DANIEL DE MIRANDA E SILVA FERREIRA
MARCO ANTÔNIO DE ALMEIDA SILVA

Especificação e Desenvolvimento de um Controlador E-MFG para Sistemas Flexíveis de Produção

Trabalho de formatura apresentado à
Escola Politécnica da Universidade de São Paulo

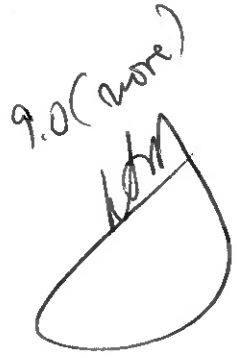
São Paulo
1998

DANIEL DE MIRANDA E SILVA FERREIRA
MARCO ANTÔNIO DE ALMEIDA SILVA

Especificação e Desenvolvimento de um Controlador E-MFG para Sistemas Flexíveis de Produção

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo

Área de Concentração:
Engenharia Mecânica - Automação e Sistemas
Orientador:
Prof. Dr. Diolino José dos Santos Filho



São Paulo
1998

Aos nossos pais, pela oportunidade de realizarmos este curso e a todos aqueles que nos apoiram nesta conquista

AGRADECIMENTOS

Ao Prof. Dr. Diolino José dos Santos Filho, pelo apoio, orientação e confiança durante o decorrer do curso.

Aos colegas Engenheiros André, Guilherme, Jairo, Jaime, João, Maurício, Ricardo, Valdir pela amizade e apoio.

Aos Engenheiros Marcio e Fábio pela colaboração, revisões e discussões que muito auxiliaram nesta conquista.

Ao Engenheiro Vinícius pela confiança no sucesso.

A todos na Mecatrônica pelo apoio.

À Giselle pelo apoio e auxílio nas revisões e pela confiança no sucesso.

Enfim, a todos aqueles que de alguma forma colaboraram na execução deste trabalho.

Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia Mecânica - Automação e Sistemas

Especificação e Desenvolvimento de um Controlador E-MFG para Sistemas Flexíveis de Produção

Orientador: Prof. Dr. Diolino José dos Santos Filho
Daniel de Miranda e Silva Ferreira *NUSP: 221111*
Marco Antônio de Almeida Silva *NUSP: 1517676*

São Paulo
Dezembro de 1998

ÍNDICE

ÍNDICE DE FIGURAS

Resumo

O controle de processos de manufatura flexíveis exige a utilização de ferramentas que permitam a modelagem e simulação de sistemas produtivos e seus algoritmos de controle. No entanto, atualmente existe uma barreira entre as etapas de modelagem e simulação do sistema e a implementação real do controle devido ao fato de se utilizar técnicas diferentes em cada fase. O E-MFG é uma ferramenta adequada para a modelagem, simulação e especificação do controle de Sistemas de Eventos Discretos (e.g. sistemas flexíveis de manufatura). Neste trabalho, procura-se especificar e implementar um controlador programável, cuja a estratégia de controle é descrita através de um grafo E-MFG bem como especificar uma linguagem textual de programação para a descrição de algoritmos de controle em E-MFG.

"Abstract"

The control of flexible manufacturing systems requires the use of tools for modeling and simulating the production systems and their control algorithms. However, there is a gap between the stage of modeling and simulation and the actual implementation of the control strategy due to the use of different techniques in each stage. The E-MFG is a proper tool for modeling, simulating and specifying the control of Discrete Event Systems (e.g. flexible manufacturing system). The aims of this work are the specification and implementation a programmable controller whose control strategy is described through a E-MFG graph, as well as the specification of a textual programming language for the description of control algorithms in E-MFG.

1 - Introdução

Em ambientes automatizados e integrados de produção identifica-se claramente 3 níveis hierárquicos de controle da produção. (SANTOS FILHO, 1993)

O primeiro nível é caracterizado pelo controle das variáveis contínuas do processo produtivo (e.g. velocidade de corte em uma indústria de manufatura ou controle de temperatura e pressão em uma indústria de processos). Para o controle dos sistemas nesse nível utiliza-se técnicas de controle de variáveis contínuas (controladores PID, I-PD, PI, equações de estado, Redes Neurais Artificiais, etc.) pois essas variáveis são regidas pelas leis físicas e seu comportamento dinâmico pode ser analisado por modelos matemáticos baseados em seu comportamento em função do tempo, através de equações diferenciais e modelos de integração numérica. A implementação desses sistemas de controle se faz, normalmente, através de sistemas dedicados ou controladores lógicos programáveis e em muitas situações através de *'hardware'* dedicado, fornecido pelo próprio fabricante do equipamento que se deseja controlar.

O segundo nível corresponde a seqüencialização e o intertravamento das ações necessárias à realização do processo produtivo. Portanto, está intimamente ligado a atividades de movimentação de materiais (gestão da cadeia de suprimentos ou *'supply chain'* interno da unidade produtora), carga e descarga dos *'pallets'*, preparação (*'setup'*) de máquinas e ferramentas, diagnose e reparo de falhas, supervisão e coordenação dos eventos de produção, bem como monitoração do sistema produtivo. Logo, verifica-se que as regras de comportamento dinâmico dos sistemas deste nível são determinadas pelo homem, não obedecendo a leis invariantes da Física, tal como ocorre com os Sistemas de Variáveis Contínuas (SVCs). Assim, a característica fundamental dos sistemas neste nível é o fato da sua evolução dinâmica ser determinada pela seqüência de eventos ocorridos, sendo que estes não possuem necessariamente um instante de tempo pré-determinado para a sua ocorrência. Esta classe de sistemas, os Sistemas de Eventos Discretos (SEDs) não podem ser representados adequadamente através das técnicas convencionais aplicadas a Sistemas de Variáveis Contínuas. Devido a essa dificuldade, desenvolveram-se modelos para esses sistemas fundamentados em Redes de Filas, autômatos finitos, modelos algébricos (Álgebra Min-Max) e Redes de Petri. A implementação do

controle neste nível se dá normalmente através de computadores industriais e CLP's, de acordo com o grau de flexibilidade exigida do sistema. A natureza do controle realizado neste nível pode ser classificada como controle da seqüência e regras de produção e portanto, está intimamente ligado a organização e execução do processo produtivo.

O terceiro nível de controle corresponde às atividades de programação de produção, planejamento da produção e gestão da cadeia de fornecedores (*'supply chain'*) da unidade produtora. Assim, de acordo com o nível anterior, o comportamento do sistema é determinado pelo homem. No entanto, esses sistemas apresentam uma necessidade de inteligência e flexibilidade ainda maior pois os modelos necessitam adequar-se a restrições variáveis de produção e suprimentos, e gerar como saída uma seqüência ótima de ordens de produção e pedidos de compra. A modelagem dos sistemas neste nível depende de fatores como escala e tipo de produção, bem como de fatores estratégicos de cada companhia, sendo portanto muito difícil a obtenção de um modelo adequado baseado em uma única técnica, principalmente quando o objetivo é unificar as etapas de planejamento e controle aplicando-se uma ferramenta única para o desenvolvimento do sistema. Logo, para o planejamento de produção, por exemplo, a modelagem e otimização desses sistemas abrange técnicas baseadas em redes de filas, modelos de regras de inteligência artificial, técnicas de identificação do caminho crítico de produção (e.g. grafos PERT/CPM, entre outras) a fim de se estabelecer a melhor estratégia de produção. A implementação do controle neste nível se dá através de computadores, geralmente máquinas com grande poder de processamento. Esse nível corresponde ao nível de controle da estratégia de produção.

As referências neste trabalho ao controle e flexibilidade de Sistemas Integrados de Manufatura (SIM's) ou mais genericamente a Sistemas Integrados de Produção, se aplicam ao segundo nível de controle descrito, isto é, ao controle da seqüencialização, das regras e execução da produção.

1.1 - Automação da Manufatura e Sistemas Flexíveis de Manufatura

O processo de manufatura pode ser encarado como um conjunto de processos de transformação de matéria prima que visa a obtenção de um produto acabado que satisfaça as especificações técnicas preestabelecidas. Identifica-se nesse processo dois tipos de atividades que contribuem para a obtenção do produto final: as atividades mecânicas repetitivas e as atividades decisórias (de avaliação entre o início e o fim de uma atividade, seqüencialização das atividades, avaliação da qualidade, enfim as regras de produção).(SANTOS FILHO, 1993)

A fim de se automatizar esse processo pode-se adotar três abordagens distintas, de acordo com a natureza do processo: a automação fixa, a automação programável e a automação flexível.

A automação fixa caracteriza-se pela automatização de uma seqüência do processo produtivo, e está fortemente associada à configuração das máquinas e equipamentos que compõem a linha. É caracterizada também por altas taxas produtivas e uma baixa adaptabilidade a mudanças no processo (exigindo normalmente longos intervalos de tempo de *'setup'* para pequenas variações no produto, uma vez que pode implicar em alterações físicas na configuração da linha e componentes da máquina).

A automação programável é o passo seguinte à automação fixa, podendo-se programar novos processos e portanto diminuir o tempo de preparação das máquinas.

Já a automação flexível implica na capacidade de se processar uma variedade de produtos sem que haja perda de tempo com a adaptação do sistema durante a mudança de produtos, diferenciando-se da automação programável pela inexistência de perdas de tempo com reprogramação e *'setup'* das máquinas.

Assim, a flexibilidade de um sistema de manufatura pode ser avaliada em função da flexibilidade de alguns parâmetros (SANTOS FILHO, 1993):

- Flexibilidade da Máquina-Ferramenta (variedade de peças que podem ser processadas)
- Flexibilidade do sistema de transporte de materiais (habilidade de se manipular peças por diferentes rotas)
- Flexibilidade do sistema de computacional (grau de adaptabilidade do sistema para a integração de novas funções)

- Flexibilidade funcional (capacidade de processamento de um mix de produtos e variações de roteamento de produtos dentro de um mesmo processo)

1.2 - A aplicação do MFG na modelagem e controle de SIM's

Modelar um sistema consiste em descrever o seu comportamento dinâmico. O modelo de um sistema pode representar os seus elementos e interconexões (modelo estrutural) ou o funcionamento de um sistema (modelo funcional).

Sistemas de manufatura possuem a característica de ter o seu comportamento dinâmico definido pela transição de estados resultante da ocorrência de eventos discretos, estando portanto o problema de controlar esses sistemas relacionado com a supervisão dos estados a fim de se assegurar uma determinada seqüência de eventos ou regra de produção.

A capacidade de representação formal e concisão das Redes de Petri [(PETERSON, 1981), (REISIG, 1985), (REISIG, 1992)] as tornam adequadas para a modelagem desses sistemas, assegurando a possibilidade de se representar a dinâmica desses sistemas (e suas regras de produção) de forma estrutural e hierárquica.

Logo, o MFG ou Mark Flow Graph (HASEGAWA, 1984) sendo um grafo derivado das Redes de Petri torna-se uma ferramenta ideal para a modelagem de SIMs, pois além de possuir toda a representatividade das Redes de Petri o MFG possui elementos inerentes ao modelo para a representação dos sinais da planta e dos sinais de controle (os '*gates*').

1.3 - Motivação e Objetivos do Trabalho

O controle de processos de manufatura flexíveis exige a utilização de ferramentas que permitam a modelagem e simulação de sistemas produtivos e seus algoritmos de controle. No entanto, atualmente existe uma barreira entre a modelagem e simulação do sistema e a especificação do sistema de controle devido ao fato de se utilizar técnicas diferentes para a modelagem, simulação e controle (e.g. utiliza-se Redes de Filas para a simulação e '*Ladder Diagram*' para o controle). Neste contexto, a capacidade de representação do E-MFG (Enhanced Mark Flow Graph) o torna uma ferramenta ideal pois permite a modelagem, simulação e especificação do algoritmo de controle de SEDs (SANTOS FILHO, 1993). O objetivo deste trabalho é especificar e implementar um controlador cuja a estratégia de

controle é descrita através de um grafo E-MFG. Será necessário, também, especificar uma linguagem textual de programação para a descrição do algoritmo de controle em E-MFG.

1.4 - Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- Capítulo 1 - É realizada uma introdução a respeito da natureza dos sistemas de manufatura e os problemas relacionados ao seu controle
- Capítulo 2 - Apresenta-se a técnica de modelagem de sistemas de eventos discretos através do PFS/MFG e PFS/E-MFG.
- Capítulo 3 - Procura-se detalhar as estratégias de controle de sistemas produtivos bem como analisar os requerimentos de controle desses sistemas e a adequação do E-MFG ao controle desses sistemas.
- Capítulo 4 - São apresentadas as especificações do controlador e analisadas as alternativas de implementação do controlador
- Capítulo 5 - É apresentada a linguagem de programação E-MFG Script
- Capítulo 6 - Procura-se especificar a estrutura de dados e a funcionalidade dos módulos do controlador
- Capítulo 7 - Procura-se detalhar a estrutura da implementação do controlador
- Capítulo 8 - Define-se os procedimentos para os testes do programa
- Capítulo 9 - Comentários e Conclusões
- Apêndice I - Apresenta o cronograma de desenvolvimento
- Apêndice II - Apresenta alguns dos testes de validação realizados
- Apêndice III - Apresenta o manual do aplicativo
- Apêndice IV - Apresenta o código fonte do aplicativo escrito em Visual C++ 4.0

2 - Modelagem de Sistemas Integrados de Manufatura

2.1 - Introdução

A capacidade de representação e a concisão são requisitos básicos de qualquer ferramenta que se utilize para a modelagem e especificação de controle de um SIM. Neste contexto, as técnicas de modelagem de sistemas de eventos discretos derivadas de Redes de Petri como o MFG e as suas extensões como F-MFG, ou MFG Funcional, (HASEGAWA; TAKHASHI, 1987) e o E-MFG (Enhanced Mark Flow Graph) constituem uma classe de ferramentas com essas características que permitem a representação do modelo global ou modelo estrutural-funcional do sistema (SANTOS FILHO, 1993).

Neste capítulo pretende-se realizar uma introdução à modelagem de sistemas de eventos discretos através do PFS/MFG (Production Flow Schema/Mark Flow Graph) (MIYAGI, 1988), discutir as limitações do MFG para a modelagem de sistemas que apresentam regras flexíveis de produção e, por fim, realizar uma introdução à modelagem de Sistemas Flexíveis de Manufatura através do PFS/E-MFG.

2.2 - Modelagem utilizando o MFG

O MFG (Mark Flow Graph) é um grafo bipartido seguro derivado de Redes de Petri que permite a representação de sistemas de eventos discretos complexos (e.g. Sistemas de Manufatura).

O grafo é composto por elementos estruturais que representam as condições e os eventos associados ao processo que se deseja modelar. Os elementos estruturais que compõe o grafo estão representados na figura 2.2-1.

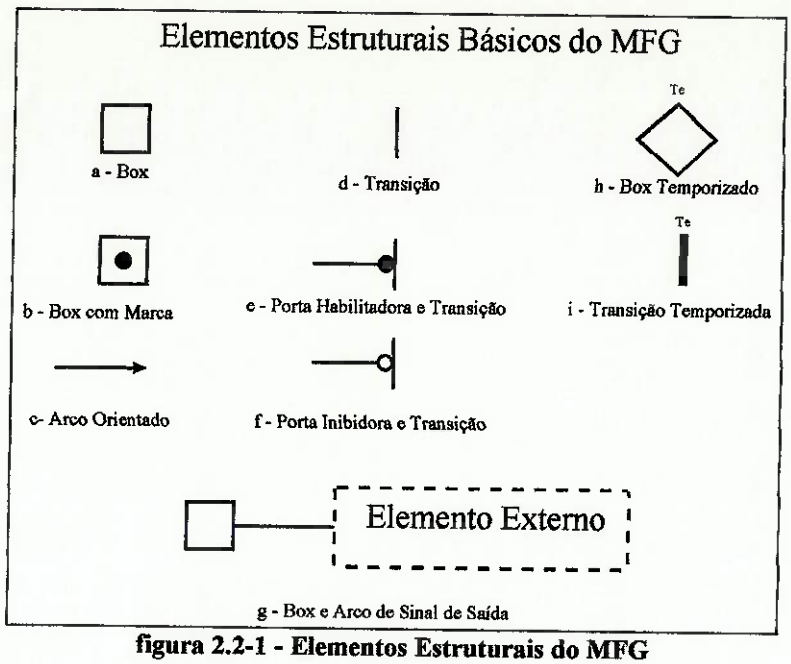


figura 2.2-1 - Elementos Estruturais do MFG

Os elementos estruturais representados na figura 2.2-1 possuem a seguinte funcionalidade:

- Box: corresponde a uma condição relacionada com a disponibilidade de recursos ou modo de operação do sistema (representado por um quadrado);
- Transição: corresponde a um evento que causa uma mudança no estado do sistema (representada por uma barra vertical);
- Arco Orientado: elemento que relaciona um box e uma transição, definindo o pré evento e o pós-evento de cada condição (são representados por setas);
- Marca: representa a manutenção de uma condição (a existência de uma marca em um box indica que esta condição está satisfeita) e alocação das marcas em um grafo constituem a sua marcação;
- Porta: existem basicamente dois tipos de portas, isto é, as habilitadoras e as inibidoras. As habilitadoras habilitam o disparo de uma transição se o sinal de origem corresponder ao nível lógico '1' e desabilitam o disparo da transição se o sinal de origem corresponder ao nível lógico '0'. Por outro lado as portas inibidoras habilitam o disparo de uma transição se o sinal de origem corresponder ao nível lógico '0' e inibem o disparo da transição se o sinal de origem corresponder ao nível lógico '1'. As portas podem ainda ser subclassificadas em internas e externas sendo que as internas estão sempre conectadas a um box onde a presença de marca indica nível lógico '1' e a ausência nível lógico '0'; as portas

externas estão associadas a um elemento externo ao modelo que gera um sinal binário;

- Arco de Sinal de Saída: é constituído por um arco conectado a um box que gera um sinal binário para um elemento externo ao modelo, onde a presença de marca no box indica nível lógico '1' e a ausência nível lógico '0';
- Elementos temporizados: a fim de se representar o tempo de duração de processos dispõe-se de dois elementos temporizados:
  - boxes temporizados: que retêm a marca por um intervalo de tempo pré-determinado (representados por losangos e uma constante de tempo);
  - transições temporizadas: que atrasam o seu disparo por um período de tempo pré-determinado a partir do instante em que as condições de disparo estão satisfeitas. Se as condições não estiverem mais satisfeitas antes que ocorra o disparo da transição esta passa ao estado de desabilitada e a contagem de tempo retorna ao zero.

Além desses elementos estruturais básicos foram propostas extensões que se baseiam em um agrupamento desses elementos e constituem os macro-elementos do MFG funcional ou F-MFG ilustrados na figura 2.2-2.

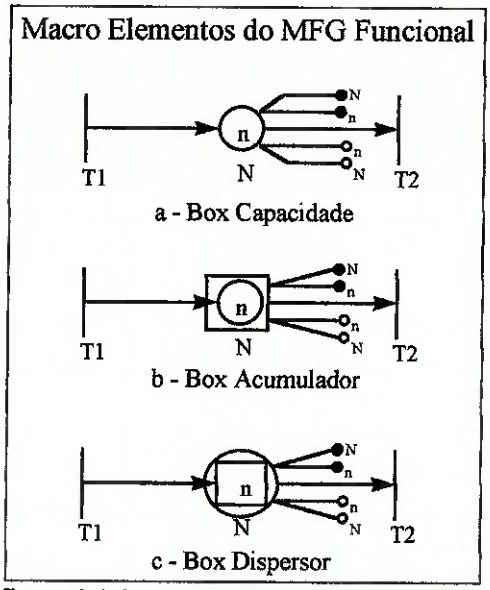


figura 2.2-2 - Macro Elementos do F-MFG

Esses elementos permitem a modelagem de *'buffers'* e operações de agrupamento e desagrupamento de marcas (e.g. montagem e desmontagem de *'pallets'*) e são constituídos por:

- Box Capacidade: o box capacidade (representado por um círculo) representa um *'buffer'* com capacidade de armazenamento igual a N. Assim, a transição T1 pode disparar sempre que o número de marcas no box for menor que N e T2 pode disparar sempre que existir alguma marca armazenada. Os *'gates'* ou portas com índice N possuem sinal lógico '1' na origem se o número de marcas no Box for igual a N e os *'gates'* ou portas com índice n possuem sinal lógico '1' na origem se existir alguma marca armazenada no box;
- Box agrupador: representa uma operação de agrupamento de N peças. Assim a transição T1 pode disparar sempre que o número de marcas no box for menor que N e T2 pode disparar sempre que o número de marcas for igual a N. Os *'gates'* ou portas com índice N possuem sinal lógico '1' na origem se o número de marcas no Box for igual a N e os *'gates'* ou portas com índice n possuem sinal lógico '1' na origem se existir alguma marca armazenada no box;
- Box dispersor: representa uma operação de desagrupamento de N peças. Assim a transição T1 pode disparar sempre que o número de marcas no box for igual a zero e T2 pode disparar sempre que o número de marcas for maior que zero. Os *'gates'* ou portas com índice N possuem sinal lógico '1' na origem se o número de marcas no Box for igual a N e os *'gates'* ou portas com índice n possuem sinal lógico '1' na origem se existir alguma marca armazenada no box;

Com os elementos estruturais descritos é possível modelar sistemas de eventos discretos complexos e simular a evolução dos estados do sistema.

O estado do sistema modelado através do MFG é dado pela disposição das marcas no grafo (marcação). A evolução das marcas no grafo é determinada pelo disparo das transições ou eventos.

Uma transição é considerada habilitada para disparo se e somente se:

1 - todos os boxes de entrada desta transição estiverem marcados

2 - todos os boxes de saída desta transição estiverem sem marcas

3 - todas as portas habilitadoras internas ligadas nesta transição tiverem sinal de origem '1'

4 - todas as portas inibidoras internas ligadas nesta transição tiverem sinal de origem '0'

Uma transição é considerada disparável se além de obedecer as condições que a tornam habilitada ela obedecer as seguintes condições:

1 - todas as portas habilitadoras externas ligadas nesta transição tiverem sinal de origem '1'

2 - todas as portas inibidoras externas ligadas nesta transição tiverem sinal de origem '0'

Se a transição é disparável ocorre o disparo de maneira imediata e instantânea, a menos que haja um conflito ou atraso de tempo (transição temporizada ou box temporizado). O disparo de uma transição remove todas as marcas dos boxes de entrada da transição e coloca marcas nos boxes de saída da transição, como ilustra a figura a 2.2-3.

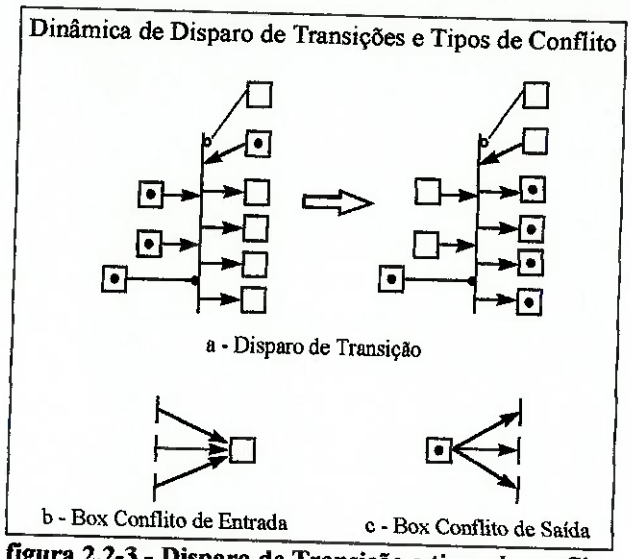


figura 2.2-3 - Disparo de Transição e tipos de conflito

A figura 2.2-3 representa também um tipo de situação onde dois ou mais eventos estão ativados e a ocorrência de um implica na desativação dos outros. Em outras palavras, a situação de conflito ocorre quando existe a convergência ou divergência de eventos mutuamente exclusivos. Nestas situações, existem duas linhas de ações que se pode seguir.

A primeira linha de ações se aplica quando os conflitos são inerentes ao sistema e não é interessante dar prioridade a um dos eventos. A ação tomada neste caso é não arbitrar o conflito e o evento que será disparado é sorteado aleatoriamente.

A segunda linha de ações se aplica quando se deseja arbitrar efetivamente o conflito e consiste em se impor uma dinâmica de disparo ao sistema através de um árbitro externo ao grafo conectado ao grafo por portas habilitadoras ou inibidoras externas ou através de um sub-grafo que descreve a estratégia de resolução do conflito conectado às transições em conflito por meio de portas inibidoras ou habilitadoras.

2.3 - Modelagem hierárquica utilizando o PFS/MFG

Com as estruturas existentes no F-MFG é possível descrever o comportamento de sistemas de eventos discretos complexos. No entanto, modelar uma complexa planta industrial a partir de seu nível máximo de detalhe é uma tarefa árdua e sujeita a muitas iterações em virtude de uma modelagem incompleta, visto que sempre deve-se ter o nível máximo de detalhe em todas as atividades que compõem o processo produtivo.

No sentido de se facilitar a modelagem de sistemas complexos foi proposta uma metodologia de modelagem hierárquica para a modelagem de SIM's, o PFS/MFG (MIYAGI, 1988), baseada na técnica de Production Flow Schema (PFS), que é uma rede do tipo canal-agência adequada para a modelagem de sistemas produtivos (REISIG, 1992).

No PFS tem-se basicamente 3 tipos de elementos estruturais, os elementos atividade (representados por colchetes), os elementos distribuidores (representados por círculos) e os arcos orientados (representados por setas). Os elementos atividades representam os elementos ativos do sistema, ou seja aqueles que realizam alguma atividade relativa à produção, transporte e modificação de itens. Os elementos distribuidores são os elementos passivos do sistema e, portanto, não realizam transformação de materiais sendo apenas capazes de armazenar itens que serão utilizados nas atividades. Por fim, os arcos orientados determinam as relações de conexão entre os elementos distribuidores e as atividades. Na figura 2.3-1 tem-se os elementos estruturais básicos do PFS.

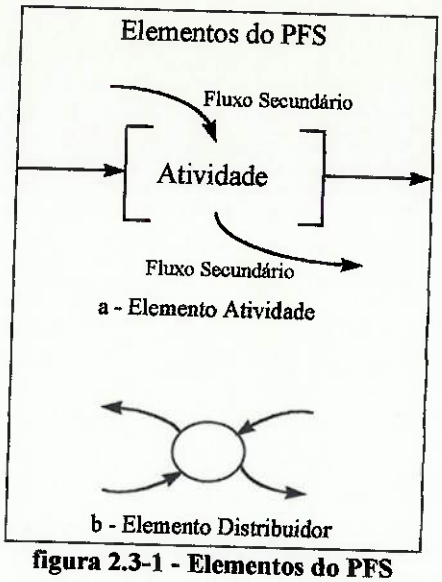


figura 2.3-1 - Elementos do PFS

A metodologia PFS/MFG estabelece a modelagem do fluxo principal de produção e um detalhamento hierárquico das atividades e elementos de distribuição (substituindo-as gradativamente por sub-redes MFG). Por exemplo, pode-se considerar uma planta composta por 3 magazines, uma célula de manufatura, uma fresadora CNC e um veículo autônomo de transporte (VAT ou AGV) que realiza o fluxo de materiais entre a célula e a fresadora. A figura 2.3-2 apresenta uma concepção estrutural do modelo e o modelo PFS correspondente e na figura 2.3-3 tem-se o detalhamento das atividades na fresadora.

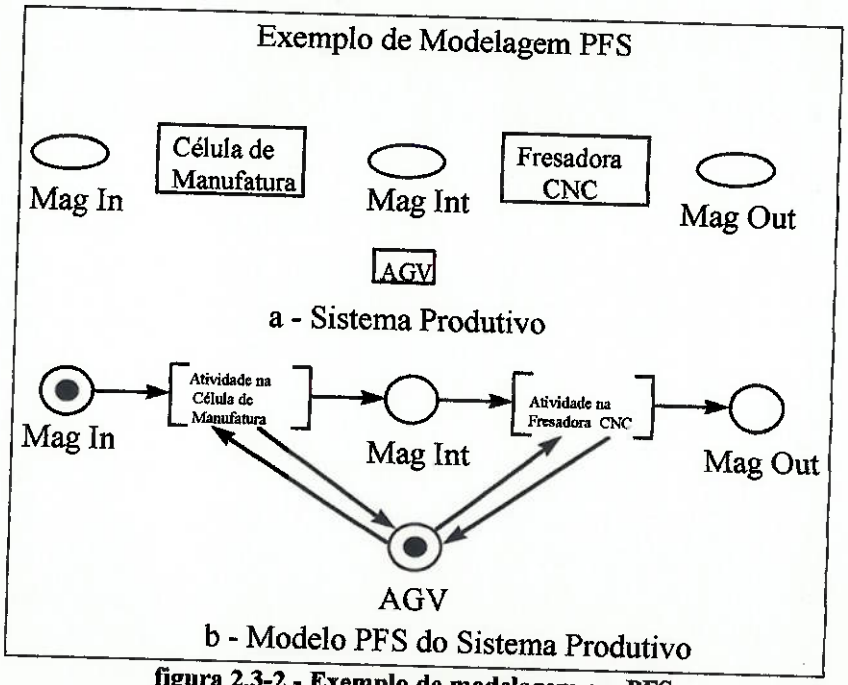


figura 2.3-2 - Exemplo de modelagem em PFS

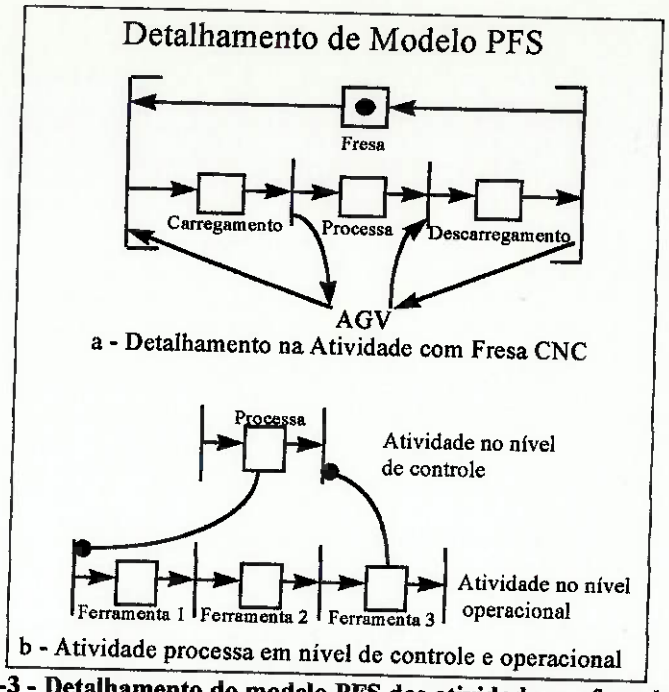


figura 2.3-3 - Detalhamento do modelo PFS das atividades na fresadora CNC

2.4 - Considerações quanto a Sistemas Flexíveis de Manufatura

A metodologia PFS/F-MFG apresentada na seção 2.3 deste capítulo propõe uma abordagem hierárquica para a modelagem do sistema produtivo, o que contribui significativamente para a obtenção de um modelo de forma progressiva e estruturada. No entanto, devido a ausência de flexibilidade modeladora do MFG a obtenção de um modelo através dessa ferramenta ainda pode ser muito difícil dependendo do grau de complexidade e flexibilidade do sistema. Essa ausência de flexibilidade modeladora do F-MFG pode ser ilustrada pelo seguinte exemplo: considere um sistema que processa 2 tipos de peças (A e B) composto por um robô manipulador, uma máquina de comando numérico e dois magazines (um de entrada e um de saída), como o ilustrado na figura 2.4-1.

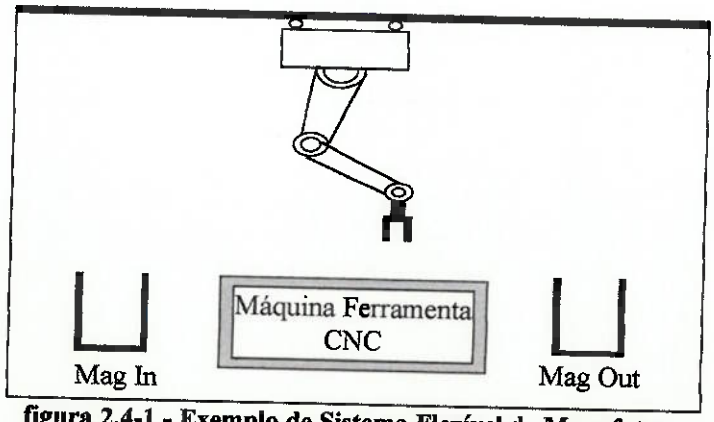


figura 2.4-1 - Exemplo de Sistema Flexível de Manufatura

Nesse sistema, existem duas alternativas de processamento das peças: em lotes ou misto.

Quando se processa as peças em lote, primeiro se processa as peças de um tipo, faz-se um ajuste da máquina para a troca do programa de usinagem e então processa-se a segunda categoria de peças. A figura 2.4-2 apresenta o modelo F-MFG do processamento em lotes.

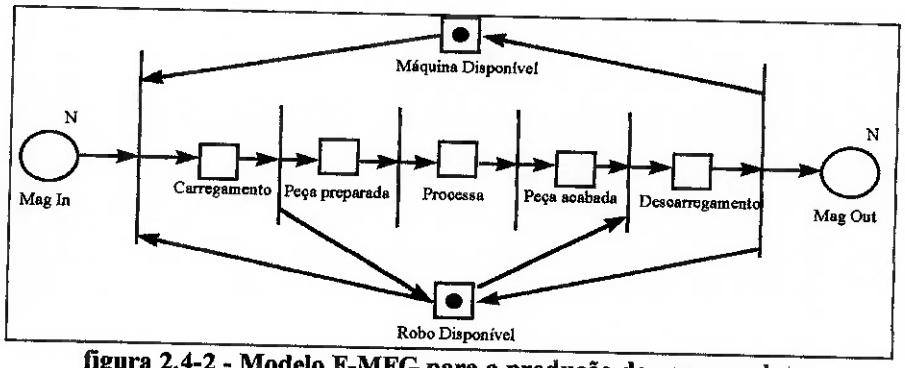


figura 2.4-2 - Modelo F-MFG para a produção de peças em lotes

Tomando a segunda alternativa, ou o processamento misto das peças, torna-se necessário indicar os dois processos no modelo, pois é necessário saber qual peça se está processando para o acionamento do programa de usinagem correto. A figura 2.4-3 apresenta o modelo F-MFG do sistema para o processamento misto dos dois tipos de peça.

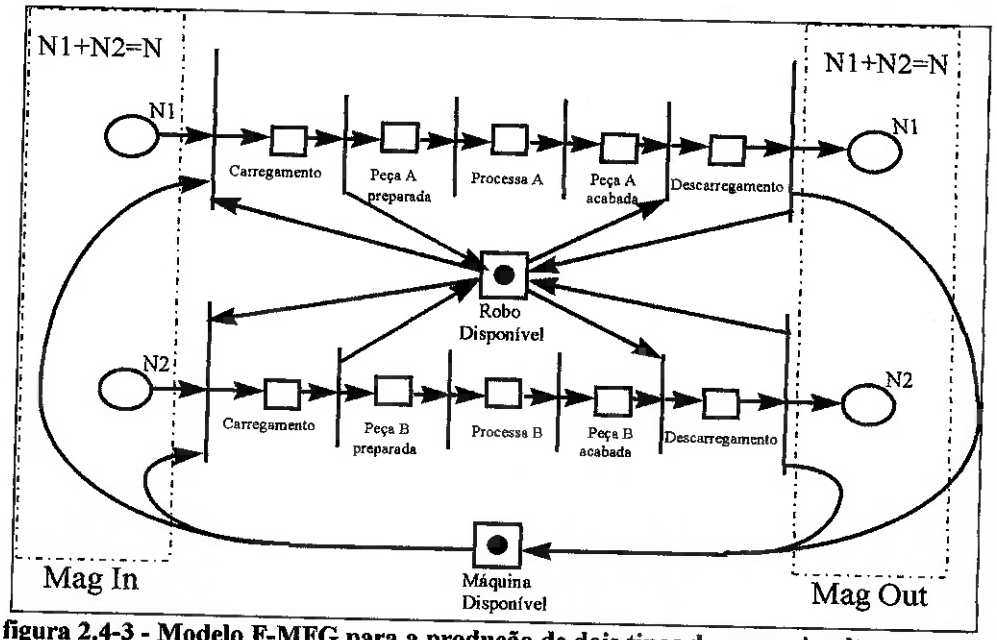


figura 2.4-3 - Modelo F-MFG para a produção de dois tipos de peças simultaneamente

Nota-se que na figura 2.4-3 existe uma duplicação de diversos elementos do modelo em virtude da necessidade de se representar os dois processos, tornando-o mais complexo. Pode-se notar que o grau de complexidade do modelo cresce muito com o grau de flexibilidade do sistema.

No sentido de se adicionar uma maior flexibilidade à modelagem de SEDs foi proposta uma extensão ao MFG onde se adiciona regras de disparo adicionais às transições e atributos às marcas, individualizando-as (SANTOS FILHO, 1993). Esse modelo foi chamado de '*Enhanced Mark Flow Graph*' ou E-MFG.

2.5 - Modelagem utilizando o E-MFG

O E-MFG possui basicamente os mesmos tipos de elementos estruturais básicos do MFG. No entanto, foram adicionadas propriedades a esses elementos no sentido de se elevar a capacidade de representação do grafo. Assim, tem-se os seguintes elementos estruturais básicos:

- as transições: indicam a ocorrência de eventos e podem possuir inscrições que representam regras adicionais restritivas para a evolução dinâmica do sistema, estas regras podem observar o estado global do grafo para autorizar ou não um disparo de transição;
- os boxes: indicam as pré e pós condições para a ocorrência de um evento;
- as marcas: indicam a manutenção de uma condição e podem ser individualizadas por atributos;
- as portas: habilitam ou inibem a ocorrência de um evento, podendo ter inscrições representando condições relacionadas com os atributos das marcas a fim de habilitar ou inibir a ocorrência de um evento;
- os arcos orientados: estabelecem a relação causal entre os eventos e as condições e podem conter inscrições que controlam a manutenção dos atributos das marcas realizando uma filtragem seletiva desses atributos;
- arcos de sinal de saída: podem transmitir informações ao meio externo relativas ao estado do atributo de uma marca ou ao estado de uma condição.

Como já foi mencionado, as marcas no E-MFG são acompanhadas de um conjunto de atributos que as caracterizam individualmente. A esses atributos podem estar associadas informações referentes ao produto, processo e ao controle. Foi

proposto que esses atributos fossem representados por um vetor de valores inteiros ou lógicos (e portanto referenciados através de índices), conforme a representação da figura 2.5-1a (SANTOS FILHO, 1993). Nesta representação pode-se considerar que um atributo com valor nulo (zero) indique a ausência desse atributo. No entanto, devido a algumas características desejáveis aos sistemas de controle e à especificação do controle através de uma representação textual, conforme a discussão na seção 3.2, propõe-se neste trabalho que os atributos das marcas sejam representados por uma lista de atributos de tipo pré declarado (e.g. valores inteiros, 'strings' e valores booleanos), sendo referenciados através de nomes mnemônicos, conforme a figura 2.5-1b. Deve-se ressaltar que essa alteração de representação não afeta a dinâmica do grafo, alterando apenas a especificação do controle através de uma descrição textual e a implementação de um controlador baseado nesta descrição.

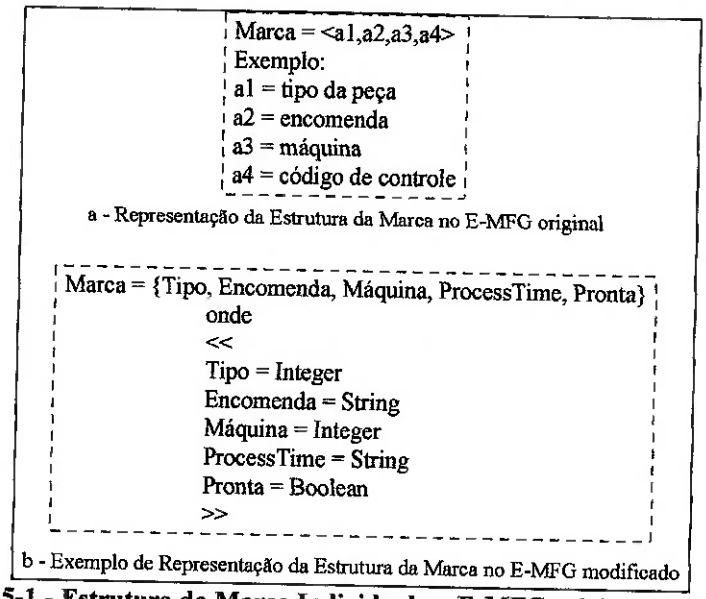


figura 2.5-1 - Estrutura da Marca Individual no E-MFG original e modificado

Ainda em relação à marcação no E-MFG, é necessário definir o conceito de marca composta, que corresponde ao agrupamento de diversas marcas individuais em uma única marca individual, que mantêm a estrutura das marcas que a compõe. Isso pode ser obtido associando à marca individual composta um atributo que representa um código de controle para a composição conforme o exemplo da figura 2.5-2. Esse tipo de marca pode ser utilizada para representar o agrupamento e desagrupamento de peças e componentes nos processos industriais.

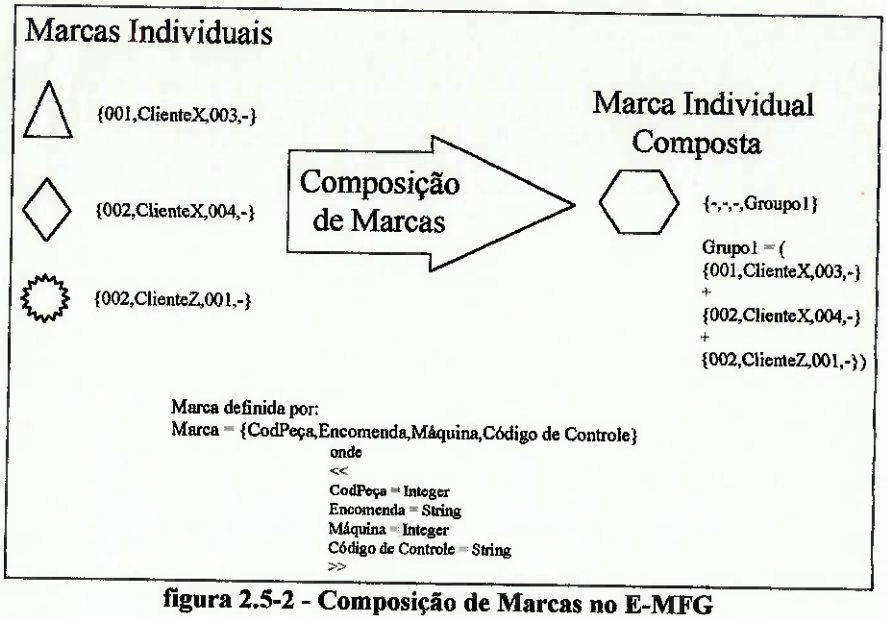


figura 2.5-2 - Composição de Marcas no E-MFG

Outro aspecto relevante do E-MFG é a extensão da funcionalidade dos macro-elementos do F-MFG. No E-MFG os macro elementos possuem as seguintes características:

- Box Capacidade: esses elementos modelam '*buffers*' para o armazenamento temporário de itens. No E-MFG, esses elementos também devem possuir um método de acesso ou saída das marcas, pois como estas apresentam características individuais, a ordem de saída passa a ter importância. São definidos dois métodos de saída para as marcas LIFO (*'Last In First Out'*), onde as marcas são armazenadas como em uma pilha e FIFO (*'First In First Out'*), onde as marcas são dispostas como em uma fila (ver figura 2.5-3).

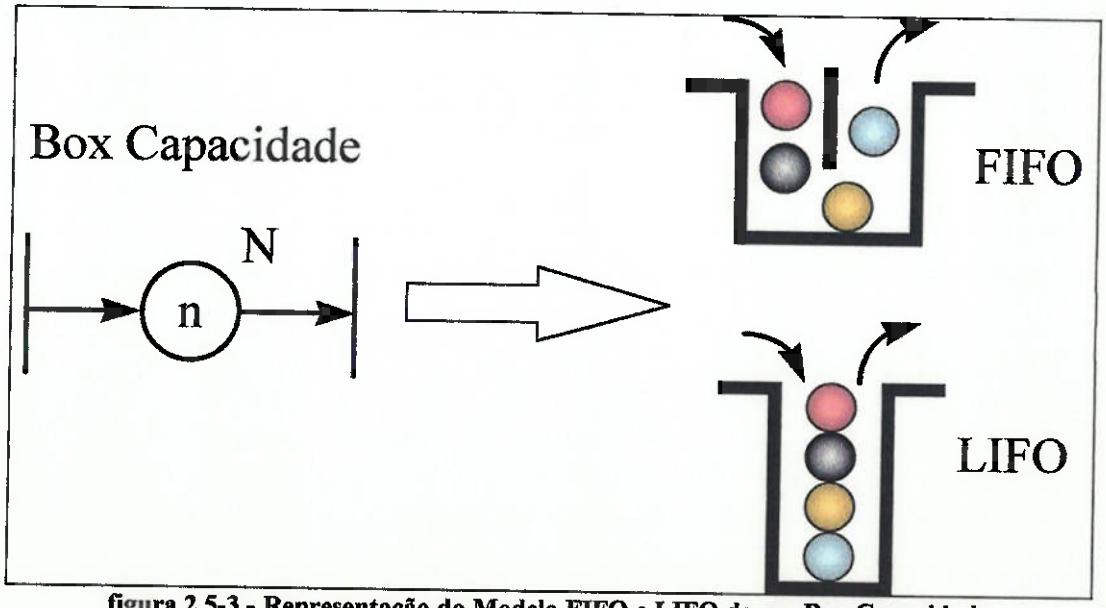


figura 2.5-3 - Representação do Modelo FIFO e LIFO de um Box Capacidade

- Box Agrupador: esses elementos modelam atividades de agrupamento de peças (e.g. *palletização*). Ao agrupar as marcas em uma única marca de saída esse elemento realiza a composição de marcas (figura 2.5-4).

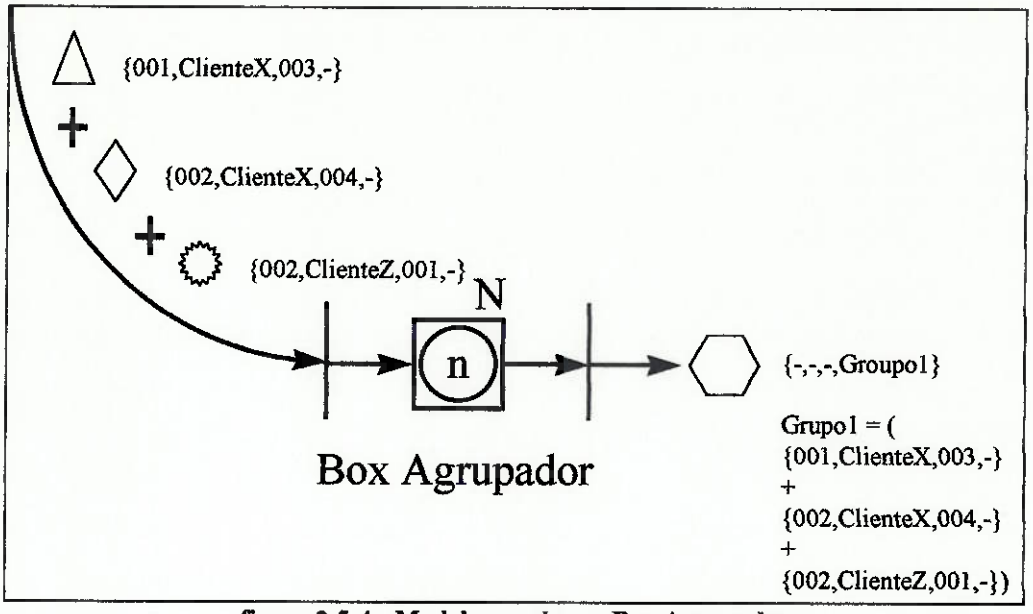


figura 2.5-4 - Modelagem de um Box Agrupador

- Box Dispersor: esses elementos modelam atividades de desagrupamento de peças (e.g. *despalletização*). Ao desagrupar uma marca composta em diversas marcas de saída esse elemento pode realizar dois métodos FO (*'First Out'*) ou LO (*'Last Out'*), pois aqui é importante a ordem de desmontagem ou desagrupamento da marca individual composta (figura 2.5-5).

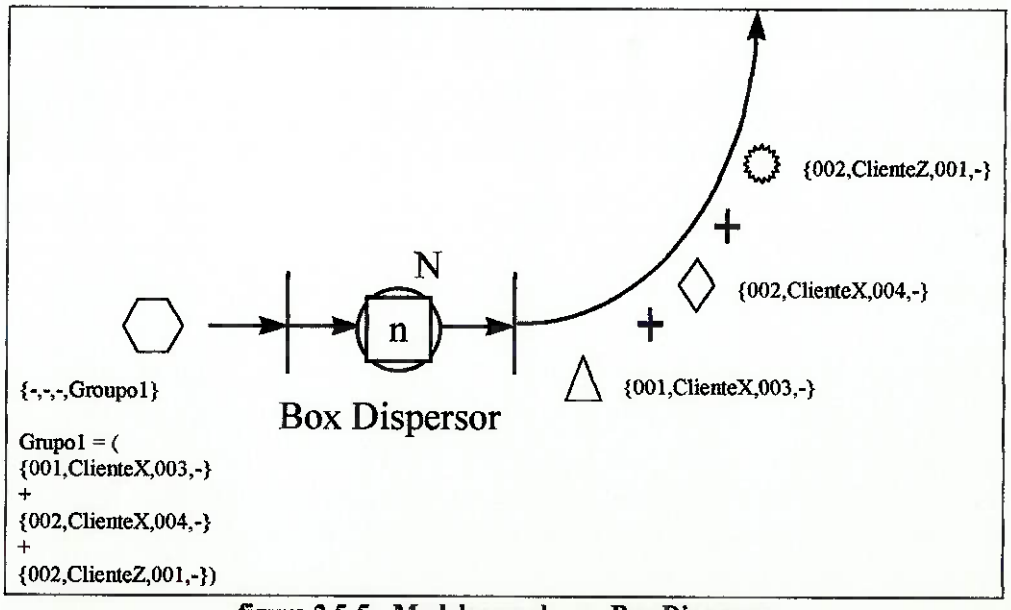


figura 2.5-5 - Modelagem de um Box Dispersor

Um outro elemento importante no E-MFG é o Box Controlador ou Transformador (representado por um quadrado com um losango em seu interior), este box possui um conjunto de regras de controle descritas através de estruturas SE ... ENTÃO... SENÃO (IF-THEN-ELSE). Estas regras são regras de produção que observam o estado atual dos elementos do grafo (e.g. valores dos atributos das marcas) e caso estejam satisfeitas executam alguma atribuição de valores aos atributos de uma marca, esta dinâmica é ilustrada na figura 2.5-6.

