

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

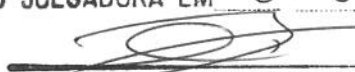
**Técnica de Especificação para Sistemas de
Manufatura
(Um Enfoque Orientado a Objeto)**

Autor : Orlando Mauricio Durán Acevedo

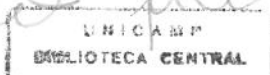
Orientador: Dr. Antonio Batocchio

01/96

ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL
DA TESE DEFENDIDA POR Orlando Mauricio
Durán Acevedo E APROVADA PELA
COMISSÃO JULGADORA EM 09 07 96


ORIENTADOR

O processo ficou na PRPG



UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE FABRICAÇÃO

Técnica de Especificação para Sistemas de Manufatura
(Um Enfoque Orientado a Objeto)

Autor : Orlando Mauricio Durán Acevedo

Orientador: Dr. Antonio Batocchio

Curso: Engenharia Mecânica.

Área de concentração: Materiais e Processos

Tese de Doutorado apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para obtenção do título de Doutor em Engenharia Mecânica.

Campinas, 1996

S.P. - Brasil

BC
UNICAMP
31t
30064
283197
0 x
R\$ 11,00
10/05/97

-00098420-3

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Ac37t

Acevedo, Orlando Mauricio Durán

Técnica de especificação para sistemas de manufatura
(um enfoque orientado a objeto) / Orlando Mauricio
Durán Acevedo.--Campinas, SP: [s.n.], 1996.

Orientador: Antonio Batocchio.

Tese (doutorado) - Universidade Estadual de Campinas,
Faculdade de Engenharia Mecânica.

1. Sistemas de fabricação integrada por computador.
 2. Programação orientada a objetos (Computação).
 3. Controle automático.
 4. Programação lógica.
- I. Batocchio, Antonio. II. Universidade Estadual de
Campinas. Faculdade de Engenharia Mecânica. III. Título.

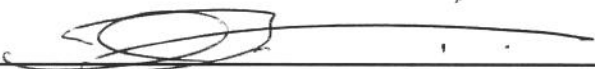
UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE FABRICAÇÃO

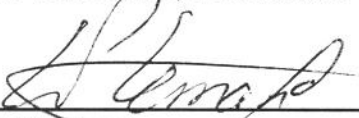
Tese de Doutorado

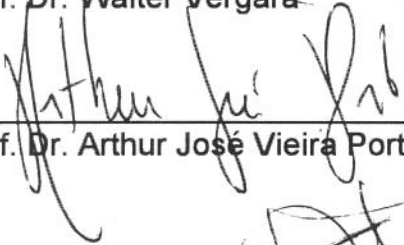
**Técnica de Especificação para Sistemas de
Manufatura
(Um enfoque orientado a Objeto)**

Autor : Orlando Mauricio Durán Acevedo

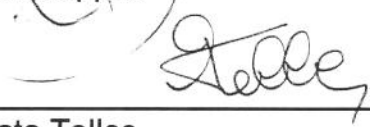
Orientador: Dr. Antonio Batocchio


Prof. Dr. Antonio Batocchio, Presidente


Prof. Dr. Walter Vergara


Prof. Dr. Arthur José Vieira Porto


Prof. Dr. Nivaldo Lemos Coppini


Prof. Dr. Geraldo Nonato Telles

Campinas, 9 de julho de 1996

Dedicatória:

Dedico mi esfuerzo a mis padres,
Orlando y Pilar
y a mis hermanos ,
Paulo y Pily

Agradecimentos

A Deus, Ele sabe porquê.

Ao Professor Doutor, Antonio Batocchio, pela orientação, apoio e por sobretudo pela amizade sem restrições.

Ao Professor Doutor Nelson Durán, meu tio, exemplo de amor pela pesquisa e vocação de ensino, obrigado!

À Sheiloca, pois sem a sua ajuda, força, fê e confiança, desde o primeiro dia, este trabalho e outras empreitadas teriam parecido o dobro de difíceis (e quensabe, inatingíveis)

À minha família (aqui e na distância) pela confiança depositada em mim, pelo seu apoio incondicional e amor que sempre me brindaram durante estes anos.

A todos meus amigos e colegas do DEMA e do DEF, especialmente a Sandro, cuja sensível amizade significou muito durante este tempo.

À Fapesp pelo apoio financeiro

Algo só é impossível
até que alguém duvida
e acaba provando o contrário.

Einstein

Sumário

Introdução	1
1.1. Justificativas:	2
1.2. Objetivos:	5
1.3. Conteúdo do trabalho:	6
Marco Teórico	7
2.1. Técnicas de especificação para sistemas de manufatura	7
2.1.1. Structured Analysis and Design Technique (SADT):	11
2.1.2. Redes de Petri (PN- Petri Nets) :	13
2.1.3. O uso de Linguagem Natural:	18
2.1.5. Exemplo de Aplicação:	24
2.1.6.Comentários sobre as Técnicas Analisadas:	30
2.2. Técnicas de Representação de Linguagens	31
2.3. Gramáticas:	35
2.3.1. Expressão de CFGs na forma de Cláusulas	39
2.4. Redes de Transição Aumentadas (ATN - Augmented Transition Networks):	43
2.4.1. Aplicação das ATNs:	48
2.5. Comparação entre as ATNs e as DCGs:	50
2.6. Aplicações e outros métodos:	52
O Paradigma de Objeto	60
3.1. Conceitos do Paradigma de Objeto:	60
3.1.1.Conceitos:	62
3.2. Contextualizando os Processos :	65
3.3. Especificação de Sistemas de Manufatura Discreta e Gramáticas de Casos	67

3.4. <i>Relação entre os Referenciais de Casos e o Paradigma de Objeto:</i>	70
3.5. <i>O Paradigma de Objeto aplicado à Especificação de Sistemas de Manufatura:</i>	72
3.5.1. <i>A Especificação de Sistemas de Manufatura:</i>	73
Técnica de Especificação para Sistemas de Manufatura (um enfoque orientado a objeto)	75
4.1. <i>Implementação da hierarquia usando Prolog:</i>	78
4.2. <i>Gramática da Técnica de Especificação:</i>	84
4.3. <i>Comentários Finais</i>	94
Aplicação : Um sistema de Geração Automática de Programas para CLPs	95
5.1. <i>Os Controladores Lógicos Programáveis:</i>	95
5.2. <i>Paradigmas de Programação de CLPs:</i>	99
5.2.1. <i>Diagramas de Contatos (Ladder Diagrams):</i>	99
5.2.2. <i>Listas de Instruções:</i>	100
5.2.3. <i>Textos Estruturados (Structured Text - ST):</i>	100
5.2.4. <i>Sequential Function Charts (SFC) :</i>	100
5.2.5. <i>Function Block Diagrams:</i>	100
5.3. <i>Caracterização das operações padrões realizadas por um CLP:</i>	102
5.3.1. <i>Instruções Básicas:</i>	105
5.3.2. <i>Outras Instruções:</i>	107
5.4. <i>Regras para a eleição do conjunto de instruções da Técnica de Especificação:</i>	109
5.5. <i>Linguagens de Programação para CLPs e a sua representação usando DCGs:</i>	110
5.6. <i>Estrutura do Sistema:</i>	113
5.6.1. <i>O Gerador Automático de Software:</i>	114
5.6.2. <i>Um exemplo simples do uso do Gerador:</i>	115
5.7. <i>Testes para Interfaceamento com uma interface em Windows:</i>	118
5.8. <i>O sistema em operação:</i>	120
5.8.1. <i>Editor de Especificações:</i>	121
5.8.2. <i>O Object Browser:</i>	123
5.8.3. <i>Gerador Automático de Software:</i>	129

Conclusões	132
Referências Bibliográficas	134

Resumo

DURÁN ACEVEDO, Orlando Mauricio, *Técnica de Especificação para Sistemas de Manufatura (Um enfoque orientado a objeto)*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 172 p. Tese (Doutorado)

Este trabalho apresenta a criação de uma Técnica de Especificação para sistemas de manufatura baseado no paradigma de orientação a objeto (TEOO). Através desta técnica podem ser descritos a estrutura dos sistemas e os seus componentes como objetos membros de uma hierarquia. Como aplicação, foi desenvolvido um sistema gerador automático de software para controladores lógicos programáveis (CLPs) que utiliza como entrada descrições de lógicas de controle para sistemas de manufatura discreta escritas usando a TEOO e gera como saída um programa no formato de um CLP específico.

Palavras Chave

- Automação da Manufatura, Geração Automática de Software, Controladores Lógicos Programáveis.

Abstract

DURÁN ACEVEDO, Orlando Mauricio, *Técnica de Especificação para Sistemas de Manufatura (Um enfoque orientado a objeto)*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 172 p. Tese (Doutorado)

In this work the definition of a manufacturing systems specification technique is reported. This technique is based on the object oriented paradigm. Through the use of this technique system structure and components can be described. These components are members of a user defined object hierarchy. As an application of the technique an automatic PLC software generator was developed (PLC Programmable Logic Controller). This system uses as an input textual descriptions of the control logic that will be implemented in a system. The textual description is written using the specification technique and generates automatically a source program ready for downloading in a selected PLC.

Keywords

Manufacturing Automation, Automatic Software Generation, Programmable Logic Controllers.

Lista de Figuras

Fig. 2.1.a. Enfoque tradicional nos projetos de Sistemas de Controle	8
Fig.2.1.b. Enfoque simultâneo no projeto de sistemas de controle	8
Fig.2.2.a.Unidade Básica do enfoque SADT	12
Fig.2.2.b. Relação entre níveis (layers) das atividades	13
Fig.2.3.a. Exemplo de uma rede de Petri básica	15
Fig.2.4. Sistema de manufatura exemplo	25
Fig.2.5.Primeiro nível para um enfoque SADT	25
Fig.2.6. Subdivisão da atividade “Manufaturar Componentes”	26
Fig.2.7. Desdobramento da atividade “Tornear Peça”	27
Fig.2.8. Rede de Petri representando o funcionamento do sistema descrito	27
Fig.2.9. Diagrama de Fluxo de Dados relativo à célula exemplo	29
figura 2.10. Representação da sequência de compilação de um programa	35
Figura 2.11. Representação de uma Máquina de Estados exemplo	44
Figura 2.12.Exemplo de uso de Sub redes	44
Figura 2.13.Exemplo de representação gráfica para uma ATN	47
Figura 2.14. Representação gráfica dos comandos: seq, either, optional e optional*	47
Figura 3.1. O usuário pode montar as suas aplicações a partir de um conjunto de componentes adquiridos de diferentes fornecedores.	61
Figura 3.2. O conceito de abstração. O programador visualiza uma entidade do mundo real e logo cria uma representação simbólica no espaço solução. Esta representação é chamada de "objeto".	62
Figura 3.3. O conceito de Abstração usado para representar uma máquina ferramenta.	63
Figura 3.4.- O Conceito de Encapsulamento. Existe uma muralha rodeando cada objeto. O usuário apenas especifica o quê ele deseja que o objeto faça através do envio de uma mensagem. A forma como o objeto realiza a ação desejada, ficará oculta numa parte privada do objeto.	64
Figura 3.5. - O conceito de Herança. Uma classe primária origina diferentes sub-classes. Estas subclasses podem herdar algumas características das superclasses e, adicionalmente, incorporar outras características próprias.	65
Figura 3.6. Divisão dos processos	66
Figura 4.1. Esquema dos diversos níveis de controle para a manufatura	75
Figura 4.2. Hierarquia de Objetos para os dispositivos de Manufatura.	77
Fig.5.1. Um exemplo genérico das aplicações baseadas em CLPs (extraído de BONANOMI, 1994)	96
Figura 5.2. Funções de um CLP e os meios de ligação ao sistema automatizado (adaptado de BITTAR, 1993)	96
Figura 5.3. Conceito de Modularidade do Sistema - Partes que dependem do CLP selecionado.	99

Figura 5.4. Exemplos de programação de CLPs usando formatos gráficos a) diagramas de contato b) Sequential Function Charts c)Function Block Diagrams	101
Figura 5.5. Cronograma das iniciativas de padronização para linguagens de programação de CLPs (CHRISTENSEN, 1993)	102
Figura 5.6. Representação das operações básicas realizadas por um CLP	109
Figura 5.7. Uso relativo das instruções (por tipo) nos programas para CLPs	110
Figura 5.8. Estrutura do Sistema	113
Figura 5.9. Exemplo de sistema de manuseio de materiais	116
Figura 5.10. Histogramas de tempos para as instruções geradas.	118
Figura 5.11. Estrutura e relacionamento do sistema que interfaceia Visual Basic e Arity Prolog	119
Figura 5.12. Estrutura do Sistema Gerador Automático de Software para CLPs.	120
Figura 5.13. Tela inicial do sistema mostrando todas as opções do menu principal	121
Figura 5.14. Tela do Editor de Especificações usando TEOO.	122
Figura 5.15. Caixa de diálogo da opção Abrir Arquivo do editor	122
Figura 5.16. Caixa de Diálogo da opção Seek, para procurar arquivos.	123
Figura 5.17. Opções implementadas no Object Browser.	124
Figura 5.18. Caixa de diálogo para definir um novo objeto.	125
Figura 5.19. Caixa de diálogo que permite apagar uma instância de objetos.	126
Figura 5.18. Caixa de diálogo para adicionar um novo método a uma classe existente.	127
Figura 5.19. Caixa de diálogo para remover um método existente pertencente a uma classe.	128
Figura 5.20. Caixa de diálogo para inserção de uma novo classe.	128
Figura 5.21. caixa de diálogo para remover uma classe existente.	129
Figura 5.22. Menu de opções para o gerador de código de CLPs.	129
Figura 5.23. Caixa de diálogo para definir o esquema de fiação do CLP escolhido	130
Figura 5.24. Caixa de diálogo para selecionar o programa a ser gerado.	131

Lista de Tabelas

TABELA 2.1. - Detalhe dos Locais da Rede de Petri	28
TABELA 2.2.-Detalhamento das Transições.	28
TABELA 2.3.- Operações sofridas e requeridas pelos objetos	30
Tabela 4.1. Identificação de instâncias de objeto <i>sensor</i> e os seus métodos	78
Tabela 4.2. Identificação de instâncias de objeto <i>atuador</i> e os seus métodos	79
Tabela 5.1.- Comparação entre a automação programável e outros métodos de automação	97

Capítulo 1

Introdução

O processo de definição e modelagem de um controlador de sistemas de manufatura requer um grande esforço por parte dos técnicos que são responsáveis por esta tarefa. Primeiro, devido às dificuldades que eles encontram ao comunicarem suas ideias e modelos ao computador onde são desenvolvidos os modelos e, segundo, devido às dificuldades de se comunicarem com outros técnicos. Isto é especialmente comum quando existem num projeto diversas pessoas com diversos tipos de preparação e conhecimentos em áreas diferentes.

Boucher e Jafari (1992) salientam as dificuldades existentes na transição entre o projeto do sistema de manufatura e o projeto do sistema de controle. Estas dificuldades, segundo o trabalho antes mencionado, é devido à falta de ferramentas padronizadas que possibilitem a tradução dos requerimentos de controle dos sistemas de manufatura para o sistema de controle. Assim, é primordial a existência de ferramentas que possibilitem a comunicação efetiva, de modo a que os técnicos que participam deste processo possam trocar ideias e discutir diversas possibilidades para um determinado projeto. Conseqüentemente, de posse de uma ferramenta deste tipo, a produtividade dos especialistas será incrementada substancialmente.

O processo de especificação e projeto de um controlador de sistema de manufatura automatizado, inicia-se com a determinação de um conjunto de requerimentos (elementos, atributos e comportamento que o sistema deverá ter), e se completa quando o sistema é suficientemente detalhado, de modo a ter uma implementação facilitada. Silva e Miyagi (1995) detalham as etapas necessárias para a implementação bem sucedida de sistemas de controle para sistemas de manufatura discreta.

Do ponto de vista do controle sequencial de sistemas de manufatura, os controladores lógicos programáveis (CLPs) se apresentam como uma ferramenta adequada e confiável para implementação de ditos controladores. Apesar de que os CLPs não possibilitam a tomada de decisões mais avançadas, eles continuam a ser um meio efetivo para realizar as tarefas de controle onde a lógica sequencial é requerida (Lee e Sen, 1994).

A programação dos CLPs é ainda uma tarefa considerada complexa e suscetível de erros. Além disto, os programas desenvolvidos são difíceis de manter e alterar, provocando dificuldades do ponto de vista da flexibilidade e rápida capacidade de reconfiguração dos sistemas de manufatura.

Objetivando a construção de um sistema que permita a rápida prototipação de novos controladores de sistemas de manufatura ou a modificação de sistemas já existentes, este trabalho propõe o desenvolvimento de um ambiente geral assistido por computador para a descrição de controladores para sistemas automatizados de manufatura discreta e a posterior geração de código para CLPs. Este ambiente baseia-se na utilização de uma Técnica de Especificação Orientada a Objeto (TEOO) cuja gramática foi desenvolvida incorporando técnicas destinadas ao tratamento de linguagem natural. O gerador automático de programas também está baseado na mesma estrutura (Definitive Clause Grammar) e implementado em Arity Prolog.

Através das descrições textuais escritas usando a TEOO, são representadas tanto a estrutura como o funcionamento de sistemas automatizados de manufatura (lógica de controle), para posteriormente traduzi-las em código fonte de um determinado CLP. Com isto pretende-se facilitar a criação de novos programas e permitir o reaproveitamento dos já existentes.

A seguir são enunciados os objetivos pretendidos e as justificativas para a elaboração deste trabalho.

1.1. Justificativas:

O propósito de toda técnica de especificação é levar aos projetistas por um processo formal de descrição de requerimentos para transformá-los em construções que possam ser facilmente implementadas. Assim, a partir de uma boa metodologia de especificação ou modelagem é possível facilitar o processo completo, desde a definição inicial, até mesmo a geração de código do controlador do sistema. Este fator torna-se importante considerando-se a flexibilidade dos sistemas de manufatura como um fator crítico de sucesso para atingir a eficiência e produtividade. Segundo Rocha e Ramos (1994) a ênfase deve-se focar a versatilidade e inteligência dos sistemas, privilegiando o desenvolvimento de sistemas automáticos de planejamento e programação.

Segundo Lee e Sen (1994) é altamente desejável que o modelamento de um sistema de controle para sistemas de manufatura seja feito a partir de módulos reutilizáveis que possam ser facilmente modificados e mantidos, segundo as alterações no sistema. Esta estratégia de modelamento deriva naturalmente na utilização do paradigma de orientação a objeto. Ainda, os mesmos autores salientam que este paradigma fornece um método unificado, consistente e eficiente para a representação de conhecimento e para o modelamento de sistemas de manufatura. Os autores argumentam que existe uma correspondência um-a-um entre os objetos “físicos” e os objetos do software que os representam. A característica principal da técnica que este trabalho propõe é o uso do paradigma de objeto para realizar a modelagem dos sistemas. O uso desta técnica proporciona um claro ponto de partida na definição de modelos, através do mapeamento direto dos objetos do mundo real em objetos internos do modelo. Este conceito se faz facilmente aplicável ao domínio dos sistemas de manufatura pelas seguintes razões :

- A função de um controlador de sistema automatizado de manufatura, é exercer controle sobre um número de dispositivos (cooperativos) que produzem uma ação desejada. Estes dispositivos podem ser facilmente “traduzidos” a objetos abstratos do modelo de definição (Lee e Sen, 1994).
- Devido ao fato de muitos destes dispositivos compartilharem características similares, por exemplo, robôs, máquinas ferramentas de comando numérico, dispositivos de

medição, controladores programáveis, etc., estes podem ser facilmente agrupados em classes de objetos (ou seja, cada um deles pode ser indicado como uma instância de uma classe de objeto). Este conceito é facilmente suportado por modelos orientados-a-objeto, permitindo a reutilização de uma dada especificação ou parte dela (Barata et al., 1995).

- Na realidade, muitos dispositivos são reutilizados em outros sistemas ou rearranjados dentro do mesmo sistema. Através do paradigma de objeto isto é simplificado com a incorporação de bibliotecas de objetos. Segundo Ho-Sang Ham et al. (1996), uma das principais vantagens das metodologias orientadas a objeto é que elas suportam a reutilização de modelos. A reutilização de componentes é conseguida através da criação de objetos baseados em atributos e operações pré-definidos e que são herdados de uma classe (conceito de herança). Assim será necessário especificar apenas de que forma o novo objeto difere da classe, ao invés de definir todas as características do novo objeto. Estes conceitos são vastamente discutidos por Aguiar et al. (1996).

Do ponto de vista do desenvolvimento de software para CLPs, French (1990) detetou a necessidade de aplicar uma metodologia estruturada no desenvolvimento de software para CLPs, e com isto ganhar em consistência e confiabilidade, facilitando o processo de programação e alteração dos programas para ditos dispositivos.

Bonanomi (1994), reconhece a dificuldade na manutenção de programas escritos para um CLP e a quase total impossibilidade de reaproveitamento de código em situações de manufatura similares às consideradas no desenvolvimento de um programa dado. Por outro lado, Witzerman e Nof (1995) salientam que os processos de carregamento, teste, ajustagem e correção das lógicas de controle resultam em quantidades significativas de tempo improdutivo para um sistema de manufatura. Ainda mais, quando a integração no nível de hardware é atingida, um erro de programa ou lógicas de controle improprias pode trazer resultados catastróficos para o sistema.

Recentemente Hyunbo e Mooyoung (1996), destacaram a conveniência de adotar a filosofia orientada a componentes de software (uma simplificação do paradigma de objeto) para o desenvolvimento de sistemas de controle para sistemas de manufatura discreta. Assim, ainda existe a necessidade de criar meios e ferramentas que facilitem tanto o desenvolvimento de programas de controle, como também o seu reaproveitamento e

manunção. Isto acelerará o lançamento de soluções de acordo com os novos cenários que os sistemas de manufatura enfrentam, mantendo a confiabilidade e segurança.

1.2. Objetivos:

O objetivo principal deste projeto é criar uma técnica textual de especificação orientada-a-objeto (baseada na língua portuguesa), através da qual seja possível executar a implantação real de um controlador para sistema de manufatura.

A técnica textual de especificação para controladores de sistemas automatizados de manufatura, será validada através da criação do módulo de geração automática de software para controladores lógicos programáveis (CLPs). Este módulo transferirá as especificações desde o formato orientado-a-objeto para o formato específico de um dispositivo de controle escolhido. Isto é, será gerado código fonte a partir da representação textual da lógica de controle do processo especificado pelo usuário. Este módulo terá como entrada a descrição do sistema (estrutura e comportamento) escrita em TEOO e, em seguida, o processamento fornecerá o programa no código específico do dispositivo ou controlador que está sendo definido. Outros objetivos parciais são descritos a seguir:

- Identificação das Classes de Objetos possíveis de existir numa situação da manufatura discreta automatizada. Assim por exemplo, podem ser identificadas classes de objetos que representem o robô, os dispositivos de fixação, os alimentadores, dispositivos de medição, etc.
- Desenvolvimento de um ambiente interativo (usando WINDOWS) para apoiar o usuário na definição da estrutura do sistema e das tarefas específicas que vão ser realizadas em dito sistema. Nela, através do uso de menus e ícones, poderão ser identificados todos os elementos que participam da aplicação e os seus atributos e relações, para, a partir desta identificação realizar a geração automática de software para controladores lógicos programáveis.
- Obtenção de um mecanismo textual e orientado a objeto para a definição de controladores para sistemas de manufatura automatizada, seus componentes e a interação entre eles.

- Definir um ambiente que suporte efetivamente a geração automática do software para cada um dispositivos que participam do controle de uma dada configuração.

Além disso, esta técnica pode ser considerada como uma linguagem de programação de alto nível que isolará os projetistas das características específicas do hardware na definição das tarefas para cada um dos dispositivos automatizados. Através das especificações de sistemas, a técnica pretende se converter em uma poderosa ferramenta de representação de conhecimento podendo representar em forma estruturada todas as limitações e possibilidades tecnológicas de cada um dos sistemas projetados.

1.3. Conteúdo do trabalho:

O trabalho está estruturado em seis capítulos mais um anexo. No capítulo número dois é feita uma revisão bibliográfica de tópicos relevantes para a realização do projeto. O terceiro capítulo encarrega-se do estudo detalhado dos conceitos do paradigma de objetos e das gramáticas de casos, e a sua relação na especificação de processos. O capítulo quatro, traz os detalhes da criação da técnica de especificação para sistemas de manufatura discreta. Nele é relatada a criação de uma hierarquia de objetos pertencentes ao domínio da manufatura discreta, e a implementação de dita hierarquia usando a linguagem de programação Prolog. Na parte final deste capítulo é discutida a definição da gramática da técnica. No capítulo 5, é mostrada a aplicação desta técnica na geração automática de programas para controladores lógicos programáveis. Junto com isto, é feito um pequeno exemplos da operação do sistema de geração automática. No capítulo final, o número 6, são enunciadas algumas conclusões e feitas algumas projeções para trabalhos futuros. No anexo, são representados quadros resumos sobre as possibilidades de construção de sentenças que a gramática tem, além de uma lista de todas as publicações que este trabalho gerou.

Capítulo 2

Marco Teórico

Este capítulo foi estruturado em duas partes principais, a primeira parte se dedica ao estudo de algumas técnicas de especificação para sistemas de manufatura, e a segunda parte estuda as duas principais metodologias para representar linguagens de programação e gerais, com o objetivo de modelar as relações sintáticas e semânticas existentes numa determinada linguagem.

2.1. Técnicas de especificação para sistemas de manufatura

À medida que os sistemas de manufatura têm começado a dar passagem à utilização de novas tecnologias e à automação, para automatizar as operações e flexibilizar os processos, a etapa de definição e projeto destes sistemas tem se tornado evidentemente mais importante.

Ainda mais, a adição da capacidade de decisão dos sistemas, identificada com a definição e implantação do software de controle, em alguns sistemas, fazem com que esta etapa se torne crucial para o sucesso na implantação e operação dos sistemas.

Quando as máquinas e sistemas de manipulação de materiais eram individuais ou pouco integrados, o processo de projeto de um novo sistema de manufatura podia ocorrer independente da tarefa de definição da lógica de controle. Assim, o projeto de sistemas de manufatura, usualmente no campo dos Engenheiros Industriais ou de Manufatura, estava mais voltado a fatores tais como Organização do Chão de Fábrica, Necessidade de Inventários Intermediários, Tempos de Preparação, Capacidade, etc. Por outro lado, a definição da lógica de controle - campo dos Engenheiros Elétricos e Mecânicos - se

concentrava em itens tais como Comunicação, Controladores Lógicos Programáveis, Sequenciamento, Recuperação de Erros,etc.

Hoje em dia, devido ao maior uso de controle automático, integração de processos e à adição de uma maior capacidade de decisão própria dos sistemas, os dois processos transcorrem paralelamente. Existe então, uma necessidade de alta integração entre estes dois campos. E desta integração, conseqüentemente, surge a necessidade de uma comunicação mais fluente entre todos os especialistas que participam de um projeto de sistema de manufatura (veja figuras 2.1.a e 2.1.b.).

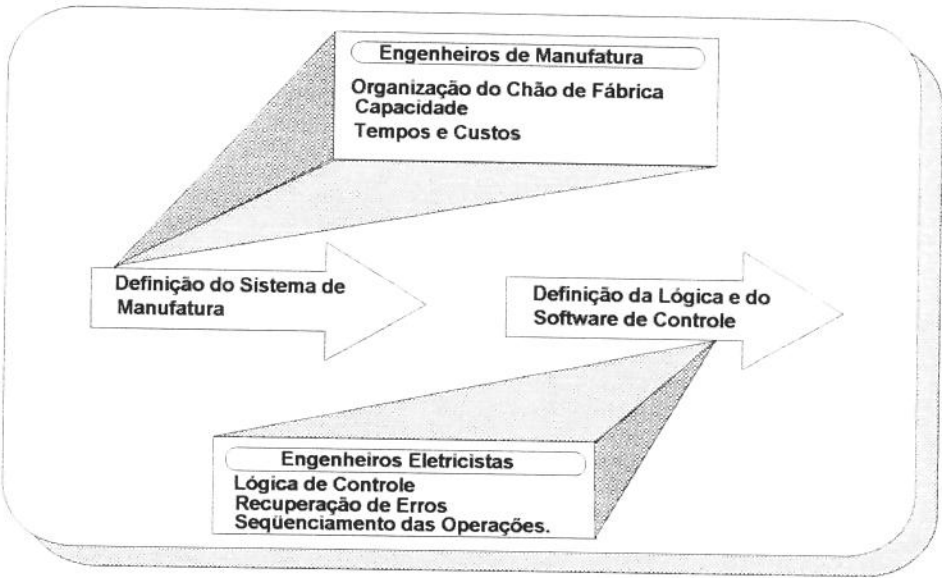


Fig. 2.1.a. Enfoque tradicional nos projetos de Sistemas de Controle

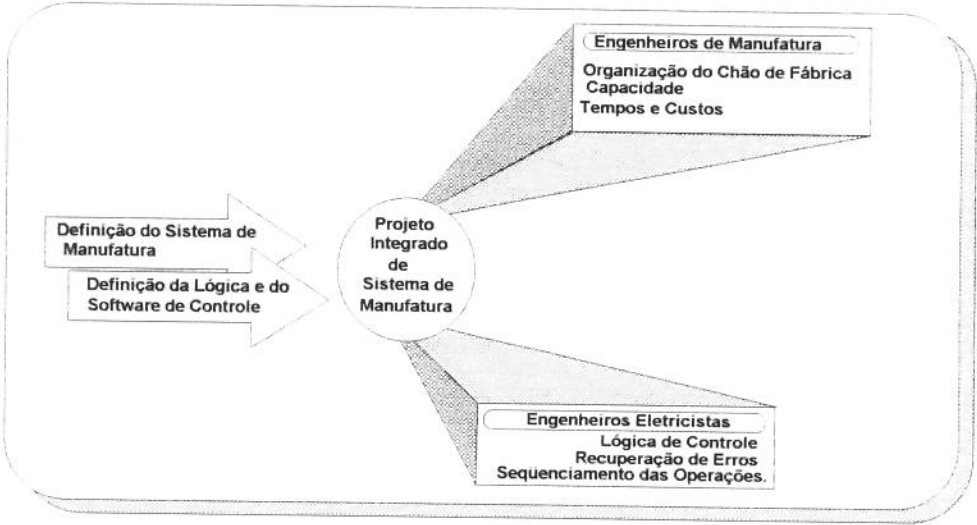


Fig.2.1.b. Enfoque simultâneo no projeto de sistemas de controle

Esta integração se ve dificultada, principalmente devido ao fato de estes especialistas entenderem e usarem diferentes linguagens e técnicas de representação dos seus modelos. Mesmo nestas áreas de conhecimento encontramos diversas técnicas que tentam representar, de maneira acurada e resumida, as configurações dos sistemas desenvolvidos. Algumas destas técnicas estão altamente difundidas e são de conhecimento de um grande número de profissionais. Porém, o outro objetivo importante perseguido, ou seja, a sua abrangência de aplicação, às vezes é sub-atingido. Assim, é comum encontrar diagramas de controle escritos em Diagramas de Relés ou de Contatos (RLL- *Relay Ladder Logic*), Diagramas Lógicos ou Diagramas de Estado. Já em projetos de Sistemas de Manufatura, as ferramentas são menos estruturadas e difundidas. Existem contadas exceções, tais como os Diagramas Pneumáticos, e as Redes de Petri (PN - *Petri Nets*), usadas principalmente na definição de modelos de simulação. Mais adiante voltar-se-á às PN mais detalhadamente e comentar-se-á o seu uso como ferramenta formal de especificação nos projetos de sistemas de manufatura automatizados.

Como pode-se supor, a integração e o trabalho conjunto entre estes profissionais técnicos é difícil, para não dizer quase impossível. Esta desfasagem entre as duas etapas traz consigo uma alta taxa de erros e problemas.

A importância da etapa de definição dos sistemas de manufatura e da sua lógica de controle é manifestada com as estatísticas reportadas por DIMITROV E TODOROV (1988). Elas salientam a necessidade de um esforço importante durante a etapa de definição de sistemas de manufatura automatizados, e do software de controle. Assim, 2/3 dos custos incorridos num novo sistema de manufatura flexível são identificados durante a etapa de implantação. Estes custos se devem, principalmente, a erros na etapa de projeto do software de controle. As despesas para solucionar os problemas são maiores quanto mais tarde estes são descobertos durante a etapa de implantação. Assim, apenas 19% destes erros são descobertos na fase de desenvolvimento e cerca de 54% dos erros são localizados durante a operação do sistema.

JOSHI, METTALA e WYSK (1992) salientam que os dois caminhos indicados para reduzir este alto custo devido a erros na etapa de definição e escrita do software de controle são:

- A geração automática de Software
- O uso de componentes reutilisáveis de Software.

Estes dois fatores estão intimamente ligados ao uso de ferramentas formais de especificação. Nos últimos anos estas ferramentas formais de especificação têm sido incorporadas ao ambiente "Computer-Aided" (HAREL E LACHOVER, 1990), formando o conceito conhecido hoje em dia como CASE (Computer Aided Software Engineering). Esta técnica apareceu em cena no começo da década de 80 junto com a introdução de pacotes de documentação para aplicativos e ferramentas de diagramação dos processos.

O avanço destas ferramentas CASE tem sido contínuo e está próximo da sua meta final, a geração completa de software a partir de um conjunto integrado de ferramentas para dito objetivo (ROBERTS E BEAUMARIAGE, 1993; RICH, 1988).

Como foi dito no parágrafo, anterior a importância de ferramentas de especificação formais fica em evidência. Pois, o uso de uma técnica de especificação formal se constitui na interface entre o *Projeto de Sistemas de Manufatura* e o *Desenvolvimento do Software de Controle*. O "gap" entre estas duas tarefas é cada dia mais estreito, considerando o nível crescente de automação existente e a maior necessidade de flexibilidade nas instalações atuais de manufatura.

Neste contexto, a maioria destas técnicas constitui-se em um meio adequado à comunicação entre os diversos especialistas participantes de um projeto de um Sistema de Manufatura.

Ocasionalmente estas técnicas são usadas para estruturar os requerimentos do novo sistema. Isto é, o especialista através de uma conversação informal com o cliente será capaz de traduzir esta descrição pouco estruturada em um modelo (HILAL E SOLTAN, 1992). Através deste modelo serão representados fielmente os requisitos de operação do novo sistema.

Apesar de algumas destas técnicas serem provenientes de outras áreas de conhecimento, tais como a Engenharia de Software ou da área de Organização e Métodos e Engenharia de Sistemas, elas têm demonstrado excelentes resultados e importantes ganhos.

Neste capítulo relata-se o estudo de técnicas estruturadas para o Projeto de Sistemas de Manufatura. Estas técnicas foram criadas com diferentes objetivos - simulação, simples representação esquemática, validação, etc, mas, o seu uso inicial como ferramentas puras de definição, fazem classificá-las como *Técnicas Formais de Especificação de Sistemas de Manufatura*.

Apresenta-se uma revisão de técnicas já conhecidas, como as Redes de Petri, e a técnica Structured Analysis and Design Technique (SADT), amplamente usadas na definição de Sistemas de Manufatura. Além destas duas técnicas será analisada uma outra técnica baseada na Linguagem Natural para a descrição de Modelos de Simulação, a qual define uma linha de pesquisa inovadora, no sentido de facilitar a interface com o usuário na definição destes modelos.

Aquí no Brasil existem algumas iniciativas destinadas à descrição de Sistemas Automatizados de Manufatura. Entre elas podem ser destacadas o uso de Redes de Petri para a definição de Gerenciadores de Células (trabalho realizado pelo GRUCON na Universidade Federal de Santa Catarina), os chamados Mark Flow Graphs (destinados à definição de controladores seqüenciais, iniciativa do grupo de Mecatrônica da Universidade de São Paulo), e os State Charts (usados como ferramenta de definição de Sistemas de Manufatura no Laboratório de Máquinas Ferramentas da Escola de Engenharia de São Carlos (USP)). Estas três metodologias serão também descritas e comentadas no decorrer deste capítulo.

2.1.1. Structured Analysis and Design Technique (SADT):

Esta técnica foi criada por D.J.ROSS em 1977 com o intuito de criar modelos de Arquiteturas para Sistemas Integrados por Computador (CIM). Mais recentemente, a Força Aérea Americana desenvolveu uma técnica baseada na SADT, chamada *Integrated Computer Aided Manufacturing Definition 0 (IDEF0)*. Ambas as técnicas destinam-se à definição e documentação de sistemas hierárquicos, e com estrutura modular. Este enfoque permite a abstração do sistema em diferentes camadas, cada uma contendo maior detalhe do que os elementos da camada superior.

A unidade básica desta técnica é a chamada *Activity Box*, neste trabalho esta será chamada simplesmente como *Processo* (figura 2.2.a.).

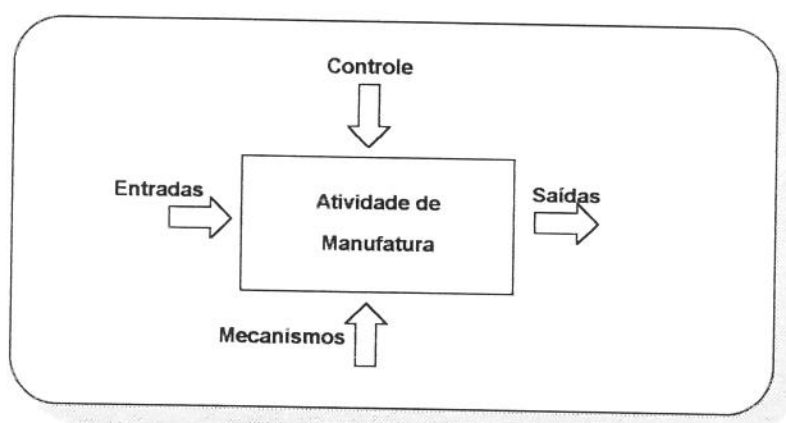


Fig.2.2.a.Unidade Básica do enfoque SADT

O *processo* define uma atividade da organização a qual está sendo modelada. Esta atividade pode ser uma Tomada de Decisão ou uma atividade de Conversão de Material. As *entradas* são colocadas no lado esquerdo da caixa. Estas entradas podem ser itens de material ou itens de informação, os quais serão transformados pelo processo. As *saídas* são colocadas no lado direito da caixa e representam o resultado do processo de transformação sobre os itens de entrada. O *controle* é localizado entrando na parte superior do processo e representa as condições que governam o comportamento da atividade. Por exemplo, o controle pode ser um conjunto de regras comandando o processo ou uma condição que deve existir antes da atividade possa ser realizada. O *mecanismo* é representado pela flecha entrando no lado inferior da caixa. O mecanismo representa o meio pelo qual a atividade é executada, por exemplo, um mecanismo pode ser uma máquina, um conjunto delas ou um operário.

Os processos representam ações sendo executadas e são rotuladas com o uso de verbos ou predicados. O restante dos elementos desta técnica são objetos e são representados por substantivos.

Esta técnica é aplicada na forma TOP-DOWN, ou seja existe uma decomposição hierárquica, onde cada elemento pode ser expandido ou desdobrado em uma camada (*layer*) com um nível de detalhe e complexidade maior. A figura 2.2.b. representa estes conceitos.

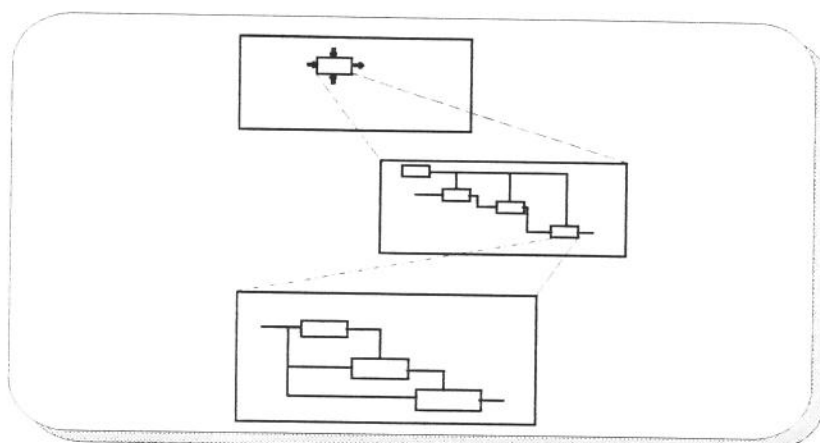


Fig.2.2.b. Relação entre níveis (layers) das atividades

No topo da hierarquia se encontra o propósito global do modelo, chamado de processo global, o qual está sendo sujeito a uma modelagem. O processo global é decomposto num conjunto de componentes que, por sua vez podem ser decompostos em novos processos que, em conjunto, compõem o processo pai. Este processo de decomposição continua até um grau de detalhe suficiente para o objetivo do projetista.

Esta técnica tem sido usada amplamente para o projeto de sistemas de informação para o nível de chão de fábrica e para as funções de controle (BOUCHER e JAFARI, 1992). Alguns incrementos foram feitos nesta técnica para incorporar melhores ferramentas de representação para situações frequentes em sistemas dinâmicos (paralelismo, conflito, etc.) assim nasceram as versões IDEF0, IDEF1 e IDEF2 (ANG et al., 1995).

Na seção final deste capítulo será apresentado um exemplo de aplicação desta técnica, quando serão comparadas todas as ferramentas de especificação consideradas, através do uso de um exemplo de Sistema de Manufatura.

2.1.2. Redes de Petri (PN- Petri Nets) :

As Redes de Petri foram criadas por Petri na sua Tese de Doutorado em 1962 onde ele formulou uma teoria de comunicação entre componentes assíncronos de um sistema computacional. Desde esta época até hoje as PN têm sido aplicadas para uma imensa gama de situações, como também têm sofrido uma grande quantidade de modificações e complementos.

Diversos grupos de pesquisa do mundo todo têm estudado a representação de processos automatizados através do uso de PN. Na França, por exemplo, foi proposto um padrão para a especificação do controle seqüencial em sistemas de manufatura. Este padrão é conhecido como GRAFCET (FRACHET, 1993), e é considerado como um derivado das PN.

No Japão, a HITACHI desenvolveu um controlador de estação de trabalho, chamado de SCR, baseado numa variação das PN denominada Redes de Petri Ampliadas. E assim por diante, a lista de pesquisadores desenvolvendo aplicações e variações para as PN é enorme.

A representação da PN sob a forma de um grafo, também chamada de parte de controle da especificação de um sistema discreto, possui grande poder de expressão, simplificando a comunicação entre o projetista e o futuro usuário. As noções de sequencialidade, paralelismo, conflito e sincronismo são expressas facilmente através de PN. O formalismo matemático associado às PN permite a validação de uma especificação, por meio de diversas abordagens.

Formalismo matemático das Redes de Petri:

Uma descrição matemática de uma PN é a seguinte:

$$R = \langle L, T, \alpha, \beta \rangle$$

onde :

L = Conjunto finito e não-vazio de Estados do sistema.

T = Conjunto finito e não-vazio de Transições ou condições do Sistema

α = Função de Incidência Prêvia para a Transição t

β = Função de Incidência Posterior para a Transição t

de modo que $\alpha(t)$ define o conjunto de estados de entrada e $\beta(t)$ define o conjunto de estados de saída. A transição t representa um evento que pode ocorrer. Os estados representam as condições que são requeridas antes que o evento ocorra. Por exemplo, uma transição pode representar o começo de uma operação e carga de uma máquina, e os estados podem representar a existência de uma peça a ser carregada e a disponibilidade do

mecanismo para carregar tal peça na determinada máquina. Só quando ambas as condições estiverem cumpridas é que o evento (transição) de carga poderá começar.

Para uma PN ter sentido na modelagem de sistemas físicos, esta deve ser provida de interpretação. Isto é, cada um dos parâmetros e a evolução através da rede devem ter significado. Na sequência é dado um pequeno e simples exemplo do uso das PN na definição de um Sistema Discreto Sequencial, com o objetivo de esclarecer os conceitos básicos desta teoria e a técnica gráfica de representação (QUIROZ, 1993). Mais adiante, no final do capítulo, será analisado um exemplo com maior detalhe. Junto com isto será comparada a aplicabilidade das PN com as outras técnicas estudadas no decorrer deste trabalho.

