PUCPR – Pontifícia Universidade Católica do Paraná

PPGEPS – Programa de Pós Graduação em Engenharia de Produção e Sistemas

JONERLAM ROBERTO CARVALHO

CONTRIBUIÇÕES A IMPLEMENTAÇÃO DA ESTRUTURA DE CONTROLE MODULAR LOCAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

CURITIBA

2007

JONERLAM ROBERTO CARVALHO

CONTRIBUIÇÕES A IMPLEMENTAÇÃO DA ESTRUTURA DE CONTROLE MODULAR LOCAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

Área de Concentração: Automação e Controle de Sistemas

Orientador: Prof. Dr. Eduardo Alves Portela Santos

CURITIBA

2007

CARVALHO, Jonerlam Roberto

Contribuições a Implementação da Estrutura de Controle Modular Local. Curitiba, 2007. 136p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

1. Sistemas a Eventos Discretos; Teoria de Controle Supervisório; Controle Modular Local; Automação de Baixo Custo; Controladores Lógico Progamaveis; Microcontroladores; Implementação. I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

Ata de defesa e termo de aprovação.

À minha esposa Cristiane.

... aprendi que o mais valioso não é **o que temos** em nossas vidas, mas sim, **quem temos** em nossas vidas.

(Ademir Goulart)

Agradecimentos

Ao meu orientador Professor Dr. Eduardo Alves Portela Santos pelo estímulo e parceria, além de sua constante orientação e permanente incentivo para o desenvolvimento deste trabalho.

Ao Professor Dr. Eduardo Rocha Loures e Professor Dr. Agnelo D. Vieira pelas inúmeras sugestões e contribuições realizadas durante o desenvolvimento deste trabalho.

À ISOTRON por ter liberado muitas das minhas horas de trabalho e pelo auxílio concedido, sem o qual este trabalho não poderia ter sido realizado. Aos meus chefes Edgar Wilkens e Frankin Wilkens pela confiança que depositam em mim, encarando minhas ausências como um investimento profissional.

À PUCPR, pela bolsa de mestrado concedida.

Ao meu irmão Jonathan Marcelo Carvalho, pela colaboração e apoio durante a última fase deste trabalho e pelo companheirismo e boas conversas que tivemos durante todos estes anos.

À minha esposa e familiares, por terem suportado o meu mau-humor nos períodos em que estive sob grande pressão e pela minha ausência em inúmeras ocasiões.

A todos os demais, não nomeados, que de forma direta ou indireta contribuíram para a realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta contribuições a implementação da Estrutura de Controle Modular Local, a partir da utilização de plataformas de controle industriais. Propõe-se a implementação desta estrutura em dois dispositivos de larga utilização – Controladores Lógico Programáveis (CLP) e Microcontroladores – de forma distribuida. Na implementação em CLP, o trabalho faz um estudo sobre a forma de utilização de variáveis, identificando uma redução de memória de dados pela escolha de variáveis com maior grau de registros (variáveis do tipo inteiros). Nesta implementação é feito um comparativo entre duas formas de construção do programa, antecipando, através de fórmulas, todos os recursos de memória necessários ao CLP. Na utilização do microcontrolador, o modelo encontrado na literatura em três níveis, é implementado em linguagem C ANSI, padrão a esses dispositivos. Neste desenvolvimento são descritas as particularidades de adaptação da teoria com o processamento seqüencial dos microcontroladores. O presente trabalho também propõe a criação de um novo método de implementação, que considera uma nova forma de comunicação entre os níveis de controle da estrutura modular local. Estas novas abordagens visam uma redução no cursor global do projeto de um sistema automatizado.

Palavras-Chave: Sistemas a Eventos Discretos; Controle Modular Local; Automação de Baixo Custo; Implementação; Supervisão; Controladores Lógico Programáveis; Microcontroladores.

Abstract

This work presents contributions for the implementation of the Structure of Local Modular Control from the use of industrial control platforms. It proposes the implementation of this structure into two devices of wide use – Program Logic Control (PLC) and Microcontrollers – in a distributive way.

In the CLP implementation, the work studies the way of the use of variable, identifying a data memory reduction for the choice of variable with larger registration degree (entire type variable). In this implementation a comparison is between two forms of program construction, for anticipating, through formulas, all the necessary resources of memory to the CLP.

In the use of the microcontroller, the model found in literature in three levels, is implemented in C ANSI language, standard to these devices.

In this development, adapting the theory particularities are described with the sequential microcontrollers processing. This present work also considers a new method creation of implementation that considers a new communication form between the local modular structure control levels. These new approaches aim a reduction in the global cursor of an automatized system project.

Key-words: Discrete Events Systems, Local Modular Control, Automation of Low Cost, Implementation, Supervision, Program Logical Control, Microcontrollers

Capítulo 1

Introdução

A flexibilidade exigida dos modernos sistemas automatizados de manufatura conduziu a uma evolução tecnológica dos dispositivos de controle, trazendo a este uma maior capacidade de memória, mais funcionalidades e maior capacidade de conectividade a outros dispositivos. Por outro lado, a utilização de novas abordagens e ferramentas formais no projeto de sistemas de controle para sistemas flexíveis de manufatura ainda é considerada incipiente em aplicações industriais.

De acordo com Erbe (2002), muitas empresas têm encontrado dificuldades em adotar a automação como uma estratégia competitiva. De acordo com o autor, a principal razão para tal fato é a insuficiente flexibilidade observada em sistemas altamente automatizados. A combinação de perdas em conseqüência da conversão, o tempo ocioso de equipamentos e o alto custo de manutenção especializada restringem sensivelmente o lucro esperado. Nesse contexto, muitas empresas têm sistematicamente diminuído o nível de automação do chão-de-fábrica ou planejam fazê-lo.

Para a solução de problemas relacionados à concepção de Sistemas de Controle (SC) é fundamental estabelecer procedimentos sistemáticos que caracterizam o ciclo de desenvolvimento do SC. Essa sistematização consiste em utilizar modelos formais para a análise, síntese e implementação do SC. Cassandras e Lafortune (1999) enumeram os principais modelos utilizados: Redes de Petri com e sem temporização, Redes de Petri Controladas com e sem temporização, Cadeias de Markov, Teoria das Filas, Processos Semi-Markovianos Generalizados e Simulação, Álgebra de Processos, Álgebra Max-Plus, Lógica Temporal e Lógica Temporal de Tempo Real, Teoria de Linguagens e Autômatos.

Dentre os modelos citados anteriormente, as Redes de Petri Controladas (Holloway et al., 1997) e o modelo de Ramadge e Wonham (1989), denominado Teoria de Controle Supervisório (TCS), baseado em Linguagens e Autômatos (Carrol e Long, 1989), são mais adequados ao desenvolvimento de SC. Diferentemente dos outros modelos que enfatizam a análise de sistemas, os dois modelos citados são dotados de procedimentos de síntese de controladores.

O modelo proposto por Ramadge e Wonham (1989) propicia um processo automático de síntese do controle, ao invés dos usuais procedimentos manuais ou heurísticos. Além desta vantagem, o procedimento de síntese impõe que o supervisor obtido sempre atende as especificações de controle. Assim, novos SC podem ser rapidamente e automaticamente projetados quando são necessárias modificações, tais como redefinição de especificações e mudanças físicas. Além disso, a idéia de síntese do supervisor minimamente restritivo, característica desta abordagem, atribui um maior grau de liberdade ao SC. Por estas razões, o presente trabalho utiliza a TCS como ferramenta formal de obtenção de um SC para sistemas automatizados de manufatura.

1.1 JUSTIFICATIVAS

A flexibilidade trás um aumento exponencial na lógica de controle, em função da quantidade de especificações (tarefas a serem cumpridas) e conseqüente número de estados a serem observados e controlados. Dessa forma, o projeto de um SC requer um cuidado em relação não somente a capacidade do hardware, memória de dados e programas, como também em relação a métodos formais mais eficazes. Nesse contexto, é relevante abordar conjuntamente sistemáticas baseadas em formalismos para o projeto de SC bem como dispositivos físicos (tecnologias) que reduzam o custo global do projeto. A sistemática formal adotada deve considerar então não somente o projeto teórico do SC, mas também a atual capacidade tecnológica dos dispositivos eletrônicos, computacionais e a realidade da economia local.

1.2 OBJETIVOS

Este trabalho tem por objetivo geral propor soluções otimizadas para a implementação da estrutura de controle supervisório em dispositivos industriais de controle. Para tanto, é adotada a Teoria de Controle Modular Local – TCML – (derivada da TCS) e um modelo de implementação da sua estrutura de controle. Para alcançar o objetivo geral, lista-se a seguir os seguintes objetivos específicos:

1. Propor uma nova forma de programação da estrutura de controle modular local em Controladores Lógico Programáveis;
2. Comparar a nova forma com a encontrada na literatura, em relação ao tamanho do programa (quantidade de memória requerida);
3. Propor a implementação da estrutura de controle modular local em microcontroladores;
4. Analisar a factibilidade da realização do SC (baseado na TCML) em microcontroladores;
5. Propor uma sistemática para a geração de código a partir de ferramentas computacionais de síntese de controladores.

Espera-se como resultado final que seja proposta uma sistemática em que ocorra redução no custo do projeto, implementação e implantação de um SC. O emprego de dispositivos programáveis de baixo custo (microcontroladores), novas formas de programação, a geração automática do código (vislumbrando antecipadamente a capacidade de memória) e a utilização de formalismos, serão os aspectos chaves para tal resultado.

1.3 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está estruturada da seguinte forma: no capítulo 2 é apresentada a teoria de modelagem e controle de Sistemas a Eventos Discretos proposta por Ramadge e Wonham (1989). Inicialmente apresentam-se os conceitos básicos da teoria de Linguagens Formais e Autômatos, necessários para o entendimento da TCS. Em seguida, uma extensão da TCS, a Teoria de Controle Modular Local (TCML) proposta por Queiroz e Cury (2000) é

apresentada de forma a complementar o embasamento teórico necessário. Ao final, o um modelo de implementação da estrutura de controle modular local é apresentado.

No início do capítulo 3 é aprofundada a proposta de implementação da Estrutura de Controle Modular Local, estruturada em três níveis, segundo os moldes Queiroz et al. (2001), Queiroz e Cury (2002) e Vieira (2004). Para tanto é utilizado como modelo de planta a célula de manufatura didática denominada MPS, sendo esta implementada na linguagem Ladder. No decorrer do desenvolvimento é mostrado que mesmo no refinamento mais específico pode-se obter um ganho na implementação de controle. Segue-se comparando duas formas de implementação, uma utilizando variáveis de dois valores e a outra utilizando as variáveis inteiros. No final é apresentado um equacionamento para antecipar os recursos necessários para a implementação do sistema.

O Capítulo 4 inicia apresentando emprego de um dispositivo de baixo custo (microcontrolador) e uma linguagem de padronizada no gerenciamento de subsistemas. Defini as diferenças de processamento entre duas tecnologias (microcontroladores e CLP’s). Segue diferenciando a forma de execução dos dispositivos e modifica a transcrição do controle para o microcontrolador. No final do capítulo é sugerido e validado um novo método de implementação da lógica de controle.

Capítulo 2

Teoria de Controle Supervisório

2.1 Introdução

O avanço tecnológico tem produzido sistemas que necessitam de um tratamento formal específico para sua otimização e automação. Estes sistemas estão presentes em uma série de aplicações, incluindo a automação da manufatura, a robótica, a supervisão de tráfego, a logística (canalização e armazenamento de produtos, organização e prestação de serviços), sistemas operacionais, redes de comunicação de computadores, concepção de software, gerenciamento de base de dados e otimização de processos distribuídos (Cury, 2001).

Tais sistemas têm em comum a maneira pela qual percebem as ocorrências no ambiente a sua volta, o que se dá pela recepção de estímulos, denominados eventos. Estes eventos são, por sua natureza, instantâneos, o que lhes confere um caráter discreto no tempo. Sistemas com estas características são denominados Sistemas a Eventos Discretos (SED), em oposição ao sistema de variáveis contínuas, tratados pela teoria de controle clássica (Cury, 2001).

A modelagem e controle de SED é uma área de pesquisa de grande interesse atual, estimulada pela diversidade de aplicações, como é o caso de sistemas de manufatura. Nesse contexto, a teoria de controle de sistemas a eventos discretos proposta por Ramadge e Wonham (1989) se destaca pela sua aplicabilidade nos modernos sistemas de manufatura e pelo seu potencial para desenvolvimento de ferramentas teóricas.

O presente capítulo tem como um dos objetivos apresentar os fundamentos da teoria de controle de SED, introduzida por Ramadge e Wonham (1989). Apresenta ainda extensões da teoria clássica, especificamente a proposta por Queiroz e Cury (2001), que aborda o

problema de modelagem e controle de forma modular. Em seguida, são apresentados alguns modelos de implementação das abordagens propostas por Ramadge e Wonham (1989) e Queiroz e Cury (2001). A seção 2.2 do presente capítulo apresenta o formalismo utilizado nas referidas abordagens.

2.2 Linguagens e Autômatos como modelos para SED

Nesta seção apresentam-se os conceitos básicos da teoria de linguagens e autômatos, modelos estes requeridos para o entendimento da teoria de controle de SED introduzida por Ramadge e Wonham (1989). Para maior detalhamento dos assuntos tratados nesta seção, o leitor poderá consultar Carrol e Long (1989), Hopcroft e Ullmann (1979), Ziller (1993) ou Cassandras e Lafortune (1999), onde encontrará a origem do material apresentado.

De acordo com a definição de SED, este tem associado um conjunto de eventos, que por sua vez formam seqüências que descrevem o seu comportamento. Uma seqüência então especifica a ordem em que vários eventos ocorrem sobre o tempo, mas sem prover os instantes associados com suas ocorrências. Então, a palavra 'linguagem' vem do fato que se pode pensar num conjunto de eventos como um 'alfabeto' e seqüências de eventos como 'palavras' (Cassandras e Lafortune, 1999). A seguir as definições são apresentadas de maneira formal.

Seja um conjunto finito, não vazio, de símbolos distintos, denominado alfabeto S. De forma análoga à escrita habitual, a justaposição de símbolos de um alfabeto é denominada *palavra*. O conjunto de todas as palavras finitas compostas com elementos de S é denotado S^*, incluindo a cadeia nula, denotada por ε. Define-se uma linguagem L sobre o alfabeto S como sendo um subconjunto de S^*. Segundo esta definição, tanto S^* quanto $\emptyset$ são linguagens. Note que a linguagem vazia, $\emptyset = \{\}$, é diferente da linguagem formada apenas pela palavra nula ε.

O prefixo de uma cadeia w sobre um alfabeto S é qualquer cadeia $u \in S^*$ que possa ser completada com outra cadeia $v \in S^*$ para formar a cadeia w. Denota-se $u \leq w$, u prefixo de w. O prefixo fechamento de uma linguagem L é dado por $\overline{L} = \{u : \exists v \in S^* \wedge uv \in L\}$. Em palavras, $\overline{L}$ consiste de todas as cadeias de S^* que são prefixos de L. Em geral, $L \subseteq \overline{L}$. A

linguagem *L* é dita prefixo fechada se $L = \overline{L}$, ou seja, uma linguagem *L* é prefixo fechada se qualquer prefixo de *L* é também uma cadeia de *L*.

Considerando a evolução seqüencial de um SED e um alfabeto *S* correspondendo ao conjunto de eventos que afetam o sistema, pode-se afirmar que se um dado sistema produziu uma cadeia qualquer *w*, então produziu anteriormente todos os seus prefixos. Portanto, o comportamento lógico de qualquer sistema a eventos discretos em que não ocorram eventos simultâneos, pode ser representado por uma linguagem prefixo fechada.

O comportamento de um sistema a eventos discretos pode ser descrito através de um par de linguagens. A evolução seqüencial do SED, ou seu comportamento lógico, pode então ser modelado através de uma dupla *D = (L, Lm).*

No modelo *D*, *L Í S** é a linguagem prefixo fechada que descreve o comportamento gerado pelo sistema, ou seja, o conjunto de todas as cadeias de eventos fisicamente possíveis de ocorrerem no sistema. Após partir do estado inicial e percorrer uma determinada trajetória em seu espaço de estados, um SED acabará, via de regra, completando uma ou mais tarefas. As seqüências de eventos que levam à tarefas completas formam também uma linguagem. Assim, no modelo *D*, *Lm Í L* é a linguagem que descreve o comportamento marcado do sistema, ou seja, o conjunto de cadeias em *L* que correspondem a tarefas completas que o sistema pode realizar.

Se *L* é a linguagem gerada de um sistema e *Lm* sua linguagem marcada, observa-se que, para gerar qualquer palavra da linguagem marcada, o SED em questão tem que gerar todos os seus prefixos. Em outras palavras, um SED que produza as palavras contidas numa linguagem *Lm,* também produzirá as palavras em *L*. Então,

$$L_m \subseteq \overline{L_m} \subseteq L = \overline{L}$$

As observações acima podem ser sintetizadas formalmente nas seguintes propriedades de linguagens *L* e *Lm* que representam um SED:

1. *L Ê Lm*, ou seja, o comportamento gerado contém o comportamento marcado de um SED;
2. $L = \overline{L}$, ou seja, o comportamento gerado de um SED é prefixo fechado.

A descrição de uma linguagem feita pela enumeração das cadeias que a definem, pode ser uma tarefa com pouca aplicabilidade. É conveniente dispor de uma forma de representação de linguagens que seja simples, concisa, clara e sem ambigüidade. É necessário

então utilizar estruturas compactas que possam representar estas linguagens. No presente trabalho serão apresentadas duas estruturas: as expressões regulares e os autômatos.

As expressões regulares são seqüências de símbolos obtidas pela aplicação repetitiva de um conjunto de regras de formação. Para um alfabeto *S* dado, define-se recursivamente uma expressão regular da seguinte forma:

1. ∅ é uma expressão regular que representa a linguagem vazia;
2. ε é uma expressão regular denotando a linguagem formada pela palavra vazia { ε};
3. u é uma expressão regular representando a linguagem {u} ⊂ Σ*, para todo u ⊂ Σ;

Se r e s são expressões regulares, então rs, (r + s), r* e s* são expressões regulares; toda expressão regular é obtida pela aplicação das propriedades 1 e 2 um número finito de vezes.

A classe de linguagens expressa pelas chamadas expressões regulares são denominadas linguagens regulares. Por sua vez, tal classe pode ser representada por autômatos finitos determinísticos. Expressões regulares e autômatos finitos têm o mesmo poder de representação e modelam as linguagens regulares com a mesma expressividade. Autômatos são modelos matemáticos que reconhecem um conjunto de cadeias sobre um dado alfabeto.

Um autômato finito determinístico é uma quíntupla *A = (S, Q, d, qo, Qm)*, onde *S* é um alfabeto, *Q* é um conjunto finito não vazio de estados, *d : Q X S ® Q* é uma função de transição de estados, *qo Î Q* é o estado inicial, *Qm Í Q* é o conjunto de estados marcados. Formalmente, a função de transição é definida sobre todo o domínio de *Q X S*, mas considera-se que a função de transição é possivelmente parcial, ou seja, não há a necessidade da função ser definida para todo elemento de *S* em cada estado de *Q* (Cassandras e Lafortune, 1999).

Os autômatos podem ser ilustrados por diagramas de transição de estados, que são grafos direcionados onde os nós representam os estados e os ramos representam os eventos. Nesses diagramas, os estados marcados são caracterizados por nós desenhados com linhas duplas e o estado inicial é identificado por uma seta. A figura 2.1 mostra um exemplo de autômato.

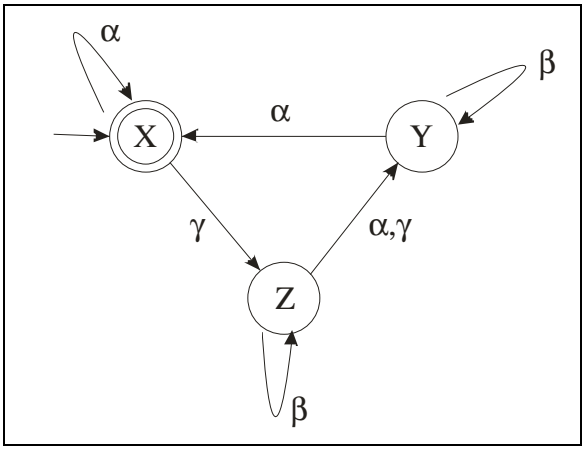

Figura 2.1 - Representação em Diagrama de Estados

$Q = \{X, Y, Z\}; \Sigma = \{a, b, g\}$

$d(X, a) = X, d(X, g) = Z, d(Y, a) = X, d(Y, b) = Y, d(Z, b) = Z$ e

$d(Z, a) = d(Z, g) = Y; x_0 = \{X\}; Q_m = \{X\}$

Dado um autômato *G*, define-se o conjunto ativo de eventos num estado $q \in Q$ como $G(q) = \{s \in S : d(q, s) \text{ é definida}\}$. O autômato *G* pode ser visto como um dispositivo que opera como segue. Inicia a partir do estado inicial q_o e lá permanece até a ocorrência de um evento $s \in G(q_o) \cap S$ que disparará a transição $d(q_o, s) \in Q$. Este processo continua baseado nas transições definidas em *d*.

A função de transição *d* pode ser naturalmente estendida para cadeias de eventos. Assim, a função de transição estendida denotada por $\hat{d}$, é uma função $Q \times S^* \rightarrow Q$ tal que, $\hat{d}(q, e) = q$, $\hat{d}(q, s) = d(q, s)$ e $\hat{d}(q, ss) = d(\hat{d}(q, s), s)$ para $s \in S^*$ e $s \in S$.

A um autômato *G* estão associadas a duas linguagens, a linguagem gerada *L(G)* e a linguagem marcada *Lm(G)*. A linguagem gerada por *G =(Q, S, d, qo, Qm)* é:

1. $L(G) := \{s \in \Sigma^* : \delta(qo, s) \text{ é definida}\}$.
2. A linguagem marcada de G é:
3. $Lm(G) := \{s \in \Sigma^* : \delta(qo, s) \in Qm\}$.

Em palavras, a linguagem *L(G)* representa todas as cadeias que podem ser seguidas no autômato, partindo do estado inicial. A linguagem *Lm(G)* considera todas as cadeias que

partindo do estado inicial chegam a um estado marcado. Desta forma, um SED pode ser modelado por um autômato *G*, onde *L(G)* é o comportamento gerado pelo sistema e *Lm(G)* é o comportamento marcado ou conjunto de tarefas completas do sistema.

É possível definir um autômato com estados inacessíveis, isto é, estados que jamais podem ser alcançados a partir do estado inicial. Tais estados formam 'ilhas' que não são ligadas ao estado inicial por qualquer caminho. Formalmente, um estado $q \in Q$ é acessível se $q = d(q_o, u)$ para algum $u \in S^*$. Um autômato *G* é dito ser acessível se *q* é acessível para todo $q \in Q$. A componente acessível, *Gac*, de um autômato *G* é obtida pela eliminação de seus estados não acessíveis e das transições associadas a eles.

Por outro lado *G* é dito ser coacessível, ou não bloqueante, se cada cadeia $u \in L(G)$ pode ser completada por algum $w \in S^*$ tal que $uw \in Lm(G)$, ou seja, se cada cadeia $u \in L(G)$ for um prefixo de uma cadeia em *Lm(G)*. Esta definição diz que um autômato é co-acessível se, a partir de qualquer um dos seus estados, existir ao menos um caminho que leve a um estado marcado. A condição de co-acessibilidade de um autômato pode ainda ser descrita pela equação:

$$L(G) = \overline{L_m(G)} \qquad 2.1$$

A equação acima permite definir a idéia de ausência de bloqueio num sistema a eventos discretos.

Um SED com comportamento *L(G)* e *Lm(G)* é dito ser não bloqueante se, e somente se, satisfaz as condições da equação 2.1, isto é, $L(G) = \overline{L_m(G)}$. A condição de bloqueio ($L(G) \neq \overline{L_m(G)}$) corresponde à existência de cadeia(s) geradas pelo sistema ($u \in L(G)$), a partir da(s) qual(is) não se pode completar alguma tarefa no sistema ($u \notin \overline{L_m(G)}$).

Os autômatos da figura 2.2 e figura 2.3 modelam um autômato não-bloqueante e um bloqueante, respectivamente. No autômato *A* a condição de SED não-bloqueante é satisfeita ($L(A) = \overline{L_m(A)}$), ainda que a função de transição de estados não esteja definida para qualquer evento no estado H. De fato, o estado H corresponde a uma tarefa completa do sistema, o que caracteriza o não bloqueio. No autômato *B* um estado de bloqueio pode ser identificado (estado Z), a partir do qual não se pode completar nenhuma tarefa no sistema. Observe ainda que, a partir deste estado existe um evento possível de ocorrer.

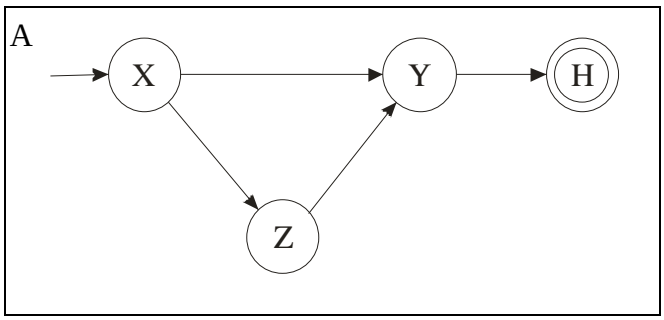

Figura 2.2 - Exemplo Autômato não Bloqueante

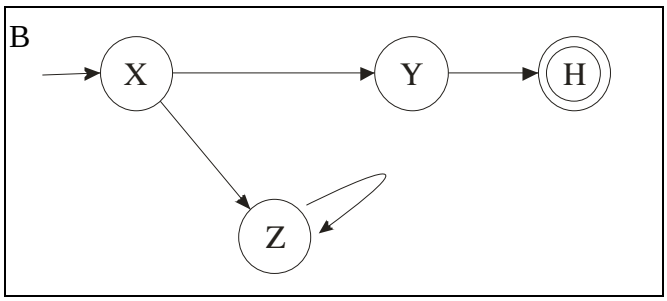

Figura 2.3 - Exemplo Autômato Bloqueante

De acordo com Cury (2001), a modelagem de SED por autômatos pode ser abordada de duas formas: uma abordagem global e uma abordagem local. Na abordagem global o sistema é analisado como um todo e procura-se um autômato que represente todas as possíveis seqüências de eventos que ele pode gerar e tarefas que pode completar. Para sistemas de maior porte, esta pode ser uma tarefa de grande complexidade. Por outro lado, muitos sistemas de interesse prático apresentam características modulares e/ou distribuídas, de modo que podem ser vistos como constituídos de diversos subsistemas concorrentes, cada qual gerando certos eventos. O comportamento do sistema como um todo é determinado então pelos eventos produzidos nesses subsistemas.

A abordagem local sugere maior facilidade na obtenção de modelos de sistemas de grande porte. Além disso, permite pressupor que alterações num subsistema somente exigirão uma mudança no modelo específico correspondente. A aplicabilidade da abordagem local para a modelagem de SEDs por autômatos é garantida pela operação de composição de autômatos, como definida a seguir.

Sejam dois autômatos *G1 = (Q1, S1, d1, qo1, Qm1)* e *G2 = (Q2, S2, d2, qo2, Qm2).* A composição síncrona de *G1* e *G2*, denotada por *G1 || G2* é definida como:

G1 || G2 = (Q1 X Q2, S1 È S2, d1|d|2, (qo1, qo2), Qm1 X Qm2)

sendo:

d1||2 : (Q1 X Q2) X (S1 È S2) ® (Q1 X Q2)

ou seja,

d1||2 ((q1, q2), s) = (d1(q1, s), d2(q2, s)) se s Î S1 Ç S2 e s Î S1(q1) È S2(q2) = (d1(q1, s), q2) se s Î S1 e s Ï S2 e s Î S1(q1)

= (q1, d2(q2, s)) se s Î S2 e s Ï S1e s Î S2(q2)

= indefinida, caso contrário.

Um evento comum a *S1* e *S2* só pode ser executado sincronamente nos dois autômatos; os demais ocorrem assincronamente, ou seja, de modo independente em cada autômato. Pode-se interpretar *G1 || G2* como a ação cooperativa entre os dois autômatos, onde os eventos comuns aos respectivos alfabetos são sincronizados. Se os alfabetos são iguais entre si *S1 = S2*, a composição é completamente síncrona, isto é, todos os eventos estão sincronizados. No caso oposto, *S1 Ç S2 = Æ*, não existe nenhuma sincronização entre os eventos dos dois autômatos. A próxima figura mostra a operação de composição de dois autômatos. O conjunto de eventos comuns é $\{a_1\}$.

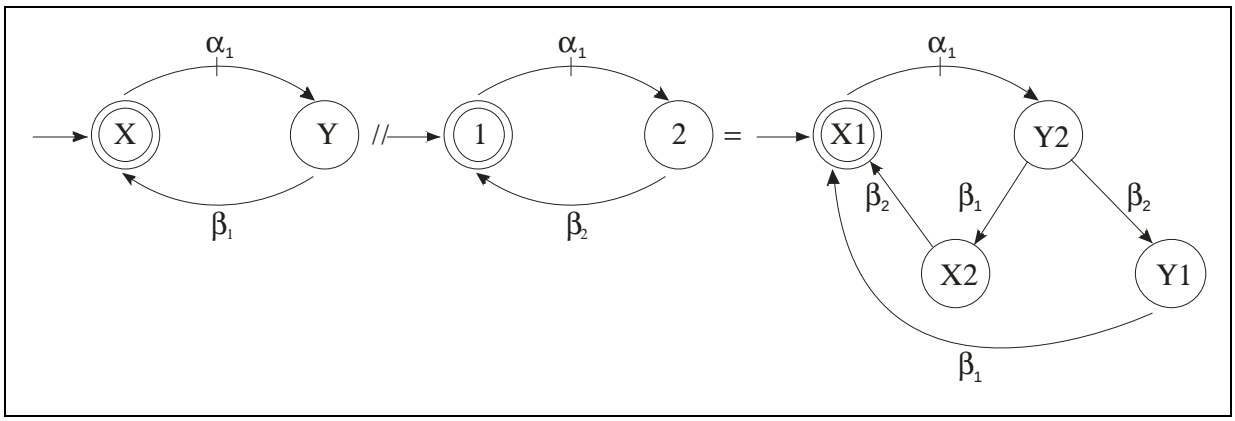

Figura 2.4 - Exemplo Composição de Dois Autômatos Passo 3

2.3 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) (Ramadge e Wonham, 1989) foi desenvolvida com o objetivo de prover uma metodologia formal para a síntese automática de controladores para Sistemas a Eventos Discretos (SED). Esta teoria faz uma distinção clara

entre o sistema a ser controlado, denominado planta, e a entidade que o controla, que recebe o nome de supervisor. Considerando que o sistema a ser controlado é constituído por diversos subsistemas,

o modelo da planta reflete o comportamento fisicamente possível dos mesmos, isto é, todas as ações que estes são capazes de executar na ausência de qualquer ação de controle. O papel do supervisor na TCS é, então, o de exercer uma ação de controle restritiva sobre os subsistemas, de modo a confinar seus comportamentos àqueles que correspondem a um conjunto de especificações.

Na abordagem proposta por Ramadge e Wonham (1989) (abordagem RW), o SED a ser controlado, ou planta na terminologia de controle tradicional, é representado por uma linguagem gerada *L* e por uma linguagem marcada *Lm*. Conforme discutido na seção anterior, assume-se aqui que a planta *G* é modelada por um autômato. A notação *G* será então usada indistintamente para referenciar a planta ou o seu modelo em autômato.

Dessa forma, as linguagens *L(G)* e *Lm(G)* podem conter cadeias indesejáveis de eventos por violarem alguma condição que se deseja impor ao sistema. Pela junção de uma estrutura de controle (supervisor), será possível modificar a linguagem gerada pelo sistema dentro de certos limites, evitando aquelas cadeias indesejadas de eventos. A característica de controle é introduzida ao se considerar que certos eventos podem ser desabilitados por um controlador externo, podendo influenciar na evolução do SED pela proibição da ocorrência de eventos chaves em certos momentos.

O autômato *G* modela então o comportamento não controlado do SED, ou o comportamento em malha aberta analogamente à teoria de controle clássica. A premissa é que este comportamento não é satisfatório e dever ser modificado por uma ação de controle. A modificação deste comportamento deve ser entendida como uma restrição do comportamento a um subconjunto de *L(G)*. Para alterar o comportamento introduz-se um supervisor, denotado por *S*.