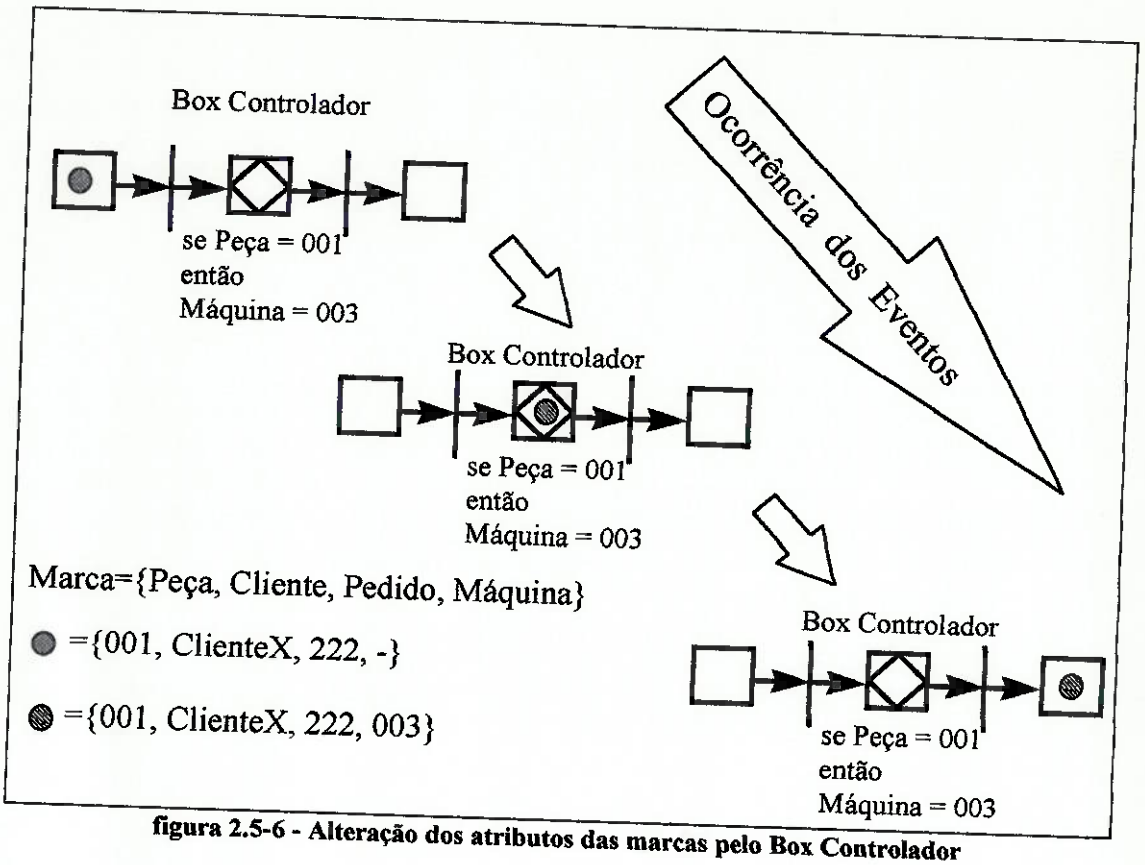


figura 2.5-6 - Alteração dos atributos das marcas pelo Box Controlador

O mecanismo de alteração dos atributos das marcas através dos boxes controladores pode ser chamado de alteração condicional. O segundo mecanismo de manutenção dos atributos das marcas é a filtragem seletiva dos atributos relevantes ao próximo estado do grafo. Este processo de filtragem seletiva é realizado pelos arcos orientados, que podem permitir ou não a passagem de determinados atributos para o próximo estado. O mecanismo de filtragem seletiva e a representação dos filtros nos arcos orientados estão ilustrados na figura 2.5-7.

O mecanismo de filtragem seletiva nos arcos orientados é especialmente interessante para se garantir a integridade da informação carregada pela marca em seus atributos. Foi proposto que quando uma transição disparasse os atributos das marcas pertencentes às pré-condições passassem pelo processo de filtragem seletiva e fossem então compostos através de uma operação lógica (AND, OR, XOR) ou algébrica(+,-) de acordo com a sua natureza (SANTOS FILHO,1993). No entanto, ao se adotar essa estratégia pode-se observar que ao se disparar um dado evento onde não existe apenas uma única marca com um determinado atributo após a filtragem seletiva dos atributos das marcas que pertencem a pré-condição, o disparo da transição alterará o valor do atributo. Isto exige um cuidado muito maior por parte do projetista do sistema para se garantir que a informação carregada por esse atributo seja sempre coerente. Assim, neste trabalho propõe-se que o projetista adote a seguinte abordagem: especificar claramente nos filtros dos arcos orientados quais atributos são relevantes para o próximo estado (não permitindo dessa forma a geração das inconsistências mencionadas) e apenas realizar uma alteração do valor de um atributo em um box controlador (onde essa alteração é explícita).

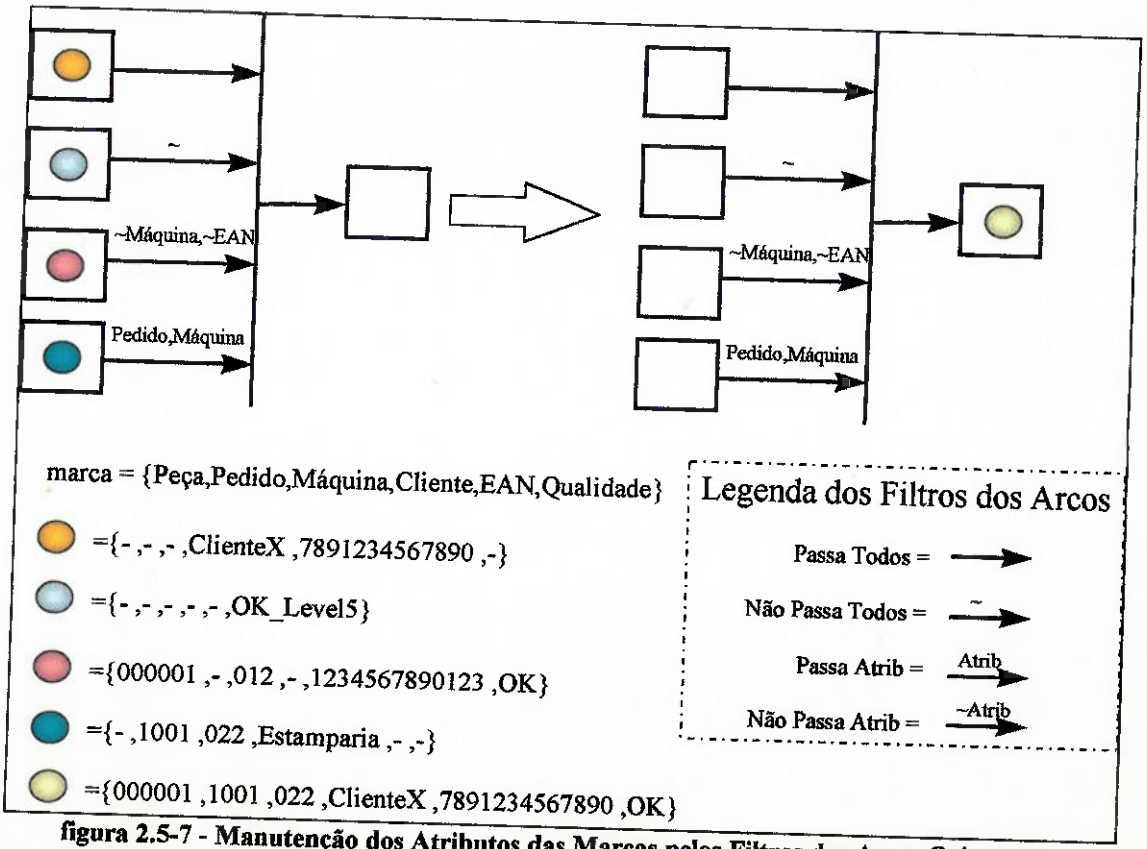


figura 2.5-7 - Manutenção dos Atributos das Marcas pelos Filtros dos Arcos Orientados

Tendo em vista as alterações nos elementos estruturais do grafo é natural que a regras de disparo de transições do E-MFG também sejam uma extensão das regras do MFG, existindo apenas algumas condições adicionais que devem estar satisfeitas para a efetivação do disparo de uma transição.

As regras de disparo de transições são resolvidas de acordo com uma hierarquia, correspondendo a três níveis de decisão:

- o primeiro nível corresponde às restrições adicionais de disparo, ou às funções lógicas agregadas às transições, necessárias à representação de estratégias de controle específicas. Essas regras são representadas pelas inscrições nas transições, conforme discutido anteriormente. Se não existirem inscrições em uma transição, então não existem regras adicionais que limitam o seu disparo. Uma transição que não possui regras adicionais ou aquela que apresenta essas regras satisfeitas é denominada transição em prontidão;

- o segundo nível corresponde às regras de habilitação de disparo que constituem as mesmas regras válidas para o MFG. Assim, uma transição é considerada habilitada para o disparo se for uma transição em prontidão e satisfizer as seguintes condições: todas as pré-condições estão satisfeitas (marcação nos boxes de entrada), todas as pós condições estão satisfeitas (ausência de marcação nos boxes de saída), não existe porta habilitadora interna ou externa conectada a esta transição que esteja em estado de desabilitação, não existe porta inibidora interna ou externa conectada a esta transição que esteja em estado de inibição.

- o terceiro nível corresponde às regras de disparo e envolvem a resolução de conflitos e a verificação das regras de filtragem seletiva dos atributos das marcas pelos filtros dos arcos orientados.

Uma transição habilitada é denominada disparável quando possui as condições de conflito resolvidas. Uma transição disparável dispara imediatamente, há menos que existam atrasos de tempo devidos a elementos temporizados.

No E-MFG, assim como no MFG pode-se optar por não arbitrar um conflito (disparando aleatoriamente uma das transições), ou quando se deseja arbitrar efetivamente o conflito pode-se proceder das seguintes formas:

- impor uma dinâmica de disparo ao sistema através de um árbitro externo ao grafo conectado ao grafo por portas habilitadoras ou inibidoras externas;

• impor uma dinâmica através de um sub-grafo que descreve a estratégia de resolução do conflito conectado às transições em conflito por meio de portas inibidoras ou habilitadoras;

• definir a transição disparável através de portas inibidoras ou habilitadoras com ou sem inscrições provenientes do próprio box-conflito.

2.6 - A aplicação do PFS/E-MFG à modelagem de Sistemas Flexíveis de Manufatura

A metodologia PFS/E-MFG é uma extensão da metodologia PFS/MFG apresentada na seção 2.3. A principal diferença entre as duas metodologias reside no fato de no PFS/E-MFG o sistema final está descrito através de um grafo E-MFG. No sentido de se manter a característica hierárquica da modelagem foi proposto que a estrutura da marcação fosse sendo detalhada na medida em que se avança na modelagem dos diversos níveis hierárquicos (SANTOS FILHO, 1993). A figura 2.6-1 apresenta um exemplo de modelagem através do PFS/E-MFG do sistema descrito na seção 2.4 e figura 2.4-1.

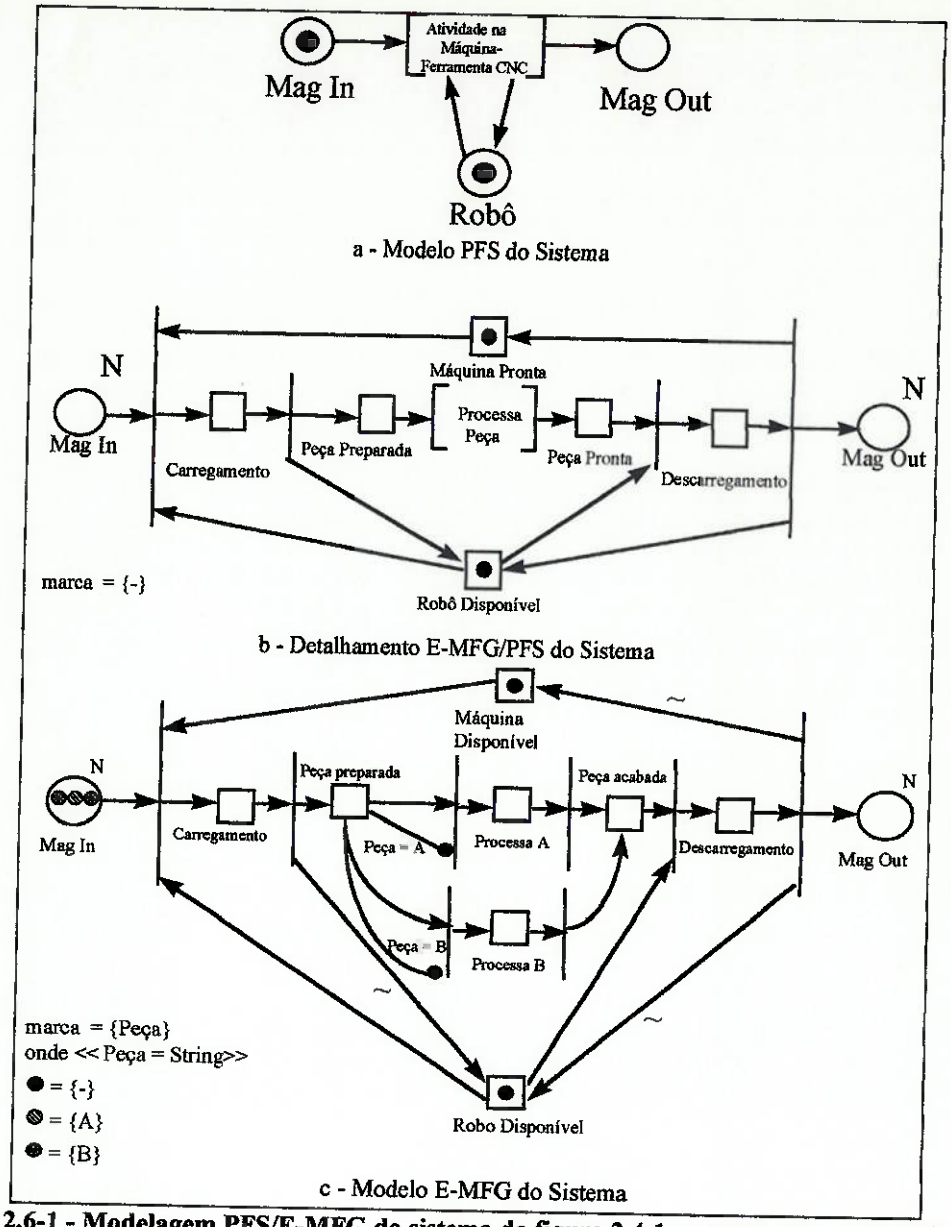


figura 2.6-1 - Modelagem PFS/E-MFG do sistema da figura 2.4-1 para processamento misto de dois tipos de peças

3 - Arquiteturas e Especificação de Controle em um Sistema Flexível de Manufatura

3.1 - Arquiteturas de Controle

Existem duas classificações básicas para a arquitetura de um sistema de controle: o controle centralizado e o controle distribuído. Cada uma dessas implementações traz vantagens e desvantagens de acordo com a aplicação em questão e portanto nenhum desses paradigmas será adotado como requisito de arquitetura do controlador. No entanto, uma breve discussão a respeito de quais são as principais características de cada abordagem se faz necessária a fim de mostrar que a aplicabilidade da metodologia PFS/E-MFG para a modelagem e especificação do controle não está ligada a nenhum desses paradigmas, sendo adequada a sua utilização nas duas modalidades de implementação.

3.1.1 - Controle Centralizado

O controle centralizado é caracterizado pela concentração de toda a tomada de decisão e supervisão do estado geral do processo produtivo por um único controlador (CLP, ou computador industrial). Essa implementação possui a vantagem de se ter disponível nessa máquina toda a informação relevante do processo produtivo. No entanto, a medida que o porte do sistema aumenta o gerenciamento de todas as informações começa a se tornar extremamente complexo, pois a todo momento deve-se olhar para o estado global da planta em seu nível de maior detalhe. Portanto o modelo utilizado deve estar totalmente detalhado. Assim uma metodologia 'top-down', como o PFS/E-MFG, é de grande valor para a obtenção de um modelo correto para o sistema. Através dela pode-se dividir o problema em módulos e resolvê-los separadamente, aumentando assim a precisão do modelo de cada subsistema.

Um cuidado que se deve tomar em relação ao controle centralizado é o fluxo de informações na via de dados, pois dependo da arquitetura desta, ela pode ficar congestionada, degradando a performance do sistema e até inviabilizando o controle.

3.1.2 - Controle Distribuído

Existem duas abordagens para o controle distribuído: o controle simplesmente distribuído e o controle hierárquico distribuído.

No controle simplesmente distribuído, a divisão da tomada de decisão é feita entre os diversos módulos, que se comunicam entre si sem regras de comunicação pré-determinadas (todos os módulos podem se comunicar com todos os módulos).

O controle hierárquico distribuído é caracterizado pela divisão da tomada de decisão entre os diversos sub-sistemas e supervisão do estado geral do processo produtivo entre diversos níveis hierárquicos de controladores (CLP's e/ou computadores industriais). Essa implementação possui a vantagem de se ter em cada máquina apenas a informação relevante a parte do processo produtivo que ela visa controlar. Assim, pode-se garantir um melhor gerenciamento de informações independentemente do porte do sistema. Portanto, passa-se a utilizar modelos que possuem apenas o nível de detalhe relevante para o nível hierárquico que se está controlando. Uma metodologia 'top-down' como o PFS/E-MFG permite a obtenção e identificação dos diversos modelos necessários. Através dela pode-se dividir o problema em módulos e resolvê-los separadamente, identificando com precisão do modelo de cada subsistema.

Quando se utiliza o controle hierárquico distribuído deve-se evitar o fluxo vertical de informações desnecessárias na via de dados, utilizando quando possível vias independentes de dados para o nível de controle direto (ou nível de ações) e os níveis de supervisão (ou nível de tarefas).

3.2 - Aplicabilidade do E-MFG ao controle de Sistemas Flexíveis de Manufatura

A viabilidade técnica e a performance de qualquer sistema de controle depende fundamentalmente da disponibilidade de um modelo que traduza de modo preciso as características dinâmicas do sistema controlado. No capítulo 2, foi mostrada a adequação do E-MFG a modelagem de sistemas flexíveis e integrados de manufatura. Sob o ponto de vista do controle desses sistemas, o E-MFG apresenta a vantagem de possibilitar a representação dos sinais de controle e vias de dados através de arcos de sinal de saída e *'gates'* de entrada, que são elementos inerentes ao modelo. Do mesmo modo, observando sob o prisma do controle distribuído, a metodologia PFS/E-MFG, permite uma fácil identificação dos níveis hierárquicos de

controle e suas interdependências, facilitando a especificação do algoritmo e informações de controle necessários em cada nível hierárquico. Isso não impede, no entanto, a obtenção de um modelo global para a realização do controle centralizado (especialmente em instalações de pequeno porte onde o fluxo de informações na via de dados não é um parâmetro crítico).

Também é de fundamental importância rever neste tópico as mudanças propostas na estrutura do E-MFG neste trabalho e o E-MFG original (SANTOS FILHO, 1993).

Dentro do conceito do E-MFG original, cada marca possui um vetor de atributos referenciados por índices, sendo que a existência de um dado atributo depende do seu valor ser diferente de nulo. O conteúdo de cada atributo seria representado sempre por um número inteiro ou valor lógico.

Neste trabalho propõe-se que:

- a fim de se aumentar a clareza de representação dos atributos do E-MFG, os atributos das marcas em um dado nível hierárquico sejam referenciados por um nome ou *'label'*;
- o conjunto de todos atributos possíveis para cada nível hierárquico seja pré-declarado, e todas as marcas desse nível hierárquico herdem essa estrutura. Se um atributo na marca apresentar valor nulo este pode ser considerado inexistente.
- cada atributo seja associado a um tipo de dado, que no escopo deste trabalho podem ser valores inteiros ou strings.

Outra mudança de concepção em relação ao E-MFG originalmente proposto está na definição de um critério para a composição de atributos de marcas. Originalmente, previa-se a utilização de uma operação lógica (AND, OR, XOR, etc.) ou algébrica (+,-, etc.) para compor os atributos da marca que irá para o box que representa a pós-condição de um evento a partir dos atributos das marcas que contidas pelos boxes que representam a pré-condição de um evento.

No entanto, essa abordagem só faz sentido quando se tratam de atributos binários ou codificações binárias de atributos. Isto exige um cuidado muito maior por parte do projetista do sistema para se garantir que a informação carregada por esse atributo seja sempre coerente.

Neste trabalho propõe-se que o projetista deva especificar claramente nos filtros dos arcos orientados quais são os atributos relevantes ao próximo estado, não permitindo a geração de inconsistências no estado seguinte.

Tendo em vista a necessidade de integração entre o programa controlador e os demais sub-sistemas que compõem a planta, é interessante acessar dados referentes ao processo externos ao grafo. Esses dados podem ser então utilizados tanto nas atribuições, quanto nas condições booleanas inscritas nos elementos do grafo. Neste sentido, propõe-se a inclusão de elementos de entrada de dados no grafo (valores inteiros ou *'strings'*). Esse elemento passa a ser descrito como um arco de sinal de entrada de dados (representado na figura 3.2-1a).

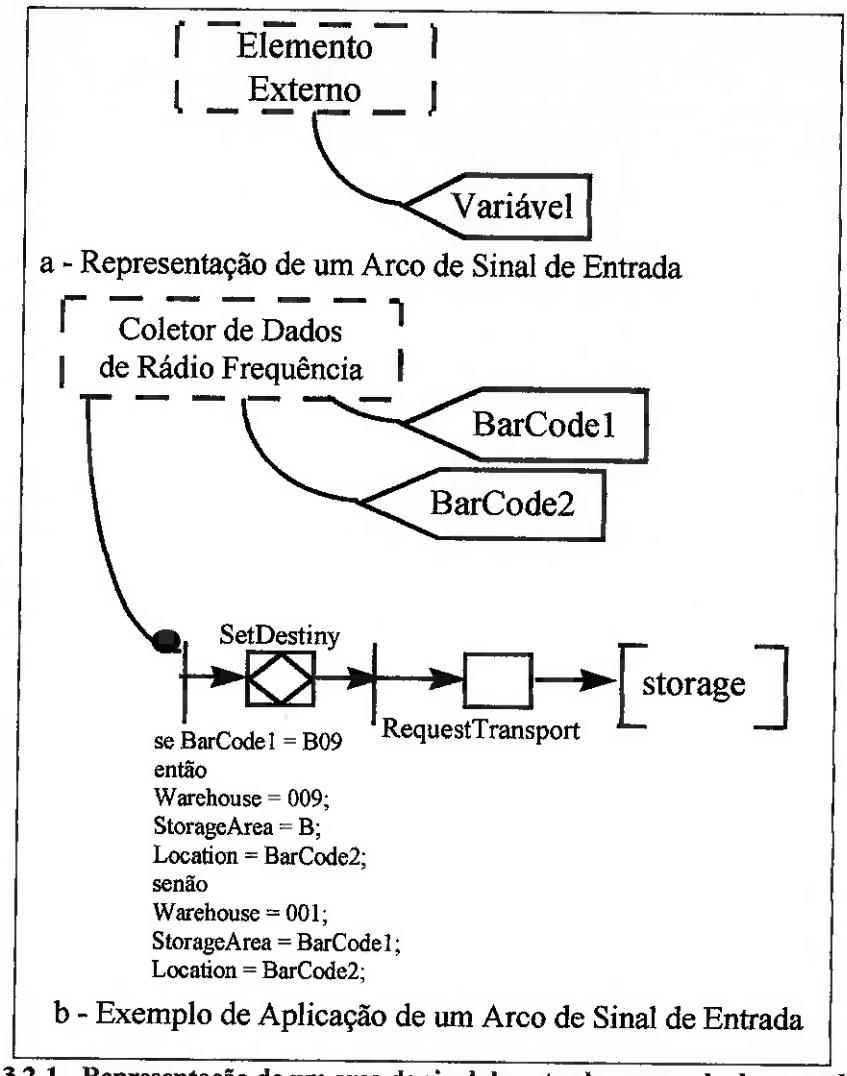


figura 3.2-1 - Representação de um arco de sinal de entrada e exemplo de sua aplicação

As vantagens trazidas pelas propostas deste trabalho são:

• maior capacidade de representação da marca e facilidade de interpretação do modelo, permitindo dessa forma a representação de estados da marca em linguagem natural sem a necessidade de se recorrer a tabelas de códigos.

• maior alinhamento com as tendências atuais de rastreabilidade e identificação de materiais, pois a representação de *'strings'* como atributos permite a utilização de códigos alfanuméricos, tais como códigos de barras, como parte do modelo.

• maior alinhamento com os recursos de tecnologia de informação atuais, especialmente com a tecnologia de bancos de dados, que são aplicados em praticamente todas as aplicações industriais, pois permite a descrição dos atributos das marcas através de tabelas de bancos de dados relacionais, uma vez que se identifica *a priori* todos os tipos de dados e representatividade desses dados de maneira unívoca.

• maior facilidade para se garantir no modelo a coerência de informações que são passadas de um estado para o outro, pois todos os atributos que devem ser mantidos devem ser claramente especificados nas inscrições dos arcos orientados.

• possibilidade de se utilizar *'labels'* mnemônicas para os atributos, conferindo uma maior facilidade na utilização destes *'labels'* para se especificar as condições e atribuições inscritas nos elementos do grafo;

• possibilidade de uma maior integração com fontes de dados externos ao grafo, por meio dos arcos de sinal de entrada (vide exemplo na figura 3.2-1 b).

3.3 - Especificação de sistemas de controle através do E-MFG

Para se especificar o algoritmo de controle através do E-MFG, é necessário que o modelo contenha apenas elementos E-MFG para o módulo do nível hierárquico que deseja-se controlar (caso se esteja utilizando uma metodologia "top-down" como o PFS/E-MFG, o sub-grafo que representa o módulo que está sendo controlado deve estar totalmente detalhado).

Por outro lado, o grafo que representa o sistema de controle deve gerar e ler sinais de controle apenas e tão somente através de portas (*'gates'*) de entrada e saída, impossibilitando dessa forma, o fluxo vertical de marcas. Este procedimento

preserva a representatividade única dos atributos da marca em cada nível de controle e evita o problema de composição de atributos de marcas que apresentam diferentes níveis de detalhamento dos atributos.

Dessa maneira, uma linguagem de programação E-MFG (e portanto uma linguagem formal de especificação de controle) deve ser orientada de modo a manter a coerência dos grafos em cada nível hierárquico.

Assim, a descrição do algoritmo de controle deve possuir um formalismo rígido para a representação dos elementos e conexões do grafo. Por outro lado, deve ser flexível para se adaptar aos novos elementos tecnológicos que são introduzidos na indústria. Portanto, apesar da rigidez de sintaxe requerida para uma descrição formal do grafo, a descrição dos elementos de I/O (gates) deve ser flexível quanto aos parâmetros requeridos para se identificar a origem ou o destino do sinal de controle. Assim pode-se abranger o maior número possível de protocolos e sistemas de informação e permitir que a sintaxe da linguagem acompanhe a rápida evolução tecnológica existente hoje, especialmente no que diz respeito a tecnologia de informação.

Adotando essa abordagem na especificação de uma linguagem de programação E-MFG, tem-se a certeza que um modelo de controle já utilizado em um sistema produtivo sofrerá pouco ou nenhum impacto devido a uma evolução tecnológica na transmissão e coleta desses dados. Desta maneira, se exigirá uma quantidade mínima de esforço (homens/hora) para revalidar o programa de controle dentro do novo ambiente.

4 - Controlador E-MFG

4.1 - Especificações

O propósito de um controlador para um Sistema Integrado e Flexível de Produção no nível de controle da seqüencialização e execução da produção é garantir a ocorrência dos eventos de produção na ordem adequada. Ou seja, obedecer as restrições dinâmicas impostas pelo projetista do programa de controle, ou regras de produção, bem como realizar a interface com os sistemas do chão de fábrica e os sistemas estratégicos da empresa.

Portanto, tendo identificado as necessidades do sistema de controle tem-se o seguintes objetivos para o projeto:

- Especificar uma linguagem de programação do controlador;
- Especificar os requisitos de projeto para o controlador;
- Desenvolver o programa controlador.

Tomando por base esses objetivos, bem como a discussão quanto a modelagem e o controle de Sistemas Flexíveis e Integrados de Produção nos capítulos 2 e 3 deste trabalho, o controlador e a sua linguagem de programação devem cumprir minimamente, os seguintes requisitos:

- receber a estratégia de controle descrita sob a forma de um grafo E-MFG, descrito de forma textual de acordo com o *EMFG- Script* especificado no capítulo 5;
- permitir a interface com os sistemas do chão de fábrica e os sistemas estratégicos da empresa através de um método padrão;
- rodar em um ambiente padrão de indústria;
- minimizar o impacto de avanços e mudanças de tecnologia no código ou programa de controle;
- desenvolvimento orientado a objetos do código do controlador e seus módulos a fim de se facilitar a sua manutenção e possibilitar a adição de novas funções ao controlador sem provocar grandes impactos nos demais módulos do programa;
- baixo custo do conjunto plataforma e sistema operacional utilizados;

4.2 - Análise de Alternativas

Com base nas especificações do item 4.1 tem-se dois grupos de alternativas para o desenvolvimento do controlador. O primeiro grupo constitui nas opções de Hardware e Software básicos (plataforma e sistema operacional) onde o sistema de controle será desenvolvido e o segundo grupo trata da estratégia de desenvolvimento ou seja a arquitetura do programa e a estrutura de dados.

4.2.1 - Plataformas de Hardware/Sistema Operacional

Tendo em vista os requisitos do sistema restringe-se o conjunto de possibilidades de hardware e sistema operacional a duas arquiteturas básicas. Sendo essas:

- ***Arquitetura RISC/UNIX(e.g. RS 6000/IBM AIX):*** Nesta arquitetura existem um grande número de processadores e versões do sistema operacional. É um tipo de plataforma de custo relativamente alto consolidada na indústria devido a grande velocidade dos processadores RISC e a confiabilidade dos diversos sistemas operacionais utilizados. No entanto, não existe uma posição clara de qual conjunto processador/Sistema Operacional se destaca como líder de mercado, sendo esta escolha muito dependente das opções de software existentes para cada tipo de indústria (e.g. O CATIA é uma solução de CAD/CAM cuja a plataforma de desenvolvimento é baseada em RS6000/AIX). Outra desvantagem é a difícil gerenciamento do ambiente Unix e o alto custo de treinamento do pessoal envolvido com o sistema.
- ***Arquitetura Intel/Microsoft Windows 95/NT***: Esta arquitetura tem se expandido com grande velocidade no mercado substituindo máquinas RISC em diversas aplicações principalmente devido ao grande aumento da velocidade de processamento dos processadores Intel e a familiaridade de interface fornecida pelo Windows 95/NT, sendo o Windows 95 o padrão mundial em computadores pessoais. Estas máquinas são de custo inferior a máquinas RISC, podendo assim como estas últimas, ser multiprocessadas quando se utiliza o Windows NT. A menor necessidade de treinamento e a facilidade de gerenciamento destas estações são as vantagens dessa plataforma. No entanto, esta é uma tecnologia que está se consolidando apenas recentemente na indústria.

Na tabela 4.2.1-1, tem-se uma análise comparativa entre essas duas plataformas:

Tabela 4.2.1-1: Comparação entre as plataformas de Hardware e Software

Plataforma	Vantagens	Desvantagens
RISC/UNIX (e.g. RS6000/IBM AIX)	• plataforma já consolidada no mercado • diversas soluções de software para cada tipo de indústria • velocidade de processamento das máquinas RISC • confiabilidade dos sistemas operacionais	• alto custo do hardware • alto custo do software básico (Sistema Operacional) • não existe um conjunto Processador/Sistema Operacional que se destaque como líder de mercado (escolha muito dependente do segmento de indústria) • alto custo de treinamento de pessoal
INTEL/WINDOWS 95/NT	• plataforma em expansão no setor industrial (tendência de mercado) • baixo custo de hardware • baixo custo de software básico (Sistema Operacional) • plataforma padrão mundial em microcomputadores pessoais (pouca necessidade de treinamento)	• Velocidade das máquinas INTEL ainda é inferior à máquinas RISC • tecnologia recente em ambiente industrial • migração das soluções industriais para essa plataforma só ocorreu recentemente

Observando as vantagens e desvantagens de cada plataforma e tendo em vista a rápida expansão da arquitetura Intel/Windows NT e a conseqüente migração dos softwares específicos de cada indústria para essa plataforma única (e.g. as soluções de CAD/CAM da IBM/Dassault Systémes - o CATIA e a solução de ERP da SAP - o SAP R/3, entre outras) optou-se por desenvolver o controlador para esta plataforma de modo a se alinhar com a tendência atual de mercado.

4.2.2 - Arquitetura de Software e Estruturas de Dados

Um dos requisitos do projeto é o desenvolvimento do código do controlador de forma modular e orientada a objetos. Assim, analisando a estrutura mínima necessária para o controlador identifica-se os seguintes módulos: a Representação Interna do grafo (estrutura de dados), um Interpretador do grafo (módulo que constrói o grafo a partir da descrição textual), um Gerenciador de Marcas (módulo responsável pelo disparo de transições), um árbitro interno responsável pela análise das condições booleanas, um controlador de I/O responsável pela conectividade do grafo com o meio externo e um módulo de interface com o usuário.

A fim de se definir como deveria ser realizada a representação interna do controlador levantou-se algumas alternativas de estruturas de dados para essas representações:

1. ***Utilização de tabelas para representar cada elementos do grafo e suas respectivas conexões:*** dessa maneira cada tipo de elemento do grafo seria representado através de uma tabela onde cada coluna corresponde-se a uma determinada propriedade ou conexão deste elemento com os demais. Por exemplo, na tabela de boxes constariam o índice na tabela marcas da marca que está presente no box, a sua constante de tempo (no caso de boxes temporizados), os índices nas tabelas de arcos dos arcos que partem ou chegam nesse box, etc. Nesta representação as tabelas constituiriam os macro-objetos que deveriam ser manipulados para se executar a dinâmica. Vantagens: fácil transcrição da linguagem textual para a tabela, possibilidade de se utilizar um banco de dados (integrabilidade). Desvantagens: difícil controle da consistência do disparo de transições (sempre se tem uma visão global do grafo), necessidade de se interpretar o texto da tabela durante todo passo de disparo de transições devido a existência de elementos com número variável de parâmetros (e.g. número de arcos que chegam em um box ou transição).
2. ***Utilização de vetores de classes de objetos em diferentes níveis de abstração:*** nesta alternativa seriam definidas classes de objetos hierarquizadas para representar os elementos do grafo e seus atributos (que constituiriam objetos auto-contidos). Dessa forma os macro-objetos que deveriam ser manipulados para se executar a dinâmica do sistema seriam os próprios elemento do grafo. Vantagens:

fácil controle do disparo de transição (visão local do grafo), elementos do grafo podem ter dinâmica própria (eliminando a checagem do tipo de elemento no nível de gerenciamento de disparos de transições), número flexível de parâmetros. Desvantagens: difícil construção do grafo a partir da descrição textual.

3. ***Utilização de Matrizes de incidência e listas de atributos:*** nesta representação as conexões do grafo estariam representadas através de matrizes de incidência e as propriedades dos elementos do grafo estariam relacionadas em listas de propriedades, assim os macro-objetos a serem manipulados na execução da dinâmica seriam as próprias matrizes de incidência e as listas de atributos para o cálculo do vetor de disparo das transições. Vantagens: permite posteriormente o acoplamento de algorítimos de análise matemática sobre o grafo. Desvantagens: difícil construção do grafo a partir da descrição textual, difícil gerenciamento dos atributos dos elementos, difícil modelagem matemática das características do E-MFG.

Na tabela 4.2.2-1 é feita uma análise comparativa entre as 3 arquiteturas de dados apresentadas, para se construir a tabela considerou-se 3 tipos de características: características que fazem parte da especificação do software (peso 5), características ligadas a performance, testes e codificação (peso 4) e características desejáveis (demais pesos).

Tabela 4.2.2-1: Comparação entre as alternativas de estruturas de dados				
	Peso	**Tabelas de Elementos**	**Vetores de Objetos**	**Matrizes de Incidência**
Facilidade de alterações na funcionalidade dos elementos do grafo	**5**	7	9	6
Facilidade de construção do grafo a partir da descrição textual	**4**	9	5	5
Performance do ciclo de disparo de transições	**5**	5	7	8
Flexibilidade quanto ao número de parâmetros do grafo (encapsulamento)	**5**	7	10	7
Facilidade de representação dos dados através de banco de dados	**2**	10	5	5
Facilidade no gerenciamento de disparo de transições	**4**	5	9	5
Facilidade de se adicionar algoritmos de análise matemática sobre o modelo	**3**	5	5	10
Facilidade de gerar casos de teste	**4**	8	8	7
Classificação Geral da Solução		6.81	7.59	6.65
Classificação Normalizada		0.90	1.00	0.87

Analisando-se, portanto, a tabela adotou-se a segunda solução para a estrutura de dados interna do controlador. O detalhamento desta estrutura de dados e módulos funcionais encontra-se no capítulo 6.

4.3 - Modelagem Estrutural/Funcional do Controlador

Analisando-se os requisitos de funcionalidade do controlador pode-se identificar um conjunto de tarefas que o controlador deve executar. Sob um ponto de vista macroscópico essas tarefas podem ser assim enumeradas: a leitura e interpretação da descrição textual do grafo, a construção da representação interna de dados, o gerenciamento de disparo de transições, o gerenciamento das portas de comunicação e interfaces com o usuário. Essas tarefas compõe a estrutura mínima do controlador.

Tomando a estrutura mínima necessária para o controlador pode-se modelar o controlador através dos seguintes módulos funcionais: a Representação Interna do grafo (estrutura de dados), um Interpretador do grafo (módulo que constrói o grafo a

partir da descrição textual), um Gerenciador de Marcas (módulo responsável pelo disparo de transições), um árbitro interno responsável pela análise das condições booleanas, um controlador de I/O responsável pela conectividade do grafo com o meio externo e um módulo de interface com o usuário. A interligação ou fluxo de dados entre esses módulos está representada na figura 4.3-1:

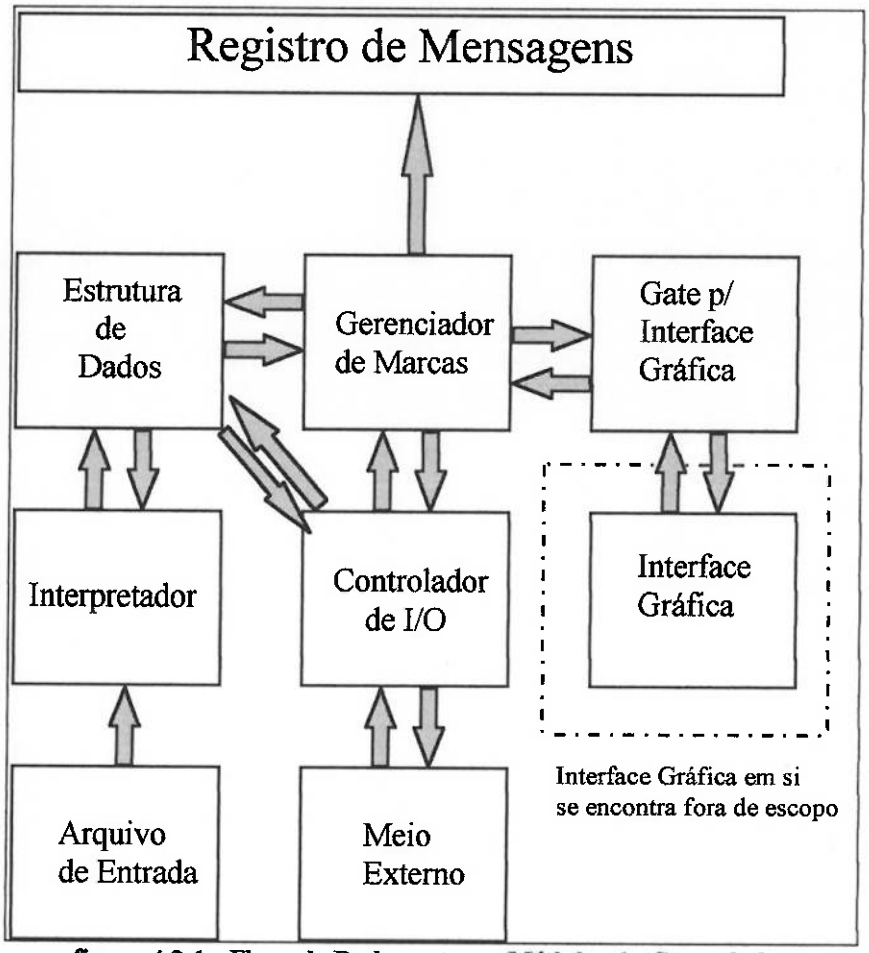


figura 4.3-1 - Fluxo de Dados entre os Módulos do Controlador

4.4 - Interfaces de Comunicação e Sinais de Controle

A fim de exercer a suas atividades de supervisão e controle o controlador deve receber sinais da planta e enviar sinais de controle para esta última. Para se implementar o controlador o mais genérico possível dentro da plataforma adotada, bem como facilitar a realização de testes de comunicação e controle, optou-se por implementar, inicialmente, apenas um método de comunicação baseado no protocolo DDE (Dynamic Data Exchange), que é um método padrão de troca de dados entre softwares dentro da plataforma Windows. Essa abordagem possui a vantagem de tornar a leitura e escrita no controlador independente do hardware com o qual se está comunicando, pois toda a comunicação é parametrizada, não havendo a necessidade

de se desenvolver um driver de comunicação no controlador para cada hardware em que se deseja ler ou escrever dados. Outra razão para se optar pelo protocolo DDE é o crescente interesse dos fabricantes de equipamentos de instrumentação e controle na disponibilização dos dados desses equipamentos via DDE na plataforma Windows. Exemplos desse comportamento são a Moeller e a Siemens, que possuem DDE servers para disponibilizar a leitura e escrita de dados em seus PLC's.

É interessante que o controlador rode em ambiente de rede, o que é possível utilizando o protocolo DDE diretamente numa rede Windows NT, ou indiretamente através de uma outra aplicação (escrita em Java, ou Visual Basic, por exemplo) que faça a interface entre o protocolo da rede e o padrão DDE.

No futuro, poderão ser adicionados módulos ao objeto controlador de I/O que permitam a comunicação diretamente com o protocolo de rede (TCP/IP, por exemplo).

5 - Linguagem de Programação *E-MFG Script*

Na seção 4.1 foi introduzida a necessidade de especificar uma linguagem de programação para o controlador E-MFG.

No presente caso, uma linguagem de programação nada mais seria que uma representação textual dos elementos que compõem um grafo E-MFG. Essa é a premissa básica do *E-MFG Script*.

Sendo o *EMFG script* a principal forma de interação entre o usuário e o controlador, a linguagem deve ser simples de usar e ao mesmo tempo garantir a representação de todos os elementos básicos da metodologia E-MFG, conforme discutido na seção 3.3.

A fim de facilitar a compreensão do código e a documentação dos grafos, o *E-MFG script* deve permitir a entrada de comentários.

Tendo em vista o desenvolvimento de uma futura interface para a construção de grafos que trabalhe em conjunto com o controlador, a estrutura do *E-MFG script* deve ser mecânica o suficiente para facilitar sua geração automática por uma interface gráfica.

5.1 - Estrutura do *E-MFG Script*

A estrutura proposta para o script baseia-se na declaração de *'tags'*, que indicam o começo e o fim de cada seção. Tanto a ordem quanto a nomenclatura das seções é rígida de forma a facilitar a interpretação pelo controlador. Mesmo que um grafo não contenha elementos de um determinado tipo, sua seção deve estar presente, apesar de se encontrar vazia.

Os comandos são separados pelo ponto-e-vírgula e a presença ou não de espaços entre os comandos é irrelevante. Todos as comparações de caracter são case-insensitive, ou seja não há distinção entre maiúsculas e minúsculas inclusive para nomes de atributos.

Os comentários são precedidos de caracteres de porcentagem, o resto da linha é desconsiderado pelo interpretador.

5.1.1 – Endereçamento

Conforme já mencionado anteriormente, prevê-se a comunicação do controlador via DDE. Tendo isto em vista, propõe-se uma forma de endereçamento que permita localizar um atributo de uma marca específica em qualquer grafo.

O endereçamento é feito de maneira hierárquica, o nome de um atributo sozinho é considerado local, ou pertencendo à própria box (e.g. ID).

O nome da box, seguida por uma exclamação, seguido por pelo nome do atributo se refere a uma box do próprio grafo (e.g. BOX1!ID).

O nome do grafo e o nome de um gate separados pelo sinal de exclamação representam um gate pertencente a outro grafo (e.g. GRAFO1!GATE).

Observe que no último caso, a comunicação só é realizada caso o gate do grafo de origem esteja declarado e disponível. Assim assegura-se que o usuário tem controle total sobre os dados que disponibiliza para outros grafos e aplicações DDE.

5.1.2 - Inclusão de Grafos Templates

A fim de tornar a descrição do grafo o mais modularizada possível, *o E-MFG Script* deve prever a inclusão de *'templates'* que representem elementos que se repetem no modelo. Em outras palavras, deve permitir a inclusão de sub-grafos armazenados em outros arquivos, constituindo assim uma biblioteca de objetos de modelagem reutilizáveis.

Como a identificação dos elementos do grafo deve ser única, será utilizado um método de concatenação para os *'labels'* do elementos do objeto a ser incluído. Assim, ao incluir um objeto do tipo 'torno' deve ser especificado o seu nome, como por exemplo 't1_'. Desse modo, um box interno ao 'torno' chamado 'prepara' passaria a ser referenciado como 't1_prepara'.

A única exceção à esta regra são os atributos de marca, que são adicionados ao conjunto de atributos do nível hierárquico original. Caso haja incompatibilidade nos tipos dos atributos, uma mensagem de erro deve ser gerada no momento da compilação.

5.2 – Seções do *E-MFG Script*

5.2.1 – Seção Inicial

Todo o script deve começar com as seguintes informações:

EMFG Script version : XX.XX;
Program Title: STRING;
Description: STRING;

A versão do script é usada para controle interno durante a fase de desenvolvimento. A número da versão do interpretador será fornecido para o usuário.

O título deve ser sempre de uma palavra. O grafo é identificado por essa palavra, inclusive para o registro na comunicação DDE. No caso de mais uma palavra ser usada, só a primeira é considerada.

A descrição serve para controle do usuário e pode conter até 255 caracteres.

5.2.2 - Tags <INCLUDE>

Essas 'tags' servem para a inclusão de sub-grafos reutilizáveis que representam elementos de modelagem repetidos:

```
<INCLUDE>
ADD(arquivo,prefixo);
...
</INCLUDE >
```

Esse comando define que **arquivo** será processado em conjunto com o presente grafo, sendo que todas os nomes de seus elementos serão precedidos de **prefixo**.

5.2.3 – Tags <EMFG>

Essas *'tags'* servem para controle do objeto interpretador, sendo que todas as demais seções devem estar contida entre essas *'tags'*:

```
<EMFG>
...
demais seções
...
</EMFG>
```

5.2.4 – Tags <MARKS>

Essa seção define os atributos da marca. Cada comando especifica o nome e o tipo de atributo. A sintaxe dos comandos é a seguinte:

```
<MARK>
INTEGER(n);
STRING(ID);
</MARK>
```

O primeiro comando adiciona um atributo inteiro chamado **n** e o segundo um atributo string de nome **ID**.

5.2.5 – Tags <BOXES>

A segunda seção define as boxes usadas. A sintaxe dos comandos é a seguinte:

```
<BOXES>
COMMON(BOX1);
CAPACITY(BOX2,2, LIFO);
PACKING(BOX3,10,FIFO);
UNPACKING(BOX4,LO);
TRANSFORMATOR(BOX5);
{
        IF (n == 1)
        THEN
                {
                TO_NUM(n,2);
                TO_STRING(NOME,FIRST);
                TO_ATTRIB(n,BOX1!N);
                TO_GATE(n, GATE1);
                }
        ELSE
                {
                ...
                }
}
</BOXES>
```

Uma box do tipo comum só precisa especificar seu nome (e.g. **BOX1**).

Uma box do tipo capacidade especifica seu nome, sua capacidade e seu método de fila (i.e. FIFO ou LIFO).

Uma box do tipo agrupador especifica seu nome, o tamanho do pacote e a ordem de inserção na lista.

Uma box do tipo dispersor precisa especificar seu nome e a forma de saída das marcas que pode ser tanto LO (*'Last Out'*) quando FO (*'First Out'*).

Um box transformador especifica seu nome e uma série de sentenças condicionais associadas a uma série de atribuições dentro dos blocos **THEN** e **ELSE**.

A sentença condicional aparece dentro dos parênteses depois do comando **IF** e deve apresentar uma sintaxe semelhante à linguagem C (ver capítulo 6 ítem 6.1.3). Os operadores suportados são:

Operação	Operador
igualdade	==
maior que	>
menor que	<
negação	~
AND	&&
OR	||

A meta é atingir um nível de detalhamento que permita a declaração do condicional da forma mais natural possível, como num programa em C. Isso implica inclusive na utilização de parênteses para alterar a ordem de avaliação.

As atribuições seguem os comandos **THEN** e **ELSE**. As atribuições dentro do **THEN** são executadas caso o condicional seja verdadeiro. Ambos o **ELSE** e o **THEN** não são opcionais, mas podem se encontrar vazios.

Espera-se pelo menos a presença de um bloco condicional, mas no caso dele não ser necessário a palavra **TRUE** deve estar entre parênteses. Assim garante-se que o conjunto de atribuições associado ao bloco **THEN** é sempre executado.

Existem quatro tipos de atribuição possíveis:

Tipo	Descrição
TO_NUM	atributo inteiro para constante inteira
TO_STRING	atributo string para constante string
TO_GATE	atributo inteiro/string para valor num gate
TO_ATTRIB	atributo inteiro/string para outro atributo do mesmo tipo pertencente à outra box

O interpretador não deve permitir a declaração de atribuições entre atributos de tipos diferentes.

