9.1 (more um)

DUK YOUNG CHOI

ROGÉRIO AUGUSTO VICENTINI PIARDI

TRADUTOR E-MFG / SFC

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Graduação em Engenharia.

São Paulo

2000

DUK YOUNG CHOI

ROGÉRIO AUGUSTO VICENTINI PIARDI

TRADUTOR E-MFG / SFC

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Graduação em Engenharia.

Área de Concentração:
Engenharia Mecânica / Mecatrônica

Orientador:
DIOLINO JOSÉ DOS SANTOS FILHO .

São Paulo

2000

A meus pais e toda minha família
Duk

A meus pais José Mário e Maria
Odete, minha avó Dirce e minhas tias
Rogério

AGRADECIMENTOS

Gostaríamos de agradecer ao nosso orientador Prof. Dr. Diolino José dos Santos Filho pela amizade e pelo apoio fundamental para conclusão deste trabalho de formatura.

Agradecemos também às nossas famílias, que nos amaram e apoiaram em todos os momentos de nossos vidas, nada esperando em troca.

Agradecemos também à nossos colegas e amigos que fizemos nesses cincos anos de faculdade, com os quais aprendemos muito e tivemos momentos marcantes.

Para finalizar. agradecemos ao colega Eng. Fábio Luís Lopes Magalhães pelo tempo e trabalho dedicados no início do desenvolvimento desse trabalho de formatura.

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO

O objetivo principal deste é documentar o desenvolvimento de trabalho de formatura, que consiste em desenvolver-se um software "tradutor" entre E-MFG e SFC, o qual seria utilizado na automação do processo de testes de validação do modelo projetado em E-MFG em CLPs (Controladores Lógicos Programáveis).

1.1 - OBJETIVOS

O objetivo final deste projeto é obter-se uma ferramenta que auxiliasse um projetista a poder implementar e simular o algoritmo de controle para um sistema regido a eventos discretos modelado segundo a metodologia PFS/MFG proposta por Miyagi em [1]. Para representação do sistema seria utilizado o grafo orientado E-MFG e o algoritmo de controle necessariamente deve ser obtido de forma a ser utilizado diretamente na programação de um CLP.

Para obter-se o algoritmo de controle escolheu-se seguir uma linguagem de programação de CLPs que fosse padronizada e bastante utilizado no mercado. Dessa forma, pesquisou-se e com isso encontrou-se a norma IEC 1131-3, que regulamenta cinco linguagens de programação de CLPs as quais devem tornar-se linguagens-padrão de programação até o ano de 2003. Dentre estas cinco linguagens regulamentadas escolheu-se utilizar uma

linguagem de programação na forma de grafo conhecida como Sequential Flow Chart ou SFC.

1.2 - MOTIVAÇÕES

As principais motivações para a implementação desta ferramenta tradutora de um sistema modelado em E-MFG para um programa SFC seriam suprir a necessidade da aplicação de uma metodologia sistematizada para obtenção do algoritmo de controle em uma linguagem que poderia ser utilizada para programação em CLPs.

Com o desenvolvimento deste tradutor a necessidade de refazer-se todo o modelo na linguagem de programação específica de um CLP, uma tarefa realmente árdua para sistemas que apresentem alto grau de complexidade e que corresponde ao principal impecílio para implementação do algoritmo de controle, seria drasticamente minimizado, possibilitando a projetistas com conhecimento pouco aprofundado sobre programação de CLPs implementar e simular seu sistema modelado em E-MFG em um CLP, restando-lhes a tarefa de verificar-se a fidelidade do resultado final da tradução em relação ao comportamento esperado para o modelo original.

1.3 - ORGANIZAÇÃO DO TEXTO

No Capítulo 2 são apresentados os requisitos básicos de projeto para obtenção de uma metodologia de tradução E-MFG / SFC, apresentando-se a norma IEC 1131-3 e as linguagens padronizadas para programação de CLPs.

No Capítulo 3 é apresentada a metodologia de tradução E-MFG / SFC que será utilizada pela ferramenta tradutora, mostrando-se a correspondência entre os elementos estruturais do E-MFG e os elementos do SFC.

No Capítulo 4 o projeto do software de tradução é apresentado, a metodologia de tradução E-MFG / SFC que será utilizada pela ferramenta tradutora, mostrando-se a correspondência entre os elementos estruturais do E-MFG e os elementos do SFC.

No Capítulo 5 é mostrado um estudo de caso comparativo sobre os resultados de uma tradução realizada por um protótipo da ferramenta tradutora e também realizada manualmente seguindo-se a metodologia de tradução proposta.

No Capítulo 6 são apresentadas considerações finais sobre o trabalho de formatura, destacando-se os pontos fortes do projeto e propondo-se expansões e melhorias a ele.

Nos Anexos A e B são mostrados alguns fundamentos necessários sobre os grafos E-MFG, destacando-se as expansões que os diferenciam de grafos MFG, e sobre a programação para CLPs em SFC, com foco nas características funcionais, respectivamente. Já no Anexo C é apresentado o código fonte do programa-protótipo utilizado no caso apresentado no Capítulo 5.

CAPÍTULO 2 - DEFINIÇÃO DOS REQUISITOS DE PROJETO

Inicialmente foi proposto a construção de uma ferramenta computacional para conversão de um modelo representado em um grafo E-MFG em uma linguagem de programação de CLPs conhecida como SFC, visando posterior simulação e validação em CLPs. Mas para isso faz-se necessário estabelecer-se quais seriam os requisitos para obter-se essa ferramenta.

Antes de iniciar-se a projetar-se a ferramenta faz-se necessário adquirir-se conhecimentos embasados sobre os grafos E-MFG, focando-se em suas características, sua estrutura funcional e a dinâmica das marcas no grafo.

Além disso, necessita-se um aprendizado sobre programação de CLPs, sobre a programação em SFC e consequentemente sobre a norma que definiu a utilização do SFC como linguagem padrão para a programação de CLPs, a norma IEC 1131-3.

Segundo esta norma, SFC pode ser usado como uma linguagem simples, mas a sua principal função é trabalhar como integrador de módulos escritos em outras linguagens em um programa de alto nível. Assim, torna-se necessário escolher-se uma das demais linguagens de programação de CLPs definidas na norma para utilização no processo de tradução.

Para realizar-se a escolha da linguagem de programação de CLPs a ser utilizada concumitantemente com o SFC haviam quatro opções dadas pela norma IEC 1131-3:

- Diagrama de Relés ("Ladder Diagram");
- Diagrama de Blocos de Função ("Function Block Diagram");
- Lista de Instruções ("Instruction List") e
- Texto Estruturado ("Structured Text").

Diagrama de relés consiste em uma linguagem gráfica de programação de CLPs e é a mais comum das linguagens de programação utilizadas. A origem dessa linguagem de programação é a notação de diagrama de relés que consiste na representação de combinações de contatos e relés para implementação de funções lógicas. Já o Diagrama de Bloco de funções é uma linguagem gráfica baseada em diagramas de blocos interconectando blocos que contenham as funções.

Texto Estruturado e Lista de Instruções são as linguagens textuais definidas pela norma. A Lista de instruções é uma linguagem de mais baixo nível, sendo correspondente à linguagem Assembly para programação voltada a PCs. E o texto estruturado é uma linguagem de mias alto nível, com a possibilidade utilizar-se de sub-rotinas, controles de seleção e iteração na programação, isto devido à sintaxe ser predominantemente derivada da linguagem Pascal para programação de PCs.

Adquirindo-se conhecimentos sobre a modelagem em E-MFG e sobre a programação de CLPs utilizando-se SFC segundo a norma IEC 1131-3, pode-se iniciar a definição de uma

metodologia de tradução entre E-MFG e SFC, baseando-se nas possibilidades de tradução entre os elementos que compõem cada um dos grafos.

Estando esta metodologia pronta e validada, pode-se iniciar o projeto do tradutor, escolhendo-se a ferramenta a ser utilizada para o desenvolvimento do programa tradutor e determinando-se as entradas do programa, o fluxo de informações, as funções necessárias e desejadas para a tradução e a implementação de protótipos do tradutor para a realização de testes.

CAPÍTULO 3 - TRADUÇÃO SFC / E-MFG

3.1 - TRADUÇÃO DOS ELEMENTOS BÁSICOS

Após ter-se um contato mais aprofundado com E-MFG e com o SFC, podem-se definir algumas regras básicas de tradução entre estes tipos de grafo.

Para cada "*box*" do E-MFG pode-se associar um "*step*" do SFC, assim como cada *transição* pode ser associada a cada "*transition*". Já os *arcos* estão relacionados com os links.

Para representar um *arco de sinal de saída* será usado no "*step*" correspondente ao "*box*" do qual sai esse dado arco um "*action*" que contém um comando mandando um sinal para a saída do CLP.

A representação das *portas* que vêm do sistema para o grafo E-MFG será realizada por condições setadas pelas entradas do CLP que habilitariam ou inibiriam as "*transitions*" correspondentes às transições em que essas portas habilitariam ou inibiriam. As demais *portas* habilitadores ou inibidores serão representados de forma semelhante: a habilitação ou inibição será realizada por condições nas "*transitions*", sendo que essas condições seriam setadas nas "*actions*" localizadas nos "*steps*" correspondentes aos "*boxes*" de que saem esses portas.

Já a representação dos atributos das marcas podem ser realizada a partir de variáveis em cada um dos "*steps*", as quais seriam ocupadas pela lista de atributos da marca que

estivesse ativando aquele "*step*". Outra solução possível seria utilizar-se vetores representando cada uma das marcas que percorre o grafo, com um "atributo" extra numerando a posição da marca no grafo. Nota-se que ambas as implementações são viáveis e possíveis, com amplas possibilidades de implementação.

Para finalizar resta mencionar as inscrições nos arcos e transições. Como essas não foi encontrado nada na literatura sobre inscrições nos links além das ordens de preferências para eventos exclusivos, todas as inscrições necessárias podem ser realizadas nas "*transitions*" através do código usando-se uma das linguagens definidas pela norma IEC 1131-3. Devido à possibilidade de obtenção de programas equivalentes utilizando-se qualquer das linguagens de programação definidas pela norma IEC 1131-3 decidiu-se protelar a definição da linguagem de programação para o projeto da ferramenta. Na Figura 7 é mostrada a conversão de elementos simples do E-MFG em SFC.

