3 LINGUAGENS E AUTÔMATOS COMO MODELOS PARA SED 24
2.3.1 Bloqueio em um SED 25
2.4 TEORIA DE CONTROLE SUPERVISÓRIO 25
2.4.1 Abordagem Monolítica 27
2.4.2 Abordagem Modular Local 28
3 REVISÃO SOBRE IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO . . 31
3.1 IMPLEMENTAÇÃO DO CONTROLE SUPERVISÓRIO 31
3.1.1 Causalidade 35
3.1.2 Problema da Escolha - Determinismo 36
3.1.3 Sincronismo entre Controlador e Planta Física 38
3.2 ESTRUTURAS E ABORDAGENS DE IMPLEMENTAÇÃO 44
3.3 CONCLUSÕES DO CAPÍTULO 49
4 MÉTODO E ARQUITETURA GENÉRICA DE IMPLEMENTAÇÃO 51
4.1 MÉTODO PARA O PROJETO DO CONTROLE DE SISTEMAS EMBARCADOS 51
4.2 DESCRIÇÃO DA ARQUITETURA DE IMPLEMENTAÇÃO 53
4.2.1 Problema da Escolha - Determinismo 58
4.2.2 Sincronismo entre Controlador e Planta 60

4.2.3 Causalidade 62
4.3 CONCLUSÕES DO CAPÍTULO 63
5 APLICAÇÃO DO MÉTODO EM ESTUDO DE CASO 66
5.1 REFRIGERADOR AUTÔNOMO DE DOIS COMPARTIMENTOS 66
5.1.1 Estratégia de Controle do Refrigerador Autônomo 67
5.1.2 Descrição dos Eventos e Modelos da Planta 69
5.1.3 Modelos das Especificações de Controle 73
5.1.4 Síntese dos Supervisores 74
5.2 IMPLEMENTAÇÃO DO CONTROLE EMBARCADO 76
5.2.1 Estrutura de arquivos utilizando linguagem ANSI C 79
5.2.2 Detalhamento da implementação em linguagem ANSI C 81
5.3 TESTES DE VALIDAÇÃO DO CONTROLE 88
5.3.1 Teste de atuação do Determinador 89
5.3.2 Teste dos ciclos para diferentes valores de temperatura ambiente 91
5.4 CONCLUSÕES DO CAPÍTULO 92
6 APLICAÇÃO DO MÉTODO EM REFRIGERADOR COMERCIAL 94
6.1 DESCRIÇÃO DO REFRIGERADOR COMERCIAL 94
6.2 MODELAGEM DO CONTROLE 95
6.3 INTEGRAÇÃO DO CONTROLE SUPERVISÓRIO NO CÓDIGO DO PRODUTO . 99
6.3.1 Testes do Controle Supervisório 100
6.3.2 Comparações do Consumo de Memória do Microcontrolador 103
6.4 CONCLUSÕES DO CAPÍTULO 108
7 CONCLUSÃO 110
REFERÊNCIAS 113
A PLUGIN DE GERAÇÃO AUTOMÁTICA DE CÓDIGO 118
A.1 INSTALAÇÃO DE *PLUGINS* NO IDES3 118
A.2 *PLUGIN* PARA GERAÇÃO DE CÓDIGO 118

Capítulo 1

Introdução

O projeto da lógica de controle em sistemas embarcados nas mais variadas técnicas e abordagens é suscetível à ocorrência de defeitos. Desde o simples mau funcionamento ou perda de funcionalidade até defeitos que levam a riscos à segurança de usuários ou do próprio sistema. Com o aumento da complexidade de algoritmos e rotinas incorporados em tais sistemas também elevam-se os riscos associados (HECHT, 2005).

Tendo conhecimento da criticidade desses defeitos a indústria de maneira geral investe em tempo de engenharia para testes e validação de seus produtos, não sendo diferente para as empresas da linha de bens de consumo. Os extensivos processos de validação e aprovação de produtos são aplicados a nível de *hardware* e de *software*. Nesses casos os testes já iniciam na depuração do código em desenvolvimento que, por vezes, também consome elevado tempo da etapa de execução do projeto.

Em outra vertente, as empresas necessitam de certificações em produtos para a comercialização em determinados mercados. Normas e padrões devem ser adotados no desenvolvimento para que institutos como *Underwriters Laboratories* (UL) e *Association for Electrical, Electronic and Information Technologies e.V.* (VDE) emitam tais certificações. As exigências aumentam os requisitos de projeto dos sistemas com controle embarcado a exemplo do *software* Classe B IEC60730 (IEC, 2010).

Quando a etapa de desenvolvimento (implementação) é baseada na experiência do projetista a possibilidade de ocorrência de erros no sistema de controle é maior do que quando utilizadas ferramentas formais (CRUZ, 2011). Nesse caso, os testes e validações são normalmente usados para a identificação desses erros, o que torna a fase de validação demorada e cara, pois segundo Teixeira (2008) quanto mais cedo erros são identificados mais baixo o custo para a solução do problema.

Neste contexto torna-se cada vez mais recorrente a busca pela prática de novas técnicas, abordagens e métodos de desenvolvimento de projeto de sistemas embarcados (SANGIOVANNI-

VINCENTELLI; MARTIN, 2001). Abordagens de desenvolvimento ágil, direcionadas a testes, baseadas em simulação e em modelos são alguns exemplos de técnicas com aceitação no meio industrial. Tais práticas visam a melhoria na captura dos requisitos e auxiliam a etapa de execução a não desviar de seus atendimentos. Em especial o desenvolvimento baseado em modelos (*Model-Based Design*) é utilizado ao longo de todo o ciclo de projeto possibilitando a identificação de problemas mais cedo e reduzindo riscos (NATIONAL INSTRUMENTS, 2012b).

As diversas abordagens são aplicáveis aos vários níveis do projeto de um sistema embarcado, desde a concepção de um produto à implementação do *hardware* e do *software* necessários ao seu controle. Dentre tais abordagens, a Teoria de Controle Supervisório (TCS) (RAMADGE; WONHAM, 1989) apresenta um procedimento formal de obtenção de um controlador, denominado supervisor, que atua de forma minimamente restritiva e garante o atendimento às especificações de controle. Apesar do processo formal de síntese do controle, sua implementação não é trivial (FABIAN; HELLGREN, 1998). Dessa forma, são encontradas na literatura algumas referências para definição de adequadas estruturas de controle e estudos relacionados aos problemas de implementação das quais destacam-se (QUEIROZ, 2004; VIEIRA et al., 2006; LEAL et al., 2012; DIETRICH et al., 2001; MALIK, 2002; FLORDAL et al., 2007; BASILE; CHIACCHIO, 2007).

Na linha de desenvolvimento de sistemas embarcados para linha branca (refrigeradores, condicionadores de ar, lava-roupas) especificamente encontram-se aplicações da TCS a exemplo de Wood (2005) que implementa o controle para uma máquina de lavar roupas. A implementação no referido trabalho é baseada na linguagem de programação *assembly*, que não apresenta fácil portabilidade para outros microcontroladores. Linguagens como o C (KERNIGHAN; RITCHIE, 1988) além de possuírem melhor portabilidade dentre os compiladores, ao se utilizar padrões como definidos pela *American National Standards Institute* (ANSI), auxiliam na legibilidade e manutenção de código, importantes pontos levados em consideração no desenvolvimento de *software*. Destaca-se nesse quesito o trabalho de Teixeira (2008), uma aplicação da TCS para o controle de refrigeradores.

1.1 Objetivos Gerais e Específicos

Com base no exposto, o objetivo geral do presente trabalho é apresentar um método baseado na abordagem modular local de síntese de supervisores (QUEIROZ, 2000) para o projeto de controle em sistemas embarcados. O procedimento formal de síntese de supervisores auxilia na identificação de possíveis defeitos no sistema de controle ainda nas fases iniciais

de desenvolvimento. Tal método incorpora a análise de propriedades em supervisores as quais indicam possíveis problemas na etapa de implementação. A estrutura de controle proposta possui características de implementação para a solução dos problemas identificados de acordo com as propriedades atendidas pelos supervisores, tais como o problema da *causalidade*, da *sincronização inexata* e da *escolha* (FABIAN; HELLGREN, 1998).

Com intuito de aplicação do método no meio industrial propõe-se uma ferramenta para auxílio à implementação. Esta implementação apresenta uma estrutura modular e genérica em linguagem ANSI C para facilitar a portabilidade para diferentes plataformas de microcontroladores e aplicações. Ainda, o método considera uma etapa de validação para uso dos testes realizados industrialmente a fim de identificar se o sistema de controle está aprovado dando prosseguimento ao processo de liberação de um produto ao mercado.

1.2 Organização do Documento

O documento está organizado da seguinte maneira. No Capítulo 2 apresenta-se uma revisão da literatura abordando os principais conceitos e embasamento teórico em relação à Teoria de Controle Supervisório de Sistemas a Eventos Discretos. No Capítulo 3 faz-se um levantamento bibliográfico acerca da implementação do controle supervisório. Destacam-se as principais referências e identifica-se o estado da arte. Posteriormente, no Capítulo 4, apresenta-se o método de projeto para sistemas embarcados e a arquitetura de implementação desenvolvida. O Capítulo 5 é dedicado a apresentação de um estudo de caso elaborado para exemplificação da aplicação do método e do detalhamento da implementação em linguagem ANSI C adotando-se estratégias de modularidade para reuso de código. A mesma abordagem de implementação é utilizada no Capítulo 6 para o controle de temperatura em um refrigerador comercial, apresentando-se os resultados de testes de validação, funcionamento do produto em conformidade com as especificações de controle e comparações acerca do consumo de memória. As considerações finais e perspectivas para trabalhos futuros são apresentadas no Capítulo 7.

Capítulo 2

Revisão Bibliográfica da Teoria de Controle Supervisório

Este capítulo destina-se a apresentar uma breve revisão dos principais conceitos da Teoria de Controle Supervisório (TCS). Tais conhecimentos são essenciais para o melhor entendimento das contribuições apresentadas ao longo deste trabalho e, portanto, para maior aprofundamento as seguintes referências são indicadas: (RAMADGE; WONHAM, 1989; QUEIROZ, 2000; CURY, 2001) e (CUNHA, 2003).

2.1 Sistemas a Eventos Discretos

Um Sistema a Eventos Discretos (SED) é definido como um sistema dinâmico no qual eventos ocorridos em intervalos de tempos irregulares ou desconhecidos determinam sua evolução (CURY, 2001). Por eventos entende-se qualquer acontecimento ocorrido no sistema. Exemplos de eventos são o acionamento e desligamento de um motor elétrico, o início e término de operação de uma máquina ou ainda o apertar de um botão realizado por um usuário.

A condição do sistema entre a ocorrência de eventos é denominada de estado (CURY, 2001). A ocorrência de eventos resulta em uma transição ou evolução dentro do espaço de estados do sistema. Portanto, a simples passagem do tempo não determina a mudança de estados, mas sim a ocorrência de eventos (CURY, 2001). O estado inicial em um SED é especialmente considerado por se tratar da condição em que nenhum evento ainda ocorreu. É possível que um sistema retorne ao seu estado inicial, esse processo é denominado de reinicialização (CURY, 2001).

Na modelagem de um SED não importa o momento da ocorrência de eventos, mas sim a ordem em que estes ocorrem. Além disso, considera-se que não há ocorrência simultânea de eventos (PENA, 2007). A modelagem de um SED é especialmente importante para conhecer o comportamento do mesmo e, a partir desta, estabelecer uma lógica de controle para garantia de

um fluxo desejável de eventos (PENA, 2007).

Segundo Brandin (1996) são encontrados exemplos de aplicações do controle de SEDs em diversas áreas, tais como na manufatura, robótica, redes de computadores e de comunicação, tráfego e logística. São também observadas aplicações em sistemas embarcados (HUBBARD, 2000), no controle de eletrodomésticos (WOOD, 2005) e mais recentemente na área de biomédica (THEUNISSEN et al., 2009).

De acordo com Teixeira (2008), diversos modelos para SEDs são encontrados para fins de verificação, controle ou análise de desempenho. Uma abordagem para o controle de SEDs é proposta por Ramadge e Wonham (1989). Nessa utilizam-se modelos baseados na teoria de linguagens e autômatos e apresenta-se um procedimento de síntese de controladores. Os conceitos fundamentais desta modelagem são apresentados nas seções a seguir.

2.2 Linguagens e Operações sobre Linguagens

Denomina-se alfabeto um conjunto finito Σ de símbolos distintos. O conjunto Σ^* é formado por todas as cadeias finitas de símbolos, também chamadas de sequências ou palavras, da forma $\sigma_1\sigma_2\ldots\sigma_k$, com $\sigma_i \in \Sigma$, onde $i \in \{1,2,\ldots\}$, que podem ser formados a partir de Σ, além de uma palavra de comprimento nulo chamada de cadeia vazia e denotada por ε. Uma linguagem sobre Σ é um subconjunto de Σ^* (PENA, 2007).

No contexto de um SED o conjunto formado por seus eventos é seu alfabeto Σ, sendo que este SED pode ser representado por uma linguagem $L \subseteq \Sigma^*$ descrevendo as sequências (cadeias) factíveis de ocorrer dentro do total de possibilidades (SILVA, 2010).

Sendo uma linguagem um conjunto de palavras, as propriedades matemáticas da teoria de conjuntos são aplicáveis (CRUZ, 2011). Na utilização de linguagens também devem ser consideradas operações sobre as mesmas, das quais as principais são relacionadas adiante.

2.2.1 Concatenação

À operação de concatenação para duas linguagens L_1 e L_2 definidas sobre Σ, denotada por L_1L_2, é definida como $L_1L_2 = \{s_1s_2 \in \Sigma^* : (s_1 \in L_1)e(s_2 \in L_2)\}$.

O *prefixo* de uma sequência $s \in \Sigma^*$ é qualquer sequência $t \in \Sigma^*$ que concatenada a uma segunda sequência $u \in \Sigma^*$ forma a sequência $s = tu$, sendo u o *sufixo* de s. O fato de t ser

o *prefixo* de s é denotado por $t \leq s$.

Outra nomenclatura obtida a partir de concatenações é o entrelaçamento entre cadeias. Um entrelaçamento das sequências s_1,$s_2 \in \Sigma^*$ é a sequência obtida pela concatenação do prefixo de uma das cadeias com o prefixo da outra, concatenadas aos sufixos restantes. O conjunto de todos os entrelaçamentos é denotado por $s_1|||s_2$.

2.2.2 Prefixo-Fechamento

À operação que obtém o conjunto formado por todos os prefixos das palavras de uma linguagem L sobre um alfabeto Σ dá-se o nome de *prefixo-fechamento*, denotada por $\overline{L}$. Dessa forma, $\overline{L} = \{t \in \Sigma^* : (\exists s \in L) t \leq s\}$. Uma linguagem é dita prefixo-fechada quando $L = \overline{L}$.

2.2.3 Projeção

Segundo Cunha (2003) dados um alfabeto Σ e um alfabeto $\Sigma_i \subseteq \Sigma$, a projeção de palavras sobre Σ em palavras sobre Σ_i é uma função recursivamente definida por:

$$p_i(\epsilon) = \epsilon$$

$$p_i(\sigma) = \begin{cases} \epsilon, & \text{se } \sigma \notin \Sigma_i \\ \sigma, & \text{se } \sigma \in \Sigma_i \end{cases}, \text{ para } \sigma \in \Sigma$$

$$p_i(s\sigma) = p_i(s)p_i(\sigma), \text{ para } \sigma \in \Sigma \text{ e } s \in \Sigma^*$$

A ação da projeção p_i sobre a palavra s é apagar todas as ocorrências de símbolos σ que não estejam em Σ_i, resultando uma palavra $p_i(s) \in \Sigma_i^*$. A imagem de uma linguagem $L \subseteq \Sigma^*$ pela projeção p_i é definida por $p_i(L) = \{p_i(s) \in \Sigma_i^* : s \in L\}$ e a imagem inversa de uma linguagem $L_i \subseteq \Sigma_i^*$ pela projeção p_i é definida por $p_i^{-1}(L_i) = \{s \in \Sigma^* : (\exists t \in L_i) p_i(s) = t\}$ (CUNHA, 2003).

2.2.4 Produto Síncrono

Dados dois alfabetos Σ_1 e Σ_2, tal que $\Sigma_1 \cap \Sigma_2 \neq \emptyset$, e dadas duas linguagens $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$. Define-se o *produto síncrono* de L_1 com L_2, denotado $L_1||L_2$, como sendo uma linguagem sobre $\Sigma = \Sigma_1 \cup \Sigma_2$ definida por $L_1||L_2 = p_1^{-1}(L_1) \cap p_2^{-1}(L_2)$, com $p_i^{-1}(L_i)$ sendo a imagem inversa

da linguagem $L_i \subseteq \Sigma_i^*$ (CUNHA, 2003). A composição síncrona de n linguagens neste trabalho é denotada por $\|_{i=1}^{n} L_i$.

2.3 Linguagens e Autômatos como modelos para SED

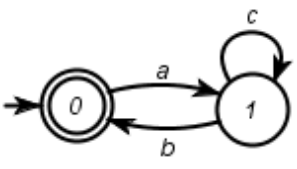
Dado um SED descrito por uma linguagem regular, esta pode ser gerada por um dispositivo chamado autômato (PENA, 2007). Um autômato determinístico de estados finitos A sobre um alfabeto Σ é definido por uma quíntupla $A = (Q,\Sigma,q_0,Q_m,\delta)$, sendo Q um conjunto de estados, Σ um alfabeto finito de eventos, $q_0 \in Q$ um estado inicial, $Q_m \subseteq Q$ um conjunto de estados marcados, e $\delta : Q \times \Sigma \rightarrow Q$ uma função parcial de transição de estados, portanto não definida em todos os estados de Q para todos os eventos de Σ (CURY, 2001).

A função parcial de transição δ, definida para um evento $\sigma \in \Sigma$, pode ser estendida para uma cadeia $s \in \Sigma^*$ como $\widehat{\delta} : Q \times \Sigma^* \rightarrow Q$, tal que $\widehat{\delta}(q,\varepsilon) = q$, $\widehat{\delta}(q,\sigma) = \delta(q,\sigma)$ e $\widehat{\delta}(q,s\sigma) = \delta(\widehat{\delta}(q,s),\sigma)$, quando $\delta(q,\sigma)$, $\widehat{\delta}(q,s)$ e $\delta(\widehat{\delta}(q,s),\sigma)$ são definidas. Essa função mapeia o estado em Q para o qual um SED transita após a ocorrência de uma cadeia s (TEIXEIRA, 2008).

Ao autômato A estão associadas duas linguagens, a linguagem gerada $L(A) = \{s \in \Sigma^* : \widehat{\delta}(q_0,s)$ é definida$\}$ e a linguagem marcada $L_m(A) = \{s \in L(A) : \widehat{\delta}(q_0,s) \in Q_m\}$. A linguagem gerada $L(A)$ representa todas as sequências de símbolos que podem ser seguidas no autômato partindo do estado inicial. A linguagem marcada $L_m(A)$ representa todas as sequências de símbolos que, partindo do estado inicial, alcançam um estado marcado. De acordo com Cury (2001), no contexto de um SED modelado por um autômato, $L(A)$ é o comportamento gerado pelo sistema e $L_m(A)$ o comportamento marcado ou conjunto de tarefas completas no sistema.

Um autômato é representado por um grafo dirigido ou diagrama de transição de estados onde os nós representam estados e arcos etiquetados denotam os eventos. Estados marcados são representados com dois círculos concêntricos e o estado inicial indicado por uma seta, conforme a Figura 2.1.

Figura 2.1: Representação de um autômato por um grafo dirigido



Fonte: produção do próprio autor

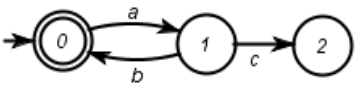
Denota-se $\Sigma(q)$ o conjunto de eventos ativos no estado q de um autômato. Por exemplo, para o autômato da Figura 2.1 $\Sigma(0) = \{a\}$ e $\Sigma(1) = \{b,c\}$.

2.3.1 Bloqueio em um SED

Um SED com comportamento representado pelas linguagens $L(A)$ e $L_m(A)$ é dito ser *não bloqueante*, se satisfaz a condição $L(A) = \overline{L_m(A)}$ (CURY, 2001). Portanto, existindo uma cadeia gerada pelo sistema ($s \in L(A)$), a partir da qual o sistema não completa uma tarefa ($s \notin \overline{L_m(A)}$), o SED apresenta *bloqueio*.

O autômato da Figura 2.2 modela um SED bloqueante. A partir da ocorrência do evento *c* o sistema encontra-se em uma situação de bloqueio pois no estado *2* não se completa uma tarefa.

Figura 2.2: Autômato bloqueante



Fonte: produção do próprio autor

2.4 Teoria de Controle Supervisório

Dado o autômato de estado finitos G, definindo o comportamento de um SED, e E um autômato para as especificações de controle, a Teoria de Controle Supervisório (TCS) provê um procedimento para síntese automática de um supervisor S não bloqueante e minimamente restritivo tal que o sistema controlado atenda às especificações E (RAMADGE; WONHAM, 1989).

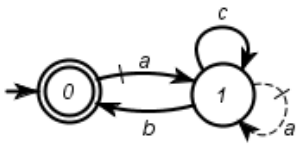
A planta G modela todo o comportamento livre do sistema em malha aberta, incluindo aqueles indesejáveis no sistema controlado, e de forma geral é formado por um conjunto de n subplantas G_i atuando paralelamente para realizar uma dada tarefa. Cada subplanta G_i possui um comportamento livre individual e a composição síncrona de todas as subplantas define o comportamento global $G = \|_{i=1}^{n} G_i$.

O conjunto dos possíveis eventos, portanto o alfabeto de G, é denotado por Σ. Na TCS o conjunto de eventos é subdividido em dois conjuntos disjuntos, um de eventos controláveis Σ_c e um conjunto de eventos não controláveis Σ_u de forma que $\Sigma = \Sigma_c \cup \Sigma_u$. Eventos controláveis são aqueles possíveis de serem desabilitados pela ação de controle, em oposição aos

não controláveis que estão permanentemente habilitados (CURY, 2001). Normalmente, comandos dados ao sistema são modelados como eventos controláveis e resposta recebidas do sistema associadas à ocorrência de eventos não controláveis. Tais eventos presentes em SEDs, na TCS, são gerados espontaneamente pela planta e observados por um supervisor, capaz de gerar uma ação de controle de desabilitação de eventos controláveis.

Autômatos utilizados como modelos na TCS representam as transições diferentemente de acordo com a controlabilidade dos eventos. Um traço em um arco indica que o evento da referida transição é controlável. Neste trabalho, representam-se eventos (controláveis) desabilitados em um estado por um arco em linhas tracejadas partindo e retornando ao estado de desabilitação, como o evento *a* no estado *1* do autômato da Figura 2.3.

Figura 2.3: Um autômato como modelo na Teoria de Controle Supervisório



Fonte: produção do próprio autor

O conjunto dos eventos desabilitados no estado q do autômato é representado por $\Phi(q)$. Tem-se que, para o exemplo, $\Phi(1) = \{a\}$. O operador $\setminus$ é utilizado para denotar a remoção de eventos de um determinado conjunto de eventos, o resultado é o mesmo que uma operação de diferença entre conjuntos. Por exemplo, o resultado da operação $\Sigma(0) \setminus \Phi(1)$ é um conjunto vazio.

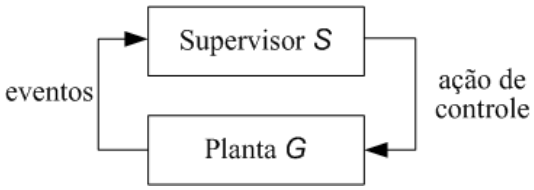
Ramadge e Wonham (1989) introduzem o procedimento para a síntese de um supervisor para um SED no qual a planta é formalmente definida por duas linguagens, uma linguagem gerada $L(G) \subseteq \Sigma^*$, e uma linguagem marcada $L_m(G) \subseteq L(G)$ contendo as conclusões de tarefas do sistema. As abordagens para a obtenção dos supervisores são denominadas de abordagem *monolítica*, *modular* e *modular local*. Na abordagem modular (RAMADGE; WONHAM, 1989) explora-se a modularidade das especificações de controle. Uma evolução na técnica modular, desenvolvida por Queiroz (2000) explora a modularidade das especificações de controle e das plantas sob ação dos supervisores modulares, método conhecido como a abordagem *modular local* (QUEIROZ, 2000). Os autômatos dos supervisores obtidos através da técnica modular local são normalmente menores em número de estados em relação aos da técnica modular, pois o modelo de planta utilizado para cálculo dos supervisores modulares contempla apenas a parte da planta afetada pela ação do supervisor.

A seguir são apresentadas as características da síntese de supervisores por intermédio das abordagens monolítica e modular local. A abordagem modular clássica não é discutida por não ser objeto de estudo deste trabalho.

2.4.1 Abordagem Monolítica

Nesta abordagem um único supervisor S é sintetizado para o controle da planta G, sendo que o supervisor S tem a capacidade de, em qualquer um de seus estados $q \in Q$, definir uma *ação de controle* a fim de desabilitar uma série de eventos controláveis pertencentes à Σ_c, conforme ilustra a Figura 2.4. Em malha fechada, o supervisor S observa a ocorrência de eventos gerados espontaneamente por G, a ação de controle do supervisor é notadamente passiva, não forçando a ocorrência de um determinado evento em G, apenas desabilitando eventos controláveis. O supervisor S é dito *controlável* em relação a planta G se, e apenas se, é sempre capaz de seguir eventos não controláveis ocorridos na planta G (CURY, 2001).

Figura 2.4: Ação de controle de um Supervisor Monolítico



Fonte: Adaptado de Cury (2001)

O sistema em malha fechada S/G, com o supervisor S atuando sobre a planta G, pode ser descrito pelo produto síncrono entre S e G, isto é $S||G$. De fato, no produto síncrono $S||G$ apenas as transições possíveis em ambos S e G são passíveis de ocorrência, então o comportamento em malha fechada é tal que $L(S/G) = L(S||G)$ e $L_m(S/G) = L_m(S||G)$ (CURY, 2001).

Segundo Pena (2007) dada uma linguagem $K \subseteq \Sigma^*$ esta é *controlável* em relação à $L(G)$ se $\overline{K}\Sigma_u \cap L(G) \subseteq K$. Propriedade que garante que o supervisor S que implementa a linguagem K não tenta desabilitar eventos não controláveis na planta.

A fim de restringir o comportamento livre da planta G, um autômato E é definido. Geralmente E é obtido pela composição síncrona de um conjunto de m *especificações de controle* de forma que $E = \|_{i=1}^{m} E_i$. O comportamento da planta G que atende às especificações de controle é denotado por uma *linguagem alvo* $K = E||L_m(G)$. De acordo com Cury (2001),

dada a planta G, com comportamento ($L(G)$,$L_m(G)$) e $K \subseteq L_m(G)$ existe um supervisor S não bloqueante para G tal que $L_m(S/G) = K$ se, e somente se, K for controlável.

Caso K não seja controlável, existe o conjunto não vazio de sublinguagens controláveis de K definido como $C(K,L(G)) = \{J : J \subseteq K$ com J controlável em relação à $G\}$. Ramadge e Wonham (1989) apresentam o procedimento para obtenção da máxima (suprema) linguagem controlável $SupC(K,L(G))$ contida em K. O supervisor S representando tal linguagem é dito *ótimo* no sentido de que S/G é o comportamento minimamente restritivo e não bloqueante que satisfaz a especificação E.

2.4.2 Abordagem Modular Local

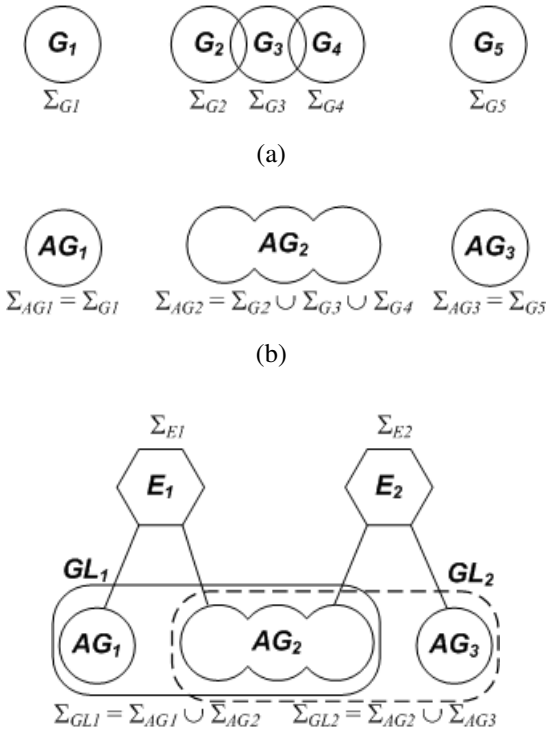
Para um SED de grande porte composto por diversos modelos para a planta, a abordagem modular local reduz consideravelmente a explosão do número de estados que tipicamente ocorre na síntese monolítica de supervisores (TEIXEIRA, 2008). Nesta metodologia um supervisor local SL_i é obtido para cada especificação (ou conjunto de especificações) E_i, levando em consideração apenas a planta local GL_i. Desta forma, a complexidade computacional associada a esta síntese e o tamanho dos supervisores são reduzidos explorando a modularidade das especificações e da estrutura descentralizada da composição das subplantas (QUEIROZ, 2000). A abordagem leva a melhores resultados quanto mais refinada for a Representação por Sistema Produto (RSP) (PENA, 2007). Nessa, dados os modelos de cada subplanta, conceitualmente representados na Figura 2.6a, apenas as subplantas síncronas, ou seja, aquelas que apresentam eventos em comum, são compostas formando um conjunto completamente assíncrono de subplantas AG_j (Figura 2.6b).

Por intermédio da RSP mais refinada de uma planta G definida sob um alfabeto Σ, forma-se o conjunto de subplantas assíncronas AG_j com alfabeto Σ_{AG_j} e $j \in J = \{1,2,\ldots n\}$. Dada uma especificação de controle E_i com alfabeto Σ_{E_i}, a *planta local* é obtida pela composição das subplantas assíncronas as quais possuem eventos em comum com a especificação, como representado na Figura 2.6c. Portanto, $GL_i = \|_{k \in N} AG_k$ com a definição do conjunto de índices N dado por $N = \{k \in J | (\Sigma_{AG_k} \cap \Sigma_{E_i}) \neq \emptyset\}$.

O comportamento da planta local GL_i que satisfaz a especificação E_i é representado pela linguagem alvo $K_i = E_i || L_m(GL_i)$, e os supervisores modulares SL_i, são obtidos de forma que $L_m(SL_i/GL_i) = SupC(K_i,L(GL_i))$.

Considerando o exemplo para dois supervisores locais, a atuação dos supervisores

Figura 2.5: Abordagem modular local: (a) Subplantas (b) Representação por Sistema Produto (c) Plantas locais



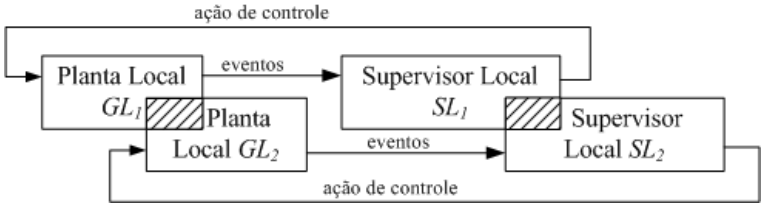
Fonte: Adaptado de Teixeira (2008)

sob as correspondentes plantas locais é ilustrada na Figura 2.6. Determinado supervisor local atua com uma ação de controle sobre uma parcela da planta, a qual pode ser parcialmente compartilhada com um segundo supervisor local. Dessa forma, é necessário verificar se a ação conjunta de tais supervisores não levará a um bloqueio na planta sob supervisão devido às desabilitações realizadas. Ao sintetizar supervisores pela abordagem modular local realiza-se a verificação da *condição de modularidade* dada por $\|_{i=1}^{n}\overline{L_m(SL_i/GL_i)} = \overline{\|_{i=1}^{n}L_m(SL_i/GL_i)}$ (QUEIROZ, 2000). A condição de modularidade garante que a malha fechada do sistema sob ação dos supervisores seja não bloqueante. Na prática, a modularidade pode ser verificada quando a composição síncrona dos supervisores modulares locais $\|_{i=1}^{n}SL_i$ é *trim* (CURY, 2001), ou seja, não contém estados não-acessíveis ou não-coacessíveis, portanto não havendo bloqueio.