5.2.6 – Tags <TRANSITS>

Essa seção define as transições usadas. A sintaxe dos comandos é a seguinte:

```
<TRANSITS>
COMMON(TRANS1);
TEMP(TEMP1,2);
CONDITIONAL(TRANS1,((BOX!N=1)&&(GATE<5)));
</TRANSITS>
```

A transição comum deve apenas especificar seu nome.

A transição temporizada especifica seu nome e o tempo de disparo em segundos.

O comando **CONDITIONAL** adiciona um condicional a uma dada transição, habilitando o disparo através de uma lógica do tipo AND (ver capítulo 6 ítem 6.1.3).

5.2.7 – Tags <ARCS>

Essa seção define os arcos. A sintaxe dos comandos é a seguinte:

```
<ARCS>
FROM_TRANSITION(ARC1,TRANS1,BOX1);
FROM_BOX(ARC2,TRANS2,BOX2);
ADD_FILTER (ARC1, NOT_PASS_COMPOSITE, PASS);
{
n; %apenas o atributo n da marca passará para o estado seguinte
}
ADD_FILTER(ARC2,NOT_PASS_COMPOSITE,NOT_PASS);
{
NOME; % o atributo NOME será anulado durante o disparo da transição
}
ADD_FILTER(ARC3,NOT_PASS_COMPOSITE,NOT_PASS_ALL);
{ % nenhum atributo passará adiante
}
</ARCS>
```

Os arcos que vem de transições são criados por **FROM_TRANSITION**, que especifica o nome do arco, a nome da transição de origem e o nome da box de destino.

Os arcos que vem de boxes são criados por **FROM_BOX**, que especifica o nome do arco, a nome da box de origem e o nome da transição de destino.

O comando **ADD_FILTER** adiciona um filtro ao arco especificado (um arco que não possui um comando ADD_FILTER associado recebe automaticamente um filtro com os valores padrão - PASS_COMPOSITE e PASS_ALL).

O parâmetro **PASS_COMPOSITE** em um filtro seletivo no arco permite a passagem da composição da marca. O parâmetro **NOT_PASS_COMPOSITE** anula a composição da marca.

O parâmetro **PASS** em um filtro seletivo no arco permite a passagem apenas dos atributos especificados, os demais são anulados. O parâmetro **NOT_PASS** anula os atributos especificados e permite a passagem dos demais. O parâmetro **NOT_PASS_ALL** anula todos os atributos de um arco. O parâmetro **PASS_ALL** permite a passagem de todos os atributos de um arco.

5.2.8 – Tags <GATES>

Essa seção define os gates. Na linguagem *EMFG Script*, todo e qualquer acesso à via de dados é feita através dos comandos desta seção (ampliando o conceito de *'gates'* para um elemento que transporta dados ou sinais de controle internos ou externos).

Os arcos de sinais de saída são criados pelos comandos que disponibilizam os dados das marcas para o meio externo ao grafo e o arcos de sinal de entrada (definidos na seção 3.3) são representados por meio dos comandos que trazem dados externos para o grafo. Assim, a diferenciação entre as portas internas, externas e os arcos de sinal de saída e entrada se dará pela funcionalidade imposta ao elemento de transporte de dados criado. Esta funcionalidade é determinada pelo comando utilizado na criação do objeto (NOTA: este elemento será chamado no *E-MFG Script* de *'gate'*, independentemente da sua função).

É importante frisar que apesar de se utilizar genericamente o nome *'gate'* para todos os elementos que transportam dados ou sinais de controle, pode-se ainda

distinguir (pela funcionalidade apresentada) os elementos estruturais do E-MFG que realizam esse transporte (e.g. portas habilitadoras internas, arcos de sinal de saída, etc.). Logo, não houve perda de representatividade ou de generalidade do *E-MFG Script* para a representação de grafos E-MFG.

A sintaxe dos comandos desta seção é a seguinte:

```
<GATES>
INTERNAL_PERMIT( label, boxorig, transdest);
INTERNAL_NOT_PERMIT( label, boxorig, transdest);
INTERNAL_FULL_PERMIT( label, boxorig, transdest);
INTERNAL_FULL_NOT_PERMIT(label, boxorig, transdest);
EXTERNAL_INPUT_PERMIT(label, transdest,mtcom,parcom,ativ);
EXTERNAL_INPUT_NOT_PERMIT(label,transdest,mtcom,parcom,ativ);
EXTERNAL_OUTPUT_ PERMIT(label,boxorig,mtcom,parcom,ativ);
EXTERNAL_OUTPUT_NOT_PERMIT(label,boxorig,mtcom,parcom,ativ);
EXTERNAL_OUTPUT_ FULL_PERMIT(label,boxorig,mtcom,parcom,ativ);
EXTERNAL_OUTPUT_ FULL_NOT_PERMIT(label,boxorig,mtcom,parcom,ativ);
INTERNAL_N(label,box_orig);
EXTERNAL_OUTPUT_DATA(label,boxorig,atriborig,mtcom,parcom,ativ);
EXTERNAL_OUTPUT_N (label,boxorig,atriborig,mtcom,parcom,ativ);
EXTERNAL_INPUT_DATA(label,tpret,mtcom,parcom,ativ);
ADD_CONDITIONAL(cond);
</GATES>
```

O *'gate'* interno de *permit* (habilitador) é criado pelo comando **INT_PERMIT** que especifica o nome do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a presença de marca na box de origem.

O *'gate'* interno de *not-permit* (inibidor) é criado pelo comando **INT_NOT_PERMIT** que especifica o nome do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a ausência de marca na box de origem.

O *'gate'* interno de *permit* específico para boxes capacidade, agrupador e dispersor é criado pelo comando **INT_FULL_PERMIT** que especifica o nome do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a box atingir sua capacidade máxima de marcas.

O *'gate'* interno de *not-permit* específico para boxes capacidade, agrupador e dispersor é criado pelo comando **INT_FULL_NOT_PERMIT** que especifica o nome

do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a box não ter atingido sua capacidade máxima de marcas.

O *'gate'* externo de entrada *permit* é criado pelo comando **EXT_INPUT_PERMIT** que especifica o nome do *'gate'*, a transição de destino, o método e a string concatenada para comunicação. O retorno é interpretado como booleano.

O *'gate'* externo de entrada *not-permit* é criado pelo comando **EXT_INPUT_NOT_PERMIT** que especifica o nome do *'gate'*, a transição de destino, o método e a string concatenada para comunicação. O retorno é interpretado como booleano.

O *'gate'* externo de saída *permit* é criado pelo comando **EXT_OUTPUT_PERMIT** que especifica o nome do *'gate'* e a box de origem. O retorno é TRUE quando há presença de marca no box.

O *'gate'* externo de saída *not-permit* é criado pelo comando **EXT_OUTPUT_NOT_PERMIT** que especifica o nome do *'gate'* e a box de origem. O retorno é TRUE quando há ausência de marca na box.

O *'gate'* externo de saída *permit* específico para boxes capacidade, agrupador e dispersor é criado pelo comando **EXT_OUTPUT_FULL_PERMIT** que especifica o nome do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a box atingir sua capacidade máxima de marcas.

O *'gate'* externo de saída *not-permit* específico para boxes capacidade, agrupador e dispersor é criado pelo comando **EXT_OUTPUT_FULL_NOT_PERMIT** que especifica o nome do *'gate'*, a box de origem e a transição de destino. A condição para retornar TRUE é a box não ter atingido sua capacidade máxima de marcas.

O *'gate'* interno de dados é criado pelo comando **INT_DATA** que especifica o nome do *'gate'*, a box de origem e o atributo. O tipo de retorno depende do tipo do atributo.

O *'gate'* externo de saída de dados é criado pelo comando **EXT_OUTPUT_DATA** que especifica o nome do *'gate'*, o box e o atributo de saída. O tipo de retorno dependo do tipo do atributo.

O *'gate'* externo de entrada de dados é criado pelo comando **EXT_INPUT_DATA** que especifica o nome do *'gate'*, o método e a string de comunicação. No caso DDE, a string de comunicação deve estar no formato "Aplicação!Tópico!Ítem". O tipo de retorno depende do tipo especificado.

O comando **CONDITIONAL** adiciona um condicional com os apresentados anteriormente. Esse condicional altera o valor de retorno do *'gate'* através de uma lógica AND (ver capítulo 6 ítem 6.1.3).

A tabela 5.2.8a apresenta um resumo da sintaxe dos comandos:

Tabela 5.2.8a - Comandos para Declaração de Gates

Comando	1	2	3	4	5	6
INTERNAL_PERMIT	label	box origem	trans. dest.			
INTERNAL_NOT_PERMIT	label	box origem	trans. dest.			
INTERNAL_FULL_PERMIT	label	box origem	trans. dest.			
INTERNAL_FULL_NOT_PERMIT	label	box origem	trans. dest.			
EXTERNAL_INPUT_PERMIT	label	trans. dest.	mét. com.	par. com.	ativação	
EXTERNAL_INPUT_NOT_PERMIT	label	trans. dest.	mét. com.	par. com.	ativação	
EXTERNAL_OUTPUT_ PERMIT	label	box origem	mét. com.	par. com.	ativação	
EXTERNAL_OUTPUT_ NOT_PERMIT	label	box origem	mét. com.	par. com.	ativação	
EXTERNAL_OUTPUT_ FULL_PERMIT	label	box origem	mét. com.	par. com.	ativação	
EXTERNAL_OUTPUT_ FULL_NOT_PERMIT	label	box origem	mét. com.	par. com.	ativação	
INTERNAL_N	label	box origem				
EXTERNAL_OUTPUT_DATA	label	box origem	atrib. origem	mét. com.	par. com.	ativação
EXTERNAL_OUTPUT_N	label	box origem	mét. com.	par. com.	ativação	
EXTERNAL_INPUT_DATA	label	tipo de ret.	mét. com.	par. com.	ativação	
ADD_CONDITIONAL	cond					

Onde define-se:

- label : um nome que define o gate unicamente;
- cond : uma sentença condicional
- box origem : box de onde o gate sai;
- atrib. origem : atributo da marca dentro do box de origem;
- trans. dest. : transição para onde o gate vai;
- mét. com. : método de comunicação, (só pode ser DDE);
- par. com. : parâmetro de comunicação concatenada (vide abaixo);
- ativação : ACTIVE ou PASSIVE.

No controlador, o único método de comunicação implementado é o DDE - (Dynamic Data Exchange). Nesse método são definidos três parâmetros para identificar unicamente um dado a ser transmitido: aplicação, tópico e ítem. Definiu-se a seguinte estrutura para a string concatenada de comunicação:

aplicação ! tópico ! ítem

Para maiores detalhes sobre a comunicação DDE, vide capítulo 6 ítem 6.6 sobre o objeto comunicador.

Quando se declara um gate externo passivo, tanto de entrada quanto saída, não é necessário se preocupar com a string concatenada, já que ela não será mesmo usada. Porém, não pode se deixar de preencher esse campo, uma vez que *todos* os parâmetros são esperados. Observe o exemplo abaixo:

EXTERNAL_INPUT_PERMIT (*exemplo1,trans1,DDE,dummy,PASSIVE*);

5.2.9 – Tags <INITIAL>

Essa seção define as condições iniciais do grafo. A sintaxe dos comandos é:

```
<INITIAL>
ADD_MARK(BOX1)
{
TO_NUM(N,0);
TO_STRING(ID,NOME1);
}
</INITIAL>
```

O único comando dessa seção **ADD_MARK** adiciona 1 marca na box indicada entre os parênteses. Os comandos de atribuição entre chaves indicam os valores iniciais para a marca.

5.3 – Exemplo de um Grafo descrito por *E-MFG Script*

Como exemplo será usado o grafo da figura 5.3-1, retirado da seção 2.6:

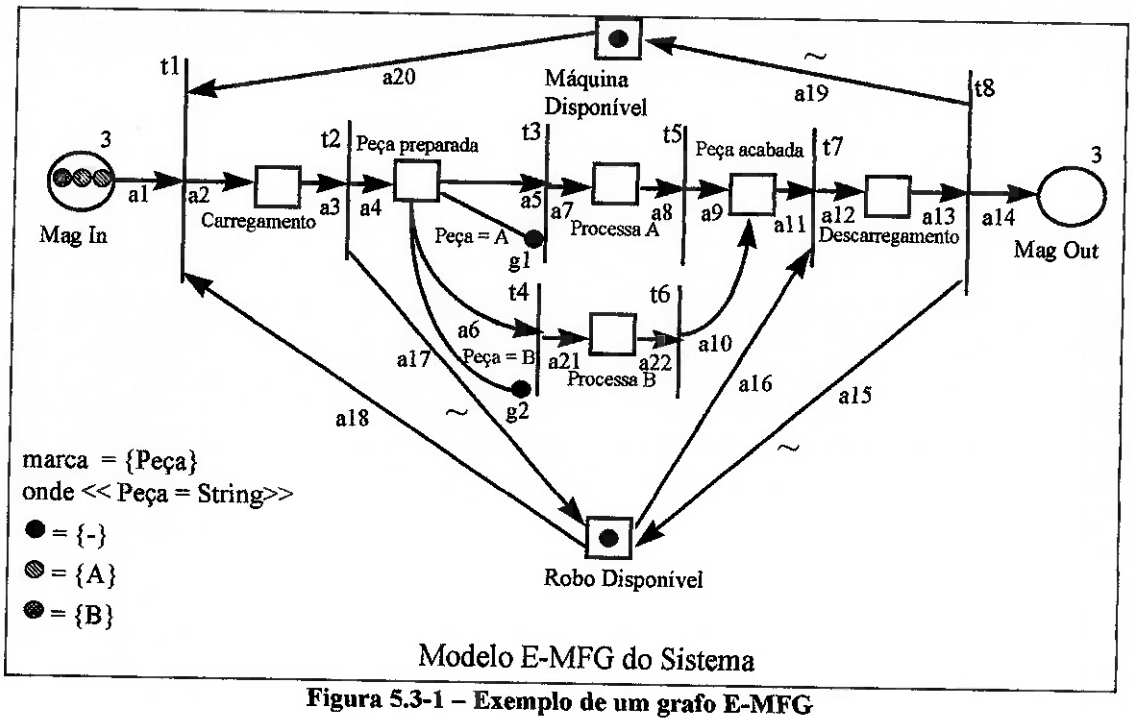


Figura 5.3-1 – Exemplo de um grafo E-MFG

EMFG Script version : 1.0;

Program Title: Exemplo;

Description: exemplo de script E-MFG;

% a próxima seção não inclui nenhum objeto reutilizável

<INCLUDE>

</INCLUDE>

% início da declaração do grafo

<EMFG>

% template dos atributos da marca

<MARKS>

STRING (PEÇA);

</MARKS>

```
% declaração das boxes
<BOXES>
CAPACITY(MAGOUT,3,FIFO);
COMMON (CARREGAMENTO);
COMMON (PREPARADA);
COMMON (MAQDISP);
COMMON (PROCA);
COMMON (PROCB);
COMMON (ROBDISP);
COMMON (ACABADA);
COMMON (DESCARREGAMENTO)
CAPACITY(MAGIN,3,FIFO);
</BOXES>
% declaração das transições
<TRANSITS>
COMMON (T1);
COMMON (T2);
COMMON (T3);
COMMON (T4);
COMMON (T5);
COMMON (T6);
COMMON (T7);
COMMON (T8);
</TRANSITS>
```

```
% declaração dos arcos orientados
<ARCS>
FROM_BOX(A1,MAGIN,T1);
FROM_TRANSITION(A2,T1,CARREGAMENTO);
FROM_BOX(A3,CARREGAMENTO,T2);
FROM_TRANSITION(A4,T2,PREPARADA);
FROM_BOX(A5,PREPARADA,T3);
FROM_BOX(A6,PREPARADA,T4);
FROM_TRANSITION(A7,T3,PROCA);
FROM_TRANSITION(A21,T4,PROCB);
FROM_BOX(A8,PROCA,T5);
FROM_BOX(A22,PROCB,T6);
FROM_TRANSITION(A9,T5,ACABADA);
FROM_TRANSITION(A10,T6,ACABADA);
FROM_BOX(A11,ACABADA,T7);
FROM_TRANSITION(A12,T7,DESCARREGAMENTO);
FROM_BOX(A13,DESCARREGAMENTO,T8);
FROM_TRANSITION(A14,T8,MAGOUT);
FROM_TRANSITION(A15,T8,ROBDISP);
FROM_BOX(A16,ROBDISP,T7);
FROM_TRANSITION(A17,T2,ROBDISP);
FROM_BOX(A18,ROBDISP,T1);
FROM_TRANSITION(A19,T8,MAQDISP);
FROM_BOX(A20,MAQDISP,T1);
% declaração dos filtros
ADD_FILTER(A15, NOT_PASS_COMPOSITE, NOT_PASS_ALL);
{
}
ADD_FILTER(A17, NOT_PASS_COMPOSITE, NOT_PASS_ALL);
{
}
ADD_FILTER(A19, NOT_PASS_COMPOSITE, NOT_PASS_ALL);
{
}
</ARCS>
```

```
% declaração dos gates
<GATES>
INT_PERMIT(G1,PREPARADA,T3);
INT_PERMIT(G2,PREPARADA,T4);
% declaração das condições para os gates
CONDITIONAL(G1,PEÇA='A');
CONDITIONAL(G2,PEÇA='B');
</GATES>
% declaração das marcações iniciais
<INITIAL>
ADD_MARK(MAGIN,2);
        {
        TO_STRING(PEÇA,A);
        }
ADD_MARK(MAGIN,1);
        {
        TO_STRING(PEÇA,B);
        }
ADD_MARK(MAQDISP,1);
        {
        }
ADD_MARK(ROBDISP,1);
        {
        }
</INITIAL>
</EMFG>
```

5.4 – Observação Sobre a Especificação do *E-MFG Script*

Esta é a primeira proposta de uma especificação de uma representação textual. Com base na experiência a ser adquirida nos próximos meses com a implementação de um interpretador, é possível que alterações possam a ser sugeridas.

6 - Estrutura de Dados

O objetivo desse capítulo é detalhar o modelo da estrutura de dados das classes de objetos que compõe o sistema.

A partir do modelo da solução apresentada no capítulo 4 (figura 4.3-1) construiu-se um modelo de estrutura de dados em 5 níveis hierárquicos (ver figura 6-1) descritos a seguir:

1. Elementos Estruturais Básicos: Constituem os micro-elementos que compõe os elementos estruturais do grafo tais como as condições booleanas, as atribuições e a estrutura dos atributos das marcas.

2. Elementos Estruturais do E-MFG: Constituem os objetos que de fato representam os elementos estruturais que compõe o grafo (boxes, transições, portas,etc..)

3. Representação do Grafo: Corresponde aos objetos que permitem o agrupamento de elementos estruturais básicos do E-MFG em listas que compõem a representação do grafo.

4. Elementos de Gerenciamento: Constituem os objetos que manipulam a dinâmica e as interfaces do grafo.

5. Aplicação: Corresponde ao nível de interação com o usuário

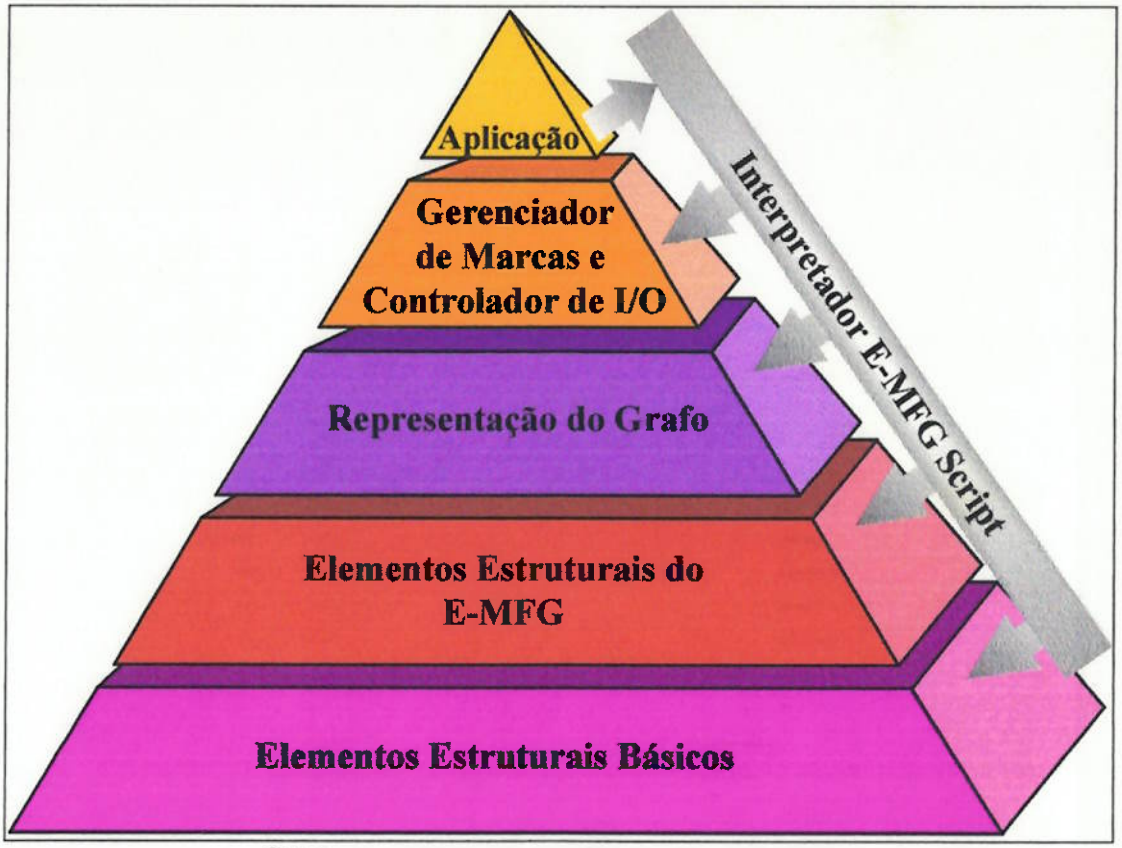


figura 6-1 - Hierarquia dos Objetos do Controlador

6.1 - Blocos Funcionais Elementares

A fim de se representar adequadamente os elementos do E-MFG, é necessário representar as propriedades desses elementos. Para facilitar essa representação serão adotadas classes de objetos que representem essas propriedades.

Identifica-se, *a priori*, 4 tipos de elementos que constituem as propriedades de um elemento do grafo, são eles os filtros dos arcos orientados, as condições booleanas, as atribuições e os atributos das marcas.

A seguir será feita a descrição da estrutura de dados de cada uma dessas representações.

6.1.1 Estrutura dos Atributos das Marcas

Conforme definido na descrição do *E-MFG Script* (Capítulo 5), cada atributo da marca é identificado por um nome ou *'label'* que o identifica de maneira unívoca. Serão suportados dois tipos básicos de atributos de marca: valores inteiros e texto (*'strings'*).

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CMarkAttribute	
Objetivo da Classe	Implementar a representação dos atributos das marcas	
Atributos da classe	**Tipo de dados**	**Descrição**
mp_Label	string pointer	Nome do atributo
m_Type	short integer(2 bytes)	Tipo de atributo (e.g.: inteiro)
mp_StringValue	string pointer	Texto do atributo se aplicável
m_IntegerValue	long integer(4 bytes)	Valor do atributo se aplicável
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: tipo e nome do atributo	void	inicializa o nome e os membros de dados do atributo da marca
SetAttrib parâmetros: o novo valor do atributo (string ou long integer, o que for aplicável para o tipo de atributo declarado)	void	atualiza o valor do atributo
GetTextValue parâmetros: nenhum	string	retorna o valor texto do atributo, se aplicável gerando um erro de depuração em caso contrário
GetIntegerValue parâmetros: nenhum	long integer	retorna o valor inteiro do atributo, se aplicável gerando um erro de depuração em caso contrário
GetLabel parâmetros: nenhum	string	retorna o nome do atributo
GetAsText parâmetros: nenhum	string	retorna o valor do atributo formatado em uma string
SetToNull parâmetros: nenhum	void	Anula o valor do atributo

6.1.2 Estrutura das Atribuições

As atribuições definidas no *E-MFG Script* (Capítulo 5) para o box transformador podem atribuir a um atributo de marca valores provenientes de constantes (texto ou valores inteiros), de um atributo de uma marca pertencente a um box do grafo, ou de um valor de um gate de dados.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CAttribution	
Objetivo da Classe	Implementar a representação das atribuições	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Type	short integer(2 bytes)	Tipo de atribuição (e.g.: =cte)
m_BoxIndex	long integer(4 bytes)	Box que contém a marca cujo o atributo vai ser alterado
m_MarkAttributeIndex	long integer(4 bytes)	Índice na lista de atributos do atributo que vai ser alterado
m_EqualStringCte	string	Constante de Texto
m_EqualIntegerCteOrElementIndex	long integer(4 bytes)	Constante Numérica ou Índice do Elemento que representa o valor a ser atribuído
m_EqualMarkAtributeIndex	long integer(4 bytes)	Índice na lista de atributos do atributo que possui o valor a ser atribuído
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: tipo, índice do Box, índice do atributo e constante numérica ou índice do gate; ou tipo, índice do Box, índice do atributo e string constante; ou tipo, índice do Box, índice do atributo, índice do segundo box e índice do segundo atributo	void	inicializa os parâmetros internos de acordo com o tipo de atribuição
DoAttribution parâmetros: listas de boxes e gates do grafo	void	realiza a atribuição a partir dos valores atuais do grafo

6.1.3 Estrutura das Condições Booleanas

As sentenças condicionais foram introduzidas no E-MFG com objetivo de explicitar regras de controle associadas a valores de atributos de marca. (SANTOS FILHO, D. J., 1993)

Sobre as sentenças condicionais, nesta seção será:

- revisto seu uso no EMFG;
- definidos seus elementos;
- apresentada sua implementação como objeto;
- definida sua interface com o usuário através do *EMFG Script*.

No EMFG foi previsto o uso de sentenças condicionais associadas aos gates inibidores ou habilitadores e em restrições adicionais associadas à transições.

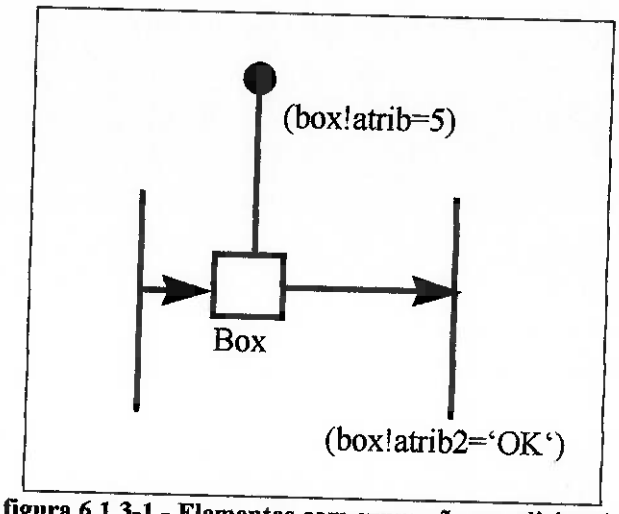


figura 6.1.3-1 - Elementos com expressões condicionais

No exemplo o gate só permite o disparo da transição caso haja uma marca na box de origem e o atributo indicado dessa marca corresponda a um valor específico. Dessa forma, fica claro que a sentença condicional representa uma regra extra de controle.

Neste trabalho, interpretou-se essa característica da seguinte maneira : *a sentença condicional sempre representa uma adição booleana às regras usuais do E-MFG.* Assim foi possível estender o uso das sentenças condicionais a outros elementos, tais como gates de dados.

Nos gates de dados, a transmissão de dados só é permitida quando a sentença condicional é verdadeira

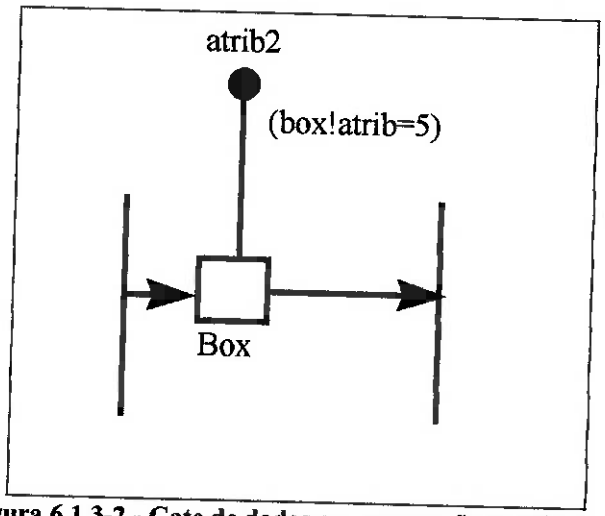


figura 6.1.3-2 - Gate de dados com expressão condicional

No caso das transições, tanto comuns quanto temporizadas, o disparo só é feito quando todas as condições usuais estão satisfeitas além da sentença condicional ser verdadeira.

Outro uso das sentenças condicionais está nos boxes transformadores . Nesse caso, a avaliação da sentença condicional indica qual bloco será executado (i.e. se verdadeira o bloco THEN, se falsa o bloco ELSE):

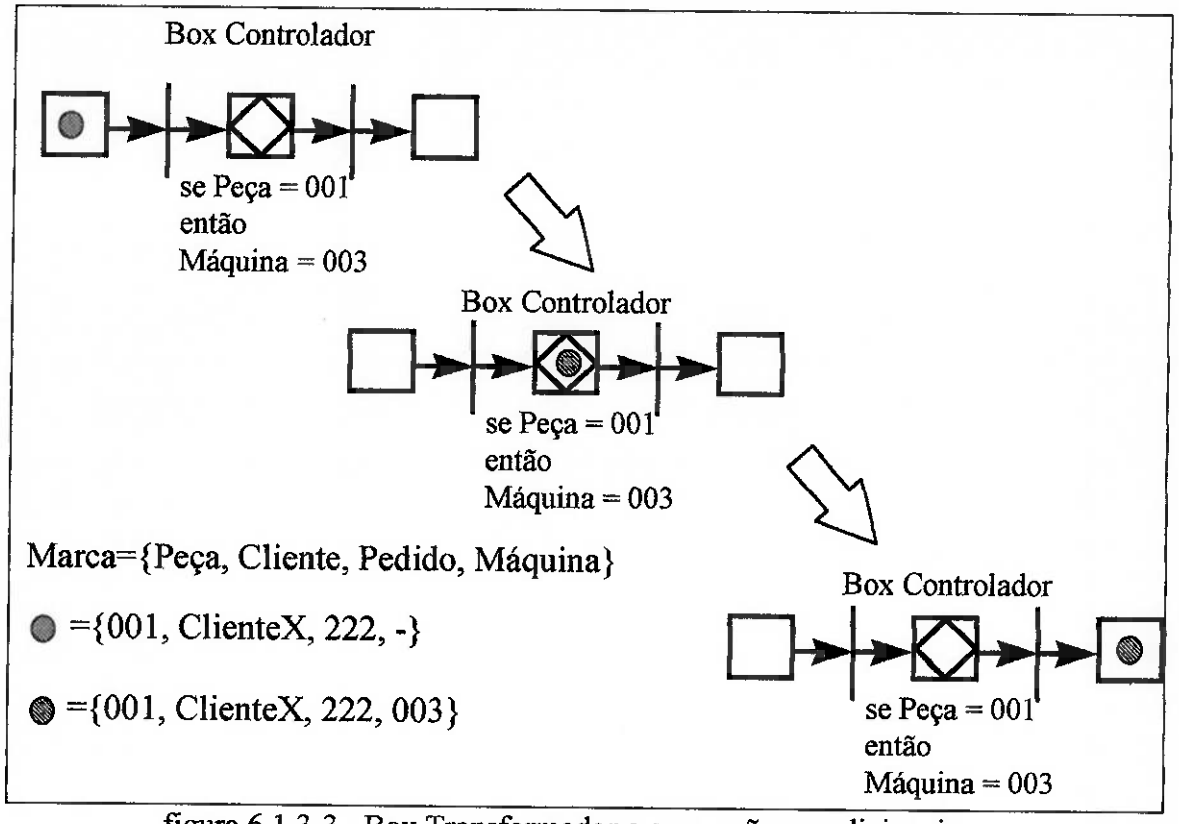


figura 6.1.3-3 - Box Transformador e expressões condicionais

As sentenças condicionais podem ser divida em: simples ou compostas.

Uma sentença simples é aquela que apresenta apenas uma relação condicional.

Já uma sentença composta apresenta várias relações concatenadas por operadores lógicos do tipo AND ou OR, e cuja ordem de avaliação pode ou não ser definida através de parenteses.

Sentenças Simples

A divisão dos elementos de uma sentença condicional simples pode ser feita da seguinte forma: (LVALUE) OP (RVALUE) = TRUE/FALSE, ou NOT((LVALUE) OP (RVALUE)) = TRUE/FALSE, onde o operador *NOT* age negando o valor final do condicional.

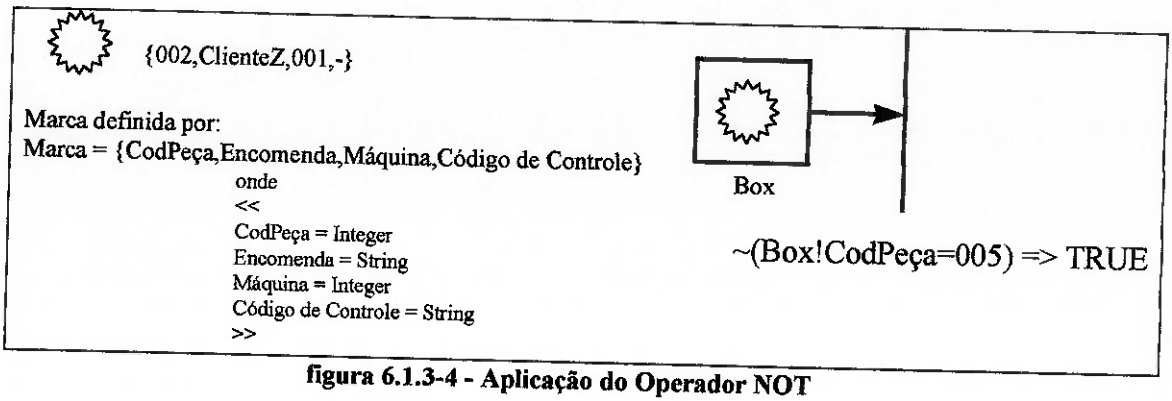


figura 6.1.3-4 - Aplicação do Operador NOT

No caso do EMFG, o *L-Value* é sempre um atributo de marca. Aqui vale introduzir a notação utilizada para se referenciar os atributos dentro de um grafo. O sinal de exclamação separa a box que contém a marca do atributo contido pela marca:

O operador relacional pode ser: *igualdade*, *diferença*, *maior que* ou *menor que*. Nesse trabalho, por simplicidade, optou-se utilizar apenas esses quatro tipos, já que conjugando com o operador NOT pode-se obter os demais tipos (i.e. *diferente*, *menor ou igual a*, *maior ou igual a*, etc...).

Já o *R-Value* pode ser interpretado de várias maneiras. Na proposta original de EMFG, o R-Value era sempre considerado como um valor inteiro constante. Tendo em vista as mudanças propostas nessa trabalho, admite-se que o *R-Value* possa ser dos seguintes tipos: inteiro constante, string constante, valor de um gate de dados ou valor do atributo de outra marca.

A avaliação do condicional dá-se em tempo real de processamento do controlador, ou seja trata-se de uma avaliação *on-line* que leva em conta o estado atual do grafo.

Sentenças compostas

Uma sentença composta corresponde a diversos condicionais simples que se relacionam através de operadores lógicos do tipo AND ou OR, na ordem imposta pelos parênteses.

Uma forma de resolver a ordem de avaliação é representar a sentença condicional como uma árvore binária, cujas folhas são os condicionais simples e cujos nós representam os operadores lógicos AND ou OR.

Um exemplo desse tipo de sentença representada através de árvore encontra-se a seguir.

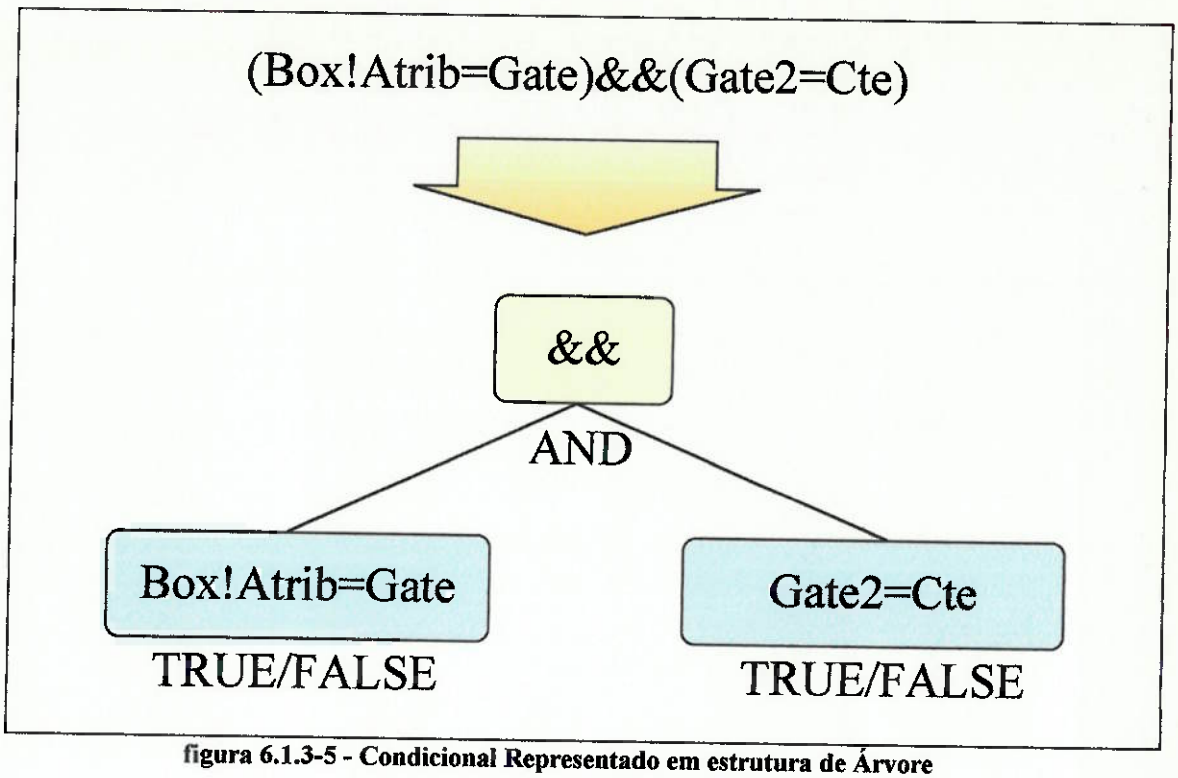


figura 6.1.3-5 - Condicional Representado em estrutura de Árvore

Implementação das Sentenças Condicionais em Objeto

Na seção anterior especificou-se que as sentenças condicionais seriam representadas por árvores binárias. Não se pretende rever os conceitos envolvidos com a representação de árvores binárias por meio de ponteiros, basta dizer que o objeto possuirá dois ponteiros, um para o filho da esquerda, outro para o filho da direita. Quando não houver filhos, os ponteiros terão valores nulos. Observe a figura 6.1.3-6, equivalente à árvore da figura 6.1.3-5.

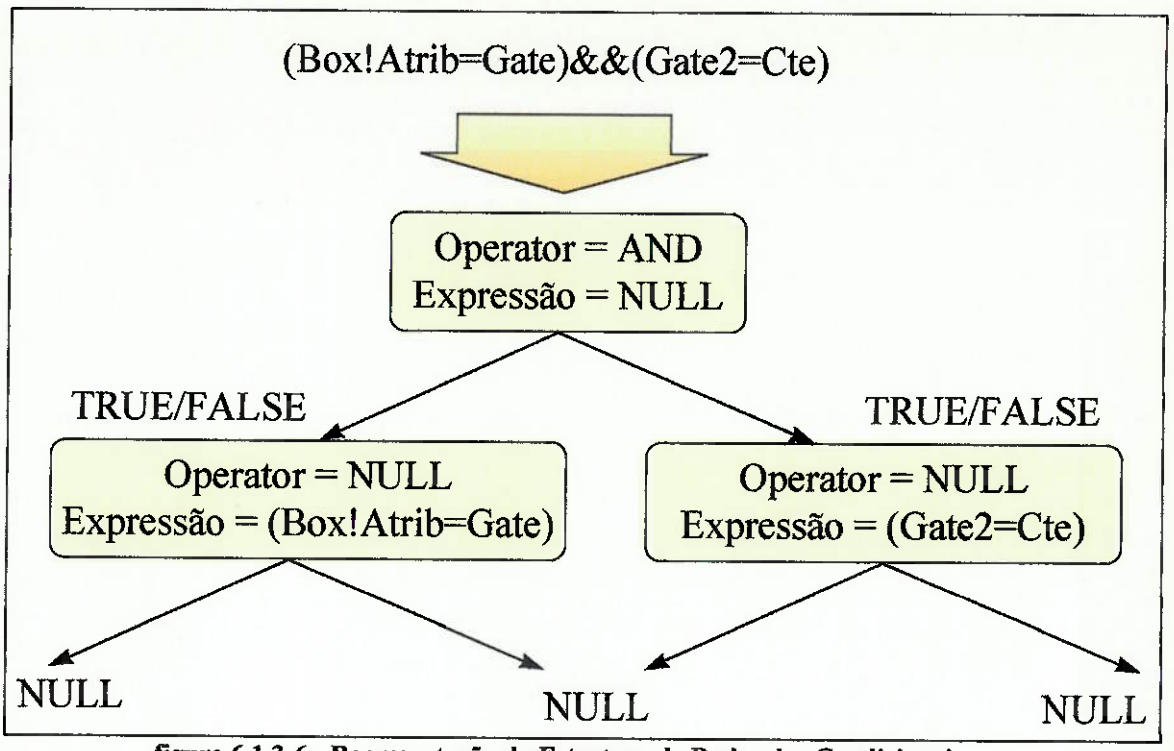


figura 6.1.3-6 - Representação da Estrutura de Dados dos Condicionais

A estrutura da classe de objeto de sentenças condicionais é dada por:

Definição da Classe de Objeto		
Nome da Classe de Objeto	CConditional	
Objetivo da Classe	Implementar a representação das sentenças condicionais	
Atributos da Classe	Tipo de Dados	Descrição
mp_Owner	pointer to CGraph	ponteiro para o grafo-dono
m_Logic	short integer (2 bytes)	lógica interna (AND,OR, etc..)
m_NOT	booleano	operador NOT
m_LBox	short integer (2 bytes)	índice da box associada ao *LValue*
m_LAttrib	short integer (2 bytes)	índice do atributo associado ao *LValue*
m_Operator	short integer (2 bytes)	operador (EQUAL,GREATER,etc...)
m_RType	short integer (2 bytes)	tipo do *RValue*
m_RIntCte	short integer (2 bytes)	constante inteira associada ao *RValue*
m_RStrCte	string	constante string associada ao *RValue*
m_RBox	short integer (2 bytes)	índice da box associada ao *RValue*
m_RAttrib	short integer (2 bytes)	índice do atributo associado ao *RValue*

m_RGate	short integer (2 bytes)	índice do gate associado ao *RValue*
mp_Right	pointer to CConditional	ponteiro para filho da direita
mp_Left	pointer to CConditional	ponteiro para filho da esquerda
Métodos da Classe	**Tipo de Retorno**	**Descrição**
Create parâmetros (lógica do nó, ponteiro p/ grafo, operador de negação, índica da LBox, índice da LAttrib, tipo do operador, tipo de RValue, parâmetro 1 do RValue, parâmetro 2 do RValue, parâmetro 3 do RValue)	void	inicializa os membros do CConditional. Os parâmetros de *RValue* são interpretados de acordo com o tipo de condicional que esteja sendo criado
Evaluate()	BOOL	avalia o condicional, tendo como parâmetros a condição atual do grafo.
SetAlwaysTrue()	void	o condicional sempre responde TRUE
SetAlwaysFalse()	void	o condicional sempre responde FALSE
AddRight parâmetro (ponteiro para o CConditional à direita)	void	adiciona um elemento à direita
AddLeft parâmetro (ponteiro para o CConditional à esquerda)	void	adiciona um elemento à esquerda

O membro *m_Logic* pode assumir os seguintes valores inteiros simbólicos:

AND, OR, ALWAYS_TRUE, ALWAYS_FALSE

O membro *m_Operator* pode assumir os valores:

EQUAL, GREATER, LESSER

O membro *m_RType* pode assumir os valores:

INT_CTE, STR_CTE,

INT_BOX_ATB, STR_BOX_ATB,

INT_GATE_VAL, STR_GATE_VAL

A função *Evaluate()* realiza a lógica associada à sentença condicional, percorrendo toda a árvore binária e avaliando suas folhas, em seguida os nós são avaliados até o valor final:

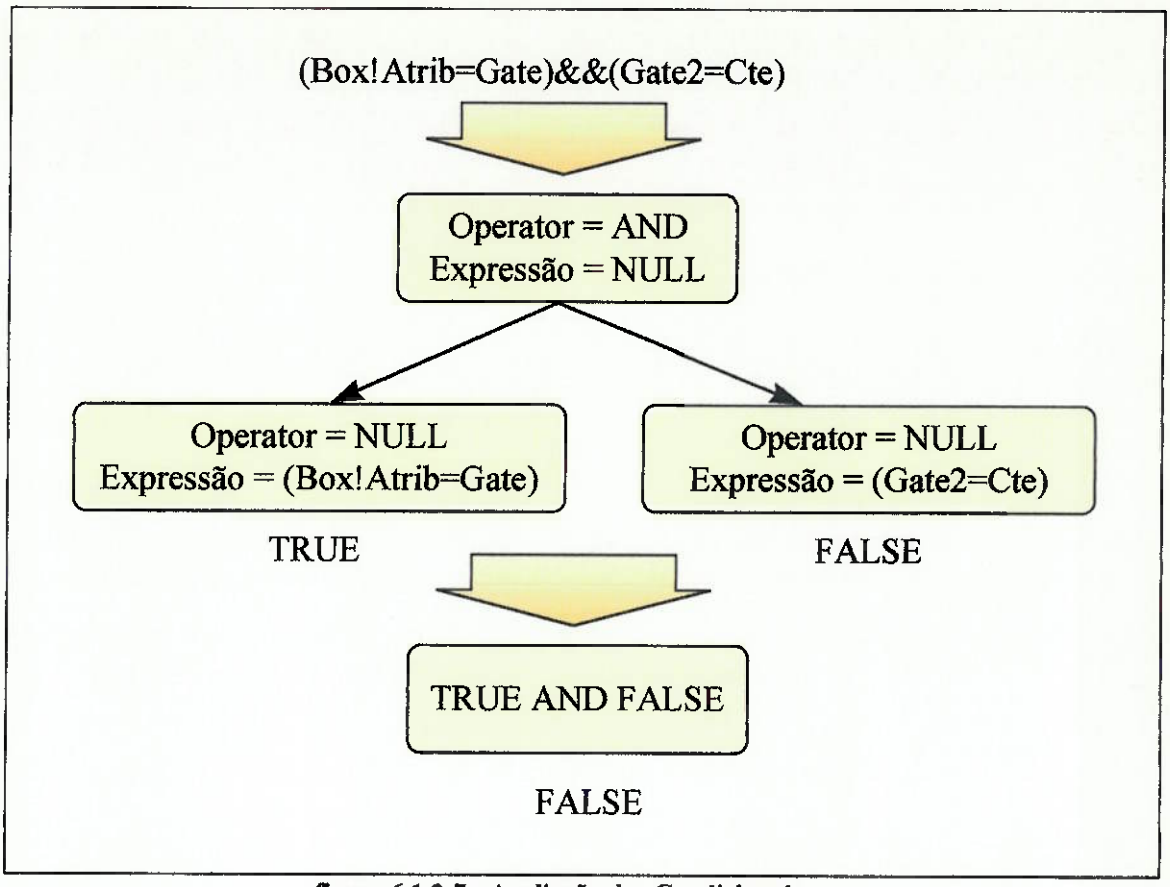


figura 6.1.3-7 - Avaliação dos Condicionais

Interface através do *EMFG Script*

Deve ficar claro que a principal preocupação ao elaborar-se a interface através do *EMFG Script* foi manter uma consistência com as principais linguagens de programação disponíveis, especialmente com C e Java.

Como já foi mencionado, o *L-Value* será sempre um atributo de marca associado através de uma referência do tipo *box!atributo*. A única exceção a essa regra é quando a sentença condicional pertence a um bloco de um box transformador. Caso não se especifique o *box*, o atributo será considerado local, ou seja, contido na marca presente no próprio box:

Os operadores são definidos da seguinte forma:

Operador	Símbolo
NOT	~
igualdade	==
maior que	>
menor que	<

A única observação é que o operador NOT deve sempre ser utilizado em conjunto com parênteses.

Conforme descrito anteriormente, o *R-Value* pode ser interpretado de quatro formas diferentes:

- valor inteiro : um número inteiro qualquer (pode ser negativo);
- valor string : um string delimitado por aspas simples;
- gate de dados : label do gate;
- atributo : *referência box!atributo.*

Usando essas regras para o declaração as sentenças condicionais, o EMFG Script se aproxima das linguagens de programação mais comuns, especialmente ao C ou Java.

6.1.4 Estrutura dos Filtros dos Arcos Orientados

Os filtros dos arcos orientados definidos no *E-MFG Script* (Capítulo 5) executam a filtragem seletiva dos atributos de uma marca quando ocorre o disparo de uma transição. O filtro pode ser dos seguintes tipos: passa tudo, passa nada, passa atributo e não passa atributo de acordo com a definição dada no Capítulo 2, seção 2.5.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CArcFilter	
Objetivo da Classe	Implementar a representação dos Filtros das Condições Booleanas	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Type	short integer(2 bytes)	Tipo do filtro (e.g.: PASSA)
m_AttribNumber	long integer(4 bytes)	Número de Atributos de Filtragem
mp_AttribIndex	long integer pointer	Lista de índices na lista de atributos dos atributos que formam o filtro
mp_Graph	CGraph	Ponteiro para o grafo
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: tipo, número de atributos, lista de atributos, grafo	void	inicializa os parâmetros internos de acordo com o tipo de filtro
Apply parâmetros: CMark	CMark	Aplica o filtro na marca passada como parâmetro anulando os atributos relevantes e retornando a marca filtrada

6.2 - Representação dos Elementos do E-MFG

O E-MFG possui basicamente 5 tipos de elementos estruturais boxes, transições, *'gates'*, arcos orientados e marcas. Na representação da estrutura de dados adotada os elementos do MFG foram considerados um caso particular do E-MFG.