O sistema da Figura 2.3.a. mostra uma Prensa simples modelada através de Redes de Petri. A interpretação é a seguinte:

L1 : Estado Inicial

L2 : Estado de Descida

L3 : Estado de Subida

t1 : Recebe o sinal "liga"

t2 : Recebe o sinal "Prensa na Posição Inferior"

t3 : Recebe o sinal "Prensa na Posição Superior"

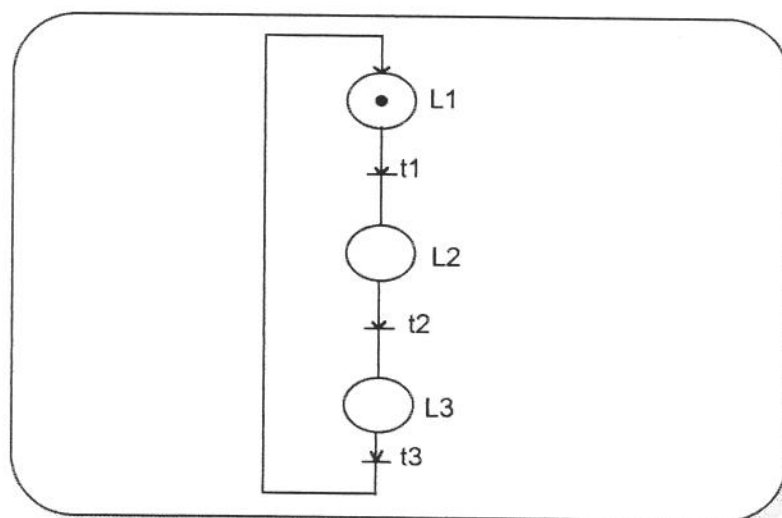


Fig.2.3.a. Exemplo de uma rede de Petri básica

Observando este simples exemplo podemos estabelecer o seguinte:

- 1. Os estados, podem conter marcas, as quais nos indicam o estado ativo. Quando um estado está ativo ele pode disparar uma transição. No exemplo, a marca está no estado inicial, ou seja a máquina está detida.
- 2. Os estados contêm transições de entrada e de saída. Analogamente pode-se dizer que uma transição tem estado de entrada e estado de saída.
- 3. As funções de Incidência α e β podem ser representadas como:

$\alpha : \{ L1, L2, L3 \}$ e $\beta : \{ t1, t2, t3 \}$. Matricialmente,

α	t1	t2	t3
L1	1	0	0
L2	0	1	0
L3	0	0	1

β	t1	t2	t3
L1	0	0	1
L2	1	0	0
L3	0	1	0

Em particular, o caso anterior é uma rede ordinária, já que as suas funções de incidência só podem tomar os valores 0 e 1. Adicionalmente, pode-se dizer que esta rede é pura, pois nenhuma transição contém um estado que seja simultaneamente de entrada e saída.

A matriz de incidência definida como Γ é representada pela a subtração de $\beta - \alpha$. No exemplo em questão, a matriz de incidência é a seguinte

Γ	t1	t2	t3
L1	-1	0	0
L2	1	-1	0
L3	0	1	-1

4. Como foi dito anteriormente, as PN podem conter marcas indicando o estado ativo na Rede. Esta marcagem pode ser representada por um vetor de dimensão igual ao número de estados da rede. Este vetor é chamado de M_0 . No caso do exemplo, o vetor de marcagem para a Rede no Estado Inicial é:

$$M_0 = (1 \ 0 \ 0)^t$$

* A^t = transposta da matriz A.

Um aspecto importante da marcagem, é a incorporação da dinâmica da Rede. Isto implica a transferência de Estado Ativo mediante o disparo da transição sensibilizada. Isto é feito eliminando-se as $\alpha(L,T)$ do estado ativo e adicionando-se $\beta(L,T)$ nos estados seguintes. Isto pode ser representado esquematicamente da seguinte forma:

$$M_i \rightarrow M_j \Rightarrow M_j(L) = M_i + \beta(T,L) - \alpha(L,T)$$

Assim pode-se concluir que existe uma seqüência de disparos, de maneira que cada disparo conduz a uma marca que sensibiliza a transição seguinte. Desta forma para o exemplo em questão tem-se:

$$\begin{array}{ccccccc} 1 & t1 & 0 & t2 & 0 & t3 & 1 \\ 0 & \rightarrow & 1 & \rightarrow & 0 & \rightarrow & 0 \\ 0 & M_0 & 0 & M_1 & 1 & M_2 & 0 \end{array}$$

Ao existir uma marcagem inicial M_0 e uma seqüência de disparos σ , então pode-se dizer que uma marcagem pode ser atingida a partir da marcagem inicial, M : $M_0 \xrightarrow{\sigma} M_i$.

Esta regra define a equação de estado de uma rede :

$$M_k = M_{k+1} + \Gamma * \theta$$

onde

M_k : é a marcagem obtida do K-ésimo disparo.

θ : é o vetor de componentes nulos, exceto o i-ésimo, se t_j for a transição de disparo no k-ésimo estado $\{\theta_k(i) = 1, \theta_k(j) = 0 \text{ para todo } j > i\}$.

Comentários finais sobre as PN:

As PN são particularmente poderosas para a modelagem de sistemas onde existem muitas atividades que ocorrem paralelamente e de maneira assíncrona (CAPKOVIC, 1993; KNAPP, 1992). A principal virtude da PN é que elas fornecem um alto grau de flexibilidade e facilidade na modelagem junto com bons resultados a nível de análise. Assim as PN são uma poderosa ferramenta para o desenvolvimento de modelos e com uma alta velocidade de execução. Finalmente, as PN podem manipular satisfatoriamente a existência de conflitos (duas ou mais atividades competem por um mesmo recurso).

2.1.3. O uso de Linguagem Natural:

Um enfoque inteiramente inovador, mas que vem sendo estudado quase que desde o surgimento da computação digital, é o uso da Linguagem Natural (NL) como meio de comunicação entre o usuário e os sistemas computacionais.

Esta área de conhecimento tem como objetivo final facilitar a interação do homem com a máquina. A modelagem de sistemas é uma área que nos últimos anos tem entrado na lista de aplicações que procuram fornecer uma interface inteiramente amigável para o usuário final (o modelador) de maneira a evitar a necessidade de conhecimento mais profundo em técnicas específicas, sejam estas matemáticas ou simbólicas.

Se se considera o grande esforço necessário na modelagem de um sistema e as dificuldades existentes para que o usuário possa definir os requerimentos e características no novo sistema, faz-se justificável a criação de uma técnica de modelagem desenhada para assistir este proceso. Através desta técnica pode-se minimizar o tempo e o esforço necessários para o processo de formulação.

As vantagens desta técnica parecem óbvias: Em primeiro lugar, pela sua facilidade de uso. Em segundo, a redução do grau de habilidade necessário para realizar uma formulação. Com isso, a produtividade do processo é incrementada significativamente.

Ademais, o modelo desenvolvido em NL pode ser usado como um eficiente meio de comunicação entre diversas pessoas com graus de preparação diferentes e em distintas áreas. Assim, um sistema descrito com NL, servirá de ferramenta para a efetiva troca de idéias entre as pessoas que participam do projeto.

Conceitos de Modelagem usando NL:

Existem diversas formas de representar informações. Elas podem ser representadas usando diferentes palavras ou diferentes estruturas de sentenças. Para representar todas as possíveis combinações de palavras que formam sentenças "*legais*" para uma determinada linguagem, esta deve ser descrita através do uso das *Gramáticas*. A alta complexidade e o quase "infinito" número de combinações possíveis na NL, tornando seu tratamento automatizado, uma tarefa quase impossível. É por isto que as abordagens encontradas na literatura reduzem o número de estruturas permitidas a um conjunto reduzido, permitindo manter as vantagens do uso da NL e possibilitando o seu processamento computacional.

Será analisado um trabalho que faz uso da modelagem baseada em NL para a construção de Modelos de Simulação e Sistemas de Manufatura. O enfoque inovador obedece ao uso das Gramáticas de Casos. Estas Gramáticas de Casos podem ser definidas da seguinte forma: A proposição embutida numa sentença simples contém uma estrutura consistente em um verbo (o componente central) e um ou mais objetos gramaticais. Cada um destes objetos está associado ao verbo de uma maneira particular. Tais relações são chamadas "*relações sintáticas semanticamente significativas*"

Um exemplo de gramática de casos, é:

Agente (A)
Contra-Agente(C)
Objeto(O)
Resultado(R)
Instrumento (I)
Meta (G)
Fonte (F)
Experiência(E)

Suponha o seguinte exemplo:

O operário move a peça da Estação 1 para a Estação 2 com um carro

"Operário" é o agente do verbo "move", "peça" é o objeto, "Estação 1" e "Estação 2" são a Fonte e a Meta, respectivamente, enquanto que "carro" é o instrumento. Junto com isto, os verbos podem ser categorizados de acordo com "tipos de sentenças", para indicar qual a relação que pode existir entre um verbo e os seus substantivos. Esta propriedade dá a possibilidade de considerar diversos verbos com um significado similar, compartilhando assim, o mesmo tipo de sentença. A vantagem desta metodologia é que quando duas sentenças, com formas diferentes, descrevem o mesmo evento, a representação de ambas as sentenças terá a mesma forma. Portanto, as descrições serão idênticas.

Atualmente, não existe uma Gramática de Casos que seja universalmente aceita. As variações de casos e de sistemas de aplicação são numerosas. O número de casos nestas aplicações também é variável, e pode ir de 5 até cerca de 30.

É importante salientar que o princípio usado para determinar os tipos de casos numa gramática está em estreita relação com o domínio do problema que está sendo atingido e da informação requerida para a solução deste. Por exemplo SU e KACHITVICHYANUKUL (1989) sugerem os seguintes casos para o domínio da manufatura discreta:

- Atividade
- Peça
- Estação
- Equipamento de Manuseio de Material
- Atributos
- Fonte
- Meta
- Estado
- Valor

O procedimento para processar uma descrição em NL, consiste em examinar as sentenças de entrada e gerar a representação semântica correspondente. O processamento desta representação foi analisado em (DURÁN e BATOCCHIO, 1993).

O processo realizado para descrever um sistema é o seguinte:

- Identificação das entidades, das atividades, dos estados e dos recursos existentes no sistema e as relações existentes entre eles. Este procedimento inclui:

- Definição das mudanças de estado
- Descrição das atividades nas quais as entidades participam
- Descrição dos processos através dos quais as entidades fluem.

- Uso de um diagrama de fluxo para descrever a estrutura global do sistema.

- Conversão da estrutura emitida no passo anterior para estruturas pertencentes à linguagem de simulação específica.

Do ponto de vista da Simulação os três elementos principais encontrados são: Entidade, Atividade e Estado. A relação entre estes elementos e um Sistema de Manufatura é explicado brevemente a seguir:

- Entidade : Existem três tipos principais de entidades no domínio da manufatura. Duas delas possuem mobilidade, a peça e o transportador. A terceira entidade, a estação, é estacionária.

Atividade : Chamada também de evento. Existem também três tipos básicos de eventos num sistema de manufatura. Um é a chegada de peças numa entidade, o segundo é a atividade de processamento de uma peça por parte de uma estação. O terceiro tipo de atividade é o transporte de uma peça de uma estação para uma outra, com a participação de um equipamento de manuseio de material.

- Estado : Usualmente identificado através do "status" de uma estação ou equipamento de manuseio de material. Este pode ser: ocupado, livre ou fora de serviço.

Adicionalmente, cada entidade e/ou evento tem associada a si um conjunto de atributos. Por exemplo, uma entidade "peça" tem atributos tais como, tamanho do lote, sequência de operação, etc. Uma entidade "máquina" tem atributos tais como número de máquinas, tamanho dos buffers, etc. Finalmente, uma atividade de processamento poderá ter associada a si a duração dela.

2.1.4. Iniciativas para a especificação de Sistemas de Manufatura levadas a cabo no Brasil:

Como já foi comentado anteriormente, aqui no Brasil existem algumas iniciativas destinadas à descrição de Sistema Automatizados de Manufatura. Estes esforços destinam-se à aplicação de algumas técnicas conhecidas na literatura na construção de modelos de simulação, no processo de validação destes modelos e como metodologias para a construção de controladores ou gerenciadores de células flexíveis de manufatura.

O primeiro trabalho a ser comentado está sendo realizado pelo GRUCON na Universidade Federal de Santa Catarina. Este grupo trabalha atualmente no modelamento de um gerenciador de células flexíveis de manufatura (CFM) baseado no uso de PN (BRAGA et al. 1993). A técnica baseia-se numa extensão das PN tradicionais, para as tarefas de especificação, validação e implementação de módulos de um software gerenciador. A justificativa dada pelos autores para a escolha das PN é que elas podem representar claramente as relações de conflito e paralelismo entre atividades de controle de uma célula. Assim, as PN são uma ferramenta adequada para atender as necessidades de tarefas de um Controlador de CFMs.

Os esforços do grupo apontam à geração automática de software que implemente as tarefas do controlador de célula. A extensão das PN baseia-se principalmente em dois aspectos: a manipulação de dados pela rede e a interação da rede com o ambiente externo. Nesta extensão, as transições da PN são aumentadas com condições suplementares ao tiro e de ações associadas ao disparo. A existência de uma condição suplementar de tiro associada a uma transição implica que, além de existir um número suficiente de 'tokens' nos lugares de entrada, esta condição deve ser satisfeita para que a transição possa ser disparada.

Para validar as PN, está sendo usado um Analisador construído na Faculdade de Eng. Elétrica da própria Universidade de Santa Catarina. A entrada deste analisador é uma descrição textual da Rede de Petri.

A ferramenta utilizada na geração de código, chamada SPP (Sistema de Produção Proposicional), gera código em C a partir de um sistema de regras de produção. A estrutura do código fonte gerado pelo SPP é totalmente transparente para o projetista e é totalmente

neutro o que o faz independente do hardware, facilitando, assim, a sua transferência. A entrada de informações no SPP é uma descrição textual do grafo. A sintaxe desta descrição textual obedece ao formato "IF-THEN".

O segundo trabalho na área é desenvolvido no Laboratório de Máquinas Ferramentas da Escola de Engenharia de São Carlos (USP). Este grupo trabalha atualmente na aplicação de um tipo de diagrama de estados chamados StateCharts na especificação de um controlador para CFMs. A utilização de StateCharts é feita nas tarefas de definição do modelo e simulação do funcionamento do sistema. Os StateCharts fornecem um formalismo visual para a representação dos relacionamentos entre os estados em que um sistema de tempo real pode se encontrar em (SILVA et al., 1993).

Num diagrama StateChart os retângulos com cantos arredondados são utilizados para denotar os estados em qualquer nível, utilizando o conceito de "encapsulamento" para expressar a relação de hierarquia. As setas, que representam as transições podem-se originar e terminar em estados de diferentes níveis. Uma seta pode ser rotulada como uma expressão lógica envolvendo eventos e condições que, se forem verdadeiras num certo instante no sistema, causarão o cumprimento da transição. Junto com a mudança de estado, a transição pode provocar instantaneamente ações que podem, por sua vez, dar origem a novos eventos. Esses eventos poderão causar o disparo de novas ações em outros componentes ortogonais (paralelos) (SILVA et al., 1993).

Para representar paralelismo são usadas linhas tracejadas delimitando os estados que estão em paralelo. A principal desvantagem é a dificuldade para se validar manualmente uma especificação completa.

As principais vantagens destacadas pelos autores são o alto poder de representação e concisão dos StateCharts e a facilidade de visualização dos componentes físicos do sistema.

Uma iniciativa para aplicar os chamados Mark Flow Graphs (ferramentas derivadas das PN), à definição de controladores sequenciais para Sistemas de Manufatura está sendo levada a cabo no grupo de Mecatrônica da Universidade de São Paulo. A aplicação dos MFG neste contexto deriva na criação dos chamados MFG / Production Flow Schema (MIYAGI, 1993; MIYAGI, MOTOHASHI e Dos SANTOS, 1993; MIYAGI, MOTOHASHI e KAGOHARA, 1993).

Esta metodologia se inicia com uma descrição conceitual chamada PFS, e a criação de uma hierarquia funcional para modelar sistemas produtivos descritos. A abordagem conclui com uma iteração de refinamentos sucessivos chegando ao estabelecimento de um modelo MFG detalhado.

Em (SANTOS et al. 1992) é proposta uma extensão dos MFG. Nela são incorporadas marcas aos grafos, numa forma similar às redes de petri coloridas, junto com a adição de regras de produção às transições.

O uso desta técnica parte do princípio que um sistema de Eventos Discretos pode ser representado pelo fluxo de materiais e de um fluxo de informações. Assim, a dinâmica dos eventos é representada pela evolução das marcas no grafo.

Segundo os autores, a aplicabilidade aos sistemas automatizados e inteligentes de fabricação é imediata, representando uma contribuição significativa para as atividades de planejamento e operação de indústrias manufatureiras (MIYAGI, MOTOHASHI e KAGOHARA, 1993).

A seguir será apresentado um exemplo de aplicação das técnicas enumeradas na seção anterior. Para encerrar, a modo de conclusões, com uma breve discussão sobre a aplicabilidade destas técnicas e uma comparação entre elas.

2.1.5. Exemplo de Aplicação:

Nesta seção serão aplicados os conceitos principais de três técnicas descritas anteriormente: SADT, Redes de Petri e Object-Oriented Development. Primeiramente, será apresentado um sistema físico, o qual será modelado, posteriormente, usando estas três técnicas. Na parte final desta seção serão feitas algumas comparações e serão analisadas as vantagens e desvantagens de cada uma delas.

O sistema a ser modelado representa uma célula de manufatura para torneamento. Nela trabalham coordenadamente, um torno, uma máquina de inspeção e um robô industrial que está encarregado de transportar as peças das máquinas e para as máquinas e dos buffers e para os buffers. Estes buffers são três: Input Buffer, WIP buffer e um buffer de produtos

acabados. Neste último buffer é feita uma separação entre as peças que são aceitas e as peças rejeitadas pelo controle dimensional.

O sistema é mostrado na seguinte figura (Fig.2.4):

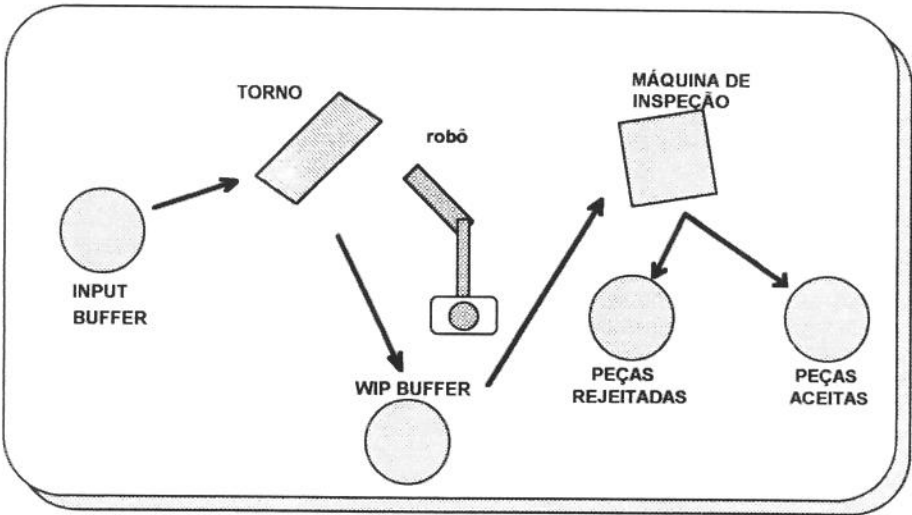


Fig.2.4. Sistema de manufatura exemplo

A figura 2.5. mostra, usando a técnica SADT, o conceito global do funcionamento do sistema. Este é o primeiro nível de análise e nele podemos apreciar o propósito do sistema (o de manufaturar componentes), o seus Inputs, os materiais, as suas saídas, componentes manufaturados aceitos e/ou rejeitados, os mecanismos que são encarregados de realizar as atividades (chamados aqui de máquinas) e o controle presente na forma de programas de controle para o sistema.

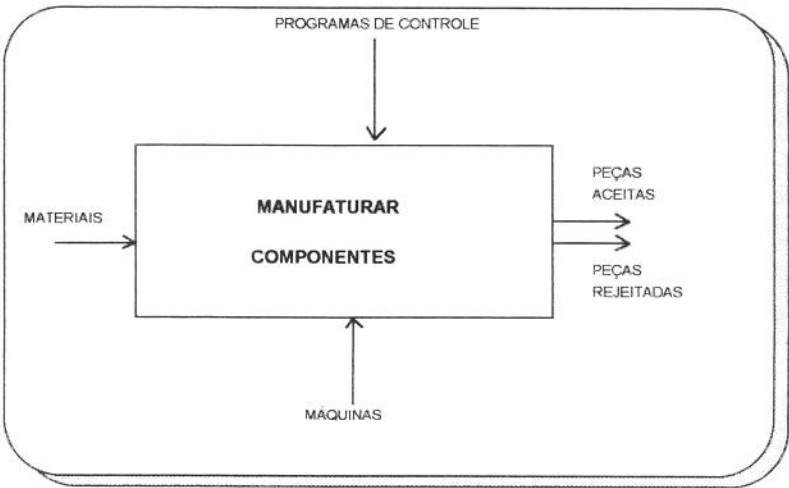


Fig.2.5.Primeiro nível para um enfoque SADT

Logo, a atividade de manufaturar componentes é dividida em outras duas atividades, num segundo nível de análise. Estas duas atividades são chamadas de Tornear Peça e Inspeccionar Peça. Este novo desdobramento é mostrado na figura 2.6. Note que SADT precisa de uma cuidadosa coordenação entre os níveis de diagrama. As Entradas e Saídas dos diagramas de nível superior devem corresponder às Entradas e Saídas do nível inferior.

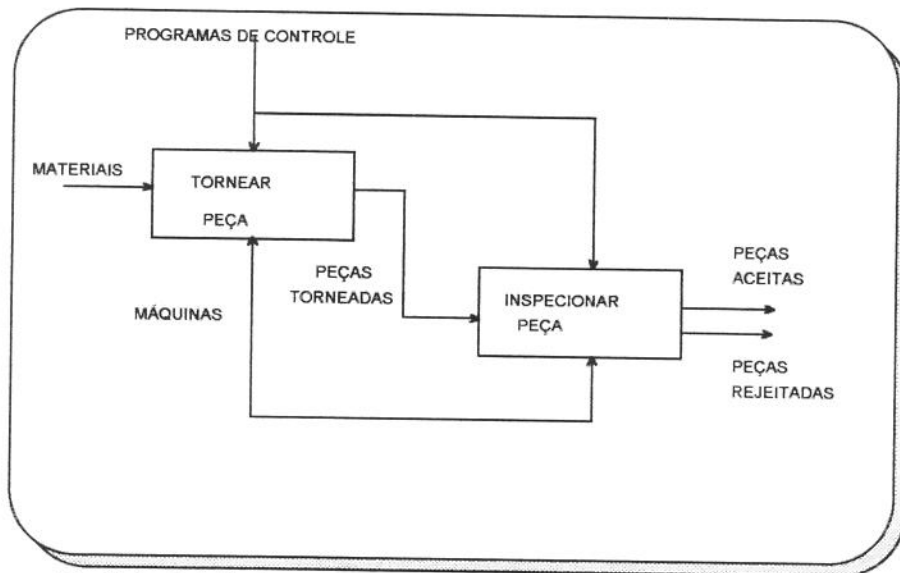


Fig.2.6. Subdivisão da atividade “Manufaturar Componentes”

Para finalizar com o exemplo de aplicação desta técnica, apresenta-se na figura 2.7., o detalhamento da atividade de Tornear Peça. Um desdobramento similar pode ser feito com a atividade de Inspeção; mas, este diagrama não será mostrado, pois não é relevante para o desenvolvimento do exemplo.

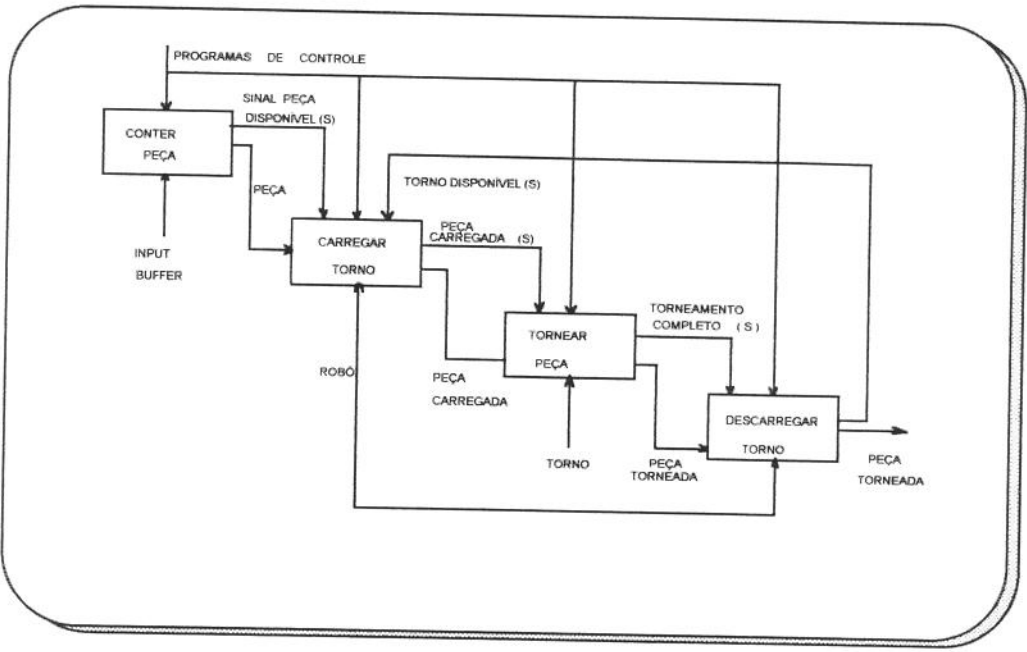


Fig.2.7. Desdobramento da atividade “Tornear Peça”

A seguir, na figura 2.8., é apresentada uma Rede de Petri representando o funcionamento do Sistema antes descrito. A descrição dos locais e das transições é fornecida nas tabelas 2.1 e 2.2, a seguir:

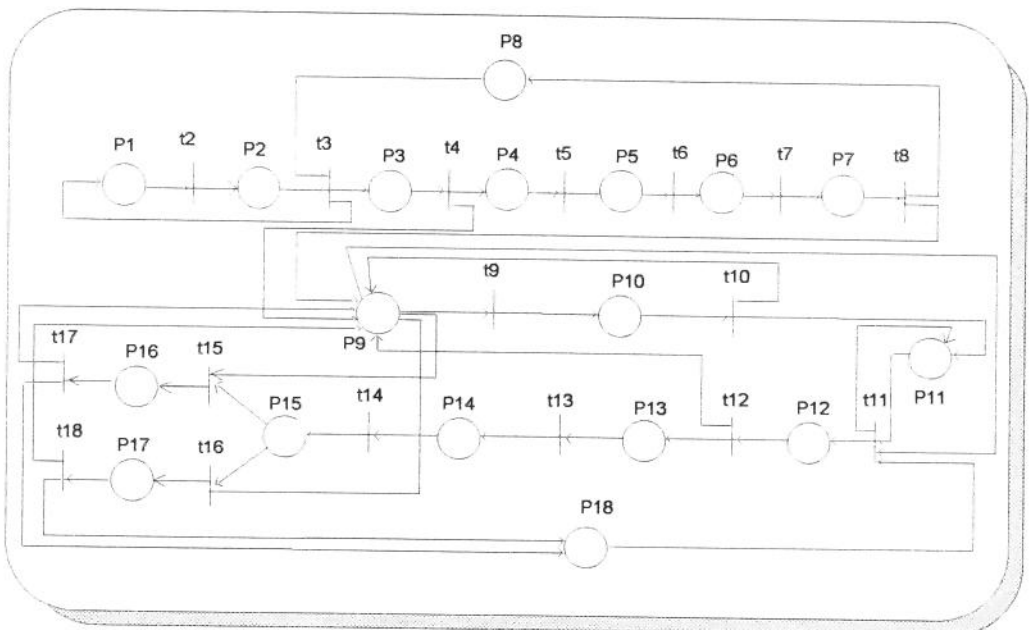


Fig.2.8. Rede de Petri representando o funcionamento do sistema descrito

TABELA 2.1. - Detalhe dos Locais da Rede de Petri

P1	Peça sendo colocada no Input Buffer
P2	Peça disponível
P3	Torno sendo carregado
P4	Peça no torno
P5	Peça sendo usinada
P6	Peça usinada
P7	Torno sendo descarregado
P8	Torno disponível
P9	Robô disponível
P10	WIP Buffer sendo carregado
P11	WIP Buffer carregado
P12	Máquina de Inspeção sendo carregada
P13	Máquina de Inspeção carregada
P14	Componente sendo medido
P15	Componente medido
P16	Comp. sendo carregado no Buffer de Rejeitados
P17	Comp. sendo carregado no Buffer de Aceitos
P18	Máquina de Inspeção disponível

TABELA 2.2.-Detalhamento das Transições.

t2	A peça é colocada no Input Buffer
t3	Robô começa a carregar o torno
t4	Robô termina de carregar o torno
t5	Torno começa usinagem
t6	Usinagem terminada
t7	Robô começa descarregar o torno
t8	Robô termina a descarga do torno
t9	Robô começa carregar WIP buffer
t10	Robô termina a carga do WIP buffer
t11	Robô começa a carga da Maq. de Inspeção
t12	Robô termina de carregar a Maq. de Inspeção
t13	Começa operação de medição
t14	A operação de Medição é completada
t15	Robô começa carregar peça no Pallet de Rejeição
t16	Robô começa carregar peça no Pallet de Aceitados
t17	Robô termina carregar peça no Pallet de Rejeição
t18	Robô começa carregar peça no Pallet de Aceitados

Grady Booch apresentou (BOOCH, 1986) uma técnica chamada de Object Oriented Development. Esta técnica pretende realizar uma aplicação do paradigma de objeto nas técnicas de Engenharia de Software para, a partir de um Diagrama de Fluxo de Dados (DFD) (GANE e SARSON, 1979), obter um conjunto de "packages" em linguagem ADA. Esta técnica está orientada à construção de Softwares em tal linguagem e mostra algumas limitações na aplicação em situações relativas à manufatura. Apesar disto, esta metodologia será usada neste trabalho para aplicar os conceitos Objeto-Orientados para o caso de estudo, e como um simples meio de estruturar a visão de objetos (e outros conceitos) para desenvolver um modelamento da célula exemplo. Na figura 2.9., está apresentado o DFD relativo à célula exemplo:

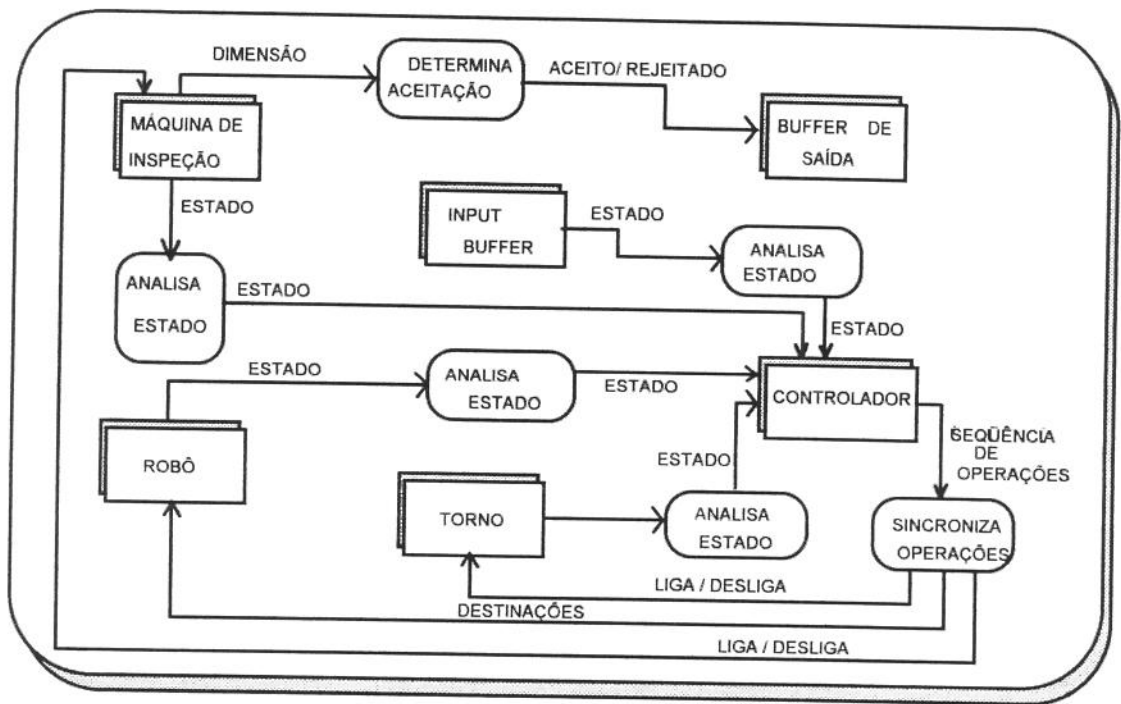


Fig.2.9. Diagrama de Fluxo de Dados relativo à célula exemplo

O segundo passo é a identificação dos objetos existentes e dos seus atributos. De maneira geral os objetos são equivalentes às entidades do DFD. Em algumas circunstâncias outros elementos do DFD podem ser considerados como objetos do nosso modelo. Desta forma, em determinadas situações um repositório ou armazém de dados pode ser considerado como objeto, já menos freqüente é a incorporação de fluxos de dados como objetos, mas isto não é proibido pela técnica. Os processos que realizam transformações

sobre os dados devem ser alocados a um objeto que sirva como agente para tal ação. O passo seguinte é a identificação das operações sofridas e requeridas pelos objetos. A tabela 2.3. mostra estas informações:

TABELA 2.3.- Operações sofridas e requeridas pelos objetos

OBJETOS	OPERAÇÃO REQUERIDA	OPERAÇÃO SOFRIDA
CONTROLADOR	Sequência de Operações	Obtém os estados dos restantes objetos
TORNO	Estado do Torno	Liga / desliga
ROBÔ	Estado do Robô	Destino do movimento
MÁQUINA DE INSPEÇÃO	Estado da Máquina	Liga / desliga
INPUT BUFFER	Estado	Nenhuma
OUTPUT BUFFER	Nenhuma	Aceita / rejeita

Note que deve existir um balanceamento entre as operações sofridas e requeridas pelos objetos. Para cada operação sofrida por um objeto, tem-se um outro objeto ou conjunto de objetos que requerem tal ação. Isto tem relação com o princípio de envio de mensagens entre objetos. Todos estes requerimentos são comunicados através da passagem de mensagens que transitam pelas interfaces de cada um destes objetos. Isto tem relação com o último passo da técnica, que é o estudo e estabelecimento das Interfaces de cada um destes objetos. Aqui o analista decide o que é encapsulado pelo objeto, ou seja, o que fica escondido do exterior do objeto e o que é público no objeto, ou ainda, que detalhes da implementação dos objetos podem ser "vistos" por outros objetos.

2.1.6.Comentários sobre as Técnicas Analisadas:

A maior parte destas técnicas, provêm de outras áreas de aplicação e ultimamente têm sido incorporadas ao contexto dos Sistemas de Manufatura Automatizada.

Pelo fato destas técnicas serem provenientes de aplicações diferentes, estas apresentam algumas limitações, especificamente na representação de algumas situações críticas, tais como sincronismos, paralelismo e conflito. Por este motivo, tais metodologias têm

acrescentado ou incorporado novos elementos (tokens, regras de produção, ações semânticas, etc) na tentativa de resolver as mencionadas dificuldades. Isto conspira contra a sua simplicidade original, tornado-as mais pesadas e difíceis de se entender.

Por outro lado, nem todas estas técnicas podem atuar na totalidade do ciclo de vida de um sistema de manufatura ou de um controlador de sistemas de manufatura. Este fato obriga os usuários à transferência de técnicas entre etapas de desenvolvimento e/ou implementação.

O enfoque mais inovador parece ser a orientação a objetos e a técnica associada a ela - o uso de sistemas operados por linguagem natural. A incorporação de técnicas objeto-orientadas está trazendo ganhos significativos na produtividade e tempo de desenvolvimento de novos sistemas. A outra grande vantagem é que o paradigma de objeto pode ser aplicado na totalidade do ciclo de vida de um Sistema de Manufatura, desde uma especificação inicial (com uma alta taxa de informalidade), até o processo de Geração Automática de Software.

Atualmente, na literatura, são encontrados diferentes tipos de diagramas para representar um projeto orientado a objeto, revelando a falta de um padrão gráfico para a representação das especificações, das classes de objetos, dos seus relacionamentos, etc. Assim, pode-se concluir que, apesar de existir uma alta difusão dos conceitos da filosofia de objetos, ainda existe um caminho para percorrer em direção à criação de uma técnica padrão para a representação dos projetos. Estes conceitos serão discutidos com mais detalhe no capítulo 4, no desenvolvimento da técnica de especificação orientada segundo este paradigma.

2.2. Técnicas de Representação de Linguagens

Desde os primeiros dias da computação tem existido uma evolução permanente em direção à sofisticação (no bom sentido da palavra) das interfaces de comunicação entre o usuário e a máquina. Estes avanços estão fundados principalmente em três conceitos: mnemônica, o paradigma e a estrutura de dados.

A capacidade mnemônica é a habilidade de substituir símbolos e frases que fazem sentido num nível mais alto, por construções num nível menor. Por exemplo, a maioria das

linguagens de programação possuem um comando "goto" cuja função é transferir o controle do fluxo do programa para uma outra localização dentro do código. Este comando, na verdade, corresponde diretamente a um comando à nível da máquina simples expresso como uma sequência de bits ou símbolos numéricos. Da mesma forma o argumento do comando "goto" é também uma construção mnemônica.

O conceito de paradigma é a capacidade de acumular várias instruções de ordem menor em uma instrução de ordem maior, que no conjunto fazem maior sentido para o programador. As subrotinas, as funções e as macros permitem ao programador construir estereótipos mais complexos. Esta facilidade de estruturar paradigmas que possuem algumas linguagens de alto nível permitem incrementar a complexidade dos sistemas que podem ser construídos, aumentando a produtividade do programador.

A última função que influencia as linguagens de programação, é a estrutura de dados. A estrutura básica de dados na maioria dos computadores é um arranjo linear de unidades compostas de um número fixo de bits. Estas unidades de armazenamento estão caracterizadas por ter um único nome projetado para elas. Porém, este tipo de ordenamento é extremamente restritivo para trabalhar com a ampla variedade de tipos de dados usados pelos programadores. Assim, existe a estrutura lógica de dados, usada pelo programador ao desenvolver o algoritmo. Desta forma, muitas linguagens têm sido projetadas para permitir ao programador estabelecer seus algoritmos em termos da estrutura lógica de dados, ao invés de requerer que sejam escritos segundo a estrutura física.

Existe uma quarta motivação para as linguagens de programação de maior ordem; esta é a independência do hardware. Apesar de ser muito difícil achar linguagens com total independência da máquina, os programas escritos em FORTRAN, COBOL, C, etc., podem ser executados em diversos computadores com relativamente poucas modificações.

Este trabalho de pesquisa visa a criação de um ambiente computacional interativo através do qual o usuário entrará com uma descrição de alto nível das tarefas de um sistema automatizado de manufatura. Assim, o programador terá à sua disposição linguagens de mais alto nível, deixando ao computador a tarefa de ler estes programas de nível mais alto e de deduzir a sequência de instruções de máquinas necessárias. Este processo é similar ao processo de compilação (ou interpretação). Por esta razão descrevem-se sucintamente os conceitos relacionados a este processo.

O compilador é um programa que usualmente está dividido em duas partes: a tradução do programa fonte e a geração do código de nível inferior. O processo de tradução revela as entidades do código fonte numa estrutura intermediária, que contém a informação suficiente para a etapa de geração. GRIES (1971) define este processo da seguinte forma: "Um compilador realiza uma análise do programa fonte e depois realiza uma síntese do programa objeto. Primeiro decompõe o programa fonte em suas partes básicas e depois constrói as partes do programa objeto equivalentes para cada uma destas partes básicas".

O processo de compilação está dividido em diversas etapas, a saber:

O Scanner:

Com certeza esta é a parte mais simples do compilador, algumas vezes chamado de Analisador Léxico. Ele analisa os caracteres do programa fonte e constrói uma lista de símbolos (às vezes chamados de "tokens" ou "átomos"). O scanner pode também colocar identificadores na lista de símbolos e outras tarefas simples. Os símbolos são geralmente passados pelo scanner ao analisador numa forma interna.

Analisador Sintático:

Estes analisadores constróem a forma interna do programa. Durante esta fase, é feito um teste completo tanto da sintaxe, quanto da semântica do programa durante esta fase. A tendência é separar a sintaxe da semântica o mais possível, assim, por exemplo quando o analisador sintático - também chamado de "parser" - reconhece uma construção determinada, ele ativa um procedimento semântico específico para tal construção. Este procedimento semântico verifica a exatidão desta construção e armazena a informação necessária à estrutura sob análise na tabela de símbolos. Por exemplo, quando uma declaração de variável é reconhecida, uma rotina semântica verificará os identificadores declarados, para garantir que não está sendo declarado duas vezes e os adicionará à tabela de símbolos, junto com os atributos declarados. Assim, considerando a seguinte estrutura:

< variável > := < expressão >

uma rotina do tipo semântico verifica se <variável> e <expressão> são compatíveis em tipo e, posteriormente, colocará tal estrutura no interior da representação interna do programa.

Representação interna do programa:

Esta representação depende em grande parte de como esta informação será manipulada posteriormente. Assim, ela pode ser uma árvore representando a sintaxe do programa fonte, pode ser o programa fonte representado segundo a chamada "Polish Notation", ou pode ser uma lista de quádruplas (operador, operando, operando, resultado) na ordem que elas serão executadas.

Preparação para a geração de código:

Antes que o código possa ser realmente gerado, geralmente é necessário manipular e/ou modificar o programa (representação interna) . Nesta etapa é possível realizar alguma classe de otimização do programa de forma a reduzir o tempo de execução do programa objeto.

Geração de Código:

Este é o processo de tradução da representação interna do programa fonte em linguagem assembler ou algum código de máquina. Assumindo que temos uma forma interna de quádruplas, o código é gerado para cada uma das quádruplas na ordem específica. Num interpretador, esta parte do compilador pode ser substituída pela própria execução do programa.

Como foi dito anteriormente, um processo de compilação está dividido basicamente em duas fases :

- Análise do programa fonte.
- Síntese do programa objeto.

Entretanto STERLING e SHAPIRO (1986) dividem estas duas fases, em cinco etapas, onde a primeira etapa transforma o texto de entrada numa lista de tokens. Esta lista de tokens é analisada na segunda etapa, gerando uma estrutura fonte. A terceira e quarta etapas transformam esta estrutura fonte em código realocável, e monta o programa objeto. A figura 2.10. mostra um esquema do funcionamento geral de um compilador.

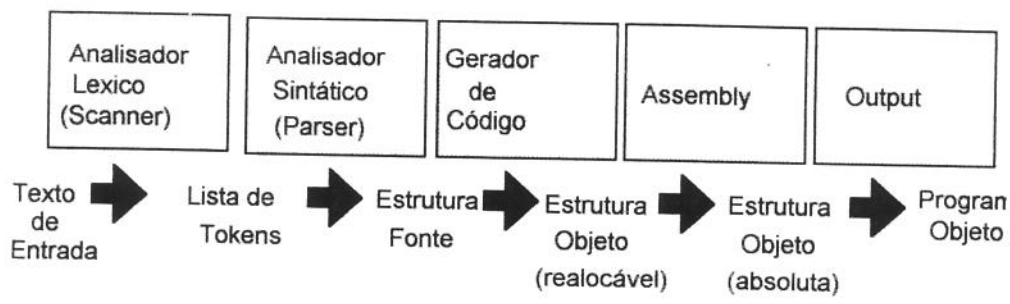


figura 2.10. Representação da sequência de compilação de um programa

Analisando as dificuldades existentes na implantação de um compilador, o scanner parece ser de uma dificuldade mínima, quase trivial, bem como a construção de analisadores sintáticos, em se tratando de linguagens formais comuns. De fato esta parte pode ser automatizada com relativa facilidade. Por outro lado, a parte com a maior dificuldade de implantação parece ser a destinada à análise sintática, como também as dedicadas à preparação do programa e à geração de código. Estas três partes são independentes, porém devem ser projetadas em forma paralela e interrelacionada.

Com esta breve introdução sobre os conceitos de processamento de linguagens de programação, tem-se condições de começar o estudo mais detalhado da teoria das linguagens formais (especificamente as Gramáticas) e a sua aplicação na construção de Processadores para Linguagens de Programação.

2.3. Gramáticas:

A fim de decompor sentenças mecanicamente deve-se fornecer regras precisas que guiem esta decomposição. Estas regras devem indicar as possibilidades combinacionais das palavras básicas ou estruturas menos complexas, para gerar todas as sentenças pertencentes à linguagem sob estudo. Ao conjunto de regras que guiam a descrição chama-se gramáticas.