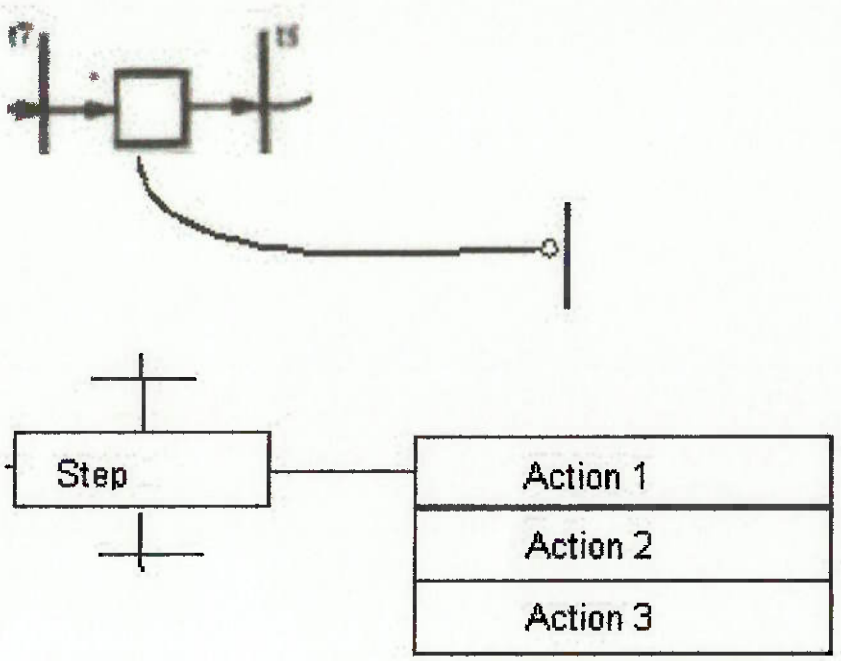

Figura 7 – Conversão E-MFG/SFC

3.2 - TRADUÇÃO DOS MACRO-ELEMENTOS

Já para a representação dos macro-elementos do E-MFG é necessário que sejam decompostos utilizando-se os elementos estruturais como os "*boxes*", as *transições* e os *arcos* .

Para representar-se o "*box*" capacidade em SFC, como mostrado na Figura 8, deve-se utilizar a possibilidade de indicar-se a ordem de entrada em um seqüenciamento exclusivo de "n" steps (vários links saindo de uma mesma transition) e para a saída utiliza-

se também um seqüenciamento exclusivo da mesma dimensão com ordem de saída definida nos links, utilizando-se de condições na transição para controlar a saída.

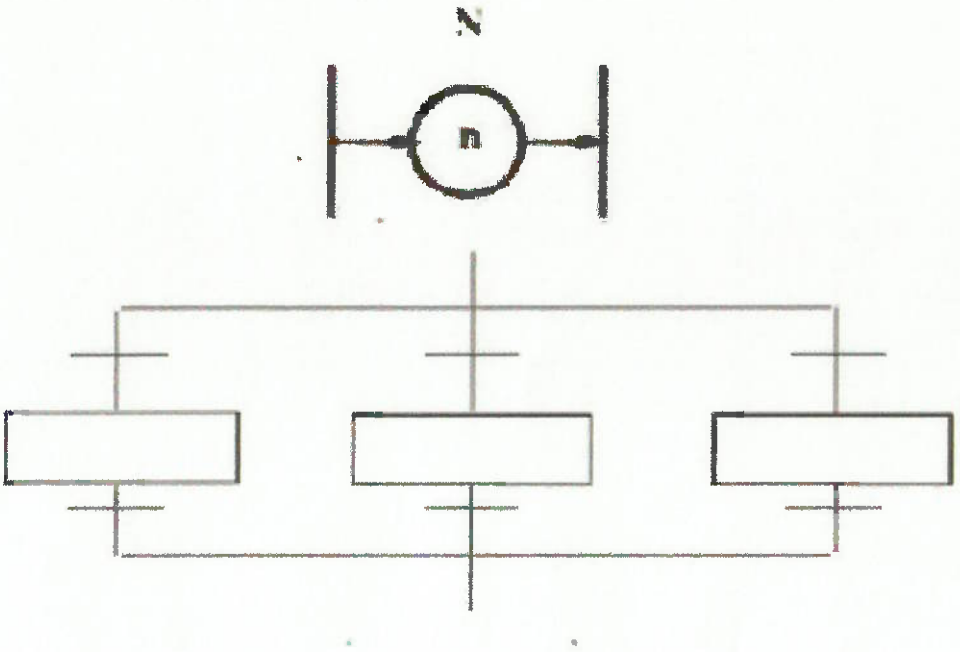

Figura 8 – Conversão E-MFG/SFC – "*box*" capacidade

Já o "*box*" agrupador pode ser modelado por um seqüenciamento exclusivo de "n" steps na entrada e um seqüenciamento em paralelo da mesma dimensão na saída, isto é, só quando todos os steps estiverem ativados, recebendo cada um uma marca simples, será realizado o disparo, com todos os parâmetros armazenados em uma variável que eqüivaleria ao código de controle. A Figura 9 mostra um exemplo dessa conversão.

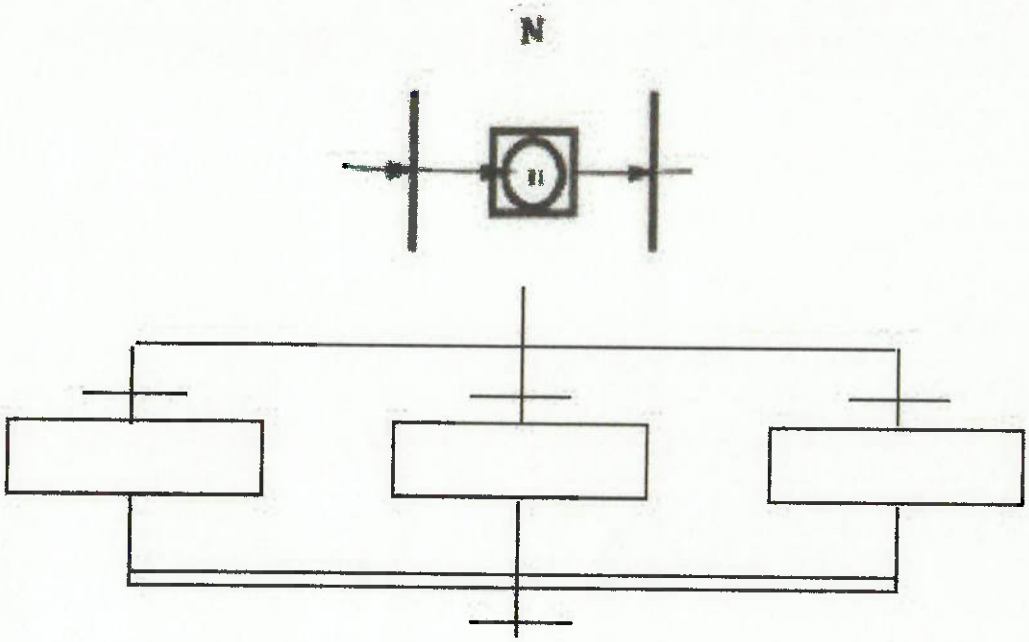

Figura 9 – Conversão E-MFG/SFC – "*box*" agrupador

O "*box*" dispersor seria modelado de forma inversa ao agrupador: seria iniciado por um seqüenciamento de "n" steps em paralelo e fechado por um seqüenciamento exclusivo com ordem definida na saída, acionando todos os steps quando a marca composta entra no dispersor e desativando uma por vez na saída de cada uma das marcas individuais na ordem pré-determinada. Um exemplo é mostrado na Figura 10.

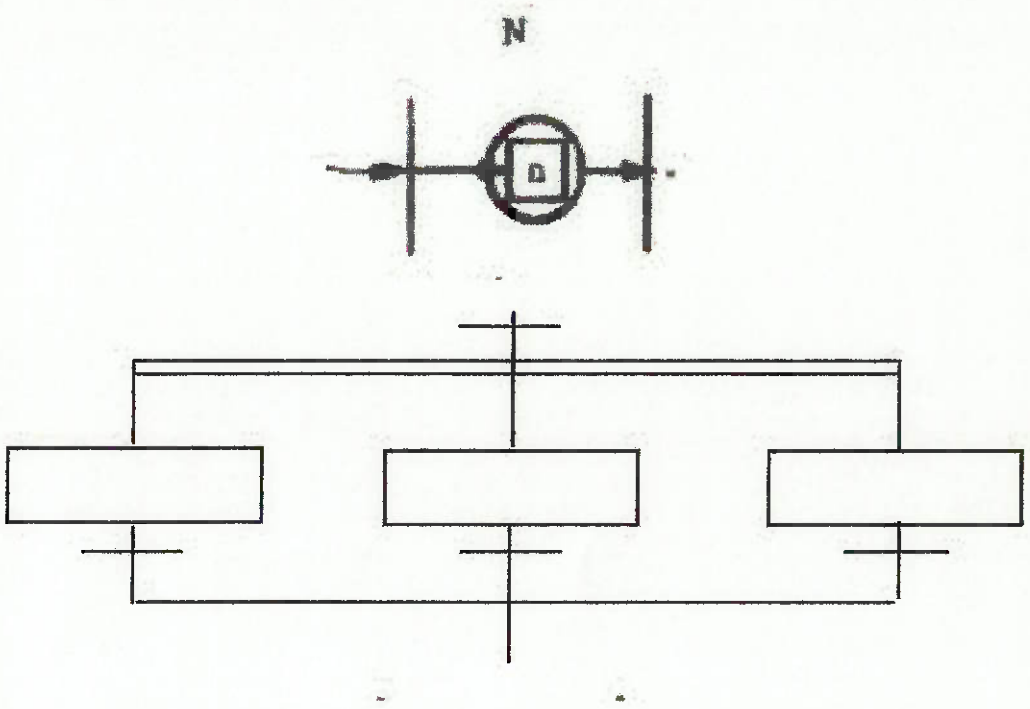

Figura 10 – Conversão E-MFG/SFC – "*box*" dispersor

O "*box*" controlador será modelado por um step com uma action com instruções para alteração dos parâmetros e o "*box*" temporizador pode ser modelado por um step temporizado ou por um step com um elemento de temporização no action.

CAPÍTULO 4 - PROJETO DO SOFTWARE

Com a definição de uma metodologia sistematizada para obtenção de um programa SFC a partir de um grafo E-MFG, pode-se passar ao projeto da ferramenta que aplicará essa metodologia.

Primeiramente, para ter-se uma ferramenta que aplicaria a metodologia de tradução deve-se especificar como serão os dados de entrada (o grafo E-MFG e suas características) e como deverão ser lidos e armazenados pelo programa. Na seqüência, deve-se determinar como será o tratamento desses dados para que a metodologia de tradução seja aplicada e, depois da tradução ser realizada, a forma de representar-se o grafo SFC resultante da tradução.

4.1 - REPRESENTAÇÃO DO E-MFG

Na definição dos dados de entrada, é necessário definir-se uma linguagem de representação textual dos elementos de um grafo E-MFG com mínima (ou nenhuma) perda de características. Escolheu-se uma representação textual para que o usuário possa escrever a representação do grafo diretamente em programas editores de texto, sem apresentar maiores dificuldades. Mas, para que essa linguagem seja interpretada pelo programa ela deve seguir uma padronização, a ser mostrada na seqüência desse capítulo.

Para que essa linguagem seja consistente com o grafo E-MFG modelado, ela deve representar todas as características do mesmo, que podem ser resumidas às características de cada um dos elementos estruturais do grafo: transição, "*box*", arcos orientados e portas.

Os "*boxes*" serão representados por números de duas casa decimais ("01", "02" , "64", etc.), com um caracter ":" representando o início da definição das características do mesmo e um ";" representando o final da definição. Também especificou-se as "*tags*" "<box>" e "</box>" para delimitar-se a região em que podem ser definidos os "*boxes*", com a descrição de cada "*box*" ocupando uma linha diferente. Para exemplificar, algumas representações textuais de "*boxes*":

- "01:;" para um "*box*" simples, sem nenhuma condição especial ou macro-característica;
- "02:OUT(16);" para um "*box*" com um arco de sinal de saída, acionando a saída 16 do CP;
- "03:AGR(05);" para um "*box*" agrupador com capacidade para 5 marcas.