A idéia central é construir um supervisor tal que os eventos que ele desabilita num dado instante dependem do comportamento passado do SED, esta abordagem denominada Controle Supervisório Monolítico, tem como objetivo projetar um único controlador cuja função é habilitar e desabilitar certos eventos, conforme a seqüência de eventos observados na planta.

Dentro desta abordagem, considera-se que o supervisor *S* interage com a planta *G*, numa estrutura em malha fechada, onde *S* observa os eventos ocorridos em *G* e define que eventos, dentre os fisicamente possíveis de ocorrerem no estado atual, são permitidos de ocorrerem a seguir. Sob este aspecto, a forma de controle é dita permissiva, no sentido que eventos inibidos não podem ocorrer e os autorizados não ocorrem obrigatoriamente. O conjunto de eventos habilitados num dado instante pelo supervisor define uma entrada de controle. Esta é atualizada a cada nova ocorrência de evento observada em *G*.

Para associar estruturas de controle a um SED ou a uma planta *G*, particiona-se o alfabeto *S* em um conjunto S_c de eventos controláveis que podem ser inibidos de ocorrer, e um conjunto S_u de eventos não-controláveis, sobre os quais o agente de controle não tem influência. Para que seja possível interferir no funcionamento da planta *G*, este precisa ser dotado de uma interface através da qual se possa informar quais eventos devem ser habilitados e quais devem ser inibidos. Considera-se o conjunto de eventos que se deseja habilitar como uma entrada de controle. Naturalmente, esta entrada de controle não deve tentar inibir eventos não-controláveis. Formalmente define-se uma estrutura de controle associada a *G* como o conjunto de entradas de controle:

$$\Gamma = \left\{ g \in 2^{\Sigma} : g \supseteq \Sigma_u \right\}$$

sendo que a condição $g \supseteq \Sigma_u$ indica simplesmente que os eventos não-controláveis são necessariamente habilitados.

Quando se aplica uma entrada de controle g a uma planta, esta se comporta como se os eventos inibidos fossem momentaneamente apagados da sua estrutura de transição, afetando com isso a linguagem gerada. É este o princípio de funcionamento do mecanismo de controle adotado no modelo RW, que consiste em chavear as entradas de controle em resposta ao comportamento observado do sistema, de modo a confinar a linguagem gerada. Considera-se então que a função de transição de um autômato cujo alfabeto foi particionado em eventos controláveis e eventos não-controláveis deixa de ser definida para os eventos inibidos por uma entrada de controle aplicada, enquanto esta estiver presente, sem explicitar este fato na notação.

O agente controlador é denominado supervisor. Formalmente, um supervisor *S* é um mapeamento $S: L \rightarrow G$ que especifica, para cada cadeia possível de eventos gerados $w \in L$, um conjunto de eventos habilitados (entrada de controle) $g = S(w) \in \Gamma$. O SED *G* controlado

por *S* é denotado por *S/G*. O comportamento do sistema sob a ação do supervisor, definido pela linguagem $L(S/G) \subseteq L(G)$ satisfaz:

$i) e \in L(S/G)$

$ii) ws \in L(S/G)\ sse\ w \in L(S/G), ws \in L(G) e\ s \in S(w).$

Diz-se que um supervisor *S* para a planta *G* é não-bloqueante se, e somente se, $\overline{L_m(S/G)} = L(S/G)$. Isso implica que, de qualquer estado do comportamento em malha fechada da planta, uma tarefa pode ser completada no sistema.

O funcionamento do sistema controlado *S/G* pode ser descrito por um SED resultante da composição síncrona de *S* e *G*, isto é, *S* || *G*. De fato, na composição síncrona *S* || *G* somente as transições permitidas tanto no sistema controlado *G*, como no supervisor *S* são permitidas.

O comportamento em malha fechada do sistema é então dado por:

$L(S/G) = L(S \| G)$ e $L_m(S/G) = L_m(S \| G)$.

De um modo geral, um problema de síntese de supervisores supõe que se represente o comportamento fisicamente possível do sistema e o comportamento desejado sob supervisão por linguagens, sendo o objetivo construir um supervisor para a planta de forma que o comportamento do sistema em malha fechada se limite ao comportamento desejado, segundo especificações, representados também em autômatos.

As especificações são interpretadas da seguinte forma. A linguagem gerada *L(G)* contém palavras indesejáveis, pois violam alguma condição que se deseja impor ao sistema. Uma especificação pode ser definida em termos de estados proibidos: certos estados de *G* que são indesejados e devem ser evitados, sendo estes estados causadores de bloqueio ou então fisicamente inadmissíveis (por exemplo, a colisão de um robô com um veículo auto guiado ou a tentativa de colocar uma peça em um armazém cheio) (Cury, 2001). Uma especificação também pode ser definida em termos de prefixos que não são permitidos em *L(G)*, pois violam uma seqüência desejada de certos eventos.

A utilização da Teoria de Controle Supervisório proposta por Ramadge e Wonham (1989) quer determinar sob que condições existe um supervisor que realize uma determinada especificação e, caso exista, como encontrá-lo. Dada uma planta *G* definida sobre o alfabeto $\Sigma = \Sigma_u \cup \Sigma_c$ e uma sublinguagem *K* de *L(G)*, a linguagem *K* é dita ***controlável*** em relação a *L(G)* se $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$. Esta definição exige que qualquer prefixo *w* de uma palavra de *K*

($w \in \overline{K}$), quando seguido de um evento não controlável $s \in \Sigma_u$, tal que $ws \in L(G)$, deve ser ainda prefixo de uma palavra de K ($ws \in \overline{K}$). A classe de linguagens controláveis contidas numa linguagem K, denotada por $C(K) = \{E \subseteq K : \overline{E}\Sigma_u \cap L(G) \subseteq \overline{E}\}$, é não-vazia e fechada para a operação de união de conjuntos. Sendo assim, $C(K)$ contém um único elemento supremo, chamado de $\sup C(K, L(G))$.

No caso em que a linguagem K que especifica o comportamento desejado não é controlável, é possível projetar uma aproximação de K que é a máxima linguagem controlável contida em K (denotado por $\sup C(K, L(G))$). O supervisor que implementa esta linguagem é chamado supervisor minimamente restritivo ou ótimo (Ramadge e Wonham, 1989; Kumar e Garg, 1995; Wonham, 1998).

Considerando o exposto na presente seção, pode-se definir as etapas de desenvolvimento desta abordagem:

- Identificar o conjunto de subsistemas envolvido no caso estudado;
- Construir um modelo em autômato G que representa o comportamento do sistema (abordagem global) ou construir um modelo G_i de cada subsistema envolvido (abordagem local), onde $i\{1,...,n\}$ e n é o número de subsistemas envolvidos;
- Identificar os eventos controláveis e não-controláveis;
- Modelar cada especificação isoladamente, considerando apenas os eventos relevantes;
- Obter o modelo da planta fazendo-se a composição síncrona dos autômatos G_i (no caso da abordagem local);
- Obter o modelo da especificação E fazendo-se a composição dos autômatos que representam as especificações;
- Calcular a linguagem da planta que satisfaz a especificação, através da composição do autômato G com o autômato E;
- Calcular a máxima linguagem controlável contida na especificação.

O exemplo a seguir, extraído de Wonham (1999), ilustra uma aplicação da TCS na supervisão de uma célula de manufatura. A célula em questão, ilustrada na figura 2.5, é formada por duas máquinas e um ***buffer*** intermediário de capacidade unitária.

$\sum_{M1} = \{a_1, b_1\}$

$\Sigma_{M_2} = \{a_2, b_2\}$

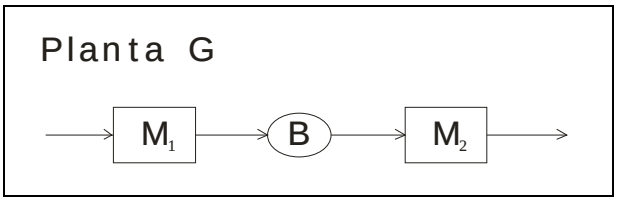

Figura 2.5 - Exemplo “Pequena Fábrica” Wonham (1999)

Os modelos das máquinas são representados pelos autômatos mostrados na figura 2.6. Ambos representam dois estados - ativo Y_n e repouso X_n - e têm na sua estrutura de transição os eventos iniciar operação (a_n) e operação finalizada (b_n), $n = \{1,2\}$. A figura 2.7 apresenta o autômato resultante da composição dos dois autômatos que representam as máquinas, sendo esta a representação do comportamento global do sistema. Esse modelo é considerado a planta.

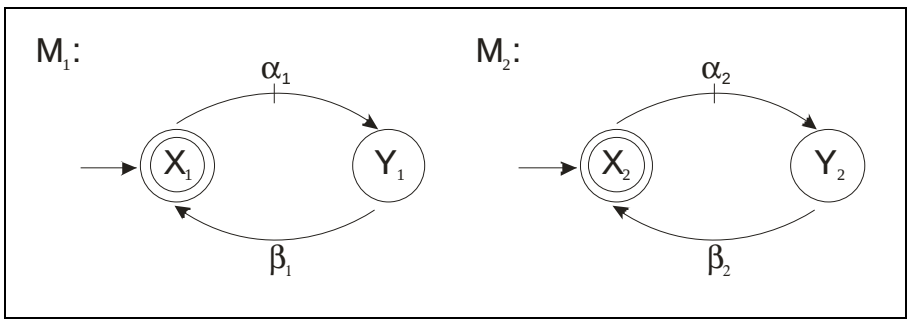

Figura 2.6 - Representação em Autômatos do comportamento das máquinas

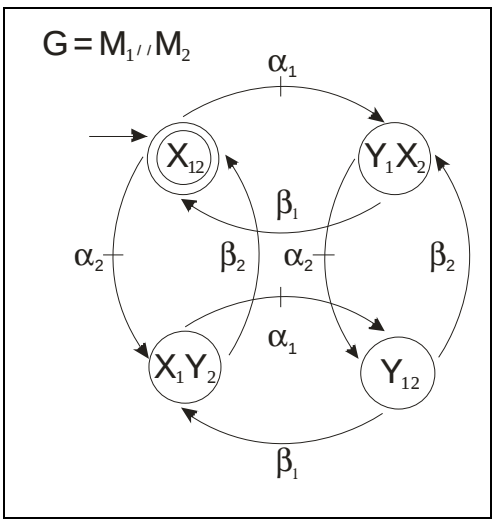

Figura 2.7 – Representação global do comportamento das máquinas

Para fim de controle, é necessário que se evite o *overflow* e o *underflow* do buffer intermediário. Nesse caso, deseja-se que M_1 coloque peça no *buffer* somente se o mesmo estiver vazio e que M_2 só retire peça do mesmo somente se estiver cheio. Esta especificação pode ser representada pelo autômato E_1 mostrado na figura 2.8.

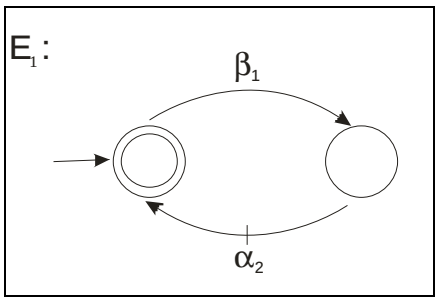

Figura 2.8 - Especificação Evitando overflow e underflow

Compondo-se a planta com a especificação desejada, tem-se o supervisor R mostrado na figura 2.9, figura 2.9 sendo este o primeiro candidato a supervisor ótimo.

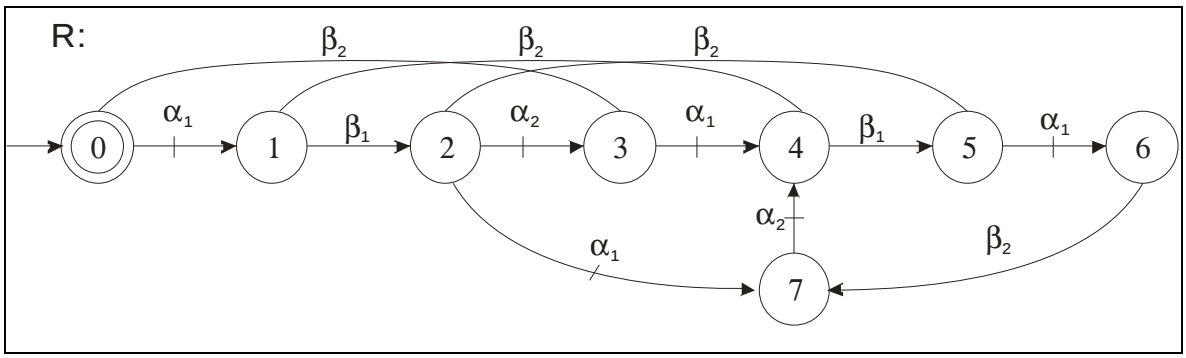

Figura 2.9 – Candidato a supervisor R

Em seguida, verifica-se a propriedade de controlabilidade do candidato a supervisor R, utilizando-se o algoritmo proposto por Ramadge e Wonham (1989).

Em linhas gerais, esse algoritmo identifica os “maus estados” do modelo, que são aqueles em que eventos não controláveis estão sendo desabilitados.

Analisando-se o modelo apresentado na figura 2.9, verifica-se que os estados 6 e 7 são “maus estados”, pois ambos estão desabilitando o evento b_1. Excluindo-se estes dois estados, obtém-se a máxima linguagem controlável ou o supervisor ótimo, apresentado na figura 2.10.

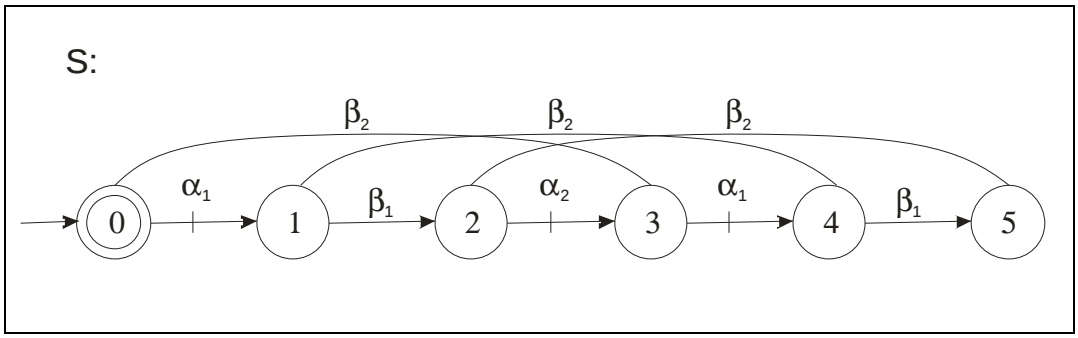

Figura 2.10 – Supervisor da Planta

2.4 Controle Supervisório Modular Local

A grande vantagem da TCS é que esta permite a síntese automática de supervisores. Também, a noção de máxima linguagem controlável garante a síntese de controladores de forma minimamente restritiva. No entanto, quando um grande número de tarefas necessita ser executada pelo sistema de controle em plantas maiores, a abordagem monolítica pode ter um desempenho computacional bastante desfavorável. Isso porque a composição síncrona das especificações gera um crescimento exponencial no número de estados do modelo e, por conseguinte, na complexidade computacional do problema.

Queiroz e Cury (2000a,b) propõem uma solução alternativa para a síntese de controle monolítico que explora a modularidade das especificações e a própria modularidade da planta. A abordagem proposta pelos autores é denominada Controle Modular Local (CML).

Segundo esta abordagem, o sistema físico deve ser decomposto em diversos subsistemas assíncronos, constituindo assim uma representação por sistema produto. Cada um destes subsistemas deve ser representado por um autômato G_i de forma a representar seu comportamento da forma mais abstrata possível. Cada um destes autômatos terá a estrutura $G_i = (S_i, Q_i, d_i, q_{0i}, Q_{mi})$, $i \hat{I} I = \{1..p\}$, sendo que p é o nº de subsistemas físicos, S_i é o sub-alfabeto de eventos exclusivo ao subsistema em questão, Q_i é o conjunto de estados, d_i é a função de transição de estados na forma $d_i : S_i \acute{} Q_i \ ® \ Q_i$, $q_{0i} \hat{I} \ Q_i$ é o estado inicial e $Q_{mi} \acute{I} \ Q_i$ é o conjunto de estados marcados. O sub-alfabeto S_i é particionado em eventos controláveis $Sc_i \acute{I} \ S_i$ e eventos não-controláveis $\Sigma u_i \subseteq \Sigma_i$. O alfabeto do sistema físico é obtido pela união dos diversos sub-alfabetos.

Na abordagem modular local o diferencial do tratamento das especificações está na obtenção da especificação local, sendo esta resultado do sincronismo de uma especificação com os subsistemas. Cada uma das *m* especificações genéricas, representada por um autômato E_i e definida em $\Sigma_i \subseteq \Sigma$ para $i \in I = \{1..m\}$, restringe o comportamento de apenas uma porção do sistema físico. A composição dos subsistemas cujo comportamento é restringido por uma determinada especificação determina uma planta local, e a composição de uma especificação genérica com a respectiva planta local define a especificação local. Com este procedimento é possível realizar a síntese de um supervisor local para cada uma das especificações estabelecidas. Cada um destes supervisores locais poderá ser representado através de um autômato SL_i cuja linguagem marcada corresponde à máxima linguagem controlável da planta local contida na linguagem da especificação local.

Conforme demonstrado em Queiroz e Cury (2000a,b), caso seja verificada a modularidade local do conjunto de supervisores locais, é assegurado que a ação conjunta de todos os supervisores é não bloqueante.

De acordo com Queiroz (2000), a sistematização da aplicação do CML é implementada nas seguintes etapas:

1. Identificar o conjunto de subsistemas envolvidos no caso tratado;
2. Construir o modelo básico ADEF (Autômato Determinístico de Estados Finitos) G_i, de cada subsistema *i* envolvido de forma mais sintética possível;
3. Calcular a mais refinada Representação por Sistema Produto (RSP), fazendo-se a composição dos subsistemas síncronos.
4. Modelar cada especificação isoladamente, considerando apenas os eventos relevantes;
5. Obter a planta local para cada especificação compondo-se os subsistemas da RSP que tenham eventos em comum com ela;
6. Calcular a linguagem de cada planta local que satisfaça a especificação, através do produto síncrono da cada planta local com sua respectiva especificação;
7. Calcular a máxima linguagem controlável contida em cada especificação local;
8. Verificar a modularidade local das linguagens resultantes;
9. Se não forem modulares, procurar resolver o problema de não modularidade por outra abordagem;

10. Se forem modulares, implementar um supervisor local para cada linguagem controlável.

A verificação da modularidade pode ser realizada através da composição síncrona dos autômatos correspondentes às máximas linguagens controláveis obtidas na etapa 7. Caso o autômato obtido resultante dessa composição seja Trim, pode-se afirmar que as linguagens resultantes são modulares entre si.

O exemplo a seguir, extraído de Vieira *et al.* (2004) e apresentado na figura 2.11, ilustra a aplicação do CML. Neste exemplo, uma célula de manufatura é composta de duas máquinas (M1 e M2), um manipulador robótico (M3) e dois *buffers*, um entre as máquinas 1 e 3 e outro entre as máquinas 3 e 2. Os modelos destas máquinas são vistos na figura 2.12: M1 no estado 0 significa máquina em repouso e no estado 1, trabalhando; o mesmo ocorre para M2; M3 no estado 0 significa manipulador robótico em repouso, no estado 1, levando peça de B1 para B2 e no estado 2, retornando a posição de repouso.

O objetivo é usar ao máximo os recursos disponíveis. Porém, o processamento de várias peças no sistema não pode ocasionar *underflow* ou *overflow* nos *buffers* (sobreposição de duas ou mais peças, ou as máquinas tentarem retirar peças dos *buffers* sem que haja alguma neles). A figura 2.13 apresenta os modelos das duas especificações que representam o comportamento desejado a planta em questão.

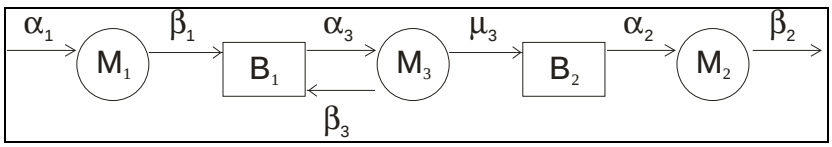

Figura 2.11 - Exemplo Célula de Manufatura

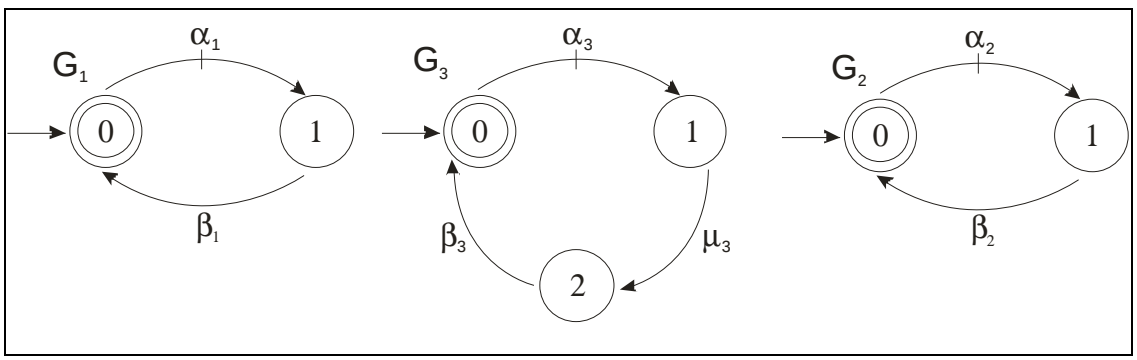

Figura 2.12 - Representação em Autômatos Célula de Manufatura

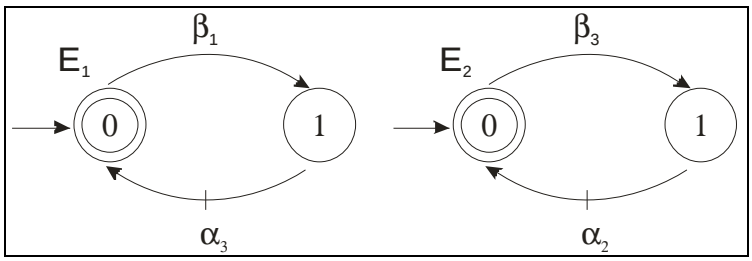

Figura 2.13 – Especificações Evitando overflow e underflow

De acordo com a aplicação do controle modular local, as plantas locais são obtidas através da composição dos subsistemas que compartilham eventos com suas respectivas especificações *E1* e *E2*. Os seguintes autômatos são então obtidos: $G_{loc,1} = G1 \,||\, G3$, $G_{loc,2} = G2 \,||\, G3$. As especificações locais são obtidas através da composição síncrona das especificações genéricas *E1 e E2* com suas respectivas plantas locais: $E_{loc,\,1} = E1 \,||\, Lm\,(G_{loc,1})$, $E_{loc,\,2} = E2 \,||\, Lm\,(G_{loc,2})$.

É possível então calcular as máximas linguagens controláveis contidas em $E_{loc,j}$ $(j=1,2)$, isto é, $SupC\,(E_{loc,j}, G_{loc,\,j})$. Por meio de uma ferramenta computacional adequada (por exemplo, o programa TCT) (WONHAM, 1999), pode-se verificar que as especificações controláveis são localmente modulares, isto é, pode-se calcular e verificar a igualdade de $\left\| \overline{SupC(E_{loc,\,j}, G_{loc,\,j})} \right.$ e $\left\| \overline{SupC\left(E_{loc,\,j}, G_{loc,\,j}\right)} \right.$.

2.5 Implementação da Estrutura de Controle Supervisório

Em relação à implementação de uma estrutura de controle baseada na TCS, muitas abordagens são encontradas na literatura (Balemi *et al.*, 1993; Brandin, 1996; Fabian e Hellgren, 1998; Hellgren *et al.*, 2002; Leduc, 1996; Queiroz e Cury, 2002; Liu e Darabi, 2002; Hasdemir *et al.*, 2004; Vieira e Cury, 2004; Vieira *et al.*, 2004) sem, no entanto, existir relatos de aplicações no meio industrial. A não difusão da TCS no meio industrial relaciona-se com algumas dificuldades na sua implementação física. Em linhas gerais, existe a necessidade de uma sistemática que traduza o modelo teórico da TCS numa linguagem própria de controladores industriais. O maior desafio nesse sentido é fazer com que o código implementado em tais controladores execute a mesma ação de controle imposta pela TCS.

Algumas dificuldades de implementação surgem em função de algumas hipóteses seguidas pelas TCS. Por exemplo, assume-se que os eventos ocorrem espontaneamente na planta, sendo que a única forma do supervisor afetar o seu comportamento é através da habilitação ou desabilitação de eventos controláveis. Também, a TCS provém o supervisor minimamente restritivo, de forma a manter o comportamento da planta confinado à especificação. Como conseqüência, é possível que o supervisor permita a ocorrência de mais de um evento controlável em um estado. No caso geral, tais hipóteses não são compatíveis com os sistemas industriais, uma vez que se espera do sistema de controle forçar a ocorrência de certos eventos, e não somente suas habilitações ou desabilitações.

Além disto, quando o comportamento livre da planta é descrito através de um conjunto de autômatos os detalhes irrelevantes para a coordenação da operação conjunta devem ser abstraídos do modelo, evitando a explosão do número de estados do supervisor. Estas questões caracterizam de forma bastante sucinta a origem dos problemas inerentes à implementação do controle de SED quando abordados segundo a TCS. Tais problemas são identificados e discutidos detalhadamente em (Dietrich *et al.*, 2002), (Fabian e Hellgren, 1998) e (Malik, 2002).

Fabian e Hellgren (1998) apresentam um procedimento para representar um supervisor em uma linguagem de programação de CLP. Nesta abordagem de implementação o papel do supervisor é o de gerar os eventos controláveis, ao invés de definir a desabilitação dos mesmos. Os eventos não-controláveis são gerados espontaneamente pelo sistema a ser controlado em decorrência das atividades executadas pelo mesmo. Em Fabian e Hellgren (1998) não é discutido o tratamento para a questão da representação abstrata do comportamento livre da planta.

Queiroz *et al.* (2001) e Queiroz e Cury (2002) propõem implementar a estrutura de controle modular local (baseada no CML) numa estrutura dividida em três níveis: Supervisores Modulares, Sistema Produto e Seqüências Operacionais, de acordo com a figura 2.14. No nível denominado “Supervisores Modulares” (SM) é implementado o conjunto de autômatos que representam os supervisores sintetizados. O conjunto de estados ativos destes autômatos é determinado pela seqüência de eventos gerada no sistema produto. Em função deste conjunto de estados são estabelecidas desabilitações de eventos controláveis.

Como a TCS prevê a geração espontânea de eventos pelo sistema físico, os níveis denominados “Sistema Produto” (SP) e “Seqüências Operacionais” (SO) atuam como uma

interface entre o modelo previsto pela TCS e o sistema físico. Nesta estrutura de controle há uma Seqüência Operacional (SO) diretamente relacionada a cada módulo do sistema produto. No nível SO, estas observam os sinais de entrada do CLP e ajustam os sinais de saída, de acordo com os ciclos de funcionamento de cada dispositivo. Em Queiroz *et al.* (2001) e Queiroz e Cury (2002), a estrutura de controle é implementada em um único Controlador Lógico Programável (CLP).

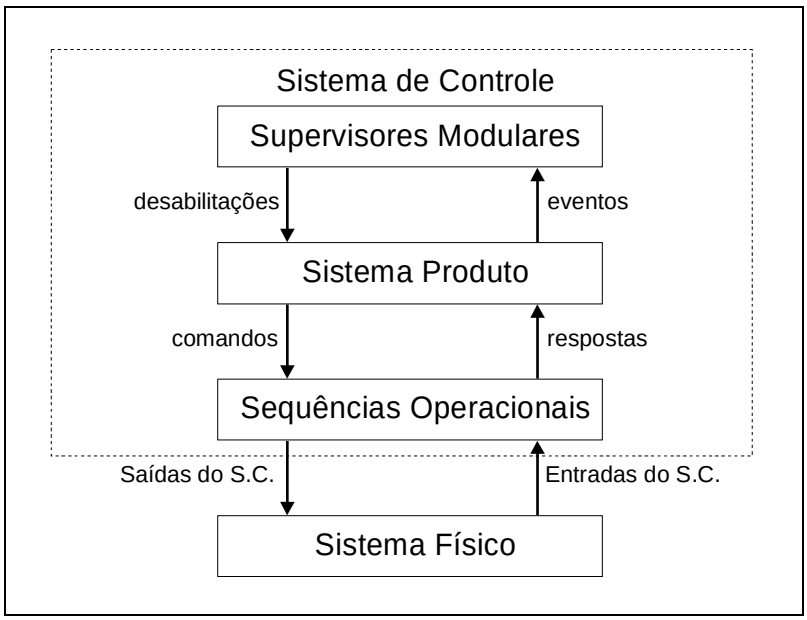

Figura 2.14 - Modelo de Implementação

Outra importante questão relacionada à implementação da TCS é a distribuição da estrutura de controle. De acordo com Vieira e Cury (2004), a distribuição é fundamental quando se consideram fatores como a limitação das plataformas utilizadas (número de entradas e saídas, recursos internos) e a modularidade do sistema físico (geralmente formado por células de trabalhos com respectivos controladores locais). Importante também citar que a distribuição do sistema físico de controle permite uma melhor organização do comissionamento da planta, alterações futuras na instalação, manutenção, ou mesmo, a operação parcial do sistema em caso de falha de um determinado módulo.

A distribuição vertical proposta por Vieira e Cury (2004) e Vieira *et al.* (2004) consiste fundamentalmente em implementar as seqüências operacionais em CLP distintos. Considera-se, desta forma, a existência de diversos CLP interligados através de um canal bidirecional (redes de comunicação ou mesmo entradas e saídas digitais). Em um desses CLP

são implementados integralmente os níveis SM e SP. É possível ainda a implementação de algum conjunto de seqüências operacionais neste mesmo CLP.

Para alcançar o correto funcionamento da estrutura distribuída, deverá ser estabelecida a comunicação entre os diversos CLP's e considerados aspectos como a falta de sincronismo e a possibilidade de diferentes tempos de ciclo de varredura dos CLP (Vieira e Cury, 2004). De forma a sanar estes problemas é proposto que seja implementada uma interface no CLP correspondente ao nó raiz e nos CLP correspondentes aos nós folhas, ilustradas nas figura 2.15 e figura 2.16.

Para a construção destas interfaces devem ser definidas funções que relacionam um comando ou uma resposta a, respectivamente, um comando externo ou uma resposta externa. Para exemplificar estas sinalizações, considera-se uma estrutura formada por um número n de CLP, $i \in I = \{1,...,n\}$. A ocorrência de um evento controlável a_i resultará na ativação de um comando $cmda_i$ no nível SP, que por sua vez ativará um comando externo $cmdexta_i$ na interface raiz. A interface folha receberá esse comando externo e irá gerar o comando $cmda_i$ responsável pelo início da *SO* correspondente. Se o evento for parte da linguagem de um dos SM, este obrigatoriamente deve ser atualizado antes da continuidade do programa.

Quando ocorrer uma resposta $rplb_i$ no nível *SO*, esta ativará uma resposta externa $rplextb_i$ na interface folha. A interface raiz receberá essa resposta externa e ativará a resposta $rplb_i$ no nível SP, esta por sua vez, resulta na ativação do evento incontrolável b_i. Se este evento for parte da linguagem de um dos SM, este deve ser atualizado antes da continuidade do programa.