Na abordagem modular local projetam-se supervisores cada especificação de controle. O resultado da ação conjunta dos supervisores locais é equivalente a atuação de supervisor único sintetizado através da abordagem monolítica.

Ainda, a abordagem modular local explora a estrutura do sistema produto, buscando

Figura 2.6: Ações de controle de Supervisores Modulares Locais



Fonte: Adaptado de Queiroz (2000)

compôr o mínimo de modelos de subsistemas necessários para obtenção de cada supervisor. Segundo Queiroz (2000), isso reduz consideravelmente a explosão de estados no processo de modelagem e síntese dos supervisores. Em sistemas de grande porte, maior ainda é o número de subsistemas a serem modelados, e utilizando do refinamento do sistema produto, a abordagem modular local apresenta melhores resultados. Por estas vantagens, neste trabalho é dado maior foco no uso da abordagem modular local.

Capítulo 3

Revisão sobre Implementação de Controle Supervisório

Neste capítulo apresenta-se uma revisão de aspectos de implementação da estrutura de controle supervisório encontrados na literatura. Fazem-se comparativos entre as abordagens e as características de similaridade entre implementações voltadas ao controle de sistemas de manufatura, na maioria controladas via Controladores Lógicos Programáveis (CLPs), com implementações em microcontroladores, portanto aplicáveis ao controle em sistemas embarcados.

3.1 Implementação do Controle Supervisório

A Teoria de Controle Supervisório TCS (RAMADGE; WONHAM, 1989) fornece supervisores minimamente restritivos não bloqueantes cuja ação de controle atende às especificações de controle. Isso significa que o controle realizado pelo supervisor permite que a planta atue da maneira mais livre possível. Quando o controle sobre um sistema é obtido sem uma abordagem formal, é natural que este atue na planta de uma maneira mais restritiva (QUEIROZ, 2004). Por maior que seja a experiência de um projetista desenvolvendo a lógica de controle, há tendências de que a abordagem empírica reduza as possibilidades ou opções que a planta poderia gerar, pelo fato de que normalmente o foco seja dado ao cálculo de obtenção do controle. Portanto, ao utilizar-se da TCS, quanto mais fiel a implementação do controle estiver em relação ao comportamento modelado pelo supervisor, maior a tendência de se permitir que a planta execute as tarefas de maneira livre sem que haja prejuízo do controle sob a mesma (QUEIROZ, 2004; BASILE; CHIACCHIO, 2007).

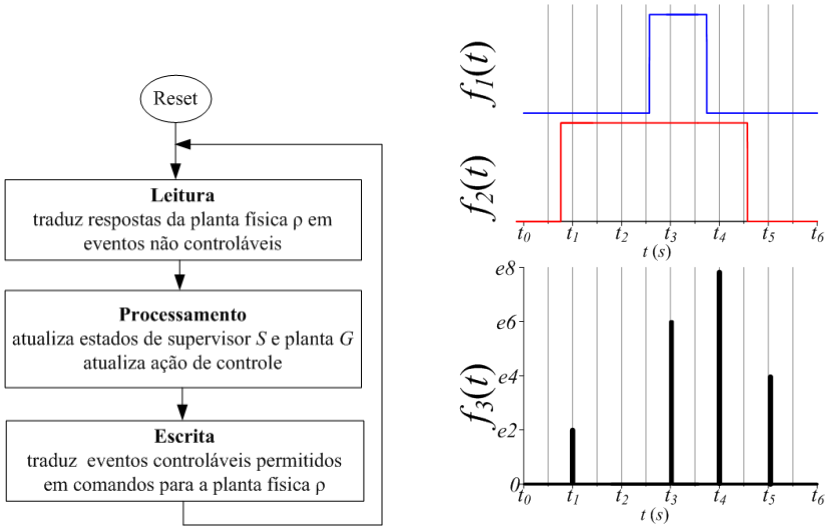
A utilização do procedimento formal da TCS no projeto do controle para SED, baseada na consistência dos modelos concebidos, auxilia na identificação de erros ainda durante a fase de desenvolvimento do projeto, aumentando a confiabilidade das soluções (TEIXEIRA, 2008).

Contudo, a conversão de supervisores em controladores tem sido tema de diversos trabalhos por haverem diferenças entre as hipóteses por parte da teoria e o funcionamento dos dispositivos de controle para os quais tais controladores são projetados (FABIAN; HELLGREN, 1998). Primeiramente, enquanto que no controle supervisório de um SED o comportamento em malha fechada é *síncrono* entre planta e supervisor, na sua implementação, sendo esta em Controlador Lógico Programável (CLP), microcontrolador ou um Computador Pessoal (PC), esta sincronização imediata não é satisfeita. Devido a características de funcionamento e operação existem atrasos inerentes ao processamento sequencial e comunicação entre controlador e planta física. Um segundo ponto de divergência ressaltado na literatura é que enquanto na TCS assume-se que a ocorrência de eventos é espontaneamente determinada pela planta, na sua implementação, por consequência das características dos dispositivos, ações de controle devem ser tomadas pelo controlador, em alguns casos inclusive decidindo entre vários possíveis eventos controláveis habilitados.

A tarefa de um supervisor *S* é, na ocorrência de eventos na planta *G*, evoluir (transitar de estado quando aplicável) em sincronismo com a planta e executar a ação de desabilitação de eventos controláveis de acordo com a característica da composição síncrona, na qual apenas eventos habilitados em ambos os autômatos *S* e *G* são possíveis no autômato $S||G$. Como eventos não controláveis são provenientes de respostas de uma planta física ρ, enquanto eventos controláveis causam ações em ρ, a solução usualmente adotada é a verificação cíclica por parte do controlador, levando um período T_{ciclo} de execução, em três passos básicos (Figura 3.2a): *(i) Leitura:* tradução das respostas da planta física ρ em eventos não controláveis, *(ii) Processamento*: atualização dos estados do supervisor *S* e da planta *G* e atualização da ação de controle, *(iii) Escrita:* tradução de eventos controláveis permitidos em comandos para a planta física ρ.

Abstraindo-se os detalhes de implementação, o comportamento cíclico de *leitura-processamento-escrita* é observado independentemente do dispositivo de controle utilizado, como por exemplo por Queiroz e Cury (2002), Vieira et al. (2006) e Leal et al. (2012) para aplicações em CLP e para aplicações em microcontrolador por Costa (2005) e Teixeira (2008). Segundo Carvalho (2007) existem diferenças na forma como os dispositivos acionam as saídas e leem as entradas, já que em um CLP os níveis lógicos são atualizados no meio físico (planta ρ) de uma só vez, enquanto o microcontrolador os efetua sequencialmente. Portanto, é importante destacar que o ciclo de execução da Figura 3.2a refere-se ao funcionamento lógico do algoritmo de controle, e as etapas de leitura e saídas não dizem respeito somente aos sinais físicos. A exemplo disso, analisando os fluxogramas de implementação propostos por Carvalho (2007)

Figura 3.1: Dinâmica de processamento do controle: (a) ciclo de execução (b) monitoramento periódico de sinais de resposta e tradução em eventos não controláveis



(a) (b)

Fonte: produção do próprio autor

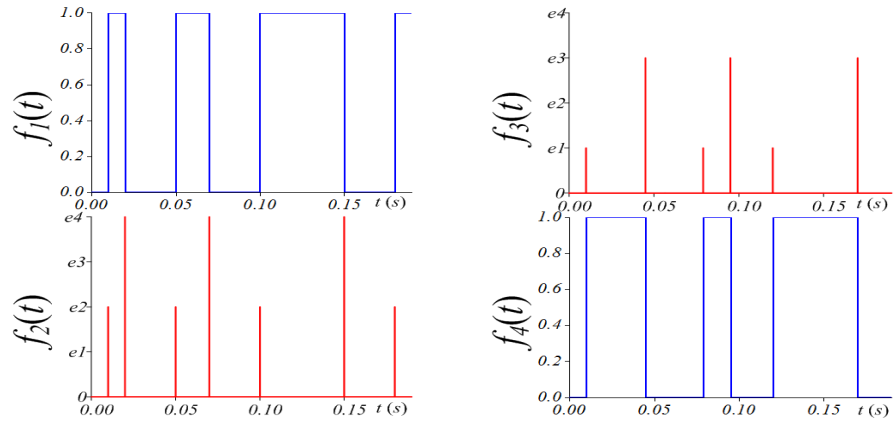
para microcontroladores, verifica-se que durante o *processamento* para atualização de estados e desabilitação, não são realizadas *leituras* ou *acionamentos*, apenas tratados eventos já ocorridos.

A detecção das mudanças de nível nos sinais de resposta é realizada pelo monitoramento periódico do sinal e comparação entre duas leituras consecutivas, detectando-se as bordas de subida e descida. A Figura 3.2b representa sinais de entrada $f_1(t)$ e $f_2(t)$ em um sistema hipotético com uma determinada taxa de leitura. Devido à mudança do nível no sinal de $f_2(t)$ entre t_0 e $t = t_1$, há ocorrência do evento *e2* no ciclo de leitura em $t = t_1$. Na Figura 3.2b eventos em $f_3(t)$ são simbolizados por traços de duração instantânea e com a amplitude definindo o evento de ocorrência.

De acordo com Fabian e Hellgren (1998) um sinal aplicado a um *CLP* é traduzido no componente mais elementar que assume um valor binário (*booleano*) "zero"ou "um". A mesma constatação pode estendida a microcontroladores, no qual o componente elementar é um *bit*. Por outro lado, na TCS supervisores transitam de estados baseados na ocorrência assíncrona e instantânea de eventos (símbolos).

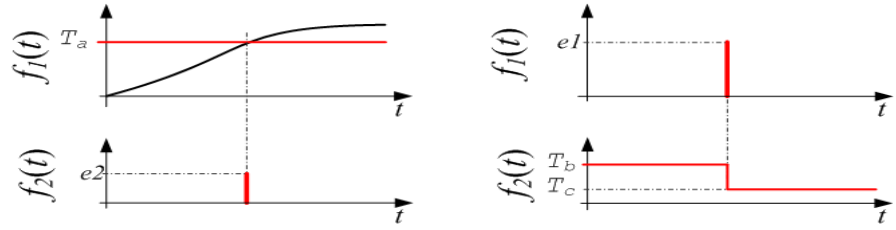
Ao longo do tempo informações binárias podem ser interpretadas como sinais binários, a exemplo de $f_1(t)$ na Figura 3.3a. Assim, para relação dos contextos de sinais e eventos, as alterações de nível no sinal binário de resposta do sinal de entrada $f_1(t)$ são traduzidas em eventos não controláveis na Figura 3.3a. Nessa, a detecção da borda de subida é convertida para

Figura 3.2: Conversões entre sinais e eventos



(a) Conversão sinal binário em eventos

(b) Conversão eventos em sinal binário



(c) Conversão sinal contínuo em evento

(d) Conversão evento em sinal contínuo

Fonte: produção do próprio autor

o evento *e2*. Já as mudanças de nível lógico "um"para "zero"são convertidas no evento *e4*. De forma análoga, os eventos controláveis *e1* e *e3*, na Figura 3.3b, são convertidos em alterações no sinal binário de comando (saída) $f_4(t)$, gerando respectivamente as mudanças de subida e de descida de nível.

Outros tipos de sinais também podem ser considerados tais como um sinal de resposta da contagem de um temporizador ou o valor decorrente de conversões Analógico-Digital (AD) como apresentado por Teixeira (2008). Nesse caso, utiliza-se a identificação do cruzamento do valor por um limiar (*threshold*) específico. Para sinalizar a ocorrência de um evento, por exemplo na Figura 3.3c, o cruzamento pelo limiar T_a define a ocorrência do evento não controlável *e2*. Eventos controláveis também podem gerar uma mudança de nível em um sinal contínuo, como na Figura 3.3d, na qual o evento *e1* modifica o valor limite T_b para um nível inferior T_c.

As diferenças entre sinais e eventos, características síncronas da TCS em relação à execução cíclica dos dispositivos de controle, e a já mencionada contextualização da espontaneidade de eventos, levam a estudos acerca da estrutura de controle supervisório. Trabalhos como o de Fabian e Hellgren (1998) apontam características nos modelos de supervisores que podem

gerar problemas de acordo com arquiteturas de implementação. Tais problemas são revisados e discutidos nas seções a seguir.

3.1.1 Causalidade

Segundo Fabian e Hellgren (1998) o problema da *Causalidade* trata da relação entre a geração espontânea de eventos por parte da planta, como assumido pela TCS (RAMADGE; WONHAM, 1989), e a necessidade de atuação do controlador na geração de eventos em alguns sistemas. O problema é definir a natureza dos eventos, em termos da controlabilidade e de quem é responsável pela sua geração, controlador ou planta física.

Os autores reportam a análise previamente realizada por Balemi (1992), o qual introduz o conceito de sistema de *entrada/saída* para o controle de um SED. A diferença nos conceitos reside essencialmente na origem determinada para os eventos. Na obra de Balemi (1992) eventos não controláveis são sempre respostas provenientes da planta física, por outro lado eventos controláveis são sempre comandos fornecidos pelo controlador. Já para Fabian e Hellgren (1998) nem todos os eventos não controláveis necessariamente provém da planta física, por exemplo, o início de um temporizador interno ao controlador, e como precisam ser simbolicamente gerados para interpretação do supervisor cabe também ao controlador fazê-lo.

Ao assumir que o controlador gera eventos, é necessário destacar as diferenças entre o controlador ζ (sistema de controle) do supervisor S (modelo do comportamento em malha fechada) (MALIK, 2002). Queiroz (2004) utiliza uma interface para compatibilização entre respostas e comandos da planta física ρ, concentrando a geração dos eventos, os símbolos observados pelo supervisor, nesta interface.

A abordagem utilizando interfaces facilita a percepção de que sob a perspectiva do supervisor a planta é integralmente responsável pela ocorrência de eventos. Os eventos traduzidos pela interface são observados no controlador ζ e, na perspectiva do supervisor *S*, interface e planta *G* são vistos como parte da planta teorizada por Ramadge e Wonham (1989).

A interface proposta por Queiroz (2004) é constituída implicitamente pelos blocos dos modelos de sistema produto e das sequências operacionais. Uma segunda abordagem é adotada por Teixeira (2008), o qual utiliza um bloco lógico explicitamente denominado de interface. Independente da forma como a interface é composta a função lógica permanece a mesma, tendo o objetivo de manter coerência entre a implementação no sistema de controle

físico e as hipóteses estabelecidas pela TCS.

3.1.2 Problema da Escolha - Determinismo

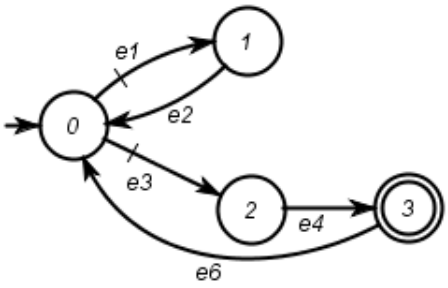
Devido à sua característica de atuação minimamente restritiva, após a atualização de estado, a ação de desabilitação do supervisor *S* pode permitir que mais do que um evento controlável permaneçam habilitados, deixando à cargo da planta a escolha sobre qual destes eventos permitidos será gerado (FABIAN; HELLGREN, 1998).

Quando o controlador, com auxílio de uma interface, necessita gerar eventos, na situação descrita anteriormente é preciso que este defina uma ação de controle, *escolhendo* um evento controlável dentre os que não foram desabilitados pelo supervisor. O problema da escolha consiste na determinação de *qual* evento é executado. Caso a escolha não seja tratada, a execução do controle a fará implicitamente. O problema torna-se especialmente relevante pois dependendo da implementação, pode-se gerar uma condição de bloqueio mesmo sendo o supervisor não bloqueante (DIETRICH et al., 2001).

No intuito de ilustrar o problema da escolha, considere o supervisor não bloqueante da Figura 3.4a. Uma possível implementação em linguagem C (KERNIGHAN; RITCHIE, 1988) para o controle das atualizações de estado do autômato, pode ser baseada no estado atual e na condição de desabilitação dos eventos para retornar por qual evento será realizada a transição. Observa-se na Figura 3.3 que a implementação codificada para o estado *0* retorna a transição sempre por intermédio da ocorrência do evento controlável *e1*. Mesmo que no estado *0* ambos os eventos *e1* e *e3* estejam habilitados, a primeira condição de *e1* não desabilitado faz a função retornar que a transição seja por *e1* e a condição de desabilitação de *e3* nunca é verificada pelo programa. Nesse caso, mesmo o supervisor não sendo bloqueante, a implementação nunca permite que o estado marcado *3* seja alcançado, portanto a condição de conclusão de tarefa não é atingida e o sistema implementado torna-se bloqueante.

Para avaliar se implementações de um determinado supervisor podem gerar condições de bloqueio, como a da exemplificada na Figura 3.3, Dietrich et al. (2001) apresentam uma série de propriedades, que quando satisfeitas, garantem que a implementação pode ser realizada sem preocupações com o problema da escolha. Malik (2002) aborda a questão de escolha sob o tema do *determinismo*, no sentido de que o controlador precisa determinar o evento controlável a ser gerado. Um algoritmo é proposto por Malik (2002) para verificação se, para toda e qualquer escolha, um estado marcado será alcançável.

Figura 3.3: Supervisor com problema de escolha e implementação bloqueante



(a) Supervisor Não Bloqueante

```
unsigned char retornaEventoTransicao(unsigned
    char estado)
{
 if(estado == 0)
 {
  //Verifica se evento e1 esta desabilitado
  if(d_ev_e1 == FALSE)
   return ev_e1; //se permitido retorna e1

  //Verifica se evento e3 esta desabilitado
  if(d_ev_e3 == FALSE)
   return ev_e3; //se permitido retorna e3
 }
 if(estado == 1)
  [...]
}
```

(b) Código em linguagem C de implementação bloqueante

Fonte: produção do próprio autor

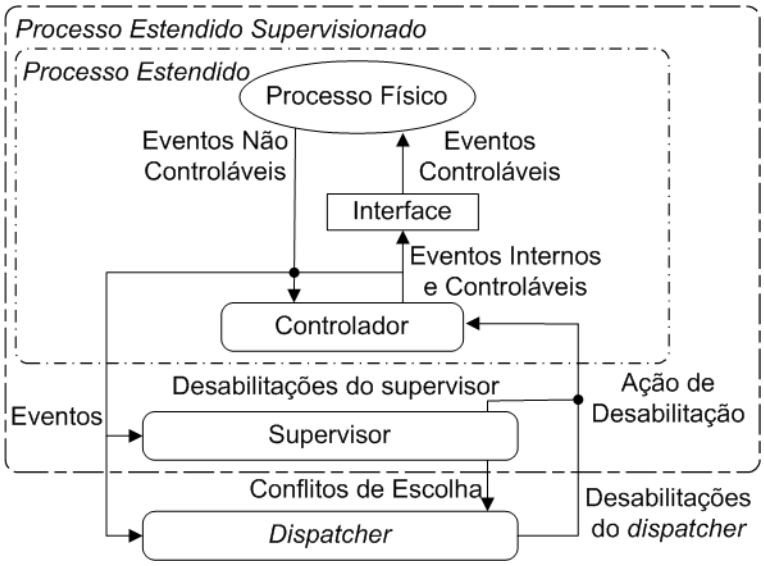
Neste trabalho as propostas encontradas na literatura que contemplam a solução do problema da escolha, são classificadas em soluções *offline* e *online*. Nas soluções por métodos *offline*, como as utilizadas por Huang e Kumar (2008) e Cantarelli e Roussel (2008), um modelo sem problema da escolha é extraído do supervisor. Esse modelo é então utilizado na implementação do controlador, tendo-se como resultado um controlador determinístico com no máximo um evento controlável permitido por estado. Nessa técnica, otimizações podem ou não ser consideradas para determinação dos caminhos definidos para o controlador. Abordagens são classificadas como *online* quando permitem que a escolha do evento seja determinada em tempo de execução, enquanto o controle atua no sistema. Tal técnica é utilizada na proposta de Basile e Chiacchio (2007).

É necessário atentar que mesmo quando o problema não leva a uma implementação bloqueante, pode levar a uma situação em que a escolha define o uso de determinada parte da planta (MALIK, 2002). Mesmo que o estado marcado, no exemplo da Figura 3.4a, seja alterado para o estado *0*, um controlador extraído do supervisor sob essa técnica permite apenas que *e1* ou *e3* esteja presente na implementação. A abordagem *offline* garante que um estado marcado é alcançado pelas implementações, embora possa falhar em utilizar todos os recursos disponíveis, ou seja, passa a restringir desnecessariamente o comportamento da planta, um aspecto contrário ao objetivo de síntese de supervisores da TCS.

A técnica de solução *online* da escolha é identificada claramente na proposta de

Basile e Chiacchio (2007) através da utilização de um dispositivo denominado *dispatcher* (determinador) que observa a ação de controle do supervisor e executa a escolha na forma de complemento às desabilitações de eventos controláveis, conforme representado na Figura 3.4.

Figura 3.4: Atuação do *dispatcher* proposto por Basile e Chiacchio (2007)



Fonte: Adaptado de Basile e Chiacchio (2007)

O método de determinação da escolha é deixado em aberto pelos autores para investigação a cada implementação. Como políticas de escolha pode-se adotar um esquema de prioridade como proposto por Vieira et al. (2006) e Teixeira (2008) ou procedimentos baseados na alteração aleatória de variáveis que definem os eventos controláveis a serem desabilitados como usados por Pinotti et al. (2010), Cruz (2011) e Leal et al. (2012).

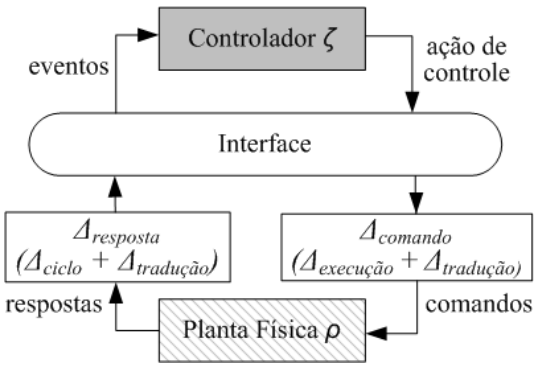
3.1.3 Sincronismo entre Controlador e Planta Física

Um controlador ζ executando o ciclo *leitura-processamento-escrita* com periodicidade T_{ciclo} define a dinâmica e a taxa de execução da interface. De tal maneira, define a geração de eventos controláveis e tradução em transições nos sinais de comando enviados à planta física ρ. De modo análogo, a planta física ρ, de acordo com suas contantes de tempo, gera mudanças de nível nos sinais de resposta. Essas mudanças são traduzidas em informações que passam a estar disponibilizadas para geração de eventos não controláveis enviados ao controlador ζ. A periodicidade de ocorrência das alterações na planta física ρ está relacionada com a controlabilidade do evento. Eventos controláveis, por serem iniciados por intermédio do controlador, tem uma periodicidade

mínima determinada por T_{ciclo}. Já a própria dinâmica da planta física define, para cada evento não controlável, um período mínimo de ocorrência representado por $T_{min\_\sigma_u}$.

Porém, não são apenas as periodicidades que influenciam o sincronismo entre o controlador e planta física. Eventos não controláveis ocorridos na planta física ρ, provocam uma mudança no sinal de resposta podendo este ser traduzido e identificado com um atraso $\Delta_{resposta}$ pelo controlador ζ. Os comandos, partindo do controlador, apresentam um atraso $\Delta_{comando}$ para gerarem a alteração de nível no sinal da planta física ρ como ilustra a Figura 3.5. O atraso $\Delta_{resposta}$ é influenciado por um Δ_{ciclo} e um atraso $\Delta_{traducao}$. O primeiro é decorrente da latência por parte do controlador em interpretar a informação de resposta devido à sua característica cíclica. Δ_{ciclo} está relacionado com T_{ciclo}, mas estes não são necessariamente iguais. Por exemplo, em implementações que interpretam as informações para gerar um evento por ciclo, tal como (VIEIRA, 2003), quando vários eventos precisam ser tratados a informação de um determinado evento estará disponibilizada por mais de um ciclo (T_{ciclo}), embora a transição seja realizada com maior atraso Δ_{ciclo}. A segunda influência no atraso de resposta é o atraso $\Delta_{traducao}$, que engloba atrasos no meio de comunicação e adequação dos sinais provenientes dos sensores para tradução na informação utilizada para interpretação de eventos pelo controlador. Um atraso $\Delta_{traducao}$ também está presente na transferência do sinal de comando para planta, gerando um atraso $\Delta_{comando}$, especialmente quando há protocolos de comunicação envolvidos e não somente sinais elétricos. No caso dos eventos controláveis a geração de evento e comando é realizada dentro do mesmo ciclo. Portanto existe apenas um pequeno tempo de execução entre a etapa de geração do evento controlável e sua correspondente ação no sinal de saída, dada por um atraso $\Delta_{execucao}$.

Figura 3.5: Atrasos entre Controlador e Planta Física presentes na estrutura lógica do controle



Fonte: produção do próprio autor

Segundo Basile e Chiacchio (2007) o atraso Δ_{ciclo} está diretamente relacionado

com o período T_{ciclo} e isto requer que a periodicidade para um evento não controlável σ_u seja tal que $T_{min\_\sigma_u} > T_{ciclo}$. Dessa maneira garante-se que o controlador ζ é capaz de detectar qualquer ocorrência do evento não controlável, pois este não ocorre mais que uma vez no mesmo ciclo. Quando esta condição não é satisfeita, Basile e Chiacchio (2007) propõe a utilização de interrupções (*hardware* ou *software*) para detecção dos sinais de resposta. Em tais casos, um *buffer* para armazenamento de eventos não controláveis ainda não processados é necessário. O tamanho do *buffer* depende da relação entre os períodos, pois quanto maior a diferença na relação $T_{min\_\sigma_u} < T_{ciclo}$ maior será o tamanho do *buffer*.

Os atrasos relacionados ao meio físico representados por $\Delta_{traducao}$ não podem ser eliminados com o uso de interrupções para conversão dos sinais em eventos. Mesmo assim, na maioria das aplicações o meio de comunicação entre controlador e planta física é realizado por sinais elétricos, conexões elétricas que resultam em ínfimos valores de atraso. Quando protocolos de comunicação são utilizados, os principais atrasos são relativos às taxas de transmissão dos protocolos. Esses atrasos são caracterizados por Xu e Kumar (2008) utilizando uma abordagem que os incorpora na modelagem. Entretanto, segundo Malik (2002) essa técnica pode levar a um grande aumento do número de estados do modelo.

Dependendo da periodicidade T_{ciclo} do controlador ζ e da dinâmica da planta física ρ, os atrasos $\Delta_{resposta}$ e $\Delta_{comando}$ podem tornar-se prejudiciais à correta operação do controle supervisório, resultando em dois problemas: *(i)* atualização dos estados dos modelos do supervisor *S* por uma sequência de eventos diferente da ocorrida na planta, e *(ii)* atuação do controlador ζ com uma ação de controle baseada em estados incoerentes com o estado atual da planta física ρ. A consequência de ambos problemas é a falha no *sincronismo* entre o controlador e a planta física (FABIAN; HELLGREN, 1998). Portanto, a correta operação está garantida quando a linguagem em malha fechada do sistema $L(S) = L(S/G)$ satisfaz certas propriedades (FABIAN; HELLGREN, 1998).

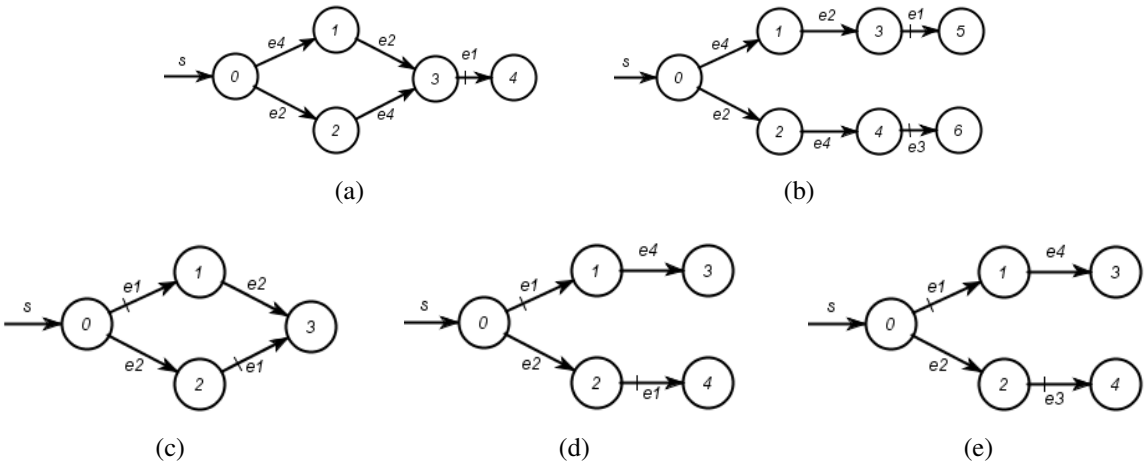
Fabian e Hellgren (1998) apresentam duas propriedades relacionadas aos atrasos e dinâmicas existentes na prática. Um modelo com linguagem de malha fechada atendendo às propriedades da Definição 3.1 e Definição 3.2 tem seu funcionamento teórico garantido mesmo com a presença de qualquer tipo de atraso que ocorra na prática.

Definição 3.1 (Fabian e Hellgren (1998))**.** *Um supervisor S é dito* insensível ao entrelaçamento *de eventos não controláveis com relação a uma planta G (definida sob um alfabeto* $\Sigma = \Sigma_c \cup \Sigma_u$*), se para* $s \in \Sigma^*$*,* s_1,$s_2 \in \Sigma_u$ *e* $\sigma \in \Sigma_c$*.* $ss_1s_2\sigma \in L(S/G) \Rightarrow s(s_1|||s_2)\sigma \subseteq L(S/G)$.

Definição 3.2 (Fabian e Hellgren (1998))**.** *Uma linguagem* $K = L_m(S/G)$ *é dita* insensível ao atraso *se dados* $s \in \overline{K}, \sigma_u \in \Sigma_u$ *e* $\sigma_c \in \Sigma_c$, $s\sigma_c$,$s\sigma_u \in \overline{K} \Rightarrow s\sigma_c\sigma_u$,$s\sigma_u\sigma_c \in \overline{K}$.

A Definição 3.1 afirma que um supervisor S é insensível ao entrelaçamento se após uma sequência s e as cadeias de eventos não controláveis s_1 e s_2 um evento controlável σ estiver habilitado, o mesmo também deverá sempre estar habilitado após a ocorrência de qualquer entrelaçamento das sequências s_1 e s_2. No intuito de exemplificar a propriedade apresentada na Definição 3.1, considere o autômato da Figura 3.7a. Em tal modelo, o mesmo estado é alcançando qualquer que seja a ordem de ocorrência dos eventos *e2* e *e4*. Esta propriedade deve ser satisfeita quando um sistema possui atraso $\Delta_{resposta}$ com a mesma ou maior ordem de grandeza que a dinâmica de processamento T_{ciclo}. Em tal sistema a propriedade é importante pois o processamento pode não ser capaz de distinguir a ordem dos eventos. Por exemplo, caso em um sistema que não atenda a propriedade, como o modelo representado na Figura 3.7b, a identificação da ocorrência dos eventos e_2 e e_4 ocorra no mesmo ciclo, durante a etapa de processamento, o controlador pode transitar para um estado de forma diferente daquela da ocorrida na planta, pois o controlador é incapaz de identificar a ordem dos eventos.

Figura 3.6: Autômatos com linguagens atendendo a diferentes propriedades



Fonte: produção do próprio autor

Quanto aos atrasos Δ_{ciclo} e $\Delta_{comando}$ nenhuma adequação é necessária quando o modelo de malha fechada atende à propriedade da Definição 3.2, tal como a linguagem representada pelo autômato da Figura 3.7c. Mesmo havendo um atraso Δ_{ciclo} na identificação do evento e_2, a atuação do controlador permitindo o evento e_1 estará em concordância com a planta, independente da ordem pela qual a transição de estados seja realizada. Nesse caso a implementação não está sujeita ao problema conhecido como *sincronização inexata* (FABIAN;

HELLGREN, 1998), quando o atraso Δ_{ciclo} da leitura de informação da entrada faz com que o controlador execute uma ação de controle indevida, em virtude de que na planta o evento não controlável já ocorreu, embora o controlador não a tenha identificado.