Para representar-se as transições é levado em consideração que cada transição apresenta "*boxes*" anteriores à transição e "*boxes*" posteriores à transição, com um caracter "-" representando o separador entre os "*boxes*" anteriores e posteriores, um caracter "," representando o separador entre cada um dos diversos "*boxes*" anteriores ou posteriores, um caracter ":" representando o início da definição das características e um ";" representando o final dessa definição. Também definiu-se as "*tags*" "<transition>" e "</transition>" para delimitar-se a região em que podem ser definidas as transições, com a

descrição de cada transição ocupando uma linha diferente. Na seqüência, alguns exemplos da representação proposta:

- "01-02:;" representa uma transição simples entre os "*boxes*" "01" e "02";
- "11,13-14:TMP(30);" representa uma transição temporizada com período igual a 30 unidades de tempo, que ligaria os "*boxes*" "11" e "13" ao "*box*" "14";
- "12,14,15-16,17:;" representa uma transição simples ligando os "*boxes*" "12", "14" e "15" aos "*boxes*" "16" e "17".

Já para representar-se as portas habilitadoras e inibidoras considerou-se cada uma como característica específica da transição em que a habilitação (ou desabilitação) é realizada. Alguns exemplos seriam:

- "01-02:HGT(13);" representa uma transição simples entre os "*boxes*" "01" e "02" com uma porta habilitadora saindo do "*box*" "13";
- "11-12:NGT(05), HGT(03);" representa uma transição simples entre os "*boxes*" "11" e "12" com uma porta habilitadora saindo do "*box*" "05" e com uma porta inibidora saindo do "*box*" "03".

A representação das características das marcas individualizadas é feita de forma análoga: definiu-se "*tags*" " "<mark>" e "</mark>" para delimitar-se a região em que podem ser definidas as características da marcas, com cada tipo de marca individual e suas características ocupando uma linha diferente. Cada tipo de marca é representado por um número com duas casas decimais, de forma análoga à representação dos "*boxes*" ("01",

"12" , "45", etc.), com um caracter ":" representando o início da definição das variantes daquela característica da marca, um caracter ";" representando o final dessa definição e um caracter "," representando o separador entre cada uma das variantes da característica definida. Cada uma dessas variantes é representada por uma palavra de dois caracteres, conforme mostrado abaixo:

- "01:AZ,PT,BR;" representaria a característica "01" da marca, a qual varia entre os valores "AZ", "PT" e "BR".

Outras características, como inscrições nos arcos e nas transições podem ser representadas de maneira análoga como características das transições em que ocorrem (no caso de inscrições nas transições) ou nas transições posteriores ou anteriores (para inscrições nos arcos). A linguagem permite expansões nas características dos elementos, as quais devem também ser aplicadas ao código-fonte da ferramenta.

Para finalizar, um exemplo de um grafo E-MFG mostrado pela Figura 11 e que será estudado posteriormente, e sua respectiva representação textual, mostrada pela Tabela 1:

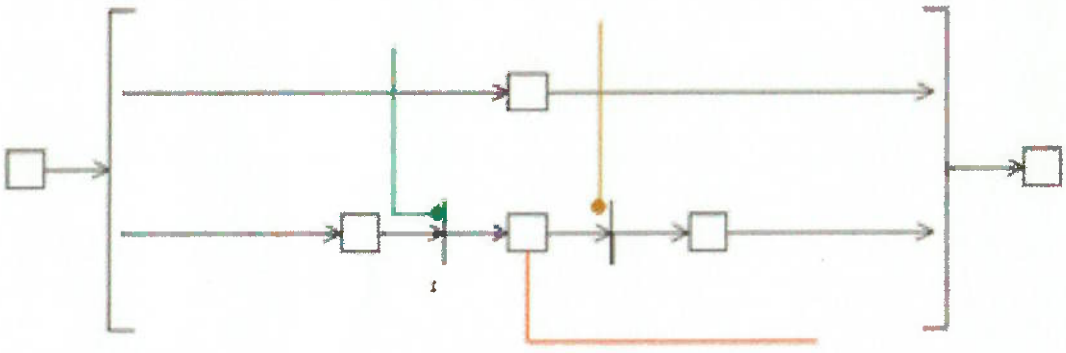

Figura 11 – grafo MFG

Tabela 1 – Representação Textual do MFG

```
<mark>
</mark>
<transition>
</transition>
<box>
01:;
02:;
03:;
04:OUT(01);
05:;
06:;
</box>
<transition>
01-02,03:;
03-04:HAE(01);
04-05:HAE(01);
03,05-06:;
</transition>
```

4.2 - MODELAGEM DO PROGRAMA E TRATAMENTO DOS DADOS

Havendo-se definido como será a entrada do programa, agora é necessário definir-se como esses dados serão tratados pelo programa. Para explicitar como os dados devem fluir no programa será usado um fluxograma (mostrado pela Figura 12) com as funções que serão utilizadas.

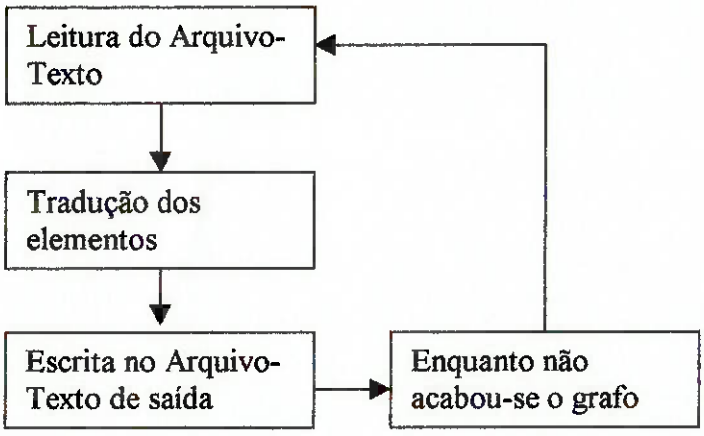

Figura 12 – Esquema básico de fluxo de informações

O programa primeiramente fará a leitura do arquivo-texto de entrada, e após à leitura, é feita seqüencialmente a tradução dos elementos e a escrita do resultado no arquivo-texto de saída até que termine-se o grafo.

Na leitura do arquivo-texto de entrada o conteúdo é separado em três "arrays", um representando o primeiro a lista de caracteristicas da marca, o segundo a lista dos boxes do grafo e o terceiro a lista de transições do grafo, conforme mostrado pela Figura 13.

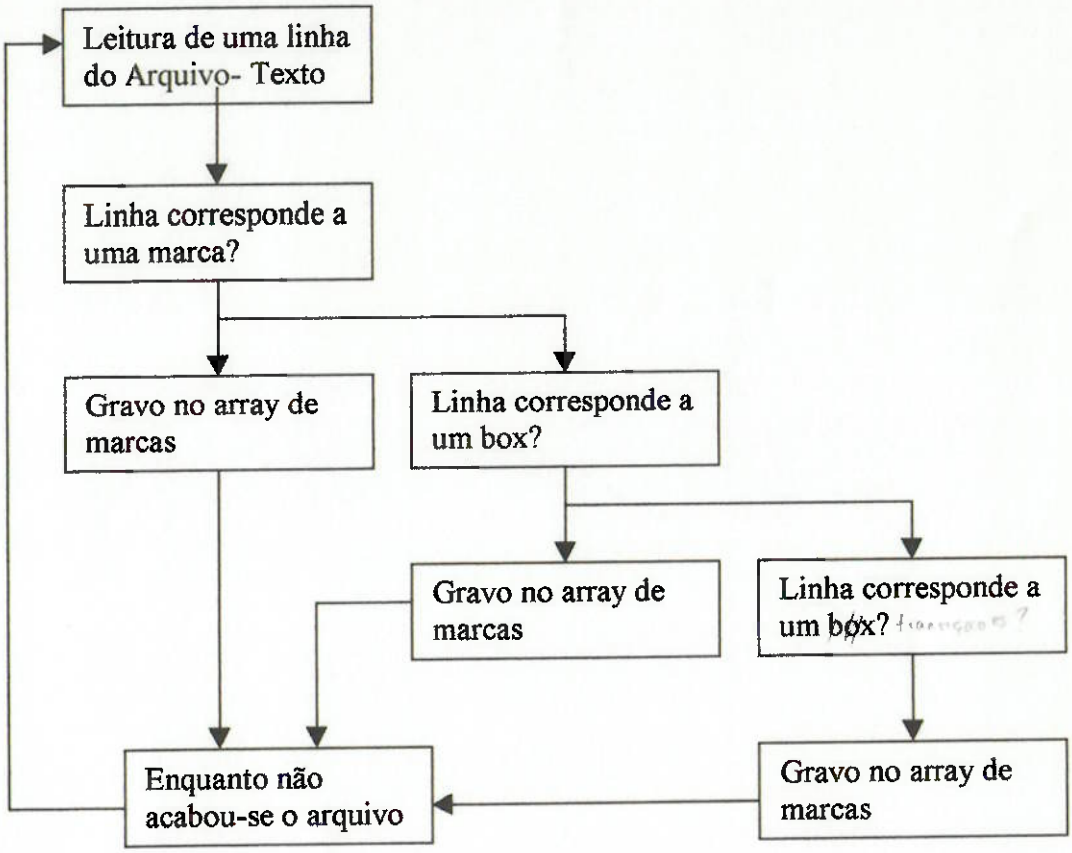

Figura 13 – Organograma - Leitura do Arquivo-Texto

Já a tradução dos elementos E-MFG e a transcrição dos elementos SFC resultantes é realizada utilizando-se o "array" de transições como base: a partir de uma dada transição

são traduzidos os "boxes" antecedentes e os "boxes" posteriores àquela transição um a um, caso já não tenham sido traduzidos. Na tradução cada um dos "boxes" é separado para posterior escrita na saída conforme sua localização relativa à transição. Após o término da tradução dos "boxes" ligados àquela transição, é traduzida a própria transição e as portas ligadas a ela.

Terminada a tradução de todos os elementos relacionados com aquela transição, é escrito no arquivo-texto de saída os elementos SFC resultantes da tradução dos "boxes" antecessores da transição ("steps" e "actions"), na seqüência a tradução da transição e das portas ligadas a ela ("transition" e condições) e para finalizar os elementos traduzidos a partir dos "boxes" posteriores à transição ("steps" e "actions"). Todos estes procedimentos são realizados para todos os elementos do "array" de transições.