Assim os diversos subtipos de elementos do grafo serão definidos por um parâmetro interno de cada elemento.

6.2.1 - Representação das Marcas

As marcas no *E-MFG Script* (Capítulo 5) podem ser de dois tipos: compostas ou simples, sendo que todas as marcas em um grafo possuem a mesma estrutura de atributos e a presença ou não de um atributo está associada ao valor desse atributo ser igual ou diferente de nulo.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CMark	
Objetivo da Classe	Implementar a representação das marcas do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
m_CompositeMark	BOOL	Tipo de marca (e.g.: comum = FALSE ou composta=TRUE)
mp_AttributesArray	CMarkAttribArray	Lista de atributos da marca
mp_MarkList	CMarkArray	Lista de marcas que compõem uma marca composta
m_WasExploded	BOOL	Flag se a marca composta vai ser decomposta nas suas marcas originais
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: lista de atributos ou lista de atributos e lista de marcas	void	inicializa a marca com os atributos apropriados e atualiza as flags internas e a lista de marcas se aplicável
GetAttrib parâmetros: índice do atributo	CMarkAttribute	Retorna o atributo requisitado do vetor de atributos
SetAttrib parâmetros: índice do atributo e valor numérico ou string	void	Aplica o valor passado como parâmetro ao atributo especificado
SetAllAttributesToNull parâmetros: nenhum	void	Anula os valores de todos os atributos do vetor de atributos
ResetComposition parâmetros: nenhum	void	Anula a composição da marca transformando-a em uma marca simples
MergeMark	CMark	Mescla os atributos da marca atual

parâmetros: CMark		(this) e a marca recebida como parâmetro, mesclando também a composição de marcas
RemoveMark parâmetros: índice da marca	void	Remove uma marca da lista de marcas que compõem uma marca composta
ExplodeComposite parâmetros: índice da marca	CMarkArray pointer	Retorna um ponteiro para a lista de marcas de uma marca composta colacando a flag m_WasExploded para TRUE
GetCompositeMarks parâmetros: índice da marca	CMarkArray pointer	Retorna um ponteiro para a lista de marcas de uma marca composta

6.2.2 - Representação dos Boxes

Os boxes no *E-MFG Script* (Capítulo 5) podem ser dos seguintes tipos: simples, temporizado, agrupador, dispersor, transformador e capacidade. A implementação desses tipos de boxes pode ser realizada através da manutenção dos atributos de uma classe genérica de objeto.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CBox	
Objetivo da Classe	Implementar a representação dos boxes do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Type	short integer (2 bytes)	Tipo de Box (e.g.: comum, capacidade, etc)
m_Label	string	Nome do Box
mp_Graph	CGraph pointer	Ponteiro para o grafo
mp_BoxMarkArray	CMarkArray	Lista de marcas que o box possui
m_TimeCte	long integer (4bytes)	Constante de tempo do box em segundos (comum=0)
m_ActualTimeCount	long integer (4bytes)	contagem de tempo atual
m_TimerOn	BOOL	marca a temporização ativa
m_TimerRunned	BOOL	marca se a condição de temporização está satisfeita
m_LastCheck	CTime	timestamp da última checagem de tempo
m_Capacity	long integer (4bytes)	quantidade de marcas que oo box pode conter (comum = 1)
m_FIFO	BOOL	identifica se a regra de saída do box é FIFO ou LIFO
m_HasAnyMark	BOOL	Identifica se o box possui alguma marca
m_NumAttribuitions	long integer (4bytes)	Número de atribuições do box (somente aplicável para o box transformador)
m_NumOriginArcs	long integer (4bytes)	número de arcos de origem

m_NumDestinyArcs	long integer (4bytes)	número de arcos de destino
mp_Attribuitions	long integer pointer	lista de índices de atribuições na tabela de atribuições
mp_OriginArcs	long integer pointer	lista de índices de arcos de origem
mp_DestinyArcs	long integer pointer	lista de índices de arcos de destino
m_NumCondition	long integer (4bytes)	Número de condições do box (somente aplicável para o box transformador)
mp_ConditionList	long integer pointer	lista de índices para as condições
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: tipo, nome, número de arcos de origem, lista de arcos de origem, número de arcos de destino, lista de arcos de destino, capacidade, regra de saída(FIFO ou LIFO), constante de tempo, número de atribuições, lista de atribuições, número de condições e lista de condições	void	inicializa o box com os atributos apropriados e atualiza as flags internas e os valores default quando aplicável
AddMark parâmetros: CMark	void	Coloca uma marca no box
GetMarkAttribute parâmetros: índice do atributo	CMarkAttribute	Retorna o atributo requisitado do vetor de atributos da marca
SetMarkAttribute parâmetros: índice do atributo e valor numérico ou string	void	Aplica o valor passado como parâmetro ao atributo especificado da marca
HasMark parâmetros: nenhum	BOOL	Retorna a existência de marca no box
HasNorMoreMarks parâmetros: quantidade N de marcas	BOOL	Retorna a existência de N ou mais marcas no box
GetType parâmetros: nenhum	short integer (2 bytes)	retorna o tipo de box
RemoveMark parâmetros: nenhum	CMark	Remove uma marca do box
GetNumOriginArcs parâmetros: nenhum	long integer (4bytes)	Retorna o número de arcos de origem que chegam ao box
GetNumDestinyArcs parâmetros: nenhum	long integer (4bytes)	Retorna o número de arcos de destino que saem do box
GetOriginArcsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de arcos de origem que chegam ao box
GetDestinyArcsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de arcos de destino que saem do box
GetOriginTrantitionsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de transições de origem que possuem arcos de saída que chegam ao box
GetDestinyTransitionsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de transições de destino que possuem arcos de entrada que saem do box
RunTransformationEngine parâmetros: nenhum	void	Realiza as atribuições do box transformador
IncrementTimeCount parâmetros: delta T	void	Incrementa a contagem de tempo atual
ResetTimeCount parâmetros: nenhum	void	Zera a contagem de tempo
CheckTime	BOOL	Retorna se a temporização já foi

parâmetros: nenhum		satisfeita
StartTimer parâmetros: nenhum	void	Inicia a contagem de tempo
UpdateTimer parâmetros: nenhum	void	atualiza a contagem de tempo

6.2.3 - Representação dos Gates

No EMFG os gates são estruturas que representam regras adicionais para o disparo de transições. Alternativamente, pode-se interpretar o funcionamento dos gates como responsáveis pela transmissão de dados num grafo.

Tendo em vista um sistema onde a estratégia de controle de uma planta é representada por meio de um grafo, pode-se dizer que os gates representam sinais de entrada da planta para o controlador, tanto quanto sinais de saída do controlador para a planta.

Assim, é possível ampliar o conceito de gates apresentado na definição original do E-MFG (SANTOS FILHO, 1993). Neste trabalho propõe-se a inclusão de gates de dados que sejam responsáveis por transmitir valores de atributos pertencentes a marcas internas do grafo. Note que isso não viola o princípio que impede o trânsito de marcas entre os diversos níveis hierárquicos de controle, uma vez que só valores de atributos individuais são transmitidos, assim a integridade dos dados é mantida.

Para o controlador, o gate é uma estrutura que armazena os diversos índices internos dos grafo e todos os parâmetros para comunicação externa.

As seções seguintes abordam os seguintes tópicos:

- a classificação dos diferentes tipos de gates;
- a estrutura da classe de objetos associada aos gates;

Classificação dos Gates

O controlador apresentado nesse trabalho disponibiliza os seguintes tipos de gate:

- permit internos;
- non-permit internos;
- permit internos de capacidade N;
- non-permit internos de capacidade N;
- permit externo de entrada;
- non-permit externo de entrada;

- permit externo de saída;
- non-permit externo de saída;
- permit externo de saída de capacidade N;
- non-permit externo de saída de capacidade N;
- dados externo de saída;
- dados externo de entrada;

Os gates são considerados internos quando estão inteiramente contidos dentro do grafo representado no controlador. São considerados externos quando implicam numa comunicação com a planta, nesse caso ainda pode ser sub-divididos em de saída ou entrada.

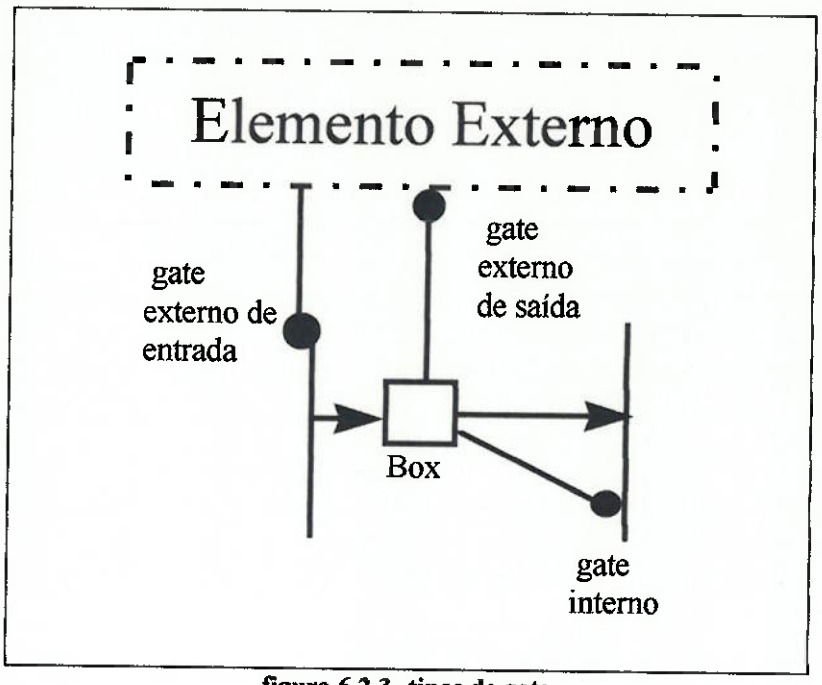


figura 6.2.3- tipos de gate

No contexto do controlador, os gates estão intimamemente ligados ao objeto comunicador (ítem 6.6). Assim, quanto aos gates externos é necessário fazer uma classificação adicional quanto a sua ativação.

Os gates são considerados passivos quando aguardam que uma aplicação externa realize a comunicação. Em outras palavras, não fica a cargo do controlador alterar os sinais de entrada ou saída do grafo.

Os gates são considerados ativos quando o próprio controlador é responsável por alterar os sinais de entrada ou saída.

Finalmente, quando um gates externo de entrada de dados é declarado, torna-se necessário especificar o seu tipo de retorno, que tanto pode ser inteiro quanto

string. Isso é necessário para se determinar qual a função será chamada para que a conversão de dados seja apropriada. Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição da Classe de Objeto		
Nome da Classe de Objeto	CGate	
Objetivo da Classe	Implementar a representação dos gates	
Atributos da Classe	**Tipo de Dados**	**Descrição**
mp_Owner	ponteiro p/ CGraph	ponteiro para o grafo-dono
m_Type	short integer (2 bytes)	tipo do gate
m_Label	string	nome do gate
m_Box	short integer (2 bytes)	box de origem
m_Attrib	short integer (2 bytes)	atributo na marca do box de origem
m_Transition	short integer (2 bytes)	transição de destino
m_Method	string	método de comunicação
m_Parameter	string	parâmetro de comunicação
m_Activation	short integer (2 bytes)	tipo de ativação
m_RetType	short integer (2 bytes)	tipo de retorno
m_String	string	valor string de retorno
m_Boolean	BOOL	valor booleano de retorno
m_Integer	short integer (2 bytes)	valor Inteiro de retorno
mp_Cond	pointer to CConditional	ponteiro p/ condicional associada
Métodos da Classe	**Tipo de Retorno**	**Descrição**
Create parâmetros(ponteiro p/ grafo, tipo do gate, tipo de retorno do gate, label do gate, box de origem, atributo de origem, transição de destino, método de comunicação, parâmetro de comunicação, ativação)	void	inicializa o gate com os valores fornecidos. A forma com que os parâmetros são interpretados pode variar conforme o tipo de gate que está sendo construído.
SetBOOL parâmetros : (valor booleano)	void	altera o valor do membro de retorno booleano

GetBOOL()	BOOL	retorna o valor booleano do gate
SetInteger parâmetros : (valor inteiro)	void	altera o valor do membro de retorno inteiro
GetInteger()	short integer (2 bytes)	retorna o valor inteiro do gate
SetString parâmetros : (valor string)	void	altera o valor do membro de retorno string
GetString()	string	retorna o valor string do gate
AddConditional parâmetro(condicional)	void	adiciona uma sentença condicional ao gate
RemoveConditional()	void	remove sentença condicional

O membro m_Type pode assumir os seguintes valores inteiros simbólicos:

INT_PERMIT
INT_NOT_PERMIT
INT_FULL_PERMIT
INT_FULL_NOT_PERMIT
EXTERNAL_INPUT_PERMIT
EXTERNAL_INPUT_NOT_PERMIT
EXTERNAL_OUTPUT_PERMIT
EXTERNAL_OUTPUT_NOT_PERMIT
EXTERNAL_OUTPUT_FULL_PERMIT
EXTERNAL_OUTPUT_FULL_NOT_PERMIT
INT_N
EXTERNAL_OUTPUT_DATA
EXTERNAL_OUTPUT_N
EXTERNAL_INPUT_DATA

O membro m_Activation pode assumir os seguintes valores inteiros simbólicos:

ACTIVE

PASSIVE

O membro m_RetType pode assumir os seguintes valores inteiros simbólicos:

NUMERICAL_DATA

STRING_DATA

BOOLEAN_DATA

O valor interno do gate (seja inteiro, string ou booleano) representa o estado do grafo no último momento de atualização. Em outras palavras, as funções

GetInteger(), GoAhead() e GetString() retornam o valor da última amostragem realizada pelo controlador.

A função AddConditional() adiciona uma sentença condicinal ao gate, de forma a desabilitar sua saída quando é falsa. Por 'desabilitar a saída' entende-se o seguinte:

- GetInteger() retorna valor NULO;
- GetString() retorna string nula "";
- GoAhead() retorna FALSE.

6.2.4 - Representação das Transições

As transições no E-MFG Script (Capítulo 5) podem ser dos seguintes tipos: comuns, ou temporizadas, podendo ou não possuir restrições adicionais. As transições também são os elementos responsáveis pela atualização do estado do grafo e são elas que devem interagir com os boxes ao se efetuar o disparo de uma transição, cabendo ao jogador de marcas apenas resolver os conflitos e autorizar o disparo.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CTransition	
Objetivo da Classe	Implementar a representação das transições do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Type	short integer (2 bytes)	Tipo de Transição (e.g.: comum ou temporizada)
m_Label	string	Nome da Transição
mp_Graph	CGraph pointer	Ponteiro para o grafo
m_TimeCte	long integer (4bytes)	Constante de tempo da transição em segundos (comum=0)
m_ActualTimeCount	long integer (4bytes)	contagem de tempo atual
m_TimerOn	BOOL	marca a temporização ativa
m_TimerRunned	BOOL	marca se a condição de temporização está satisfeita
m_LastCheck	CTime	timestamp da última checagem de tempo
m_NumOriginArcs	long integer (4bytes)	número de arcos de origem
m_NumDestinyArcs	long integer (4bytes)	número de arcos de destino
mp_OriginArcs	long integer pointer	lista de índices de arcos de origem
mp_DestinyArcs	long integer pointer	lista de índices de arcos de destino
m_NumCondition	long integer (4bytes)	número de condições
mp_ConditionList	long integer pointer	lista de índices para as condições
m_NumGates	long integer (4bytes)	número de condições
mp_GatesList	long integer pointer	lista de índices para os gates
m_IsPossibilyFireable	BOOL	flag de condições satisfeitas
m_IsEnabled	BOOL	flag de gates satisfeitos

m_IsFireable	BOOL	flag de restrições adicionais satisfeitas
m_IsFlaggedForNextStateChange	BOOL	flag de mudança de estado
m_HasCondition	BOOL	flag de presença de restrições adicionais
m_IsASolvedConflict	BOOL	flag de resolução de conflitos
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: tipo, nome, número de arcos de origem, lista de arcos de origem, número de arcos de destino, lista de arcos de destino, número de gates, lista de gates, número de condições, lista de condições, constante de tempo	void	inicializa a transição com os atributos apropriados e atualiza as flags internas
GetLabel parâmetros: void	string	retorna o nome da transição
IncrementTimeCount parâmetros: delta T	void	Incrementa a contagem de tempo atual
ResetTimeCount parâmetros: nenhum	void	Zera a contagem de tempo
CheckTime parâmetros: nenhum	BOOL	Retorna se a temporização já foi satisfeita
StartTimer parâmetros: nenhum	void	Inicia a contagem de tempo
UpdateTimer parâmetros: nenhum	void	atualiza a contagem de tempo
GetNumOriginArcs parâmetros: nenhum	long integer (4bytes)	Retorna o número de arcos de origem que chegam na transição
GetNumDestinyArcs parâmetros: nenhum	long integer (4bytes)	Retorna o número de arcos de destino que saem da transição
GetOriginArcsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de arcos de origem que chegam na transição
GetDestinyArcsIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de arcos de destino que saem da transição
GetNumGates parâmetros: nenhum	long integer (4bytes)	Retorna o número de gates ligados à transição
GetGatesIndexList parâmetros: nenhum	long integer pointer	Retorna a lista de índices de gates ligados à transição
IsChangeFlagged parâmetros: void	BOOL	retorna o status da flag
SetChangeFlag parâmetros: BOOL	BOOL	atualiza e retorna o status da flag
IsFireable parâmetros: void	BOOL	retorna o status da flag
SetFireableFlag parâmetros: BOOL	BOOL	atualiza e retorna o status da flag
IsEnabled parâmetros: void	BOOL	retorna o status da flag
SetEnabledFlag parâmetros: BOOL	BOOL	atualiza e retorna o status da flag
IsPossibilyFireable parâmetros: void	BOOL	retorna o status da flag
SetPossibiltyFireableFlag parâmetros: BOOL	BOOL	atualiza e retorna o status da flag
IsASolvedConflict parâmetros: void	BOOL	retorna o status da flag

SetConflictFlag parâmetros: BOOL	BOOL	atualiza e retorna o status da flag
CanBeFired parâmetros: void	BOOL	retorna se a transição pode ou não ser disparada na mudança de estado do grafo
UpdateFlags parâmetros: void	void	atualiza as flags da transição
ResetFlags parâmetros: void	void	reinicializa as flags da transição
FireTransition parâmetros: void	void	dispara a transição de estado

6.2.5 - Representação dos Arcos Orientados

Os arcos orientados definidos no E-MFG Script (Capítulo 5) podem ser dos seguintes tipos: FROM_BOX e FROM_TRANSITION, de acordo com a sua origem e podem possuir filtros que realizam a filtragem seletiva dos atributos das marcas.

Dessa forma, a estrutura de uma classe de objetos que implementa essa representação é dada por:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CArc	
Objetivo da Classe	Implementar a representação dos arcos do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Label	string	Nome do arco
mp_Graph	CGraph pointer	ponteiro para o Grafo
m_Origin	long integer (4 bytes)	Índice do Elemento de Origem
m_Destiny	long integer (4 bytes)	Índice do Elemento de Destino
m_Type	short integer	Tipo do arco
m_Filter	CArcFilter	Filtro do Arco (*default* = PASS_ALL)
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: Nome do Arco, Tipo, Grafo, Origem e Destino	void	inicializa o arco com os atributos apropriados
SetFilter parâmetros: filtro do arco	void	Inicializa o filtro do arco
ApplyFilter parâmetros: CMark	CMark	Aplica o filtro do arco na marca passada como parâmetro e retorna a marca com os atributos filtrados
GetOrigin parâmetros: void	long integer	Retorna o índice do elemento de origem
GetDestiny parâmetros: void	long integer	Retorna o índice do elemento de destino
SetOrigin parâmetros: índice do elemento	void	Atualiza a origem do arco
SetDestiny parâmetros: índice do elemento	void	Atualiza o destino do arco
GetLabel parâmetros: void	string	retorna o nome do arco
GetType	short integer	retorna o tipo do arco

parâmetros: void		

6.3 - Representação do Grafo

A fim de se representar o grafo é necessário agrupar os elementos que o formam através de listas de elementos do mesmo tipo. Logo, será necessária a codificação de classes de objetos que implementem essas listas para cada tipo de objeto. Para permitir a alocação dinâmica de memória, serão utilizadas listas simplesmente ligadas como a descrita no modelo da figura 6.3-1.

Para se evitar confusão quanto a nomenclatura, foi definido que uma lista para uma determinada classe terá o mesmo nome que essa classe de objetos concatenado com a palavra *'Array'*, portanto uma lista de marcas receberá o nome de CMarkArray (pois a marca é implementada pela classe CMark).

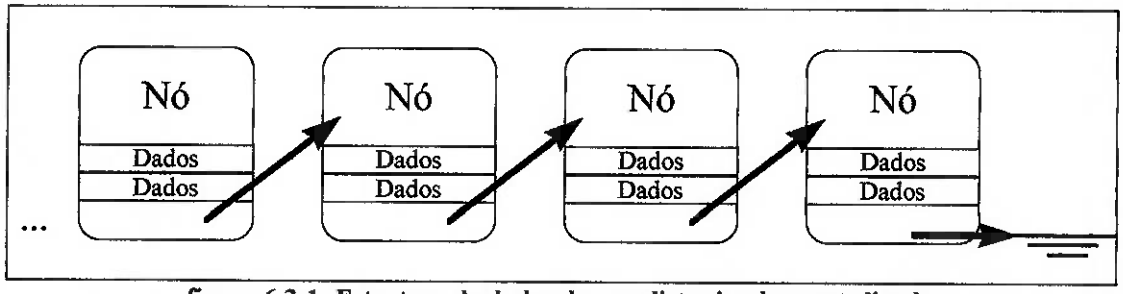


figura 6.3-1: Estrutura de dados de uma lista simplesmente ligada

Adotando esse agrupamento de objetos, tem-se o grafo representado por um conjunto de listas para os seus elementos estruturais(boxes, transições, arcos e portas), uma lista contendo os *'templates'* dos atributos de marca, uma lista de atribuições e uma lista de condições booleanas às quais os elementos estruturais se referenciam.

Portanto, a representação do grafo será realizada pela seguinte classe de objeto:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CGraph	
Objetivo da Classe	Implementar a representação do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
m_Title	string	Título do Grafo
m_Descr	string	Descrição do Grafo
mp_BoxesTable	CBoxArray pointer	Lista de boxes
mp_TransitionsTable	CTransitionArray pointer	Lista de Transições
mp_ArcsTable	CArcArray pointer	Lista de Arcos
mp_GatesTable	CGateArray pointer	Lista de Gates
mp_AttribTemplTable	CMarkAttribArray pointer	Lista de *'Templates'* de Atributos de Marca

mp_AttributionsTable	CAttributionArray pointer	Lista de Atribuições
mp_ConditionsTable	CConditionArray pointer	Lista de Condições
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: Título e Descrição	void	inicializa o grafo
GetBoxes parâmetros: void	CBoxArray pointer	retorna o vetor de boxes
GetTransitions parâmetros: void	CTransitionArray pointer	retorna o vetor de transições
GetArcs parâmetros: void	CArcArray pointer	retorna o vetor de arcos
GetGates parâmetros: void	CGateArray pointer	retorna o vetor de gates
GetAttribTemplates parâmetros: void	CMarkAttribArray pointer	retorna o vetor de templates de atributos de marca
GetAttributions parâmetros: void	CAttributionArray pointer	retorna o vetor de atribuições
GetConditions parâmetros: void	CConditionArray pointer	retorna o vetor de condições
GetBoxesConnectionDump parâmetros: CLinesArray pointer to DumpLineTable	void	Faz o dump das ligações entre os boxes e as transições
GetBoxesPropertiesDump parâmetros: ClinesArray pointer to DumpLineTable	void	Faz o dump das propriedades dos boxes e suas marcas

6.4 - Interpretador

O interpretador é o objeto responsável por tranformar a representação textual *E-MFG Script* (capítulo 5) na representação interna de objetos definida na seção 6.2.

Para especificá-lo é necessário antes definir quais são os passos para a execução dessa tarefa. Em termos gerais é preciso:

1. verificar a inclusão de arquivos e montar uma tabela de inclusão, checando para não haver recursão que causaria loops infinitos;
2. para cada arquivo da lista de includes, ler o arquivo texto, eliminando os espaços em branco, separando os comandos e ignorando os comentários. O resultado é guardado numa tabela de strings;
3. processar a tabela do arquivo, avaliando cada seção conforme a sintaxe apresentada no capítulo 5. Para cada seção uma nova tabela de strings é construída com a adição do prefixo especificado na tag <INCLUDE> do arquivo específico, correspondendo aos parâmetros de constução de cada objeto. Ao final dessa etapa, há uma tabela literal para cada um desses

elementos: atributos, atribuições, condicionais, boxes, transições, arcos, gates e marcas iniciais do grafo;

4. para cada tabela literal construída na etapa anterior executa-se uma referência cruzada que transforma os nomes dos elementos (string) em índices (inteiros). Essa referência inclui a checagem de não duplicação de nomes entre os diversos elementos.
5. caso ainda haja arquivos a processar na tabela de includes, retornar ao passo 2;
6. com base nos índices obtidos na etapa anterior os objetos são criados e adicionados nas listas internas do CGraph.

Observe que o interpretador é responsável por manter a consistência do grafo representado pelo arquivo de entrada em *E-MFG Script*. O interpretador deve detectar problemas como:

- erros de digitação, ou ausência de seções obrigatórias;
- referências a elementos não declarados em suas respectivas seções.

As tabelas literais são na verdade matrizes bi-dimensionais de strings, ou em outras palavras arrays de arrays de string. Cada linha corresponde a um elemento E-MFG declarado no arquivo de entrada.

Dessa forma, pode-se definir a estrutura do objeto interpretador:

Definição de Classe de Objeto		
Nome da Classe de Objetos	Cinterpreter	
Objetivo da Classe	Implementar a representação do grafo	
Atributos da classe	**Tipo de dados**	**Descrição**
mp_Graph	CGraph pointer	ponteiro para o grafo a ser construído
m_FileTable	array de string	tabela do arquivo de entrada
m_AttribTable	matriz de string	tabela literal do template de atributos
m_AttributionTable	matriz de string	tabela literal das atribuições
m_CondTable	matriz de string	tabela literal das condições
m_BoxesTable	matriz de string	tabela literal das boxes
m_TransitionsTable	matriz de string	tabela literal das transições
m_ArcsTable	matriz de string	tabela literal dos arcos
m_GatesTable	matriz de string	tabela literal dos gates
m_MarksTable	matriz de string	tabela literal das marcas iniciais
title	string	título do grafo

description	string	descrição do grafo
Métodos da classe	**Tipo de retorno**	**Descrição**
Compile parâmetros: nome do arquivo e ponteiro para o grafo	BOOL	gerencia a ordem de chamada das demais funções
Read parâmetros: nome do arquivo	BOOL	lê o arquivo de entrada e cria a tabela literal dos comandos
BuildTables parâmetros: void	BOOL	constrói as tabelas literais
CrossReference parâmetros: void	BOOL	realiza a referência cruzada e cria os objetos

Essa é a interface externa da classe CInterpreter, que apresenta uma série de outras funções internas responsáveis pela leitura das linhas de comando, procura em tabelas literais, interpretação e montagem de sentenças condicionais.

Dentre essas funções internas é interessante focar a atenção naquelas que interpretam e constroem as sentenças condicionais, uma vez que conforme especificado no capítulo 5, essas apresentam uma sintaxe distante da representação interna em objeto.

A função chamada BuildCond() recebe como parâmetros uma referência ao condicional a ser montado e a string contendo a expressão definida no *EMFG Script*. Essa função pode ser dividida em duas partes:

- inicialmente é determinado se a sentença condicional é simples ou composta;
- caso seja simples, aciona-se a primeira parte:
  - executa-se a divisão dos ítens *L-Value*, *Operator* e *R-Value;*
  - realiza-se a referência cruzada desses ítens e a montagem do condicional;
  - a função retorna.
- caso seja composta, aciona a segunda parte:
  - extrai-se uma sub-string correspondente a sentença condicional dentro delimitada pelo parênteses de nível mais alto;
  - chama-se recursivamente a própria BuildCond() com a sub-string como parâmetro;
  - quando o BuildCond() acima retorna o condicional criado é adicionado numa lista local;

- como se trata de uma sentença composta, procura-se pelo operador lógico e adiciona-se este à outra lista local;
- repete-se o processo acima para os todas as sub-strings dentro do nível local de parênteses;
- ao chegar aqui, tem-se uma lista de condicionais com *n* elementos e uma lista de operadores com *(n-1)* elementos;
- monta-se a árvore correspondente às listas acima, uma vez que todos esses condicionais pertencem ao mesmo nível de parênteses;
- a função retorna.

Note que no caso de uma sentença simples, somente a primeira parte da função será acionada. Caso seja composta, sucessivas chamadas recursivas são feitas até que se encontrem os condicionais simples. A própria estrutura da heap de execução é responsável por representar a ordem de criação da árvore binária.

Na verdade, o processo executado pela BuildCond() é um pouco mais complicado. Isso se deve a alocação de memória, especialmente por causa da recursão. Ao final da cada chamada de função, seus membros locais são destruídos e a memória associada é perdida.

Essa dificuldade pode ser ultrapassada com o uso cuidadoso dos operadores de cópia e em especial do mecanismo de referência disponível no C++.

6.5 - Gerenciador de Marcas

O Gerenciador de Marcas consiste em um objeto que observa o estado atual do grafo e executa os passos necessários para a evolução para o próximo estado. No entanto, o Gerenciador de Marcas não têm como avaliar as condições do meio externo ao grafo que ele controla, cabendo essa tarefa a um segundo objeto chamado de Controlador de I/O, de acordo com o modelo conceitual do controlador descrito na figura 4.3-1. Logo, o escopo do trabalho do Gerenciador de Marcas se restringe a uma única evolução de estado, sendo necessário que o nível hierárquico superior no modelo de dados do controlador (ou seja a aplicação - vide figura 6-1), faça as chamadas de função adequadas para a atualização dos estados dos sinais externos ao controlador.

Dessa forma, o objeto Gerenciador de Marcas deve apenas possuir um ponteiro para a representação do grafo e uma seqüência de métodos que executem o

ciclo de disparo. Outro aspecto importante é que a avaliação das condições de disparo de transição é realizada internamente pela própria transição. O Gerenciador de Marcas apenas requisita a atualização do *'status'* da transição sem precisar conhecer seu tipo ou suas restrições, que são tratados localmente. Assim, o papel fundamental do Gerenciador de Marcas é a resolução de conflitos (onde se necessita de uma visão global do grafo) e a requisição de disparo das transições ainda disparáveis após a resolução do conflito.

O Gerenciador de Marcas deve resolver os conflitos não arbitrados no grafo. O algoritmo de resolução de conflitos é o seguinte:

1 - Enquanto não foi alcançado o final da lista, percorra a lista de pré-condições;

2 - Se o box é um box conflito de saída identifique as transições em conflito;

2.1 - Se não existirem transições marcadas como vencedoras de conflito;

2.1.1 - para cada transição em conflito é atribuída uma chance igual de disparo;

2.1.2 - sorteia-se uma transição vencedora do conflito e a marca como tal;

2.1.3 - desabilita as demais transições em conflito;

2.1.4 - retorna ao passo 1;

2.2 - Se existirem transições vencedoras de conflito a primeira transição vencedora da lista é marcada como vencedora e as demais desabilitadas, retornado em seguida ao passo 1;

3 - Se o box não é um box conflito retorne ao passo 1

4 - Enquanto não foi alcançado o final da lista, percorra a lista de pós-condições;

5 - Se o box é um box conflito de entrada identifique as transições em conflito;

5.1 - Se não existirem transições marcadas como vencedoras de conflito;

5.1.1 - para cada transição em conflito é atribuída uma chance igual de disparo;

5.1.2 - sorteia-se uma transição vencedora do conflito e a marca como tal;

5.1.3 - desabilita as demais transições em conflito;

5.1.4 - retorna ao passo 4;

5.2 - Se existirem transições vencedoras de conflito a primeira transição vencedora da lista é marcada como vencedora e as demais desabilitadas, retornado em seguida ao passo 4;

6 - Se o box não é um box conflito retorne ao passo 4;

Tendo esses aspectos do modelo do Gerenciador de Marcas em vista, tem-se a seguinte estrutura para o objeto:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CMarkManager	
Objetivo da Classe	Implementar a representação do Gerenciador de Marcas	
Atributos da classe	**Tipo de dados**	**Descrição**
mp_Graph	CGraph pointer	ponteiro para o grafo
m_ActualTime	CTime	TimeStamp atual
m_GraphCycleTime	CTimeSpan	Duração do último ciclo de controle
m_FileLog	BOOL	Flag para armazenar ou não o disparo de transições em disco
Métodos da classe	**Tipo de retorno**	**Descrição**
Create parâmetros: ponteiro para o Grafo	void	inicializa o Gerenciador de Marcas
SetupTransitions parâmetros: void	void	percorre a lista de transições e atualiza as flags de dinâmica de disparo
FireTransitions parâmetros: ponteiro para o arquivo de log	void	dispara as transições marcadas como disparáveis
LogGraph parâmetros: ponteiro para o arquivo de log	void	realiza um dump em arquivo texto do grafo
SetFileLog parâmetros: valor booleano	void	atualiza a flag booleana para logar ou não o disparo de transições
IsLogging parâmetros: void	BOOL	retorna o valor da flag booleana para logar ou não o disparo de transições
GetTimeStamp parâmetros: void	string	retorna a TimeStamp Atual
GetTimeSpanStamp parâmetros: void	string	retorna o tempo transcorrido desde o último ciclo de controle
GetLastFireTime parâmetros: void	CTime	retorna a TimeStamp Atual
GetLastCycleSpan parâmetros: void	string	retorna o tempo transcorrido desde o último ciclo de controle
SeekAndSolveConflicts parâmetros: void	void	resolve as transições em conflito
SeekAndSolveInputConflicts parâmetros: void	void	resolve os conflitos de entrada
SeekAndSolveOutputConflicts parâmetros: void	void	resolve os conflitos de saída

LogFiredTransition parâmetros: ponteiro para o arquivo e Nome da transição	void	faz o log em um arquivo texto d as transições disparadas

6.6 - Controlador de I/O

O controlador de I/O é responsável por realizar a comunicação com o mundo externo, ou seja, ler os sinais dos *'gates'* de entrada e disponibilizar os sinais dos *'gates'* de saída para outras aplicações.

Para que o controlador de I/O seja o mais genérico possível, sua estrutura deverá permitir a inserção de módulos que permitam a comunicação através de vários protocolos.

6.6.1 – Comunicação DDE

Dentro da proposta de controlador adotada nesse trabalho, especificou-se a implementação da comunicação através de DDE, que é padrão para a plataforma Windows.

No compilador VC++ 4.0 toda comunicação DDE é feita através de uma biblioteca chamada DDEML (DDE Management Library). Para funcionar como servidor, fica a cargo de cada aplicação escrever uma função de *'callback'* que trate a requisição de dados de acordo com a figura 6.6.1-1.

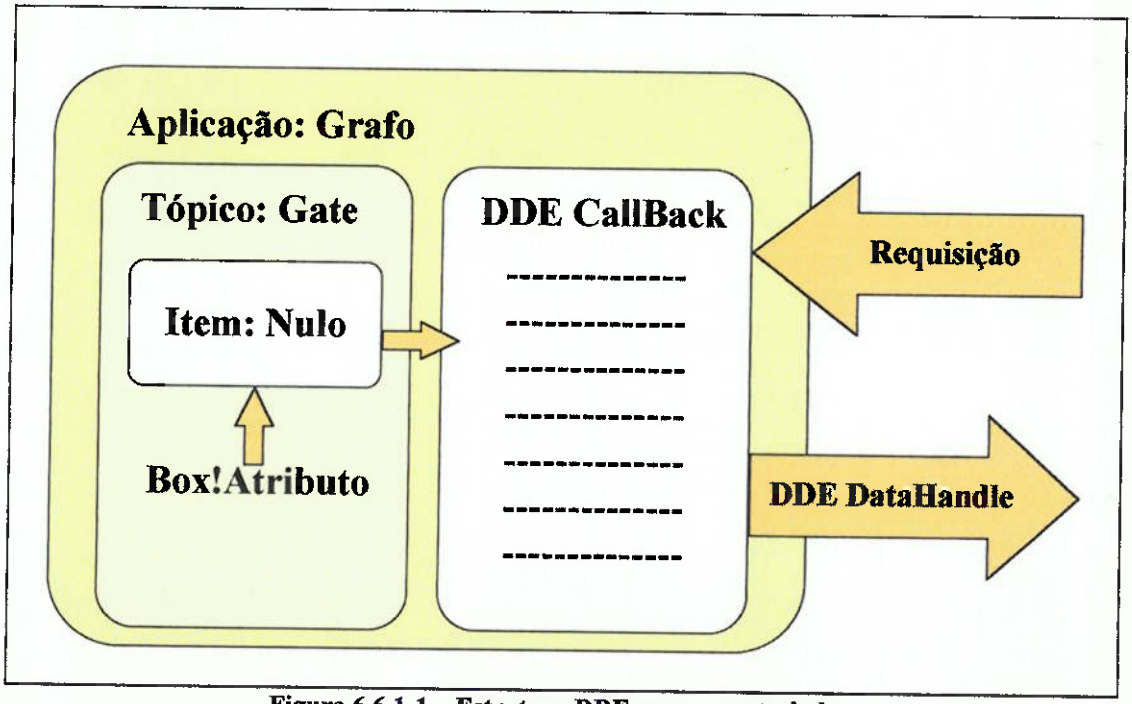


Figura 6.6.1-1 – Estrutura DDE para o controlador

A função DDE Callback associada ao controlador tem como missão disponibilizar os dados para outras aplicações interessadas em receber sinais ou

escrever sinais do grafo. No caso da estrutura de gates definida no capítulo 5, a DDE Callback está diretamente ligada aos gates passivos.

A biblioteca DDEML.H define uma mensagem para cada uma dessas operações:

- XTYP_REQUEST, para receber dados;
- XTYP_POKE, para escrever dados.

Ao receber uma mensagem XTYP_REQUEST, o procedimento na DDE Callback é o seguinte:

- percorre a lista de gates, para determinar se existe um gate com o nome do ítem;
- caso exista, checa se esse gate é do tipo passivo;
- caso seja, recupera o valor do gate;
- realiza a transformação para um DDE Data Handle de acordo com o tipo do gate.

Ao receber uma mensagem XTYP_POKE, o procedimento na DDE Callback é o seguinte:

- percorre a lista de gates, para determinar se existe um gate com o nome do ítem;
- caso exista, checa se esse gate é do tipo passivo;
- caso seja, interpreta o DDE Data Handle de acordo com o tipo do gate;
- atualiza o valor do gate.

Por outro lado, o controlador também deve ler dados de outras aplicações, funcionando como um cliente. Nesse caso o compilador não impõe nenhuma condição quanto ao nome e estrutura das funções (essas são as funções associadas aos gates ativos).

Para implementar o cliente DDE é uma boa opção encapsular as funções do DDEML dentro de um objeto, tomando como parâmetro básico uma string do tipo: "Aplicação!Tópico!Item". O objeto se encarrega de dividir essa string, estabelecer a comunicação, recuperar o DDE DATA HANDLE e realizar a conversão para o tipo correspondente. Observe que o único formato de *'clipboard'* utilizado será CF_TEXT (que é o formato mais utilizado), mas isto fica transparente para o usuário já que todas as conversões de dados são feitas internamente ao objeto.

Definição de Classe de Objeto		
Nome da Classe de Objetos	CDDESocket	
Objetivo da Classe	Encapsular o acesso a DDEML.H	
Atributos da classe	**Tipo de dados**	**Descrição**
m_title	string	nome do grafo

m_AppName	string	nome da presente aplicação
m_App	string	nome da aplicação servidora
m_Topic	string	nome do tópico no servidor
m_Item	string	nome do ítem no servidor
m_Registered	boolean	flag de registro
m_Connected	boolean	flag de conexão
m_timeout	short integer (2 bytes)	tempo para TIMEOUT em ms
m_error	DWORD	ID do erro DDE
m_idInst	DWORD	ID da instância DDE
m_hConv	HCONV	handle para a conversação
Métodos da classe	**Tipo de retorno**	**Descrição**
RegisterApp parâmetros: nome do grafo	void	registra o grafo como uma aplicação DDE
GetString parâmetros: string de comunicação	string	retorna a string correspondente ao DDE DATA HANDLE obtido na comunicação
GetInteger parâmetros: string de comunicação	integer	retorna o inteiro correspondente ao DDE DATA HANDLE obtido na comunicação
GetBOOL parâmetros: string de comunicação	BOOL	retorna o BOOL correspondente ao DDE DATA HANDLE obtido na comunicação
PutString parâmetros: (string de comunicação, valor string)	void	coloca o valor string na aplicação, tópico e ítem determinados pela string de comunicação
PutInteger parâmetros: (string de comunicação, valor inteiro)	void	coloca o valor inteiro na aplicação, tópico e ítem determinados pela string de comunicação
PutBOOL parâmetros: (string de comunicação, booleano)	void	coloca o valor bool na aplicação, tópico e ítem determinados pela string de comunicação
SetTimeOut parâmetros: (valor inteiro)	void	determina qual o tempo de TIMEOUT em ms

A estrutura do controlador de I/O deve incluir uma instância do objeto CDDESocket e uma função DDE Callback global para que o controlador E-MFG possa funcionar tanto como um servidor quanto como cliente DDE.

6.6.2 – Ciclo de Update

O ciclo de *'update'* executa a leitura dos sinais externos e a atualiza os valores internos dos *'gates'* de saída.

Esse ciclo é feito percorrendo toda a lista de *'gates'* do grafo. Para os *'gates'* internos e de saída, os valores internos de retorno do CGates são alterados. Para os *'gates'* externos, as requisições de dados são enviadas para a aplicações servidoras.

No momento que todos os *'gates'* se encontram atualizados, guarda-se a diferença entre a último *'update'* (Δt do último ciclo) e o valor do momento do atual *'update'*.

6.6.3 – Estrutura Interna

A definição do controlador de I/O fica da seguinte forma:

Definição de Classe de Objeto		
Nome da Classe de Objetos	CCommunicator	
Objetivo da Classe	Realizar a comunicação com o meio-externo	
Atributos da classe	**Tipo de dados**	**Descrição**
mp_Graph	CGraph pointer	ponteiro para o grafo
m_DDESock	CDDESocket	objeto encapsulador DDE
m_Last	CTime	valor do último momento de update dos dados
m_Comm	short integer (2 bytes)	valor do delta-tempo em segundos para o último ciclo de comunicação
m_Control	short integer (2 bytes)	valor do delta-tempo em segundos para o último ciclo de controle
m_LogErrors	BOOL	flag de log de erros
m_LogFile	string	nome do arquivo de log
m_LogFileSet	BOOL	flag do arquivo de log
Métodos da classe	**Tipo de retorno**	**Descrição**
UpdateAllGates()	BOOL	atualiza todos os valores dos gates, tanto internos quanto externos
GetLastCommCycleDelta()	long integer (4 bytes)	retorna os segundos de duração do último ciclo de comunicação
GetLastControlCycleDelta ()	long integer (4 bytes)	retorna o segundos de duração do último ciclo de controle
GetLastUpdate()	CTime	retorna o CTime correspondente ao último update
Log()	void	loga os erros de comunicação

Um comentário final sobre a estrutura definida fica por conta do tratamento dado aos erros de comunicação. Toda vez que o controlador não consegue acessar a aplicação server, ou quando o tempo de TIMEOUT expira, uma mensagem de erro é gerada. São possíveis duas ações:

- uma mensagem é apresentada ao usuário, aguardando confirmação para prosseguir;
- a mensagem de erro é gravada num arquivo para análise posterior.

Por motivos óbvios, a primeira ação não se presta quando o controlador está em operando em ciclo contínuo. Porém, essa opção pode ser interessante durante a realização de testes de comunicação.

O flag m_LogErrors indica se os erros serão gravados num arquivo texto. O nome do arquivo é o mesmo do grafo, seguido da extensão DDE (i.e. grafo.dde). O formato da mensagem de erro é dado por:

[*dia/mês/ano hora:minuto:segundo*] - DDE Comunication Error - *String de Erro*

Assim, pode-se analisar os erros de comunicação *à posteriori*, sem interromper a execução do ciclo de controle.

Cabe esclarecer que apesar da comunicação DDE prever três parâmetros, a estrutura definida só necessita de dois parâmetros para identificar unicamente um gate pertencente à outro grafo. Assim foi usada a palavra 'data' no lugar do tópico, deixando assim aberta a possibilidade para extensões do objeto controlador de I/O utilizando o próprio DDE. Seria o caso, por exemplo, de um gate para uma interface gráfica.

A estrutura do controlador de I/O permite a extensão dos métodos de comunicação, uma vez que basta incluir membros nos moldes do CDDESocket, ou seja, encapsulamentos de outros métodos de comunicação (e.g. TCP/IP).

Observe que a comunicação com uma interface gráfica ainda não está definida, mas cabe mencionar que uma possibilidade seria através de um *'loop'* DDE Advise. Isso implicaria numa alteração na rotina DDE Callback para permitir este tipo de serviço. Também seria necessário definir quais dados e em que ordem estariam disponíveis para o programa de interface gráfica.

6.7 - Interface

O formato final da interface não está ainda definido, mas pode-se especificar algumas características desejáveis, tais como permitir:

- a visualização das tabelas literais de elementos;
- a visualização textual da dinâmica do grafo;
- a execução passo-a-passo do ciclo de controle;
- o histórico das transições disparadas e dos sinais externos de dados.

A princípio, o controlador rodará numa instância para cada grafo.

Já existe uma interface definida para Debug mostrada na figura 6.7-1:

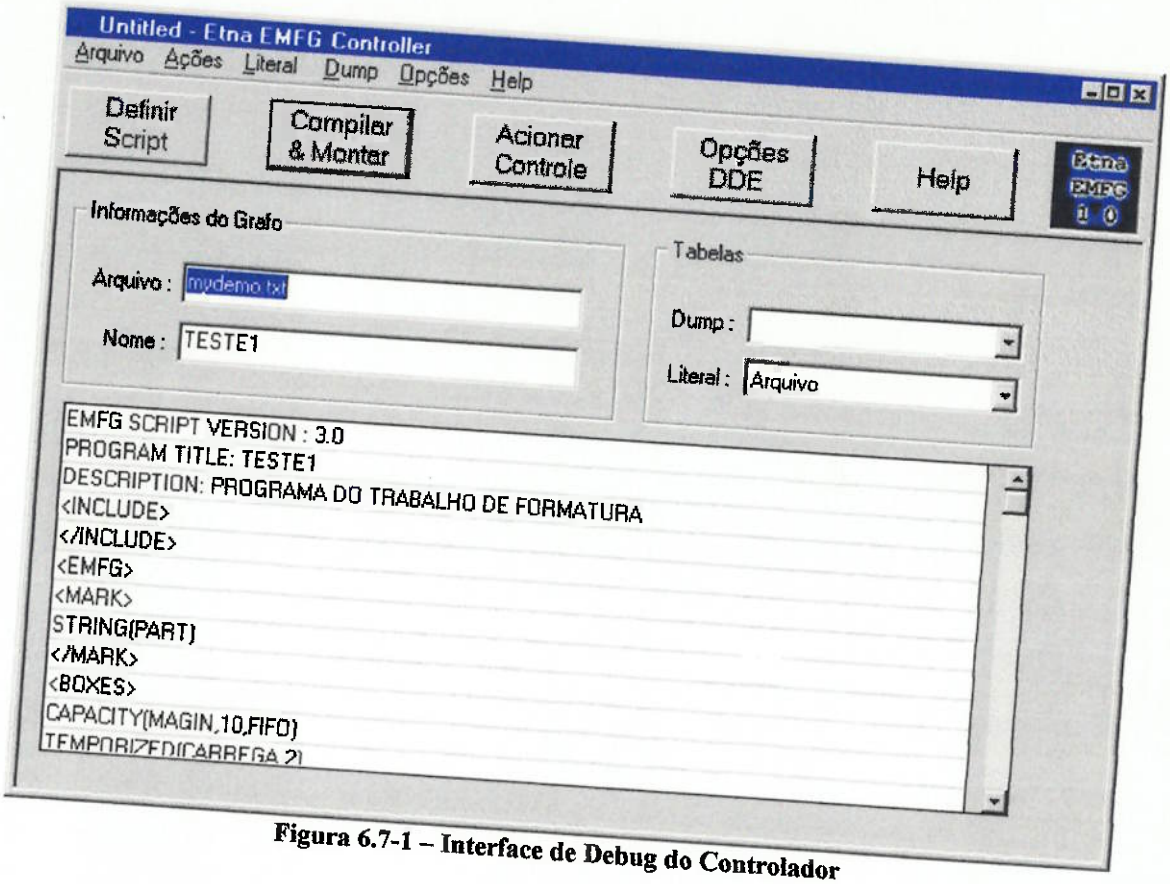


Figura 6.7-1 – Interface de Debug do Controlador

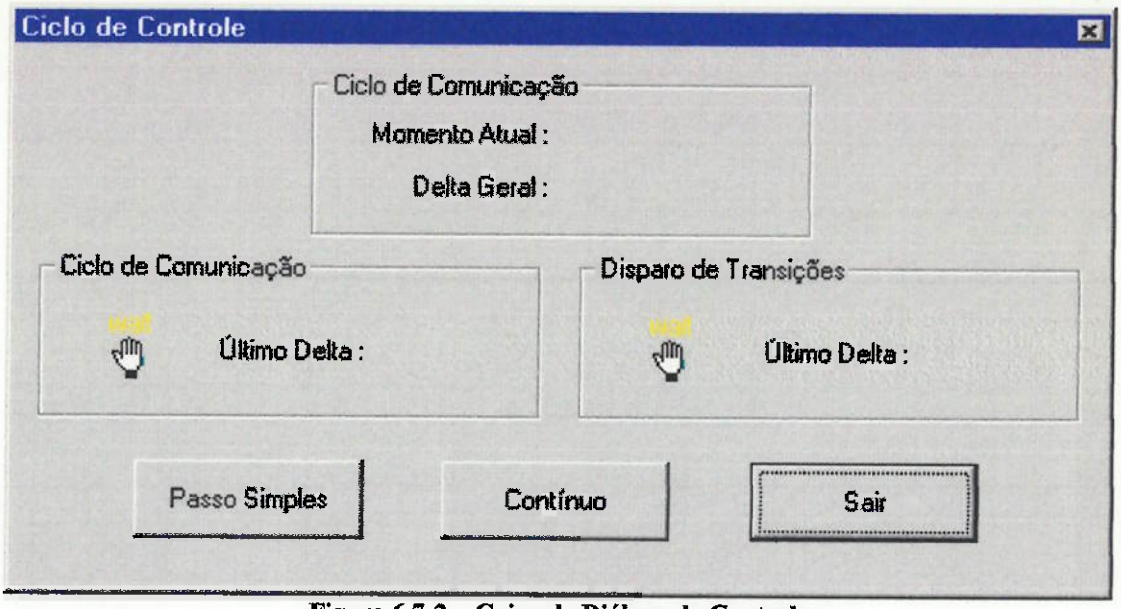


Figura 6.7-2 – Caixa de Diálogo de Controle

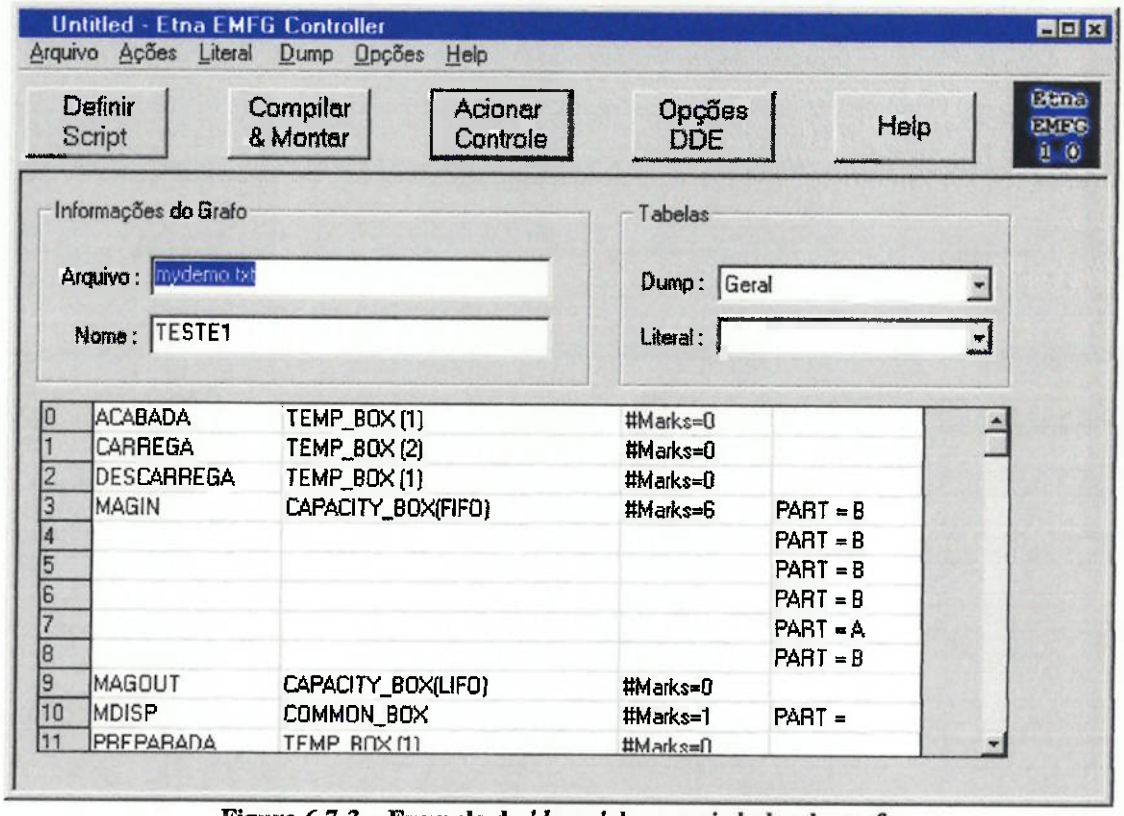


Figura 6.7-3 – Exemplo de *'dump'* de propriedades do grafo

Maiores detalhes quanto à implementação da interface pode ser vistos no Apêndice II.