Malik (2002) apresenta o conceito de autômato $\Sigma_c - \Sigma_u - commuting$, tratando de uma abordagem de identificação à robustez em relação ao problema de *comunicação*. Apesar da propriedade da Definição 3.3 representar a mesma característica dada pela Definição 3.2 (FABIAN; HELLGREN, 1998), a proposta de Malik (2002) é que a propriedade seja verificada no modelo da planta, não no supervisor. Segundo Malik (2002) o modelo de malha fechada, contendo a planta, apresenta a propriedade e qualquer controlador gerado a partir de tal modelo, aceita sinais da planta mesmo que esta esteja com atraso. Além disso, caso no modelo em malha fechada ocorra a desabilitação do evento controlável, o problema de *insensibilidade ao atraso* não está caracterizado e portanto ausente no sistema.

Definição 3.3 (Malik (2002))**.** *Sendo $A = (\Sigma,Q,\delta,q_0,M)$ um autômato determinístico. O autômato A é $\Sigma_c-\Sigma_u-commuting$, se para todo $q \in Q$, $\sigma_u \in \Sigma_u$ e $\sigma_c \in \Sigma_c$ de forma que $\sigma_c \in \Sigma(q)$ e $\sigma_u \in \Sigma(q)$ as transições $\delta(q,\sigma_c\sigma_u)$ e $\delta(q,\sigma_u\sigma_c)$ são definidas e ainda $\delta(q,\sigma_c\sigma_u) = \delta(q,\sigma_u\sigma_c)$.*

Uma versão menos restritiva da propriedade de *insensibilidade ao atraso* (Definição 3.2) é introduzida por Basile e Chiacchio (2007).

Definição 3.4 (Basile e Chiacchio (2007))**.** *Uma linguagem $K = L(S||G)$ é dita* insensível ao atraso *se dados $s \in \overline{K}, \sigma_u \in \Sigma_u$ e $\sigma_c \in \Sigma_c$, $s\sigma_c$,$s\sigma_u \in \overline{K} \Rightarrow s\sigma_u\sigma_c \in \overline{K}$.*

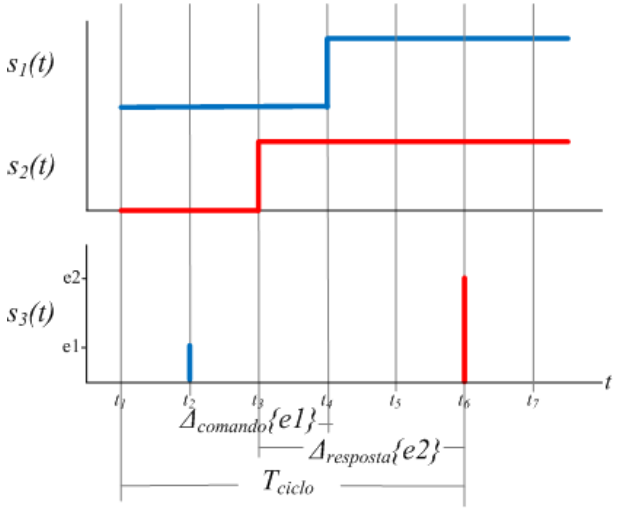
Pela análise da propriedade da Definição 3.4 conclui-se que em um sistema no qual o evento não controlável σ_u é fisicamente impossível após a ocorrência de σ_c, o modelo de malha fechada não apresenta o caminho $s\sigma_c\sigma_u$. Por este motivo não pode ser considerado *sensível ao atraso*, o que seria concluído caso a propriedade da Definição 3.2 fosse utilizada como condição para satisfazer o critério de insensibilidade ao atraso.

Segundo Fabian e Hellgren (1998) e Queiroz (2004) deve haver priorização ao tratamento (transição de estados) de eventos não controláveis nas implementações da estrutura de controle supervisório. Basile e Chiacchio (2007) também utilizam dessa premissa e desta forma, modelos com linguagem de malha fechada que atendam às propriedades da Definição 3.1 e Definição 3.4 operam corretamente em cenários práticos susceptíveis a atrasos. Porém, Basile e Chiacchio (2007) não deixam claro que ao se utilizar da propriedade da Definição 3.4 apenas a priorização no tratamento de eventos não é suficiente para se garantir que não haverá

sincronização inexata devido ao atraso Δ_{ciclo}, conforme é abordado nas contribuições deste trabalho no Capítulo 4.

No intuito de exemplificar o problema que pode ocorrer em uma implementação apresenta-se na Figura 3.7 uma dinâmica de ocorrência e processamento de eventos para o autômato da Figura 3.7d, cuja linguagem atende à Definição 3.4. Considere o estado *0* como inicial em *t1*, no qual também se inicia um ciclo de execução. Mudanças de nível nos sinais $s_1(t)$ e $s_2(t)$ devem ser traduzidas na ocorrência dos eventos *e1* e *e2* respectivamente. No início do ciclo, na leitura dos sinais de entrada, não há mudança no nível de $s_2(t)$ não sendo registrada ocorrência do evento *e2* e o controlador permite que o evento controlável *e1* seja simbolicamente gerado em *t2*. Durante a continuidade do *processamento* ocorre a mudança no sinal $s_2(t)$ na planta física em *t3*, antes do sinal de comando relativo à *e1* ser enviado à planta na etapa de *escrita* do ciclo em *t4*. Caso a atualização de estados seja realizada neste ciclo, por exemplo em *t5*, o controlador observa apenas a ocorrência do evento *e1* e efetua uma transição para o estado *1*, uma vez que a detecção de *e2* apenas é possível na etapa de *leitura* do próximo ciclo em *t6*. Contudo, na planta ambos *e1* e *e2* ocorreram e o estado atual seria o estado *4*.

Figura 3.7: Atraso na conversão do sinal de resposta em evento



Fonte: produção do próprio autor

Por fim um autômato como o da Figura 3.7e, o qual representa uma linguagem de malha fechada que não atende às propriedades descritas, está fatalmente sujeito ao problema da *sincronização inexata*. Para estes casos não se encontra na literatura soluções para o problema,

caso as dinâmicas de ocorrência de eventos e processamento sejam da mesma ordem de grandeza.

3.2 Estruturas e Abordagens de Implementação

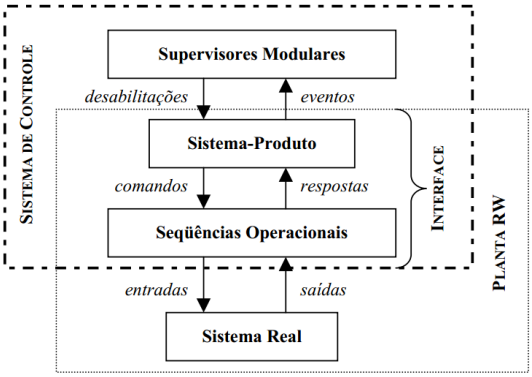
Os problemas e propriedades descritos anteriormente podem ser observados nos modelos de malhada fechada obtidos por intermédio da TCS. Não apenas a característica dos supervisores influencia na ocorrência de problemas, mas também a própria maneira de como a implementação do controle supervisório é realizada. Nesta seção são apresentadas estruturas e abordagens encontradas na literatura para a implementação do controle supervisório e estratégias adotadas em relação aos problemas apresentados anteriormente.

Estruturas e arquiteturas de implementação do controle supervisório são encontradas na literatura visando a adequação aos sistemas práticos, com destaque para (BRANDIN, 1996; QUEIROZ, 2004; VIEIRA et al., 2006; BASILE; CHIACCHIO, 2007; FLORDAL et al., 2007; HASDEMIR et al., 2008; LEAL et al., 2012), onde o dispositivo de controle alvo da implementação é um CLP. Já dentre as obras com implementação para microcontrolador destacam-se (HUBBARD, 2000; COSTA, 2005; CARVALHO, 2007; TEIXEIRA, 2008; SILVA, 2010). Apesar das diferenças nas linguagens de programação e controle de periféricos desses dispositivos, há semelhanças no princípio de funcionamento e portanto as arquiteturas de controle e conceitos introduzidos pelos métodos de implementação podem ser usadas indiferentemente do dispositivo.

A exemplo disso, a estrutura de implementação genérica proposta por Queiroz (2004) tem sido utilizada como base para aplicações do controle supervisório na automação da manufatura assim como em sistemas embarcados. A arquitetura de controle de Queiroz (2004) lida com o problema da causalidade através da introdução de uma interface entre a implementação dos supervisores modulares locais e a planta física. A interface é divida entre os modelos do *sistema produto* responsáveis pela geração de eventos e interação com as *sequências operacionais*, onde implementam-se funções abstraídas do modelo da planta como mostra a Figura 3.8. Os modelos da planta apresentam um evento controlável de comando e um evento não controlável de resposta. A partir do evento de comando as sequências operacionais são responsáveis pela execução dos passos necessários para a conclusão da tarefa sinalizando o término à camada do sistema produto. O sistema produto gera os eventos e efetua as transições de estado nos modelos da planta. Nessa arquitetura os supervisores modulares são observadores dos eventos, atualizando os estados de acordo com a ocorrência desses eventos e a ação de controle é dada pelas desabilitações de eventos controláveis mantendo-se a relação supervisor-planta

estabelecida por Ramadge e Wonham (1989).

Figura 3.8: Estrutura de Implementação proposta por Queiroz (2004)



Fonte: (QUEIROZ, 2004)

Para demonstração e validação da arquitetura, Queiroz (2004) apresenta uma implementação voltada ao controle de sistema de manufatura, utilizando CLP como elemento de controle. Nessa, observa-se que para lidar com o problema da sincronização inexata, faz-se com que para cada evento (controlável ou não) gerado a nível de sistema produto ocorra uma atualização nos estados de supervisores, dando-se prioridade ao tratamento de eventos não controláveis. Para cada evento gerado em um ciclo atualizam-se os estados de supervisores modulares e o processo repete-se enquanto houverem eventos a serem atualizados. Durante esse processo de atualização não ocorrem atualizações de saídas ou leituras das entradas físicas.

Observando a característica cíclica de atualização dos supervisores locais, Vieira et al. (2006) introduz o conceito de *células de controle*, para implementações baseadas na estrutura de Queiroz (2004). Uma célula de controle é constituída por um supervisor local SL_i e por todos os modelos da planta utilizados para formação da planta local GL_i referente ao supervisor SL_i. Portanto, cada supervisor local está associado a apenas uma célula de controle, mas um elemento da planta pode pertencer a várias células de controle. Ao adotar a abordagem de células, permite-se que, após a ocorrência de um evento em um subsistema da planta, apenas os subsistemas das células de controle envolvidas com o subsistema que gerou o evento entrem em espera até que ocorra atualização dos estados dos supervisores. Dessa forma permite-se que mais eventos possam ser gerados no mesmo ciclo de leitura-processamento-escrita. Vieira et al. (2006) propõem que sejam verificadas as propriedades em supervisores para detecção dos problemas de implementação discutidos anteriormente. Especificamente para o caso do problema da escolha, deixa a ordem de chamada das funções de sistema produto determinar o evento controlável a ser gerado. Portanto, o código de implementação é que determina a prioridade na

escolha dos eventos controláveis, sendo necessário que os supervisores atendam à propriedade do determinismo (MALIK, 2002).

Quando a modelagem dos subsistemas da planta não contempla a forma sugerida na implementação de Queiroz (2004), na qual há uma transição controlável de comando e um retorno por evento não controlável de resposta, encontram-se dificuldades na utilização de sequências operacionais. Pinotti et al. (2010) propõem o uso da mesma arquitetura de implementação de Queiroz (2004), embora prevendo-se o uso de modelos que representam dispositivos apenas de comando, como um atuador sem fim ou um dispositivo apenas de resposta como um sensor. Dessa forma, o sistema produto pode acessar respostas provenientes de uma sequência operacional ou diretamente do meio físico, assim como comandos podem ser enviados às sequências ou à planta física. Portanto, a mesma estrutura de Queiroz (2004) é utilizada, embora a implementação de Pinotti et al. (2010) separa o tratamento dos eventos de acordo com a controlabilidade. Transições por eventos não controláveis ocorrem no início do ciclo após a etapa de leitura das entradas. Posteriormente são obtidas as desabilitações e após são tratadas transições por eventos controláveis. A divisão no bloco principal de chamada das funções permite que diversos eventos não controláveis, aqueles já ocorridos no sistema físico, sejam tratados, assim como em um mesmo ciclo, eventos controláveis sejam gerados de acordo com as células de controle. Nesse caso, as células de controle são formadas apenas para avaliação dos eventos controláveis, diferentemente do inicialmente proposto por Vieira et al. (2006). Isso é possível pois as transições por eventos não controláveis ocorrem antes da obtenção das desabilitações e análise das células de controle. Além disso, é proposta uma abordagem para a solução de escolha entre diversos eventos controláveis, através da análise das desabilitações dos supervisores, utilizando da atuação *online* de um *dispatcher* (BASILE; CHIACCHIO, 2007) que complementa as desabilitações quando necessário.

Cruz (2011) conclui que ao se separar a implementação dos blocos que executam a atualização de estados pela controlabilidade e solucionando-se o problema da escolha, não há necessidade da implementação de células de controle. Neste trabalho é adotada uma solução *online* para o problema da escolha e um algoritmo para identificação das escolhas presentes em supervisor é proposto.

Muito embora os trabalhos revisados anteriormente contemplem a implementação em CLP, a arquitetura de Queiroz (2004) também pode ser utilizada quando o dispositivo de controle é um microcontrolador. A exemplo, Costa (2005) apresenta uma implementação utilizando a arquitetura para o controle de um sistema de manufatura. Foi realizada a comunicação entre

CLPs que executam as sequências operacionais com o microcontrolador onde implementam-se supervisores e sistema produto em linguagem *Assembly*. O código foi obtido por intermédio de uma ferramenta de geração automática e foram realizadas simulações para validação do sistema de controle.

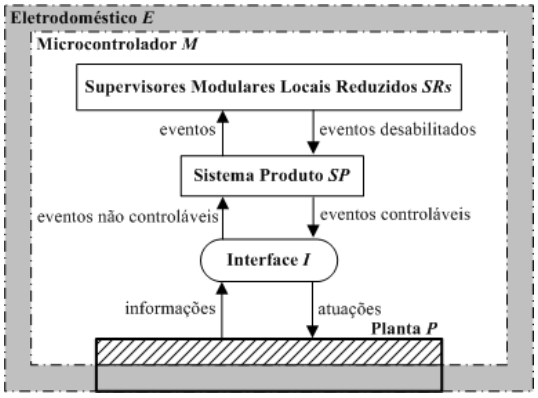
A ocorrência do problema da escolha pode ser identificada no trabalho de Carvalho (2007). Embora no referido trabalho, a solução apresentada não analisa explicitamente a situação de escolha complementando a ação de desabilitação dos supervisores. Carvalho (2007) utiliza uma estratégia de chamada para os elementos do sistema produto de acordo com uma função aleatória. A abordagem evita a chamada de alguns elementos do sistema produto evitando a ocorrência da escolha e, devido à aleatoriedade, permite que as diferentes máquinas sejam acionadas ao longo da execução. Mas neste tipo de solução, quando os supervisores não estão nos estados em que o problema da escolha está presente, a aleatoriedade pode acarretar na chamada de um subsistema cujo evento não está habilitado. No ciclo de execução em que isso ocorrer nenhum evento será tratado pelo controle.

Ao se aplicar a arquitetura em microcontroladores, fica evidente a possibilidade da aplicação em sistemas embarcados. O controle de veículos auto-guiados em um sistema de manufatura, no qual o controle supervisório é aplicado nos controladores embarcados nos veículos é apresentado por Silva (2010). Neste trabalho é utilizada a estrutura de Queiroz (2004) implementada em microcontroladores em linguagem C. A estrutura de controle foi emulada em uma ferramenta computacional e testada em uma bancada experimental.

Por fim, o controle supervisório aplicado a eletrodomésticos é discutido em Teixeira (2008), no qual a arquitetura de Queiroz (2004) é detalhada visando a implementação em microcontroladores conforme a Figura 3.9.

Na concepção dessa arquitetura entende-se que parte dos elementos constituintes do microcontrolador podem fazer parte da planta a ser controlada. Como por exemplo, temporizadores, contadores ou conversores Analógico-Digitais podem ser modelados. Uma interface é concebida para tradução de informações e atuações do sistema físico para o nível de sistema produto. No trabalho desenvolvido por Teixeira (2008) os eventos não controláveis são gerados por funções de leitura na interface e processados para atualização dos estados de supervisores modulares. Eventos controláveis são gerados utilizando-se um esquema de prioridade fixa, e para garantir-se que a implementação não acarreta em um bloqueio é necessário que a propriedade do determinismo de Malik (2002) seja utilizada. No entanto, eventos controláveis e não controláveis

Figura 3.9: Estrutura de Implementação de Teixeira (2008)



Fonte: (TEIXEIRA, 2008)

são gerados em diferentes níveis da estrutura. Para os eventos controláveis as funções de escrita recebem os eventos gerados a nível de sistema produto, e apenas definem a ação a qual o referido evento executa na planta. Diferentemente, os eventos não controláveis são gerados nas funções de leitura da interface.

As estruturas de controle descritas definem de que forma os elementos constituintes da arquitetura interagem entre si, mas não necessariamente definem as formas como devem ser codificados. Como observa-se nas diferentes implementações utilizando a mesma estrutura de controle de Queiroz (2000). Segundo Barreta e Torrico (2008) as formas mais usuais de implementação dos modelos (autômatos) são utilizando vetores, matrizes ou listas encadeadas. Barreta e Torrico (2008) implementam supervisores obtidos pela abordagem monolítica utilizando de listas encadeadas visando redução no consumo de memória do microcontrolador. O trabalho propõe também uma estrutura de controle, apresenta uma ferramenta de geração de código e resultados para validação das estruturas usadas em uma bancada experimental. Ferigollo et al. (2011) utilizam a estrutura de controle monolítica de Barreta e Torrico (2008) e apresenta sua extensão para o controle modular local. Além disso, realiza estudos comparativos entre as duas abordagens e evidencia o aumento no consumo de memória quando aumenta-se o número de modelos implementados.

Outra estrutura para implementação de supervisores que visa redução em consumo de memória é apresentada por Lopes et al. (2011). No referido trabalho também apresenta-se uma arquitetura de implementação para o controle supervisório sob abordagem monolítica. O problema da escolha é estudo no trabalho, utilizando aleatoriedade para execução de eventos controláveis nos estados onde há mais que um evento não desabilitado. Tal técnica é possível pois na abordagem monolítica tem-se apenas um supervisor, portanto não há necessidade de análise

de uma ação conjunta de desabilitação. Segundo Lopes et al. (2011) a estrutura de codificação de supervisores e a ferramenta geração de código utilizada podem ser expandidas para a abordagem modular local. Mas não são evidenciadas as alterações necessárias na arquitetura de controle ou na análise do problema da escolha neste caso.

A Tabela 3.1 apresenta um comparativo entre diferentes propostas de estruturas de implementação do controle supervisório voltadas à microcontroladores.

Tabela 3.1: Comparativo entre Estruturas para Microcontrolador

Autor	**Estrutura**	**Escolha**	**Sincr. Inexata**	**Linguagem**	**Geração Código**	**Validação**
Costa 2005	Max	N	N	Assembly	S	Simulaçao
Carvalho 2007	Max	I	N	C	S	Simulação
Teixeira 2008	Max	Malik	Fabian	C	S	Protótipo
Torrico 2008	do Autor	N	N	C	S	Bancada
Silva 2010	Max	N	N	C	N	Bancada
Lopes 2011	do Autor	Aleatória	N	C	S	Bancada

Fonte: produção do próprio autor

3.3 Conclusões do Capítulo

A revisão acerca da implementação da TCS ressalta a existência de problemas que podem ocorrer na aplicação do controle supervisório quando implementados em dispositivos de funcionamento sequencial tais como CLP e microcontroladores. Características nos supervisores que modelam o controle em malha fechada do sistema podem ser observadas em comparação à propriedades que evidenciam que possíveis implementações práticas podem levar a bloqueios ou a perda de sincronismo entre controlador e planta.

Arquiteturas para a implementação do controle supervisório visando a solução de tais problemas são encontradas na literatura. Embora, algumas características ainda podem ser mais exploradas ou necessitam de cuidados não claramente evidenciados. Por exemplo, Basile e Chiacchio (2007) introduzem o uso de um bloco determinador para a solução da escolha, deixando a critério de cada aplicação definir a política de escolha. Além disso, não há menção de como o determinador se comporta ao utilizarem-se supervisores reduzidos. Os autores também

apresentam uma importante análise para a propriedade de insensibilidade ao atraso, utilizada como avaliação para o problema da sincronização inexata. Entretanto, não evidenciam cuidados a serem tomados na atualização de estados dos supervisores (e sistema produto), nos casos em que a propriedade está presente no supervisor (ou em um supervisor modular), conforme discutido neste capítulo.

A solução usual para o problema da causalidade, adotada em diversos trabalhos revisados, é dada pela utilização de uma interface entre supervisores e planta física. Apesar disso, ainda pode-se notar certa discrepância na geração de eventos em alguns casos. Por exemplo, na implementação da interface proposta por Teixeira (2008), eventos não controláveis são gerados pela interface, mas eventos controláveis são provenientes do sistema produto.

Diferentes abordagens e estruturas de implementação dos modelos também são encontradas. Contudo, observa-se que muitas dependem, por exemplo, da quantidade de elementos modelados e número de eventos envolvidos na modelagem. Portanto, soluções mais genéricas para a codificação podem ser elaboradas aumentando a possibilidade de reuso da implementação.

Dessa forma, o próximo capítulo visa apresentar um método para o projeto do controle supervisório modular local visando suprir as deficiências nos métodos analisados neste capítulo.

Capítulo 4

Método e Arquitetura Genérica de Implementação

Neste capítulo apresenta-se um método para o projeto do controle supervisório em sistemas embarcados. Nesses sistemas o dispositivo de controle mais comumente utilizado é um micro-controlador. Para a etapa de implementação é apresentada e discutida uma arquitetura proposta com base nas estruturas lógicas discutidas no Capítulo 3.

A arquitetura foi elaborada principalmente a partir dos conceitos introduzidos por Queiroz (2004), Basile e Chiacchio (2007) e Teixeira (2008). Apresenta-se o diagrama das estruturas de implementação, algumas diferenças em relação às arquiteturas de referência, contribuições e recomendações no uso da arquitetura visando sua aplicação ao controle de sistemas embarcados.

Com o intuito de solucionar algumas lacunas presentes em abordagens encontradas na literatura, as formas de como a estrutura lida com os problemas de implementação da TCS como *escolha*, *sincronização controlador-planta* e *causalidade* são analisados e discutidos neste capítulo, apresentando-se a fase de implementação com maior destaque.

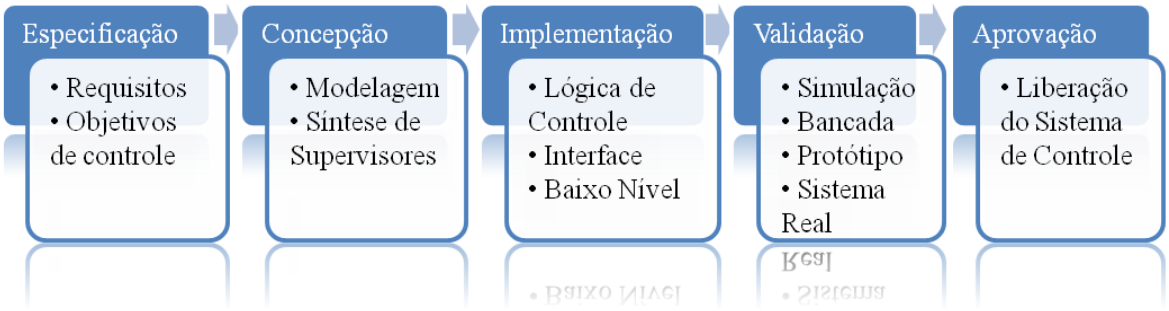
4.1 Método para o Projeto do Controle de Sistemas Embarcados

O método para o projeto do controle supervisório de Sistemas Embarcados constitui-se de etapas que definem tarefas a serem executadas em cada fase do projeto, conforme ilustrado na Figura 4.1.

Inicialmente, na etapa de *Especificação* são coletados os requisitos e os objetivos de controle, ou seja, determina-se qual a função do controle no sistema. O levantamento de requisitos é importante pois auxilia no entendimento das funções do sistema e fornece o suporte para a etapa de *Concepção*.

De posse das informações do sistema e dos objetivos de controle é possível obter a

Figura 4.1: Etapas do Método de Projeto baseado no Controle Supervisório



Fonte: produção do próprio autor

modelagem dos subsistemas de maneira mais clara. Conforme o número de subsistemas e de especificações de controle, maior o número de elementos a serem modelados para obtenção do controle supervisório. Por este motivo e devido ao aumento da complexidade dos algoritmos de controle nas mais variadas aplicações em sistema embarcados, ficam mais evidentes as vantagens do uso da abordagem *modular local* (QUEIROZ, 2000) para a síntese de supervisores.

A partir dos modelos de supervisores modulares locais (QUEIROZ, 2000) algoritmos de redução (VAZ; WONHAM, 1986; SU; WONHAM, 2004) são aplicados a fim de reduzir o número de estados dos supervisores a serem implementados. O uso dessas estratégias visam a economia no consumo de memória do controlador do sistema (QUEIROZ; CURY, 2002; FERIGOLLO et al., 2011). É possível que durante o processo de síntese de supervisores se obtenha uma linguagem alvo K *controlável*, nestes casos não se faz necessário o uso de algoritmos de redução de supervisores, uma vez que a própria especificação de controle representa um modelo de supervisor reduzido.

A etapa de concepção disponibiliza um conjunto de supervisores modulares reduzidos e elementos do sistema produto a serem implementados para obtenção da lógica de controle, já na fase de *Implementação*. Nessa etapa, pode-se utilizar de geradores automáticos de código, de acordo com a estrutura de implementação desejada. O uso de ferramentas auxilia na obtenção da lógica de controle, ficando a cargo do projetista a implementação da interface e rotinas de baixo nível, assim como as possíveis configurações necessárias para o dispositivo de controle (microcontrolador) utilizado. Uma arquitetura de implementação é proposta neste capítulo, e esta é utilizada como a lógica de controle para um gerador de código desenvolvido neste trabalho.

Este trabalho apresenta maior enfoque na etapa de implementação. A arquitetura de controle proposta é dividida em camadas de forma que apenas a camada de mais baixo nível tem dependência com o microcontrolador. Dessa forma, as camadas de mais alto nível não variam caso o controlador seja alterado. Da mesma forma, alterações em uma especificação de controle

concentram-se apenas na parte de implementação dos modelos de supervisores. Não sendo necessária a geração de todo o código para qualquer modificação ou até mesmo para controle de outras aplicações.

Após a fase de implementação o sistema de controle é submetido a uma etapa de *Validação*. Nessa fase são efetuados testes baseados em simulações, experimentos em bancada, em protótipos ou ainda no sistema real com uso de instrumentações adequadas. Essa etapa avalia se o controle atua de acordo com as especificações inicialmente estabelecidas. O não atendimento a algum requisito normalmente acarreta no retorno à fase de concepção para análise, especialmente da modelagem utilizada para tradução dos objetivos nas especificações de controle. Quando entende-se que os requisitos estão atendidos o projeto evolui para a fase de *Aprovação* na qual ocorre a liberação do sistema de controle.

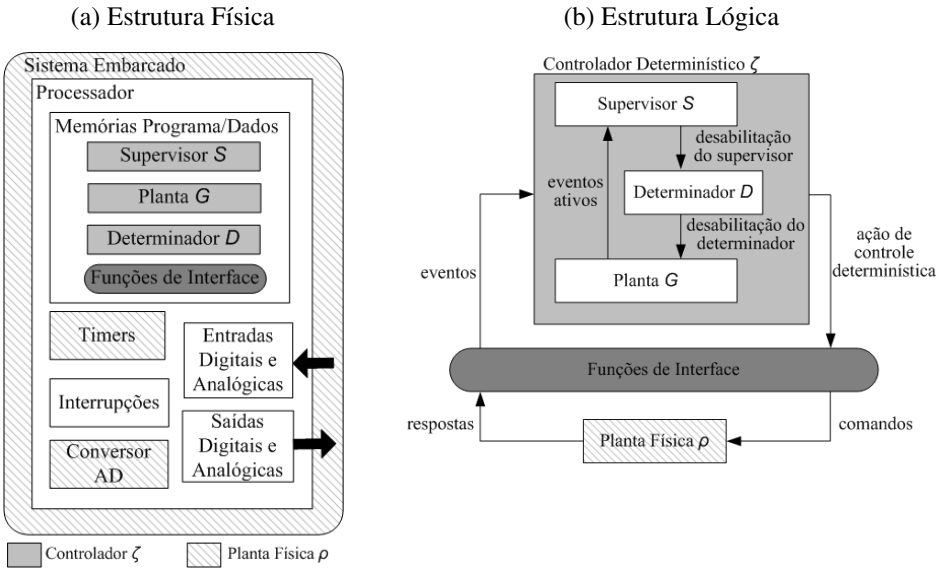
4.2 Descrição da Arquitetura de Implementação

A estrutura de um microcontrolador em um sistema embarcado usualmente consiste nas memórias de programa e de dados, nos registradores e nos periféricos tais como entradas e saídas digitais, conversores AD, temporizadores e interrupções. Com isso, os modelos do supervisor S e da planta G, assim como funções de interface são implementados na memória do processador como apresentado na Figura 4.2a. O sistema embarcado como um todo contém elementos do meio físico, como atuadores, cargas e sensores, por exemplo, e nele encontra-se embarcado o processador com o controle. A interação do processador com o meio físico se dá por intermédio de suas entradas e saídas. A estrutura física da Figura 4.2a ainda denota que parte dos periféricos presentes no microcontrolador podem ser parte da planta física ρ, ora por fazerem parte da modelagem ora por terem sido abstraídos nos modelos, mas sendo necessários para a ocorrência de eventos na planta por intermédio de comandos e respostas, assim como são para os demais elementos.

A estrutura lógica da Figura 4.2b apresenta os mesmos elementos de maneira funcional. A ação de desabilitação do supervisor S com auxílio de um determinador D sobre a planta G (uma abstração de ρ) determina o conjunto de eventos controláveis desabilitados em um estado. Esta atuação define a *ação de controle determinística* do Controlador ζ sobre a planta física ρ.

A arquitetura de implementação proposta neste trabalho está baseada na estrutura lógica da Figura 4.2b, que está fundamentada na estrutura de Queiroz (2004). Ela é constituída de três camadas de acordo com o nível lógico e hierárquico da execução das funções nelas

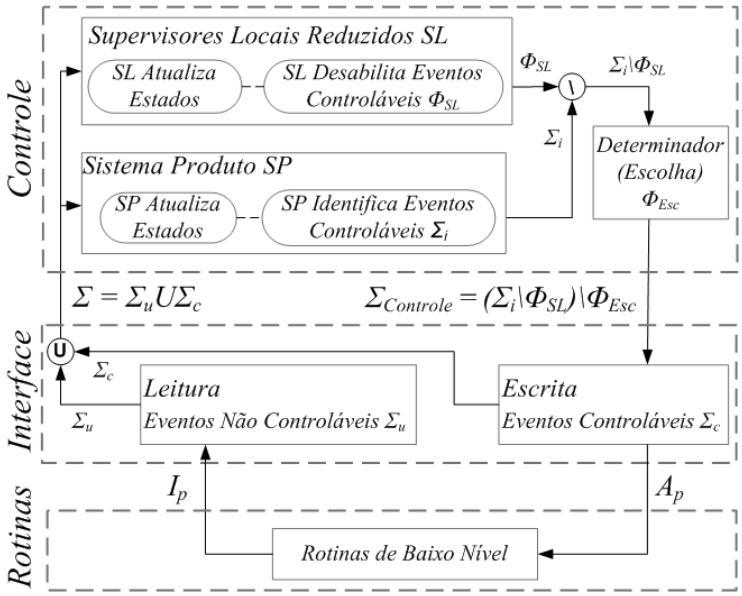
Figura 4.2: Estruturas (a) Física e (b) Lógica do controle supervisório em Sistemas Embarcados



Fonte: produção do próprio autor

implementadas, conforme ilustrado na Figura 4.3. Lembra-se que a estrutura apresenta os blocos *lógicos* do sistema de controle, portanto parte do que se entende por meio físico (planta física) pode estar embarcado no controlador, dependendo das necessidades do problema e da modelagem.