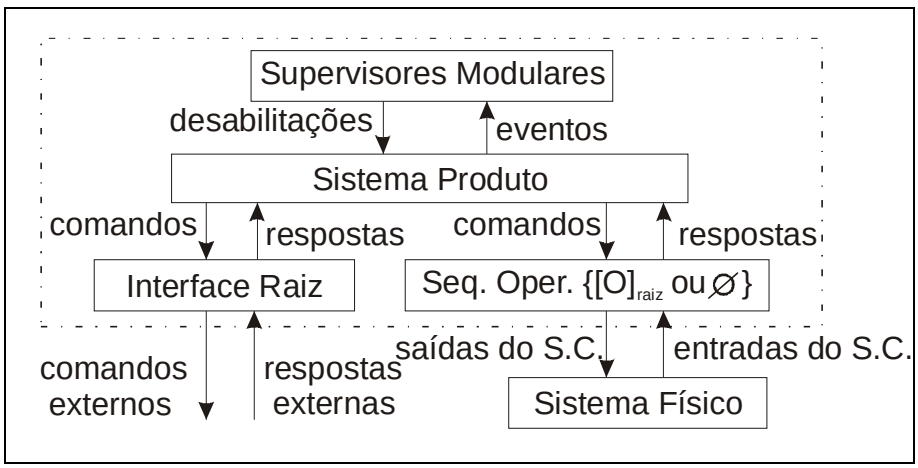

Figura 2.15 - Interface de Comunicação Sistema de Controle

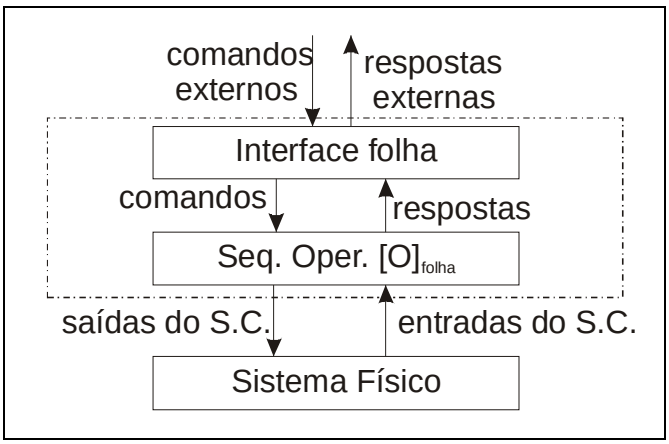

Figura 2.16 - Interface de Comunicação Seqüências Operacionais

Capítulo 3

Proposta de Implementação da Estrutura de Controle Modular Local

3.1 Introdução

Esta seção tem por objetivo apresentar um estudo comparativo entre formas de implementação da estrutura de controle modular local. Para tal fim, é utilizada como aplicação uma célula de manufatura didática denominada MPS, que se encontra no Laboratório de Automação de Sistemas da Pontifícia Universidade Católica do Paraná. A célula apresentada foi objeto de estudo nos trabalhos desenvolvidos em Queiroz *et al.* (2001), Queiroz e Cury (2002), Queiroz (2004), Vieira (2004), Costa et al. (2004) e Gilvan et al. (2005), sendo que na presente dissertação será utilizada toda a modelagem dos subsistemas e especificações apresentada nos referidos trabalhos.

Em Queiroz *et al.* (2001) e Queiroz e Cury (2002), foi utilizado a implementação concentrada (um único CLP) da estrutura de controle modular local (Supervisores Modulares, Sistema Produto e Seqüências Operacionais) para o controle da célula de manufatura. Vieira (2004) propõe a utilização de uma arquitetura distribuída verticalmente, em que as seqüências operacionais são implementadas em diferentes CLP (ver seção 2.3). Em Costa *et al.* (2004) e Costa *et al.* (2005), também foi utilizada a distribuição vertical, com a diferença que os níveis dos supervisores modulares e do sistema produto foram implementados em um microcontrolador. As seqüências operacionais foram implementadas em diferentes CLP.

O estudo comparativo terá como critério fundamental o tamanho do programa (memória) resultante da aplicação da estrutura de controle proposta inicialmente por Queiroz

et al. (2001). Num primeiro momento, é descrita uma implementação nos moldes daquela encontrada em Queiroz *et al.* (2001), Queiroz e Cury (2002) e Vieira (2004), em que as variáveis utilizadas na codificação da estrutura de controle são do tipo booleanas. Num segundo momento, é descrita uma implementação, seguindo o modelo básico de Queiroz *et al.* (2001), em que as variáveis utilizadas são do tipo Word (variável de inteiros). Em ambos os casos consideram-se a distribuição vertical da estrutura de controle conforme Vieira (2004) (ver seção 2.3). A partir dos resultados chega-se a uma estimativa de capacidade de memória exigida para o CLP utilizados no controle da célula.

3.2 Descrição da célula de manufatura utilizada

A célula de manufatura estudada, ilustrada na figura 3.1, é composta por uma mesa giratória de quatro posições (M0), onde são efetuadas operações de furo e teste de peças plásticas e metálicas, e de mais quatro dispositivos operacionais: a esteira de entrada (M1), a furadeira (M2), o aparelho de teste (M3) e o manipulador robótico (M4).

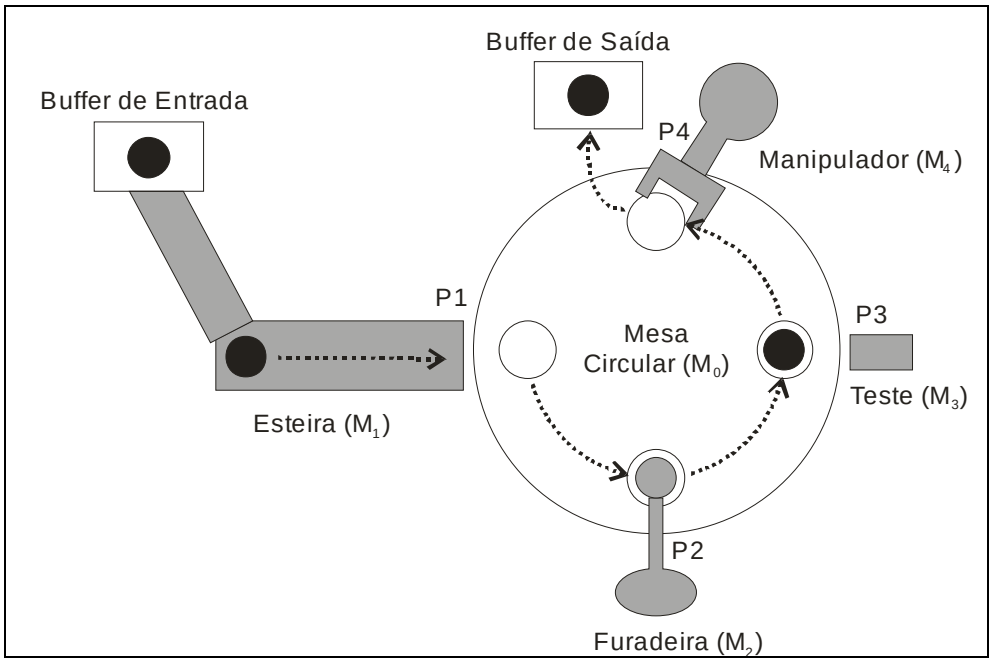

Figura 3.1 - Exemplo Sistema MPS

As peças a serem processadas possuem uma rota definida: é retirada do buffer de entrada e conduzida pela Esteira (M_1) até o ponto P1 pertencente à mesa giratória (M_0); a mesa giratória é responsável pelo transporte das peças entre as posições P1, P2, P3 e P4, sendo que a cada giro de 90º uma peça é conduzida a uma nova posição; nas posições P2, P3 e P4 as peças sofrem processamentos e transporte por M2, M3 e M4, respectivamente.

De acordo com a metodologia proposta por Queiroz e Cury (2000b), o primeiro passo para a síntese de controladores locais é a modelagem dos subsistemas envolvidos. O funcionamento de cada dispositivo corresponde a uma seqüência de operações específicas, como acionamento de motores e atuadores pneumáticos e leitura de sensores. No entanto, os problemas operacionais descritos não ocorrem nas seqüências operacionais particulares de cada subsistema, mas, sim, na descoordenação entre o início e o final das diversas seqüências. Com isso, para a síntese da lógica de controle, os modelos dos dispositivos podem ser abstraídos, bastando representá-los em termos dos eventos de início e final de operação. Essa simplificação ajuda a reduzir o tamanho dos modelos e, por conseqüência, a complexidade do processo de síntese e o número de estados dos supervisores resultantes.

A abstração descrita no parágrafo anterior permite a utilização de um modelo em autômato similar para todos os subsistemas, tendo suas operações iniciadas por um evento controlável a_i e finalizadas através de um evento não controlável b_i. No estado inicial o subsistema encontra-se em repouso, enquanto no segundo estado o subsistema encontra-se em operação. Os modelos em autômatos G_i são apresentados na figura 3.2.

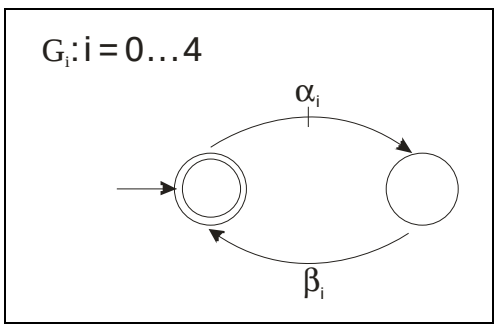

Figura 3.2 - Representação Autômatos Subsistemas MPS

Conforme a Teoria de Controle Modular Local (TCML), o próximo passo é o desenvolvimento das especificações condizentes com a necessidade de controle. O autômato E_a, ilustrado na figura 3.3, modela uma especificação que garante que a mesa não vai girar à

toa, isto é, sem ao menos uma peça bruta em P1 ou uma peça furada em P2 ou uma peça testada em P3.

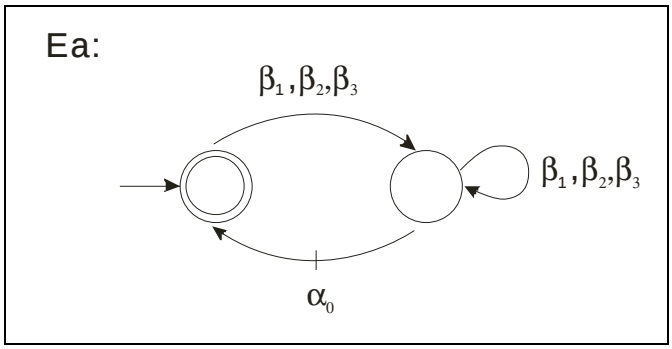

Figura 3.3 - Representação Autômatos Especificação E_a

As quatro especificações de segurança que impedem a mesa de girar enquanto a esteira (E_{b1}), a furadeira (E_{b2}), o teste (E_{b3}) ou/e o manipulador (E_{b4}) estiverem operando podem ser descritas pelo autômato E_{bi} de índice i da figura 3.4.

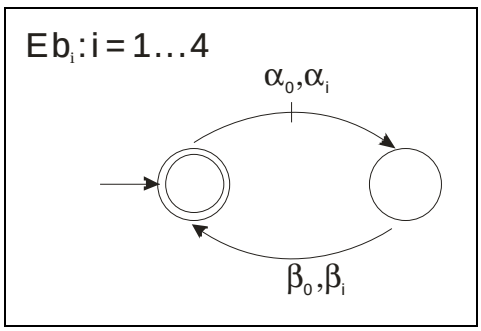

Figura 3.4 - Representação Autômatos Especificação E_b

Os possíveis problemas decorrentes do fluxo de múltiplas peças na mesa podem ser evitados pelas seguintes especificações: E_{c1}, relativa à movimentação de peças brutas entre as posições P1 e P2; E_{c2}, para a manipulação de peças furadas entre P2 e P3; E_{c3}, correspondente ao fluxo de peças testadas entre P3 e P4. Para tanto, a especificação E_{c1} evita sobrepor peças em P1, furar sem peça bruta em P2 e girar a mesa com peça bruta em P2. Já a especificação E_{c2} proíbe furar duas vezes a mesma peça, testar sem peça furada em P3 e girar a mesa com peça furada e não testada em P3, enquanto E_{c3} impede testar duas vezes a mesma peça,

acionar o manipulador sem peça em P4 e girar a mesa com peça em P4. Como especificações têm a mesma estrutura, pode-se ilustrá-las pelo modelo indexado E_{ci} da figura 3.5.

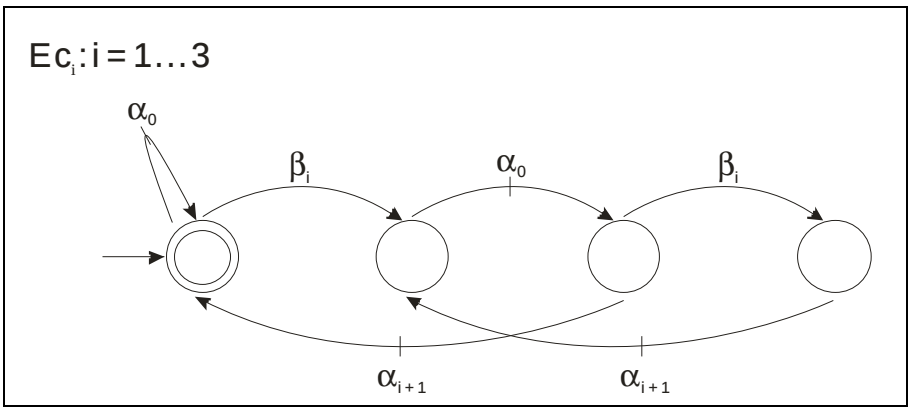

Figura 3.5 - Representação Autômatos Especificação E_c

Como os subsistemas já estão modelados de forma assíncrona, o sistema encontra-se na mais refinada RSP. Assim, podem-se obter as plantas locais $G_{loc,a}$, $G_{loc,b1}$, $G_{loc,b2}$, $G_{loc,b3}$, $G_{loc,b4}$, $G_{loc,c1}$, $G_{loc,c2}$ e $G_{loc,c3}$ respectivas às especificações E_a, E_{b1}, E_{b2}, E_{b3}, E_{b4}, E_{c1}, E_{c2} e E_{c3} fazendo-se a composição dos subsistemas que tenham eventos comuns a elas, figura 3.6.

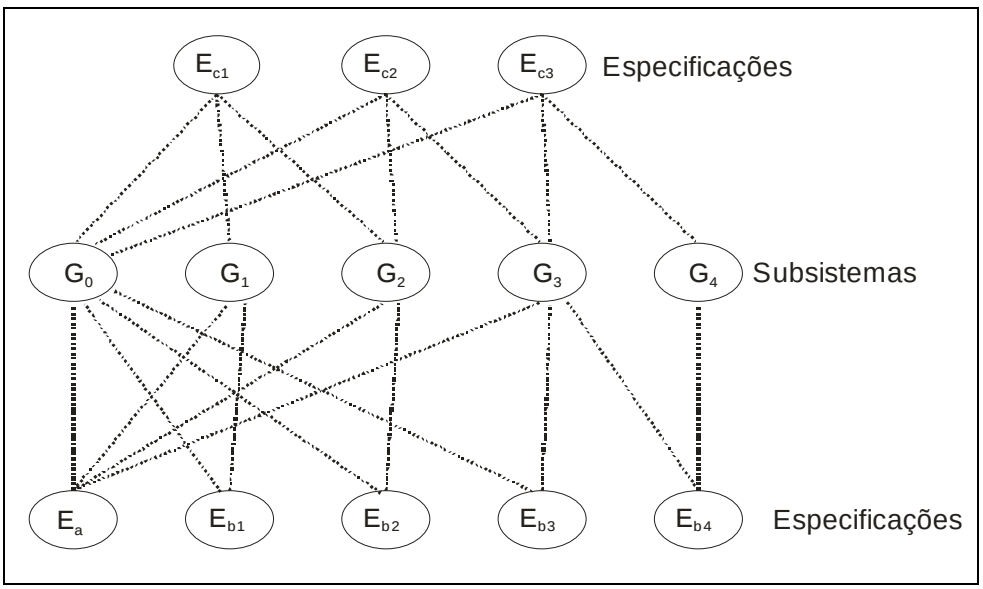

Figura 3.6 - Composição Plantas Locais

$$G_{loc,a} = G_0 // G_1 // G_2 // G_3$$

$$G_{loc,b1} = G_0 // G_1$$

$$G_{loc,b2} = G_0 // G_2$$

$$G_{loc,b3} = G_0 // G_3$$

$$G_{loc,b4} = G_0 // G_4$$

$$G_{loc,c1} = G_0 // G_1 // G_2$$

$$G_{loc,c2} = G_0 // G_2 // G_3$$

$$G_{loc,c3} = G_0 // G_3 // G_4$$

Após a obtenção das plantas locais, calculam-se as especificações locais $E_{loc,a}$, $E_{loc,b1}$, $E_{loc,b2}$, $E_{loc,b3}$, $E_{loc,b4}$, $E_{loc,c1}$, $E_{loc,c2}$ e $E_{loc,c3}$,figura 3.7.

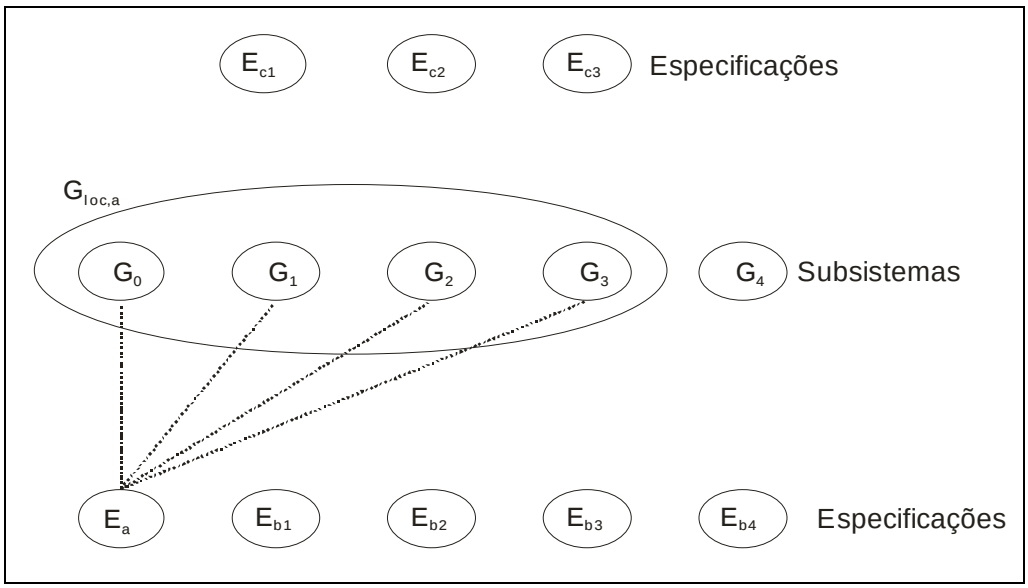

Figura 3.7 - Composição Especificações Locais

$$E_{loc,a} = G_{loc,a} \,//\, E_a$$
$$E_{loc,b1} = G_{loc,b1} \,//\, E_{b1}$$
$$E_{loc,b2} = G_{loc,b2} \,//\, E_{b2}$$
$$E_{loc,b3} = G_{loc,b3} \,//\, E_{b3}$$
$$E_{loc,b4} = G_{loc,b4} \,//\, E_{b4}$$
$$E_{loc,c1} = G_{loc,c1} \,//\, E_{c1}$$
$$E_{loc,c2} = G_{loc,c2} \,//\, E_{c2}$$
$$E_{loc,c3} = G_{loc,c3} \,//\, E_{c3}$$

O passo seguinte é o cálculo das máximas linguagens controláveis contidas nas especificações locais. Fazendo a composição síncrona de todas as linguagens resultantes, obtém-se o autômato S, que é TRIM. Isso indica que a condição de modularidade local é verificada e, por conseguinte, a abordagem modular é equivalente à monolítica.

$$SLa = SupC(E_A, G_{loc,A})$$
$$SLb_1 = SupC(E_{B1}, G_{loc,B1})$$
$$SLb_2 = SupC(E_{B2}, G_{loc,B2})$$
$$SLb_3 = SupC(E_{B3}, G_{loc,B3})$$
$$SLb_4 = SupC(E_{B4,} G_{loc,B4})$$
$$SLc_1 = SupC(E_{C1,} G_{loc,C1})$$
$$SLc_2 = SupC(E_{C2,} G_{loc,C2})$$
$$SLc_3 = SupC(E_{C3}, G_{loc,C3})$$

$$(SLa \,//\, SLb_1 \,//\, SLb_2 \,//\, SLb_3 \,//\, SLb_4 \,//\, SLc_1 \,//\, SLc_2 \,//\, SLc_3)$$
$$= TRIM(SLa \,//\, SLb_1 \,//\, SLb_2 \,//\, SLb_3 \,//\, SLb_4 \,//\, SLc_1 \,//\, SLc_2 \,//\, SLc_3)$$

3.3 Primeira forma de implementação da estrutura de controle modular local

Queiroz *et al.* (2001) e Queiroz e Cury (2002) introduzem o modelo de implementação em três níveis visto na figura 2.14, sendo que sob o ponto de vista da estrutura física de controle, foi utilizado um único CLP para realizar tais níveis. Dessa forma, utiliza-se o termo 'implementação concentrada'. Na presente dissertação, opta-se por uma implementação denominada 'distribuição vertical', segundo Vieira (2004), Vieira *et al.* (2004) e Vieira e Cury (2004), ilustrada na figura 3.8. A justificativa para esta opção se dá em função de um dos objetivos do trabalho: deseja-se analisar a capacidade de memória exigida em um CLP, mas somente dos dois níveis da estrutura de controle – supervisores modulares e sistema produto. Sabendo-se que as seqüências operacionais correspondem ao detalhamento da abstração do modelo do sistema físico, o programa relacionado pode variar mesmo mantendo os níveis supervisores modulares e sistema produto. Dessa forma, não será considerada para fins de análise a memória exigida das seqüências operacionais do problema tratado.

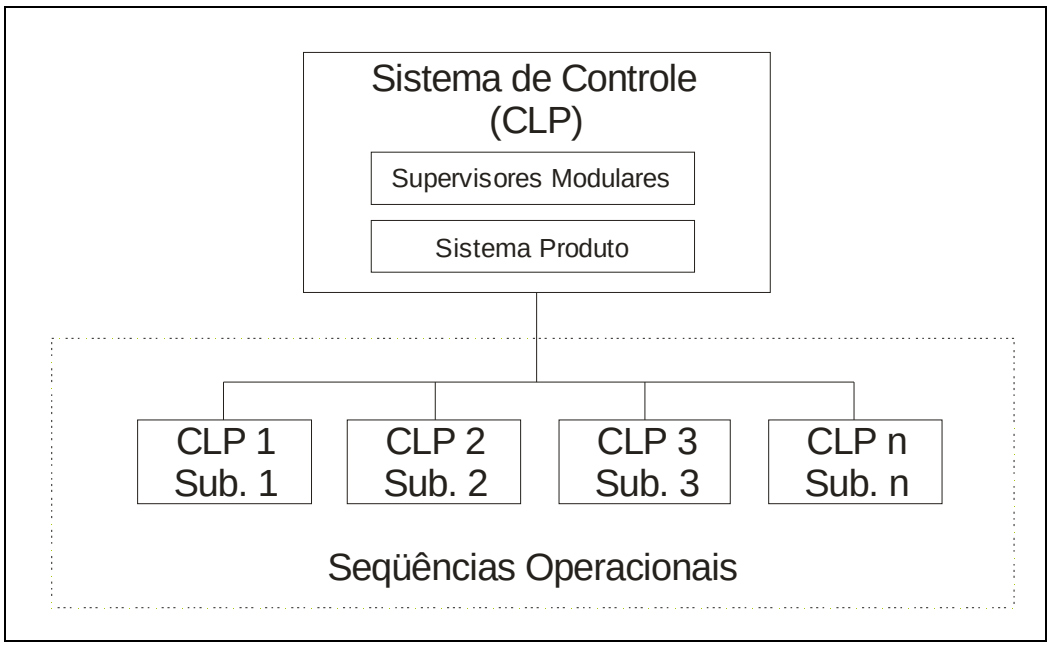

Figura 3.8 - Estrutura Física de Controle.

O experimento é realizado em um CLP específico, D0-06USER-M (2004), mas os resultados obtidos são válidos também para os demais controladores lógicos que utilizem a linguagem diagrama escada ou também denominada diagrama Ladder (IEC 61131-3, 1993).

A especificação básica do CLP é descrita a seguir:

- CLP Koyo DL06-DD;
- Linguagem de Programação – Ladder;
- Capacidade 14,8k word's de Memória de programas;
- 2 Portas de comunicação;
- 16 entradas digitais;
- 20 saídas digitais;
- Protocolo de comunicação: Modbus RTU;

Seguindo o modelo de implementação de Queiroz *et al.* (2001), programam-se os supervisores modulares e o sistema produto. Para ilustrar tal programação, apresenta-se na figura 3.9 o supervisor modular local $SLc_1 = SupC(E_{c1}, G_{loc,c1})$.

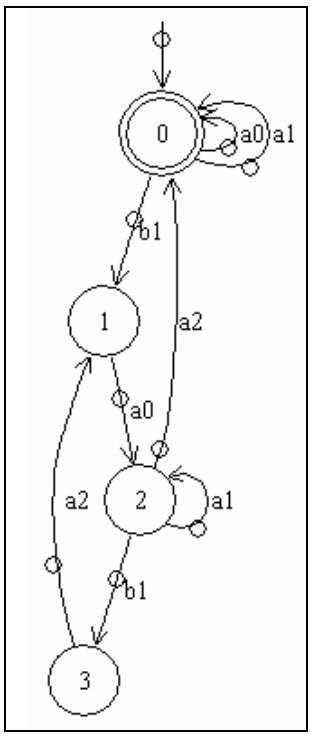

Figura 3.9 - Representação em Autômato do Supervisor Modular Local SLc_1

Nesta forma de implementação considera-se $S_{x,y}$ o estado do supervisor, $G_{n,y}$ o estado dos subsistemas, a_n os eventos controláveis e b_n os eventos não-controláveis, tal que x corresponde ao número do supervisor, n ao número do subsistema e y corresponde ao

estado ativo de cada autômato. Nesta implementação cada estado $S_{x,y}, G_{n,y}$ e evento a_n, b_n de um autômato é associado a um sinalizador booleana C_j, “0” ou “1”. Onde j corresponde ao número do sinalizador escolhido livremente pelo programador, conforme pode ser visto na tabela 3.1.

Tabela 3.1: - Lista de Atribuição de Variáveis.

Autômatos	Variável Alocada para os Estados		Variável Alocada para os Eventos	
Supervisor SLc_1	$C0$	Estado 0 S_0 ($S_{0.0}$)	Alocado pelos Subsistemas	
	$C1$	Estado 1 S_0 ($S_{0.1}$)		
	$C2$	Estado 2 S_0 ($S_{0.2}$)		
	$C3$	Estado 3 S_0 ($S_{0.3}$)		
Subsistema G_0	$C200$	Estado 0 G_0	$C300$	Evento a_0
	$C201$	Estado 1 G_0	$C301$	Evento b_0
Subsistema G_1	$C202$	Estado 0 G_1	$C302$	Evento a_1
	$C203$	Estado 1 G_1	$C303$	Evento b_1
Subsistema G_2	$C204$	Estado 0 G_2	$C304$	Evento a_2
	$C205$	Estado 1 G_2	$C305$	Evento b_2

Para garantir a correta evolução dos supervisores e do sistema produto deve-se codificar a inicialização dos mesmos, ou seja, garantir que o programa inicie com todos os autômatos nos seus respectivos estados iniciais. As figura 3.10 e figura 3.11 representa essa codificação, sendo que o contato SP0 está somente ativo no primeiro ciclo de varredura do CLP. Dessa forma, todas as variáveis correspondentes aos estados iniciais dos autômatos são carregados na primeira execução do programa.

Figura 3.10 - Representação Ladder Estágio Inicial 1ª Parte

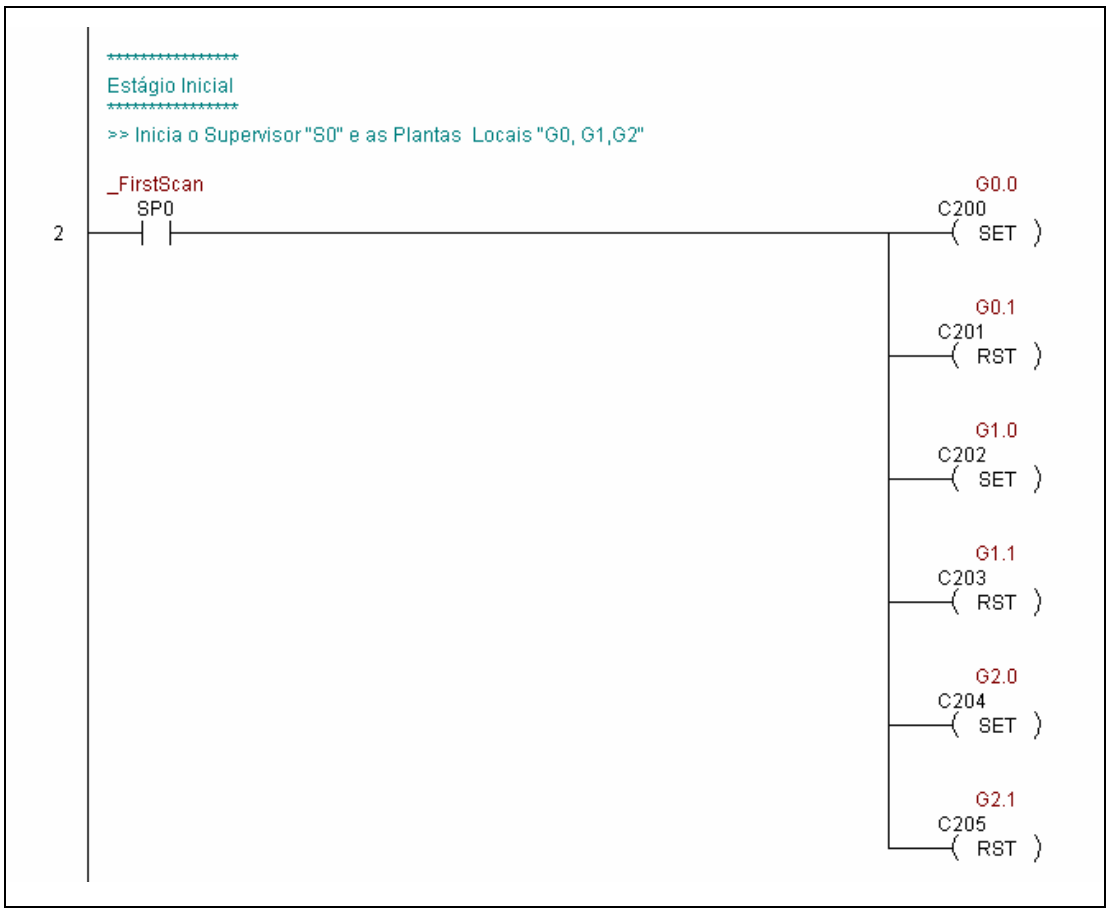

Figura 3.11 - Representação Ladder Estágio Inicial 2ª Parte

Os supervisores resultantes do processo de síntese apresentados na seção anterior são descritos como máquinas de estados finitos em que, para cada estado ativo, um conjunto de eventos controláveis deve ser desabilitado. Desta forma, a implementação do programa de controle consiste basicamente em fazer o CLP se comportar como um jogador de autômatos.

A programação dos supervisores é realizada através de duas estruturas: a primeira, relacionada a evolução de estados de acordo com os eventos ocorridos no sistema produto; e a segunda, a estrutura de desabilitação de eventos controláveis do sistema produto.

A figura 3.12 e a figura 3.13 apresentam o segmento do diagrama escada correspondente a evolução do supervisor SLc_1. Este supervisor evolui de acordo com a ocorrência dos eventos gerados nos subsistemas G_0, G_1 e G_2. Por exemplo, ma linha 6 do diagrama escada (figura 3.13), codifica-se a evolução do estado 2 ($S_{0.2}$) para o estado 0 ($S_{0.0}$) do supervisor SLc_1, com a ocorrência do evento $a_2 (C_2)$ gerado no subsistema G_2.

