MARCOS ROBERTO FAUSTINO

**NORMA IEC61131-3: ASPECTOS HISTÓRICOS,
TÉCNICOS E UM EXEMPLO DE APLICAÇÃO**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do Título de
Mestre em Engenharia.

São Paulo

2005

MARCOS ROBERTO FAUSTINO

**NORMA IEC61131-3: ASPECTOS HISTÓRICOS,
TÉCNICOS E UM EXEMPLO DE APLICAÇÃO**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do Título de
Mestre em Engenharia.

Área de Concentração:
Sistemas de Potência

Orientador:
Prof. Dr. Clovis Goldemberg

São Paulo
2005

FICHA CATALOGRÁFICA

Faustino, Marcos Roberto

Norma IEC 61131-3: aspectos históricos, técnicos e um exemplo de aplicação.

/ Marcos Roberto Faustino -- São Paulo, 2005.

123 p.

Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Energia e Automação Elétricas

1. Automação industrial 2. Controladores programáveis 3. Motores elétricos I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Energia e Automação Elétricas II.t

À minha esposa Patrícia, pela paciência e ajuda.
Aos meus pais Manuel e Joaquina, por terem oferecido
antecipadamente uma herança que nunca se dissipará:
Educação.

AGRADECIMENTOS

Ao Prof. Dr. Clovis Goldemberg pela orientação e ajuda oferecida desde os tempos do Trabalho de Conclusão de Curso da Graduação, por oferecer-me a oportunidade de trabalhar no Projeto de Modernização dos Navios-Varredores, pelas diversas lições de hardware e controle ensinadas durante o desenvolvimento deste projeto e pela tolerância com o meu horário de trabalho randômico.

Aos meus chefes na Photon Engenharia e Desenvolvimento Ltda., Eng. Calípio Luiz Rocha Neto e Eng. Nicolau Henrique Longo Neto, por não criarem nenhuma dificuldade durante o período que trabalhei simultaneamente nessa empresa e no Projeto de Modernização dos Navios-Varredores, por terem oferecido um ambiente onde pela primeira vez tomei contato e usei os conceitos da IEC61131-3, por permitirem que a versão original do tradutor de SFC para IL por mim desenvolvida na Photon fosse adaptada para uso no Projeto dos Navios-Varredores e por cederem inúmeros livros citados nessa dissertação.

Ao IPqM (Instituto de Pesquisas da Marinha), representado pelo Capitão de Fragata Vianna Tavares, por ter permitido apresentação do Projeto de Modernização dos Navios-Varredores no estudo de caso desenvolvido nesta dissertação. Aos engenheiros que trabalham no grupo de Sistemas Digitais do IPqM pelas críticas e sugestões que ajudaram a melhorar as ferramentas e metodologias aqui apresentadas.

À FUSP (Fundação de Apoio à Universidade de São Paulo), por ter viabilizado o acordo institucional com a Marinha do Brasil, visando o Projeto de Modernização dos Navios-Varredores.

Aos Professores Cícero Couto de Moraes e Fuad Kassab Junior pelos comentários no Exame de Qualificação do Mestrado, que contrubuíram para o aprimoramento deste trabalho. Ao Professor Walter Kaiser por me convencer que a norma IEC61131-3 seria um assunto apropriado para o desenvolvimento de uma dissertação de mestrado.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) por oferecer a todos os alunos da Escola Politécnica acesso ao banco de artigos técnicos do IEEE (*Institute of Electrical and Electronics Engineers*) sem o qual a lista de referências bibliográficas seria significativamente menor.

Aos meus pais Manuel e Joaquina por terem investido na minha educação; Às minhas irmãs Maria Elena, Maria Inez e Maria Raquel pelos conselhos e apoio que viabilizaram minha admissão na Escola Politécnica; À minha esposa Patrícia pelo incentivo, ajuda e compreensão durante os momentos que estive afastado trabalhando no Projeto de Modernização dos Navios-Varredores e escrevendo esta dissertação.

À todos que direta ou indiretamente, colaboraram na execução deste trabalho

RESUMO

Este trabalho traça um panorama dos PLCs e tecnologias associadas em momentos anteriores e posteriores à publicação da norma IEC61131-3 e discute aspectos relativos à sua adoção. A aplicação desta norma pode trazer ganhos de produtividade no projeto e implementação de sistemas de automação industrial. Esta dissertação apresenta o caso do “Projeto de modernização dos navios-varredores da Marinha do Brasil” no qual alguns conceitos desta norma foram utilizados com sucesso. Ferramentas e metodologias desenvolvidas para adequar o PLC existente a alguns requisitos da norma são descritas ao longo da dissertação. A operação do novo sistema de varredura pode ser verificada através dos resultados experimentais apresentados.

ABSTRACT

This work presents an overview of the PLC and associated technologies before and after the publication of the IEC61131-3 standard. Some aspects concerning the adoption of this standard are also discussed. The application of this standard can increase productivity of the design and implementation of industrial automation systems. Some concepts of this standard were applied, successfully, to the modernization of the Brazilian Navy mine-sweepers. Tools and methods that were required in order to adapt the existing PLC to the IEC standard are described. The operation of the newly developed system can be verified by experimental results.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1 INTRODUÇÃO	1
1.1 O conteúdo da dissertação	2
1.2 Observações quanto à forma	4
 2 HISTÓRICO DO PLC	 5
2.1 O início	5
2.2 <i>Hardware</i> e <i>firmware</i>	7
2.3 Periféricos de programação e documentação	11
2.4 Periféricos de interface com o usuário	13
2.5 Redes de comunicação de dados	14
2.6 Linguagens de programação	17
2.6.1 <i>Ladder</i>	17
2.6.2 GRAFCET	22
2.6.3 Outras linguagens de programação	24
2.7 A evolução do PLC vista por meio das referências bibliográficas	24
 3 A NORMA IEC61131	 29
3.1 Histórico da IEC61131-3	29
3.2 Características da IEC61131-3	31
3.2.1 O modelo de <i>software</i>	31
3.2.2 Linguagens de programação	36
3.3 IEC61131-3: Prós e contras	38
3.4 O mercado e a IEC61131-3	40
3.4.1 Informações quantitativas	40
3.4.2 Aspectos estratégicos	42
3.4.2.1 Ameaça de produtos e serviços substitutos	42
3.4.2.2 O poder de negociação dos clientes	45
3.5 A adoção da IEC61131-3	47

4 A MODERNIZAÇÃO DOS NAVIOS-VARREDORES	50
4.1 Introdução ao sistema de varredura	50
4.2 Visão geral do novo <i>hardware</i>	52
4.3 Visão geral do novo <i>software</i>	55
4.4 Comentários finais	59
5 CONTROLE	60
5.1 A tradução dos diagramas de controle	60
5.2 O processador de macros M4	62
5.3 A construção dos blocos de função	62
5.4 A estratégia de controle utilizada no Sistema Médio Tom (MT)	63
5.5 Principais blocos de função usados	64
5.5.1 Blocos de geração de referência	64
5.5.2 Blocos de controle	66
5.5.3 Blocos adicionais	66
6 LÓGICA DE DETECÇÃO DE ERROS	68
6.1 O motivo da lógica de detecção de erros	68
6.2 A implementação da detecção de erros	69
6.3 Macros de configuração da lógica da detecção de erros	73
6.4 A detecção de erros e a norma IEC61131-3	75
7 SEQÜENCIAMENTO	78
7.1 A tradução de SFC para IL	78
7.2 Visão geral do seqüenciamento	79
7.3 O funcionamento de um grafo SFC superior	83
7.4 O diagrama SFC do sistema acústico MT	84
8 DEPURAÇÃO E TESTE	89
8.1 Depuração do seqüenciamento	89
8.2 Depuração e monitoração do controle	90
8.3 Teste	91

9 RESULTADOS	93
9.1 O sistema MT em funcionamento	93
9.1.1 A preparação do sistema MT	94
9.1.2 A varredura do sistema MT	97
9.1.3 O desmonte do sistema MT	99
9.2 O sistema magnético da cauda em funcionamento	99
9.3 O gerador de sinal PRBS e o sistema magnético	101
 10 CONCLUSÃO	 104
 ANEXO A CÓDIGO FONTE DA MACRO ESCALONA	 106
 ANEXO B TRADUÇÃO DE SFC PARA IL: SINTAXE E EXEMPLOS	 108
B.1 Apresentação informal da sintaxe da linguagem de descrição textual de SFC ..	108
B.2 Exemplos de representação textual de SFCs	110
B.3 Código resultante da tradução dos exemplos	114
B.3.1 Código resultante da tradução do primeiro exemplo	114
B.3.2 Código resultante da tradução do segundo exemplo	115
B.3.3 Código resultante da tradução do terceiro exemplo	116
B.3.4 Código resultante da tradução do quarto exemplo	117
 LISTA DE REFERÊNCIAS	 118

LISTA DE FIGURAS

Fig. 3.1 - Exemplo ilustrativo do modelo de <i>software</i> da norma IEC 61131-3	34
Fig. 3.2 - Ciclo de vida de um produto de nova tecnologia	48
Fig. 4.1 - Funcionamento do navio-varredor	51
Fig. 4.2 - Trajetória percorrida por um navio-varredor	51
Fig. 4.3 - Diagrama de blocos do <i>hardware</i> do sistema de varredura acústica de médio tom (MT)	54
Fig. 4.4 - Diagrama de blocos do <i>software</i> do sistema de varredura acústica de médio tom (MT)	56
Fig. 4.5 - Diagrama SFC do sistema MT	58
Fig. 5.1 - Exemplo de diagrama de controle	61
Fig. 5.2 - Bloco de função hipotético em sua versão gráfica e textual	63
Fig. 5.3 - Diagrama de blocos de controle do sistema MT	64
Fig. 5.4 - Perfil de velocidade utilizado na varredura acústica de médio tom	65
Fig. 6.1 - Exemplos de tabelas de erros utilizadas nos navios-varredores	70
Fig. 6.2 - Lógica de detecção de erros	71
Fig. 7.1 - Hierarquia dos SFCs	81
Fig. 7.2 - Exemplo de SFC superior	84
Fig. 7.3 - Macro-estado de Espera de Comandos (M1)	85
Fig. 7.4 - Macro-estado de Preparação de Varredura (M2) (continua)	86
Fig. 7.5 - Macro-estado de Preparação de Varredura (M2) (conclusão)	87
Fig. 7.6 - Macro-estado de Diminuição de Referência (M3)	88
Fig. 7.7 - Macro-estado de Desmonte da Varredura (M4)	88
Fig. 9.1 - Comportamento da velocidade do martelo MT durante ciclo completo de operação	94
Fig. 9.2 - Variáveis referentes ao controle de corrente de campo durante a partida	95
Fig. 9.3 - Variáveis referentes ao controle de velocidade durante a partida	96
Fig. 9.4 - Variáveis referentes ao controle de corrente de armadura durante a partida	96
Fig. 9.5 - Variáveis referentes ao controle de velocidade durante a varredura	98
Fig. 9.6 - Variáveis referentes ao controle de corrente de armadura durante a varredura	98
Fig. 9.7 - Referência do <i>chopper</i> de armadura durante a varredura	98
Fig. 9.8 - Variáveis referentes ao controle de velocidade durante o desmonte	99
Fig. 9.9 - Diagrama de blocos de controle do sistema magnético	100

Fig. 9.10 - Referência e realimentação da corrente de cauda (sistema magnético)	101
Fig. 9.11 - Resposta do campo do gerador de varredura ao sinal pseudo-aleatório . . .	102
Fig. 9.12 - Diagrama de blocos do sistema para identificação do campo do gerador de varredura	102
Fig. B.1 - Primeiro exemplo de SFC	110
Fig. B.2 - Segundo exemplo de SFC	111
Fig. B.3 - Terceiro exemplo de SFC	112
Fig. B.4 - Quarto exemplo de SFC	113

LISTA DE TABELAS

Tabela 2.1 - A evolução do PLC	7
Tabela 2.2 - Linguagens utilizadas nos livros consultados	25
Tabela 2.3 - Alguns assuntos abordados nos livros consultados	27
Tabela 3.1 - Partes da norma IEC61131	30
Tabela 3.2 - Métodos de comunicação disponíveis aos elementos de configuração e aos POU's	35
Tabela 3.3 - Tipos de variáveis de interface permitidas nos POU's	36
Tabela 6.1 - Estado dos <i>bits</i> das tabelas para as diversas configurações úteis de detecção de erro	74

1 INTRODUÇÃO

*Não pensem que eu vim trazer paz à terra;
eu não vim trazer a paz, e sim a espada.
Mateus, Capítulo 10, Versículo 34*

Passaram-se 12 anos desde a publicação da IEC61131-3, mas muitos dos seus conceitos ainda são ignorados por um grande número de pessoas envolvidas com automação industrial. Tal constatação motivou o autor desta dissertação a desenvolvê-la no sentido de mostrar os ganhos de produtividade obtidos com o uso desta norma no projeto, implementação e manutenção de um sistema real. Esta dissertação também apresenta a evolução dos equipamentos destinados à automação industrial (especialmente os PLCs) e o seu mercado nos momentos anteriores e posteriores à publicação da norma.

O *hardware* destinado à automação vive um momento de convergência em que equipamentos com fins antes bastante específicos estão se tornando mais genéricos, por exemplo, PLCs, PCs, SDCDs têm áreas de aplicação cada vez mais sobrepostas. Em contraste, ainda existe uma discussão acirrada sobre qual o melhor caminho a trilhar em relação às soluções de *software* destinadas ao controle.

Durante anos, a linguagem *ladder* foi a única usada para programar PLCs. Embora *ladder* também seja uma linguagem normalizada pela IEC61131-3, esta abrange outras linguagens. A norma da IEC também traça uma metodologia voltada para decomposição dos programas em módulos, reuso de código e criação de estruturas de dados definidas pelo programador. Grandes polêmicas têm ocorrido (KRIGMAN, 1985); (CONTROL.COM, 2001); (CONTROL.COM, 2002) entre os defensores do *ladder* como linguagem única (POLLARD, 1994) e os que defendem a multiplicidade de linguagens trazida pela norma (LEWIS, 1995); (LEWIS, 1996); (DAVIDSON *et al.*, 1997). Em outro extremo, existem os que consideram antiquadas as linguagens padronizadas pela norma e criticam sua falta de coesão (CRATER, 2005A). Não é objetivo deste trabalho tomar partido nestas polêmicas, tarefa impossível, pois:

- Os problemas a serem resolvidos pela automação são muito diversos e as necessidades dos clientes desse mercado são muito amplas. Torna-se difícil obter métricas que quantifiquem méritos e deficiências de cada uma das linguagens;
- Não é, economicamente, viável, em projetos reais, solucionar problemas usando cada um dos enfoques para se estabelecer uma comparação entre eles. Mesmo que tal experimento fosse viável, estabelecer parâmetros justos de comparação seria difícil;
- Em algumas situações, as discussões podem ser polarizadas pelos interesses de empresas fornecedoras de equipamentos de automação.

O objetivo deste trabalho é muito mais humilde, restringindo-se a fazer a análise de um caso em que o uso dos conceitos da norma mostrou-se proveitoso.

Ao contrário de grande parte dos livros de automação nos quais exemplos fictícios são usados, esta dissertação se baseou em um projeto real: “*A modernização dos sistemas de controle de varredura de um conjunto de navios da Marinha do Brasil*”. Estes sistemas de controle de varredura, depois de quase trinta anos de uso, se mostravam bastante desgastados. Tendo sido implementados com amplificadores operacionais e relés, a obtenção de peças de reposição para a sua manutenção era cada vez mais difícil. Os navios que continham tais sistemas, denominados de navios-varredores, possuem características construtivas específicas que os tornam inadequados para outras tarefas que não sejam a eliminação de minas marítimas. Para que esses navios-varredores pudessem ser capazes de realizar sua função primordial, a solução encontrada foi a substituição do antigo sistema de controle por um sistema digital implementado com PLC.

Todas as tarefas envolveram a documentação do *software* e a programação do PLC foram implementadas por um único engenheiro, o que implicou na necessidade de metodologias de trabalho produtivas. Por ser um sistema de uso militar, o *software* desenvolvido deve ser bastante confiável, fácil de testar e de dar manutenção. Tais circunstâncias levaram à adoção dos conceitos da IEC61131-3 neste projeto.

A definição do PLC usado no projeto foi feita por meio de uma licitação e o equipamento escolhido não atendia a todos os requisitos da norma IEC61131-3, o que obrigou ao desenvolvimento de ferramentas e metodologias (descritas nesta dissertação) para suprir tais deficiências.

1.1 O conteúdo da dissertação

Esta dissertação não apresenta os conceitos básicos sobre operação, aplicação e/ou programação de PLCs, exigindo do leitor alguma familiaridade com esses conceitos. Na lista de referências, existem vários livros que fornecem essa visão introdutória (NATALE, 1989); (MICHEL, 1990); (LEWIS, 1995); (SILVEIRA, 1998); (MORAES *et al.*, 2001).

Os Capítulos 2 e 3 apresentam uma breve história dos PLCs e outras tecnologias associadas antes e depois da publicação da norma IEC61131-3. O Capítulo 3 também discute alguns aspectos relativos à aceitação da norma pelo mercado de automação.

Os Capítulos 4 a 9 apresentam a aplicação dos conceitos da norma ao projeto de modernização dos navios-varredores, buscando um equilíbrio (necessariamente difícil) entre as informações sobre a constituição do sistema de varredura e os conceitos da norma. Se nenhuma informação sobre os navios-varredores fosse dada, as idéias da norma seriam apresentadas apenas

num caso abstrato. Caso a dissertação tratasse apenas nos navios-varredores, o espaço para apresentar as idéias da norma tornar-se-ia restrito. Esta é a razão pela qual a dissertação apresenta apenas os resultados do “*subsistema acústico de médio tom*” apesar de os conceitos da norma terem sido aplicados igualmente a todos os três sub-sistemas de varredura existentes (apresentados no Capítulo 4).

O Capítulo 4 dá noções básicas sobre minas marítimas e do processo usado para sua varredura. Também são apresentados o *hardware* do novo sistema de controle de varredura e uma visão geral do *software* desenvolvido para o PLC.

O Capítulo 5 detalha o módulo de controle das variáveis de varredura. O projeto de controle produziu uma série de diagramas de blocos que seriam implementados diretamente caso o PLC usado dispusesse da linguagem FBD (*Function Block Description*). É apresentada a metodologia adotada para traduzir os diagramas de blocos para linguagem IL (*Instruction List*), a qual utilizou largamente o processador de macros M4. Finalmente, são apresentados os principais blocos usados no controle desenvolvidos durante o projeto.

O Capítulo 6 apresenta a lógica de detecção de erros. Trata-se de um módulo que faz o processamento unificado de erros e falhas do sistema. Esta lógica tem relação muito próxima com o seqüenciamento, de tal maneira que, quando um erro ocorre, o sistema é conduzido a um estado seguro. A separação entre as tarefas de seqüenciamento e detecção de erros permitiu uma implementação e manutenção mais fácil do módulo de seqüenciamento. Neste capítulo também é apresentado um conjunto de macros que permitem a configuração dinâmica da detecção de erros conforme o seqüenciamento evolui.

O Capítulo 7 mostra como foi implementado o seqüenciamento, ou seja, a habilitação e desabilitação ao longo do tempo de diversos componentes de *hardware* e tarefas feitas pelo *software*. O projeto deste módulo resultou em uma série de diagramas SFC (*Sequential Function Chart*). Como o PLC usado não dispunha desta linguagem, foi desenvolvido um tradutor para auxiliar na tradução destes diagramas para código IL (*Instruction List*). Apresentou-se também como o seqüenciamento do sistema foi distribuído hierarquicamente entre vários grafos SFC de modo a termos um tratamento de falha e erros sensato. Detalhes da seqüência de operações seguida por um dos subsistemas durante sua partida, varredura e parada finalizam este capítulo.

O Capítulo 8 apresenta as técnicas desenvolvidas para depuração e teste do sistema.

O Capítulo 9 apresenta resultados experimentais, por meio de registros adquiridos pelo próprio PLC.

O Capítulo 10 apresenta as conclusões.

1.2 Observações quanto à forma

Algumas expressões foram escritas em inglês, preservando a nomenclatura utilizada nas normas técnicas e em referências bibliográficas. Nestas circunstâncias, as expressões estão em itálico. Em algumas das figuras apresentadas também podem existir termos na língua inglesa.

Alguns termos podem ter sentido diferente conforme o contexto em que são usados. Por exemplo, “erro” pode significar valor de referência menos realimentação. Em outros contextos, “erro” significa algum evento que leva o sistema a uma condição indevida.

Todas as imagens e marcas referentes a produtos contidas ou citadas nesta dissertação pertencem aos seus respectivos proprietários, sendo usadas exclusivamente em caráter educacional.

2 HISTÓRICO DO PLC

*Vamos descer e confundir a língua deles,
para que um não entenda a língua do outro.
Gênesis, Capítulo 11, Versículo 7*

Neste capítulo, será exposto um histórico do desenvolvimento do PLC e das tecnologias a ele associadas. As empresas que fabricam estes equipamentos geralmente são bastante reservadas, liberando pouca informação sobre o seu *hardware* e *firmware* aos usuários. Os livros existentes sobre PLCs dedicam poucas páginas aos aspectos históricos, encontrando-se até versões conflitantes. Convém salientar que não se pretende reconstituir o passado de modo detalhado ou rigoroso, mas trazer subsídios para as discussões sobre a IEC61131-3 que serão abordadas nos capítulos posteriores.

A revisão bibliográfica efetuada levantou os seguintes aspectos dignos de nota:

- *hardware* e *firmware*;
- periféricos de programação e documentação;
- periféricos de interface com o usuário;
- redes de comunicação de dados;
- linguagens de programação.

Após o detalhamento dos tópicos da lista acima, serão analisadas as referências bibliográficas existentes. Será mostrado como a evolução dos PLCs faz com que assuntos centrais em livros mais antigos sequer sejam tratados nos mais recentes.

2.1 O início

Nos anos 60 e 70, os clientes que desejavam comprar um automóvel numa dada cor eram freqüentemente obrigados a esperar longos períodos, como se estes fossem produzidos em “safras”. Isto era consequência dos grandes lotes produzidos. O *setup* de uma linha de produção tinha um custo proibitivo, pois as fábricas daquela época não haviam sido projetadas para serem flexíveis devido às limitações da tecnologia de automação então usada. Antes do PLC, os intertravamentos para controle e segurança eram feitos usando relés e contadores. Esta técnica tinha várias desvantagens:

- Baixa flexibilidade. Mudar os intertravamentos implementados implicava em alterar a fiação e mecânica dos painéis de relés, levando a longas paradas de produção;

- Alto custo de operação. Os componentes usados eram pesados e volumosos e os seus painéis ocupavam grandes áreas das fábricas. Por serem equipamentos eletromecânicos, estavam sujeitos a desgaste;
- Alto custo de engenharia e manutenção. As lógicas implementadas com relés tinham que ser minimizadas para diminuir o custo de componentes e montagem. Lógicas muito minimizadas trazem dificuldades de entendimento e documentação ao pessoal de manutenção (SILVEIRA *et al.*, 1998) .

Segundo Kissel (1986), Simpson (1994) e Moraes *et al.* (2001), o primeiro PLC foi desenvolvido a partir da divisão Hydramatic da General Motors buscando uma empresa interessada em produzir um equipamento com as seguintes características:

- ser facilmente programado e ter sua seqüência de operação prontamente mudada, de preferência na própria planta;
- ter a manutenção e reparo facilitados usando uma montagem de módulos “plugáveis”;
- ter capacidade de operar na planta de modo mais confiável que um painel de relés;
- ser fisicamente menor que um painel de relés para minimizar o custo de ocupação do chão de fábrica;
- produzir dados para um sistema central de coleta de informações;
- ser competitivo quanto a custo em relação a painéis de relé e de estado sólido então em uso.

Em 1969, a Bedford Associates, uma empresa de consultoria em engenharia, foi a selecionada para o fornecimento. Mais tarde, esta empresa mudou seu nome para Modicon (*Modular Digital Controller*). Kissel (1986) cita que o produto apresentado se chamava 084 por ter sido desenvolvido depois de 83 modificações de projeto. Outras empresas que participaram daquela licitação (entre as quais podemos destacar a Allen-Bradley), apesar de não terem sido escolhidas, logo estavam oferecendo seus produtos a outros clientes. No início dos anos 70, outros fornecedores entraram neste mercado como a Texas Instruments e a Square D.

No Brasil, os PLCs vieram a se proliferar na indústria apenas nos anos 80 (SILVEIRA *et al.*, 1998), primeiramente, pela absorção de tecnologias utilizadas na matriz das multinacionais. Simpson (1994) indicava que havia mais de 50 fabricantes de PLC, o que mostra o sucesso deste produto. O mesmo autor apresenta uma tabela com eventos-chave na evolução do PLC que foram reproduzidos na tabela a seguir:

Tabela 2.1 - A evolução do PLC

Ano	Evento
1968	Concepção do PLC
1969	Unidade central de processamento baseada em <i>hardware</i>
1972	Edição de código fonte
1974	PLC multiprocessado
1976	Entrada e saída remota
1977	PLC baseado em microprocessador com processador lógico
1978	Estrutura de entrada e saída universal
1979	Arquitetura de processador <i>bit-slice</i>
1980	Entrada e saída remota de alto desempenho/inteligente
1981	Transmissão de dados em média velocidade
1983	Coprocessador de linguagem Basic
1986	PLCs flexíveis multilinguagem

2.2 *Hardware e firmware*

A maior parte dos livros não entra em detalhes sobre a arquitetura e implementação de *hardware* ou *firmware* do PLC. Isto até faz sentido, pois o objetivo destes trabalhos não é ensinar a projetar ou dar manutenção, mas sim ensinar a programar e operar o equipamento. A exceção é Michel (1990), que se detém um pouco mais explicando a constituição interna dos controladores programáveis (memória, processador, entrada/saída etc).

A memória é um dos elementos de *hardware* essenciais de um PLC e deve ser não volátil. Dados armazenados de modo confiável são uma condição necessária para que o equipamento seja robusto. A memória no PLC pode desempenhar as seguintes funções:

- armazenamento do *firmware*;
- armazenamento do código do programa desenvolvido pelo usuário;
- armazenamento dos dados do programa desenvolvido pelo usuário;

Memórias onde cada *bit* era armazenado usando um pequeno núcleo de ferrite foram as usadas nos primeiros PLCs. A direção de magnetização indicava o estado 0 ou 1 do *bit* associado a este núcleo. Michel (1990) classifica estas memórias como caras, volumosas e de baixo desempenho. Os núcleos de ferrite tinham dimensão muito maior que os agrupamentos de

transistores hoje usados para armazenar dados em *chips*. Para magnetizá-los, eram necessárias correntes relativamente altas implicando no uso de volumosos amplificadores de potência. Apesar destas desvantagens, aquelas eram as memórias mais confiáveis na época. Morley (2005) comenta que, no projeto dos primeiros PLCs, foram escolhidas memórias com grandes núcleos de ferrite capazes de armazenar maior energia magnética. Tais memórias operavam com uma relação sinal-ruído alta, o que implicava em retenção mais confiável dos dados.

O desenvolvimento da microeletrônica tornou viável o uso de memórias semicondutoras.

A princípio, as memórias RAM (*Random Access Memory*) foram usadas, mas, por serem intrinsicamente voláteis, eram sempre acompanhadas por baterias que as mantinham permanentemente alimentadas. As baterias sempre foram um elo fraco na confiabilidade de dados críticos e um problema de manutenção.

Mais tarde, passaram a ser utilizadas memórias PROM (*Programmable Read Only Memory*), que não eram reprogramáveis, onde pequenos fusíveis eram queimados para armazenar os estados binários dos *bits*. O próximo passo, foram as memórias não voláteis EPROM (*Erasable Programmable Read Only Memory*) que só podiam ser apagadas pela exposição à luz ultravioleta. A necessidade da exposição à luz ultravioleta foi abolida com o surgimento de memórias que podiam ser apagadas eletricamente (*EEPROM* ou *Electrically Erasable Read Only Memory*).

Todas as variantes cujo nome termina com ROM não necessitavam de baterias como as RAM, e normalmente não eram reprogramáveis sem a sua retirada do PLC. Devido a estas características, elas eram úteis para armazenar o *firmware* e, eventualmente, o código do programa desenvolvido pelo usuário. Para o armazenamento de dados do programa, RAMs continuaram sendo usadas.

Kissel (1986) assinala que memórias não voláteis como as PROM podiam ser usadas como *backup* de programas de PLCs a serem guardados em áreas seguras como uma gaveta do escritório. O mesmo autor indica que diversos *chips* de memória contendo programas diferentes para um mesmo PLC eram uma maneira fácil de reprogramar uma máquina. Quando se desejasse iniciar a produção de um produto que necessitasse de um comportamento diferente da máquina, bastava trocar a sua memória não-volátil. De acordo com Michel (1990), menos de cinco PLCs eram equipados com EEPROM em 1981, sendo que dois anos mais tarde, o número havia aumentado para, aproximadamente, 20 (provavelmente no mercado francês) mostrando o sucesso deste tipo de memória.

O *firmware* é um programa desenvolvido pelo fabricante do produto com a função de executar as tarefas mais primárias do equipamento, tais como a interface básica com o *hardware*

do sistema. Tal programa tem a função de facilitar o uso do equipamento por parte do usuário. Dentre os que trabalham com PLCs, este programa também é conhecido como “sistema executivo”, “sistema operacional” ou “monitor”. Segundo Simpson (1994), é o sistema executivo que transforma a capacidade de computação genérica de um microprocessador em um equipamento especializado como o PLC. O *firmware* pode, por exemplo, executar a interpretação das linguagens de alto nível de que o usuário dispõe para programar os PLCs.

O processador é outro componente essencial dos PLCs. O primeiro microprocessador foi o Intel 4004, lançado em 1971. Entretanto, só mais tarde tais dispositivos adquiriram a maturidade e confiabilidade para serem incluídos nos projetos dos robustos controladores programáveis. Os primeiros PLCs tinham o seu processador implementado em lógica discreta. De acordo com Morley (2005), o projeto do primeiro PLC previa duas placas: a fonte e a placa processadora. Todo o processamento deveria ser feito em *software*. Um protótipo assim construído revelou-se bastante lento, o que exigiu o acréscimo de mais uma placa chamada de *Logic Solver*. Seu propósito era implementar em *hardware* algumas funções muito usadas pelo *software* acelerando o processamento. A capacidade chamada de *microcoded*, que alguns processadores tinham, foi algo importante para o desenvolvimento dos PLCs. Segundo Michel (1990), alguns processadores tinham seu conjunto de instruções formado a partir da combinação de um conjunto de operações mais básicas. Uma memória associada a estes processadores era usada para programar essas combinações de operações básicas de modo a produzir um conjunto de instruções desejado.

O arranjo de processadores chamado *bit-slice* também foi relevante durante o desenvolvimento dos PLCs. Tal técnica consiste em combinar vários processadores com um número menor de *bits* (por exemplo, 4 processadores de 4 *bits*) para obter um processador com um número maior de bits (resultando no exemplo anterior um processador de 16 bits).

No final dos anos 70 e início dos anos 80, o preço dos microprocessadores caiu dramaticamente, e estes se tornaram componentes padrão dos PLCs. Segundo Michel (1990), o uso de PLCs em sistemas de automação que envolviam lógica e seqüenciamento já havia sido testado e aprovado. Isto fez com que os fabricantes de PLC enxergassem um potencial de crescimento de mercado muito forte caso estes equipamentos comesçassem a ser usados com outras aplicações, tais como:

- processamento analógico de sinais (controle de processos);
- comunicações entre máquinas e entre máquinas e homem;
- processamento numérico.