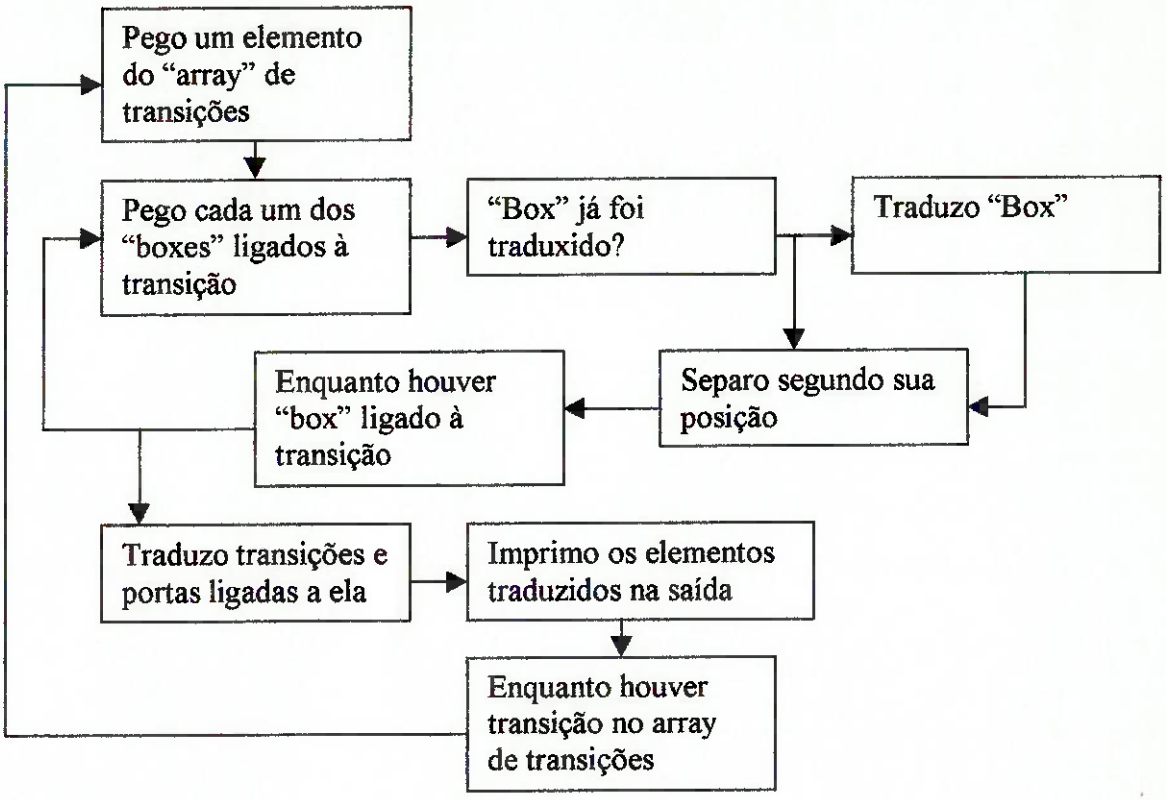

Figura 14 – Organograma - Tradução e escrita no arquivo-texto de saída

4.3 - REPRESENTAÇÃO DO GRAFO SFC

Para a representação do grafo SFC obtido pela ferramenta de tradução adoutou-se uma representação textual baseada em "tags"que representariam os elementos estruturais do SFC ("step", "transition" e "action"), de maneira semelhante à representação utilizada para descrever o grafo E-MFG lido pelo programa.

Segundo essa representação, um step seria delimitado por tags <step xx> e </step>, com "xx" representando o número que fora utilizado para denominar o box que corresponderia àquele step.

Já uma action ligada ao um step não seria representada explicitamente, mas as instruções que deverão ser executadas por essa action estarão escritas nas linhas que separam as tags <step xx> e </step>. Essas instruções poderiam ser escritas em qualquer uma das quatro linguagens definidas pela norma IEC-1131-3, e assim optou-se pela utilização de Texto Estruturado devido à maior facilidade de representação dessa linguagem em forma de texto.

A representação de uma transition é feita de forma análoga à representação de um step, utilizando-se tags delimitadoras (<trans xxx> e </trans>), com "xxx" representando os números dos steps ligados à transição, com um "-"separando os steps antecedentes dos steps posteriores à transição e com um "," separando os diversos steps anteriores (ou posteriores) entre si. E as condições que se aplicam em uma transition são escritas em Texto Estruturado nas linhas que separam as tags <trans xxx> e </trans>.

Um exemplo da representação textual proposta para o SFC será mostrado no caso estudado no capítulo seguinte.

CAPÍTULO 5 - ESTUDO DE CASO – IMPLEMENTAÇÃO DA FERRAMENTA

Na implementação da ferramenta de tradução optou-se pela implementação de um programa simplificado, o qual interpretaria e traduziria apenas um tipo específico de grafo E-MFG. Assim, escolheu-se como grafo a ser traduzido o grafo mostrado pela Figura 15, o qual já foi mostrado em capítulos anteriores.

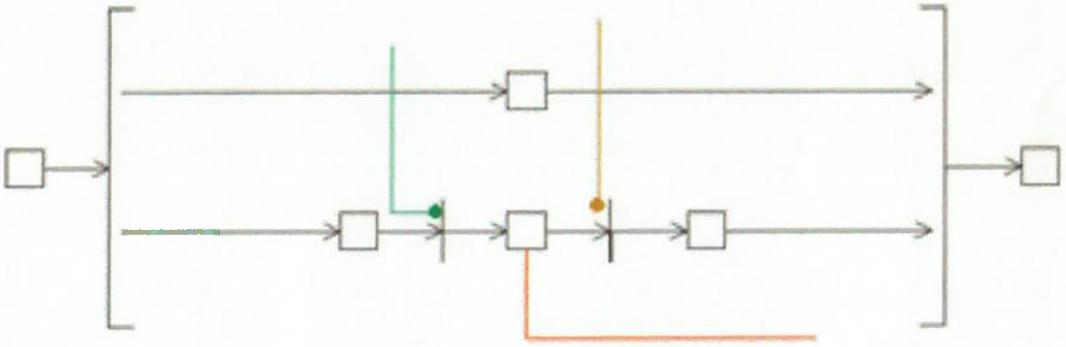

Figura 16 – grafo MFG

Este grafo foi escolhido devido à capacidade de modularização apresentada por um sistema modelado segundo a metodologia PFS/MFG proposta por Miyagi em [1]. Em um sistema modelado segundo essa metodologia é possível a divisão do modelo em blocos funcionais menores, facilitando-se o detalhamento da solução.

Na seqüência, a representação textual desse grafo é mostrada na Tabela 2, a representação do grafo SFC obtido pela ferramenta de tradução é mostrada na Tabela 3 e o grafo SFC

obtido pela tradução manual seguindo-se a metodologia proposta é mostrado pela Figura 16.

Tabela 2 – Representação Textual do grafo MFG estudado

```
<mark>
</mark>
<transition>
</transition>
<box>
01:;
02:;
03:;
04:OUT(01);
05:;
06:;
</box>
<transition>
01-02,03:;
03-04:HAE(01);
04-05:HAE(01);
03,05-06:;
</transition>
```

Tabela 3 – Representação Textual do grafo SFC traduzido pela ferramenta

<step 1> s1 = TRUE </step> <trans 01-02,03> </trans> <step 2> s2 = TRUE </step> <step 3> s3 = TRUE </step> <trans 03-04> == INPUT1 </trans>	<step 4> s4 = TRUE OUTPUT1 == TRUE </step> <trans 04-05> == INPUT2 </trans> <step 5> s5 = TRUE </step> <trans 03,05-06> </trans> <step 6> s6 = TRUE </step>

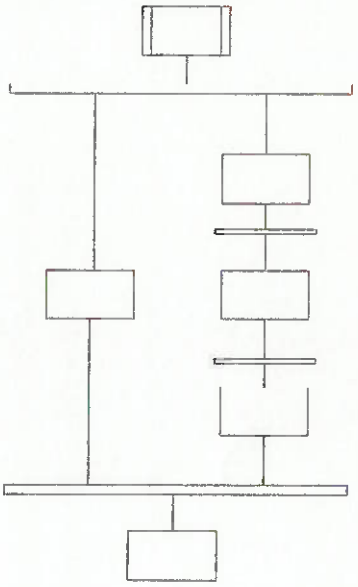

Figura 16 - grafo SFC traduzido manualmente

Comparando-se os resultados obtidos pode-se notar que o modelo obtido através do tradutor é idêntico ao grafo obtido aplicando-se manualmente a metodologia de tradução, sendo que obteve-se a mesma quantidade de steps e a mesma ligação a partir das transitions.

CAPÍTULO 6 - CONSIDERAÇÕES FINAIS

Para finalizar este trabalho, se faz necessário apresentar as melhorias decorrentes do desenvolvimento da ferramenta tradutora E-MFG / SFC projetada neste trabalho de formatura:

- Definição de uma metodologia sistematizada de tradução entre E-MFG / SFC;

Sem a definição de uma metodologia sistematizada para tradução de um sistema modelado em E-MFG para um programa SFC, a implementação do algoritmo de controle seria muito mais trabalhosa e onerosa, sendo necessário praticamente refazer o sistema na linguagem de programação do CLP.

- Geração de documentação do algoritmo de controle que será utilizado nos CLPs;

Como atualmente os ciclos de vida de algoritmos de controle tem se mostrado cada vez menores, a existência de uma documentação sobre o algoritmo de controle facilitaria e agilizaria a consulta para posteriores alterações ou correções no algoritmo.

- Desenvolvimento de uma ferramenta de arquitetura aberta, possibilitando expansão e atualizações da ferramenta;

No momento atual é forte a tendência da utilização de plantas com múltiplos processamentos simultâneos. Assim, para identificação de múltiplos processos se faz necessário um ferramenta de modelagem de sistemas discretos que apresente marcas

individualizadas, como a expansão do grafo MFG conhecida como E-MFG. Em conseqüência disto, tornaria-se oportuno a expansão do tradutor projetado neste trabalho de formatura para traduzir-se grafos E-MFG que apresentariam maior grau de complexidade que o caso estudado neste trabalho.

Algumas melhorias que poderiam ser feitas à ferramenta, além da expansão para casos mais complexos, seriam:

- Utilizar-se de uma interface gráfica para formatar-se tanto o modelo E-MFG de entrada como o programa SFC obtido, integrando-se a modelagem com a geração do algoritmo de controle e facilitando-se a interpretação dos resultados;

- Integração da ferramenta com compiladores SFC, gerando uma saída que pudesse diretamente ser usada em CLPs.

ANEXO A - E-MFG – MARK FLOW GRAPH ESTENDIDO

O E-MFG (Mark Flow Graph Estendido) é um grafo derivado do MFG (Mark Flow Graph), ao qual foram acrescidas algumas propriedades e elementos visando a modelagem e o controle um sistema a eventos discretos (SED) de modo a representar melhor a flexibilidade operacional do sistema, como por exemplo um sistema fabril no qual são produzidas diferentes peças com compartilhamento de recursos fabris como máquinas-ferramenta, transportadores e robôs.

As principais capacidades acrescentadas ao E-MFG para são a inserção de atributos à marca, de maneira semelhante às Redes de Petri coloridas, e também a associação de regras de produção às transições para representar regras adicionais de controle.

A.1 - ELEMENTOS BÁSICOS

Os elementos que constituem o E-MFG são basicamente os mesmos que constituem o MFG com a inserção de extensões, de modo a adaptá-los ao nível elevado de representação do grafo E-MFG. Os elementos básicos são:

- as *transições* que indicam a ocorrência de eventos e permitem inscrições para representar regras para a evolução do sistema;
- os *"boxes"*, indicando as condições para ocorrência dos eventos;

- as *marcas* que indicam a manutenção de uma condição, sendo individualizadas por atributos ou não;
- os *arcos orientados*, estabelecendo uma relação entre as condições e os eventos, podendo conter inscrições variáveis para controle da transmissão de atributos às marcas;
- as *portas* que habilitam ou inibem a ocorrência de eventos, admitindo inscrições fixas relacionadas com os atributos das marcas para controlar a ocorrência dos eventos de forma mais efetiva e;
- os *arcos de sinal de saída* para transmissão de informações a dispositivos externos, sendo que estas informações podem ser relativas ao estado dos atributos de uma marca através de inscrições variáveis nestes arcos.