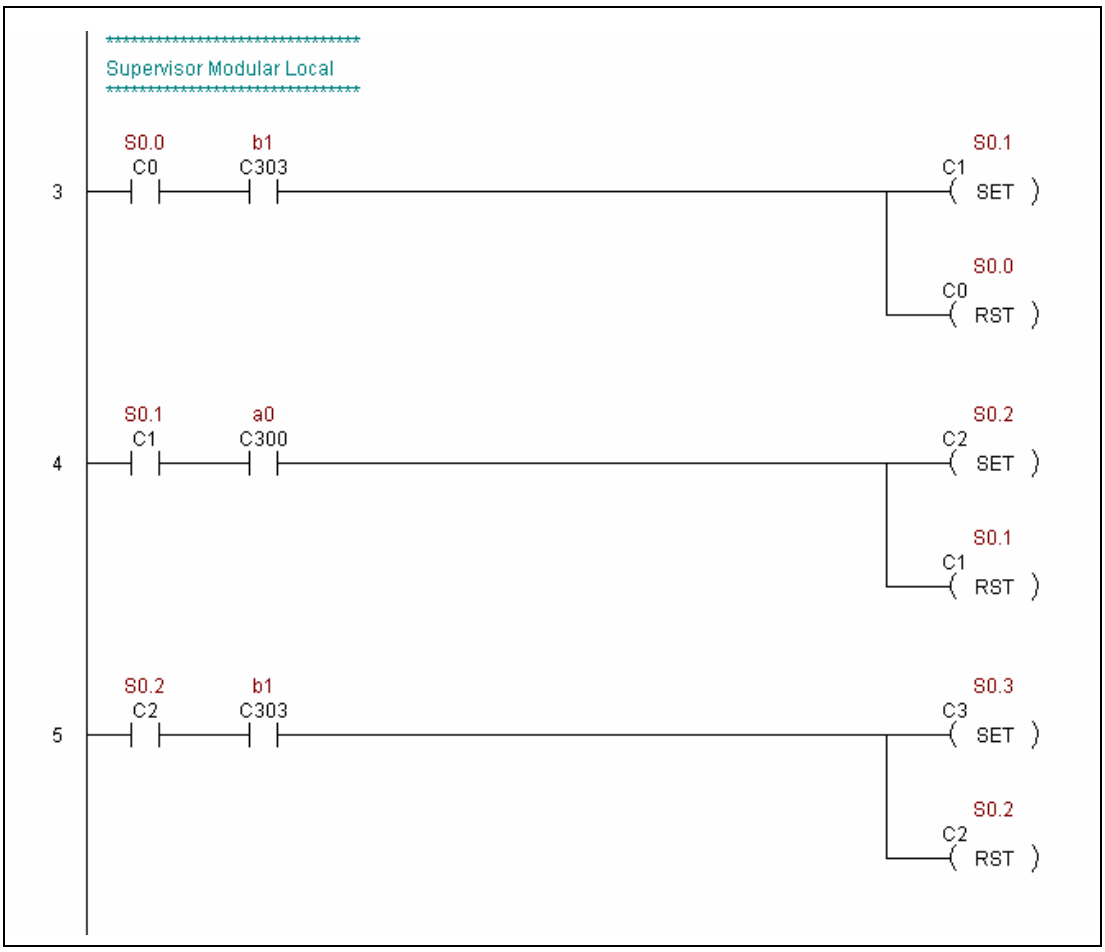

Figura 3.12 - Representação Ladder Supervisor Modular Local 1ª Parte

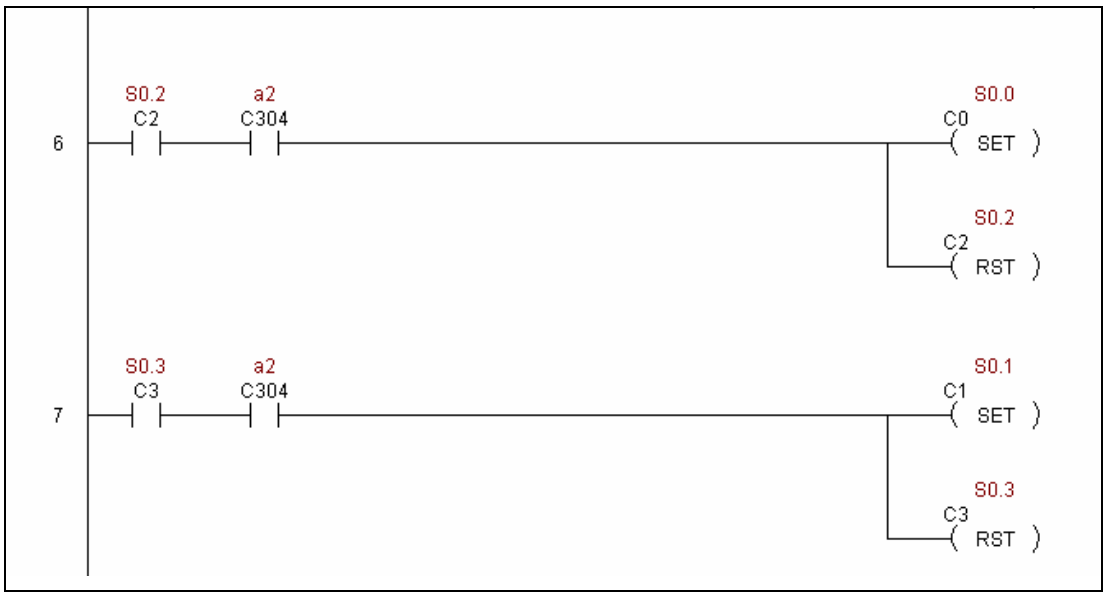

Figura 3.13 - Representação Ladder Supervisor Modular Local 2ª Parte

A estrutura de desabilitação de eventos controláveis é realizada mediante a geração de variáveis, de acordo com o estado ativo do supervisor. Dependendo do estado atual do supervisor, um conjunto de eventos controláveis é desabilitado. Para tanto, para cada evento controlável é associado um sinal de desabilitação, conforme tabela 3.2.

Tabela 3.2: - Lista de Atribuição Associado a Desabilitação de Eventos Controláveis.

Autômatos	Variável Alocada para os Eventos Controláveis		Variável Alocada para os Sinais de Desabilitação	
Subsistema G_0	$C300$	Evento a_0	$C100$	Evento a_0
Subsistema G_1	$C302$	Evento a_1	$C101$	Evento a_1
Subsistema G_2	$C304$	Evento a_2	$C102$	Evento a_2

A figura 3.14 e a figura 3.15 apresentam a estrutura de desabilitação do supervisor SLc_1. Por exemplo, a linha 9 do diagrama escada (figura 3.15) codifica a desabilitação do evento a_1 através da variável d_a_1. Isso acontece sempre que supervisor esteja nos estados 1 ($S_{0.2}$) ou 3 ($S_{0.3}$).

Os subsistemas G_n são implementados no nível denominado sistema produto, conforme modelo proposto por Queiroz *et al.* (2001). Os eventos controláveis são automaticamente disparados quando não são desabilitados pelo supervisor. Os eventos não

controláveis são sinalizados pelo nível Seqüências Operacionais. Cada evento também sinaliza uma mudança de estado do supervisor, atualizando-o continuamente conforme descrito anteriormente. Desta forma, evita-se que duas transições seguidas ocorram no sistema produto sem que os supervisores tenham sido atualizados.

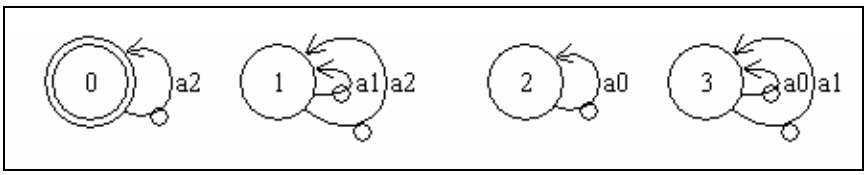

Figura 3.14 - Representação Autômato Eventos Desabilitados em cada Estado

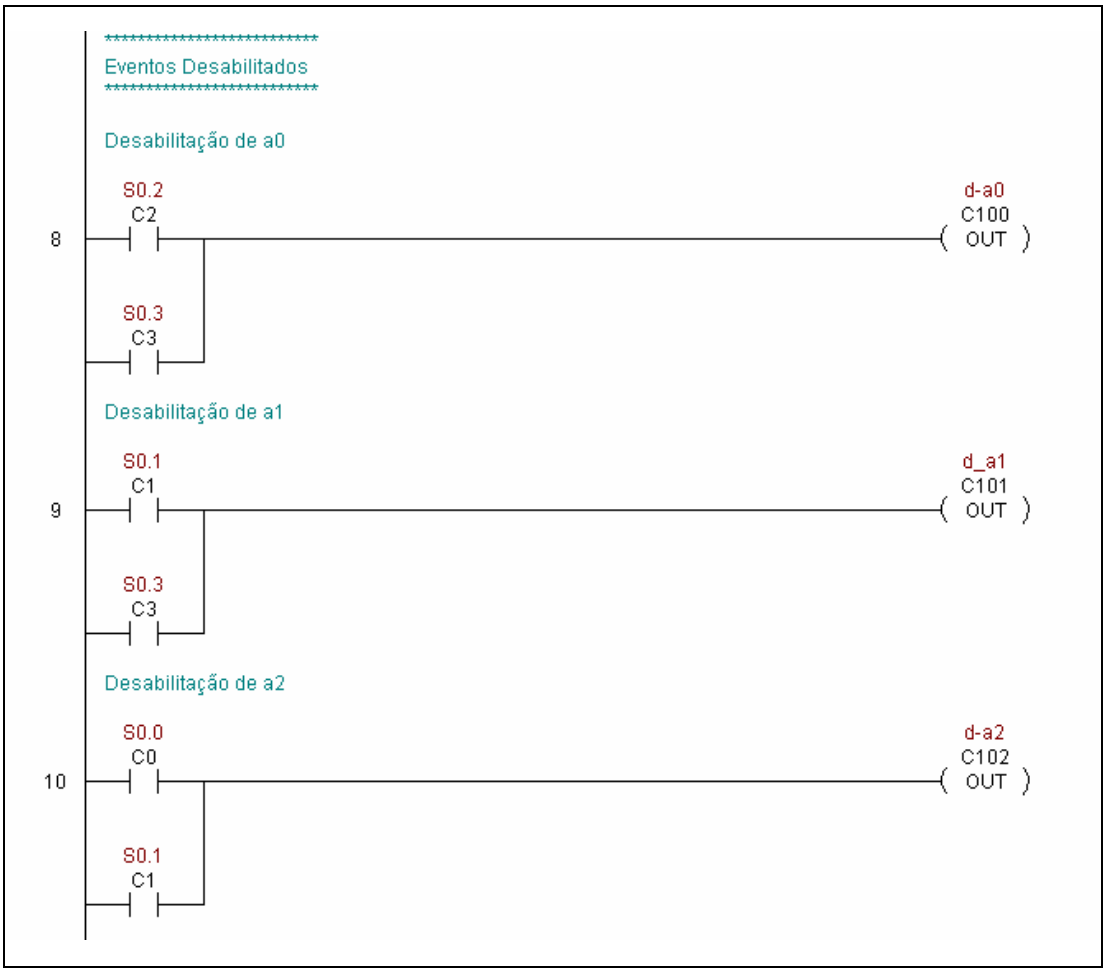

Figura 3.15 - Representação Ladder Eventos Desabilitados pelo Supervisor

A figura 3.17 e a figura 3.18 apresentam os segmentos do diagrama escada correspondente ao subsistema G_0, mostrado na figura 3.16. Na figura 3.17, observa-se que o

sinal de desabilitação d_a_0 enviado pelo supervisor é colocado como condição de sinalização do evento a_0 (C_0) e também do sinal de início da seqüência operacional G_0 e ($e\_ini\_G_0$). Caso o sinal d_a_0 não esteja sendo gerado pelo supervisor, o programa evolui da seguinte forma: é gerado a variável C_0 (correspondente ao evento a_0) e é sinalizado o início da seqüência operacional G_0 (através de $e\_ini\_G_0$).

A figura 3.18 apresenta o código correspondente a sinalização do evento b_0 do subsistema G_0. A variável $e\_fim\_G_0$ corresponde ao final do ciclo de operação da seqüência operacional G_0. No momento que esta variável é gerada, o subsistema G_0 evolui do estado 1 ($G_{0.1}$) para o estado inicial 0 ($G_{0.0}$).

Da figura 3.19 até a figura 3.24 apresentam-se a codificação dos subsistemas G_1 e G_2, de forma similar ao apresentado anteriormente para o subsistema G_0.

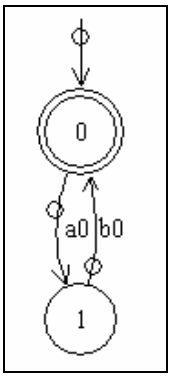

Figura 3.16 - Representação Autômato Subsistema G_0

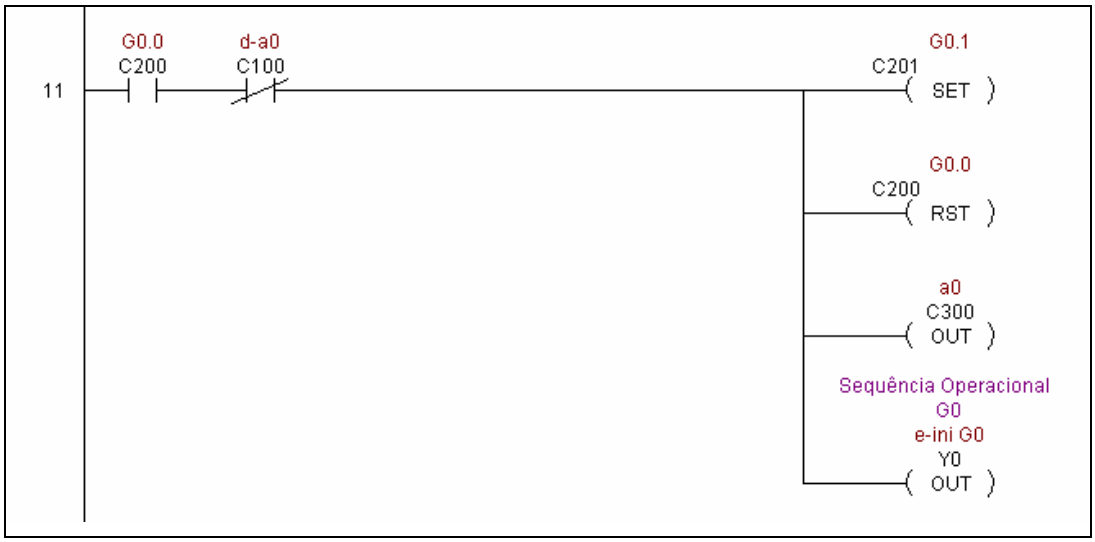

Figura 3.17 - Representação Ladder Subsistema G_0 1ª Parte

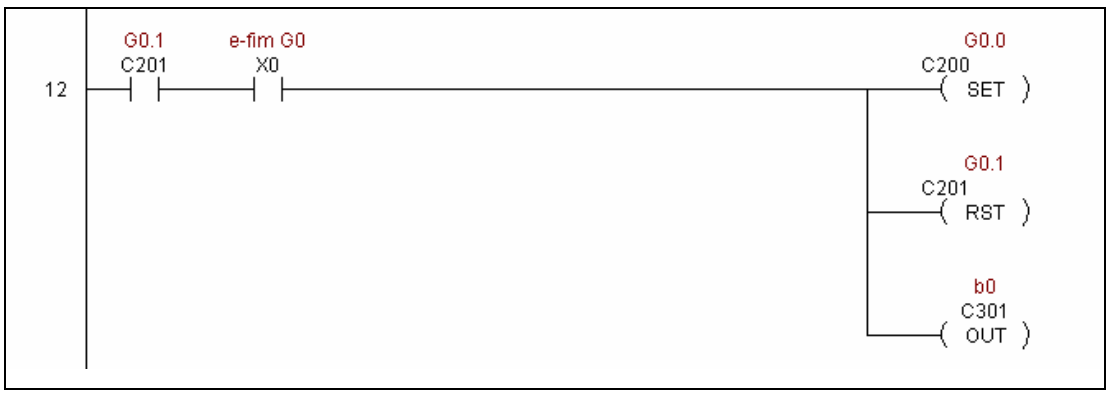

Figura 3.18 - Representação Ladder Subsistema G_0 2ª Parte

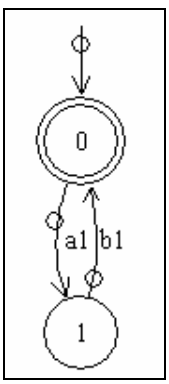

Figura 3.19 - Representação Autômato Subsistema G_1

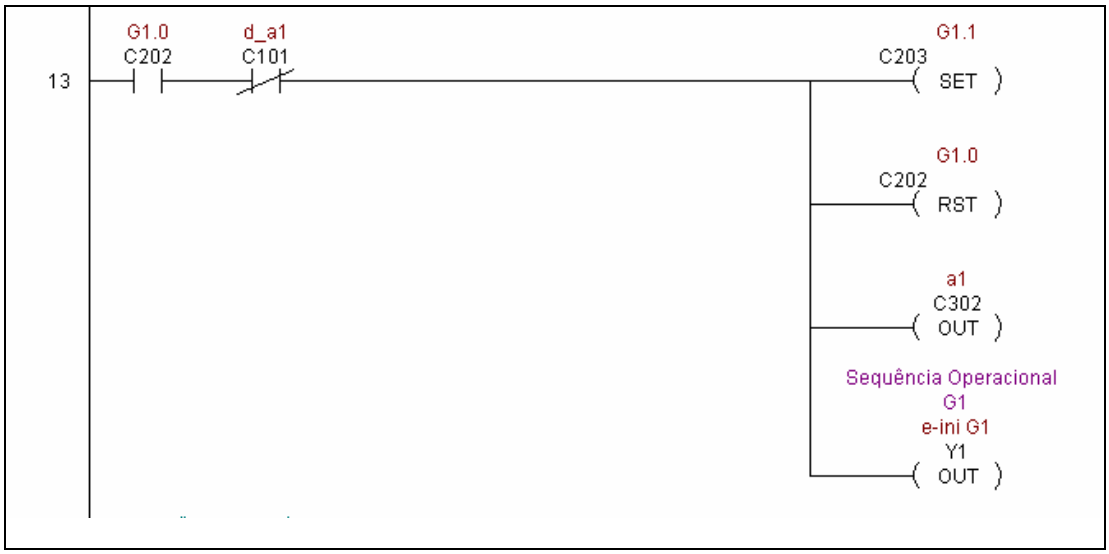

Figura 3.20 - Representação Ladder Subsistema G_1 1ª Parte

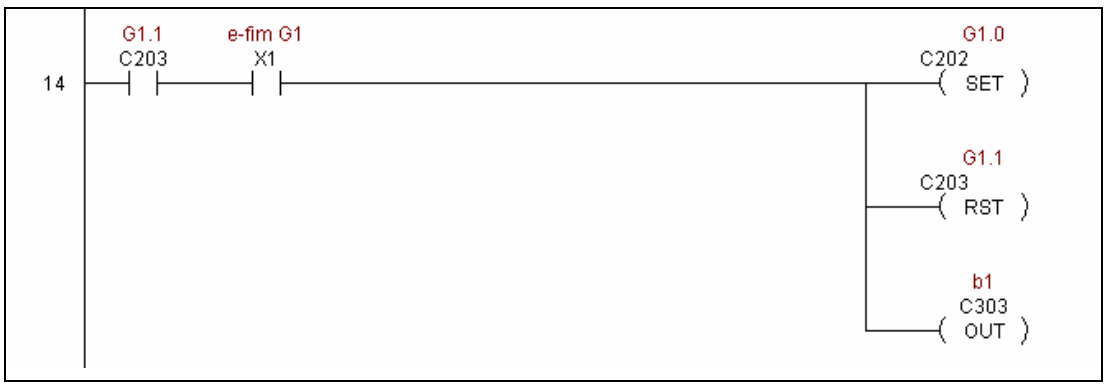

Figura 3.21 - Representação Ladder Subsistema G_1 2ª Parte

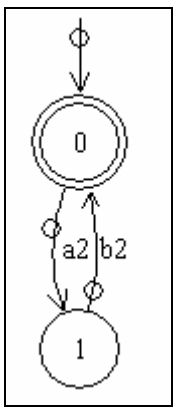

Figura 3.22 - Representação Autômato Subsistema G_2

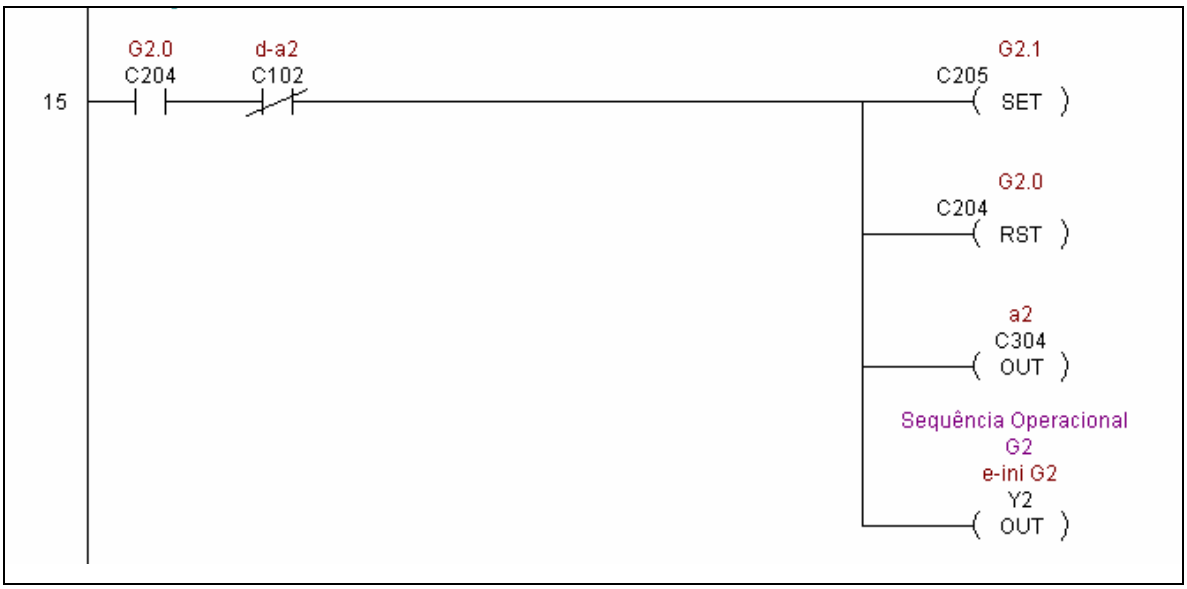

Figura 3.23 - Representação Ladder Subsistema G_2 1ª Parte

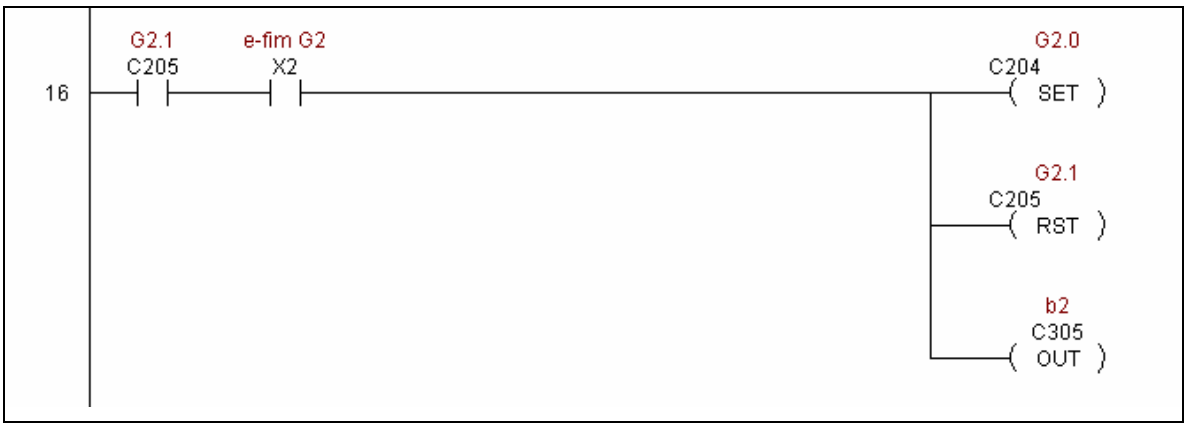

Figura 3.24 - Representação Ladder Subsistema G_2 2ª Parte

3.4 Segunda forma de implementação da estrutura de controle modular local

Em linhas gerais, os CLP tratam e distinguem dois tipos de variáveis de dados: a variável de bit e a variável de palavra (Byte, Word, Inteiros, etc), mesmo que aparentemente alguns não dividam ou aloquem endereços distintos para essa utilização, quando usadas por funções que as caracterizam, estas são indicadas no programa como variável de bits ou variável de palavras.

As variáveis de bit são aquelas que assumem apenas dois estados “0” ou “1” e são geralmente utilizadas como sinalizadores (*flags*) ou como endereçamento das entradas e saídas digitais do CLP. As variáveis de palavra, em maior número, são variáveis com capacidades limitadas segundo o número de bits do processador ou o arranjo dessas variáveis. Por exemplo, para uma palavra de 16 bits pode-se registrar até 65536 posições.

Na forma anterior, pelo critério de implementação, cada estado e evento é vinculado a um bit de memória e o número máximo de informação que cada variável pode registrar limita-se ao número de bits do processador. Por exemplo, uma variável de 16 bits pode registrar apenas 16 posições.

Na segunda forma de implementação, os estados e eventos dos níveis SM e SP são programados utilizando-se de variáveis de palavras. Dessa maneira, pretende-se diminuir o número de variáveis envolvidas bem como estruturar melhor o programa. A concepção básica de troca de sinais entre os níveis da estrutura de controle é mantida, conforme *Queiroz et al.* (2001) e *Queiroz e Cury* (2002).

A figura 3.25 apresenta novamente o supervisor $SLc_1 = S0 = SupC(E_{C1}, G_{loc,C1})$ para o melhor acompanhamento.

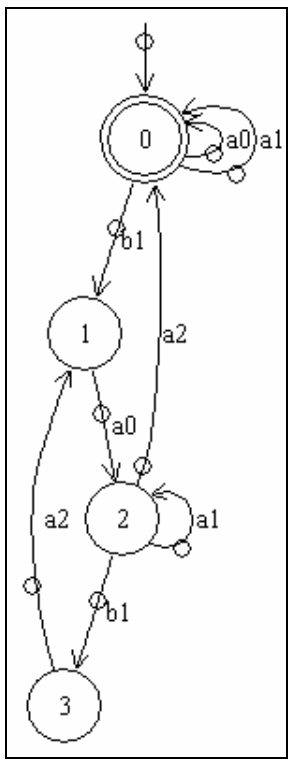

Figura 3.25 - Representação Autômato Supervisor Modular Local

Nota-se a seguir, na implementação Ladder a atribuição de cada conjunto de estado e evento a uma variável de palavra, pois tanto um estado como evento somente pode assumir um valor de cada vez. A

tabela 3.3 mostra quais variáveis foram alocadas para representar o conjunto dos estados e eventos dos autômatos, relacionado cada estado ou evento a um valor inteiro (k).

Tabela 3.3: - Lista de Atribuição de Variáveis.

Autômatos	Variável Alocada para os Estados		Variável Alocada para os Eventos	
Supervisor SLc_1	V1200		Alocado pelos Subsistemas	
	$K0$	Estado 0 S_0		
	$K1$	Estado 1 S_0		
	$K2$	Estado 2 S_0		
	$K3$	Estado 3 S_0		
Subsistema G_0	V1400		V400	
	$K0$	Estado 0 G_0	$K0$	Evento a_0
	$K1$	Estado 1 G_0	$K1$	Evento b_0
Subsistema G_1	V1401		V401	
	$K0$	Estado 0 G_1	$K0$	Evento a_1
	$K1$	Estado 1 G_1	$K1$	Evento b_1
Subsistema G_2	V1401		V402	
	$K0$	Estado 0 G_2	$K0$	Evento a_2
	$K1$	Estado 1 G_2	$K1$	Evento b_2

Conforme o raciocínio utilizado na primeira forma de implementação, figura 3.10 e figura 3.11, o código em Ladder da figura 3.26 e figura 3.27 garante o correto funcionamento pelo início de todos os autômatos em seus estados iniciais.

Nesta linguagem a função LD (loaud) carrega a constante K para uma variável interna do processador (acumulador) e a função OUT (out) retira o valor dessa variável interna e atribui a variável indicada pela função, pode-se dizer então que as variáveis recebem o valor da constante. A nova forma de implementação compara e valida cada linha escada através das funções de manipulação de variáveis. Um contato com o símbolo de igual, visto na figura

3.28, significa que este somente será ativo se o valor da variável for igual ao valor da constante K.

O novo desenvolvimento pode ser melhor compreendido no confronto direto da figura 3.28 a figura 3.36 com as figuras anteriores da primeira forma de implementação (figura 3.12 a figura 3.24).