Figura 4.3: Arquitetura de Implementação



Fonte: produção do próprio autor

Na camada de mais alto nível, denominada de *Controle*, são implementados os blocos lógicos dos Supervisores Modulares Locais Reduzidos, Sistema Produto e o bloco lógico do

Determinador para solução *online* do problema da escolha, conforme a proposta de Basile e Chiacchio (2007).

Segundo essa estrutura, os supervisores locais são implementados por um bloco de *atualização de estados* e um bloco para obtenção da *ação de desabilitação*. Há uma conexão entre as duas partes, pois a cada estado atual novas ações de desabilitações sob os eventos controláveis são obtidas, principalmente pelo fato de se utilizarem supervisores reduzidos, pois nestes a mudança de estado pressupõe uma nova ação de desabilitação (VAZ; WONHAM, 1986; SU; WONHAM, 2004).

O bloco de implementação dos supervisores locais recebe a informação dos eventos ocorridos e executa a transição de estados de acordo com os modelos. A partir do estado atual dos supervisores obtêm-se as *desabilitações dos supervisores* as quais consistem no conjunto Φ_{SL} de eventos controláveis desabilitados pela ação conjunta dos supervisores. Dessa maneira, para os supervisores, todas as demais estruturas da arquitetura de implementação comportam-se como a planta de Ramadge e Wonham (1989).

De forma similar, o bloco de implementação do Sistema Produto é dividido entre a *atualização de estados* e a *identificação* do conjunto de eventos controláveis ativos Σ_I no estado atual da planta. A informação conjunta fornecida por Supervisores e Sistema Produto resulta no conjunto de eventos controláveis ativos que não foram desabilitados pela ação conjunta dos supervisores, representado por $\Sigma_I \setminus \Phi_{SL}$. Tal conjunto é observado pelo bloco do determinador para solução do problema da escolha. Suas características de funcionamento são discutidas adiante neste capítulo, embora se resumam a complementar a ação de supervisão Φ_{SL} com o conjunto de eventos controláveis desabilitados na escolha Φ_{Esc}.

O conjunto de eventos controláveis desabilitados $\Sigma_{Controle} = (\Sigma_I \setminus \Phi_{SL}) \setminus \Phi_{Esc}$ é fornecido à camada da *Interface* para ser processado. A interface executa a análise para gerar (ou não) o evento. Conforme previamente discutido no Capítulo 3 existem diferenciações na realização das interfaces, a serem destacadas a seguir para posterior comparação com a concepção da interface proposta pela arquitetura da Figura 4.3.

No trabalho de Queiroz (2004) a interface é constituída pelo conjunto de implementação de sistema produto e sequências operacionais. O bloco de sistema produto é o responsável pela geração de eventos para a camada superior onde estão presentes apenas os supervisores modulares. Faz-se necessário que a camada de sistema produto contenha funções de interpretação de *respostas* (fim das sequências operacionais) e geração de *comandos* (início das sequências

operacionais), além da geração de eventos.

Na estrutura de implementação de Teixeira (2008) a tarefa de geração de eventos para supervisores modulares também cabe ao sistema produto. Com o intuito de promover a implementação de supervisores em microcontroladores, é concebida uma interface, para isolar a interpretação de *informações* e gerenciamento de *atuações* dos sinais para a planta física. Apesar disso, ao se analisar a sequência de execução utilizada para implementação, nota-se que a interface trata eventos diferentemente de acordo com a controlabilidade. A partir de informações da camada de baixo nível a interface *gera* eventos não controláveis por intermédio de funções de leitura. Porém, a interface *interpreta* eventos controláveis provenientes da camada superior (sistema produto) para realizar atuações através de funções de escrita.

Portanto, seguindo a estratégia de implementação proposta por Teixeira (2008), na arquitetura da Figura 4.3 a Interface é divida nos blocos das funções de Leitura e Escrita. Porém, nessa estrutura, as funções presentes na camada de interface são integralmente responsáveis pela *geração* de eventos sendo estes controláveis ou não. Ainda nessa arquitetura, a interface realiza tais tarefas, cabendo ao bloco do sistema produto apenas a implementação dos modelos. Por isso modelos de sistema produto são usados apenas a nível de *Controle* para identificação de eventos ativos, e não constituem parte da interface.

De forma a isolar a camada de alto nível *Controle* da camada de baixo nível, na arquitetura da Figura 4.3 a interface interage de acordo com o nível de informação esperado pelas outras camadas. A camada de controle lida com eventos e desabilitações de eventos, já as *Rotinas* lidam com sinais.

A partir do nível de Controle, a interface interpreta o conjunto das desabilitações de *eventos controláveis* $\Sigma_{Controle}$ através das funções de *Escrita*. Essas, por sua vez, *geram* os eventos controláveis, representados pelo conjunto Σ_c, ao mesmo tempo em que *atuam* nas rotinas de baixo nível com os *sinais* A_p. Das informações I_p provenientes das rotinas, as funções de *Leitura* da interface também *geram* eventos não controláveis, denotados pelo conjunto Σ_u. O conjunto dos *eventos* ocorridos $\Sigma = \Sigma_c \cup \Sigma_u$ é fornecido pela interface para os blocos da camada *Controle*.

Na camada de mais baixo nível (*Rotinas*) são previstas funções a serem utilizadas pela interface para interpretação e posterior geração de eventos conforme explicado anteriormente. Além dessas, a camada pode conter funções para configuração e comunicação com periféricos do microcontrolador, outros módulos ou algoritmos que possam servir para o correto funcionamento

do sistema embarcado, mas não fazem parte ou foram abstraídas na modelagem para obtenção do controle supervisório.

Considera-se que as funções presentes na camada das *Rotinas*, especialmente as utilizadas pela interface para a análise e geração dos eventos, são executadas com maior frequência pelo programa do microcontrolador do que as funções envolvidas nas camadas superiores, à exemplo das funções de leitura e de atualização de estados. Implementações com esse tipo de execução possuem diferentes *faixas de tempo* para a chamada das funções e são consideradas como a codificação de um Sistema Operacional primitivo (TANENBAUM, 2009).

Um exemplo para a necessidade da execução das funções das diferentes camadas em bases de tempos distintas é o caso da leitura dos sinais de entrada do microcontrolador provenientes de botões de interface com o usuário. A presença de ruídos devido ao chaveamento dos componentes faz com que seja necessário o uso de técnicas de *debounce* (GANSSLE, 2008) desses sinais de entrada. Uma rotina de baixo nível pode ser codificada para filtrar o sinal do pino do microcontrolador e interpretar se há um sinal válido, ou seja, realmente há um pressionamento do botão por parte do usuário. Exemplos de rotinas de *debounce* são apresentadas por Ganssle (2008).

Como a função de leitura é executada em intervalos maiores de tempo, a cada chamada a informação filtrada mais atual na rotina de baixo nível é a disponibilizada para a interface. Um exemplo seria o uso de funções de leitura executadas a cada *1 segundo* com rotinas de baixo nível chamadas em intervalos de *25 milisegundos*. A rotina é executada 40 vezes, tratando o sinal, para uma chamada de interpretação da interface. Portanto, a preocupação de se detectar as mudanças de sinal no meio físico são concentradas na camada *Rotinas* destinada a tratar os sinais de baixo nível.

Outros exemplos são rotinas que realizam a conversão de canais AD, atualização (*refresh*) de pinos de entrada e saída ou controle de periféricos para fins de comunicação como *Universal Asynchronous Receiver Transmitter* (UART), *Controller Area Network* (CAN) ou *Inter-Integrated Circuit* (I2C). As rotinas de baixo nível não necessariamente precisam estar ligadas a funções do microcontrolador, podendo também ser códigos relacionados com funções do sistema embarcado vinculadas ao acionamento de cargas, verificação de erros ou ainda algoritmos de proteção.

Características da arquitetura apresentada na Figura 4.3, como a taxa de execução das funções da camada *Rotinas* em relação à da *Interface* e de *Controle*, são utilizadas para auxiliar

na solução dos problemas de implementação discutidos no Capítulo 3. Essas características são destacadas nas seções a seguir de acordo com o problema a que estão relacionadas.

4.2.1 Problema da Escolha - Determinismo

O método para solução do problema da escolha proposto na arquitetura de implementação é o da análise dinâmica da ação de controle dos supervisores através do bloco *determinador*. O determinador analisa a atual ação de desabilitação dos supervisores e, de acordo com a política e lógica adotadas, complementa a ação com a desabilitação de eventos controláveis envolvidos na situação de escolha.

Diferentes políticas de desabilitação podem ser adotadas pelo bloco de solução da escolha. Alternativas mais comuns utilizam variáveis com atualizações aleatórias para capacitar o determinador a atuar com diferentes desabilitações quando requisitado. Carvalho (2007) utiliza uma biblioteca padrão da linguagem C, a *cstdlib* para que a função *rand()*, que retorna um valor inteiro pseudo-aleatório, seja usada para estabelecer uma escolha, definindo um entre os possíveis subsistemas a serem chamados. Porém, como mencionado no Capítulo 3, nessa estratégia é possível a chamada de um subsistema que não tenha eventos controláveis ativos. Além disso, o uso dessa função leva a um maior consumo de memória do microcontrolador em relação à outras técnicas para obtenção de valores pseudo-aleatórios. A função *rand()* pode ser substituída por algoritmos que implementam a obtenção de valores randômicos ou por uso de variáveis modificadas constantemente a bases de tempo independentes da chamada do bloco de solução. Trabalhos como os de Cruz (2011) e Pinotti et al. (2010) apresentam soluções nessa linha de desenvolvimento que apesar de voltadas à aplicação em CLP também são aplicáveis em microcontrolador.

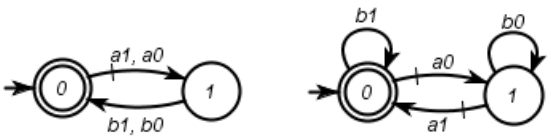
Neste trabalho, a proposta de atuação do bloco de solução de escolha é que este seja executado apenas quando o controle deixa de desabilitar os eventos envolvidos no problema. Algo evidenciado nesta arquitetura é que ao utilizar da implementação de supervisores reduzidos perde-se a informação da condição atual da planta. Portanto, propõe-se que o bloco do determinador atue analisando o conjunto de eventos controláveis ativos na planta sob a ação dos supervisores $\Sigma_I \setminus \Phi_{SL}$. Dessa forma, garante-se que, ao se utilizar supervisores reduzidos, não serão realizadas desabilitações pelo bloco determinador em momentos inadequados. De modo que a ação de desabilitação age igualmente àquela na qual implementam-se supervisores completos, aumentando a eficiência do algoritmo de controle, no sentido de que as mesmas

desabilitações são realizadas, embora modelos menores de supervisores são utilizados.

A análise das situações de escolha, quando utilizada a abordagem modular local (QUEIROZ, 2000), pode ser realizada em cada supervisor local. Embora, caso identificado um problema de escolha em um supervisor local não há garantia que esta ocorra na ação conjunta dos supervisores. Um exemplo é apresentado na Figura 4.4. A análise do supervisor local SL_1 (Figura 4.5a) resulta na identificação de uma situação de escolha entre os eventos controláveis $a0$ e $a1$ no estado *0*. Embora a ação conjunta de SL_1 com o supervisor local SL_2 mostrado na Figura 4.5b não apresente o problema da escolha.

Os problemas de controle pesquisados no Grupo de Automação de Sistemas e Robótica (GASR) da UDESC, como por exemplo (TEIXEIRA, 2008), (LEAL et al., 2012) e (ALMEIDA, 2012), têm demonstrado que, não havendo problema da escolha em supervisores locais, este também não ocorre na ação conjunta dos supervisores. No entanto, para total garantia de que não haverá problemas de escolha na ação conjunta dos supervisores, avalia-se o autômato obtido pela composição síncrona dos supervisores locais, que contém no alfabeto os eventos controláveis sob análise.

Figura 4.4: Ação conjunta de supervisores que elimina a condição do problema de escolha



(a) SL_1 (b) SL_2

Fonte: produção do próprio autor

Caso o problema de escolha não seja adequadamente analisado nos supervisores locais, pode-se implementar uma política de atuação do determinador para uma situação que nunca estará ativa devido a ação conjunta dos supervisores. A implementação da solução do problema da escolha nesse caso será simplesmente um código a mais que não será executado. Contudo, apesar do consumo de memória desnecessário, não há mais prejuízos à implementação, uma vez que a situação nunca estará ativa e o determinador nunca executa uma ação de desabilitação complementar.

4.2.2 Sincronismo entre Controlador e Planta

Conforme discutido no Capítulo 3, o problema da *sincronização inexata* pode ocorrer em supervisores nos quais a linguagem não apresenta certas propriedades. Uma propriedade para a insensibilidade ao atraso foi definida por Fabian e Hellgren (1998) (ver Definição 3.2). Essa foi posteriormente relaxada por Basile e Chiacchio (2007) (ver Definição 3.4), a fim de obter uma propriedade a ser utilizada em um sistema no qual o evento não controlável envolvido na análise, não é fisicamente possível após a ocorrência de um determinado evento controlável.

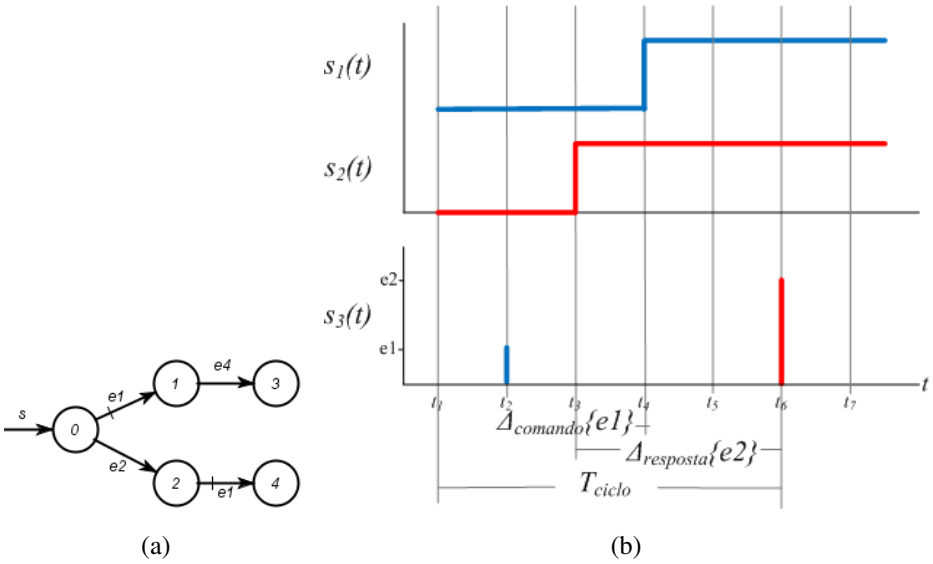
De acordo com Queiroz (2004), para garantir a consistência do controle sobre a planta, é necessário que as transições por eventos não controláveis que já ocorreram no sistema físico sejam executas antes das transições devidas a eventos controláveis. Dessa forma a ação de desabilitação baseia-se no atual estado da planta.

Portanto, os blocos que implementam as atualizações de estado de Supervisores e Sistema Produto no diagrama da Figura 4.3 devem ser executados de forma a tratar os eventos não controláveis prioritariamente. Além disso, no caso em que supervisores que atendem à Definição 3.4 são implementados, alguns critérios na ordem de execução das funções devem ser seguidos para garantir que não haverá problema da sincronização inexata devido ao atraso de resposta sob a influência de Δ_{ciclo}.

Retomando a análise iniciada no Capítulo 3 a respeito da propriedade de insensibilidade ao atraso de Basile e Chiacchio (2007) apresenta-se novamente a dinâmica de processamento na Figura 4.5. Nessa figura tem-se o autômato cuja linguagem atende à propriedade de *insensibilidade ao atraso* de Basile e Chiacchio (2007). Os atrasos na conversão de sinais de comando e resposta em eventos são ilustrados na Figura 4.6b, que apresenta o ciclo de execução.

Neste trabalho propõe-se que as funções de leitura sejam executadas no início de cada ciclo t_1. Em seguida são efetuadas as transições de estado dando-se prioridade aos eventos não controláveis. Realizando-se a verificação das funções de Leitura, ou seja, realizando a identificação e tradução dos sinais disponibilizados nas entradas, antes de qualquer transição permite-se que a ocorrência de um evento não controlável, detectada com atraso, seja capturada pela interface, sendo tal evento disponibilizado à camada de *Controle*. Não se realizam atualizações de estados por eventos controláveis no final do ciclo, por exemplo em t_5 na Figura 4.6b, de modo que supervisores e planta efetuam transições na ordem como os eventos ocorreram na planta física.

Figura 4.5: (a) Autômato cuja linguagem atende à Definição 3.4 (b) Atrasos na conversão de sinais em eventos



(a) (b)

Fonte: produção do próprio autor

Dessa forma, eventos não controláveis são gerados e transições de estado por tais eventos são efetuadas no mesmo ciclo. No caso dos eventos controláveis, estes são gerados durante a etapa de execução das funções de escrita na interface, mas as transições de estado ocorrem no próximo ciclo, após as funções de entrada terem sido novamente executas. Utilizando-se de *buffers* a interface pode facilmente lidar com este tipo de armazenamento, podendo-se ter um mesmo *buffer* ordenado para priorizar eventos não controláveis ou simplesmente utilizar-se de *buffers* distintos.

Na arquitetura proposta neste trabalho a camada de *Rotinas de Baixo Nível* possui funções executadas a menores intervalos de tempo. Assim, garantindo-se que essas chamadas ocorram a taxas adequadas para a captura dos sinais provenientes do meio físico, a preocupação com detecção de sinais não é elevada às camadas superiores. Garante-se que pela chamada das funções de leitura eventos não controláveis ocorridos durante um ciclo de execução são capturados pelo algoritmo na camada de *Controle*, tendo-se o sincronismo entre controlador e planta.

Vale salientar que supervisores que não atendam à nenhuma das propriedades, portanto ditos *sensíveis ao atraso* não apresentam necessariamente problemas na implementação. É sempre importante que o projetista avalie o significado físico dos eventos envolvidos ao se deparar com supervisores contendo esta característica. Um exemplo é quando há uma dependência física entre os eventos modelados. Em um sistema de refrigeração, um evento não controlável

modelado para detecção da passagem por um limiar de temperatura fria só é fisicamente viável de ocorrer se o dispositivo que realiza a troca de calor tiver sido ligado. No entanto, em um dado estado do supervisor ambos os eventos podem estar ativos, como pode ser observado em supervisores do estudo de caso apresentado no Capítulo 5.

Há outra alternativa que pode ser considerada quando a linguagem de um supervisor atender à Definição 3.4 e não adota-se essa estratégia de transições de estados. Adicionando-se o caminho ausente $\sigma_c\sigma_u$ ao modelo implementado (*e1e2* no exemplo da Figura 4.6a), faz-se com que a implementação atenda à Definição 3.2 e há garantia de não ocorrência do problema da sincronização inexata. Uma vez que os eventos são gerados pela planta, a ausência do evento não controlável no modelo significa que ele é fisicamente impossível de ocorrer *após* o evento controlável. A transição pelo caminho adicionado apenas será feita na condição de ter havido um atraso na identificação do evento não controlável e dessa forma tem-se a mesma transição ocorrendo no supervisor. Apesar de possível, durante este trabalho não foram realizados testes com implementações utilizando esta abordagem, pois considerou-se que o uso de armazenamento dos eventos em *buffers* não prejudica ou traz impactos significativos ao funcionamento do controle.

Um segundo problema relacionado ao sincronismo entre a planta e controlador gerado pela ocorrência de eventos não controláveis é o da *sensibilidade ao entrelaçamento*. Quando um supervisor não atende à propriedade da Definição 3.1, sugere-se utilizar interrupções para as entradas dos sinais relativos aos eventos envolvidos. Ao detectar a interrupção o programa pode utilizar a função de interface para determinar a ocorrência do evento. O uso de *buffers* garante que as transições de estado se deem na ordem de ocorrência dos eventos. Porém, tendo em vista que este tipo de problema nunca foi detectado em problemas reais pelos pesquisadores do GASR da UDESC, o mesmo não foi objeto de estudo do presente trabalho.

4.2.3 Causalidade

A TCS prevê que eventos são gerados espontaneamente pela planta enquanto o supervisor observa tais eventos e age sobre a planta com uma ação de desabilitação de eventos controláveis (RAMADGE; WONHAM, 1989). Conforme discutido no capítulo anterior, nas aplicações práticas essa condição não é encontrada, pois neste caso, o controle normalmente precisa efetuar ações na planta. A questão de determinar se controlador ou planta física é a fonte de geração de eventos é chamada de problema da *causalidade* (FABIAN; HELLGREN, 1998). Contudo, na implementação

do controle supervisório é possível separá-lo por hierarquia e constituir uma interface para que os supervisores se comportem como nas hipóteses adotadas pela TCS (RAMADGE; WONHAM, 1989), conforme já preconizado por Queiroz (2004).

Na estrutura de implementação proposta neste trabalho a mesma abordagem de hierarquia é estabelecida. Com o auxílio da interface os eventos $\Sigma = \Sigma_c \cup \Sigma_u$ são gerados e apenas observados pelo *nível de controle*, que por sua vez traduz o estado atual na ação de desabilitação $\Sigma_{Controle}$.

A camada de menor nível hierárquico, das *Rotinas*, é observada pela *Interface* para que esta sinalize a ocorrência dos eventos. Portanto, não é necessário que uma execução de uma função de *Leitura* ou de *Escrita* obrigatoriamente resulte na ocorrência de um evento. Nos trabalhos de Teixeira (2008) e Cruz (2011), por exemplo, as leituras podem não acarretar em eventos não controláveis, embora as chamadas de escrita obrigatoriamente disparam ações e eventos controláveis.

Neste trabalho as funções de escrita são recorrentemente executadas enquanto o evento controlável correspondente não está desabilitado e a interface gera o evento ao reconhecer que a planta executou a ação, isto através do algoritmo na rotina de baixo nível. Considera-se que Rotinas de Baixo Nível podem conter funções que podem estar vinculadas ao funcionamento de planta, dependendo do nível de abstração dado aos modelos, de tal forma que a chamada de uma função de escrita na interface requisita a ação vinculada ao evento controlável, embora ao analisar o baixo nível, observa que a ação não foi atendida. Essa abordagem aproxima ainda mais a implementação da teoria, uma vez que eventos controláveis também são gerados a partir da informação da planta.

4.3 Conclusões do Capítulo

A arquitetura e as características apresentadas nesse capítulo visam tratar dos principais aspectos de implementação da estrutura de controle supervisório em microcontroladores. A estrutura contempla os blocos necessários à execução do controle supervisório baseando-se numa divisão hierárquica proposta por Queiroz (2004), adequação de uma interface de *leitura* e *escrita* (TEIXEIRA, 2008) e aplicação da verificação *online* da ação de controle dos supervisores para complemento e determinismo da ação de desabilitação (BASILE; CHIACCHIO, 2007).

Na etapa de implementação do método proposto utiliza-se uma arquitetura que se

mostra adequada para lidar com os problemas de implementação discutidos no Capítulo 3.

O bloco determinador proposto na estrutura de implementação atua analisando o conjunto de eventos controláveis desabilitados pelos supervisores locais e ativos pela planta. Tal estratégia deve ser aplicada quando os modelos de supervisores reduzidos são utilizados pela implementação. Desse modo, as desabilitações complementares do determinador ocorrem da mesma forma como se fossem utilizados supervisores não reduzidos. Dessa maneira, consegue-se o mesmo resultado de atuação do controle, embora o tamanho dos modelos implementados são menores.

A estrutura aplica uma estratégia de execução das funções de leitura no início do ciclo e posterior execução das transições de estados priorizando eventos não controláveis. Portanto, transições por meio de eventos controláveis não ocorrem no mesmo ciclo de execução em que tais eventos são gerados. Abordagem que permite utilizar a propriedade mais relaxada da sensibilidade ao atraso (Definição 3.4 de Basile e Chiacchio (2007).

Para as funções de escrita da interface propõe-se uma estratégia quando são abstraídas do modelo características da planta que podem ser extraídas das rotinas de baixo nível. Dessa forma, a execução das funções de escrita geram eventos controláveis somente quando uma informação proveniente da planta é obtida, assim como ocorre para os eventos não controláveis. Dessa forma a implementação torna-se mais consistente com a teoria, reduzindo os efeitos do problema da causalidade.

Além disso, utilizam-se taxas de execução diferenciadas para as rotinas de baixo nível, chamadas a intervalos de tempo menores que as funções de maior nível da interface e de controle. Portanto, permite-se que apenas esse nível lide adequadamente com os sinais de entrada e saída, disponibilizando a informação mais atual para os níveis superiores.

Por fim, o processo que constitui as etapas de Concepção e Implementação do método pode ser resumido pelo seguinte procedimento:

i) Modelar os subsistemas assíncronos AG_j do sistema produto SP, que constituem o modelo da planta e representam o comportamento livre do sistema a ser controlado.

ii) Definir e modelar as especificações de controle E_i que restringem o comportamento do SP ao desejado.

iii) Obter a composição síncrona (CURY, 2001) das especificações E_i com as subplantas locais GL_i a partir de SP, obtendo a linguagem alvo $K_i = E_i||L_m(GL_i)$.

iv) Sintetizar os supervisores modulares locais SL_i tais que $L_m(SL_i/GL_i) = SupC(K_i,L(GL_i))$ para cada especificação de controle.

v) Verificar a condição de não conflito para os supervisores modulares locais. Se não conflitantes ir para o passo seguinte, se conflitantes verificar estratégias de resolução do conflito (QUEIROZ, 2000).

vi) Obter redução dos supervisores através de algoritmos de redução (VAZ; WONHAM, 1986; SU; WONHAM, 2004) ou utilizando a especificação E_i quando K_i é controlável.

vii) Verificar o atendimento às propriedades nos supervisores a fim de detectar problemas de *escolha*, *sensibilidade ao delay* e *sensibilidade ao entrelaçamento*.

viii) Implementar os blocos da arquitetura da Figura 4.3, solucionando os problemas caso identificados no passo anterior. Nessa etapa parte dos blocos pode ser gerada automaticamente, como blocos de supervisores e sistema produto, sendo necessário ao programador apenas implementar o corpo das funções de interface que traduzem os sinais das rotinas de baixo nível em eventos.

Para fins de validação e maior detalhamento da estrutura de implementação o método é empregado para a obtenção do controle de temperatura em um estudo de caso no Capítulo 5.

Capítulo 5

Aplicação do Método em Estudo de Caso

Neste capítulo descreve-se a aplicação do método apresentado no Capítulo 4 para obtenção do controle de um refrigerador autônomo de dois compartimentos proposto como estudo de caso. Inicialmente se descreve o funcionamento e modelagem do problema e em seguida são detalhadas as estruturas implementadas utilizando linguagem ANSI C (KERNIGHAN; RITCHIE, 1988) permitindo que seja compilada para diferentes microcontroladores.

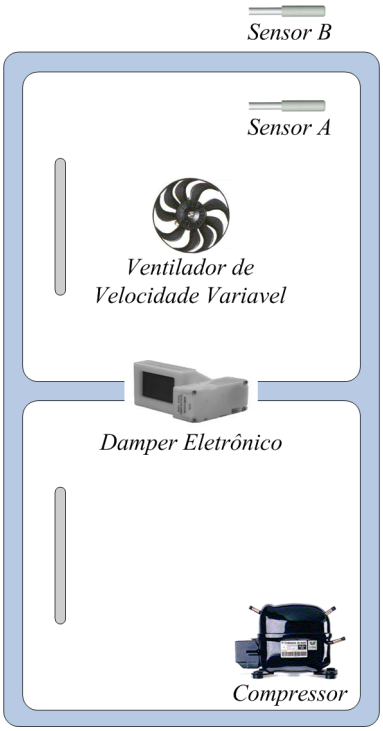
O código obtido foi embarcado em um controlador eletrônico utilizado comercialmente pela empresa Whirlpool Eletrodomésticos (WHIRLPOOL LATIN AMERICA, 2012) e testes realizados para validação do método e implementação adotados. A estrutura de arquivos utilizada como estratégia de implementação visa obter modularidade e possibilidade de reuso de código quando diferentes aplicações (problemas) são modeladas. Um gerador de código integrado como *plugin* do *software IDES3* (RUDIE, 2006) foi desenvolvido e utilizado para obtenção da estrutura de controle modular local.

5.1 Refrigerador Autônomo de Dois Compartimentos

Para demonstrar a aplicação do método descrito no Capítulo 4, propõe-se um estudo de caso baseado no controle de um refrigerador. O problema, inicialmente introduzido por Teixeira (2008), consiste em manter a temperatura interna de um refrigerador mesmo com alterações da temperatura externa, sem a necessidade de intervenção do usuário.

Teixeira (2008) considera um refrigerador com um único compartimento. Neste trabalho, novos elementos são adicionados para exemplificar e trazer à tona algumas características nos supervisores como as discutidas nos capítulos antecedentes. O refrigerador proposto neste estudo de caso é composto por dois compartimentos, tendo como elementos de controle um compressor, um ventilador de velocidade variável e um *damper* eletrônico. Dois sensores são utilizados para monitorar a temperatura do compartimento superior (*Sensor A*) e temperatura ambiente (*Sensor B*) conforme o diagrama da Figura 5.1.

Figura 5.1: Diagrama do Refrigerador adaptado de Teixeira (2008)



Fonte: Adaptado de Teixeira (2008)

5.1.1 Estratégia de Controle do Refrigerador Autônomo

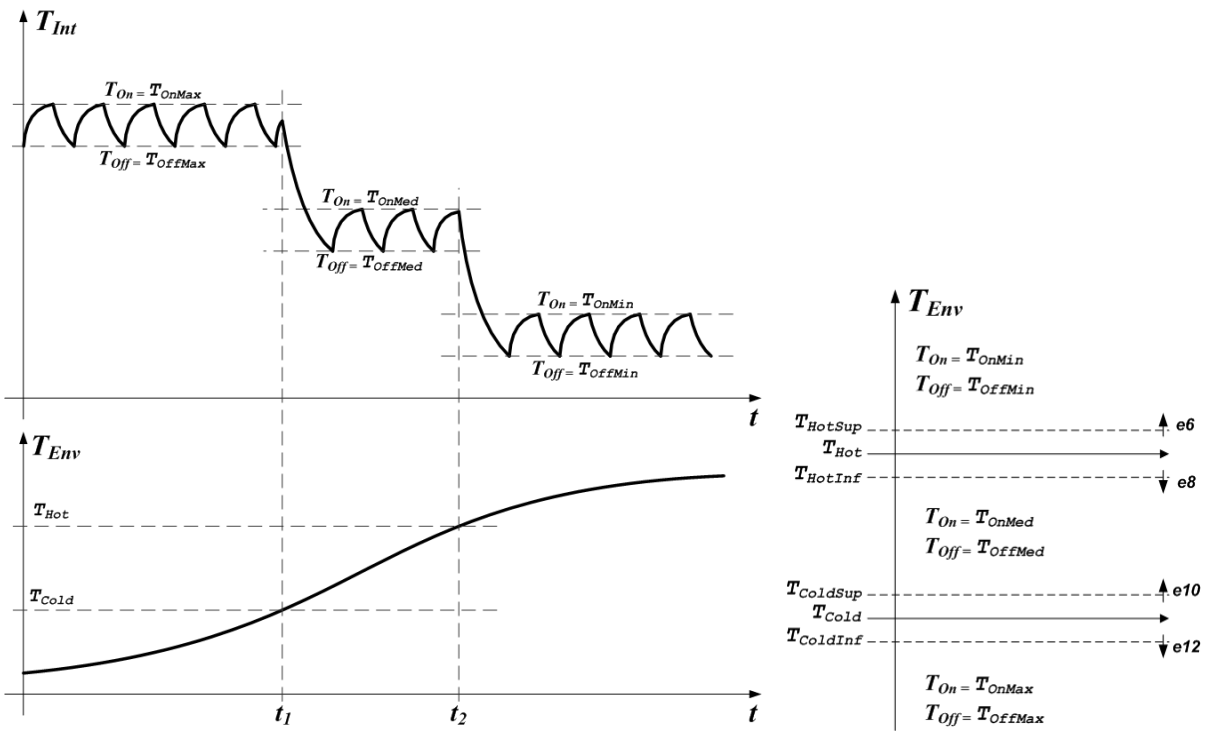
Para o controle da temperatura do compartimento superior o controlador embarcado aciona o compressor em ciclos de liga/desliga. De forma conjunta o controle coordena a abertura e fechamento do *damper*, assim como a velocidade do ventilador variável para manter a temperatura do compartimento inferior também insensível à variação da temperatura externa. A coordenação a ser garantida é basicamente manter o *damper* fechado enquanto o compressor estiver ligado e aberto quando o compressor estiver desligado. Ao mesmo tempo, o controle atua na velocidade do ventilador variável sendo esta maior quando o *damper* eletrônico está aberto.