Os PLCs com estrutura baseada em um único processador se mostravam insuficientes para cumprir tais tarefas pois:

- o acesso à memória se constituía num gargalo;
- o barramento ficava permanentemente ocupado;
- a sobreposição de atividades de natureza diferente (seqüencial, computacional, controle, comunicação) necessitava de um gerenciamento complexo com conseqüências para a estrutura, volume e desempenho do sistema operacional;
- o aumento da complexidade do sistema levava a uma diminuição da sua confiabilidade.

O menor custo dos microprocessadores somado a uma necessidade de maior capacidade de processamento dos PLCs resultou no uso de multiprocessamento nestes equipamentos. O multiprocessamento apareceu sob diversas formas, como, por exemplo:

- executando tarefas específicas dentro das unidades de processamento central. Um processador podia ficar dedicado a executar o programa desenvolvido pelo usuário e gerenciar as entradas e saídas da máquina enquanto outro processor ficava inteiramente dedicado a cuidar das tarefas de comunicação com outros equipamentos;
- dentro da unidade central de processamento executando múltiplas tarefas genéricas (o que exige um sistema operacional mais refinado) ou como coprocessador de outras linguagens de programação;
- dentro de módulos entrada e saída inteligentes, por exemplo, os cartões de PID ou cartões para condicionamento de sinais de entrada analógicos especiais como os vindos de termopares. Kissel (1986) tem um capítulo dedicado a “Módulos de entrada e saída avançados” em que são mostradas fotos de um cartão dedicado à função de controlador PID fabricado pela Allen Bradley.

Mais recentemente, a disponibilidade de processadores cada vez mais poderosos fez com que muitas das tarefas executadas por processadores antes localizados em módulos de entrada e saída inteligentes passassem a ser executadas em *software* pela unidade central de processamento.

Rillo (1983) descreve o projeto e a implementação do *hardware* e *firmware* de um PLC especificamente voltado para executar a linguagem GRAFCET. Apesar de ser um trabalho acadêmico, trata-se de um exemplo representativo da constituição interna dos PLCs então comercializados. Provavelmente, os equipamentos industriais disponíveis na época não diferissem muito, pois eram construídos usando componentes semelhantes oferecidos pela indústria de

semicondutores. Enquanto o processador usado por Rillo(1983) foi um Intel 8085, Michel (1990) cita que o PLC da Siemens modelo S5-135U usava os microcontroladores Intel 8031 e o PLC da Festo FPC404 usava processadores Zilog Z80.

2.3 Periféricos de programação e documentação

Antes da popularização dos computadores pessoais (PC ou *Personal Computer*), a programação de PLCs era feita usando-se terminais de programação dedicados. Estes terminais eram equipamentos que deveriam ser tão robustos quanto os PLCs que iriam programar. Dispunham, normalmente, de teclado e um tubo de raios catódicos. Kissel (1986), Michel (1990), Pérez *et al.* (1992) e Simpson (1994) contêm capítulos ou seções inteiras dedicadas a estes dispositivos onde podem ser vistas fotos destes equipamentos produzidos por diversos fabricantes.

Michel (1990) indica que os terminais de programação mais primitivos e antigos usavam a memória e o processador do próprio PLC a ser programado. O projeto destes era muito próximo dos terminais “burros” usados para acesso aos computadores *mainframes*. Esta solução foi adotada pois processadores e memória eram componentes caros na fase inicial da história do PLC. Era economicamente inviável dedicar mais componentes à função de terminal, necessária apenas durante o desenvolvimento do programa do usuário e em eventuais manutenções.

Mais tarde foram desenvolvidos terminais inteligentes (com sua própria memória e processador). Estes terminais traziam a grande vantagem da programação chamada *offline*, ou seja, o programa podia ser escrito sem conexão de dados com o PLC. Isto permitia que grande parte do programa fosse desenvolvido nos escritórios. Um vez escrita uma primeira versão do programa, era necessário ir ao chão de fábrica para gravá-la no PLC e fazer seu teste, retornando ao escritório caso houvesse necessidade de grandes mudanças.

Os terminais de programação se comunicavam com os PLCs usando protocolos proprietários desenvolvidos pelos seus fabricantes. Ou seja, se uma indústria possuísse três PLCs de fabricantes diferentes, era necessário ter três terminais diferentes de programação. Segundo Kissel (1986), cada fabricante dava até um nome diferente ao seus terminais de programação criando uma terminologia complicada:

- a Allen Bradley os chamava de “terminal industrial” (*industrial terminal*);
- a General Eletric os chamava de “terminal de desenvolvimento de programa” (“PDT” ou *Program Development Terminal*);
- a Texas Instruments os chamava de “unidade de programação em vídeo” (“VPU” ou *Video Programming Unit*);

- a Square D os chamava de “programador com tubo de raios catódicos” (“CRT Programmer” ou *Catode Ray Tube Programmer*).

Para dificultar ainda mais, as linguagens de programação não eram padronizadas, como será detalhado em seções que se seguem. Cada um destes terminais tinha teclas de atalho específicas para acelerar a chamada de funções usadas nas linguagens definidas por cada fabricante. O resultado é que nem mesmo eram padronizados o formato e as funções dos teclados.

Existiam também terminais portáteis de programação com um número restrito de teclas e pequenos *displays* de LEDs ou cristal líquido. Alguns PLCs dispunham destes terminais incorporados a suas tampas frontais. Normalmente, estes pequenos terminais tinham uma funcionalidade mais restrita, sendo utilizados apenas para alterar dados ou pequenos trechos de código do programa do usuário.

Associados aos terminais de programação, existiam uma série de outros dispositivos com diversas funções. Para armazenar programas já desenvolvidos existiam unidades de fita perfurada e de fita magnética. Natale (1989) considera as unidades de fita perfurada como sendo mais baratas e resistentes às agressões do ambiente industrial que as fitas magnéticas. Outro dispositivo associado aos terminais era a impressora. Na época dos terminais burros, elas eram diretamente conectadas aos PLCs, imprimindo os programas desenvolvidos pelo usuário sem muitos refinamentos (por exemplo, comentários). A razão do estilo lacônico das impressões era a economia de memória. Sanduski (1989) exhibe listagens de programas *ladder* com estas características.

A partir de meados dos anos 80, os computadores pessoais (PCs) começaram a se tornar populares. Foram, então, desenvolvidos programas para que estes computadores cumprissem as funções antes desempenhadas pelos terminais de programação. Ao longo deste texto, chamaremos estes programas de interfaces de programação. A princípio, os PCs foram recebidos com ceticismo na nova função, pois tinham elevado preço e eram considerados frágeis para resistir ao ambiente industrial. O principal ponto de preocupação eram os discos fixos responsáveis por armazenar o trabalho desenvolvido no chão de fábrica. Um falha destes discos devido a agressividade do ambiente industrial (calor, poeira, umidade) faria perder muitas horas de trabalho de depuração do programa de uma máquina complexa. Para tentar contornar esta dificuldade, alguns fabricantes de PLC produziram computadores pessoais com projeto especial para torná-los mais resistentes (SIMPSON, 1994).

Um aspecto interessante sobre terminologia deve ser mencionado: era comum no passado o uso da sigla PC (*Programmable Controller*) para se referir aos controladores programáveis. Para evitar confusão, o uso da sigla PC com este sentido diminuiu muito a partir da popularização

dos computadores pessoais. A sigla PLC, apesar de ser marca registrada da empresa Allen Bradley (posteriormente adquirida pela Rockwell Automation), passou a ser utilizada para se referir a controladores de qualquer fabricante. Entretanto, os redatores da norma IEC61131-3 (IEC, 1993) ainda se referem aos controladores programáveis como PCs, provavelmente, para evitar eventuais disputas quanto aos direitos da marca.

Sandusky (1989) escreveu um artigo sobre o desenvolvimento dos dispositivos e programas usados para o desenvolvimento e, especialmente, a documentação de programas de PLC.

2.4 Periféricos de interface com o usuário

Na fase inicial de desenvolvimento dos PLCs, a interface com o usuário era muito semelhante à existente nos painéis de relés. Dados booleanos podiam entrar através de botoeiras e sair por lâmpadas de sinalização.

PLCs com grande número de *bits* de entrada e com capacidade de processamento aritmético permitiram a leitura de dados numéricos sendo usadas chaves *thumbwheel*. Tratava-se de uma chave com 10 posições onde cada uma representava um número. Quando conectada a entradas digitais do PLC, a *thumbwheel* indicava qual número o usuário escolheu. A indicação podia ser em código hexadecimal ou BCD. Várias chaves podiam ser combinadas para entrada de números com vários dígitos.

A partir dos anos 90, com o barateamento dos *displays* de cristal líquido, surgiram as chamadas IHM (Interface Homem-Máquina). Tais dispositivos consistem em um teclado, um display e um processador conectados, por meio de uma rede de comunicação de dados, a um ou mais PLCs. A invenção do PLC tornou a fiação de painéis bem mais simples que a existente na época dos relés. Entretanto, a parte da fiação relacionada à interface com o usuário, ou seja, a conexão de botoeiras, lâmpadas de sinalização e chaves *thumbwheel* ao PLC continuou existindo. Esses fios foram eliminados com a introdução das IHMs no mercado. Um aumento na quantidade de dados de entrada ou exibidos ao usuário pode ser feito a um custo muito baixo, bastando reprogramar novas telas. Na época das botoeiras, a entrada de mais dados implicava na compra de mais *hardware* (as botoeiras ou lâmpadas de sinalização bem como cartões de entrada e saída do PLC). Além disto os sistemas se tornaram muito mais flexíveis, permitindo uma melhor interação com os usuários.

Outro elemento de interface que surgiu após o desenvolvimento de redes de dados entre PLCs e da popularização dos PCs foram os sistemas supervisórios. O sistema supervisório tem

a função de facilitar o monitoramento e a operação de plantas complexas controladas com equipamentos digitais, por exemplo, os PLCs. Usando PCs conectados em rede com os equipamentos de controle digital, é possível avisar ao operador da ocorrência de alarmes, obter dele o seu reconhecimento e definir modos de operação em contingência. Todas estas ocorrências podem ser armazenadas em um banco de dados para posterior auditoria das ações tomadas pelo operador e para diagnóstico das falhas ocorridas. Moraes *et al.* (2001) dedica um capítulo inteiro aos “Sistemas supervisórios e interfaces homem-máquina (IHM)”. Em breves artigos, Maciel (2005) apresenta uma série de conceitos relacionados aos *softwares* de supervisão enquanto Vieira (2004) e Maciel *et al.* (2004) comentam diversos aspectos sobre IHMs.

2.5 Redes de comunicação de dados

Interligar os antigos painéis de relés a plantas industriais remotas era uma operação complicada e cara. Igualmente desafiador era interligar vários desses painéis para trocarem informações entre si obtendo-se uma operação coordenada. Praticamente, para cada *bit* de informação enviado a um local remoto era necessário um par de fios. As soluções obtidas eram limitadas e pouco flexíveis. Uma das grandes vantagens dos sistemas de controle e automação digitais é a facilidade com que se pode conectar estes equipamentos em rede com outros tais como microcomputadores, IHMs ou dispositivos de entrada e saída remota.

Michel (1990) classifica o período inicial da conexão de PLCs em redes como aquele entre os anos de 1978 e 1980. Nessa época alguns fabricantes de rede criaram o que este autor chama de redes homogêneas, ou seja, aquelas em que todos os equipamentos conectados deviam pertencer ao mesmo fabricante. Segundo Kissel (1986), a rede de dados MODBUS foi projetada pela Modicon em 1978, e o primeiro sistema usando esta rede tornou-se operacional em novembro de 1979. Assim como os terminais de programação, a rede desenvolvida por cada fabricante tinha um nome e protocolo diferente. Kissel (1996) lista alguns desses nomes:

- a Texas Instruments chamava sua rede de TIWAY;
- a Modicon chamava sua rede de MODBUS, denominação mantida até hoje;
- a Allen Bradley chamava sua rede de Data Highway;
- a GE chamava sua rede de GENet.

Michel (1990) comenta que a criação de redes proprietárias tem suas razões, assegurando ao fabricante todo o controle sobre as especificações de *hardware* e *software* da rede, que poderia adaptá-las melhor aos PLCs que ele próprio produzia.

No início dos anos 80, um consórcio entre Xerox, Intel e DEC (três grandes empresas da área de processamento de dados) definia a rede Ethernet. Ao longo dos anos seguintes, esta rede iria evoluir e se tornar o padrão de fato para comunicação de dados entre computadores nas áreas administrativas das empresas. A longo dos anos 80, ocorreu o aumento da concorrência entre indústrias onde os PLCs eram muito usados, em especial a automobilística. Isto levou essas indústrias a participarem de uma corrida em busca de mais eficiência, flexibilidade e qualidade. Um dos meios encontrados para obter sucesso nesta corrida foi um melhor gerenciamento e integração das informações dentro destas indústrias e da sua cadeia de suprimentos (fornecedores e distribuidores). Nesse período surgiram, por exemplo, as técnicas de suprimento *just in time*. Dentro desse contexto de integração e melhor gerenciamento das informações, principalmente as relacionadas com a produção, as redes de PLC proprietárias eram um problema sério.

Segundo Michel (1990), o protocolo MAP (*Manufacturing Automation Protocol*) foi definido pela General Motors, em meados dos anos 80, para permitir a comunicação entre segmentos espalhados de suas várias fábricas automatizadas com o uso de tecnologias e equipamentos heterogêneos. Lewis (1995) considera o MAP como sendo a mais notável tentativa de harmonizar os dispositivos de controle, instrumentos, protocolos de comunicação e PLCs existentes na indústria. De acordo com o mesmo autor, o MAP é um padrão agora estabilizado, mas não foi, particularmente, bem-sucedido pelas seguintes razões :

- os custos de *hardware* para prover interface de comunicação entre um PLC e uma rede MAP podiam ser muito altos;
- a performance de comunicação de um ponto a outro não era muito impressionante;
- mesmo que um dispositivo de controle completamente compatível fosse conectado a uma rede MAP, um problema significativo ainda permanecia: como transferir informação entre dispositivos que foram programados de maneiras diferentes ?

Simpson (1994) indica que o protocolo MAP é baseado em *Token Passing* e estruturado segundo o modelo ISO/OSI. O modelo OSI (*Open Systems Interconnection Reference Model*), produzido pelo instituto de normalização ISO (*International Standards Organization*), é composto por sete camadas. Cada camada representa algum tipo de transformação executada, por meio de *softwares* específicos, nas informações que trafegam na rede. Um conjunto de dados produzido por um nó da rede passa sequencialmente pelas diversas camadas implementadas neste nó até estar pronto para ser colocado no meio físico de transmissão da rede. Esses dados trafegam pela rede até chegarem ao nó de destino onde em ordem inversa eles passam pelas diversas camadas lá implementadas até poderem ser finalmente utilizados. Maiores informações sobre o

modelo ISO/OSI podem ser encontradas em (MICHEL, 1990); (SIMPSON, 1994); (SILVEIRA *et al.*, 1998) e (MORAES *et al.*, 2001) e em outros livros específicos sobre redes de comunicação de dados.

Normalmente, as redes de dados entre PLCs usam o conceito de mestre e escravo. Mestre é o equipamento na rede que tem o privilégio de iniciar uma comunicação com algum outro equipamento. Todos os outros equipamentos na rede sem a condição de mestre são chamados de escravos, pois devem apenas se pronunciar quando interrogados pelo mestre. Em alguns protocolos de rede, o estado de mestre fica permanentemente associado a um dado elemento da rede. Tal configuração seria apropriada, por exemplo, a uma rede composta de um PLC e várias entradas e saídas remotas por ele controladas.

Em outros protocolos, a condição de mestre pode ser repartida entre os diversos elementos que constituem a rede. Quando um dispositivo mestre termina de enviar e receber todas as mensagens então disponíveis, este transfere a condição de mestre para algum outro dispositivo (que era previamente escravo). Este mecanismo é chamado de *Token Passing* e é uma maneira de arbitrar o uso da rede para evitar que, num dado momento, todos falem e ninguém escute.

O mecanismo de arbitragem utilizado em redes Ethernet pode ser resumido na seguinte política: “os elementos da rede começam a usá-la apenas quando ela está desocupada”. Se, porventura, dois elementos começarem a usar a rede no mesmo instante, ocorre o que se chama de colisão. Os elementos participantes da rede têm mecanismos para detectar que ocorreu a colisão e, então, param de usar a rede. Cada um dos elementos da rede espera, então, uma quantidade de tempo aleatória antes de tentar novamente usar a rede, suficiente para evitar que ocorra uma nova colisão. Embora, conceitualmente, mais simples que o mecanismo de *token passing*, a detecção de colisão introduz atrasos não-determinísticos na transmissão de dados. Isto pode ser muito comprometedor em equipamentos que rodam em tempo real, ou seja, aqueles que precisam ter uma resposta temporalmente determinística e dentro de certos limites. É comum encontrar PLCs operando dentro desta condição. Devido ao uso generalizado de redes Ethernet, atualmente alguns PLCs dispõem de interface para ter acesso a esta rede. Seu uso em aplicações com tempo de resposta não-críticas, como gravação de programas e interface de monitoração de variáveis para depuração, é bastante interessante e viável. Entretanto, seu uso em sistemas de tempo real deve ser considerado com muito cuidado.

Uma série de artigos escritos por Shirasuna (2004; 2005A; 2005B) detalha aspectos do uso industrial das redes Ethernet. Mata (2005) apresenta uma série de informações sobre protocolos para comunicação voltados à automação e controle.

O instituto de normalização IEC (*International Electrotechnical Commission*) está

desenvolvendo, atualmente, a norma IEC61499 para a sistemas de PLC distribuídos. Nestes sistemas, o processamento é compartilhado entre diversos equipamentos conectados via rede. Maiores informações sobre esta norma podem ser encontradas em (JOHN *et al.*, 2001); (LEWIS, 2001).

2.6 Linguagens de programação

2.6.1 *Ladder*

A linguagem mais utilizada e mais conhecida para programação de PLCs é o *ladder*. Trata-se de uma linguagem gráfica onde a lógica a ser executada pelo controlador é descrita por um diagrama de interligação de contatos e bobinas. A forma pela qual estes contatos e bobinas são organizados no diagrama os torna semelhante ao desenho de uma escada, dando origem ao seu nome em inglês. Também é conhecida como diagrama de contatos (MORAES *et al.*, 2001); (NATALE, 1989) ou diagrama de relés (MIYAGI, 1996) .

O raciocínio necessário para entender e programar em *ladder* é muito semelhante ao usado nos diagramas elétricos dos painéis de relés. Esta é a principal razão para o enorme sucesso que esta linguagem obteve, pois ela transformou os PLCs em emulações digitais dos antigos painéis de relés. O uso do *ladder* permitiu uma das mais suaves transições de tecnologias já acontecida na história fazendo com que os usuários pouco ou nada sentissem, apesar da mudança abrupta na implementação da automação industrial. A aceitação da linguagem foi tão grande que, durante anos, muitos fabricantes produziram PLCs que tinham o *ladder* como única linguagem. Atualmente, a grande maioria dos técnicos de automação industrial conhece e trabalha apenas com esta linguagem. O mercado norte-americano aderiu incondicionalmente ao *ladder*, a ponto de John *et al.* (2001) ter uma seção em seu livro chamada “O modo americano de programação *ladder*”.

A predominância do uso do *ladder* não foi unânime, sendo objeto de polêmica. Os simpatizantes desta linguagem argumentam que esta é suficientemente poderosa para ser usada em qualquer tarefa de programação de PLC. Outros argumentam que a utilidade do *ladder* se restringe a certo universo de aplicações, apontando, nela, uma série de deficiências.

Pollard (1994) apresentou as seguintes vantagens do *ladder* :

- é uma linguagem gráfica e simbólica;
- é muito simples para interpretar;
- engenheiros de controle estão familiarizados com ela;
- os membros das equipes de manutenção conseguem entendê-la;

- é executada rapidamente;
- é produtiva no projeto e depuração;
- tem vasto suporte por parte dos seus fornecedores e terceiros;
- permite programação *on-line* com compilação em tempo real;
- cada uma das suas instruções é um objeto, permitindo futuros desenvolvimentos e integração;
- permite, facilmente, extensões do usuário.

Um dos entrevistados por Pollard no artigo citado anteriormente declarou: “*Não podemos nos tornar mais simbólicos do que com lógica ladder. Não há razão para abandoná-la.*”

O fato de a maior parte dos PLCs permitir que um programa em *ladder* seja alterado sem que a máquina por ele controlada seja parada (desde que não seja necessário modificar também o *hardware*) é muito apreciado pelos seus usuários. Por ser uma linguagem de uso específico permite que o *firmware* sobre a qual ela se apóia seja mais simples e, portanto, mais confiável do que os sistemas operacionais necessários para executar linguagens mais genéricas.

Segundo Lewis (1995), uma outra razão para a popularidade do *ladder* é a facilidade com que programas envolvendo lógica booleana podem ser depurados. Geralmente, as interfaces de programação distinguem os estados lógicos dos contatos e bobinas com cores distintas. Isto permite identificar rapidamente erros num diagrama, bastando seguir um caminho no qual os contatos estejam ativados.

O artigo de Krigman (1985) discute a utilização da linguagem *ladder* no contexto das indústrias que fazem controle de processos em lotes. Este tipo de aplicação combina controle de variáveis analógicas com o controle de eventos discretos, abrangendo o seqüenciamento, intertravamento e até a otimização da produção. A partir da opinião de vários usuários, consultores e engenheiros que trabalham em indústrias que utilizam o controle de processos por lotes, foram levantadas as seguintes vantagens do *ladder*:

- familiaridade do pessoal de manutenção com esta linguagem;
- simplicidade para treinamento de técnicos;
- facilidade que permite a programação de PLCs.

O mesmo artigo apresenta críticas a esta linguagem:

- é muito apropriada para as tarefas antes executadas por relés eletromecânicos, mas inadequada para descrever o controle de processos ou a interface com o operador;
- é inflexível para ser usada no controle de processos;
- não oferece recursos para definição de receitas de produção;
- produz programas difíceis de documentar.

Ao longo do tempo, os vários fabricantes de PLC foram criando dialetos de *ladder* nos quais um grande número de pequenas diferenças foram introduzidas. Alguns exemplos de mudanças entre os fabricantes:

- regras de conexão dos contatos e bobinas;
- o modo de endereçamento das entradas e saídas;
- a máxima quantidade de contatos e bobinas colocadas por linha;
- a ordem de execução das instruções aritméticas;
- a direção em que o diagrama é executado pelo PLC (da direita para esquerda ou ao contrário).

Muitas destas diferenças se devem ao fato de o *ladder* ser interpretado pelo sistema executivo do PLC. Como cada fabricante implementa o interpretador da maneira que lhe é mais conveniente, fica difícil obter um comportamento padronizado. Miyagi (1996) apresenta uma metodologia para calcular as saídas de uma lógica representada por um diagrama *ladder* a partir das suas entradas. Esta metodologia consiste em transformar informações do diagrama em vetores e matrizes e usá-los na resolução de um conjunto de equações. Estes cálculos podem ser usados para implementar um interpretador de *ladder*.

Zhou *et al.* (1995) apresentou algumas limitações do *ladder*:

“... quanto maior o sistema de controle, mais difícil se torna determinar as especificações iniciais de projeto (como o sistema opera) examinando a lógica de controle. Isto torna o diagnóstico de problemas neste sistema difícil.

Programas ladder curtos (menos de 100 linhas) não são difíceis de entender. Entretanto, conforme o sistema se torna mais complexo, ele pode facilmente crescer para 1000 linhas ou mais, com as seguintes consequências:

- 1. A complexidade para validar e entender a lógica cresce exponencialmente (em geral).*
- 2. Torna-se mais difícil fazer modificações no programa para refletir mudanças das especificações funcionais do sistema.*
- 3. A tarefa de diagnóstico e manutenção do programa torna-se mais difícil.”*

Peshek *et al.* (1993) indica que uma aplicação de controle em tempo real possui um ou mais dos seguintes elementos:

- controle de eventos discretos;
- controle sequencial;
- processamento de variáveis analógicas.

Segundo estes autores, a linguagem *ladder* é adequada nas aplicações que envolvem controle de eventos discretos mas tem tido um sucesso menor quando usada com os demais elementos anteriormente citados.

A linguagem *ladder* foi muito apropriada nos primeiros anos da história do PLC pois a memória dos equipamentos era limitada devido ao custo, o que não permitia programas muito grandes. Os equipamentos eram usados apenas para executar a lógica booleana dos intertravamentos antes feitos com relés. A grande capacidade de memória dos PLCs atuais e a sua aplicação com finalidades além da lógica booleana tornou discutível a utilização exclusiva da linguagem *ladder*.

Para contornar estas deficiências e, ao mesmo tempo, se beneficiar da grande quantidade de *softwares* que suportam *ladder* e de técnicos de manutenção treinados no seu uso, Jiménez *et al.* (2001) ou Zhou *et al.* (1998) propõe o modelamento dos sistemas de automação usando redes de Petri e sua posterior implementação pela tradução para diagrama de contatos. Enfoque semelhante é apresentado por Moraes *et al.* (2001).

Lewis (1995) aponta uma série de deficiências associadas ao *ladder*:

- a) Geralmente, há um suporte limitado ou inexistente a procedimentos, funções ou blocos de programa. Resultam programas pouco estruturados e sem uma hierarquia clara. Em vários dialetos de *ladder* pode-se definir e chamar funções mas, na maioria dos casos, não há suporte para passagem de parâmetros, o que acaba sendo feito por variáveis globais. As funções resultantes não têm uma interface muito clara, o que desestimula o reuso de código. Muitas vezes um trecho de código idêntico manipulando variáveis diferentes acaba sendo replicado várias vezes ao longo do programa. Em geral, não existe o conceito de variáveis locais ou escopo de variáveis, ou seja, todas as variáveis da função devem ser globais. O programador acaba se ocupando com o gerenciamento de variáveis globais que poderiam ser locais. Na depuração, também muito tempo se gasta para diagnosticar problemas devido a efeitos colaterais causados por atribuições indevidas a estas variáveis globais.
- b) Não há suporte a tipos de dados estruturados definidos pelo usuário. Normalmente, são disponíveis apenas os tipos booleano, inteiro de 16 e/ou 32 *bits* e, mais raramente, ponto flutuante. Pode-se também usar arranjos ou seqüências destes tipos. Entretanto, não é permitida a mistura de tipos para formar uma variável para agrupar as informações associadas a alguma entidade do processo. O programador é obrigado a ficar gerenciando conjuntos dispersos de variáveis, algo que poderia ser feito automaticamente, como nas linguagens de alto nível para computadores (C ou PASCAL).

- c) Não existe suporte para seqüenciamento. Muitas aplicações industriais requerem uma ou mais seqüências de operação dependendo do estado operacional do processo controlado. É comum existir uma instrução chamada seqüenciador (*sequencer*) na linguagem *ladder* (KISSEL, 1986); (SIMPSON, 1994). Entretanto, falta flexibilidade a esta instrução, pois ela permite apenas um seqüência circular de eventos, ou seja, eles são executados um após o outro e chegando-se ao último retorna-se ao primeiro. Existe a possibilidade de se implementar máquinas de estado usando *ladder*, mas tal solução é trabalhosa, pois o programa que resulta, geralmente é bem grande, cuja manutenção é muito difícil.
- d) Não existe controle rigoroso sobre a taxa de execução do programa. Especialmente nos PLCs mais antigos, esta taxa era determinada pelo tamanho do programa: programas menores implicavam em taxas mais rápidas; programas maiores acarretavam taxas de execução mais lentas. Trabalhar nestas condições em sistemas onde tempo de resposta é crítico (tempo real) torna-se muito difícil. Algumas versões de *ladder* usando a instrução *Master Control Relay* permitem desabilitar ou não um conjunto de linhas do programa. Entretanto, esta instrução não oferece meios precisos para configurar uma parte do programa de modo que ela seja executada com maior freqüência que as demais. Trechos de programas não tão prioritários acabam tendo a mesma importância que trechos muito relevantes. Por exemplo, a rotina de verificação de erro e operação em emergência deve ser executada com maior taxa que outra para atualizar o estado das lâmpadas piloto do painel da máquina.
- e) Executar operações aritméticas usando *ladder* pode ser complicado. Um enfoque comum é chamar as operações aritméticas como blocos que podem ser habilitados ou não por meio de contatos. Normalmente, os endereços das variáveis analógicas de entrada e saída são passados como parâmetros para estes blocos. Fazer cálculos um pouco mais extensos irá necessitar de um grande número de blocos. Isto tornará o diagrama visualmente congestionado e difícil de trabalhar já que é raro as telas das interfaces de programação mostrarem mais do que cinco destes blocos ao mesmo tempo. O gerenciamento dos endereços das variáveis passadas aos blocos também se torna uma pesada tarefa que o programador deve executar com cuidado por ser altamente sujeita a erro.
- f) Nas versões mais antigas de *ladder*, não era possível associar nomes simbólicos aos endereços de memória usados como entrada, saída ou registradores de rascunho. Versões um pouco melhores permitem a associação de nomes simbólicos, mas é muito trabalhoso alterar o endereço inicialmente associado ao nome. Mesmo em PLCs mais recentes, o programador é obrigado a definir o endereço das variáveis internas. Não existem mecanismos para a própria interface de programação fazer isto como nas linguagens de alto nível para programação de

computadores. Novamente o tempo do programador é desperdiçado executando tarefas com alto potencial de erros e que poderiam ser automatizadas.

Segundo Lewis (1995), essas deficiências podem ser contornadas com o uso de bons padrões de programação, geradores de programas, ferramentas de documentação e bancos de dados para o gerenciamento das variáveis.

Alguns dos defeitos aqui citados não são exatamente causados pelo *ladder* mas pela infraestrutura (*firmware*) dos controladores sobre a qual a linguagem foi implementada. Como o *ladder* é dominante no mercado, as questões relacionadas a infra-estrutura acabam sendo associadas a ele.

No próximo capítulo será exposta a norma IEC61131 e como ela definiu um novo modelo de *software* em que uma série de recursos para resolver essas deficiências de infra-estrutura foram previstos. Os simpatizantes do *ladder* argumentam que muitas das críticas referidas anteriormente são procedentes apenas em dialetos antigos desta linguagem e que as suas versões mais modernas apresentam uma série de novas instruções e recursos capazes de contornar as deficiências apresentadas. O *site* da Rockwell (2005A) comenta que vários de seus PLCs, desenvolvidos antes da conclusão da norma IEC61131-3, já incorporavam uma grande quantidade de características desta norma.

Polêmicas à parte, *ladder* continua sendo a linguagem mais usada para programar PLCs como, por exemplo, atesta uma pesquisa feita no ano 2000 entre os leitores da revista americana Control Engineering (2000) indicando que 93% dos que responderam programavam nesta linguagem. A tradição do uso é um fator importante a ser considerado, principalmente neste contexto em que ela significa um grande montante investido no treinamento de equipes de projeto e manutenção.