A.2 - CARACTERÍSTICAS DO E-MFG

Alguns outros conceitos relacionados ao E-MFG que são importantes para a modelagem e serão destacados são :

- a estrutura das marcas individuais;
- o conceito de marca composta;
- a representação de inscrições nas transições;
- a modularização do E-MFG.

A.2.1 - Estrutura das Marcas Individuais

No E-MFG as marcas são acompanhadas por atributos que as diferenciam umas das outras, individualizando-as. Aos atributos podem ser associadas diversas informações referentes ao sistema modelado, como por exemplo o tipo de processo ou o produto envolvido. Os atributos são definidos por um vetor de dimensão *n* e os componentes desse vetor representam cada um dos atributos, os quais, por sua vez, podem ser representados por variáveis lógicas ou numéricas. A Figura 1, retirada de Santos Filho [3], representa um exemplo de estrutura de marca individual.

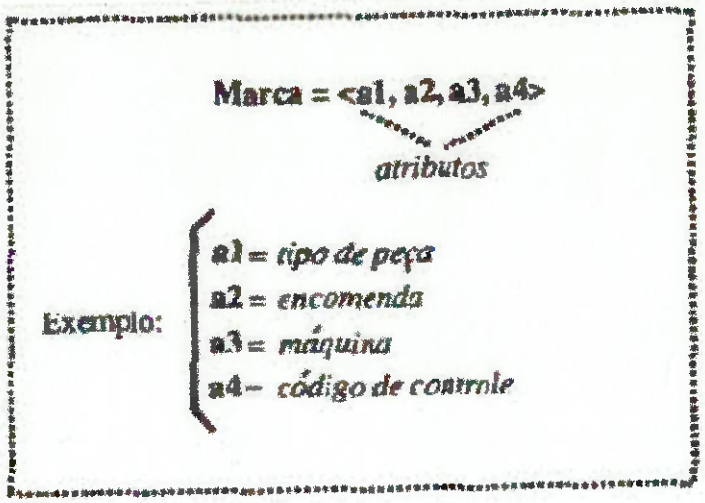

Figura 1- Representação da estrutura de uma marca individual

A.2.2 - Marca Composta

Já uma marca composta consiste na marca que contém os atributos referentes à composição de diversas marcas individuais simples. Este tipo de marca é necessário na representação,

sem perda de informação, de quais itens são agrupados ("palletizados"), ou então da origem de itens desagrupados ("despalletizados") durante um processo. A criação de uma marca individual composta baseia-se no estabelecimento de uma matriz de atributos, sendo que cada uma das linhas dessa matriz corresponde a um vetor de atributos de uma das marcas originais.

Para representar-se uma marca individual composta, é necessário que na estrutura da marca contenha um atributo especial representando um código de controle de acesso às informações dessa matriz. Na representação desse código de controle seria reservar-se o último componente do vetor de atributos o código de controle. A Figura 2 mostra um exemplo de agrupamento de três marcas individuais simples com vetores de quatro atributos em uma marca individual composta e o código de controle de composição resultante.

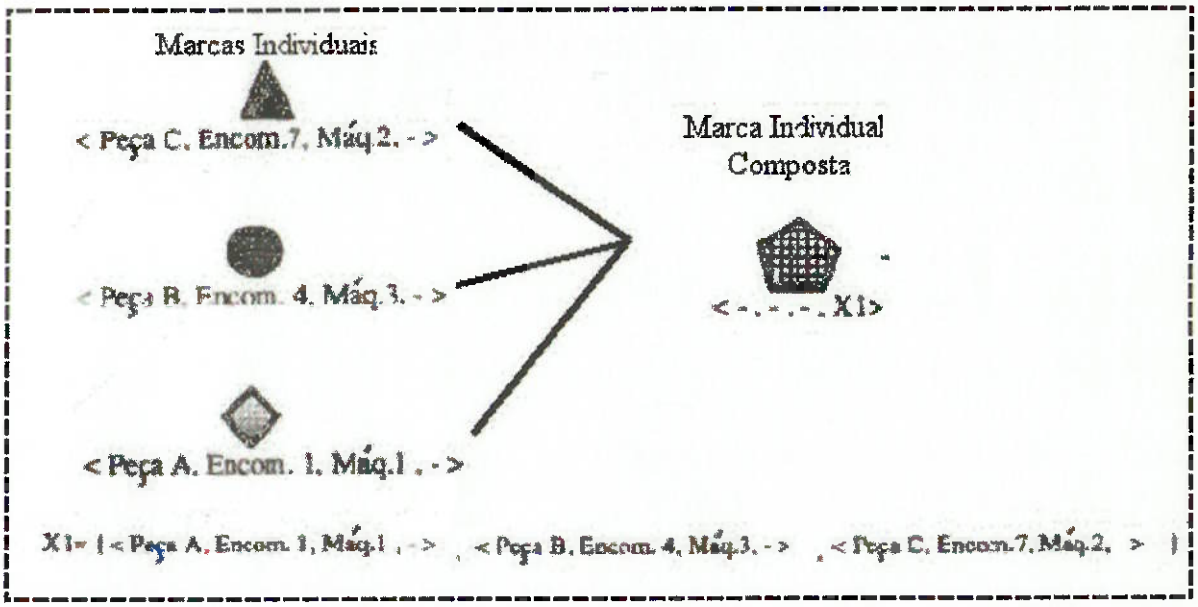

Figura 2- Exemplo de marca composta com quatro atributos

A.2.3 - Modularização do E-MFG: Macro-Elementos

Em relação à modularização, o E-MFG apresenta alguns macro-elementos cuja finalidade é simplificar a representação de dispositivos presentes na manufatura, como magazines de armazenamento temporário, empacotadores para transporte em "pallets" e os desempacotadores. Para representar estes elementos foram adicionados ao E-MFG os "*boxes*" funcionais do tipo capacidade, agrupador e dispersor, os quais são capazes de controlar a quantidade de peças envolvidas na dada etapa e também a seqüência de entrada e saída de materiais, além de poder registrar o conteúdo das cargas, isso devido à individualização das marcas graças aos atributos.

O "*box*" capacidade modela magazines para armazenamento temporário de itens ou a disponibilidade de recursos armazenada. Este tipo de "*box*" apresenta flexibilidade na forma de saída das marcas - FIFO ("First-In First-Out") ou FILO ("First-In Last-Out") - devido à individualização das mesmas e, por ser um componente do E-MFG, é possível especificar-se as regras de saída de itens deste "*box*", além da capacidade máxima. A Figura 3, retirada de Santos Filho [3] mostra um exemplo de "*box*" capacidade.

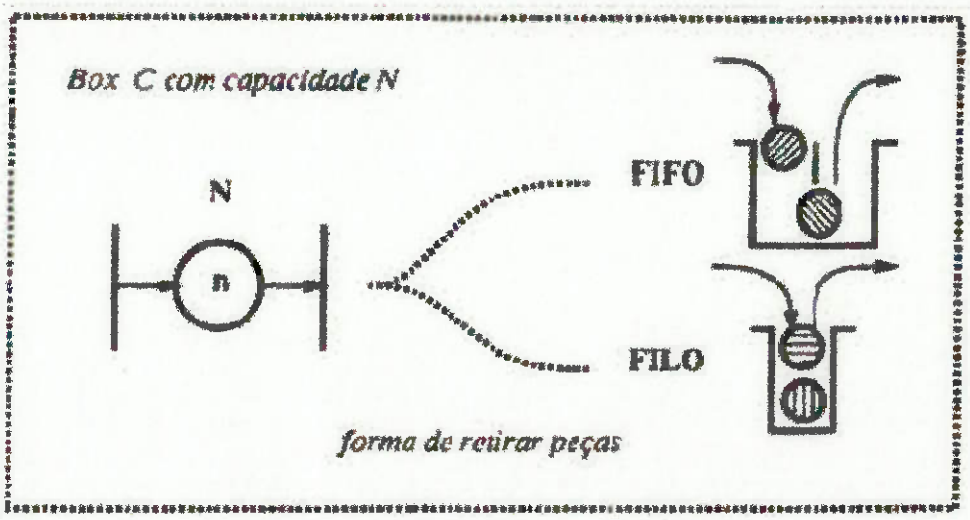

Figura 3-"Box" Capacidade

O "*box*" agrupador representa os empacotadores para transporte em "pallets", com a entrada de diversas marcas individuais simples uma a uma resultando na em uma marca individual composta, armazenando-se os atributos associados às marcas na ordem de entrada. No instante em que a capacidade do "*box*" agrupador é atingida ocorre o disparo da transição de saída, seguindo-se a marca composta e desaparecendo-se as marcas simples. Um exemplo de "*box*" agrupador é mostrado pela Figura 4, retirada de Santos Filho [3].

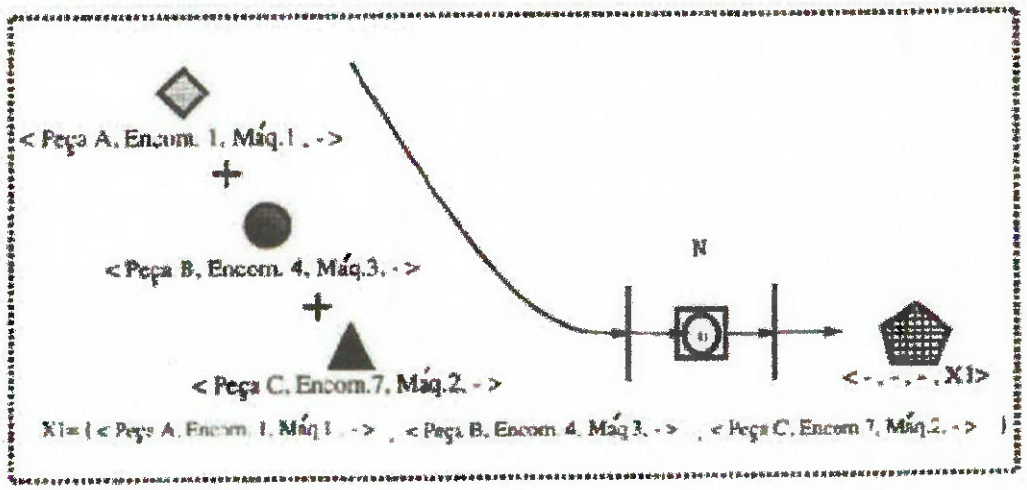

Figura 4- "Box" Agrupador

O "*box*" dispersor somente pode receber uma marca composta como entrada, e a partir do atributo referente à composição pode-se recuperar o conteúdo e a ordem das marcas individuais primitivas. Com isso pode ser feita a decomposição da marca composta em duas formas: FO ("First-Out") ou LO ("Last-Out"). Após a decomposição, o "*box*"só estará disponível após a saída de todas as marcas individuais simples. Na Figura 5, retirada de Santos Filho [3], é mostrado um exemplo de "*box*" dispersor.