7 - Estrutura do Controlador

Conforme descrito anteriormente, o programa do controlador é constituído por uma estrutura hierárquica de dados (figura 6-1), onde no topo da hierarquia encontram-se os objetos responsáveis pelo comportamento macroscópico do controlador, e é com esses objetos que o nível da aplicação deve interagir a fim de realizar a ponte entre o usuário e o programa de controle descrito na representação interna de dados.

7.1 - Ciclo Geral do Programa

A figura 7.1-1 apresenta um grafo E-MFG que descreve o comportamento macroscópico esperado do nível do programa do controlador identificado como aplicação na figura 6.1. Este é o comportamento para o controlador funcionando em modo automático (o modo *'single-step'* exige que o usuário autorize cada ciclo de controle).

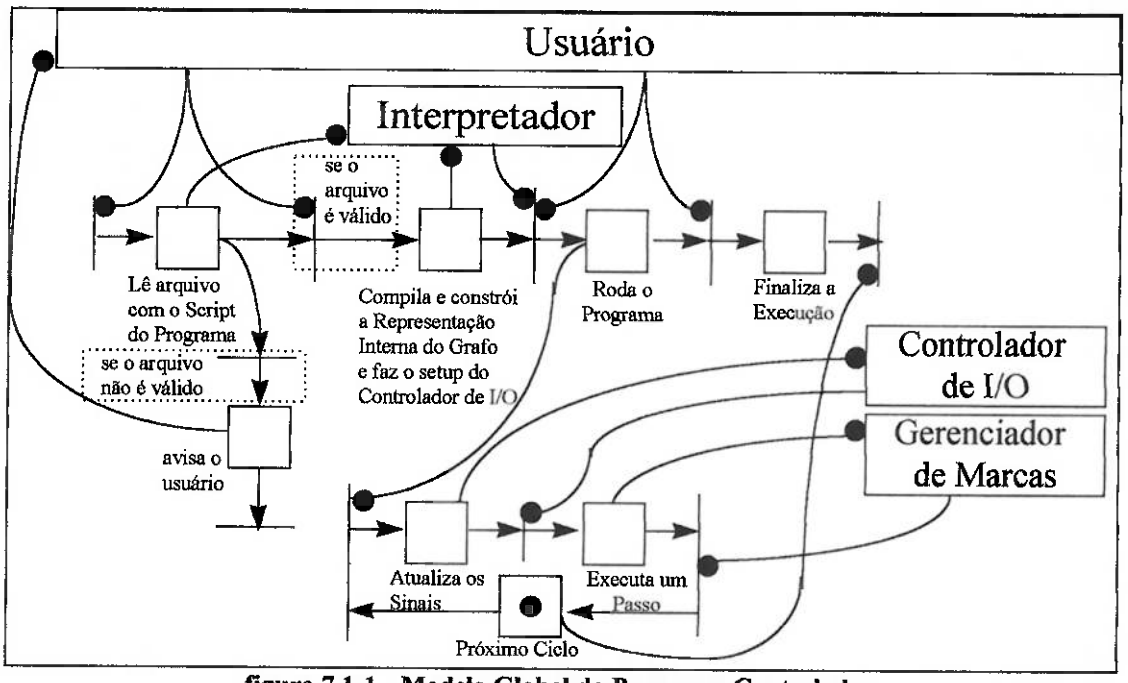


figura 7.1-1 - Modelo Global do Programa Controlador

A fim de se executar um programa de controle, o usuário deve dar um comando de leitura do programa de controle e autorizar a sua compilação. Se o arquivo do programa for um arquivo inválido o programa deve exibir uma mensagem de erro, caso contrário uma vez autorizado pelo usuário o programa deve construir a

representação interna do grafo (através do objeto Interpretador). Uma vez construída a representação interna o usuário deve autorizar a execução do programa. Uma vez autorizada a execução, a aplicação fará uma chamada do Controlador de I/O para que este atualize os sinais da via de dados e este ao terminar a atualização autoriza a aplicação a chamar o Gerenciador de Marcas para a execução da evolução para o próximo estado do programa de controle. Após a evolução de estado a aplicação poderá novamente chamar o Controlador de I/O e repetir o ciclo, até que o usuário ordene a finalização do programa. Neste caso o ciclo de controle em andamento será executado até o final então o programa será finalizado.

7.2 - Ciclo de Controle

Durante o ciclo de controle, a aplicação é responsável por chamar na ordem adequada os métodos do Controlador de I/O e do Gerenciador de Marcas, de modo que a evolução dinâmica do grafo se dê corretamente. De maneira resumida, a figura 7.2-1 ilustra o ciclo de controle com as chamadas de métodos desses objetos no domínio da aplicação e um exemplo da chamada de métodos no domínio do Gerenciador de Marcas (durante a execução da função FireTransitions). Deve-se notar que para a aplicação a estrutura de dados de representação interna do grafo é transparente, ou seja apenas os métodos do Gerenciador de Marcas e do Controlador de I/O que podem alterar os elementos do grafo. Da mesma maneira, apenas as transições do grafo é que podem efetuar a dinâmica de disparo e podem interagir com os outros objetos do grafo (retirando a marcação das pré-condições, aplicando os filtros dos arcos e marcando as pós condições). Garante-se, dessa maneira, a manutenção da hierarquia de objetos proposta.

É importante notar que a hierarquia também é mantida pelos elementos de I/O do grafo, ou os *'gates'*, que a cada ciclo possuem apenas um valor armazenado (booleano ou não). Como os *'gates'* não apresentam métodos que os permitam se auto-atualizar, eles podem ser tratados indistintamente (a menos da sua função: inibidor, habilitador ou transporte de dados) pelos outros elementos do grafo. Neste modelo, a atualização de todos os gates fica a cargo do controlador de I/O, o que permite que futuramente novos métodos de I/O sejam implementados sem se alterar a estrutura do grafo.

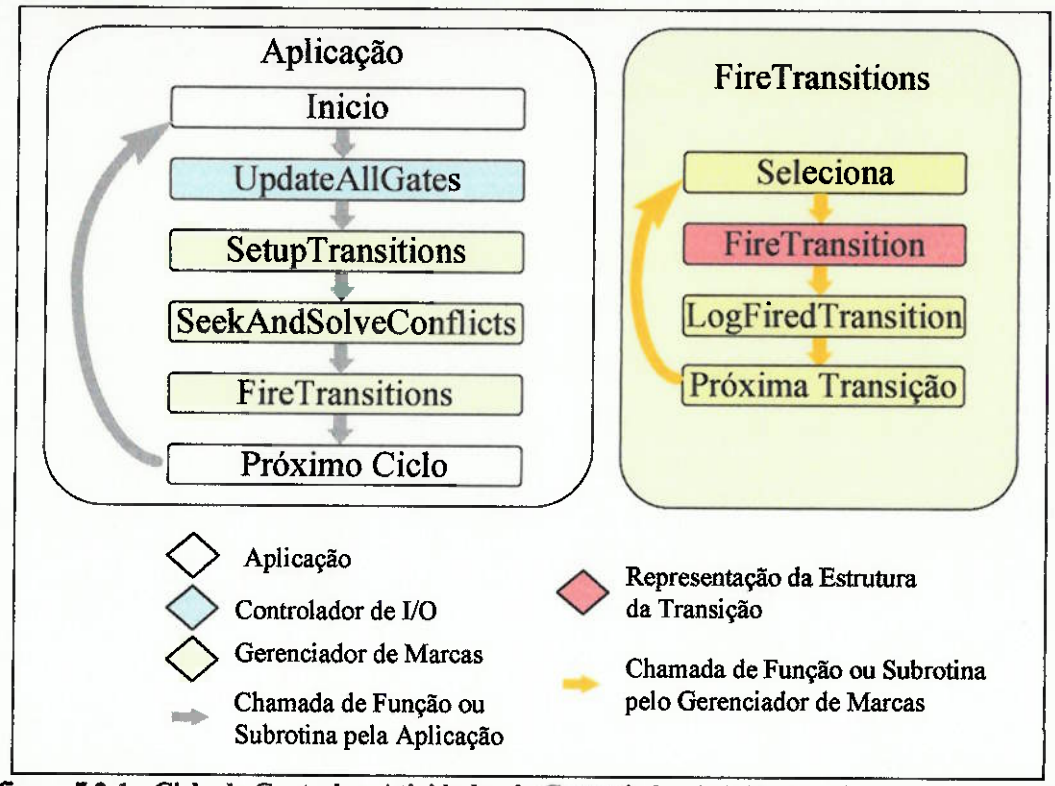


figura 7.2-1 - Ciclo de Controle e Atividades do Gerenciador de Marcas e Controlador de I/O

Torna-se claro, observando a figura, que o Gerenciador de marcas e o Controlador de I/O só possuem a visão do estado atual do programa de controle, ficando a cargo da aplicação controlar se o programa é executado em um loop contínuo ou passo a passo. Logo, esse tipo de controle fica mais próximo da camada de software que realiza a interface com o usuário, não sendo relevante à estrutura de dados que representa o grafo ou executa a sua dinâmica. Este exemplo ilustra o encapsulamento de dados que se pretende obter na codificação do controlador, onde cada objeto implementado possui métodos que estão dentro do seu domínio de dados (ou nível hierárquico) e pode chamar métodos do nível imediatamente inferior. A única exceção para esse modelo é dados é o objeto Interpretador que pode chamar as funções de criação de todos os tipos de elementos do grafo, bem como acessar as listas de elementos do objeto grafo, além de sua estrutura de dados interna (figura 6.1).

8 - Seqüências de Testes

A fim de se evitar uma maior complexidade para a realização dos testes, a seqüência de testes foi realizada de forma a validar o funcionamento de cada módulo funcional do controlador separadamente sempre que possível. A abordagem adotada para a realização dos testes foi a seguinte:

a) Validação da seqüência de chamada de funções e métodos de cada objeto.

b) Validação da construção das estruturas de dados auxiliares (e.g. tabelas literais utilizadas para o primeiro nível de compilação do programa de controle)

c) Validação da construção dos objetos e seus relacionamentos (e. g. tabelas de *'dump'* de conexões e propriedades dos elementos.

d) Validação do algoritmo de gerenciamento de marcas (evolução de estado), através do aumento contínuo da complexidade dos grafos (utilização de elementos estruturais mais sofisticados).

e) Validação do módulo de comunicação (Controlador de I/O)

f) Validação do passo de controle (seqüência de ações necessárias à evolução de estado para a integração do Controlador de I/O e Gerenciador de Marcas)

h) Validação do modo de controle em *'loop'* contínuo.

Alguns dos testes realizados encontram-se no Apêndice I, este conjunto de testes demonstra a filosofia de testes adotada (de complexidade crescente). Não foram incluídos todos os testes realizados neste apêndice para se evitar a redundância, pois foram realizados testes ao longo de todo o desenvolvimento, utilizando diferentes versões do *E-MFG Script* com diversos grafos simples e complexos que agregariam pouco valor a este documento. Devido a engenharia de software adotada, a arquitetura orientada a objetos utilizada no desenvolvimento do programa permitiu a validação progressiva da funcionalidade, uma vez que o aumento de complexidade representa apenas a adição de um método novo associado a um novo valor de uma característica do objeto e o encapsulamento de dados inerente a esta arquitetura de desenvolvimento permite isolar os métodos já validados daqueles que estão sendo agregados.

8.1 - Testes de Montagem da Representação Interna

Nestes testes será verificada o funcionamento do módulo interpretador e da estrutura de dados.

Serão utilizados exemplos comuns de grafos, com gradual aumento na complexidade nas conexões e usando maior número de elementos E-MFG.

Nessa fase deseja-se testar:

- a validade do *E-MFG Script* como uma representação dos diferentes tipos de grafo possíveis;
- a capacidade do objeto CInterpreter de localizar falhas na consistência do grafo (nomes duplicados e referências faltosas);
- o gerenciamento da memória, com a liberação após o término do programa de todas as estruturas alocadas;
- a conexão entre os elementos E-MFG.

8.2 - Testes de Dinâmica de Disparo

Nesta fase será testado com mais intensidade o objeto CMarkManager, responsável por gerenciar a dinâmica de disparo das transições. Para tanto, utiliza-se um botão STEP na interface que permite a execução do apenas um ciclo de controle.

Com grafos cujas dinâmicas de disparo sejam conhecidas por simulação manual, testa-se o seguinte:

- a avaliação das sentenças condicionais;
- a realização das atribuições;
- o funcionamento dos gates internos;
- as condições de disparo;
- a resolução de conflitos.

8.3 - Testes de Comunicação

Os testes de comunicação devem validar o funcionamento dos gates externos, tanto de saída quanto entrada. Serão feitos testes entre instâncias do controlador, representando diversos níveis hierárquicos de um sistema.

Também serão feitos testes de comunicação com outros aplicativos, tais como o Excel, ou outras aplicações específicas escritas em Visual Basic ou VC++.

8.4 - Validação Geral

Com o funcionamento do controlador testado nas fases anteriores, a validação geral será feita em conjunto com a simulação de um sistema real.

Para tanto serão utilizados programas de simulação que disponibilizem seus dados via DDE e sejam aptos para controle por E-MFG.

9 - Comentários e Conclusões

Neste trabalho buscou-se especificar uma representação textual para os grafos E-MFG, a fim de se utilizar essa representação textual como linguagem de programação do controlador desenvolvido.

Durante a fase de levantamento dos requisitos da descrição textual do E-MFG e o confronto destes requisitos com as características necessárias ao controle de sistemas flexíveis de produção, foram levantadas as propostas de melhoria na representação do E-MFG para a especificação de sistemas de controle (seção 3.3).

De um modo geral, essas propostas tiveram por objetivo tornar a codificação do algoritmo de controle mais clara e menos susceptível a erros, praticamente não afetando a metodologia proposta inicialmente. No entanto, uma das propostas introduziu um novo elemento estrutural à representação do grafo, os arcos de sinal de entrada (ou arcos de dados), que se aplicam para alterar dinamicamente a estratégia de controle (ou seja uma inscrição em um box transformador, transição, ou porta) a partir de um dado coletado externamente ao grafo. Portanto, os arcos de sinal de entrada possibilitam uma maior integrabilidade, flexibilidade e capacidade de representação do grafo, permitindo a parametrização da estratégia de controle (vide a figura 3.3-1b). Por outro lado, a introdução deste elemento não altera o mecanismo de disparo das transições (continuando válidos os 3 níveis de decisão para o disparo discutidos na seção 2.5), ou seja, não altera as características de evolução dinâmica do grafo preservando a característica de Sistema de Eventos Discretos do E-MFG e suas propriedades (como *'safeness'*, por exemplo).

Em relação ao *E-MFG Script* (Capítulo 5), procurou-se desenvolver a linguagem de forma que as declarações dos elementos do grafo estivessem muito claramente definidas. Esta atitude foi tomada com o intuito de tornar a descrição do grafo o mais mecanizada possível a fim de se facilitar a construção do grafo através de uma futura Interface Gráfica de Programação (que se encontra fora do escopo deste trabalho), pois é evidente que a representação gráfica do sistema é muito mais concisa e fácil de ser manipulada pelo projetista do sistema de controle do que uma representação textual. O *E-MFG Script*, ainda apresenta a característica de agrupar todos os elementos de transporte de dados e sinais de controle internos e externos ao grafo em um único bloco de comandos, sem no entanto perder a representatividade

desses elementos estruturais do E-MFG (e.g. portas habilitadoras internas, arcos de sinal de saída, etc.), conforme a discussão da seção 5.2.8.

A partir da especificação do controlador e do *E-MFG Script*, construiu-se um modelo de dados que visando o maior encapsulamento possível (Capítulo 6), a fim de permitir a reutilização do código gerado em outras aplicações, servindo o controlador como o núcleo de um futuro ambiente de projeto de sistemas de controle para SEDs (que envolveriam ferramentas de simulação baseadas no controlador, geração de relatórios e estatísticas, etc.), o que contribuiria muito para a redução do *'lead-time'* para o *'design'* desses sistemas de controle.

Portanto, são subprodutos deste trabalho:

- A disponibilização de uma biblioteca em Visual C++ de elementos utilizados no E-MFG que facilite o desenvolvimento de outras ferramentas;
- A disponibilização de um modelo de desenvolvimento de aplicações e ferramentas baseadas na biblioteca de elementos E-MFG;

Assim, o controlador e seu modelo de dados (detalhados no Capítulo 6) constituem o ponto de partida para futuros projetos visando um ambiente de modelagem e design de sistemas de controle para SEDs. Entre as possibilidades de futuros trabalhos se encontram o desenvolvimento dos seguintes módulos:

- Interface de visualização gráfica para supervisão;
- Interface de programação gráfica em dois níveis (Grafo e Objetos);
- Interface com banco de dados para realizar o rastreamento e a geração das ordens de produção;
- Interface com outros controladores (CLP's, Fieldbus, etc.) através de outros métodos ou protocolos (e.g. TCP/IP, XML, MQSeries);
- Simulador E-MFG ou PFS/E-MFG baseado na estrutura de dados do controlador (e integrado ao controlador);
- Ferramentas de Análise on-line do sistema produtivo com geração de relatórios e estatísticas;
- Biblioteca de grafos E-MFG para a simulação e validação do controle.

Uma característica importante do controlador é a parametrização da comunicação com o meio externo através do protocolo DDE, esta abordagem torna-se transparente para o controlador informações provenientes de outros *'softwares'* como

drivers de PLC's ou programas que simulem a planta controlada. Assim, pode-se validar o programa de controle sem a necessidade de se estar junto à planta, pois a informação proveniente de um programa de simulação ou de um driver para um *'hardware'* específico é tratada igualmente pelo Controlador de I/O.

Bibliografia

- ARAKAKI, J., **Análise de Sistemas de Manufatura Através da Metodologia MFG/PFS e Regras De Produção**, Tese de Mestrado, EPUSP, São Paulo 1993

- BEHFOROOZ, A. & HUDSON, F. J., **Software Engineering Fundamentals**, Oxford University Press, New York, 1996

- HASEGAWA,K. et al., Proposal of Mark Flow Graph for Discrete System Control, **Trans. of SICE**, Tokyo, v.20, n.2, p. 122-129, 1984 (em japonês)

- HASEGAWA,K. e TAKASHI, K. Simulation of Discrete Production Systems Based on Mark Flow Graph. **System Science**, Wroclaw, v.13, n.1-2, 1987(em japonês)

- MIYAGI, P. E., **A Study on Mark Flow Graph Based Programming Method for Robots**, Tokyo Institute of Technology, Tese de Mestrado, Japan,1985

- MIYAGI, P. E., **Control System Design, Programming and Implementation for Discrete Event Production Systems by Using Mark Flow Graph**, Tokyo Institute of Technology, Tese de Doutorado, Japan, 1988

- MIYAGI, P.E ; CAMARINHA-MATOS, L. M. SANTOS FILHO, D. J.; BARATA, J; ARAKAKI, J.; Application of Enhanced Mark Flow Graph in Real Time Control Systems., In: **IFAC 4th SYSMPOSIUM ON LOW COST AUTOMATION, Proceedings**, IFAC/AADECA, Buenos Aires, p163 - 170, 1995

- MIYAGI, P.E; KAGOHARA, M. Y.; MOTOHASHI, C. T.; TSUGAWA, M. F.; PELLICER, J. E.; CARELLI, R. Training System for Control of Discrete Event Systems. In: **IFAC 4th SYSMPOSIUM on LOW COST AUTOMATION, Proceedings**. IFAC/AADECA, Buenos Aires, p157-162, 1995

- MIYAGI, P. E., **Controle Programável**, Editora Edgard Blücher Ltda., São Paulo, 1996

- PETERSON, J.L., **Petri Net Theory and the Modeling of Systems**, Prentice-Hall, 1981

- REISIG, W., **Petri Nets an Introduction**, Springer-Verlag, New York, 1985

- REISIG, W., **A Primer in Petri Net Design**, Springer-Verlag, Berlin Heidelberg, 1992

- SANTOS FILHO, D. J. - **Proposta do Mark Flow Graph Estendido para a Modelagem e Controle de Sistemas Integrados de Manufatura**, Dissertação de Mestrado, EPUSP, São Paulo, 1993

- SANTOS FILHO, D. J.; MIYAGI, P.E. Enhanced Mark Flow Graph to Control Flexible Manufacturing Systems, **Revista Brasileira de Ciências Mecânicas**, ABCM, Rio de Janeiro, RJ, v XVII, n2, p232-248, 1995

- SANTOS FILHO, D. J.; MIYAGI, P.E Enhanced Mark Flow Graph to Control Autonomous Guided Vehicle. In: **CAPE'95 COMPUTER APPLICATIONS IN PRODUCTION ENGINEERING. Proceedings** (livro publicado pela Chapman& Hall, Londres, 1995) IFIP The International Federation for Information Processing, Beijing,, p856-865, 1995

Apêndice I - Cronograma de Atividades

O projeto foi desenvolvido de acordo com o cronograma apresentado na figura 9-1. Houve uma grande aderência ao cronograma durante a evolução do projeto. Apesar desta aderência ao cronograma, durante a etapa de desenvolvimento (codificação e testes) foi necessário um cuidado especial nos testes de construção e funcionalidade dos objetos implementados (devido a intensa interdependência dos objetos que compõem a representação do grafo), a fim de se evitar atrasos no cronograma de testes de validação geral.

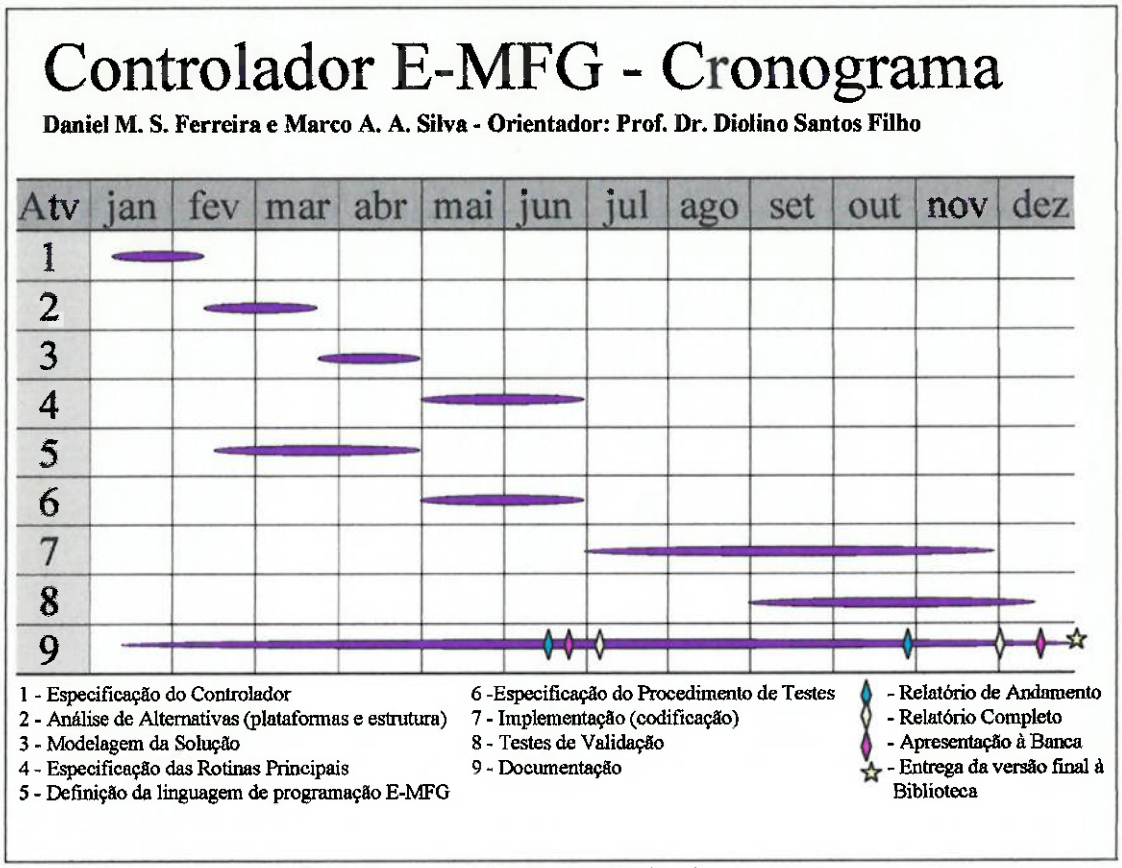


figura I-1 - Controlador E-MFG - Cronograma

Apêndice II - Seqüência de Testes

Neste apêndice procura-se mostrar alguns exemplos de grafos e procedimentos que foram utilizados para testar o controlador. Os testes não se restringiram a estes apresentados nesta seção, sendo estes testes apenas ilustrativos.

a. Seqüência 1 - Testes de Construção do Grafo e Dinâmica de Disparo

A fim de se realizar os testes progressivos de construção e dinâmica de disparo utilizou-se de um conjunto de dois sub-grafos para simular a integração entre dois processos industriais (figura I-a1).

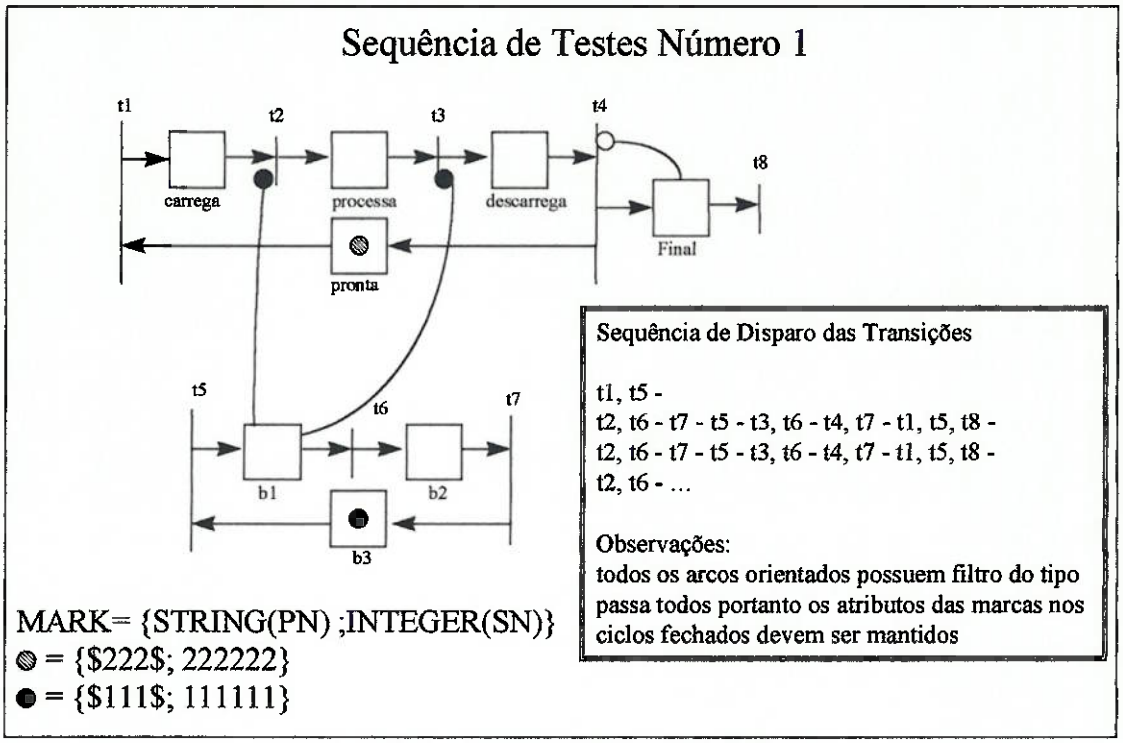


figura II-a1 - Grafo da Seqüência de Testes 1

```
EMFG Script version : 3.0;
Program Title: Sequência de Testes 1;
Description: Testes de Dinâmica de
Disparo;

<INCLUDE>
</INCLUDE>

<EMFG>

<MARK>
STRING(pn);
INTEGER(sn);
</MARK>

<BOXES>
COMMON(carrega);
COMMON(Processa);
COMMON(Descarrega);
COMMON(Final);
COMMON(ProNTa);
COMMON(b1);
COMMON(b2);
COMMON(b3);
</BOXES>
```

```
<TRANSITIONS>
COMMON(t1);
COMMON(t2);
COMMON(t3);
COMMON(t4);
COMMON(t5);
COMMON(t6);
COMMON(t7);
COMMON(t8);
</TRANSITIONS>
<ARCS>
FROM_TRANSITION(a1,t1,carrega);
FROM_TRANSITION(a2,t2,processa);
FROM_TRANSITION(a3,t3,descarrega);
FROM_BOX(a4,carrega,t2);
FROM_BOX(a5,processa,t3);
FROM_BOX(a6,descarrega,t4);
FROM_TRANSITION(a7,t4,final);
FROM_TRANSITION(a8,t4,pronta);
FROM_BOX(a9,final,t8);
FROM_BOX(a10,pronta,t1);
FROM_TRANSITION(a11,t5,b1);
FROM_TRANSITION(a12,t6,b2);
```

```
FROM_TRANSITION(a13,t7,b3);
FROM_BOX(a14,b3,t5);
FROM_BOX(a15,b1,t6);
FROM_BOX(a16,b2,t7);
</ARCS>
<GATES>
INTERNAL_PERMIT(g1,b1,t2);
INTERNAL_PERMIT(g2,b1,t3);
INTERNAL_NOT_PERMIT(g3,final,t4);
</GATES>
<INITIAL>
ADD_MARK(pronta);
{
TO_STR(pn,$222$);
TO_NUM(sn,222222);
}
ADD_MARK(b3);
{
TO_STR(pn,$111$);
TO_NUM(sn,111111);
}
</INITIAL>

</EMFG>
```

O arquivo *"SEQUÊNCIA DE TESTES 1.DMP"* é o histórico de disparo de transições para o grafo da figura I-a1. A estrutura do arquivo é a seguinte:

CT [dia/mês/ano] [hora : minuto : segundo] Fired: LABEL

Onde CT é o Cycle Tag que é um valor que alterna entre "<>" e "><" a cada ciclo de disparo de transições e LABEL é o nome da Transição conforme definido no arquivo *E-MFG Script*.

SEQUÊNCIA DE TESTES 1.DMP	
<> [01/12/98][22:45:36] Fired: T1	>< [01/12/98][22:45:38] Fired: T4
<> [01/12/98][22:45:36] Fired: T5	>< [01/12/98][22:45:38] Fired: T7
>< [01/12/98][22:45:36] Fired: T2	<> [01/12/98][22:45:39] Fired: T1
>< [01/12/98][22:45:36] Fired: T6	<> [01/12/98][22:45:39] Fired: T5
<> [01/12/98][22:45:36] Fired: T7	<> [01/12/98][22:45:39] Fired: T8
>< [01/12/98][22:45:36] Fired: T5	>< [01/12/98][22:45:39] Fired: T2
<> [01/12/98][22:45:37] Fired: T3	>< [01/12/98][22:45:39] Fired: T6
<> [01/12/98][22:45:37] Fired: T6	<> [01/12/98][22:45:39] Fired: T7
>< [01/12/98][22:45:37] Fired: T4	>< [01/12/98][22:45:39] Fired: T5
>< [01/12/98][22:45:37] Fired: T7	<> [01/12/98][22:45:39] Fired: T3
<> [01/12/98][22:45:37] Fired: T1	<> [01/12/98][22:45:39] Fired: T6
<> [01/12/98][22:45:37] Fired: T5	>< [01/12/98][22:45:40] Fired: T4
<> [01/12/98][22:45:37] Fired: T8	>< [01/12/98][22:45:40] Fired: T7
>< [01/12/98][22:45:37] Fired: T2	<> [01/12/98][22:45:40] Fired: T1
>< [01/12/98][22:45:37] Fired: T6	<> [01/12/98][22:45:40] Fired: T5
<> [01/12/98][22:45:38] Fired: T7	<> [01/12/98][22:45:40] Fired: T8
>< [01/12/98][22:45:38] Fired: T5	>< [01/12/98][22:45:40] Fired: T2
<> [01/12/98][22:45:38] Fired: T3	>< [01/12/98][22:45:40] Fired: T6
<> [01/12/98][22:45:38] Fired: T6	<> [01/12/98][22:45:40] Fired: T7

Como se pode perceber comparando o arquivo com a saída esperada verifica-se o correto funcionamento do programa para este caso.

b. Seqüência 2 - Testes de Construção do Grafo e Dinâmica de Disparo

Dando continuidade aos testes progressivos de construção e dinâmica de disparo utilizou-se um grafo simples para testar a manutenção de atributos de marcas individuais e a regra de composição de atributos das marcas (figura I-b1).

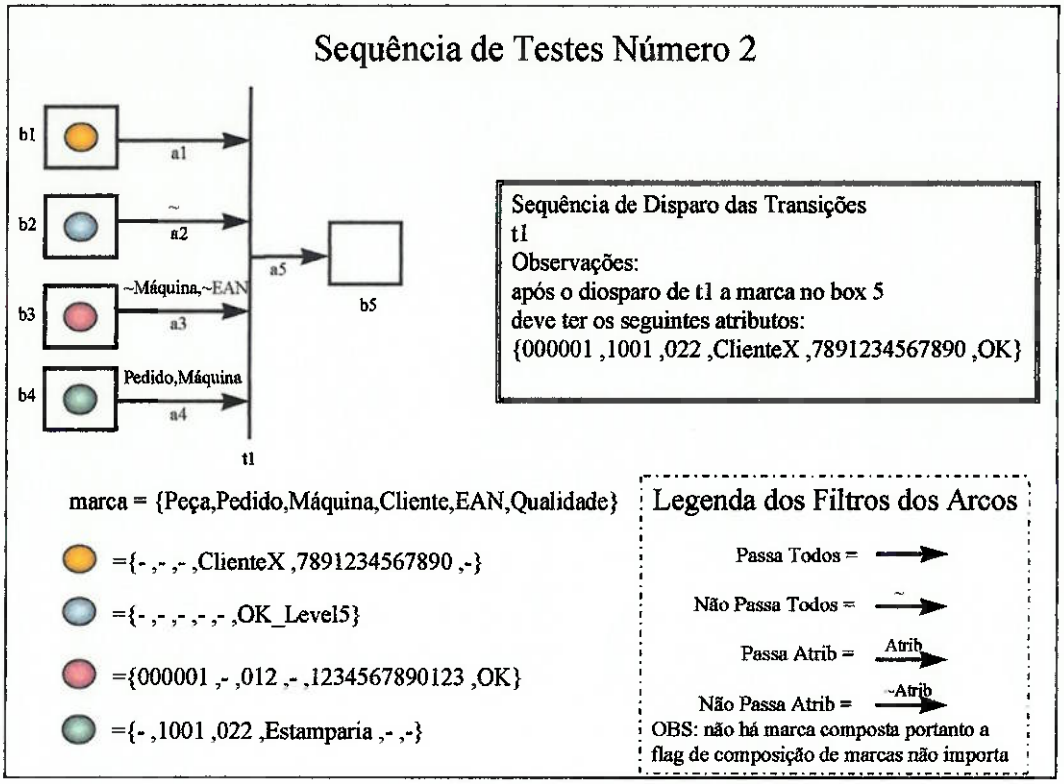


figura II-b1 - Grafo da Seqüência de Testes 2

```
TimeStamp: [d/m/y][H:M:S]
TimeStamp: [15/12/98][23:26:18]
/* EMFG Script Language Version 3.0 */
/* EMFG Interpreter Version 1.0 */
/* EMFG Linker Version 1.0 */
/* EMFG MarkManager Version 1.0 */

/* EMFG Communicator Version 1.0 */

<< E-MFG Boxes Connections Dump >>

Dumping Format:
 [Origin Transition Label]->(BoxLabel)->[Destiny Transition
Label]
if a box has a mark it's label will be marked as follows:
[Origin Transition Label]->(** BoxLabel **)->[Destiny
Transition Label]

[ ]-> (B1) ->[ T1 ]
[ ]-> (B2) ->[ T1 ]
[ ]-> (B3) ->[ T1 ]
[ ]-> (B4) ->[ T1 ]
[ T1 ]-> (** B5 **) ->[ ]
```

```
<<END: E-MFG Boxes Connections Dump >>
<< E-MFG Boxes Properties Dump >>

Dumping Format:/nName , Type , # Marks: Atrib1 = XXX &
Atrib2 = XXX & ... & AtribN = XXX

B1 , COMMON_BOX , #Marks=0
B2 , COMMON_BOX , #Marks=0
B3 , COMMON_BOX , #Marks=0
B4 , COMMON_BOX , #Marks=0
B5 , COMMON_BOX , #Marks=1: CLI = CLIENTEX & E =
1234567890123 & MÁQ = 012 & PED = 0 & PEÇA =
000001 & QDE = OK


<<END: E-MFG Boxes Properties Dump >>
<< E-MFG Gates Values Dump >>

Dumping Format:
(GateLabel)=Value
(GateLabel)='String'
<<END: E-MFG Gates Values Dump >>
```

```
EMFG Script version : 3.0;
Program Title: Sequência de Testes 2;
Description: Filtragem Seletiva;

<INCLUDE>
</INCLUDE>
<EMFG>
<MARK>
STRING(peça);
INTEGER(ped);
STRING(máq);
STRING(cli);
STRING(E);
STRING(qde);
</MARK>
<BOXES>
COMMON(b1);
COMMON(b2);
COMMON(b3);
COMMON(b4);
COMMON(b5);
</BOXES>
<TRANSITIONS>
COMMON(t1);
</TRANSITIONS>
```

```
<ARCS>
FROM_BOX(a1,b1,t1);
ADD_FILTER(a1,NOT_PASS_COMPO
SITE,PASS_ALL);
{
}
FROM_BOX(a2,b2,t1);
ADD_FILTER(a2,NOT_PASS_COMPO
SITE,NOT_PASS_ALL);
{
}
FROM_BOX(a3,b3,t1);
ADD_FILTER(a3,NOT_PASS_COMPO
SITE,NOT_PASS);
{
E;
máq;
}
FROM_BOX(a4,b4,t1);
ADD_FILTER(a4,NOT_PASS_COMPO
SITE,PASS);
{
ped;
máq;
}
FROM_TRANSITION(a5,t1,b5);
% omitindo o filtro do arco a5
</ARCS>
```

```
<GATES>
</GATES>
<INITIAL>
ADD_MARK(b1);
{
TO_STR(cli, clientex);
TO_STR(E,7891234567890);
}
ADD_MARK(b2);
{
TO_STR(qde,OK_LEVEL5);
}
ADD_MARK(b3);
{
TO_STR(peça,000001);
TO_STR(máq,012);
TO_STR(E,1234567890123);
TO_STR(qde,OK);
}
ADD_MARK(b4);
{
TO_NUM(ped,1001);
TO_STR(máq,022);
TO_STR(cli,estamparia);
}
</INITIAL>
</EMFG>
```

c. Seqüência 3 - Testes de Construção do Grafo, Dinâmica de Disparo e integração com outros aplicativos

A fim de aumentar a complexidade dos testes progressivos de construção e dinâmica de disparo utilizou-se um grafo representando a dinâmica de um processo industrial (figura I-c1) ligado a um sistema de controle representado pelo grafo da figura I-c2. Este teste foi realizado utilizando-se duas instâncias do controlador conectadas através de DDE.

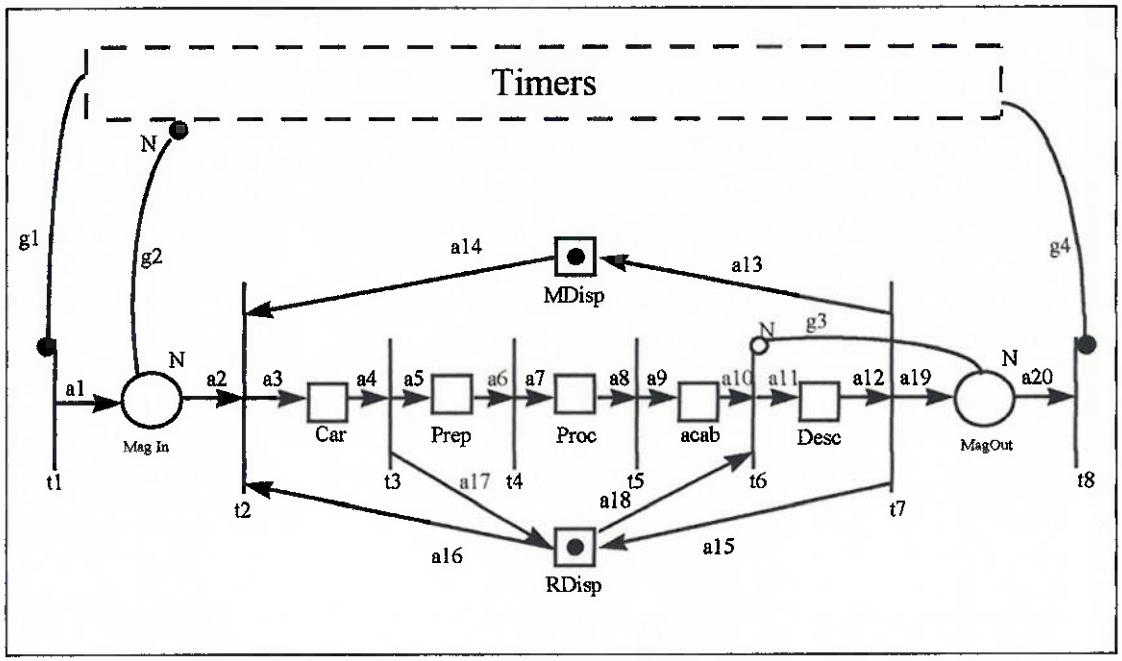


figura II-c1 - Grafo da Seqüência de Testes 3 - Grafo do Processo

```
EMFG Script version : 3.0;

Program Title: SeqTest3-Plant;

Description: Planta da
Sequência de Testes 3;
<INCLUDE>
</INCLUDE>

<EMFG>

<MARK>
</MARK>

<BOXES>
CAPACITY(MagIn,5,Fifo);
COMMON(Car);
COMMON(Prep);
COMMON(Proc);
COMMON(Acab);
COMMON(Desc);
COMMON(MDISP);
COMMON(RDISP);
CAPACITY(MagOut,5,Fifo);
</BOXES>
```

```
<TRANSITIONS>
COMMON(t1);
COMMON(t2);
COMMON(t3);
COMMON(t4);
COMMON(t5);
COMMON(t6);
COMMON(t7);
COMMON(t8);
</TRANSITIONS>
<ARCS>
FROM_TRANSITION(a1,t1,MagIn);
FROM_BOX(a2,MagIn,t2);
FROM_TRANSITION(a3,t2,Car);
FROM_BOX(a4,Car,t3);
FROM_TRANSITION(a5,t3,Prep);
FROM_TRANSITION(a6,t3,RDisp);
FROM_BOX(a7,Prep,t4);
FROM_BOX(a8,RDisp,t2);
FROM_TRANSITION(a9,t4,Proc);
FROM_BOX(a10,Proc,t5);
FROM_TRANSITION(a11,t5,Acab);
FROM_BOX(a12,Acab,t6);
FROM_BOX(a13,RDisp,t6);
FROM_TRANSITION(a14,t6,Desc);
FROM_BOX(a15,Desc,t7);
```

```
FROM_TRANSITION(a16,t7,MagOut);
FROM_TRANSITION(a17,t7,MDisp);
FROM_TRANSITION(a18,t7,RDisp);
FROM_BOX(a19,MDisp,t2);
FROM_BOX(a20,MagOut,t8);
</ARCS>
<GATES>
EXTERNAL_INPUT_PERMIT(gate1,t1,DDE,SeqT
est3-Timers!data!gate1,ACTIVE);
EXTERNAL_OUTPUT_FULL_PERMIT(gate2,Ma
gIn,DDE,SeqTest3-Timers!data!gate2,ACTIVE);
INTERNAL_FULL_NOT_PERMIT(gate3,MagOut,t
6);
EXTERNAL_INPUT_PERMIT(gate4,t8,DDE,SeqT
est3-Timers!data!gate2,ACTIVE);
ADD_CONDITION(gate1,((TRUE)&&(TRUE)));
</GATES>
<INITIAL>
ADD_MARK(Mdisp);
{
}
ADD_MARK(Rdisp);
{
}
</INITIAL>
</EMFG>
```

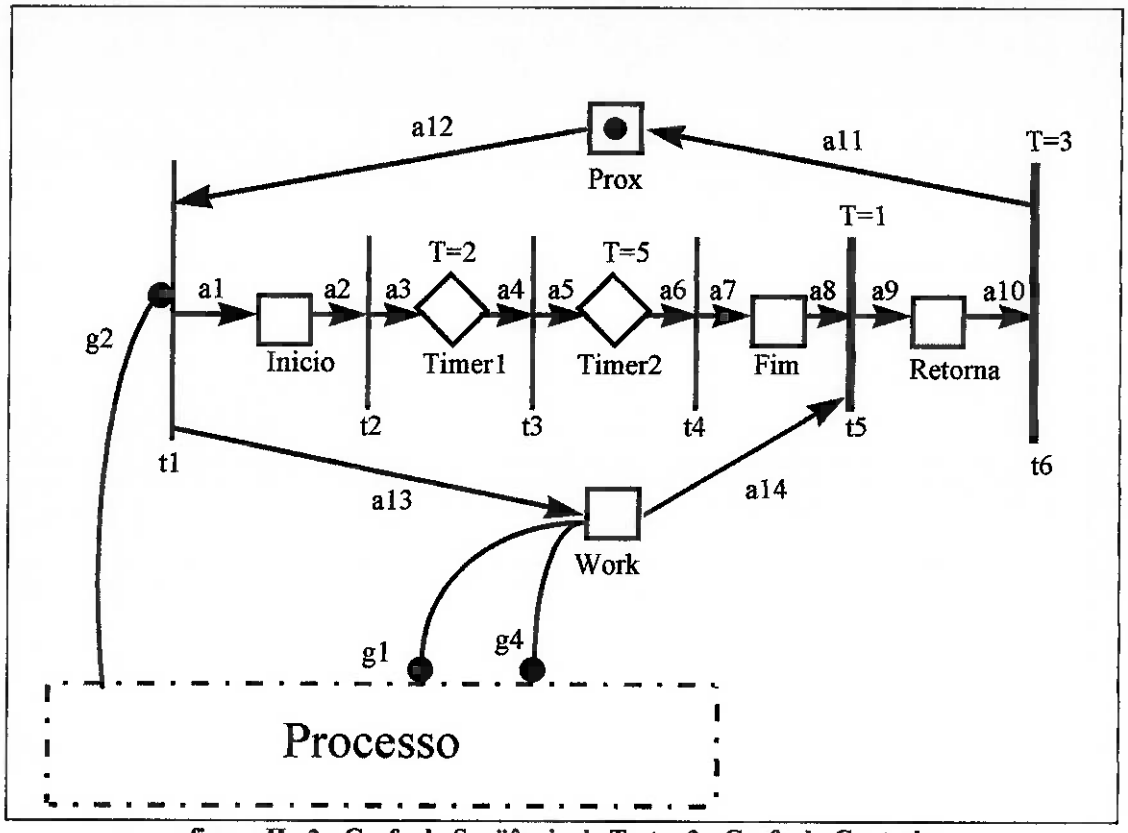


figura II-c2 - Grafo da Seqüência de Testes 3 - Grafo do Controle

EMFG Script version : 3.0;
Program Title: SeqTest3-Timers;
 	
Description: Timers da Sequência de Testes 4;
 	
<INCLUDE>
</INCLUDE>
 	
<EMFG>
 	
<MARK>
</MARK>
 	
<BOXES>
COMMON(Inicio);
TEMPORIZED(Timer1,2);
TEMPORIZED(Timer2,5);
COMMON(Fim);
COMMON(Retorna);
COMMON(Work);
COMMON(Prox);
</BOXES>
	<TRANSITIONS>
	COMMON(t1);
	COMMON(t2);
	COMMON(t3);
	COMMON(t4);
	TEMPORIZED(t5,1);
	TEMPORIZED(t6,3);
	</TRANSITIONS>
	
	<ARCS>
	FROM_TRANSITION(a1,t1,Inicio);
	FROM_BOX(a2,Inicio,t2);
	FROM_TRANSITION(a3,t2,Timer1);
	FROM_BOX(a4,Timer1,t3);
	FROM_TRANSITION(a5,t3,Timer2);
	FROM_BOX(a6,Timer2,t4);
	FROM_TRANSITION(a7,t4,Fim);
	FROM_BOX(a8,Fim,t5);
	FROM_TRANSITION(a9,t5,Retorna);
	FROM_BOX(a10,Retorna,t6);
	FROM_TRANSITION(a11,t6,Prox);
	FROM_BOX(a12,Prox,t1);
	FROM_TRANSITION(a13,t1,Work);
	FROM_BOX(a14,Work,t5);
	</ARCS>
		<GATES>
		EXTERNAL_OUTPUT_PERMIT(gate1,Work,
		DDE,PASSIVE);
		EXTERNAL_OUTPUT_PERMIT(gate4,Work,
		DDE,PASSIVE);
		EXTERNAL_INPUT_PERMIT(gate2,t1,DDE,
		,PASSIVE);
		</GATES>
		
		<INITIAL>
		ADD_MARK(Prox);
		{
		}
		</INITIAL>
		
		</EMFG>

d. Seqüência 4 - Testes de Integração com outros aplicativos

Utilizando o grafo da figura II-c1, descrito através do programa a seguir foram monitorados a evolução dinâmica do sistema através de arcos de sinal de saída que se conectavam via DDE a um segundo aplicativo.