O compressor é ligado no momento em que a temperatura interna *T_Int* atinge o valor de limiar (*threshold*) *T_On* e é desligado quando *T_Int* cruza o limiar *T_Off* (Figura 5.3a). Os limiares de temperatura *T_On* e *T_Off* assumem valores mínimos (*T_OnMin* e *T_OffMin*), médios (*T_OnMed* e *T_OffMed*) e máximos (*T_OnMax* e *T_OffMax*) conforme a variação da temperatura ambiente *T_Env* como ilustra a Figura 5.3b. Desta forma, com a temperatura ambiente fria (*T_Env* < *T_Cold*) o compressor apresenta ciclos de operação (liga e desliga) em temperaturas mais elevadas (*T_OnMax* e *T_OffMax*), enquanto que para uma temperatura externa elevada (*T_Env* > *T_Hot*) tais limiares são ajustados para valores mínimos (*T_OnMin*

e *T_OffMin*). A histerese em *T_Env* evita alterações bruscas nos limiares de operação para pequenas oscilações da temperatura ambiente.

O princípio fundamental por trás da operação autônoma é que a temperatura interna não depende apenas da carga térmica presente nos compartimentos e o quanto o compressor e *damper* operam, mas também da influência da temperatura ambiente na troca de calor necessária para refrigerar os compartimentos. Portanto, para manter-se uma mesma temperatura interna apesar de variações de temperatura no ambiente, a operação do compressor, determinada pelos valores limiares *T_On* e *T_Off*, precisam ser alteradas.

Figura 5.2: (a) Comportamento autônomo de acordo com a variação da temperatura ambiente (b) Histerese para a temperatura ambiente definindo os *setpoints* de funcionamento do compressor



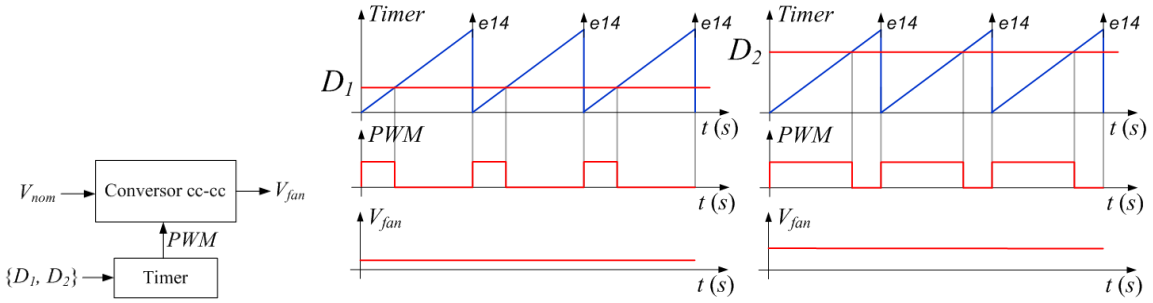
(a) (b)

Fonte: (TEIXEIRA, 2008)

O controle da velocidade do ventilador é obtido pela variação da amplitude da tensão Corrente Contínua (CC) aplicada a ele. A tensão CC V_{Fan} é obtida a partir da tensão fixa de referência V_{Nom} através de um conversor CC-CC cujo diagrama é mostrado na Figura 5.4a. O conversor recebe um sinal de comando *Pulse-Width Modulation* (PWM) de acordo com uma razão cíclica *D*(também chamado de ciclo de trabalho, do inglês *duty cycle*), sendo a tensão de saída proporcional a *D* dada por $V_{Fan} = D.V_{Nom}$. O sinal PWM é gerado por um *temporizador*

periférico do microcontrolador. O ciclo de trabalho D pode ser atualizado apenas no início de cada ciclo do sinal PWM desta forma evita-se a ocorrência de vários pulsos PWM no mesmo ciclo de chaveamento. Com a alteração no valor de comparação do sinal do temporizador, ou seja, mudança na razão cíclica, obtém-se diferentes valores médios de tensão V_{Fan} resultando em diferentes velocidades, como ilustrado pelos sinais da Figura 5.4b.

Figura 5.3: Diagrama do comando PWM do conversor CC-CC para acionamento do ventilador



(a) (b)

Fonte: produção do próprio autor

5.1.2 Descrição dos Eventos e Modelos da Planta

Um dos primeiros passos para a síntese de supervisores para a solução de um problema de controle supervisório (RAMADGE; WONHAM, 1989) de SED é a definição dos eventos que descrevem o comportamento do sistema a ser controlado. A TCS distingue tais eventos entre controláveis e não controláveis. Eventos controláveis são aqueles que podem ser desabilitados pela ação de controle sobre a planta. Por outro lado, os eventos não controláveis não podem ser desabilitados, ou seja, sua ocorrência não pode ser proibida pela ação do controlador.

A Tabela 5.1 apresenta os eventos não controláveis definidos para a modelagem do refrigerador autônomo de dois compartimentos.

Os eventos controláveis para o refrigerador autônomo com dois compartimentos são descritos na Tabela 5.2.

A definição dos eventos é realizada em conjunto com a modelagem do comportamento livre dos subsistemas existentes na planta a ser controlada. Observa-se os elementos (subplantas) que compõem o sistema para definir o comportamento individual de tal forma que a reunião (ou sincronização) dos modelos dos elementos descreverá o comportamento livre da planta como um todo.

Tabela 5.1: Relação dos Eventos Não Controláveis do Refrigerador Autônomo

Evento	**Descrição**
e2	*T_Int* maior que o limiar determinado por *T_On*
e4	*T_Int* menor que o limiar determinado por *T_Off*
e6	*T_Env* maior que o limiar determinado por *T_HotSup*
e8	*T_Env* menor que o limiar determinado por *T_HotInf*
e10	*T_Env* maior que o limiar determinado por *T_ColdSup*
e12	*T_Env* menor que o limiar determinado por *T_ColdInf*
e14	Término de um período do temporizador do PWM do ventilador

Fonte: produção do próprio autor

Tabela 5.2: Relação dos Eventos Controláveis do Refrigerador Autônomo

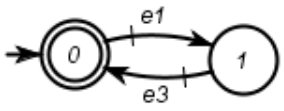
Evento	**Descrição**
e1	Liga compressor
e3	Desliga compressor
e5	Seleciona *T_On=T_OnMax* e *T_Off=T_OffMax*
e7	Seleciona *T_On=T_OnMed* e *T_Off=T_OffMed*
e9	Seleciona *T_On=T_OnMin* e *T_Off=T_OffMin*
e11	Abre *damper*
e13	Fecha *damper*
e15	Seleciona razão cíclica D_1 para comando PWM do ventilador
e17	Seleciona razão cíclica D_2>D_1 para comando PWM do ventilador

Fonte: produção do próprio autor

O compressor é o principal elemento de controle do sistema de refrigeração, responsável pela circulação do gás refrigerante que realiza a troca de calor com os compartimentos do produto (EMBRACO, 2012). Nesta aplicação, os comandos de acionamento e desligamento do compressor são modelados como descrito no modelo da Figura 5.4.

Apesar de não representar um elemento físico do sistema, a seleção dos limiares de temperatura da ciclagem do compressor são importantes pois determinam os valores de temperatura máximos, médios e mínimos que, por sua vez, estabelecem a operação do compressor. A ação de seleção é modelada pelo autômato da Figura 5.5. O evento *e5* corresponde a ação

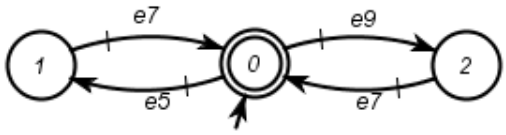
Figura 5.4: G_1 Modelo do compressor



Fonte: produção do próprio autor

de seleção dos valores máximos de *setpoint T_On=T_OnMax* e *T_Off=T_OffMax*, assim como *e7* seta *T_On=T_OnMed* e *T_Off=T_OffMed* e por fim *e9* que seleciona *T_On=T_OnMin* e *T_Off=T_OffMin*.

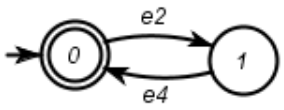
Figura 5.5: G_2 Modelo para a seleção dos *setpoints* de funcionamento do compressor



Fonte: produção do próprio autor

O *Sensor A* posicionado no evaporador do refrigerador mede a temperatura interior *T_Int* do refrigerador. O aumento da temperatura acima do limiar determinado por *T_On* é capturado pela ocorrência do evento *e2*. Já o evento *e4* ocorre quando a temperatura interna *T_Int* ultrapassa o limite inferior definido por *T_Off*, conforme o modelo representado pelo autômato da Figura 5.6.

Figura 5.6: G_3 Modelo do *Sensor A* posicionado no evaporador

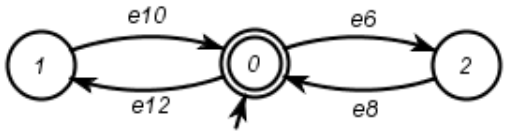


Fonte: produção do próprio autor

O autômato da Figura 5.7 modela o comportamento do *Sensor B*. Os eventos considerados na modelagem desse elemento são referentes às mudanças na temperatura ambiente *T_Env* quando esta ultrapassa valores limiares estabelecidos. Para se evitar a ocorrência de eventos devido a pequenas oscilações da temperatura ambiente ou por erros de medição determinam-se histereses em torno das temperaturas que definem um ambiente estar quente ou frio. Os eventos modelados são: *e6* quando *T_Env* torna-se maior que o limiar determinado por *T_HotSup*; *e8* se *T_Env* menor que *T_HotInf*, quando a temperatura externa deixa de ser considerada como

quente; *e10* quando a temperatura *T_Env* torna-se mais quente que *T_ColdSup*; *e12* no momento em que *T_Env* alcança um valor inferior à *T_ColdInf*.

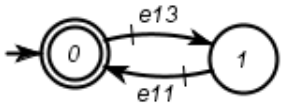
Figura 5.7: G_4 Modelo do *Sensor B* para medição da temperatura ambiente



Fonte: produção do próprio autor

De uma forma similar ao compressor, considera-se para o modelo do *damper* eletrônico os eventos de abrir (*e11*) e fechar (*e13*). Este dispositivo é utilizado para regular a troca de calor e circulação do ar entre os compartimentos do refrigerador, sendo seu modelo apresentado na Figura 5.8.

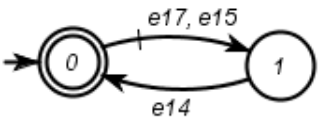
Figura 5.8: G_5 Modelo do *damper* eletrônico



Fonte: produção do próprio autor

O modelo para o ventilador de velocidade variável, representado pelo autômato da Figura 5.9, considera o acionamento via comando PWM descrito anteriormente. O evento *e15* determina que o ciclo de trabalho do PWM seja $D = D_1$. Para obtenção de uma maior velocidade do ventilador, o evento *e17* seleciona $D = D_2$, sendo que $D_2 > D_1$. Para que uma nova razão cíclica seja dada ao comando PWM apenas no término de um período, o término do temporizador que controla o comando é acusado pela ocorrência do evento não controlável *e14*.

Figura 5.9: G_6 Modelo do ventilador de velocidade variável



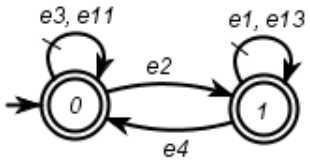
Fonte: produção do próprio autor

Os modelos das plantas foram construídos utilizando o *software IDES* (RUDIE, 2006), também utilizado para os modelos das especificações de controle e cálculo dos supervisores.

5.1.3 Modelos das Especificações de Controle

A especificação de controle modelada pelo autômato E_1 da Figura 5.10 determina que o compressor está habilitado a ligar e o *damper* a fechar quando *T_Int* atingir um valor superior à *T_On*. Também permite que o controlador desligue o compressor e abra o *damper* caso *T_Int* se torne inferior à *T_Off*.

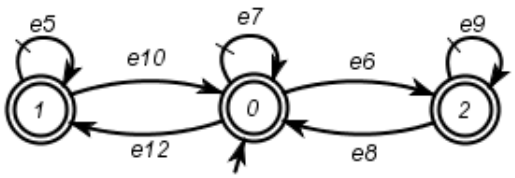
Figura 5.10: E_1 Modelo da especificação para a relação de *T_Int* com o compressor e *damper*



Fonte: produção do próprio autor

O modelo da Figura 5.11 especifica a mudança autônoma nos valores dos limiares *T_On* e *T_Off* de acordo com a variação da temperatura ambiente *T_Env*. Por exemplo, quando a temperatura ambiente *T_Env* ultrapassa o valor de limiar *T_HotSup* (evento *e6*) a especificação de controle E_2 proíbe a ocorrência dos eventos de seleção de valores médios e máximos, permitindo apenas que o os valores mínimos de temperatura *T_OnMin* e *T_OffMin* sejam atribuídos, para que o refrigerador opere em valores baixos em dias quentes.

Figura 5.11: E_2 Modelo da especificação de controle para a seleção do *setpoint* com a variação de *T_Env*

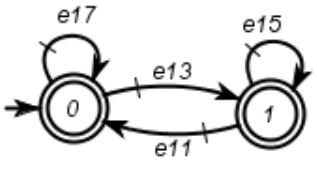


Fonte: produção do próprio autor

A terceira especificação de controle determina a velocidade do ventilador que força a circulação de ar no refrigerador (Figura 5.12). Com o fechamento do *damper* (evento *e13*) o ar circula apenas pelo compartimento superior, exigindo uma menor velocidade do ventilador (evento *e15*), portanto a especificação permite *e15* após a ocorrência do evento *e13*. Com a abertura do *damper* (evento *e11)*, a velocidade do ventilador deve ser maior (evento *e17*). A especificação E_3 dessa maneira permite o evento *e17*, forçando a circulação de ar entre os dois

compartimentos com uma maior rotação no ventilador.

Figura 5.12: E_3 Modelo da especificação da velocidade do ventilador de acordo com o estado do *damper*



Fonte: produção do próprio autor

5.1.4 Síntese dos Supervisores

Seguindo a abordagem Modular Local (QUEIROZ, 2000) de síntese de supervisores os modelos do subsistemas devem ser assíncronos, ou seja, subsistemas cujo alfabeto compartilhe um mesmo evento devem ser sincronizadas (reunidas através da operação de composição síncrona), formando a mais refinada RSP. Os autômatos que modelam os elementos da planta deste estudo de caso são assíncronos, compondo a RSP.

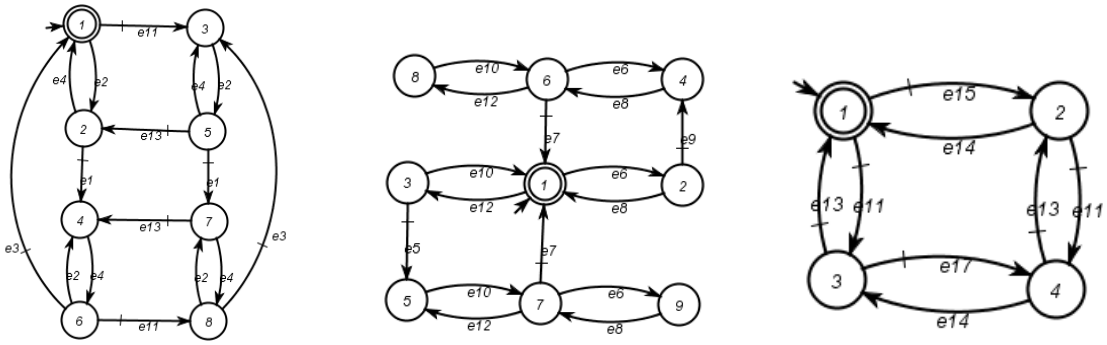
Para cada especificação de controle são definidas as denominadas plantas locais LG_i obtidas pela composição síncrona dos elementos de planta cujo alfabeto contém pelo menos um evento também pertencente ao alfabeto da especificação de controle (QUEIROZ, 2000). Dessa forma, as plantas locais definidas neste estudo são: $GL_1 = G_1||G_3||G_6$, $GL_2 = G_2||G_4$ e $GL_3 = G_5||G_6$.

Em seguida, a linguagem alvo é calculada por $K_i = E_i||GL_i$ com $i = 1,2,3$. Por fim, com o uso do cálculo *SupCon* do *software* IDES (RUDIE, 2006) obtém-se a máxima sublinguagem de K_i que é controlável em relação à GL_i dada por $L_m(S_i/GL_i) = SupC(GL_i,K_i)$. Os autômatos para os supervisores S_i são apresentados na Figura 5.13.

O teste da modularidade (QUEIROZ, 2000) realizado pela verificação da ausência de estados não acessíveis e co-acessíveis no autômato resultante da composição síncrona dos supervisores obtidos (operação *trim*$(S_1||S_2||S_3)$) determina que os supervisores locais obtidos são não conflitantes.

O método apresentado no Capítulo 4 considera a implementação de supervisores reduzidos por estes terem a mesma ação de controle com menor número de estados que supervi-

Figura 5.13: Supervisores obtidos para o controle do Refrigerador Autônomo



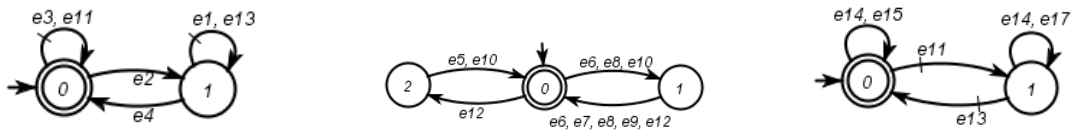
(a) S_1 (b) S_2 (c) S_3

Fonte: produção do próprio autor

sores completos. É possível verificar que neste problema as linguagens alvo K_i são controláveis em relação às plantas sob supervisão. Uma maneira simples de realizar tal verificação é pela comparação do número de estados e transições da linguagem alvo após a operação *trim()*, que remove estados não acessíveis e co-acessíveis (CURY, 2001), com o número de estados e transições do supervisor, ou seja, se $trim(K_i) = S_i$ então K_i é controlável.

Nos casos em que K_i é controlável a própria especificação de controle pode ser usada como supervisor reduzido. Embora esta condição seja verdadeira para todos os K_i desse estudo de caso, para fins de comparação os supervisores S_i foram reduzidos por meio do algoritmo de redução *supreduce()* disponibilizado pelo *software TCT* (FENG; WONHAM, 2006). Os autômatos dos supervisores reduzidos obtidos encontram-se na Figura 5.14.

Figura 5.14: Supervisores reduzidos com o uso do TCT (FENG; WONHAM, 2006)



(a) SL_1 (b) SL_2 (c) SL_3

Fonte: produção do próprio autor

Observa-se que SL_1 é idêntico à especificação de controle E_1, mas os supervisores reduzidos SL_2 e SL_3, embora contenham o mesmo número de estados que as especificações E_2 e E_3, contém mais transições. Quando isso acontece é mais vantajoso implementar as especificações de controle sobre os supervisores reduzidos, uma vez que quanto maior o número

de transições presentes no supervisor maior o consumo em memória da implementação.

5.2 Implementação do Controle Embarcado

O método de implementação apresentado no Capítulo 4 é genérico o suficiente para permitir diferentes formas de implementar os blocos propostos assim como é flexível a linguagem de programação que pode ser utilizada.

Para tornar a implementação mais próxima à utilizada industrialmente para o controle de eletrodomésticos, o controle do refrigerador de dois compartimentos com operação autônoma foi implementado em linguagem ANSI C (KERNIGHAN; RITCHIE, 1988) para ser embarcado em um microcontrolador de 8 *bits* da família STM8 (STMICROELECTRONICS, 2012).

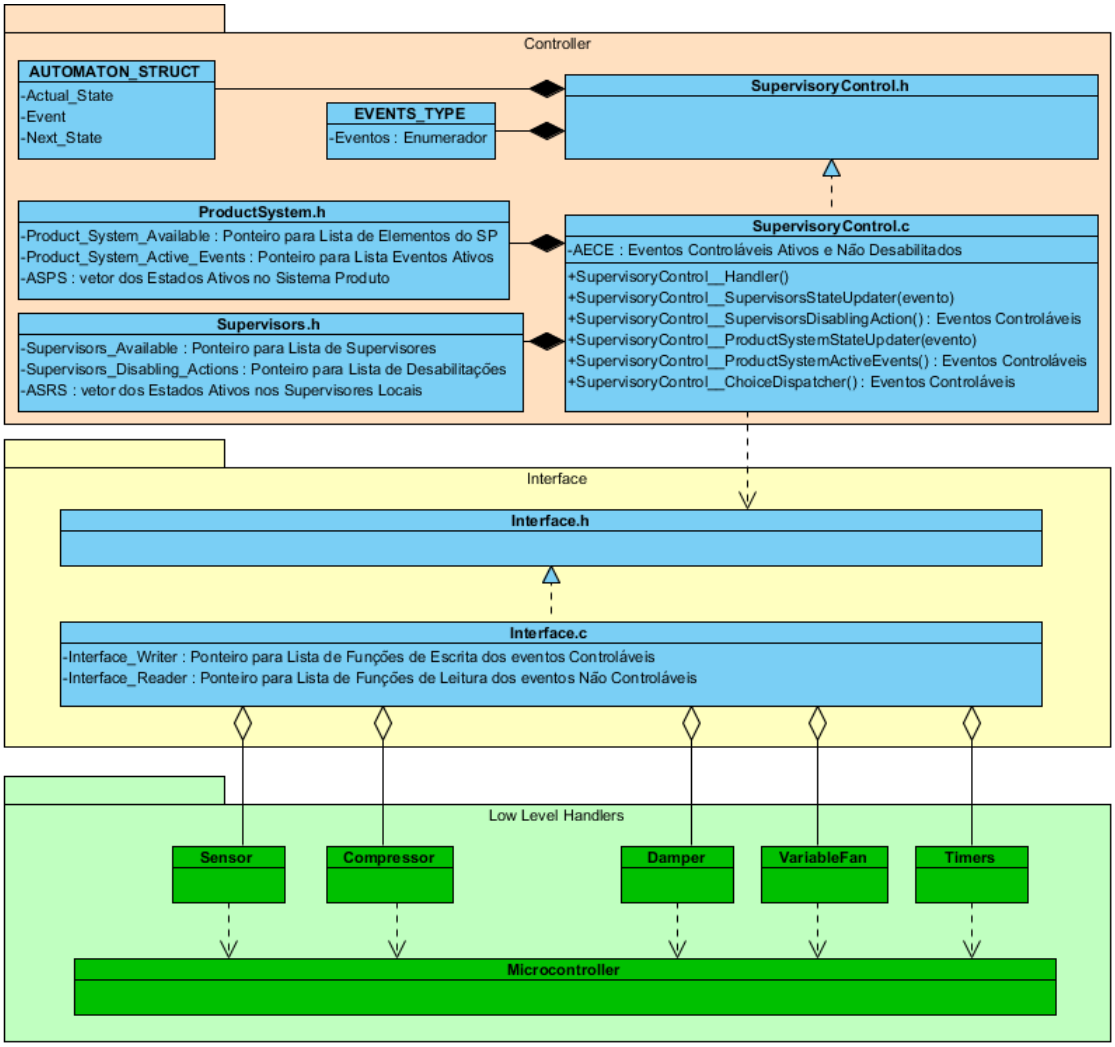
Segundo Barreta e Torrico (2008) mesmo definida a linguagem de programação ainda existem diferentes maneiras de se implementar supervisores como as que utilizam, vetores, matrizes, listas encadeadas ou ainda outras estruturas de dados. A implementação do refrigerador autônomo baseia-se no uso misto de vetores, matrizes e listas de ponteiros, utilizando-se das estruturas definidas por Teixeira (2008) para os modelos de supervisores e sistema produto. Além dessas, acrescenta-se o uso de ponteiros para funções para criar um código principal modular, que pode ser reutilizado independente da aplicação.

A estrutura de implementação da Figura 4.3 prevê a divisão do *software* em três camadas *Controle*, *Interface* e *Rotinas*. Cada uma dessas camadas contém blocos a serem implementados para obtenção do controle supervisório.

O diagrama de classe (BOOCH et al., 2006) da Figura 5.15 apresenta a estrutura de arquivos da implementação em linguagem ANSI C utilizada para obtenção do controle do refrigerador autônomo de dois compartimentos proposto para este estudo de caso. O diagrama é utilizado como representação dos diferentes módulos existentes na implementação, pois a linguagem C não é orientada a objetos.

No nível do *Controle* encontra-se o módulo principal ***Supervisory Control***. Neste módulo são codificadas as funções que realizam as tarefas dos blocos de atualização de estados de supervisores e sistema produto, análise das situações de escolha e complemento das desabilitações por parte do determinador. Além dessas, implementa-se uma *função principal* responsável pela chamada de todas as funções para que se obtenha a sequência de execução desejada como discutido na apresentação do método (Capítulo 4). Dessa forma, pode-se estruturar o programa

Figura 5.15: Diagrama de Classes em UML da Estrutura dos Arquivos



Fonte: produção do próprio autor

do microcontrolador para periodicamente executar a função principal do módulo *Supervisory Control* e este coordena a execução de todas as tarefas seguintes.

Como se observa na Figura 5.15 o módulo *Supervisory Control* contém dois arquivos, *SupervisoryControl.h* e *SupervisoryControl.c*. O arquivo de cabeçalho *SupervisoryControl.h* contém os protótipos das funções, a declaração da estrutura de dados dos autômatos e a lista (enumerador) de eventos. O segundo arquivo *SupervisoryControl.c* é a realização do módulo, arquivo no qual as funções são codificadas. Estes arquivos constituem o módulo principal da arquitetura de implementação que é codificado de forma a ser independente da aplicação modelada.

Na camada do *Controle* ainda são definidas as estruturas de dados para implementação dos modelos dos supervisores e sistema produto. Nesta implementação utilizam-se arquivos

separados para a implementação das estruturas que representam os modelos dos supervisores em *Supervisors.h* e dos elementos do sistema produto em *ProductSystem.h*. Os arquivos são incluídos em *SupervisoryControl.c* que contém as funções que utilizam as informações dos modelos para executar as transições e análise das desabilitações.

A *Interface* é constituída pelo módulo de mesmo nome, dado pelo arquivo cabeçalho (*header*) *Interface.h* e pelo arquivo *Interface.c* o qual contém a implementação das funções. Na interface faz-se a conversão dos valores lidos e escritos nas variáveis de entrada e saída do sistema de controle em eventos modelados no sistema produto. As funções de leitura e escrita da *Interface* são chamadas pela *função principal* do módulo *Supervisory Control* nos momentos apropriados, sendo este portanto dependente da camada de interface.

Ao executar as funções de leitura e escrita da *Interface* realiza-se a interpretação de variáveis, principalmente de módulos na camada de mais baixo nível (*Rotinas de Baixo Nível*) e adicionam-se eventos em *buffers*. Nessa implementação utiliza-se um *buffer* para eventos controláveis e um segundo para eventos não controláveis. Dessa forma, a *função principal* pode priorizar a verificação e atualização de estados inicialmente por parte dos eventos não controláveis analisando todos os eventos armazenados em um *buffer* por vez, utilizando uma ordem de análise do tipo *First In First Out* (FIFO).

A terceira e última camada da implementação do controle é denominada de *Rotinas de Baixo Nível*. Para este estudo de caso esta camada foi inteiramente obtida pelo reuso de módulos previamente existentes nas bibliotecas de desenvolvimento de *software* da Whirlpool Eletrodomésticos. Como objetivo secundário este estudo de caso, além de validar o método proposto, propõe-se apresentar uma maneira de empregar o controle obtido através da TCS em uma aplicação industrial. Os elementos (cargas) controlados nos sistemas de refrigeração possuem características de acionamento distintas e, em diversos casos, são necessárias rotinas de auto verificação para identificação de erros e/ou proteção para evitar falhas críticas e garantir segurança na interação do usuário com o produto. Tais rotinas são previstas pelo método de aplicação apresentado no Capítulo 4 na camada de baixo nível. Nesse caso, os módulos para as cargas contemplam rotinas que foram abstraídas da modelagem para obtenção do controle, mas exercem função fundamental para garantia de funcionamento do produto.

Nessa implementação as chamadas de funções são executadas utilizando uma estrutura conhecida como *time slice* que se baseia na execução de tarefas a taxas de tempo definidas (TANENBAUM, 2009). Dessa forma funções de mais alto nível, como a *função principal* do

módulo *Supervisory Control* e outras funções de aplicação, são executadas com intervalos de tempo maiores que as funções de baixo nível, por exemplo, as diretamente ligadas aos periféricos do microcontrolador na camada *Rotinas de Baixo Nível*.

A Figura 5.15 apresenta os módulos de baixo nível acessados pela interface para gerenciar a leitura e o acionamento das cargas para então armazenar a ocorrência dos eventos modelados.

A utilização dessa implementação distribuída em camadas e estruturação dos arquivos traz alguns benefícios como modularidade e facilidade no reuso que serão destacados a seguir.

5.2.1 Estrutura de arquivos utilizando linguagem ANSI C

Conforme se aumenta a demanda pela utilização de *software* embarcado para o controle de sistemas, como por exemplo de eletrodomésticos, também se observa a necessidade da redução no tempo de desenvolvimento de tais aplicações (SANGIOVANNI-VINCENTELLI; MARTIN, 2001). Portanto, busca-se estruturar o *software* de maneira modular para que a utilização de códigos previamente desenvolvidos possam ser reaproveitados em aplicações futuras. Uma maior relevância ainda é observada quando se analisa a necessidade de testes que são executados para validação de *software*. A reutilização de módulos eleva o grau de confiabilidade do código uma vez que será testado em diferentes aplicações, permitindo que, em alguns casos, o teste seja inteiramente voltado à integração do módulo à nova aplicação e novamente o tempo de desenvolvimento é reduzido.

Nesse contexto, a implementação desenvolvida a partir do método discutido no Capítulo 4 teve a estrutura organizada para apresentar modularidade, no sentido de independência com estruturas específicas de uma aplicação, e como consequência ter a possibilidade de reutilização para diversas aplicações.

O módulo principal *Supervisory Control* é codificado de forma a não depender do número de elementos que constituem o sistema produto e do número de supervisores presentes em uma determinada aplicação. Cada modelo de supervisor e planta do sistema produto pode ter diferentes números de estados e transições sem que mudanças nas funções deste módulo sejam necessárias. As funções de atualização de estados precisam identificar nos diversos modelos as transições a serem realizadas. Da mesma forma, estas funções são implementadas para não depender das características específicas dos modelos. A independência do módulo *Supervisory*

Control, conforme é discutido ao longo do exemplo, permite que este seja gerado apenas uma vez e que diversas aplicações, ou seja, diferentes problemas cujo controle é obtido pela TCS, possam ser executados sem a necessidade de alteração nessa parte do código. Tal característica facilita a criação de um gerador automático de código. Um *plugin* para o IDES (RUDIE, 2006) gerador dessas rotinas foi desenvolvido e detalhes de sua utilização são encontrados nos anexos deste trabalho.

Além de permitir o uso em aplicações distintas, a estrutura concentra eventuais alterações em modelos apenas em arquivos específicos. A alteração na modelagem de uma especificação de controle, por exemplo, exige que apenas o arquivo *Supervisor.h* seja atualizado com o novo modelo. Assim como a adição de um novo elemento no sistema produto demanda a atualização de *ProductSystem.h*.

No caso em que uma nova estrutura de dados para a implementação da modelagem dos autômatos é necessária (ou desejada) é preciso a reformulação das funções de transição presentes no módulo *Supervisory Control*, mas novamente, sendo atualizado uma vez poderá ser reutilizado para diversas implementações do controle supervisório.