Para este propósito são projetadas as "metalinguagens" - linguagens capazes de descrever uma outra linguagem.

Uma árvore sintática descreve a sintaxe (estrutura) de uma sentença através da decomposição em partes constituintes. Para realizar esta descrição são usados símbolos, chamados de "entidades sintáticas" ou "classes sintáticas"; Estes símbolos devem ser diferenciados das palavras básicas (as vezes chamadas de palavras reservadas em algumas linguagens de programação) da linguagem, pois estas últimas constituem as unidades indivisíveis da estrutura de uma linguagem, enquanto que as entidades sintáticas podem ser divididas em combinações de outras entidades sintáticas, combinações de entidades sintáticas com palavras básicas ou simplesmente, divididas em uma ou mais palavras básicas da linguagem.

SAVADOVSKY (1977) define claramente o conceito de linguagem. Através desta definição fica mais fácil entender a tentativa de modelamento dessas linguagens por meio de estruturas matemáticas. Assim, ele define uma linguagem como "um conjunto determinado de cadeias finitas sobre elementos de um alfabeto de símbolos terminais dado". Formalmente uma gramática de estrutura frasal é uma quadrúpla da forma (V, SG, P, S) onde:

V é um conjunto finito não vazio chamado vocabulário

SG é um subconjunto não vazio e finito de V chamado alfabeto de terminais.

S é elemento de $V - SG = N$ e é chamado de símbolo inicial

N é chamado de alfabeto de símbolos não terminais.

P é um conjunto finito de regras ou produções da seguinte forma:

$A \rightarrow B$

onde A é elemento de V^*NV^* e B é elemento de V^* , com $*$ indicando que o símbolo que o antecede poderá ocorrer qualquer número de vezes (inclusive zero), assim:

$V^* = \{ \emptyset, V, VV, VVV, \dots \}$

Então, a linguagem gerada por uma dada gramática G , $L(G)$ é o conjunto de cadeias de terminais que são deriváveis do símbolo inicial. Dada uma linguagem L , com algumas restrições razoáveis, existirá um número infinito de gramáticas G tais que $L = L(G)$.

Podemos classificar as gramáticas de estrutura frasal (baseada em frases) em diversos tipos de acordo com o tipo de suas produções. Esta classificação é chamada de Hierarquia de Chomsky (CHOMSKY, 1956). RICH (1988) apresenta uma tabela onde resume a chamada Hierarquia de Chomsky.

Tipo	Classe de Gramática	Restrições da regra
0	Enumerável recursivamente	Nenhuma
1	Sensível ao Contexto	Nenhuma regra de redução de comprimento
2	Livre Contexto	Lado esquerdo deverá ser um único não terminal
3	Regular	Lado esquerdo deverá ser um único não terminal e lado direito deverá ser um único terminal ou um único terminal seguido de um único não terminal

DE GRIES (1971) dá uma definição mais geral para estes conceitos. Uma produção ou regra é um par ordenado (U,x) usualmente escrito $U ::= x$, onde U é um símbolo e x é uma cadeia finita de símbolos (não vazia). Assim, uma gramática $G[Z]$ é um conjunto finito, não vazio de regras, onde Z é um símbolo que deve aparecer como a parte esquerda de pelo menos uma regra. Z é o chamado símbolo inicial. Por outro lado, todos os símbolos usados nas partes esquerda e direita formam o vocabulário V. Estas definições ficam mais claras através do seguinte exemplo:

$G1[<número>]$ é uma gramática definida pelo seguinte conjunto de regras:

$<número> ::= <número><dígito> | <dígito>$

$<dígito> ::= 0|1|2|3|4|5|6|7|8|9$

Esta notação é conhecida como Backus-Naur Form e é muito usada pela comunidade ligada à ciencia da computação para representar linguagens de programação. O símbolo | corresponde ao "OR" condicional. Desta forma, uma regra pode ser lida da seguinte forma : Um número pode ser representado por um número, seguido de um dígito; ou apenas por um dígito. Da mesma forma um dígito pode ser 0, ou 1, ou qualquer dígito até o dígito 9.

Uma classe fundamental de gramáticas é a do tipo 2 da hierarquia de Chomsky, chamada Gramática de Livre Contexto (CFG - Context-free Grammars). Nas CFG as palavras fundamentais da linguagem são identificadas como símbolos terminais, enquanto que a categoria de frases são identificadas por símbolos não terminais. Cada regra de uma CFG define a forma possível de expressar um símbolo não terminal como uma sequência de símbolos terminais e/ou de símbolos não terminais. A maior parte das linguagens de programação pode ser incluída nesta categoria, já que o sentido de uma sentença (ou de um símbolo dentro de uma sentença) não depende do contexto onde ela está presente. Dito de outra forma, estas estruturas não mudam o seu significado dependendo dos símbolos ou sentenças das quais elas estão acompanhadas.

COLMERAUER (1977) e KOWALSKI (1974) sugeriram traduzir os formalismos das CFGs num formato de propósito múltiplo chamado lógica de predicados de primeira ordem. Assim eles foram capazes de representar regras de livre contexto como estruturas lógicas conhecidas como Cláusulas Definidas (Definitive Clauses). Assim o problema de reconhecimento de cadeias pertencentes ou não a uma linguagem derivou num problema de demonstração de teoremas derivados de um axioma escrito na forma de cláusulas.

Estes princípios impulsionaram a criação da linguagem de programação PROLOG, através da qual os conjuntos de cláusulas podiam ser definidos na forma de Programas em Lógica (Logic Programming) e através da execução desses programas obter uma dedução automática desses teoremas (LEGEARD, 1995). Em outras palavras, eles criaram uma linguagem de programação que tem como uma estrutura de dados interna dita notação (As gramáticas em Cláusulas Definidas) e através da qual podem ser estruturadas análises de linguagens de programação e/ou de mais alto nível (linguagem natural) (RUEHER ET AL., 1987; RUEHER E LEGEARD, 1995).

Este fato, torna-se ainda mais importante considerando-se que a técnica de tradução de CFGs para Cláusulas definidas é apenas uma generalização, e resulta num formalismo ainda mais poderoso conhecido como Gramáticas em Cláusulas Definidas (DCG - Definite Clause Grammars) (SZPAKOWICZ, 1987).

Na atualidade, é sabido que as CFGs não são completamente adequadas para a descrição de Linguagens Naturais, bem como para algumas linguagens de programação.

Assim as DCGs superam estes problemas através da extensão das CFGs em três formas básicas:

- As DCGs fornecem o conceito de Dependência do Contexto a uma gramática. Dessa forma, as frases "legais" de uma linguagem irão depender do contexto no qual essa frase ocorre (este conceito está aplicado principalmente à interpretação de linguagem natural).
- As DCGs permitem construir estruturas (árvores) durante a execução da análise apesar do carácter recursivo da estrutura da gramática; assim, através destas estruturas pode-se representar o "significado" de uma frase.
- As DCGs permitem a inclusão de condições adicionais à gramática.

2.3.1. Expressão de CFGs na forma de Cláusulas

Para descrever como uma gramática pode ser expressa em lógica usaremos um exemplo de CFG mais geral. A seguir, é apresentada uma gramática segundo a notação Backus-Naur para uma linguagem de programação simplificada que servirá como exemplo:

```

program      ::= <statements><restatements>
program      ::= []
statement    ::= [id(v)] [:=] <expressão>
statement    ::= [if] <condição> [then] program [endif]
condição    ::= [not] <relação>
condição    ::= <relação>
relação      ::= <expressão><comp_op> <expr>
comp_op      ::= ['=']
comp_op      ::= ['<']
expressão    ::= <primitiva>
expressão    ::= <expressão><op_arit><primitiva>
primitiva    ::= [id(V)]
primitiva    ::= [num(N)]
op_arit      ::= ['+']
op_arit      ::= ['-']
op_arit      ::= ['*']
op_arit      ::= ['/']

```

Num primeiro nível de interpretação, um programa pode ser interpretado com uma sentença ("statement") seguida de um grupo de sentenças ("reststatements"). Considerando a representação de uma sentença como uma lista de símbolos, a representação de um programa na linguagem exemplo, usando a forma de regras em PROLOG é a seguinte :

`program(A,C) :- statement(A,B) , reststatement(B,C)`

As variáveis A,B,C representam listas de símbolos. No cabeçalho da cláusula, isto é, o predicado "program", o primeiro argumento A, representa a lista de símbolos a serem analisados como um programa, e o segundo argumento, C, representa a lista de símbolos restantes, ou seja, o que restará depois do análise. Se este resto não existir e o predicado "program" for completamente satisfeito, o programa estará correto sintaticamente. Assim, será atribuído a C a lista vazia. É claro que ambos os predicados do lado direito da cláusula devem ser propriamente definidos para que isto aconteça.

Os predicados do lado direito da cláusula trabalham da mesma forma que o predicado "program". Em ambos os predicados o primeiro argumento é a lista de símbolos de entrada a ser analisada, e o segundo argumento é a lista de símbolos restantes da análise. Para o predicado "statement(A,B)" a lista de entrada é a mesma usada como entrada no predicado do cabeçalho da regra. O segundo argumento - B, é a lista de símbolos restantes, que por sua vez constituirá a lista de entrada do predicado "restatement(B,C)". Se esta lista prova ser um "reststatement" válido, não restarão mais símbolos a serem testados nesta cláusula e será atribuída a lista vazia ao segundo argumento do predicado "reststatement".

Quando uma gramática é chamada de livre de contexto, significa que a análise de uma entidade em particular não depende da análise de outras entidades fora daquela que está sendo especificamente analisada. Também significa que no interior de uma entidade a análise de uma parte dela independe da análise de outra parte.

Por outro lado, uma DCG tem um conjunto finito de regras, cuja dimensão dependerá diretamente da complexidade da linguagem que está querendo ser representada. Assim, quanto mais complexa for a gramática, maior número de regras necessárias.

Diversas versões do PROLOG têm incorporadas uma notação especial de predicados para representar as regras de uma DCG. Estas fornecem uma técnica de substituição que

abrevia a notação comum dos predicados usados no exemplo anterior. O símbolo " $\rightarrow$ " é usado para indicar uma regra de gramática. Assim, a cláusula seguinte

`program(A,C) :- statement (A,B) , reststatement (B,C)`

pode ser definida como :

`program  $\rightarrow$  statement, reststatement.`

A que pode ser interpretada como : "Um programa pode tomar a forma de um statement seguido por um conjunto de outros statments chamados aqui de restatement".

Este tipo de cláusula se comporta exatamente da mesma forma que as cláusulas expressas usando a notação " : - ". A vantagem de se usar a notação de regras gramaticais, é que esta economiza código e é mais fácil de ser entendida. Os símbolos terminais também ficam mais claramente definidos através desta segunda notação :

`op_arit  $\rightarrow$  [ + ].`

A seguir é mostrada uma parte da gramática descrita acima, usando agora o formalismo DCG:

<code>program</code>	<code>$\rightarrow$</code>	<code>statements, restatements.</code>
<code>program</code>	<code>$\rightarrow$</code>	<code>[].</code>
<code>statement</code>	<code>$\rightarrow$</code>	<code>[id(v)] , [:=] , expressão.</code>
<code>statement</code>	<code>$\rightarrow$</code>	<code>[if] , condição , [then] , program , [endif].</code>
<code>condição</code>	<code>$\rightarrow$</code>	<code>[not] , relação .</code>
<code>condição</code>	<code>$\rightarrow$</code>	<code>relação .</code>
<code>relação</code>	<code>$\rightarrow$</code>	<code>expressão, comp_op, expr .</code>
<code>comp_op</code>	<code>$\rightarrow$</code>	<code>[=].</code>

PEREIRA e WARREN (1980) discutem três mecanismos importantes na análise de linguagens fornecidos pelo formalismo de DCGs. Estes mecanismos são: construção de estruturas para descrever a árvore de análise de alguma entidade, a imposição de condições adicionais para alguns constituintes de uma entidade e um tratamento geral para a dependência de contexto.

Devido ao fato de que a grande maioria das linguagens de programação de dispositivos automatizados serem do tipo livre de contexto, o terceiro mecanismo foge do escopo do trabalho, portanto, serão analisados apenas os dois primeiros.

a) Construção de árvores de análise :

Através da adição de um argumento adicional aos símbolos não_terminais pode-se construir uma estrutura que descreva todo o processo de análise em função de uma dada gramática. À medida que os símbolos "não_terminais" vão sendo expandidos através da unificação com regras específicas de uma gramática, as estruturas são progressivamente construídas. Apresenta-se a seguir um exemplo simplificado. A CFG anterior será modificada para produzir explicitamente um ramo da árvore de análise para cada interpretação feita. Assim, teremos um símbolo não_terminal aumentado tal como, "statement(S)" que significa "um statement com interpretação S". Desta forma a primeira regra:

$$\text{program}(p(S, Ss)) \rightarrow \text{statement}(S), \text{reststatement}(Ss).$$

pode ser lida como "um programa com interpretação p(S,Ss), consistindo em um statement com interpretação S, seguido de um reststatement com interpretação Ss".

b) Condições Extras:

O uso de chamadas cláusulas explícitas para procedimentos específicos no corpo de uma regra, para restringir a aceitação dos constituintes pode ser ilustrado pela seguinte regra:

$$\text{pl_integer}(X) \rightarrow [X], \{\text{integer}(X)\}.$$

Esta regra pode ser lida como "um símbolo representando o inteiro X, ou pode ser escrita como uma entidade representando o próprio símbolo X, onde X é um inteiro".

Mais adiante voltar-se-á a este tipo de notação e realizar-se-á uma aplicação mais completa dela.

2.4. Redes de Transição Aumentadas (ATN - Augmented Transition Networks):

O conceito de ATN para realizar uma análise sintática integra diversos conceitos da teoria escrita por Chomsky. Estas técnicas são consideradas pertencentes às técnicas de reconhecimento recursivo de padrões (padrões sintáticos) - recursive pattern matching - nas quais a cadeia de símbolos de uma frase é estruturada numa forma mais representativa do ponto de vista sintático. As ATN são uma progressão desde os diagramas de transição de estados finitos, passando pelas redes de transição recursivas, onde cada nível desta progressão representa maior grau de incorporação do carácter semântico elevado às entradas de um programa para análise sintática.

O ponto de partida no estudo das ATN devem ser as máquinas de Estado Finito. Estas máquinas consistem de um conjunto de nós e arcos ligando tais nós de acordo com as seguintes regras:

- Todos os símbolos terminais são representados por arcos.
- Um estado inicial (ou nodo) é definido como "S"; o estado final é etiquetado como "***".
- Uma máquina de estado finito especifica um processo dinâmico através da operação sobre sentenças arbitrárias.

Uma Máquina de Estado Finito (MEF) processa a sentença de entrada palavra por palavra, testando se dita palavra encaixa com a estrutura pré-definida. Se isto acontecer a palavra é localizada no arco apropriado e é removida da sentença de entrada. Quando o estado final for alcançado e não existirem mais palavras na sentença de entrada, diz-se que a máquina aceitou a sentença.

A seguinte máquina (figura 2.11) aceitará qualquer sentença que começar com o símbolo 'the', terminar com um símbolo pertencente à classe '<noun>' (substantivo), e que tiver qualquer número de símbolos pertencentes à classe <adjetivo> entre eles.

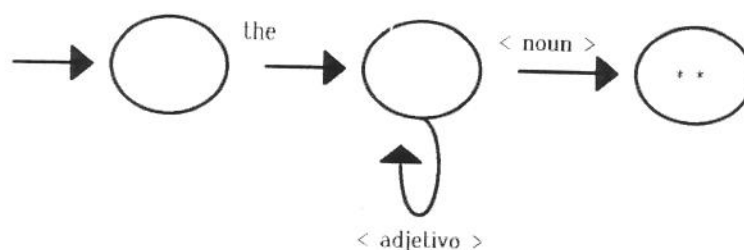


Figura 2.11. Representação de uma Máquina de Estados exemplo

A principal limitação de um MEF é que ela só é capaz de reconhecer apenas sentenças escritas como linguagens regulares (tipo 3 da Hierarquia de Chomsky), a categoria mais restritiva dentro das 4 identificadas pelo Chomsky.

Assim, para estender a capacidade das máquinas para reconhecer linguagens livre de contexto, foi adicionado um mecanismo recursivo gerando as redes de transição recursivas (RTN - recursive transition networks). Esta adição pode ser entendida como a capacidade de colocar sub-rotinas em cada arco. Deste modo, um arco pode conter o nome de uma sub-rede para a qual será transferido o controle se o arco for ativado. Quando uma sub-rede alcançar o seu estado final o controle é transferido para o nodo subsequente continuando com o processamento.

Este recurso é mostrado na figura 2.12.:

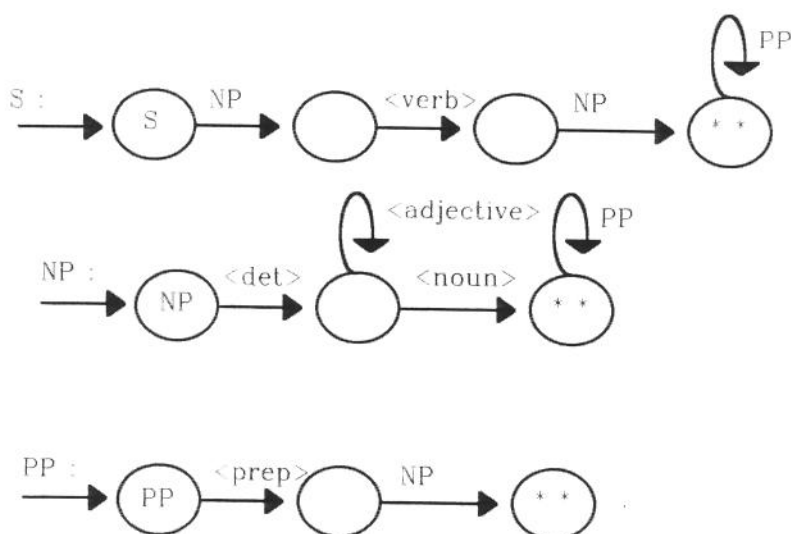


Figura 2.12.Exemplo de uso de Sub redes

A possibilidade de recursão dá às RTN uma vantagem considerável sobre as suas predecessoras, as MEF. Não existem restrições quanto aos níveis de recursão possíveis e quanto ao nível máximo de hierarquia. Também não existe limitação no número de sub-redes que podem ser chamadas de um dado arco (as RTN têm a capacidade de processamento paralelo). Para estender as capacidades dos analisadores sintáticos foram criadas as Redes de Transição Aumentadas (ATN - Augmented Transition Networks), o acréscimo foi o seguinte:

- Adição de um conjunto de registros, os quais armazenam informação, tais como árvores sintáticas parciais.
- Permite que um arco seja executado condicionalmente.
- Adiciona determinadas ações aos arcos para, por exemplo, modificar a estrutura dos dados retornados.

Estas adições foram feitas com o intuito de expandir a habilidade da ATN para manipular linguagem natural. Uma de suas dificuldades, é que não pode-se entrar com uma ATN num computador num formato gráfico - apesar de existirem ferramentas CASE para a definição de estruturas diretamente na tela usando símbolos. Isto resulta numa dificuldade adicional, que pode ter um custo maior do que os benefícios trazidos por este formato. Além disso, o formato gráfico torna-se pesado na descrição de gramáticas muito grandes. Assim, foi criado um formato textual, muito mais adequado ao tratamento computacional das ATN.

Existem dois modos em que uma prova pode operar num arco. Se o analisador está tentando achar um símbolo, por exemplo 'noun', então o analisador tenta provar se na sentença de entrada o símbolo seguinte é um 'noun'. Se for assim, ele é removido da sentença e a prova é satisfeita. Este tipo de prova é chamada "category test" e é denotado como:

(category símbolo_terminal)

Por outro lado, se o analisador desejar achar um símbolo não terminal, por exemplo um 'NP', ele irá na sub-rede para os NPs. Esta prova é definida como

(parse símbolo_não_terminal)

Adicionalmente a prova 'PEEK' atua de forma semelhante à prova 'CATEGORY', exceto que ela não remove a palavra desde a sentença de entrada.

(peek símbolo_terminal)

A prova 'WORD' procura ver se a próxima palavra tem a raiz 'root'.

(word root)

Por exemplo, (word be) será satisfeita (verdadeira) se a palavra de entrada for 'be', 'is' ou 'was'. Para permitir a completa representação textual de uma ATN são adicionados outros quatro comandos. Estes comandos podem ser entendidos como as estruturas de controle da "linguagem para ATN".

-SEQ (para denotar uma sequência de estados e arcos).

-EITHER (equivalente ao comando 'do-case' das linguagens de programação de propósito múltiplo).

-OPCIONAL (equivalente ao IF-THEN-ELSE).

-OPCIONAL* (usado quando um arco é executado qualquer número de vezes, inclusive nenhuma vez, voltando ao mesmo estado inicial).

A seguir é dado um exemplo de ATN nos formatos textual e gráfico:

```
(EITHER (CATEGORY proper-noun)
  (CATEGORY pronoun )
  (SEQ (CATEGORY det )
    (OPCIONAL* (CATEGORY adj))
    (CATEGORY noun      )
    (OPCIONAL* (PARSE pp ))))
```

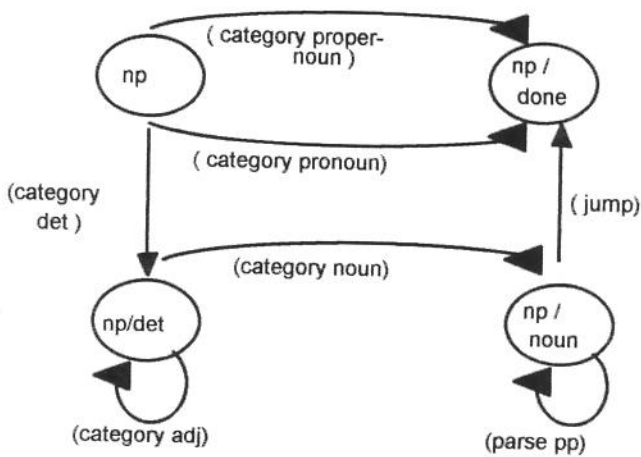


Figura 2.13.Exemplo de representação gráfica para uma ATN

Uma convenção adotada usualmente para etiquetar os nodos de uma rede é colocar o nome da rede seguido por um 'slash' ('/'), seguido pelo último símbolo analisado. Assim, vp/v é um nodo da rede vp, o qual é alcançado depois de ser achado 'v'.

A seguir são mostrados as representações gráficas para os quatro comandos definidos acima (figura 2.14.).

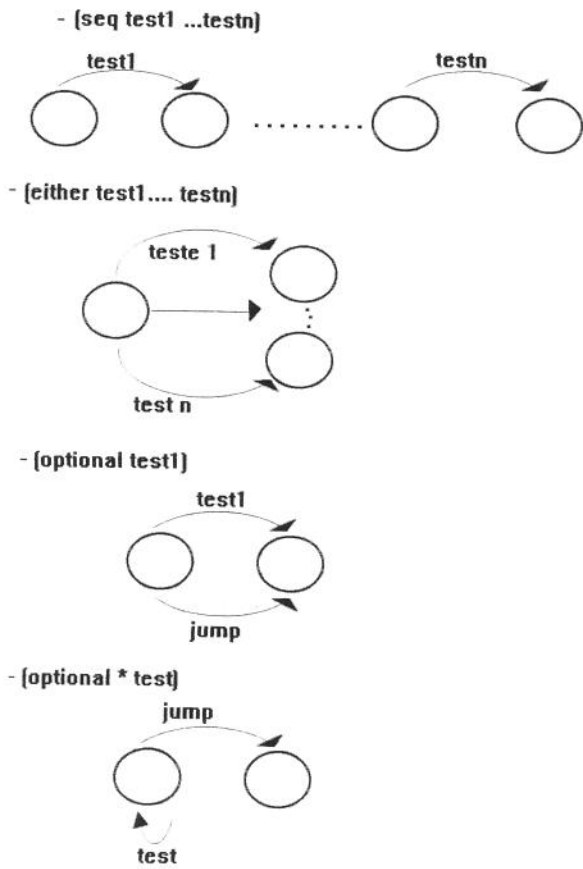


Figura 2.14. Representação gráfica dos comandos: seq, either, optional e optional*

2.4.1. Aplicação das ATNs:

Uma aplicação interessante das ATN é descrita em (SU e KACHITVICHYANUKUL, 1989) que reporta um sistema para assistir a formulação e construção automática de modelos de simulação. A estratégia usada é que, a partir de uma descrição de um sistema de manufatura feita na linguagem natural, o sistema gerará automaticamente um modelo de simulação escrito numa linguagem dedicada para fins de simulação.

A análise do texto que descreve um dado sistema é feita por um módulo chamado Natural Language Modelling Environment, que extrai a informação através de um processo de análise semântica. O objetivo de toda análise semântica é relacionar símbolos com conceitos. Como já foi dito, a maioria das regras gramaticais é usada para interpretar a estrutura sintática tais como, sujeito, verbo, objetos, etc. Porém, neste trabalho são usadas as Case Grammars (Gramáticas de Casos), as quais fornecem um método para combinar a estrutura sintática junto com uma interpretação semântica. Como já foi comentado no início deste capítulo, a idéia central de uma gramática de casos pode ser representada da seguinte forma: uma proposição embutida numa frase possui uma estrutura constituída por um verbo (componente central) e um ou mais argumentos. Cada um dos argumentos associa-se com o verbo numa relação particular. Tais relações, as quais FILLMORE (1968) caracterizou como "relações sintáticas semanticamente relevantes", são chamadas de casos.

Não há um consenso sobre o que deve ser um conjunto de casos, mas são evidentes as seguintes classes de casos:

- Agent(A): Agente, tipicamente animado, instigador da ação.
- Instrument(I): Instrumento, tipicamente inanimado, causa do evento ou objeto utilizado para causar o evento .
- Dative(D): Dativo, tipicamente animado, entidade afetada pela ação.
- Factive(F): Transitivo - predicativo, objeto ou ser resultante do evento.
- Locative(L): Locativo, local do evento.
- Source(S): Origem, local de onde algo se desloca.
- Goal(G): Meta, local para onde algo se desloca.

- Goal(G): Meta, local para onde algo se desloca.
- Beneficiary(B): Beneficiário, tipicamente animado, ser em cujo benefício o evento ocorreu.
- Time (T) : Tempo em que o evento ocorreu.
- Object (O) : Objeto, entidade que se modifica ou é modificada; é o caso mais geral.

Esta classificação foi extraída de (RICH, 1988). Adicionalmente, FILLMORE sugeriu que os verbos podem ser categorizados de acordo com os "tipos de sentenças", os quais indicam que tipo de relação pode existir entre um verbo e os seus argumentos. Esta propriedade fornece a possibilidade de existirem vários verbos, cada um deles compartilhando uma mesma classe de relação. A vantagem de tal metodologia é que quando duas sentenças descrevem o mesmo evento e ambas possuem o mesmo sentido global, mas têm pequenas diferenças, então a descrição na base do conhecimento para ambas as sentenças será a mesma.

A variação do número de casos varia com o tipo do domínio da aplicação e a informação requerida. Por exemplo, os autores definiram o seguinte conjunto de casos para trabalhar no domínio dos sistemas de Manufatura:

- Evento / atividade : os eventos ou atividades associadas com as peças ou facilidades no sistema.
- peça
- estação de trabalho
- equipamento de manuseio de materiais
- atributo: tipo de atributo que corresponde a um evento/atividade, a uma peça, a uma estação de trabalho, a um equipamento de manipulação de materiais ou ao sistema.
- origem : o local desde onde a peça é movida
- destino : local para onde a peça é movida
- estado: o estado de uma facilidade ou sistema.
- valor : valor de um atributo

O procedimento para processar uma descrição em linguagem natural, consiste em examinar as sentenças de entradas e gerar a representação semântica correspondente. Para atingir tal objetivo, cada componente do módulo processador é projetado para realizar uma função específica.

São do interesse desta revisão a análise sintática e a geração do modelo escrito numa linguagem de simulação. Estas tarefas são feitas por um conjunto de três módulos, a saber :

- Analisador léxico: este módulo analisa com base num dicionário de símbolos (chamado de **lexicon**) se todas as palavras do texto de entradas já foram pré-definidas e são conhecidas pelo sistema. Este dicionário armazena em forma sintetizada toda a informação tanto sintática quanto semântica das palavras (para usar a notação comentada mais atrás, símbolos terminais).

- Uma rede de transição aumentada: esta serve como base para o processo de análise sintático. Neste processo são analisadas todas as sentenças do texto de entrada para ver se elas são componentes legais da linguagem definida pela ATN.

- Sintetizador Semântico: este módulo gera uma representação semântica da sentença de entrada usando como informação básica a rede ATN e o dicionário. Esta representação semântica é depois interpretada com o intuito de se construir o programa em linguagem dedicada à simulação.

A rede de transição aumentada e os analisadores foram implementados numa estação APOLLO usando a linguagem LISP.

2.5. Comparação entre as ATNs e as DCGs:

PEREIRA e WARREN (1980) realizaram uma série de comparações entre as duas técnicas e concluíram que as DCGs representam um significativo avanço frente às ATNs. Isto é devido a que as DCGs são mais poderosas, mais flexíveis e mais fáceis de entender do que as ATNs.

Por outro lado, do ponto de vista prático, a maior clareza e modularidade de uma DCG é um fator vital no desenvolvimento de sistemas de alta complexidade e maior tamanho. As DCGs são mais fáceis de serem extendidas ou modificadas devido, principalmente, ao fato de que estas consistem de conjuntos de pequenas regras escritas de forma declarativa (autoexplicativa). Este conceito fica mais claro considerando que ambas as técnicas podem ser entendidas como "máquinas" para a análise de linguagens, porém apenas as DCGs podem ser visualizadas como técnicas de descrição formal de uma linguagem. O próprio WOODS (1970), criador das ATN, comenta que "ao observar uma regra, as consequências de tal regra para um determinado tipo de construção permitido ficam imediatamente aparentes".

Outro ponto de vantagem para as DCGs, é o seu caráter modular. Como foi dito anteriormente, as DCGs são construídas a partir de pequenos componentes (cláusulas) as quais se comunicam exclusivamente através de argumentos. Não existem variáveis globais - o escopo de cada variável está limitado a uma cláusula determinada. Como resultado, o comportamento de cada cláusula numa DCG pode ser entendido de maneira independente das outras cláusulas. Por outro lado, numa ATN, a menor unidade é uma sub-rede, isto prende-se ao fato de que o escopo de um registro é uma rede por completo. Assim, em aplicações complexas as sub-redes tendem a ser muito grandes, contendo um grande número de detalhes para que elas possam funcionar de maneira correta.

Um fator que faz as DCGs mais fáceis de serem entendidas, é que nelas não existem as funções de atribuição de variáveis; o valor de uma variável, uma vez fixado, ele não muda. Afortunadamente a maioria das versões de ATN usam a atribuição de valores em forma restrita. Assim, elas também são fáceis de serem entendidas neste sentido e fáceis de serem traduzidas para DCGs.

Segundo o seu criador (WOODS, 1970) as ATNs podem ser usadas tanto para fins de geração quanto para de análise. Agora, para realizar a primeira dessas tarefas é preciso executar mudanças substanciais na interpretação das operações (que etiquetam os arcos). Assim, a viabilidade deste mecanismo é questionável. Em contraste, é perfeitamente possível realizar um processo de geração com uma DCG sem mudança alguma. Ainda mais, através de um mesmo procedimento (em PROLOG, por exemplo) pode-se adequá-la tanto para processos de análise quanto para geração.

Ao comparar as DCGs e ATNs equivalentes no seu formato textual, observa-se que as DCGs são significativamente menores. Segundo (PEREIRA e WARREN, 1980) em média as DCGs correspondem a 50 % do tamanho de uma ATN. Pode-se dizer que as DCGs são, em geral, mais concisas do que as ATNs, pela mesma razão que programas em lógica são em geral, mais concisos que os programas escritos em linguagens convencionais.

Finalmente, o formalismo das DCGs é mais flexível do que o das ATNs, já que o primeiro não está baseado num mecanismo específico de análise e/ou execução. Assim, torna-se mais fácil realizar experimentos com diversas estratégias de análise através da descrição de uma linguagem usando o método DCG.

2.6. Aplicações e outros métodos:

As gramáticas e outros métodos formais têm sido usados no desenvolvimento de sistemas em vários aspectos da engenharia de manufatura. Por exemplo WU et al. (1986) aplicaram as gramáticas para um problema de formação de células usando uma técnica de reconhecimento de padrões sintáticos. Por outro lado, GRAHAM e SARIDIS (1982) desenvolveram uma estrutura hierarquica de controle baseada numa análise lingüística, a qual é capaz de interagir com o operador humano numa linguagem de alto nível, no nível mais alto da hierarquia, e é capaz de controlar os movimentos num sistema físico mais complexo, no nível mais baixo da estrutura. Para cada nível da hierarquia são descritas gramáticas formais as quais podem gerar exatamente as ações de controle adequadas para esse nível.

Para relacionar elementos lingüísticos em dois níveis adjacentes, isto é, ligar duas formas de se expressar uma mesma ação, foi criada uma estrutura chamada "Linguistic Decision Schema", a qual incorpora capacidade de aprendizado para obter automaticamente relações lingüísticas entre dois níveis da hierarquia.

Uma aplicação diferente das gramáticas formais pode ser encontrada em TAM (1989), que apresenta um método para modelar linhas de produção. Esta técnica está destinada a estruturar modelos de simulação para avaliar as políticas de agrupamentos, filas, etc.

NAYLOR e VOLZ (1987) desenvolveram uma estrutura para a criação de software de control em tempo real para sistemas de Manufatura Integrada. A técnica usada baseia-se na fusão de conceitos modernos de Engenharia de Software e modelos semânticos formais. Assim, os programas são construídos como um conjunto de componentes de software (montagem), todos escritos numa linguagem de programação distribuída comum. Estes componentes e montagens são construídos de maneira a serem genéricos, permitindo ao usuário reutilizar estes blocos.

Uma observação importante feita por estes autores é a necessidade futura de uma linguagem comum para o controle de sistemas de manufatura. Isto não significa a criação de uma estrutura monolítica que contenha todos os softwares dos subsistemas, mas o desenvolvimento de um ambiente básico e comum que forneça uma estrutura global contendo as linguagens específicas dos dispositivos e que seja capaz de se interfacear com eles através da programação semi ou totalmente automática.

A metodologia usada para realizar as traduções entre a linguagem distribuída e as linguagens específicas foi contruir um pré-tradutor que lê um programa, escrito em ADA, e dá como saída um conjunto de programas, também escritos em ADA, para cada processador específico. Este enfoque permite a distribuição de software entre um conjunto de processadores na forma de bibliotecas geradas por uma instrução PRAGMA (semelhante a um processo de compilação embutido em ADA). Uma versão primária do sistema foi implantada numa rede VAX junto com um sistema de simulação que fornece a representação dos componentes (dispositivos) a nível de chão de fábrica.

As Redes de Petri, uma importante ferramenta de modelagem formal, também foram aplicadas ao projeto, teste e desenvolvimento de controladores para sistemas de manufatura. Por exemplo, BRUNO e MARCHETTO (1987) desenvolveram uma extensão de Redes de Petri, chamadas PROT nets (PROcess Tranlatable Nets), onde a implantação dos processos e a sincronização dos mesmos é gerada automaticamente a partir da Rede usando um tradutor para a linguagem ADA. Desta forma, é possível atingir uma rápida prototipação de sistemas. PROT fornece um ambiente adequado para suporte das atividades de construção de um

modelo de especificação operacional, avaliação, simulação e validação do modelo, e a tradução automática para estruturas de programas. Especificamente, o sistema consiste em um ambiente para a criação e edição de Redes de Petri, e de um mecanismo de tradução automática em programas escritos na linguagem ADA.

O processo de tradução das Redes PROT para programas em ADA envolve dois estágios: a identificação dos tipos de processos e a sua implantação, e a sincronização de acordo com a lógica expressa na Rede. Esta metodologia foi implantada através da criação do AISPE (Advanced Industrial Software Production Environment), um sistema que integra as seguintes estruturas:

- Editor gráfico de Redes.
- Tradutor de Redes.
- Editor de Programas

O sistema foi escrito em Pascal e implantado numa máquina DEC VAX / 11.

GUNASENA e LEHTIHET (1991) desenvolveram um sistema para a geração automática de diagramas de contato, cuja entrada é uma lista com relações de precedência entre atividades (Project Network, um tipo de CPM, o método do caminho crítico), junto com uma descrição funcional dos dispositivos de entrada e saída. Em adição, requer-se do usuário o fornecimento dos estados de cada um dos sensores (ON/OFF) durante cada atividade, como também as condições necessárias para desativar uma atividade.

JOSHI e SUPINSKI (1992) reportam o desenvolvimento de um simulador genérico para uso em microcomputadores de programas para CLPs. O desenvolvimento do simulador compreende duas etapas : uma para a criação de um CLP virtual e a outra para a manipulação deste CLP virtual e a interação com o usuário.

O CLP virtual compreende um analisador léxico e um parser, os quais funcionam como compiladores, analisando a sintaxe e reconhecendo os comandos de uma linguagem específica para um CLP para, posteriormente, criar uma representação interna do funcionamento do CLP durante a execução do programa na memória do microcomputador, assim, é feita uma verificação do programa através da emulação diminuindo assim os tempos de depuração e o risco de dano no equipamento real.

A técnica desenvolvida não é específica para um sistema em particular, e pode ser aplicada genericamente a qualquer controlador. A linguagem de programação de um CLP é apresentada ao sistema na forma de CFG. As ações a serem realizadas, assim que as instruções são lidas e reconhecidas, são apresentadas como ações semânticas associadas a uma ou mais regras da gramática. Estas ações são usadas para criar uma representação interna da lógica do programa para CLP na memória do microcomputador.

A construção do analisador léxico e sintático foi assistida com o uso de técnicas de geração automática extraídas de (AHO, 1985). A análise semântica é realizada durante o processo de análise sintático, e as ações semânticas associadas às regras de produção da gramática são executadas quando uma dada produção é "disparada" durante o processo de redução do "parser".

Para incorporar uma nova linguagem de programação para um outro CLP que se deseja simular é preciso escrever esta linguagem usando o formalismo de CFG. Os autores usaram linguagem C para tal tarefa. Mais adiante,voltar-se-a à contrução destas gramáticas para representar uma linguagem de programação de um CLP no formato de CFG e DCGs.

BALA RAM e LAI (1992) propõem uma técnica de especificação a nível de hardware, isto é, identificação dos sensores e dispositivos que participam da aplicação. O argumento usado para isto, é que este é um método mais natural para realizar a programação de um CLP. A segunda etapa da especificação do programa, é a definição das condições sob as quais o estado de cada dispositivo irá mudar. Esta especificação das condições de mudança é estruturada na forma de uma tabela identificando o dispositivo, o seu estado inicial e a condição (estado de um determinado dispositivo ou chave) sob a qual mudará o estado inicial. Veja a seguir um exemplo desta especificação:

Dispositivo	estado inicial	On	Off	Delay ON
P1	OFF	h1		
P2	OFF	P2		
P3	ON	w1		
P4	OFF	w1		
P5	ON			C1,2
C1	ON		h2	
C2	OFF		h3	

A tradução destas especificações para um programa de CLP envolve dois passos:

- Tradução de cada condição de mudança de estado em um ou mais laços do diagrama de contatos.
- Atribuição de localizações válidas para cada chave, dispositivo ou temporizador.

Todas estas especificações são estruturadas na forma de tabelas (uma para chaves, outra para dispositivos, e uma terceira tabela para a descrição das condições de mudança de estados). Por exemplo, veja a seguir a tabela contendo a definição dos dispositivos ligados ao CLP em questão:

Dispositivo	Descrição
C1	Conveyer principal
C2, C3, C4	Conveyers secundários
P1, P2, P3	manipuladores ligados ao conveyer C1
P4, P5, P6	manipuladores ligados aos conveyer sec.
S1	leitor de código de barras.

Para realizar uma atribuição correta, esta dependerá diretamente do tipo de fabricante de CLP que se deseja programar. Assim, existirá uma tabela com os requerimentos específicos de endereçamento para cada tipo de CLP em uso. A única limitação aparente deste pacote de software é que ele não suporta instruções de tipo "Counters".

TOGINO et al. (1980), descreveram uma ferramenta de software chamada Automatic Programming Software System junto com uma linguagem de programação que é usada para a criação automática de diagramas de contato. Apesar de o artigo não apresentar os detalhes do processo de geração dos diagramas, algumas limitações são evidentes, como por exemplo, a impossibilidade de trabalhar com atividades paralelas e temporizadores.

O sistema de GARR e HAMMER (1988) inclui uma linguagem de alto nível chamado "Sequential Function Chart", usado nos equipamentos da Texas Instruments, que é capaz de representar problemas tanto a nível discreto como de controle seqüencial.

BOUCHER e JAFARY (1992) descrevem um sistema para a definição de controladores sequenciais a partir de uma especificação de alto nível. Através deste sistema são alcançadas diversas vantagens, tais como um melhoramento na comunicação entre especialistas, a geração automática da lógica de controle a partir desta descrição de alto nível, e a permissão de uma descrição analítica das decisões operacionais e funções de controle especificadas durante a fase de projeto.

Como técnica de especificação foi usada a metodologia IDEF0 (Integrated Computer Aided Manufacturing Definition 0 - SoftTech Inc. sob encomenda da USAF). Por outro lado, a técnica usada para a definição das lógicas de controle foram as redes de Petri. Devido ao fato de esta técnica fornecer grandes facilidades para a especificação, análise e implementação de processos de controle, e à possibilidade destas serem usadas tanto como uma interface com controladores sequenciais convencionais, como também poderem ser usadas junto com um computador para integrar diretamente com o sistema sob controle.

BHATNAGAR e LINN (1990) criaram um sistema de programação automática para controladores lógicos a partir de uma descrição de alto nível fornecida pelo usuário. Esta descrição está baseada numa série de atividades. Estas atividades são obtidas através da decomposição de um processo em diversos estágios de controle, as quais são chamadas de operações. Por sua vez, uma operação pode ser dividida em uma série de condições e ações. Os autores definiram um formato único para a entrada da especificação do processo (que não passa de uma tabela em formato ASCII). Cada operação é escrita na forma de uma "n-upla" onde os elementos estão divididos na seguinte forma:

- Nome da operação.
- Número sequencial correspondente à operação.
- Condição # 1.
- :
- :
- Condição # n.
- Ação.

O sistema está baseado num padrão de especificação da lógica de controle (SCLS- Standard Control Logic Specification), cuja sintaxe é similar ao formato do LISP, usando uma lista de parâmetros agrupados entre parêntesis para representar uma operação. As condições, as funções, os dispositivos e as ações são representados por um símbolo e um número (o sistema possui um dicionário para a identificação dos diversos símbolos). Para a especificação das lógicas são usados operadores booleanos. Assim, uma operação consiste principalmente de ANDs e ORs combinados com as condições, por exemplo:

[OR { AND X1 (NOT X2) } {AND X1 X3 } Y1]

Posteriormente, esta representação é transferida ao código de um CLP em particular. Para realizar esta tarefa o sistema possui uma base de dados contendo os detalhes do

hardware para cada CLP. Assim, cada uma das ações e condições são "etiquetadas" com o código específico que identificará cada um dos dispositivos presentes na aplicação.

As duas maiores virtudes do sistema são que o processo é bidirecional, isto é, o sistema pode gerar um programa para um CLP específico a partir do SCLS, como também a partir do código específico chegar na especificação em SCLS. Isto permite uma rápida adaptação do código existente para um outro tipo de CLP que suporte as operações programadas. A outra capacidade importante é que o sistema possui um módulo de detecção de erros. Isto é interessante, já que os CLPs podem ter regras de programação e restrições diferentes, assim o sistema examina se as regras de programação para um CLP específico, têm sido cumpridas, se as funções definidas estão disponíveis para o CLP em questão e se tais funções têm o seus operandos na sequência requerida.