O arquivo de entrada e a evolução dinâmica do grafo se encontram a seguir.

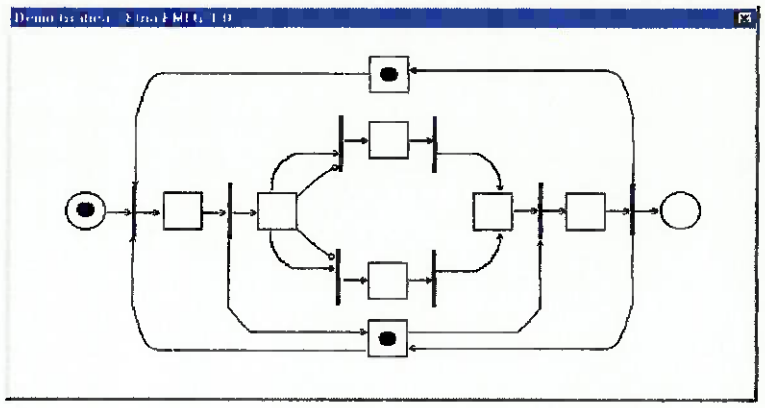


figura II-d1 - Estado 1

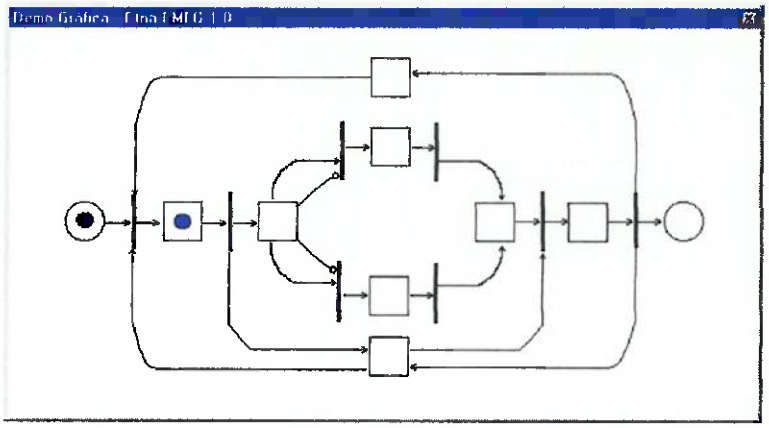


figura II-d2 - Estado 2

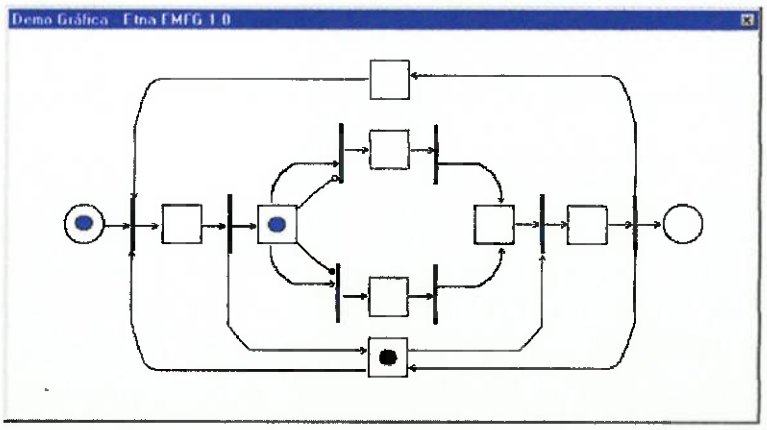


figura II-d3 - Estado 3

```
EMFG Script version : 3.0;
Program Title: TESTE1;
Description: programa do trabalho de formatura;
<INCLUDE>
</INCLUDE>
<EMFG>
<MARK>
STRING(PART);
</MARK>
<BOXES>
CAPACITY(MAGIN,10,FIFO);
TEMPORIZED(CARREGA,2);
TEMPORIZED(PREPARADA,1);
COMMON(MDISP);
COMMON(RDISP);
TEMPORIZED(PROC_A,5);
TEMPORIZED(PROC_B,5);
TEMPORIZED(ACABADA,1);
TEMPORIZED(DESCARREGA,1)
CAPACITY(MAGOUT,10,LIFO);
</BOXES>
<TRANSITIONS>
COMMON(T1);
COMMON(T2);
COMMON(T3);
COMMON(T4);
COMMON(T5);
COMMON(T6);
COMMON(T7);
COMMON(T8);
</TRANSITIONS>
<ARCS>
FROM_BOX(a1,MAGIN,T1);
FROM_TRANSITION(a2,T1,CARREGA);
FROM_BOX(a3,CARREGA,T2);
FROM_TRANSITION(a4,T2,PREPARADA);
FROM_BOX(a5,PREPARADA,T3);
FROM_TRANSITION(a6,T3,PROC_A);
FROM_BOX(a7,PREPARADA,T4);
FROM_TRANSITION(a8,T4,PROC_B);
FROM_BOX(a9,PROC_A,T5);
FROM_TRANSITION(a10,T5,ACABADA);
FROM_BOX(a11,PROC_B,T6);
FROM_TRANSITION(a12,T6,ACABADA);
FROM_BOX(a13,ACABADA,T7);
FROM_TRANSITION(a14,T7,DESCARREGA);
FROM_BOX(a15,DESCARREGA,T8);
FROM_TRANSITION(a16,T8,MAGOUT);
%robô e máquina
FROM_TRANSITION(a17,T2,RDISP);
ADD_FILTER(a17,NOT_PASS_COMPOSITE,NOT_PASS_ALL);
{
}
FROM_BOX(a18,RDISP,T7);
FROM_TRANSITION(a19,T8,RDISP);
ADD_FILTER(a19,NOT_PASS_COMPOSITE,NOT_PASS_ALL);
{
}
FROM_BOX(a20,RDISP,T1);
FROM_TRANSITION(a21,T8,MDISP);
ADD_FILTER(a21,NOT_PASS_COMPOSITE,NOT_PASS_ALL);
{
}
FROM_BOX(a22,MDISP,T1);
</ARCS>
<GATES>
INTERNAL_PERMIT(GATE1,PREPARADA,T3);
ADD_CONDITION(GATE1,(PREPARADA!PART=='A'));
INTERNAL_PERMIT(GATE2,PREPARADA,T4);
ADD_CONDITION(GATE2,(PREPARADA!PART=='B'));

%Gates de interface

EXTERNAL_OUTPUT_DATA(MAGIN_A2,MAGIN,PART,DDE,DE
MO!DUMMY!MAGIN,ACTIVE);
EXTERNAL_OUTPUT_DATA(CARREGA_A3,CARREGA,PART,DD
E,DEMO!DUMMY!CARR,ACTIVE);
EXTERNAL_OUTPUT_DATA(PREPARADA_A4,PREPARADA,PAR
T,DDE,DEMO!DUMMY!PREP,ACTIVE);
EXTERNAL_OUTPUT_DATA(PROCA_A5,PROC_A,PART,DDE,DE
MO!DUMMY!PROCA,ACTIVE);
EXTERNAL_OUTPUT_DATA(PROCB_A6,PROC_B,PART,DDE,DE
MO!DUMMY!PROCB,ACTIVE);
EXTERNAL_OUTPUT_DATA(ACABADA_A7,ACABADA,PART,D
DE,DEMO!DUMMY!ACAB,ACTIVE);
EXTERNAL_OUTPUT_DATA(DESCARREGA_A8,DESCARREGA,P
ART,DDE,DEMO!DUMMY!DESC,ACTIVE);
EXTERNAL_OUTPUT_DATA(MAGOUT_A9,MAGOUT,PART,DDE
,DEMO!DUMMY!MAGOUT,ACTIVE);
EXTERNAL_OUTPUT_PERMIT(RDISP_A10,RDISP,DDE,DEMO!D
UMMY!GATE2,ACTIVE);
EXTERNAL_OUTPUT_PERMIT(MDISP_A11,MDISP,DDE,DEMO!
DUMMY!GATE1,ACTIVE);


</GATES>
<INITIAL>
ADD_MARK(MAGIN)
{
TO_STR(PART,B);
}
ADD_MARK(MAGIN)
{
TO_STR(PART,B);
}
ADD_MARK(MAGIN)
{
TO_STR(PART,B);
}
ADD_MARK(MAGIN)
{
TO_STR(PART,B);
}
ADD_MARK(MAGIN)
{
TO_STR(PART,A);
}
ADD_MARK(MAGIN)
{
TO_STR(PART,B);
}
ADD_MARK(RDISP)
{
}
ADD_MARK(MDISP)
{
}
</INITIAL>
</EMFG>
```

Apêndice III - Manual do Aplicativo

O aplicativo controlador é bastante simples sob o ponto de vista do usuário, existindo poucos comandos que podem ser utilizados. A interface do controlador pode ser visualizada na figura III-1.

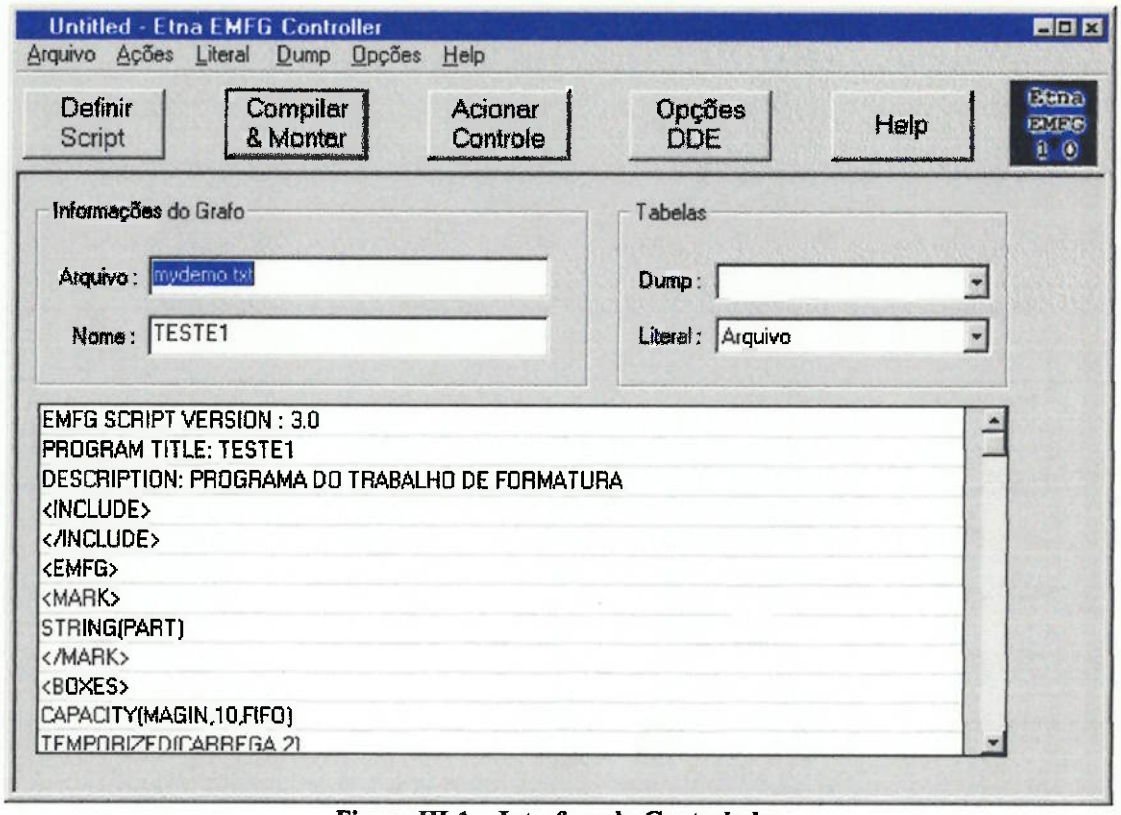


Figura III-1 – Interface do Controlador

A seguir, são apresentados os processos de uso do controlador:

Carregando um programa de controle:

Escolha o botão ***Definir Script*** e escolha o arquivo de dados a ser carregado. Este arquivo deve estar escrito de acordo com o E-MFG Script descrito no capítulo 5.

Selecione então o botão ***Compilar & Montar*** e o grafo será compilado e estará pronto para rodar. *Se existirem gates ativos de dados externos os seus valores iniciais serão carregados neste momento.*

Executando um programa de Controle:

Escolha o botão ***Acionar Controle***, a caixa de diálogo da figura III-2 será exibida. Nesta caixa de diálogo serão apresentados os tempos do ciclo de disparo de transição e do ciclo de comunicação, bem como 3 botões com as funções de executar um passo simples de controle (***Passo Simples***), executar o programa em modo contínuo (***Contínuo***) e sair (***Sair***).

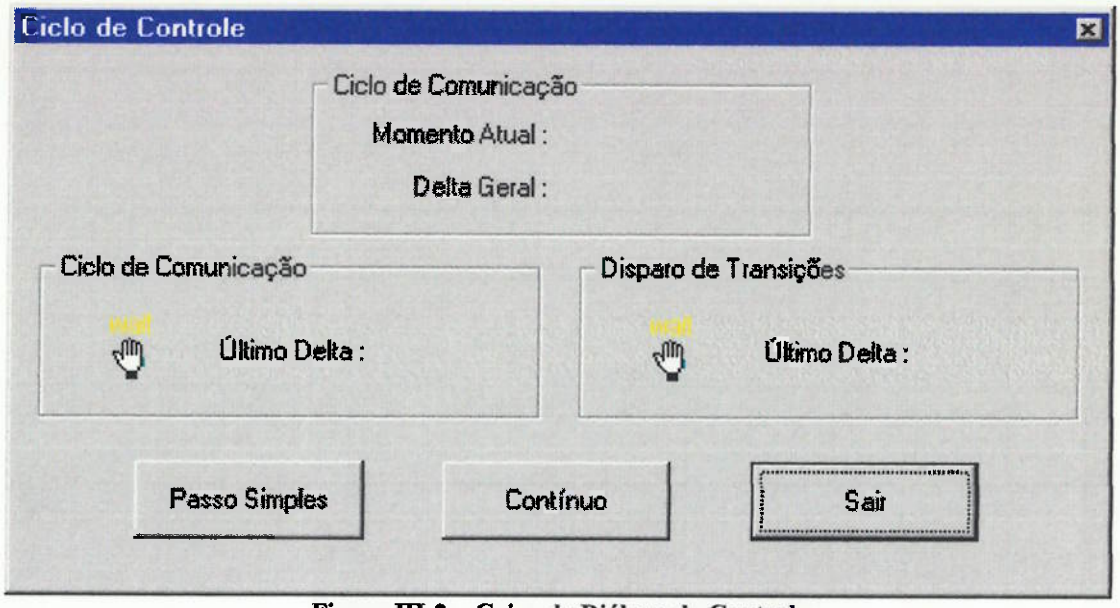


Figura III-2 – Caixa de Diálogo de Controle

Ao sair da caixa de diálogo de ciclo de controle o usuário pode selecionar uma das funções de *Dump* do menu dump e visualizar as propriedades do grafo (marcação e conexões, valores dos gates e propriedades dos boxes).

Testando e configurando a comunicação via DDE:

O botão **Opções DDE** permite que sejam configurados os parâmetros de comunicação e que sejam realizados testes com outros aplicativos.

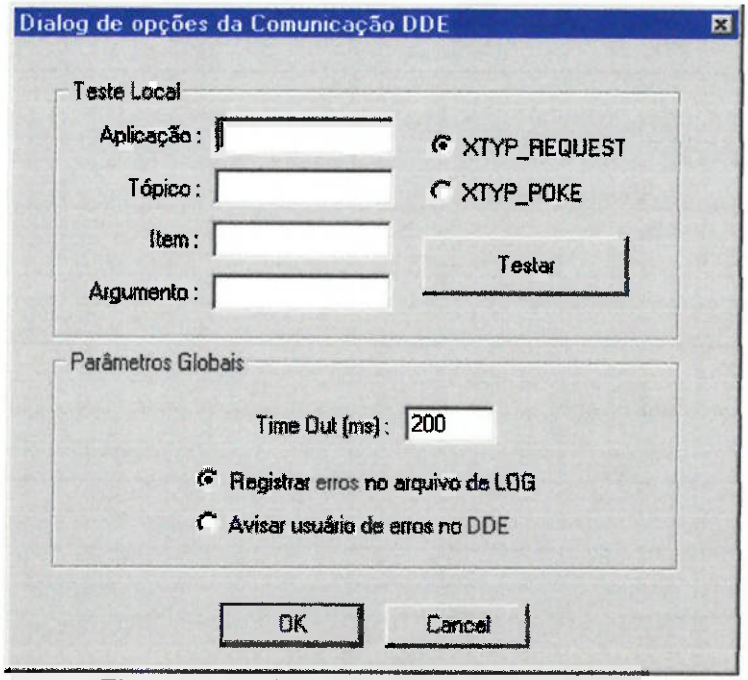


Figura III-3 – Caixa de Diálogo de Opções DDE

Histórico de Transações

O programa gera 3 arquivos de log, NOME.DMP (registro do disparo de transições), NOME.GPH (dump do estado do grafo) e NOME.DDE (registro dos erros de comunicação). Onde NOME é o nome do grafo.

Apêndice IV - Código Fonte do Controlador

Definições Globais

Files Elements.h and Definitions.h - Global

```
//include file for E-MFG Elements
//Definition Date: July 4th, 1998
//Last changed:        July 4th, 1998
//Defined by : Marco A. A. Silva
//Last changed by: Marco A. A. Silva

#include "definitions.h"                    //E-MFG Constant Definitions
#include "wordsarray.h"                     //Extension of String Arrays
#include "basis.h"                          //E-MFG Basis Class
#include "graph.h"                          //E-MFG Graph Components
#include "marks.h"                          //E-MFG Marks Definition
#include "arcfilt.h"        //E-MFG Arc Filter Definition
#include "arcs.h"                           //E-MFG Arcs Definition
#include "boxes.h"                          //E-MFG Boxes Definition
#include "transit.h"                        //E-MFG Transition Definition
#include "gates.h"                          //E-MFG Gates Definition
#include "list.h"                           //Linked List Definition
#include "arrays.h"                         //Arrays Extension Class

// definitions.h - Definitions of eMFG constants:
// version:1.0
// Definition Date: 28/01/98
// Last Modified: 06/07/98
// programed by: Marco A. A. Silva

// Objetivos: colocar todos as constantes num só arquivo para ser
// incluído toda a vez que for necessário (basis.h, interp.h)
// para deixar as definições mais limpas

// Tipo Indefinido (Uso múltiplo)
#define NULO -1

// Versão Atual do SCRIPT
#define SCRIPT_VERSION "3.0"

// Número máximo de elementos de cada tipo no grafo
#define NMAX 255

// Tipos de Dados
#define BOOLEAN          3
#define TEXT_ATRIB       2
#define INTEGER_ATRIB    1
#define STILL_UNDEFINED  0

// Tipos de Nodes de Array
#define STRING_NODE      1
#define MARKAT_NODE      2
#define ATTRIB_NODE      3
#define ARC_NODE         4
#define BOX_NODE         5
#define TRANSIT_NODE     6
#define GATE_NODE        7
#define MARKS_NODE       8
#define CONDS_NODE       9
#define ATTRI_NODE       10
#define ATTRIARRAY_NODE  11
#define INT_NODE         12


// Possible Atribuitions and/or Conditions (Types):
#define UNDEFINED_TYPE   0
#define EQUAL_TO_NUM_CTE 1
#define EQUAL_TO_STRING_CTE 2
#define EQUAL_TO_GATE    3
```

```
#define EQUAL_TO_MARKATRIB        4
#define NOT_EQUAL_TO_NUM_CTE      5
#define NOT_EQUAL_TO_STRING_CTE   6
#define NOT_EQUAL_TO_GATE         7
#define NOT_EQUAL_TO_MARKATRIB    8
#define GREATER_THAN_NUM_CTE      9
#define GREATER_THAN_STRING_CTE   10
#define GREATER_THAN_GATE         11
#define GREATER_THAN_MARKATRIB    12
#define LOWER_THAN_NUM_CTE        13
#define LOWER_THAN_STRING_CTE     14
#define LOWER_THAN_GATE           15
#define LOWER_THAN_MARKATRIB      16
#define EQUAL_OR_GREATER_THAN_NUM_CTE          17
#define EQUAL_OR_GREATER_THAN_STRING_CTE       18
#define EQUAL_OR_GREATER_THAN_GATE             19
#define EQUAL_OR_GREATER_THAN_MARKATRIB        20
#define EQUAL_OR_LOWER_THAN_NUM_CTE            21
#define EQUAL_OR_LOWER_THAN_STRING_CTE         22
#define EQUAL_OR_LOWER_THAN_GATE               23
#define EQUAL_OR_LOWER_THAN_MARKATRIB          24

// Possible Left Elements of Conditional Expressions:
#define COMPARE_MARK 1
#define COMPARE_GATE 0

// tipos de boxes
#define COMMON_BOX              0
#define TEMP_BOX                1
#define PACKING_BOX             2
#define UNPACKING_BOX           3
#define TRANSFORMATOR_BOX       4
#define CAPACITY_BOX            5

//tipos de transição
#define COMMON_TRANSITION       0
#define TEMPORIZED_TRANSITION   1

//tipos de gates
#define INT_PERMIT                          0
#define INT_NOT_PERMIT                      1
#define INT_FULL_PERMIT                     2
#define INT_FULL_NOT_PERMIT                 3
#define EXTERNAL_INPUT_PERMIT               4
#define EXTERNAL_INPUT_NOT_PERMIT           5
#define EXTERNAL_OUTPUT_PERMIT              6
#define EXTERNAL_OUTPUT_NOT_PERMIT          7
#define EXTERNAL_OUTPUT_FULL_PERMIT         8
#define EXTERNAL_OUTPUT_FULL_NOT_PERMIT     9
#define INT_DATA                            10
#define INT_N                               11
#define EXTERNAL_OUTPUT_DATA                12
#define EXTERNAL_OUTPUT_N                   13
#define EXTERNAL_INPUT_DATA                 14

// Tipos de Ativação
#define ACTIVE   0
#define PASSIVE  1

// Tipos de Métodos de Comunicação
#define INTERNAL  0
#define DDE       1

// Return Data
#define NUMERICAL_DATA          0
#define STRING_DATA             1
#define BOOLEAN_DATA            2

// Tipos de Arco
#define NOT_FILTER          0
#define FILTER              1
```

```
// Tipos de Origem do Arco:
#define UNKNOWN                 0
#define FROM_TRANSITION         1
#define FROM_BOX                2

// Tipos Especiais de Condicionais

// Operadores Lógicos
#define AND                     0
#define OR                      1
#define ALWAYS_TRUE             2
#define ALWAYS_FALSE            3

// Operadores de comparação
#define EQUAL                   0
#define GREATER                 1
#define LESSER                  2
#define DIFFERENT               3

// Tipos de R-VALUE
#define INT_CTE                 0
#define STR_CTE                 1
#define INT_BOX_ATB             3
#define STR_BOX_ATB             4
#define INT_GATE_VAL            5
#define STR_GATE_VAL            6

// Defines do Interpretador
#define MAX_LINE_SIZE           255;

// Tipos de Filtro
#define PASS_ALL                0
#define NOT_PASS_ALL            1
#define PASS                    2
#define NOT_PASS                3
#define PASS_COMPOSITE          4
#define NOT_PASS_COMPOSITE      5

// Tipos de Endereço
#define SINGLE      0
#define DOUBLE      1

//Tipos de Ação
#define NONE        0
#define STEP        1
#define CONT        2
#define LEAVE       3
```

A - Elementos Estruturais Básicos

Files Basis.h and Basis.cpp - E-MFG base build elements

```
// basis.h - Definition of EMFG base build elements:
// version:2.0
// Definition Date: 11/12/97
// Last Modified: 08/11/98
// Programed by: Daniel M. S. Ferreira
// Last Modified by: Daniel M. S. Ferreira

// Objetivos: Definir as estruturas de dados auxiliares
// para a implementação do grafo (condições e atribuições e atributos)
//


class CMarkAttribute:public CObject
{
private:
        // Atribute Label
        CString* mp_Label;
        // Atribute Type
```

```
        short m_Type;
        // Atribute Data for Text Atributes
        CString* mp_TextAtribute;
        // Atribute Data for Integer Atributes
        long m_IntegerAtribute;

public:
        // Construction & Destruction:
        CMarkAttribute();
        ~CMarkAttribute();

        CMarkAttribute(CString MyLabel, short MyType);

        // Build Functions:
        void Create(CString MyLabel, short MyType);

        void ResetAtrib(); // Resets attribute
        void SetToNull(); // Set Attribute to Null values

        // Data Management:
        void SetAtrib(CString MyTextAtrib); //Sets the attribute value
        void SetAtrib(long MyIntegerAtrib); //Sets the attribute value
        CString GetTextAtrib(); //Returns the mp_TextAtribute Member
        long GetIntegerAtrib(); //Returns the m_IntegerAtribute Member
        short GetType();  // Returns the attribute type

        CString GetLabel(); //Returns the attribute label
        CString GetAsText();//Returns the attribute value as a string
        CString GetDump(); // Returns the attribute in dump format

        //serialização
        DECLARE_SERIAL (CMarkAttribute);
        virtual void Serialize( CArchive& ar );


        // Construtor de Cópia e operador igual:
        CMarkAttribute(const CMarkAttribute& AtribSer);
        const CMarkAttribute& operator =( const CMarkAttribute& AtribSrc );

 // operador de igualdade
 BOOL operator ==(CMarkAttribute& AtribSrc );
};


////////////////////////////////////////////////////////////////////////////
//CAttribution Object Definition
//
class CAttribution:public CObject
{
protected:
        // Tipo de Atribuição
        short m_Type;
        // Índice do Box que contem a marca
        long m_ThisMarkBoxIndex;
        // Atributo da Marca
        long m_MarkAtrib;
        // Constantes de igualdade:
        long m_EqualNumConstant;
        CString m_EqualStringConstant;
        // Variáveis de igualdade:
        long m_EqualGateIndex;
        long m_EqualMarkBoxIndex;
        long m_EqualMarkBoxAtrib;

public:
        // Construction & Destruction
        CAttribution();
        ~CAttribution();
        CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualConstantOrGateIndex);
        CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,CString MyEqualConstant);
        CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualBoxIndex, long MyEqualAtribute);
```

```
        // Build Functions:
        void Create(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualConstant);
        void Create(short MyType,long MyMarkBox,long MyMarkAtrib,CString MyEqualConstant);
        void Create(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualBoxIndex, long MyEqualAtribute);



        // Data Management:
        BOOL DoAttribution(class CBoxArray* p_MyBoxArray,class CGateArray* p_MyGateArray); // Executa a atribuição
        //serialização
        DECLARE_SERIAL (CAttribution);
        virtual void Serialize( CArchive& ar );

        // Copy Constructor and Equal Operator
        CAttribution(const CAttribution& AtribSrc);
        const CAttribution& operator =( const CAttribution& AtribSrc );

};


////////////////////////////////////////////////////////////////////////////
//CConditional Object Definition
//eMFG Conditional Object
//version 2.0
//Definition date:06/07/98
//Last Modified:22/10/98
//Programmed by: Marco A. A. Silva
//Last changed by: Marco A. A. Silva

class CConditional : public CObject
{
//constructor & destructor
public:
        CConditional();
        ~CConditional();

//functions
public:
        void Create (       short myLogic,
                                                        class CGraph* myGraph=NULL,
                                                        BOOL myNegative=FALSE,
                                                        short myLBox=NULO, short
myLAttrib=NULO,
                                                        short myOperator=NULO,
                                                        short myRType=NULO,
                                                        short myRParam1=NULO, short
myRParam2=NULO, CString myRParam3="");

        void AddRight (CConditional& myRBrother);
        void AddDown  (CConditional& myDDaughter);

 void SetAlwaysTRUE();
 void SetAlwaysFALSE();

        BOOL Evaluate();

// Copy Constructor and Equal Operator
        CConditional(const CConditional& CondSrc);
        const CConditional& operator =( const CConditional& CondSrc );

//data members
private:
        class CGraph* mp_Owner; // Owner Graph

        short           m_Logic;                //Node Logical Operator
        BOOL            m_NOT;                          //Node Not Operator
        short           m_LBox;                         //L-Value Box
        short           m_LAttrib;              //L-Value Attribute
        short           m_Operator;             //operator type
        short           m_RType;                //R-Value type
        short           m_RIntCte;              //R-Value Integer Constant
        CString     m_RStrCte;              //R-Value String Constant
```

```
        short           m_RBox;                         //R-Value Box
        short           m_RAttrib;              //R-Value Attribute
        short           m_RGate;                //R-Value Gate

//pointer members
private:
        CConditional*   mp_Right; //Right brother conditional
        CConditional* mp_Down;    //Down daughter conditional
};

// basis.cpp - Implementation of EMFG base build elements
// version:2.0
// Definition Date: 11/12/97
// Last Modified: 20/06/98
// Programed by: Daniel M. S. Ferreira
// Last Modified by: Marco A. A. Silva

// Objetivos: Definir as estruturas de dados auxiliares
// para a implementação do grafo (condições e atribuições e atributos)
//

#include "stdafx.h"                     // standard windows aplication
#include "elements.h"           // EMFG Elements include list

////////////////////////////////////////////////////////////
// CMarkAtrib class:
#define CMARKATRIB_VERSION 1
        // Construction & Destruction:


CMarkAttribute::CMarkAttribute()
 {
        mp_Label=NULL;
        m_Type = STILL_UNDEFINED;
        mp_TextAtribute = NULL;
        m_IntegerAtribute = 0;
        }

CMarkAttribute::CMarkAttribute(CString MyLabel, short MyType)
        {
        ASSERT(MyLabel!="");
        ASSERT((MyType=INTEGER_ATRIB)||(MyType=TEXT_ATRIB));

        mp_Label = new CString(MyLabel);
        m_Type = MyType;
        switch (m_Type)
                {
                case INTEGER_ATRIB:
                        {
                        mp_TextAtribute = NULL;
                        m_IntegerAtribute = 0;
                        break;
                        }
                case TEXT_ATRIB:
                        {
                        mp_TextAtribute = new CString;
                        m_IntegerAtribute = 0;
                        break;
                        }
                };
        }

        CMarkAttribute::~CMarkAttribute()
        {
        if (mp_Label!=NULL)
                {
                delete mp_Label;
                mp_Label = NULL;
                }

        if (mp_TextAtribute!=NULL)
                {
```

```
		delete mp_TextAtribute;
		mp_TextAtribute = NULL;
		}
}


void CMarkAttribute::Create(CString MyLabel, short MyType)
{
ResetAtrib();
ASSERT(MyLabel!="");
ASSERT((MyType==INTEGER_ATRIB)||(MyType==TEXT_ATRIB));
mp_Label = new CString(MyLabel);
m_Type = MyType;
switch (m_Type)
		{
		case INTEGER_ATRIB:
			{
			mp_TextAtribute = NULL;
			m_IntegerAtribute = 0;
			break;
			}
		case TEXT_ATRIB:
			{
			mp_TextAtribute = new CString;
			m_IntegerAtribute = 0;
			break;
			}
		};
}


void CMarkAttribute::SetToNull()
{
switch (GetType())
		{
		case TEXT_ATRIB:
				{
				if(mp_TextAtribute!=NULL)
					{
					delete mp_TextAtribute;
					mp_TextAtribute = NULL;
					}
				mp_TextAtribute=new CString;
				break;
				}
		case INTEGER_ATRIB:
				{
				m_IntegerAtribute=0;
				break;
				}
		};
}

void CMarkAttribute::ResetAtrib()
{
if (mp_Label!=NULL)
		{
		delete mp_Label;
		mp_Label = NULL;
		}

if(mp_TextAtribute!=NULL)
		{
		delete mp_TextAtribute;
		mp_TextAtribute = NULL;
		}
		m_Type = STILL_UNDEFINED;
		m_IntegerAtribute = 0;
}
```

```
// Data Management:
void CMarkAttribute::SetAtrib(CString MyTextAtrib)
{
ASSERT(GetType()==TEXT_ATRIB);
if(mp_TextAtribute!=NULL)
        {
        delete mp_TextAtribute;
        mp_TextAtribute = NULL;
        }
mp_TextAtribute = new CString(MyTextAtrib);
}

void CMarkAttribute::SetAtrib(long MyIntegerAtrib)
{
ASSERT(GetType()==INTEGER_ATRIB);
m_IntegerAtribute = MyIntegerAtrib;
}

CString CMarkAttribute::GetTextAtrib()
{
ASSERT(GetType()==TEXT_ATRIB);
ASSERT(mp_TextAtribute!=NULL);
return (CString)*mp_TextAtribute;
}

long CMarkAttribute::GetIntegerAtrib()
{
ASSERT(GetType()==INTEGER_ATRIB);
return m_IntegerAtribute;
}

short CMarkAttribute::GetType()
{
return m_Type;
}

CString CMarkAttribute::GetLabel()
{
ASSERT (mp_Label!=NULL);
return *mp_Label;
}

CString CMarkAttribute::GetAsText()
{
CString MyReturn;
switch (GetType())
        {
        case TEXT_ATRIB:
                {
                MyReturn = GetTextAtrib();
                break;
                }
        case INTEGER_ATRIB:
                {
                MyReturn.Format("%ld",GetIntegerAtrib());
                break;
                }
        };
return MyReturn;
}

CString CMarkAttribute::GetDump()
{
CString MyReturn = GetLabel() + " = " + GetAsText();
return MyReturn;
}

//serialização
IMPLEMENT_SERIAL (CMarkAttribute,CObject,CMARKATRIB_VERSION);
void CMarkAttribute::Serialize( CArchive& ar )
        {
        if (ar.IsStoring())
```

```
                {
                ar<<(CString)*mp_Label;
                ar<<m_Type;
                if (m_Type==TEXT_ATRIB)
                                ar<<(CString)*mp_TextAtribute;
                if (m_Type==INTEGER_ATRIB)
                                ar<<m_IntegerAtribute;
                }
        else
                {
                ResetAtrib();
                mp_Label = new CString;
                ar>>(CString)*mp_Label;
                ar>>m_Type;
                if (m_Type==TEXT_ATRIB)
                                {
                                mp_TextAtribute = new CString;
                                ar>>(CString)*mp_TextAtribute;
                                }
                if (m_Type==INTEGER_ATRIB)
                                ar>>m_IntegerAtribute;
                }
        }

// Construtor de Cópia e operador igual:
        CMarkAttribute::CMarkAttribute(const CMarkAttribute& AtribScr)
        {
        ASSERT(AtribScr.mp_Label!=NULL);
        mp_Label = new CString(*AtribScr.mp_Label);
        m_Type =AtribScr.m_Type;
        if (m_Type==TEXT_ATRIB)
                {
                mp_TextAtribute = new CString(*AtribScr.mp_TextAtribute);
                m_IntegerAtribute = 0;
                }
        else
                {
                if (m_Type==INTEGER_ATRIB)
                                {
                                m_IntegerAtribute = AtribScr.m_IntegerAtribute;
                                mp_TextAtribute = NULL;
                                }
                else
                                {
                                mp_TextAtribute = NULL;
                                m_IntegerAtribute = 0;
                                }
                }
        }

        const CMarkAttribute& CMarkAttribute::operator =( const CMarkAttribute& AtribSrc )
        {
        ASSERT(AtribSrc.mp_Label!=NULL);
        if (this==&AtribSrc)
                {
                return *this;
                }
        else
                {
                if (mp_Label != NULL)
                        {
                        delete mp_Label;
                        mp_Label = NULL;
                        }
                if (mp_TextAtribute != NULL)
                        {
                        delete mp_TextAtribute;
                        mp_TextAtribute = NULL;
                        }
                mp_Label = new CString(*AtribSrc.mp_Label);
                m_Type =AtribSrc.m_Type;
                if (m_Type==TEXT_ATRIB)
```

```
                    {
                    mp_TextAtribute = new CString(*AtribSrc.mp_TextAtribute);
                    m_IntegerAtribute = 0;
                    }
                else
                    {
                    if (m_Type==INTEGER_ATRIB)
                        {
                        m_IntegerAtribute = AtribSrc.m_IntegerAtribute;
                        mp_TextAtribute = NULL;
                        }
                    else
                        {
                        mp_TextAtribute = NULL;
                        m_IntegerAtribute = 0;
                        }
                    }
                return *this;
                }
        }

BOOL CMarkAttribute::operator ==(CMarkAttribute& AtribSrc )
{
BOOL Return=FALSE;
if (GetType()==AtribSrc.GetType())
                {
                switch (GetType())
                        {
                        case INTEGER_ATRIB:
                                {
                                if (GetIntegerAtrib()==AtribSrc.GetIntegerAtrib())
                                        {
                                        Return=TRUE;
                                        }
                                else
                                        {
                                        Return = FALSE;
                                        }
                                break;
                                }
                        case TEXT_ATRIB:
                                {
                                if (GetTextAtrib()==AtribSrc.GetTextAtrib())
                                        {
                                        Return=TRUE;
                                        }
                                else
                                        {
                                        Return = FALSE;
                                        }
                                break;
                                }
                        };
                }
else
                {
                Return=FALSE;
                }
return Return;
}

//
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
// CAttribution class Implementation
//

// Obs: see definitions.h for CAttribution types

        CAttribution::CAttribution()
        {
        // Tipo de Atribuição
```

```
m_Type=UNDEFINED_TYPE;
// Índice do Box que contem a marca
m_ThisMarkBoxIndex = -1;
// Atributo da Marca
m_MarkAtrib = 0;
// Constantes de igualdade:
m_EqualNumConstant=0;
m_EqualStringConstant="";
// Variáveis de igualdade:
m_EqualGateIndex=-1;
m_EqualMarkBoxIndex=-1;
m_EqualMarkBoxAtrib=-1;
}

CAttribution::~CAttribution()
{
//void destructor (there is no object dynamic allocation)
}

CAttribution::CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualConstantOrGateIndex)
{
ASSERT((MyType==EQUAL_TO_NUM_CTE)||(MyType==EQUAL_TO_GATE));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);

// Tipo de Atribuição
m_Type=MyType;
// Índice do Box que contem a marca
m_ThisMarkBoxIndex = MyMarkBox;
// Atributo da Marca
m_MarkAtrib = MyMarkAtrib;

if (MyType==EQUAL_TO_NUM_CTE)
                        {
                        // Constantes de igualdade:
                        m_EqualNumConstant=MyEqualConstantOrGateIndex;
                        m_EqualStringConstant="";
                        // Variáveis de igualdade:
                        m_EqualGateIndex=-1;
                        m_EqualMarkBoxIndex=-1;
                        m_EqualMarkBoxAtrib=-1;
                        }
else
                        {
                        // Constantes de igualdade:
                        m_EqualNumConstant=0;
                        m_EqualStringConstant="";
                        // Variáveis de igualdade:
                        m_EqualGateIndex=MyEqualConstantOrGateIndex;
                        m_EqualMarkBoxIndex=-1;
                        m_EqualMarkBoxAtrib=-1;
                        }

}

CAttribution::CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,CString MyEqualConstant)
{
ASSERT((MyType==EQUAL_TO_STRING_CTE));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);
ASSERT(MyEqualConstant!="");

m_Type=MyType;// Tipo de Atribuição
m_ThisMarkBoxIndex = MyMarkBox;// Índice do Box que contem a marca
m_MarkAtrib = MyMarkAtrib;// Atributo da Marca

// Constantes de igualdade:
m_EqualNumConstant=0;
m_EqualStringConstant=MyEqualConstant;
// Variáveis de igualdade:
m_EqualGateIndex=-1;
m_EqualMarkBoxIndex=-1;
```

```
m_EqualMarkBoxAtrib=-1;
}

CAttribution::CAttribution(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualBoxIndex, long
MyEqualAtribute)
{
ASSERT((MyType==EQUAL_TO_MARKATRIB));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);
ASSERT(MyEqualBoxIndex>=0);
ASSERT(MyEqualAtribute>=0);

m_Type=MyType;// Tipo de Atribuição
m_ThisMarkBoxIndex = MyMarkBox;// Índice do Box que contem a marca
m_MarkAtrib = MyMarkAtrib;// Atributo da Marca

// Constantes de igualdade:
m_EqualNumConstant=0;
m_EqualStringConstant="";
// Variáveis de igualdade:
m_EqualGateIndex=-1;
m_EqualMarkBoxIndex=MyEqualBoxIndex;
m_EqualMarkBoxAtrib=MyEqualAtribute;

}


// Build Functions:
void CAttribution::Create(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualConstantOrGateIndex)
{
ASSERT((MyType==EQUAL_TO_NUM_CTE)||(MyType==EQUAL_TO_GATE));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);

m_Type=MyType;// Tipo de Atribuição
m_ThisMarkBoxIndex = MyMarkBox;// Índice do Box que contem a marca
m_MarkAtrib = MyMarkAtrib;// Atributo da Marca
if (MyType==EQUAL_TO_NUM_CTE)
                        {
                        // Constantes de igualdade:
                        m_EqualNumConstant=MyEqualConstantOrGateIndex;
                        m_EqualStringConstant="";
                        // Variáveis de igualdade:
                        m_EqualGateIndex=-1;
                        m_EqualMarkBoxIndex=-1;
                        m_EqualMarkBoxAtrib=-1;
                        }
else
                        {
                        // Constantes de igualdade:
                        m_EqualNumConstant=0;
                        m_EqualStringConstant="";
                        // Variáveis de igualdade:
                        m_EqualGateIndex=MyEqualConstantOrGateIndex;
                        m_EqualMarkBoxIndex=-1;
                        m_EqualMarkBoxAtrib=-1;
                        }
}

void CAttribution::Create(short MyType,long MyMarkBox,long MyMarkAtrib,CString MyEqualConstant)
{
ASSERT((MyType==EQUAL_TO_STRING_CTE));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);
ASSERT(MyEqualConstant!="");

m_Type=MyType;// Tipo de Atribuição
m_ThisMarkBoxIndex = MyMarkBox;// Índice do Box que contem a marca
m_MarkAtrib = MyMarkAtrib;// Atributo da Marca

// Constantes de igualdade:
m_EqualNumConstant=0;
```

```
m_EqualStringConstant=MyEqualConstant;
// Variáveis de igualdade:
m_EqualGateIndex=-1;
m_EqualMarkBoxIndex=-1;
m_EqualMarkBoxAtrib=-1;
}

void CAttribution::Create(short MyType,long MyMarkBox,long MyMarkAtrib,long MyEqualBoxIndex, long
MyEqualAtribute)
{
ASSERT((MyType==EQUAL_TO_MARKATRIB));
ASSERT(MyMarkBox>=0);
ASSERT(MyMarkAtrib>=0);
ASSERT(MyEqualBoxIndex>=0);
ASSERT(MyEqualAtribute>=0);

m_Type=MyType;// Tipo de Atribuição
m_ThisMarkBoxIndex = MyMarkBox;// Índice do Box que contem a marca
m_MarkAtrib = MyMarkAtrib;// Atributo da Marca

// Constantes de igualdade:
m_EqualNumConstant=0;
m_EqualStringConstant="";
// Variáveis de igualdade:
m_EqualGateIndex=-1;
m_EqualMarkBoxIndex=MyEqualBoxIndex;
m_EqualMarkBoxAtrib=MyEqualAtribute;
}


// Data Management:
BOOL CAttribution::DoAttribution(CBoxArray* p_MyBoxArray,CGateArray* p_MyGateArray)
{
ASSERT((m_Type!=UNDEFINED_TYPE)&&((m_Type==EQUAL_TO_MARKATRIB)||(m_Type==EQUAL_TO_S
TRING_CTE)||(m_Type==EQUAL_TO_NUM_CTE)||(m_Type==EQUAL_TO_GATE)));
ASSERT ((p_MyBoxArray!=NULL)&&(p_MyGateArray!=NULL));
BOOL ReturnFlag = FALSE;

// find register:
CMarkAttribute TempAtrib = (p_MyBoxArray->GetAt(m_ThisMarkBoxIndex)).GetMarkAtrib(m_MarkAtrib);
if (!(p_MyBoxArray->GetAt(m_ThisMarkBoxIndex)).HasMark())
                {
                ReturnFlag = FALSE;
                ASSERT(FALSE);
                }

switch (m_Type)
        {
        case EQUAL_TO_NUM_CTE:
                {
                ASSERT(TempAtrib.GetType()==INTEGER_ATRIB);
                TempAtrib.SetAtrib(m_EqualNumConstant);
                ReturnFlag = TRUE;
                break;
                }
        case EQUAL_TO_STRING_CTE:
                {
                ASSERT(TempAtrib.GetType()==TEXT_ATRIB);
                TempAtrib.SetAtrib(m_EqualStringConstant);
                ReturnFlag = TRUE;
                break;
                }
        case EQUAL_TO_GATE:
                {
                if (TempAtrib.GetType()==INTEGER_ATRIB)
                        {
                        CGate TempGate = (p_MyGateArray-
>GetAt(m_EqualGateIndex));
                        TempAtrib.SetAtrib(TempGate.GetInteger());
                        ReturnFlag = TRUE;
                        }
                if (TempAtrib.GetType()==TEXT_ATRIB)
```

```
                                                {
                                                CGate TempGate = (p_MyGateArray-
>GetAt(m_EqualGateIndex));
                                                TempAtrib.SetAtrib(TempGate.GetString());
                                                ReturnFlag = TRUE;
                                                }
                                        break;
                                        }

                        case EQUAL_TO_MARKATRIB:
                                        {
                                        ASSERT(m_EqualMarkBoxIndex>=0);
                                        if (p_MyBoxArray->GetAt(m_EqualMarkBoxIndex).HasMark())
                                                {
                                                CMarkAttribute RightAtrib = p_MyBoxArray-
>GetAt(m_EqualMarkBoxIndex).GetMarkAtrib(m_EqualMarkBoxAtrib);
                                                ASSERT(TempAtrib.GetType()==RightAtrib.GetType());
                                                if (RightAtrib.GetType()==INTEGER_ATRIB)
                                                        {

        TempAtrib.SetAtrib(RightAtrib.GetIntegerAtrib());
                                                        ReturnFlag = TRUE;
                                                        }
                                                if (RightAtrib.GetType()==TEXT_ATRIB)
                                                        {

        TempAtrib.SetAtrib(RightAtrib.GetTextAtrib());
                                                        ReturnFlag = TRUE;
                                                        }
                                                }
                                        else
                                                {
                                                if (TempAtrib.GetType()==INTEGER_ATRIB)
                                                        {
                                                        TempAtrib.SetAtrib(0);
                                                        ReturnFlag = TRUE;
                                                        }
                                                if (TempAtrib.GetType()==TEXT_ATRIB)
                                                        {
                                                        TempAtrib.SetAtrib("");
                                                        ReturnFlag = TRUE;
                                                        }
                                                }
                                        break;
                                        }
                        };
                        if (ReturnFlag)
                                        {
                                        (p_MyBoxArray->ElementAt(m_ThisMarkBoxIndex)).SetMarkAtribute(TempAtrib);
                                        }
                        return ReturnFlag;
        }

//serialização
 IMPLEMENT_SERIAL (CAttribution,CObject, 1);
 void CAttribution::Serialize( CArchive& ar )
        {
        ASSERT (FALSE);
        }

        // Implementar Construtor de Cópia e Operador Igual
        CAttribution::CAttribution(const CAttribution& AtribSrc)
        {
        // Tipo de Atribuição
        m_Type=AtribSrc.m_Type;
        // Índice do Box que contem a marca
        m_ThisMarkBoxIndex = AtribSrc.m_ThisMarkBoxIndex;
        // Atributo da Marca
        m_MarkAtrib = AtribSrc.m_MarkAtrib;
        // Constantes de igualdade:
        m_EqualNumConstant=AtribSrc.m_EqualNumConstant;
        m_EqualStringConstant=AtribSrc.m_EqualStringConstant;
```

```
        // Variáveis de igualdade:
        m_EqualGateIndex=AtribSrc.m_EqualGateIndex;
        m_EqualMarkBoxIndex=AtribSrc.m_EqualMarkBoxIndex;
        m_EqualMarkBoxAtrib=AtribSrc.m_EqualMarkBoxAtrib;
        }

        const CAttribution& CAttribution::operator =( const CAttribution& AtribSrc )
        {
        // Tipo de Atribuição
        m_Type=AtribSrc.m_Type;
        // Índice do Box que contem a marca
        m_ThisMarkBoxIndex = AtribSrc.m_ThisMarkBoxIndex;
        // Atributo da Marca
        m_MarkAtrib = AtribSrc.m_MarkAtrib;
        // Constantes de igualdade:
        m_EqualNumConstant=AtribSrc.m_EqualNumConstant;
        m_EqualStringConstant=AtribSrc.m_EqualStringConstant;
        // Variáveis de igualdade:
        m_EqualGateIndex=AtribSrc.m_EqualGateIndex;
        m_EqualMarkBoxIndex=AtribSrc.m_EqualMarkBoxIndex;
        m_EqualMarkBoxAtrib=AtribSrc.m_EqualMarkBoxAtrib;
        return *this;
        }



////////////////////////////////////////////////////////////////////////////
//CConditional Object Implementation
//eMFG Conditional Object
//version 2.0
//Definition date:06/07/98
//Last Modified:06/07/98
//Programmed by: Marco A. A. Silva
//Last changed by: Marco A. A. Silva

//Construtor Vazio
//todas os condicionais devem ser inicializados usando Create
CConditional::CConditional()
{
        mp_Owner                = NULL;
        m_Logic                 = NULO;
        m_NOT                           = FALSE;
        m_LBox                  = NULO;
        m_LAttrib       = NULO;
        m_Operator      = NULO;
        m_RType                 = NULO;
        m_RIntCte               = NULO;
        m_RStrCte               = "";
        m_RBox                  = NULO;
        m_RAttrib       = NULO;
        m_RGate                 = NULO;
        mp_Right        = NULL;
        mp_Down                 = NULL;
}

//Função de inicialização Create
//observe exemplos de condicionais simples abaixo:

//(box!atributo<1)
//CConditional* cond1 = new CConditional;
//cond1->Create(NULO, myGraph,FALSE, box, atributo, LESSER, INT_CTE, 1);

//(~(box!atributo=='string'))
//CConditional* cond2 = new CConditional;
//cond2->Create(NULO, myGraph, TRUE, box, atributo, EQUAL, STR_CTE,NULO,NULO,"string");

//(box1!atributo>box2!atributo)
//CConditional* cond3 = new CConditional;
//cond3->Create(NULO, myGraph, FALSE, box1, atributo1, GREATER, INT_BOX_ATB, box2, atributo2);

//atributo string e gate string
```

```
//(box!atributo!=gate)
//CConditional* cond4 = new CConditional;
//cond4->Create(NULO, myGraph, TRUE, box, atributo, EQUAL, STR_GATE_VAL, gate);


//todas as checagens de tipo devem ser feitas pelo interpretador

void CConditional::Create (     short myLogic,

                                CGraph* myGraph,

                                BOOL myNegative,

                                short myLBox, short myLAttrib,

                                short myOperator,

                                short myRType, short myRParam1, short myRParam2, CString myRParam3 )
{
        m_Logic                 = myLogic;
        mp_Owner                = myGraph;
        m_NOT                   = myNegative;
        m_LBox                  = myLBox;
        m_LAttrib       = myLAttrib;
        m_Operator      = myOperator;
        m_RType                 = myRType;

        switch (m_RType)
                {
                case NULO:
                        break;
                case INT_CTE:
                        {
                        m_RIntCte = myRParam1;
                        break;
                        }
                case STR_CTE:
                        {
                        m_RStrCte = myRParam3;
                        break;
                        }
                case INT_BOX_ATB:
                        {
                        m_RBox          = myRParam1;
                        m_RAttrib = myRParam2;
                        break;
                        }
                case STR_BOX_ATB:
                        {
                        m_RBox          = myRParam1;
                        m_RAttrib = myRParam2;
                        break;
                        }
                case INT_GATE_VAL:
                        {
                        m_RGate         = myRParam1;
                        break;
                        }
                case STR_GATE_VAL:
                        {
                        m_RGate         = myRParam1;
                        break;
                        }
                };
}

//funções para composição da estrutura de avaliação
//((box1!atributo1==1)&&(~(box2!atributo2=='string'))
//CConditional* cond1 = new CConditional;
//CConditional* cond2 = new CConditional;
//CConditional* cond3 = new CConditional;
```

```
//cond1->Create (AND);
//cond2->Create (NULO, myGraph, FALSE, box1, atributo1, EQUAL, INT_CTE, 1);
//cond3->Create (NULO, myGraph, TRUE, box2, atributo2, EQUAL, STR_CTE, NULO, NULO, "string");
//cond1->AddRight(cond2);
//cond1->AddDown(cond3);

void CConditional::AddRight(CConditional& myRBrother)
{
 if (mp_Right!=NULL)
   {
   delete mp_Right;
   mp_Right=NULL;
   }
 mp_Right = new CConditional;
          *(mp_Right) = myRBrother;
}

void CConditional::AddDown(CConditional& myDDaughter)
{
 if (mp_Down!=NULL)
   {
   delete mp_Down;
   mp_Down=NULL;
   }
 mp_Down = new CConditional;
          *(mp_Down) = myDDaughter;
}

void CConditional::SetAlwaysTRUE()
{
 m_Logic = ALWAYS_TRUE;
}

void CConditional::SetAlwaysFALSE()
{
 m_Logic = ALWAYS_FALSE;
}

//função de avaliação
//BOOL success = cond1->Evaluate();
BOOL CConditional::Evaluate()
{
          BOOL result;                              //overall result

          BOOL brother;                             //left brother result
          BOOL daughter;                  //down daughter result

          long              RI, LI;                 //Integer R and L-Values
          CString    RS, LS;              //String    R and L-Values

          if (mp_Right!=NULL)
                    brother = mp_Right->Evaluate();

          if (mp_Down!=NULL)
                    daughter = mp_Down->Evaluate();

          switch (m_Logic)
                    {
                    case NULO:
                              {
                              switch (m_Operator)
                                        {
                                        case EQUAL:
                                                  {
                                                  switch (m_RType)
                                                            {
                                                            case INT_CTE:
                                                                      {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
```

```
        }
                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                    RI = m_RIntCte;
                                                    result = (LI==RI);
                                                    break;
                                                    }
                                                case STR_CTE:
                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                    LS = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetTextAtrib();
                                                    RS = m_RStrCte;
                                                    result = (LS==RS);
                                                    break;
                                                    }
                                                case INT_BOX_ATB:
                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
        if (!mp_Owner->GetBoxes()->GetAt(m_RBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                    RI = mp_Owner->GetBoxes()-
>GetAt(m_RBox).GetMarkAtrib(m_RAttrib).GetIntegerAtrib();
                                                    result = (LI==RI);
                                                    break;
                                                    }
                                                case STR_BOX_ATB:
                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
        if (!mp_Owner->GetBoxes()->GetAt(m_RBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                    LS = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetTextAtrib();
                                                    RS = mp_Owner->GetBoxes()-
>GetAt(m_RBox).GetMarkAtrib(m_RAttrib).GetTextAtrib();
                                                    result = (LS==RS);
                                                    break;
                                                    }
                                                case INT_GATE_VAL:
                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                    RI = mp_Owner->GetGates()-
>GetAt(m_RGate).GetInteger();
                                                    result = (LI==RI);
                                                    break;
                                                    }
```

```
                                                    case STR_GATE_VAL:
                                                                {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                LS = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetTextAtrib();
                                                                RS = mp_Owner->GetGates()-
>GetAt(m_RGate).GetString();
                                                                result = (LS==RS);
                                                                break;
                                                                }
                                                    };
                                        break;
                                        }
                                case GREATER:
                                        {
                                        switch (m_RType)
                                                {
                                                case INT_CTE:
                                                                {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                RI = m_RIntCte;
                                                                result = (LI>RI);
                                                                break;
                                                                }
                                                case INT_BOX_ATB:
                                                                {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
        if (!mp_Owner->GetBoxes()->GetAt(m_RBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                RI = mp_Owner->GetBoxes()-
>GetAt(m_RBox).GetMarkAtrib(m_RAttrib).GetIntegerAtrib();
                                                                result = (LI>RI);
                                                                break;
                                                                }
                                                case INT_GATE_VAL:
                                                                {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                RI = mp_Owner->GetGates()-
>GetAt(m_RGate).GetInteger();
                                                                result = (LI>RI);
                                                                break;
                                                                }
                                                };
                                        break;
                                        }
                                case LESSER:
                                        {
```

```
                                                    switch (m_RType)
                                                            {
                                                            case INT_CTE:
                                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                    RI = m_RIntCte;
                                                                    result = (LI<RI);
                                                                    break;
                                                                    }
                                                            case INT_BOX_ATB:
                                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
        if (!mp_Owner->GetBoxes()->GetAt(m_RBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                    RI = mp_Owner->GetBoxes()-
>GetAt(m_RBox).GetMarkAtrib(m_RAttrib).GetIntegerAtrib();
                                                                    result = (LI<RI);
                                                                    break;
                                                                    }
                                                            case INT_GATE_VAL:
                                                                    {
        if (!mp_Owner->GetBoxes()->GetAt(m_LBox).HasMark())
         {
         result = FALSE;
         break;
         }
                                                                    LI = mp_Owner->GetBoxes()-
>GetAt(m_LBox).GetMarkAtrib(m_LAttrib).GetIntegerAtrib();
                                                                    RI = mp_Owner->GetGates()-
>GetAt(m_RGate).GetInteger();
                                                                    result = (LI<RI);
                                                                    break;
                                                                    }
                                                            };
                                                    break;
                                                    }
                                            };
                                    break;
                                    }
  case ALWAYS_TRUE:
   {
   return (TRUE);
   break;
   }
  case ALWAYS_FALSE:
   {
   return (FALSE);
   break;
   }
                        case AND:
                                {
                                result = (brother&&daughter);
                                break;
                                }
                        case OR:
                                {
                                result = (brother||daughter);
```

```
                                break;
                                }
                        };

        if (m_NOT)
                return !(result);
        else
                return (result);
}

// Implementar Construtor de Cópia e Operador Igual
CConditional::CConditional(const CConditional& CondSrc)
{
        m_Logic=CondSrc.m_Logic;                        //Node Logical Operator
        m_NOT=CondSrc.m_NOT;                            //Node Not Operator
        m_LBox=CondSrc.m_LBox;                          //L-Value Box
        m_LAttrib=CondSrc.m_LAttrib;        //L-Value Attribute
        m_Operator=CondSrc.m_Operator;      //operator type
        m_RType=CondSrc.m_RType;                        //R-Value type
        m_RIntCte=CondSrc.m_RIntCte;                    //R-Value Integer Constant
        m_RStrCte=CondSrc.m_RStrCte;                    //R-Value String Constant
        m_RBox=CondSrc.m_RBox;                          //R-Value Box
        m_RAttrib=CondSrc.m_RAttrib;        //R-Value Attribute
        m_RGate=CondSrc.m_RGate;                        //R-Value Gate

 mp_Owner=CondSrc.mp_Owner;     //CGraph pointer

 mp_Right = NULL;
 mp_Right = NULL;

 if (CondSrc.mp_Right!=NULL)
   mp_Right = new CConditional (*CondSrc.mp_Right);
 else
   mp_Right = NULL;

 if (CondSrc.mp_Down!=NULL)
   mp_Down = new CConditional (*CondSrc.mp_Down);
 else
   mp_Down = NULL;
}

const CConditional& CConditional::operator =( const CConditional& CondSrc )
{
        m_Logic=CondSrc.m_Logic;                        //Node Logical Operator
        m_NOT=CondSrc.m_NOT;                            //Node Not Operator
        m_LBox=CondSrc.m_LBox;                          //L-Value Box
        m_LAttrib=CondSrc.m_LAttrib;        //L-Value Attribute
        m_Operator=CondSrc.m_Operator;      //operator type
        m_RType=CondSrc.m_RType;                        //R-Value type
        m_RIntCte=CondSrc.m_RIntCte;                    //R-Value Integer Constant
        m_RStrCte=CondSrc.m_RStrCte;                    //R-Value String Constant
        m_RBox=CondSrc.m_RBox;                          //R-Value Box
        m_RAttrib=CondSrc.m_RAttrib;        //R-Value Attribute
        m_RGate=CondSrc.m_RGate;                        //R-Value Gate

 mp_Owner=CondSrc.mp_Owner;     //CGraph pointer

 if (CondSrc.mp_Right!=NULL)
   {
   if (mp_Right!=NULL)
     delete mp_Right;
   mp_Right = new CConditional (*CondSrc.mp_Right);
   }
 else
   mp_Right = NULL;

 if (CondSrc.mp_Down!=NULL)
   {
   if (mp_Down!=NULL)
     delete mp_Down;
   mp_Down = new CConditional (*CondSrc.mp_Down);
   }
```

```
  else
    mp_Down = NULL;

  return *this;
}


//destrutor funciona recursivamente
//liberando toda a estrutura
CConditional::~CConditional()
{
        if (mp_Right!=NULL)
                delete mp_Right;
        if (mp_Down!=NULL)
                delete mp_Down;
}
```