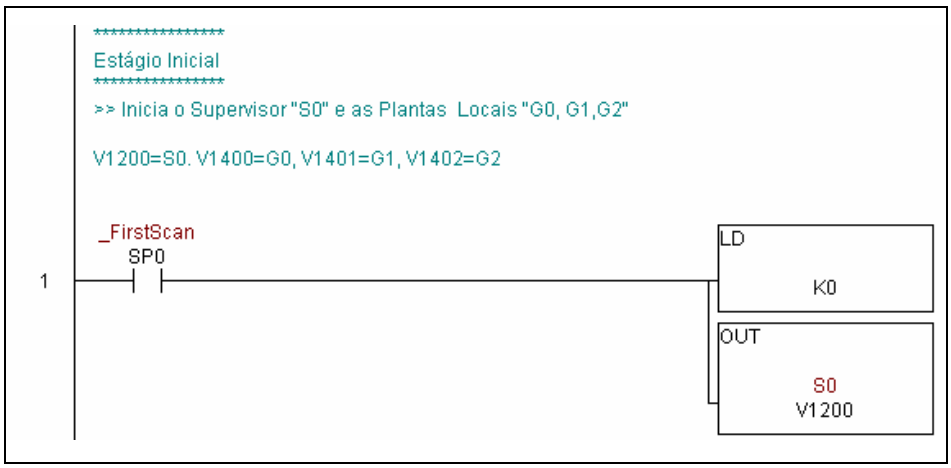

Figura 3.26 - Representação Ladder Estágio Inicial 1ª Parte

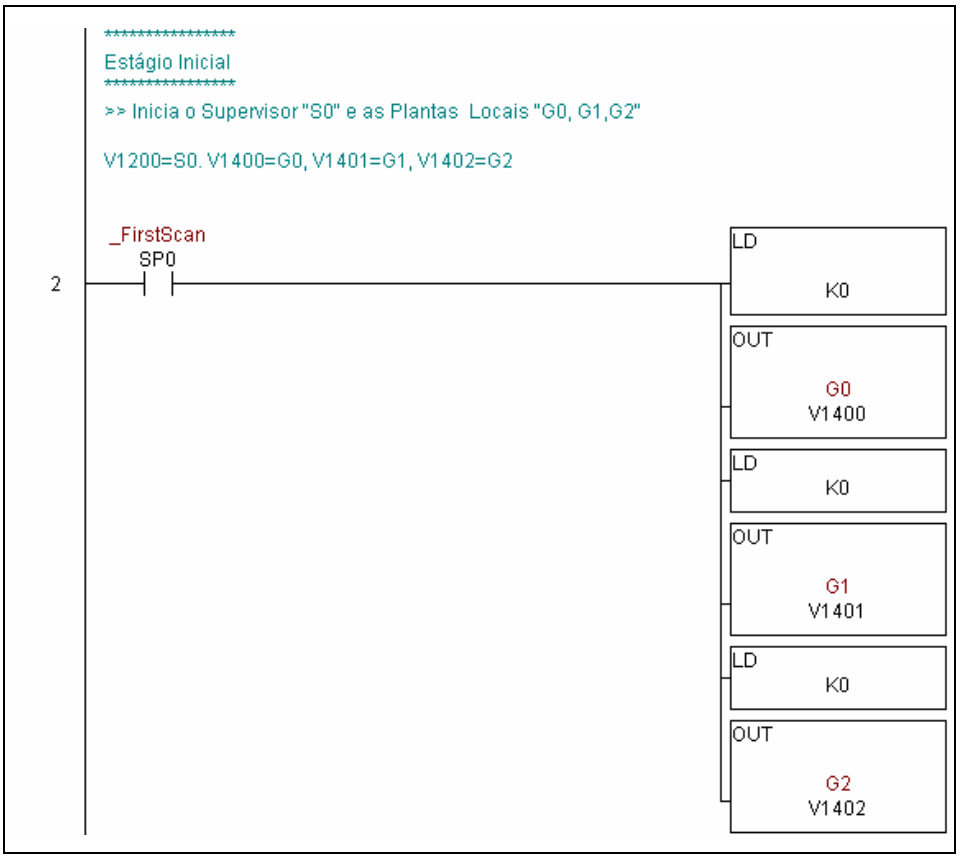

Figura 3.27 - Representação Ladder Estágio Inicial 2ª Parte

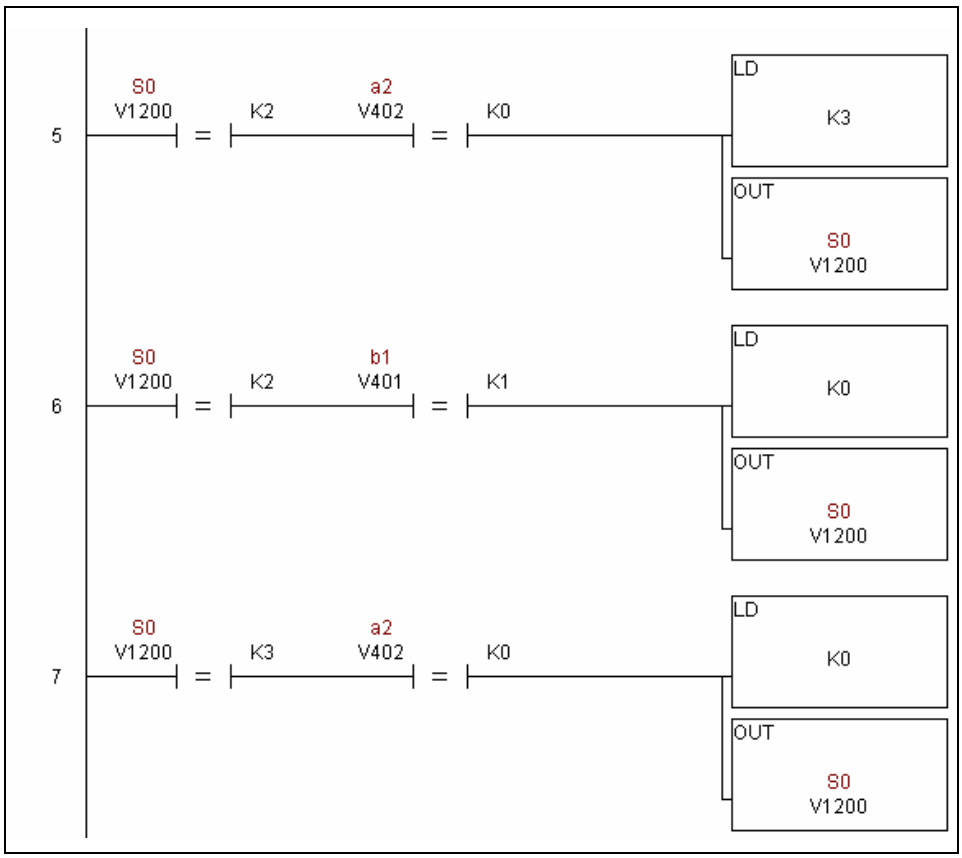

Figura 3.28 - Representação Ladder Supervisor Modular Local 1ª Parte

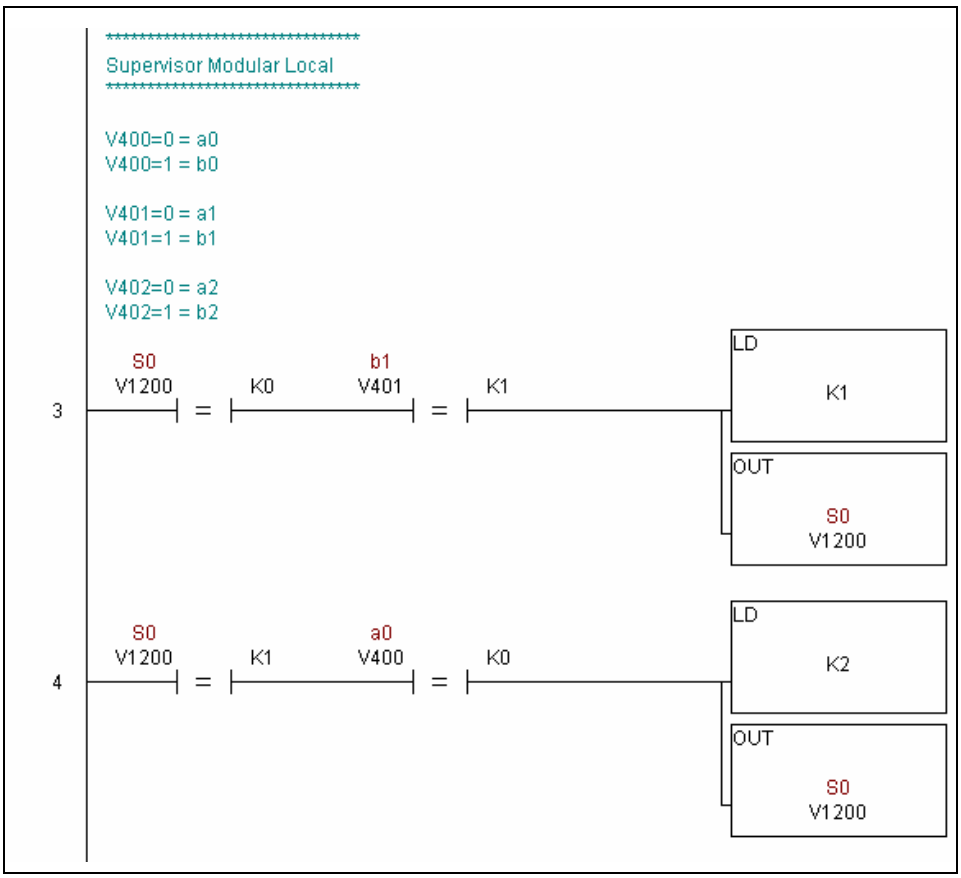

Figura 3.29 - Representação Ladder Supervisor Modular Local 2ª Parte

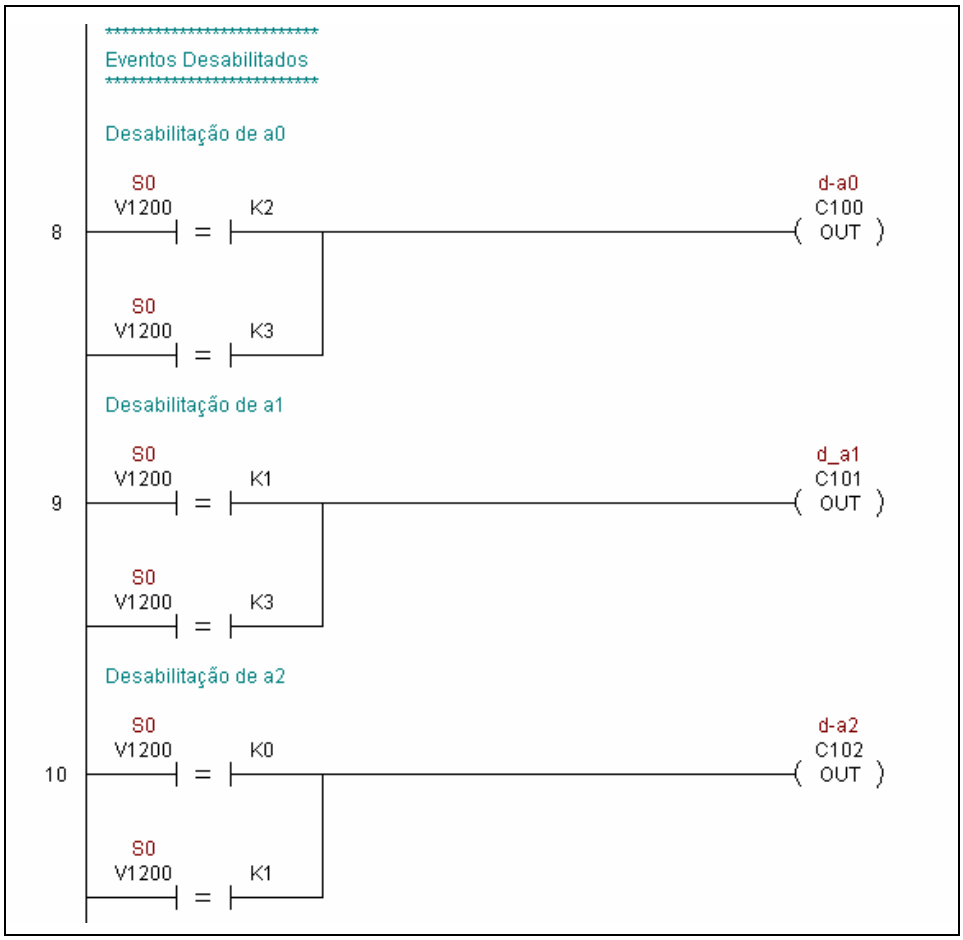

Figura 3.30 - Representação Ladder Eventos Desabilitados pelo Supervisor

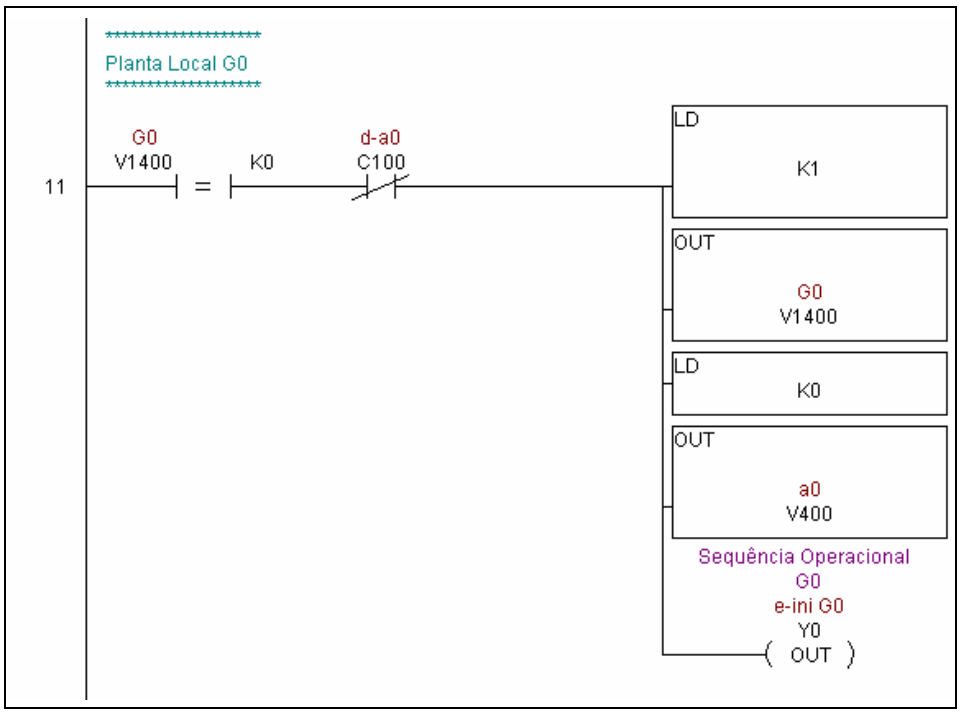

Figura 3.31 - Representação Ladder Subsistema G_0 1ª Parte

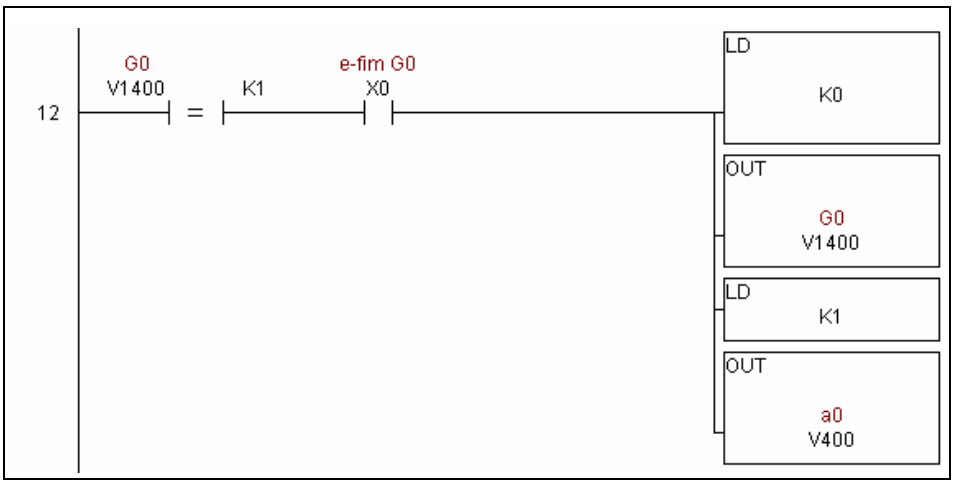

Figura 3.32 - Representação Ladder Subsistema G_0 2ª Parte

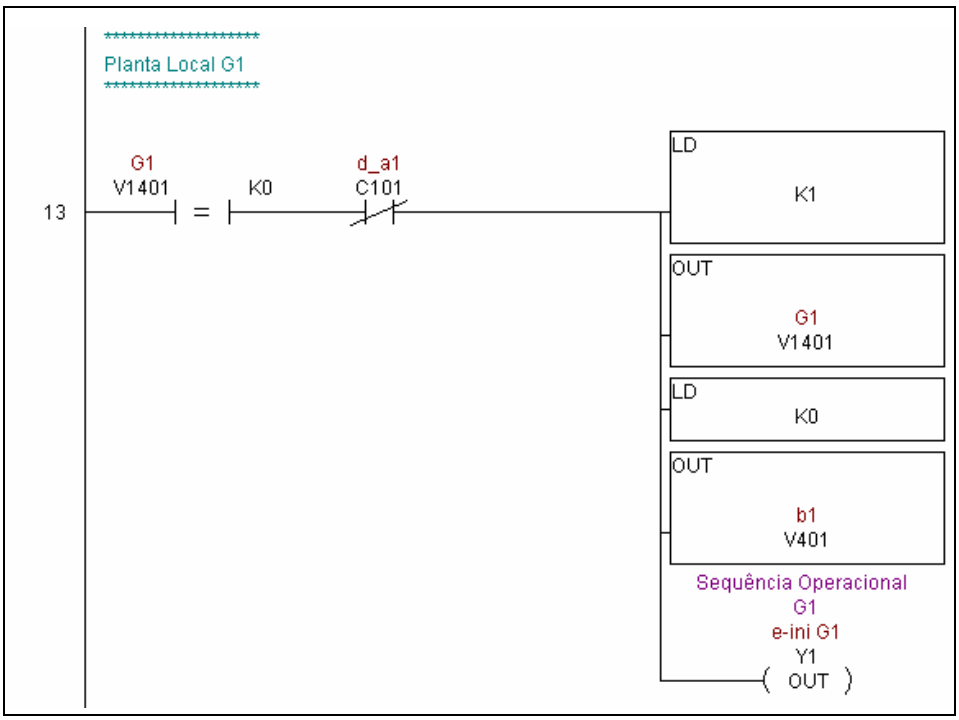

Figura 3.33 - Representação Ladder Subsistema G_1 1ª Parte

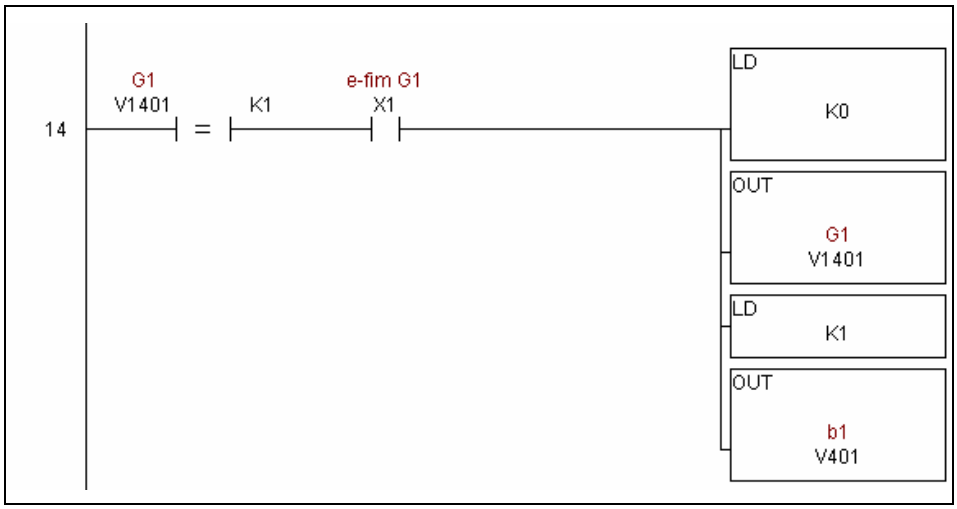

Figura 3.34 - Representação Ladder Subsistema G_1 2ª Parte

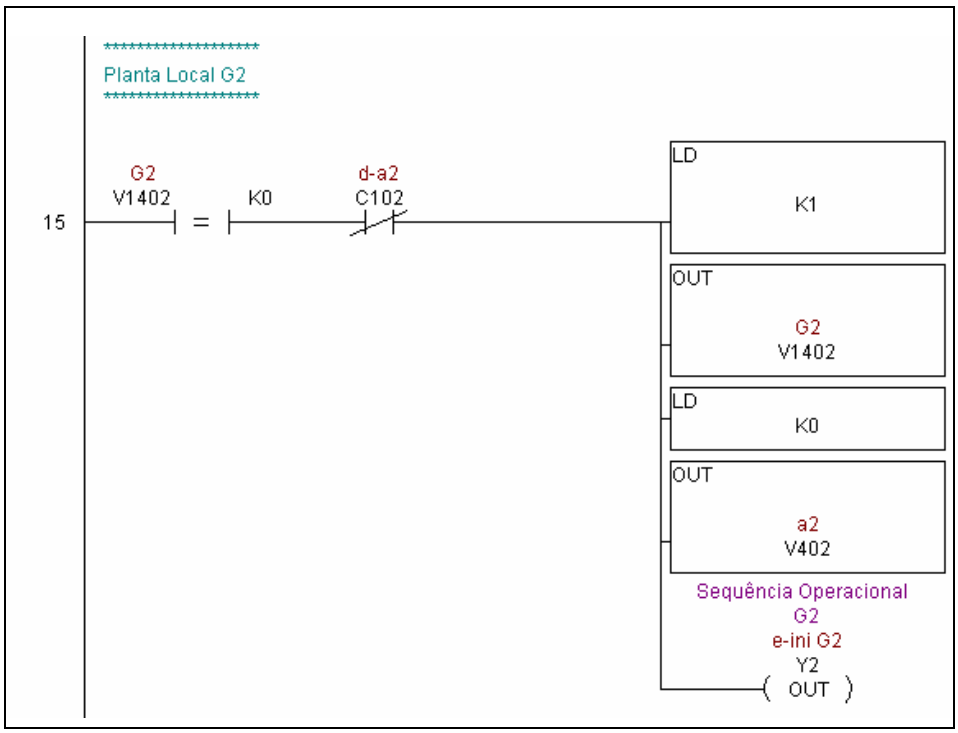

Figura 3.35 - Representação Ladder Subsistema G_2 1ª Parte

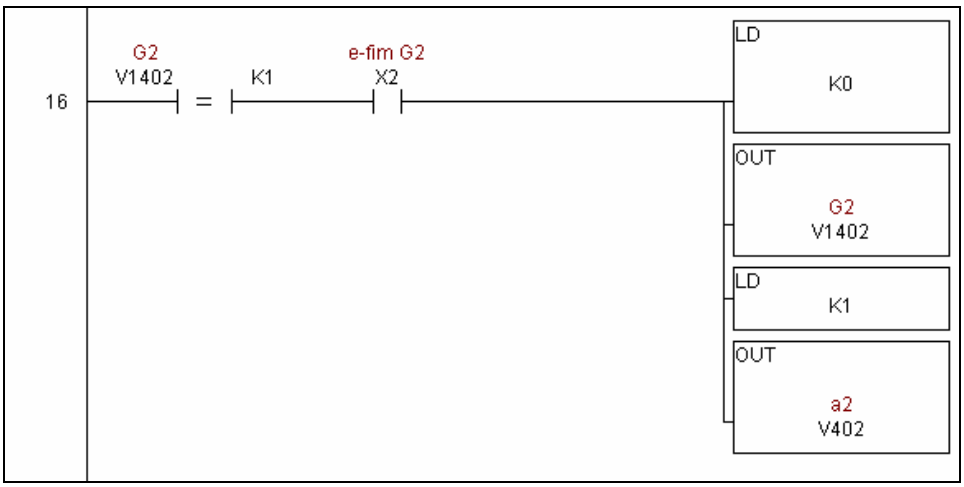

Figura 3.36- Representação Ladder Subsistema G_2 2ª Parte

3.5 Análise dos resultados

Na primeira forma de implementação, descrita no Capítulo 3.3, o autor utiliza-se da variável booleana (variável de bit) para a codificação dos estados e de todos os eventos dos supervisores e do sistema produto. Pode-se então estabelecer o número de variáveis total

utilizado na programação da estrutura de controle, supervisores modulares e sistema produto. O número de sinalizadores utilizados no programa da primeira proposta é dado: pelo número de estados de todos os supervisores n, mais o total de todos os eventos utilizados, mais o número total dos estados de todos os subsistemas n, mais o número dos eventos desabilitados por todos os supervisores modulares locais. Esse número pode ser obtido empregando-se a equação abaixo.

$$N^{o}\text{ de V} = \sum_{n} N^{o}\text{ de Estados Supervisor n} + \sum_{n} N^{o}\text{ de Eventos Subsistema n} +$$
$$\sum_{n} N^{o}\text{ de Estados Subsistema n} + \sum_{n} N^{o}\text{ de Eventos Desabilitáveis Utilizados Supervisor}$$

$$N^{o}\text{ de V} = (4) + (2+2+2) + (2+2+2) + (3) = 19$$

Na segunda forma de implementação utilizam-se variáveis de palavra. Nesta cada supervisor e cada subsistema do sistema produto ocupará uma variável de dados. Na nova proposta o número de variáveis utilizado é dado: pelo número total de supervisores correspondente aos seus estados, mais duas vezes o número total de subsistemas correspondente aos seus estados e eventos, mais o número total de eventos desabilitáveis; conforme a fórmula a seguir:

$$N^{o}\text{ de V} = N^{o}\text{ Supervisores} + (N^{o}\,\textit{Subsistema}) \times 2 +$$
$$\sum_{n} N^{o}\text{ de Eventos Desabilitáveis Utilizados Supervisor}$$

$$N^{o}\,deV = 1 + (3) \times 2 + (3) = 10$$

Os sinalizadores para as seqüências operacionais não considerados nas fórmulas devem ser incluídos no cálculo final de ambas, pois estas compõem igualmente as duas formas de implementação, totalizando assim as variáveis, Bit's mais Word's.

Utilizando as duas fórmulas apresentadas, faz-se novamente uma comparação com todos os supervisores da MPS, tabela 3.4, com objetivo de visualizar o distanciamento das variáveis num sistema completo.

<u>Proposta 1 MPS:</u>

$$N^{o}\text{ de V} = \sum_{n} N^{o}\text{ de Estados Supervisor n} + \sum_{n} N^{o}\text{ de Eventos Subsistema n} +$$
$$\sum_{n} N^{o}\text{ de Estados Subsistema n} + \sum_{n} N^{o}\text{ de Eventos Desabilitáveis Utilizados Supervisor}$$

$$N^{o}\text{ de V} = (2+2+2+2+2) + (4+4+4) + (2+2+2+2+2) + (2+2+2+2+2) +$$
$$(1+2+2+2+2+3+3+3) = 60$$

Proposta 2 MPS:

$$N^{o}\text{ de V} = N^{o}\text{ Supervisores} + (N^{o}\,Subsistema)\times 2 + \sum_{n} N^{o}\text{ de Eventos Desabilitáveis Utilizados Supervisor}$$

$$N^{o}\,deV = (1+1+1+1+1+1+1+1)+(1+1+1+1+1)\times 2+(1+2+2+2+2+3+3+3) = 36$$

Tabela 3.4: - Relação do Número de Estados e Eventos

Supervisores	Estados	Eventos Desabilitados
SLa	2	1
SLb_1	2	2
SLb_2	2	2
SLb_3	2	2
SLb_4	2	2
SLc_1	4	3
SLc_2	4	3
SLc_3	4	3
Subsistemas	Estados	Eventos
G_0	2	2
G_1	2	2
G_2	2	2
G_3	2	2
G_4	2	2

3.6 Comentários finais

Este capítulo mostra na prática o uso do método de Controle Supervisor Modular Local. É apresentada a concepção da estrutura física em uma distribuição vertical. Nesta distribuição o sistema de controle, supervisores modulares e sistema produto, pertencem a um sistema de processamento lógico e as seqüências operacionais, programas das diversas células, pertencem, cada uma, a outro controlador lógico programável.

O número crescente de estados e eventos de um sistema com maior número de células resulta na explosão de estados observados pelos supervisores (*Balemi et al*., 1993) (*Brandin*, 1996). A nova implementação busca adequar a estrutura á capacidade e manipulação de variáveis, uma vez que a teoria de autômatos empregada no controle da manufatura é um sistema determinístico, em que seus estados são unitários na evolução de eventos. Procurou-se através de um exemplo real (MPS) comparar o número de variáveis utilizando as duas formas de implementação. Mesmo observando o distanciamento, este não foi maior, pois os supervisores e os subsistemas desse exemplo apresentam um número reduzido de estados e eventos. A forma de implementação proposta por variável de inteiros faz com que o programa seja incapaz de ativar mais de um estado ou gerar mais de um evento simultâneo em um mesmo autômato.

A obtenção de fórmulas que possam expressar o número de variáveis a serem utilizadas em um projeto de controle torna-se muito importante na requisição da capacidade dos dispositivos. O projetista pode, antecipadamente, saber quais são as melhores ferramentas para seu desenvolvimento.

Capítulo 4

Proposta de Implementação da Estrutura de Controle Modular Local em Microcontrolador

4.1 Introdução

Este capítulo aponta o emprego de microcontroladores nos níveis de controle (Supervisores Modulares e Sistema Produto), pois apresentam os mesmos recursos de um Controlador Lógico Programável (CLP) para integrar o nível das seqüências operacionais, tais como velocidade de processamento, número similar de memória de dados, memória de programa e D*rives* de comunicação de redes industriais; não possuindo o custo de conexões elétricas (bornes) e níveis de sinais de entradas e saídas (24V, relés, proteções, etc.) preparados para a montagem de chão de fábrica.

Neste capítulo é utilizado o microcontrolador, dispositivo de baixo custo, para gerenciar o sistema de controle físico, mais precisamente os níveis de supervisores modulares e o sistema produto (figura 4.1), numa distribuição vertical, em que as seqüências operacionais já estão implementadas nos seus respectivos controladores lógicos programáveis. Com esse estudo busca-se verificar as particularidades do uso de um dispositivo de processamento seqüencial e ação instantânea na execução de tarefas, buscando na forma de escrita do código de controle respeitar questões propostas pela Teoria de Controle Supervisório Modular Local, não permitindo que tal escrita interfira na modularidade do controle e nem na evolução espontânea dos subsistemas. O uso de uma linguagem padrão para os microcontroladores, linguagem C ANSI, beneficia o desenvolvimento e a aquisição de

qualquer dispositivo microcontrolado, reaproveitando também o código já escrito para outros desenvolvimentos de controle.

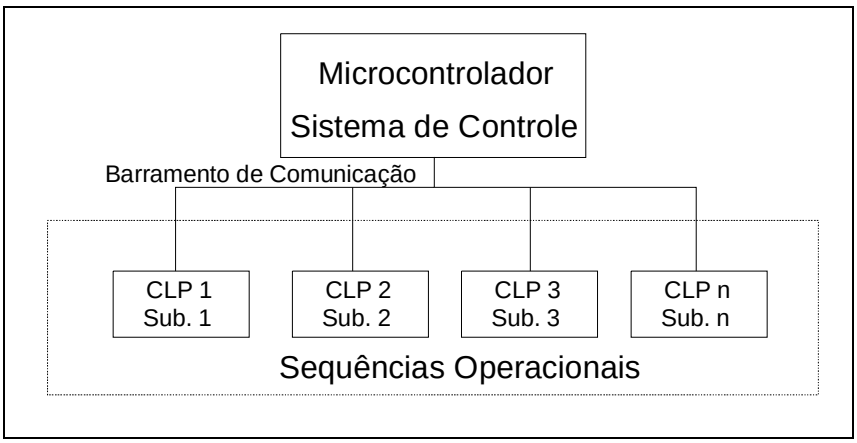

Figura 4.1 – Estrutura Física de Controle.

4.2 Descrição e comparação dos dispositivos empregados

4.2.1 Microcontroladores

O microcontrolador é considerado como um computador de um único chip, pois difere do microprocessador por possuir muitas funcionalidades embutidas, tais como memória de dados e programa, protocolo de comunicação, conversores de sinal, temporizadores, contadores e outros. Os microcontroladores, cada qual com suas características particulares, são projetados segundo a necessidade da indústria de desenvolvimento, em lógica e processamento, para desenvolver seu *hardware* segundo sua aplicação. Os microcontroladores são utilizados em eletrodomésticos, alarmes, celulares, brinquedos, entre outros. Os CLP´s, dispositivos destinados ao controle fabril, podem possuir um microcontrolador para gerenciar o seu sistema.

Segundo Lima & Rosa (2001), o conhecimento do microcontrolador é um elemento indispensável para o engenheiro eletricista, eletrônico ou ainda para o técnico de nível médio da área, em função de sua versatilidade e da enorme aplicação.

4.2.2 Linguagem C ANSI

Para programação de um microcontrolador é necessário conhecer as instruções do fabricante, ou seja, cada microcontrolador possui sua própria linguagem, *Assembly*, e uma mudança de dispositivo para outro fabricante exige que o programador reescreva todo o código mudando também a lógica de pensamento na elaboração de rotinas e na escrita de operações matemáticas, pois cada qual utiliza funções específicas e formas de tratamento distintos na maipulação de variáveis. Como forma de universalizar a programação dos microcontroladores muitas empresas de software desenvolvem compiladores em C com objetivo de reuso do mesmo código, melhor estruturação e compreensão do programa pelo emprego de uma linguagem de mais alto nível. A linguagem C não é somente utilizada em microcontroladores, é também empregada nos diversos sistemas embarcados, tais como IHM´s, sistemas operacionais e programação de dispositivos microprocessados. Esses compiladores traduzem a linguagem programada para a linguagem do dispositivo.

4.2.3 Controlador Lógico Programável

Os controladores Lógicos Programáveis são projetados para controlar dispositivos industriais, seus sinais elétricos estão em conformidade com os sinais de sensores, atuadores; possuindo *drives* de corrente, tensão, motores de passo, *enconders*, interface analógica e placas de expansão segundo a necessidade. Além disso, seu projeto físico prevê a fácil instalação em painel e ligação de seus sinais ao ambiente industrial, agilizando o desenvolvimento e manutenção. Os CLP’s priorizam também questões de segurança, como a transferência de novas rotinas de programas no processamento contínuo.

Historicamente, os Controladores Lógicos foram desenvolvidos para substituir as lógicas de controles a relês ou lógicas pneumáticas, proporcionado redução do painel de controle, custo de implementação e manutenção, maior potencialidade de controle e expansão. Os CLP´s no começo de sua utilização eram programáveis no próprio dispositivo, pois ainda eram resumidos a controles simples e de pouca adaptação, por isso optou-se por linguagens que os próprios técnicos e operadores pudessem programá-los, com uma estrutura lógica não muito diferente do que estavam habituados.

A Linguagem Ladder foi a primeira linguagem criada para programação de CLP's, por ser uma linguagem gráfica baseada em símbolos, igual aos encontrados nos esquemas elétricos de relés e contatores. Nos Estados Unidos foi a mais utilizada, já na Europa outra linguagem teve maior reconhecimento. A linguagem chamada de Lista de Instruções baseia-se na estrutura semelhante ao *Assembly* e junto com a Ladder compõe as chamadas linguagens básicas. A norma IEC 61131-3 define cinco Linguagens de Programação: Linguagem Ladder, Lista de Instruções, Texto estruturado, Diagrama de Blocos de Função, Diagrama Funcional Seqüencial. Cada uma com sua particularidade, aplicação e preferência entre os integradores. Existem também alguns fabricantes de PLC's que disponibilizam programação em Linguagem C e BASIC.

4.2.4 Diferença de Processamento Entre Duas Arquiteturas

A leitura das entradas, a ativação das saídas e o preenchimento de valores das memórias internas diferem nas duas arquiteturas. O microcontrolador utiliza-se de processamento seqüencial e ação imediata em cada lógica programada e os Controladores Lógicos fazem por varredura de tempo (*scan cycle*) sua execução (figura 4.2). Estas características mudam a forma de raciocínio lógico, tal qual o desenvolvimento do programa. Como exemplos podem-se considerar o seguinte caso: se no microcontrolador precisa-se ativar 3 saídas (Y0,Y1e Y2), cada uma será ativada seqüencialmente segundo a velocidade do *Clock* do processador, tendo assim um pequeno atraso de Y1 para Y0, de Y2 para Y1 e Y0. No caso do CLP as saídas serão ativadas todos ao mesmo tempo, pois primeiramente é lido todo o código escrito e depois processado sua ação. Na prática os controladores lógicos programáveis em termos de tempo de execução de processamento são mais lentos que os microcontroladores, uma vez que os Controladores Lógicos Programáveis são fabricados com estes dispositivos. Quando programa-se em microcontrolador utiliza-se do processamento puro desse dispositivo e quando programa-se em um CLP está programando para um sistema operacional, que processa todas as informações segundo o ciclo de execução da figura 4.2.