Uma solução similar, só que desta vez aplicada à programação de Robôs Industriais foi publicada por LEE e CHEN (1991), no seu sistema AMPLE. AMPLE (Automated Manufacturing Programming Language Environment) é um sistema que, a partir de uma descrição das tarefas robotizadas, escolhe um robô apropriado e traduz esta descrição num programa de controle no formato do robô específico. Para realizar esta tarefa, foram identificadas as operações realizadas mais frequentemente pelos robôs, posteriormente foi criado um conjunto de "templates" (uma espécie de máscaras algorítmicas ou gabaritos) para permitir a tradução das operações para instruções em código do robô. O sistema está dividido em dois grandes módulos, um pré-processador e um processador. O pré-processador interage com o usuário para realizar a descrição da tarefas e na escolha (semi-automatizada) das máscaras adequadas às tarefas descritas. O processador avalia as tarefas a serem realizadas, escolhe o robô apropriado e realiza o sequenciamento das operações. A característica mais importante do sistema é a representação interna dos elementos operacionais das tarefas robotizadas. Os autores identificaram um conjunto de oito classes de categorias de aplicações robotizadas no ambiente da manufatura discreta. Cada uma destas categorias pode ser caracterizada através de um verbo e uma lista de parâmetros. Assim, estes verbos passam a ser as palavras reservadas ou símbolos terminais da especificação. Dependendo da natureza da tarefa, os parâmetros da descrição conterão diferentes informações. O pré-processador identifica para cada tarefa uma máscara, pede para o usuário entrar com os parâmetros necessários, e armazena as estruturas em memória. Em função

dessas estruturas o processador escolhe o robô adequado e realiza a tradução para a linguagem de programação deste robô.

No próximo capítulo, são feitas algumas conceituações sobre o paradigma de objeto e dos processos e sistemas de manufatura necessários para a definição da técnica de especificação.

Capítulo 3

O Paradigma de Objeto

O projeto orientado a objeto pode ser definido como a técnica que, diferentemente das técnicas tradicionais de projeto (normalmente com um enfoque funcional), está baseada na decomposição modular em classes de objetos que compõem o sistema, e não na decomposição em funções que o sistema realiza (BOOCH, 1986).

Este conceito provém da engenharia de software (WASSERMAN et al., 1990), e trouxe mudanças revolucionárias para a escrita e projeto de software. Com a aplicação deste paradigma os conceitos de produtividade, confiabilidade e reutilização têm observado importantes incrementos (BAKOUCH, 1994). Isto se verifica, principalmente, nas tarefas de especificação de requerimentos, projeto e programação. Este fato torna-se mais claro com a incorporação de novas linguagens de programação (SMALLTALK, ADA, C++, etc.). Na continuação analisar-se-ão os principais conceitos existentes por trás deste novo paradigma, e comentar-se-á sua aplicação no projeto de Sistemas de Manufatura.

3.1. Conceitos do Paradigma de Objeto:

Como COX e NOVOBILSKI (1991) comentaram, desafortunadamente a indústria do software possui um "record" negativo no fornecimento dos seus produtos. Os sintomas fatais, desgraçadamente, parecem familiares: erro no planejamento, falta total de controle das operações, baixa qualidade, etc. Estes problemas podem derivar eventualmente no cancelamento do projeto e no descrédito para o fornecedor. Em ambientes tão competitivos, que atualmente se apresentam, isto pode resultar mortal para uma empresa.

O problema real é a incapacidade dos produtores de software de enfrentar e responder a uma série de situações que atentam contra a qualidade do produto e do serviço que estes fornecem. Entre estas situações pode-se citar: as mudanças nos requerimentos, alterações do projeto, descrição ambigua das necessidades, etc.. Dito isto, pode-se concluir que existe a necessidade da criação de novas ferramentas de desenvolvimento e metodologias para melhorar a tarefa de construção de software. O paradigma de orientação a objeto, aplicado tanto nas fases de análise como na fase de programação, parece a técnica mais adequada com as tendências atuais do mercado de software (STEFIK, 1991). Isto pode ser estendido para as tarefas de geração de software de controle, e em geral, na produção de aplicações de gerenciamento de processos automatizados.

O Paradigma de Objeto, à diferença das metodologias de análise e programação tradicionais (com uma visão funcional da realidade), divide o domínio num conjunto de classes de objetos (WALKER, 1992). Este conceito, usado inicialmente na Engenharia de Software (WASSERMAN, 1990), causou mudanças revolucionárias no projeto e escrita de software.

O conceito de software orientado a objeto foi chamado por COX e NOVOBILSKI (1991) de Integrated Circuit Software (software ICs) para enfatizar a semelhança existente entre estes módulos de software e os chip de circuitos integrados. Assim, um programador não precisa mais construir programas inteiros a partir do nada, agora ele pode construir novos programas através da montagem de componentes de software reutilizáveis de outros programadores (Figura 3.1.)(WOLCZKO, 1992).

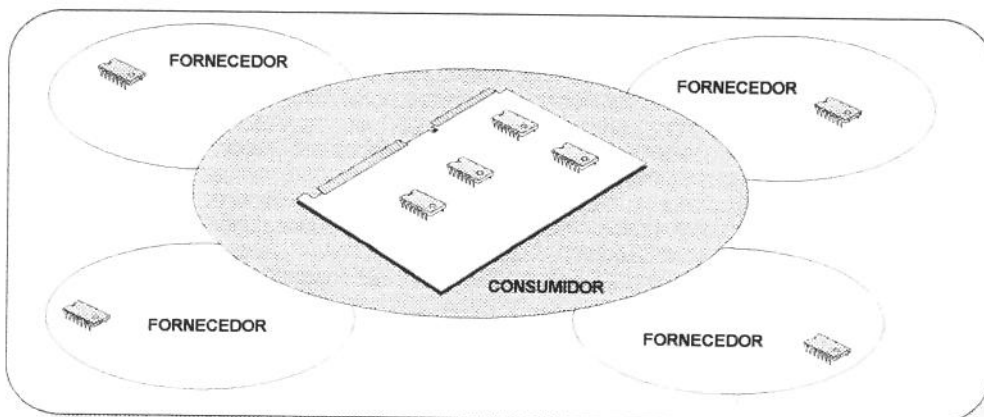


Figura 3.1. O usuário pode montar as suas aplicações a partir de um conjunto de componentes adquiridos de diferentes fornecedores.

A tendência é que a construção de software se torne cada vez mais semelhante às linhas de montagem industriais, onde os montadores apenas compram componentes no mercado e os combinam, de maneira tal, a obter um produto.

3.1.1. Conceitos:

Antes de dar continuidade, deve-se entender o conceito de OBJETO. Objeto é uma entidade a qual encapsula dados (atributos) e procedimentos (métodos). Qualquer objeto do mundo real pode ser enxergado como um objeto e, portanto, qualquer objeto pode ser representado como uma peça de software. O processo de representação de um objeto do mundo real numa peça de software na memória de um computador ou num modelo orientado a objeto, é chamado de **Abstração**. Assim, através da abstração, a percepção de um objeto (no modelo) será muito próxima à percepção do objeto real. Isto é, para cada objeto considerado do mundo real o modelo associa um conjunto de atributos e métodos, os quais são inerentes ao objeto do mundo real (BOOCH, 1986). O processo de abstração é esquematizado na figura 3.2.

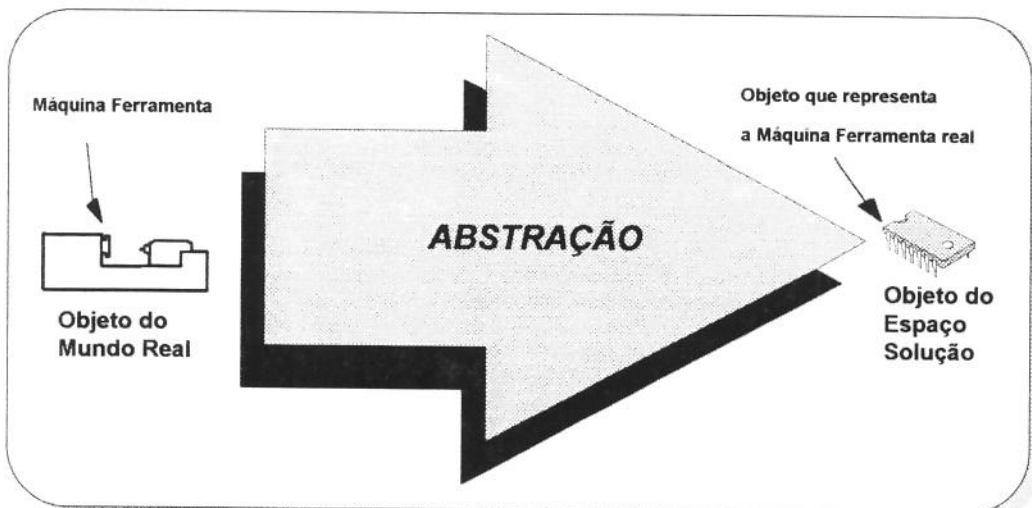


Figura 3.2. O conceito de abstração. O programador visualiza uma entidade do mundo real e logo cria uma representação simbólica no espaço solução. Esta representação é chamada de "objeto".

No caso de estar-se trabalhando no domínio da manufatura, a abstração permite a representação de um objeto qualquer pertencente a este domínio, por exemplo, uma máquina ferramenta, e estruturar uma entidade dentro do modelo que a represente e identifique (figura 3. 3.).

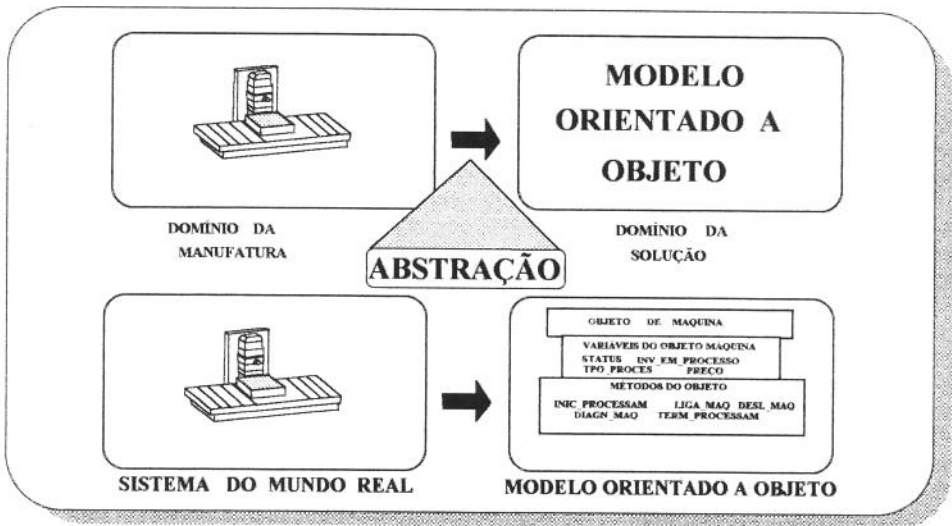


Figura 3.3. O conceito de Abstração usado para representar uma máquina ferramenta.

Encapsulamento é provavelmente o conceito mais importante, sobre o qual o paradigma se baseia. Como mencionado anteriormente, cada objeto é associado a um conjunto de operações e atributos; estes atributos apenas podem ser acessados pelas operações definidas no interior do objeto. Usualmente, estas operações estão localizadas numa parte privada do objeto e são acessadas através de mensagens. A identificação dos métodos e atributos de um objeto é feita numa seção pública chamada de Interface, através da qual as mensagens passam para o "interior do objeto". COX e NOVIBILISKI (1991) descrevem este conceito como "uma muralha de código ao redor de uma estrutura de dados" (Figura 3.4.).

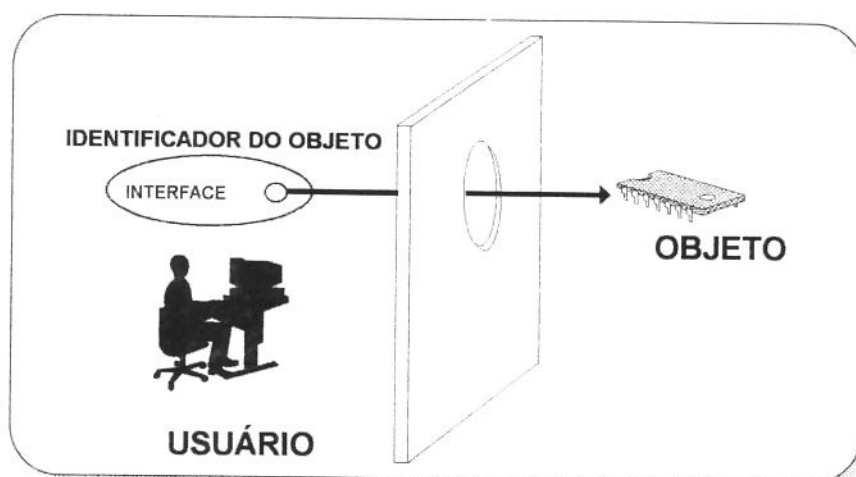


Figura 3.4.- O Conceito de Encapsulamento. Existe uma muralha rodeando cada objeto. O usuário apenas especifica o quê ele deseja que o objeto faça através do envio de uma mensagem. A forma como o objeto realiza a ação desejada, ficará oculta numa parte privada do objeto.

Através deste conceito os detalhes da implementação são escondidos numa caixa preta, a seção privada do objeto. Assim, o usuário do objeto pode especificar apenas o que o objeto deve realizar, deixando ao objeto a tarefa de escolher o código que é mais apropriado para o tipo ou classe.

O terceiro conceito é a **Herança**. Esta é a capacidade de construção de hierarquias de objetos. Em forma similar a uma árvore genealógica, vários objetos podem ser originados a partir de um objeto comum ou classe comum. Consequentemente, os objetos herdados podem ser estendidos, incorporando novas capacidades e adicionando atributos. HODGE e MOCK (1992) identificam este mecanismo como uma "relação generalização-especialização", onde a superclasse é a versão geral de uma subclasse mais especializada. Este conceito é o aspecto mais inovativo do paradigma, já que a maior parte das técnicas tradicionais (funcionais) não incorporam a capacidade de gerar subclasses a partir de uma superclasse existente.

A capacidade de herdar atributos e operações fornece as ferramentas para reutilizar blocos de software. Isto permite ao programador ou analista definir um novo software, ou novo modelo, na mesma forma que um professor (o especialista) introduz um conceito ao aluno (ou aprendiz), através da comparação com alguma situação similar, que resulta familiar e, que com pequenas modificações, poderá ser estendida para a nova situação. Assim, um

programador poderá pegar uma peça existente de software com objetos em comum com o domínio do problema que está sendo resolvido, e com mínimas alterações e/ou adições gerará um solução para tal problema (figura 3.5.).

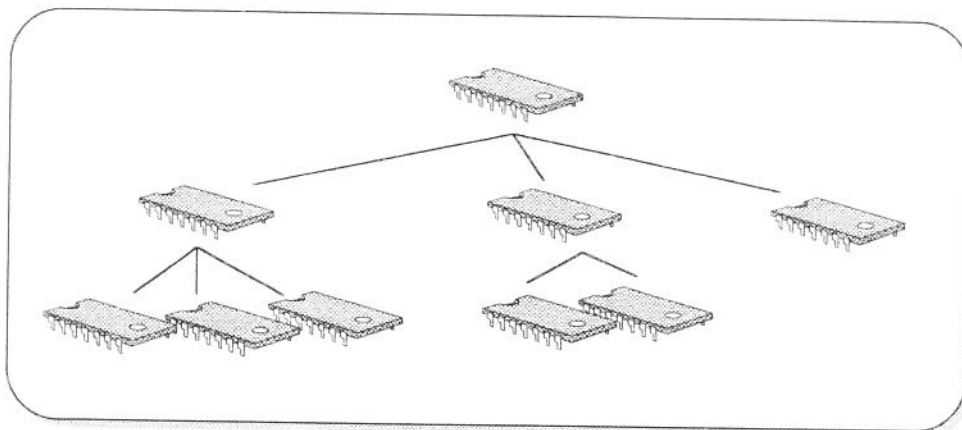


Figura 3.5. - O conceito de Herança. Uma classe primária origina diferentes subclasses. Estas subclasses podem herdar algumas características das superclasses e, adicionalmente, incorporar outras características próprias.

3.2. Contextualizando os Processos :

Processo é uma série sistemática de ações, que quando executadas provocam mudanças de estado num sistema. Como mostrado na figura 3.6., um processo pode ser subdividido em vários passos, chamados aqui de operações. Desta forma, uma operação é um passo de controle do processo completo. Por sua vez, uma operação pode ser decomposta num conjunto de condições e ações; quando uma ou mais destas condições se torna verdadeira, a ação correspondente é executada.

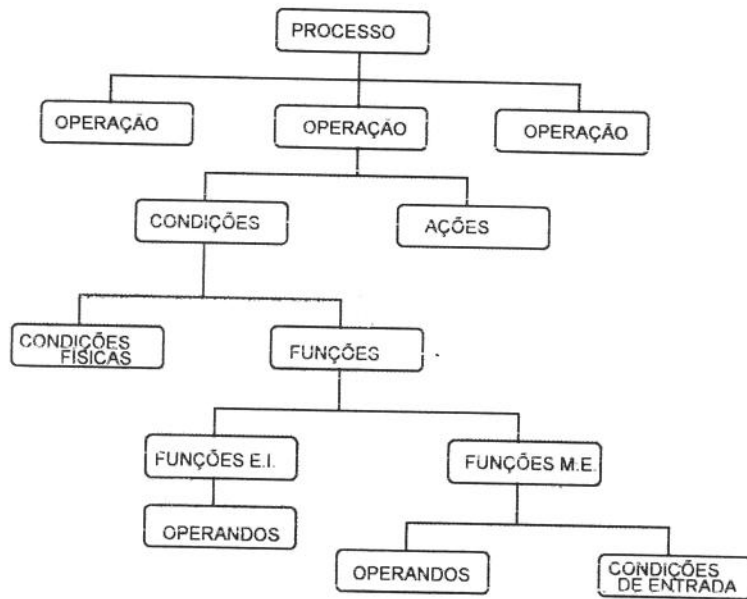


Figura 3.6. Divisão dos processos

As condições de uma operação podem consistir de condições físicas e de funções. As condições físicas estão associadas ao estado de entrada de dispositivos de estados discretos, tais como chaves de fim de curso, fotodetectores, botões, relés de controle interno, etc.

Às vezes pode se tornar necessária a existência de uma função para realizar o controle de uma operação. Algumas destas funções podem ser funções de contagem, funções de temporização, funções do tipo aritmético, funções de manipulação de bits, etc. Entretanto, uma função pode possuir apenas uma condição de entrada para habilitar sua execução, por exemplo, a função que é usada para somar dois números existentes em duas localizações de memória, pode ser considerada uma função de entrada individual (E.I. no diagrama), pois quando a condição dada se tornar verdadeira, a adição dos números será realizada. Por outro lado, as funções de múltiplas condições de entrada (M.E. no diagrama), requerem uma ou duas condições de entradas adicionais. Estas condições de entrada múltiplas podem ser requeridas para desabilitar os operandos da função e/ou habilitar ou inicializar a função. A função de contagem é um exemplo comum de função com múltiplas condições de entrada, pois são requeridas, no mínimo, duas condições de entrada para habilitar o contador.

As ações de uma operação são normalmente a alteração do estado de um dispositivo de saída, ou a execução de uma instrução interna. Os dispositivos de saída incluem dispositivos tais como: motores, alarmes, luzes, válvulas, solenóides, etc. As instruções internas correspondem a relés de controle interno, relés de controle mestre, fim de execução incondicional, etc..

3.3. Especificação de Sistemas de Manufatura Discreta e Gramáticas de Casos

Utilizando-se das definições do ponto anterior, podemos estabelecer uma descrição para o controle de sistemas de manufatura discreta, nosso domínio. O controle de um sistema de manufatura discreta pode ser definido como a "aplicação de uma seqüência pré-planejada de ações, para que aquilo que se considera como o objeto de controle atinja certos objetivos" (MIYAGI, 1991). Em outros termos, o controle corresponde à execução de uma série de operações conforme um procedimento pré-estabelecido ou numa lógica fixa governada por um conjunto de condições.

A partir deste conjunto de definições de controle de sistemas de manufatura discreta (SMD) é claro que as operações que se sucedem umas a outras e as transições (condições satisfeitas) que governam dita seqüência são os elementos centrais na especificação de um sistema deste tipo e dos seus procedimentos de controle.

Voltando atrás nas definições, uma operação, pode ser definida como um conjunto de elementos que quando ativados provoca ações caracterizando uma etapa de um processo. A ativação sucessiva de cada estado determinará a seqüência de funcionamento de um sistema na sua totalidade. Transição, é a combinação de eventos que condicionam uma mudança de estados num sistema. Isto é, a transferência na execução de uma operação para uma outra provocando alterações no estado e, portanto no comportamento do sistema.

As principais características do sistema de eventos discretos podem ser definidas como a seguir:

- O sistema trabalha com um número finito de estados e eventos discretos.
- O controle de SMDs envolve o controle qualitativo
- A estrutura de controle deste tipo de sistema não é necessariamente em malha fechada.
- Os comandos de controle possuem um número finito (muitas vezes binário) de informações, isto é, informações discretas e/ou digitais.

Geralmente um modelo para representar um sistema de manufatura discreta é construído recriando eventos e atividades as quais tomam lugar no sistema modelado. Portanto, é primordial identificar tais elementos nas descrições dos sistemas. BRUCE (1975) realizou uma revisão completa das técnicas usadas para a identificação de eventos a partir de descrições textuais usando Linguagem Natural. Estas técnicas possuem alguns aspectos que são aplicáveis na identificação dos eventos que retratam o funcionamento de sistemas de manufatura discreta, e portanto, úteis para os objetivos deste trabalho. Alguns destes aspectos serão discutidos a seguir.

Segundo SU e KACHITVICHNUKUL (1989) as gramáticas de casos, analisadas no capítulo 2, fornecem um meio eficaz para a identificação e categorização de verbos de acordo ao tipo de relação que eles mantêm com os outros elementos de uma sentença. BRUCE (1975) assegura que um conjunto pequeno de relações pode ser usado para definir os eventos como configurações que facilitam os processos de análise sintático e entendimento de sentenças.

Chamados de *referenciais de casos* (ou *Case Frames*), estes indicam os conjuntos de relações de casos que podem existir entre um verbo e os outros elementos gramaticais de uma sentença. Segundo Fillmore (1971) estas relações entre um verbo e os elementos gramaticais são similares às existentes, em particular, entre operações e os estados existentes numa descrição de um processo, portanto, este princípio poderia ser aplicado na modelagem textual de sistemas de manufatura discreta. No caso de uma especificação objeto orientada para um sistema de manufatura discreta, esta relação materializa-se através do encapsulamento que existe entre os objetos (ou instâncias) e os seus métodos, bem como na relação que existe entre os métodos e os seus parâmetros. Outra forma de interpretar isto, sob o ponto de vista do paradigma de objeto é a existência da relação mensagem-objeto, que pode equivaler à relação que existe entre AGENTE e o VERBO numa gramática de casos.

Adicionalmente junto com estes referenciais de casos, existe outro tipo de relações mais específicas que identificam os limites e extensões no espaço/tempo requeridos por verbos que indicam movimento, localização, duração, etc. Estas relações podem ser identificadas com certos parâmetros usados por alguns métodos de alguns objetos de manufatura.

Para um melhor entendimento dos referenciais de casos, analisaremos o verbo *abrir*. Um referencial de casos para este verbo pode ser descrito através de quatro relações:

A porta <i>aberta</i>	(O)
João <i>abriu</i> a porta	(A O)
O vento <i>abriu</i> a porta	(I O)
João <i>abriu</i> a porta com uma chave	(A O I)

- O : Objeto, a entidade que se mexe ou muda ou cuja posição ou existência está em consideração.
 A : Agente, o instigador de um evento.
 I : Instrumento, o estímulo ou causa física imediata que provoca um evento.

Assim pode-se representar o referencial de casos para o verbo *abrir* como:

[O (A) (I)]

Onde os caracteres entre parêntesis são opcionais, e podem aparecer ou não numa sentença. Ou seja, para este referencial de casos, o verbo *abrir*, precisa no mínimo estar acompanhado de um objeto para conter algum significado útil. Neste ponto devemos nos deter numa característica muito importante dos referenciais de casos, a qual é o valor semântico das relações de casos. Normalmente quando está-se definindo um referencial de casos, deve-se individualizar as relações relevantes ao domínio da aplicação e centrar a atenção nas informações requeridas pelo modelo. Este conceito é conhecido como Dependência Contextual, ou seja o conjunto de casos que dá uma noção de importância para um contexto dado.

Existem alguns fatores adicionais relacionados aos referenciais de casos que servem para determinar as relações entre os verbos e os outros elementos gramaticais, como por exemplo: diferentes terminações verbais, o uso de preposições, a ordem dos símbolos gramaticais, fatores tais como gênero e número, etc.

Para descrever um SMD são identificadas seis categorias principais dentro de uma sentença, estas são : Entidade, eventos/atividades, atributos, localidade, estado e valor. Este agrupamento foi feito a partir de um referencial de casos utilizado em (SU e KACHITVICHNUKUL, 1989). A seguir apresenta-se cada uma destas categorias:

Entidade	Existem três tipos principais de entidades no domínio da manufatura discreta, uma é a peça que esta sendo manufaturada, outra é o elemento de transporte das peças e o último é a estação de trabalho, uma máquina ou um conjunto de equipamentos.
Eventos/at- vidades	Eventos podem ser caracterizados como a operação propriamente dita, o transporte da peça e a peça em espera
Atributos	Algum tipo de atributo que caracterize um evento ou entidade
Localidade	Lugar de origem ou de destino de uma operação de transporte
Estado	Normalmente utilizado para identificar o estado de uma entidade, por exemplo em uso, desocupado, com defeito, etc.
Valor	Valor de um atributo, normalmente numérico.

Uma outra metodologia aplicada à modelagem de sistemas de eventos discretos, usa a programação em lógica para construir modelos de simulação. Isto é feito através de uma linguagem de descrição de condições, baseado na idéia de que o comportamento de um sistema de eventos discretos pode ser representado como um conjunto de tuplas condição/ação as quais são análogas às cláusulas lógicas das Linguagens de Programação Lógica. Este princípio já foi usado por FUTO e GERGELY (1983) que desenvolveram uma base para simulação usando lógica, chamada TS-PROLOG. Neste sistema, todo é descrito usando cláusulas, as quais são contidas numa base de dados, e o controle destas cláusulas é feito por mecanismos de resolução de problemas.

Propõe-se combinar estes dois elementos : a gramática de casos e a programação lógica com o paradigma de orientação a objeto. A seguir é analisada uma relação entre as Gramáticas de Casos e os conceitos do Paradigma de Objeto.

3.4. Relação entre os Referenciais de Casos e o Paradigma de Objeto:

BOOCH (1986), um dos autores mais citados na área da especificação e análise orientada a objeto, criou uma metodologia para o desenvolvimento de modelos usando o paradigma de objetos. Nesta metodologia, a primeira fase é a identificação no sistema de objetos ou classes de objetos existentes no domínio do problema. Os seguintes são os passos da metodologia criada por Booch:

- Identificação dos objetos e os seus atributos.
- Identificação das operações executadas por cada um dos objetos.
- Estabelecimento da interface de cada um dos objetos

- Implementação de cada objeto.

O primeiro passo consiste na identificação dos objetos e os seus atributos. Isto envolve, segundo o autor, o reconhecimento dos atores principais, agentes e servidores existentes no domínio, além do seu papel no modelo. Frequentemente, os objetos identificados nesta etapa derivam dos substantivos usados na descrição do problema o sistema. ABBOTT (1983) fornece uma estratégia mais completa sobre a identificação dos objetos (e classes de objetos) a partir da especificação informal de um sistema. Segundo ele, substantivos comuns equivalem ao nome de uma classe de objetos. Nomes próprios equivalem a nomes de instâncias de objetos. Substantivos abstratos (incluindo unidades de medidas) servirão para indicar características limitantes ou agrupamentos específicos para objetos ou classes de objetos. Outro trabalho que sugere uma classificação para os objetos físicos presentes num sistema de manufatura é o de SU e KACHITVICHNUKUL (1989). Estes podem ser máquinas ferramentas, equipamentos de manuseio de materiais, e peças. Este trabalho propõe mais duas categorias identificadas como dispositivos de medição e sensoramento e equipamentos de interação com o exterior do sistema. Estas duas categorias foram adicionadas devido a que foi identificada uma limitação no trabalho do Su e Kachitvichnukul, principalmente porque este está direcionado à construção automática de modelos de simulação para sistemas de manufatura, e não à especificação de sistemas de manufatura com o objetivo de definir lógicas de controle que governem o seu funcionamento.

Junto com isto, alguns atributos podem ser capturados nesta etapa através da identificação dos adjetivos existentes na descrição informal. Estes atributos serão facilmente relacionados com os objetos ou classes determinadas, pois sempre acompanham os substantivos na descrição informal. Estes atributos foram subdivididos nas seguintes categorias: quantidade física, identificador de uma instância de um objeto, capacidade de processamento ou manuseio de algum dispositivo (objeto) e duração de uma operação.

O passo seguinte, a identificação das operações executadas por cada um dos objetos, serve para caracterizar o comportamento de cada objeto. Claramente estas operações também podem ser extraídas da especificação informal do sistema, e derivam dos verbos usados junto com os substantivos achados durante a etapa anterior. Isto permite uma ligação facilitada entre objetos e operações, visando já a construção de estruturas que encapsulam,

atributos e operações (objetos). Os verbos denotam ações e/ou ocorrências. As ações passíveis de serem executadas num sistema de manufatura podem ser : processar (no amplo sentido), inspecionar e/ou transportar ou deslocar. Já este agrupamento fornece alguma direção sobre os tipos de verbos que poderão ser usados numa descrição para um sistema de manufatura, e de como estes podem ser relacionados com os métodos (ou serviços) de um objeto.

3.5. O Paradigma de Objeto aplicado à Especificação de Sistemas de Manufatura:

Os Sistemas de Manufatura consistem em duas partes principais : um conjunto de recursos que requerem ser controlados (ou apenas administrados), o qual chama-se de *entidade controlada* ou *parte operativa* e o *controlador* ou *comando* (entidade que realiza as tarefas de controle e/ou administração). Do ponto de vista do controlador, a entidade controlada forma o meio ambiente no qual ele deverá operar. Ele receberá informação sobre o ambiente através de "*sensores*" e enviará respostas (ou ações) a estas informações para alterar o estado do meio através de "*atuadores*". A especificação da entidade controladora do sistema pode ser equacionado através do mapeamento das entradas e saídas do sistema. A construção do modelo de especificação pode assim, ser desdobrado nos seguintes passos (JOANNIS e KRIEGER, 1992):

- Identificação dos objetos do mundo real que são requeridos pelo modelo.
- Identificação dos eventos no ambiente, os quais requerem uma resposta (ação) do comando.
- Identificação das operações requeridas dos objetos para executar estas respostas.
- Definição da estrutura do modelo, por exemplo, hierarquia entre os objetos.

Para especificar completamente a entidade controlada, estes passos devem ser executados de forma iterativa. A melhor forma é começar pela especificação de um pequeno número de objetos principais (os quais correspondem aos "grandes objetos" do mundo real), e ir desdobrando estes objetos em "objetos menores" ou secundários, até que o sistema seja suficientemente detalhado de maneira a facilitar a sua implementação. Algumas aplicações

interessantes aplicadas à especificação de sistemas através do paradigma de objeto podem ser vistas em (MOURANT, 1992; DROLET e MOREAU, 1992; ULGEN e THOMASMA, 1990; LEVAS E JAYARAMAN, 1989; MIZE et al., 1992).

3.5.1. A Especificação de Sistemas de Manufatura:

Considerando todos os conceitos enunciados nos capítulos anteriores pode-se estruturar uma definição para sistema, sob um enfoque Orientado-a-Objeto. Assim, sistema será um objeto com características próprias, que o identificam como uma entidade única, com fronteiras definidas, e formado por um grupo de objetos que, através de interações, conduzem o objeto maior para o cumprimento do seu propósito final. Muitas vezes o propósito final é a simples missão de se manter ativo, ou seja, continuar sendo viável com o passar do tempo. Entretanto, na maior parte das vezes, existem outros objetivos principais para tal objeto maior (o sistema), por exemplo, produzir um bem ou serviço, prever algum acontecimento, agir em função de uma situação, etc.

Como na definição clássica de sistema, estes objetos maiores também estão compostos por elementos, que são, por sua vez, outros objetos com missões específicas, características particulares e relações com outros objetos através de *mensagens* que passam pelas *interfaces* destes objetos.

Outra analogia que pode ser feita em função dos conceitos clássicos de Sistema, é em função da hierarquia dos sistemas, ou seja um objeto pode ser parte de um objeto maior; por exemplo, um robô (um objeto per si) é parte de um objeto maior que chamamos de *célula de manufatura*, que por sua vez, é uma parte de um objeto ainda maior chamado *fábrica*.

Os componentes de um sistema, como já foi dito, são objetos; estes objetos podem ser *objetos estáticos*, ou seja, os objetos que estão presentes desde o início da operação do sistema até o desaparecimento final dele. Em contrapartida, existem os *objetos dinâmicos*, que apenas passam através do sistema, ou são criados por outros objetos com um objetivo específico, e que em algum momento podem ser destruídos pelo objeto criador, quando o objetivo perseguido for atingido. Exemplos comuns de *objetos dinâmicos* são itens de informação, materias primas que após um processamento formam parte de um produto semi-

acabado, etc. Estes objetos correspondem aos *componentes de fluxo* no enfoque clássico de sistema.

Num enfoque Orientado-a-Objeto, assim como no enfoque clássico da Teoria de Sistemas, um sistema pode alterar os seus estados (situações ou condições). Estas alterações são impulsionadas através de transições, ou condições a serem cumpridas pelo objeto. Estas mudanças de estados podem ser representadas através de *autômatos* ou *máquinas de estados*.

A seguir, no capítulo 4 é descrito o desenvolvimento da técnica de especificação de sistemas de manufatura, usando os conceitos até agora discutidos.

Capítulo 4

Técnica de Especificação para Sistemas de Manufatura (um enfoque orientado a objeto)

Visando a realização de um protótipo de programação automática para controladores lógicos programáveis (CLPs) usando a Técnica de Especificação Orientada a Objeto (TEOO), neste ponto, enfoca-se as aplicações de controle discreto usando estes equipamentos. Entretanto considera-se também as aplicações que usam a lógica e informações analógicas.

As tarefas de controle podem ser consideradas segundo várias camadas (figura 4.1.). Assim podem ser controladas as operações globais da fábrica, as operações das seções, das células, dos centros de trabalhos, dos equipamentos ou inclusive no nível dos mecanismos que compõem cada um destes equipamentos. O grau de detalhe irá depender dos objetivos perseguidos pelos responsáveis do sistema.

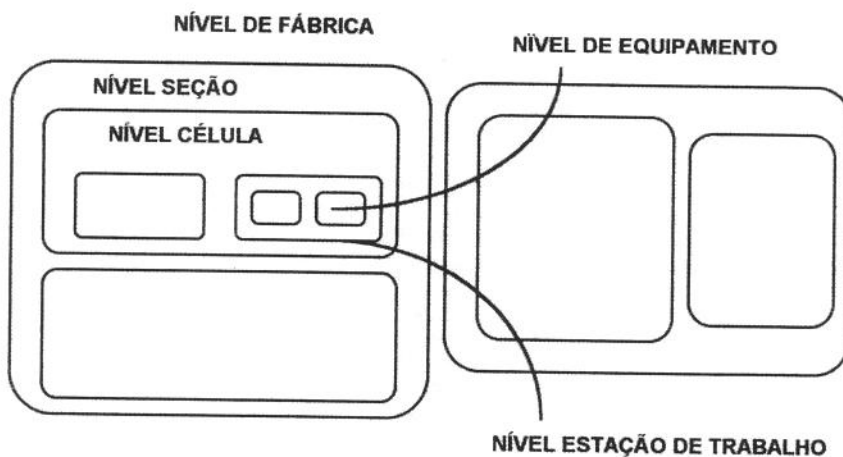


Figura 4.1. Esquema dos diversos níveis de controle para a manufatura

Este trabalho concentra a atenção apenas no controle das operações ao interior de uma célula de manufatura, uma célula de montagem ou de um sistema flexível de manufatura (que combina estes dois sistemas com sistemas de movimentação e gerenciamento de materiais). Um controlador neste nível está encarregado do

sequenciamento das operações ao interior da célula, coordenando as tarefas inter-equipamentos (HALANG E KRAMER, 1992).

Uma aplicação baseada no uso de CLPs consiste principalmente no projeto e/ou seleção de dispositivos mecânicos, tais como: equipamentos de processamento de materiais e de manuseio de materiais, e sensores, tais como: chaves de fim de curso e células fotoelétricas, e da confecção do programa de aplicação no formato adequado ao CLP escolhido.

Para auxiliar o processo de escrita de programas para automatizar tarefas de controle discreto usando CLPs pretende-se usar a Técnica de Especificação Orientada a Objeto (TEOO) para descrever a Lógica de Controle deste tipo de sistema. Com o uso desta técnica, o programador de CLPs terá sua produtividade aumentada incrementalmente. Isto, principalmente, com o uso de bibliotecas de objetos, nas quais ele encontrará representações abstratas dos dispositivos existentes nas aplicações reais. Estas representações abstratas, chamadas de objetos, encapsulam características e capacidades inerentes a eles. Assim, o programador poderá fazer uso destes objetos na descrição da estrutura e funcionamento de um sistema de Manufatura. Com uma descrição feita usando estes conceitos, o nível de abstração da descrição toda se verá elevado, chegando em níveis quase naturais (muito próximo à Linguagem Natural). Este fato provoca um aumento da reutilização de descrições existentes modificando-as e obtendo novas aplicações.

Com o intuito de obter um mecanismo preciso (objeto-orientado) para a definição de sistemas de manufatura, dos seus componentes e da interrelação entre eles, e finalmente estruturar as bibliotecas de objetos nas quais a técnica está baseada, foi realizado um levantamento na literatura dos dispositivos mais frequentemente encontrados na Manufatura Discreta. Este levantamento serviu como base para a criação de Classes de Dispositivos. Estas classes agruparão elementos com características (chamadas de atributos no Paradigma de Orientação a Objeto) e capacidades (Métodos) similares, de maneira a obter uma hierarquia na qual a Técnica de Especificação possa estar baseada.

Deve salientar-se neste ponto que esta hierarquia incorpora dispositivos que possuem diferentes níveis de complexidade. Isto é facilmente suportado pelo paradigma de objeto através do conceito de abstração. Para este tipo de aplicações o controlador não faz nenhum tipo de distinção entre os dispositivos conectados a ele, podendo ser eles dispositivos simples tais como, sensores ou dispositivos com uma complexidade muito maior como robôs ou centros de usinagem. Portanto, o nível de abstração dos objetos, bem como a topologia da hierarquia entre classes de objetos, irá diferir entre uma aplicação e outra. Isto dá uma grande liberdade na manipulação dos objetos existentes e na incorporação de novas

instâncias ao sistema. Este fato é considerado como uma das grandes fortalezas do sistema, pois permite ao sistema "crescer" ou "aprender" à medida que vai sendo usado para especificar novas situações, incorporando um conhecimento que facilitará novas definições (MENGA E MORISIO, 1989).

Os resultados obtidos a partir deste levantamento, são relatados a seguir.

Na figura 4.2. é mostrada uma hierarquia inicial que contém os elementos antes mencionados e a relação entre eles.

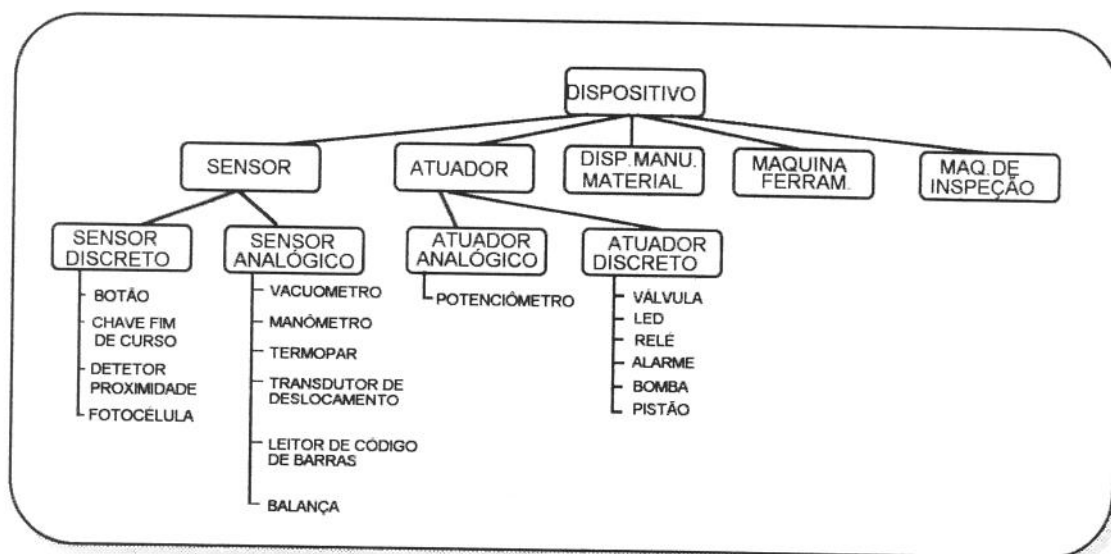


Figura 4.2. Hierarquia de Objetos para os dispositivos de Manufatura.

Esta hierarquia origina-se numa Classe de Objetos chamada de *Dispositivo de Manufatura*, dela estão derivadas outras cinco sub-classes identificadas como *Sensor*, *Atuador*, *Dispositivos de Manuseio de Material*, *Máquinas Ferramenta* e *Máquinas de Inspeção*. A primeira sub-classe, foi dividida em outras duas sub-classes. Os sensores que são capazes de transmitir informação do ambiente que apenas poderá tomar dois estados, esta subclasse foi chamada de *Sensor Digital*. A outra sub-classe, logicamente agrupa a todos os sensores que transmitem informação analógica para o controle do Sistema de Manufatura, esta sub-classe é identificada como *Sensor Analógico*. Algumas instâncias de objetos foram definidas na figura, de maneira a exemplificar cada uma destas subclasses. Classes equivalentes foram definidas para o caso da classe *Atuador*.

Para melhor exemplificar, podem ser considerados algumas outras instâncias, dependentes das outras classes existentes na hierarquia. Assim por exemplo, na classe *Dispositivo de Manuseio de Material*, podemos considerar três instâncias: AGVs, Robôs e Conveyers. Na classe de *Máquinas Ferramenta* podemos considerar instâncias tais como: Tornos, Fresadoras, Centros de Usinagem, etc.

O segundo passo, é determinar quais são os métodos relevantes para cada uma destas classes e que serão usados na aplicação em desenvolvimento. As tabela 4.1. e 4.2. mostram um resumo dos principais métodos relacionados com as classes *Sensor* e *Atuador*.

Tabela 4.1. Identificação de instâncias de objeto *sensor* e os seus métodos

CLASSE: Sensor	MÉTODO
Botão	apertado não_apertado
Chave_fim_de_curso	ativada desativada
Detector_proximidade	detecta não_detecta
Foto_célula	detecta não_detecta
Vacuometro	pressão de vacuo
Manometro	pressão
Termopar	temperatura
Transductor de Deslocamento	deslocamento longitudinal
Leitor de Código de Barras	número/código
Balança	massa

Do ponto de vista do desenvolvimento de um Gerador automático de software, um atributo importante para cada uma destas instâncias, é a porta de Entrada ou Saída mediante a qual estes dispositivos estão ligados ao CLP. Esta informação é colocada como atributo de cada objeto, e será usada pelo módulo gerador de software. Mais adiante será visto de que maneira esta informação é encapsulada no objeto.

4.1. Implementação da hierarquia usando Prolog:

Neste capítulo aborda-se a construção de um conjunto de predicados destinados à manipulação dos conceitos do Paradigma de Objeto, usando como ferramenta computacional o Arity Prolog. Aqui serão aplicadas algumas das técnicas estudadas no decorrer do projeto e discutidas anteriormente.

Tabela 4.2. Identificação de instâncias de objeto *atuador* e os seus métodos

CLASSE: Atuador	MÉTODO
Potenciometro	Potência
Válvula	Liga Desliga
LED	Liga Desliga
Alarme	Liga Desliga
Relé	Abre Fecha
Bomba	Liga Desliga
Pistão	Liga Desliga

Um programa ou descrição objeto-orientada é estabelecido em termos de um conjunto de objetos que enviam mensagens os uns aos outros. Cada objeto possui uma parte privada onde armazena os seus atributos e uma outra seção onde existe um conjunto de procedimentos, chamados de métodos, os quais implementam as capacidades de ação destes objetos.

Como já foi dito, estes objetos podem ser criados como extensões de maior detalhe de classes de objetos existentes, estas extensões herdarão os métodos e atributos dos objetos "pais" (conceito de herança do Paradigma de Objeto). Um objeto responde a uma mensagem executando um dos seus métodos. A mensagem determina qual método irá ser executado (o envio de uma mensagem para um objeto, é equivalente à chamada de um procedimento no paradigma de programação e análise estruturado).