2.6.2 GRAFCET

Segundo Baracos (1992) o GRAFCET (*GRAPhe Fonctionnel de Commande Etape/Transition*) foi inventado na França em 1979 por uma equipe criada pela AFCET (*Association Française pour la Cybernétique Economique et Technique*) que incluía tanto membros da universidade como da indústria. Os membros acadêmicos contribuíram tornando a linguagem rigorosa e poderosa, enquanto os da indústria contribuíram tornando-a aplicável aos problemas da vida real. Michel (1990) comenta que o GRAFCET inicialmente era uma metodologia destinada para o projeto e representação gráfica de algoritmos de controle de sistemas automatizados, ou, em outras palavras, para efetuar a análise funcional destes sistemas. A forma sintética de sua representação e a sua correspondência rigorosa com a lógica a ser programada

levou certos fabricantes a transformar o GRAFCET em uma linguagem de programação. O exemplo dado pela Telemecanique foi seguido por outros concorrentes, a ponto de, atualmente, podermos falar muito mais do GRAFCET como linguagem do que como metodologia. Trata-se de uma linguagem com fundamentos matemáticos sólidos, baseada em conceitos da teoria de automação como redes de Petri e máquinas de estado. Usando uma analogia informal e sem nenhum rigor matemático, podemos dizer que GRAFCET parece como uma Rede de Petri simplificada ou como uma máquina de estado melhorada. Assim como as máquinas de estado são um grafo onde estados são ligados através de transições, o GRAFCET é um grafo de passos também conectados por transições. O formalismo tradicional das máquinas de estado permite um e apenas um estado ativo a cada momento enquanto o GRAFCET, para modelar paralelismo de ações, permite mais de uma etapa ativa ao mesmo tempo. Uma rede de Petri, em que os pesos de seus arcos é convenientemente atribuído de modo que os lugares desta rede só possam ter marcação 0 ou 1, terá um comportamento muito semelhante ao de um grafo GRAFCET. Um tratamento matematicamente mais rigoroso sobre esta linguagem pode ser encontrado em (RILLO,1983).

Desde a sua criação, GRAFCET foi uma linguagem construída para programar de modo conveniente e flexível as seqüências de operação de acordo com a situação da máquina controlada (ao contrário do *ladder* que não é muito apropriado para esta tarefa). Existem inúmeros relatos nos quais o sistema de automação só teve sucesso utilizando o GRAFCET como linguagem de programação. Baracos (1992), por exemplo, apresenta um estudo de caso acontecido em 1984, no qual a concessionária de eletricidade canadense Hydro-Québec, ao notar a ineficiência dos demais métodos existentes para a programação de controladores, adota com sucesso o uso da linguagem GRAFCET. O caso descreve a implementação de um programa chaveador automático de linha para subestação de alta tensão. A versão anterior ao uso do GRAFCET havia sido implementada por um programa de 25 páginas de *ladder* enquanto o programa em GRAFCET ocupava uma única página. O programa que usa a nova linguagem é mais curto, mais intuitivo e com menor risco de erros de programação.

GRAFCET tem grande aceitação na França (onde foi criado) e no restante da Europa. A norma francesa NFC-03-190 e a norma internacional IEC848 definem a linguagem GRAFCET. Como veremos nos capítulos que se seguem, a norma IEC61131-3 rebatizou a linguagem GRAFCET como SFC (*Sequential Function Chart*). Esta norma também alterou alguns aspectos superficiais do GRAFCET ao definir o SFC de modo que a nova linguagem pudesse ser usada em conjunto com outras linguagens muito usadas para a programação de PLCs. Maiores detalhes sobre esta linguagem podem ser encontrados em (RILLO,1983); (MICHEL,1990);

(BARACOS,1992); (PÉREZ *et al.*,1992); (NATALE,1989) e (MORAES *et al.*, 2001). Maiores detalhes sobre SFC (a versão IEC61131-3 do GRAFCET) podem ser encontrados em (LEWIS,1995); (MIYAGI,1996); (BONFATTI *et al.*,1997) e (JOHN *et al.*,2001).

2.6.3 Outras linguagens de programação

Além do *ladder* e GRAFCET outras linguagens foram e são usadas para a programação de PLCs, especialmente na Europa. Uma delas é a conhecida como lista de instruções (IL - *Instruction List*). É uma linguagem textual onde a lógica é descrita como uma seqüência de mnemônicos. Normalmente, essas instruções executam operações tendo como entrada variáveis e/ou uma pilha interna do controlador e produzem saídas que são acumuladas na pilha ou em variáveis. Uma listagem de um programa escrito nesta linguagem tem uma aparência muito semelhante a um programa escrito em *assembler* de computador.

Diagramas de blocos também são comuns entre os que usam os PLCs para executar o controle envolvendo variáveis analógicas.

Linguagens de alto nível, como BASIC ou algo parecido, com textos de receitas descrevendo como o controle da máquina deve ser executado, são usadas bem mais raramente. Geralmente, essas linguagens são usadas para programar coprocessadores ou módulos inteligentes com algum uso específico como o controle de servomotores. Elas são apropriadas a este fim por descreverem de modo conciso expressões aritméticas. Encontra-se na literatura a citação de outras linguagens que não se consolidaram, o que resultou em um nível de utilização muitíssimo pequeno. Entre estas, podemos enumerar a tradução direta de fluxogramas (*flowcharts*), linguagens gráficas e textuais para a descrição de máquinas de estado, linguagens gráficas para descrição de lógica utilizando símbolos de portas lógicas, linguagens textuais baseadas em expressões booleanas entre muitas outras. Informações suplementares podem ser encontradas em (MICHEL,1990) que comenta um conjunto relativamente extenso de linguagens.

2.7 A evolução do PLC vista por meio das referências bibliográficas

Durante a elaboração desta revisão bibliográfica, notaram-se algumas correlações entre as linguagens usadas, os assuntos abordados nos livros, a data da sua publicação e a origem dos seus autores. Tais correlações foram constatadas de forma empírica a partir da análise direta de um número relativamente pequeno de livros consultados. Apesar desta metodologia ser questionável, ainda assim pode produzir resultados qualitativos interessantes.

A Tabela 2.2 mostra as diversas linguagens utilizadas pelos livros consultados. As linguagens que constam na tabela são *ladder*, GRAFCET, lista de instruções, diagrama de blocos, todas essas em suas versões não normalizadas anteriores à norma IEC61131-3, além das linguagens padronizadas por esta norma e um agrupamento para conter as outras. No capítulo que se segue, será apresentada a norma IEC61131-3, que criou versões padronizadas de *ladder*, GRAFCET, lista de instruções e diagrama de blocos. Além de padronizar estas linguagens, esta norma sugere uma série de recursos para tornar mais simples e consistente a programação dos PLCs. Por esta razão, criamos uma coluna separada para as linguagens normalizadas pela IEC. Ao construirmos a Tabela 2.2, não consideramos uma linguagem como sendo utilizada se ela foi apenas apresentada ou citada de modo breve. Por exemplo, isto ocorre, em relação ao GRAFCET (NATALE,1989) e (PÉREZ *et al.*,1992) e, por isso esta linguagem não consta como utilizada por estas referências.

Tabela 2.2 - Linguagens utilizadas nos livros consultados

Referência	Origem	<i>Ladder</i>	GRAFCET	Lista de instruções	Diagrama de blocos	IEC 61131-3	Outras
(KISSEL,1986)	Estados Unidos	✓					
(NATALE,1989)	Alemanha	✓		✓	✓		
(MICHEL,1990)	França	✓	✓	✓	✓		✓
(BARACOS,1992)	Canadá	✓	✓				
(PÉREZ <i>et al.</i> ,1992)	Espanha	✓		✓	✓		
(OLIVEIRA,1993)	Brasil	✓					
(SIMPSON,1994)	Estados Unidos	✓					
(LEWIS,1995)	Inglaterra					✓	
(MIYAGI,1996)	Brasil	✓				✓	
(BONFATTI <i>et al.</i> ,1997)	Itália					✓	
(SILVEIRA <i>et al.</i> ,1998)	Brasil	✓	✓		✓	✓	
(MORAES <i>et al.</i> ,2001)	Brasil	✓				✓	
(JOHN <i>et al.</i> ,2001)	Alemanha					✓	

Um outro ajuste feito na Tabela 2.2 foi em relação à origem de alguns autores. O livro escrito por Michel (1990), embora publicado na Inglaterra, foi uma tradução de um original francês. O livro escrito por Natale (1989), embora publicado no Brasil, foi considerado como tendo origem na Alemanha, pois foi patrocinado por uma empresa com matriz nesse país (Siemens) e também porque se refere a PLCs lá originados.

A Tabela 2.2 mostra claramente dois períodos:

- o anterior à IEC61131-3, que termina em 1994;
- o posterior à IEC61131-3, que se inicia em 1995 (a primeira versão da norma foi lançada em 1993).

Nota-se, no momento anterior à norma, que os livros com origem nos Estados Unidos usam apenas o *ladder* enquanto os livros de origem européia também utilizam outras linguagens.

No período posterior à IEC61131-3, todos os livros citam e utilizam as linguagens por ela padronizadas. Além das linguagens da norma, algumas referências ainda usam o *ladder* e o GRAFCET não normalizados enquanto o uso de outras linguagens não normalizadas quase desaparece.

Na Tabela 2.3 foram assinaladas a presença de alguns assuntos nos livros da bibliografia. Os assuntos foram escolhidos por mostrarem alguns aspectos interessantes da evolução dos PLCs. Por exemplo, o assunto "Terminais de programação" era abordado em praticamente todos os livros mais antigos. A popularização dos PCs e o desenvolvimento de interfaces de programação executadas por esses equipamentos tornou o assunto ausente dos livros mais recentes.

"Fundamentos de eletrônica digital" é um assunto que está deixando de ser abordado. Nos primeiros tempos dos PLCs, a tecnologia digital era usada de uma maneira muito mais restrita do que hoje. Talvez por isso, a maioria dos autores que publicaram antes de 1994 expunham os conceitos básicos de eletrônica digital tais como conversão de base de numeração, álgebra de Boole e mapa de Karnaugh. Os autores de publicações mais atuais consideram esses conceitos como pré-requisitos para quem deseja aprender sobre PLC.

Enquanto alguns assuntos estão perdendo espaço, outros estão ganhando. A apresentação de "Conceitos de engenharia de *software*" desenvolvidos a partir dos anos 70 para tornar a programação de computadores mais racional e eficiente, tem aparecido em alguns livros publicados após 1994. Os conceitos das "Redes de Petri" também têm sido abordados de modo mais freqüente neste período.

Tabela 2.3 - Alguns assuntos abordados nos livros consultados

Referência	Fundamentos de eletrônica digital	Terminais de programação	Metodologia de automação e exemplos	Conceitos de engenharia de software	Redes de Petri
(KISSEL,1986)	✓	✓			
(NATALE,1989)	✓	✓	✓		
(MICHEL,1990)		✓	✓		
(BARACOS,1992)		✓			✓
(PÉREZ <i>et al.</i> ,1992)	✓	✓	✓		
(OLIVEIRA,1993)	✓	✓			
(SIMPSON,1994)	✓	✓			
(LEWIS,1995)			✓	✓	✓
(MIYAGI,1996)			✓		✓
(BONFATTI <i>et al.</i> ,1997)			✓	✓	✓
(SILVEIRA <i>et al.</i> ,1998)	✓		✓		
(MORAES <i>et al.</i> ,2001)			✓	✓	✓
(JOHN <i>et al.</i> ,2001)			✓		

A maioria dos livros ensina algum tipo de metodologia para resolver os problemas encontrados em automação industrial. Coincidência ou não, os poucos livros que não fazem isto ficam restritos ao uso da linguagem *ladder* (Compare nas Tabelas 2.2 e 2.3 as referências (KISSEL,1986); (SIMPSON,1994) e (OLIVEIRA,1993)).

Restrições podem ser levantadas em relação à validade das correlações como:

- a quantidade de livros utilizada não é suficiente para ser usada como uma amostra estatisticamente significativa.
- alguns dos livros citados são destinados a técnicos enquanto, outros, a engenheiros. Isto pode tornar polarizadas as correlações relativas ao assunto.

- para maior precisão, o uso das linguagens e a presença dos assuntos nos livros deveriam ser quantificados por intermédio de alguma métrica. Como seria complicado definir esta métrica de maneira equilibrada então optou-se por uma classificação do tipo assunto ou linguagem presente ou não presente.

Apesar de estarem sujeitas a essas críticas, tais correlações fornecem uma noção do conjunto das referências consultadas. A mudança de importância dos assuntos abordados pelos livros sobre PLC também mostra que esta é uma área de conhecimento bastante dinâmica.

3 A NORMA IEC61131

Como é que cada um de nós os ouve na própria língua materna?
Atos dos Apóstolos, Capítulo 2, Versículo 8

A primeira parte deste capítulo apresenta um breve histórico da norma IEC61131-3 e suas principais características. Na medida em que existem livros inteiros (LEWIS, 1995) ; (BONFATTI *et al.*, 1997) ; (JOHN *et al.*, 2001) dedicados exclusivamente a este assunto e também artigos (LEWIS, 1996);(VAN DER WAL, 1999) que dão uma visão breve dos conceitos da norma, torna-se desnecessário uma apresentação dos seus detalhes. A segunda parte deste capítulo traça um panorama de como o mercado de automação industrial tem se posicionado em relação à norma.

3.1 Histórico da IEC61131-3

Segundo Lewis (1995), a maturidade de um ramo industrial pode ser verificada quando este precisa consolidar diferentes projetos e enfoques, sendo inevitável o aparecimento de algum tipo de padronização. O progresso de diversas tecnologias surgidas durante a Revolução Industrial foi atravancado enquanto não foram desenvolvidos padrões. É o caso da definição de bitolas comuns para os trilhos das ferrovias e de tensões e frequências padrão nominais para as empresas geradoras e distribuidoras de energia elétrica. Segundo Womack *et al.* (1992), Henry Ford obteve sucesso na criação das primeiras linhas de montagem automobilísticas após investir num sistema de padronização de medidas das peças montadas por estas linhas.

Ao final dos anos 70 e início dos 80, surgiram algumas normas nacionais para definir a programação de PLCs. John *et al.* (2001) cita a norma alemã DIN 40719-6 para padronizar o uso de diagramas de blocos de função enquanto Baracos (1992) cita a norma francesa NFC-03-190 para definição do GRAFCET. Tais normas não impediram que cada fabricante implementasse seus próprios controladores e linguagens do modo que lhe fosse mais conveniente.

Em 1979, a IEC (*International Electrotechnical Commission*) começou a definir uma norma relativa a vários assuntos ligados aos controladores programáveis como seu projeto de *hardware*, instalação, testes, documentação e programação. O comitê técnico dessa organização, ligado à medição e ao controle de processos industriais (TC75, ou seja, *Technical Committee 75*), criou um grupo de trabalho (WG7, ou seja, *Working Group 7*) com a função de discutir e escrever a norma. Tal grupo logo percebeu que a padronização de todos os assuntos relacionados a PLCs estava além de suas capacidades. Algumas forças-tarefas de especialistas foram então estabelecidas para desenvolver as diferentes partes do padrão. A força-tarefa 3 recebeu o objetivo primário de desenvolver um novo padrão de linguagens que se tornou a parte 3 da então chamada

norma IEC1131 (abreviadamente referida como IEC1131-3). A partir de 1998, a IEC1131 passou a ser chamada de IEC61131. Esta norma foi a primeira a receber uma aceitação internacional nos meios industriais. Dentre os envolvidos na construção da norma, há representantes de diferentes fabricantes de PLC, fabricantes de *software* e usuários.

Outras partes da norma foram dedicadas aos assuntos mostrados na Tabela 3.1, reproduzida do livro escrito por Lewis (1995) .

Tabela 3.1 - Partes da norma IEC61131

Parte	Título	Conteúdo	Data publicação
Parte 1	Informação geral	Definição de terminologia e conceitos básicos	1992
Parte 2	Requisitos dos equipamentos e testes	Construção mecânica e eletrônica e testes de verificação	1992
Parte 3	Linguagens programáveis	Estrutura do <i>software</i> do PLC, linguagens e execução de PLCs	1993
Parte 4	Linhas mestras para usuários	Orientação para seleção, instalação e manutenção de PLCs	1995
Parte 5	Especificação de serviços de mensagens	Facilidades de comunicação usando protocolo baseado no MAP	Previsto para 1998

A coluna mais à direita da Tabela 3.1 indica o ano de publicação da primeira versão de uma parte da norma, o que não significa que o trabalho em torno deste assunto tenha terminado naquela data. Por exemplo, nos anos posteriores a 1993, foram publicadas correções e emendas pertinentes à parte 3 (IEC61131-3).

John *et al.* (2001) comenta que a IEC61131-3 é um guia para a programação de PLCs e não um conjunto rígido de regras. Devido ao número enorme de detalhes, espera-se que os controladores programáveis estejam apenas parcialmente conformes. Esta norma fornece um conjunto de 62 tabelas de requisitos que devem ser declarados implementados ou não pelo fabricante do PLC. Obtém-se assim um *benchmark* que permite tanto a fabricantes quanto a clientes determinarem quantitativamente a conformidade de cada produto em relação à norma.

Segundo Lewis (1995), um dado fabricante pode dizer que atende à norma implementando apenas uma ou duas características simples de linguagem de programação. Para impedir que os requisitos fracos de conformidade da norma IEC61131-3 inviabilizem o sonho de *software* de controle industrial realmente portátil, foi criado um organismo chamado PLCopen.

Trata-se de uma organização neutra, não ligada a nenhum fabricante ou produto, fundada em 1992, com o propósito de promover o desenvolvimento e o uso de *softwares* compatíveis para PLCs. Dentre as atividades desta instituição estão a criação de classes de conformidade com a norma e a habilitação de instituições independentes para a execução desse tipo de certificação.

3.2 Características da IEC61131-3

3.2.1 O modelo de *software*

A característica mais comentada (tanto positivamente quanto negativamente) da norma é a padronização de cinco linguagens diferentes para a programação de controladores programáveis. Entretanto, o modelo de *software* (*software model*) proposto pela norma é muito importante apesar de não ser tão enfatizado quanto seu caráter poliglota. Trata-se de um conjunto de conceitos que definem uma infra-estrutura para decomposição da solução dos problemas de automação em partes com interfaces formais e claras. O uso desta característica permite o desenvolvimento de grandes projetos de forma mais simples e estruturada do que acontecia nos PLCs anteriores à IEC61131-3.

A estrutura de decomposição do problema definida por este modelo é construída em camadas. Cada camada tem funções e características específicas. As camadas mais baixas na hierarquia estão mais ligadas ao controle do processo e são controladas pelas camadas superiores. Esta estrutura permite o encapsulamento de dados e da complexidade: os elementos hierarquicamente inferiores cuidam dos detalhes mais próximos do processo ficando aos superiores a atribuição de coordenar o sistema. A síntese, análise ou manutenção de sistemas construídos de acordo com essa arquitetura torna-se muito mais simples devido à sua estruturação mais consistente.

Os elementos mais inferiores na hierarquia do modelo de *software* são genericamente chamados de POUs (*Program Organisation Units* ou Unidades de Organização do Programa). Existem três tipos diferentes de POUs :

- programa (*Program*);
- blocos de função (*Function Blocks*);
- funções (*Function*).

Segundo a norma, “*programa é uma montagem lógica de todos os elementos de linguagens e construções necessárias para o processamento de sinal requerido para o controle de uma máquina ou processo por um sistema controlador programável*”.

Blocos de função são POU's que produzem um ou mais valores ao serem executados. Múltiplas instâncias (cópias) desses elementos podem ser criadas. Associada a cada uma dessas instâncias existe uma estrutura de dados contendo suas saídas e variáveis internas, cujos valores devem permanecer inalterados entre uma chamada e outra deste elemento de *software*. Comparações têm sido feitas entre blocos de função e objetos encontrados na programação orientada a objetos. Ambos contêm dados encapsulados e têm métodos associados a estes dados. Entretanto, outras características encontradas nas linguagens orientadas a objetos como herança e polimorfismo não são suportadas pelos blocos de função.

Funções produzem um único valor de saída e podem ser chamadas como operandos em expressões algébricas ou lógicas em linguagens textuais (tal como as funções trigonométricas). Não é possível armazenar qualquer informação de estado em funções uma vez que estas não podem conter dados internos. Isto significa que uma função chamada com os mesmos parâmetros de entrada sempre produzirá o mesmo resultado.

De maneira geral, os programas são usados para construir soluções mais específicas, como o controle especializado de uma dada máquina ou parte dela. Os blocos de função e funções têm uma utilização mais geral sendo, portanto, elementos muito apropriados para reuso em outras partes do projeto. Por serem mais específicos, normalmente, têm um tamanho menor que os programas. Enquanto as funções devem ser escolhidas para resolver problemas combinatoriais (onde não são necessárias variáveis internas para registro dos estados), os blocos de função são apropriados para a solução de problemas seqüenciais. Como exemplos, temos:

- uma função seria muito apropriada para cálculo do erro de uma planta controlada em malha fechada, pois este depende exclusivamente do valor da referência e da realimentação;
- um controlador do tipo PID (Proporcional Integral Derivativo) deve ser implementado como um bloco de função, pois o termo integral precisa ser armazenado entre uma chamada e outra deste POU;
- o controle de aquecimento de uma extrusora ou injetora seria melhor implementado usando-se um programa pois se trata de um problema mais específico.

Um programa pode conter blocos de função e funções mas não outros programas. Um bloco de função pode conter funções ou outros blocos de função.

Subindo na hierarquia, temos os *resources*. Estes elementos oferecem todas as características necessárias para se rodar programas. Este elemento pode ser visto como uma máquina virtual capaz de executar um ou mais programas. Uma característica interessante dos *resources* é que eles são uma divisão de *software* mas que também podem refletir uma divisão no

hardware. Por exemplo, um PLC com múltiplos processadores pode ser projetado para ter um *resource* associado a cada um desses processadores. Um *resource* de um PLC deve ser independente dos demais. Dentro de um *resource*, podem existir uma ou mais tarefas (*tasks*) para comandar a execução de programas e blocos de função contidos neste recurso. Um programa só é executado quando explicitamente associado a uma tarefa. Um bloco de função pode ser executado explicitamente quando acionado por uma tarefa. Caso não exista um bloco de função que esteja associado a nenhuma tarefa, então ele será executado quando o programa que o contém for chamado. Uma tarefa pode ser configurada para executar periodicamente os programas a ela subordinados ou apenas quando um evento ocorre (por exemplo, pela mudança de uma entrada digital). A periodicidade e a prioridade com que os programas são executados são definidos pelo usuário.

Configuração (*configuration*) é a camada hierarquicamente mais alta no modelo de *software* da norma. Uma configuração geralmente contém todos os elementos de *software* processados dentro de um PLC podendo conter um ou mais *resources*. A Fig. 3.1 mostra um exemplo dos diversos elementos que compõem o modelo de *software*.

A norma também definiu vários modos para permitir a troca de dados entre os elementos de *software* descritos aqui e também com elementos externos ao PLC, por meio de redes de comunicação de dados. Os principais fluxos de informações num sistema que usa PLC ocorrem entre:

- as entradas e saídas (digitais ou analógicas) e a unidade de processamento central;
- os diversos elementos de *software* em que o programa desenvolvido pelo usuário é dividido (funções, blocos de função etc.);
- a unidade de processamento central e sua interface homem-máquina, outras unidades de processamento ou sistema supervisório via rede de comunicação;
- entre o programa desenvolvido pelo usuário e o sistema executivo (*firmware*) do controlador para que ocorram as corretas inicializações deste programa ou para que ele seja executado, monitorado, parado etc.

Fig. 3.1 Exemplo ilustrativo do modelo de *software* da norma IEC 61131-3

O modelo de *software* (*software model*) prevê modos de definir e controlar hierarquicamente todos esses fluxos de informações por meio dos seguintes conceitos:

- caminhos de acesso (*access paths*) que permitem a troca de dados entre a configuração de um PLC com elementos externos, por meio de redes de comunicação (tal assunto é detalhado na parte 5 da norma);
- variáveis representadas diretamente que tornam possível endereçar posições bem definidas da memória do PLC. Normalmente, o compilador determina o endereço das variáveis declaradas, aliviando o programador do trabalho de definir e assegurar a coerência dos endereços. Entretanto, em alguns casos específicos, é desejável o acesso direto a determinadas posições de memória que cumprem alguma função especial em função do *hardware* (acesso a cartões de entrada e saída, por exemplo);

- variáveis globais que, definidas num determinado nível hierárquico, se tornam visíveis a todos os níveis subordinados. Uma variável global declarada numa configuração poderá ser usada em qualquer elemento de *software* do PLC enquanto outra declarada num *resource* só será visível aos POU's nele contidos.
- variáveis externas que podem ser importadas por programas e blocos de função se anteriormente tiverem sido declaradas como globais;
- blocos de comunicação (este tópico também está relacionado com a parte 5 da norma) que permitem a troca de dados. Tais blocos de função ficam ligados a um programa e não são visíveis externamente;
- parâmetros de chamada são usados como entradas e saídas dos POU's.

A Tabela 3.2 mostra quais dos mecanismos para a troca de dados são permitidos em cada tipo de elemento definido pelo modelo de *software*. Como não é possível definir dados nas tarefas (*tasks*), estas não participam da Tabela 3.2.

Tabela 3.2 - Métodos de comunicação disponíveis aos elementos de configuração e aos POU's

Método de comunicação	Configuração <i>Configuration</i>	Recurso <i>Resource</i>	Programa <i>Program</i>	Bloco de função <i>Function block</i>	Função <i>Function</i>
Caminhos de acesso (<i>access path</i>)	✓		✓		
Variáveis representadas diretamente	✓	✓	✓		
Variáveis globais	✓	✓	✓		
Variáveis externas			✓	✓	
Blocos de comunicação			✓	✓	
Parâmetros de chamada		✓	✓	✓	✓

As seguintes palavras reservadas podem ser usadas para definir a interface com os POU's:

- VAR_INPUT - parâmetro de entrada;
- VAR_OUTPUT - parâmetro de saída;
- VAR_IN_OUT - parâmetro de entrada e saída;
- VAR_EXTERNAL - variável externa;
- VAR_GLOBAL - variável global;
- VAR_ACCESS - variável que participa de um caminho de acesso (*access path*).

Na Tabela 3.3, são assinaladas quais das palavras reservadas podem ser usadas nos diversos tipos de POU.

Tabela 3.3 - Tipos de variáveis de interface permitidas nos POU

Tipo de variável	Programa <i>Program</i>	Bloco de função <i>Function Block</i>	Função <i>Function</i>
VAR_INPUT	✓	✓	✓
VAR_OUTPUT	✓	✓	
VAR_IN_OUT	✓	✓	
VAR_EXTERNAL	✓	✓	
VAR_GLOBAL	✓		
VAR_ACCESS	✓		

A configuração de um sistema e seus dados podem ser definidos usando-se tanto as linguagens textuais como gráficas padronizadas pela norma. Uma linguagem textual chamada “elementos comuns” (*common elements*) foi criada com esta finalidade e possui esse nome pois pode ser usada em conjunto com as demais linguagens destinadas à programação dos PLCs. A versão textual dos POU tem uma listagem definida em dois trechos:

- um cabeçalho escrito em “elementos comuns” no qual são definidos seus dados e a sua interface;
- um complemento escrito em qualquer uma das outras cinco linguagens de programação contendo o código a ser executado.

Os “elementos comuns” também possuem recursos poderosos para definir vetores, matrizes e dados estruturados criados pelo usuário do PLC para modelar as entidades por ele controladas.

3.2.2 Linguagens de programação

O primeiro passo tomado pelo grupo de trabalho que produziu a norma foi rever as técnicas e linguagens oferecidas por diversos fabricantes de PLC. O grupo, então, racionalizou as diferentes linguagens e os numerosos dialetos de *ladder* para criar um conjunto de cinco novas linguagens. A obtenção de um consenso entre os elaboradores da norma foi um feito notável,

apesar da participação de fabricantes de PLCs com tradições distintas, tais como Allen-Bradley e Siemens.

A IEC 61131-3, ao fornecer várias linguagens de programação, permite que o projetista do *software* escolha a linguagem mais apropriada para resolver um certo problema. Tal escolha pode ser feita por razões técnicas e/ou por familiaridade com alguma linguagem específica. Elementos de *software*, como blocos de função, podem ser programados em qualquer linguagem.

Em teoria, o maior benefício trazido pela utilização da IEC 61131-3 seria a possibilidade de executar um mesmo sistema de controle em diferentes PLCs, o que teria um grande impacto no mercado de PLCs, tornando-o mais aberto. Na prática, estamos longe desta situação, pois os requisitos fracos de conformidade que a norma coloca permitem que cada fabricante implemente um conjunto diferente de linguagens e/ou outros conceitos do modelo de *software*. Os formatos dos arquivos utilizados para armazenar os programas de PLCs não foram padronizados, tornando difícil portar código entre PLCs de fabricantes diferentes. A organização PLCopen definiu um formato baseado em ASCII para blocos textuais chamado FxF, mas sua implementação pelos fabricantes não é muito comum. Existem interfaces de programação que armazenam seus programas usando XML (*eXtensible Markup Language*), mas também não existe uma padronização para que isto seja feito de modo uniforme. Williams (1999) comenta essas dificuldades, entretanto, vê a criação de uma biblioteca básica normalizada pela IEC61131-3 como ponto positivo, pois a sua utilização traz alguma facilidade aos que desejam portar código entre equipamentos diferentes.

As cinco linguagens definidas pela norma são:

- ST (*Structured Text*) é uma linguagem textual de alto nível que encoraja a programação estruturada e cuja sintaxe lembra fortemente o PASCAL;
- FBD (*Function Block Description*) é uma linguagem gráfica construída a partir da conexão de blocos de função (*function blocks*) que processam sinais e fluxos de dados;
- LD (*Ladder Diagram*) é uma linguagem gráfica baseada nos diagramas de lógica de relés;
- IL (*Instruction List*) é uma linguagem textual de baixo nível parecida com assembler;
- SFC (*Sequential Function Chart*) é uma linguagem gráfica que permite definir o comportamento seqüencial de um sistema de controle. Tal evolução pode ser determinada por eventos específicos ou simplesmente por intervalos de tempo.

Segundo Bonfatti *et al.* (1997), SFC é a linguagem mais apropriada para ser usada nas etapas iniciais e nas fases mais cruciais do ciclo de desenvolvimento de *software*. As razões para isto são:

- o SFC é a linguagem com maior poder de expressão, sendo apropriada para descrever problemas dinâmicos (inclusive aqueles onde ocorre o paralelismo de ações);
- é uma linguagem gráfica de fácil utilização;
- tem grandes vantagens como ferramenta de modelamento do sistema, ajudando a criar a primeira representação formal do seu comportamento, quando muitos aspectos ainda não foram definidos ou ainda são ignorados pelo projetista;
- a utilização do SFC também evita as ambigüidades surgidas pelo uso da linguagem natural na comunicação entre clientes, projetistas e programadores. Convém lembrar que o GRAFCET (antecessor do SFC) foi criado como uma metodologia de análise e, só posteriormente, se transformou numa linguagem de programação;
- permite o desenvolvimento e refinamento paulatino do projeto. Por exemplo, ações associadas a um passo de um SFC podem ser implementadas como um outro SFC.
- permite uma conexão natural com as demais linguagens que podem ser usadas para descrever transições e ações elementares.

A norma define a sintaxe de suas linguagens textuais (inclusive os “elementos comuns” - *common elements*) de modo formal e sem ambigüidades fornecendo, num de seus apêndices, a BNF¹ delas. A semântica das linguagens é definida por meio de comentários feitos ao longo do texto da IEC61131-3.