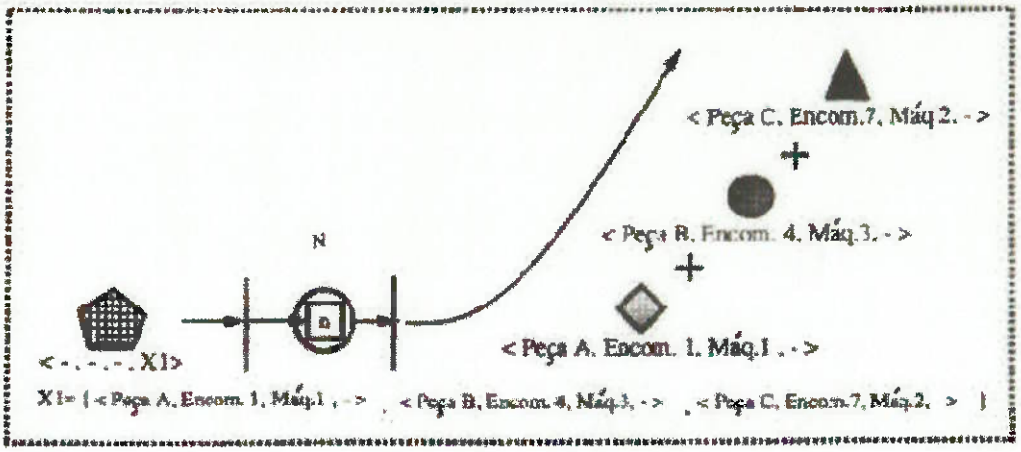

Figura 5- "Box" Dispersor

O "*box*" controlador tem como finalidade atualizar-se os atributos das marcas. Esta atualização é realizada a partir através de regras de controle determinadas pelo projetista. Estas regras de controle são especificadas por regras de produção do tipo "se ... então" e referem-se ao estado dos atributos, e a atualização faz-se alterando-se o estado desses atributos. A Figura 6, retirada de Santos Filho [3], ilustra um exemplo de "*box*" controlador alterando os atributos de uma marca.

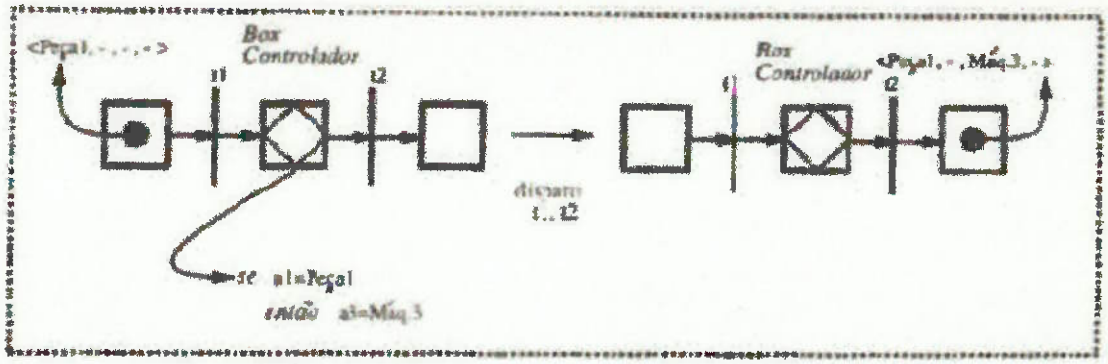

Figura 6-"Box" Controlador

O box temporizador é um box de capacidade unitária que retém a marca em seu interior por um determinado intervalo de tempo, sendo que a duração desse intervalo pode estar condicionada ao estado de algum dos atributos. Este tipo de box é adequado na representação de uma máquina-ferramenta que processa diferentes tipos de peça, envolvendo diferentes tipos de processamento.

ANEXO B - SFC (SEQUENTIAL FLOW CHARTS)

Programas PLC SFC não imitam as linguagens tradicionais de alto nível. Os SFCs são ferramentas de estruturação de programas que apresentam a visualização de fluxo de controle. A estrutura de SFC inclui steps, transitions, actions e links unindo esses elementos. Cada step e transition é implementado com uma linguagem IEC 1131, tais como ladder logic, linguagem estruturada, lista de instruções, diagrama de blocos, implementada. Os SFCs são adequados para aplicações onde a execução pode ser dividida em steps distintos. Na Figura 6 é mostrado um exemplo de programa SFC.

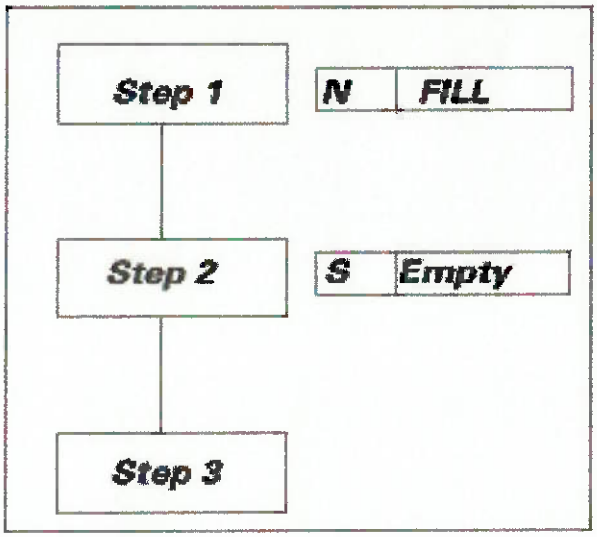

Figura 7-"Box" Controlador

B.1 - ELEMENTOS

B.1.1 - Step (etapa)

O step representa uma etapa de uma seqüência e as ações atribuídas ao step são definidas pelo action block do step. O step pode permanecer em um dos dois estados lógicos: ON (ativo) ou OFF (inativo). O estado da seqüência é determinado em qualquer instante pelo conjunto dos steps ON e pelos valores das variáveis internas e de saída destes steps. Ele pode ser representado gráfica ou textualmente. O step flag indica se o step está ON ou OFF. O tempo de ativação (passagem) do step mede o tempo desde o instante em que o step tornou-se ON até o instante em que se torna OFF, mantendo este valor mesmo depois do step ser OFF.

B.1.2 - Transition ou transition condition (transição ou condição de transição)

Quando existe um (ou mais) steps conectados através de links a um (ou mais) steps, a transition indica a condição para que o estado ON dos steps antecedentes passe para os steps subseqüentes. A transition é representada por um traço horizontal (perpendicular) ao link. A informação escrita ao lado da transition é a condição da transition, que é uma condição lógica para que a evolução ocorra.

B.1.3 - Link ou directed link (conexão ou conexão orientada)

O link conecta os steps na representação gráfica do SFC, indicando o caminho da evolução. Quando nada está explicitamente indicado, seu sentido é de cima para baixo. Assim, nos jumps utilizam-se de rótulos (labels) para a identificação dos pontos de confluência.

B.1.4 - Action (ação)

Um step pode ou não possuir actions que são acionadas quando o step fica ON. A action pode ser conectada diretamente à direita do step ou, através da utilização das declarações STEP e END_STEP e colocados (descritos) em outra parte.

A execução da action ocorre sempre que o step está ON, mas quando este fica OFF, o step flag assume o valor lógico 0 e a action é executada uma única vez.

B.2 - ATRIBUTOS PARA SFC NO NÍVEL DE IMPLEMENTAÇÃO NO PLC

Aqui veremos os atributos genéricos de PLC SFCs.

B.2.1 - Confiabilidade

Confiabilidade se traduz como a capacidade de executar funções exigidas sob certas condições para certo período de tempo (IEEE, 1990) ou a probabilidade de suceder operação enquanto houver a demanda. A confiabilidade depende da capacidade de prever os seguintes itens durante a execução:

- Utilização de memória;
- Fluxo de controle;
- Tempo.

Utilização de memória

Os programas SFC não alocam memória diretamente. Mas esta abordagem é aplicável apenas no nível de implementação.

Fluxo de controle

A capacidade de prever fluxo de controle é a de determinar facilmente e sem ambigüidade que caminho o programa deve seguir sob condições especificadas.

Os atributos básicos relacionados são seguintes:

- Maximizar estruturação – deve evitar ao máximo o uso de GOTO para que não aconteça a desestruturação do programa. Apesar de SFCs permitirem o uso dele, seu uso deve ser evitado exceto uma situação: tratamento de eventos fora da seqüência em situação anormal.
- Minimizar a complexidade de fluxo de controle – para fazer isso segue as seguintes instruções:
  - Limitar o número de caminhos paralelos – o número de caminhos paralelos do início ao fim no trecho lógico deve ser limitado a sete.

- Limitar uso de SFC para operações não seqüenciais – uso de SFCs em aplicações não seqüenciais vai resultar em grande número de links diretos e divergência de seleções seqüenciais, resultando uma complexidade enorme para SFC.

- Iniciar variáveis antes do uso – SFCs não tratam a inicialização de variáveis, pois eles não possuem variáveis internos. Mas as variáveis podem ser tratadas no outro nível:

  - Inicialização como parte do programa – as variáveis específicos de SFC, quando existem são inicializados e mantidos por sistema PLC.

  - Inicialização de processo steps e transitions – dentro de cada step e transition, a inicialização é feita na linguagem IEC 1131 de nível baixo (ex. Ladder Logic)

- Entrada e saída únicas para sub-programas – A "gramática" do SFC permite apenas única entrada e a única saída para cada step.

- Especificar tipo de dados – algumas implementações de SFC não possuem variáveis, portanto não há tipo de dados. Outras possuem variáveis associados com cada step. Cada step também possui temporizador de step que indica o tempo pelo qual step deve estar ativado. Entretanto, os tipos de dados associados a variáveis de cada step são fixos.

- Contabilidade de exatidão e precisão de dados – não é aplicável para qualquer caso, pois algumas implementações não possuem variáveis.

- Pedido de precedência de operadores aritmético, lógico e funcional

- Evitar funções com efeitos colaterais

Tempo

A capacidade de prever o tempo ~e crucial num sistema seguro usado sistema de controle de tempo real. As especificações relacionadas são:

- Minimização de tarefa – no nível de código fonte, SFC não suporta multitarefa, entretanto, em termo de sistema operacional, alguns PLCs podem suportar multitarefa, executando vários SFCs independentes ao mesmo tempo. Este tipo de multitarefa com SFCs independentes deve ser evitado.

- Minimização de processamento de interrupção – SFC em si não suporta interrupções. Mesmo que haja alguma ocorrência que exija um tratamento imediato, SFC não consegue fazê-lo devido a sua natureza de execução seqüencial. Entretanto, a interrupção poderia ser utilizada a nível de PLC, portanto, é necessário que a implementação software/hardware assegure o funcionamento seguro sob as condições com interrupções.

- Divergência de seqüências – a divergência de seqüência é representada em SFC por uma linha horizontal abaixo de step, seguido por múltiplos transitions paralelos. Os seguintes itens são instruções:

- Definir mutuamente condições de transition exclusivo – todas as condições de transition em uma divergência de seqüência de seleção deveria ser programada para ser mutuamente exclusivo.

- Assegurar convergência de seqüência seguindo divergência dela – qualquer divergência de escolha de seqüência deve eventualmente seguido por uma convergência de seqüências, onde os caminhos alternativos se reúnem.

- Contagem para limites de número de transitions – limites de número de transitions poderia ser alocado numa divergência de seleção de seqüência varia de implementação para implementação. Esses limites deveriam ser contados na elaboração.