Para traduzir estes conceitos em instruções em Prolog, é mostrado um pequeno exemplo com algumas das classes definidas anteriormente. Os objetos e os seus respectivos métodos escolhidos são os seguintes:

- Botão (apertado / desapertado)
- Chave_fim_de_curso (ativada / desativada)

- Alarme (liga / desliga)
- Válvula (liga / desliga)

Estas instâncias são escritas em prolog usando a forma

`object(Objeto,Métodos).`

onde Objeto é um termo, que identifica a instância e Métodos, é uma lista de termos de Prolog que especificam os Métodos encapsulados neste objeto. Estes termos tomam a forma de cláusulas (ou seja regras ou fatos). Os objetos anteriormente escolhidos podem ser escritos em Prolog da seguinte forma:

```
object(botão,[apertado,desapertado]).
object(chave_fim_de_curso,[ativada, desativada]).
object(alarme, [liga, desliga]).
object(válvula, [liga, desliga]).
```

Verifica-se neste exemplo, que existem algumas instâncias de objetos que possuem alguns métodos em comum; para usar o conceito de herança, estes métodos serão afixados então à classe superior, de maneira que, as instâncias de objetos definidas sob esta super-classe herdem estes métodos. Assim, pode-se afixar os métodos *liga/desliga* à classe *Atuador Digital*. Isto foi feito definindo um predicado escrito em Arity Prolog e chamado de "is_a". A seguir dois exemplos destes predicados:

```
is_a(valvula,atuador_digital).
is_a(alarme,atuador_digital).
```

E atribuindo os dois métodos no objeto *atuador_digital* extraíndo os mesmos das duas instâncias (válvula e alarme) que passam agora a herdar estes métodos. Isto é feito da seguinte forma:

```
object(atuador_digital,[liga,desliga]).
object(alarme, [ ] ).
object(válvula, [ ] ).
```

A construção da gramática para representar a técnica de especificação das lógicas de controle para sistemas de manufatura está em estreita relação com a parte pública dos objetos que a compõem. Esta parte pública, chamada de Interface, contém os nomes dos

métodos associados a cada um dos objetos. Portanto, a gramática deve conter como símbolos terminais os nomes de todos os objetos e os seus métodos. Em outras palavras, o dicionário de símbolos da gramática está formado pelos nomes dos objetos definidos no domínio e pelos procedimentos que cada um destes objetos pode executar como resposta ao envio de mensagens.

Para construir a gramática da Técnica de Especificação para o Gerador Automático de Software para CLPs precisa-se inicialmente identificar as condições nas quais o estado de um dispositivo poderá mudar. Existem diversos tipos de mudanças de estado possíveis. B. RAM e S.H.Y. LAI (1992) definiram quatro tipos de condições para mudança de estado, estes são:

- Uma mudança de estado momentânea do estado "on" para o estado "off", ou vice-versa, dada uma condição.
- Uma mudança de estado que tarda um determinado período de tempo (delay), ou seja "on delay" ou "off delay".
- Uma mudança de estado que perdura enquanto uma certa condição é verdadeira.
- Uma mudança de estado que é provocada por uma determinada condição e que se encerra (ou reverte-se) por uma outra condição.

No total são oito categorias possíveis de mudança de estados, pois devem considerar-se tanto as passagens do estado ON para o estado OFF e vice-versa. Estas categorias foram modificadas com o objetivo de se adaptarem melhor aos conceitos de gramáticas de casos e às configurações frasais que irão representar lógicas de controle para sistemas de manufatura discreta. Assim baseando-se nos quatro tipos comentados acima, passou-se a um conjunto de cinco categorias que abrangem de maneira mais adequada a especificação da estrutura de sistemas de manufatura e os seus processos. A seguir, estas novas categorias são enunciadas:

1. *Mudanças momentâneas de estado de ON para OFF (ou viceversa), provocadas pelo início ou final de um dado evento (ou combinação de eventos)*
2. *Mudanças de estado provocadas e apenas mantidas na existência de um dado evento (ou combinação destes).*

3. *Mudanças de estado permanentes de ON para OFF (ou viceversa), provocadas pela existência de um dado evento (ou combinação de eventos) sem depender da manutenção durante o tempo do(s) eventos(s).*
4. *Mudanças de estado provocadas pela existência de um dado evento (ou combinação destes) um determinado número de vezes.*
5. *Mudanças de estado gatilhadas pela ocorrência de um evento (ou combinação de eventos) durante um determinado intervalo de tempo.*

A combinação de eventos mencionadas acima corresponde a combinações lógicas de um conjunto de eventos individuais, estas combinações lógicas fazem uso da lógica de Boole. Assim os operadores lógicos tradicionais, E, OU, NÃO e XOR deverão ser incluídos na sintaxe da gramática. Estas inclusões são feitas na forma de símbolos terminais, ou seja, farão parte do dicionário de termos da técnica de especificação.

Para estruturar a gramática devem ser consideradas todas estas possibilidades combinatoriais. O objetivo desta gramática é refletir a representação do sistema real (estruturada em base a objetos e os seus métodos e escrita usando uma estrutura frasal ou textual) em representações internas. Para cada símbolo na especificação o processo identifica elementos individuais componentes do modelo tais como: objetos, métodos e/ou atributos. Estes elementos do modelo são então, agrupados num formato pré-definido. A forma como estes elementos estão organizados e como eles se relacionam os uns aos outros é determinada pelas regras da gramática.

Uma das características mais relevantes do paradigma de orientação a objeto é a facilidade de reutilizar modelos ou as descrições existentes. Isto aumenta a produtividade dos projetistas e programadores que já não precisam mais começar do zero para especificar e escrever um programa. Com o uso de um paradigma deste tipo eles poderão procurar por aplicações similares executadas anteriormente e com poucas mudanças obterão uma nova aplicação.

Provavelmente estas modificações derivem na criação de novas instâncias de objetos, que representem novos dispositivos presentes no sistema real. Esta adição de novos objetos à hierarquia permite a estruturação de bibliotecas de objetos, e à medida que estas vão se populando, e crescendo, estas irão ficando mais completas, provocando assim um aumento maior na produção de novas soluções.

O protótipo desenvolvido, possui a capacidade de incorporar novos objetos, na forma de instâncias de classes existentes e que, portanto, herdarão os atributos e métodos dos seus

ancestrais, como também podem incorporar novos métodos próprios da instância que está sendo criada.

Foi criado um predicado escrito em Prolog para a incorporação de novas instâncias de objetos.

Esta predicado, num primeiro momento requer do usuário o nome da nova instância e posteriormente, o nome da classe da qual ele origina-se. Finalmente, requer que o usuário forneça os métodos que o novo objeto encapsulará.

A seguir é mostrado o detalhe do predicado escrito em Prolog:

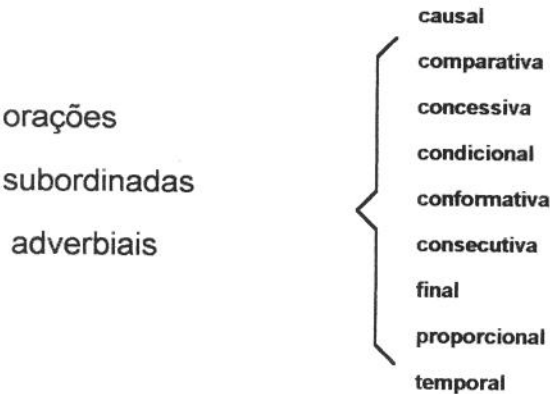
```
entra_novos_objetos:-
    write($Nome do Novo Objeto : ? $),
    nl,read(Nome),nl,
    write($Porta do Objeto : ? $),
    nl,read(Porta),nl,
    write($E uma instancia de : ?$),
    nl,read(SuperClasse),nl,
    write($Entre com os nomes dos novos métodos do objeto $),nl,
    write($separados por virgulas e um ponto final (usando só minúsculas) $),nl,
    entra_lista(Lista_Metodos),
    assertz((instance(Nome))),
    assertz((object(Nome,Lista_Metodos))),
    assertz((isa(Nome,SuperClasse))).
entra_lista(Lista_Metodos):-
    get0(Car),
    getrest(Car,Lista_Metodos).
getrest(46,[],):-!.
getrest(32,Lista_Metodos):- !,
    entra_lista(Lista_Metodos).
getrest(H,Letra,[Metodo|Lista_Metodos]):-
    getletters(Letra,Letra,Proximo_Car),
    name(Metodo,Letra),
    getrest(Proximo_Car,Lista_Metodos).
getletters(46,[],46):-!.
getletters(32,[],32):-!.
getletters(Letra,[Letra|Letra],Proximo_Car):-
    get0(Car),
    getletters(Car,Letra,Proximo_Car).
```

4.2. Gramática da Técnica de Especificação:

Foi feito um estudo em duas grámaticas da lingua portuguesa (CEGALLA, 1980; CUNHA, 1978) para definir a morfologia das sentenças que serão usadas para a descrição das lógicas de controle dos sistemas de manufatura. Este estudo possibilitou a determinação das entidades gramaticais que comporão cada uma destas sentenças, descrevendo as categorias mencionadas acima, e as regras que as governam. Também foram determinados os referenciais de casos que representam os verbos escolhidos (métodos dos objetos).

Como se comentou no capítulo 3 deste trabalho, um processo pode ser descrito como um conjunto de operações, as que por sua vez estão compostas de uma ou mais condições e de uma ou mais ações. Gramaticalmente, estas condições e ações podem ser descritas através de uma dupla de orações do tipo subordinadas. Isto é, cada descrição de uma operação consta de uma oração principal e de uma ou mais orações dependentes ou subordinadas. Segundo CEGALLA (1980), uma oração subordinada é a que depende de uma oração principal: serve-lhe de termo e completa-lhe o sentido. Normalmente as orações subordinadas se apresentam ligadas por um conectivo subordinativo, (geralmente uma conjunção). Quando este conectivo subordinativo está escrito de forma explícita na oração, se diz que ela está na forma desenvolvida, e quando ela está implícita, a oração chama-se reduzida.

Dentre as orações subordinadas existem três tipos diferentes, as substantivas, as adjetivas e as adverbiais. Estas últimas são as que interessam para este trabalho, pois através delas pode-se expressar circunstâncias de tempo, modo, fim, causa, condição, hipótese, etc. Estas orações subordinadas adverbiais estão divididas em categorias, segundo o tipo de conjunção subordinativa que as encabeça, assim tem-se as seguintes categorias:



Feita uma análise sobre quais destas categorias se adequam melhor na definição das operações de um sistema de manufatura, foram escolhidos três tipos orações subordinadas

adverbiais, a saber: as causais, que exprimem causa, motivo ou razão; as temporais, que indicam o tempo em que se realiza o fato expresso na oração principal; e as condicionais, que expressam condição ou hipótese. Vejamos alguns exemplos para estes três tipos de orações subordinativas adverbiais (O.S.A):

- Se a fresadora começar a usinar, acender a luz vermelha. (condicional)
- Visto que a fresadora iniciou a usinagem, ascenda a luz vermelha (causal)
- Não desligar a turbina principal, enquanto a bomba secundária estiver ligada (temporal)

A seguinte tabela mostra alguns exemplos de conjunções subordinativas que são usadas pela gramática da técnica de especificação desenvolvida:

Causais	porque, dado que, visto que, já que, uma vez que, desde que, pois, etc.
Condicionais	se, caso, a menos que, a não ser que, salvo se, etc.
Temporais	enquanto, quando, logo que, depois que, etc.

A seguir são descritas as sentenças definidas para cada uma das categorias de mudanças de estado descritas no capítulo anterior. Esta descrição tem sido escrita utilizando uma notação criada especificamente para este trabalho com o objetivo de facilitar o entendimento da sintaxe de cada categoria. Esta notação foi chamada de Equação Sintática, e que facilitou a tradução posterior de toda a gramática da técnica para a notação Backus-Naur e a posterior implantação em Prolog (usando as DCGs).

1. Mudanças momentâneas de estado de ON para OFF (ou viceversa), provocadas pelo início o final de um evento.

Estas mudanças momentâneas de estado de um ou mais dispositivos é provocada pelo início ou término de um evento.

As estruturas permitidas pela Técnica de Especificação para esta categoria são as seguintes:

TIPO A : CONJUNÇÃO + (AGENTE + VERBO_AUX + VERBO) + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO + OBJETO

TIPO B : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

TIPO C : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO + [E/OU + (AGENTE + NEGAÇÃO + VERBO_AUX + VERBO).....]

TIPO D : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

O VERBO_AUX são palavras reservadas da técnica que denotam os eventos que condicionam as ações a serem executadas. Estes verbos são: **COMEÇAR, INICIAR, DAR COMEÇO, DAR INÍCIO, ACABAR, TERMINAR, PÔR FIM E PÔR TERMINO.**

O AGENTE é um objeto de manufatura pertencente à alguma classe definida e já incorporada à gramática. O VERBO deve ser algum método próprio do objeto especificado em AGENTE ou herdado de uma classe ancestral. A forma de este verbo estará restrita a algumas poucas possibilidades, as quais procedem em orações do tipo que está sendo utilizado, podendo ser o Infinitivo, o Gerúndio, o Particípio, ou a forma nominal, chamado às vezes de verbo substantivado. Dependendo do tempo verbal que estiver sendo usado no VERBO_AUX, as conjunções são aplicáveis ou não. Assim para cada uma das três estruturas definidas acima existirão três tempos verbais que poderão ser usados, e para cada um deles existirá um conjunto pré-definido de conjunções aplicáveis. As reticências significam que pode existir qualquer número de vezes a estrutura: E/OU + (AGENTE + VERBO_AUX + ARTIGO + VERBO). O símbolo "/" indica que apenas um dos dois símbolos pode ser usado. Os colchetes indicam opcionalidade dos símbolos usados no interior deles. A NEGAÇÃO apenas indica a palavra NÃO. A seguir apresenta-se em detalhe a construção de cada um dos três tipos enunciados acima:

TIPO A:

Este tipo foi subdividido em três grupos diferentes:

i) Quando o VERBO_AUX estiver na forma de futuro do presente do modo subjuntivo. Neste caso as conjunções aplicáveis são: assim que, quando, se, sempre que, depois que, logo que, uma vez que. As formas verbais que podem ser utilizadas neste caso são: **COMEÇAR, INICIAR, DER COMEÇO, DER INÍCIO, ACABAR, TERMINAR, PUSER FIM E PUSER TÉRMINO**

ii) No segundo caso o VERBO_AUX esta no presente do modo subjuntivo:

COMECE, INICIE, DÊ COMEÇO, DÊ INÍCIO, ACABE, TERMINE, PONHA FIM E PONHA TÉRMINO.

As conjunções que poderão ser usadas neste caso são apenas duas: **CASO, CONTANTO QUE** e **DESDE QUE**.

iii) O último caso é quando o VERBO_AUX estiver no pretérito perfeito do modo indicativo, ou seja: **COMEÇOU, INICIOU, DEU COMEÇO, DEU INÍCIO, ACABOU, TERMINOU, PÓS FIM** e **PÓS TÉRMINO**

Sendo as conjunções que se utilizam nesse caso: **DADO QUE, VISTO QUE, JÁ QUE** e **UMA VEZ QUE**.

TIPO B:

No segundo tipo, o VERBO_AUX pode estar em dois tempos verbais diferentes:

- i) no presente do modo subjuntivo, ou
- ii) futuro do presente do modo subjuntivo.

E as conjunções que podem ser usadas são : a não ser que e a menos que, para o grupo i) e salvo se para o grupo ii). A NEGAÇÃO é a palavra **NÃO**.

TIPO C:

O terceiro tipo contempla a simples negação dos grupos ii) e iii) do *TIPO A*. Isto é feito antepondo a palavra reservada "**NÃO**" ao VERBO_AUX e ao verbo da segunda oração.

TIPO D:

No último tipo o VERBO_AUX pode estar apenas no futuro do presente do subjuntivo e as conjunções que podem ser usadas são apenas duas : **SE** e **ENQUANTO**. A negação é feita antepondo a palavra reservada "**NÃO**" aos VERBO e VERBO_AUX.

Em relação ao VERBO da oração principal, e como foi mencionado anteriormente, este verbo deve ser um método relacionado ao objeto especificado como agente na mesma oração. Agora bem, este verbo pode tomar duas formas diferentes: infinitivo, acompanhado de uma preposição, por exemplo: a **USINAR**, a **SE MEXER**, a **OPERAR**, etc., ou na forma nominal (substantivada), acompanhado de um artigo, por exemplo: **A USINAGEM**, **O**

MOVIMENTO, A OPERAÇÃO, etc. Isto só é válido no caso dos seguintes verbos: **COMEÇAR, INICIAR, TERMINAR E ACABAR**. No caso dos verbos **PÔR FIM, PÔR TÊRMINO, DAR INÍCIO** e **DAR COMEÇO**, o verbo só pode tomar a forma nominal e estar acompanhado de uma preposição e um artigo, por exemplo: **À USINAGEM, À OPERAÇÃO, AO MOVIMENTO**, etc.

Na segunda parte destas orações uma estrutura se repete nos quatro tipos mencionados: VERBO + OBJETO, que representa as ações a serem tomadas se as condições especificadas na primeira das orações subordinadas (ou principal) são satisfeitas. O verbo apenas pode estar na forma do infinitivo ou imperativo, e deve corresponder a um método pertencente ao objeto que o acompanha.

2. Mudanças de estado provocadas e apenas mantidas na existência de um evento dado.

A ocorrência de um evento provocará uma mudança de estado num ou mais dispositivos. O estado assumido pelo(s) dispositivo(s) é mantido até o evento associado acabar. Quando este evento não existir mais o estado do(s) dispositivo(s) voltará ao estado inicial. As equações sintáticas a seguir definem as estruturas das sentenças que podem ser usadas para descrever esta categoria de mudanças de estado.

TIPO A : CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO_1 + OBJETO.

TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

TIPO C : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

TIPO A :

Este tipo pode ser subdividido em quatro grupos dependendo da conjugação do VERBO_AUX, estes grupos são definidos a seguir:

i) as conjunções permitidas são : **ENQUANTO, SEMPRE QUE, QUANDO** e **SE**. O VERBO_AUX apenas pode ser a palavra reservada "**ESTIVER**" (futuro simples do subjuntivo).

ii) as conjunções permitidas são : **CASO, DESDE QUE** e **CONTANTO QUE**. O VERBO_AUX é a palavra "**ESTEJA**" (modo subjuntivo, presente)

iii) as conjunções permitidas são : **JÁ QUE, VISTO QUE, COMO, DADO QUE e UMA VEZ QUE**. O VERBO_AUX é a palavra reservada "**ESTÁ**" (modo indicativo, no presente).

iv) O quarto grupo é similar ao terceiro. A ordem dos componentes deste tipo de sentenças mudou e foram adicionadas algumas conjunções, a saber: **POIS, PORQUE e PORQUANTO**.

Para os quatro casos o VERBO da oração principal, e como nos casos anteriores, este verbo deve ser um método relacionado ao objeto especificado como agente na mesma oração. Agora bem, este verbo pode estar conjugado apenas na forma de gerúndio. Ou seja, se um objeto tem um método "**USINAR**", no caso de lógica deste tipo o verbo será "**USINANDO**".

TIPO B :

As conjunções permitidas por este tipo são: **A NÃO SER QUE, SALVO SE e A MENOS QUE**. O VERBO_AUX deve ser a forma verbal do presente do subjuntivo do verbo estar, ou seja, "**ESTEJA**". O VERBO da oração principal, ao igual ao do tipo A, deve estar no gerúndio, e deve ser um método associado ao AGENTE expresso na mesma oração. Por exemplo, **USINANDO, OPERANDO**, etc.

TIPO C:

O terceiro tipo contempla a negação dos grupos ii), iii) e iv) definidos no TIPO A. Estes podem ser negados antepondo a palavra reservada "**NÃO**" ao VERBO da oração subordinada e ao VERBO_AUX da oração principal.

3 Mudanças de estado permanentes de ON para OFF (ou viceversa), sem depender da presença permanente do evento que gatilhou esta mudança.

A ocorrência de um evento pode provocar uma mudança de estado permanente num dispositivo ou em vários dispositivos. Esta mudança de estado só será revertida quando for satisfeita uma outra condição dada.

TIPO A : CONJUNÇÃO_1 + AGENTE_1 + VERBO_AUX_1 + VERBO_1 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO_3 + OBJETO + CONJUNÇÃO_2 + AGENTE_2 + VERBO_AUX_2 + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

TIPO B : CONJUNÇÃO_1 + AGENTE_1 + VERBO_AUX_1 + VERBO_1 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO_3 + OBJETO + CONJUNÇÃO_2 + AGENTE_2 + NEGAÇÃO + VERBO_AUX_2 + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

TIPO A :

Este tipo foi subdividido em quatro grupos, a saber:

- i) A CONJUNÇÃO_1, o VERBO_AUX_1 e o VERBO_1 cumprem todas as especificações descritas para os tres grupos do TIPO A da categoria 2 definida acima. Agora bem, a CONJUNÇÃO_2 pode ser : a menos que, a não ser que ou até que. O VERBO_AUX_2 pode ser **COMEÇAR, INICIAR, DAR COMEÇO, DAR INÍCIO, ACABAR, TERMINAR, PÔR FIM e PÔR TERMINO**, no presente do modo subjuntivo.
- ii) Idem ao grupo i) , com a única diferença de o VERBO_AUX_1 estar no presente do subjuntivo, ou seja "**ESTEJA**".
- iii) Idem ao grupo i) , com a única diferença de o VERBO_AUX_1 estar no presente do indicativo, ou seja "**ESTÁ**".
- iv) Este grupo se diferencia dos anteriores em que a ordem dos elementos foi trocada. Isto é, a oração subordinada vai antes da oração principal, unidas por alguma das seguintes conjunções causais: **POIS, PORQUANTO, DADO QUE, VISTO QUE, JÁ QUE, UMA VEZ QUE, COMO, DESDE QUE e PORQUE**. O VERBO_AUX_2 cumpre com as regras especificadas no grupo i). O VERBO_AUX_1 é a forma verbal "**ESTÁ**", ou seja presente do indicativo.

TIPO B :

Este tipo foi subdividido em três grupos diferentes, a saber:

- i) O primeiro grupo consta de duas possibilidades:
 - Com as conjunções : **ENQUANTO, SEMPRE QUE, QUANDO e SE**. O VERBO_AUX_1 é a forma verbal "**ESTIVER**" (futuro simples do subjuntivo)

- Com as conjunções : **CASO, DESDE QUE** e **CONTANTO QUE**. O VERBO_AUX_1 é a forma verbal "**ESTEJA**" (presente do subjuntivo)

ii) O segundo grupo tem as seguintes conjunções: **DADO QUE, VISTO QUE, JÁ QUE, UMA VEZ QUE** e **COMO**. O VERBO_AUX_1 é a forma verbal "**ESTA**" (presente do indicativo).

Para todos estes grupos os componentes restantes cumprem as seguintes regras: o VERBO_1 é um método relacionado ao objeto que cumpre a função de AGENTE na oração, na forma do gerúndio, por exemplo, **USINANDO, OPERANDO**,etc. Agora bem, a CONJUNÇÃO_2 pode ser : **SE** e **ENQUANTO**. O VERBO_AUX_2 pode ser **COMEÇAR, INICIAR, DAR COMEÇO, DAR INÍCIO, ACABAR, TERMINAR, PÔR FIM** e **PÔR TERMO**, no futuro do presente do modo subjuntivo.

4. Mudanças de estado provocadas pela existência de um evento um determinado número de vezes

A ocorrência repetida de um evento um determinado número de vezes provocará uma mudança de estado no(s) dispositivo(s) definido(s) na instrução. O estado assumido pelo(s) dispositivo(s) é mantido até um novo evento associado à instrução acontecer. Este novo evento associado é chamado evento de rearme. As seguintes equações sintáticas definem as sentenças permitidas pela Técnica de Especificação para esta categoria de mudanças:

TIPO A : VERBO_1 + OBJETO + CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + NUMERO + UNID_CONTAB

TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + PREPOSIÇÃO + NUMERO + UNID_TEMPO

TIPO A:

i) conjunções: **DEPOIS QUE, SE** e **LOGO QUE**. O VERBO_AUX pode ser a palavra reservada "**TIVER**" ou "**HOUVER**" (futuro simples do subjuntivo).

ii) as conjunções são : **CASO, DESDE QUE** e **CONTANTO QUE**. O VERBO_AUX é a palavra "**TENHA**" e "**HAJA**" (modo subjuntivo, presente)

iii) conjunção permitida: **APÓS**. O VERBO_AUX é a palavra reservada "**TER**" ou "**HAVER**" (modo infinitivo).

Para os três casos o VERBO_2, e como nos casos anteriores, deve ser um método relacionado ao objeto especificado como agente na mesma oração. Note que este verbo deve estar conjugado no particípio. Ou seja, se um objeto tem um método "**USINAR**", no caso deste tipo de sentenças, o verbo será "**USINADO**".

TIPO B:

Neste tipo de sentença, existem dois grupos diferentes, a saber:

- i) No primeiro grupo, o VERBO_AUX apenas pode ser a palavra reservada "**TIVER**" ou "**HOVER**" (futuro simples do modo subjuntivo)
- ii) No segundo grupo, o VERBO_AUX é a palavra reservada "**TENHA**" e "**HAJA**" (presente do modo subjuntivo)

Como no tipo anterior, para os dois casos o VERBO_2, deve ser um método relacionado ao objeto especificado como agente na mesma oração. Note que este verbo deve estar conjugado no particípio. Ou seja, se um objeto tem um método "**USINAR**", no caso deste tipo de sentenças, o verbo será "**USINADO**".

Para os dois tipos anteriormente definidos o NÚMERO é um número qualquer podendo este ser inteiro ou real. Por sua vez, UNID_CONTAB, representa algum objeto que tenha sofrido a ação especificada pelo VERBO_2.

5 . Mudanças de estado gatilhadas pela presença de um evento um determinado período de tempo

Um dispositivo modificará seu valor só depois da ocorrência de um evento dado durante um determinado tempo. Enquanto o evento tiver um valor (lógico) verdadeiro, o estado do(s) dispositivo(s) será alterado e mantido. Quando o estado do evento tiver um valor negativo o estado do(s) dispositivo(s) voltará ao seu estado inicial.

TIPO A : CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + CONJUNÇÃO_2 + NUMERO + UNID_TEMPO , + VERBO_1 + OBJETO.

TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + CONJUNÇÃO_2 + NUMERO + UNID_TEMPO

TIPO A:

Este tipo está dividido em quatro grupos diferentes:

- i) As conjunções permitidas são: **ENQUANTO**, **SEMPRE QUE**, **QUANDO**, **SE**, **CASO**, **DESDE QUE** e **CONTANTO QUE**. O VERBO_AUX apenas pode ser "**TIVER**". O VERBO_2, deve ser um método relacionado ao objeto especificado como agente na mesma oração. Note que este verbo deve estar conjugado no particípio. Ou seja, se um objeto tem um método "**USINAR**", no caso deste tipo de sentenças, o verbo será "**USINADO**".
- ii) O segundo grupo permite as seguintes conjunções: **CASO**, **DESDE QUE** e **CONTANTO QUE**. O VERBO_AUX tem duas possibilidades, os verbos "**TENHA**" e "**ESTEJA**" (presente do subjuntivo). Estes dois verbos são combinados com os VERBO_2 conjugado no gerúndio, no primeiro caso, e no particípio, no segundo. Este verbo deve ser um método associado ao objeto especificado no AGENTE da oração. As CONJUNÇÃO_2 podem ser: **DURANTE** e **POR**, mas para o segundo caso, ou seja com o VERBO_AUX na forma de **TENHA**, pode se adicionar a CONJUNÇÃO_2 o verbo "**HÁ**".
- iii) O terceiro grupo, de maneira similar ao anterior pode ser dividido em dois casos;
 - com CONJUNÇÃO_1 sendo: **POIS**, **PORQUANTO**, **DADO QUE**, **VISTO QUE**, **JÁ QUE**, **UMA VEZ QUE**, **COMO**, **DESDE QUE** e **PORQUE**. O VERBO_AUX sendo a palavra reservada "**ESTÁ**" (presente do modo indicativo), combinado com um VERBO_1 conjugado em gerúndio. Este verbo deve ser um método associado ao objeto especificado no AGENTE da oração. As CONJUNÇÃO_2 permitidas neste caso são: **DURANTE**, **POR** e **HÁ**.
 - com CONJUNÇÃO_1 sendo apenas **DEPOIS QUE**. O VERBO_AUX sendo a palavra reservada "**TIVER**" (futuro simples do modo subjuntivo), combinado com um VERBO_1 conjugado em particípio. Este verbo deve ser um método associado ao objeto especificado no AGENTE da oração. As CONJUNÇÃO_2 permitidas neste caso são: **DURANTE** e **POR**.
- iv) As conjunções permitidas para CONJUNÇÃO_1 são: **DADO QUE**, **VISTO QUE**, **JÁ QUE**, **UMA VEZ QUE** e **COMO**. O VERBO_AUX é a palavra **ESTÁ** (presente do subjuntivo). A possibilidades para a CONJUNÇÃO_2 são: **DURANTE**, **HÁ** e **POR**.

TIPO B:

Este tipo está subdividido em dois grupos, a saber:

i) CONJUNÇÃO_1 apenas pode ser : ***SALVO SE***. O VERBO_AUX pode tomar duas formas:

- ESTIVER combinado com um VERBO_2 conjugado no gerúndio e com as seguintes CONJUNÇÃO_2 : ***POR, HÁ e DURANTE***.

- TIVER combinado com um VERBO_2 conjugado no particípio e com as seguintes CONJUNÇÃO_2 : ***POR e DURANTE***.

ii) CONJUNÇÃO_2 podendo ser: ***A NÃO SER QUE e A MENOS QUE***. O VERBO_AUX é "***ESTEJA***" (presente do subjuntivo) combinado com um VERBO_2 conjugado no gerúndio.

Nos dois tipos anteriores, e para todos os seus grupos, o NÚMERO pode ser um inteiro ou um real. Igualmente, a UNID_TEMPO, pode ser alguma das seguintes unidades de medida de tempo a seguir: ***MILISEGUNDO, SEGUNDO, MINUTO e/ou HORA***.

Um conjunto de quadros resumidos foi adicionado no apêndice para facilitar o entendimento global das sentenças permitidas pela Técnica Orientada a Objeto para a especificação de Sistemas de Manufatura.

4.3. Comentários Finais

A caracterização das instruções padrões para Controladores Programáveis descrita no próximo capítulo, serviu como meio de validação das categorias de mudanças de estado e uma tentativa de abranger a maior quantidade de dispositivos, mantendo uma simplicidade que se manifesta posteriormente numa programação facilitada ao usar a técnica.

A gramática definida neste capítulo foi facilmente traduzida para DCGs escritas em Prolog, especificamente pela relação direta que existe entre estas e a notação Backus Naur. Além das regras definidas nesta gramática, foram anexadas algumas regras para gerenciar os conceitos de encapsulamento e herança. Um conjunto de quadros resumos sobre as possibilidades combinatoriais da gramática é mostrado no apêndice deste trabalho.

A identificação e estudo das linguagens de programação para CLPs existentes no mercado brasileiro, fornece uma base sólida para a construção de módulos geradores de códigos para estas linguagens, e isto será discutido no capítulo seguinte.

Capítulo 5

Aplicação : Um sistema de Geração Automática de Programas para CLPs

5.1. Os Controladores Lógicos Programáveis:

Os Controladores Lógicos Programáveis (CLPs) estão sendo usados intensivamente para o controle de processos de manufatura discreta. Estes equipamentos foram criados na década de 60 com o intuito de substituir os grandes e complexos painéis de controle, baseados em relés. Com a evolução da microeletrônica e, conseqüentemente, o avanço no desenvolvimento de novas gerações de microprocessadores, os CLPs passaram a coordenar e controlar tarefas cada vez mais complexas, demandando deles uma alta capacidade e flexibilidade (COX, 1986). Além disso, as vantagens obtidas ao comparar a aplicação de CLPs com sistemas baseados em relés e outros baseados na automação fixa, indicam que esta tendência aumentará. Esta corrente têm a ver com a flexibilidade, economias e menores custos de manutenção, reduzindo-se também os tempos para a posta em marcha e operação dos sistemas baseados em CLPs. A tabela 5.1 mostra uma comparação entre o uso da automação programável e outros métodos menos flexíveis.

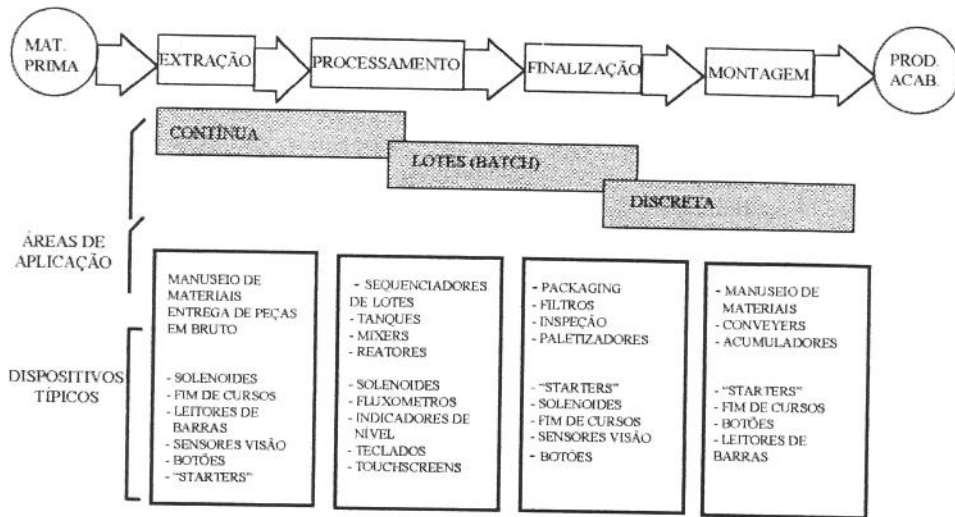


Fig.5.1. Um exemplo genérico das aplicações baseadas em CLPs (extraído de BONANOMI, 1994)

As aplicações baseadas em CLPs incluem atualmente três diferentes tipos de controle: contínuo, em lotes e discreto (BONANOMI, 1994). A figura 5.1. mostra um ciclo de produção genérico que poderia formar parte de uma instalação qualquer na indústria de processos ou de manufatura discreta. O ciclo é dividido em etapas, que mostram os tipos de controle e operações realizadas, e os típicos dispositivos de campo que realizam as operações.

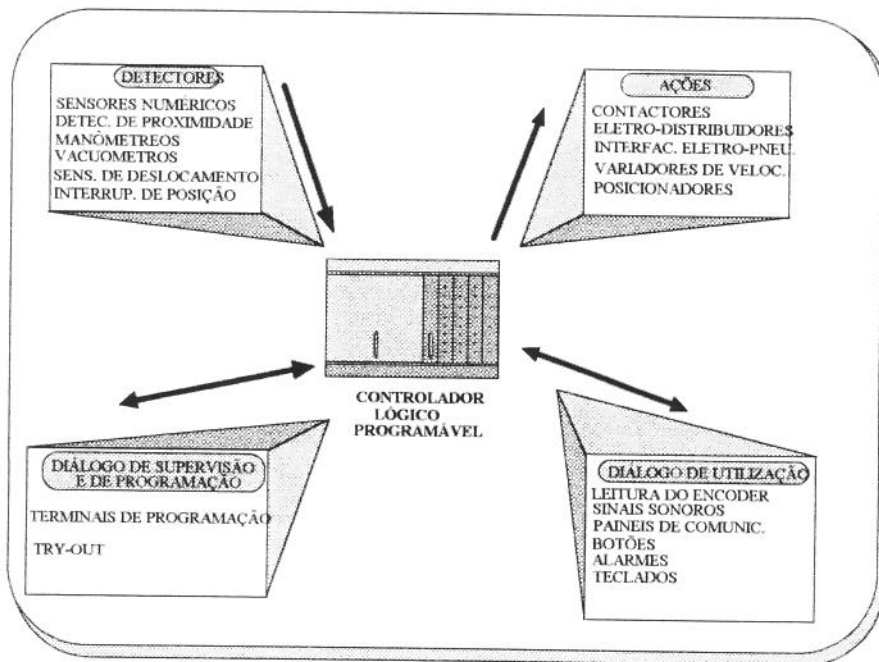


Figura 5.2. Funções de um CLP e os meios de ligação ao sistema automatizado (adaptado de BITTAR, 1993)

Assim, hoje em dia, linhas de transferência, robôs, equipamentos de manuseio de materiais, etc, podem ser controlados e integrados com facilidade, fornecendo grande flexibilidade nas operações. A figura 5.2 mostra as principais funções de um CLP e os elementos encarregados de ligar estas funções com o sistema automatizado (adaptação de BITTAR, 1993).

Tabela 5.1.- Comparação entre a automação programável e outros métodos de automação

	SISTEMAS BASEADOS EM RELES	LÓGICA FIXA	CONTROLE PROGRAMAVEL
Implantação e Modificações	Modificações conduzem a diversas alterações e aumentos nos tamanhos. Muitas vezes estas alterações não são possíveis por problemas de espaço dentro dos painéis.	Idem aos relés	As alterações se traduzem reprogramação sem necessidade de alterações na fiação.
Profissionais que têm acesso	Engenheiros Elétricos e Eletromecânicos	Técnicos e Engenheiros Eletrônicos	Todos os tipos de profissionais mencionados anteriormente, mais Eng. Mecânicos, especialistas em Instrumentação, e de Processamento de Dados.
Linguagem de Especificação e Comunicação	Diagramas Elétricos	Diagramas de Eletrônica	Listas de Instruções, Diagramas de Contatos, Function Block Diagrams, Cartas de Funções Sequenciais
Manuntenção	Os componentes precisam ser trocados. Existem problemas com a corrosão e agentes poluintes (pó, humidade, etc.)	Aqui não existem esses problemas, pois os componentes são de estado sólidos	Idem aos controles de lógica fixa
“Throughput”	Tempos para abrir e fechar contatos de 5 a 20 ms. Os tempos se somam se os componentes estão encadeados.	Tempos para abrir e fechar contatos são de 2 ms. Os tempos se somam se os componentes estão encadeados.	A diferença entre a entrada e a saída se estende de 2 a 5 ms

Apesar desta flexibilidade, o desenvolvimento de programas para um CLP é uma tarefa demorada e susceptível a erros. Isto aumenta consideravelmente os custos relacionados às áreas dedicadas a esta função, especialmente se se considerar instalações com grande quantidade de controladores lógicos programáveis. Esta situação se vê agravada

se se pensar que cada fabricante incorpora aos seus produtos uma linguagem de programação específica o que impossibilita o intercâmbio de software entre controladores reduzindo a portabilidade e o reaproveitamento dos programas de aplicação desenvolvidos (PERRON, 1992).

Assim, surge a necessidade de um formato neutro de programação, por meio do qual o programador possa se liberar de linguagens de programação específicas e realizar o desenvolvimento do software de controle num formato independente do controlador onde será implantado tal programa. Para satisfazer esta necessidade, foi desenvolvido um sistema de programação automática de software para controladores lógicos programáveis. Este sistema trabalha com um formato de especificação neutro baseado na técnica de especificação orientada a objeto (TEOO) descrita no capítulo anterior, onde o usuário define as variáveis e lógica de controle, e o sistema se encarrega de gerar como saída o programa especificado no código fonte do CLP selecionado. Este sistema possui uma estrutura totalmente modular garantindo a rápida incorporação de novas linguagens de programação de CLPs ao sistema. Como já foi dito, o sistema usa a técnica de especificação orientada a objeto (TEOO) para estruturar a lógica de controle dada pelo usuário, portanto independe do hardware (CLP) selecionado. Porém, a saída será específica para o CLP escolhido, e, em princípio, só é executado em tal equipamento. A metodologia usada aqui soluciona o problema, pois a técnica de especificação tem o papel de fornecer uma Interface entre o usuário e os diferentes CLPs. Este conceito é mostrado na figura 5.3. A área encerrada pelo retângulo tracejado representa a única parte do sistema que depende do CLP selecionado para ser programado.

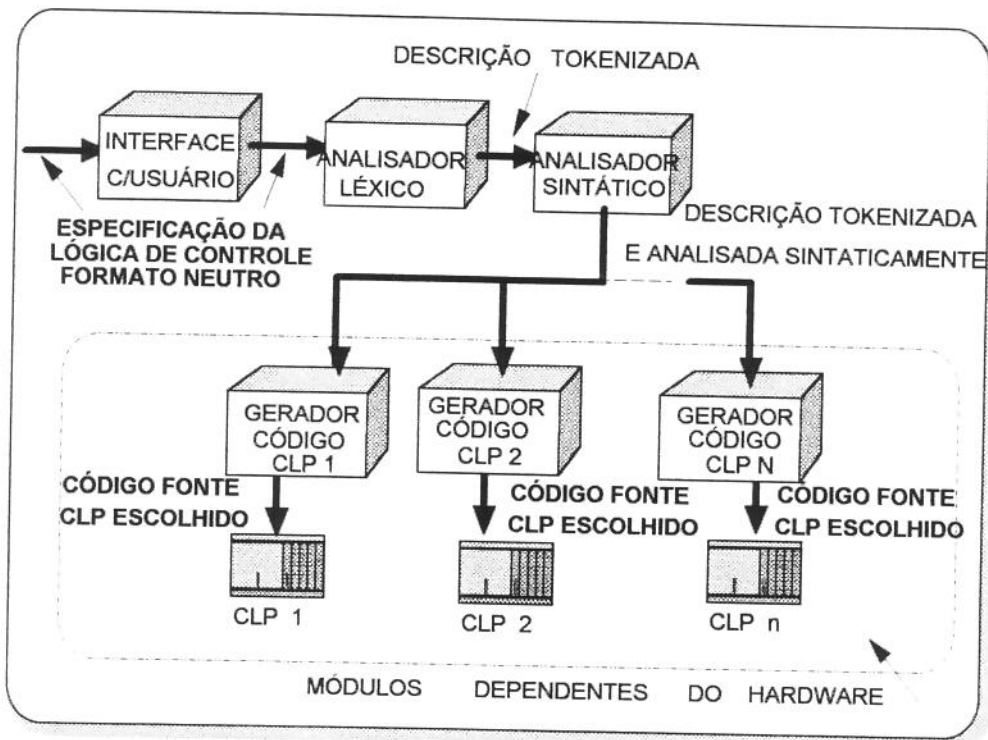


Figura 5.3. Conceito de Modularidade do Sistema - Partes que dependem do CLP selecionado.

5.2. Paradigmas de Programação de CLPs:

Existem diversas metodologias para a programação de CLPs. Cada fabricante incorpora uma ou mais destas metodologias aos seus equipamentos (FRENCH, 1990). A seguir são descritos brevemente os formatos de programação mais comuns:

5.2.1. Diagramas de Contatos (Ladder Diagrams):

Esta é uma variação da bem conhecida linguagem baseada na técnicas de representação de diagramas de relés. O elemento principal de um diagrama de contato é o “laço”. Cada laço é composto de contatos (entradas e saídas). As entradas representam dispositivos de sensoramento externo e/ou localizações internas da memória. As entradas estão agrupadas em arranjos de tipo seriais ou paralelos formando as condições de um determinado laço. No lado direito do laço se encontram as saídas as quais estão conectadas aos dispositivos externos ou a localizações internas da memória, correspondendo às ações do laço dado. Quando as condições são verdadeiras, as ações são realizadas, e os contatos definidos na seção das saídas do laço mudam os seus estados.

5.2.2. Listas de Instruções:

Esta é uma linguagem textual de baixo nível onde uma cada linha está formada por apenas uma instrução simples. Este tipo de linguagens é dedicada a programadores mais experientes, e é utilizada para realizar operações simples. Normalmente programas escritos com este formato, são diretamente interpretados pelo próprio CLP.

5.2.3. Textos Estruturados (Structured Text - ST):

Linguagens de programação de mais alto nível, que permitem a programação estruturada. Sua sintaxe lembra o Pascal. Estas linguagens incorporam operações baseadas na lógica de Boole e operações aritméticas, suportando um grande número de tipos de dados. À diferença dos programas escritos usando Listas de Instruções, programas escritos em ST precisam ser compilados antes de serem carregados no CLP.

5.2.4. Sequential Function Charts (SFC) :

Esta é uma linguagem do tipo gráfico que possibilita a representação diagramática de sequências de controle. Programas escritos usando SFC permitem ao programador decompor uma sequência em passos e transições. Os passos representam as ações a serem realizadas quando uma condição é satisfeita (FOUNTAIN, 1994). Uma transição define uma condição que deve ser verdadeira antes de um novo passo se ativar. Este formato está baseado na norma IEC 848, a qual é uma versão internacional da linguagem GRAFCET desenvolvida por pesquisadores franceses.