Files Arcfilt.h and Arcfilt.cpp - E-MFG Arc's Filters

```
// arcfilt.h - eMFG Arc Filter Definition:
// version: 2.0
// Definition Date: 08/07/98
// Last Modified:18/07/98
// programed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira


// Objetivos: Definir a estrutura de dados dos filtros de
// atributos de marca dos elementos de conexão
// do grafo (arcos direcionais)


#ifndef ARCFILTERINCLUDED
class CArcFilter : public CObject
{
protected:
  class CGraph* mp_Owner;                       // Owner Graph
        short m_Type;           // Tipo de Filtro
        long m_AttribNumber;    // Número de atributos
        long* mp_AttribIndex;   // Índices dos atributos

  short m_CompositeFlag;    // Flag para passagem ou não de marcas compostas

public:
        // Default Construction & Destruction
        CArcFilter();
        ~CArcFilter();


        // Build Functions

  void Create(class CGraph* MyOwner, short MyType, long MyAttribNumber, long* MyAttribIndex, short MyCompositeFlag);


//      CMark AplyFilter(CMark& SrcMark);
        CMark ApplyFilter(CMark& SrcMark);

        CArcFilter( const CArcFilter& FilterSrc );
  CArcFilter& operator = (const CArcFilter& FilterSrc);
};
#define ARCFILTERINCLUDED 1
#endif

// arcfilt.cpp - eMFG Arc Filter Implementation:
// version: 2.0
// Definition Date: 08/07/98
// Last Modified:18/07/98
// programed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira


// Objetivos: Implementar a estrutura de dados dos filtros de
// atributos de marca dos elementos de conexão
```

```
// do grafo (arcos direcionais)

#include "stdafx.h"                    // standard windows aplication

#include "elements.h"   //E-MFG Standard Elements

#include"arcfilt.h"    // E-MFG Arc Filter Definition

//#define PASS_ALL 0;
//#define NOT_PASS_ALL 1;
//#define PASS 2;
//#define NOT_PASS 3;


        // Default Construction & Destruction
        CArcFilter::CArcFilter()
{
mp_Owner=NULL;                         // Owner Graph
        m_Type=PASS_ALL;   // Tipo de Filtro
        m_AttribNumber=0;  // Número de atributos
        mp_AttribIndex=NULL;// Índices dos atributos
m_CompositeFlag = PASS_COMPOSITE; // Flag para marcas compostas
}

        CArcFilter::~CArcFilter()
{
if (mp_AttribIndex!=NULL)
   {
   delete []mp_AttribIndex;
   mp_AttribIndex = NULL;
   }
}


        // Build Functions

 void CArcFilter::Create(class CGraph* MyOwner, short MyType, long MyAttribNumber, long* MyAttribIndex, short
MyCompositeFlag)
 {
 ASSERT(MyOwner!=NULL);
 ASSERT(
     (MyType==PASS_ALL)||
     (MyType==PASS)||
     (MyType==NOT_PASS_ALL)||
     (MyType==NOT_PASS)
     );
 mp_Owner=MyOwner;                                    // Owner Graph
        m_Type=MyType;              // Tipo de Filtro
        m_AttribNumber=MyAttribNumber;   // Número de atributos
 if (MyAttribNumber>0&&MyAttribIndex!=NULL)
  {
  mp_AttribIndex = new long[MyAttribNumber];
  for (long index = 0;index<MyAttribNumber; index++)
            {
   mp_AttribIndex[index]=MyAttribIndex[index];    // Índices dos atributos
   }
  }
 else
  {
  mp_AttribIndex = NULL;
  }

ASSERT((MyCompositeFlag == PASS_COMPOSITE)||(MyCompositeFlag == NOT_PASS_COMPOSITE));
m_CompositeFlag = MyCompositeFlag;

}


        CMark CArcFilter::ApplyFilter(CMark& SrcMark)
{
CMark ReturnMark = SrcMark;
switch (m_Type)
```

```
  {
  case PASS_ALL:
   {
   // Do Nothing
   break;
   }
  case NOT_PASS_ALL:
   {
   // Sets All Attributes to NULL
   ReturnMark.SetAllAttributesToNull();
   break;
   }
  case PASS:
   {
   ASSERT(m_AttribNumber>0);
   ASSERT(mp_AttribIndex!=NULL);
   for (long index = 0; index<mp_Owner->GetAttribTemplates()->GetSize(); index++)
    {
    // for all attributes verify if they are in the PASS list
    BOOL Keep = FALSE;
    for (long k=0;k<m_AttribNumber;k++)
     {
     if (index==mp_AttribIndex[k])
      {
      Keep = TRUE;
      }
     }
     if (!Keep)
      {
      // if they aren't in the PASS list Set them to NULL
      ReturnMark.SetAttribToNull(index);
      }
    }
   break;
   }
  case NOT_PASS:
   {
   ASSERT(m_AttribNumber>0);
   ASSERT(mp_AttribIndex!=NULL);
   for (long index = 0; index<m_AttribNumber; index++)
    {
    ASSERT(mp_AttribIndex[index]<mp_Owner->GetAttribTemplates()->GetSize());
    ReturnMark.SetAttribToNull(mp_AttribIndex[index]);
    }
   break;
   }
  }

switch (m_CompositeFlag)
 {
 case PASS_COMPOSITE:
  {
  // Do Nothing
  break;
  }
 case NOT_PASS_COMPOSITE:
  {
  // Sets All Attributes to NULL
  ReturnMark.ResetComposition();
  break;
  }
 };

return ReturnMark;
}

        CArcFilter::CArcFilter( const CArcFilter& FilterSrc )
{
mp_Owner=FilterSrc.mp_Owner;                                    // Owner Graph
        m_Type=FilterSrc.m_Type;            // Tipo de Filtro
        m_AttribNumber=FilterSrc.m_AttribNumber;    // Número de atributos
if (FilterSrc.m_AttribNumber>0&&FilterSrc.mp_AttribIndex!=NULL)
```

```
    {
    mp_AttribIndex = new long[FilterSrc.m_AttribNumber];
    for (long index = 0;index<FilterSrc.m_AttribNumber; index++)
                {
     mp_AttribIndex[index]=FilterSrc.mp_AttribIndex[index];    // Índices dos atributos
     }
    }
   else
    {
    mp_AttribIndex = NULL;
    }
   m_CompositeFlag = FilterSrc.m_CompositeFlag;
   }

   CArcFilter& CArcFilter::operator = (const CArcFilter& FilterSrc)
   {
   mp_Owner=FilterSrc.mp_Owner;                                          // Owner Graph
              m_Type=FilterSrc.m_Type;              // Tipo de Filtro
              m_AttribNumber=FilterSrc.m_AttribNumber;   // Número de atributos
   if (mp_AttribIndex!=NULL)
      {
      delete []mp_AttribIndex;
      mp_AttribIndex = NULL;
      }

   if (FilterSrc.m_AttribNumber>0&&FilterSrc.mp_AttribIndex!=NULL)
    {
    mp_AttribIndex = new long[FilterSrc.m_AttribNumber];
    for (long index = 0;index<FilterSrc.m_AttribNumber; index++)
                {
     mp_AttribIndex[index]=FilterSrc.mp_AttribIndex[index];    // Índices dos atributos
     }
    }
   else
    {
    mp_AttribIndex = NULL;
    }
   m_CompositeFlag = FilterSrc.m_CompositeFlag;
   return *this;
   }
```

Files Array.h and Array.cpp - Structural Elements Arrays

```
// CLinkedList Extension Classes
// Defined at : 23/07/98
// Last Modified : 24/07/98
// Programmed by : Marco A. A. Silva
// Modified by : Marco A. A. Silva

//CLinesArray
//CLinkedList Extension Class
//array of CWordsArray
//
#ifndef ARRAYS_INCLUDED

class CLinesArray : public CLinkedList
{
public:
            CLinesArray();
            ~CLinesArray();

public:
   void AddOrdered(CWordsArray StrArrArg, short col);
   void Add(CWordsArray StrArrArg);
            CWordsArray  GetAt(int n);
            CWordsArray& ElementAt(int n);
};
```

```
//CMarkAttribArray
//CLinkedList Extension Class
//array of CMarkAttribute
//
class CMarkAttribArray : public CLinkedList
{
public:
        CMarkAttribArray();
        ~CMarkAttribArray();

public:
        void Add(CMarkAttribute Arg);
        CMarkAttribute  GetAt(int n);
        CMarkAttribute& ElementAt(int n);
};

//CArcArray
//CLinkedList Extension Class
//array of CArc
//
class CArcArray : public CLinkedList
{
public:
        CArcArray();
        ~CArcArray();

public:
        void Add(CArc Arg);
        CArc  GetAt(int n);
        CArc& ElementAt(int n);
};

//CGateArray
//CLinkedList Extension Class
//array of CGate
//
class CGateArray : public CLinkedList
{
public:
        CGateArray();
        ~CGateArray();

public:
        void Add(CGate Arg);
        CGate  GetAt(int n);
        CGate& ElementAt(int n);
};

//CTransitionArray
//CLinkedList Extension Class
//array of CTransition
//
class CTransitionArray : public CLinkedList
{
public:
        CTransitionArray();
        ~CTransitionArray();

public:
        void Add(CTransition Arg);
        CTransition  GetAt(int n);
        CTransition& ElementAt(int n);
};

//CBoxArray
//CLinkedList Extension Class
//array of CBox
//
class CBoxArray : public CLinkedList
{
public:
```

```
            CBoxArray();
            ~CBoxArray();

public:
            void Add(CBox Arg);
            CBox  GetAt(int n);
            CBox& ElementAt(int n);
};

//CMarkArray
//CLinkedList Extension Class
//array of CMark
//
class CMarkArray : public CLinkedList
{
public:
            CMarkArray();
            ~CMarkArray();

public:
            void Add(CMark Arg);
            CMark  GetAt(int n);
            CMark& ElementAt(int n);
};

//CCondArray
//CLinkedList Extension Class
//array of CConditional
//
class CCondArray : public CLinkedList
{
public:
            CCondArray();
            ~CCondArray();
 CCondArray::CCondArray(const CCondArray& arg):/// Daniel
 CCondArray& operator =(const CCondArray& arg);

public:
            void Add(CConditional& Arg);
            CConditional  GetAt(int n);
            CConditional& ElementAt(int n);
};

//CAttrbnArray
//CLinkedList Extension Class
//array of CAttribution
//
class CAttrbnArray : public CLinkedList
{
public:
            CAttrbnArray();
            ~CAttrbnArray();

            CAttrbnArray(const CAttrbnArray& arg);
            CAttrbnArray& operator =(CAttrbnArray& arg);

public:
            void Add(CAttribution Arg);
            CAttribution  GetAt(int n);
            CAttribution& ElementAt(int n);
};

//CIntArray
//CLinkedList Extension Class
//array of long
//
class CIntArray : public CLinkedList
{
public:
            CIntArray();
            ~CIntArray();
```

```
public:
        void  Add(short Arg);
        short  GetAt(int n);
        short& ElementAt(int n);
};


//CAtbnMatrix
//CLinkedList Extension Class
//List of Atribution Arrays
class CAtbnMatrix : public CLinkedList
{
public:
 CAtbnMatrix();
 ~CAtbnMatrix();
 CAtbnMatrix(const CAtbnMatrix& arg);//Daniel
 CAtbnMatrix& operator =(const CAtbnMatrix& arg);

public:
        void Add(CAttrbnArray Arg);
        CAttrbnArray  GetAt(int n);
        CAttrbnArray& ElementAt(int n);
};

#endif

#define ARRAYS_INCLUDED 1
// CLinkedList Extension Classes
// Defined at : 23/07/98
// Last Modified : 24/07/98
// Programmed by : Marco A. A. Silva
// Modified by : Marco A. A. Silva

#pragma warning(4270:disable)

#include "stdafx.h"

#include "definitions.h"                    //E-MFG Constant Definitions
#include "wordsarray.h"                     //Extension of String Arrays
#include "basis.h"                          //E-MFG Basis Class
#include "graph.h"                          //E-MFG Graph Components
#include "marks.h"                          //E-MFG Marks Definition
#include "arcs.h"                           //E-MFG Arcs Definition
#include "boxes.h"                          //E-MFG Boxes Definition
#include "transit.h"                        //E-MFG Transition Definition
#include "gates.h"                          //E-MFG Gates Definition
#include "list.h"                           //Linked List Definition

#include "arrays.h"
//////////////////////////////////////////////////////////////////////
//CLinesArray
//CLinkedList Extension Class
//array of CWordsArray
//
CLinesArray::CLinesArray()
{
}

CLinesArray::~CLinesArray()
{
}

void CLinesArray::AddOrdered(CWordsArray StrArrArg, short col)
{
        CNode* NewNode;
        CNode* cursor;
  CNode* previous;

  CString key1, key2;

  //não insere ainda
  BOOL INSERT = FALSE;
```

```
    //cria nova string array baseada no argumento
    CWordsArray* NewString = new CWordsArray (StrArrArg);

    //cria novo nódulo apontando para nova string array
    NewNode = new CNode(STRING_NODE);
            NewNode->mp_StringArray = NewString;

    //coloca cursor na sentinela
            cursor  = mp_Sentry;
    previous = mp_Sentry;

    //enquanto não for último elemento ou hora de inserir
            while ((cursor->mp_NextNode!=NULL)&&(!INSERT))
     {
     //guarda o último
     previous = cursor;
     //avança na lista
     cursor = cursor->mp_NextNode;
     //compara as chaves
     key1 = cursor->mp_StringArray->GetAt(col);
     key2 = NewString->GetAt(col);
     if (key1>key2)
       INSERT = TRUE; //hora de inserir
     }

    //acresce de um elemento
            m_n++;

    if (INSERT)
     {
     previous->mp_NextNode = NewNode;
     NewNode->mp_NextNode  = cursor;
     }
    else
     {
     //no final da lista
     cursor->mp_NextNode  = NewNode;
     }
}

void CLinesArray::Add(CWordsArray StrArrArg)
{
            AddString(StrArrArg);
}


CWordsArray  CLinesArray::GetAt(int n)
{
            return(GetStringAt(n));
}

CWordsArray& CLinesArray::ElementAt(int n)
{
            return(StringElementAt(n));
}

////////////////////////////////////////////////////////////////////////////
//CMarkAttribArray
//CLinkedList Extension Class
//array of CMarkAttribute
//
CMarkAttribArray::CMarkAttribArray()
{
}

CMarkAttribArray::~CMarkAttribArray()
{
}

void CMarkAttribArray::Add(CMarkAttribute Arg)
{
```

```
        AddAttribute(Arg);
}


CMarkAttribute  CMarkAttribArray::GetAt(int n)
{
        return(GetAttributeAt(n));
}

CMarkAttribute& CMarkAttribArray::ElementAt(int n)
{
        return(AttributeElementAt(n));
}

////////////////////////////////////////////////////////////////////
//CArcArray
//CLinkedList Extension Class
//array of CArc
//
CArcArray::CArcArray()
{
}

CArcArray::~CArcArray()
{
}

void CArcArray::Add(CArc Arg)
{
        AddArc(Arg);
}


CArc  CArcArray::GetAt(int n)
{
        return(GetArcAt(n));
}

CArc& CArcArray::ElementAt(int n)
{
        return(ArcElementAt(n));
}
////////////////////////////////////////////////////////////////////
//CGateArray
//CLinkedList Extension Class
//array of CGate
//
CGateArray::CGateArray()
{
}

CGateArray::~CGateArray()
{
}

void CGateArray::Add(CGate Arg)
{
        AddGate(Arg);
}


CGate  CGateArray::GetAt(int n)
{
        return(GetGateAt(n));
}

CGate& CGateArray::ElementAt(int n)
{
        return(GateElementAt(n));
}

////////////////////////////////////////////////////////////////////
```

```
//CTransitionArray
//CLinkedList Extension Class
//array of CTransition
//
CTransitionArray::CTransitionArray()
{
}

CTransitionArray::~CTransitionArray()
{
}

void CTransitionArray::Add(CTransition Arg)
{
        AddTransit(Arg);
}


CTransition CTransitionArray::GetAt(int n)
{
        return(GetTransitAt(n));
}

CTransition& CTransitionArray::ElementAt(int n)
{
        return(TransitElementAt(n));
}

////////////////////////////////////////////////////////////////////////
//CBoxArray
//CLinkedList Extension Class
//array of CBox
//
CBoxArray::CBoxArray()
{
}

CBoxArray::~CBoxArray()
{
}

void CBoxArray::Add(CBox Arg)
{
        AddBox(Arg);
}


CBox  CBoxArray::GetAt(int n)
{
        return(GetBoxAt(n));
}

CBox& CBoxArray::ElementAt(int n)
{
        return(BoxElementAt(n));
}

////////////////////////////////////////////////////////////////////////
//CArray
//CLinkedList Extension Class
//array of CMark
//
CMarkArray::CMarkArray()
{
}

CMarkArray::~CMarkArray()
{
}

void CMarkArray::Add(CMark Arg)
{
```

```
            AddMark(Arg);
}

CMark  CMarkArray::GetAt(int n)
{
            return(GetMarkAt(n));
}

CMark& CMarkArray::ElementAt(int n)
{
            return(MarkElementAt(n));
}

////////////////////////////////////////////////////////////////////////
//CCondArray
//CLinkedList Extension Class
//array of CConditional
//
CCondArray::CCondArray()
{
}

CCondArray::~CCondArray()
{
}
// Daniel
CCondArray::CCondArray(const CCondArray& arg)
{
 short i,j;
 CNode* cursor;
 for (i=0;i<arg.m_n;i++)
  {
  cursor = arg.mp_Sentry;//daniel
  for (j=0;j<=i;j++)
   cursor = cursor->mp_NextNode;
            ASSERT(cursor->m_type==CONDS_NODE);
  AddConditional(*(cursor->mp_CondsArray));
  }
}

CCondArray& CCondArray::operator =(const CCondArray& arg)
{
 short i,j;
 CNode* cursor;

 if (&arg!=this)
  {
  RemoveAll();
  for (i=0;i<arg.m_n;i++)
    {
    cursor = arg.mp_Sentry;//daniel
    for (j=0;j<=i;j++)
     cursor = cursor->mp_NextNode;
              ASSERT(cursor->m_type==CONDS_NODE);
    AddConditional(*(cursor->mp_CondsArray));
    }
  }
 return *(this);
}


void CCondArray::Add(CConditional& Arg)
{
            AddConditional(Arg);
}

CConditional  CCondArray::GetAt(int n)
{
            return(GetConditionalAt(n));
}

CConditional& CCondArray::ElementAt(int n)
```

```
{
        return(ConditionalElementAt(n));
}

////////////////////////////////////////////////////////////////////
//CIntArray
//CLinkedList Extension Class
//array of long
//
CIntArray::CIntArray()
{
}

CIntArray::~CIntArray()
{
}

void CIntArray::Add(short Arg)
{
        AddInteger(Arg);
}

short CIntArray::GetAt(int n)
{
        return(GetIntegerAt(n));
}

short& CIntArray::ElementAt(int n)
{
        return(IntegerElementAt(n));
}

////////////////////////////////////////////////////////////////////
//CAttrbnArray
//CLinkedList Extension Class
//array of CAttribution
//
CAttrbnArray::CAttrbnArray()
{
}

CAttrbnArray::~CAttrbnArray()
{
}

CAttrbnArray::CAttrbnArray(const CAttrbnArray& arg)
{
 RemoveAll();
 CNode* myNode;
 if (arg.m_n>0)
   myNode = arg.mp_Sentry->mp_NextNode;
 for (short i=0;i<arg.m_n;i++)
   {
   AddAttribution(*(myNode->mp_AttribArray));
   myNode = myNode->mp_NextNode;
   }
}

CAttrbnArray& CAttrbnArray::operator =(CAttrbnArray& arg)
{
 if (&arg!=this)
   {
   RemoveAll();
   CNode* myNode;
   if (arg.m_n>0)
     myNode = arg.mp_Sentry->mp_NextNode;
   for (short i=0;i<arg.m_n;i++)
     {
     AddAttribution(*(myNode->mp_AttribArray));
     myNode = myNode->mp_NextNode;
     }
```

```
    }
  return *(this);
}

void CAttrbnArray::Add(CAttribution Arg)
{
        AddAttribution(Arg);
}

CAttribution  CAttrbnArray::GetAt(int n)
{
        return(GetAttributionAt(n));
}

CAttribution& CAttrbnArray::ElementAt(int n)
{
        return(AttributionElementAt(n));
}

//////////////////////////////////////////////////////////////////////
//CAttrbnMatrix
//CLinkedList Extension Class
//List of Atribution Arrays
//
CAtbnMatrix::CAtbnMatrix()
{
}

CAtbnMatrix::~CAtbnMatrix()
{
}

CAtbnMatrix::CAtbnMatrix(const CAtbnMatrix& arg)// Daniel
{
  CNode* cursor;
  for (short i=0;i<arg.m_n;i++)
   {
   cursor = arg.mp_Sentry;//daniel
   for (short j=0;j<=i;j++)
    cursor = cursor->mp_NextNode;
   ASSERT(cursor->m_type==ATTRIARRAY_NODE);
   AddAttribArray(*(cursor->mp_AtbnMatrix));
   }
}

CAtbnMatrix& CAtbnMatrix::operator =(const CAtbnMatrix& arg)
{
 if (&arg!=this)
   {
   CNode* cursor;
   RemoveAll();
   for (short i=0;i<arg.m_n;i++)
    {
    cursor = arg.mp_Sentry;//daniel
    for (short j=0;j<=i;j++)
      cursor = cursor->mp_NextNode;
    ASSERT(cursor->m_type==ATTRIARRAY_NODE);
    AddAttribArray(*(cursor->mp_AtbnMatrix));
    }
   }
 return *(this);
}


void CAtbnMatrix::Add(CAttrbnArray Arg)
{
        AddAttribArray(Arg);
}

CAttrbnArray  CAtbnMatrix::GetAt(int n)
{
 return(GetAttribArrayAt(n));
```

```
}

CAttrbnArray& CAtbnMatrix::ElementAt(int n)
{
        return(AttribArrayElementAt(n));
}
```

Files List.h and List.cpp - CLinkedList Node and Implementation

```
//Linked List Definition
//version 1.0
//Definition Date: 19/06/98
//Last Modification: 20/06/98
//Programmed by : Marco A. A. Silva
//Last modified by : Marco A. A. Silva

//Definir estrutura alternativa para arrays

class CNode : public CObject
{
public:
        CNode(long MyType = NULO);
        ~CNode();
public:
        int m_type;
        CNode* mp_NextNode;

        CWordsArray*            mp_StringArray;
        CMarkAttribute*     mp_MarkAtArray;
        CArc*                                           mp_ArcsArray;
 CBox*                                          mp_BoxesArray;
        CGate*                                  mp_GatesArray;
        CTransition*            mp_TransArray;
        CMark*                                  mp_MarksArray;
 CConditional*  mp_CondsArray;
 CAttribution*  mp_AttribArray;
 short*       mp_IntArray;

 class CAttrbnArray* mp_AtbnMatrix;
};


class CLinkedList: public CObject
{
public:
        CLinkedList();
        ~CLinkedList();

public:
        int GetSize();
 void RemoveAll();

//CWordsArray Array
public:
        void AddString(CWordsArray StrArrArg);
        CWordsArray  GetStringAt(int n);
        CWordsArray& StringElementAt(int n);

//CMarkAttribute Array
public:
        void AddAttribute(CMarkAttribute AttributeArg);
        CMarkAttribute  GetAttributeAt(int n);
        CMarkAttribute& AttributeElementAt(int n);

//CArcs Array
public:
        void AddArc(CArc ArcArg);
        CArc  GetArcAt(int n);
        CArc& ArcElementAt(int n);

//CBoxes Array
public:
```

```
            void AddBox(CBox BoxArg);
            CBox  GetBoxAt(int n);
            CBox& BoxElementAt(int n);

//CGates Array
public:
            void AddGate(CGate GateArg);
            CGate  GetGateAt(int n);
            CGate& GateElementAt(int n);

//CTransitions Array
public:
            void AddTransit(CTransition TransitArg);
            CTransition  GetTransitAt(int n);
            CTransition& TransitElementAt(int n);

//CMarks Array
public:
            void AddMark(CMark MarkArg);
            CMark  GetMarkAt(int n);
            CMark& MarkElementAt(int n);

            CMark  RemoveMarkAt(int n);

//CConditional Array
public:
            void AddConditional(CConditional& CondArg);
            CConditional  GetConditionalAt(int n);
            CConditional& ConditionalElementAt(int n);

//CAttribution Array
public:
            void AddAttribution(CAttribution AttrbArg);
            CAttribution  GetAttributionAt(int n);
            CAttribution& AttributionElementAt(int n);

//short Array
public:
            void   AddInteger(short IntArg);
            short  GetIntegerAt(int n);
            short& IntegerElementAt(int n);

//CAttribution Matrix
public:
            void AddAttribArray(class CAttrbnArray Arg);
            class CAttrbnArray  GetAttribArrayAt(int n);
            class CAttrbnArray& AttribArrayElementAt(int n);

public:
 long               m_n;
            CNode*  mp_Sentry;
};

//Linked List Implementation
//version 1.0
//Definition Date: 19/06/98
//Last Modification: 08/07/98
//Programmed by : Marco A. A. Silva
//Last modified by : Daniel M. S. Ferreira

//Implementar estrutura alternativa para arrays

#include <stdafx.h>

#include "definitions.h"                    //E-MFG Constant Definitions
#include "wordsarray.h"                     //Extension of String Arrays
#include "basis.h"                          //E-MFG Basis Class
#include "graph.h"                          //E-MFG Graph Components
#include "marks.h"                          //E-MFG Marks Definition
#include "arcfilt.h"      //E-MFG Arc Filter Definition
#include "arcs.h"                                   //E-MFG Arcs Definition
#include "boxes.h"                          //E-MFG Boxes Definition
```

```
#include "transit.h"                    //E-MFG Transition Definition
#include "gates.h"                      //E-MFG Gates Definition
#include "list.h"                       //Linked List Definition
#include "arrays.h"                     //Arrays Extension Class

CNode::CNode(long MyType)
{
        m_type = MyType;
        mp_NextNode=NULL;

        mp_StringArray  = NULL;
        mp_MarkAtArray  = NULL;
        mp_ArcsArray    = NULL;
        mp_BoxesArray   = NULL;
        mp_GatesArray   = NULL;
        mp_TransArray   = NULL;
 mp_MarksArray  = NULL;
 mp_CondsArray  = NULL;
 mp_AttribArray  = NULL;
 mp_IntArray    = NULL;
 mp_AtbnMatrix  = NULL;
}

CNode::~CNode()
{
 if (mp_StringArray!=NULL)
   delete mp_StringArray;

 if (mp_MarkAtArray!=NULL)
   delete mp_MarkAtArray;

 if (mp_MarksArray!=NULL)
   delete mp_MarksArray;

 if (mp_ArcsArray!=NULL)
   delete mp_ArcsArray;

 if (mp_BoxesArray!=NULL)
   delete mp_BoxesArray;

 if (mp_TransArray!=NULL)
   delete mp_TransArray;

 if (mp_GatesArray!=NULL)
   delete mp_GatesArray;

 if (mp_CondsArray!=NULL)
   delete mp_CondsArray;

 if (mp_AttribArray!=NULL)
   delete mp_AttribArray;

 if (mp_AtbnMatrix!=NULL)
   delete mp_AtbnMatrix;

 if (mp_IntArray!=NULL)
   delete mp_IntArray;
}

CLinkedList::CLinkedList()
{
        m_n = 0;
        mp_Sentry = new CNode(NULO);
}

CLinkedList::~CLinkedList()
{
        short i;
        CNode* next;
        CNode* trash;
        next = mp_Sentry;
        for (i=0; i<=m_n; i++)
```

```
                    {
                    trash = next;
                    next  = trash->mp_NextNode;
                    delete trash;
                    }
}

void CLinkedList::RemoveAll()
{
        short i;
        CNode* next;
        CNode* trash;
        next = mp_Sentry;
        for (i=0; i<=m_n; i++)
                {
                trash = next;
                next  = trash->mp_NextNode;
                delete trash;
                }

        m_n = 0;
        mp_Sentry = new CNode(NULO);
}


int CLinkedList::GetSize()
{
        return m_n;
}


//Array de CWordsArray
void CLinkedList::AddString(CWordsArray StrArrArg)
{
        CNode* NewNode;
        CNode* cursor;
        CWordsArray* NewString;
        NewString = new CWordsArray;
        *(NewString)=StrArrArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(STRING_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_StringArray = NewString;
}

CWordsArray CLinkedList::GetStringAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==STRING_NODE);

        return *(cursor->mp_StringArray);
}

CWordsArray& CLinkedList::StringElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;
```

```
	ASSERT(cursor->m_type==STRING_NODE);

	return *(cursor->mp_StringArray);
}


//Array de CMarkAttribute
void CLinkedList::AddAttribute(CMarkAttribute AttributeArg)
{
	CNode* NewNode;
	CNode* cursor;
	CMarkAttribute* NewAttribute;
	NewAttribute = new CMarkAttribute;
	*(NewAttribute)=AttributeArg;
	cursor = mp_Sentry;
	while (cursor->mp_NextNode!=NULL)
		cursor = cursor->mp_NextNode;
	m_n++;
	NewNode = new CNode(MARKAT_NODE);
	cursor->mp_NextNode = NewNode;
	NewNode->mp_MarkAtArray = NewAttribute;
}

CMarkAttribute CLinkedList::GetAttributeAt(int n)
{
	ASSERT (n>=0);
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==MARKAT_NODE);

	return *(cursor->mp_MarkAtArray);
}

CMarkAttribute& CLinkedList::AttributeElementAt(int n)
{
	ASSERT (n>=0);// Daniel
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==MARKAT_NODE);

	return *(cursor->mp_MarkAtArray);
}


//Array de CArc
void CLinkedList::AddArc(CArc ArcArg)
{
	CNode* NewNode;
	CNode* cursor;
	CArc* NewArc;
	NewArc = new CArc;
	*(NewArc)=ArcArg;
	cursor = mp_Sentry;
	while (cursor->mp_NextNode!=NULL)
		cursor = cursor->mp_NextNode;
	m_n++;
	NewNode = new CNode(ARC_NODE);
	cursor->mp_NextNode = NewNode;
	NewNode->mp_ArcsArray = NewArc;
}
```

```
CArc CLinkedList::GetArcAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ARC_NODE);

        return *(cursor->mp_ArcsArray);
}

CArc& CLinkedList::ArcElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ARC_NODE);

        return *(cursor->mp_ArcsArray);
}


//Array de CBox
void CLinkedList::AddBox(CBox BoxArg)
{
        CNode* NewNode;
        CNode* cursor;
        CBox* NewBox;
        NewBox = new CBox;
        *(NewBox)=BoxArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(BOX_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_BoxesArray = NewBox;
}

CBox CLinkedList::GetBoxAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==BOX_NODE);

        return *(cursor->mp_BoxesArray);
}

CBox& CLinkedList::BoxElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;
```

```
        ASSERT(cursor->m_type==BOX_NODE);

        return *(cursor->mp_BoxesArray);
}

//Array de CGate
void CLinkedList::AddGate(CGate GateArg)
{
        CNode* NewNode;
        CNode* cursor;
        CGate* NewGate;
        NewGate = new CGate;
        *(NewGate)=GateArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(GATE_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_GatesArray = NewGate;
}

CGate CLinkedList::GetGateAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==GATE_NODE);

        return *(cursor->mp_GatesArray);
}

CGate& CLinkedList::GateElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==GATE_NODE);

        return *(cursor->mp_GatesArray);
}

//Array de CTransition
void CLinkedList::AddTransit(CTransition TransitArg)
{
        CNode* NewNode;
        CNode* cursor;
        CTransition* NewTransit;
        NewTransit = new CTransition;
        *(NewTransit)=TransitArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(TRANSIT_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_TransArray = NewTransit;
}

CTransition CLinkedList::GetTransitAt(int n)
{
```

```
	ASSERT (n>=0);
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==TRANSIT_NODE);

	return *(cursor->mp_TransArray);
}

CTransition& CLinkedList::TransitElementAt(int n)
{
	ASSERT (n>=0);// Daniel
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==TRANSIT_NODE);

	return *(cursor->mp_TransArray);
}

//Array de CMark
void CLinkedList::AddMark(CMark MarkArg)
{
	CNode* NewNode;
	CNode* cursor;
	CMark* NewMark;
	NewMark = new CMark;
	*(NewMark)=MarkArg;
	cursor = mp_Sentry;
	while (cursor->mp_NextNode!=NULL)
		cursor = cursor->mp_NextNode;
	m_n++;
	NewNode = new CNode(MARKS_NODE);
	cursor->mp_NextNode = NewNode;
	NewNode->mp_MarksArray = NewMark;
}

CMark CLinkedList::GetMarkAt(int n)
{
	ASSERT (n>=0);
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==MARKS_NODE);

	return *(cursor->mp_MarksArray);
}

CMark& CLinkedList::MarkElementAt(int n)
{
	ASSERT (n>=0);// Daniel
	ASSERT (n<m_n);
	short i;
	CNode* cursor;
	cursor = mp_Sentry;
	for (i=0;i<=n;i++)
		cursor = cursor->mp_NextNode;

	ASSERT(cursor->m_type==MARKS_NODE);
```

```
		return *(cursor->mp_MarksArray);
}

CMark CLinkedList::RemoveMarkAt(int n)
{
		ASSERT (n>=0);
		ASSERT (n<m_n);
		short i;
		CNode* cursor;
		CNode* removed;
		CMark  retMark;
		cursor = mp_Sentry;
		for (i=0;i<n;i++)
  {
  if (cursor->mp_NextNode!=NULL)
   {
		cursor = cursor->mp_NextNode;
   }
  else
   {
   ASSERT(FALSE);
   }
  }

		ASSERT(cursor->m_type==MARKS_NODE||((cursor->m_type==NULO)&&(n==0)));

		removed = cursor->mp_NextNode;
 if (removed!=NULL)
  {
		//short-circuits the connection
		if (cursor->mp_NextNode->mp_NextNode != NULL)
			cursor->mp_NextNode = cursor->mp_NextNode->mp_NextNode;
		else
			cursor->mp_NextNode = NULL;
		m_n--;
		retMark = *(removed->mp_MarksArray);
  removed->mp_NextNode=NULL;// Daniel
		 delete removed;
  }
		return retMark;
}

//Array de CConditional
void CLinkedList::AddConditional(CConditional& ConditionalArg)
{
		CNode* NewNode;
		CNode* cursor;
		NewNode = new CNode(CONDS_NODE);
		NewNode->mp_CondsArray = new CConditional (ConditionalArg);
		cursor = mp_Sentry;
		while (cursor->mp_NextNode!=NULL)
			cursor = cursor->mp_NextNode;
		m_n++;
		cursor->mp_NextNode = NewNode;

}

CConditional CLinkedList::GetConditionalAt(int n)
{
		ASSERT (n>=0);
		ASSERT (n<m_n);
		short i;
		CNode* cursor;
		cursor = mp_Sentry;
		for (i=0;i<=n;i++)
			cursor = cursor->mp_NextNode;

		ASSERT(cursor->m_type==CONDS_NODE);

		return *(cursor->mp_CondsArray);
}
```

```
CConditional& CLinkedList::ConditionalElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==CONDS_NODE);

        return *(cursor->mp_CondsArray);
}

//Array de CAttribution
void CLinkedList::AddAttribution(CAttribution AttributionArg)
{
        CNode* NewNode;
        CNode* cursor;
        CAttribution* NewAttribution;
        NewAttribution = new CAttribution;
        *(NewAttribution)=AttributionArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(ATTRI_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_AttribArray = NewAttribution;
}

CAttribution CLinkedList::GetAttributionAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ATTRI_NODE);

        return *(cursor->mp_AttribArray);
}

CAttribution& CLinkedList::AttributionElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ATTRI_NODE);

        return *(cursor->mp_AttribArray);
}

//Array de Integer
void CLinkedList::AddInteger(short IntegerArg)
{
        CNode* NewNode;
        CNode* cursor;
        short* NewInteger;
        NewInteger = new short;
        *(NewInteger)=IntegerArg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
```

```
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(INT_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_IntArray = NewInteger;
}

short CLinkedList::GetIntegerAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==INT_NODE);

        return *(cursor->mp_IntArray);
}

short& CLinkedList::IntegerElementAt(int n)
{
        ASSERT (n>=0);// Daniel
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==INT_NODE);

        return *(cursor->mp_IntArray);
}

//Array de CAttribArray
void CLinkedList::AddAttribArray(CAttrbnArray Arg)
{
        CNode* NewNode;
        CNode* cursor;
        CAttrbnArray* NewArray;
        NewArray = new CAttrbnArray;
        *(NewArray)=Arg;
        cursor = mp_Sentry;
        while (cursor->mp_NextNode!=NULL)
                cursor = cursor->mp_NextNode;
        m_n++;
        NewNode = new CNode(ATTRIARRAY_NODE);
        cursor->mp_NextNode = NewNode;
        NewNode->mp_AtbnMatrix = NewArray;
}

CAttrbnArray CLinkedList::GetAttribArrayAt(int n)
{
        ASSERT (n>=0);
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ATTRIARRAY_NODE);

        return *(cursor->mp_AtbnMatrix);
}

CAttrbnArray& CLinkedList::AttribArrayElementAt(int n)
{
        ASSERT (n>=0);// Daniel
```

```
        ASSERT (n<m_n);
        short i;
        CNode* cursor;
        cursor = mp_Sentry;
        for (i=0;i<=n;i++)
                cursor = cursor->mp_NextNode;

        ASSERT(cursor->m_type==ATTRIARRAY_NODE);

        return *(cursor->mp_AtbnMatrix);
}
```