• Seqüências simultâneas – num evento que condições de múltiplas de transitions são validados como verdadeiro simultaneamente, diferentes implementações de SFC resultarão em comportamento diferente.

- Evitar dependência na ordem de execução – em alguns sistemas, o ramos da esquerda é ativada; em outras, todas as seqüências seguindo condições de transition verdadeiras são ativadas. Portanto, É considerado prática ruim de programação ter operação de seqüência simultânea depender da ordem de processamento de steps ativos.

- Usar seqüências simultâneas apenas quando sincronização é requerida – seqüências simultâneas são usadas quando processos paralelos precisam ser sincronizados no seu início e no seu fim. Onde seqüências assíncronas que não

exigem este tipo de sincronização são desejadas, eles poderiam ser codificados como SFCs independentes.

B.2.2 - Robustez

Robustez refere 'a capacidade de o programa sobreviver a desligamento ou outras condições não-antecipadas.

Transparência de diversidade funcional

SFCs são bem adequados para implementar algoritmos diversos. AND pode forçar diferentes steps para validar a mesma condição. OR pode ser usado para causar uma transição se for desejado programar o sistema tal que qualquer número de diversos algoritmos paralelos cause a transição.

- Ordem de execução – o projeto deve avaliar sobre impacto seguro da ordem de execução de diversos steps. A ordenação de SFC deve refletir a intenção do projeto.
- Interfaces – O projeto de sistema seguro deve avaliar todas as variáveis locais e globais necessárias para suportar processo replicado em arquivos transition. Como parte da implementação, deve ser verificado se alguma variável em arquivos transition será inicializado ou sobrescrito.

Tratamento de exceções

O nível, natureza e função de tratamento de exceções de SFC varia significativamente entre implementações. Função de tratamento de exceções em SFC abrange desde nada, através de

ativação de seqüência de falha designada, até a habilidade de para varrer o status de ativação de um diagrama SFC sob controle de opções do programa PLC não dentro do SFC.

Apesar de haver variações significativas, a seguinte diretriz aplica para a maioria das implementações SFC:

- Uso de GOTO ou JMP para assegurar a interrupção de fluxo de controle
- Evitar conflitos
- Comportamento do mecanismo de tratamento de exceções durante o processo step
- Comportamento do mecanismo de tratamento de exceções durante transition

Checagem de entrada e saída

Corrupção de dado num processo step ou transition pode ter sérias conseqüências se for permitido propagar para outros processos steps. SFCs não têm explicitamente mecanismo para checagem de entrada e saída. Entretanto, uma instrução geral poderia se aplicar para o programa steps e transitions implementado. A instrução específica para a checagem de entrada e saída é que deve ser tratado no nível da linguagem e não no nível de SFC. A propagação de erro pode ser reduzida se processo step usasse a checagem prévia racional para acertar variáveis utilizadas para outros steps.

B.2.3 - Manutenibilidade

Esta seção se discute a manutenibilidade para SFCs.

Legibilidade

Ela permite software ser entendido por desenvolvedores qualificados. É fundamental para segurança, pois ela permite rever e reduzir os erros durante manutenção

- Nomes de identificação descritivos – Muitos sistemas SFC permite dar nome a steps e transitions. Identificadores são usados para etiquetar os steps e transitions de SFC. Cada identificador refere ao arquivo de programa que contém processo step e transition. Os identificadores poderiam ser definidos para providenciar informações adequadas sobre a natureza e conteúdo de cada arquivo.

- Comentários e documentos internos

– Descrição de steps – descrições de processos step limpas e sem ambigüidade precisam ser utilizadas. Estas descrições poderiam incluir o processo executado pelo step, temporização e outras operações.

– Descrição de interfaces – as descrições das interfaces para cada step e transition devem incluir a identificação completa dos parâmetros de entrada, de variáveis globais (e qualquer efeito colateral) e parâmetros de saída.

- Descrição de condição de transição – as condições de transição devem ser claramente autenticadas. Todas as variáveis de saída e variáveis globais devem ser identificadas.

- Limitações de tamanho do sub-programa – as implementações SFC variam em limitações da soma de código contido num step.

  - Limitação de step simples para função simples – o código em step simples deve ser limitado para executar uma ação. Desde que cada SFC step é tipicamente uma subrotina que usa linguagem suportada por PLC, a regra para subrotina deve se aplicar a steps – uma função claramente definida. Múltiplas ações num step devem ser evitadas.

  - Limitação em transition para uma expressão – as condições de transition estão limitadas para uma expressão.

- Minimização de mistura de linguagem de programação – cada uma de linguagens IEC 1131-3 para PLC é específico para tratar certos problemas particulares do controle. Os programas simples estão menos sujeitos a erros e são mais manutenível do que aqueles que usam linguagens IEC 1131-3 para seus fins específicos.

  - Usar SFC para seqüênciamento – SFC é especialmente voltado para programação de seqüência de máquina. A notação SFC para este fim é mais adequada do que Ladder Logic ou Texto Estruturado.

- Não use SFC para intertravamento ou validação de relações lógicas – Ladder Logic é mais adequado para intertravamento e outras aplicações que exigem validação de relações booleanas. SFC não é adequado para esses fins.

- Não use SFC para operações matemáticas ou validação de relações matemáticas – Texto Estruturado é mais adequado de todos para esses fins.

- Minimizar programação obscura.

  - Evitar embutir subrotinas dentro de um step de SFC – um step sugere que certo subrotina de PLC será executa nesse step. Ao final do programa ou RET é executado por essa subrotina, o arquivo de transição é checado e o fluxo continua dali. Chamar subrotinas embutidas de qualquer linguagem a partir de step é obscuro, por causa de step já é uma subrotina.

  - Não use construções de SFC não relacionadas ao seqüenciamento – SFC é uma linguagem voltada para controle seqüencial. Há algumas construções SFC que permite outros usos para SFC. Esse tipo de construção deve ser evitada.

- Minimizar dispersão de elementos relacionados – dispersão pode ser um destaque de SFC devido a sua organização gráfica. Poucos detalhes são apresentados no nível do SFC e muitas ações relacionadas a variáveis são contidos nos arquivos step e transition. A dispersão pode ocorrer devido ao fato de step poder estar organizado em várias subrotinas, cada uma dentro de arquivos separados. A minimização da dispersão dos componentes críticos deve ser projetada durante o desenvolvimento.

Abstração de dado

Abstração de dado é a combinação de dado, operações em cima deste dado numa entidade única, estabelecimento de uma interface que permite acesso, manipulação e armazenamento do dado apenas através de operações permitidas. Isso reduz ou elimina os efeitos colaterais de troca de variáveis tanto durante a execução quanto na manutenção. Programas SFC providenciam uma abstração de sistema de controle para ser executado pelo PLC. E deve seguir as seguintes instruções:

- Minimizar o uso de variáveis globais – esta instrução tem limite para aplicar, pois PLCs utilizam apenas variáveis globais. Entretanto, há algumas aplicações que suportam a distinção entre variáveis globais e locais. Se variáveis locais forem suportadas por linguagem para implementação para step ou transition, elas deveriam ser utilizadas para todas as operações internas.

- Minimizar a complexidade de interfaces – a interface primária é a interação entre arquivos step e transition. Eles deveriam ser endereçados através de linguagem IEC 1131 de implementação.

Maleabilidade

Maleabilidade é a habilidade de sistema de software de acomodar modificações na exigência funcional. Para implementar sistema de software maleável, é necessário primeiro identificar o que vai ser constante e o que vai ser modificado, e então diferenciar o que vai ser modificado para ser facilmente notado e modificado para minimizar os efeitos colaterais.

Portabilidade

O advento de padronização de IEC-1131 software vai criar plataforma comum e o tratamento padronizado. Entretanto, isso poderia ser prejudicado pelos vendedores de hardware que adicionam extensões que rodam apenas nas suas próprias máquinas. A extensibilidade poderia ser útil para a aplicação, porém, inútil para a padronização.

Apenas sistemas SFC com IEC 1131-3 deveriam se utilizados. Sem o uso de constantes IEC 1131-3, SFC não vai ser portátil entre plataformas. As implementação de SFC variam. Por exemplo, GRAFCET difere de implementações americanas. SFC de Allen-Bradley não é uma implementação completa do padrão IEC 1131-3.

ANEXO C - CÓDIGOS-FONTES - PROTÓTIPO

C.1 - ARQUIVO TRANSLATOR.DSW

```
Microsoft Developer Studio Workspace File, Format Version 5.00
# WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###############################################################################
######
Project: "Translator"=".\Translator.dsp" - Package Owner=<4>
Package=<5>
{{{
}}}
Package=<4>
{{{
}}}
###############################################################################
######
Global:
Package=<5>
{{{
}}}
Package=<3>
{{{
}}}
###############################################################################
######
```

C.2 - ARQUIVO TRANSLATOR.DSP

```
# Microsoft Developer Studio Project File - Name="Translator" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 5.00
# ** DO NOT EDIT **
```

```
# TARGTYPE "Win32 (x86) Console Application" 0x0103

CFG=Translator - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Translator.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Translator.mak" CFG="Translator - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Translator - Win32 Release" (based on\
 "Win32 (x86) Console Application")
!MESSAGE "Translator - Win32 Debug" (based on\
 "Win32 (x86) Console Application")
!MESSAGE

# Begin Project
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Translator - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE"
/D "_MBCS" /YX /FD /c
# ADD CPP /nologo /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D
"_MBCS" /YX /FD /c
# ADD BASE RSC /l 0x416 /d "NDEBUG"
# ADD RSC /l 0x416 /d "NDEBUG"
BSC32=bscmake.exe
```

```
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:console /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:console /machine:I386

!ELSEIF  "$(CFG)" == "Translator - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 2
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /YX /FD /c
# ADD CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /D "_AFXDLL" /YX /FD /c
# ADD BASE RSC /l 0x416 /d "_DEBUG"
# ADD RSC /l 0x416 /d "_DEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept
# ADD LINK32 /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept

!ENDIF

# Begin Target

# Name "Translator - Win32 Release"
# Name "Translator - Win32 Debug"
# Begin Source File

SOURCE=.\main.cpp
```

```
# End Source File
# Begin Source File

SOURCE=.\main.h
# End Source File
# Begin Source File

SOURCE=.\TreatFile.cpp
# End Source File
# Begin Source File

SOURCE=.\TreatFile.h
# End Source File
# End Target
# End Project
```

C.3 - ARQUIVO MAIN.H

```
#include <stdio.h>
//#include <stdlib.h>
#include <iostream.h>
#include <afx.h>
//#include <afxwin.h>
```

C.4 - ARQUIVO MAIN.CPP

```
#include "main.h"
#include "TreatFile.h"

void main (void)
{
        Arquivo arqTxt1;

        arqTxt1.Ler("C:\\sample.txt");
        arqTxt1.Escrever("C:\\sample2.txt");
}
```