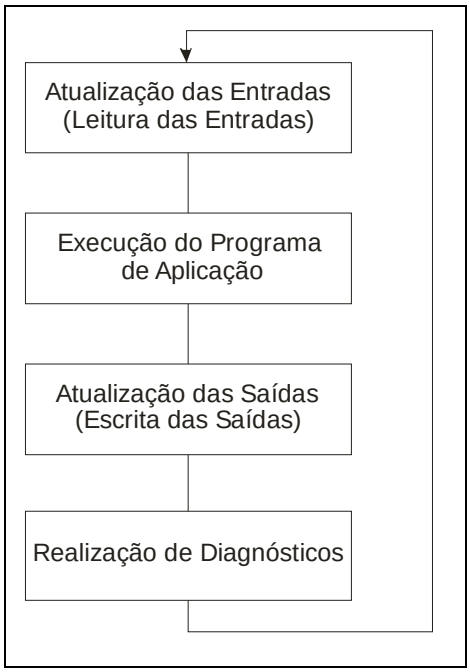

Figura 4.2 – Ciclo de Execução Básico de um CLP.

4.3 Geração Automática do Código de Controle para Microcontroladores

4.3.1 Geração Automática do Código de Controle Segundo Modelo de Implementação em Três Níveis

O modelo em três níveis, descrito no capítulo 3, proposto por Queiroz *et al.* (2001) e Queiroz e Cury (2002) e a distribuição vertical, segundo Vieira e Cury (2004) é conservado nesta utilização com microcontroladores. Conforme descrito anteriormente, a forma de processamento do dispositivo empregado e sua linguagem de programação acrescenta mudanças à lógica de implementação do código de controle. Para tanto este desenvolvimento busca adequar e adaptar a linguagem e o processamento dos microcontroladores à Teoria de Controle Supervisório Modular Local; visando garantir aspectos tais como a espontaneidade dos eventos gerados pelos subsistemas e a supervisão imediata dos supervisores modulares.

Com objetivo de verificar a atuação simultânea dos supervisores modulares e possíveis erros gerados na comunicação com os subsistemas, bem como observar o número de ocorrências dos subsistemas, G_0, G_1 e G_2, verificando assim a interferência do

processamento na evolução dos subsistemas, optou-se pela escolha de dois supervisores modulares simples do sistema MPS, SLb_1 e SLb_2.

No estudo de caso o nível de Supervisores Modulares e o nível Sistema Produto são implementados no microcontrolador PIC16F877A e as seqüências operacionais no CLP DirectLogic06-DD. A comunicação entre os dois sistemas é através de uma comunicação serial simples, código ASCII, simplificando a integração. A figura 4.3 exemplifica em fluxograma a ordem de execução da estrutura em três níveis, sendo esta chamada e implementada na rotina principal do programa. O nível Supervisor Modular e Sistema Produto são implementados em três rotinas secundárias que são chamadas pela rotina principal. Segundo o fluxograma, primeiramente todas as variáveis que representam os estados dos autômatos são zerados, significando o estado inicial.

Na rotina secundária Supervisores Modulares é verificada a ocorrência de eventos dos subsistemas observados, atualizando o estado dos autômatos supervisores modulares. Após o fim dessa execução são chamados pela rotina principal à rotina Eventos Desabilitáveis que desabilita os eventos controláveis impróprios para o bom funcionamento da planta, segundo o estado dos supervisores. Na rotina Subsistemas estão às abstrações do funcionamento espontâneo das diversas células que podem ser inibidos segundo a desabilitação dos eventos controláveis. Nota-se a dependência seqüencial de cada rotina secundária com as demais rotinas, fechando após a rotina Estados Iniciais um ciclo de execução que se repete indefinidamente, como pode ser visto na figura 4.3. Para cada rotina chamada existe um programa específico executado, com início e fim.

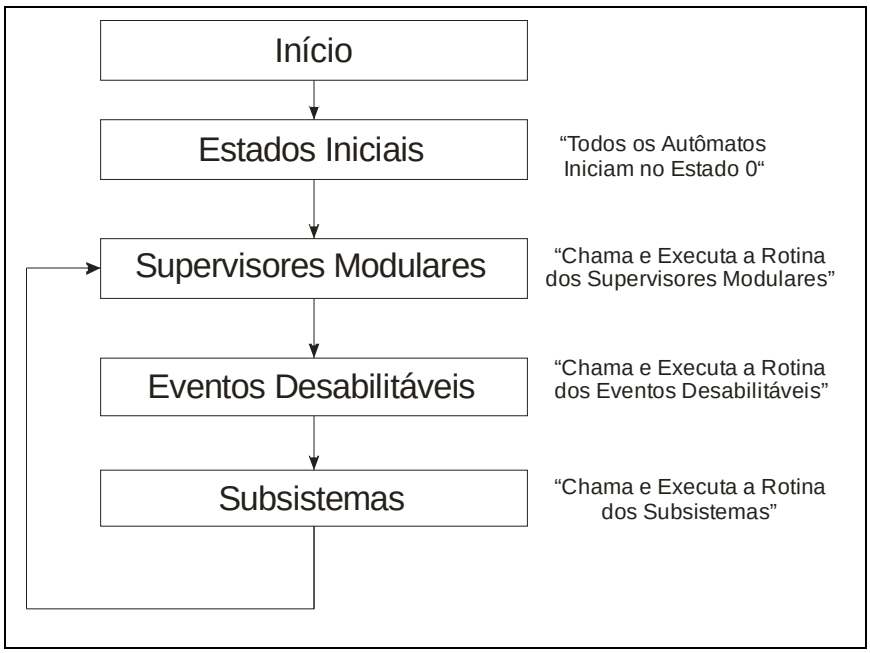

Figura 4.3 - Fluxograma Rotina Principal "Main"

A evolução em autômatos para outro estado $d : Q X S$ ® é uma função dependente do estado atual e do evento ocorrido. No código em C ANSI, representado nos fluxogramas abaixo, nota-se diretamente essa condição. Primeiramente é verificado qual é o estado do autômato para depois verificar se houve a ocorrência de algum evento associado a ele, atualizando assim para outro estado.

Nota-se no fluxograma dos Supervisores Modulares, figura 4.4 e figura 4.5, a dependência de evolução de um novo estado ao estado atual e ao evento a ele atribuído, sendo executados por duas verificações. Exemplo: Se $SLb_1 = 0$ (estado) e $eventos = a_0$ (evento) então $SLb_1 = 1$ (novo estado).

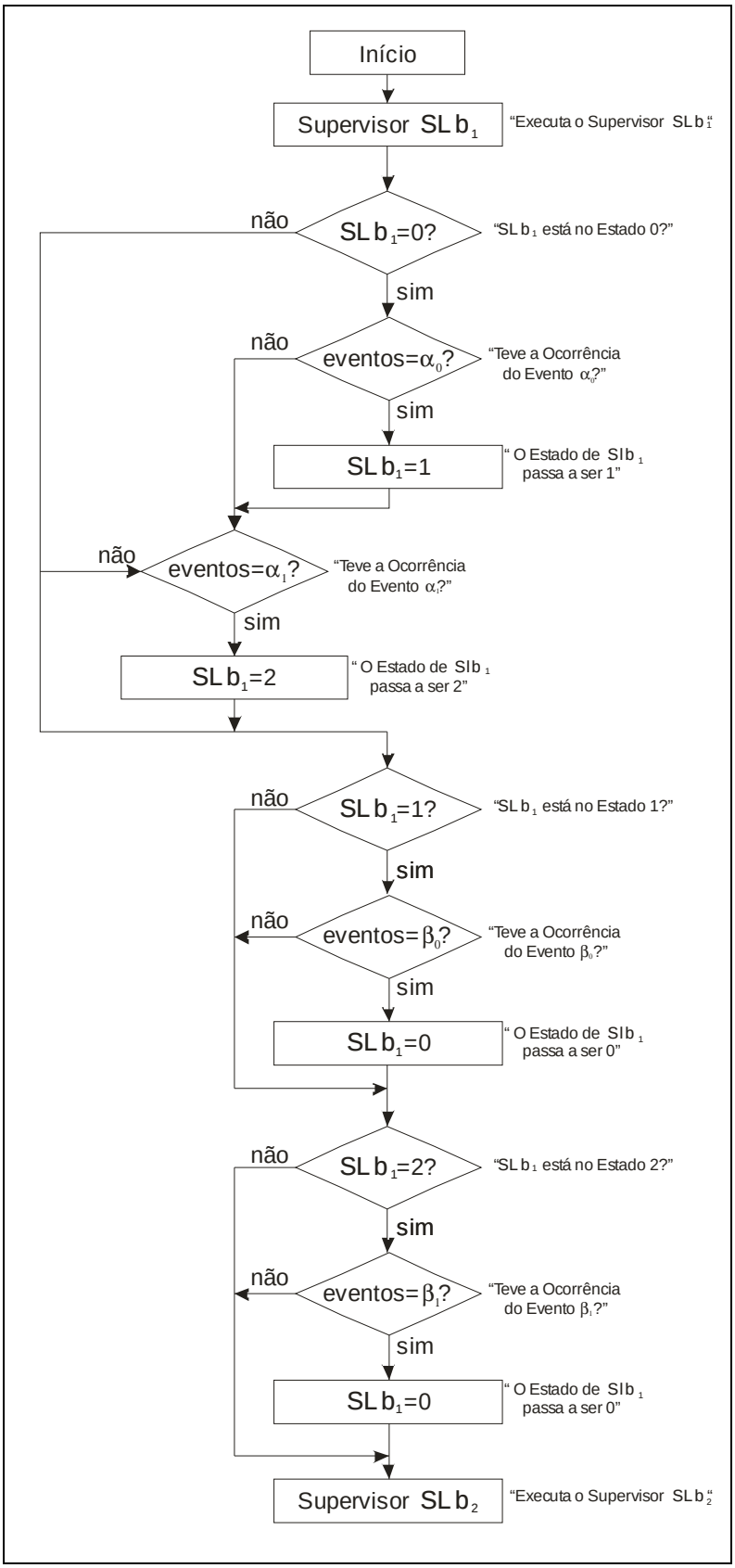

Figura 4.4 - Fluxograma Rotina Supervisores Modulares 1ª Parte

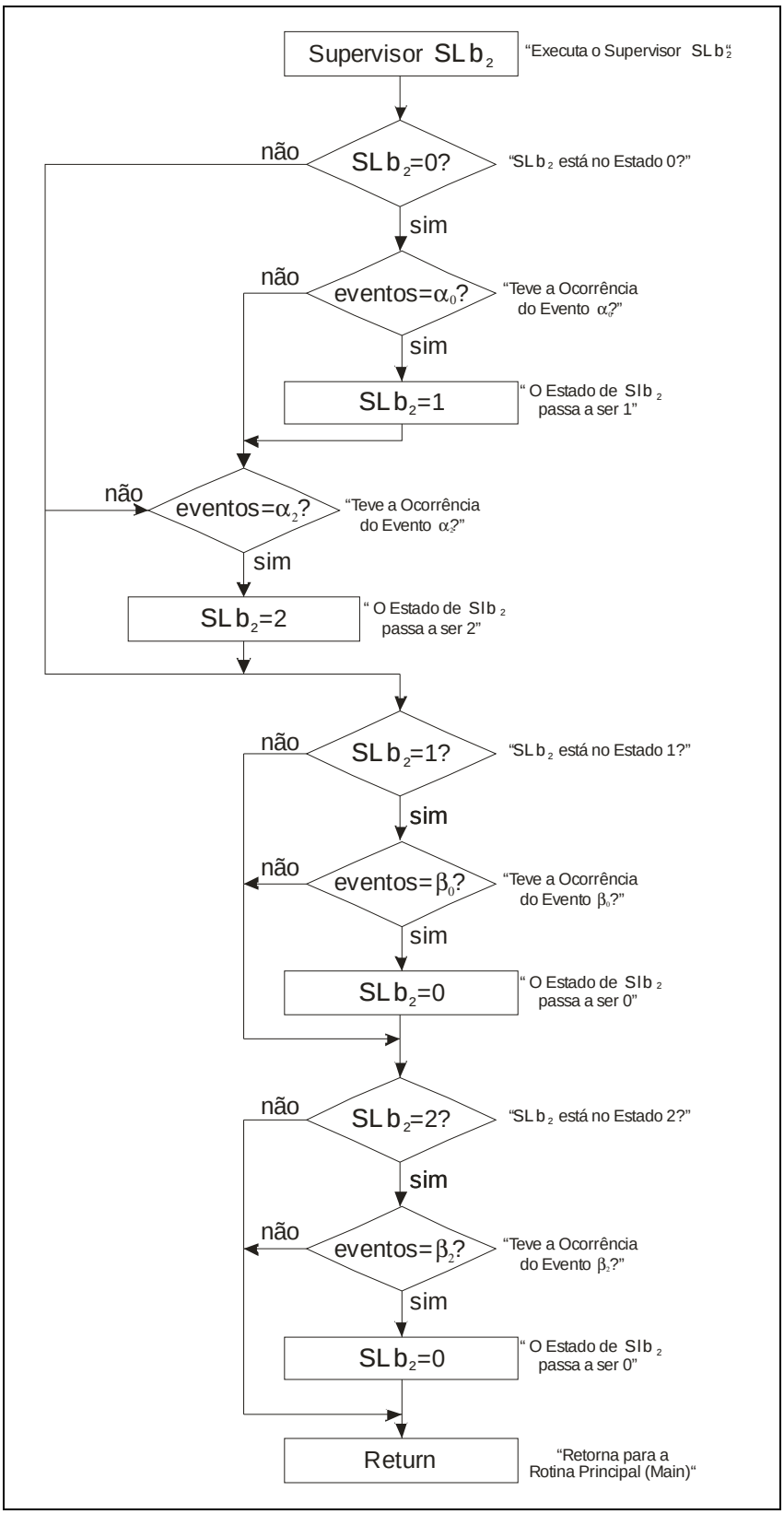

Figura 4.5 - Fluxograma Rotina Supervisores Modulares 2ª Parte

Em cada estado de um supervisor existe um conjunto de eventos desabilitados, figura 4.6, esta rotina verifica o estado dos supervisores desabilitando os eventos controláveis associados a eles.

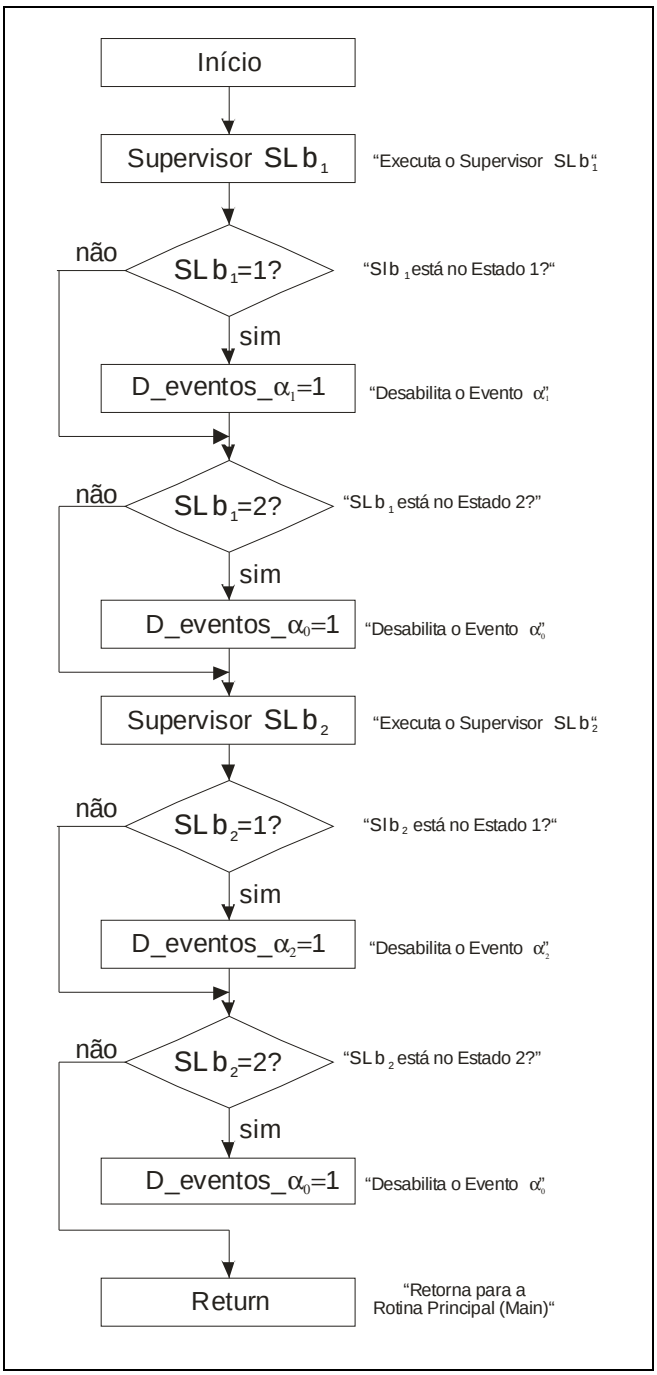

Figura 4.6 - Fluxograma Rotina Eventos Desabilitáveis

A Teoria de Controle Supervisório Modular Local permite a espontaneidade na evolução de eventos, estando a planta livre para evoluir e seguir qualquer rota de estados,

sendo inibida e controlada somente pelos supervisores locais. Durante o desenvolvimento do estudo de caso e a execução do programa em C ANSI, verificou-se que a escrita em uma programação seqüencial cria uma rota específica e cíclica para a execução desses subsistemas, limitando planta. Acompanhando a figura 4.8 e a figura 4.9, pode-se entender melhor descrição a seguir: como o programa dos subsistemas é executado segundo sua ordem de escrita na linguagem, o subsistema G_0 sempre será o primeiro e único subsistema a evoluir, pois a cada geração de evento e atualização de seu estado a rotina Subsistema deve ser finalizada e suas ações informadas para os supervisores. Esta finalização se faz necessária para não permitir a dessincronização dos supervisores com os subsistemas.

Sendo assim, após a evolução de G_0 por a_0, os supervisores SLb_1 e SLb_2 impedem a ocorrência dos eventos a_1 e a_2, porque estes estão respectivamente no estado "1". Para sair dessa condição de desabilitação dos subsistemas G_1 e G_2 ambos os supervisores devem estar no estado "0", o que ocorrerá após a finalização do subsistema G_0 por b_0, mas após isso ocorrer G_0 será novamente o primeiro subsistema a ser atendido, pois na escrita do programa o subsistema G_0 é o primeiro.

Observa-se hipoteticamente também que após o cumprimento de G_0 e a não evolução de a_0 novamente, o próximo evento prioritário no sistema seria a_1, isto porque G_1 está ordenadamente após G_0. Um outro ponto a ser levantado, se não houvesse a limitação da questão anterior, seria a impossibilidade de evolução do subsistema G_0 após a execução dos subsistemas G_1 e G_2. Para a ocorrência de a_0 os eventos b_1 e b_2 deveriam ocorrer e ser tratados simultaneamente pelo programa. Nesses moldes isso seria impossível de ocorrer.

Para resolução destas questões foi criada uma variável aleatória, cujo objetivo é distribuir aleatoriamente a execução dos subsistemas, retirando a mesma ordem de execução pela escrita do programa, esta forma garante uma rota espontânea, com início espontâneo, conforme pode ser visto no fluxograma da figura 4.7. No início da rotina Subsistemas é sorteado um número entre 0 a 100 e este número armazenado na variável aleatória será a base de escolha para a seleção do subsistema.

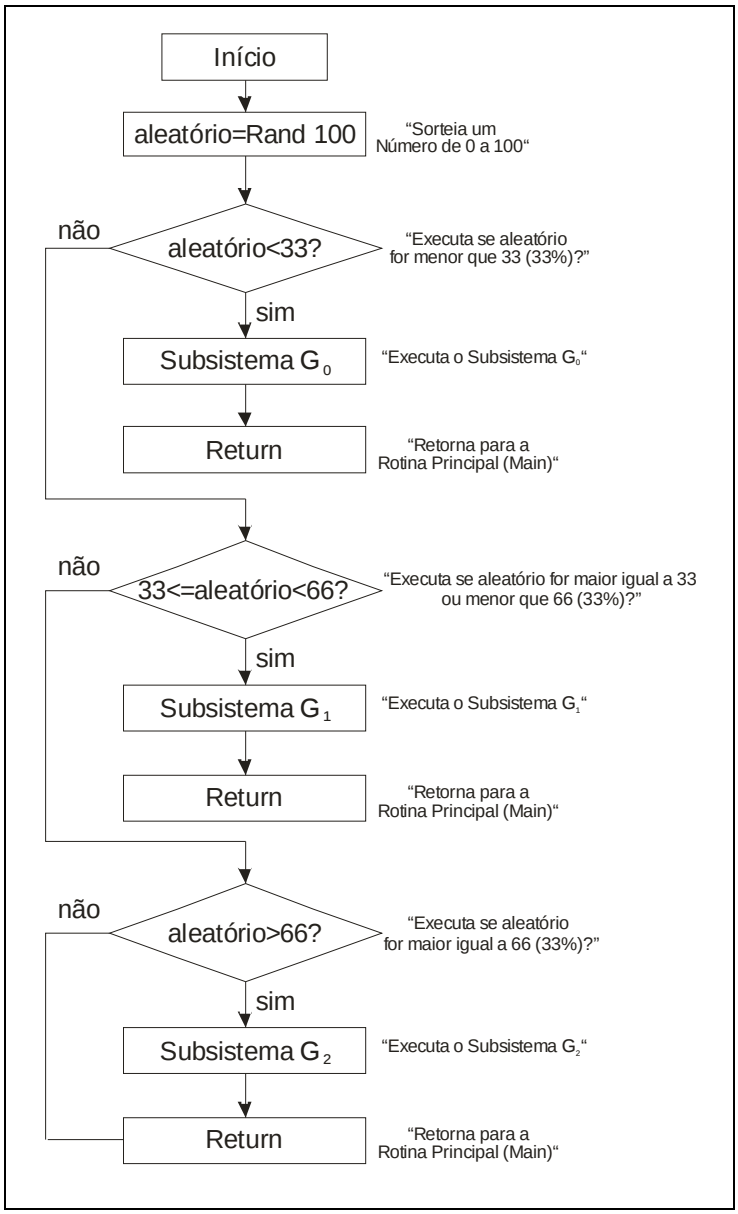

Figura 4.7 - Fluxograma Rotina Distribuição Subsistemas

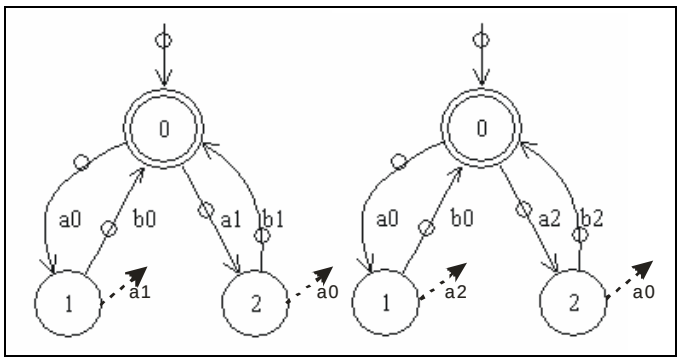

Figura 4.8 – Supervisores Modulares SLb_1 e SLb_2

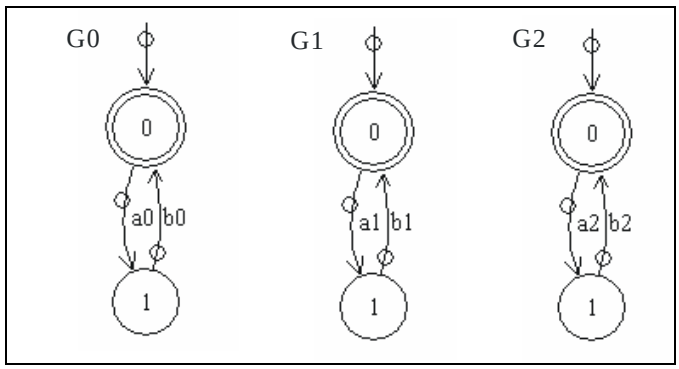

Figura 4.9 – Subsistemas

Nos Fluxogramas das rotinas dos Subsistemas a seguir, todos os eventos controláveis estão vinculados à condição de desabilitação e quando não desabilitados geram o próprio evento, o novo estado e o comando para o início das seqüências operacionais (figura 4.10, figura 4.11, figura 4.12).

Para a evolução de um evento não controlável, a condição é satisfeita através da finalização da seqüência operacional que são passados para o nível de controle por comunicação serial.

Em cada condição satisfeita, seja o início ou o fim de uma seqüência operacional a rotina Subsistema é finalizado mediante o comando *return*, cujo objetivo é informar imediatamente para os supervisores sua evolução.

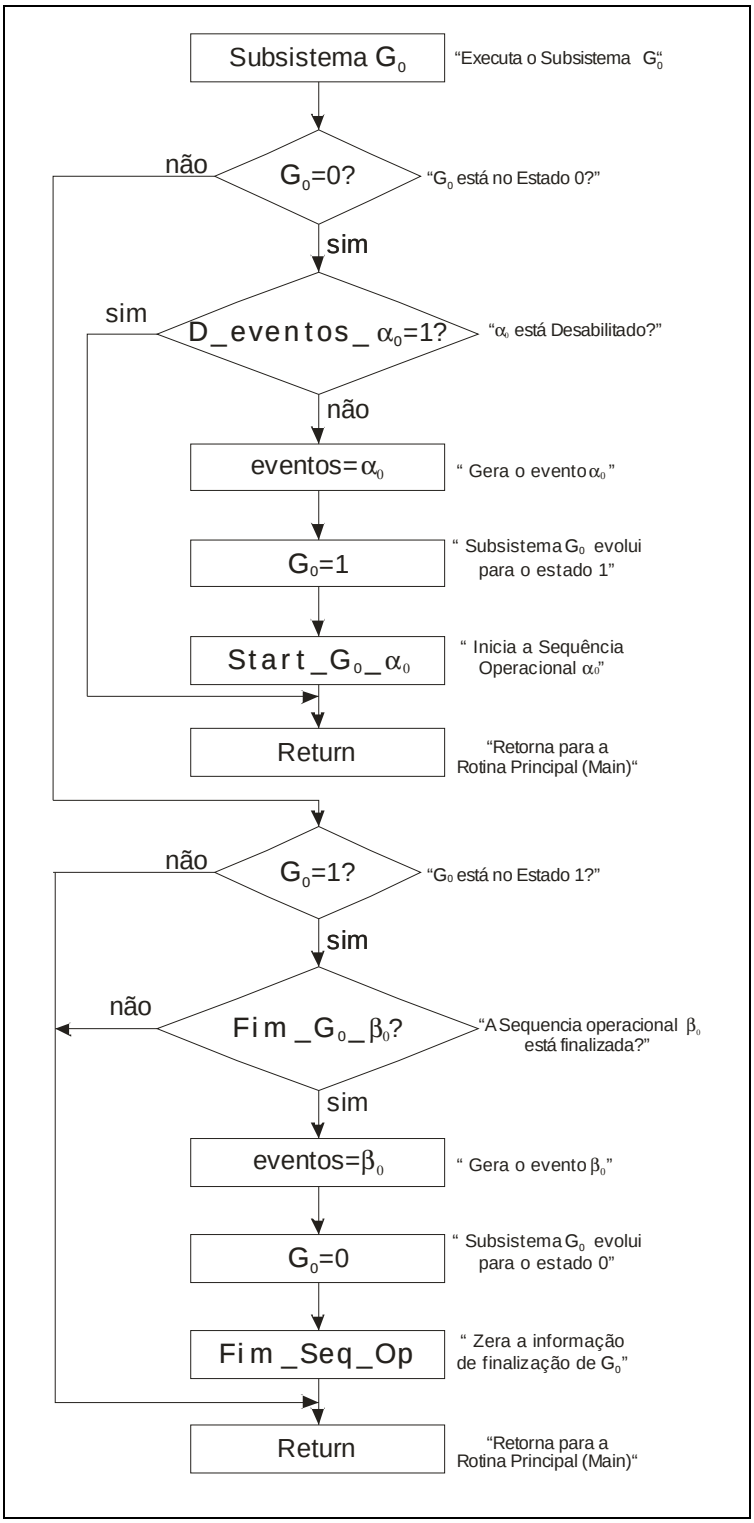

Figura 4.10 - Fluxograma Rotina Subsistema G_0

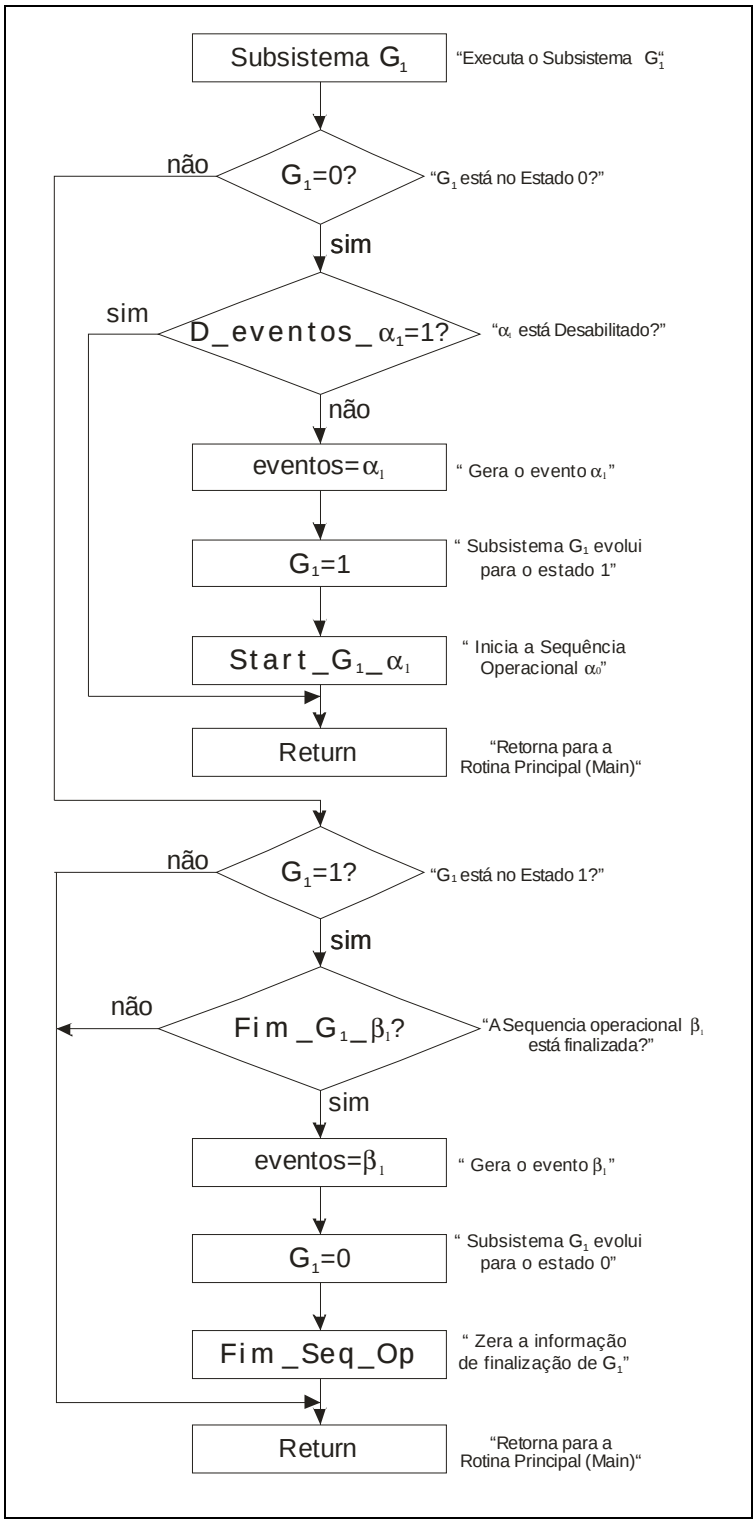

Figura 4.11 - Fluxograma Rotina Subsistema G_1

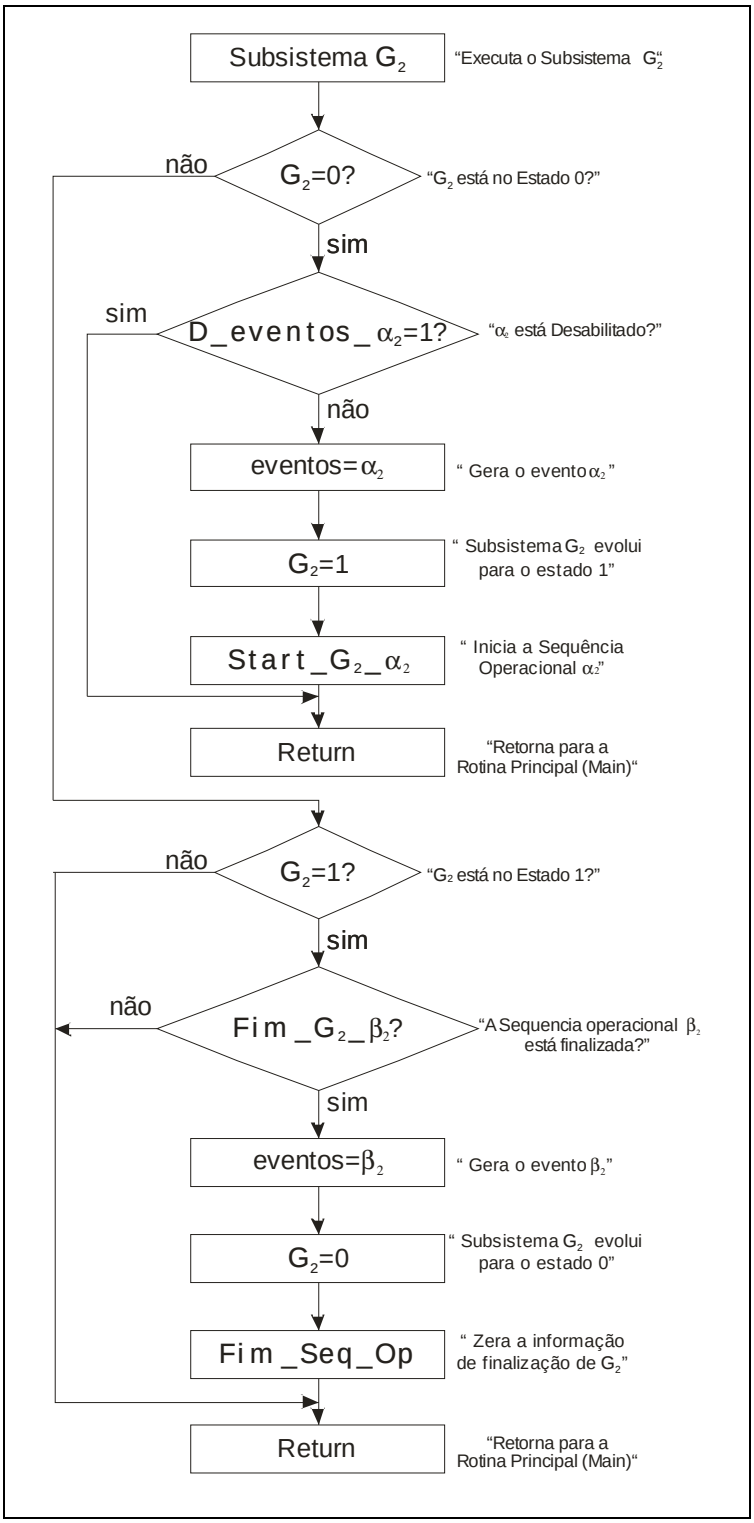

Figura 4.12 - Fluxograma Rotina Subsistema G_2

A seguir mostra-se a transcrição direta dos autômatos em Linguagem C. Pode-se através do código Grail fazer uma implementação direta baseada na estrutura de máquina de estados.