3.3 IEC61131-3: Prós e contras

Apesar do esforço feito pelos redatores da IEC61131-3 no sentido de padronizar linguagens e de criar um modelo de *software* consistente, esta norma não obteve uma aceitação unânime. Como exemplo, será reproduzida uma série de críticas apresentadas contra a norma em (CRATER, 2005A). Segundo o autor deste artigo, o conceito de um conjunto (suíte) de linguagens padronizadas vai contra o interesse da base de usuários de PLC e o padrão proposto é retrógrado e específico a ponto de ser medíocre. A seguir, é apresentado um resumo das principais críticas de Crater:

- As linguagens escolhidas são antiquadas e de origem européia sem qualquer interesse substancial para o mercado americano;

¹ BNF (*Backus Naur Form*) é uma linguagem para descrição da sintaxe de linguagens de uso muito comum entre os que estudam e implementam compiladores.

- A normalização de linguagens não é aconselhável, pois pretende congelar o tempo em 1992;
- A esperança de portabilidade é ilusória, pois a própria norma permite que os fabricantes de equipamentos incluam extensões ao padrão. Pressões competitivas ou oportunismo irão fazer com que esses fabricantes criem numerosas extensões aos seus conjuntos de linguagens. Buscando os usuários de PLC obter sempre o melhor desempenho, essas extensões serão prontamente aceitas, resultando na proliferação de programas não portáteis;
- Os benefícios da norma serão colhidos por pequeno conjunto de fabricantes, deixando de fora a comunidade de usuários. Por definir a constituição interna de uma classe inteira de produtos em um nível tecnologicamente medíocre, os produtos no mercado serão parecidos, retirando do usuário a liberdade de escolha;
- O modelo de comunicação proposto é obsoleto. O uso de variáveis globais e *links* conceituais para comunicação interprocessos irá, no futuro, impor limitações substantivas, principalmente quando se tentar integrar controladores usando este modelo de comunicação a ambientes orientados a objetos;
- ST, FBD e LD são caracterizadas como de baixo nível;
- A linguagem ST não tem uma personalidade própria. Embora ela incorpore algumas das funcionalidades das linguagens convencionais de programação de computadores, também difere em um número suficiente de aspectos de modo a aborrecer seus usuários. Se a norma devesse conter uma linguagem orientada a programação de computadores, por que não se adotou C , talvez com algumas extensões para processamento em tempo real ?
- SFC, após um exame mais próximo, tem o efeito de introduzir mais um nível de complexidade aos programas. Em vez de uma linguagem unificada com um número relativamente pequeno de elementos estruturais, SFC força o usuário a pensar em dois domínios simultaneamente: o seqüencial, em nível SFC, e o combinatorial, em nível de seus estados. Tem-se a impressão de que o SFC tentou, sem sucesso, agradar a todos.

O mesmo autor apresenta em outro artigo (CRATER, 2005B) uma linguagem chamada *State Language* como alternativa às linguagens da IEC61131, especialmente o *ladder* e o SFC.

Por outro lado, Lewis (1995) aponta as seguintes vantagens no uso dos conceitos da norma:

- o encorajamento da construção de programas bem estruturados pelo uso do modelo de *software* e de POUs (*Program Organization Units*);
- a “tipagem” (*typing*) forte de dados, o que significa que a interface de programação do PLC irá detectar quando o programador tenta atribuir um dado de formato errado a uma variável;

- a possibilidade de executar partes diferentes do programa com taxas e prioridades diferentes;
- a descrição de comportamentos sequenciais complexos pode ser feita de forma clara utilizando-se a linguagem SFC;
- a linguagem de elementos comuns (*commom elements*) permite a criação de estruturas de dados que agregam variáveis de tipos diferentes para descrever entidades controladas pelo programa;
- seleção flexível de linguagens de acordo com o problema e as habilidades dos projetistas e programadores;
- a possibilidade de criar soluções portáteis e apropriadas ao uso em PLCs de fabricantes diferentes. Na prática, esta vantagem ainda sofre algumas restrições comentadas anteriormente.

Convém notar que, quase todas as vantagens listadas, são um contraponto à lista de deficiências dos sistemas programados exclusivamente em *ladder*, que foram mencionadas no Capítulo 2. Tal resultado não é casual, sendo um dos objetivos do comitê que produziu a norma IEC 61131-3.

3.4 O mercado e a IEC61131-3

3.4.1 Informações quantitativas

Existem poucas informações quantitativas sobre o mercado de PLCs. A empresa Frost & Sullivan realizou, em 1995, um estudo sobre o mercado mundial de PLCs, dado citado por (PLCOPEN,2005). Tal estudo previa (para o período entre 1993 e 2000) uma participação maior no mercado dos PLCs de pequeno porte e menor preço unitário. O aumento no número de unidades vendidas seria maior do que o aumento na receita dos fabricantes desses equipamentos.

Um artigo mais recente, publicado em 2004, na versão europeia da revista Control Engineering (LLOYD,2004) indica que o crescimento da receita das empresas no mercado de PLCs cresce numa taxa que não excede 5%, sendo menor do que muitos esperariam. Este artigo endossa as idéias do relatório da Frost & Sullivan em relação ao crescimento da participação dos PLCs de pequeno porte. O estudo da Control Engineering indica que a região asiática deverá assumir o posto de segundo maior participante do mercado mundial de PLCs, ultrapassando a América. A Europa é apresentada como o principal participante do mercado mundial de PLCs devido ao seu grande número de fabricantes de máquinas industriais.

Informações quantitativas gerais sobre o mercado de controladores programáveis no Brasil são precárias. A revista Mecatrônica Atual (VIERA,2005) tentou, sem sucesso, fazer um levantamento entre os fornecedores desses equipamentos no Brasil. Segundo o mesmo artigo,

nem mesmo a própria ABINEE (Associação Brasileira da Indústria Elétrica e Eletrônica) conseguiu levantar tais dados no ano de 2000. A revista acaba apresentando dados obtidos junto à Receita Federal sobre a quantidade de equipamentos importados, que não permitem uma avaliação consistente do mercado de PLCs. Segundo o levantamento efetuado, existem, no Brasil, 38 empresas atuando neste mercado.

Quanto à adoção da norma, pesquisas e estudos publicados pela revista Control Engineering mostram dados interessantes, mas representativos apenas do mercado americano.

No ano de 1994 (POLLARD,1994), o *ladder* era apresentado como linguagem única de programação de controladores programáveis da maioria dos fornecedores americanos. Um dos executivos de uma empresa produtora de *software* para PLCs entrevistado declarou: “*Nós estamos preparados para a IEC1131 a qualquer momento em que ela se tornar viável. Entretanto, achamos que o ladder irá prevalecer entre nós por um período significativo.*”

No ano de 1995 (POLLARD,1995), a revista Control Engineering comentou os desafios colocados aos principais fabricantes de PLCs, sendo a adoção da IEC1131 um deles. Os resultados de um questionário enviado a diversos fabricantes de *software* para PLCs mostram que todos eles tinham intenção de se tornarem conformes à norma de maneira paulatina, adotando-a apenas parcialmente (apenas em alguns modelos de seus PLC e não necessariamente implementado todas as linguagens). Posição semelhante pode ser encontrada, por exemplo, no *site* da empresa americana Rockwell Automation (ROCKWELL,2005A) que comenta a sua participação ativa nos comitês que definiram a norma e informa que seus principais produtos foram lançados antes do término dos trabalhos da IEC. Anunciava também um trabalho de migração em direção ao padrão internacional. Um ano antes da publicação da IEC61131-3, um gerente da Allen-Bradley (empresa antecessora da Rockwell Automation) já havia publicado um artigo antecipando os principais pontos da norma (OULTON,1992). Mais recentemente esta companhia (ROCKWELL,2005B) apresentou uma interface de programação que dispõe do FBD, ST e SFC, além do *ladder* como linguagens de programação.

No ano 2000, (CONTROL ENGINEERING,2000) a revista Control Engineering incluiu em uma pesquisa entre seus leitores uma questão para levantar a importância da IEC61131-3. Dentre os que responderam à pesquisa:

- 41% atribuíram alguma importância à norma;
- 39% atribuíram nenhuma importância;
- 13% não responderam a esta questão;
- 7% indicaram a norma como uma necessidade.

3.4.2 Aspectos estratégicos

Para uma análise do impacto estratégico da IEC61131-3 no mercado de automação, serão usadas as idéias de Michael Porter (1986) desenvolvidas no início dos anos 80 para analisar a competição entre empresas de um determinado ramo. Segundo esse autor a concorrência entre empresas é definida por cinco forças:

- a rivalidade entre as empresas existentes;
- ameaça de novas empresas ingressantes;
- ameaça de produtos ou serviços substitutos;
- o poder de negociação dos clientes;
- o poder de negociação dos fornecedores.

As cinco forças listadas refletem o fato de a concorrência em um determinado ramo não estar limitada aos participantes já estabelecidos. Clientes, fornecedores, substitutos e ingressantes potenciais são todos concorrentes para as empresas já atuantes no sentido de ter a capacidade de diminuir-lhes a competitividade e/ou rentabilidade. Essas forças podem ter maior o menor importância dependendo de circunstâncias particulares do ramo analisado. Nas próximas seções, serão considerados aspectos relativos à “ameaça de produtos e serviços substitutos” e “ao poder de negociação dos clientes” aplicados ao mercado de PLCs. O número relativamente grande de fornecedores de controladores programáveis e a diminuição do crescimento da receita dessas empresas, mencionados anteriormente, indicam a existência de forte concorrência nesse ramo.

3.4.2.1 Ameaça de produtos e serviços substitutos

O aumento de capacidade de processamento trazido pela microeletrônica derrubou barreiras existentes dentro do mercado de automação, fazendo com que áreas de aplicação de equipamentos antes distintas tornem-se cada vez mais sobrepostas. Isto aumenta o número de clientes potenciais de uma empresa ao mesmo tempo em que faz crescer o seu número de concorrentes. Tal mecanismo pode ser ilustrado pelas publicações:

- “Quando controles convergem: CNC, PLC & PC” (GOULD,1999);
- Um texto da Foxboro tentando levar os usuários de PLC a considerarem a possibilidade de usar um DCS de baixo custo (FOXBORO,2005);
- Uma nota de aplicação (ARNOLD,2005), comparando os produtos da Foxboro em relação aos PLCs fabricados pela Allen Bradley.

Segundo Porter (1986), “*um ramo onde entram concorrentes de origem diversa pode exibir instabilidades na competição pois os novos participantes podem trazer novas estratégias deixando os demais com dificuldades de entender as regras do jogo e portanto para definir suas ações.*”

Os computadores pessoais (PC) vêm se firmando como possíveis substitutos ao PLC em determinados tipos de aplicação. Inicialmente, os PCs eram vistos com desconfiança pelos engenheiros de automação, pois:

- os sistemas operacionais tradicionalmente usados nessas máquinas, além de pouco robustos, não são adequados para uso em aplicações de tempo real;
- esses equipamentos não podiam ser programados usando as linguagens de programação de PLC já consolidadas entre as equipes de manutenção existentes nas fábricas.

No sentido de contornar essas dificuldades, foram desenvolvidas soluções chamadas *soft logic* ou *soft PLC*. Esses sistemas consistem em um módulo *run-time* e uma interface de programação. O módulo *run-time* interfaceia com o sistema operacional do PC ou reimplementa parte dele de modo a ter uma operação em tempo real robusta e confiável, e executa as tarefas típicas de controle. A interface de programação permite criar e baixar programas para o *run-time* utilizando linguagens típicas de PLC, geralmente as propostas pela norma IEC61131-3. Uma listagem dos fornecedores de *soft PLCs* pode ser encontradas em (RALIZE, 2004). ISAGRAF é produto digno de nota por ser um dos primeiros a oferecer as cinco linguagens da norma ((POLLARD,1995,p.88);(BONFATTI *et al*,1997, p.241, p.260); (ISAGRAF,2005).

Alguns artigos (CONTROL ENGINEERING, 2000); (STROTHMAN, 2002) mostram que o caminho adotado pelos fabricantes tradicionais de PLC para escapar da concorrência dos PCs é dar ênfase a pequenas unidades com capacidade capazes de interligação via rede (inclusive Ethernet) e funcionalidades específicas (tais como controle de movimento) antes presentes apenas em controladores de maior porte. O custo inicial relativamente grande de *hardware* e *software* de uma solução *soft logic* a torna pouco competitiva onde existe a necessidade de controlar pequeno número de pontos. Em projetos maiores, onde se justifica o pagamento desse custo inicial, o baixo custo incremental das entradas e saídas de um sistema baseado em PC aliado à sua flexibilidade e alta capacidade de processamento o torna uma opção bastante interessante.

Uma outra alternativa à competição entre PCs e PLCs é apresentada pela National Instruments (NATIONAL, 2005A); (NATIONAL, 2005B). Esta empresa se consolidou no mercado vendendo *hardware* digital e *software* voltado para instrumentação e aquisição de dados e agora oferece um *hardware* para controle baseado em componentes programáveis (FPGA, ou

seja, *Field Programmable Gate Array*). Este *hardware* é programado usando LabVIEW (2005) na expectativa de que a grande base de usuários acostumados a configurar sistemas de aquisição de dados dessa empresa usando esta linguagem possam prontamente desenvolver aplicações de controle.

A IEC61131-3, de certo modo, também pode ser vista como uma ameaça aos fabricantes tradicionais de PLC pelas seguintes razões:

- Antes da existência da norma, os PLCs originais eram baseados em *hardwares* aliados a *firmwares* altamente especializados na interpretação de *ladder*. As interfaces de programação eram construídas de maneira a recusarem uma instrução não válida (ou alguma que o interpretador de *ladder* não fosse capaz de interpretar) assim que o programador a tentasse introduzir no editor de *ladder* mantendo o programa editado sempre sintaticamente correto. Desse modo, o processo de compilação comum nas linguagens usadas em computador tornava-se desnecessário nos antigos PLCs. Ao definir cinco linguagens, algumas delas com sintaxes bem mais complexas que o *ladder*, a norma tornou impossível ou muito difícil o enfoque anteriormente adotado. Assim, os novos PLCs devem utilizar o processo de compilação ou compilação mais interpretação (como o usado por Java (SUN, 2005), Perl (CPAN, 2005) ou Python (PYTHON, 2005)) para tornarem-se viáveis. Para os fabricantes de PLC tradicionais, esta nova realidade implica em gastos de desenvolvimento e na retirada de uso de sistemas antigos nos quais já foram feitos grandes investimentos (que eventualmente ainda não foram totalmente amortizados). (JUER *et al.*, 1993); (KIM *et al.*, 1999) discutem o desenvolvimento de compiladores e interfaces gráficas dentro do contexto de IEC61131-3. Mais difícil para os fabricantes do que lidar com a multiplicidade de linguagens, talvez seja a implementação de uma série de características do modelo de *software* da norma que provavelmente demandam um sistema operacional de tempo real suportando as aplicações de automação.
- As tradições consolidadas entre os fabricantes e usuários de PLCs criaram todo um conjunto de usos e costumes nem sempre compatíveis com a norma. A IEC61131-3 traz um dilema para tais atores: abandonar as tradições e aderir à norma ou criar mecanismos (nem sempre simples) de compatibilidade com o passado.
- Como já foi comentado, existem dificuldades para o desenvolvimento de *software* de PLC realmente portátil baseado na norma. Caso esses obstáculos fossem removidos, existiria a possibilidade de o *software* (e em certa extensão também o *hardware*) se tornar uma *commodity* aumentando potencialmente a concorrência no mercado de automação.

Para os que oferecem soluções *soft logic* (ou outros equipamentos programáveis baseados em PCs como controladores de robôs), o uso das linguagens padronizadas pela norma traz vantagens, entre as quais podem ser citadas:

- O trabalho de definição das linguagens de programação (uma tarefa que consome muitas horas de engenharia) já foi feito pela IEC;
- As dificuldades de desenvolvimento de um compilador e criação de um *run-time* adequado não são tão complicadas, pois o *hardware* dos PCs tem requisitos de processamento e memória bem menos restritos que os encontrados em PLCs.

Pode-se encontrar sistemas *soft logic* ou outros equipamentos programáveis com um grau de conformidade com a norma maior que o encontrado em PLCs (KUKA,2005).

3.4.2.2 O poder de negociação dos clientes

Os clientes do mercado de automação industrial são reconhecidamente conservadores. Morley (2005) cita que o primeiro fornecedor de PLCs levou anos para convencer alguns dos seus clientes a substituir seus painéis de relés pelo novo produto. Lewis (1995) também comenta, no prefácio de seu livro, a maneira pouco entusiasmada com que a indústria reage à novidades. Segundo Pollard (1994), o custo de máquinas paradas era estimado em aproximadamente US\$4000 por minuto em uma empresa automobilística canadense. Este conservadorismo e pragmatismo fazem sentido num contexto onde uma escolha errada pode custar o emprego (ou a carreira) de um (ou vários) engenheiros. O conservadorismo, somado a um ambiente de padrões fechados, faz com que os clientes de PLCs tradicionais tenham pouco poder de negociação frente a seus fornecedores. Grande parte dos PLCs adquiridos pelo cliente final estão integrados em máquinas compradas de indústrias que as fabricam, sem que o cliente final possa interferir na escolha. Quando o cliente final compra PLCs diretamente (para a reforma de uma máquina, por exemplo) normalmente o faz em pequeno volume. Esses fatos diminuem ainda mais o poder de negociação desses clientes frente aos fabricantes de controladores.

Uma exceção notável a este comportamento conservador por parte dos clientes de PLC, é um documento chamado OMAC (*Open, Modular Architecture Controllers for Applications in the Automotive Industry*) produzido por um conjunto de empresas automobilísticas (OMAC,2005). Nesse documento é apresentada uma série de requisitos para a obtenção de equipamentos de controle econômicos, manuteníveis, abertos, modulares e escaláveis. Segundo esse documento, controladores com estas características irão prover benefícios tais como custo

inicial reduzido, tarefas de integração simplificadas, incorporação mais fácil de funções de diagnóstico do controlador, da máquina e do processo. Entre os requisitos apresentados, temos o uso da norma IEC61131-3 para a coordenação de tarefas entre controladores e para o controle de eventos discretos.

Uma reformulação no ensino das técnicas de automação industrial seria um fator capaz de contrabalançar o conservadorismo dos clientes de PLC sem comprometer a confiabilidade dos seus sistemas de automação industrial. Verwer (1999) indica a necessidade do ensino de engenharia ser multidisciplinar e equilibrado entre seus aspectos práticos e teóricos e constata que as aplicações de PLCs não são amplamente ensinadas na universidade apontando causas para isso:

- A programação de PLC tende a ser classificada como atividade de nível técnico em vez de atividade de engenharia;
- PLCs ainda são vistos como limitados à aplicações de controle lógico;
- Os métodos de programação dos PLCs são muito específicos com relação aos fabricantes. O ensino centrado num equipamento ou nos produtos de um determinado fabricante pode limitar a amplitude de conhecimentos transmitidos;
- A ausência de bases matemáticas teóricas faz com que a programação de PLCs seja pensada pelos acadêmicos como menos relevante;
- A teoria matemática associada ao controle quantitativo pode ser, predominantemente, ensinada em sala de aula. Programação de PLCs requer atividades mais ligadas ao laboratório, o que traz uma série de problemas, tais como limitação do número de estudantes devido a disponibilidade restrita de equipamentos e pessoal de apoio.

Segundo o mesmo autor (VERWER, 1999); (VERWER,2000) a adoção da norma IEC61131-3 usada para controlar plantas simuladas traz as vantagens apontadas ao final da Seção 3.3 para o ambiente do ensino de engenharia de controle.

Galceran *et al.* (2001) mostra o uso da norma aplicado ao que ele chama de laboratórios virtuais voltados ao ensino de automação industrial. Tais laboratórios consistem em sistemas desenvolvidos em PCs que permitem o projeto, programação e simulação de sistemas automatizados, sendo que estes podem ser utilizados, interativamente, por vários usuários por meio de uma rede local ou mesmo por meio da Internet. Os criadores do laboratório virtual estavam cientes da batalha entre sistemas baseados em PC e PLC pelo mercado de automação, assim os dois tipos de equipamentos podem ser usados para a construção de controladores dentro do laboratório. Além do uso do conceitos da IEC61131-3, outras técnicas, como o uso de redes

de Petri para o projeto e simulação feita em computador podem ser exercitadas pelos usuários desse laboratório. Uma série de *softwares* foram utilizados, por exemplo:

- IsaGRAF, um dos primeiros *softwares* para implementação de *soft-logic* disponíveis no mercado com as cinco linguagens da IEC61131-3; (POLLARD,1995,p.88); (BONFATTI *et al*,1997, p.241, p.260); (ISAGRAF,2005);
- LabVIEW, linguagem da National Instruments utilizada, inicialmente, em instrumentação, mas agora também para controle; (LABVIEW, 2005);
- MatLab 5.3, usado para simulação de sistemas físicos; (MATLAB,2005);
- Visual Object Net 2.0, utilizada para o desenvolvimento de redes de Petri; (VISUAL OBJECT,2005).

O laboratório das universidades tem um papel importante, seja pesquisando alternativas para as técnicas tradicionais de implementação da automação e controle, seja ensinando aos seus alunos essas técnicas. Nas universidades, ao contrário das indústrias, não existe a responsabilidade do alto custo associado às linhas de produção paradas ou compromissos com um ou outro determinado fornecedor de equipamento, constituindo-se em um ambiente onde os alunos podem exercitar e tornarem-se seguros quanto à aplicação de novas técnicas de automação e controle. Alunos assim formados, provavelmente, serão bem menos conservadores e avessos às novas técnicas quando forem trabalhar na indústria.

3.5 A adoção da IEC61131-3

A conclusão de um artigo escrito por Oulton (1992) merece ser reproduzida a seguir, tanto por ser escrita um ano antes da publicação da IEC61131-3 e manter-se incrivelmente atual, quanto por abordar questões relativas à adoção da norma dignas de nota:

“Apesar de o ainda não publicado padrão IEC 65 A/B (IEC1131) representar um significativo passo à frente, desafios consideráveis permanecem. Eles incluem:

- *Treinamento. Novas metodologias trazem consigo novas ferramentas e técnicas. Ganhar vantagens com a norma IEC 65A (IEC1131) requer investimentos em treinamento.*
- *Resistência. As técnicas trazidas pela nova norma requerem um enfoque requerem enfoques para a programação diferentes daqueles usados pela lógica ladder, e os usuários podem resistir às mudanças a menos que grandes benefícios estejam aparentes.*
- *Aplicação correta. Como qualquer outro conjunto de ferramentas, a nova norma terá sucesso apenas*

se aplicada corretamente. Uso efetivo da norma é algo que os programadores irão desenvolver com o tempo, talvez com a ajuda dos fabricantes.

Apesar desses desafios, este autor acredita que os benefícios que o novo padrão trará serão reconhecidos por um número crescente de usuários, e proverão impulso suficiente para vencer os desafios que restarem. Aplicando o novo padrão, os usuários não só podem melhorar a eficiência de seus sistemas de controle como reduzir, significativamente, seus custos (tempo e dinheiro) de projeto, desenvolvimento de programas, comissionamento e manutenção.”

Serão apresentadas, brevemente, as idéias de Geoffrey Moore (1996); (HUTT *et al.*, 1998, pp. 307 a 313), sobre a adoção de novas tecnologias. Moore classifica os clientes em cinco classes diferentes:

Fig.3.2 Ciclo de vida de um produto de nova tecnologia

- Entusiastas: são os interessados em explorar as últimas inovações, mas possuem pouco poder de influência sobre como a nova tecnologia é vista nas organizações em que trabalham, não tendo, portanto, poder para causar uma adoção mais geral desta.
- Visionários: têm o interesse em explorar novas tecnologias visando obter vantagem competitiva e têm poder para definir a adoção da tecnologia dentro das organizações em que trabalham.
- Pragmáticos: acreditam na evolução tecnológica e não em revolução. Procuram os produtos do líder de mercado com histórico reconhecido de benefícios e confiabilidade. As pessoas com este perfil costumam ter o poder para definir a grande maioria das decisões de compra dentro das organizações.
- Conservadores: sensíveis a preço e relutantes, adotam novas tecnologias apenas para não ficar para trás.
- Céticos: são críticos sempre presentes em relação aos modismos que circundam os produtos de novas tecnologias.

O ciclo de vida de um produto de nova tecnologia é apresentado em um gráfico na Fig. 3.2 onde são mostradas, no eixo horizontal, as classes de clientes que, predominantemente, compram o produto em cada fase da sua evolução, e o eixo vertical dá uma idéia do seu volume de vendas. Podemos dividir o ciclo de vida desses produtos em duas partes: Na inicial, ele é adotado pelos entusiastas e pelos visionários, enquanto, na segunda parte o volume de vendas cresce bastante devido à adoção do produto por um grande número de pragmáticos seguido pela decadência do produto quando esse passa a ser usado, predominantemente, pelos conservadores e céticos. Entre essas duas fases existe um momento crítico (semelhante a uma adolescência) chamado por Moore de abismo. O abismo só é vencido se a nova tecnologia atender 100% das necessidades dos pragmáticos, e muitas novas tecnologias não alcançam a segunda parte do ciclo de vida apresentado na Fig. 3.2 por fracassarem em atender esse requisito.

O estudo das idéias de Geoffrey Moore (1996) traz questões interessantes:

- Estará a norma IEC61131-3 no abismo tentando provar aos pragmáticos que é digna de uma adoção real e generalizada por parte destes?
- Caso ela esteja vivendo essa fase crítica, será ela capaz de ultrapassá-la com sucesso?

Tais questões são bastante polêmicas e difíceis de responder de forma objetiva (como tantas outras suscitadas pelo mercado de automação). Talvez parte da resposta esteja na definição dada por Moore (HUTT *et al.*, 1998, p 308) para o que ele chamou de *inovações descontínuas*, ou seja, aquelas que: “*requerem do usuário final e do mercado mudanças significativas em seu comportamento passado, com a promessa de benefícios igualmente significativos*”.

4 A MODERNIZAÇÃO DOS NAVIOS-VARREDORES

É doce morrer no mar...
Dorival Caymmi e Jorge Amado

Os capítulos anteriores mostraram um panorama dos PLCs antes e depois da IEC61131. Neste capítulo, iremos apresentar o projeto de modernização dos navios-varredores da Marinha do Brasil, que utilizou largamente os conceitos da norma.

4.1 Introdução ao sistema de varredura

Minas marítimas são artefatos de guerra lançados em áreas estratégicas tais como canais de entrada/saída de portos com o objetivo de impedir a passagem de navios nessas áreas. Caso um desses artefatos detecte a passagem de um navio, o mesmo é detonado, causando uma explosão submarina que danifica ou destrói a embarcação detectada. As minas mais simples e mais antigas detectam a presença de navios pelo contato físico. As mais elaboradas, operam por influência, sendo capazes de detectar à distância a variação de alguma grandeza física que indique a passagem de um navio. As principais grandezas físicas que podem ser monitoradas são:

- o ruído produzido pelo hélice do navio (influência acústica);
- as distorções no campo magnético da terra causadas pela massa metálica do navio-alvo (influência magnética);

A varredura consiste no processo de destruir a maior quantidade possível de minas presentes numa dada área do mar, ou seja, ‘varrer’ esta área. O resultado desse processo nunca é determinístico e tem sua eficiência medida pela probabilidade de um navio ser explodido após passar pela área varrida. Quanto menor esta probabilidade, maior a qualidade do processo de varredura. O processo de varredura recebe como entrada uma grande quantidade de informações: desde indicações sobre o tipo de mina existente e outras informações sobre a área a ser varrida tais como a profundidade, o relevo do fundo do mar, salinidade e correntes marítimas.

A varredura é realizada por uma classe especializada de navios, denominados navios-varredores (MARINHA,2005) que rebocam, a uma distância segura, dispositivos que simulam a presença de um suposto navio-alvo (ver Fig. 4.1), fazendo com que as eventuais minas existentes sejam detonadas. Dispositivos para varrer minas de contato são puramente mecânicos, enquanto os utilizados na varredura de influência são eletromecânicos, necessitando, para o seu funcionamento, de equipamentos de apoio que ficam alojados no navio varredor.

Fig. 4.1 Funcionamento do navio-varredor

O navio-varredor percorre, numa trajetória em “zig-zag” (ver Fig. 4.2), a área a ser varrida. Muitas vezes, esse percurso precisa ser executado mais de uma vez para se alcançar uma qualidade razoável na varredura.

Fig. 4.2 Trajetória percorrida por um navio-varredor

A simulação da perturbação no campo magnético da Terra causada por um navio-alvo é denominada varredura magnética e pode ser realizada:

- por meio de uma cauda magnética, que são eletrodos metálicos que ficam em contato direto com a água do mar e através dos quais irá circular uma corrente elétrica;
- por meio de um eletro-ímã, com carcaça suficientemente robusta para resistir a explosões, denominada HFG (solenóide de impulso magnético, em alemão).

O navio-varredor possui um conjunto de geradores e equipamentos de controle associados que produzem as correntes elétricas usadas na varredura transmitidas aos dispositivos por um cabo de reboque.

A simulação do ruído mecânico do navio-alvo é denominada varredura acústica e é feita por um dispositivo chamado martelo acústico. Tal dispositivo consiste de uma cápsula que contém um motor elétrico cuja velocidade é controlada, o qual faz girar um eixo excêntrico que golpeia um diafragma de aço em contato com a água. Variando-se a velocidade do motor num determinado intervalo, obtém-se um espectro de frequências. Existem martelos construídos para cobrir várias bandas de frequência.

A Marinha do Brasil possui uma frota de seis navios que ficam sediados na Base Naval de Aratu, próximo a Salvador/BA. Tais navios-varredores, denominados classe “Aratu”, foram construídos nos anos 70 e possuem dois equipamentos de varredura acústica (martelos de baixo e médio tom) e dois equipamentos de varredura magnética (cauda magnética e HFG).

Os equipamentos eletrônicos de programação e controle do sistema antigo, fabricados há quase 30 anos, precisavam ser modernizados, pois havia uma dificuldade crescente de obtenção de sobressalentes, com o conseqüente aumento do número de avarias e redução da disponibilidade dos navios.

4.2 Visão geral do novo *hardware*

Um dos objetivos deste trabalho é abordar os conceitos da IEC61131-3 aplicados ao novo *software* de PLC produzido a partir da modernização dos varredores. Nesta seção, será dada apenas uma descrição superficial do *hardware* do novo sistema de varredura para permitir o entendimento da interação deste com o *software*.

O sistema de varredura é dividido em três sub-sistemas:

- sistema acústico de baixo tom;
- sistema acústico de médio tom;
- sistema magnético que controla a cauda ou o HFG.

O processo de modernização do *hardware* abrangeu os seguintes aspectos:

- os amplificadores de potência (*choppers*) de cada um dos sub-sistemas foram substituídos;
- os transdutores originais de cada um dos sub-sistemas foram substituídos;
- o sistema de controle original (implementado com amplificadores operacionais, relés e circuitos discretos) foi substituído por um PLC;

- o painel de operação original (implementado com chaves, lâmpadas, potenciômetros e instrumentos de quadro móvel) foi substituído por uma interface homem-máquina;
- foram mantidos os motores e sistemas mecânicos originais.

Como os conceitos da norma foram aplicados igualmente em todos os três sub-sistemas, iremos nos restringir ao detalhamento do sub-sistema acústico de médio tom ou, de forma abreviada, sistema MT. A Fig. 4.3 apresenta um diagrama de blocos simplificado do *hardware* do sistema MT.