5.2.5. Function Block Diagrams:

Este é um outro formato gráfico que está sendo intensivamente usado por programadores europeus. Os elementos deste formato de programação são os blocos, os quais são ligados como se fossem componentes de um circuito. Os blocos de função são

entidades computacionais que têm entradas que produzem saídas de qualquer tipo, como se fossem circuitos integrados. Cada bloco de função é descrito como uma caixa retangular ligadas a outras conformando uma rede de blocos. A figura 5.4. mostra um simples exemplo dos três formatos gráficos descritos acima.

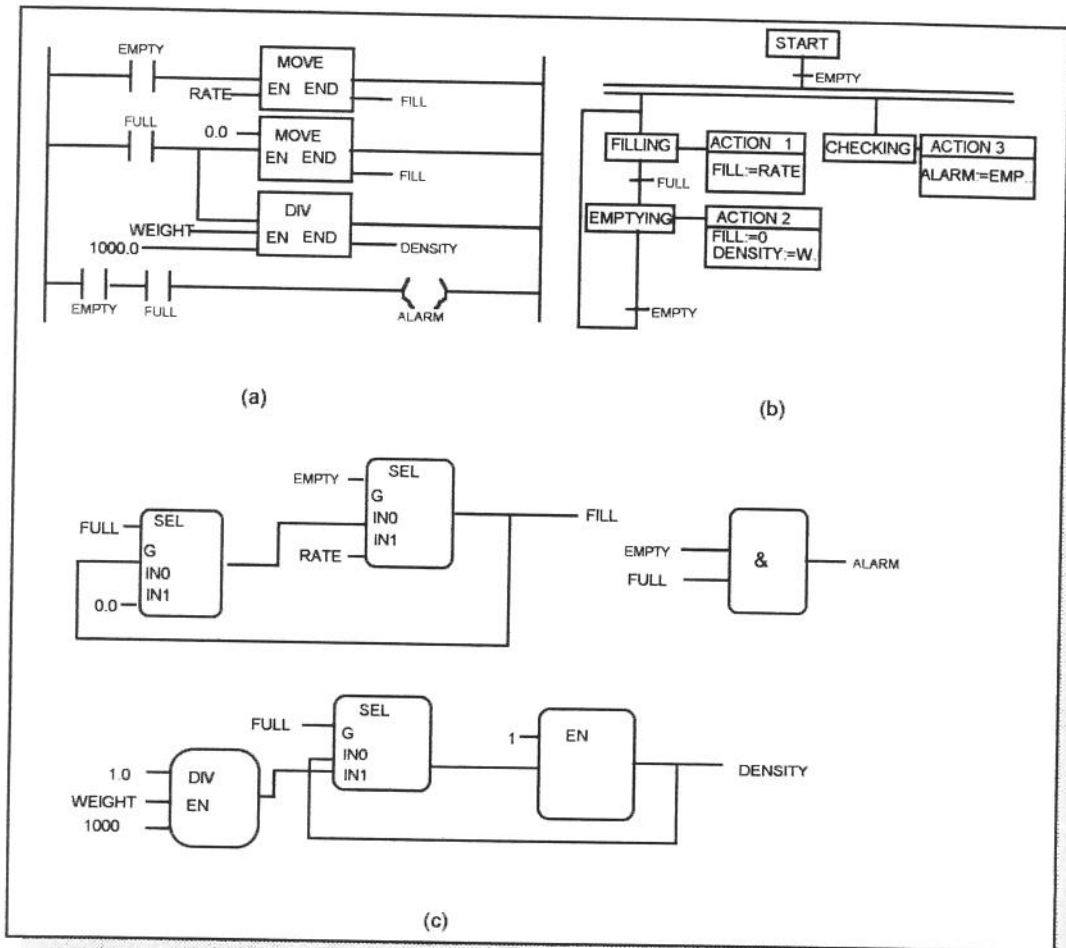


Figura 5.4. Exemplos de programação de CLPs usando formatos gráficos a) diagramas de contato b) Sequential Function Charts c) Function Block Diagrams

Uma série de instituições internacionais têm trabalhado para definir um padrão de programação para CLPs. Pode-se destacar as iniciativas da DIN, NEMA, IEC, etc. A figura 5.5. mostra um cronograma das tentativas de padronização para linguagens de programação para CLPs. Desafortunadamente a maioria destas iniciativas têm falhado porque existem tantas versões de linguagens de programação como fabricantes de CLPs no mercado, e a aceitação e assimilação de um padrão significaria um alto investimento tanto a nível econômico como no filosófico para a maior parte destes fabricantes. A última iniciativa de padronização, identificada com o número IEC 1131-3, foi liderada por uma série de

Instruções Sequenciais : 17 tipos

Instruções Aritméticas e de Aplicação : 44 tipos

Relógio Calendário

Ano, mes, dia, dia da semana, hora, minuto, segundo.

Dispositivos Periféricos

Programador por Instruções e diagrama ladder, programador gráfico portátil (HITACHI, 1994).

CONTROLADOR CLP- 5 ALLEN BRADLEY

O controlador CLP-5 é apresentado em duas versões:

CLP-5 / 15 (1785-LT)

CLP-5 / 25 (1785-LT2)

Programação por Diagramas Ladder

Programação Estruturada através do Controle Sequencial de Funções

Capacidade de Memória

Versão CLP-5/15 : 6K palavras

Versão CLP-5/25 : 13K palavras

Pontos de E/S

Versão CLP-5/15 : 512

Versão CLP-5/25 : 1024

Varredura de Programa

2ms/Kpalavras (lógica de bit)

8ms/Kpalavras (típico)

MODICON PROGRAMMABLE CONTROLLER 984(b) (MODICON, 1991)

Capacidade de Memória

CPU 24 bits

Mémoria de programas 32K/64K

Pontos de E/S

1024in/1024out

256in/256out

Varredura de Programa

0.75 ms/Kpalavra

Programação por Set de Instruções

Relays Instructions

Funções Aritméticas

Contadores e Temporizadores
 Instruções de Movimentação
 Instruções Lógicas.

PC SIMATIC S5-110A SIEMENS

Capacidade de Memória

Depende da CPU usada:

6ES5 900-7AA11,21 2K

-7AB11,21 2K

-7AC11,21 4K

Programação por Set de Instruções

Relays Instructions

Funções Aritméticas

Contadores e Temporizadores

Instruções Lógicas.

CONTROLADOR BCM 1086

O controlador BCM é o modelo de menor porte da linha BCM, ele apresenta em uma só unidade a CPU, display de teclado para programação, entradas e saídas digitais, relógio, comunicação serial, etc.

Pontos de E/S

Entradas Digitais : 16

Saídas Digitais : 12

Capacidade de Memória

16 Kbytes de EPROM para o usuário

A área total de memória RAM do controlador programável é de 8 Kbytes, dividida em três partes:

- Área de serviço do sistema operacional: 512 bytes
- Área de variáveis de E/S e atribuição : 2048 bytes
- Área de variáveis de atribuição livre : 5632 bytes

Programação por Set de Instruções: A estrutura fundamental de programação em linguagem BCM é o Diagrama de Estados.

Como já foi dito, foi feita uma análise nas linguagens de programação destes equipamentos, na tentativa de padronizar um conjunto de operações comuns à maioria destes equipamentos. Este conjunto padrão servirá para direccionar a formação da técnica de

especificação para o sistema de manufatura orientado por objeto. Foram detectados agrupamentos naturais entre estas instruções. Estes agrupamentos foram catalogados como: Instruções Básicas, Instruções de Tipo Aritmético, Instruções para Manipular Dados, e Instruções mais complexas (Controle de Processos). A seguir é feita a descrição para cada um destes grupos de instruções.

5.3.1. Instruções Básicas:

i) O primeiro grupo de instruções analisado, é o do grupo de instruções básicas-sequenciais e se refere ao exame de estado para dispositivos de entrada (em forma individual ou em grupos) como condição para um determinado valor de saída (seja este um valor de uma localização de memória interna ou o acionamento de um dispositivo de saída).

A seguir são feitas breves descrições explicativas para cada uma destas instruções básicas, do tipo sequencial.

(1) Examina Energizado: Quando um dispositivo de entrada (E) fecha seu circuito, o terminal de entrada conectado ao mesmo indica um estado energizado, fato que é registrado na memória (bit correspondente à entrada). Quando o controlador localiza uma instrução que usa este endereço a lógica para este tipo de instrução é verdadeira, tornando o valor da saída (S) verdadeira. De maneira análoga, quando o estado do dispositivo de entrada for desenergizado, a lógica da instrução será falsa, provocando a mudança de valor do dispositivo de saída de acionado para desabilitado.

(2) Examina Desenergizado: De maneira similar ao caso anterior, quando o dispositivo de entrada não é acionado, o estado é desenergizado, fato que é registrado no bit de memória correspondente. Quando o controlador localiza uma instrução deste tipo que usa o mesmo endereço de memória, e determina que a entrada (E) está desenergizada a lógica da instrução se torna verdadeira, habilitando a saída (S). Quando o dispositivo de entrada for acionado, a lógica da instrução se torna falsa, desabilitando a saída correspondente.

(3) Detecta Borda de Subida: Este tipo de instrução provoca uma alteração no estado de uma saída (S) de desabilitado para habilitado de duração de um scan (ciclo de varredura) quando o dispositivo de entrada (E) mudar de estado de não acionado para acionado.

(4) Detecta Borda de Descida: Caso equivalente à instrução anterior, que provoca uma saída de um pulso (duração de um scan) cada vez que o estado do dispositivo de entrada mudar de acionado para não acionado.

- (5) Ligação em série de contactos normalmente abertos: A saída é somente ligada quando ambas as entradas E1 e E2 estiverem em estado energizado. Se um dos dois ou ambos os dispositivos de entrada estiverem ou mudarem para estado desenergizado, a saída (S) será desabilitada.
- (6) Ligação em paralelo de contactos normalmente abertos: A saída é somente ligada quando ambas ou pelo menos uma das entradas E1 e E2 estiver em estado energizado. Se ambos os dispositivos de entrada estiverem ou mudarem para estado desenergizado, a saída (S) será desabilitada.
- (7) Ligação em série de contactos normalmente fechados: A saída é somente ligada quando ambas as entradas E1 e E2 estiverem em estado desenergizado. Se um dos dois ou ambos os dispositivos de entrada estiverem ou mudarem para estado energizado, a saída (S) será desabilitada.
- (8) Ligação em paralelo de contactos normalmente fechados: A saída é somente ligada quando ambas ou pelo menos uma das entradas E1 e E2 estiver em estado desenergizado. Se ambos os dispositivos de entrada estiverem ou mudarem para estado energizado, a saída (S) será desabilitada.
- (9) Energiza Saída com retenção: Instruções de saída retentiva. O dispositivo de saída conectado a este tipo de instruções será energizado assim que a lógica da linha de instruções se tornar verdadeira. Entretanto, se existir uma mudança na lógica da instrução de entrada, o estado do dispositivo de saída e do bit de memória correspondente se manterá abilitado, permanecerá assim, mesmo após as condições de entrada se tornarem falsas.
- (10) Desenergiza Saída com retenção: Esta instrução rearma (desabilita ou desenergiza) o bit de memória correspondente a uma saída, quando a lógica de linha de instruções se tornar verdadeira. Entretanto, se existir alguma mudança na lógica das instruções de entrada, o estado lógico do dispositivo de saída se manterá desligado.

Nota: As ligações de contactos em série e paralelo, na realidade podem combinar contatos normalmente abertos (NA) e contatos normalmente fechados (NF) na mesma instrução. Os exemplos aqui mencionados, apenas foram feitos para ilustrar o funcionamento das ligações em série e em paralelo.

ii) O segundo grupo de instruções básicas, catalogado como de temporização e contabilização, agrupa muitas tarefas para as quais, frequentemente, se utilizam relés de tempo ou dispositivos de temporização e de contagem de estado sólido.

As instruções de Temporizador e de Contador são instruções de saídas que são condicionadas por instruções de entrada. Os temporizadores contam intervalos de tempo e os contadores contam eventos. Na figura 5.6 são apresentados os diagramas de tempo para três tipos de operações pertencentes a este grupo:

(11) Temporizador na energização: Esta instrução pode ser utilizada para energizar ou desenergizar um dispositivo quando tiver transcorrido um intervalo de tempo pré-determinado (é chamado também de ON-delay timer). Quando as condições de entrada se tornarem verdadeiras, o temporizador começará a contar intervalos de tempo. Enquanto as condições permanecerem verdadeiras, ele incrementará o valor acumulado a cada intervalo contado. Quando o valor acumulado é igual ou maior que o valor pré-selecionado o temporizador pára de incrementar seu valor acumulado e energiza a saída. Se as condições de entrada se tornarem falsas antes da contagem atingir o valor pré-selecionado, o temporizador será desenergizado, e só reiniciará uma nova contagem quando as condições de entradas novamente se tornarem verdadeiras.

(12) Temporizador na Desenergização: Este tipo de instrução difere da anterior em vários aspectos: o temporizador na desenergização começa a temporizar um intervalo assim que as condições de entrada se tornarem falsas. Enquanto as condições se tornarem falsas, o contador continuará com a sua contagem. Quando a contagem do temporizador atingir o valor pré selecionado, a saída será desenergizada. São chamados também de OFF-delay timers.

(13) Contador: As instruções de contador contam transições de falso para verdadeiro, as quais podem ser causadas por eventos que ocorrem no sistema controlado. As instruções de contagem possuem um valor pré-selecionado e um valor acumulado. Quando as condições de entrada passam de falsa para verdadeira, o valor acumulado é incrementado em um, quando isto ocorre sucessivamente até o valor pré-selecionado, uma localização de memória, chamado de bit de executado, é habilitada, e a saída é energizada. Existem contadores do tipo crescente e do tipo decrescente, como também contadores crescente/decrescente podendo, este último funcionar tanto como o primeiro tipo ou como o segundo.

5.3.2.Outras Instruções:

Além das instruções estudadas acima existe um outro grupo importante de instruções destinadas principalmente à manipulação de dados, a operar aritmeticamente estes e à comparação de pares de dados. A maior parte das linguagens de programação de CLPs

possuem conjuntos de instruções pertencentes a estes tipos de instruções para complementar os cálculos básicos da lógica combinatorial que se preocupa principalmente de examinar os estados dos dispositivos de entrada, combiná-los usando alguma tipo de equação lógica e alterar o estado de um ou mais dispositivos de saída. A seguir é feita uma breve descrição destes tipos de instruções.

Instruções para manipular dados:

Estas transferem ou comparam informações armazenadas em bytes na memória de dados de um CLP. Para transferir dados de um endereço de memória para um outro existe uma instrução do tipo MOVE a qual "copia" o valor existente num dado endereço de memória para um outro. A instrução COMPARA verifica se o dado existente num endereço de memória determinado é igual, maior ou menor que o dado existente num outro endereço de memória. Isto permite que o CLP realize diferentes funções básicas baseadas num dado de referência (PEIXOTO, 1993).

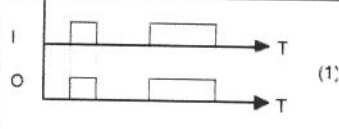
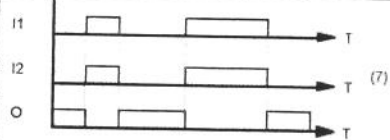
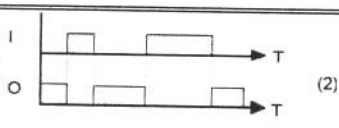
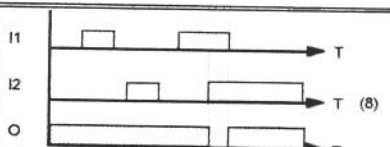
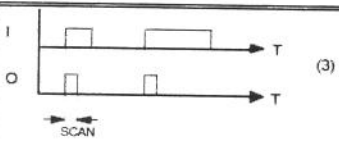
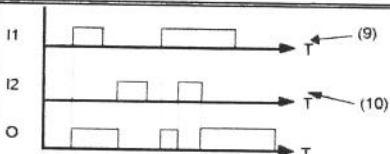
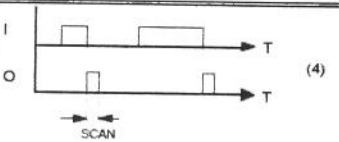
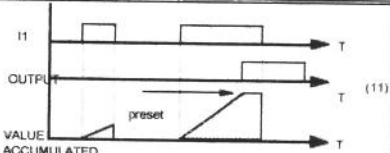
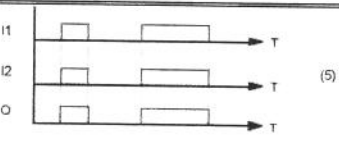
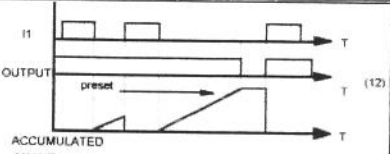
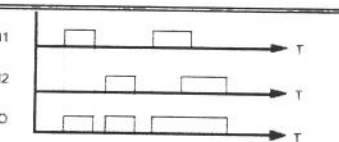
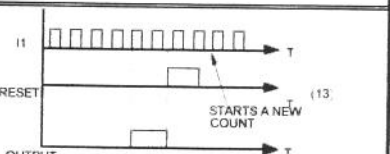
Normalmente Abertos	 (1)	Coneção em série de contatos normalmente fechados	 (7)
Normalmente Fechados	 (2)	Coneção em paralelos de contatos normalmente fechados	 (8)
Detecção de borda de subida	 SCAN (3)	Energiza e desenergiza contato com retenção	 (9) (10)
Detecção de borda de descida	 SCAN (4)	Temporizador na energização	 OUTPUT VALUE ACCUMULATED (11)
Coneção em série de contatos normalmente abertos	 (5)	Temporizador na desenergização	 OUTPUT ACCUMULATED VALUE (12)
Coneção paralela de contatos normalmente abertos	 (13)	Contador	 RESET STARTS A NEW COUNT (13)

Figura 5.6. Representação das operações básicas realizadas por um CLP

Instruções do tipo Aritméticas:

As instruções do tipo aritméticas são empregadas para realizar cálculos com dados armazenados na memória de dados. Através destas instruções matemáticas podemos somar, subtrair, multiplicar e /ou dividir dois dados localizados em dois diferentes localizações da memória de dados.

Instruções Complexas:

Destinadas principalmente ao controle de processos, elas são capazes de implementar funções de controle do tipo PID, e controle de movimentos, como também em gerações mais modernas de controladores para implantar controle baseado em lógica fuzzy. Estas instruções fogem do escopo do nosso trabalho e portanto são brevemente definidas somente para o leitor ter conhecimento da sua existência maiores detalhes destas instruções podem ser encontrados em (MURPHY (1992) e QUIROZ (1993).

5.4. Regras para a eleição do conjunto de instruções da Técnica de Especificação:

Um dos objetivos da criação de uma técnica padronizada de especificação de lógicas de controle para um CLP é tirar vantagem da redução do número de instruções necessárias. Este conjunto de instruções deverá fornecer confiabilidade e a “capabilidade” necessárias para cobrir as necessidades de programação de uma aplicação baseada em CLPs evitando duplicidades e exclusões. Segundo ZAPOLIN (1990), existem três critérios básicos a serem considerados na eleição do conjunto de instruções ótimo. Ao satisfazer simultaneamente estes três critérios estará garantida uma boa qualidade do conjunto. Estes três critérios são os seguintes:

- Cobrir as necessidades reais dos usuários das aplicações.
- Fornecer as instruções mais populares.
- Evitar restrições desnecessárias.

Estudos recentes (ZAPOLIN, 1990) sobre o uso das instruções para programação de CLPs e sua frequência, indicam que a maior parte dos programas estão formados por instruções que realizam a lógica combinatorial tradicional, examinando o estado aberto/fechado dos dispositivos de entrada e alterando o valor de estado dos dispositivos de

saída. Estas instruções não representam menos do que 75% e usualmente até 90% das instruções usadas em aplicações típicas baseadas em CLPs. Ao considerarmos contadores, temporizadores e as instruções de movimentação de bits, esta porcentagem atinge o 98% das instruções achadas nos programas de aplicação cobertos pelo estudo. A seguir, em importância estão as instruções do tipo aritmético, que fornecem as ferramentas necessárias para implantar o processamento matemático necessário na maioria dos programas. A figura 5.7. resume as frequências relativas do uso de instruções em programas para CLPs.

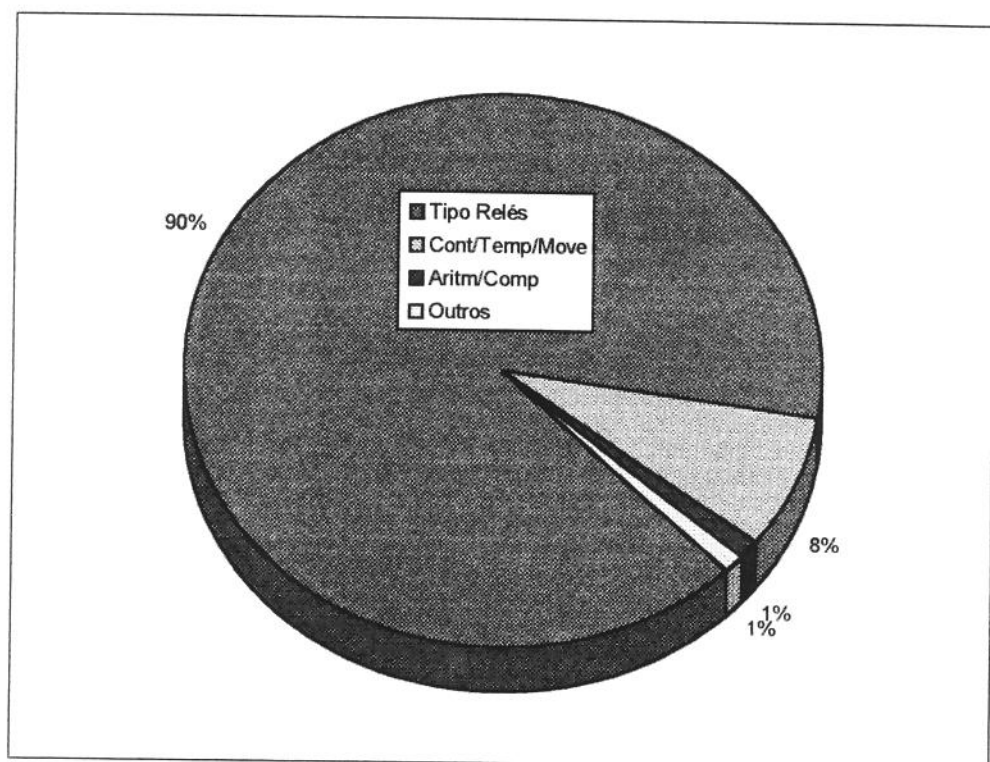


Figura 5.7. Uso relativo das instruções (por tipo) nos programas para CLPs

5.5. Linguagens de Programação para CLPs e a sua representação usando DCGs:

Os últimos avanços no hardware de computadores digitais e na tecnologia de software permitiram aos CLPs resolver uma maior quantidade de problemas de controle sequencial. O software para CLPs cada dia torna-se mais importante e é ele que direciona as decisões de escolha e seleção do tipo de controlador a serem adquiridos por parte de um grande setor da indústria.

Atualmente os diagramas de contatos (Relay Ladder Logic - RLL) constituem uma espécie de interface gráfica com o programador/usuário. Assim, a programação baseada na RLL permite aos engenheiros escrever programas de controle para tempo real, de maneira fácil e rápida.

A difusão deste método pode ser verificada através das estatísticas do uso dele nos Estados Unidos de América, onde no ano 1990 foram calculados em 250.000 os engenheiros de manutenção elétricos que estavam familiarizados com este tipo de aplicações (CHRISTENSEN, 1993; ASFAHL, 1990).

Virtualmente, todos os fabricantes importantes de CLPs incorporam esta técnica de programação aos seus produtos. Num estudo feito nos E.U.A. concluiu-se que 70% dos programadores preferiam os diagramas de contato a outros métodos e programação.

Nesta secção, são aplicados os conceitos de representação gramatical numa linguagem de programação para CLPS simplificada. Um programa escrito em diagramas de contatos é uma malha composta por uma série de laços, com cada um desses laços correspondendo a uma instrução. Considerando esta primeira definição podemos escrever imediatamente nossa primeira regra de produção de mais alto nível:

malha = laço | malha.

Com o objetivo de gerar as demais regras, define-se primeiro o set de instruções para um CLP imaginário. A seguinte lista ilustra um conjunto de instruções típicas de qualquer CLP comum:

BIT EXAMINING	--[]--	Examine ON (examinar energizado)
	--[]--	Examine OFF (examinar desenergizado)
	--()--	Energize Location (energizar saída)
BIT CONTROL	--(L)--	Latch Location (energizar com retenção)
	--(U)--	Unlatch Location (desenergizar saída com retenção)

Pode-se observar que existem duas categorias principais de instruções: Bit Examining e Bit Control (LU ET. AL., 1992). Ao analisar-se um laço simples, pode-se observar que eles estão compostos por um número arbitrário de condições de entrada, seguido por uma única condição de saída. Dado isto, é escrito o próximo conjunto de regras de produção:

laço = Condlist, Output

Condlist = Instruction | Condlist, Instruction.

Até agora, as regras de produção derivadas são genéricas. Virtualmente, qualquer controlador programado em RLL segue este formato básico. Já no nível seguinte, as regras começam a ser mais específicas e começam a se diferenciar segundo o CLP que está sendo programado:

Instruction = -[]-, Locação |
 -[/]-, Locação.

Output = -()-, Locação |
 -(L)-, Locação |
 -(U)-, Locação.

A seguir, é apresentada a representação em CFGs e DCGs deste conjunto de regras, usando a linguagem de programação Arity PROLOG :

```
lacos(S0,S):- laco(S0,S1)|';,lacos(S1,S).
laco(S0,S) :- condicao(S0,S1),saida(S1,S).
condicao(S0,S) :- instrucao(S0,S1)|condicao(S1,S).
instrucao([biton(I)],S) :- instrucao(('(-[',I,'(-)')'),S).
instrucao([bitoff(I)],S) :- instrucao(('(-[/',I,'(-)')'),S).
instrucao(('(-[',I,'(-)')'),S).
instrucao(('(-[/',I,'(-)')'),S).
saida([energ(O)|S],S).
saida([latch(O)|S],S).
saida([unlatch(O)|S],S).
```

```
%Este programa deve ser executado através da chamada%
% laco(X,[biton,bitoff,latch],[ ]).%
lacos(lacos(A,B))--> laco(A),lacos(B).
lacos(_)--> [].
laco(laco(T,S)) -->condicoes(T),saida(S).
condicoes(cond_entrada(I,C)) --> instrucao(I),condicoes(C).
condicoes(_)--> [].
instrucao(biton) --> [biton].
instrucao(bitoff) --> [bitoff].
saida(energ) --> [energ].
saida(latch) --> [latch].
saida(unlatch) --> [unlatch].
```

5.6. Estrutura do Sistema:

Este sistema é composto por três módulos principais. O primeiro módulo é encarregado da interação entre o usuário/programador e as demais partes do sistema. Através deste módulo pode-se realizar a especificação da lógica de controle, no formato neutro através da entrada de "frases" usando a gramática da TEOO. No segundo módulo estas frases são analisadas para conferir se todas as palavras usadas na descrição pertencem ao "vocabulário" permitido pela gramática da técnica de descrição. Este módulo é chamado de analisador léxico ou "scanner". Após a análise léxica de todo o arquivo de entrada, este arquivo é convertido numa sequência de símbolos (tokens). O último módulo tem uma função dupla: primeiro, verifica se as seqüências de tokens são sintaticamente corretas; isto é feito confrontando as "frases" da especificação com a Gramática da TEOO; se existirem erros do tipo sintático, estes serão reportados nesta fase do processo. A segunda função deste módulo é criar a saída final do sistema; esta saída corresponde ao programa num código fonte do CLP selecionado. Esta transformação é feita a partir de ações semânticas associadas a cada uma das regras de produção da gramática. A estrutura do sistema é apresentada na figura 5.8.

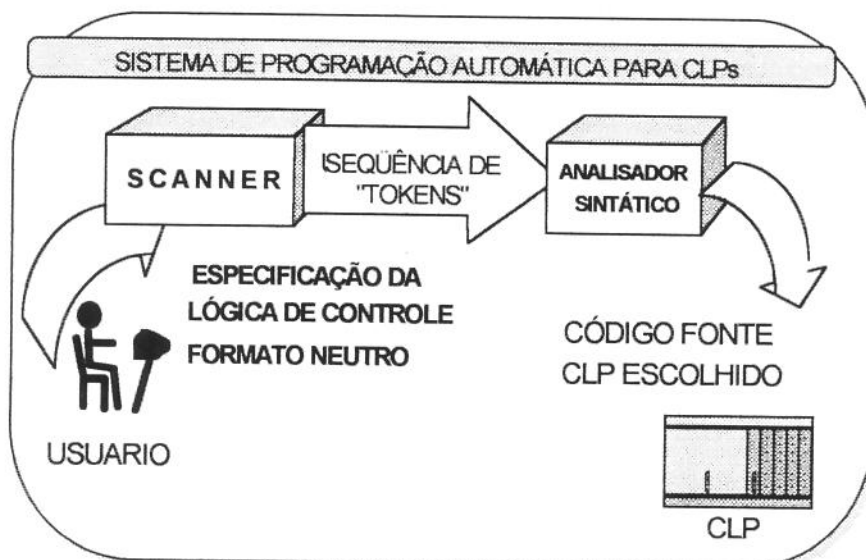


Figura 5.8. Estrutura do Sistema

5.6.1. O Gerador Automático de Software:

O módulo de geração automática de programas para CLPs construído é capaz de receber a saída do Analisador Sintático com base nos comandos orientados a objeto, gerando, como saída, programas escritos no formato do CLP escolhido na opção SET-UP PLC do menu principal do sistema gerador.

A principal característica do gerador reside no fato de incorporar a capacidade de gerenciar a herança de métodos com a finalidade de obter o número da porta de entrada ou saída de cada dispositivo referenciado na descrição. Isto foi feito através da definição de um método chamado *getport* na classe *Dispositivo*. Este método se encarrega de avaliar todos os predicados definidos para descrever cada um dos objetos e capturar o número da porta definida. Parte do conjunto de predicados encarregados da geração automática do programa é mostrada a seguir:

%%%%%%%%%% gerador de codigo%%%%%%%%%%

```
ger_code(laco( (ativ(A,O)) ,(eventos(evento3(O1,A1,O2,A2)))) -->
    ger_code(evento1(O1,A1)), ger_code(ativ(liga,O)),
    ger_code(evento1(O2,A2)), ger_code(ativ(desliga,O)),
    ger_code(ativ(desliga,O)).
ger_code(laco(A,E)) --> ger_code(E), ger_code(A).
ger_code(eventos(E,R,Es)) --> ger_code(E),ger_code(R),ger_code(Es).
ger_code(eventos(E)) --> ger_code(E).
ger_code(rel(e)) --> {write($ AND $)}.
ger_code(evento1(O,on)) --> {write($ BITON $)},
    {heranca(O,getport(O,Porta))}, {write(Porta)}.
ger_code(evento1(O,off)) --> {write($ BITOFF $)},
    {heranca(O,getport(O,Porta))}, {write(Porta)}.
ger_code(evento2(O,off,T)) --> {write($ BITOFF $)},
    {heranca(O,getport(O,Porta))},
    {write(Porta)}, {write($ TON $),write(T),nl},
    {write($ BITOFF $),write(Porta),nl}.
ger_code(evento2(O,on,T)) --> {write($ BITSET $)},
    {heranca(O,getport(O,Porta))},
    {write(Porta)}, {write($ TON $),write(T),nl},
    {write($ BITSET $),write(Porta),nl}.
ger_code(ativ_afirm(liga,O)) --> {write($ BITSET $)},
```

```

{heranca(O,getport(O,Porta))}, {write(Porta),nl}.
ger_code(ativ(liga,O)) --> {write( $ LATCH $)},
{heranca(O,getport(O,Porta))}, {write(Porta),nl}.
ger_code(ativ(desliga,O)) --> {write( $ BITUNLATCH $)},
{heranca(O,getport(O,Porta))}, {write(Porta),nl}.

```

5.6.2. Um exemplo simples do uso do Gerador:

Apresenta-se um exemplo simples da utilização do sistema gerador automático de código baseado na TEOO. Este foi adaptado de um exemplo existente em (RAM e LAI, 1992). A figura 5.9. mostra uma representação esquemática de um sistema manipulação e distribuição de materiais. Três manipuladores, chamados de MAN_01, MAN_02 e MAN_03 na figura 5.9, carregam três tipos de peças com diferentes tamanhos. Três pistões, chamados PIST_01, PIST_02 e PIST_03 são os encarregados de empurrar desde a estação de entrada os blocos até uma esteira. Um sensor ótico envia um sinal para o controlador que, por sua vez ativa o pistão correto através de uma eletro-válvula. O sistema total entra em operação quando o switch geral, chamado SWG_01 é apertado e desliga só depois que o “switch” de desligamento geral SWG_02 for acionado. A esteira, chamada de CONVEYER_01, está constantemente em movimento, até o SWG_01 ser desligado. As peças que se transladam através do CONVEYER_01 são detetadas através de um leitor de código de barras, chamado de SCANNER_01. Este detetor fornece três tipos de sinais, dependendo da peça que foi detetada, SINAL_01, SINAL_02 ou SINAL_03. Cada um destes sinais ativará um pistão de saída, identificados aqui como PIST_04, PIST_05 e PIST_06. Estes pistões direcionam as peças para esteiras distribuidoras identificadas como CONVEYER_02, CONVEYER_03 e CONVEYER_04. Estas esteiras são também ativadas pelos sinais SINAL_01, SINAL_02 e SINAL_03, respectivamente. As esteiras CONVEYER_02 e CONVEYER_03 ficam ativadas por 5 segundos. Pelo contrário, o CONVEYER_04 permanece em movimento até que a chave de fim de curso CHAVE_FIM_01, é ativada, isto acontece quando a peça faz contato com ela.

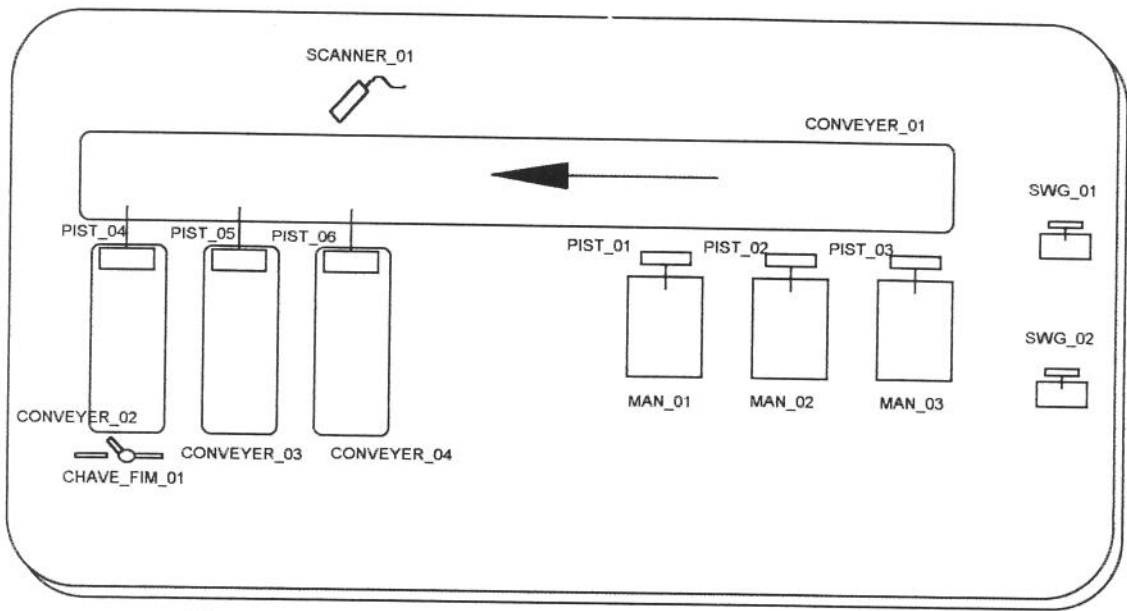


Figura 5.9. Exemplo de sistema de manuseio de materiais

Utiliza-se um conjunto de objetos definidos pelo nome e pelo número da porta na qual eles estarão ligados. Estes objetos são os seguintes:

OBJETO	MÉTODOS	PORTA
CONVEYER_01	LIGAR	000/00
CONVEYER_02	LIGAR	000/01
CONVEYER_03	LIGAR	000/02
CONVEYER_04	LIGAR	000/03
PIST_01	ATIVAR	000/04
PIST_02	ATIVAR	000/05
PIST_03	ATIVAR	000/06
PIST_04	ATIVAR	000/07
PIST_05	ATIVAR	000/10
PIST_06	ATIVAR	000/11
SINAL_01	SINALIZAR	000/13
SINAL_02	SINALIZAR	000/14
SINAL_03	SINALIZAR	000/15
FLAG_01	LIGAR	030
MAN_01	FIXAR	100/01
MAN_02	FIXAR	100/02
MAN_03	FIXAR	100/03
CHAVE_FIM_01	SINALIZAR	100/04
SWG_01	APERTAR	100/05
SWG_02	APERTAR	100/06

A terceira coluna da tabela indica os endereços das portas de entrada e saída para cada um dos dispositivos que estarão ligados com o CLP, para o qual está sendo gerado o código automaticamente. Note que foram tabelados apenas os métodos que cada um dos objetos usará nesta aplicação. Cada um destes objetos ou as classes às quais pertencem, poderão ter outros métodos associados, inclusive, palavras que sejam sinônimos das usadas neste exemplo. Este fato mostra ainda mais, a flexibilidade com que as descrições poderão ser feitas, pois cada classe ou instância poderá ter associada a si um número ilimitado de métodos (ou verbos na linguagem natural) para representar o seu acionar dentro de uma aplicação de manufatura. Assim por exemplo, os manipuladores pneumáticos declarados como MAN_01, etc, poderão ter associados a si outros métodos, tais como : ORIENTAR, RETIRAR, DESMONTAR, MONTAR, etc. cada um deles adaptando-se melhor a uma determinada situação.

A seguir mostra-se um conjunto de sentenças, usando a Técnica de Especificação Orientada a Objeto e, posteriormente, o programa gerado usando os comandos e instruções do PLC-5 (ALLEN BRADLEY):

Caso SWG_01 esteja apertado, ligar CONVEYER_01, a menos que SWG_02 seja apertado. Assim que SINAL_01 começar a sinalizar, ativar PIST_04. Quando SINAL_02 iniciar a sinalização, ativar PIST_05. Caso SINAL_03 dê início à sinalização, ativar PIST_06. Caso SINAL_03 esteja sinalizando, ligar CONVEYER_04 enquanto CHAVE_FIM_01 não terminar a sinalização. Enquanto SINAL_01 estiver sinalizando, ativar CONVEYER_02. Caso SINAL_01 esteja sinalizando por 5 segundos, desative CONVEYER_02. Uma vez que CONVEYER_02 terminar o movimento, desativar SINAL_01. Enquanto SINAL_02 estiver sinalizando, ativar CONVEYER_03. Caso SINAL_02 esteja sinalizando por 5 segundos, desative CONVEYER_03. Uma vez que CONVEYER_03 terminar o movimento, desativar SINAL_02. Se MAN_01 não acabar a fixação, não ativar PIST_01. Uma vez que MAN_02 terminou a fixação, ative PIST_02. Depois que MAN_03 puser fim à fixação, ative PIST_03.

ABR SINAL_01	ABR SINAL_02	ABR SINAL_03	ABR SWG_01
PLS PIST_04	OU CONVEYER_03	OU CONVEYER_04	OU PIST_04
ABF CHAVE_FIM_01	EF T05	EF CHAVE_FIM_01	EF SWG_02
BOB CONVEYER_04	BOB CONVEYER_03	BOB CONVEYER_04	BOB PIST_04
ABF MAN_03	K5.0	ABR SINAL_02	
BOB PIST_03	ABR SINAL_01	PLS PIST_05	
ABF MAN_02	OU CONVEYER_02	ABR SINAL_01	
BOB PIST_02	EF T06	BOB CONVEYER_02	
ABF MAN_01	BOB CONVEYER_02	ABR SINAL_03	
BOB PIST_01	K5.0	PLS PIST_06	

No programa gerado foram modificados códigos que representam as entradas e saídas para facilitar assim o entendimento. A figura 5.10. mostra os diagramas ou histogramas de tempo equivalentes a algumas instruções geradas pelo sistema.

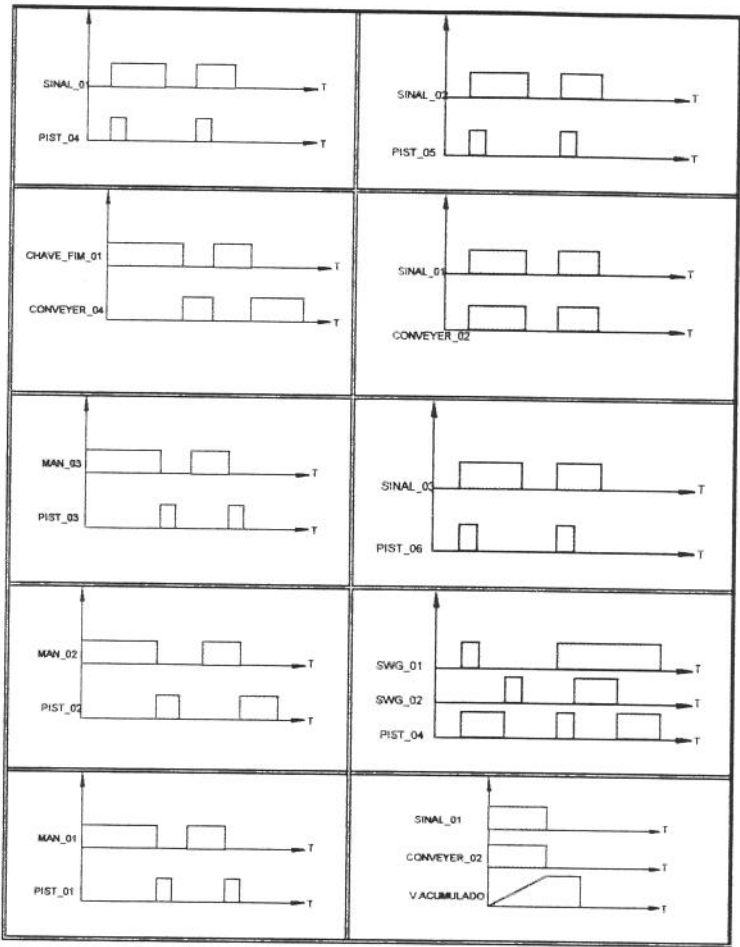


Figura 5.10. Histogramas de tempos para as instruções geradas.

5.7. Testes para Interfaceamento com uma interface em Windows:

Foram realizados paralelamente os testes para implementar uma união entre um módulo encarregado da interface gráfica com o usuário, escrita em Visual BASIC e diversos módulos executáveis escritos em Arity Prolog para DOS. A estratégia fica mais clara ao analisar-se a figura 5.11.