4.3.1.1 Transcrição dos Supervisores

Onde: X_SLb_1 e X_SLb_2 representam os estados dos Supervisores Modulares e *eventos* os eventos $a_0, b_0, a_1, b_1, a_2, b_2$ dos respectivos subsistemas.

Autômatos:

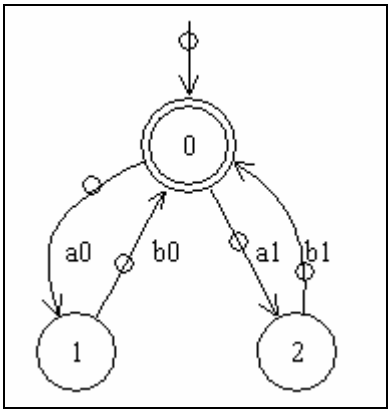

Figura 4.13 – Supervisor SLb_1

Código em TXT:

```
(START) |- 0
1 b0 0
2 b1 0
0 a0 1
0 a1 2
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_Slb1==0)
{
     if (eventos==a0)
     {
        X_Slb1=1;
     }
     if (eventos==a1)
     {
        X_Slb1=2;
     }
}
```

```
if (X_Slb1==1)
{
     if (eventos==b0)
     {
        X_Slb1=0;
     }
}
if (X_Slb1==2)
{
     if (eventos==b1)
     {
        X_Slb1=0;
     }
}
```

Autômatos:

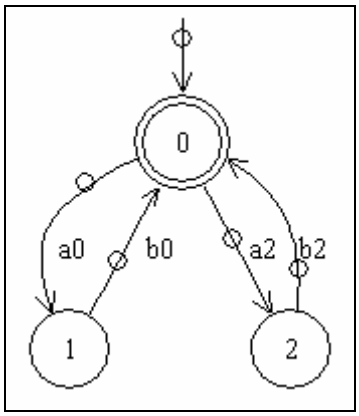

Figura 4.14 – Supervisor SLb_2

Código em TXT:

```
(START) |- 0
1 b0 0
2 b2 0
0 a0 1
0 a2 2
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_Slb2==0)
{
     if (eventos==a0)
     {
        X_Slb2=1;
     }
     if (eventos==a2)
     {
        X_Slb2=2;
     }
}
if (X_Slb2==1)
```

```
{
     if (eventos==b0)
     {
        X_Slb2=0;
     }
}
if (X_Slb2==2)
{
     if (eventos==b2)
     {
        X_Slb2=0;
     }
}
```

4.3.1.2 Transcrição dos Eventos Desabilitáveis

Onde: X_SLb_1 e X_SLb_2 representam os estados dos Supervisores Modulares e $D\_eventos\_a_0$, $D\_eventos\_a_1$, $D\_eventos\_a_2$ os eventos controláveis dos subsistemas que serão desabilitados pelos supervisores. O sinal “1” indicará para os subsistemas a desabilitação e o sinal “0” a permissão dos eventos em ocorrer.

Após o final da rotina Subsistemas, antes do chamando da rotina Supervisores Modulares, todas as variáveis de desabilitação recebem o valor “0” (Rotina *main* do anexo 1), cabendo novamente aos supervisores desabilitar os eventos segundo sua supervisão.

Autômatos:

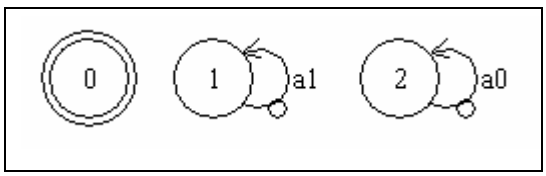

Figura 4.15 – Eventos Desabilitáveis pelo Supervisor SLb_1

Código em TXT:

```
1 a1 1
2 a0 2
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_Slb1==1)
{
     D_eventos_a1=1;
}
if (X_Slb1==2)
```

```
{
     D_eventos_a0=1;
}
```

Autômatos:

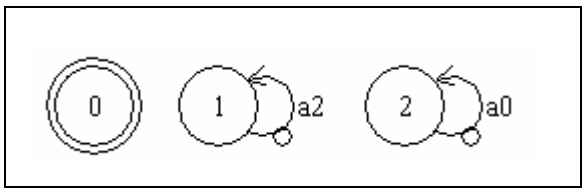

Figura 4.16 – Eventos Desabilitáveis pelo Supervisor SLb_2.

Código em TXT:

```
1 a2 1
2 a0 2
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_Slb2==1)
{
     D_eventos_a2=1;
}
if (X_Slb2==2)
{
     D_eventos_a0=1;
}
```

4.3.1.3 Transcrição dos Subsistemas

Onde: X_G_0, X_G_1, X_G_2 representam os estados dos Subsistemas, $a_0, b_0, a_1, b_1, a_2, b_2$ os eventos gerados e $D\_eventos\_a_0$, $D\_eventos\_a_1$, $D\_eventos\_a_2$ os eventos controláveis desabilitados pelos supervisores. A variável Fim_Seq_Op registra a informação de finalização de qualquer seqüência operacional que são passadas através da comunicação serial: $Fim\_G_0\_b_0$ é o registro de G_0, $Fim\_G_1\_b_1$ o registro de G_1 e $Fim\_G_2\_b_2$ o registro de G_2.

Autômatos:

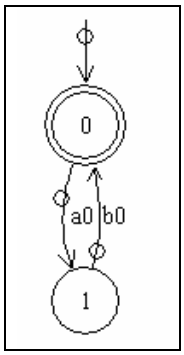

Figura 4.17. – Subsistema G_0

Código em TXT:

```
(START) |- 0
0 a0 1
1 b0 0
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_G0==0)
{
     if (D_eventos_a0!=1)
     {
        eventos=a0;
        X_G0=1;
     }
}
if (X_G0==1)
{
     if (Fim_Seq_Op==Fim_G0_b0)
     {
        eventos=b0;
        X_G0=0;
     }
}
```

Autômatos:

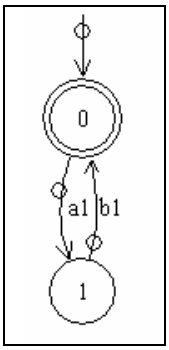

Figura 4.18 – Subsistema G_1

Código em TXT:

```
(START) |- 0
0 a1 1
1 b1 0
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_G1==0)
{
     if (D_eventos_a1!=1)
     {
        eventos=a1;
        X_G1=1;
     }
}
if (X_G1==1)
{
     if (Fim_Seq_Op==Fim_G1_b1)
     {
        eventos=b1;
        X_G1=0;
     }
}
```

Autômatos:

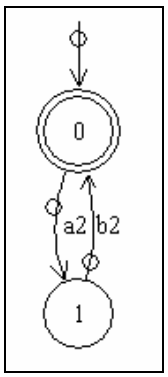

Figura 4.19 – Subsistema G_2

Código em TXT:

```
(START) |- 0
0 a2 1
1 b2 0
0 -| (FINAL)
```

Código em Linguagem C:

```
if (X_G2==0)
{
     if (D_eventos_a2!=1)
     {
```

```
            eventos=a2;
            X_G2=1;
        }
    }
    if (X_G2==1)
    {
        if (Fim_Seq_Op==Fim_G2_b2)
        {
            eventos=b2;
            X_G2=0;
        }
    }
```

4.3.2 Geração Automática do Código de Controle Segundo Nova Abordagem

A estrutura de implementação da nova abordagem transfere a importância para o sistema produto, pois ele representa o comportamento livre da planta. Esta afirmativa é justificada no acompanhamento da programação de todo o código de controle, porque o código que representa os subsistemas está estruturado na rotina principal do programa (Figura 4.20 –figura 4.20).

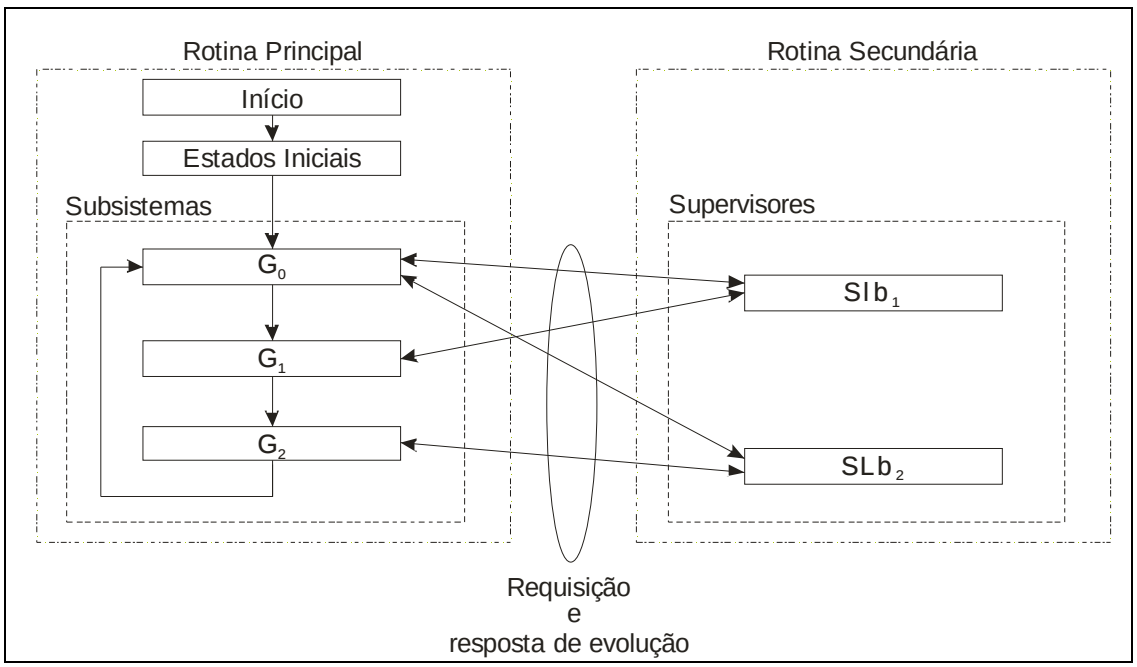

Figura 4.20 – Modelo da Nova Abordagem

Nesta abordagem o sistema produto envia os eventos espontâneos como forma de requisição e os respectivos supervisores modulares confirmam para o subsistema a possibilidade de ocorrer (figura 4.21, figura 4.22). Se houver a permissão de todos os seus

supervisores o subsistema pode evoluir e os estados dos supervisores são atualizados. Mas se algum supervisor não evoluir, o subsistema não recebe a confirmação e os supervisores permanecem em seus estados originais. Após a transição confirmada, o estado do subsistema é atualizado e o programa da seqüência operacional correspondente é iniciado.

Na proposta de implementação de controle, dois tipos modelos serão necessários: a abstração dos Subsistemas e a obtenção dos Supervisores Modulares Locais. Para compor o projeto e realizar a transcrição para linguagem de programação, o projetista deverá também relacionar quais subsistemas são afetados pelos supervisores modulares, conforme tabela 4.1. Com esse recurso não será necessário obter o conjunto de eventos desabilitados de cada supervisor.

Esta estrutura não considera os eventos desabilitados para a implementação do controle, mas são necessários os eventos de mudanças do próprio estado para o correto controle (***selfloop)*** se estes existirem na obtenção dos supervisores. A permissão dos supervisores modulares está ligada nas suas evoluções e se qualquer supervisor relacionado com o subsistema não evoluir pelo evento requisitado, o evento controlável desse subsistema será bloqueado. Vale lembrar que na forma de implementação anterior podem-se desconsiderar os ***selfloop's*** para minimizar os supervisores, pois esses não representarão nenhuma mudanças na execução da rotina Eventos Desabilitáveis.

Utilizam-se os mesmos subsistemas e supervisores da implementação do Capítulo 4.3.1.

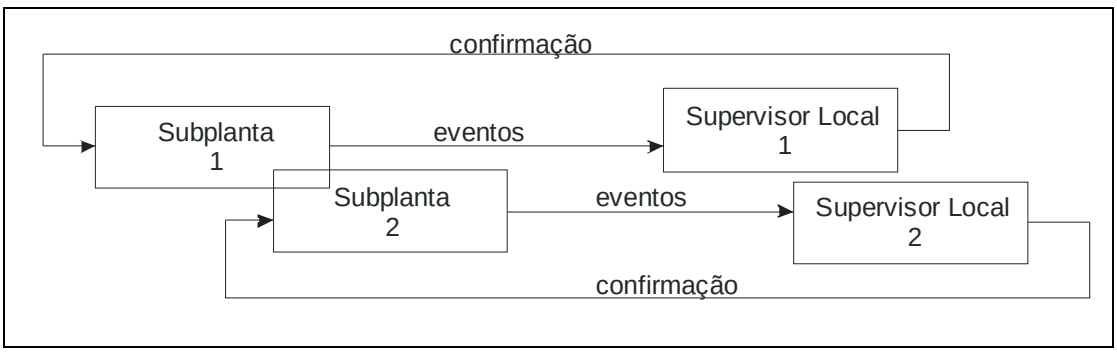

Figura 4.21 – Atuação do Supervisor Local

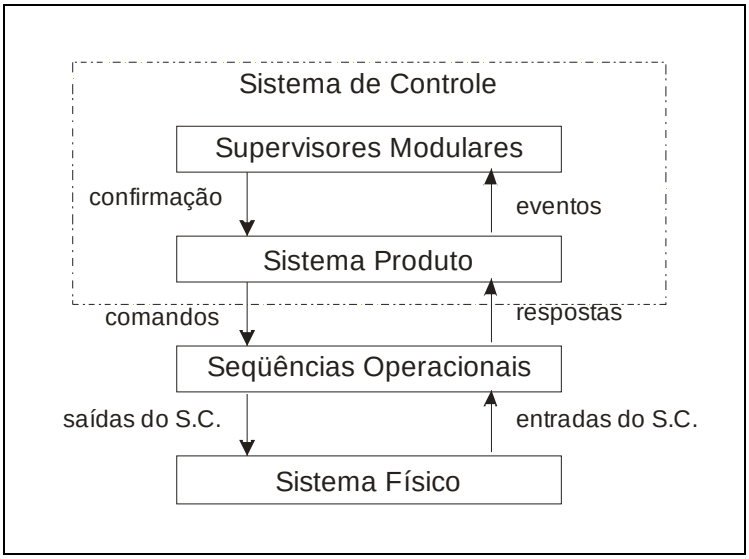

Figura 4.22 – Estrutura de Controle

Tabela 4.1 – Subsistemas Afetados pelos Supervisores

SUPERVISORES	SUBPLANTAS		
	G0	G1	G2
SLb_1	X	X	
SLb_2	X		X

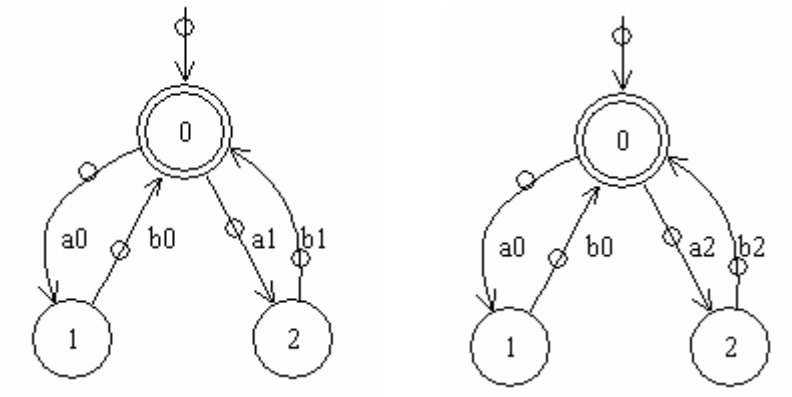

Figura 4.23 – Supervisores Modulares SLb_1 e SLb_2

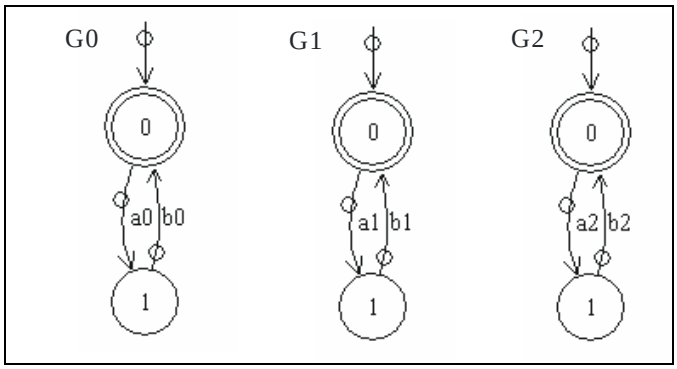

Figura 4.24 – Subsistemas

A linguagem de programação da nova proposta utiliza-se da forma de verificação de supervisão dos supervisores modulares em conjunto com os subsistemas. Nesta forma, analisam-se os subsistemas junto com os supervisores modulares e após o acompanhamento dos supervisores na evolução dos eventos atualizam-se os estados dos subsistemas e dos supervisores (figura 4.23 e figura 4.24). No momento em que o sistema é iniciado todos os autômatos estão no estado zero e qualquer análise é feito inicialmente nestes estados. Observam-se os supervisores modulares: SLb_1 observa G_0, G_1 e SLb_2 observa G_0, G_2. Se a primeira hipótese de evolução for a_0 (G_0), deve-se verificar a possibilidade de evolução nos dois supervisores e se houver atualiza-se os estados dos autômatos. Neste momento SLb_1 e SLb_2 estão no estado 1, G_0 está no estado 1, G_1 e G_2 permanecem no estado 0. Continuando a análise, escolhe-se a_1 para evoluir e verifica a possibilidade nos supervisores, como somente SLb_1 observa G_1 a análise é restrita a ele. O evento a_1 não é acompanhado pelo estado 1 do supervisor SLb_1 e por essa razão não pode ocorrer. Esta mesma observação se faz para o evento a_2 em relação ao supervisor SLb_2. Os eventos a_1 e a_2 só estarão novamente aptos a evoluir quando os supervisores também puderem através dos mesmos atualizarem seus estados. Nessa análise lança-se primeiramente uma hipótese de evolução (a_0, a_1, a_2), observa-se o acompanhamento pelos supervisores e se confirmado atualiza os estados dos autômatos. A ocorrência de um evento não-controlado (b_0, b_1, b_2) sempre provoca a atualização dos supervisores (critério de controlabilidade).

A utilização da geração e verificação do número aleatório para escolher os subsistemas é justificada pelas mesmas razões descritas no Capítulo 4.3.1, sendo essa uma

forma de garantir espontaneidade ao sistema (figura 4.25). A lógica de programação pode ser acompanhada nos fluxogramas seguintes (figura 4.26 à figura 4.30).

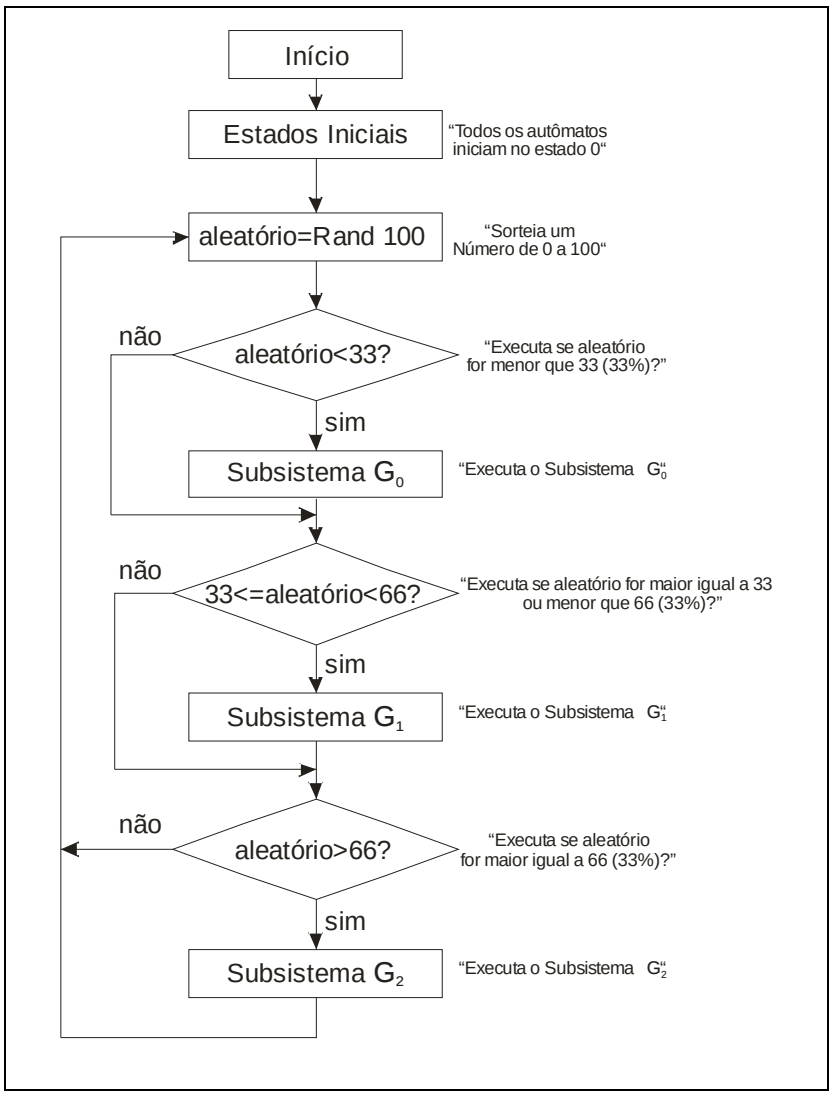

Figura 4.25 – Fluxograma Rotina Principal "main"