C.5 - ARQUIVO TREATFILE.H

```
#ifndef _TREATFILE_H_
#define _TREATFILE_H_

#include "afx.h"
#include <fstream.h>
#include <fcntl.h>
#include <io.h>


class Arquivo : public CObject          // Classe que le um arquivo texto e separa as partes
{
public:
        Arquivo(){}
        ~Arquivo(){}

        void Ler (CString NomeArquivo);
// Le o arquivo
        void Escrever(CString NomeArquivo);
// retorna o arquivo texto com os steps e transitions
        CString Mark(int i) { return sMark[i];}
// retona a parte referente a marca
        CString Box(int i)  { return sBox[i];}
// retona a parte referente ao box
        CString Transition(int i){ return sTransition[i];}
// retona a parte referente a transicao

private:
        // arrays para armazenar os boxes, transicoes e tipos de marcas
        CStringArray sMark;
        CStringArray sBox;
        CStringArray sTransition;

        CString TrataStep(CString strStep);
// tratamento do conteudo do step
// arcos de sinal de saida, gates
        CString TrataTrans(CString strTrans);
// conversao das condicoes de uma transicao – gates
// inibidores ou habilitadores
        void FormataStep(CString * strExpAT, CString strAux, int ntStAut);


};
#endif //_TREATFILE_H_
```

C.6 - ARQUIVO TREATFILE.CPP

```
#include "TreatFile.h"

CString Arquivo::TrataStep(CString strStep)
{
	CString strAux, strResp;
	int intNumST;

	intNumST = atoi(strStep.Left(2));
	strAux = strStep.Right(strStep.GetLength()-3);

	strResp.Format("s%d = TRUE\n", atoi(strStep.Left(2)));

	while( strAux != ";")//do
	{
		//strAux = strAux.Right(strAux.GetLength()-3);

		//CString str2 = strAux.Left(3);
		if ( /*str2*/ strAux.Left(3) == "OUT")
		{
			CString strTemp;//, str = strAux.Mid(4,2);
			strTemp.Format("OUTPUT%d == TRUE\n",
atoi(/*str*/strAux.Mid(4,2)));
			strResp +=strTemp;
			strAux = strAux.Right(strAux.GetLength()-7);
		}


	}

	return strResp;
}

void Arquivo::FormataStep(CString *strExpAT, CString strAux, int intStAut)
{
	*strExpAT += strAux;
	*strExpAT += TrataStep(Box(intStAut-1));//.Right(Box(intStAut-1).GetLength()-
3));//sBox[intStAut-1].Right(sBox[intStAut-1].GetLength()-3));
	*strExpAT += "</step>\n";
}


CString Arquivo::TrataTrans(CString strTrans)
{
```

```
        CString strAux, strResp;

        strAux = strTrans;

        while( strAux != ";")
        {

                if ( strAux.Left(3) == "HGT")
                {
                        CString strTemp;
                        strTemp.Format(" == S%d", atoi(strAux.Mid(4,2)));
                        strResp +=strTemp;
                        strAux = strAux.Right(strAux.GetLength()-7);
                }
                if ( strAux.Left(3) == "NGT")
                {
                        CString strTemp;
                        strTemp.Format(" == NOT(S%d)", atoi(strAux.Mid(4,2)));
                        strResp +=strTemp;
                        strAux = strAux.Right(strAux.GetLength()-7);
                }
                if ( strAux.Left(3) == "HAE")
                {
                        CString strTemp;
                        strTemp.Format(" == INPUT%d", atoi(strAux.Mid(4,2)));
                        strResp +=strTemp;
                        strAux = strAux.Right(strAux.GetLength()-7);
                }
                if ( strAux.Left(3) == "NAE")
                {
                        CString strTemp;
                        strTemp.Format(" == NOT(INPUT%d)", atoi(strAux.Mid(4,2)));
                        strResp +=strTemp;
                        strAux = strAux.Right(strAux.GetLength()-7);
                }


        }

        strResp += "\n";

        return strResp;
}

/**/
void Arquivo::Ler( CString NomeArquivo )
```

```
{
        CString Temp;
        char temp[256];
        ifstream ifile;
// Inicializacao
        ifile.open((char*)(LPCSTR) NomeArquivo, ios::nocreate);                    //abro    o
arquivo para leitura


// Leitura do arquivo texto com o modelo do grafo MFG
        do
        {
                ifile.get(temp, 256);              //pego uma linha inteira    em  um  char[]  e
copio para um CString
                Temp = temp;
                ifile.get();                                  //pego o caractere de pula-linha

                if (Temp == "<mark>")          // se far a area dar marcas
                {                                                          // leio até chegar no tag do
final da area
                        do
                        {
                                ifile.get(temp, 256);
                                Temp = temp;

                                if (Temp != "</mark>")
                                        sMark.Add(Temp);
                                ifile.get();

                        }while (Temp != "</mark>");

                }
                else
                {
                        if (Temp == "<box>")           // se for a area dos boxes
                        {                                                      // leio até chegar no tag
do final da area
                                do
                                {
                                        ifile.get(temp, 256);
                                        Temp = temp;

                                        if (Temp != "</box>")
                                                sBox.Add(Temp);
                                        ifile.get();
```

```
                    }while (Temp != "</box>");
                }
                else
                {
                    if (Temp == "<transition>")          // se for transicao
                    {                                                    //
leio até chegar no tag do final da area
                        do
                        {
                            ifile.get(temp, 256);
                            Temp = temp;

                            if (Temp != "</transition>")
                                sTransition.Add(Temp);
                            ifile.get();

                        } while (Temp != "</transition>");
                    }
                    else
                        ifile.get();
                }
            }

    }while(ifile.good()); // leio enquanto o arquivo teiver condicoes de ser lido
                          // (nao esta corrompido ou atingiu o eof


// Finalizacao
    ifile.close();          // fecho o arquivo

}

void Arquivo::Escrever(CString NomeArquivo)
{

    int intTrans, intStep, intContT, intContS;
    CString strTemp, str;
    ofstream ofile;
    bool *blStepPrt;
    bool blPosTrans;

// Inicializacao das variaveis

    intTrans = sTransition.GetSize( ) ;
    intStep =  sBox.GetSize( ) ;
```

```
        // aloco o vetor de flags de impressao dos steps
        blStepPrt = new bool [intStep];

        for(int i = 0; i < intStep; i++)
                blStepPrt[i] = true;

        ofile.open("c:\\output.txt", ios::app); // abro o arquivo para escrita

        if(!ofile.good())
        {
                delete [] blStepPrt;  // deleto o vetor de flags
                return;
        }

        intContT = 1;
        intContS = 1;

//      Pego os vetores correspondentes aos boxes e transicoes e imprimo o fonte simplificado em sfc
        do
        {
                // pego a transicao
                strTemp = Transition(intContT-1);//sTransition[intContT-1];

                // variaveis auxiliares
                CString strExp, strExpAT, strExpDT, strUno, strAux;
                int intStAut, intContCh;

                // Inicializacao das variaveis auxiliares
                strAux = "";
                strUno = ",";
                intContCh = 0;
                blPosTrans = false;

                do
                {

                        intStAut = atoi((LPCSTR)strTemp.Left(2));// pego os caracteres correspondentes ao numero do step
                        intContCh +=2;

                        strAux.Format("<step %d>\n",intStAut);

                        if (strUno ==  "," ) // verifico se há um box 'entrando' na transicao ou 'saindo' da transicao
                        {
```

```
                        if(blStepPrt[intStAut-1] == true)  // se a flag de impressao
do box e'OK eu imprimo no arquivo
                        {
                                blStepPrt[intStAut-1] = false;        // seto esse step
como lido

                                if (blPosTrans == false)        // verifico se o step e'
antes da transicao  ou depois
                                {
                                        // coloco o step no string correspondente a
antes da transicao
                                        FormataStep(&strExpAT, strAux, intStAut);

                                //      strExpAT += strAux;
                                //      strExpAT      +=      TrataStep(Box(intStAut-
1));//.Right(Box(intStAut-1).GetLength()-3));//sBox[intStAut-1].Right(sBox[intStAut-
1].GetLength()-3));
                                //      strExpAT += "</step>\n";
                                }
                                else
                                {
                                        // coloco o step no string correspondente a
antes da transicao
                                        FormataStep(&strExpDT, strAux, intStAut);
                                        //strExpDT += strAux;
                                        //strExpDT      +=      TrataStep(Box(intStAut-
1));//.Right(Box(intStAut-1).GetLength()-3));//sBox[intStAut-1].Right(sBox[intStAut-
1].GetLength()-3));
                                        //strExpDT += "</step>\n";
                                }

                        }
                }
                else
                {
                        if (strUno == "-") // verifico o caractere
correspondente a passagem de antes da transicao
                        {                                       //para depois da
transicao
                                blPosTrans = true ;     // seto que os steps
seguintes serao apos a transicao e
                                                                        //
        coloco o step no string correspondente a antes da transicao
                                                                                //
verificando se ele ja foi impresso ou nao
                                if(blStepPrt[intStAut-1] == true)
```

```
                                        {
                                                blStepPrt[intStAut-1] = false; // seto
esse step como lido
                                                FormataStep(&strExpDT,      strAux,
intStAut);
                                                //strExpDT += strAux;
                                                //strExpDT += TrataStep(Box(intStAut-
1));//.Right(Box(intStAut-1).GetLength()-3));//sBox[intStAut-1].Right(sBox[intStAut-
1].GetLength()-3));
                                                //strExpDT += "</step>\n";
                                        }
                                }
                        }


                        strTemp = strTemp.Right(strTemp.GetLength()-2);  //

                        strUno = strTemp.Left(1); // Pego o caractere que mostra a posicao
do step (antes ou depois da transition)
                        intContCh +=1;

                        strTemp = strTemp.Right(strTemp.GetLength()-1);

                }while (strUno != ":"); //(strTemp.GetLength()>0);

                // Formato a saida correspondente a transicao dada, com os steps antes e
depois
                strExp.Format("%s\n<trans %s>\n%s\n</trans>\n\n%s",
                        strExpAT,
                        Transition(intContT-1).Left(intContCh-1),//sTransition[intContT-
1].Left(intContCh-1),
                        TrataTrans(strTemp),
                        strExpDT);

                ofile << (char*)(LPCSTR)strExp;    // mando para o arquivo

                intContT++;

        }while (intContT <= intTrans);       // comparo a transicao atual com a qt de
transicoes

//Finalizacao
        ofile.close();              // fecho o arquivo

        delete [] blStepPrt;  // deleto o vetor de flags
}
```

CAPÍTULO 7 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] Miyagi, P. E. **Controle Programável - Fundamentos do Controle de Sistemas a Eventos Discretos**, Ed. Edgard Blüncher, São Paulo, p. 149-183, 1996.

[2] Lewis, R. W. **Programming using IEC 1131-3**, 1998.

[3] Santos Filho, D. J. **Proposta de Mark Flow Graph Estendido para a modelagem e controle de sistemas integrados de manufatura**, Dissertação de Mestrado, Escola Politécnica da USP, São Paulo, 1993.

Os seguintes websites também foram consultados:

[4] http://www.nrc.gov/NRC/NUREGS/CR6463/appa.htm

[5] http://www.nrc.gov/NRC/NUREGS/CR6463/ch5.htm

[6] http://www.nrc.gov/NRC/NUREGS/CR6463/ch6.htm

[7] http://www.nrc.gov/NRC/NUREGS/CR6463/ch10.htm

[8] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131.html

[9] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-01.html

[10] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-02.html

[11] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-03.html

[12] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-04.html

[13] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-05.html

[14] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-06.html

[15] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-07.html

[16] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-08.html

[17] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-09.html

[18] http://www.isa.org/isaolp/journals/g-html/other/9906iec1131-10.html

[19] http://www.asapinc.com/white_papers/IEC1131%20Article.htm