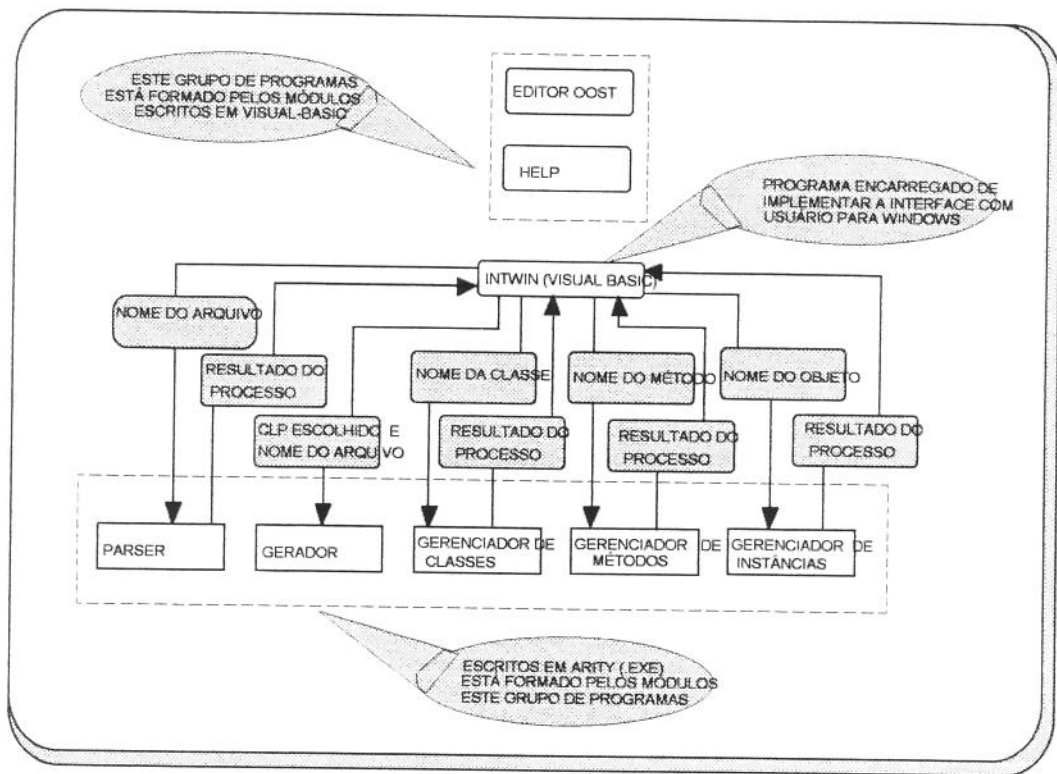


Figura 5.11. Estrutura e relacionamento do sistema que interfaceia Visual Basic e Arity Prolog

O sistema todo está coordenado pelo módulo INTWIN executado no ambiente Windows. Neste módulo o usuário dispõe das opções para edição de descrições usando a TEOO, gerenciar a hierarquia de classes, especificar e escolher um dado CLP e gerar código automaticamente para tal CLP. Cada uma destas opções estará ligada a um módulo independente escrito em Prolog e já compilado. Escolhida alguma das opções do Menu principal, o programa em Visual Basic, possui uma função especial para executar programas executáveis. Esta função, chamada de SHELL, tem a seguinte sintaxe:

`shell("nomeprog.exe", windowstyle)`

onde nomeprog.exe é o nome de programa que será executado desde a aplicação Visual Basic, e *windowstyle* indica o estilo de execução dentro do windows no qual o programa será executado. De maneira geral, este estilo permite executar o programa no modo minimizado ou no estilo normal.

Antes de executar qualquer programa, devem ser inicializadas as variáveis de cada módulo executável através da opção File Properties do Program Manager do Windows. Nesta opção são definidos os caminhos de busca para o programa, o arquivo de trabalho, um ícone associado e o modo de execução. No caso deste trabalho, decidiu-se rodar todos os módulos em modo minimizado. Isto permite que o usuário não perceba que o programa, ou melhor dito, o módulo não foi escrito para Windows, e ele está operando na forma de um “Shell” (similar com o que é feito quando se executa programas para DOS desde dentro do Windows usando a opção MSDOS Prompt). Isto só é possível devido a que são extraídos todos os comandos de tela dos programas escritos em Prolog. Lamentavelmente existe uma limitação, a qual até hoje não foi superada, a impossibilidade de se passar parâmetros desde o programa principal (em V.Basic) para algum módulo executável escrito em Prolog, ou vice-versa. Isto está sendo realizado através de arquivos em formato ASCII que fazem as vezes de ponte entre estes módulos (estes arquivos aparecem em cinza na figura 5.11.). Isto diminui a eficiência de programa, mas, é uma solução que viabiliza o interfaceamento entre programas escritos para dois padrões diferentes.

A seguir são comentadas algumas telas da interface com usuário escrita para o padrão Windows em Visual Basic 3.0.

5.8. O sistema em operação:

Foi implementado um gerador automático de software para Controladores Lógicos Programáveis baseado na Técnica de Especificação Orientada a Objeto (Object-Oriented Specification Technique - OOST). Esta versão incorpora a gramática definida e descrita durante o capítulo anterior. A estrutura do sistema está detalhada na figura 5.12.

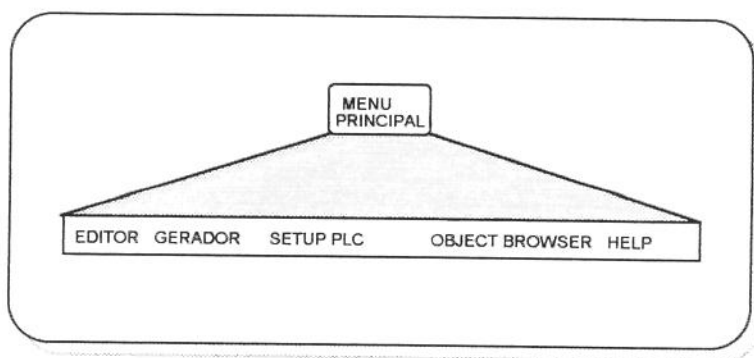


Figura 5.12. Estrutura do Sistema Gerador Automático de Software para CLPs.

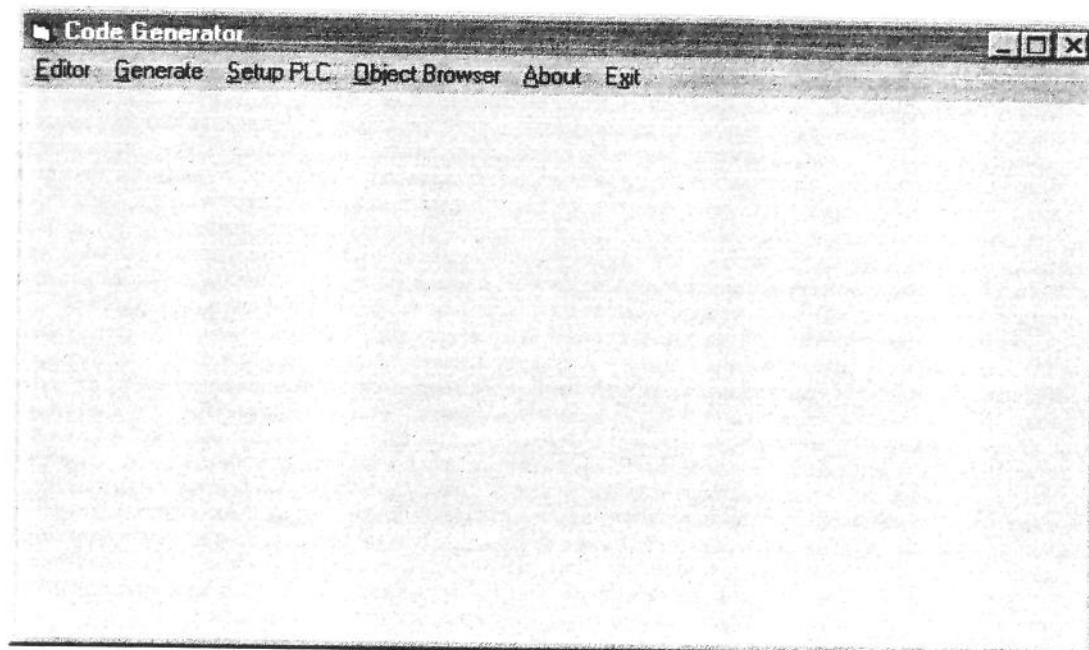


Figura 5.13. Tela inicial do sistema mostrando todas as opções do menu principal

A seguir, serão detalhados cada um dos módulos existentes neste sistema:

5.8.1. Editor de Especificações:

Através deste módulo o usuário ou programador pode realizar a escrita do *script* que descreve o comportamento e estrutura do sistema que ele deseja programar. Este editor possui todas as características de um editor de textos comum, podendo o usuário inserir, apagar e modificar blocos completos de textos. A figura 5.14. mostra o aspecto desta opção.

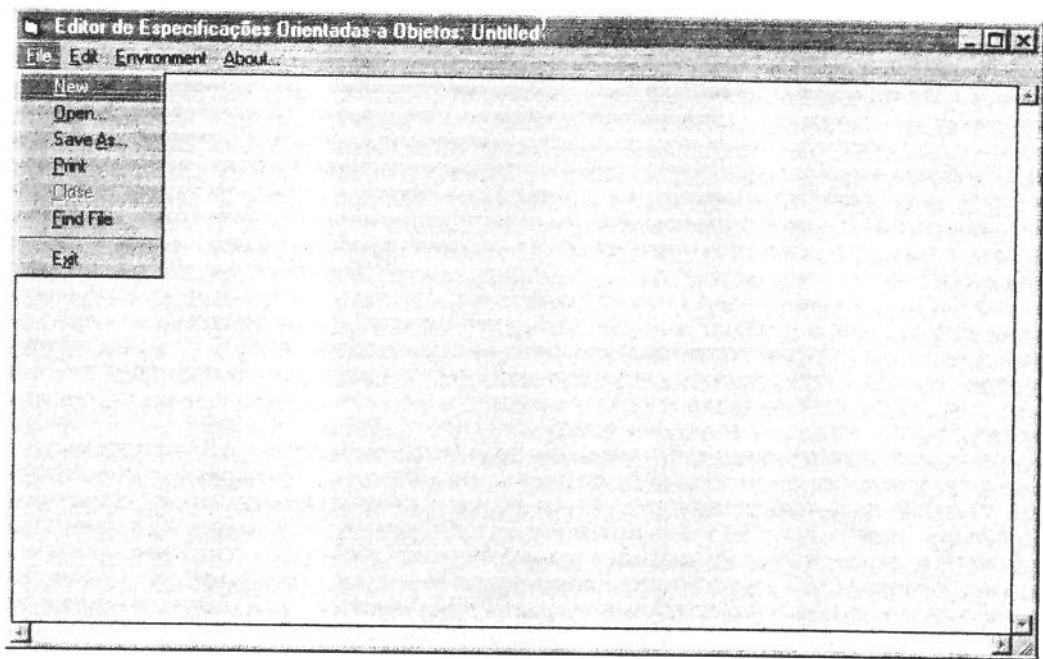


Figura 5.14. Tela do Editor de Especificações usando TEOO.

Pode-se apreciar a série de recursos gráficos com que o usuário conta. Estas telas usam de diversos recursos ou ferramentas disponíveis no Visual Basic para desenvolver Interfaces Gráficas com o usuário. Na figura 5.15, é mostrada a caixa de diálogo que o sistema ativa quando o usuário desejar abrir um novo arquivo, na extremidade superior esquerda existe uma *list box*, através da qual o usuário pode carregar uma especificação já existente (com extensão .oop) e produzir as alterações. Uma *list box* apresenta uma fita vertical de textos ou uma série de itens, um dos quais pode ser escolhido pelo usuário para algum efeito, neste caso, o nome de arquivos com extensão *oop* que armazenam especificações feitas com o uso do sistema.

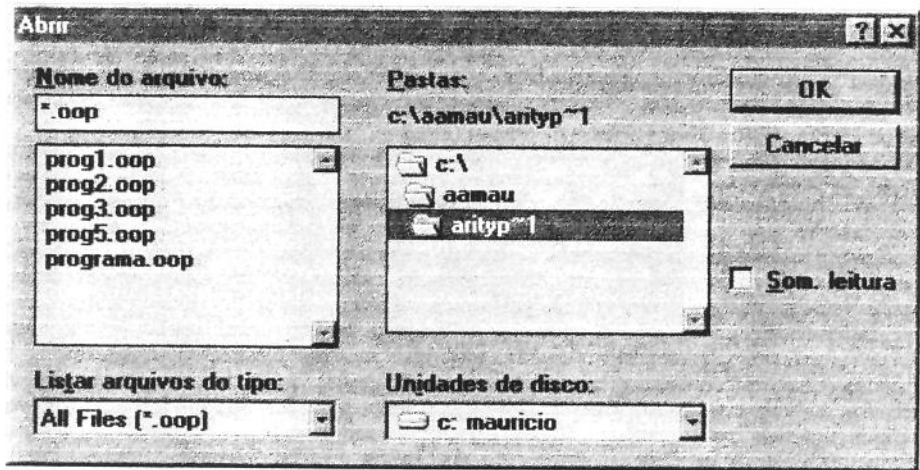


Figura 5.15. Caixa de diálogo da opção Abrir Arquivo do editor

O editor consiste numa região de tela que apresenta toda a funcionalidade de um editor de textos comum. Uma vez que o usuário escolhe um arquivo na *list box*, o conteúdo deste arquivo aparecerá na *edit box*. As capacidades desta *edit box* são as mesmas de qualquer editor de textos, tendo-se teclas de navegação rápida, deslocamento de cursor, avanço e retrocesso de páginas, possibilidade de levar o cursor para um número de linha específico, inserção e deleção de um bloco e opções de busca e substituição de textos.

O usuário pode preferir usar a função New para começar a edição de um arquivo a partir de uma “folha em branco”. Quando o usuário ativar a opção New, a *edit box* ficará em branco à espera do usuário/programador começar uma nova especificação. Com o botão Cancel, o sistema volta o controle ao menu principal. Finalmente, com o botão Save, o usuário poderá realizar a gravação do texto editado por ele. Se o texto foi editado a partir da opção New, o usuário deverá fornecer o nome do novo arquivo criado, para este efeito será mostrada uma caixa de diálogo associada à opção Save As. Este recurso permite ao usuário entrar com o nome do arquivo recém-criado. O sistema também fornece a possibilidade ao usuário de procurar por um arquivo num determinado subdiretório ou disco. Para isto dispõe da opção “Seek”, cuja de diálogo é mostrada na figura 5.16.

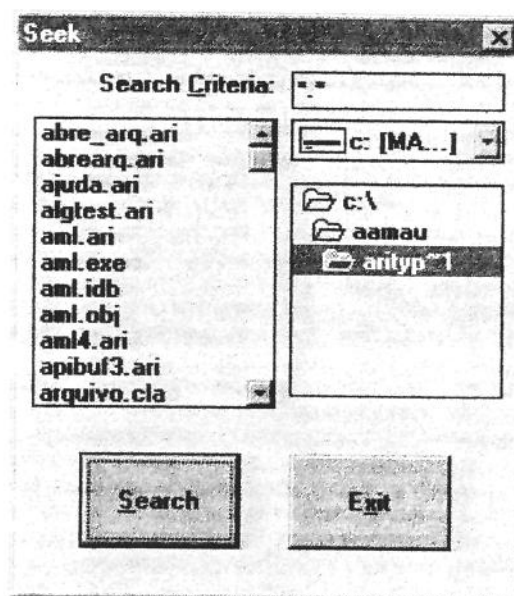


Figura 5.16. Caixa de Diálogo da opção Seek, para procurar arquivos.

5.8.2. O Object Browser:

O Object Browser permite ao usuário fazer uso de todas as capacidades do paradigma de objeto suportadas pela técnica de especificação. Ou seja através desta opção, o usuário pode incorporar ou extrair objetos da hierarquias de classes suportadas pela gramática da técnica de especificação. A figura 5.17 mostra as opções implementadas.

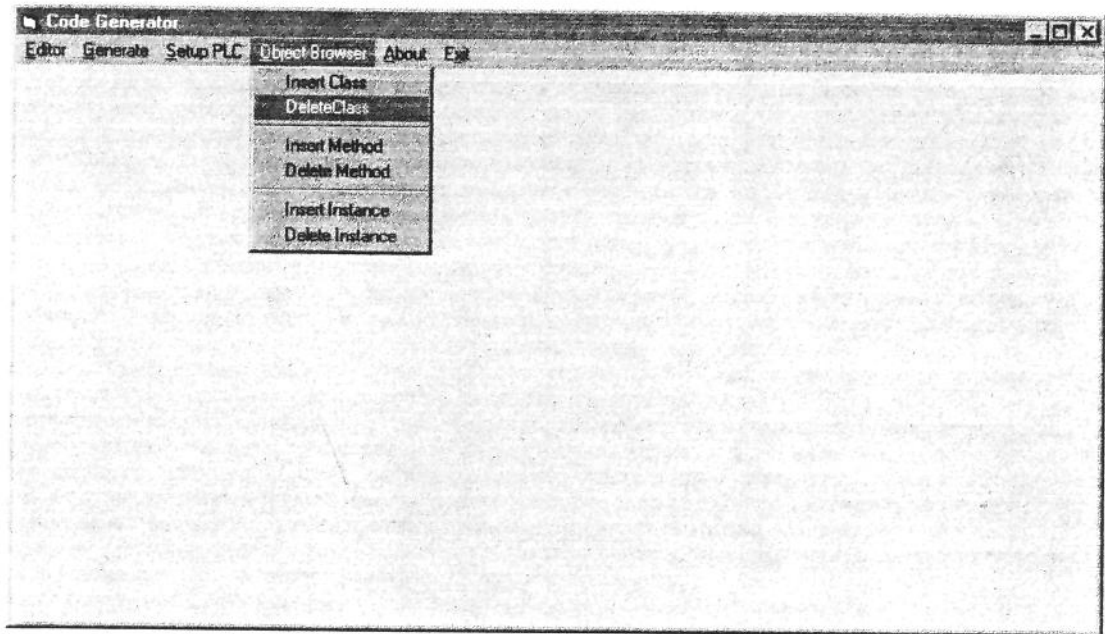


Figura 5.17. Opções implementadas no Object Browser.

As opções que o usuário tem são: Inserir um novo objeto (entenda-se instância), apagar um objeto, junto com a capacidade de alterar os métodos associados a alguma classe existente na hierarquia definida. Também é possível alterar a hierarquia de classes incorporando ou excluindo uma classe de objetos. Analisar-se-á mais detalhadamente a opção de inserir um novo objeto. A figura 5.18 mostra a caixa de diálogo que o sistema abre uma vez que o usuário escolhe esta opção.

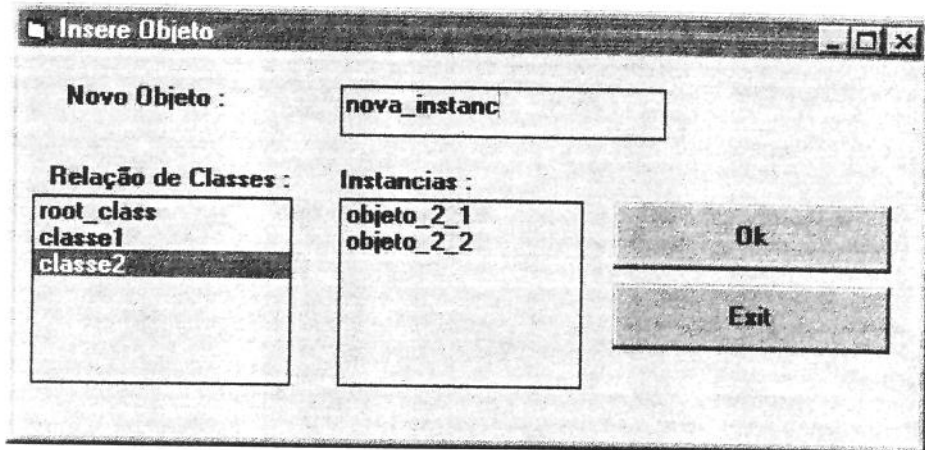


Figura 5.18. Caixa de diálogo para definir um novo objeto.

Nesta caixa de diálogo existem os seguintes recursos de interface: uma *list box*, onde são fornecidas as classes existentes na hierarquia. Neste ponto é interessante se deter num aspecto importante deste sistema: quando o usuário for usar o sistema pela primeira vez, a única classe existente a hierarquia será a classe raiz. Só através do uso contínuo deste sistema e da incorporação de novas classes e instâncias de objetos definidas pelo usuário é que o sistema, e mais especificamente a gramática da técnica irá aumentar.

Voltando atrás, na *list box* aparecerão as classes definidas anteriormente. Através desta *list box*, o usuário poderá especificar a qual das classes o objeto (ou melhor dito, a instância) será atribuído. Junto com esta *list box*, existe uma segunda que mostra todas as instâncias associadas a classe que foi escolhida, e que aparece mais escura na figura. De acordo com o paradigma de orientação a objeto, esta atribuição resulta na herança de todos os métodos definidos para a classe superior. Junto com esta *list box* aparece um *edit field* no qual o usuário informa o nome da nova instância que está sendo criada e adicionada à hierarquia, uma vez feito isto, o usuário deve ativar o controle do botão OK e confirmar a inserção. Se o usuário desejar anular a ação o botão EXIT deve ser ativado, voltando o controle para o Menu Principal.

Se o usuário desejar eliminar uma instância da hierarquia, ele deve escolher a segunda opção do Menu do tipo Pull Down que aparece na figura 5.17. Após isto, aparecerá a seguinte caixa de diálogo na tela do computador (figura 5.19).

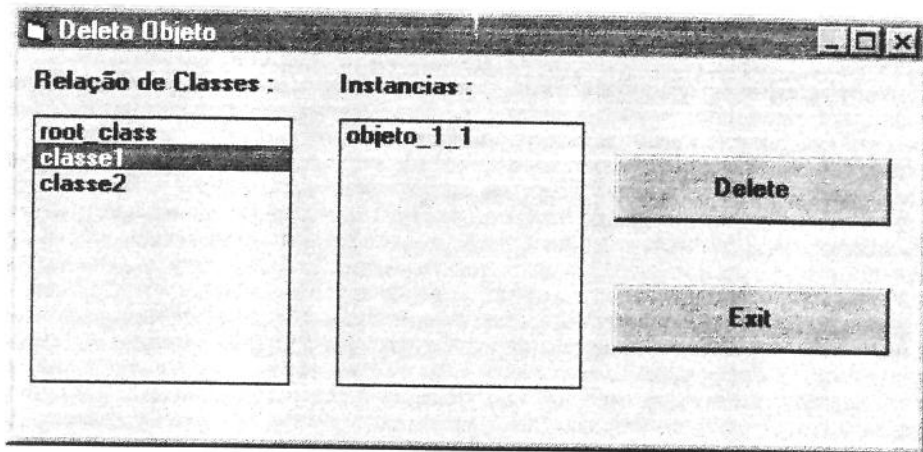


Figura 5.19. Caixa de diálogo que permite apagar uma instância de objetos.

Nesta caixa de diálogo podem ser observadas duas *list boxes* numa delas aparecerão todas as classes já definidas e que fazem parte da gramática da técnica. Uma vez que o usuário escolheu um das classes, na segunda *list box* aparecerão todas as instâncias que estão associadas à classe escolhida. Assim que o usuário identificar a instância que ele deseja apagar (com o mouse), ele deverá levar o controle para o botão OK para efetivar a operação. Para cancelar esta opção o usuário dispõe do botão EXIT.

No segundo grupo de instruções do menu pull down da figura 5.17, o usuário tem a possibilidade de inserir ou apagar um método associado a uma das classes existentes na gramática. Faz-se uma análise, em separado, destas duas opções. Se o usuário escolher a opção *Add Method*, o sistema apresentará a seguinte caixa de diálogo (figura 5.18).

Insert a New Method

classe3
classe3

Ok Exit

Name os the new Method
(infinitive form)

Name os the new Method
(participle form)

Name os the new Method
(gerund form)

Name os the new Method
(substantivized form)

Figura 5.18. Caixa de diálogo para adicionar um novo método a uma classe existente.

Esta caixa de diálogo possui uma *list box* para identificar qual é a classe de objetos na qual será adicionado um novo método. Feito isto pelo usuário, ele deverá fornecer o nome do novo método que esta sendo inserido. Este método deve ter a forma de um verbo em infinitivo, junto com isto o sistema apresentará ao usuário perguntas sobre outras formas verbais para este verbo (verbo em gerúndio, verbo em particípio e verbo substantivado), de maneira a satisfazer as necessidades do analisador sintático da gramática da TEOO.

Se o usuário escolher a opção *Remove Method*, o sistema fornecerá a caixa de dialogo mostrada na figura 5.19. Nela aparece uma *list box* através da qual o usuário poderá identificar qual é a classe da qual será removido um método. Uma vez identificada a classe, serão mostrados, na segunda *list box* todos os métodos associados à classe escolhida. Nesta lista o usuário escolherá qual o método a ser removido da gramática. Esta escolha é confirmada pelo botão OK. Se o usuário quiser cancelar esta opção, a caixa de diálogo dispõe de um botão EXIT, o qual devolve o controle para o Menu Principal.

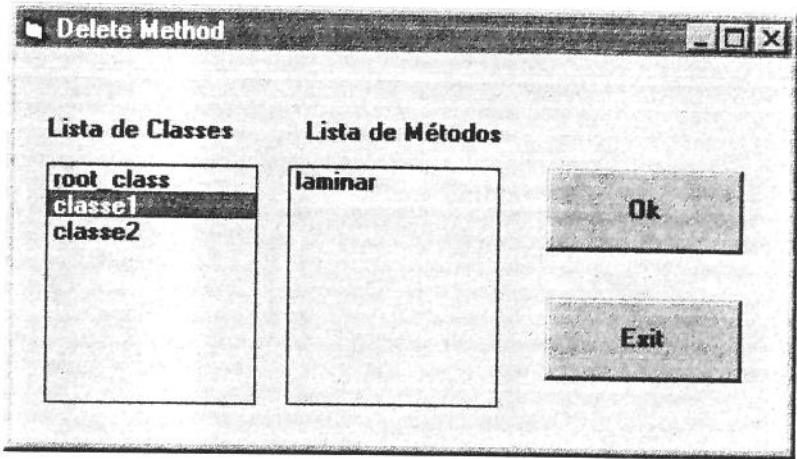


Figura 5.19. Caixa de diálogo para remover um método existente pertencente a uma classe.

Para as opções *Insert Classe* o sistema fornece uma caixa de diálogo (figura 5.20). Nela o usuário dispõe de uma *list box* através da qual ele poderá identificar qual a classe que originará esta nova classe e que, por herança, passará os métodos que ela possui, para esta nova classe. Se nenhuma das classes origina a nova classe, o usuário deve identificar a classe RAIZ como a originária.

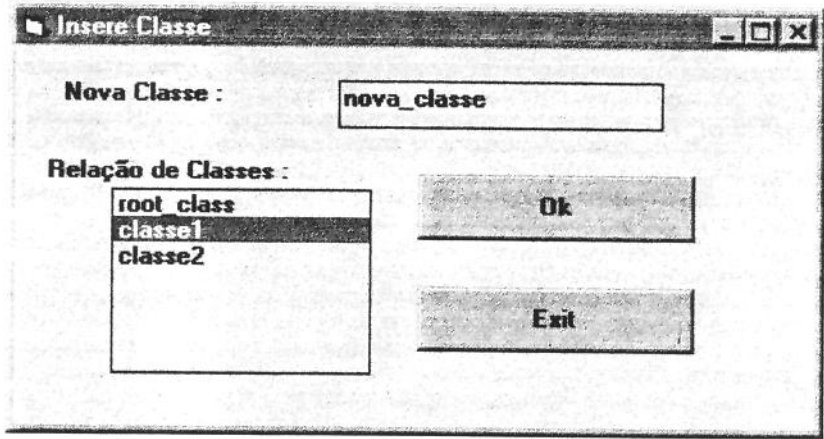


Figura 5.20. Caixa de diálogo para inserção de uma novo classe.

No caso de o usuário escolher a opção *Delete Class* ele disporá de uma caixa de diálogo na qual aparece uma *list box* onde o sistema apresenta o nome de todas as classes que foram definidas anteriormente, e que fazem parte da gramática (figura 5.21). Uma vez que o usuário escolher o nome da classe a ser removida ele deve confirmar através do botão OK. Por outro lado, se ele desejar sair desta caixa de diálogo sem realizar a operação, dispõe-se do botão EXIT que devolve o controle do programa ao Menu Principal. É

necessário salientar que ao apagar uma classe existente serão removidos todos os métodos associados a ela junto com todas as instâncias e classes que são geradas a partir da classe que está sendo apagada.

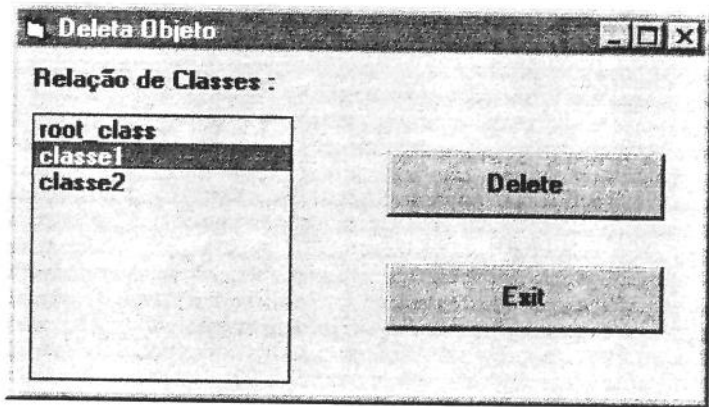


Figura 5.21. caixa de diálogo para remover uma classe existente.

5.8.3. Gerador Automático de Software:

Ao escolher esta opção, o sistema apresentará o seguinte (figura 5.22) Menu de Opções (tipo pull down), mostrando todos os tipos de CLPs para os quais o sistema é capaz de gerar código.

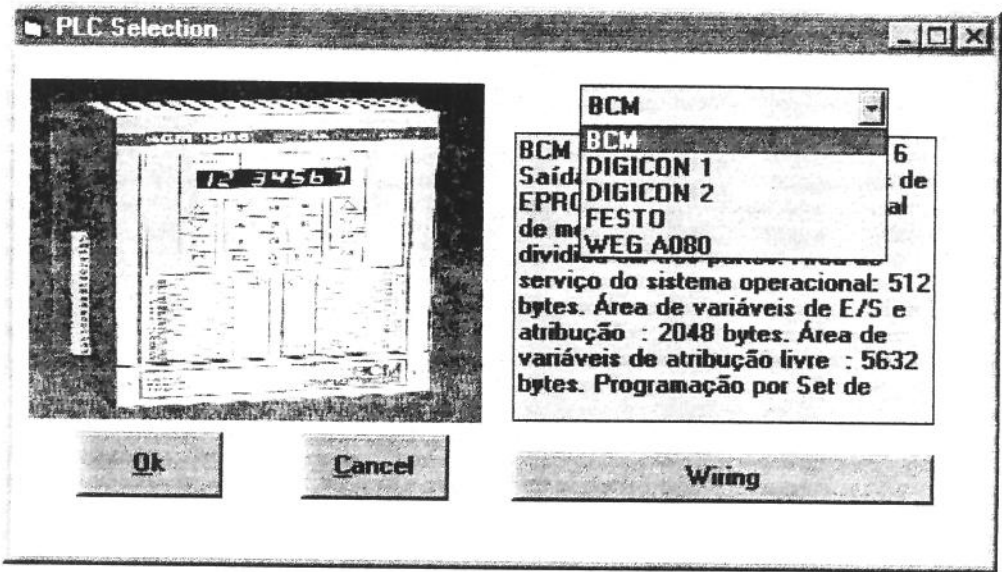


Figura 5.22. Menu de opções para o gerador de código de CLPs.

Neste menu aparecem todas as opções de formatos de programação para controladores lógicos programáveis, para os quais o sistema gerará código fonte em forma automática e a partir das especificações escritas usando a técnica de especificação orientada a objeto. Uma vez que o usuário escolher o CLP para o qual vai ser gerado o programa, o sistema apresenta uma fotografia digitalizada e uma série de informações sobre o equipamento em questão. Na atualidade está operando apenas um módulo protótipo que gera programas num sub-conjunto da linguagem de programação do Allen Bradley (PLC-5) e outro para CLPs da Shrader Bellows. Uma vez escolhido o CLP destino, o usuário deve especificar os detalhes de como o CLP está ligado aos dispositivos do sistema de produção que ele vai controlar. Aqui devem ser identificados os objetos (já definidos na hierarquia através do Object Browser) e os identificadores nos módulos de entrada ou saída segundo onde eles estejam ligados. Ao apertar o botão WIRING na caixa de diálogo da figura 5.22, aparecerá na tela do vídeo a seguinte caixa de diálogo (Figura 5.23).

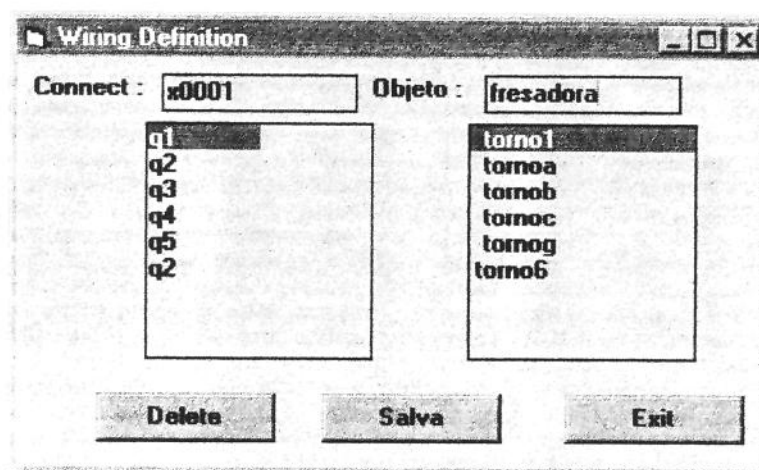


Figura 5.23. Caixa de diálogo para definir o esquema de fiação do CLP escolhido

Assim existem duas list boxes para a especificação do identificador de entrada ou saída e o objeto que estará ligado a ele. Aqui o usuário poderá apagar alguma especificação já feita e que não faz mais parte da configuração física do sistema. Só basta identificar o número ou objeto e, na sequência clicar sobre o botão DELETE. Da mesma forma o usuário poderá definir uma nova ligação usando os edit fields da parte superior da caixa de diálogo e confirmar a sua incorporação através do botão SALVA.

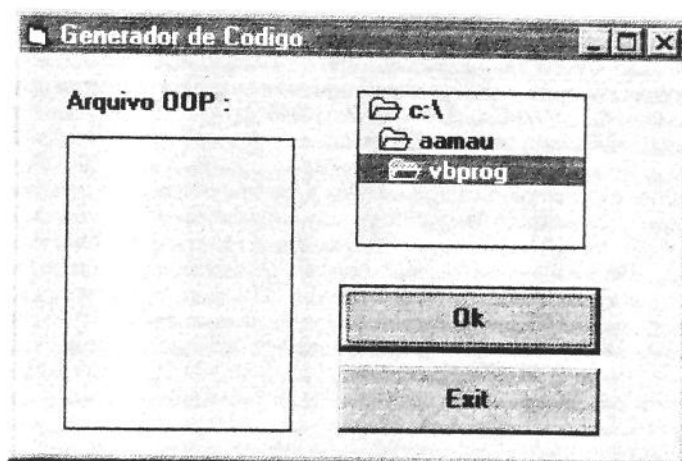


Figura 5.24. Caixa de diálogo para selecionar o programa a ser gerado.

Para gerar o programa, o usuário deve escolher o nome da especificação escrita usando a TEOO, na caixa de diálogo mostrada na figura 5.24. Nesta caixa de diálogo existe uma *list box* onde aparecem os nomes de todos os arquivos já criados usando a técnica (estes arquivos têm a extensão *oot*). Usando esta *list box* o usuário poderá especificar o nome do arquivo que será utilizado como entrada para o processo de geração automática do programa para o CLP escolhido. Uma vez selecionado o arquivo o usuário deve indicar a sua escolha através do botão OK. Este botão iniciará o processo de geração. Finalizado este processo, o usuário pode escolher um novo arquivo, ou pode abandonar esta opção através do botão EXIT.

A seguir, no último capítulo deste trabalho, são apresentadas as conclusões, e sugestões para trabalhos futuros.

Capítulo 6

Conclusões

O principal objetivo deste trabalho foi mostrar a criação de uma técnica textual de especificação baseada num conjunto de objetos concorrentes e cooperativos que representam objetos reais do domínio da manufatura discreta. Esta técnica é aplicável à definição de controladores para sistemas de manufatura e pode ser usada em tarefas de modelagem propriamente dita, tarefas de geração automática de "software", ou, simplesmente como um meio de armazenar conhecimento sobre projetos de sistemas de manufatura e do comportamento deles.

Conjuntamente, através deste tipo de modelo podem ser estruturadas formalmente as especificações de sistemas, integrando diversos tipos de especialistas e conhecimentos numa linguagem comum. Com isto pretende-se eliminar a ambigüidade nas definições e facilitar o intercâmbio de informações.

A Técnica de Especificação Orientada a Objeto (TEOO) pode ser considerada como um eficiente meio para realizar a descrição informal de requerimentos de um novo sistema, e, a partir desta, facilitar todo o ciclo de vida de um sistema, até inclusive a geração automática de software de controle para tais sistemas.

Quando a técnica é aplicada nas fases iniciais do ciclo de vida de um sistema de manufatura, tais como a especificação informal e a análise de requerimentos, o contato entre usuário e projetista é melhorado devido a que esta comunicação pode ser feita de uma forma natural, minimizando assim os erros de interpretação e superando as barreiras que possam existir entre profissionais, que pelo fato de atuarem em áreas diferentes da engenharia, não possuam uma linguagem comum para a comunicação destes requerimentos.

Esta metodologia permite a criação de bibliotecas de objetos que acelerarão o desenvolvimentno de novos modelos e fomentarão a reutilização de modelos já existentes. Este fato traz importantes avanços na produtividade e confiabilidade das tarefas de desenvolvimento e implantação de sistemas de manufatura.

Como aplicação da TEOO, foi desenvolvido um sistema de programação automática para controladores lógicos programáveis. Este sistema trabalha com entradas escritas usando a TEOO, e gera automaticamente programas para um CLP específico, respeitando o seu formato de programação.

O sistema faz uso das gramáticas de cláusulas definidas, cuja manipulação a nível de software é facilitada com a utilização do Arity Prolog. Esta gramática permite a análise sintática das especificações feitas em TEOO, e posteriormente a construção de geradores de código de uma maneira similar à utilizada para a escrita do analisador sintático.

Para validar o conjunto de construções gramaticais que representam as possibilidades de mudanças de estado num sistema controlado por CLPs, foi feito um levantamento em seis linguagens de programação de CLPs existentes no mercado brasileiro, e feita uma seleção das instruções mais utilizadas. Este levantamento resultou num conjunto de instruções “padrão” que de acordo com a literatura incorpora mais do 90% das instruções utilizadas nas atividades de programação de CLP para aplicações industriais.

Para finalizar é importante destacar um aspecto da metodologia usada. Se desejarmos incorporar a geração automática de uma outra linguagem de programação, só será necessário trocar o módulo gerador de código. Assim, o “parser” permanecerá inalterado. Isto garante uma alta modularidade e uma rápida adaptação a outras linguagens de programação para novos CLPs.

Para o futuro pretende-se completar este sistema com a incorporação de outras linguagens de programação para CLPs reais, bem como a adição de novos comandos à técnica de especificação.

Através deste tipo de sistema pretende-se aumentar a produtividade das atividades de programação e incrementar a portabilidade dos programas desenvolvidos. O sistema foi inteiramente descrito em Arity Prolog Visual BASIC e implantado num microcomputador da linha PC (compatível IBM).

Capítulo 7

Referências Bibliográficas

ABBOTT, R. Program Design by informal english descriptions. *Comm. ACM*, Vol.26, No.11, p. 884, Nov. 1983.

AGUIAR, M.W.C., MURGATROYD, I.S., EDWARDS, J.M. Object-Oriented resource models: their role in specifying components of integrated manufacturing systems. *Computers Integrated Manufacturing Systems*, Vol.9, No.1, p. 33-48, 1996.

ALLEN BRADLEY, Controlador Programável CLP-5 *Manual de Programação e Operação*, Allen Bradley Controles Eletronicos Ltda. São Paulo, Brasil. 1993.

SILVA, J.C.ANACLETO, P.R.POLITANO, R.P. DE M. FORTES, A.J.V. PORTO. Uma especificação do controle de um FMS através de StateCharts. In: *Proceedings XII ENEGEP*, Florianópolis, SC, Brasil, 1993.

ANG, C.L., LUO, M., R.K.L.GAY. Knowledge-based approach to the generation of IDEF0 models. *Computer Integrated Manufacturing Systems*, Vol.8, No.4, p.279-290, 1995.

ASFAHL, C.R., A. BALAGAMWALA. SIMLOG: A prolog-based simulator for industrial logic control systems. *Computers and Industrial Engineering*, Vol.19, nos.1-4, p. 195-199, 1990

BITTAR, R. DE C.S.M., *A utilização do GRAFCET como Ferramenta na Automação Industrial*. Campinas : FEM, Universidade Estadual de Campinas, 1993. Tese (mestrado).

BARATA, J., CAMARINHA-MATOS, L.M., COLOMBO, W., CARELLI, R. Integration of Object-Oriented Programming and Petri Nets for Modelling and Supervision of FMS/FAS.

Proceedings of the IEEE/ECLA/IFIP International conference on architectures and design methods for balanced automation systems, Vitoria (ES), Brasil, p.107-120, 1995.

BAKOUCH, M. A program Transformation System for Software reuse. In: *Proceedings of the FIFTH INTERNATIONAL CONFERENCE ON INFORMATION PROCESSING AND MANAGMENT OF UNCERTAINTY IN KNOWLEDGE-BASED SYSTEMS*, Paris, July, 1994.

BHATNAGAR, S., R.J. LINN. Automatic Programmable Logic Controller Program Generator with Standard Interface. *Manufacturing Review*, Vol.3, no.2, p. 98-105, 1990

BONANOMI, D. PLC soultions range from simple on/off control to sophisticated automation. *I & CS*, p. 65-93, October, 1994

BOOCH, G. Object Oriented Design. *IEEE Transactions on Software Engineering*, v. SE-12, n.2, p. 211-221, 1986.

BOUCHER, T.O., M.A. JAFARI. Design of a Factory Floor Sequence Controller from a High Level System Speification. *Journal of Manufacturing Systems*, v.11, n.6, p. 401-417, 1992.

BRAGA, M.C.L.F., C.A. MARTIN. Um modelo de desenvolvimento de Software baseado em redes de Petri utilizado no desenvolvimento do gerente de uma célula flexível de manufatura". In: *Proceedings I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE*, Rio Claro, SP, Brasil. p. 437-444, 1993.

BRAGA, M.C.L.F., C.A. MARTIN, J. CARDOSO. Uma experiência na utilização de Redes de Petri no desenvolvimento de Software para gerenciar uma célula flexível de manufatura. In: *Anais do XII CONGRESSO BRASILEIRO DE ENGENHARIA MECÂNICA*, Brasília, DF, Brasil. p. 425-428, 1993.

BRUCE, B. Case systems for Natural Language. *Artificial Intelligence* 6 (1975), p. 327-360.

BRUNO, G. E MARCHETTO. Process Translatable Petri Nets for the Rapid Prototyping of Control Systems. *IEEE Transactions on Software Engineering*, Vol. SE -12, no.2, p. 346-357. 1987.

CAPKOVIC, F. Modelling and Justifying discrete production processes by Petri Nets. *Computer-Integrated Manufacturing Systems*, v.6, n.1, p. 27-35, 1993.

CASTRO, P.T.S. Automação Industrial Orientada a Objetos. *Automação e Indústria*, v. VI, n. 49, p. 11-12, 1993.

CEGALLA, D.P. *Novíssima Gramática da Língua Portuguesa*, 21ª Edição, Companhia Editora Nacional, 1980.

CHOMSKY, N. Three Models for the Descriptions of Languages. *IEEE Transactions on Informations Theory*. Vol. IT2, p. 113- 124. 1956.

CHRISTENSEN, J. Programmable controller users and makers go global with IEC 1131-3. *I&CS*, October, PP:73-76, 1993.

COLMERAUER, A. *An interesting natural language subset*. Groupe d'Intelligence Artificielle, Université de Marseille- Luminy , 1977.

COX, B.J., A.J. NOVOBILSKI, *Object Oriented Programming*, 2nd. ed. Addison Wesley Pub. Co., USA, 1991.

COX, R.A., *Technician's Guide to Programmable Controllers*, Delmar Publishers Inc., Albaanny, New York, USA, 1989.

CUNHA, C., *Gramática do Português Contemporâneo*, 7ª Edição revista, Editora Bernardo Álvares S.A., Belo Horizonte (MG), 1978

DIMITROV, D., N.TEODOROV, Software Development Approach in FMS, *Computers in Industry*, v.10, p. 171- 175, 1988.