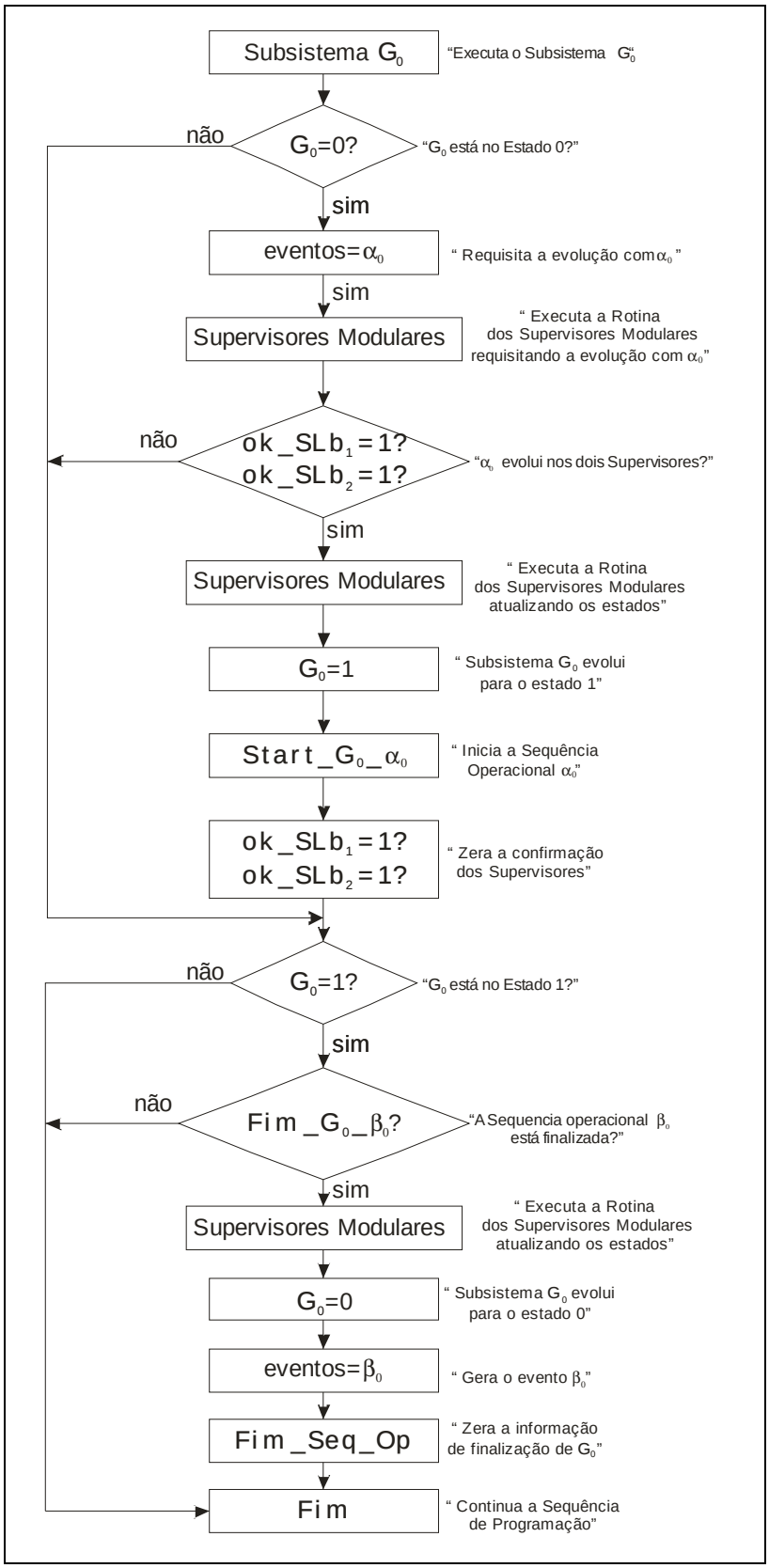

Figura 4.26 – Fluxograma Rotina Principal Subsistema

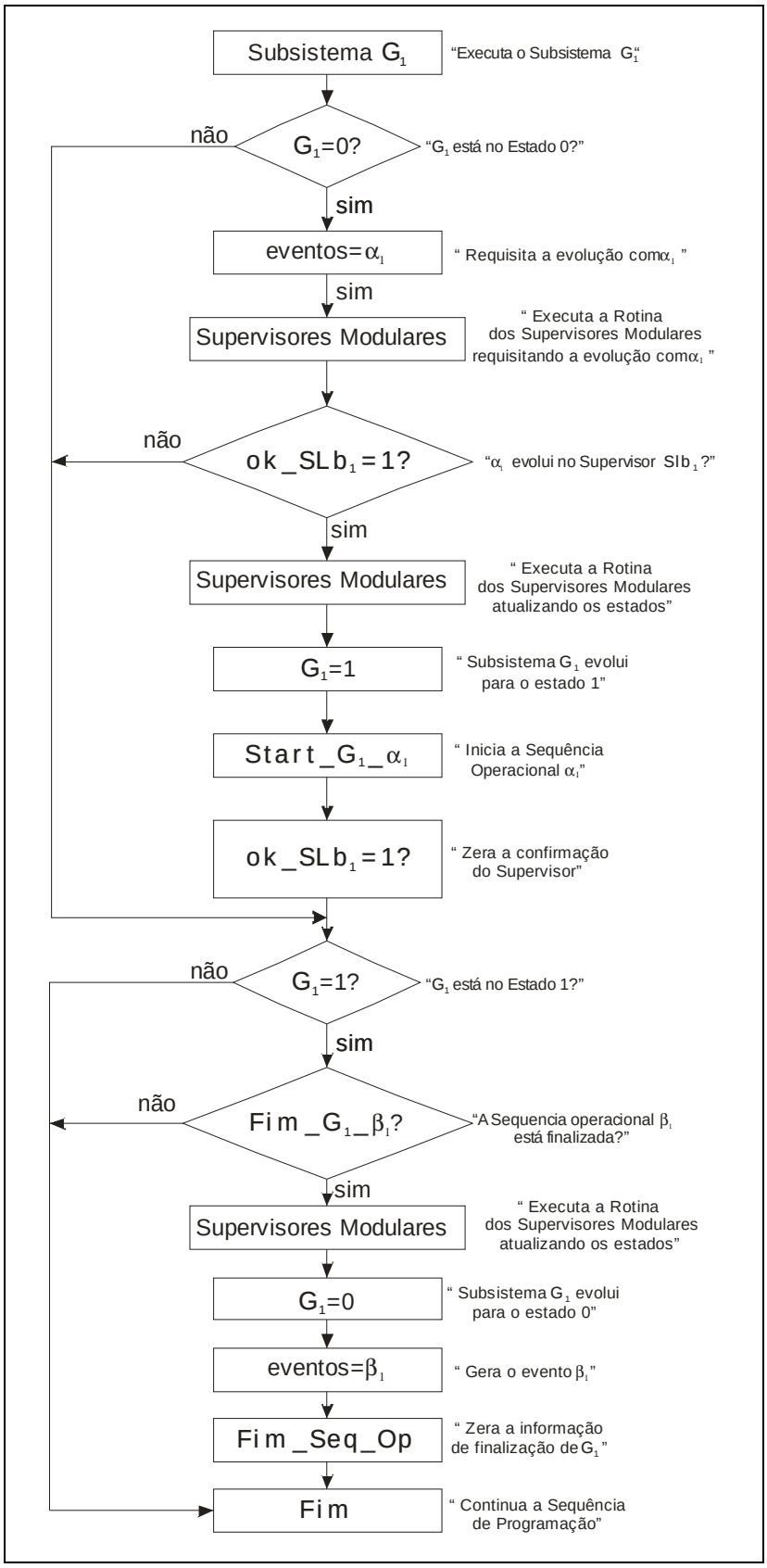

Figura 4.27 - Fluxograma Rotina Principal Subsistema G_1

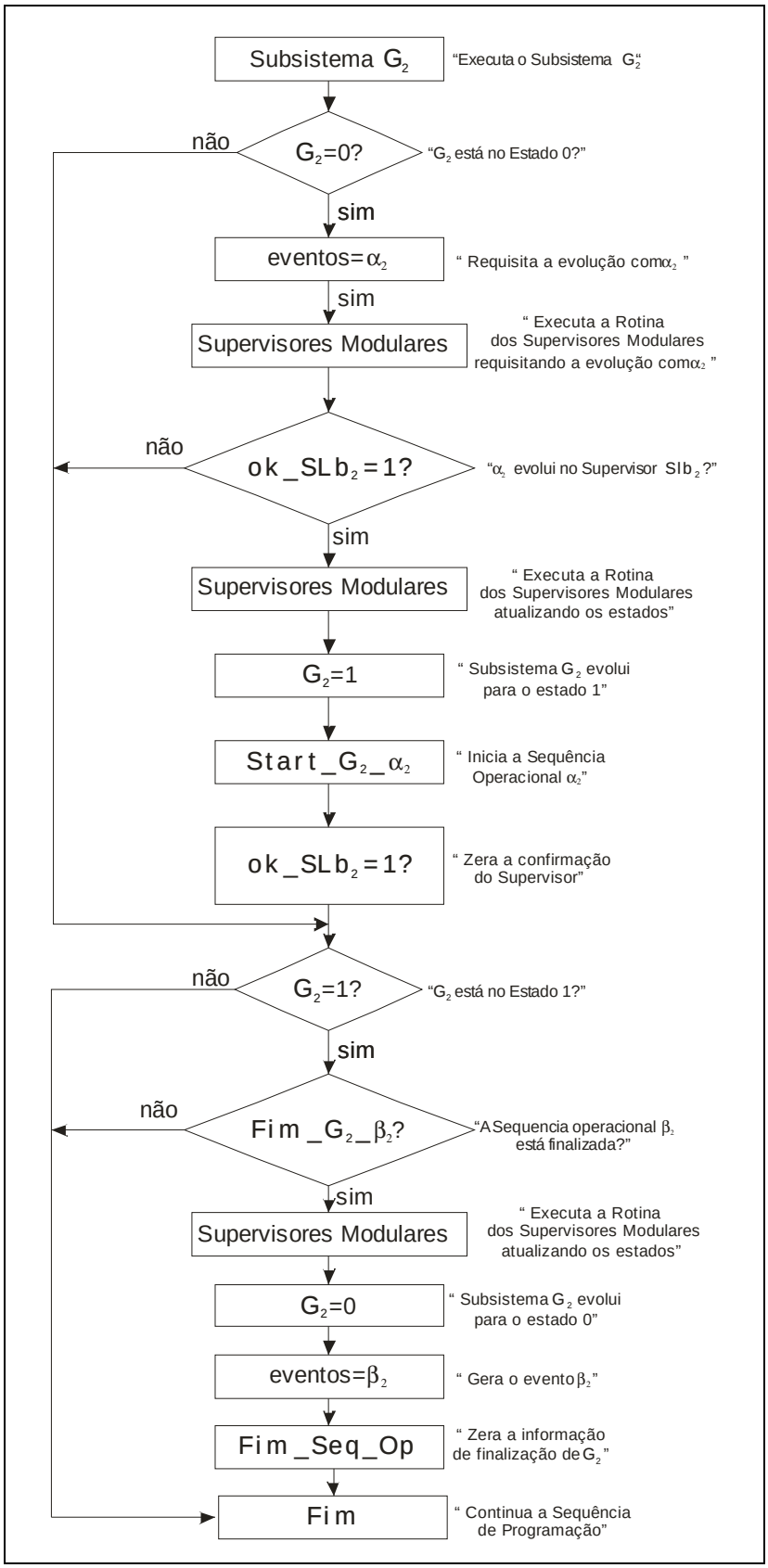

Figura 4.28 - Fluxograma Rotina Principal Subsistema G_2

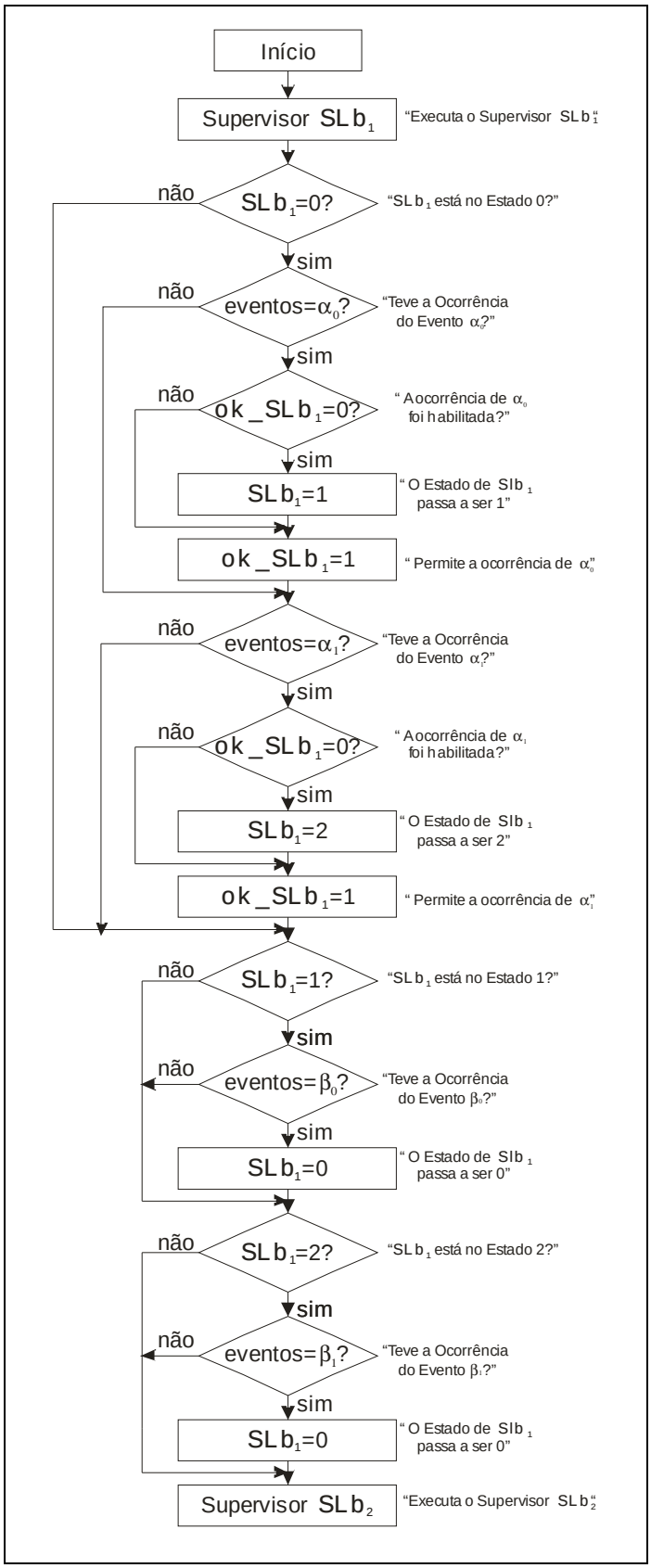

Figura 4.29 – Fluxograma Rotina Supervisores Modulares SLb_1

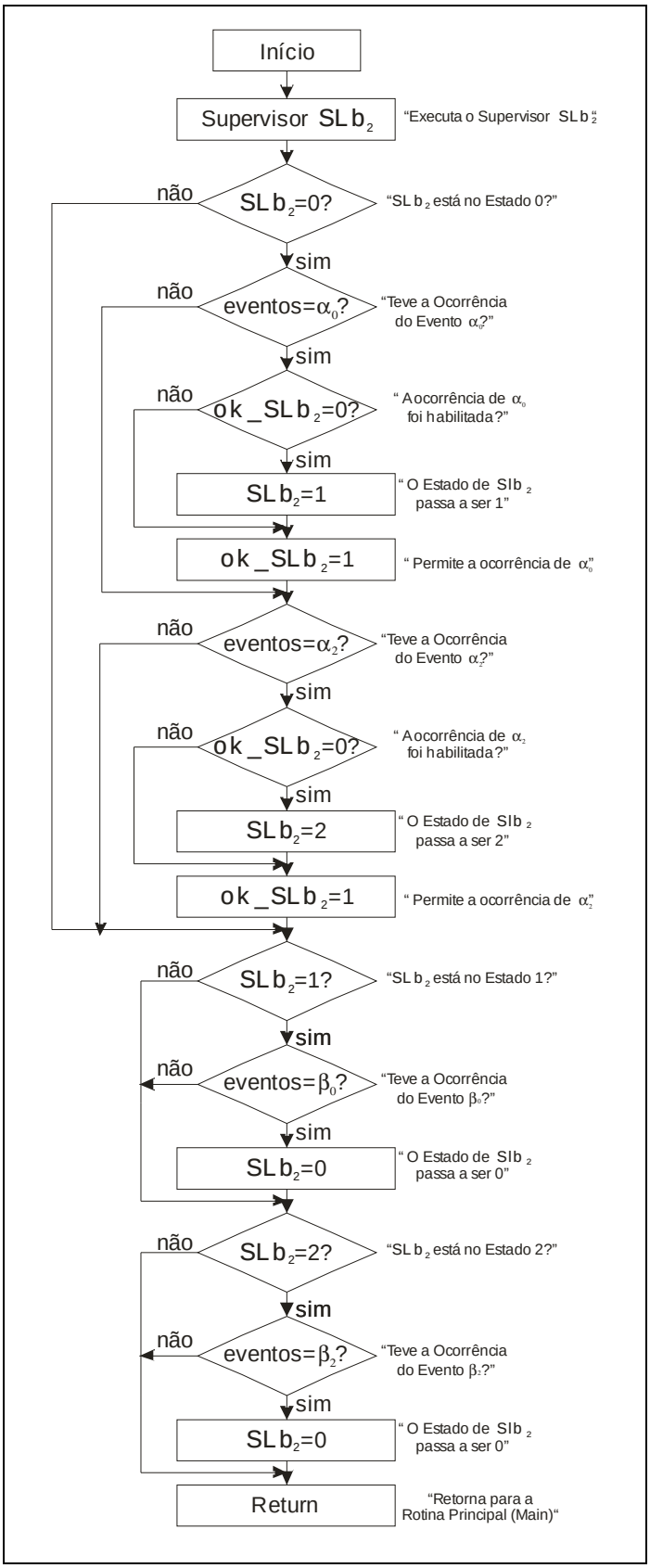

Figura 4.30 - Fluxograma Rotina Supervisores Modulares SLb_2

Todo o código escrito para a nova proposta em microcontrolador encontra-se em Anexo 2. Neste pode-se verificar também todas as funções necessárias para a correta operação do dispositivo: configuração, comunicação serial e funções específicas.

4.4 Comentários finais

O Capítulo 4 propôs a utilização do microcontrolador e uma linguagem padrão para o desenvolvimento e implementação do controle supervisório, desenvolvendo um *hardware* de baixo custo para supervisionar células de um sistema de manufatura.

Desenvolver novas formas de programação e métodos de implementação para a Teoria de Controle Supervisório Modular Local foi também o objetivo deste capítulo, esperando estimular o desenvolvimento prático da teoria e com isso contribuir com o crescimento e aplicabilidade industrial.

No estudo de caso buscou-se uma transcrição mais direta e simples do programa, sem utilização de qualquer função mais complexa da linguagem C (ponteiros ou variáveis diferentes de inteiros), permitindo redução no código, menor número de memória de dados, rapidez na compreensão e conferência na correta transcrição dos autômatos para o código fonte do microcontrolador.

Antes de utilizar-se do compilador específico para o dispositivo (microcontrolador), o código escrito foi validado no próprio compilador BorlandC para o PC, que atou semelhante a supervisão de controle do microcontrolador. Este sistema tem maior potencial de simulação da ferramenta e permite a execução passo a passo da rotina escrita, sendo necessária para a conferência do desenvolvimento do novo método. Esta plataforma (PC) confirma a reusabilidade da linguagem nos diferentes dispositivos.

Durante o desenvolvimento dos dois códigos e formas de implementação (abordagem Max Queiroz ou a desenvolvida nesta dissertação) este trabalho apontou a impossibilidade da espontaneidade na ocorrência dos subsistemas. Como forma de resolução propôs-se a utilização de número aleatório.

O novo método de implementação não exclui o antigo, mas trás uma nova opção para a obtenção do controle prático. Novos exemplos de plantas devem ser usados para confrontar os dois métodos, com intuito de melhorar ambos, criar um mais adequado ou simplesmente classificá-los segundo a necessidade. Pela comparação da capacidade de memória de dados e de programa requisitada nos dois códigos de controle, observou-se uma pequena redução na

memória de dados. Esta redução não representa um ganho considerável em relação ao antigo método, mas prova que o novo se equipara a este, confirmando ser mais uma opção de implementação.

Tabela 4.2 – Comparação entre as duas implementações.

Microcontrolador PIC16F877A	Memória de Dados "RAM"	Memória de Programa "ROM"
Implementação Max Queiroz	29(17%) – Nível Principal	455bytes (6%)
Nova Implementação	28(16%) – Nível Principal	455bytes (6%)

Capítulo 5

Conclusão

A Teoria de Controle Supervisório Modular Local, derivada da Teoria de Controle Supervisório, justifica-se em função da limitação do modelo clássico em adequar o número crescente de estados e eventos à capacidade física de implementação, ao desenvolvimento e análise computacional, à capacidade de memória dos equipamentos e à complexidade do acréscimo de novos subsistemas ao sistema já instituído.

Apesar da validação da Teoria de Controle Supervisório Modular Local em diversas aplicações, existem ainda diversos aspectos a serem explorados que necessitam de formulações, comparações, outros métodos de implementação e de transferência da linguagem nos mais diversos equipamentos de controle.

O objetivo geral em propor soluções otimizadas para a implementação da estrutura de controle em dispositivos industriais foi cumprido nesta dissertação. Este trabalho buscou mostrar as diferenças entre o método teórico e sua implementação prática nas tecnologias existentes, ressaltando questões tais como: otimização de memórias; padronização da linguagem para sistemas de baixo custo e uma nova forma de implementação de controle. Pela descrição das implementações em linguagem C, criou-se uma sistematização que pode gerar automaticamente o código de controle, desde que sejam desenvolvidas ferramentas mais avançadas para o estudo e obtenção dos Controladores Modulares.

Durante o desenvolvimento deste projeto, o acompanhamento da realidade fabril foi o principal direcionador para a elaboração das experiências contidas neste trabalho e busca de novos estudos de casos. Foi considerada a obrigatoriedade de uma comunicação serial e por rede do gerenciamento do sistema de controle com as seqüências operacionais, pois o

distanciamento quase sempre presente entre os controladores das células (CLP´s) distribuídos em um sistema flexíveis de manufatura encarece qualquer outra forma de comunicação.

Ressalta-se como contribuição principal da presente dissertação a proposta de uma nova abordagem de implementação do sistema de controle supervisório. A nova abordagem trabalha com os eventos observados e validados pelos supervisores modulares nos seus subsistemas relacionados, não necessitando do conjunto dos Eventos Desabilitados em cada estado do supervisor. O objetivo do novo desenvolvimento não é suceder a antiga implementação, mas sim ser um estímulo para novas soluções que busquem a redução e a melhor organização da lógica de implementação. Pode-se, ao final de cada desenvolvimento, validar a eficácia do antigo método, obter um novo paradigma ideal ou classificá-los segundo a necessidade e recurso dos equipamentos de controle.

5.1 PERSPECTIVAS FUTURAS

Este trabalho é o primeiro passo para a construção de um equipamento de controle com recursos destinados para esse fim, com uma especificação de robustez, compacto, de baixo custo e de rápida integração. No presente trabalho, em que foi utilizado um microcontrolador como *hardware* de controle supervisório, somente sua capacidade de processamento interno e suas saídas seriais foram utilizadas. Dessa forma, esse dispositivo físico torna-se um forte candidato ao projeto de um controlador dedicado, por ser um equipamento acessível e de baixíssimo custo.

Propõem-se como sugestão para trabalhos futuros a implementação em um sistema de maior porte, criação de bibliotecas de códigos para os novos projetos ou distribuição mais ampla da estrutura de controle (distribuição dos supervisores modulares em diversos microcontroladores).

ANEXOS

Anexo 1 – Código Segundo Abordagem Max Queiroz

```
#include <16f877.h>
#include<STDIO.H>
#include <STDLIB.H>
#FUSES XT,NOPROTECT,NOPUT,NOBROWNOUT,NOWDT
#use DELAY(CLOCK=4000000)
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7)
#define Start_G0_a0  65
#define Start_G1_a1  66
#define Start_G2_a2  67
#define Fim_G0_b0  68
#define Fim_G1_b1  69
#define Fim_G2_b2  70
#define a0  1
#define b0  2
#define a1  3
#define b1  4
#define a2  5
#define b2  6

//Variáveis Globais
int X_Slb1=0, X_Slb2=0;
int X_G0=0, X_G1=0, X_G2=0;
int eventos, D_eventos_a0,D_eventos_a1,D_eventos_a2;
int Fim_Seq_Op=0;
int aleatorio;
//X >> Corresponde aos Estados dos Autômatos
//eventos >> Corresponde aos Eventos
//D_eventos >> Corresponde aos Eventos desabilitáveis
```

```
//Ini_Seq_Op >> Envia para os CLP´s informação de início de Processo
//Fim_Seq_Op >> Envia para o Supervisor  informação de Fim de Processo do CLP´s

#INT_RDA
void RS232(void)
{
	Fim_Seq_Op=getc();
	return;
}

void Eventos_Desabitaveis (void)
{
//*****************************
//	Eventos Desabilitáveis
//*****************************

//Pelo Supervisor Slb1
	if (X_Slb1==1)
	{
		D_eventos_a1=1;
	}
	if (X_Slb1==2)
	{
		D_eventos_a0=1;
	}
//Pelo Supervisor Slb2
	if (X_Slb2==1)
	{
		D_eventos_a2=1;
	}
	if (X_Slb2==2)
	{
		D_eventos_a0=1;
	}
	return;
}

void Subsistemas (void)
{
```

```
//*******************************
//          Subsistemas
//*******************************
//return;
//Finaliza a evolução de um estado, não permitindo que haja mais de uma evolução na mesma execução.
//da rotina, subsistema, sem que os supervisores e os eventos desabilitados tenham sido atualizados.
//A criação da rotina aleatória permite que haja uma distribuição espontânea dos Subsistemas.
aleatorio=rand();
        if (aleatorio<33)
        {
        //Subsistema G0
                if (X_G0==0)
                {
                        if (D_eventos_a0!=1)
                        {
                                eventos=a0;
                                X_G0=1;
                                putc (Start_G0_a0);
                        }
                        return;
                }
                if (X_G0==1)
                {
                        if (Fim_Seq_Op==Fim_G0_b0)
                        {
                                eventos=b0;
                                X_G0=0;
                                Fim_Seq_Op=0;
                        }
                        return;
                }
        }
```

```
//Subsistema G1
        if (aleatorio>=33 && aleatorio<66)
        {
                if (X_G1==0)
                {
                        if (D_eventos_a1!=1)
                        {
                                eventos=a1;
                                X_G1=1;
                                putc (Start_G1_a1);
                        }
                        return;
                }
                if (X_G1==1)
                {
                        if (Fim_Seq_Op==Fim_G1_b1)
                        {
                                eventos=b1;
                                X_G1=0;
                                Fim_Seq_Op=0;
                        }
                        return;
                }
        }

        //Subsistema G2
        if (aleatorio>=66)
        {
                if (X_G2==0)
                {
                        if (D_eventos_a2!=1)
                        {
                                eventos=a2;
                                X_G2=1;
                                putc (Start_G2_a2);
                        }
                        return;
                }
                if (X_G2==1)
```

```
                {
                        if (Fim_Seq_Op==Fim_G2_b2)
                        {
                                eventos=b2;
                                X_G2=0;
                                Fim_Seq_Op=0;
                        }
                        return;
                }
        }
}

void Supervisores_Modulares  (void)
{
//*******************************
//      Supervisores Modular
//*******************************

//Supervisor Slb1
        if (X_Slb1==0)
        {
                if (eventos==a0)
                {
                        X_Slb1=1;
                }
                if (eventos==a1)
                {
                        X_Slb1=2;
                }
        }
        if (X_Slb1==1)
        {
                if (eventos==b0)
                {
                        X_Slb1=0;
                }
        }
        if (X_Slb1==2)
        {
```

```
                if (eventos==b1)
                {
                        X_Slb1=0;
                }
        }
//Supervisor Slb2
        if (X_Slb2==0)
        {
                if (eventos==a0)
                {
                        X_Slb2=1;
                }
                if (eventos==a2)
                {
                        X_Slb2=2;
                }
        }
        if (X_Slb2==1)
        {
                if (eventos==b0)
                {
                        X_Slb2=0;
                }
        }
        if (X_Slb2==2)
        {
                if (eventos==b2)
                {
                        X_Slb2=0;
                }
        }
        return;
}
```

```
void main (void)
{
enable_interrupts(GLOBAL); // all interrupts ON
enable_interrupts(INT_RDA); // RS232 ON
//1- Chama a Rotina dos Subpervisores
//2- Chama a Rotina dos Eventos Desabilitáveis
//3- Chama a Rotina dos Subsistemas
srand(1); // Semente do número aleatório
        while(1)
        {
                Supervisores_Modulares ();
                Eventos_Desabitaveis ();
                Subsistemas ();
                D_eventos_a0=0;
                D_eventos_a1=0;
                D_eventos_a2=0;
        }
}
```

Anexo 2 – Código Segundo Nova Abordagem

```
#include <16f877.h>
#include<STDIO.H>
#include <STDLIB.H>
#FUSES XT,NOPROTECT,NOPUT,NOBROWNOUT,NOWDT
#use DELAY(CLOCK=4000000)
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7)
#define Start_G0_a0  65
#define Start_G1_a1  66
#define Start_G2_a2  67
#define Fim_G0_b0  68
#define Fim_G1_b1  69
#define Fim_G2_b2  70
#define a0  1
#define b0  2
#define a1  3
#define b1  4
#define a2  5
#define b2  6

//Variáveis Globais
int X_Slb1=0, X_Slb2=0;
int X_G0=0, X_G1=0, X_G2=0;
int ok_SLb1=0,ok_SLb2=0;
int eventos;
int Fim_Seq_Op=0;
int aleatorio;

//X >> Corresponde aos Estados dos Autômatos
//eventos >> Corresponde aos Eventos
//D_eventos >> Corresponde aos Eventos desabilitáveis
//Ini_Seq_Op >> Envia para os CLP´s informação de início de Processo
//Fim_Seq_Op >> Envia para o Supervisor  informação de Fim de Processo do CLP´s
```

```
#INT_RDA
void RS232(void)
{
	Fim_Seq_Op=getc();
	return;
}

void Supervisores_Modulares  (void)
{
//*****************************
//	Supervisores Modular
//*****************************

//Supervisor Slb1
	if (X_Slb1==0)
	{
		if (eventos==a0)
		{

			if (ok_SLb1==1)
			{
				X_Slb1=1;
			}
		ok_SLb1=1;
		}
		if (eventos==a1)
		{
			if (ok_SLb1==1)
			{
				X_Slb1=2;
			}
		ok_SLb1=1;
		}
	}
```

```
        if (X_Slb1==1)
        {
                if (eventos==b0)
                {
                        X_Slb1=0;
                }
        }
        if (X_Slb1==2)
        {
                if (eventos==b1)
                {
                        X_Slb1=0;
                }
        }
//Supervisor Slb2
        if (X_Slb2==0)
        {
                if (eventos==a0)
                {
                        if (ok_SLb2==1)
                        {
                                X_Slb2=1;
                        }
                ok_SLb2=1;
                }
                if (eventos==a2)
                {
                        if (ok_SLb2==1)
                        {
                                X_Slb2=2;
                        }
                ok_SLb2=1;
                }
        }
```

```
if (X_Slb2==1)
        {
                if (eventos==b0)
                {
                        X_Slb2=0;
                }
        }
        if (X_Slb2==2)
        {
                if (eventos==b2)
                {
                        X_Slb2=0;
                }
        }
        return;
}
void main (void)
{
enable_interrupts(GLOBAL); // all interrupts ON
enable_interrupts(INT_RDA); // RS232 ON
//1- Chama a Rotina dos Subpervisores
//2- Chama a Rotina dos Eventos Desabilitáveis
//3- Chama a Rotina dos Subsistemas
srand(1); // Semente do número aleatório
        while(1)
        {
                aleatorio=rand();
                if (aleatorio<33)
                {
                        //Subsistema G0
                        if (X_G0==0)
                        {
                                eventos=a0;
                                Supervisores_Modulares();
                               if (ok_SLb1==1 && ok_SLb2==1)
                                {
                                        Supervisores_Modulares();
                                        X_G0==1;
                                        putc (Start_G0_a0);
```

```
                }
                ok_SLb1=0;
                ok_SLb2=0;
        }
        if (X_G0==1)
        {
                if (Fim_Seq_Op==Fim_G0_b0)
                {
                        Supervisores_Modulares();
                        X_G0=0;
                        eventos=b0;
                        Fim_Seq_Op=0;
                }
        }
}
if (aleatorio>=33 && aleatorio<66)
{
        //Subsistema G1
        if (X_G1==0)
        {
                eventos=a1;
                Supervisores_Modulares();
                if (ok_SLb1==1)
                {
                        Supervisores_Modulares();
                        X_G1==1;
                        putc (Start_G1_a1);
                }
                ok_SLb1=0;
        }
        if (X_G1==1)
        {
                if (Fim_Seq_Op==Fim_G1_b1)
                {
                        Supervisores_Modulares();
                        X_G1=0;
                        eventos=b1;
                        Fim_Seq_Op=0;
                }
```

```
                    }
                }
                if (aleatorio>=66)
                {
                    //Subsistema G2
                    if (X_G2==0)
                    {
                        eventos=a2;
                        Supervisores_Modulares();
                        if (ok_SLb2==1)
                        {
                            Supervisores_Modulares();
                            X_G2==1;
                            putc (Start_G2_a2);
                        }
                        ok_SLb2=0;
                    }
                    if (X_G2==1)
                    {
                        if (Fim_Seq_Op==Fim_G2_b2)
                        {
                            Supervisores_Modulares();
                            X_G2=0;
                            eventos=b2;
                            Fim_Seq_Op=0;
                        }
                    }
                }
            }
        }
```

Referências Bibliográficas

ALCÂNTARA, S; ALMEIDA FILHO, A; CAULLIRAUX, H; PACHECO DANTAS, E; DESCHAMPS, E; MARQUES, G; NOGUEIRA, A; CARVALHO, M; PROENÇA, A; SCHEER, A; SILVA FILHO, S; SIMA, R; SIMA, A; ZERBONE, E. **Manufatura Integrada por Computador**. Rio de Janeiro: Editora Campus, 1995.

ALONSO, R. M. **TKControl. Sistema de Processamento Baseado no Microcontrolador 803**. Monografia de Graduação, Ciência da Computação. Lavras-Minas Gerais, 2003.

AUTOMATION DIRECT, DL06 User Manual. Manual Number: D0-06USER-M, Volume 1 of 2.

BALEMI, S.; HOFFMANN, Gyugyi G. J. P.; WONG-TOI, H.; FRANKLIN, G.F. **Supervisory control of a rapid thermal multiprocessor**. IEEE Transaction on Automatic Control, 1993. vol. 38, pp. 1040-1059.

BRANDIN, B. A. **The real-time supervisory control of an experimental manufacturing cell**. IEEE Transactions on Robotics and Automation, 1996. v. 12, n. 1, pp.1-14.

CARROL, J.; LONG, D. **Theory of finite automata**. Prentice-Hall International Editions, 1989.

CASSANDRAS, C. G. and LAFORTUNE, S. **Introduction to discrete event systems**. Kluwer Academic Publishers, Massachusetts, USA, 1999.

CCS. **C Compiler Reference Manual**. Jully, 2003.

COSTA de OLIVEIRA, G.; TORTELLI, B; SANTOS, A. P. E.; BUSETTI, A. M. **Projeto e Implementação de um Sistema de Baixo Custo no Controle de Células de Manufaturas**. Artigo, Engenharia Mecatrônica, Pontifícia Universidade Católica do Paraná, Curitiba, 2004.

CURY, J. **Teoria de Controle Supervisório de Sistemas a Eventos Discretos**. Canela-RS: V Simpósio Brasileiro de Automação Inteligente, 2001.

CURY, J; QUEIROZ, M, . Controle **Modular de Sistemas de Manufatura Discretos**. Artigo, LCMI, Florianópolis.

DIETRICH, P.; MALIK, R.; WONHAM, W. M.; BRANDIN, B. **Implementation considerations in supervisory control. Synthesys and control of discrete event systems.** Kluwer Academic Publishers, 2002. pp. 185-201.

DINIZ, R. L. **Curso Completo de C**. Apostila, http://www.apostilando.com.br, Download 2006.

ERBE, H. **Low Cost Intelligent Automation Manufacturing**. Congresso, Barcelona, Espanha, 2002.

FABIAN, M.; HELLGREN, A. **PLC-based Implementation of Supervisory Control for Discrete Event Systems**. Proceedings of the 37th IEEE Conference on Decision and Control, 1998. v. 3, pp. 3305-3310.

GEORGINI, M. **Automação Aplicada. Descrição e Implementação de Sistemas Seqüenciais com PLCs**. São Paulo: Editora Érica, 2000. 5 ed.

HASDEMIR, I. T.; KURTULAN, S.; GÖREN L. **Implementation of local modular supervisory control for a pneumatic system using PLC**. Proceedings of the Workshop on Discrete Event Systems,2004. pp. 25-30.

HOLLOWAY, L. E.; KROGH, B. H.; GIUA, A. **A survey of Petri nets methods for controlled discrete event systems**. Discrete Event Dynamic Systems: Theory and Applications, 7, pp. 151-190. Boston: Kluwer Academic Publishers, 1997.

HOPCROFT, J. E.; ULLMAN, J. D. **Introduction to automata theory, languages and computation**. Addison-Wesley USA, 1979.

KUMAR, R.; GARG, V. **Modeling and control of logical discrete event systems**. Kluwer Academic Publishers, 1995.

LEDUC, R. J. **PLC Implementation of a DES supervisor for a manufacturing testbeb: An implementations perspective**. M.A.Sc. Thesis, Dept. of Elect. and Comp. Eng., University of Toronto, Canadá, 1996.

LIU J.; DARABI, H. **Ladder logic implementation of Ramadge-Wonham supervisory controller**. Proceedings of the Workshop on Discrete Event Systems, 2002. pp. 383-392.

MALIK, P. **Generating Controllers from Discret-Event Models**. In F. Cassez, C. Jard , F. Laroussinie, and M. D. Ryan, editors, MOVEP'2002. pp. 337-342.

MICROCHIP. **MICROCHIP, PIC16F87X 28/40-pin 8-Bit CMOS FLASH Microcontrollers**. Microchip Technology Inc., 2006.

MULLENDER. **Distributed Systems**. ACM Press.

RAMADGE, P. J; WONHAM, W. M. **The Control of Discrete Event Systems**. Proceedings of IEEE, Special Issue on Discrete Event Dynamic Systems.

SIPPER, D; BULFIN JR, R. **Production: Planning, Control, and Integration**. Singapore: McGraw Hill, 1998.

SOUZA, J. D. **Desbravando o PIC: ampliado e atualizado para PIC 16F628**. São Paulo: Editora Érica, 2003. 6 ed.

TENÓRIO, B. M. **Apostila Sobre Linguagem C**. Apostila, http://www.apostilando.com.br, Download 2006.

VIEIRA, A. **Contribuição à Implementação de Sistemas de Controle Supervisório – Documento de qualificação de Tese de Doutorado. Programa de Pós-Graduação em Engenharia Elétrica**. Florianópolis, Dezembro de 2004.

VIEIRA, A. **Implementação de Estrutura de Controle de Sistema a Eventos Discretos em Controlador Lógico Programável Utilizando a Teoria Controle Supervisório Modular Local**. Curitiba, 2003.

VIEIRA, A; CURY, J; QUEIROZ, M. **Distribuição de Estrutura de Controle Supervisório a Eventos Discretos**. Artigo, PPGEPS, Curitiba; LCMI, Florianópolis; GEMM, Florianópolis.

WONHAM, W. M.. **Notes on controls of discrete event systems. Course notes for ECE 1636F/1637S**. Dept. Of Electrical and Computer Engineering, University of Toronto, Canadá, 1999.

ZILLER, M. R. **A Abordagem Ramadge-Wonham no Controle de Sistemas a Eventos Discretos: Contribuição à Teoria**. Dissertação, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, Outubro 1993.