No canto superior direito da Fig. 4.3 temos o motor de corrente contínua que produz o ruído acústico de médio tom. Este motor fica dentro do dispositivo chamado martelo MT, que é rebocado por meio de um cabo, a uma distância segura, pelo navio-varredor. Todos os demais blocos da figura se encontram no navio-varredor.

A velocidade do motor do martelo MT é a variável controlada, sendo estimada a partir das medidas de tensão e corrente de armadura. A corrente de campo desse motor também é controlada em malha fechada. Dessa maneira, três sinais de realimentação do motor são obtidos:

- a tensão de armadura do motor (VA);
- a corrente de armadura do motor (IA);
- a corrente de campo do motor (IF).

No canto inferior direito da Fig. 4.3, existe um bloco que representa o cartão de transdutores que transforma, e isola galvanicamente, os sinais de realimentação em tensões analógicas no intervalo de ± 10 V, que podem ser ligadas diretamente às entradas analógicas do PLC. O cartão de transdutores envia ao PLC o sinal digital CTOK, indicando que suas fontes de alimentação estão em bom funcionamento.

O controle do motor é realizado por intermédio das tensões aplicadas aos seus circuitos de campo e armadura produzidas por dois *choppers*. Estes são comandados pelo PLC que executa um algoritmo de controle que calcula os valores VC1 e VC2. Estes valores são convertidos pelo cartão de saída analógica do PLC em níveis 0..10V, que são conectados às entradas dos *choppers* respectivos, que atuam, simplesmente, como amplificadores de potência. Os *choppers* operam em malha fechada de tensão sendo ajustados para produzirem uma resposta rápida sem *overshoot*. Existem dois sinais digitais (REGMT1 e REGMT2) que indicam ao PLC que essas malhas de controle estão regulando dentro de uma faixa de erro aceitável. O estado de operação do *chopper* é monitorado pelo PLC por meio do sinal MTOK, que combina o diagnóstico interno de uma série de falhas.

Os *choppers* são alimentados a partir de uma tensão contínua de 220V disponível no navio, representada no canto superior esquerdo da Fig. 4.3. Entre a fonte de 220V do navio e o *chopper* existe o disjuntor de proteção DJMT e o contator C1MT. Este contator é comandado por uma saída digital do PLC de mesmo nome, que controla a conexão de uma das extremidades da bobina desse contator. A segunda extremidade da bobina é controlada pelo circuito de emergência descrito a seguir.

Fig. 4.3 Diagrama de blocos do *hardware* do sistema de varredura acústica de médio tom (MT)

O circuito de emergência está ligado a diversas chaves espalhadas em locais estratégicos do navio. Caso alguma dessas chaves seja acionada, a segunda extremidade da bobina de C1MT é desenergizada forçando seu desligamento. Esse circuito de emergência atua de forma análoga nos contadores de entrada dos outros sub-sistemas. A opção de implementar o sistema de emergência em *hardware* foi proposital, por razões de confiabilidade. Se essa tarefa fosse atribuída ao PLC, ter-se-ia um elo a mais de falha representado pelo *software*. Apesar disso, os estados das chaves de emergência também entram no PLC, o que permite que este tome medidas adicionais de segurança caso esteja operacional.

Tanto o disjuntor DJMT como o contator C1MT possuem contatos auxiliares que são monitorados pelo PLC (por meio dos sinais STDJMT e STC1MT). As letras ST no começo de seus nomes vêm da palavra "*Status*". Tal monitoração permite detectar falhas do *hardware*, levar o sistema a um estado seguro e enviar mensagens ao sistema supervisor indicando onde ocorreram falhas, tornando mais ágil a manutenção. No estado seguro, todos os elementos de hardware do sub-sistema no qual ocorreu o erro são desligados.

Internamente ao *chopper*, existe um grande capacitor de filtragem além de um contator que implementa o circuito de pré-carga deste capacitor. Quando o PLC comanda o fechamento do contator C1MT, o PLC ainda deve aguardar o sinal digital MTREADY indicando que o processo de pré-carga foi completado. Apenas ao final desta sequência é que o PLC deverá efetivamente liberar a operação dos *choppers* (por meio dos sinais lógicos ENMT1 e ENMT2) e produzir referências não-nulas para os *choppers* (VC1 e VC2).

No canto inferior esquerdo da Fig. 4.3, temos o programador, um computador do tipo PC com sistema operacional Windows, que se conecta ao PLC por meio de uma rede Ethernet. Este programador define a referência de velocidade (um sinal periódico) para o martelo MT, por meio de um conjunto de parâmetros enviados ao PLC antes do início efetivo do processo de varredura.

O PLC e seus cartões de entrada/saída analógicos e digitais são equipamentos comerciais. A escolha desses equipamentos foi realizada por meio de um processo de licitação promovido pela Marinha. O *chopper* e o cartão de transdutores foram equipamentos desenvolvidos especificamente para o projeto de modernização dos varredores.

4.3 Visão geral do novo *software*

A Fig. 4.4 mostra os principais blocos implementados em *software* no PLC e no programador.

Fig. 4.4 Diagrama de blocos do *software* do sistema de varredura acústica de médio tom (MT)

Convém ressaltar que esta é uma representação didática e bastante simplificada do *software*, apropriada apenas para dar uma primeira idéia do seu funcionamento. Os blocos representados são, na realidade, compostos de um ou mais módulos do programa do PLC, e existem outras trocas de dados entre estes módulos que não foram representadas.

À esquerda da Fig. 4.4, temos os blocos “Supervisório” e “Planejamento Tático”, executados no programador mencionado anteriormente, por meio do qual o operador interage com o restante do sistema.

Ao centro e direita da Fig. 4.4, temos os diversos blocos de *software* implementados no PLC. O bloco “Interface com Supervisório” recebe as ordens vindas do sistema supervisório e analisa a sua coerência. Sendo coerentes com o estado atual do sistema, elas serão aceitas e repassadas para o bloco “Sequenciamento” para serem executadas. Caso um dado comando não seja compatível com o estado atual do sistema, o bloco “Interface com Supervisório” notifica ao “Supervisório” a não aceitação deste comando. Por meio do “Supervisório”, o operador é avisado sobre o envio de um comando incoerente.

A função principal do sistema acústico de médio tom é simular o ruído acústico produzido pelo navio-alvo de uma mina de influência. Isto é feito pelo bloco “Controle”, que controla, em malha fechada, a velocidade do motor do martelo acústico de modo a produzir um espectro de frequências semelhante ao gerado pelo navio-alvo. Vários sinais, entre os quais a velocidade estimada do martelo, também são enviados ao sistema supervisório para que o operador possa acompanhar o andamento da varredura.

O bloco “Seqüenciamento” faz com que os diversos elementos de *hardware* sejam ligados/desligados de modo ordenado ao início/final da varredura. A interface entre este bloco e a planta é feita a partir das saídas digitais do PLC. O bloco “Seqüenciamento” é o responsável pela habilitação do sistema de controle, que só entra em operação após a preparação do *hardware*. O bloco “Seqüenciamento” também habilita/desabilita os erros a serem monitorados em cada etapa do processo. Os módulos de *software* de “Seqüenciamento” foram implementados com o uso de SFC.

O bloco “Detecção de erros” faz com que o sistema de varredura seja conduzido a um estado seguro caso algum erro ou exceção ocorra em qualquer etapa do processo. Como já foi dito, no estado seguro, todos os elementos de hardware do sub-sistema no qual ocorreu o erro são desligados. A “Detecção de erros” atua a partir das informações vindas da planta por meio das entradas digitais. O erro ocorrido (mesmo que de modo intermitente) é memorizado para que o operador seja notificado por meio do sistema supervisório. Desse modo, o operador do sistema pode fazer o diagnóstico e corrigir o problema. Após o reconhecimento (*acknowledge*) no sistema supervisório, a falha é apagada da memória do sistema.

À direita da Fig. 4.4, temos o bloco “Planta” que consiste nos dispositivos de varredura e de apoio à varredura, representado neste diagrama de *software* por produzir e consumir os dados manipulados pelos demais blocos.

Podemos enunciar os seguintes eventos relevantes para a operação do sistema:

- Comando de “Pausa” que só pode ocorrer quando houver uma varredura em andamento. Neste caso, a referência de varredura é reduzida a um valor mínimo, mas o motor continua girando. Toda a preparação de *hardware* feita pelo “Seqüenciamento” antes do início da varredura é mantida. Tal comando é usado para fazer as manobras de guinada necessárias para executar o zig-zag ilustrado na Fig. 4.2.
- Comando de “Desligamento” que só pode ocorrer quando houver uma varredura em andamento ou quando o sistema estiver em “Pausa”. É utilizado quando se deseja o desligamento efetivo do sistema, ao final da missão ou para manutenção. A referência de varredura é zerada e o seqüenciamento executa o desligamento de todo o *hardware* de forma ordenada.
- Quando ocorre um erro ou exceção, o sistema é conduzido a um estado inicial seguro, no qual todo seu *hardware* é desligado, independentemente da condição anterior. O erro será memorizado pelo módulo “Detecção de erro” e só será apagado quando o operador executar o reconhecimento (*acknowledge*) deste erro através do ‘Supervisório’.

A Fig. 4.5 apresenta o diagrama simplificado do SFC implementado pelo módulo “Sequenciamento”. Tal diagrama é uma abstração de alto nível do diagrama de transição de estados real do sistema, no qual vários estados da implementação final foram agrupados e representados como sendo um único “macro-estado”.

O sistema, sempre que ligado ou quando da ocorrência de um erro, vai para o estado inicial seguro (X1). Caso o módulo de "Detecção de erros" não tenha memorizado nenhum erro relevante para o sistema, então o sistema passa ao macro-estado de espera (M1). Se não houver erros, o sistema permanece no macro-estado espera (M1) até que o supervisor ordene o início de varredura quando, então, o macro-estado de preparação de varredura (M2) torna-se ativo. Neste estado (M2), o *hardware* e a configuração da “Detecção de erros” são colocados em condições de se habilitar às malhas de controle do sistema. O motor começa a girar, acelerando até a velocidade mínima de varredura e chega num novo estado de espera (X15) no qual aguarda a hora programada de início da varredura. Se ocorrer um comando de desligamento antes de que

Fig. 4.5 Diagrama SFC do sistema MT

chegue a hora de iniciar a varredura o sistema passa para o macro-estado de desmonte da varredura (M4). De outra maneira, ao chegar a hora do início da varredura, o sistema passa para o estado (X16) em que se executa a varredura e no qual a velocidade varia de acordo com uma referência pré-estabelecida.

Durante o estado de varredura (X16), o operador tem as opções de pausar ou desligar. Em qualquer um dos casos, o sistema vai para o macro-estado (M3) no qual a referência de velocidade vai para seu valor mínimo. Caso o comando anterior tenha sido “Pausa” o sistema permanece em M3 até a chegada de um novo comando de “Início” que irá causar o retorno ao estado (X15) no qual se aguarda novamente a chegada da hora de início de varredura. Caso o comando anterior tenha sido “Desliga” o sistema inicia o processo de desmonte (M4) no qual as ações de inicialização de *hardware* e configuração de erros feitas na preparação (M2) são feitas em seqüência inversa. Terminado o desmonte da varredura, o sistema volta ao estado de espera de comandos(M1).

4.4 Comentários finais

Uma vez que o PLC adquirido no processo de licitação não implementava todas as linguagens da norma IEC61131-3, tornou-se necessária a criação de ferramentas e metodologias que implementassem, na plataforma de *hardware/software* existente, os conceitos ausentes da norma. Tais ferramentas e metodologias, apresentadas nos capítulos a seguir, permitiram o desenvolvimento de um sistema melhor estruturado e de mais fácil manutenção.

5 CONTROLE

*O escravo está sempre sob controle,
e não escapa de ficar marcado pelos golpes.
Eclesiástico, Capítulo 23, Versículo 10*

Neste capítulo, será discutido o controle do sistema de varredura. Apesar da epígrafe bíblica dizer que o controle **sempre** pode ser realizado, existe todo um ramo da engenharia que estuda cientificamente os modos e as dificuldades para sua implementação.

O sistema acústico de médio tom é acionado por um motor DC variando-se a sua tensão de armadura e mantendo-se seu fluxo de excitação constante. Esta tarefa é realizada por três malhas de controle do tipo PI (Proporcional-Integral). A codificação foi realizada em duas etapas: Na primeira etapa, o controle é visto num nível mais alto, importando apenas as interligações entre os diferentes blocos de controle, desconsiderando-se os detalhes internos de cada bloco. Na segunda etapa, foram codificados os blocos de controle, suas interfaces e algoritmos.

A linguagem da IEC61131-3 mais apropriada para a etapa da interligação dos blocos de controle seria o FBD (*Function Block Description*). A linguagem mais apropriada para a manipulação aritmética dentro dos blocos de controle seria ST (*Structured Text*). O PLC utilizado no projeto de modernização dos navios-varredores não oferecia nenhuma dessas linguagens. Apesar da técnica de controle ser convencional, foram utilizadas métodos não tradicionais e conceitos da norma para a sua implementação na linguagem de PLC disponível.

O processador de macros M4 (2005), a ser descrito neste capítulo, foi uma ferramenta importante por suprir a falta da linguagem ST. Também serão mostrados neste capítulo a estratégia de controle adotada no sistema de médio tom, além de uma breve descrição dos principais blocos de função utilizados.

5.1 A tradução dos diagramas de controle

O projeto do controle dos sistemas de varredura produziu diagramas compostos pela interligação de uma série de blocos de função, sendo um exemplo dado pela Fig. 5.1.

Tais diagramas foram traduzidos para IL (*Instruction List*, disponível no PLC escolhido). Esta tradução não foi feita automaticamente, exigindo um sequência de etapas de codificação. Inicialmente, cada uma das linhas do diagrama foi associada a uma variável de 16 *bits* declarada na memória do PLC. Os blocos, escritos em IL, fazem operações de escrita e leitura nas variáveis de entrada/saída correspondentes. O nome e endereço das variáveis associadas às linhas também estão representados no diagrama. Existem algumas exceções em que a passagem de dados é

realizada por meio da pilha (*stack*) e, neste caso, não existe uma associação entre nome/endereço e a linha do diagrama.

Fig. 5.1 Exemplo de diagrama de controle

O diagrama da Fig. 5.1 pode ser traduzido pelo código apresentado abaixo:

"Pre-processamento do feedback

ESCALONA(IAmt,Amt_PC_IAmt_Num,Amt_PC_IAmt_Dem,Amt_VA_IAmt_esc)

```
COMPENSA(Amt_VA_Iamt_esc,Amt_PC_Iamt_OffsetLim,Amt_IS_Iamt_Mede,
        Amt_VA_Iamt_com,Amt_VA_Iamt_Ofs,Amt_IS_OfsIamt_Ent,
        Amt_IS_Iamt_Pronto,Amt_VA_Iamt_Soma,Amt_VA_Iamt_Cont)
```

```
SELESIN(Amt_VA_IAmt_com,Com_VA_Tes_Saida,Amt_IS_Hab5,Amt_IS_Modo5,
        Amt_VA_AuxPID3b)
```

Os blocos de função acima são, na realidade, macros escritas para serem expandidas pelo processador de macros M4.

5.2 O processador de macros M4

M4 é um processador de macros de uso geral, disponível na Internet (M4, 2005) para vários sistemas operacionais. O uso dessa ferramenta só foi possível porque o programa do PLC do novo sistema de varredura foi escrito em IL (*Instruction List*), que é uma linguagem textual.

No projeto de modernização dos varredores M4, foi usado para:

- declarar macros que se tornam funções *inline* dentro do programa. O mecanismo de chamada de funções disponível no PLC utilizado não permite a passagem de parâmetros. As alternativas de codificação existentes seriam implementar uma pilha virtual (para passagem de parâmetros) ou implementar as funções *inline*. Como o PLC utilizado dispunha de bastante memória utilização de funções *inline* foi possível e trouxe benefícios quanto ao tempo de execução.
- implementar mecanismos de compilação condicional. Na etapa de desenvolvimento, foram incluídos diversos trechos de código para auxiliar a depuração, que se tornaram desnecessários na versão final. O uso de macros permite escrever um único programa, que inclui (ou não) trechos de código de depuração, em função de diretivas de compilação.
- fazer a inclusão nos arquivos fontes, um após outro, dos diversos arquivos de descrição de macros M4 que compõem o projeto. Isto facilita a manutenção do programa. Caso seja necessário corrigir ou alterar uma macro, isto será feito em um único arquivo que contém a sua descrição o qual depois é incluído nos diversos módulos que fazem uso desta macro. O processo torna-se muito mais rápido, menos tedioso e sujeito a erros.
- declarar macros usadas para facilitar os trabalhos na edição dos arquivos de descrição do diagrama SFC.

O uso da linguagem M4 foi fundamental para contornar as deficiências das linguagens de programação oferecidas pelo PLC escolhido. Nas seções e capítulos seguintes serão dados diversos exemplos da utilização da linguagem M4.

5.3 A construção dos blocos de função

A construção de bloco de função sempre começa com a definição da sua interface com o resto do programa. A Fig. 5.2 mostra a equivalência entre a interface de um bloco de função nas suas versões gráfica e textual. Em alguns blocos, além de a interface conter entradas e saídas também são incluídas variáveis internas.

Fig. 5.2 Bloco de função hipotético em sua versão gráfica e textual

Ao contrário do que acontece quando se usa a linguagem de elementos comuns existente na IEC61131-3, o processador de macros M4 não faz distinção entre entradas, saídas ou variáveis internas. Para suprir tal lacuna, é necessário documentar, por meio de um cabeçalho, tais variáveis e o tipo de argumento associado.

O passo seguinte no processo de tradução, era escrever o algoritmo de funcionamento da macro em ST. Tal código em ST era traduzido pelo programador para IL seguindo uma série de regras, tais como as utilizadas por compiladores para gerar código assembler a partir de alguma linguagem de alto nível. Tais regras não serão apresentadas neste trabalho por existirem em grande número e serem muito específicas para o subconjunto de IL implementado pelo fabricante do PLC escolhido. Exemplos de regras de tradução de linguagens de alto nível para assembler podem ser encontradas em livros sobre compiladores, especialmente nos capítulos que versam sobre os geradores de código, por exemplo (FRASER et al,1995) e (WIRTH,1996). Ao término do processo de compilação, o código em ST original era deixado como comentário ao lado do código IL, facilitando a compreensão do código produzido.

No anexo A, apresentamos a macro que implementa o bloco de função ESCALONA presente na Fig. 5.1. Uma série de cuidados para evitar o *overflow* pela multiplicação de variáveis de 16 *bits* faz com que a macro tenha um código bem mais complexo do que poderia ter se a implementação de IL tratasse de maneira conveniente esta operação matemática.

5.4 A estratégia de controle utilizada no Sistema Médio Tom (MT)

A Fig. 5.3 apresenta o diagrama de blocos do controle do sistema MT.

A variável fundamental é a velocidade do motor de corrente contínua, controlada por meio da sua tensão de armadura. O fluxo de excitação (determinado pela corrente de campo) é mantido fixo no seu valor nominal. O sinal de realimentação de velocidade (WEST) foi estimado a partir das medidas de tensão e corrente de armadura (VAMT e IAMT).

Fig. 5.3 Diagrama de blocos de controle do sistema MT

O bloco PID2 recebe a referência de velocidade (WMT^*) dada pelo operador do sistema e a estimativa de velocidade (WEST), produzindo o sinal de referência da malha de controle da corrente de armadura ($IAMT^*$). A malha de corrente de armadura é formada pelo PID3 e seus periféricos. Se o erro de velocidade for positivo, o controlador de velocidade (PID2) solicita uma maior corrente de armadura para o PID3, fazendo acelerar o motor. Se o erro de velocidade for negativo, o mecanismo funciona de maneira inversa, diminuindo a corrente de armadura e a velocidade. A saída do PID3 produz a referência (VC1) do *chopper* de armadura.

A corrente de campo (IFMT) é controlada pelo PID1, fazendo com que permaneça em seu valor nominal mesmo que ocorram variações da tensão de alimentação ou na resistência de campo do motor devido ao seu aquecimento. A saída do PID1 (VC2) é a referência do *chopper* ligado ao enrolamento de campo do martelo.

5.5 Principais blocos de função usados

5.5.1 Blocos de geração de referência

As referências dos sistemas de varredura são funções periódicas, sendo que o sistema antigo era capaz de produzir por volta de dez tipos de funções diferentes. Entretanto, no novo sistema, todas essas funções periódicas foram implementadas pelo encadeamento de quatro funções mais simples que chamaremos de “primitivas”. Isto simplificou o trabalho de programar

e testar este bloco. No sistema magnético, existe a necessidade de senóides amortecidas usadas na sua desmagnetização e também ciclos de senóides usados na operação normal. No capítulo 8, comentaremos sobre uma primitiva adicional usada para identificar a função de transferência da planta a ser controlada.

A Fig. 5.4 mostra a referência usada na varredura do sistema MT.

Fig. 5.4 Perfil de velocidade utilizado na varredura acústica de médio tom

Existem velocidades mínima e máxima absolutas, ditadas pelas características do motor e do sistema mecânico do martelo acústico, sendo, portanto, chamadas velocidades mínima e máxima do sistema. A referência da varredura é composta por dois patamares constantes, dados pelos valores mínimos e máximos da varredura também chamados de velocidades mínima e máxima de varredura. Estes patamares de velocidade são interligados por rampas de velocidade, que têm o papel de simular a variação de frequência causada pelo efeito Doppler de um navio que se aproxima de uma mina. Por meio do sistema supervisor, o operador pode configurar a referência da varredura para simular algum tipo específico de navio.

O perfil de velocidade apresentado na Fig. 5.4 (excetuando-se o período de inicialização anterior ao instante T2) é descrito por um conjunto de três primitivas:

- rampa de WMIN até WMAX com duração (T3-T2);
- patamar constante de WMAX com duração (T4-T3);
- patamar constante de WMIN com duração (T5-T4).

O trabalho de codificação foi simplificado, pois o “patamar constante” é um caso particular de rampa no qual o valor final e inicial coincidem. Ou seja, o perfil de velocidade pode ser descrito por meio de 3 primitivas do tipo “rampa”. Através de um *flag* específico, indica-se ao PLC que a seqüência deve recomeçar ao seu término (o que ocorre no instante T5).

O período inicial, anterior ao instante T2, também pode ser descrito por meio das mesmas primitivas, executando-se uma rampa até WMIN, com duração (T1-T0). Como o *flag* de recomeço não está setado, o PLC permanece em WMIN até que ocorra uma ordem para o início do processo de varredura, tal como ocorreu em T2.

O sistema de geração de referências possui quatro instâncias, sendo uma para cada sistema acústico, uma para o sistema magnético e a quarta para comissionamento e depuração do sistema. Apesar de as necessidades de cada sistema de varredura serem diferentes, a macro que implementa o bloco de função foi construída de forma genérica.

5.5.2 Blocos de controle

O elemento central das malhas de controle são os blocos PID (Proporcional Integral Derivativo) fornecidos pelo próprio fabricante do PLC. O funcionamento deste bloco foi estudado criteriosamente para verificar seu funcionamento interno, tempo de execução e o mecanismo de *anti-windup* implementado (FERNANDES et al., 2001). Os ganhos derivativos eram sempre zerados, deixando ativos apenas os termos proporcional e integral. Tal escolha é freqüente em controle de motores industriais (BUXBAUM et al., 1990) apesar de existirem exceções.

Os blocos PIDs podem ser configurados para operarem em modo automático ou manual. No modo manual, o sinal de comando é estabelecido por uma variável externa enquanto, no modo automático, o sinal de comando é calculado a partir do sinal de erro observado e dos ganhos estabelecidos.

O modo manual do PID foi utilizado durante o chamado “teste do martelo invertido”. Os varredores dispõem de dois martelos acústicos com sistemas de conexão mecânica idênticos, o que torna possível conectar o martelo MT ao sistema de varredura BT (e vice-versa). Tal erro poderia danificar os enrolamentos dos motores. Para impedir tal erro, efetua-se o “teste do martelo invertido”, antes do início da varredura, aplicando-se uma pequena tensão ao enrolamento de campo e medindo a corrente que circula. Como os dois martelos têm resistências de campo muito diferentes, o PLC pode discriminar se o martelo conectado ao sistema de varredura é correto. Durante o “teste do martelo invertido” o PID1, que comanda o *chopper* do enrolamento de campo, é colocado em modo manual, aplicando-se uma tensão reduzida nesse enrolamento.

5.5.3 Blocos adicionais

O bloco REGCALC foi construído para supervisionar o funcionamento do bloco PID e verifica se:

- o sinal de erro fica acima de determinado patamar por um período excessivamente longo;
- o sinal de comando do PID permanece saturado por um período excessivamente longo.

Nos dois casos, a lógica de detecção de erro é notificada porque a malha monitorada perdeu a capacidade de regulação. Tal tipo de evento pode ser provocado, por exemplo, pela falha de um sinal de realimentação.

O bloco ESCALONA multiplica os sinais de realimentação por um ganho, criando um sistema PU (*per unit*) internamente ao PLC, o que facilita a análise de desempenho do sistema de varredura, pois o valor nominal sempre é representado internamente ao PLC pelo valor de 10000.

O bloco COMPENSA tem a função de eliminar *offsets* introduzidos nos sinais de realimentação devido ao cartão de transdutores. Este bloco tem dois modos de operação:

- Modo “Mede”. Na partida do sistema, antes de aplicar qualquer excitação à planta (*choppers* desligados), este bloco aquisa uma quantidade razoável de amostras do sinal de realimentação a ser compensado. A média aritmética desses valores é considerada o *offset* introduzido pelos operacionais do cartão de transdutores e é armazenada. Para indicar o término desta medição, o bloco seta o *bit* de saída “Pronto”. Existe um limite para o *offset* acima do qual o bloco emite um alarme para a lógica de detecção de erros.
- Modo “Compensa”. O bloco produz, na sua saída, o valor de entrada menos o *offset* obtido anteriormente.

Tipicamente, os sinais analógicos vindos do cartão de transdutores passam pelo bloco ESCALONA e pelo bloco COMPENSA antes de chegarem ao bloco de controle PID. Existe um bloco chamado SELESIN neste percurso (ver Fig. 5.1), que será discutido no Capítulo 8.

Uma série de outros blocos, como filtros passa-baixa usados na estimação de velocidade, também foram construídos usando macros em M4.

Alguns dos blocos de função relacionados ao controle necessitavam de taxa de amostragem constante. Entre estes, podemos citar o gerador de função e os filtros passa-baixa. Como o PLC escolhido não implementava o conceito do modelo de *software* da norma IEC61131-3 que permite determinar a taxa de execução de um módulo a única solução foi utilizar o mecanismo de interrupção disponível. Todos os blocos de função que necessitavam de taxa de amostragem constante foram incluídos numa única interrupção temporizada.

6 LÓGICA DE DETECÇÃO DE ERROS

*Mais vale a sabedoria do que os instrumentos de guerra,
mas um só erro pode anular muita coisa boa.
Eclesiastes, Capítulo 9, Versículo 18*

A lógica de detecção de erros adotada no projeto de modernização dos navios-varredores permite o processamento unificado de falhas e possui relação muito próxima com o seqüenciamento. Neste capítulo, será apresentada a razão de sua existência e sua implementação. Ao final, iremos comentar certas características da norma IEC61131-3 úteis na implementação da lógica de erros, caso estivessem disponíveis no PLC escolhido.

6.1 O motivo da lógica de detecção de erros

O seqüenciamento do sistema consiste numa série de etapas nas quais se prepara o sistema para varrer, executar a varredura ou processar sua parada. A execução dessas etapas depende de:

- condições de *hardware* (por exemplo, antes de energizar um *chopper*, devemos verificar se o disjuntor que o alimenta está fechado);
- condições de *software* (por exemplo, antes de energizar o sistema, o programa do PLC verifica se o *offset* dos sinais vindos do cartão de transdutores são nulos ou bastante baixos).

A ausência destas condições pode forçar o seqüenciamento a parar o sistema ou simplesmente gerar um aviso, dependendo da gravidade da ocorrência. A maioria dessas condições precisa ser verificada não apenas como pré-requisito para as etapas do seqüenciamento mas também durante a operação do sistema. O exemplo do disjuntor do *chopper* ilustra bem esta situação, pois deve permanecer ligado enquanto o *chopper* estiver funcionando.

Num seqüenciamento implementado em SFC, é muito simples fazer verificações de erro antes de se executar uma dada operação por meio da inclusão de condições nas transições do SFC que ligam suas etapas. Entretanto, implementar o mecanismo de verificação permanente de erros no SFC exige uma abordagem específica pois, a cada nova etapa, novas condições passam a ser relevantes e devem ser vigiadas.

Uma técnica inadequada para implementar tal mecanismo seria incluir, após cada passo, uma transição que verificaria todos os possíveis erros. As consequências seriam:

- existiria um número muito grande de transições usadas para detecção de erros;
- a equação lógica associada a cada uma dessas transições seria cada vez mais longa, pois cada estado traz mais eventos a serem verificados;

- um SFC desse tipo seria de depuração e manutenção muito difícil, pelo seu tamanho e complexidade aumentados;
- a lógica de operação ficaria misturada com a lógica de detecção de erros.

A solução encontrada para vigilância permanente de erros consistiu em dividir o seqüenciamento do sistema e a detecção de erros em dois módulos distintos de *software* que se comunicam e cooperam no sentido de obter o resultado desejado. O seqüenciamento verifica se existem condições de entrar num novo estado e, ao entrar neste novo estado, configura a lógica de detecção de erros para vigiar permanentemente essas condições daquele momento em diante.

A lógica de detecção de erros produz um “*bit* geral de erro” que é o resultado do OR lógico de todos os erros vigiados num dado momento. Na lógica de implementação do SFC o “*bit* geral de erro” é usado (implicitamente), tornando ativo o estado inicial seguro do SFC e desativando todos os demais. Esta “transição por erro” tem prioridade sobre as demais transições do SFC e não é representada no grafo que o descreve, sendo esta a razão de a termos qualificado de implícita. O diagrama SFC fica simplificado, assegurando o mecanismo de vigilância permanente.

6.2 A implementação da detecção de erros

A implementação da lógica de detecção de erros utilizada nos navios-varredores foi a reimplementação parcial em IL (*Instruction List*) de uma norma interna de programação estabelecida pela Petrobrás (1996). O objetivo desta norma interna é padronizar a codificação de programas escritos em *ladder* por empresas contratadas, e seus aspectos centrais são o tratamento das saídas e entradas digitais. O tratamento das entradas tem como objetivo a detecção de erros, indicando ao sistema a existência de estados lógicos não permitidos. Essa norma interna também descreve a interface entre o PLC e o sistema supervisor com relação à sinalização de erros para o operador e o seu reconhecimento.

A diferença essencial entre a implementação do navio-varredor em relação à norma da Petrobrás é a adesão incondicional desta ao *ladder* como linguagem de programação. O uso de SFC e a emulação de blocos de função feita com o uso do processador de macros M4 trouxe vantagens em relação ao trabalho original da Petrobrás, permitindo uma configuração dinâmica dos erros conforme o seqüenciamento evolui. O tratamento (habilitação) das saídas digitais indicado pela norma da Petrobrás não foi implementado nos navios-varredores. Realizar esta tarefa usando diretamente o SFC mostrou-se uma solução melhor.

Embora a detecção de erros lide basicamente com *bits*, a norma da Petrobrás exige o uso

de *bytes* ou *words* por razões de eficiência. No caso dos navios-varredores, foram utilizadas variáveis booleanas de 16 *bits* (e as instruções *bitwise* associadas). Tendo em vista o grande número de erros possíveis, tornou-se necessário agrupar estas *words* em tabelas.