Como mencionado anteriormente, utilizaram-se módulos pré existentes como funções de baixo nível. Cabe um destaque especial para a implementação das funções mais básicas relacionadas ao uso dos periféricos do microcontrolador, representada pela classe *Microcontroller* no diagrama da Figura 5.15. As classes da camada de *Rotinas* que interagem com os periféricos do microcontrolador, tais como conversor AD, pinos de entrada e saída, PWM, utilizam interfaces padrão que independem do microcontrolador para o qual o código será compilado. Dessa maneira uma mesma implementação pode ser embarcada em diferentes microcontroladores, pela simples alteração da classe *Microcontroller* e compilação para tal plataforma.

Ainda pelo uso de uma estrutura em camadas é possível dar flexibilidade também às interfaces. Em conjunto com o uso dos módulos de baixo nível algumas das funções de interface de escrita auxiliam na sincronização entre geração de eventos na planta e atualização de tais eventos nos modelos do controle. O problema da *causalidade* abordado nos capítulos anteriores, é solucionado uma vez que mesmo eventos controláveis apenas são gerados por intermédio da interface, quando ocorreram na planta. Um exemplo utilizado no estudo de caso é o do módulo do compressor. Este módulo contém rotinas de proteção para que não se permita um desligamento e um súbito religamento do compressor. Neste caso, quando por exemplo o evento controlável *e1* não está desabilitado a rotina de escrita desse evento será chamada e uma solicitação para ligar

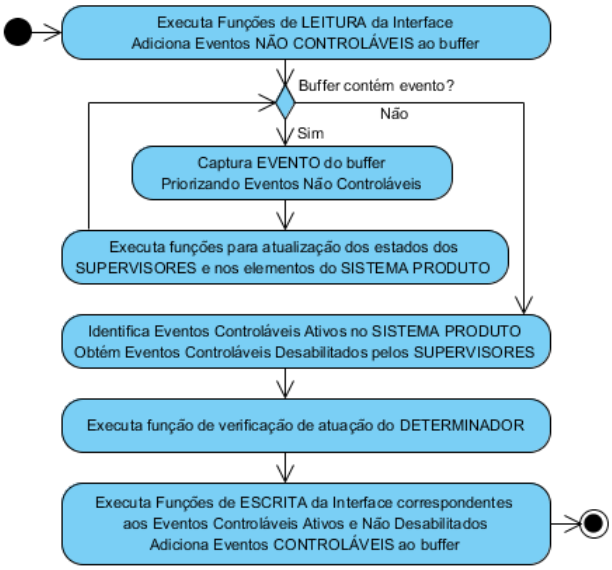
o compressor é iniciada no módulo do compressor. Essa mesma função de escrita na interface verifica o estado retornado pelo módulo compressor e apenas adiciona o evento *e1* ao *buffer* de eventos controláveis se este indica que o compressor está ligado, o que reflete com maior fidelidade o acionamento físico na planta. Se a rotina de baixo nível estiver no tempo de proteção do compressor a função de escrita continua sendo chamada até que haja efetiva ocorrência do evento controlável, o que reflete o real acionamento da carga na planta física.

5.2.2 Detalhamento da implementação em linguagem ANSI C

Nesta seção são apresentados com maiores detalhes os aspectos da implementação que permitem que apenas parte do código obtido a partir dos modelos da TCS deva ser gerado para cada problema ou alteração.

O módulo *Supervisory Control* contém a *função principal* que determina a ordem de chamada e execução das rotinas como se prevê no método de implementação do Capítulo 4. A Figura 5.16 apresenta um diagrama de atividades com esta ordem de execução. A função é periodicamente executada pelo módulo principal e é representa com uma atividade final, não com um *loop* para a tarefa inicial, pois o programa executa outras funções e módulos.

Figura 5.16: Função principal do módulo *Supervisory Control*

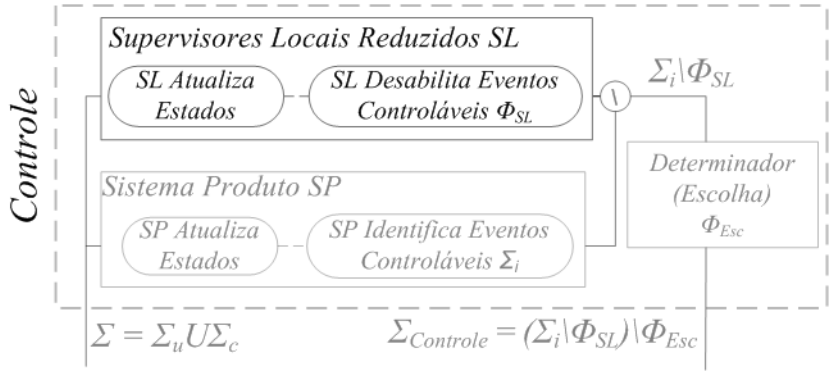


Fonte: produção do próprio autor

Para que o código do módulo *Supervisory Control* seja independente da aplicação que está sendo modelada, as estruturas estabelecidas pelo método de implementação precisam ser independentes das características dos modelos, como por exemplo, número de estados e

transições, assim como da própria quantidade de modelos. Para o bloco dos supervisores na camada *Controle* (Figura 5.17) são implementadas duas funções, uma para realizar a tarefa de atualização dos estados dos modelos dos supervisores e outra para obtenção da desabilitação de eventos controláveis determinada pelos estados ativos nos supervisores.

Figura 5.17: Supervisores na Camada *Controle*



Fonte: produção do próprio autor

Na Figura 5.18 encontra-se o código da função implementada para atualização de estados nos supervisores locais. Nessa é possível observar que o processo de varredura nos diversos supervisores ocorre pela execução de um *loop* que incrementa o ponteiro `Supervisors_Available[]`, de tal forma que este código é exatamente o mesmo para qualquer número de supervisores modelados.

O maior ganho com esta forma de implementação em relação às abordagens de atualização utilizadas em trabalhos como de Costa (2005), Teixeira (2008), Carvalho (2007), Silva (2010) é que nestes casos, para cada aplicação funções específicas para cada supervisor são necessárias uma vez que dependem das características do modelo como número de transições e eventos envolvidos. A utilização de funções independentes permite que grande parte do código não necessite ser gerado quando diferentes modelos (problemas) são estudados e implementados. O uso de uma única função reduz a complexidade na função principal que necessita chamar apenas uma função para atualizar todos os supervisores ao invés de requerer a chamada de diferentes funções, ou seja, de protótipos diferentes.

A mesma abordagem é utilizada na função para obtenção dos eventos controláveis desabilitados pelos supervisores mostrada na Figura 5.19, mas nesta o ponteiro `Supervisors_Disabling_Actions[]` é utilizado para percorrer vetores que contêm os eventos desabilitados para cada estado de cada um dos supervisores locais.

Figura 5.18: Código ANSI C da Função de Atualização de Estados Supervisores

```
void SupervisoryControl__SupervisorsStateUpdater(EVENTS_TYPE event)
{
 unsigned char supIdx;
 unsigned char transIdx;

 //Search all supervisors
 for(supIdx = 0; supIdx < SUPERVISORS_NUM; supIdx++)
 {
  //For each supervisor, check all transitions
  for(transIdx = 0; transIdx < Supervisors_Transitions[supIdx]; transIdx++)
  {
   //Find actual state
   if(Supervisors_Available[supIdx][transIdx].Actual_State == ASRS[supIdx])
   {
    //Find if "event" match with an available event of this state
    if(Supervisors_Available[supIdx][transIdx].Event == event)
    {
     //Make state transition. Update state.
     ASRS[supIdx] = Supervisors_Available[supIdx][transIdx].Next_State;

     break; //one transition for this supervisor. Move to next supervisor.
    }
   }
  }
 }
}
```

Fonte: produção do próprio autor

Figura 5.19: Código ANSI C da Função de Obtenção das Desabilitações dos Supervisores

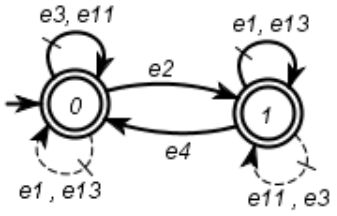
```
CONTROLLABLE_EVENTS SupervisoryControl__SupervisorsDisablingAction(void)
{
 unsigned char sup_idx;
 //Globally Disabled Controllable Events
 CONTROLLABLE_EVENTS GDCE = 0b111111111;

 for(sup_idx = 0; sup_idx < SUPERVISORS_NUM; sup_idx++)
 {
  GDCE &= Supervisors_Disabling_Actions[sup_idx][ASRS[sup_idx]];
 }

 return (GDCE);
}
```

Fonte: produção do próprio autor

Como mencionado anteriormente a descrição dos modelos dos supervisores é implementada somente no arquivo *Supervisors.h* (Figura 5.15). Neste arquivo são adicionadas as estruturas que definem as transições de estado nos autômatos e os vetores com as desabilitações

dos eventos controláveis por estado. A Figura 5.20 traz como exemplo as estruturas utilizadas para a descrição do Supervisor SL_1. Note que são implementados apenas as transições que efetivamente resultam em uma alteração de estado, ou seja, quando a ocorrência do evento resulta na alteração da ação de controle do supervisor. Para a construção da lista de eventos controláveis desabilitados por estado utiliza-se um valor binário onde cada *bit* corresponde a um evento controlável. Caso o *bit* tenha o valor 0 (zero) o evento controlável correspondente está desabilitado no estado do supervisor. Eventos controláveis não pertencentes ao alfabeto do supervisor devem ser mantidos habilitados pelo supervisor. A ordem dos eventos do *bit* mais significativo para o menos é *e1*, *e11*, *e13*, *e15*, *e17*, *e3*, *e5*, *e7* e *e9*.

Figura 5.20: Implementacao do Supervisor SL_1



(a) Modelo SL_1

```
//Define supervisor constants in a Matrix
    format
const AUTOMATON_STRUCT Supervisor_E1[] = {
  {0, e2, 1},
  {1, e4, 0}};

// Supervisors Disabling Actions.
// Event Disabled:0, Enabled:1
const CONTROLLABLE_EVENTS_STRUCT DCERS_E1[]
     = {
 0b010111111,
 0b101110111};
```

(b) Código ANSI C para implementação do supervisor

Fonte: produção do próprio autor

Além das estruturas para cada supervisor modular local existente no problema, o arquivo *Supervisors.h* define as listas (ponteiros) e o vetor utilizado para armazenar o estado ativo de cada supervisor que serão utilizados pelas funções (Figura 5.18 e Figura 5.19) do módulo *SupervisoryControl*. Para o estudo de caso do refrigerador autônomo de dois compartimentos os elementos são definidos como apresentado na Figura 5.21.

A implementação das funções dos elementos do sistema produto para atualização de estado, obtenção dos eventos controláveis ativos na planta e as estruturas de descrição dos modelos seguem a mesma abordagem utilizada na codificação dos supervisores.

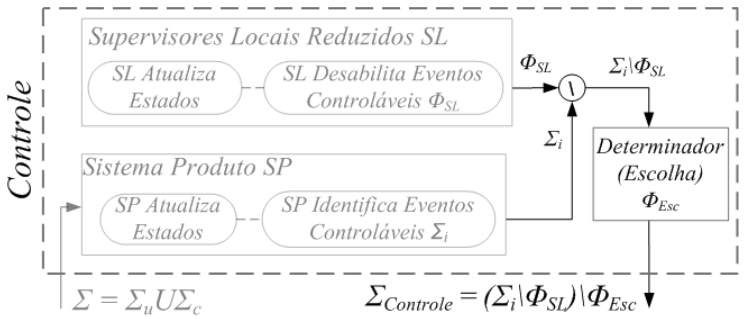
Seguindo o fluxo de execução da função principal (Figura 5.16) após a identificação dos eventos controláveis ativos pela planta e a obtenção da ação de desabilitação dos supervisores executa-se a análise do problema da escolha e possível atuação do *determinador*. Como previsto no método de implementação (Figura 5.22) a situação de escolha é analisada sobre o resultado da ação de desabilitação dos supervisores nos eventos controláveis ativos pela planta.

Figura 5.21: Código ANSI C das Listas e variável de estado atual de *Supervisors.h*

```
//Constant pointers pointing to each available Supervisor.
AUTOMATON_STRUCT *const Supervisors_Available[] = {
 (AUTOMATON_STRUCT *const) Supervisor_E1,
 (AUTOMATON_STRUCT *const) Supervisor_E2,
 (AUTOMATON_STRUCT *const) Supervisor_E3
};

//Constant pointers pointing to each Supervisor Disabling action vector
    .
CONTROLLABLE_EVENTS *const Supervisors_Disabling_Actions[] = {
 (CONTROLLABLE_EVENTS *const) DCERS_E1,
 (CONTROLLABLE_EVENTS *const) DCERS_E2,
 (CONTROLLABLE_EVENTS *const) DCERS_E3
};

//Declare vector ASRS which holds the supervisor actual state.
STATE_VAR_TYPE ASRS[SUPERVISORS_NUM] = {0, 0, 0};
```

Fonte: produção do próprio autor

Figura 5.22: Verificação da Escolha na camada *Controle*



Fonte: produção do próprio autor

A função de análise é sempre executada a partir da *função principal*, mas a função apenas retorna uma desabilitação diferente caso a situação de escolha seja detectada. A Figura 5.23 apresenta o trecho de código da *função principal* para obtenção da ação de controle $\Sigma_{Controle}$ (Figura 5.22), no código denominada de `Control_Action`. A ação de controle é inicializada com todos os eventos controláveis habilitados. Em seguida os eventos controláveis ativos são avaliados e, através da operação *E bit-a-bit* (&), a ação de controle assume o valor dado por Σ_i. A operação *&* é posteriormente utilizada para efetuar a ação de desabilitação dos supervisores ($\Sigma_i \setminus \Phi_{SL}$). Por fim, a análise de escolha determina se é necessária um complemento à ação de controle, tendo como resultado uma ação determinística a ser utilizada na interface.

No trabalho de Carvalho (2007) é proposta a utilização da função `rand()` para

Figura 5.23: Trecho do código em ANSI C da função principal para obtenção da ação de controle determinística

```
//Update Control_Action variable
Control_Action  = 0b111111111;
Control_Action &= SupervisoryControl__ProductSystemActiveEvents();
Control_Action &= SupervisoryControl__SupervisorsDisablingAction();
Control_Action &= SupervisoryControl__ChoiceDispatcher();
```

Fonte: produção do próprio autor

obtenção da aleatoriedade na escolha dos eventos a serem complementarmente desabilitados. Já nesta implementação utiliza-se o registrador de temporizador do microcontrolador já em uso pela aplicação. No momento em que uma escolha é requerida, a função realiza a tomada de decisão baseada no valor atual do registrador do temporizador. Por exemplo, para uma escolha entre dois eventos controláveis, já mapeados como situação de escolha, no momento em que estes estiverem ativos, utiliza-se o estado atual do *bit 0* do registrador para determinar o evento a ser desabilitado. O interessante dessa abordagem é que não se faz necessário adicionar rotinas (algoritmos) ou bibliotecas para se obter valores aleatórios. O uso de temporizadores é comum a diversas implementações, especialmente neste estudo onde se utiliza o conceito de *fatias de tempo* (TANENBAUM, 2009) para execução das funções. O registrador do periférico do microcontrolador está constantemente alterando seu valor e é controlado por *hardware* o que garante aleatoriedade para a implementação.

A chamada das funções de Leitura e Escrita da interface a partir da função principal (Figura 5.16) também deve ser feita de forma a não depender do número de eventos envolvidos no problema para que o módulo *Supervisory Control* seja independente. Para isso a função principal utiliza de ponteiro para as funções de interface. Não se espera nenhum retorno e não existem parâmetros para tais funções. Como se requer uma programação específica nessa camada a adição do evento nos *buffers* de eventos ocorridos fica a cargo de cada função de leitura e escrita da interface. Como apresentado na Figura 5.25a declara-se um ponteiro constante `Interface_Reader[]` que aponta para as funções de leitura correspondente a cada evento não controlável existente na aplicação. A chamada dessas funções a partir da função principal, mostrada na Figura 5.25b, portanto independe do número de eventos (funções) sendo o mesmo código para qualquer problema a ser resolvido utilizando o método proposto.

Um exemplo de função de interface é apresentado na Figura 5.25. Nessa, implementa-

Figura 5.24: Código ANSI C para execução das Funções de Leitura da *Interface*

```
//! A constant pointer to a
   function with no return value
   or parameters.
typedef void (* const
   INTERFACE_FUNCTION_TYPE)(void);

//Declare the list of Reader
   Functions for the interface.
   One for each Uncontrollable
   Event.
INTERFACE_FUNCTION_TYPE
   Interface_Reader[] = {
 Interface__ReaderEvent_e10,
 Interface__ReaderEvent_e12,
 Interface__ReaderEvent_e14,
 Interface__ReaderEvent_e2,
 Interface__ReaderEvent_e4,
 Interface__ReaderEvent_e6,
 Interface__ReaderEvent_e8
};
```

(a) Ponteiro para as funções de Leitura na interface

```
//Call Reader Interface
for(ev_idx = 0; ev_idx <
   UNCONTROLLABLE_EVENTS; ev_idx
   ++)
{
 Interface_Reader[ev_idx]();
}
```

(b) Chamada das funções no *SupervisoryControl*

Fonte: produção do próprio autor

Figura 5.25: Código ANSI C da Interface de Leitura para o evento *e2*

```
void Interface__ReaderEvent_e2(void)
{
 if((Interface_Events_Enable.Event_e2_T_ON_Enable == TRUE) &&
    (Sensor__GetCelsius(SENSOR_4) >= Compartment_Actual_SetPoint.T_On))
 {
  //Only add the event when temperature cross the threshold
  Interface_Events_Enable.Event_e2_T_ON_Enable = FALSE;
  INTERFACE_ADD_UNCONTROLLABLE_EVENT(e2);
 }
}
```

Fonte: produção do próprio autor

se a função de leitura para identificação de quando a temperatura *T_Int* torna-se maior que o valor de limiar dado por *T_On* (evento *e2*). Note que na identificação do evento *e2*, este é adicionado ao *buffer* de eventos não controláveis.

A interface também contém as funções de escrita para cada evento controlável e a mesma abordagem de ponteiro para função é utilizada. As funções da interface são responsáveis por interpretar e acusar a ocorrência de eventos. A implementação utiliza de *buffers* separados para eventos não controláveis e controláveis, dessa forma pode-se priorizar a transição de estados

por eventos não controláveis como discutido no Capítulo 4. No entanto, caso desejado, pode-se utilizar o mesmo *buffer* para os dois tipos de eventos pois a implementação das funções de iterface utilizam de macros permitindo tal flexibilidade.

5.3 Testes de validação do controle

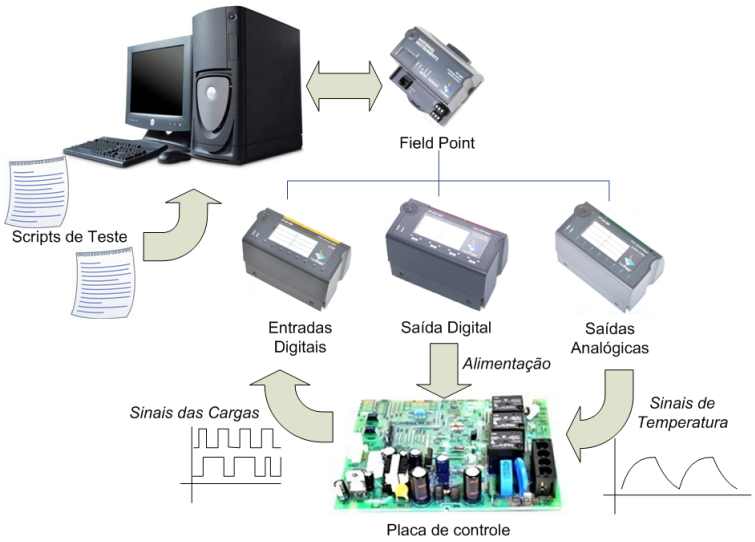
O código da implementação em linguagem ANSI C pode ser compilado para uma variedade de microcontroladores. Tem-se como objetivo validar o algoritmo de controle obtido pela execução das etapas do método e implementação desenvolvidos neste trabalho em plataformas utilizadas na indústria. Para isso, neste estudo de caso tendo como referência a Whirlpool Eletrodomésticos, o código foi compilado para ser embarcado em um microcontrolador de 8-bits da família STM8 (STMICROELECTRONICS, 2012). O *software* foi programado em uma placa eletrônica de controle voltada a aplicações em refrigeração desenvolvida pela empresa.

A empresa possui um processo para validação da eletrônica embarcada nos produtos. Parte deste processo consiste na validação do *software* de controle. Para auxiliar nesta etapa utiliza-se um sistema automatizado da National Instruments conhecido como *FieldPoint* utilizado em conjunto com o Labview (NATIONAL INSTRUMENTS, 2012a). Este sistema automatizado de validação foi utilizado para avaliar o comportamento do controle obtido para este estudo de caso.

O sistema de validação baseia-se na execução de *scripts* gerados especificamente para cada teste desejado. Um programa desenvolvido em Labview interpreta o *script*, atuando nas saídas e monitorando as entradas do *FieldPoint*. O mesmo programa armazena os dados para posterior análise das respostas do sistema sob avaliação. Não é necessário a adição de nenhum código extra, além do que se está validando, no programa gravado na placa eletrônica que está em teste. A Figura 5.26 apresenta um diagrama dos equipamentos utilizados para os testes de validação de *software*.

Foram criados dois *scripts* de teste para validação do comportamento do refrigerador autônomo. O primeiro consiste em alterar a temperatura interna do compartimento, medida pelo *Sensor A*, forçando as transições nos supervisores que determinam situações de escolha, tendo como objetivo avaliar a ocorrência de diferentes escolhas tomadas pelo controle, validando assim a ação complementar de desabilitação do determinador. O segundo teste modifica também a temperatura ambiente (*Sensor B*) para avaliar a ciclagem das cargas de acordo com os diferentes limiares de temperatura.

Figura 5.26: Diagrama da bancada para testes de validação de *software*



Fonte: produção do próprio autor

Os testes realizados são considerados em malha aberta pelo fato de que os valores de temperatura são fornecidos ao sistema, ou seja, a temperatura do compartimento não é determinada pelo funcionamento das cargas (compressor, ventilador e *damper*). O sistema de validação aplica na placa sob teste o valor AD correspondente à temperatura. Apesar disso, os testes são válidos uma vez que se avalia a resposta do controle às entradas fornecidas, portanto é possível simular condições com maior rapidez e com mesmo grau de confiança determinar se o controle supervisório atua de acordo com as especificações de controle modeladas.

5.3.1 Teste de atuação do Determinador

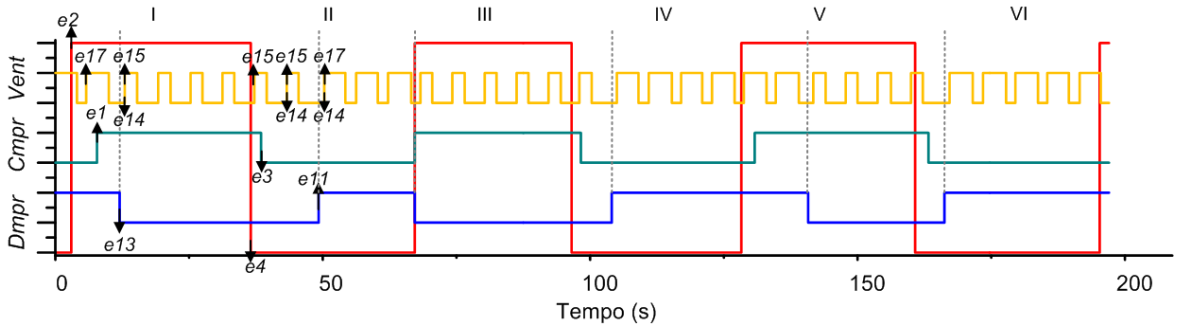
Para avaliação de diferentes escolhas realizadas pelo controlador e desta forma validar a ação de desabilitação complementar com característica aleatória dada pelo determinador, no primeiro teste modifica-se a temperatura do compartimento a fim de se habilitar as situações de escolha entre os eventos do ventilador variável (*e15* e *e17*) e *damper* eletrônico (*e11* e *e13)*.

Observando os modelos das especificações E_1 (Figura 5.10) e E_3 (Figura 5.12) utilizadas como supervisores reduzidos tem-se que a cada transição ocorrida em E_1, seja ela pelo evento *e2* ou *e4*, uma situação de escolha é ativada no supervisor E_3. Por exemplo, quando o estado atual de E_3 é o estado *0*, após a temperatura interna *T_Int* tornar-se menor que a temperatura *T_Off* (evento *e2*), ocorre uma situação de escolha entre permanecer com a rotação baixa no ventilador (*e17*) ou efetuar a abertura do *damper* (evento *e13*). Quando o controle

determinar a abertura do *damper* o estado de E_3 passa a ser *1*, no qual a velocidade do ventilador é maior (*e15*). Uma situação similar é definida na ocorrência do evento *e4*, enquanto E_3 está no estado *1*, desta vez, uma escolha entre os eventos *e15*, para manutenção da velocidade alta no ventilador, e evento *e17*, para fechamento do *damper* eletrônico.

Para melhor visualização do sinal de acionamento PWM do ventilador variável, a frequência de chaveamento foi reduzida para execução deste teste. Com um período de 6 segundos e razão cíclica $D_1 = 33\%$ e $D_2 = 67\%$, o ventilador é acionado por 2 segundos para funcionar a baixa rotação e por 4 segundos para atingir maior velocidade. O *script* desenvolvido para este teste altera a temperatura do compartimento para valores acima de *T_On* e abaixo de *T_Off* a cada 33 segundos, tempo que evita qualquer alinhamento com o ciclo do sinal PWM (não múltiplos). O teste teve um tempo de duração de 1 hora com uma taxa de aquisição dos sinais de 0,5 segundos. A Figura 5.27 apresenta os sinais coletados para 6 diferentes ciclos, os quais são indicados por números romanos no topo da figura.

Figura 5.27: Diferentes escolhas realizadas pelo controle



Fonte: produção do próprio autor

Os estágios **I**, **III** e **V** são de ciclos onde a temperatura do compartimento é maior que *T_On*. Nesses ciclos o controle deve ligar o compressor (*e1*), fechar o *damper* eletrônico (*e13*) e acionar o ventilador com baixa velocidade (*e15*). Observa-se que no primeiro ciclo há um pulso do PWM com longo ciclo de trabalho ($D = D_2$), portanto uma ocorrência do evento *e17*. Note que o evento *e1* está presente no ciclo **I**, e portanto o controle detectou a ocorrência do evento *e4*. No término do ciclo PWM (*e14*), o controle atua no *damper* para fechá-lo determinando dessa forma a ocorrência do evento *e13*. Logo em seguida observa-se um evento *e15* definindo que o ciclo de PWM do ventilador passe a ser determinado por $D = D_1$, respeitando a condição de menor rotação. Diferentemente no ciclo **III**, não há ocorrência do evento *e17*, logo após a mudança de temperatura (*e4*) o controle fecha o *damper* (*e13*) e a rotação do ventilador é alterada para menor rotação (*e15*). Já no ciclo **V** o ventilador é mantido em maior rotação por dois ciclos

PWM. Observam-se duas ocorrências do evento *e17* antes do fechamento do *damper* (*e13*).

Nas etapas onde a temperatura do compartimento está abaixo de *T_Off*, mostrados nos estágios **II**, **IV** e **VI** da Figura 5.27 a condição de temperatura está satisfeita, o controle deve atuar nas cargas a fim de desligar o compressor (*e3*), abrir o *damper* (*e11*) e ventilar o produto na velocidade mais alta (*e17*). Embora, assim como nos ciclos explicados anteriormente, ocorrem escolhas para manutenção da velocidade do ventilador antes de uma mudança no estado do *damper*. Por exemplo no ciclo **II** o controle determina a escolha em manter a baixa velocidade do ventilador (*e15*) por dois pulsos de PWM. Após a escolha do evento *e11* o controle executa o fechamento do *damper* e a velocidade no ventilador passa a ser maior (*e17*). Já no ciclo **IV** observa-se a escolha do evento *e15* por um ciclo do sinal PWM. Na sequência o *damper* eletrônico é aberto (*e11*) e o ventilador passa a ser acionado com maior rotação (*e17*). No último estágio (ciclo **VI**) não observam-se escolhas do evento *e15* após o evento *e2*.

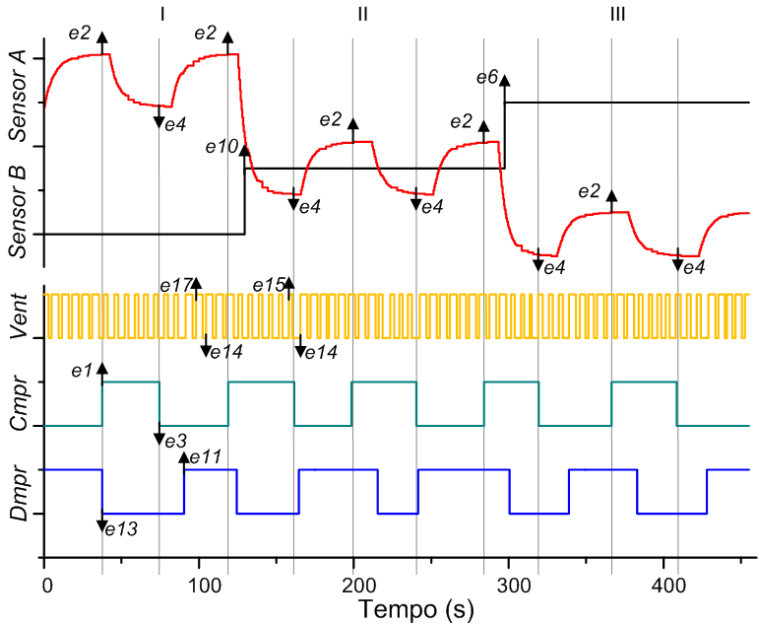
Devido à situação de escolha tem-se que por algumas vezes o controle mantém a velocidade de acionamento do ventilador antes de atuar no *damper*. Os diferentes estágios ainda apresentam número de escolhas diferentes mostrando que há aleatoriedade na determinação do evento a ser desabilitado, a exemplo de duas escolhas de *e15* no ciclo **II** e apenas uma no ciclo **IV**. Ainda é possível concluir neste teste que mesmo havendo escolhas que retardam a alteração no estado do *damper*, em algum momento este deixa de ser desabilitado, demonstrando a desejada inexistência de tendência por um subsistema ou ainda em um pior caso seu completo desuso. Em todos os casos, após a mudança no estado do *damper* a velocidade do ventilador é alterada de acordo com a especificação de controle E_3.

5.3.2 Teste dos ciclos para diferentes valores de temperatura ambiente

Para validação do comportamento das cargas com a mudança da temperatura ambiente desenvolveu-se um segundo *script* de teste. A temperatura ambiente, medida pelo *Sensor B*, na Figura 5.28 é inicialmente mantida abaixo de *T_ColdInf*. Durante o estágio **I** uma temperatura fria é detectada pelo controle e a ciclagem das cargas é determinada pelo nível *Máximo* (com maiores temperaturas estabelecidas para *T_On* e *T_Off*). A temperatura interna, medida pelo *Sensor A*, é modificada pelo sistema de validação a cada 3 segundos seguindo uma curva de característica exponencial para simular as curvas de mudança de temperatura como as que ocorrem na prática.

Durante o segundo ciclo de temperatura, inicia-se o estágio **II** aumentando a temperatura ambiente para um nível *Intermediário* superior à *T_ColdSup*. A especificação de controle

Figura 5.28: Diferentes ciclos do compressor para mudanças na temperatura ambiente



Fonte: produção do próprio autor

E_2 (utilizada como supervisor reduzido SL_2) atualiza os limiares de ciclagem e o compressor é acionado de acordo com menores limites para *T_On* e *T_Off*.

Na ocorrência do evento *e6* quando a temperatura ambiente é elevada acima de *T_HotSup* no estágio **III** o supervisor SL_2 novamente habilita a alteração de nível, desta vez para o nível *Mínimo*. O evento *e9* seleciona os menores valores de temperatura *T_OffMin* e *T_OnMin*.