- DOLET, J., R., M. MOREAU. Development of an Object-Oriented Simulator for Material Handling System Design. *Computers and Industrial Engineering*, v.23, n 1-4, p. 249-252, 1992.
- DURÁN, O., A. BATOCCHIO, Relatório de Atividades Projeto FAPESP 92/4983-0.1993.
- DURÁN, O., A. BATOCCHIO, A high-level object-oriented programmable controller program interface, In: ISIE'94, Proceedings of the IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, Santiago, Chile, May, 1993.
- FILLMORE, C., *The Case for Case*, In: Universals in Linguistic Theory, E.Bach & R.T. Harms (Eds), Holt, New York, 1968.
- FILLMORE, C.J., *Types of Lexical Information*, In: Semantics (Edited by D.Steinberg and L.Jokovitz), p. 370-392, Cambridge Univ. Press, 1971.
- FOUNTAIN, W., State Logic speeds PLC programming, *I & CS*, fevereiro, p. 83-85, 1994.
- FRACHET , J.P., Notas do Curso: *Uma Introdução à Engenharia Integrada dos Sistemas automatizados de Produção*, ministrado pelo Prof. Frachet, 01-03 Dezembro 1993, CT/UNICAMP, Brasil.
- FRENCH, A.C., Software Engineering Applied to Programmable Controller Software Design, *ISA Transactions*, Vol. 29, no. 2, p. 23-32, 1990.
- GANE, C., T. SARSON. *Análise estruturada de Sistemas*, Prentice Hall, 1979. 1ª. Edição
- GARR, D.T., A.G. HAMMER. Concepts of a Commercially Available Software Development Environment. In: *Proceedings of 14th ANNUAL ADVANCED CONTROL CONFERENCE*, September, 1988.
- GRAHAM, J.H., G. N. SARIDIS. Linguistic Decision Structures for Hierarchical Systems. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-12, p. 325- 333., 1982.

- GUNASENA, N. U., E.A. LEHTIHET. Automatic Program Generation for PC-based Software Control Problems. *Journal of Manufacturing Systems*, Vol.10, no. 2, p. 109-120, 1991.
- HALANG, W.A., B.KRÄMER. Achieving high integrity of process control software by graphical design and formal verification. *Software Engineering Journal*, January, 1992, p. 53-64.
- HAREL, D., LACHOVER, H., A.NAAMED, A. PNUELI, M.POLITI, R.SHERMAN, A.S. TRAURING, M. TRAKHTEMBROT. StateMate: A working environment for the development of complex reactive systems, *IEEE Transactions on Software Engineering*, v. SE-16, n. 4, p. 403-413, 1990.
- HILAL D.K., H.SOLTAN, To prototype or not to prototype? that is the question. *Software Engineering Journal*, november, pp: 388-392, 1992
- HITACHI H-200, Controlador Programável, *Manual de Operação*, MAN-02, Sistema Automação S.A., 1994
- HO SANG, H., SEOK CHAM, J., YOUNG-HUI, K. Real Time shop floor control system for PCB auto insertion line based on object oriented approach. *Computers and Industrial Engineering*. Vol.30, No.3, p. 543-555, 1996.
- HODGE, L.R., M.T. MOCK, A proposed object-oriented development methodology, *Software Engineering Journal*, March, 1992, p. 119-129.
- HYUNBO CHO, MOOYOUNG JUNG, MOON HO KIM. Enabling technologies of agile manufacturing and its related activities in KOREA. *Computers and Industrial Engineering*. Vol. 30, No. 3, p.323-334, 1996.
- JOANNIS, R., M. KRIEGER, Object Oriented approach to the specification of manufacturing systems. *Computer-Integrated Manufacturing Systems*, v.5, n.2, p. 27-35, 1992.

JOSHI, S.B., E.G. METTALA, R.A. WYSK. CIMGEN: A Computer Aided Software Engineering Tool for development of FMS Control Software. *IIE Transactions*, v.24, n.3, p. 84-97. 1992.

JOSHI, S.B., M.R.SUPINSKI. The development of a generic PC-based programmable logic controller simulator. *International Journal of Production Research*, 1992, vol.30, no.1, p. 151-168

KNAPP,G.M., H.P.WANG. Modelling of Automated Storage/Retrieval Systems using Petri Nets. *Journal of Manufacturing Systems*, v. 11, n.1, p. 20-28, 1992.

KOWALSKI, R.A. Predicate Logis as Programming Language. In: *Proceedings of the IFIP CONFERENCE '74*, Stockholm. 1974.

LEE, K.H., SEN, S., ICOSS: A two-layer object based intelligent cell control architecture. *Computer Integrated Manufacturing Systems*, 7 (2), p.100-112.1994.

LEE, S.M., C.S. CHEN. Robot Programming in the Automated Manufacturing Programming Language Environment (AMPLE). *Computers & Industrial Engineering*. Vol.21, nos. 1- 4. p. 525-528, 1991.

LEGEARD, B., M. RUEHER. Une démarche pour le prototypage en Prolog. *Technique et Science Informatiques*, vol.8, no.5, 1989, p. 423-437.

LEVAS, A., R. JAYARAMAN. WADE: An Object-Oriented environment for modelling and simulation of workcell applications. *IEEE Transactions on Robotics and Automation*, vol.5, no.3, Junho, 1993, p. 324-335.

LEWIS, B. Mind your Language, *Electrical Review*, August, p. 24-26, 1993.

LU, H, Z.YING, T.W. LIAO. Simulation of Programmable Logic Controller. *Computers and Industrial Engineering*, vol.23, nos.1-4, p. 351-354, 1992

MENGA G., M.MORISIO. Prototyping discrete part manufacturing systems. *Information and Software Technology*, vol.31, no.8, outubro, p. 429-437, 1989.

MIYAGI, P.E. Apostila: *Controle de Processos II, Sistemas de Eventos Discretos*. Escuela de Robótica, 1993, CINVESTAV- IPN.

MIYAGI, P.E., C.T. MOTOHASHI, D.F. DOS SANTOS. Mark Flow Graph e suas extensões para análise e controle de Sistemas de Manufatura. In: *Anais do XII CONGRESSO BRASILEIRO DE ENGENHARIA MECÂNICA*, Brasília, DF, Brasil. p. 297-300, 1993.

MIYAGI, P.E., C.T. MOTOHASHI, M.Y. KAGOHARA. Metodologia de Projeto de Sistemas de Controle de Sistemas Automatizados de fabricação. In : *Proceedings I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE*, Rio Claro, SP, Brasil. p. 483-490 , 1993.

MIZE, J.H., H.C. BHUSKUTE, D.B. PRATT, M.KAMATH. Modeling of Integrated Manufacturing Systems using an Object - Oriented Approach. *IIE Transactions* v.24, n.3, p. 14-26. 1992.

MOCK, M.T., L.R. HODGE. An exercise to prototype the object oriented development process. *Software Engineering Journal*, March, p. 114-118, 1992.

MODICON 984 Programmable Controller Systems. *Manual*, Modicon, Inc. Industrial Automation Systems, One High Street, North Andover, Massachusetts 01845.

MOURANT R.R. An Interface for hierarquical modeling in Object-Oriented Simulation. *Computers and Industrial Engineering*, v.23, n 1-4, p. 233-235, 1992.

MURPHY, P. Fuzzy Logic smooths system control. *Instrumentation & Control Systems*, March, p. 45-49, 1992.

NAYLOR, A.W., R.A. VOLZ. Design of Integrated Manufacturing System Control Software. *IEEE Transactions os Systems, Man and Cybernetics*. Vol. SMC-17, no. 6. p. 881- 897. 1987.

PEIXOTO DE OLIVEIRA, J.C. *Controlador Programável*, São Paulo: McGraw-Hill, 1993.

PEREIRA, F.C.N. E D.H.D. WARREN. Definite Clause Grammars for Language Analysis- A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, Vol.13 , p. 231- 278. 1980.

QUIROZ, J.H. Redes de Petri aplicadas a Controladores de Lógica Programable. In: *Anais do CONGRESO NACIONAL DE INGENIERÍA ELÉCTRICA*, Santiago, Chile, pp. e55-e60, 1993.

RAM, B., S.H.-Y. LAI. The Development of a Program Generator for Programmable Logic Controllers. *Computers & Industrial Engineering*, Vol.23, nos. 1- 4, p. 335- 339, 1992.

RICH, E. *Inteligência Artificial* . , São Paulo (SP): McGraw Hill Brasil, 1988.

RICH, C., R.WATERS. Automatic Programming : Myths and Prospects. *Computer*, August, 1988, p. 40-51.

ROBERTS, C.A., T. BEAUMARIAGE. A Specification Technique for Generating and Simulating Supervisory Control. *Computers and Industrial Engineering*, vol.25, nos. 1-4, p. 515-518, 1993.

ROCHA, J., RAMOS, C. Generating and Converting Plans for Intelligent Manufacturing Systems. *Proceedings of the IEEE International Symposium on Industrial Symposium on Industrial Electronics*. Santiago, Chile, p. 304-309. Maio, 1994

ROSS, D.J. Structured Analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering*, v.3, n.1, 1977.

RUEHER, M, M.C. THOMAS, A. GUBERT, D. LADRET. *A prolog based graphical approach for task level specifications*. Languages for sensor based control in Robotics, Rembold e Hörmann editores, p. 201-215. 1987.

RUEHER, M.E B. LEGEARD. Prolog et le prototypage. *Revue genie logiciel*, No.3, p. 19-22, 1995

- SACKS, T. PLC ESPERANTO: the route to lower costs? *Electrical Review*, June, 1993. p. 25-27.
- SANTOS, D.J., J. ARARAKI, P.E. MIYAGI. Controle de tarefas de um sistema de transporte por veículos Autônomo - Aplicação de Técnica de MFG / PFS em Sistemas Integrados de manufatura. In: *Proceedings 90 CBA, CONGRESSO BRASILEIRO DE AUTOMÁTICA*, SBA, Vitória, 1992.
- SHAW, M. Abstraction Techniques in Modern Programming Languages, *IEEE Software*, October, 1984, p. 10-26.
- SILVA, J.C.A., P.R.POLITANO, R.P. DE M FORTES, A.J.V. PORTO. Aplicando StateCharts à modelagem do Sistema de Controle de Produção de uma Célula Flexível de Manufatura. In: *Proceedings I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE*, Rio Claro, SP, Brasil. p. 491-499, 1993.
- SILVA, J.R., AFSARMANESH, H., COWAN, D.D., LUCENA, C.J.P. An Object-Oriented Approach to the Design of Manufacturing Systems. *Proceedings of the IEEE/ECLA/IFIP International conference on architectures and design methods for balanced automation systems*, Vitoria (ES), Brasil, p.91-106, 1995.
- SILVA, J.R., MIYAGI, P.E. PFS/MFG: A High Level Net for the Modeling of Discrete Manufacturing Systems. *Proceedings of the IEEE/ECLA/IFIP International conference on architectures and design methods for balanced automation systems*, Vitoria (ES), Brasil, p.349-362, 1995.
- STEFIK, M.J., D.G. BOBROW. Object Oriented Programming: Themes and Variations. *The AI Magazine*, p. 40-62.
- STERLING, L., E.SHAPIRO *The Art of Prolog*. The MIT Press, Cambridge MA.USA. 1986.
- SU, H.M., V. KACHITVICHYANUKUL. A Natural Language System to Aid Simulation Model Formulation. *Computers & Industrial Engineering*, Vol. 16, no. 4, p. 535-543, 1989.
- SZPAKOWICZ, S. Logic Grammars. *Byte* August 1987, p. 185-195.

TAM, K.Y. Linguistic Modelling of Flexible Manufacturing Systems. *Journal of Manufacturing Systems*. Vol.8, no.2 p. 127-137. 1989.

TOGINO, K., FURUTA, K., K.KOBASHI. A Controller Independent Programming Language for Sequentially Controlled Machine. *Advanced Manufacturing Technology*, North Holland, IFIP, 1980.

ULGEN, O.M., T. THOMASMA. SmartSim: An object oriented simulation program generator for manufacturing systems. *International Journal of Production Research*, vol. 28, no.9, p. 1713-1730, 1990.

WALKER, I.J. Requirements of an Object-oriented design method. *Software Engineering Journal*, March, p. 102-113, 1992

WASSERMAN, A.I., P.A.PIRCHER, R.J. MULLER. The Object- Oriented Structured Design Notation for Software Design Representation. *Computer*, v.23, n.3, p. 50-63, 1990.

WEG Controlador Programável A080, *Manual de Programação*.

WITZGERMAN, J.P., NOF, S.Y. Integration of Simulation and emulation with graphical design for the development of cell control programs. *International Journal of Production Research*, Vol.33, No. 11, p.3193-3206, 1995.

WOLCZKO, M. Encapsulation, delegation and inheritance in object-oriented languages. *Software Engineering Journal*, March, p. 95-101, 1992.

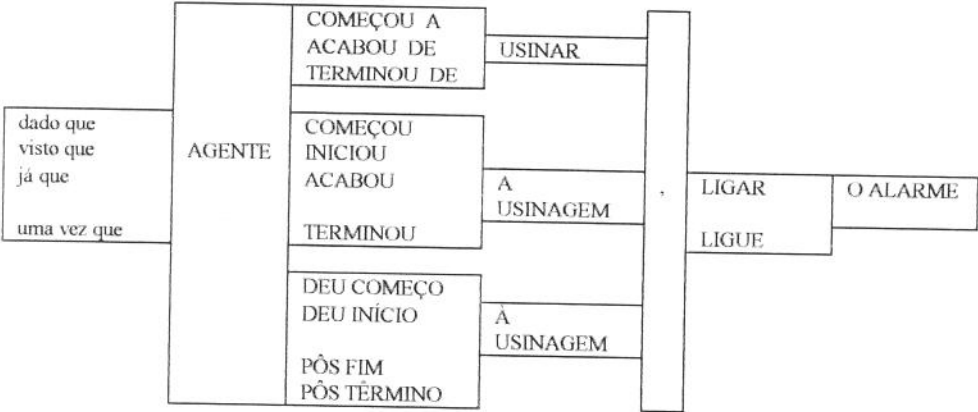
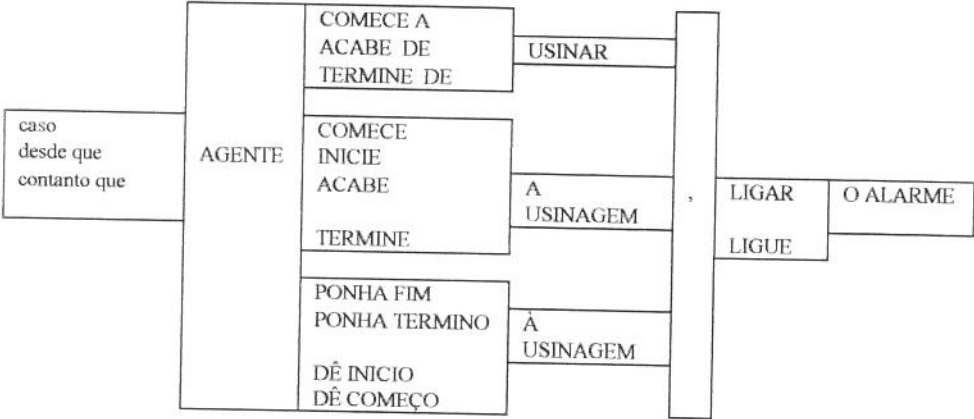
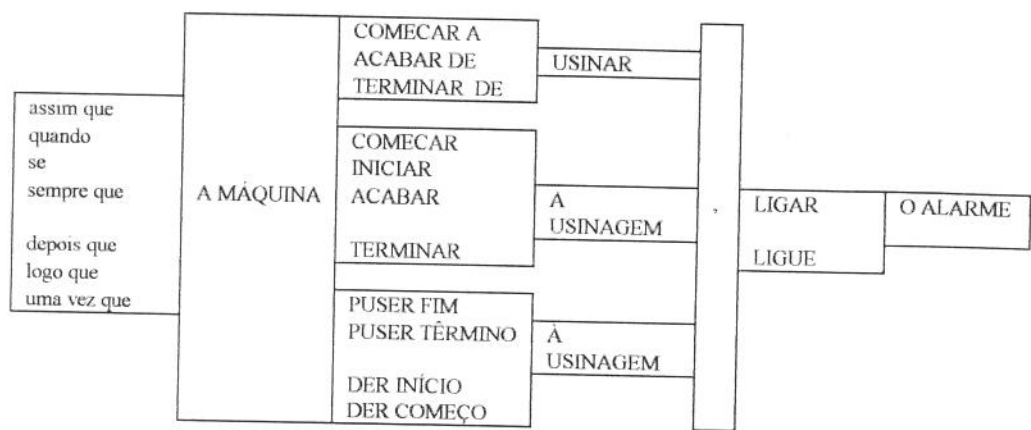
WOODS, W.A. Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*, Vol.13. p. 591-606, 1970.

WU, H.L., VENUGOPAL, R., M.M. BARASH. Design of a Cellular Manufacturing: A Syntactic Pattern Recognition Approach. *Journal of Manufacturing Systems*, Vol.5, p. 81-87, 1986.

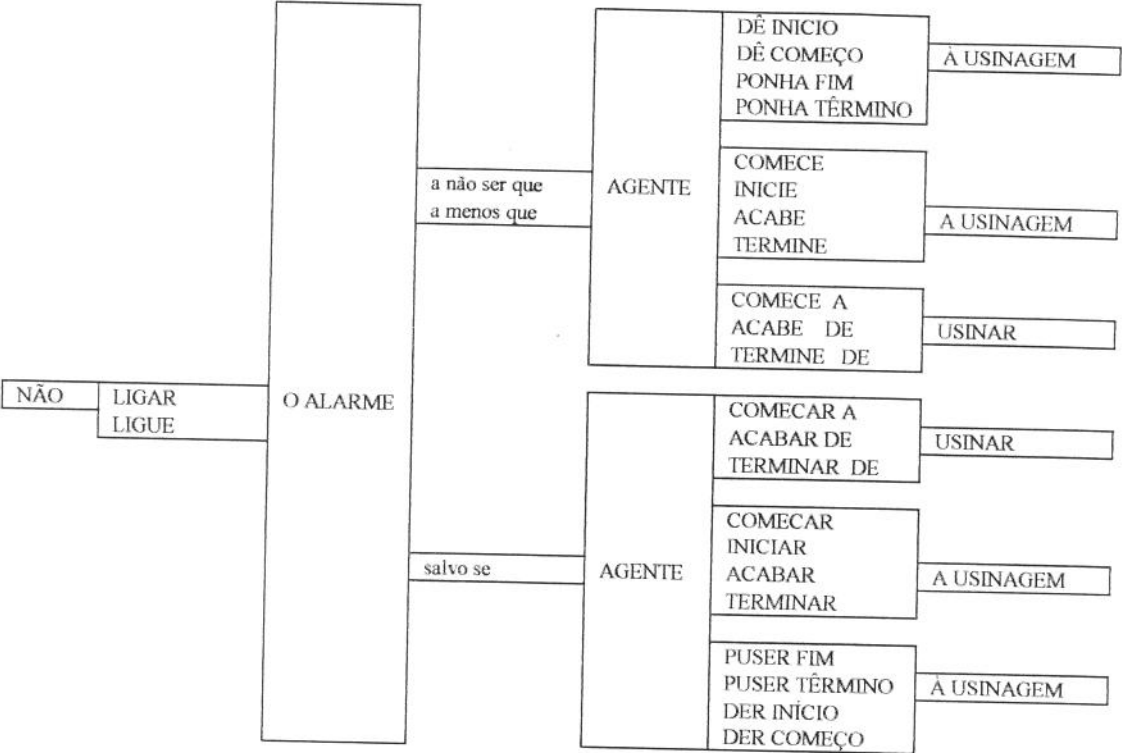
ZAPOLIN, R.E. RISC Programmable Controllers can bring Programability down to low-end remote sites. *ISA Transactions*, Vol. 29, no. 2, p. 81-87, 1990.

Anexos

TIPO A : CONJUNÇÃO + (AGENTE + VERBO_AUX + VERBO) + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO + OBJETO



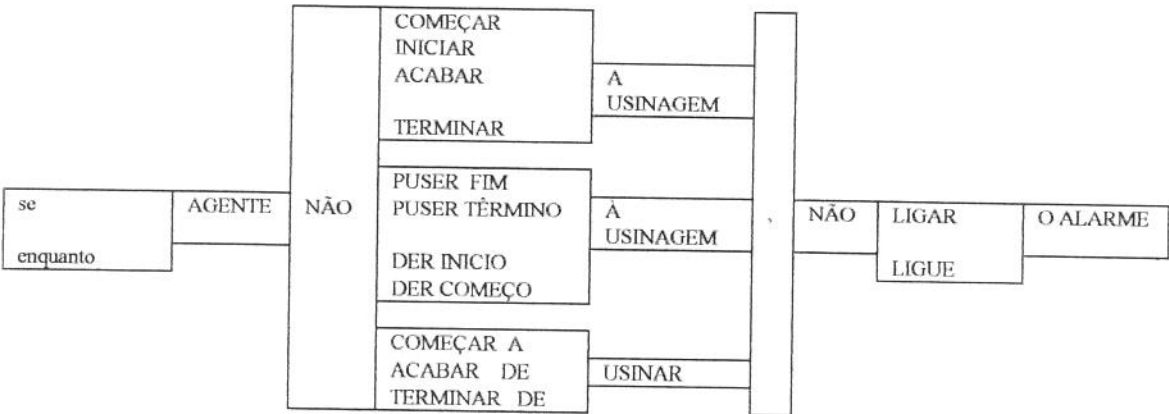
TIPO B : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO
+ [E/OU + (AGENTE + VERBO_AUX + VERBO)]



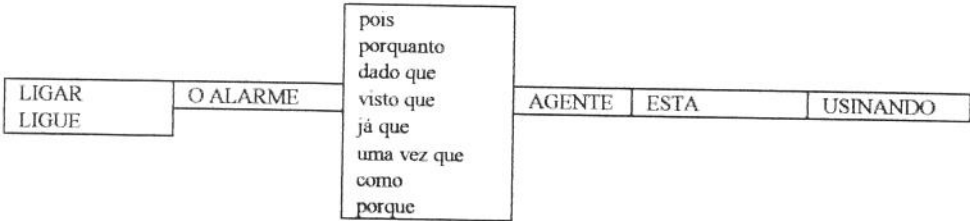
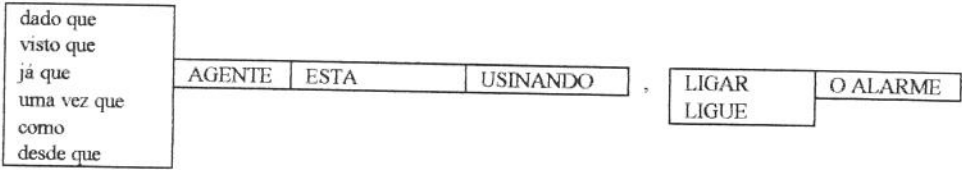
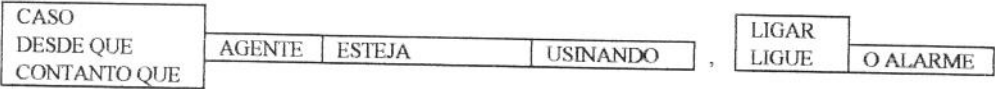
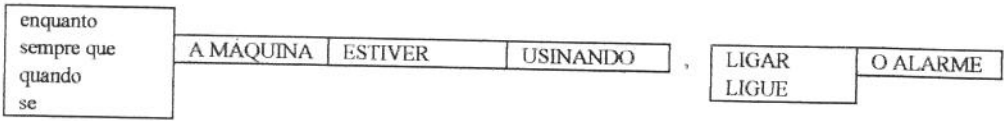
TIPO C : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO + [E/OU + (AGENTE + NEGAÇÃO + VERBO_AUX + VERBO).....]



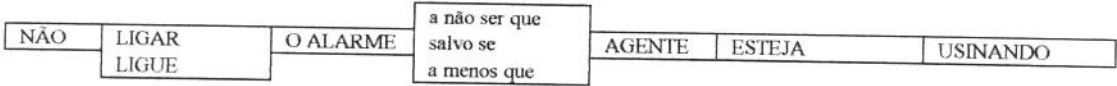
TIPO D : NEGAÇÃO + VERBO + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO + [E/OU + (AGENTE + VERBO_AUX + VERBO)]



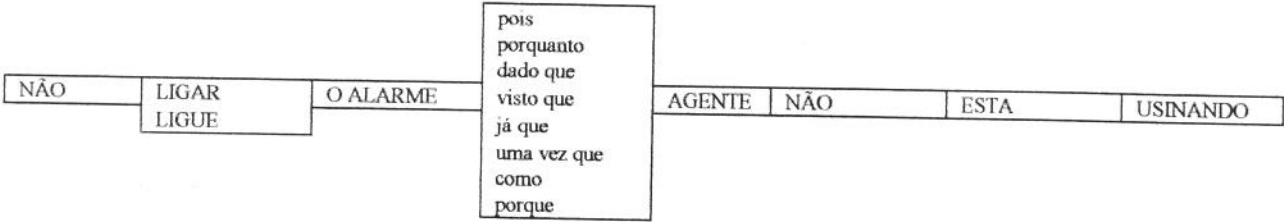
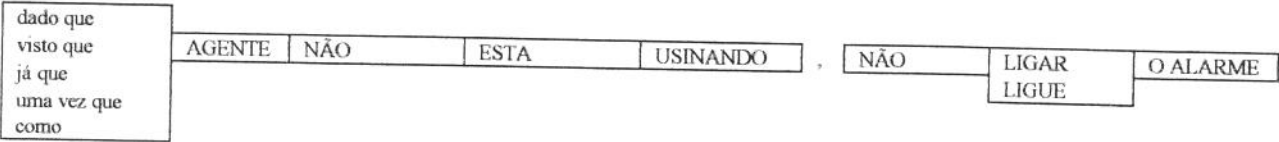
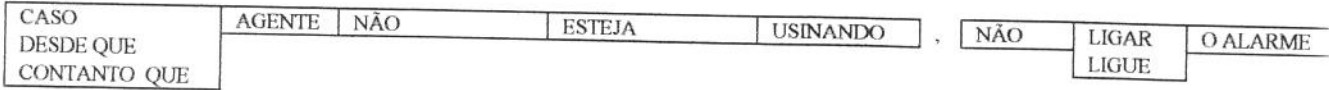
TIPO A : CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO_1 + OBJETO.



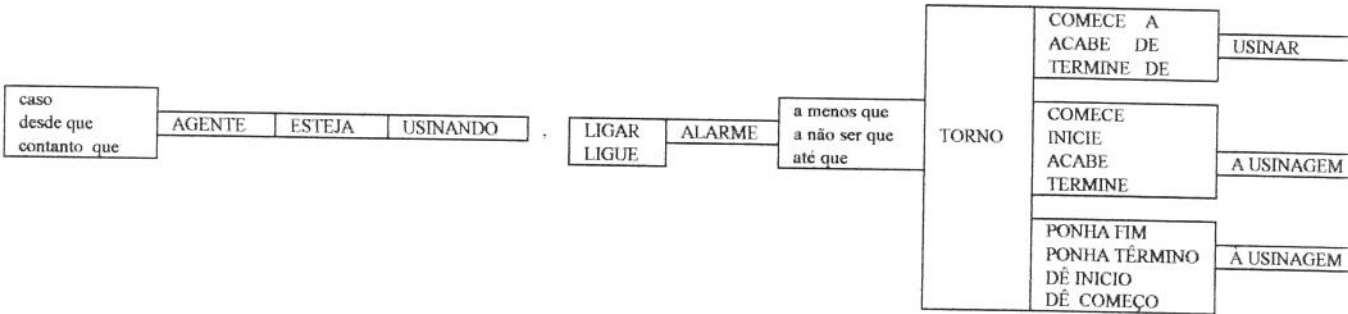
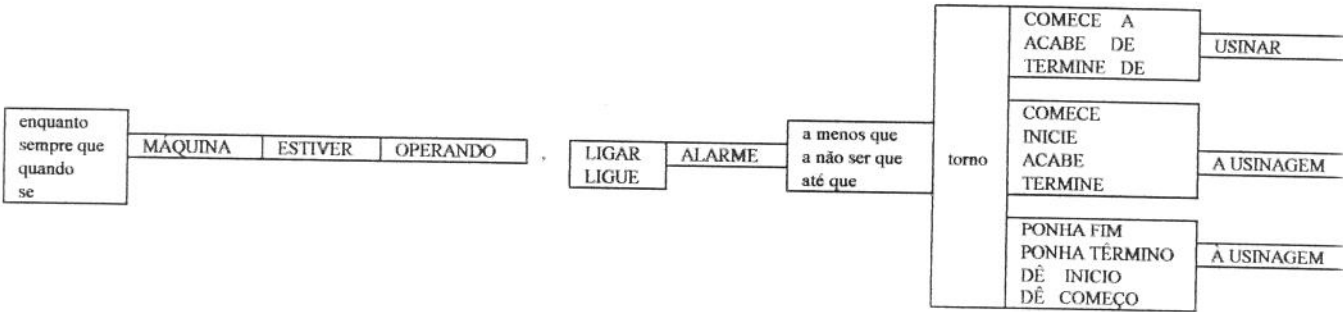
TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

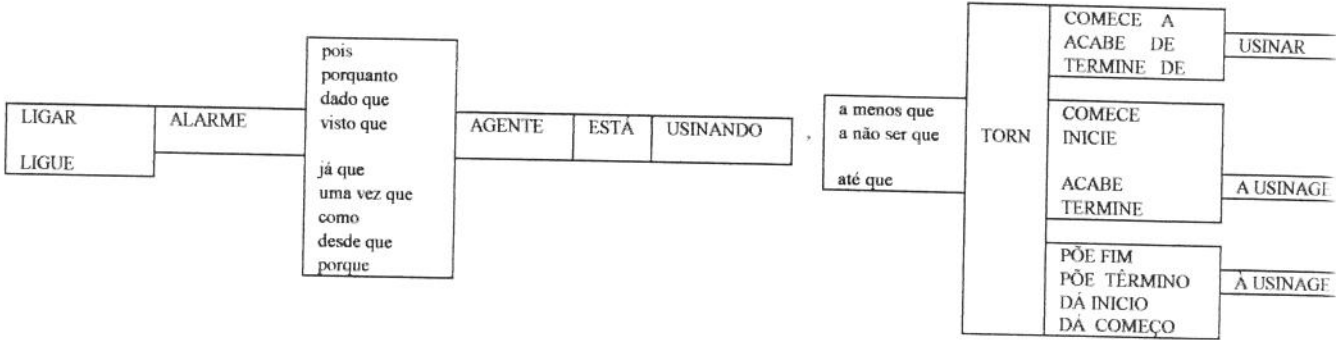
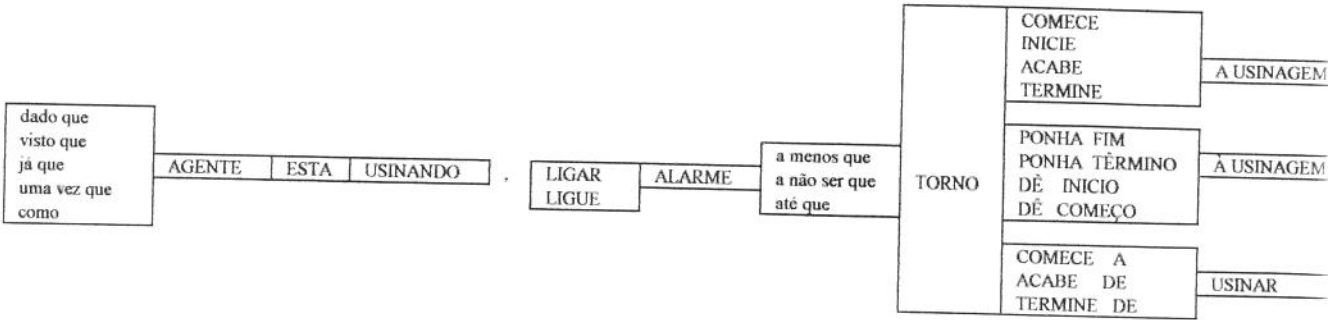


TIPO C : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO + AGENTE + NEGAÇÃO + VERBO_AUX + VERBO_2+ [E/OU + (AGENTE + VERBO_AUX + VERBO)]

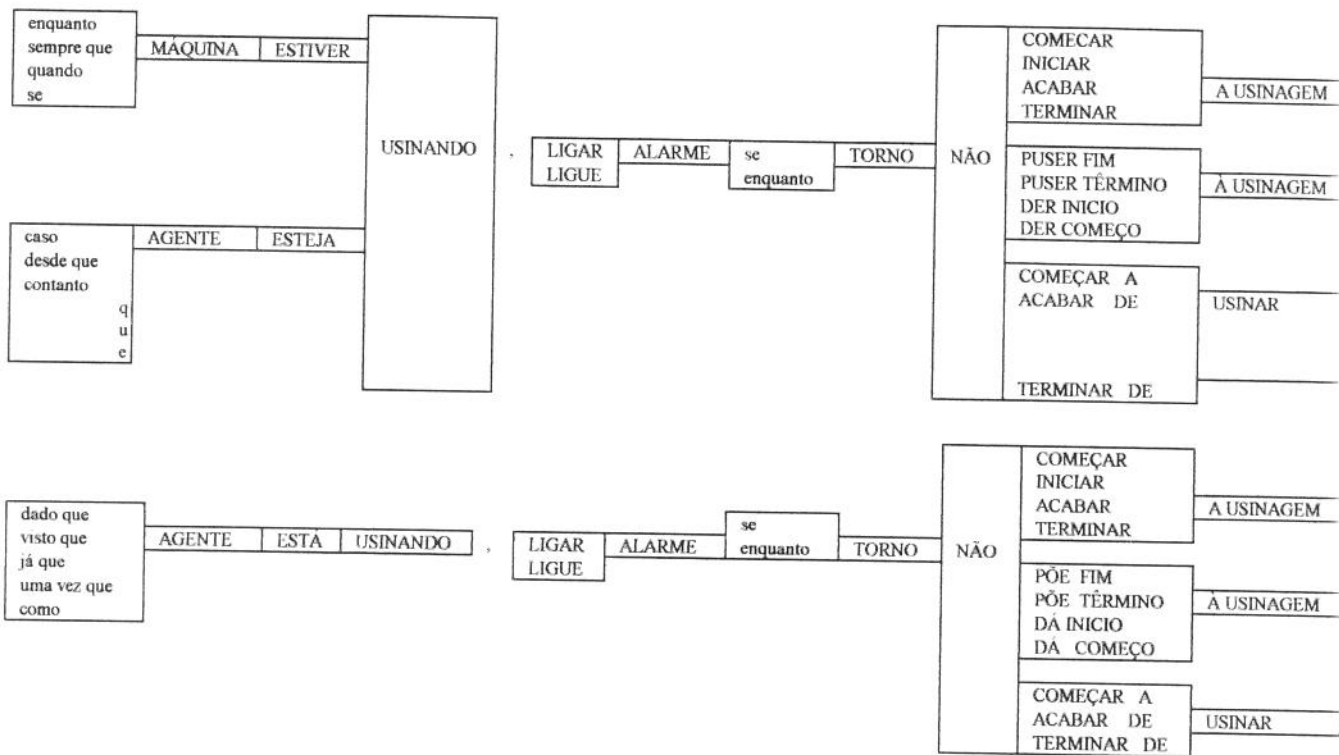


TIPO A : CONJUNÇÃO_1 + AGENTE_1 + VERBO_AUX_1 + VERBO_1 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + , + VERBO_3 + OBJETO + CONJUNÇÃO_2 + AGENTE_2 + VERBO_AUX_2 + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]

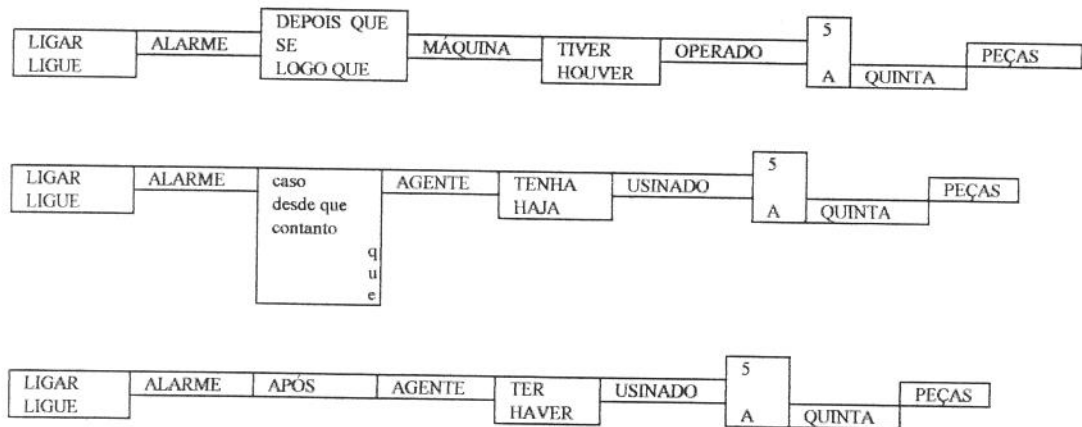




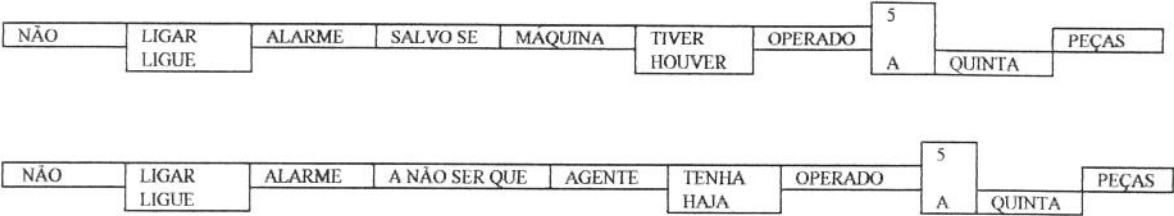
TIPO B : CONJUNÇÃO_1 + AGENTE_1 + VERBO_AUX_1 + VERBO_1 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + . + VERBO_3 + OBJETO + CONJUNÇÃO_2 + AGENTE_2 + NEGAÇÃO + VERBO_AUX_2 + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)]



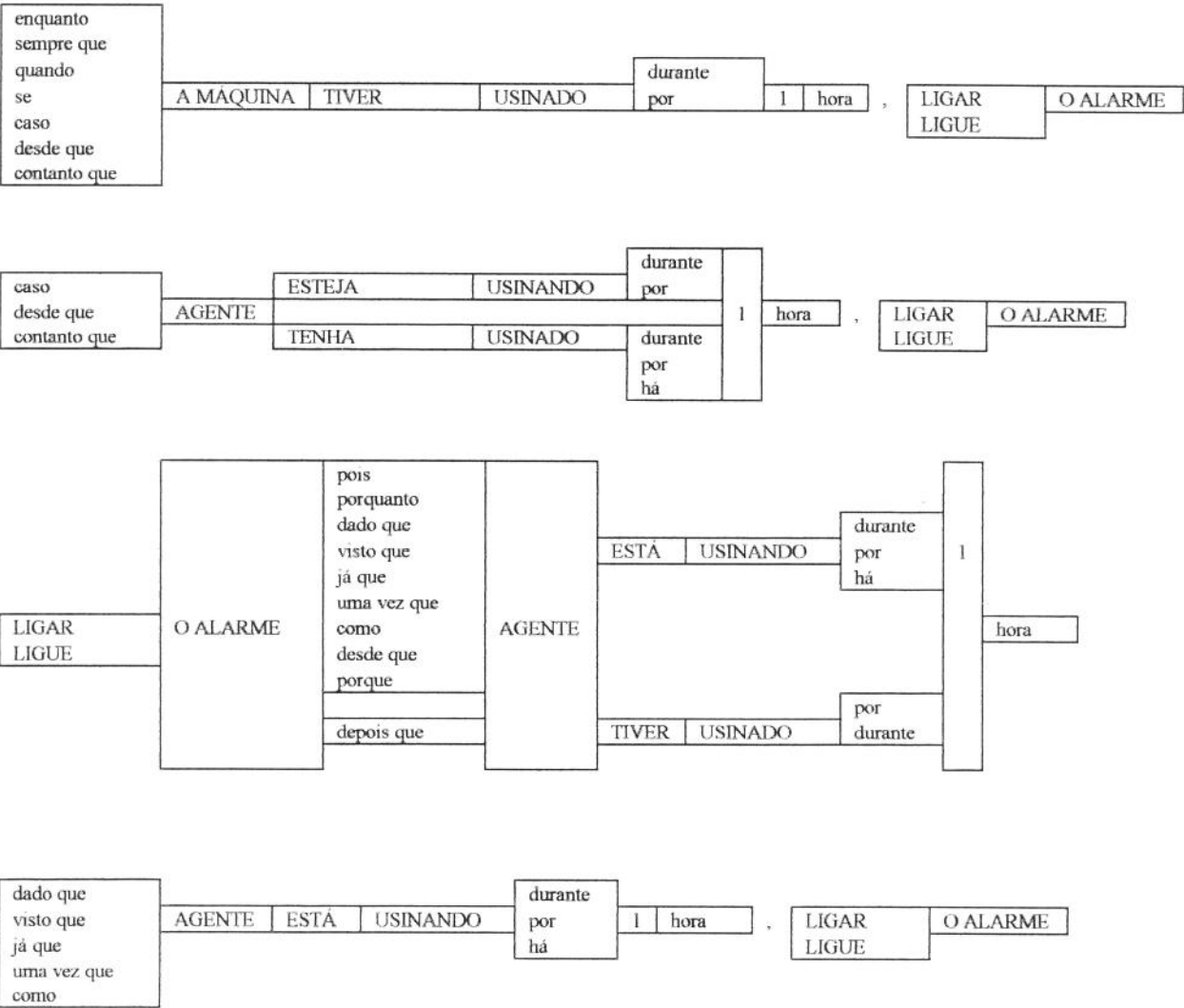
TIPO A : VERBO_1 + OBJETO + CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + NUMERO + UNID_CONTAB



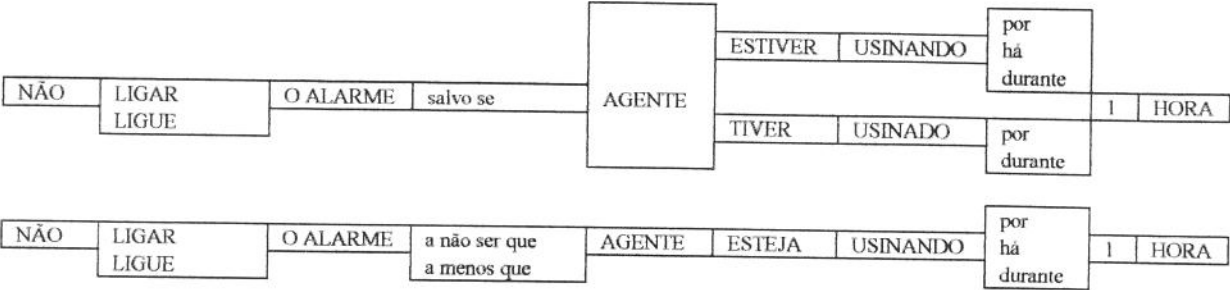
TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + PREPOSIÇÃO + NUMERO + UNID_TEMPO



TIPO A : CONJUNÇÃO_1 + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + CONJUNÇÃO_2 + NÚMERO + UNID_TEMPO + , + VERBO_1 + OBJETO.



TIPO B : NEGAÇÃO + VERBO_1 + OBJETO + CONJUNÇÃO + AGENTE + VERBO_AUX + VERBO_2 + [E/OU + (AGENTE + VERBO_AUX + VERBO)] + CONJUNÇÃO_2 + NÚMERO + UNID_TEMPO.



ARTIGOS PUBLICADOS:

"A High Level Object - Oriented Programmable Controller Programming Interface", Proceedings ISIE '94, IEEE International Symposium on Industrial Electronics, Santiago, Chile, Maio 1994.

"Geração Automática de Programas para Controladores Lógicos Programáveis através de uma Interface Neutra". Anais do III Congresso de Engenharia Mecânica Norte Nordeste, Junho de 1994, Belém (PA), Brasil.

"El Paradigma de Objeto e su uso como herramienta de Especificación para Diseño Sistemas de Manufactura". Atas do VI Congreso Nacional de Ingeniería Mecânica, Novembro de 1994, Santiago, Chile.

"Object Oriented Development Methodology for PLC software", Proceedings of the IEEE/ECLA/IFIP International Conference on Architectures and Design Methods for Balanced Automation Systems, Julho de 1995, Vitoria (ES).

"Generating Manufacturing Code with an Object-Oriented Development Methodology", Proceedings do LCA'95 IFAC, 5th. Symposium of Low Cost Automation, Buenos Aires, Argentina, Setembro de 1995.

"Caracterização das Linguagens de Programação para CLPs", aceito para ser apresentado no IV Congresso de Engenharia Mecânica Norte Nordeste, em Junho 1996, Recife (PE).

"O Paradigma de Objeto e Elementos de Inteligência Artificial para Geração Automática de Código de Controle", Revista do Instituto de Informática, PUCCAMP, vol. 2, no.2, Junho/Dezembro, 1994.