Tabelas são aglomerados de *bits* contíguos na memória do PLC associados a cada um dos possíveis erros. No caso dos navios-varredores, as tabelas possuem 160 *bits* e estão ilustradas na Fig. 6.1.

	Bit 1: STDJ0MAG	Bit 2: STDJ1MAG	Bit 3: STDJ2MAG	Bit 4: STDJMT	Bit 5: STDJBT	...	...	Bit 158: PerTaco BT	Bit 159: NPrimMT	Bit 160: NPrimBT
Tabela de Entrada	0	1	1	1	1	...	...	1	1	0
Valores Esperados	0	0	0	1	0	...	...	0	0	1
Máscara de Temporização	1	1	0	1	0	...	...	1	0	0
Máscara de Bypass	1	0	1	1	1	...	...	1	1	1
Erros Bypassados e Selados	1	1	1	1	0	...	...	1	1	1

Fig. 6.1 Exemplos de tabelas de erros utilizadas nos navios-varredores

Os primeiros *bits* das tabelas estão associados ao estado do disjuntor #0 do sistema magnético (STDJ0MAG). A sequência continua com o estado dos demais disjuntores do sistema magnético (STDJ1MAG, STDJ2MAG) e dos disjuntores dos sistemas acústicos (STDJMT e STDJBT). Continuando numa ordem definida, a associação entre erros e *bits* segue até terminar com *bits* associados aos erros de perda de taco do sistema BT e número de primitivas errado para os sistemas acústicos. Embora cada tabela tenha uma função diferente, todas elas seguem esta mesma sequência na associação de seus *bits* a erros.

A lógica de detecção de erros será explicada usando-se diagramas de blocos. Cada linha do diagrama deve ser entendida como um fluxo onde toda uma tabela é produzida por uma operação booleana, sendo usada como entrada para outra operação booleana. A Fig. 6.2 dá uma visão geral das manipulações lógicas utilizadas.

Cada uma das tabelas apresentadas como entradas na Fig. 6.2 tem função bem específica:

- A “Tabela de entrada” contém os *bits* cujos estados devem ser vigiados. Os primeiros 48 *bits* desta tabela se referem às entradas digitais do sistema (erros de *hardware*) enquanto os restantes são calculados pelo *software* (tal como o *bit* de falta de excitação dos sistemas acústicos, produzido pela verificação do valor de corrente de campo medido contra um valor limite);

- A tabela “Valores Esperados” contém os valores lógicos que são esperados para cada um dos *bits* de entrada para que não ocorra indicação de erro;
- A tabela “Máscara de temporização” indica se o erro deve ser considerado imediatamente ou se pode perdurar temporariamente antes que o sistema seja parado;
- A tabela “Máscara de *bypass*” é usada para indicar que um erro deve ser vigiado ou desconsiderado (*bypassed*);
- A tabela “Erros *bypassados* e selados” (EBS) memoriza os erros até que sejam reconhecidos.
- A tabela de “Reconhecimento de erros” permite ao operador apagar, via supervisor, um erro memorizado na EBS.

Fig. 6.2 Lógica de detecção de erros

A convenção adotada nas tabelas de erros associa um *bit* setado à presença de um erro. Numa situação real de um sistema, particularmente nos *bits* produzidos pelo *hardware*, nem sempre os comportamentos são tão regulares assim. Um exemplo é o *status* dos contadores que devem estar desligados no início do seqüenciamento, ligados enquanto ocorre a varredura e, finalmente, desligados quando a varredura termina. Os *bits* de entrada passam por um processamento chamado polarização que uniformiza a lógica fazendo como que *bits* setados

sempre representem erros. Quando os *bits* da tabela de “Entradas” coincidirem com a tabela de “Valores esperados” não há erro.

Alguns erros devem ser considerados imediatamente enquanto outros podem ser temporizados. Erros de atuação imediata presentes na entrada do bloco de temporização são copiados imediatamente em sua saída. Erros indicados como temporizados só são copiados na saída quando persistem por um dado período. Cada erro está associado a um temporizador individualizado.

Erros eventuais ou intermitentes exigem um mecanismo de memorização para que possam ser objeto de manutenção/supervisão. O bloco “Selo de falhas” registra, na tabela “Erros Bypassados e Selados” (EBS), todos os erros ocorridos. Os erros são memorizados até que o motivo que os causou seja corrigido e que tenham sido reconhecidos pelo operador através do sistema supervisorio.

A tabela de reconhecimento é uma área de dados do PLC compartilhada com o supervisorio. O reconhecimento consiste em setar temporariamente o *bit* correspondente ao erro que se quer reconhecer. Isto fará com que a informação de erro selada na EBS seja apagada. A tabela EBS também é compartilhada com o supervisorio de modo que o operador possa saber quais foram os erros ocorridos e se o processo de reconhecimento executado foi bem-sucedido.

Existem cinco grafos SFC no *software* de seqüenciamento (por razões a serem explicadas no próximo capítulo) ordenados segundo uma hierarquia. Cada um desses SFCs tem o seu próprio “estado inicial seguro” e um “*bit* geral de erro” associado. Quando o “*bit* geral de erro” de um SFC fica verdadeiro, este é parado imediatamente e vai para seu “estado inicial seguro”. Quando ocorre um erro num dado SFC não apenas este deverá parar mas também seus SFCs subordinados. Por exemplo, se o “*bit* geral de erro” do SFC COM (comum) for setado, todos os outros quatro SFCs também devem parar. Se ocorrer uma falha no SFC ACU (acústico) apenas os SFCs AMT e ABT (acústicos de médio e baixo tom) devem parar.

Os *bits* relevantes a cada sistema são separados por meio de “Máscaras de erros” específicas a cada SFC. A implementação da hierarquia de SFCs é realizada fazendo com que os SFCs subordinados herdem o “*bit* geral de erro” do SFC de nível superior.

O tratamento dos avisos possui um tratamento análogo ao tratamento de erros. Neste caso os *bits* na EBS também são setados mas, como não estão presentes em nenhuma das “Máscaras de erro” o funcionamento do sistema não é interrompido, pois nenhum “*bit* geral de erro” é setado. Entretanto, o operador continua sendo informado que estes *bits* foram setados na EBS, tornando necessário o processo de reconhecimento.

O módulo de detecção de erros foi implementado em IL e consiste na tradução para forma textual do diagrama apresentado na Figs. 6.2.

6.3 Macros de configuração da lógica da detecção de erros

Os diversos SFCs de seqüenciamento configuram dinamicamente a lógica de detecção de erros na medida em que se sucedem os diferentes estados do sistema de varredura. Existem quatro tabelas cujos *bits* precisam ser manipulados para que se consiga alterar a configuração de erros:

- valores esperados - configura se um erro ocorrerá quando a entrada correspondente ao erro estiver setada ou zerada;
- máscara de *timer* - configura se um erro será temporizado ou de atuação imediata;
- máscara de *bypass* - determina se um erro será considerado ou não levado em conta;
- máscara de erros - determina a qual SFC o erro estará associado (caso não haja associação a nenhum SFC, então teremos um aviso/*warning*).

Foi criado um conjunto de macros escritas em M4 para simplificar o trabalho de codificação. O nome da macro indica como a configuração do erro irá ficar após a chamada da macro, facilitando a compreensão da mudança introduzida na detecção de erros.

Combinando os quatro *bits* das tabelas acima podemos ter nove configurações úteis. Para cada uma delas foi criada uma macro cujo nome e descrição será apresentada em seguida:

- E1 - erro não temporizado quando a entrada é setada;
- E0 - erro não temporizado quando a entrada é zerada;
- E1T - erro temporizado quando a entrada é setada;
- E0T - erro temporizado quando a entrada é zerada;
- W1 - aviso não temporizado quando a entrada é setada;
- W0 - aviso não temporizado quando a entrada é zerada;
- W1T - aviso temporizado quando a entrada é setada;
- W0T - aviso temporizado quando a entrada é zerada;
- X - erro é *bypassado*, ou seja, não é vigiado.

Na Tabela 6.1 indicamos quais os valores booleanos atribuídos aos *bits* das tabelas de configuração de erros para obter as nove possibilidades úteis existentes. Quando o valor de um *bit* não for relevante na configuração, assinalaremos na tabela o valor X (*don't care*).

Tabela 6.1 - Estado dos *bits* das tabelas para as diversas configurações úteis de detecção de erro

Configuração	Valor esperado	Máscara <i>timer</i>	Máscara <i>bypass</i>	Máscara erros
E1	0	0	0	1
E0	1	0	0	1
E1T	0	1	0	1
E0T	1	1	0	1
W1	0	0	0	0
W0	1	0	0	0
W1T	0	1	0	0
W0T	1	1	0	0
X	X	X	1	X

Os *bits* de valor esperado sempre estão invertidos em relação ao número que participa do nome da macro. Isto ocorre porque o número do nome da macro indica com que valor de entrada o erro ou aviso deve ocorrer. Assim, E0 significa que deve ocorrer erro se uma dada entrada estiver em 0. As atribuições feitas à tabela de valores esperados refletem a situação de normalidade do sistema.

Cada tabela usada na detecção de erros tem suas *words* declaradas como variáveis para que o processamento booleano *bitwise* descrito na seção anterior possa ser feito. Também cada um dos *bits* das tabelas é declarado como uma variável para que estes possam ser usados pelas macros de configuração de erros. Os nomes das variáveis (*bits*) declarados com esta função dividem-se em duas partes:

- um prefixo que indica o nome do erro ao qual o *bit* se refere, ou seja, a sua posição relativa dentro da tabela.
- uma terminação que indica a função da tabela a que este *bit* pertence.

Cada uma das macros deve receber um ou mais parâmetros. O parâmetro(s) deve(m) ser o nome do erro que se deseja configurar. Durante a expansão da macro, os nomes dos *bits* envolvidos na configuração são obtidos concatenando o nome do erro (parâmetro da macro) com os sufixos diferenciadores de tabela. Estes *bits* são atribuídos a zero lógico (desligado) ou um (1) lógico (ligado) conforme a macro usada de acordo com a Tabela 6.1.

Por exemplo, se desejarmos configurar como erro se o estado do disjuntor 0 do sistema magnético for zero (desligado) devemos chamar a macro da seguinte maneira:

```
E0(STDJ0MAG)
```

Após o processamento por M4 a macro produz o seguinte código IL:

```
***** Inicio do bloco E0(STDJ0MAG)
LD_BOOL LIGADO
ST_BOOL STDJ0MAG_Esp
ST_BOOL STDJ0MAG_MascErro
LD_BOOL DESLIGADO
ST_BOOL STDJ0MAG_Byp
ST_BOOL STDJ0MAG_MascTimer
***** Fim do bloco E0(STDJ0MAG)
```

A primeira e última linhas da listagem acima contêm comentários que indicam o início e final do código gerado pelo M4 bem como os parâmetros originais da macro. O código usado na depuração do sistema é o produzido após a expansão das macros por M4 enquanto o código escrito pelo programador continha a chamada de tais macros. Os comentários iniciais e finais têm a função de ajudar o depurador a se localizar.

Se existissem mais erros que desejássemos configurar com E0 poderíamos incluir seu prefixos com vírgulas separando-os. Por exemplo, se desejarmos configurar o *status* do disjuntor 0 do sistema magnético e também dos disjuntores 1 e 2 iríamos chamar a macro do modo abaixo:

```
E0(STDJ0MAG,STDJ1MAG,STDJ2MAG)
```

que produziria o seguinte código IL:

```
***** Inicio do bloco E0(STDJ0MAG,STDJ1MAG,STDJ2MAG)
LD_BOOL LIGADO
ST_BOOL STDJ0MAG_Esp
ST_BOOL STDJ1MAG_Esp
ST_BOOL STDJ2MAG_Esp
ST_BOOL STDJ0MAG_MascErro
ST_BOOL STDJ1MAG_MascErro
ST_BOOL STDJ2MAG_MascErro
LD_BOOL DESLIGADO
ST_BOOL STDJ0MAG_Byp
ST_BOOL STDJ1MAG_Byp
ST_BOOL STDJ2MAG_Byp
ST_BOOL STDJ0MAG_MascTimer
ST_BOOL STDJ1MAG_MascTimer
ST_BOOL STDJ2MAG_MascTimer
***** Fim do bloco E0(STDJ0MAG,STDJ1MAG,STDJ2MAG)
```

6.4 A detecção de erros e a norma IEC61131-3

A declaração das variáveis usadas na detecção de erros foi uma tarefa difícil, pois:

- todas as tabelas devem seguir uma única seqüência de *bits* de erros, o que as torna muito sensíveis a declarações de *bits* em posições de memória erradas;

- a necessidade de acesso às tabelas tanto como *words* (quando se executa a lógica de detecção usando operadores *bitwise*) quanto como *bits* (quando se executam as configurações de cada erro) fez com que as declarações de variáveis implicassem em um processo trabalhoso.

Essas dificuldades têm um potencial de geração de erros muito grande, principalmente em um ambiente de programação como o usado no projeto em que não existia nenhum apoio à tarefa de alocar variáveis e o programador deve assumir toda a responsabilidade da definição de endereços. Para diminuir este potencial de erros, foi criada uma planilha eletrônica com uma série de fórmulas que auxiliaram no trabalho de definir o correto endereço dos *bits* nas tabelas.

Se a norma IEC61131-3 estivesse inteiramente implementada no PLC usado, parte desta dificuldade não existiria pois ela prevê meios para que o próprio compilador da linguagem do PLC defina o endereço das variáveis. Entretanto, nem mesmo uma conformidade completa da interface de programação à norma resolveria todos os problemas. Para tornar mais racional a declaração das tabelas ora como *bits* ora como *words* o ideal seria termos na linguagem de programação do PLC uma característica semelhante a declaração de *unions* na linguagem C. Esta característica permite a declaração de estruturas de dados diferentes sobre um mesmo trecho de memória. Desse modo, o conteúdo desta memória pode ser acessado num dado momento como tendo um determinado tipo (*word*, por exemplo) e, em outro momento, como tendo outro tipo (*bit*, por exemplo), de acordo com a conveniência do programador. A IEC61131-3 não permite tal arranjo, pois ele pode quebrar a consistência dos tipos de dados.

Em relação ao código, o fato de o PLC escolhido não permitir a declaração de funções ou blocos de funções com parâmetros traria problemas para a estruturação do programa, não fosse o uso do processador de macros M4. Esta ferramenta permitiu emular a criação de funções e blocos de funções com parâmetros, o que ofereceu condições de produzir um código de melhor qualidade. Entretanto, o ideal seria que o PLC implementasse o modelo de *software* apresentado pela IEC 61131-3 pois:

- a) As funções não seriam expandidas *inline* como o processador M4 fez, o que implicaria em grande economia de memória. Felizmente o PLC usado no projeto da modernização dos varredores tinha bastante memória mas, em projetos onde este recurso é mais restrito, o uso de M4 seria inviável.
- b) Poderíamos usar funções com parâmetros e variáveis com declarações de escopo mais apropriadas e não todas globais como fizemos. Como o PLC usado também não permitia o uso de variáveis locais, todos os parâmetros das macros M4 consistiam em variáveis globais, abrindo margem para erros de codificação de difícil depuração.

- c) A taxa de execução da detecção de erros poderia ser escolhida de modo mais apropriado. O PLC escolhido permitia que o código fosse executado no programa principal (todo ele numa mesma taxa) ou em interrupção temporizada (com outra taxa de execução). Na interrupção foi incluído um grande volume de código de controle que necessitava de taxa de amostragem constante. Assim, a alternativa que restou foi rodar a detecção de erros na taxa do programa principal.

O uso da detecção de erros (originalmente voltada ao *ladder*) em conjunto com uma das linguagens da norma (SFC) trouxe as seguintes vantagens:

- diagramas SFC mais leves cuja responsabilidade é apenas cuidar do seqüenciamento, ficando a detecção de erros separada em outro módulo;
- configuração dinâmica da detecção de erros conforme o seqüenciamento evolui;
- retorno automático a um estado seguro inicial quando da ocorrência de erro;
- adequação à operação hierárquica dos diagramas SFCs que formam o sistema.

7 SEQUENCIAMENTO

*Debaixo do céu há momento para tudo, e tempo certo para cada coisa:
Tempo para nascer e tempo para morrer.
Tempo para plantar e tempo para arrancar a planta.
Tempo para matar e tempo para curar.
Tempo para destruir e tempo para construir.
Eclesiastes, Capítulo 3, Versículos 1 a 3*

O sequenciamento do sistema tem três funções:

- iniciar a varredura de modo ordenado;
- parar o sistema quando em erro e levá-lo rapidamente ao estado seguro;
- desligar o sistema ordenadamente quando não mais se desejar varrer.

A implementação do sequenciamento foi feita usando-se a linguagem SFC (*Sequential Function Chart*) pelas razões apresentadas na Seção 3.2.2. Como esta linguagem não estava disponível para o PLC escolhido foi desenvolvida uma ferramenta capaz de traduzir versões textuais dos SFCs para IL.

Neste capítulo apresenta-se uma visão geral do sequenciamento e, como exemplo, detalha-se o funcionamento do SFC do sistema acústico médio tom (AMT). Descreve-se também a implementação e uso do tradutor de SFC para IL.

7.1 A tradução de SFC para IL

O ponto de partida para o tradutor de SFC para IL é um arquivo texto ASCII com a descrição textual da máquina de estados que deseja implementar. Este arquivo contém um misto de duas linguagens:

- uma linguagem para descrever a topologia do grafo SFC por meio da qual são indicados os estados iniciais e os estados regulares, além da sucessão dos estados e transições;
- as condições das transições e o código referente às ações a serem executadas por cada estado escritas diretamente em IL.

O código IL resultante pode ser utilizado diretamente na interface de programação do PLC, restando a tarefa de declarar as variáveis utilizadas pelo código em IL gerado.

O tradutor teve como inspiração a linguagem textual para descrição de SFCs da norma IEC61131-3 descrita em seu anexo B.1.6. Elementos vindos da IEC848 também foram usados. Essas idéias vindas de duas fontes diferentes, somadas à necessidade de construir um compilador simples, resultaram em uma linguagem não conforme com nenhuma das duas normas.

Neste tradutor foram implementadas apenas as características estritamente necessárias ao trabalho. O programa, em momento nenhum, pode ser considerado um produto acabado e independente do operador. A idéia foi produzir uma ferramenta que tão somente tirasse do programador o trabalho repetitivo de escrever o gerenciamento em lógica booleana associado à evolução dos grafos SFC.

O programa de tradução não se encarrega das seguintes tarefas (ficando elas a cargo do programador) :

- Interface com qualquer tipo de editor gráfico de SFC. O programador deve desenhar o SFC e traduzi-lo por conta própria para a versão textual processada pelo tradutor.
- Verificação da coerência da topologia do SFC descrito. Por exemplo, se o programador "pular" a conexão de algum estado e o deixar solto, ninguém o avisará disto. Este tipo de erro pode trazer como sintomas a geração de um erro quando a interface de programação do PLC compilar o programa em IL gerado ou um *bug* que só irá se manifestar durante a depuração do programa.
- Verificação sintática ou semântica dos trechos de programa IL. Os trechos em IL são passados para interface de programação sem qualquer verificação. Se algum erro de sintaxe do IL existir no arquivo de descrição do SFC então quem irá produzir uma mensagem de erro será a interface de programação.
- Gerenciamento da existência, tipo e localização, na memória do PLC, das variáveis do programa IL gerado.

O uso desta ferramenta de tradução, apesar de suas restrições, traz um balanço positivo de trabalho principalmente quando se utiliza este tradutor em projetos de maior porte.

O anexo B apresenta, de modo informal, a sintaxe da linguagem de descrição textual de SFC. Exemplos curtos e simples mostrando os principais elementos e características desta linguagem e sua interface com a linguagem IL também serão mostrados neste anexo.

7.2 Visão geral do seqüenciamento

Para evitar que a verificação de erros feita no seqüenciamento se tornasse ainda mais complexa foi criado um conjunto de verificações de coerência em relação aos comandos que o operador pode executar. O módulo de interface com o supervisor recebe os comandos e verifica a sua coerência: se o comando for coerente ele é enviado ao seqüenciamento que o executa; se for incoerente é produzido um aviso por meio da lógica de detecção de erros.

Um exemplo da interação entre interface com o supervisor e seqüenciamento são os comandos de indisponibilização do sistema. Quando se quer fazer a manutenção de um sistema, este deve ser colocado na condição de indisponível. Nesta condição os comandos de início de operação deste sistema não serão mais aceitos. Um comando de indisponibilização só pode ser aceito se o sistema estiver fora de operação, ou seja, o operador deve emitir o comando de fim de varredura antes de enviar o comando de indisponibilização. Todas essas restrições tornam o seqüenciamento mais simples, pois ele não tem que se preocupar com a verificação de comandos pouco sensatos que podem chegar a qualquer momento, mas que são barrados pela interface com o supervisor. As restrições usadas para verificação de coerência de comandos são um conjunto de equações booleanas que levam em consideração o estado em que o seqüenciamento se encontra. A interface com o supervisor executa apenas operações lógicas e este módulo foi implementado em *ladder*, que é uma linguagem (neste contexto) muito apropriada para este fim.

De uma maneira geral o seqüenciamento se relaciona e gerencia todas as outras partes do sistema:

- comanda o ligamento e desligamento de uma série de dispositivos de hardware;
- configura a lógica da detecção de erros para que, individualmente, estes sejam monitorados (ou não) conforme o sistema evolui;
- conduz o sistema para um estado seguro quando solicitado pela lógica de detecção de erros;
- recebe comandos validados (coerentes) do sistema supervisor e toma as providências para que estes sejam executados;
- habilita as malhas de controle a operarem quando o sistema está em condições de iniciar uma varredura.

Talvez até fosse possível construir todo o seqüenciamento usando uma única e grande máquina de estado, mas o trabalho para sua construção, depuração e manutenção seria imenso. Para gerenciar de modo mais racional o grande número de detalhes que o sistema possui foram construídos diversos diagramas SFC com um ordenamento hierárquico entre eles.

Grafos SFCs com uma disciplina hierárquica são uma solução para o compartilhamento de algumas entradas digitais por mais de um sistema. Por exemplo, se o operador acionar algum botão de emergência, então todos os sistemas que estiverem em operação devem parar. Não seria racional (e também de difícil operação) existir um botão de emergência para cada sistema. Quando a detecção de erros avisa que uma emergência ocorreu, os SFCs de todos os sistemas devem ir ao estado seguro.

A Fig. 7.1 mostra as relações de hierarquia entre os diversos SFCs. O sistema magnético

não foi desdobrado em dois (um para o HFG e outro para Cauda) pois apenas um dispositivo de varredura magnético pode ser usado de cada vez.

O estado do botão de emergência é usado no cálculo do “*bit* geral de erro” do SFC Com. Como Com é o mais alto em termos de hierarquia, seu “*bit* de erro” é usado para o cálculo de erro de todos os outros, ou seja a parada por emergência faz todos os SFCs do sistema irem para o estado seguro. No caso específico do botão de emergência o sistema também é parado por *hardware* para não depender do bom funcionamento do PLC para lidar com questões de segurança.

Fig. 7.1 Hierarquia dos SFCs

Podemos classificar os grafos SFC em dois grandes grupos:

- COM e ACU são superiores hierárquicos, que possuem um pequeno número de estados, pois apenas vigiam a ocorrência de alguns erros comuns e coordenam a operação de seus subordinados;
- MAG, AMT e ABT são grafos subordinados que estão fazendo o trabalho efetivo de seqüenciamento e que possuem, cada um, cerca de duas dezenas de estados.

Apesar do conteúdo e função de cada estado ser diferente, a estrutura da maior parte deles é semelhante, sendo constituída por:

- ações a serem executadas tais como ligar e desligar elementos do *hardware*, manipular dados na memória do PLC, etc;
- reconfiguração da lógica de detecção de erros em função das ações tomadas, sendo freqüente configurar a verificação de erros tendo em vista uma ação que só será executada no próximo estado;
- “macros M4” utilizadas na depuração do sistema, descritas no Capítulo 8.

Para exemplificar, apresenta-se a seguir, um trecho de SFC. (Os números, em sua margem esquerda, foram colocados *a posteriori*). O anexo B apresenta a linguagem de descrição textual de SFCs e a sua leitura irá facilitar o entendimento do trecho abaixo.

```

630 %step Mag_IS_X9 forceoff Mag_IS_GeralErro
632
633 " Configura blocos de compensacao para compensar OffSet
634 LD_BOOL DESLIGADO
635 ST_BOOL Mag_IS_Icauda_Mede
636 ST_BOOL Mag_IS_IcaudAux_Mede
637 ST_BOOL Mag_IS_Vcauda_Mede
638 ST_BOOL Mag_IS_IFGerVar_Mede
639 ST_BOOL Mag_IS_VExc_Mede
640 ST_BOOL Mag_IS_IHfg_Mede
641 ST_BOOL Mag_IS_IFHfg_Mede
642 ST_BOOL Mag_IS_VHfg_Mede
643 ST_BOOL Mag_IS_IFGerVar_Mede
644 ST_BOOL Mag_IS_VExc_Mede
647 " Liga C0MAG
648 LD_BOOL LIGADO
649 ST_BOOL C0MAG
650
651 " Bypassa erro de STC0Mag, MagReady e Offsets excessivos
652 X(STC0Mag,MagReady,Mag_IS_OfsICauda,Mag_IS_OfsICaudAux
    Mag_IS_OfsIHfg,Mag_IS_OfsVCauda,Mag_IS_OfsVHfg,Mag_IS_OfsIFHfg,
653   Mag_IS_OfsIFGerVar,Mag_IS_OfsVexc)
657
658 MONITESTADO(Mag)
659
660 %endstep
661
667 %delay Mag_VA_d4 from Mag_IS_X9 Mag_PC_TFechaC0Mag tenths to Mag_IS_Aux4
669
670 %transition Mag_IS_Y11 from Mag_IS_X9
671 LD_BOOL Mag_IS_Aux4
672 OR STC0MAG
673 PASSOAPASSO
674 %to Mag_IS_X10
676
677 %step Mag_IS_X10 forceoff Mag_IS_GeralErro
678 " Configura STC0MAG como erro em 0
679 E0(STC0MAG);
680
681 MONITESTADO(Mag)
682
683 %endstep

```

Iremos comentar a constituição dos estados exibidos dividindo-os nas partes a, b e c anteriormente definidas.

Estado X9:

- a) linhas 633-649: os blocos COMPENSA do sistema magnético são configurados para não medir mais o *offset*. O contator que energiza o *chopper* magnético é ligado.
- b) linhas 651-653: todos os erros de *offset* excessivo do sistema magnético são bypassados,

bem como o erro de *chopper* magnético pronto e do estado do contator que energiza o *chopper* magnético (STC0MAG).

- c) linha 658 : chamada de macro M4 para depuração do sistema.

Estado X10:

- a) inexistente neste estado.
- b) linhas 678-679 : configura a detecção de erros para vigiar se o contator que energiza o *chopper* magnético se abriu.
- C) linha 681 : chamada de macro M4 para depuração do sistema.

Na linha 667, foi incluída uma temporização para esperar o fechamento do contator C0MAG. A transição entre os estados X9 e X10 (linhas 670 a 674) espera o *status* deste contator (STC0MAG) indicar seu fechamento ou o fim desta temporização, o evento que ocorrer primeiro torna X10 ativo.

No estado X10, a detecção de erros é configurada para verificar se o contator C0MAG está aberto. Caso este estado tenha se tornado ativo porque o *status* deste contator indica que ele está fechado, então a evolução do SFC prossegue normalmente. Entretanto, se X10 tornou-se ativo devido ao fim da temporização de espera presente na linha 667, então a lógica de erros irá detectar o não-fechamento do contator C0MAG. Esse conjunto de atividades, ou seja, comando do *hardware*, espera de sua resposta e configuração de erro, foi usado de modo freqüente no seqüenciamento. Tal arranjo foi chamado de *timeout*. No exemplo anterior foi descrito o *timeout* do fechamento do contator C0MAG.

A macro PASSOAPASSO na linha 673 será explicada no Capítulo 8.

7.3 O funcionamento de um grafo SFC superior

Um exemplo das ações típicas tomadas por um SFC superior é apresentado na Fig. 7.2.

Fig. 7.2 Exemplo de SFC superior

7.4 O diagrama SFC do sistema acústico MT

Nesta seção, serão detalhadas as ações dos estados e macro-estados apresentados na Fig. 4.5.

No estado **X1** o sistema é colocado num estado seguro com as seguintes ações:

- O contator pré-*chopper* (C1MT) é desligado e os dois canais do *chopper* MT são desabilitados.
- Os PIDs são todos desabilitados e postos em modo manual.
- O gerador de função é desabilitado e os seus *bits* de configuração são desligados.
- Os *bits* de Mede que configuram os blocos COMPENSA são desligados pondo, assim, estes no modo de operação compensa.
- Os *bits* de erro produzidos pelos blocos COMPENSA são zerados.
- Os *presets* (comando de saída no modo manual) e as saídas dos PIDs são zerados.
- É feita a parada automática da aquisição contínua de dados.
- Quanto à configuração da lógica de detecção de erros, todos os erros do sistema são *bypassados*.

O desdobramento do macro-estado **M1** da Fig. 4.5 está dado na Fig. 7.3.

Fig. 7.3 Macro-estado de Espera de Comandos (M1)

O macro-estado **M2** da Fig. 4.5 é detalhado nas Figs. 7.4 e 7.5

Fig. 7.4 Macro-estado de Preparação de Varredura (M2) (continua)

Fig. 7.5 Macro-estado de Preparação de Varredura (M2) (conclusão)

X15: São copiados os parâmetros de configuração (definidos pelo operador) para a área de primitivas do gerador de referência preparando-o para produzir o perfil de velocidade trapezoidal típico da varredura. Neste estado, também se executa a macro que verifica se a hora correspondente ao início da varredura já chegou.

X16: A varredura é realizada neste estado, habilitando-se o gerador de referência.

O desdobramento do macro-estado **M3** da Fig. 4.5 está mostrado na Fig. 7.6.

Fig. 7.6 Macro-estado de Diminuição de Referência (M3)

O desdobramento do macro-estado M4 da Fig. 4.5 está na Fig. 7.7.

Fig. 7.7 Macro-estado de Desmonte da Varredura (M4)

8. DEPURAÇÃO E TESTE

*Se seu irmão pecar, vá e mostre o erro dele,
mas em particular, só entre vocês dois.
Mateus, Capítulo 18, Versículo 15*

Mesmo o *software* mais estruturado e planejado apresenta erros ao final das etapas iniciais de codificação, tornando essenciais as ferramentas de depuração. As ferramentas de apoio disponibilizadas pelo fabricante do PLC utilizado eram muito limitadas. Este capítulo irá mostrar algumas ferramentas de depuração desenvolvidas durante o projeto de modernização dos navios-varredores.

8.1 Depuração do seqüenciamento

A possibilidade de executar um programa ‘passo a passo’ é de grande valia, permitindo visualizar a dinâmica de um programa de seqüenciamento. A macro M4 chamada PASSOAPASSO foi criada com tal objetivo. As transições do SFC foram implementadas através de um código em IL que testa uma ou mais condições deixando na pilha uma variável booleana como resultado. A evolução do SFC para o próximo estado depende do valor desta variável. A macro PASSOAPASSO faz o AND lógico desta variável com um sinal proveniente de um botão ligado a uma entrada digital do PLC. Ou seja, mesmo que as condições do seqüenciamento estejam satisfeitas o sistema só irá evoluir quando o ‘botão de passo’ for acionado.