Durante a execução deste teste os eventos de compressor *e1* e *e3* são observados para diferentes valores de temperatura do compartimento, assim como os eventos do *damper* eletrônico (*e11* e *e13*) e ventilador variável (*e15* e *e17*) acompanham as mudanças de temperatura. Desta maneira valida-se o comportamento desejado para o refrigerador autônomo que ajusta os limites *T_On* e *T_Off* para manter a temperatura interna mesmo com variações na temperatura exterior.

5.4 Conclusões do Capítulo

Uma implementação utilizando linguagem ANSI C foi aplicada a um estudo de caso para aplicação da arquitetura introduzida no Capítulo 4. A estrutura de implementação foi desenvolvida visando uma maior praticidade na reutilização e portabilidade do código gerado para o controle supervisório. O intuito é que a partir desta mesma implementação outras aplicações, com diferentes modelos, possam ser facilmente obtidas.

Esse objetivo é conquistado na medida em que a implementação é dividida em diferentes arquivos (módulos) concentrando as alterações necessárias entre diferentes aplicações em arquivos específicos, não necessitando a geração de todo o código a cada mudança.

A estrutura e implementação propostos foram validados em um sistema automático de validação de *software* usado na Whirlpool. As aquisições dos sinais permitem verificar a ocorrência das situações de escolha existentes na modelagem e a ação de controle adotada pelo controlador. Fica evidente o correto acionamento das cargas conforme as especificações de controle, embora neste estudo o sistema tenha funcionado em malha aberta. Este é um dos motivos para que a implementação seja aplicada ao controle de um refrigerador comercial, com uma atuação em malha fechada, na qual os estados de cargas determinam a temperatura dos compartimentos, o que é abordado no capítulo a seguir.

Capítulo 6

Aplicação do Método em Refrigerador Comercial

No capítulo anterior o método de implementação do Capítulo 4 foi utilizada para a obtenção do controle de um refrigerador em um estudo de caso. Neste capítulo a mesma abordagem de implementação é utilizada para implementação do controle de temperatura de um refrigerador comercial para posterior comparação com o algoritmo de controle originalmente utilizado pela empresa.

Por motivos de simplificação e confidencialidade apenas o funcionamento básico do algoritmo foi modelado. As funções não essenciais foram removidas do algoritmo original e isso possibilitou a comparação do comportamento do controle do refrigerador comercial e também do consumo de memória entre as abordagem original, já implementada pela empresa, e a abordagem via controle supervisório.

Inicialmente apresenta-se uma descrição do algoritmo de controle de temperatura do refrigerador e a modelagem seguindo o método do Capítulo 4. São apresentados resultados dos testes de funcionamento realizados com o código embarcado na placa de controle utilizada pelo produto. É feita a verificação do comportamento de acordo com as especificações de controle. Por fim, as abordagens de implementação são comparadas em termos de consumo de memória do microcontrolador.

6.1 Descrição do Refrigerador Comercial

O produto escolhido para modelagem da rotina de controle de temperatura trata-se de um refrigerador de dois compartimentos, com cargas e características de operação próximas às utilizadas no estudo de caso do Capítulo 5. A constituição física deste produto porém requer certos cuidados com relação ao sistema de refrigeração. Conhecido comercialmente por *Inverse*, possui o compartimento do *freezer* (compartimento com ar mais frio) abaixo do compartimento do refrigerador, conforme apresentado na Figura 6.1.

Figura 6.1: Refrigerador comercial

Fonte: (WHIRLPOOL LATIN AMERICA, 2012)

O controle de temperatura dos compartimentos baseia-se no acionamento liga/desliga de um compressor, ventilador e *damper* eletrônico, neste caso todas cargas de acionamento em Corrente Alternada (CA). A refrigeração do compartimento do *freezer* é feita pelo compressor, ao passo que o controle de temperatura do compartimento do refrigerador é dado pela ação conjunta do *damper* e ventilador.

Como naturalmente o ar mais frio permanece no compartimento inferior, faz-se necessário a ventilação forçada deste ar para o compartimento superior (refrigerador). A passagem deste ar refrigerado é permita pela abertura do *damper* eletrônico. O acionamento do ventilador não depende somente do estado de funcionamento do *damper*, seu acionamento sofre atrasos ou prolongamentos, de acordo com o acionamento do compressor, como apresentado na descrição da modelagem do algoritmo de controle.

As cargas relacionadas a um compartimento são acionadas em valores pré estabelecidos T_{Liga} e $T_{Desliga}$ que definem uma histerese em torno de um *setpoint*, sendo este o valor médio de temperatura desejado para o compartimento. O sensoriamento da temperatura é obtido através do uso de termistores do tipo *Negative Temperature Coefficient* (NTC).

6.2 Modelagem do Controle

A implementação do controle de temperatura seguindo o método do Capítulo 4 inicia pela modelagem da planta. A definição dos eventos para caracterização dos modelos é detalhada

conforme esses são introduzidos.

O compressor deste produto é modelado da mesma forma como a utilizada no estudo de caso do Capítulo 5. É de interesse do controle determinar o estado de compressor ligado e desligado, sendo que a transição entre tais estados é definida pela ocorrência dos eventos controláveis *comp_liga* e *comp_desl* conforme modelo da Figura 6.3a. De forma similar o autômato da Figura 6.3b modela o comportamento do ventilador, sendo *vent_liga* e *vent_desl* os eventos controláveis de acionamento e desligamento do ventilador.

O autômato apresentado na Figura 6.3c modela o comportamento do *damper* eletrônico. As transições de estado para o *damper* também ocorrem por dois eventos controláveis definidos como *damper_abre* e *damper_fecha* modelando a ocorrência de acionamento para a abertura e fechamento do *damper*, respectivamente.

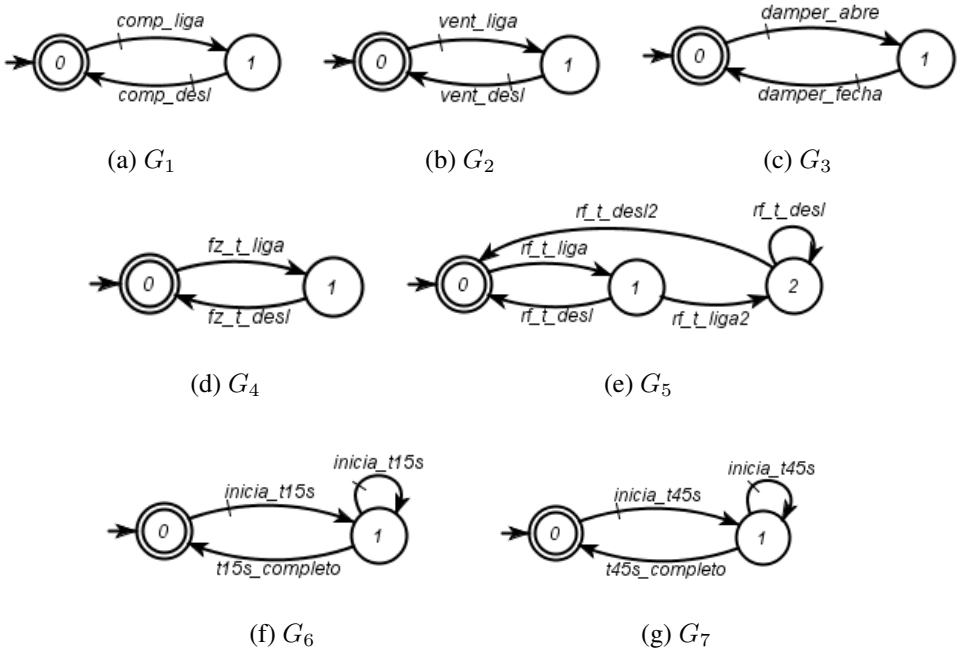
Para o usuário do produto são disponibilizadas diferentes níveis de temperatura, a exemplo *mínimo*, *médio* e *máximo*, embora as temperaturas de operação equivalentes a cada nível são previamente definidas no produto. Dessa forma, para o controle de temperatura, são relevantes apenas os valores de cruzamento com as temperaturas T_{Liga} e $T_{Desliga}$. A seleção do usuário define quais são os atuais valores para T_{Liga} e $T_{Desliga}$. Portanto, para o sensor que monitora a temperatura do compartimento do *freezer* são definidos os eventos não controláveis *fz_t_liga* acusando o cruzamento que a temperatura atual passa a ser maior que T_{Liga} e o evento *fz_t_desl* cuja ocorrência determina que a temperatura atual do compartimento passou a ser inferior à $T_{Desliga}$. O autômato que modela o sensor do *freezer* é apresentado na Figura 6.3d.

O controle de temperatura para o compartimento do refrigerador inclui uma proteção contra aquecimento excessivo. Portanto, o autômato representado na Figura 6.3e que modela o sensor do refrigerador contém um segundo par de transições por eventos não controláveis, *rf_t_liga2* e *rf_t_desl2*, que ocorrem nos cruzamentos da temperatura atual do compartimento pelos limiares determinados por T_{Liga2} e T_{Desl2}. Os valores estabelecidos para este compartimento são tais que $T_{Liga} <= T_{Liga2}$ e $T_{Desl} >= T_{Desl2}$.

Outros dois modelos de planta são utilizados para representar o comportamento de *temporizadores* necessários para o controle do ventilador. As figuras 6.3f e 6.3g apresentam os autômatos que modelam tais temporizadores de 15 e 45 segundos, respectivamente. Um evento controlável *inicia_t15s* (*inicia_t45s*) dispara a contagem de *15* (45) segundos. O término da contagem do *temporizador* é indicado pela ocorrência do evento não controlável *t15s_completo* (*t45s_completo*). Caso o contador já esteja realizando uma contagem de tempo, é possível que

este seja reiniciado antes de ter completado a anterior.

Figura 6.2: Modelos para os subsistemas da planta



(a) G_1 (b) G_2 (c) G_3

(d) G_4 (e) G_5

(f) G_6 (g) G_7

Fonte: produção do próprio autor

A partir dos modelos dos subsistemas da planta, e tendo como base o funcionamento do algoritmo de controle atualmente utilizado no produto, são definidas cinco especificações de controle para obtenção do algoritmo básico de funcionamento.

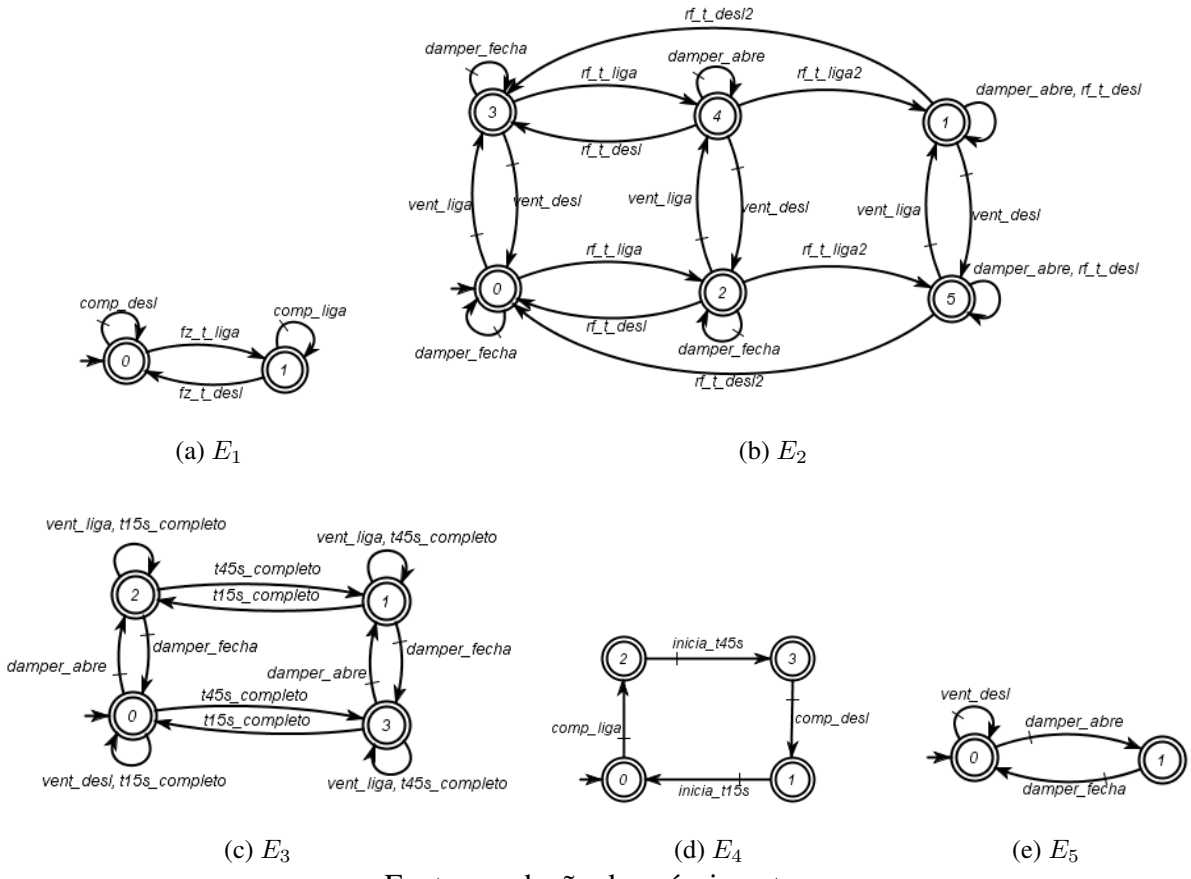
A especificação de controle representada pelo autômato da Figura 6.4a implica a proibição do acionamento do compressor enquanto a temperatura do compartimento do *freezer* não sofra um aquecimento para uma temperatura superior ao limiar determinado por T_{Liga}. A mesma especificação proíbe o desligamento do compressor até que a temperatura do *freezer* resfrie abaixo de T_{Desl}.

Na Figura 6.4b apresenta-se o autômato que modela a especificação de controle do *damper* eletrônico para o controle de temperatura do compartimento do refrigerador. Enquanto a temperatura no compartimento mantiver-se entre os limites principais T_{Liga} e T_{Desl} proíbe-se a abertura do *damper* sem que antes se tenha acionado o ventilador. Por outro lado, caso haja um aquecimento acima de T_{Liga2} a proteção do compartimento é ativada permitindo-se que o *damper* eletrônico seja aberto e que o ventilador seja ligado. Na ocorrência da ativação da proteção, é preciso que o compartimento atinja uma temperatura de recuperação abaixo de T_{Desl2} para que então se permita o fechamento do *damper* eletrônico.

O controle do acionamento do ventilador é obtido com o auxílio dos temporizadores e está relacionado com o estado do compressor. As especificações de controle representadas pelos autômatos das figuras 6.4c e 6.4d são utilizadas para determinar seu comportamento. Após o compressor ser ligado, inicia-se a contagem de 45 segundos para que haja um atraso no acionamento do ventilador. Portanto, proíbe-se que o ventilador ligue antes do *temporizador* de 45 segundos tiver finalizado a contagem. Na ocorrência do desligamento do compressor o ventilador é mantido ligado pelo tempo de 15 segundos, então permite-se que este seja desligado.

Na especificação de controle da Figura 6.4e observa-se que com o *damper* aberto proíbe-se o evento controlável *vent_desl* isto porque é necessário que o ventilador esteja ligado forçando a circulação do ar para o compartimento do refrigerador.

Figura 6.3: Modelos das especificações de controle



(a) E_1 (b) E_2

(c) E_3 (d) E_4 (e) E_5

Fonte: produção do próprio autor

Segundo a abordagem abordagem modular local (QUEIROZ, 2000) obtiveram-se os cinco supervisores para as especificações de controle modeladas. Na Tabela 6.1 apresenta-se os subsistemas da planta usados como planta local, número de estados e transições para os supervisores locais. Fazendo-se o teste da modularidade verifica-se que os supervisores são

Tabela 6.1: Supervisores para controle do refrigerador comercial

S_i	GL_i	**Estados**	**Transições**
S_1	$G_1 \| \| G_4$	4	6
S_2	$G_2 \| \| G_3 \| \| G_5$	12	38
S_3	$G_2 \| \| G_3 \| \| G_6 \| \| G_7$	32	144
S_4	$G_1 \| \| G_6 \| \| G_7$	16	32
S_5	$G_2 \| \| G_3$	4	7

Fonte: produção do próprio autor

não conflitantes. Seguindo o método de implementação apresentado no Capítulo 4 avalia-se que as linguagens alvo são controláveis. Portanto, não é necessário submeter os supervisores a algoritmos de redução uma vez que as próprias especificações de controle da Figura 6.3 podem ser utilizadas como supervisores locais reduzidos. Analisando-se o supervisor da Figura 6.4e pode-se interpretar que é necessário um bloco de solução de escolha entre os eventos *vent_desl* e *damper_abre*. No entanto, o controle dessas cargas encontra-se distribuído em diferentes supervisores locais e a ação conjunto de tais supervisores não apresenta situação de escolha. De forma que, para o controle do refrigerador comercial, não é necessário implementar uma política de escolha para o determinador.

6.3 Integração do Controle Supervisório no código do Produto

Utilizando-se o *plugin* desenvolvido neste trabalho (Apêndice A) para o IDES3 (RUDIE, 2006), foram gerados os arquivos de implementação de supervisores (*Supervisors.h*), sistema produto (*ProductSystem.h*), módulo principal constituído pelos arquivos *SupervisoryControl.c* e o *header SupervisoryControl.h*, assim como os arquivos para codificação da *Interface*.

Esses arquivos foram adicionados ao código embarcado no microcontrolador da placa de controle do produto. O módulo original do produto, que implementa o algoritmo de controle de temperatura, foi removido e substituído pelo *SupervisoryControl*. Os demais módulos de *software* foram mantidos e os módulos das cargas, compressor, *damper* e ventilador, além do módulo do sensor passaram a ser utilizados como rotinas de baixo nível pela *Interface*.

Este código foi compilado e o *software* programado no microcontrolador da placa de controle utilizada no refrigerador. Nenhum outro *software* ou programa externo, rodando em um computador por exemplo, foi utilizado para o controle no produto.

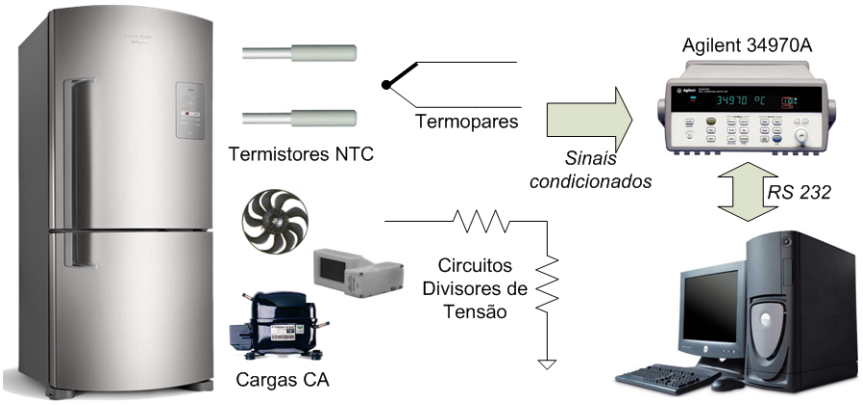
6.3.1 Testes do Controle Supervisório

Para análise da atuação do controle supervisório foram monitoradas as cargas e temperaturas dos compartimentos durante o funcionamento do produto. O *setup* utilizado durante este teste é representado pelo diagrama da Figura 6.4.

A instrumentação do produto através do uso de termopares posicionados junto aos termistores NTC e de circuitos divisores de tensão para as cargas CA permite que os sinais condicionados sejam lidos e armazenados pelo equipamento de aquisição de dados *Agilent 34970A* (AGILENT TECHNOLOGIES, 2012).

Este equipamento permite que diferentes tipos de sinais sejam monitorados ao mesmo tempo. A cada aquisição, que neste teste foi em intervalos de 6 segundos, os valores dos canais configurados foram registrados, enviados ao computador via comunicação RS232 e adicionados a um *log*, o sistema foi monitorando ao longo de 10 horas, tempo adequado para permitir a estabilização da temperatura nos compartimentos do refrigerador. Para realização deste teste, três canais do Agilent foram configurados como entradas para sinais de tensão em CA. Circuitos divisores de tensão foram adicionados em paralelo às saídas da placa eletrônica que realizam o acionamento do compressor, *damper* eletrônico e ventilador do produto. De forma que quando o controle acionava uma carga o sinal era capturado pelo equipamento de aquisição. Outros dois canais foram configurados para receberem sinais provenientes de termopares posicionados junto aos sensores de temperatura de cada compartimento. A conversão do valor do sinal do termopar para a unidade de graus Celsius foi realizada pelo próprio equipamento de aquisição.

Figura 6.4: *Setup* para monitoramento do produto



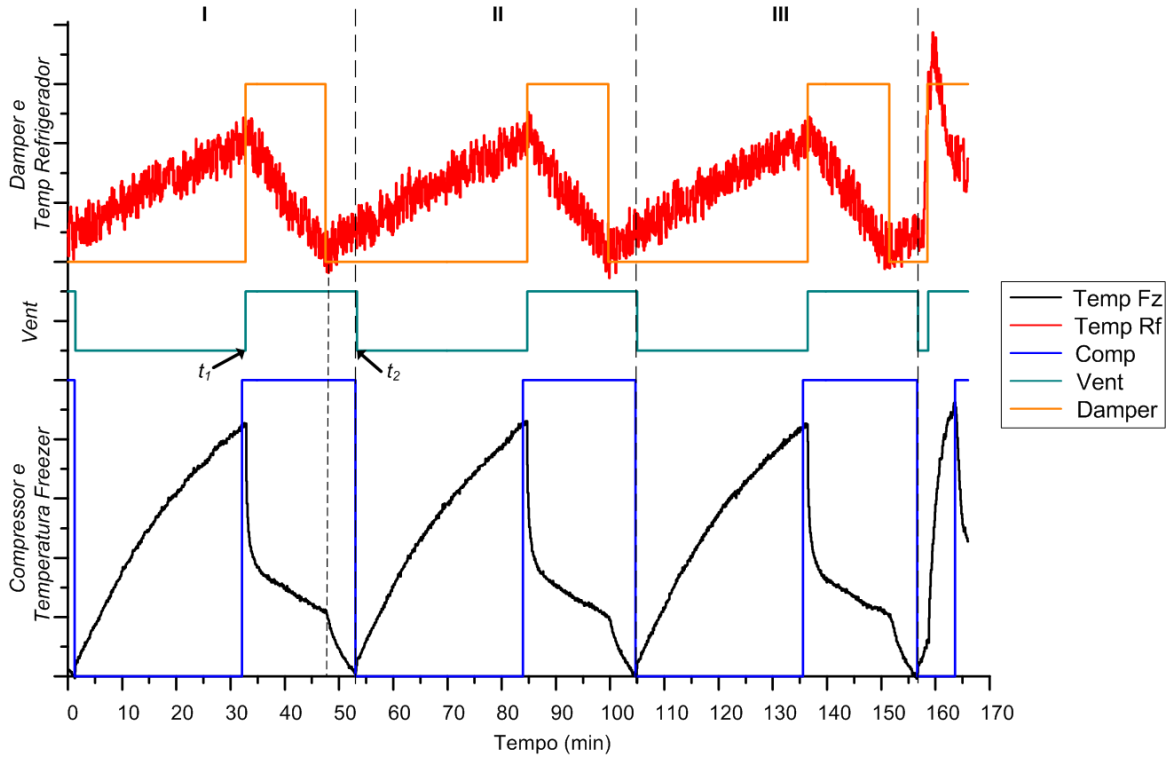
Fonte: produção do próprio autor

De posse do *log* capturado pelo Agilent fez-se uma análise dos dados para verificar se a atuação do controle supervisório estava de acordo com as especificações, e se estas refletem

o comportamento desejado para o funcionamento do produto. Os dados foram utilizados para construção de gráficos e a partir destes verificar o acionamento das cargas e o comportamento das temperaturas em cada um dos compartimentos. Durante os testes observou-se um ruído no sinal de aquisição da temperatura do compartimento do refrigerador, mas este ruído não prejudica a análise e entendimento do funcionamento do produto.

Na Figura 6.5 são visualizados três ciclos completos de refrigeração em ambos os compartimentos (do refrigerador e do *freezer*). Os dados são apresentados de maneira a facilitar a visualização entre acionamento das cargas e da temperatura do respectivo compartimento, sem no entanto destacar os valores absolutos, especialmente das temperaturas. Tomando como base o ciclo **I**, indicado no topo da Figura 6.5, note que o ciclo inicia com as cargas desligadas e consequentemente com aumento na temperatura dos compartimentos. Nesta operação normal (em regime) a refrigeração inicia com o acionamento do compressor, aproximadamente em $t = 32min$. O ventilador é acionado 45 segundos após o compressor devido ao tempo de atraso estabelecido pelo controle. Como a taxa de aquisição do teste é de 6 segundos, e são necessários apresentar vários minutos no gráfico para compreensão dos ciclos completos, o atraso de 45 segundos é indicado no instante t_1. Com o ventilador ligado o *damper* é aberto estabelecendo a passagem do ar frio também ao compartimento do refrigerador.

Figura 6.5: Ciclos de operação do refrigerador comercial

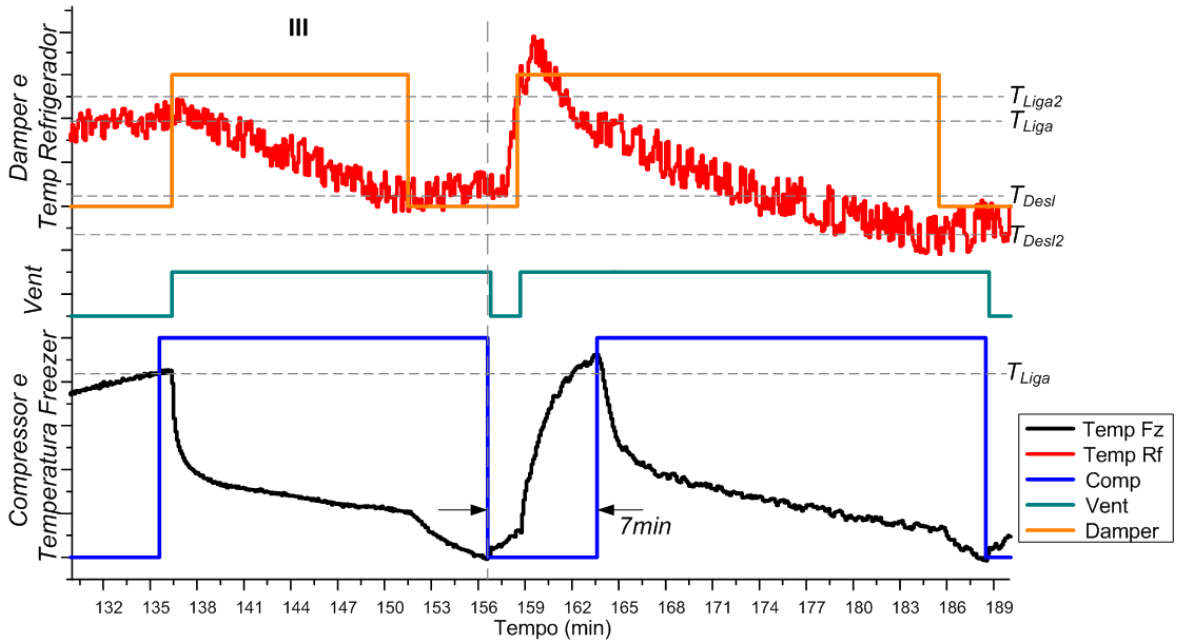


Fonte: produção do próprio autor

Na continuação do ciclo de refrigeração, a temperatura em ambos os compartimentos diminui, e em $t \approx 48min$ a temperatura T_{Desl} do compartimento do refrigerador é atingida e o *damper* é fechado pelo controle, pois a temperatura de limiar para o compartimento do refrigerador foi satisfeita. Há uma mudança na dinâmica do sistema, pois com o *damper* fechado apenas o *freezer* está sendo refrigerado. Compressor e ventilador são mantidos acionados, pois o compartimento do *freezer* ainda não atingiu a temperatura de desligamento. No entanto, rapidamente a temperatura no compartimento do *freezer* satisfaz a condição abaixo de T_{Desl}. Nesta condição o compressor é desligado ($t \approx 54min$) sendo seguido pelo desligamento do ventilador após o término do tempo de 15 segundos, indicado por t_2 no gráfico. Com as cargas desligadas inicia-se um novo ciclo de refrigeração, com o mesmo comportamento repetindo-se nos ciclos **II** e **III**.

Logo após o desligamento do compressor no terceiro ciclo da Figura 6.5 ($t \approx 155min$) a porta do compartimento superior (refrigerador) foi mantida aberta durante alguns minutos. Para melhor visualização do comportamento apresenta-se o período da referida abertura de porta na Figura 6.6, quando há um aumento brusco de temperatura em ambos os compartimentos.

Figura 6.6: Ciclo de operação do produto com abertura de porta do refrigerador



Fonte: produção do próprio autor

Com a abertura da porta do compartimento do refrigerador a temperatura deste aumenta acima do valor estabelecido pelo limiar T_{Liga2} que é maior que o valor T_{Liga} observado nos ciclos anteriores. Com o cruzamento da temperatura por T_{Liga2} a proteção contra sobreaquecimento do supervisor S_2 atua abrindo o *damper* e ligando o ventilador ($t \approx 160min$). Nesse caso,

não há tempo de atraso para acionamento do ventilador, uma vez que o acionamento não ocorre através da operação normal (em regime) logo após o compressor. Note que o compartimento do refrigerador começa a resfriar ainda com a temperatura do *freezer* aumentando.

Para evitar danos ao compressor, após seu desligamento, este precisa permanecer desligado durante um tempo antes de ser religado. Este tempo mínimo de desligamento, de 7 minutos nesta aplicação, está implementado na rotina de baixo nível do compressor analisada pela *Interface*. Observa-se que por volta do tempo $t = 162min$ na Figura 6.6 a temperatura no compartimento do *freezer* já é suficiente para requisitar o acionamento do compressor. Devido à característica de implementação baseada nas funções de escrita, a *Interface* solicita à rotina de baixo nível que este seja ligado, mas não há retorno do evento de que este está acionado até aproximadamente $t = 164min$, quando o tempo de proteção finaliza, e então a função de escrita vinculada ao acionamento do compressor retorna a ocorrência do evento *comp_liga*.

Como modelado na especificação de controle da Figura 6.4b é preciso que a temperatura do refrigerador atinja um valor menor quando a condição de proteção é estabelecida. Este comportamento é observado durante o teste com a abertura da porta, quando o controle permite o fechamento do *damper* a uma temperatura do refrigerador (T_{Desl2}) abaixo da encontrada nos ciclos anteriores (T_{Desl}) .

6.3.2 Comparações do Consumo de Memória do Microcontrolador

Além da comparação da implementação através da análise funcional no produto, foram estudados o consumo de memória do código obtido para o controle do refrigerador e do módulo original.

Quando um código é compilado para um determinado microcontrolador o compilador pode ser configurado para criar um arquivo que é um mapa de todos os símbolos, funções e variáveis utilizadas pelo programa. Este mesmo arquivo fornece o quanto de memória não volátil *Read-Only Memory* (ROM), a memória *Flash* nos microcontroladores atuais, e de memória volátil *Random Access Memory* (RAM) são consumidos por cada módulo do programa. O código em si que determina a lógica a ser executada é armazenado na memória *Flash*, assim como são as contantes. O consumo de memória RAM, por sua vez, é determinado pelas variáveis utilizadas pelos módulos.

Utilizando-se desse mapa, gerado por diferentes compilações, foram obtidos os recursos de memória necessários apresentados na Tabela 6.2. O consumo de memória *Flash* é

apresentado pela soma do consumo de texto (lógica) somado ao consumo de constantes para que sejam facilmente diferenciados, por exemplo na Compilação (a) da Tabela 6.2 o módulo Original consume 2291 *bytes* de texto (programa) e nenhum consumo em constantes.

Para efeito de comparação toma-se como referência o *software* original do produto, sem qualquer modificação. O consumo de memória inicial do módulo que implementa o controle de temperatura do refrigerador é obtido pela Compilação (a) na Tabela 6.2. Como mencionado anteriormente, parte do módulo original não foi modelado por questões de simplificação. Portanto, removendo-se do módulo original os trechos de código correspondentes às funções não modeladas tem-se como resultado a Compilação (b) que deve ser usada para comparação com o código obtido utilizando a abordagem de controle supervisório.

Tabela 6.2: Comparação de Consumo de Memória

Compilação	**Módulo Consumo**	**Original**	**SupervisoryControl**	**Interface**	**Events**	**Total**
(a)	Flash	2291+0	-	-	-	2291
	RAM	18	-	-	-	18
(b)	Flash	1397+0	-	-	-	1397
	RAM	13	-	-	-	13
(c)	Flash	-	568+231	728+48	101+0	1676
	RAM	-	13	1	23	37
(d)	Flash	-	369+675	728+48	101+0	1921
	RAM	-	6	1	23	30
(e)	Flash	-	379+133	850+48	101+0	1501
	RAM	-	6	1	23	30

Fonte: produção do próprio autor

Na Compilação (c) da Tabela 6.2 utilizaram-se as mesmas estruturas apresentadas no estudo de caso do Capítulo 5, de tal forma que o módulo principal *SupervisoryControl* contém as estruturas de Supervisores Reduzidos e elementos do Sistema Produto. A *Interface* foi codificada para se adequar aos eventos modelados neste problema. Como a aplicação original não é baseada em eventos, é necessário acrescentar os *buffers* de armazenamento de eventos e as funções gerenciamento, este consumo em memória está capturado inteiramente no módulo *Events*.

A comparação entre as compilações (b) e (c) mostra que há maior consumo de memó-

ria tanto *Flash* quanto RAM utilizando o método baseado na estrutura de controle supervisório. A implementação original do algoritmo é baseada no uso de máquinas de estados, no entanto, o controle é obtido sem um procedimento formal para tradução das especificações e requisitos no código implementado. De modo que esse aumento no consumo de memória não demonstra ser um empecilho para utilização do método proposto em aplicações comerciais. Há que se observar que o aumento da complexidade do problema leva ao aumento do número de modelos necessários, resultando no incremento do consumo de memória. Portanto outras alternativas podem ser estudas visando diminuir o impacto em consumo de memória, como uso de estruturas como as propostas por Lopes et al. (2011).

Sem alteração das estruturas de implementação usadas para codificação dos modelos, foi considerada a utilização dos modelos completos (não reduzidos) dos Supervisores. A compilação (d) da Tabela 6.2 apresenta o resultado obtido para essa alternativa. Com a utilização de supervisores completos, a informação dos eventos possíveis pela planta é implícita, não sendo necessário implementar os modelos e as funções de atualização e identificação do Sistema Produto. Os módulos *Interface* e *Events* permanecem intactos neste caso. Observa-se uma redução no consumo em código (texto) no módulo *SupervisoryControl*. Embora, devido ao tamanho dos modelos dos supervisores completos serem maiores, há um incremento no consumo em constantes, ou seja, a implementação das estruturas de supervisores completos é superior a de supervisores reduzidos e sistema produto combinados, não demonstrando ser uma boa alternativa.

Uma terceira abordagem é adotada na Compilação (e). Novamente os modelos de supervisores reduzidos são adotados. Os módulos desenvolvidos pela empresa e utilizados como rotinas de baixo nível para as cargas nesta implementação, também são baseados em máquinas de estado, de forma que é possível obter a informação do estado atual de planta. Observou-se que a implementação dos modelos do sistema produto encontravam-se redundantes no código. Uma adequação na interface para obter a informação dos estados atuais na planta foi realizada e os modelos de sistema produto puderam ser removidos.

Uma função foi adicionada à *Interface* para retornar os eventos controláveis ativos pela planta baseado na informação do estado atual obtida diretamente dos módulos de baixo nível. No módulo *SupervisoryControl* não foram adicionados os modelos de sistema produto, apenas adicionada, em relação ao estudo (d), a chamada da nova função da interface. Apesar do aumento em código de texto tanto no módulo principal como na interface, há grande redução no consumo de memória *Flash* devido à remoção das constantes relativas aos modelos do sistema

produto. Este último estudo apresenta uma aproximação ainda maior com o consumo de memória do módulo original, tendo um resultado muito satisfatório.

O consumo de memória depende das características dos modelos a serem implementados, tais como número de eventos, transições e estado, e também da plataforma alvo (microcontrolador) para a qual o código estará sendo compilado. De qualquer forma, é possível obter-se a estimativa do consumo de memória previamente. Como o programa é o mesmo independentemente da aplicação, o consumo de memória *Flash* relacionado à lógica permanece invariável, portanto a estimativa a ser feita depende das constantes codificadas, ou seja, da modelagem do problema e das variáveis utilizadas.

Na Tabela 6.3 apresentam-se as definições e características dos modelos que são as variáveis utilizadas nas equações para se obter a estimativa de consumo de memória. Nessa tabela as variáveis em letras maiúsculas têm relação ao número de bytes consumidos, já as variáveis em minúsculo são números obtidos diretamente a partir dos modelos.

Tabela 6.3: Variáveis das equações de estimativa de consumo de memória

Variável	**Descrição**
n	total de supervisores modelados no problema
m	total de elementos do sistema produto
c	total de eventos controláveis / funções de escrita
u	total de eventos não controláveis / funções de leitura
t	número de transições de um determinado modelo
q	número de estados de um determinado modelo
Q	*bytes* consumidos para definição de Estado
E	*bytes* para definição de Evento
T	*bytes* para definição do número de Transições
C	*bytes* para definição dos Vetores de Eventos Controláveis
V	*bytes* consumidos por um ponteiro para variável/constante
F	*bytes* consumidos por um ponteiro para função

Fonte: produção do próprio autor

Os valores base de consumo (variáveis em maiúsculo) estão relacionados principalmente ao tipo utilizado na declaração que por sua vez depende de alguma característica dos modelos. Por exemplo, para uma plataforma de *8 bits*, o valor de **Q** em *bytes* será **1** caso

nenhum modelo supere uma quantidade de **estados** maior que 255, pois é necessário apenas uma declaração do tipo *unsigned char* para a definição de estados. O valor de **Q** passa a ser **2** caso seja necessário a utilização da declaração de um inteiro *unsigned short int*. A Tabela 6.4 resume a base de consumo dos tipos de variável da linguagem C para uma plataforma de 8 *bits*.

Tabela 6.4: Consumo base dos tipos de variáveis em C para plataforma 8 *bits*

Variável	**Característica**	**Condição**	**Tipo**	**Consumo (*bytes*)**
Q	Estados	até 255	char	1
		entre 255 e 65535	short	2
		maior que 65536	long	4
E	Eventos	menor que 255	char	1
		menor que 65535	short	2
		maior que 65536	long	4
T	Transições	menor que 255	char	1
		menor que 65535	short	2
		maior que 65536	long	4
C	Ev. Controláveis	até 8	char	1
		entre 9 e 16	short	2
		acima de 16	long	4

Fonte: produção do próprio autor

Tendo definido as características de modelos e os valores do consumo do armazenamento em memória dos tipos de variável é possível equacionar a estimativa de consumo para a implementação utilizada neste trabalho.

O módulo *SupervisoryControl* inclui as constantes definidas tanto para os modelos de Supervisores quanto para os modelos do Sistema Produto. As constantes codificadas neste módulo definem os termos da Equação 6.1 e são referentes às: Listas do número de transições dos modelos, Matrizes das transições dos modelos, Vetores relacionados aos eventos controláveis dos estados, Ponteiro para as matrizes de transição e Ponteiro para os vetores.

$$Flash_{Ctes} = (n+m)\times T + \sum^{n+m}\{t\times(2E+Q)\} + \sum^{n+m}\{q\times C\} + (n+m)\times V + (n+m)\times V \quad (6.1)$$

$$Flash_{Ctes} = (n+m)\times(T+2V) + \sum^{n+m}\{t\times(2E+Q) + q\times C\} \quad (6.2)$$

O consumo de memória RAM no módulo *SupervisoryControl* está relacionado ao vetor que controla o estado ativo dos modelos de Supervisores e elementos do Sistema Produto, assim como a uma variável utilizada para obtenção dos eventos controláveis não desabilitados e ativos, utilizada para a chamada das funções de Escrita. A estimativa de consumo em memória RAM para este módulo é dada pela Equação 6.3.

$$RAM = (n + m) \times Q + C \tag{6.3}$$

Embora o consumo em memória *Flash* relativo à codificação da *Interface* seja variável devido à dependência com as rotinas e eventos modelados, o consumo em constantes também pode ser obtido por intermédio da Equação 6.4.

$$Flash_{Ctes} = (u + c) \times F \tag{6.4}$$

Para os estudos realizados neste capítulo, cujos resultados estão listados na Tabela 6.2, tem-se que Q = E = T = C = 1 *byte*, o ponteiro para variáveis consome V = 2 *bytes* equivalente ao endereçamento de memória da arquitetura do microcontrolador e por sua vez cada ponteiro para função da interface F = 3 *bytes*. Estes valores e dados dos modelos do problema aplicados às equações 6.2, 6.3 e 6.4 obtém-se os valores de consumo de memória apresentados na Tabela 6.2.

6.4 Conclusões do Capítulo

Neste capítulo, aplicou-se o método proposto no Capítulo 4 para o controle de temperatura em um refrigerador comercial (*Inverse*) da Whirlpool. O funcionamento básico do controle de temperatura foi modelado, e serve como estudo para aplicações do método. Foram realizados testes e monitoramento do produto que demonstram o correto funcionamento de acordo com as especificações de controle. A estrutura de controle supervisório foi implementada juntamente com outras rotinas executadas pelo produto e não foram detectados problemas na interação entre o controle e tais rotinas. O código obtido através do método proposto foi implementado e não foram necessárias adequações para o correto funcionamento do refrigerador.

Os consumos de memória entre o código já utilizado na empresa para o controle de temperatura do produto em comparação com este estudo, que implementa o controle baseado na TCS, foram bastante próximos, na ordem de centenas de *bytes*. Este resultado parece ser promissor para a utilização do método em sistemas microcontrolados aplicados comercialmente.

Embora, entende-se que o aumento da complexidade das rotinas acarretará em um

aumento no número de modelos e consequentemente no aumento de consumo de memória. Ainda assim, há vantagens no uso do método por se obter um controle através de procedimento formal, que facilita a geração automática de código. Utilizando a ferramenta desenvolvida neste trabalho (Apêndice A) obtém-se a implementação da estrutura de controle supervisório diretamente dos modelos utilizando o *software Integrated Discrete-Event Systems* (IDES) (RUDIE, 2006).

Por fim, a estrutura de implementação dos autômatos adotada neste trabalho (ver Capítulo 5) tem como principal alvo a legibilidade do modelo, ou seja, a conversão modelo-implementação é visualmente percebida no código. Diferentemente, trabalhos como o de Lopes et al. (2011) vêm sendo realizados voltados à implementação das estruturas dos modelos especialmente projetadas para redução no consumo em memória. Aliados ao uso de geradores automáticos, que fornecem ao usuário/programador a legibilidade necessária, pode-se aumentar a aplicabilidade do método do Capítulo 4 em aplicações comerciais de sistemas embarcados.

Capítulo 7

Conclusão

Neste trabalho apresenta-se um método de projeto para sistemas embarcados compreendido de etapas que vão desde o levantamento de requisitos até a aprovação do sistema de controle. A etapa de concepção do método proposto utiliza a abordagem modular local (QUEIROZ, 2000) para síntese de supervisores minimamente restritivos e não bloqueantes, que garantem o atendimento às especificações de controle.

Na fase de implementação do método propõe-se uma estrutura para o controle supervisório que visa lidar com diferentes problemas de implementação e solucionar lacunas existentes nas propostas encontradas na literatura.

Uma das contribuições deste trabalho consiste na análise sobre a aplicabilidade da propriedade relaxada de insensibilidade ao atraso (BASILE; CHIACCHIO, 2007) como meio para minimização da ocorrência do problema da sincronização inexata (FABIAN; HELLGREN, 1998). Para isso, a arquitetura de implementação prevê que seja executada a (re)leitura das entradas no início dos ciclos de execução para que sejam capturados os sinais de resposta que podem ter sido identificados com atraso. Na geração de um evento controlável a atualização de estados referida a esta ocorrência ocorre apenas após uma nova leitura das entradas. Tal condição permite que a ordem de ocorrência de eventos para as transições do controlador seja capturada na mesma ordem que ocorrem na planta física.

Além disso, apresenta-se uma arquitetura genérica para a estrutura de controle supervisório voltada à implementação em microcontroladores em um sistema embarcado. A arquitetura é composta por hierarquia prevendo que as diferentes camadas possam ser executadas em diferentes intervalos de tempo. Característica que permite restringir a necessidade de adequações aos sinais do sistema físico à camada de baixo nível.

A arquitetura utiliza a abordagem *online* de verificação do problema da escolha. A técnica permite que todos os caminhos previstos no supervisor possam ser executados pela implementação, característica não visualizada quando técnicas *offline* são consideradas. A estrutura

de implementação ainda mostra de forma clara a atuação do bloco determinador para solução do problema da escolha quando são implementados os modelos de supervisores reduzidos. Analisando-se o conjunto de eventos habilitados pela planta e não desabilitados pelo supervisor, tem-se a atuação do determinador nas mesmas condições de quando são utilizados modelos não reduzidos. Dessa forma, as mesmas atuações são obtidas, mas com um menor consumo em memória do microcontrolador, pois os modelos de supervisores reduzidos apresentam menor número de estados.

Para a interface, a arquitetura permite que chamadas das funções de escrita avaliem rotinas de baixo nível para determinar a ocorrência de eventos controláveis. Assim, eventos controláveis são gerados somente a partir de informações da planta física, aumentando a consistência entre teoria e implementação e solucionando o problema da causalidade. Essa característica torna-se especialmente útil quando as rotinas de baixo nível contém informações da planta abstraídas dos modelos utilizados na síntese dos supervisores. Diferentemente dos trabalhos encontrados na literatura nos quais a não desabilitação de um evento controlável invariavelmente dispara a geração do evento e consequente atuação na planta.

Uma ferramenta de geração automática de código em linguagem ANSI C foi desenvolvida para aplicação da estrutura de controle proposta no método de projeto. A ferramenta é integrada ao *software* IDES3 (RUDIE, 2006) na forma de um *plugin*. O aplicativo IDES3, utilizado para modelagem e síntese de supervisores, permite a exportação e importação de modelos no formato Grail e TCT, e portanto tem-se uma grande flexibilidade para a etapa de modelagem.

A arquitetura de implementação foi utilizada para obtenção do controle em um estudo de caso e em um refrigerador comercial da Whirlpool Eletrodomésticos. A modularidade da estrutura de implementação permitiu que o mesmo código principal fosse utilizado para as duas aplicações. As estruturas dos modelos foram obtidas a partir da ferramenta de geração automática, bastando adequar as funções de interface em cada problema.

Já na etapa de validação, foram utilizadas as ferramentas disponíveis na Whirlpool para validação dos sistemas de controle nos produtos. Os testes realizados demonstraram a correta atuação do controle de acordo com as especificações. Além disso, foi possível distinguir as diferentes escolhas tomadas a partir das ações do controlador, evidenciando a atuação randômica do determinador. O consumo em memória do microcontrolador também foi avaliado na aplicação de controle no refrigerador comercial. O consumo de memória ocupado pela implementação baseada no método proposto foi muito próximo ao consumo do algoritmo de controle original

utilizado no produto, com a vantagem de que o método utiliza de procedimento formal para obtenção dos supervisores.

Como perspectivas de trabalhos futuros para continuidade deste trabalho destacam-se:

- A substituição das estruturas de implementação dos modelos de supervisores e sistema produto por listas encadeadas e pela estrutura proposta por Lopes et al. (2011) para estudos no impacto em consumo de memória do microcontrolador.
- A aplicação do método para o projeto do controle do refrigerador comercial sem simplificações no algoritmo e em outros produtos comerciais e análise dos resultados obtidos.
- A aplicação da técnica de solução da sincronização inexata pela adição na implementação do caminho ausente $\sigma_c\sigma_u$ em sistemas que atendem à propriedade relaxada de insensibilidade ao atraso para validação da proposta.
- Proposição de um supervisor *sensível* ao entrelaçamento de eventos não controláveis para aplicação e avaliação da solução utilizando a técnica de interrupções para as entradas envolvidas.
- O desenvolvimento, testes e validação de algoritmos para verificação das propriedades de insensibilidade ao atraso e entrelaçamento de eventos não controláveis e para detecção do problema da escolha em supervisores modulares.
- Implementação destes algoritmos e execução de testes em diversos modelos para validação e posterior integração à ferramenta de geração de código. Tal união pode auxiliar na implementação de soluções para os problemas detectados.
- Um estudo teórico sobre as condições *necessárias* e *suficientes* para se garantir que os problemas da escolha e insensibilidade ao atraso não ocorrem na implementação dos supervisores modulares locais, especialmente no que se refere aos modelos utilizados para identificação do problema. Por exemplo, no caso do problema da escolha, para o qual é possível afirmar que a detecção do problema em supervisores locais não é suficiente para garantir que o mesmo ocorre na ação conjunta dos supervisores.

REFERÊNCIAS

AGILENT TECHNOLOGIES. **34970A Data Acquision / Data Logger Switch Unit**. 2012. Disponível em: <http://www.home.agilent.com>. Acesso em: 05/Agosto/2012.

ALMEIDA, S. R. de. **Implementação de controle supervisório em CLPs usando linguagem de alto nível**. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade do Estado de Santa Catarina - UDESC, Joinville, SC, 2012.

BALEMI, S. Communication delays in connections of input/output discrete event processes. In: 31ST IEEE CONFERENCE ON DECISION AND CONTROL, 1992. **Proceedings...** [S.l.]: IEEE, 1992. p. 3374 –3379 vol.4.

BARRETA, R.; TORRICO, C. R. Máquinas de mealy e moore para implementação de controle supervisório de sistemas a eventos discretos em microcontroladores. In: XVII CONGRESSO BRASILEIRO DE AUTOMáTICA - CBA, 2008, Juiz de Fora. **Anais...** Juiz de Fora: SBA, 2008.

BASILE, F.; CHIACCHIO, P. On the implementation of supervised control of discrete event systems. **Control Systems Technology, IEEE Transactions on**, v. 15, n. 4, p. 725 –739, july 2007. ISSN 1063-6536.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML : guia do usuário**. 2. ed. Rio de Janeiro: Campus, 2006. ISBN 8535217843 (broch.).

BRANDIN, B. The real-time supervisory control of an experimental manufacturing cell. **Robotics and Automation, IEEE Transactions on**, v. 12, n. 1, p. 1 –14, feb 1996. ISSN 1042-296X.

CANTARELLI, M.; ROUSSEL, J.-M. Reactive control system design using the supervisory control theory: Evaluation of possibilities and limits. In: 9TH INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS - WODES'08, 2008. **Proceedings...** [S.l.]: IEEE, 2008. p. 200 –205.

CARVALHO, J. R. **Contribuições a Implementação da Estrutura de Controle Modular Local**. Dissertação (Mestrado em Engenharia de Produção e Sistemas) — Programa de Pós-Graduação em Engenharia de Produção e Sistemas, Pontifícia Universidade Católica do Paraná - PUCPR, Curitiba, PR, 2007.

COSTA, G. de O. **Uma Plataforma Computacional de Suporte ao Ciclo de Desenvolvimento de Sistemas Automatizados de Manufatura**. Dissertação (Mestrado em Engenharia de Produção e Sistemas) — Programa de Pós-Graduação em Engenharia de Produção e Sistemas, Pontifícia Universidade Católica do Paraná - PUCPR, Curitiba, PR, 2005.

CRUZ, D. L. L. da. **Metodologia para Implementação de Controle Supervisório Modular Local em Controladores Lógicos Programáveis**. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade do Estado de Santa Catarina - UDESC, Joinville, SC, 2011.

CUNHA, A. E. C. da. **Contribuições ao controle hierárquico de sistemas a eventos discretos**. Tese (Doutorado em Engenharia Elétrica) — Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2003.

CURY, J. Teoria de controle supervisório de sistema a eventos discretos. In: V SIMPóSIO BRASILEIRO DE AUTOMAçãO INTELIGENTE - SBAI, 2001, Canela. **Anais...** Canela,RS, 2001.

DIETRICH, P.; MALIK, R.; WONHAM, W.; BRANDIN, B. Implementation considerations in supervisory control. In: SYNTHESIS AND CONTROL OF DISCRETE EVENT SYSTEMS - SCODES'01, 2001, Paris. **Proceedings...** Paris,France: Kluwer Academic Publishers, 2001. p. 185–201.

EMBRACO. **Conhecendo o Compressor**. 2012. Disponível em: <http://www.embraco.com/Default.aspx?tabid=71>. Acesso em: 07/Maio/2012.

FABIAN, M.; HELLGREN, A. Plc-based implementation of supervisory control for discrete event systems. In: 37TH IEEE CONFERENCE ON DECISION AND CONTROL, 1998. **Proceedings...** [S.l.]: IEEE, 1998. v. 3, p. 3305 –3310 vol.3.

FENG, L.; WONHAM, W. TCT: A computation tool for supervisory control synthesis. In: 8TH INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS - WODES'06, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 388 –389.

FERIGOLLO, C.; TORRICO, C.; LEAL, A. Análise de desempenho das abordagens monolítica e modular local da teoria de controle supervisório implementada em microcontroladores. In: X SIMPóSIO BRASILEIRO DE AUTOMáTICA - SBAI, 2011, São João del-Rei. **Anais...** São João del-Rei: SBA, 2011.

FLORDAL, H.; FABIAN, M.; ÅKESSON, K.; SPENSIERI, D. Automatic model generation and plc-code implementation for interlocking policies in industrial robot cells. **Control Engineering Practice**, Elsevier, v. 15(11), p. 1416–1426, 2007.

GANSSLE, J. G. **The Art of Designing Embedded System**. 2nd. ed. Oxford, UK: Elsevier, 2008.

GRIGOROV, L. **IDES Software**. Kingston, Ontario, 2012. Disponível em: <https://qshare.queensu.ca/Users01/rudie/www/software.html>. Acesso em: 10/Setembro/2012.

HASDEMIR, T.; KURTULAN, S.; GöREN, L. An implementation methodology for supervisory control theory. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 36 (3), p. 373–385, 2008.

HECHT, M. Products liability issues for embedded software in consumer applications. In: IEEE SYMPOSIUM ON PRODUCT SAFETY ENGINEERING, 2005, Los Angeles. **Proceedings...** The Aerosp. Corp., Los Angeles, CA, USA: IEEE, 2005. p. 42 – 48.

HUANG, J.; KUMAR, R. Directed control of discrete event systems for safety and nonblocking. **Automation Science and Engineering, IEEE Transactions on**, v. 5, n. 4, p. 620 –629, oct. 2008. ISSN 1545-5955.

HUBBARD, P. **Hierarchical Supervisory Control Systems**. Tese (Doutorado) — Department of Electrical and Comuputer Engineering - McGill University, Montréal, Canada, 2000.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. **IEC 60730**: Automatic electrical controls for household and similar use - annex h: Requirements for electronic controls. [S.l.], 2010.

KERNIGHAN, B.; RITCHIE, D. **The C Programming Language**. 2nd. ed. Englewood Cliffs,New Jersey,USA: Prentice Hall Inc., 1988.

LEAL, A. B.; CRUZ, D. L. L. da; HOUNSELL, M. da S. PLC-Based Implementation of Local Modular Supervisory Control for Manufacturing Systems. In: ____. **Manufacturing System**. InTech, 2012. cap. 8. ISBN 978-953-51-0530-5. Disponível em: <http://www.intechopen.com/books/manufacturing-system/plc-based-implementation-of-local-modular-supervisory-control-for-manufacturing-systems>.

LOPES, Y.; HARBS, E.; LEAL, A.; JR., R. R. Proposta de implementação de controle supervisório em microcontroladores. In: X SIMPóSIO BRASILEIRO DE AUTOMáTICA - SBAI, 2011, São João del-Rei. **Anais...** São João del-Rei: SBA, 2011.

MALIK, P. Generating controllers from discrete-event models. In: SUMMER SCHOOL IN MODELLING AND VERIFICATION OF PARALLEL PROCESSES (MOVEP), 2002. **Proceedings...** [S.l.], 2002. p. 1–6.

NATIONAL INSTRUMENTS. **O que é o LabVIEW?** 2012. Disponível em: <http://www.ni.com/labview/whatis/pt/>. Acesso em: 13/Agosto/2012.

NATIONAL INSTRUMENTS. **Shortening the Embedded Design Cycle with Model-Based Design**. National Instruments, October 2012. Disponível em: <http://www.ni.com/white-paper/4074/en>.

PENA, P. N. **Verificação de Conflito na Supervisão de Sistemas Concorrentes Usando Abstrações**. Tese (Doutorado em Engenharia Elétrica) — Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2007.

PINOTTI, A. J.; LEAL, A. B.; OLIVEIRA, D. S. Uma proposta de metodologia para implementação da estrutura de controle supervisório em controladores lógicos programáveis. In: XVIII CONGRESSO BRASILEIRO DE AUTOMáTICA - CBA, 2008, Bonito. **Anais...** Bonito,MS: SBA, 2010. p. 2830–2837.

QUEIROZ, M. H. de. **Controle Supervisório Modular de Sistemas de Grande Porte**. Dissertação (Mestrado em Engenharia Elétrica) — Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2000.

QUEIROZ, M. H. de. **Controle Supervisório Modular e Multitarefa de Sistemas Compostos.** Tese (Doutorado em Engenharia Elétrica) — Departamento de Automação e Sistemas - DAS, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2004.

QUEIROZ, M. H. de; CURY, J. E. R. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: SIXTH INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS - WODES, 2002. **Proceedings...** [S.l.]: IEEE, 2002. p. 377 – 382.

RAMADGE, P.; WONHAM, W. The control of discrete event systems. **Proceedings of the IEEE**, v. 77, n. 1, p. 81 –98, jan 1989. ISSN 0018-9219.

RUDIE, K. The integrated discrete-event systems tool. In: 8TH INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS - WODES'06, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 394 –395.

SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. **Design Test of Computers, IEEE**, v. 18, n. 6, p. 23 –33, nov/dec 2001. ISSN 0740-7475.

SILVA, Y. G. **Controle Supervisório Modular Local de Sistemas de Veículos Auto-Guiados**. Dissertação (Mestrado em Engenharia de Automação e Sistemas) — Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Universidade Federal de Santa Catarina - UFSC, Florianópolis, SC, 2010.

STMICROELECTRONICS. **STM8 - 8-bit MCUs**. 2012. Disponível em: <http://www.st.com/internet/mcu/class/1738.jsp>. Acesso em: 13/Agosto/2012.

SU, R.; WONHAM, W. M. Supervisor reduction for discrete-event systems. **Discrete Event Dynamic Systems**, v. 14, n. 1, p. 31 – 53, January 2004.

TANENBAUM, A. S. **Sistemas operacionais modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009. ISBN 978-85-7605-237-1.

TEIXEIRA, C. A. **Aplicação da Teoria de Controle Supervisório no Projeto de Controladores para Eletrodomésticos**. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade do Estado de Santa Catarina - UDESC, Joinville, SC, 2008.

THEUNISSEN, R. J. M.; SCHIFFELERS, R. R. H.; BEEK, D. A. v.; ROODA, J. E. Supervisory control synthesis for a patient support system. In: 10TH EUROPEAN CONTROL CONFERENCE (ECC'09), 2009, Budapest. **Proceedings...** Budapest, Hungary, 2009. p. 1–6.

VAZ, A.; WONHAM, W. M. On supervisor reduction in discrete-event systems. **Int. J. Control**, v. 44(2), p. 475–491, 1986.

VIEIRA, A. D. **Implementação de estrutura de controle de sistema a eventos discretos em controlador lógico programável utilizando a teoria ControleSupervisório Modular Local**. Pontifícia Universiadede Católica (PUC-PR) e Universidade Federal de Santa Catarina (UFSC), 2003.

VIEIRA, A. D.; CURY, J. E. R.; QUEIROZ, M. H. de. A model for plc implementation of supervisory control of discrete event systems. In: IEEE CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION - ETFA '06., 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 225 –232.

WHIRLPOOL LATIN AMERICA. **Sobre a Whirlpool**. 2012. Disponível em: <http://www.whirlpool.com.br/SobreaWhirlpool>. Acesso em: 21/Maio/2012.

WOOD, M. M. **Application, Implementation and Integration of Discrete-Event Systems Control Theory**. Tese (Mestrado (Engenharia)) — Department of Electrical and Comuputer Engineering - Queens University, Kingston, Ontario, Canada, 2005.

XU, S.; KUMAR, R. Asynchronous implementation of synchronous discrete event control. In: 9TH INTERNATIONAL WORKSHOP ON DISCRETE EVENT SYSTEMS - WODES'08, 2008. **Proceedings...** [S.l.]: IEEE, 2008. p. 181 –186.

Apêndice A

Plugin de Geração Automática de Código

O *software* IDES a partir de sua versão 3 beta 1 permite que operações com autômatos e outras customizações sejam integradas ao aplicativo através do uso de uma interface padrão (API) para *plugins*.

Durante a realização deste trabalho foi desenvolvido um *plugin* para a geração automática de código baseado na estrutura de implementação utilizada no Capítulo 5 e Capítulo 6. A seguir são apresentadas a forma de instalação de *plugins* no IDES3 e instruções básicas de utilização do *plugin* de geração de código.

A.1 Instalação de *plugins* no IDES3

Os *plugins* são disponibilizados no formato de *Java Archive* (.JAR) com a compilação das classes desenvolvidos para o *plugin* com base na API disponibilizada pelo IDES. Para maiores informações para utilização da API verificar a documentação disponível juntamente com o pacote (GRIGOROV, 2012).

Para instalação de *plugins* basta copiar o arquivo **.jar** para a pasta ***plugins*** presente na pasta de instalação do IDES. Por exemplo $C:\backslash Programas \backslash IDES3 \backslash plugins$.

A.2 *Plugin* para Geração de Código

O *plugin* **Ides2MatrixPlugin.jar** adiciona duas operações para conversão dos modelos no código em linguagem ANSI C. A operação *Ides2MatrixSup* gera os arquivos do SupervisoryControl, Supervisor e Interface, utilizando como parâmetros os supervisores reduzidos. A Figura A.1 mostra os passos para utilização do *plugin*.

1. Acesse *Operations* > *DES Operations*

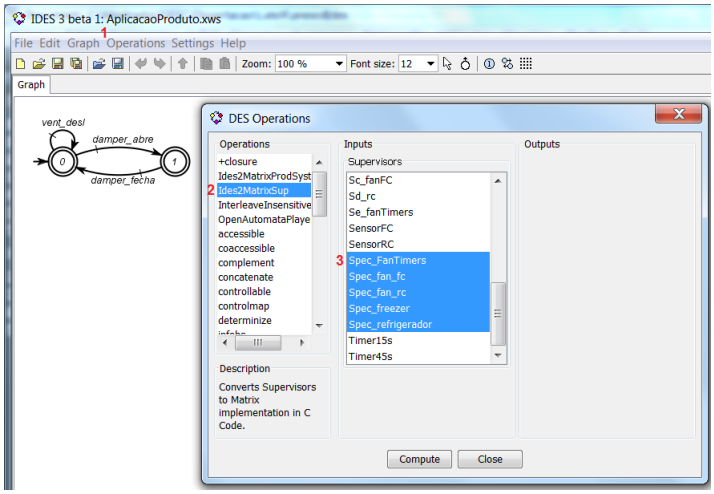
2. Selecione a operação *Ides2MatrixSup*

3. Adicione todos os modelos de supervisores a serem convertidos

4. Clique em *Compute*

5. A mensagem "Created Files Path: <caminho>"aparecerá indicado o local onde os arquivos foram gerados (mesma pasta onde estão os arquivos dos modelos).

Figura A.1: Utilização do *plugin* para geração de código



Fonte: produção do próprio autor

A segunda etapa utiliza a operação *Ides2MatrixProdSyst* para geração do arquivo ProductSystem.h utilizando os modelos das plantas como parâmetros. O procedimento de utilização segue os mesmos passos que a operação anterior.

Os arquivos gerados devem ser adicionados ao projeto no ambiente de desenvolvimento do microcontrolador no qual se implementará o controle supervisório. Codifica-se o conteúdo das funções de escrita e leitura da interface e implementam-se as rotinas de baixo nível de acordo com os requisitos da aplicação.