A macro MONITESTADO armazena o último estado executado por um dado SFC antes de ocorrer um erro (ou seja, antes do seqüenciamento deste sistema ser conduzido ao estado X1). Cada estado pertencente a um SFC tem uma variável booleana associada para indicar se tal estado está ativo ou inativo. Tais variáveis estão agrupadas em *words* na memória do PLC. A macro MONITESTADO simplesmente copia estas *words* em ‘variáveis-espelho’ independentes ao final de cada estado do seqüenciamento, exceto no estado X1. Quando ocorre um erro, a ‘variável-espelho’ irá conter o estado do SFC antes da ocorrência do erro. Tal macro foi fundamental para a depuração do seqüenciamento e da lógica de erros.

Embora úteis durante o processo de depuração, as macros descritas acima tornam o programa maior e mais lento. Usando a característica de inclusão condicional fornecida por M4 estas macros podem ter seu código eliminado em função do estado da variável de compilação DEPURA. Ao término do processo de depuração, tal variável é zerada antes de gerar a versão final do programa. Desta maneira tornou-se possível ter um único ‘programa fonte’, utilizado tanto para depuração quanto para gerar a versão final.

Exemplos de chamada das macros PASSOAPASSO e MONITESTADO pode ser vistos no código apresentado na Seção 7.2.

8.2 Depuração e monitoração do controle

Os blocos de função GERENMALHA e SELESIN têm a função de introduzir sinais de teste em pontos chave do diagrama analógico. SELESIN é um seletor de sinal controlado por dois bits. A partir do estado destes bits este bloco pode ter três comportamentos:

- o sinal de entrada do bloco é copiado na sua saída. Trata-se da operação *default* do bloco, ou seja, seu comportamento quando não está ocorrendo depuração.
- o sinal da instância do gerador de função destinada a testes é copiado na saída do bloco.
- o sinal do gerador de função somado ao sinal de entrada do bloco é copiado na saída do bloco.

Blocos SELESIN foram colocados no fluxo de processamento dos sinais de referências e realimentação de todas as malhas e antes dos cartões de saídas analógicas que produzem a referência dos *choppers*, tal como exemplificado na Fig. 5.1. Usando este bloco é possível inserir sinais de teste em diversos pontos do sistema de controle. Grande parte do comissionamento consistiu em ajustar os ganhos dos blocos PID de cada malha individualmente, tarefa que foi simplificada pela existência dos blocos SELESIN. Outro uso destes blocos é a caracterização do comportamento dinâmico da planta ou a introdução de perturbações nas realimentações para verificar a estabilidade e a imunidade à ruído dos controladores.

Para evitar erros na operação dos blocos SELESIN (tal como malhas sem realimentação) foi desenvolvido o bloco GERENMALHA que opera como um decodificador digital. A saída do bloco GERENMALHA gerencia os bits de configuração de todos os blocos SELESIN de um dado sub-sistema de uma maneira coordenada e segura. O número na entrada de GERENMALHA indica qual dos blocos SELESIN será usado e o seu modo de operação. Na inicialização do sistema, realizada quando o PLC é ligado, o bloco GERENMALHA entra no modo *default*, liberando o funcionamento normal de todos os blocos SELESIN.

Para o levantamento do comportamento dinâmico da planta foi incluída entre as primitivas do gerador de função um sinal do tipo PRBS (*Pseudo Random Binary Sequence*) (LJUNG, 1999). O uso deste tipo de sinal permitiu aprimorar os modelos matemáticos utilizados para a simulação e o ajuste das malhas de controle.

Foi desenvolvido um sistema de aquisição de dados interno ao próprio sistema, reservando-se para este fim uma área de memória do PLC. Esta área de memória permite

armazenar até 16 variáveis analógicas e existem essencialmente dois modos de operação:

- modo ‘varredura única’ (*single sweep*) onde os dados são armazenados até preencher completamente a área de memória do PLC;
- modo repetitivo, quando a área de memória funciona como *buffer* circular sempre que existe algum sub-sistema executando uma varredura.

O modo ‘varredura única’ mostra-se particularmente útil no processo de comissionamento das malhas de controle.

O modo repetitivo cria um mecanismo do tipo ‘caixa-preta’, armazenando todos os eventos anteriores a uma falha. Para tal finalidade foram armazenadas informações adicionais tais como data e hora da falha, os últimos comandos emitidos pelo operador e os últimos estados executados pelo seqüenciamento.

Os dados ficam armazenados no PLC e podem ser lidos tanto pelo sistema supervisão quanto por um *notebook* e são utilizados para fins de comissionamento ou manutenção. Um resultado destas ferramentas é a possibilidade de fazer um diagnóstico de falhas remoto, bastando enviar os arquivos coletados para técnicos experientes.

A existência deste sistema de armazenamento interno substituiu em parte o uso de osciloscópios durante o processo de modernização dos navios-varredores, oferecendo o registro simultâneo de um grande número de canais. Em alguns casos específicos, utilizou-se uma das saídas analógicas do PLC para observar através de um osciloscópio, variáveis escolhidas pelo operador.

8.3 Teste

O teste e a validação do software desenvolvido eram de fundamental importância por se tratar de um sistema de uso militar. Com este objetivo, utilizando um computador PC equipado com placas de entradas/saídas digitais/analógicas, foi construído um sistema capaz de simular a planta e o hardware presentes no navio-varredor (BARROS, 2002). Ao se conectar as entradas do PLC às saídas do sistema de simulação e vice-versa era possível executar uma série de testes relativos tanto ao controle do sistema como ao seu seqüenciamento.

A implementação do seqüenciamento numa linguagem (SFC) onde existe uma definição muito clara dos estados do sistema e qual sua evolução temporal facilitou bastante o teste. Um conjunto de planilhas descrevendo as possíveis seqüências de estados bem como quais erros deveriam ser vigiados pela lógica de erros foram construídas e o sistema foi testado contra elas

em busca de alguma discrepância. Estes testes foram feitos de modo automático. Usando uma versão modificada do simulador capaz de atuar o ‘botão de passo’ e criar de satisfazer as transições do seqüenciamento, o programa do PLC era levado até um determinado estado. Eram simulados erros na planta para verificar se realmente o PLC conduzia o sistema novamente ao estado inicial seguro. Eventuais diferenças de comportamento em relação à planilha de teste detectadas pelo sistema simulação eram verificadas novamente por um operador do sistema de modo a afastar a possibilidade de um erro no sistema simulação causar a detecção de um falso erro no programa do PLC. Não se trata de um sistema infalível, mas sem dúvida teve valor no sentido de auxiliar os responsáveis pelo teste e validação a encontrar um grande número de pequenos erros de codificação no programa do PLC.

O reuso de código, inspirado no conceito de bloco de função (*function block*) da norma IEC61131-3 e implementado usando o macro processador M4, também foi proveitoso para o teste e validação. Este conceito permitiu a definição de interfaces claras e que cada macro fosse isoladamente testada.

De Smet *et al.* citado por Frey (2002) indica que o teste formal de programas escritos em *ladder* só é viável quando estes são de pequeno ou médio porte. Hamilton (1993) indica que as características da norma IEC61131-3 que contribuem para a obtenção de programas de PLC mais estruturados e consistentes também são benéficas para tornar o processo de testes e validação mais fácil.

9 RESULTADOS

*Não existe árvore boa que dê frutos ruins,
nem árvore ruim que dê frutos bons;
porque toda árvore é conhecida pelos seus frutos.
Lucas, Capítulo 6, Versículos 43 e 44*

Nesta capítulo, serão exibidos e comentados registros referentes ao funcionamento do sistema de varredura obtidos com o mecanismo de aquisição descrito no Capítulo 8. Grande parte das curvas apresentadas foram produzidas pelo sistema de varredura MT. Por não se exporem detalhes sobre a constituição e funcionamento do sistema magnético, mostrar-se-á apenas um gráfico para ilustrar o funcionamento da malha de controle mais externa deste sistema. Resultados da aplicação do sinal PRBS (*Pseudo Random Binary Sequence*) (LJUNG, 1999) sobre a planta do sistema magnético também serão exibidos para ilustrar a implementação desta ferramenta de depuração.

9.1 O sistema MT em funcionamento

As variáveis apresentadas nos gráficos que constam nesta seção têm siglas que seguem a convenção apresentada no diagrama de blocos apresentado na Fig. 5.3. Esta figura também é apoio importante aos comentários aqui feitos sobre aspectos de controle do sistema.

A Fig 9.1 apresenta a referência e a realimentação da velocidade (a principal variável controlada pelo sistema MT) durante as etapas de preparação, varredura e desmonte.

Durante a preparação, temos uma rampa desde a velocidade zero até a velocidade mínima do sistema (440 rpm). Ao fim do período de preparação, a velocidade salta para a velocidade mínima de varredura (1100 rpm). Este comportamento não está de acordo com o sequenciamento apresentado na Fig. 7.5 da Seção 7.4, segundo o qual o sistema deveria passar pelo estado X14a e X14b que executam a uma outra rampa iniciada na velocidade mínima do sistema e terminada na velocidade mínima de varredura. A razão é que este registro foi feito numa versão do programa anterior à implementação dos estados X14a e X14b, mostrando precisamente a necessidade desses estados intermediários.

Durante a varredura, a velocidade varia segundo um perfil de velocidade trapezoidal, típico dos sistemas de varredura acústicos. A função deste perfil de velocidade é produzir ruídos semelhantes ao que seria causado pela aproximação entre o navio-alvo simulado pelo martelo e a mina a ser varrida.

No desmonte, temos uma rampa que leva a velocidade do ponto onde a varredura foi interrompida (aproximadamente 1475 rpm) até a velocidade mínima do sistema (440 rpm) seguida por uma outra que leva o sistema até o repouso.

Fig. 9.1 Comportamento da velocidade do martelo MT durante ciclo completo de operação

9.1.1 A preparação do sistema MT

O principais eventos ocorridos durante a preparação do sistema são o teste do martelo invertido e a rampa que leva o sistema do repouso até a velocidade de varredura.

A Fig. 9.2 mostra as variáveis referentes ao controle da corrente de campo durante a partida. Os sinais mostrados são a referência e a realimentação de corrente de campo. A referência de corrente de campo permanece constante no valor nominal (0.22 A) durante todo o intervalo, pois o controle de velocidade do motor DC é feito variando-se a tensão de armadura e mantendo-se o campo fixo. Apesar disso, o PID1 (responsável pelo controle da corrente de campo) é habilitado apenas no estado X12, sendo normal que, antes desse momento, a realimentação de velocidade esteja distante do valor de referência. Nos instantes anteriores à X10 (quando irá ocorrer o teste do martelo), o PID1 está desabilitado anulando o sinal de saída deste bloco bem como sua realimentação. Nos estados X10 e X11, o PID1 é habilitado em modo manual com um pequeno valor de *preset*, o que faz o *chopper* aplicar uma tensão reduzida ao enrolamento de campo do motor. A corrente de campo produzida é medida pelo controle, o que permite que o

PLC possa detectar se o martelo BT foi ligado na tomada de saída do sistema MT. Na situação apresentada na Fig. 9.2, não ocorreu erro de “martelo invertido” e o seqüenciamento prossegue normalmente.

Fig. 9.2 Variáveis referentes ao controle de corrente de campo durante a partida

O PID1 é posto em modo automático no estado X12. Como consequência, a corrente de campo sobe até o valor nominal. Após este transitório, referência e realimentação permanecem praticamente iguais até o final da etapa de desmonte da varredura.

A Fig. 9.3 exhibe o comportamento da referência e realimentação de velocidade durante a partida. No estado X13, que dura um único *scan*, a malha de corrente de armadura (PID2) é colocada em modo automático. No estado X14, são habilitados o PID2 (controle de velocidade) e o gerador de referência MT. O gerador de referência havia sido previamente configurado para executar uma rampa desde a velocidade nula até a velocidade mínima absoluta de varredura (440 rpm). O estado X14 fica ativo enquanto esta rampa é executada. Após X14, o sistema evolui para o estado X15, onde deveria esperar a hora de início de varredura. Entretanto, na ocasião em que foi feita a aquisição desses dados, a hora de início da varredura foi configurada de tal maneira que praticamente não houve espera neste estado.

Fig. 9.3 Variáveis referentes ao controle de velocidade durante a partida

Fig. 9.4 Variáveis referentes ao controle de corrente de armadura durante a partida

No estado X16, é iniciada a varredura fazendo com que ocorra uma transição abrupta da velocidade mínima do sistema (440 rpm) para a velocidade mínima de varredura (1100 rpm). A partir apenas do que pode ser visto na Fig. 9.3, tal mudança brusca, aparentemente, não traz problemas, entretanto, ao analisarmos a Fig. 9.4, poderemos perceber que ocorre um grande surto de corrente de armadura no mesmo instante da variação abrupta de velocidade. Tal transitório não é benéfico aos componentes elétricos e mecânicos do sistema. A versão do programa usada para a aquisição destes resultados ainda não continha os estados X14a e X14b apresentados na Fig. 7.5 da Seção 7.4, que foram, posteriormente, implementados para produzir, quando necessário, uma rampa desde a velocidade mínima do sistema até a velocidade mínima de varredura. A presença desta rampa adicional impede a ocorrência do pico na corrente de armadura mostrado na Fig. 9.4.

9.1.2 A varredura do sistema MT

A Fig. 9.5 detalha a evolução da velocidade durante o período de varredura. Na maior parte do tempo, o erro de velocidade é nulo (nos patamares de velocidade máxima e mínima de varredura e na rampa ascendente). Entretanto, durante a frenagem do motor (quando a referência de velocidade cai abruptamente de seu valor máximo para mínimo), existe um transitório onde aparece um erro de velocidade considerável.

A Fig. 9.6 apresenta a referência de corrente de armadura e seu valor medido. Essas duas grandezas permanecem próximas durante o intervalo de aceleração e nos patamares de velocidade constante.

No intervalo de frenagem, existe uma diferença significativa entre a corrente de armadura e seu valor medido. A malha de velocidade comporta-se corretamente, produzindo uma referência de corrente de armadura nula (o valor mínimo estabelecido para a malha de velocidade) lembrando-se que os *choppers* utilizados são de um único quadrante. O resultado natural seria uma desaceleração lenta, definida pelos torques resistentes do sistema mecânico, e tal comportamento já ocorria no sistema de varredura acústica anteriormente existente. Entretanto, nota-se na Fig. 9.7, que a referência do *chopper* de armadura (que é a saída do controlador de corrente de armadura) diminui lentamente. Tal variação lenta se deve a um ajuste do ganho proporcional da malha de corrente em nível relativamente baixo. Ganhos proporcionais mais altos chegaram a ser testados, mas o ajuste da malha de corrente de armadura mostrou-se uma tarefa delicada, podendo produzir grandes variações de corrente (a resistência de armadura é baixa).

Fig. 9.5 Variáveis referentes ao controle de velocidade durante a varredura

Fig. 9.6 Variáveis referentes ao controle de corrente de armadura durante a varredura

Fig. 9.7 Referência do *chopper* de armadura durante a varredura

9.1.3 O desmonte do sistema MT

A Fig. 9.8 exibe a referência e a realimentação de velocidade durante o desmonte da varredura.

Fig. 9.8 Variáveis referentes ao controle de velocidade durante o desmonte

No estado X18, é executada uma rampa que leva a velocidade desde o valor em que o comando de parada de varredura foi executado (aproximadamente 1475 rpm) até a velocidade mínima absoluta do sistema (440 rpm) e, no estado X21, outra rampa leva o motor ao repouso. A duração dessas rampas é um parâmetro configurável apenas pelo comissionador do sistema. As inclinações dessas duas rampas dificilmente coincidem pois os seus valores iniciais e finais são estabelecidos pelo operador quando este define a referência de varredura.

9.2 O sistema magnético da cauda em funcionamento

Embora tenham-se apresentado apenas informações bastante genéricas sobre o sistema magnético, a Fig. 9.10 ilustra o funcionamento da sua malha de controle mais externa quando operando com o dispositivo de varredura chamado cauda.

A cauda consiste em um conjunto de eletrodos submersos na água do mar através do qual circula uma corrente elétrica elevada, denominada de corrente de cauda. O objetivo desta corrente é produzir uma distorção do campo magnético terrestre similar à que seria produzida pela passagem de um navio. Nos navios da classe Aratu, a corrente nominal da cauda (regime contínuo) é de 820 A mas o sistema é capaz de produzir valores superiores durante períodos mais curtos.

Conforme mostra a Fig. 9.9, o sistema magnético consiste num conjunto de geradores ligados em cascata para produzir a corrente de cauda. A excitação desses geradores é definida pelo sistema de controle da varredura magnética.

Fig. 9.9 Diagrama de blocos de controle do sistema magnético

A variável controlada pela malha mais externa é a corrente de cauda, e sua referência é definida de acordo com as características do navio-alvo que se deseja simular, sendo comum concatenar trechos de senóides com períodos de corrente nula. A Fig. 9.10 exibe uma referência de corrente com este perfil típico e a sua respectiva realimentação.

Fig. 9.10 Referência e realimentação da corrente de cauda (sistema magnético)

Todos os conceitos da norma IEC61131-3 aplicados ao sistema acústico também foram utilizados no sistema magnético, resultando um diagrama SFC com quase 40 estados, número superior ao do sistema acústico apresentado.

9.3 O gerador de sinal PRBS e o sistema magnético

No item 8.2, foi mencionada a existência de mecanismos para auxiliar o processo de identificação do comportamento dinâmico da planta. Tal mecanismo envolvia abrir algumas das malhas de controle e aplicar um sinal do tipo PRBS (*Pseudo Random Binary Sequence*) (LJUNG,1999). A Fig. 9.11 resulta do processo de identificação dos parâmetros do circuito de campo do gerador de varredura (o gerador elétrico à direita na Fig. 9.9). Os sinais mostrados na Fig. 9.11 são a entrada da planta a identificar (tensão no enrolamento de campo do gerador de varredura designada no gráfico como *Input*) e a sua saída (corrente no enrolamento de campo deste gerador designada como *Output*). O valor calculado por meio do modelo identificado está indicado no gráfico como *Model*.

Fig. 9.11 Resposta do campo do gerador de varredura ao sinal pseudo-aleatório

Fig. 9.12 Diagrama de blocos do sistema para identificação do campo do gerador de varredura

Embora o sinal PRBS se consista de uma alternância de valores positivos e negativos com transições muito rápidas, o sinal chamado *Input* na Fig. 9.11 apresenta bordas distorcidas. Isso ocorre porque o sinal PRBS é produzido pelo PLC e aplicado às referências dos *choppers* enquanto a entrada do sistema a identificar é a tensão produzida pela armadura do amplidínamo que alimenta o enrolamento de campo do gerador de varredura (ver Fig.9.9). Conforme pode ser visto na Fig. 9.12, a entrada do sistema a identificar é um sinal PRBS que já foi distorcido pelas

respostas dos *choppers* e circuitos de campo e armadura do amplidínamo. Isto não constitui um problema, pois o processo de identificação da função de transferência leva em conta apenas relação entre a entrada e a saída efetivamente medidas.

O cabeçalho do gráfico da Fig. 9.11 mostra dados adicionais produzidos pelo programa de identificação, entre as quais a taxa de amostragem utilizada no registro (10 ms). O modelo obtido para este trecho da planta, no domínio da frequência, é dado por uma função de primeira ordem:

$$G(s) = \frac{11.94}{1.119s + 1}$$

associada a um atraso puro de 10ms.

As ferramentas ilustradas acima permitiram desenvolver modelos matemáticos para a planta, tornando possível um ajuste preliminar e simulação das malhas de controle do sistema. No caso específico da planta ilustrada pela Fig. 9.11, o atraso puro é muito menor do que a constante de tempo do sistema, viabilizando o controle estável em malha fechada.

10 CONCLUSÃO

*E cheguei à seguinte conclusão:
Deus fez o homem correto,
mas o homem inventa muitas complicações.
Eclesiastes, Capítulo 7, Versículo 29*

Do trabalho apresentado, resultam as seguintes conclusões:

- O mercado de *hardware* voltado para automação industrial vive um momento de convergência em que equipamentos antes destinados a usos diferentes estão se tornando substitutos uns dos outros. Em contraste, o mercado de *software* destinado à automação industrial ainda procura definir seus caminhos, numa situação análoga às complicações bíblicas mencionadas acima;
- A norma IEC61131-3 é uma das alternativas possíveis ao mercado de *software* para controle e automação apesar de sua aceitação ainda restrita. Existem problemas de implementação que tornam distante o desejo de portabilidade de código entre equipamentos de fabricantes distintos;
- A norma IEC61131-3 é uma fonte valiosa de idéias e conceitos que, em certas situações, trazem um aumento de produtividade frente aos métodos tradicionais de programação de PLCs. Este trabalho apresentou esses conceitos aplicados a um projeto real: “*A modernização dos navios-varredores da Marinha do Brasil*”;
- Embora o PLC adotado pelo projeto não implementasse nativamente muitas das características desejáveis da norma, um conjunto de metodologias e ferramentas foram desenvolvidas ao longo do projeto para suprir essas deficiências. Apesar do tempo gasto para construir tais ferramentas, o aumento de produtividade resultante permitiu que o projeto fosse concluído com sucesso, resultando um código consistente, fácil de testar e de dar manutenção, com um número de homens-hora relativamente baixo.

Algumas extensões futuras deste trabalho podem ser consideradas:

- Aprimorar o tradutor de SFC para IL para que ele venha a executar tarefas que a versão atual não executa (como as listadas no final da Seção 7.1);
- Desenvolver um tradutor de ST para IL. Este tradutor substituiria, com vantagens, o uso do processador de macros M4 na implementação de blocos de controle e trechos de código em que existe manipulação aritmética intensa.

- É fundamental o acompanhamento permanente do mercado de *software* e *hardware* para automação industrial, além da evolução dos aspectos relativos à adoção da norma, para que se consiga acompanhar a dinâmica deste setor;
- Um estudo mais detalhado das plantas controladas e uma otimização dos ajustes das diversas malhas de controle.

A maioria dos capítulos se iniciou com uma citação bíblica e, ao final deste trabalho, na busca de um equilíbrio, cabem as palavras de Shakespeare (1600): “...*Até o diabo pode citar as Santas Escrituras em seu proveito...*” .

ANEXO A CÓDIGO FONTE DA MACRO ESCALONA

```

dn] Macro ESCALONA ( Escalonamento de Variaveis analogicas )
dn]
dn] Multiplica uma variável de entrada por uma fração determinada pelo seu numerador e
dn] denominador
dn]
dn] $1 : Entrada, ou seja, a variável a ser escalonada, dever ser INT
dn] $2 : Numerador, ou seja, o numerador da função de escalonamento, deve ser INT
dn] $3 : Denominador, ou seja, o denominador da função de escalonamento, deve ser INT
dn] $4 : Saida, ou seja, a variável escalonada, deve ser INT
dn]
dn]
dn] Variaveis Locais usadas comuns a todas as instâncias :
dn] Com_VA_EscalonaAux e Com_VA_Escalona2Aux, devem ser DINT.
dn] Com_VA_EscalonaAux_HI e Com_VA_Escalona2Aux_HI , devem ser INT e ter enderecos contiguos ( ou seja uma
dn] posicao a mais ) aos enderecos de Com_VA_EscalonaAux
dn] e Com_VA_Escalona2Aux, respectivamente.
dn]
dn]
dn] Possiveis Bugs no uso :
dn]
dn] 1 Esta macro supoe que cada variavel de entrada serah escalonada no maximo uma
dn] unica vez. Assim exite um rotulo no programa formado pelo prefixo Pulazera e
dn] tendo como sufixo o nome da variavel de entrada. Caso o pressuposto de um unico
dn] escalonamento de variavel de entrada nao seja atendido então o referido rotulo
dn] aparecerá mais de uma vez no programa e o Versapro irah acusar um erro de rotulo
dn] definido mais de uma vez.
dn]
dn] 2 Devido ao uso de variáveis auxiliares ( Com_VA_EscalonaAux e Com_VA_Escalona2Aux) para todos as
dn] instancias deste bloco, ele NAO devera ser chamado dentro de rotinas de interrupcao
dn] pois esta variavel comum poderá ser "suja" pela interrupção produzindo resultados
dn] impreviseíveis.
dn]
dn] define(`ESCALONA',`***** Inicio do bloco `ESCALONA' ( Entrada = $1,
dn] " Numerador = $2,
dn] " Denominador = $3,
dn] " Saida = $4)
dn]
dn] " Converte a entrada de INT para DINT
dn] LD_INT $1
dn] ST_INT Com_VA_EscalonaAux " Com_VA_EscalonaAux := $1; (* Parte Low de Com_VA_EscalonaAux recebe o valor de $1 *)
dn] LD_INT 0
dn] ST_INT Com_VA_EscalonaAux_HI " Com_VA_EscalonaAux_HI := 0; (* Zera Parte hi de Com_VA_EscalonaAux *)
dn] LT_INT(
dn] IN1 := $1
dn] IN2 := 0
dn] )
dn] JMPCN Pulazera$1 " IF $1 < 0 THEN
dn] LD_INT 1
dn] ST_INT Com_VA_EscalonaAux_HI " Com_VA_EscalonaAux_HI := 1; (* Ajuste do sinal da parte high de Com_VA_EscalonaAux *)

```

```

Pula2Zera$1: " " END_IF (* $1 < 0 *)
" Converte o numerador de INT para DINT
LD_INT $2
ST_INT Com_VA_Escalona2Aux " Com_VA_Escalona2Aux recebe o valor de $2 *)
LD_INT 0
ST_INT Com_VA_Escalona2Aux_HI " Com_VA_Escalona2Aux_HI := 0; (* Zera Parte hi de Com_VA_Escalona2Aux *)
LT_INT(
  IN1 := $2
  IN2 := 0
)
JMPCN Pula2Zera$1 " IF $2 < 0 THEN
LD_INT 1
ST_INT Com_VA_Escalona2Aux_HI " Com_VA_Escalona2Aux_HI := 1; (* Ajuste do sinal da parte high de Com_VA_Escalona2Aux *)

Pula2Zera$1: " " END_IF (* $2 < 0 *)
MUL_DINT(
  IN1 := Com_VA_Escalona2Aux
  IN2 := Com_VA_Escalona2Aux
)
ST_DINT Com_VA_Escalona2Aux " Com_VA_Escalona2Aux := Com_VA_Escalona2Aux * $2; (* Variavel Auxiliar := Entrada * Numerador *)

" Converte o denominador de INT para DINT
LD_INT $3
ST_INT Com_VA_Escalona2Aux " Com_VA_Escalona2Aux := $3; (* Parte Low de Com_VA_Escalona2Aux recebe o valor de $3 *)
LD_INT 0
ST_INT Com_VA_Escalona2Aux_HI " Com_VA_Escalona2Aux_HI := 0; (* Zera Parte hi de Com_VA_Escalona2Aux *)
LT_INT(
  IN1 := $3
  IN2 := 0
)
JMPCN Pula3Zera$1 " IF $3 < 0 THEN
LD_INT 1
ST_INT Com_VA_Escalona2Aux_HI " Com_VA_Escalona2Aux_HI := 1; (* Ajuste do sinal da parte high de Com_VA_Escalona2Aux *)

Pula3Zera$1: " " END_IF (* $3 < 0 *)
DIV_DINT(
  IN1 := Com_VA_Escalona2Aux
  IN2 := Com_VA_Escalona2Aux
)
ST_DINT Com_VA_Escalona2Aux " Com_VA_Escalona2Aux := Com_VA_Escalona2Aux / $3; (* Variavel Auxiliar := (Entrada * Numerador) / Denominador; *)
LD_INT Com_VA_Escalona2Aux
ST_INT $4 " $1 := Com_VA_Escalona2Aux; (* Parte low da variável auxiliar deve é copiada na saída *)
"***** Fim do bloco `ESCALONA'($1,$2,$3,$4)'

```


ANEXO B TRADUÇÃO DE SFC PARA IL: SINTAXE E EXEMPLOS

Será apresentada, de modo informal, a sintaxe da linguagem de descrição textual de SFC. Exemplos curtos e simples mostrando os principais elementos e características desta linguagem e sua interface com a linguagem IL também serão mostrados. Para detalhes sobre o mecanismo de tradução do SFC em IL, convém consultar Baracos (1992, pp. 139 a 148). Ao fim do anexo, estão as listagens dos códigos resultantes da tradução dos exemplos apresentados.

B.1 Apresentação informal da sintaxe da linguagem de descrição textual de SFC

A descrição textual de um SFC é composta por uma sequência de diretivas que descrevem os elementos que compõem o diagrama. A ordem na qual as diretivas aparecem no texto é irrelevante. Entretanto, é aconselhável que essas diretivas sigam a mesma sequência do diagrama, facilitando o eventual processo de manutenção do código.

Passo Inicial:

```
%INITIALSTEP <NOME DO ESTADO INICIAL >  
    <CÓDIGO EM IL PARA DEFINIR A AÇÃO EXECUTADA POR ESTE PASSO>  
%ENDSTEP
```

No processo de tradução é suposto que existe uma variável booleana declarada na interface de programação com o nome igual ao nome do estado. No programa em IL gerado, essa variável estará associada à atividade deste estado, ou seja, quando ela estiver verdadeira, o estado estará ativo e vice-versa. Cabe ao programador declarar esta variável na tabela de variáveis da interface de programação, pois o programa tradutor não faz isso automaticamente.

Passo regular:

```
%STEP <NOME DO ESTADO >  
    <CÓDIGO EM IL PARA DEFINIR A AÇÃO EXECUTADA POR ESTE PASSO>  
%ENDSTEP
```

Todos os comentários feitos sobre o estado inicial valem para o estado regular.

Forces:

Nas diretivas passo inicial e passo regular, opcionalmente, após o nome do estado, podem ser colocados os modificadores:

FORCEON <BIT FORCE ON>: Faz que o estado modificado por esta expressão torne-se ativo quando o <BIT FORCE ON> for verdadeiro. Isto ocorrerá independentemente da evolução normal do

SFC, resultando daí o nome FORCE (forçar).

FORCEOFF <BIT FORCE OFF> : Faz que o estado modificado por esta expressão torne-se inativo quando o <BIT FORCE OFF> for verdadeiro. Como no modificador anterior, isto ocorrerá independentemente da evolução normal do SFC.

O mecanismo de retorno implícito ao estado inicial seguro (X1) quando ocorre um erro, citado nos capítulos 6 e 7, usa esses modificadores.

Transição:

```
%TRANSITION <NOME DA TRANSIÇÃO> FROM <ESTADO ORIGEM>
<CÓDIGO EM IL PARA DEFINIR A CONDIÇÃO DA TRANSIÇÃO>
%TO <ESTADO DESTINO>
```

Tal como nas diretivas anteriores, o programador deve declarar na interface de programação uma variável booleana com o mesmo nome da transição que irá indicar seu estado lógico no código IL gerado.

<ESTADO ORIGEM> se refere ao estado anterior à transição e <ESTADO DESTINO> se refere ao estado posterior à transição. O tradutor não faz nenhum tipo de verificação se os estados origem e destino existem. Uma transição pode ter vários estados de origem e/ou destino e este recurso é usado para implementarmos o paralelismo de operações apresentado mais adiante.

Existe uma restrição ao código em IL associado a uma transição: ele deve sempre deixar um valor booleano no topo da pilha interna do IL, para comunicarmos o estado booleano da condição associada à transição.

Temporização de transições:

Diretiva para facilitar a construção de transições disparadas por tempo. Tem a seguinte forma:

```
%DELAY <NOME DO DELAY> FROM <BIT DE INICIO> <TEMPO> <UNIDADE> TO <BIT RESULTADO>
```

<NOME DO DELAY>: É o nome da variável temporizador que será usada em IL para implementar este atraso. Será necessário que o programador declare na interface de programação esta variável.

<BIT DE INICIO>: É a variável booleana que vai deflagrar o início da contagem do temporizador. Caso esta variável não tenha ainda sido declarada, deve ser, então, incluída na lista de variáveis booleanas da interface de programação.

<TEMPO>: Conjunto de dígitos que vai indicar quantas unidades de tempo irá durar o atraso. Uma alternativa é colocar o nome de uma variável declarada na tabela de variáveis da interface de programação. Assim procedendo, a temporização irá durar o número de unidades

correspondente ao valor acumulado nesta variável.

<UNIDADE>: Pode ser uma das palavras: TENTHS, HUNDS ou THOUS para indicar, respectivamente, décimos de segundo, centésimos de segundo ou milésimos de segundo.

<BIT RESULTADO>: Variável booleana (geralmente bit auxiliar) que irá se tornar verdadeiro ao fim da contagem. Normalmente, este bit é usado numa transição temporizada.

B.2 Exemplos de representação textual de SFCs

No primeiro exemplo, iremos implementar um SFC sequencial bastante simples, no qual existem três estados (X1, X2 e X3), um deles sendo o inicial. A condição de transição da atividade desses estados (Y1, Y2 e Y3) será dada por chaves conectadas nas entradas digitais do PLC (ENTRADA1, ENTRADA2 e ENTRADA3). Como ação associada a cada estado, iremos energizar uma saída digital correspondente a cada estado (SAIDA1 ou SAIDA2 ou SAIDA3) e desligar as outras duas.

Fig. B.1 Primeiro exemplo de SFC

```

%initialstep X1
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA1
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA2
  ST_BOOL  SAIDA3
%endstep

%transition Y1 from X1
  LD_BOOL  ENTRADA1
%to X2

%step X2
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA2
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA3
%endstep

%transition Y2 from X2
  LD_BOOL  ENTRADA2
%to X3

%step X3
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA3
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA2
%endstep

%transition Y3 from X3
  LD_BOOL  ENTRADA3
%to X1
  
```

No segundo exemplo, é apresentado um grafo SFC em que existem dois caminhos paralelos não executados simultaneamente (divergência e convergência OU). A Fig. B.2 mostra o diagrama deste exemplo. Na listagem a seguir, está a transcrição textual do diagrama acima.

Fig. B.2 Segundo exemplo de SFC

O estado X1 é estado de origem de mais de uma transição (Y1 e Y2) e é estado de destino de mais de uma transição (Y3 e Y4). Este é o mecanismo usado para descrevermos caminhos alternativos na sequência de ativação de estados.

```

%InitialStep X1
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA1
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA2
  ST_BOOL  SAIDA3
%endstep

%transition Y1 from X1
  LD_BOOL  ENTRADA1
%to X2

%transition Y2 from X1
  LD_BOOL  ENTRADA2
%to X3

%step X2
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA3
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA2
%endstep

%step X3
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA2
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA3
%endstep

%transition Y3 from X2
  LD_BOOL  ENTRADA3
%to X1

%transition Y4 from X3
  LD_BOOL  ENTRADA4
%to X1
  
```

O objetivo do terceiro exemplo é descrever um grafo SFC em que existem dois caminhos paralelos executados em paralelo (divergência e convergência AND). A Fig. B.3 mostra o diagrama deste grafo. Usamos transições com mais de um estado de destino (de Y1 para X2 e X3) ou com mais de um estado de origem (de X2 e X4 para Y3) para indicar que os estado X2 e X3 ou X4 entram em atividade simultaneamente.

Fig. B.3 Terceiro exemplo de SFC

```
%initialstep X1
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA1
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA2
  ST_BOOL  SAIDA3
  ST_BOOL  SAIDA4
%endstep

%transition Y1 from X1
  LD_BOOL  ENTRADA1
%to X2 X3

%step X2
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA2
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
%endstep

%step X3
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA3
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA4
%endstep

%transition Y2 from X3
  LD_BOOL  ENTRADA2
%to X4

%step X4
  LD_BOOL  LIGADO
  ST_BOOL  SAIDA4
  LD_BOOL  DESLIGADO
  ST_BOOL  SAIDA1
  ST_BOOL  SAIDA3
%endstep

%transition Y3 from X2 X4
  LD_BOOL  ENTRADA3
```

A topologia do quarto exemplo (mostrado na Fig. B.4) é semelhante à do primeiro exemplo sendo que as transições por chaves foram substituídas por transições por tempo.

Fig. B.4 Quarto exemplo de SFC

No diagrama acima, usaram-se nas transições a notação para temporizadores dada pela norma IEC848. Existem três elementos separados por barras (/). O primeiro é a letra T, para indicar temporização. O segundo elemento mostra qual variável booleana dispara o início da contagem de tempo. O terceiro é composto de um número e uma unidade de tempo indica quanto tempo o temporizador deve contar antes de tornar sua saída verdadeira. As unidades usadas são décimos de segundo (ds), centésimos de segundo (cs) e milésimos de segundo (ms). O grafo do quarto exemplo irá ativar os estados X1, X2 e X3 em sequência durante um segundo cada. Na descrição textual do quarto exemplo, foram acrescentados modificadores *force* nas diretivas *initialstep* e *step*. Isto faz com que X1 se torne ativo e os estados X2 e X3 se tornem inativos quando o *bit* ENTRADA4 se torna verdadeiro. O SFC volta a evoluir normalmente quando ENTRADA4 se torna falsa.

```

%initialstep X1 forceon entrada4
    LD_BOOL    LIGADO
    ST_BOOL    SAIDA1
    LD_BOOL    DESLIGADO
    ST_BOOL    SAIDA2
    ST_BOOL    SAIDA3
%endstep

%delay d1 from X1 10 tenths to AUX1

%transition Y1 from X1
    LD_BOOL    AUX1
%to X2

%step X2 forceoff entrada4
    LD_BOOL    LIGADO
    ST_BOOL    SAIDA2
    LD_BOOL    DESLIGADO
    ST_BOOL    SAIDA1
    ST_BOOL    SAIDA3
%endstep

%delay d2 from X2 100 hunds to AUX2

%transition Y2 from X2
    LD_BOOL    AUX2
%to X3

%step X3 forceoff entrada4
    LD_BOOL    LIGADO
    ST_BOOL    SAIDA3
    LD_BOOL    DESLIGADO
    ST_BOOL    SAIDA1
    ST_BOOL    SAIDA2
%endstep

%delay d3 from X3 1000 thous to AUX3

%transition Y3 from X3
    LD_BOOL    AUX3
%to X1
  
```

B.3 Código resultante da tradução dos exemplos

O *bit* PRIMEIRO nas listagens abaixo é setado pelo PLC apenas no seu primeiro *scan*.

O *bit* LIGADO é sempre verdadeiro enquanto o DESLIGADO é sempre falso.

B.3.1 Código resultante da tradução do primeiro exemplo

```

                                "Codigo gerado a partir do arquivo primeiro.sfc
LD_BOOL  ENTRADA1
AND      X1
ST_BOOL  y1      "Gatilho da transicao y1
LD_BOOL  ENTRADA2
AND      X2
ST_BOOL  y2      "Gatilho da transicao y2
LD_BOOL  ENTRADA3
AND      X3
ST_BOOL  y3      "Gatilho da transicao y3
LD_BOOL  PRIMEIRO
OR      y3
OR(
LD_BOOL  x1
ANDN     y1
)
ST_BOOL  x1      "Atividade do estado x1
LDN_BOOL PRIMEIRO
AND(
LD_BOOL  y1
OR(
LD_BOOL  x2
ANDN     y2
)
)
ST_BOOL  x2      "Atividade do estado x2
LDN_BOOL PRIMEIRO
AND(
LD_BOOL  y2
OR(
LD_BOOL  x3
ANDN     y3
)
)
ST_BOOL  x3      "Atividade do estado x3
LD_BOOL  x1
JMPCN    fimx1
LD_BOOL  LIGADO
ST_BOOL  SAIDA1
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA2
ST_BOOL  SAIDA3
fimx1:    "Executa a acao ligada ao estado x1
LD_BOOL  x2
JMPCN    fimx2
LD_BOOL  LIGADO
ST_BOOL  SAIDA2
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA1
ST_BOOL  SAIDA3
fimx2:    "Executa a acao ligada ao estado x2
LD_BOOL  x3
JMPCN    fimx3
LD_BOOL  LIGADO
ST_BOOL  SAIDA3
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA1
ST_BOOL  SAIDA2
fimx3:    "Executa a acao ligada ao estado x3

```

B.3.2 Código resultante da tradução do segundo exemplo

```

                                "Codigo gerado a partir do arquivo segundo.sfc
LD_BOOL  ENTRADA1
AND      X1
ST_BOOL  y1      "Gatilho da transicao y1
LD_BOOL  ENTRADA2
AND      X1
ST_BOOL  y2      "Gatilho da transicao y2
LD_BOOL  ENTRADA3
AND      X2
ST_BOOL  y3      "Gatilho da transicao y3
LD_BOOL  ENTRADA4
AND      X3
ST_BOOL  y4      "Gatilho da transicao y4
LD_BOOL  PRIMEIRO
OR       y3
OR       y4
OR(
LD_BOOL  x1
ANDN     y1
ANDN     y2
)
ST_BOOL  x1      "Atividade do estado x1
LDN_BOOL PRIMEIRO
AND(
LD_BOOL  y1
OR(
LD_BOOL  x2
ANDN     y3
)
)
ST_BOOL  x2      "Atividade do estado x2
LDN_BOOL PRIMEIRO
AND(
LD_BOOL  y2
OR(
LD_BOOL  x3
ANDN     y4
)
)
ST_BOOL  x3      "Atividade do estado x3
LD_BOOL  x1
JMPCN    fimx1
LD_BOOL  LIGADO
ST_BOOL  SAIDA1
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA2
ST_BOOL  SAIDA3
fimx1:    "Executa a acao ligada ao estado x1
LD_BOOL  x2
JMPCN    fimx2
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA1
ST_BOOL  SAIDA3
LD_BOOL  LIGADO
ST_BOOL  SAIDA2
fimx2:    "Executa a acao ligada ao estado x2
LD_BOOL  x3
JMPCN    fimx3
LD_BOOL  DESLIGADO
ST_BOOL  SAIDA1
ST_BOOL  SAIDA2
LD_BOOL  LIGADO
ST_BOOL  SAIDA3
fimx3:    "Executa a acao ligada ao estado x3

```


B.3.3 Código resultante da tradução do terceiro exemplo

```

LD_BOOL    ENTRADA1    "Codigo gerado a partir do arquivo terceiro.sfc
AND         x1
ST_BOOL    y1          "Gatilho da transicao y1
LD_BOOL    ENTRADA2
AND         x3
ST_BOOL    y2          "Gatilho da transicao y2
LD_BOOL    ENTRADA3
AND         x2
AND         x4
ST_BOOL    y3          "Gatilho da transicao y3
LD_BOOL    PRIMEIRO
OR          y3
OR(
LD_BOOL    x1
ANDN       y1
)
ST_BOOL    x1          "Atividade do estado x1
LDN_BOOL   PRIMEIRO
AND(
LD_BOOL    y1
OR(
LD_BOOL    x2
ANDN       y3
)
)
ST_BOOL    x2          "Atividade do estado x2
LDN_BOOL   PRIMEIRO
AND(
LD_BOOL    y1
OR(
LD_BOOL    x3
ANDN       y2
)
)
ST_BOOL    x3          "Atividade do estado x3
LDN_BOOL   PRIMEIRO
AND(
LD_BOOL    y2
OR(
LD_BOOL    x4
ANDN       y3
)
)
ST_BOOL    x4          "Atividade do estado x4
LD_BOOL    x1
JMPCN      fimx1
LD_BOOL    LIGADO
ST_BOOL    SAIDA1
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA2
ST_BOOL    SAIDA3
ST_BOOL    SAIDA4
fimx1:      "Executa a acao ligada ao estado x1
LD_BOOL    x2
JMPCN      fimx2
LD_BOOL    LIGADO
ST_BOOL    SAIDA2
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA1
fimx2:      "Executa a acao ligada ao estado x2
LD_BOOL    x3
JMPCN      fimx3
LD_BOOL    LIGADO
ST_BOOL    SAIDA3
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA1
ST_BOOL    SAIDA4
fimx3:      "Executa a acao ligada ao estado x3
LD_BOOL    x4
JMPCN      fimx4
LD_BOOL    LIGADO
ST_BOOL    SAIDA4
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA1
ST_BOOL    SAIDA3
fimx4:      "Executa a acao ligada ao estado x4

```

B.3.4 Código resultante da tradução do quarto exemplo

```

LD_BOOL    AUX1          "Codigo gerado a partir do arquivo quarto.sfc
AND         X1
ST_BOOL    y1            "Gatilho da transicao y1
LD_BOOL    AUX2
AND         X2
ST_BOOL    y2            "Gatilho da transicao y2
LD_BOOL    AUX3
AND         X3
ST_BOOL    y3            "Gatilho da transicao y3
LD_BOOL    PRIMEIRO
OR          y3
OR(
LD_BOOL    x1
ANDN       y1
)
OR         entrada4
ST_BOOL    x1            "Atividade do estado x1
LDN_BOOL   PRIMEIRO
ANDN       entrada4
AND(
LD_BOOL    y1
OR(
LD_BOOL    x2
ANDN       y2
)
)
ST_BOOL    x2            "Atividade do estado x2
LDN_BOOL   PRIMEIRO
ANDN       entrada4
AND(
LD_BOOL    y2
OR(
LD_BOOL    x3
ANDN       y3
)
)
ST_BOOL    x3            "Atividade do estado x3
LD_BOOL    x1
TMR_TENTHS(
  Address  := d1
  PV       := 10
)
ST_BOOL    AUX1          "Executa o delay d1
LD_BOOL    X2
TMR_HUNDS(
  Address  := d2
  PV       := 100
)
ST_BOOL    AUX2          "Executa o delay d2
LD_BOOL    X3
TMR_THOUS(
  Address  := d3
  PV       := 1000
)
ST_BOOL    AUX3          "Executa o delay d3
LD_BOOL    x1
JMPCN      fimx1
LD_BOOL    LIGADO
ST_BOOL    SAIDA1
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA2
ST_BOOL    SAIDA3
fimx1:      "Executa a acao ligada ao estado x1
LD_BOOL    x2
JMPCN      fimx2
LD_BOOL    LIGADO
ST_BOOL    SAIDA2
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA1
ST_BOOL    SAIDA3
fimx2:      "Executa a acao ligada ao estado x2
LD_BOOL    x3
JMPCN      fimx3
LD_BOOL    LIGADO
ST_BOOL    SAIDA3
LD_BOOL    DESLIGADO
ST_BOOL    SAIDA1
ST_BOOL    SAIDA2
fimx3:      "Executa a acao ligada ao estado x3

```

LISTA DE REFERÊNCIAS

- [ARNOLD,2005] Arnold, S., **Foxboro A2 white paper case study: System cost and engineering advantages over PLCs**. Disponível em: <<http://www.vfd.com/foxboroa2whitepaper.pdf>> . Acesso em 20 de maio 2005
- [BARACOS,1992] Baracos, P., **Grafcet step by step** . St-Laurent: Famic Automation, 1992. 156p.
- [BARROS,2002] Barros, G. B., **Simulador dos sistemas de varredura acústica e magnética dos navios varredores da classe Aratu**. 2002. 100p. Trabalho de Conclusão de Curso. (Graduação em Engenharia Elétrica) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2002
- [BONFATTI *et al*,1997] Bonfatti, F.; Monari, P. D.; Sampieri, U., **IEC1131-3 programming methodology** . Seyssins: C.J. International, 1997. 377p.
- [BUXBAUM *et al*. 1990] Buxbaum, A.; Schierau, K.;Straughen, A., **Design of control systems for DC drives** . Berlim: Springer-Verlag, 1990. 237p.
- [CONTROL.COM, 2001] Control.com, **ENGR: Ladder logic, automation replacing people**. Disponível em: <http://www.control.com/996452066/index_html> . Acesso em 20 maio 2005
- [CONTROL.COM, 2002] Control.com, **Why to program with high level programming languages**. Disponível em: <http://www.control.com/1026147036/index_html> . Acesso em 20 maio 2005
- [CONTROL ENGINEERING,2000] Control Engineering, **Smaller PLCs retain popularity**. Mar. 2000Disponivelem <<http://www.manufacturing.net/ctl/article/CA191302>> . Acesso em 20 maio 2005
- [CONTROL ENGINEERING,2004] Lloyd,A.,**PLC market misconceptions abound**. Control Engineering Europe, p.14-15, Oct. 2004
- [CPAN,2005] **Comprehensive Perl Archive Network**. Disponível em: <<http://www.cpan.org/>>. Acesso em 20 maio 2005
- [CRATER,2005A] Crater, K., **White paper: When technology standards become counter-productive**. Disponível em: <<http://www.ctc-control.com/customer/elearning/whpapers/counter.asp>> . Acesso em 20 de maio 2005
- [CRATER,2005B] Crater, K., **White paper: State language for machine control**. Disponível em: <<http://www.ctc-control.com/customer/elearning/whpapers/StLangPaper.asp>> . Acesso em 20 de maio 2005
- [DAVIDSON *et al*.,1997] Davidson, C.; McWhinnie, J., **Stepping off the ladder**. IEE Review, p. 210-212, Sept.1997
- [FERNANDES *et al*.,2001] Fernandes, L. A. P.; Goldemberg, C., **Técnicas “anti-windup”** In: Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Energia e Automação Elétricas, BT/PEA/0119. 2001

- [FOXBORO,2005] Foxboro, **When to consider a DCS vs a PLC** Disponível em: <<http://www.vfd.com/whentoconsider.pdf>> . Acesso em 20 de maio 2005
- [FRASER *et al.*,1995] Fraser, C.; Hanson, D., **A retargetable C compiler: design and implementation** . Menlo Park: Addison-Wesley, 1995. 564p.
- [FREY,2002] Frey, G., **Formal methods in PLC control demonstrated at a flexible manufacturing line.** Disponível em: <www.eit.uni-kl.de/frey/papers/PDF/V150.pdf> . Acesso em 20 de maio 2005
- [GALCERAN *et al.* ,2001] Galceran, S.; Sudria, A.; Bergas, J.; Benitez, I.; Vazquez, L.; Boye, G.; Obregon, O., **Use of IEC1131 programming in virtual laboratory** In: Proceedings Conference on Emerging Technologies and Factory Automation . 2001. p. 645-649 v.1
- [GOULD,1999] Gould, L. S., **When controls converge: CNC,PLC & PC** Automotive Design Production, Jan. 1999 Disponível em: <<http://www.autofieldguide.com/articles/019903.html>>. Acesso em 20 de maio 2005
- [HAMILTON,1993] Hamilton, A., **The effect of IEC 1131-3 on the testing process for real time software applications** In: Advances in Software Engineering for PLC. 1992. p.4/1-4/3
- [HUTT,1998] Hutt, M. D.; Speh, T. W., **Bussiness marketing management** . Orlando: The Dryden Press, 1998, 777p.
- [IEC,1993] International Eletrotechnical Commission **Programmable Controllers Part 3, Programming Languages, IEC1131-3** . Geneva: IEC, 1993. 207p.
- [ISAGRAF, 2005] ISaGRAF, **ICS TRIPLEX ISaGRAF**. Disponível em: <<http://www.isagraf.com/>> . Acesso em 22 de maio 2005
- [JIMÉNEZ *et al.*,2001] Jimenez, I.; Lopez, E.; Ramirez, A., **Synthesis of ladder diagrams from Petri nets controller models** In: Proceedings of the 2001 IEEE International Symposium on Intelligent Control. 2001. p. 225-230
- [JOHN *et al.*,2001] John, K. H.; Tiegelkamp, M., **IEC61131-3: Programming industrial automation systems**. Berlin: Springer, 2001. 376p.
- [JUER *et al.*,1993] Juer, J.; Oliver,J.J., **Building and using graphical programming tools for the IEC 1131-3 standard** In: Advances in Software Engineering for PLC. 1993 . p. 5/1 - 5/4
- [KIM *et al.*,1999] Kim, H. S.; Lee, J. Y.; Kwon, W. H. **A compiler design for IEC1131-3 Standard languages for programmable logic controllers** In: 38th Annual Conference Proceedings of the SICE Annual. 1999. p. 1155-1160
- [KISSEL,1986] Kissel, T. E., **Understanding and using programmable controllers** . Englewood Cliffs: Prentice Hall,Inc., 1986. 209p.
- [KRIGMAN,1985] Krigman, A., **Relay ladder diagrams: we love them, we love them not.** InTech, v. 32, n.10, p. 39-44, 1985

- [KUKA,2005] Kuka, **Advantages of the Soft-PLC**. Disponível em: <http://www.kuka.be/main/products/soft_plc/advantages/e_soft_plc_advantages.htm>. Acesso em 20 de maio 2005
- [LABVIEW, 2005] LabVIEW, **LabVIEW - The software that powers instrumentation**. Disponível em: <<http://www.ni.com/labview/>> . Acesso em 22 de maio 2005
- [LEWIS,1995] Lewis, R. W., **Programming industrial control systems using IEC1131-3** . London: The Institution of Electrical Engineers, 1995. 281p.
- [LEWIS,1996] Lewis, R.W., **How can IEC 1131-3 improve the quality of industrial control software?** In: International Conference on Control '96, UKACC(Conf. Publ. No. 427) , Volume: 2.1996 p.1389-1393
- [LEWIS,2001] Lewis, R., **Modelling distributed control systems using IEC61499** . London: The Institution of Electrical Engineers, 2001.
- [LJUNG,1999] Ljung, L., **System Identification: Theory for the user**.Upper Saddle River: Prentice Hall, 1999
- [LLOYD,2004] Lloyd,A., **PLC market misconceptions abound**. Control Engineering Europe. Control Engineering Europe, p. 14-15, Oct. 2004
- [M4, 2005] Gnu Project - Free Software Foundation, **GNU M4**. Disponível em: <www.gnu.org/software/m4/> . Acesso em 07 de março 2005
- [MACIEL *et al.*,2004] Maciel, P.H.S.; Carvalho, P. C., **Interfaces homem-máquina**. Mecatrônica Atual, n. 17, p. 15-17, Ago. 2004
- [MACIEL,2005] Maciel, P.H., **Softwares de supervisão**. Mecatrônica Atual, n. 20, p. 15-19, Fev. 2005
- [MARINHA,2005] Marinha do Brasil, **Navio Varredor Classe Aratu**. Disponível em: <<http://www.mar.mil.br/aratu.htm>> . Acesso em 23 de maio 2005
- [MATA,2005] Mata, R. S., **Uma visão atual sobre os sistemas heterogêneos na automação industrial**. Mecatrônica Atual, n. 21, p.14-19, Abr. 2005
- [MATLAB, 2005] MATLAB, **MATLAB and Simulink for technical computing**. Disponível em: <<http://www.mathworks.com/>> . Acesso em 22 de maio 2005
- [MICHEL,1990] Michel, G., **Programmable logic controllers** . Chichester: John Wiley & Sons, 1990. 338p.
- [MIYAGI,1996] Miyagi, P. E. **Controle programável** . São Paulo: Edgard Blücher, 1996. 194p.
- [MOORE,1996] Moore, G. A. **Dentro do furacão**. São Paulo: Futura, 1996. 272p.
- [MORAES *et al.*,2001] Moraes, C. C.; Castrucci, P. L. **Engenharia de automação industrial**. Rio de Janeiro: L.T.C., 2001. 295p.

- [MORLEY,2005] Morley, D., **The history of PLC**. Disponível em: <www.barn.org/FILES/historyofplc.html> . Acesso em 07 de março 2005
- [NATALE,1989] Natale, F. **Automação industrial** . São Paulo: Nobel, 1989. 194p.
- [NATIONAL,2005A] National, **PAC: The next-generation PLC**. Disponível em: <<http://digital.ni.com/worldwide/bwcontent.nsf/web/all/57CD875381DDC91286256F43005B80FD>> . Acesso em 20 maio 2005
- [NATIONAL,2005B] National, **PACs for industrial control, the future of control**. Disponível em: <<http://zone.ni.com/devzone/conceptd.nsf/webmain/...63B424952E7EB98B86256F9B00766805>>. Acesso em 20 maio 2005
- [OLIVEIRA,1993] Oliveira, J. C. P., **Controlador programável** . São Paulo: Makron Books do Brasil, 1993. 200p.
- [OMAC,2005] OMAC, **Requirements of open, modular architecture controllers for applications in the automotive industry**. Disponível em: <<http://www.omac.org/Techdocs/omacv11.htm>>. Acesso em 20 de maio 2005
- [OULTON, 1992] Oulton, B., **Structured programming based on IEC SC 65A: using alternate programming methodologies and languages with programmable controllers** In: IEEE Conference Record of 1992 Forty-Fourth Annual Conference of Electrical Engineering Problems in the Rubber and Plastics Industries. 1992. p. 18-20
- [PERÉZ *et al.*,1992] Pérez, E. M.; Acevedo, J. M.; López, S. A. P. **Controladores lógicos y autómatas programmables** . Barcelona: Marcombo Boixareu, 1992. 393p.
- [PESHEK *et al.*,1993] Peshek, C. J.; Mellish, M.T. **Recent developments and future trends in PLC programming languages and programming tools for real-time control** In: Record of IEEE Cement Industry Technical Conference Papers. 1993. p. 219-230
- [PETROBRÁS, 1996] PETROBRÁS, **ET-3000-1200-800-PGT-003 : Padronização para execução do ladder do CP** Especificação técnica. 1996
- [PYTHON, 2005] **BeginnersGuide/Overview**. Disponível em: <<http://wiki.python.org/moin/BeginnersGuide/Overview>>. Acesso em 20 de maio 2005
- [PLCOPEN, 2005] PLCopen, **World wide PLC market**. Disponível em: <<http://www.plcopen.org/promotio/wwmarket.htm>> . Acesso em 20 de maio 2005
- [POLLARD, 1994] Pollard, J. R., **Ladder logic remains the PLC language of choice**. Control Engineering, p. 77-79, Apr. 1994
- [POLLARD, 1995] Pollard, J., **The future of PLC programming**. Control Engineering, p. 83-88, Feb. 1995
- [PORTER, 1986] Porter, M.E., **Estratégia competitiva** . Rio de Janeiro: Campus, 1986. 363p.
- [RALIZE, 2004] Ralize, C. H. C., **Sistemas de controle por PC**. Mecatrônica Atual, n. 17, p. 18-21, Ago. 2004

- [RILLO, 1983] Rillo, M., **Controlador programável utilizando grafos de comando etapa-transição**. 1983. 107p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo. São Paulo, 1983
- [ROCKWELL, 2005A] Rockwell Automation, **What is IEC1131**. Disponível em: <<http://www.software.rockwell.com/corporate/reference/Iec1131/>>. Acesso em 20 de maio 2005
- [ROCKWELL, 2005B] Rockwell Automation, **Two new programming languages increase versatility of RSLogix 5000 software**. Disponível em: <http://support.rockwellautomation.com/softwareconnection/swc02_1/twonew.asp> . Acesso em 20 de maio 2005
- [SANDUSKI, 1989] Sandusky, R.D., **PLC and PC system documentation concepts** In: IEEE Conference Record of 1989 Forty-First Annual Conference of Electrical Engineering Problems in the Rubber and Plastics Industries, 1989. p. 38-47
- [SHAKESPEARE, 1600] Shakespeare, W. **The merchant of Venice**. London, 1600
- [SHIRASUNA, 2004] Shirasuna, M., **Ethernet industrial - Parte 1 - Introdução e a norma IEC61158 (FIELDBUS)**. Mecatrônica Atual, n. 17, p. 58-60, Ago. 2004
- [SHIRASUNA, 2005A] Shirasuna, M., **Ethernet industrial - Parte 4 - Protocolos industriais**. Mecatrônica Atual, n. 20, p. 26-29, Mar. 2005
- [SHIRASUNA, 2005B] Shirasuna, M., **Ethernet industrial - Parte 5 - As organizações internacionais de Ethernet industrial**. Mecatrônica Atual, n. 21, p. 32-35, Abr. 2005
- [SILVEIRA *et al.*, 1998] Silveira, P. R.; Santos, W. E. **Automação e controle discreto** . São Paulo: Érica, 1998. 338p.
- [SIMPSON, 1994] Simpson, C. D., **Programmable logic controllers** . Englewood Cliffs: Regents/Prentice Hall, 1994. 294p.
- [STROTHMAN, 2002] Strothman, J., **PLCs' market face: looks deceiving**, InTech, Feb. 2002
Disponível em: <<http://www.isa.org/InTechTemplate.cfm?Section=InTech&template=/ContentManagement/ContentDisplay.cfm&ContentID=9459>>. Acesso em 20 de maio 2005
- [SUN, 2005] Sun, **Java technology overview**. Disponível em: <<http://java.sun.com/overview.html>>. Acesso em 20 de maio 2005
- [VAN DER WAL, 1999] van der Wal, E., **Introduction into IEC 1131-3 and PLCopen** In: IEE Colloquium on The Application of IEC 61131 to Industrial Control: Improve Your Bottom Line Through High Value Industrial Control Systems, 1999. p. 2/1-2/8
- [VERWER, 1999] Verwer, A., **The impact of IEC (6)1131-3 on the teaching of control engineering** In: IEE Colloquium on the application of IEC 61131 in industrial control: improve your bottom-line through high value industrial control systems, 1999. p. 10/1-10/4
- [VERWER, 2000] Verwer, A., **The role of process simulation in the control system software life cycle** In: IEE Colloquium on the application of IEC 61131 in industrial control: improve your bottom-line through high value industrial control systems, 2000. p. 7/1-7/6

- [VIEIRA, 2004] Vieira, S., **As novas tecnologias das IHMs**. Mecatrônica Atual, n. 17, p. 12-14, Ago. 2004
- [VIEIRA, 2005] Vieira, S., **Conheca o mercado brasileiro de CLPs**. Mecatronica Atual, n. 20, p. 12-14 Fev. 2004
- [VISUAL OBJECT, 2005] Visual Object Net, **Visual Object Net++**. Disponível em: <<http://www.systemtechnik.tu-ilmenau.de/~drath/visual.htm>> . Acesso em 22 de maio 2005
- [WILLIAMS, 1999] Williams, D., **IEC1131-3 - A systems integrator's view**. In: IEE Colloquium on the application of IEC 61131 in industrial control: improve your bottom-line through high value industrial control systems, 1999. p. 8/1-8/3
- [WIRTH, 1996] Wirth, N., **Compiler construction** . Harlow: Addison-Wesley, 1996. 176p.
- [WOMACK *et al.*, 1992] Womack, J. P.; Jones, D. T.; Roos, D., **A máquina que mudou o mundo**. São Paulo: Campus, 1992. 347p.
- [ZHOU *et al.*, 1995] Zhou, M.; Twiss, E., **A comparison of relay ladder logic programming and Petri net approach for sequential industrial control systems**. In: Proceedings of the 4th IEEE Conference on Control Applications, 1995. p. 748-753
- [ZHOU *et al.*, 1998] Zhou, M.C.; Twiss, E., **Design of industrial automated systems via relay ladder logic programming and Petri nets**. IEEE Transactions on Systems, Man and Cybernetics, Part C, v. 28, n. 1, p.137-150, Feb. 1998