Files Wordsarray.h and Wordsarray.cpp - String arrays with copy constructor

```
// CString Array Extension Definition
// Last Modified : 20/07/98
// Programmed by : Daniel M. S. Ferreira
// Modified by : Marco A. A. Silva

class CWordsArray: public CStringArray
{
public:
        CWordsArray();
        ~CWordsArray();
        CWordsArray(const CWordsArray& SrcArray);
        CWordsArray& operator=(const CWordsArray& SrcArray);
};
// CString Array Extension Implementation
// Last Modified : 20/07/98
// Programmed by : Daniel M. S. Ferreira
// Modified by : Marco A. A. Silva

#include "stdafx.h"
#include "wordsarray.h"

CWordsArray::CWordsArray()
{
}

CWordsArray::~CWordsArray()
{
        long MySize=GetUpperBound();
        for (long index=MySize;index>0;index--)
                {
                RemoveAt(index);
                }
}

CWordsArray::CWordsArray(const CWordsArray& SrcArray)
{
        ASSERT(GetUpperBound()==-1);
        for (long index=0;index<SrcArray.GetSize();index++)
                {
                Add(SrcArray.GetAt(index));
                }
}

CWordsArray& CWordsArray::operator=(const CWordsArray& SrcArray)
{
        long MySize=GetUpperBound();
        for (long index=MySize;index>0;index--)
                {
                RemoveAt(index);
                }
        for (index=0;index<SrcArray.GetSize();index++)
                {
                Add(SrcArray.GetAt(index));
                }
        return *this;
}
```

B - Elementos Estruturais do E-MFG

Files Mark.h and Mark.cpp - E-MFG Marks

```
// marks.h - eMFG marks definition:
// version:2.0
// Definition Date: 11/12/97
// Last Modified: 18/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira

// special edition for the beta version

// Objetivos: Definir a estrutura de dados para a marcação do grafo

// OBS: CMarkArray e CMarkAttribArray estão definidas no arquivo objarray.h

class CMark : public CObject
{
private:
 class CGraph* mp_Owner;
        class CMarkAttribArray* mp_MarkAtributesArray;// Array of Mark Atributes
        BOOL m_CompositeMark;// Flag para marcas compostas
        class CMarkArray* mp_MarksList; // Array of Marks
        long m_MarkID; // Identificação da marca dentro do vetor de marcas
        BOOL m_WasExploded; // Flag de decomposição de marcas

public:
        // Construction & Destruction:
        CMark();
        ~CMark();


        // Build Functions:
        void Create(class CGraph* MyGraph,class CMarkAttribArray* p_MyMarkAtribArray, long MyID);


        // Data Management:
        class CMarkAttribute GetAtrib(CString CAtribLabel);
        class CMarkAttribute GetAtrib(long AtribIndex);
        class CMarkArray* ExplodeComposite(); // returns an array of Marks
        class CMarkArray* GetCompositeMarks(); // returns a reference to the composite mark array

// Functions to initialize the mark attribute
        BOOL SetAtrib(CString MyLabel, CString MyTextAtrib);
        BOOL SetAtrib(CString MyLabel, long MyIntegerAtrib);
BOOL SetAtrib(long MyIndex, long MyAtrib);
BOOL SetAtrib(long MyIndex, CString MyTextAtrib);

// Mark ID Functions (not used)
        long GetID();
        void SetID(long MyID);

// Composite Mark Manipulation (composite marks have an internal mark vector)
        void AddMark(CMark MyMark);
        void RemoveMark(long ArrayIndex);

// Graph Dynamics Mark Functions
        CMark MergeMark(CMark SrcMark);
        void SetAllAttributesToNull();
void SetAttribToNull(long MyIndex);
        void ResetComposition();
BOOL IsComposite();

// Dump function
        void GetDump(class CWordsArray* pLine);

        //serialização
        DECLARE_SERIAL (CMark);
        virtual void Serialize( CArchive& ar );
        //
```

```
        // Copy Constructor and Equal Operator
        CMark(const CMark& SrcMark);
        const CMark& operator =( const CMark& SrcMark );
        void operator =( CMark* SrcMark );
};

// marks.cpp  CMark class implementation
// version:2.0
// Definition Date: 11/12/97
// Last Modified: 18/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira

// Objetivos: Implementar a estrutura de dados para a marcação do grafo

#include "stdafx.h"                         // standard windows aplication
#include "elements.h"   // EMFG Standard Elements

#define MARKS_VERSION 1

        CMark::CMark()
        {
 mp_Owner=NULL;
        mp_MarkAtributesArray = NULL;
        m_CompositeMark = FALSE;
        mp_MarksList = NULL;
        m_MarkID = -1;
        m_WasExploded = FALSE;

        }


        CMark::~CMark()
        {
        if (mp_MarkAtributesArray != NULL)
                {
                delete mp_MarkAtributesArray;
                mp_MarkAtributesArray = NULL;
                }
        if (mp_MarksList!=NULL)
                {
                delete mp_MarksList;
                mp_MarksList = NULL;
                }
        }



        // Build Functions:
        void CMark::Create(class CGraph* MyGraph,       class CMarkAttribArray* p_MyMarkAtribArray, long MyID)
        {
 mp_Owner= MyGraph;
        ASSERT(p_MyMarkAtribArray!=NULL);
        ASSERT(MyID>=0);

        mp_MarkAtributesArray = new CMarkAttribArray;
        for (long index=0;index<(p_MyMarkAtribArray->GetSize());index++)
                {
                mp_MarkAtributesArray->Add(p_MyMarkAtribArray->GetAt(index));
                }
        m_CompositeMark = FALSE;
        mp_MarksList = NULL;
        m_MarkID = MyID;
        m_WasExploded = FALSE;
        }

        // Data Management:
class   CMarkAttribute CMark::GetAtrib(CString CAtribLabel)
{
long ReqIndex = -1;
for (int index = 0; index<mp_MarkAtributesArray->GetSize(); index++)
                {
```

```
                    if ((mp_MarkAtributesArray->GetAt(index)).GetLabel()==CAtribLabel)
                                {
                                ReqIndex=index;
                                break;
                                }

                    }
ASSERT(ReqIndex>=0);

return (mp_MarkAtributesArray->GetAt(ReqIndex));
}

class       CMarkAttribute CMark::GetAtrib(long AtribIndex)
{
ASSERT(AtribIndex>=0&&AtribIndex<mp_MarkAtributesArray->GetSize());
return (mp_MarkAtributesArray->GetAt(AtribIndex));
}

void CMark::GetDump(CWordsArray* pLine)
{
for (long index=0;index<mp_MarkAtributesArray->GetSize();index++)
                    {
                    pLine->Add(mp_MarkAtributesArray->ElementAt(index).GetDump());
                    }
if (IsComposite())
 {
 pLine->Add("Composition:");
 for (long i=0;i<mp_MarksList->GetSize();i++)
  {
  CString MID;
  MID.Format("%ld",i);
  MID = "M" + MID;
  pLine->Add(MID);
  mp_MarksList->GetAt(i).GetDump(pLine);
  }
 }
}

class       CMarkArray* CMark::ExplodeComposite()
{
// returns an array of Marks
return mp_MarksList;
}

class       CMarkArray* CMark::GetCompositeMarks()
{
// returns a reference to the composite mark array
return mp_MarksList;
}

BOOL CMark::SetAtrib(long MyIndex, CString MyTextAtrib)
{

(mp_MarkAtributesArray->ElementAt(MyIndex)).SetAtrib(MyTextAtrib);

return TRUE;
}

BOOL CMark::SetAtrib(long MyIndex, long MyAtrib)
{

(mp_MarkAtributesArray->ElementAt(MyIndex)).SetAtrib(MyAtrib);

return TRUE;
}

BOOL CMark::SetAtrib(CString MyLabel, CString MyTextAtrib)
{
ASSERT(GetAtrib(MyLabel).GetType()==TEXT_ATRIB);
if (!(GetAtrib(MyLabel).GetType()==TEXT_ATRIB))
                    return FALSE;
```

```
long ReqIndex = -1;
for (int index = 0; index<mp_MarkAtributesArray->GetSize(); index++)
                        {
                        if ((mp_MarkAtributesArray->GetAt(index)).GetLabel()==MyLabel)
                                        {
                                        ReqIndex=index;
                                        break;
                                        }

                        }
ASSERT(ReqIndex>=0);

(mp_MarkAtributesArray->ElementAt(ReqIndex)).SetAtrib(MyTextAtrib);

return TRUE;
}

BOOL CMark::SetAtrib(CString MyLabel, long MyIntegerAtrib)
{
ASSERT(GetAtrib(MyLabel).GetType()==INTEGER_ATRIB);
if (!(GetAtrib(MyLabel).GetType()==INTEGER_ATRIB))
                        return FALSE;

long ReqIndex = -1;
for (int index = 0; index<mp_MarkAtributesArray->GetSize(); index++)
                        {
                        if ((mp_MarkAtributesArray->GetAt(index)).GetLabel()==MyLabel)
                                        {
                                        ReqIndex=index;
                                        break;
                                        }

                        }
ASSERT(ReqIndex>=0);

(mp_MarkAtributesArray->ElementAt(ReqIndex)).SetAtrib(MyIntegerAtrib);

return TRUE;

}

        long CMark::GetID()
        {
        return m_MarkID;
        }

        void CMark::SetID(long MyID)
        {
        m_MarkID = MyID;
        }


        void CMark::AddMark(CMark MyMark)
        {
 if (m_CompositeMark == FALSE)
                        {
                        m_CompositeMark=TRUE;
  ASSERT(mp_MarksList==NULL);
                        mp_MarksList = new CMarkArray;
                        mp_MarksList->Add(MyMark);
        }
        else
                        {
                        mp_MarksList->Add(MyMark);
                        }
        }

        void CMark::RemoveMark(long ArrayIndex)
        {
 ASSERT(FALSE);
//      mp_MarksList->RemoveAt(ArrayIndex);
        }
```

```
//serialização

IMPLEMENT_SERIAL (CMark,CObject,MARKS_VERSION);
void CMark::Serialize( CArchive& ar )
        {
        ASSERT(FALSE);
        }

        //
        // Construtor de Cópia e operador igual:


        CMark::CMark(const CMark& SrcMark)
        {
mp_Owner= SrcMark.mp_Owner;
        if ((SrcMark.mp_MarkAtributesArray)!=NULL)
                {
                mp_MarkAtributesArray = new CMarkAttribArray;
                for (long index=0;index<((SrcMark.mp_MarkAtributesArray)->GetSize());index++)
                        {
                        mp_MarkAtributesArray->Add(SrcMark.mp_MarkAtributesArray->GetAt(index));
                        }
                }
        else
                {
                mp_MarkAtributesArray=NULL;
                }

        m_CompositeMark = SrcMark.m_CompositeMark;

        if (SrcMark.mp_MarksList!=NULL)
                {
                        mp_MarksList = new CMarkArray;
                        for (long index=0;index<((SrcMark.mp_MarksList)->GetSize());index++)
                                {
                                mp_MarksList->Add(SrcMark.mp_MarksList->GetAt(index));
                                }
                }
        else
                {
                mp_MarksList=NULL;
                }

        m_MarkID = SrcMark.m_MarkID;//Warning: New Marks must have a different ID
        m_WasExploded = SrcMark.m_WasExploded;
        }

        const CMark& CMark::operator =( const CMark& SrcMark )
        {
mp_Owner= SrcMark.mp_Owner;
        if (mp_MarkAtributesArray != NULL)
                {
                delete mp_MarkAtributesArray;
                mp_MarkAtributesArray = NULL;
                }
        if ((SrcMark.mp_MarkAtributesArray)!=NULL)
                {
                mp_MarkAtributesArray = new CMarkAttribArray;
                for (long index=0;index<((SrcMark.mp_MarkAtributesArray)->GetSize());index++)
                        {
                        mp_MarkAtributesArray->Add(SrcMark.mp_MarkAtributesArray->GetAt(index));
                        }
                }
        else
                {
                mp_MarkAtributesArray=NULL;
                }

        m_CompositeMark = SrcMark.m_CompositeMark;
        if (mp_MarksList!=NULL)
                {
```

```
                delete mp_MarksList;
                mp_MarksList = NULL;
                }
        if (SrcMark.mp_MarksList!=NULL)
                {
                        mp_MarksList = new CMarkArray;
                        for (long index=0;index<((SrcMark.mp_MarksList)->GetSize());index++)
                                {
                                mp_MarksList->Add(SrcMark.mp_MarksList->GetAt(index));
                                }
                }
        else
                {
                mp_MarksList=NULL;
                }

        m_MarkID = SrcMark.m_MarkID;//Warning: New Marks must have a different ID
        m_WasExploded = SrcMark.m_WasExploded;
        return *this;
        }

        CMark CMark::MergeMark(CMark SrcMark)
        {
        // Merge Atributes:
        for (long index=0;index<((SrcMark.mp_MarkAtributesArray)->GetSize());index++)
                {
                switch ((SrcMark.mp_MarkAtributesArray)->GetAt(index).GetType())
                                {
                                case INTEGER_ATRIB:
                                                {
                                                if ((SrcMark.mp_MarkAtributesArray)-
>GetAt(index).GetIntegerAtrib()!=0)
                                                                {
                                                                mp_MarkAtributesArray-
>ElementAt(index).SetAtrib((SrcMark.mp_MarkAtributesArray)->GetAt(index).GetIntegerAtrib());

                                                                }
                                                break;
                                                }
                                case TEXT_ATRIB:
                                                {
                                                if ((SrcMark.mp_MarkAtributesArray)-
>GetAt(index).GetTextAtrib()!="")
                                                                {
                                                                mp_MarkAtributesArray-
>ElementAt(index).SetAtrib((SrcMark.mp_MarkAtributesArray)->GetAt(index).GetTextAtrib());

                                                                }
                                                break;
                                                }
                                };
                }
                // Merge Composition
                if (SrcMark.GetCompositeMarks()!=NULL)
                                {
                                for (long index=0;index<SrcMark.GetCompositeMarks()->GetSize();index++)
                                                {
                                                AddMark(SrcMark.GetCompositeMarks()->GetAt(index));
                                                }
                                }

        return *this;
        }

        void CMark::SetAllAttributesToNull()
        {
        ASSERT(mp_MarkAtributesArray!=NULL);
        for (long index=0;index<mp_MarkAtributesArray->GetSize();index++)
                {
                mp_MarkAtributesArray->ElementAt(index).SetToNull();
                }
        }
```

```
void CMark::SetAttribToNull(long MyIndex)
{
ASSERT(mp_MarkAtributesArray!=NULL);
ASSERT(mp_MarkAtributesArray->GetSize()>MyIndex);
ASSERT(MyIndex>=0);
mp_MarkAtributesArray->ElementAt(MyIndex).SetToNull();
}

        void CMark::ResetComposition()
        {
        m_CompositeMark = FALSE;
        if (mp_MarksList!=NULL)
                {
                delete mp_MarksList;
                mp_MarksList = NULL;
                }
        m_WasExploded = FALSE;
        }

BOOL CMark::IsComposite()
 {
 return m_CompositeMark;
 }
```

Files Boxes.h and Boxes.cpp - E-MFG Boxes

```
// boxes.h eMFG Boxes Definition:
//
// version:2.0
// Definition Date: 11/12/97
// Last Modified: 08/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira

// Objetivos: Definir a estrutura de dados dos elementos de estado
// (boxes) do grafo. Estão previstos boxes comuns, temporizados, capacidade, acumuladores (packing), dispersores(unpacking) e
transformadores

class CBox : public CObject
{
protected:
class CGraph* mp_Owner;
        // Box Type:
        short m_Type;
        // Box Label:
        CString m_Label;
        // Number of Origin and Destiny Arcs:
        long m_NumOriginArcs;
        long m_NumDestinyArcs;
        // Array of Indexes to Origin Arcs:
        long* mp_OriginArcIndex;
        // Array of Indexes to Destiny Arcs:
        long* mp_DestinyArcIndex;
        // This Box Mark:
        class CMark m_BoxMark;
        // Flag: Do I have any mark inside this box?
        BOOL m_HasAnyMark;

//Old CTransformatorBox Data Members:
protected:
// Transformator Engine array of atribuitions:

class CCondArray*  mp_ConditionsArray;
class CAtbnMatrix* mp_ThenAttributions;
class CAtbnMatrix* mp_ElseAttributions;

//Old CCapacity Box Data Members:
protected:
// Box Capacity:
        long m_Capacity;
// Box Array of Marks
```

```
class CMarkArray* mp_BoxMarkArray;

BOOL m_Deadlock;
// Picking Order Preference:
        BOOL m_Fifo;
// Temporized Boxes Data Mambers:

CTime m_StartTime;
long m_TimeCte;
long m_TimeCount;
BOOL m_IsTimerRunning;
BOOL m_HasTimerRunned;


public:
        // Construction & Destruction:
        CBox();
        ~CBox();

        // CBox Build Functions:
        void Create(CGraph* MyGraph, short MyType, CString MyLabel,long MyNumOriginArcs, long MyNumDestinyArcs,
long* MyOriginIndex, long* MyDestinyIndex);

        //CBox Data Management:
        virtual void AddMark(class CMark MyMark);
        virtual class CMark RemoveMark();
        class CMarkAttribute GetMarkAtrib(CString AtribLabel);
        class CMarkAttribute GetMarkAtrib(long AtribIndex);
        void SetMarkAtribute(CString AtribLabel, CString AtribValue);
        void SetMarkAtribute(CString AtribLabel, long AtribValue);
        void SetMarkAtribute(CMarkAttribute RefAtrib);
 void SetMarkAtribute(long Index, long AtribValue);
 void SetMarkAtribute(long Index, CString AtribValue);

        CString GetLabel();
        short GetType();

        BOOL HasMark();

        long GetNumOriginArcs();
        long GetNumDestinyArcs();
        long* GetOriginArcsIndexList();
        long* GetDestinyArcsIndexList();
        long* GetOriginTransitionsIndexList(class CArcArray* p_ArcsArray);
        long* GetDestinyTransitionsIndexList(class CArcArray* p_ArcsArray,class CGateArray* p_MyGateArray, class
CBoxArray* p_MyBoxArray);

//////////////////////////////////////////////////////
// Old CTransformatorBox Function Members:
public:
 // Adds a if then else statement
 void AddBlock(class CCondArray MyConditions, class CAtbnMatrix MyThenAtbn, class CAtbnMatrix MyElseAtbn);

protected:
        void RunTransformationEngine(); // Executes the if-then-else command (called by AddMark)

//////////////////////////////////////////////////////////////////////////
// Old CCapacityBox Packing and Unpacking Box  Function Members: // this constructor will have to be used for the Unpacking
Box
public:
 void Create(CGraph* p_MyGraph,short MyType, CString MyLabel,long MyNumOriginArcs, long MyNumDestinyArcs, long*
MyOriginIndex, long* MyDestinyIndex, long MyCapacity, BOOL IsFifo); // This is THE constructor, please keep other function as
reference for a while

        long GetNumMarks();
        BOOL IsFull();
        BOOL HasNorMoreMarks(long NumMarks);

 BOOL IsAReadyPreCondition();
 BOOL IsAReadyPosCondition();

 void ResetDeadlock();
```

```
 BOOL IsDeadlock();

// Temporized Box Functions (not implemented):
public:
        void SetBoxAsTemporized(long MyTimeCte);// only for Common Box
 void StartTimer(CTime MyTime);
 void UpdateTimer(CTime MyTime);
 BOOL IsTimeCounting();
 void ResetTimer();
 BOOL HasTimerRunned();


// Copy Constructor and Operator equal:
public:
        CBox(const CBox& BoxSrc );
        CBox& operator = (CBox& BoxSrc);
        DECLARE_SERIAL (CBox);
        virtual void Serialize( CArchive& ar );
        void GetDump(class CLinesArray* pTable);

protected:
        void ResetMarkFlag();
};

// boxes.cpp eMFG Boxes Implementation:
//
// version:2.0
// Definition Date: 11/12/97
// Daniel M. S. Ferreira & Marco A. A. Silva
// Last Modified: 08/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira
//
// Objetivos: Implementar a estrutura de dados dos elementos de estado
// (boxes) do grafo. Estão previstos boxes comuns, temporizados, capacidade, acumuladores (packing), dispersores(unpacking) e
transformadores

#include "stdafx.h"                             //standard windows aplication
#include "elements.h"       //E-MFG elements

//////////////////////////////////////////////////////////////////
// class CBox : public CObject


// Construction & Destruction:
CBox::CBox()
        {

 mp_Owner=NULL;
 // Box Type:
        m_Type = COMMON_BOX;
        // Box Label:
        m_Label="";
        // Number of Origin and Destiny Arcs:
        m_NumOriginArcs=0;
        m_NumDestinyArcs=0;
        // Array of Indexes to Origin Arcs:
        mp_OriginArcIndex=NULL;
        // Array of Indexes to Destiny Arcs:
        mp_DestinyArcIndex=NULL;
        // Flag: Do I have any mark inside this box?
        m_HasAnyMark=FALSE;



//Old CCapacity Box Data Members:
        // Box Capacity:
        m_Capacity=1;
        // Box Array of Marks
        mp_BoxMarkArray=NULL;
        // Picking Order Preference:
        m_Fifo=FALSE;
```

```
 m_Deadlock = FALSE;

 // Timer:
// CTime m_StartTime;
 m_TimeCte = 0;
 m_TimeCount = 0;
 m_IsTimerRunning = FALSE;
 m_HasTimerRunned = FALSE;

 mp_ConditionsArray = NULL;
 mp_ThenAttributions = NULL;
 mp_ElseAttributions = NULL;
        }

        CBox::~CBox()
        {
        if (mp_OriginArcIndex!=NULL)
                {
                delete [] mp_OriginArcIndex;
                mp_OriginArcIndex=NULL;
                }

        if (mp_DestinyArcIndex!=NULL)
                {
                delete [] mp_DestinyArcIndex;
                mp_DestinyArcIndex=NULL;
                }

        if (mp_BoxMarkArray!=NULL)
                {
                delete mp_BoxMarkArray;
                mp_BoxMarkArray=NULL;
                }

 if (mp_ConditionsArray!=NULL)
  {
  delete mp_ConditionsArray;
  mp_ConditionsArray = NULL;
  }

 if (mp_ThenAttributions!=NULL)
  {
  delete mp_ThenAttributions;
  mp_ThenAttributions = NULL;
  }

 if (mp_ElseAttributions!=NULL)
  {
  delete mp_ElseAttributions;
  mp_ElseAttributions = NULL;
  }
        }


        // CBox Build Functions:
        void CBox::Create(CGraph* MyGraph, short MyType, CString MyLabel,long MyNumOriginArcs, long
MyNumDestinyArcs, long* MyOriginIndex, long* MyDestinyIndex)
        {
 ASSERT(MyGraph!=NULL);
 mp_Owner = MyGraph;
        ASSERT(MyType==COMMON_BOX||MyType==TRANSFORMATOR_BOX);
        // Box Type:
        m_Type = MyType;
        // Box Label:
        m_Label=MyLabel;
        // Number of Origin and Destiny Arcs:
        m_NumOriginArcs=MyNumOriginArcs;
        m_NumDestinyArcs=MyNumDestinyArcs;
        // Array of Indexes to Origin Arcs:
        if ((m_NumOriginArcs>0)&&(MyOriginIndex!=NULL))
                {
```

```
                mp_OriginArcIndex=new long[m_NumOriginArcs];
                for (long index=0;index<m_NumOriginArcs;index++)
                        {
                        mp_OriginArcIndex[index]=MyOriginIndex[index];
                        }
                }
        else
                {
                mp_OriginArcIndex=NULL;
                }
        // Array of Indexes to Destiny Arcs:
        if ((m_NumDestinyArcs>0)&&(MyDestinyIndex!=NULL))
                {
                mp_DestinyArcIndex=new long[m_NumDestinyArcs];
                for (long index=0;index<m_NumDestinyArcs;index++)
                        {
                        mp_DestinyArcIndex[index]=MyDestinyIndex[index];
                        }
                }
        else
                {
                mp_DestinyArcIndex=NULL;
                }
        // Flag: Do I have any mark inside this box?
        m_HasAnyMark=FALSE;

//Old CTransformatorBox Data Members:
        // Transformator Engine array of atribuitions:
//      mp_AttributionArray=NULL;
//      m_NumAttributions = 0;

//Old CCapacity Box Data Members:
        // Box Capacity:
        m_Capacity=1;
// Box Array of Marks
        mp_BoxMarkArray=NULL;
        // Picking Order Preference:
        m_Fifo=FALSE;

 mp_ConditionsArray = NULL;
 mp_ThenAttributions = NULL;
 mp_ElseAttributions = NULL;
        }

        //CBox Data Management:
        void CBox::AddMark(class CMark MyMark)
        {
 switch (m_Type)
  {
  case COMMON_BOX:
   {
                ASSERT(m_HasAnyMark==FALSE);
                m_BoxMark=MyMark;
                m_HasAnyMark=TRUE;
   break;
   }
   case TEMP_BOX:
   {
                ASSERT(m_HasAnyMark==FALSE);
                m_BoxMark=MyMark;
                m_HasAnyMark=TRUE;
   m_IsTimerRunning = TRUE;
   break;
   }
  case TRANSFORMATOR_BOX:
   {
                ASSERT(m_HasAnyMark==FALSE);
                 m_BoxMark=MyMark;
                m_HasAnyMark=TRUE;
                RunTransformationEngine();
   break;
   }
```

```
case CAPACITY_BOX:
 {
 ASSERT(mp_BoxMarkArray!=NULL);//tests for correct initialization
 ASSERT((m_HasAnyMark==TRUE&&mp_BoxMarkArray->GetSize()>0)||
    (m_HasAnyMark==FALSE&&mp_BoxMarkArray->GetSize()==0));//test for correct flag use
 ASSERT(mp_BoxMarkArray->GetSize()<m_Capacity);//test for correct pos_condition flag and capacity initialization
                    mp_BoxMarkArray->Add(MyMark);
                    m_HasAnyMark=TRUE;
 break;
 }
case PACKING_BOX:
 {
 ASSERT(mp_BoxMarkArray!=NULL);//tests for correct initialization
 ASSERT((m_HasAnyMark==TRUE&&mp_BoxMarkArray->GetSize()>0)||
    (m_HasAnyMark==FALSE&&mp_BoxMarkArray->GetSize()==0));//test for correct flag use
 ASSERT(mp_BoxMarkArray->GetSize()<m_Capacity);//test for correct pos_condition flag and capacity initialization
                    mp_BoxMarkArray->Add(MyMark);
 ASSERT(mp_BoxMarkArray->GetSize()<=m_Capacity);
                    m_HasAnyMark=TRUE;
 break;
 }
case UNPACKING_BOX:
 {
 ASSERT(mp_BoxMarkArray!=NULL);//tests for correct initialization
 ASSERT((m_HasAnyMark==FALSE&&mp_BoxMarkArray->GetSize()==0));//test for correct flag use
 ASSERT(mp_BoxMarkArray->GetSize()<m_Capacity);//test for correct pos_condition flag and capacity initialization
 if (MyMark.IsComposite())
  {
  ASSERT (MyMark.GetCompositeMarks()->GetSize()==m_Capacity);
  if (MyMark.GetCompositeMarks()->GetSize()==m_Capacity)
   {
   for (long index = 0;index<m_Capacity;index++)
    {
    mp_BoxMarkArray->Add(MyMark.GetCompositeMarks()->GetAt(index));
    }
   }
  else
   {
   // The mark and the unpacking box are not compatible this should cause a deadlock
   m_Deadlock = TRUE;
   ASSERT(FALSE);
   }
  }
 else
  {
   for (long index = 0;index<m_Capacity;index++)
    {
    mp_BoxMarkArray->Add(MyMark);
    }
  }
                    m_HasAnyMark=TRUE;
 break;
 }
};
        }

        class CMark CBox::RemoveMark()
        {
        if (m_Type==COMMON_BOX||m_Type==TRANSFORMATOR_BOX)
                    {
                    ASSERT(m_HasAnyMark==TRUE);
                    m_HasAnyMark=FALSE;
                    }
        if (m_Type==TEMP_BOX)
                    {
                    ASSERT(m_HasAnyMark==TRUE);
ASSERT(m_HasTimerRunned);
ResetTimer();
                    m_HasAnyMark=FALSE;
                    }

        if (m_Type==CAPACITY_BOX||m_Type==UNPACKING_BOX)
```

```
                    {
                    ASSERT(mp_BoxMarkArray->GetSize()>0);
                    if (m_Fifo==TRUE)
                                        {
                                        m_BoxMark=mp_BoxMarkArray->GetAt(0);//
                                        mp_BoxMarkArray->RemoveMarkAt(0);
                                        }
                    else
                                        {
                                        m_BoxMark=mp_BoxMarkArray->GetAt((mp_BoxMarkArray->GetSize()-1));
                                        mp_BoxMarkArray->RemoveMarkAt((mp_BoxMarkArray->GetSize()-1));
                                        }
                    if (mp_BoxMarkArray->GetSize()>0)
                                        {
                                        m_HasAnyMark=TRUE;
                                        }
                    else
                                        {
                                        m_HasAnyMark=FALSE;
                                        }
                    }
if (m_Type==PACKING_BOX)
                    {
                    ASSERT(mp_BoxMarkArray->GetSize()==m_Capacity);
                    if (m_Fifo==TRUE)
                                        {
                                        m_BoxMark=mp_BoxMarkArray->GetAt(0);//
   m_BoxMark.SetAllAttributesToNull();
   for (long i=0;i<m_Capacity;i++)
    {
    m_BoxMark.AddMark(mp_BoxMarkArray->GetAt(0));
    mp_BoxMarkArray->RemoveMarkAt(0);
    }
                                        }
                    else
                                        {
   m_BoxMark=mp_BoxMarkArray->GetAt(0);//
   m_BoxMark.SetAllAttributesToNull();
   for (long i=m_Capacity-1;i>=0;i--)
    {
    m_BoxMark.AddMark(mp_BoxMarkArray->GetAt(i));
    mp_BoxMarkArray->RemoveMarkAt(i);
    }
                                        }
 ASSERT(mp_BoxMarkArray->GetSize()==0);
                    m_HasAnyMark=FALSE;
                    }
CMark ReturnValue = m_BoxMark;
m_BoxMark.SetAllAttributesToNull();
m_BoxMark.ResetComposition();
return ReturnValue;
          }

          class CMarkAttribute CBox::GetMarkAtrib(CString AtribLabel)
          {
/* THIS FUNCTION IS USED WHEN THE ATTRIBUTE NUMBER IS NOT KNOWN */

          CMarkAttribute ReturnValue;
          if (m_HasAnyMark==TRUE)
                    {
                    if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                              {
                              if (mp_BoxMarkArray->GetSize()>0)
                                        {
                                        if (m_Fifo==TRUE)
                                                  {
                                                  ReturnValue=(mp_BoxMarkArray-
>GetAt(0)).GetAtrib(AtribLabel);
                                                  }
                                        else
                                                  {
```

```
                                                                    ReturnValue=(mp_BoxMarkArray-
>GetAt((mp_BoxMarkArray->GetSize()-1))).GetAtrib(AtribLabel);
                                                                    }
                                                        }
                                    }
                        else
                                    {
                                    ReturnValue=m_BoxMark.GetAtrib(AtribLabel);
                                    }
                        }
            else
                        {
                        BOOL AtribIsValid = FALSE;
                        for (long index=0;index<mp_Owner->GetAttribTemplates()->GetSize();index++)
                                    {
                                    if (AtribLabel==mp_Owner->GetAttribTemplates()->GetAt(index).GetLabel())
                                                {
                                                ReturnValue = mp_Owner->GetAttribTemplates()-
>GetAt(index);
                                                AtribIsValid = TRUE;
                                                break;
                                                }
                                    }
                        ASSERT(AtribIsValid);
                        }
            return ReturnValue;
            }


class CMarkAttribute CBox::GetMarkAtrib(long AtribIndex)
            {
            CMarkAttribute ReturnValue;
            if (m_HasAnyMark==TRUE)
                        {
                        if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                                    {
                                    if (mp_BoxMarkArray->GetSize()>0)
                                                {
                                                if (m_Fifo==TRUE)
                                                            {
                                                            ReturnValue=(mp_BoxMarkArray-
>GetAt(0)).GetAtrib(AtribIndex);
                                                            }
                                                else
                                                            {
                                                            ReturnValue=(mp_BoxMarkArray-
>GetAt((mp_BoxMarkArray->GetSize()-1))).GetAtrib(AtribIndex);
                                                            }
      }
    else
      {
      ReturnValue = mp_Owner->GetAttribTemplates()->GetAt(AtribIndex);
      }
                                    }
                        else
                        {
                                    ReturnValue=m_BoxMark.GetAtrib(AtribIndex);
                         }
                        }
            else
                        {
                        ReturnValue = mp_Owner->GetAttribTemplates()->GetAt(AtribIndex);
                        }
            return ReturnValue;
            }

            void CBox::SetMarkAtribute(CString AtribLabel, CString AtribValue)
            {

            if (m_HasAnyMark==TRUE)
                        {
                        if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
```

```
                                {
                                if (mp_BoxMarkArray->GetSize()>0)
                                                {
                                                if (m_Fifo==TRUE)
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt(0).SetAtrib(AtribLabel,AtribValue);
                                                                }
                                                else
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt((mp_BoxMarkArray->GetSize()-1)).SetAtrib(AtribLabel,AtribValue);;
                                                                }
        }
      else
        {
        ASSERT(FALSE); // should never get here!
        }
                                }
                        else
                        {
                                m_BoxMark.SetAtrib(AtribLabel,AtribValue);
                         }
                        }
                else
                        {
                        ASSERT(FALSE); // should never get here!
                        }

                }

                void CBox::SetMarkAtribute(CString AtribLabel, long AtribValue)
                {
                if (m_HasAnyMark==TRUE)
                        {
                        if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                                {
                                if (mp_BoxMarkArray->GetSize()>0)
                                                {
                                                if (m_Fifo==TRUE)
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt(0).SetAtrib(AtribLabel,AtribValue);
                                                                }
                                                else
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt((mp_BoxMarkArray->GetSize()-1)).SetAtrib(AtribLabel,AtribValue);;
                                                                }
        }
      else
        {
        ASSERT(FALSE); // should never get here!
        }
                                }
                        else
                        {
                                m_BoxMark.SetAtrib(AtribLabel,AtribValue);
                         }
                        }
                else
                        {
                        ASSERT(FALSE);// should never get here!
                        }
                }

void CBox::SetMarkAtribute(long Index, long AtribValue)
                {

                if (m_HasAnyMark==TRUE)
                        {
                        if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
```

```
                                {
                                if (mp_BoxMarkArray->GetSize()>0)
                                                {
                                                if (m_Fifo==TRUE)
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt(0).SetAtrib(Index,AtribValue);
                                                                }
                                                else
                                                                {
                                                                mp_BoxMarkArray-
>ElementAt((mp_BoxMarkArray->GetSize()-1)).SetAtrib(Index,AtribValue);;
                                                                }
        }
      else
        {
        ASSERT(FALSE);// should never get here!
        }
                                }
                        else
                        {
                                m_BoxMark.SetAtrib(Index,AtribValue);
                         }
                        }
                else
                        {
                        ASSERT(FALSE);// should never get here!
                        }
        }

 void CBox::SetMarkAtribute(long Index, CString AtribValue)
        {
        if (m_HasAnyMark==TRUE)
                {
                if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                        {
                        if (mp_BoxMarkArray->GetSize()>0)
                                        {
                                        if (m_Fifo==TRUE)
                                                        {
                                                        mp_BoxMarkArray-
>ElementAt(0).SetAtrib(Index,AtribValue);
                                                        }
                                        else
                                                        {
                                                        mp_BoxMarkArray-
>ElementAt((mp_BoxMarkArray->GetSize()-1)).SetAtrib(Index,AtribValue);;
                                                        }
        }
      else
        {
        ASSERT(FALSE);// should never get here!
        }
                        }
                else
                        {
                        m_BoxMark.SetAtrib(Index,AtribValue);
                         }
                        }
                else
                        {
                        ASSERT(FALSE);// should never get here!
                        }
        }

        void CBox::SetMarkAtribute(CMarkAttribute RefAtrib)
        {
        switch (RefAtrib.GetType())
                {
                case TEXT_ATRIB:
                                {
                                SetMarkAtribute(RefAtrib.GetLabel(),RefAtrib.GetTextAtrib());
```

```
                                        break;
                                        }
                        case INTEGER_ATRIB:
                                        {
                                        SetMarkAtribute(RefAtrib.GetLabel(),RefAtrib.GetIntegerAtrib());
                                        break;
                                        }
                        };
}

CString CBox::GetLabel()
{
return m_Label;
}

void CBox::GetDump(CLinesArray* pTable)
{
CWordsArray* pLine=NULL;
pLine = new CWordsArray;
pLine->Add(GetLabel());
switch (GetType())
                {
                case TRANSFORMATOR_BOX:
                                        {
                                        pLine->Add("TRANSFORMATOR_BOX");
                                        CString Dump;
                                        if(HasMark())
                                                                {
                                                                Dump="#Marks=1";
                                                                pLine->Add(Dump);
                                                                m_BoxMark.GetDump(pLine);
                                                                }
                                        else
                                                                {
                                                                Dump="#Marks=0";
                                                                pLine->Add(Dump);
                                                                }
                                        pTable->Add(*pLine);
                                        delete pLine;
                                        pLine = NULL;
                                        break;
                                        }
                case CAPACITY_BOX:
                                        {
                if (m_Fifo)
                                                                {
                                                                pLine->Add("CAPACITY_BOX(FIFO)");
                                                                }
                                        else
                                                                {
                                                                pLine->Add("CAPACITY_BOX(LIFO)");
                                                                }
                                        CString Dump;
                                        if(HasMark())
                                                                {
                                                                Dump.Format("%ld",GetNumMarks());
                                                                Dump="#Marks=" + Dump;
                                                                pLine->Add(Dump);
                                                                mp_BoxMarkArray->ElementAt(0).GetDump(pLine);
                                                                pTable->Add(*pLine);
                                                                delete pLine;
                                                                pLine = NULL;
                                                                for (long index=1;index<mp_BoxMarkArray-
>GetSize();index++)
                                                                                                {
                                                                                                pLine = new CWordsArray;
                                                                                                pLine->Add("");
                                                                                                pLine->Add("");
                                                                                                pLine->Add("");
                                                                                                mp_BoxMarkArray-
>ElementAt(index).GetDump(pLine);
                                                                                                pTable->Add(*pLine);
```

```
                                                    delete pLine;
                                                    pLine = NULL;
                                                    }
                                        }
                                else
                                        {
                                        Dump="#Marks=0";
                                        pLine->Add(Dump);
                                        pTable->Add(*pLine);
                                        delete pLine;
                                        pLine = NULL;
                                        }

                                break;
                                }

                        case UNPACKING_BOX:
                                {
                CString SCap;
        SCap.Format("%ld",m_Capacity);
                if (m_Fifo)
                                        {
                                        pLine->Add("UNPACKING_BOX(FIFO,"+SCap+")");
                                        }
                                else
                                        {
                                        pLine->Add("UNPACKING_BOX(LIFO,"+SCap+")");
                                        }
                                CString Dump;
                                if(HasMark())
                                        {
                                        Dump.Format("%ld",GetNumMarks());
                                        Dump="#Marks=" + Dump;
                                        pLine->Add(Dump);
                                        mp_BoxMarkArray->ElementAt(0).GetDump(pLine);
                                        pTable->Add(*pLine);
                                        delete pLine;
                                        pLine = NULL;
                                        for (long index=1;index<mp_BoxMarkArray-
>GetSize();index++)
                                                {
                                                pLine = new CWordsArray;
                                                pLine->Add("");
                                                pLine->Add("");
                                                pLine->Add("");
                                                mp_BoxMarkArray-
>ElementAt(index).GetDump(pLine);
                                                pTable->Add(*pLine);
                                                delete pLine;
                                                pLine = NULL;
                                                }
                                        }
                                else
                                        {
                                        Dump="#Marks=0";
                                        pLine->Add(Dump);
                                        pTable->Add(*pLine);
                                        delete pLine;
                                        pLine = NULL;
                                        }

                                break;
                                }

                        case PACKING_BOX:
                                {
                CString SCap;
        SCap.Format("%ld",m_Capacity);
                if (m_Fifo)
                                        {
                                        pLine->Add("PACKING_BOX(FIFO,"+SCap+")");
                                        }
```

```
                                else
                                                {
                                                pLine->Add("PACKING_BOX(LIFO,"+SCap+")");
                                                }
                                CString Dump;
                                if(HasMark())
                                                {
                                                Dump.Format("%ld",GetNumMarks());
                                                Dump="#Marks=" + Dump;
                                                pLine->Add(Dump);
                                                mp_BoxMarkArray->ElementAt(0).GetDump(pLine);
                                                pTable->Add(*pLine);
                                                delete pLine;
                                                pLine = NULL;
                                                for (long index=1;index<mp_BoxMarkArray-
>GetSize();index++)
                                                                {
                                                                pLine = new CWordsArray;
                                                                pLine->Add("");
                                                                pLine->Add("");
                                                                pLine->Add("");
                                                                mp_BoxMarkArray-
>ElementAt(index).GetDump(pLine);
                                                                pTable->Add(*pLine);
                                                                delete pLine;
                                                                pLine = NULL;
                                                                }
                                                }
                                else
                                                {
                                                Dump="#Marks=0";
                                                pLine->Add(Dump);
                                                pTable->Add(*pLine);
                                                delete pLine;
                                                pLine = NULL;
                                                }
                                break;
                                }

                case COMMON_BOX:
                                {
                                pLine->Add("COMMON_BOX");
                                CString Dump;
                                if(HasMark())
                                                {
                                                Dump="#Marks=1";
                                                pLine->Add(Dump);
                                                m_BoxMark.GetDump(pLine);
                                                }
                                else
                                                {
                                                Dump="#Marks=0";
                                                pLine->Add(Dump);
                                                }
                                pTable->Add(*pLine);
                                delete pLine;
                                pLine = NULL;
                                break;
                                }
                case TEMP_BOX:
                                {
    CString TCte;
    TCte.Format("%ld",m_TimeCte);
                                pLine->Add("TEMP_BOX ("+TCte+")");
                                CString Dump;
                                if(HasMark())
                                                {
                                                Dump="#Marks=1";
                                                pLine->Add(Dump);
                                                m_BoxMark.GetDump(pLine);
                                                }
                                else
```

```
                                                {
                                                Dump="#Marks=0";
                                                pLine->Add(Dump);
                                                }
                                        pTable->Add(*pLine);
                                        delete pLine;
                                        pLine = NULL;
                                        break;
                                        }

                        }
                ASSERT (pLine==NULL);
                }

                short CBox::GetType()
                {
                return m_Type;
                }

                BOOL CBox::HasMark()
                {
                return m_HasAnyMark;
                }

                long CBox::GetNumOriginArcs()
                {
                return m_NumOriginArcs;
                }

                long CBox::GetNumDestinyArcs()
                {
                return m_NumDestinyArcs;
                }

                long* CBox::GetOriginArcsIndexList()
                {
                return mp_OriginArcIndex;
                }

                long* CBox::GetDestinyArcsIndexList()
                {
                return mp_DestinyArcIndex;
                }

                long* CBox::GetOriginTransitionsIndexList(class CArcArray* p_ArcsArray)
                {
                long* p_TransitionIndex=new long[GetNumOriginArcs()];
                for (long index=0;index<GetNumOriginArcs();index++)
                        {
                        p_TransitionIndex[index]=p_ArcsArray->GetAt(mp_OriginArcIndex[index]).GetOrigin();
                        }
                return p_TransitionIndex;
                }

        long* CBox::GetDestinyTransitionsIndexList(class CArcArray* p_ArcsArray,class CGateArray* p_MyGateArray,
class CBoxArray* p_MyBoxArray)
                {
                long* p_TransitionIndex=new long[GetNumDestinyArcs()];
                for (long index=0;index<GetNumDestinyArcs();index++)
                        {
                        p_TransitionIndex[index]=(p_ArcsArray->GetAt(mp_DestinyArcIndex[index])).GetDestiny();
                        }
                return p_TransitionIndex;
                }



                void CBox::RunTransformationEngine()
                {

                for (long CondIndex = 0; CondIndex<mp_ConditionsArray->GetSize(); CondIndex++)
                        {
```

```
    // WAITING FOR MARCO'S CONDITIONAL ELEMENTS DEFINITION AND THE PROPER
    // FUNCTIONS ON CLINKEDLIST FOR CONDITIONALS AND ATTRIBUTIONS

    if (mp_ConditionsArray->GetAt(CondIndex).Evaluate())
     {
     long NumTrueAttrib = mp_ThenAttributions->GetAt(CondIndex).GetSize();
     for (long AttIndex = 0;AttIndex<NumTrueAttrib;AttIndex++)
      {
      CAttribution Actual = mp_ThenAttributions->ElementAt(CondIndex).ElementAt(AttIndex);
      Actual.DoAttribution(mp_Owner->GetBoxes(),mp_Owner->GetGates());
      }
     }
    else
     {
     long NumFalseAttrib = mp_ElseAttributions->GetAt(CondIndex).GetSize();
     for (long AttIndex = 0;AttIndex<NumFalseAttrib;AttIndex++)
      {
      CAttribution Actual = mp_ElseAttributions->ElementAt(CondIndex).ElementAt(AttIndex);;
      Actual.DoAttribution(mp_Owner->GetBoxes(),mp_Owner->GetGates());
      }
     }
            }
            }

// Old CCapacityBox Function Members:


            void CBox::Create(CGraph *p_MyGraph,short MyType, CString MyLabel,long MyNumOriginArcs, long
MyNumDestinyArcs, long* MyOriginIndex, long* MyDestinyIndex, long MyCapacity, BOOL IsFifo)
            {
            ASSERT(MyType==CAPACITY_BOX||MyType==PACKING_BOX||MyType==UNPACKING_BOX);
 ASSERT(p_MyGraph!=NULL);
 // Owner Graph:
            mp_Owner = p_MyGraph;
 // Box Type:
            m_Type = MyType;
            // Box Label:
            m_Label=MyLabel;
            // Number of Origin and Destiny Arcs:
            m_NumOriginArcs=MyNumOriginArcs;
            m_NumDestinyArcs=MyNumDestinyArcs;
            // Array of Indexes to Origin Arcs:
            if ((m_NumOriginArcs>0)&&(MyOriginIndex!=NULL))
                        {
                        mp_OriginArcIndex=new long[m_NumOriginArcs];
                        for (long index=0;index<m_NumOriginArcs;index++)
                                                {
                                                mp_OriginArcIndex[index]=MyOriginIndex[index];
                                                }
                        }
            else
                        {
                        mp_OriginArcIndex=NULL;
                        }
            // Array of Indexes to Destiny Arcs:
            if ((m_NumDestinyArcs>0)&&(MyDestinyIndex!=NULL))
                        {
                        mp_DestinyArcIndex=new long[m_NumDestinyArcs];
                        for (long index=0;index<m_NumDestinyArcs;index++)
                                                {
                                                mp_DestinyArcIndex[index]=MyDestinyIndex[index];
                                                }
                        }
            else
                        {
                        mp_DestinyArcIndex=NULL;
                        }
            // Flag: Do I have any mark inside this box?
            m_HasAnyMark=FALSE;

//Old CTransformatorBox Data Members:
            // Transformator Engine array of Attributions:
```

```
//          mp_AttributionArray=NULL;
//          m_NumAttributions = 0;
//Old CCapacity Box Data Members:
          // Box Capacity:
          m_Capacity= MyCapacity;
          // Box Array of Marks
 mp_BoxMarkArray=new CMarkArray;

          // Picking Order Preference:
          m_Fifo= IsFifo;

 mp_ConditionsArray = NULL;
 mp_ThenAttributions = NULL;
 mp_ElseAttributions = NULL;
          }


          long CBox::GetNumMarks()
          {
          long ReturnValue=0;
          if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                    {
                    ReturnValue= mp_BoxMarkArray->GetSize();
                    }
          else
                    {
                    ReturnValue = (long)m_HasAnyMark;
                    }

          return ReturnValue;
          }

          BOOL CBox::IsFull()
          {

          if (m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX)
                    {
          ASSERT(mp_BoxMarkArray->GetSize()<=m_Capacity);
          if (mp_BoxMarkArray->GetSize()<m_Capacity)
                                        {
                                        return FALSE;
                                        }
                    else
                                        {
                                        return TRUE;
                                        }
 ASSERT(FALSE);
 return FALSE;
                    }
          else
                    {
                    return m_HasAnyMark;
                    }
          }

          BOOL CBox::HasNorMoreMarks(long NumMarks)
          {

          ASSERT(m_Type==CAPACITY_BOX||m_Type==PACKING_BOX||m_Type==UNPACKING_BOX);
          ASSERT (mp_BoxMarkArray!=NULL);
if (mp_BoxMarkArray!=NULL)
 {
 if (mp_BoxMarkArray->GetSize()<NumMarks)
                    {
                    return FALSE;
                    }
          else
                    {
                    return TRUE;
                    }
 }
if (m_Capacity==1&&NumMarks>1)
```

```
    {
    return FALSE;
    }
  if ((m_Capacity==1)&&(NumMarks==1)&&(HasMark()))
    {
    return TRUE;
    }
 // Should never get here
  ASSERT(FALSE);
  return FALSE;
            }


// Copy Constructor and Operator equal:

            CBox::CBox(const CBox& BoxSrc )
            {
            m_Type = BoxSrc.m_Type;
            // Box Label:
            m_Label=BoxSrc.m_Label;
            // Number of Origin and Destiny Arcs:
            m_NumOriginArcs=BoxSrc.m_NumOriginArcs;
            m_NumDestinyArcs=BoxSrc.m_NumDestinyArcs;
            // Array of Indexes to Origin Arcs:
            if ((m_NumOriginArcs>0)&&(BoxSrc.mp_OriginArcIndex!=NULL))
                              {
                              mp_OriginArcIndex= new long[m_NumOriginArcs];
                              for (long i=0;i<m_NumOriginArcs;i++)
                                        {
                                        mp_OriginArcIndex[i]=BoxSrc.mp_OriginArcIndex[i];
                                        }
                              }
            else
                              {
                              mp_OriginArcIndex = NULL;
                              }
            // Array of Indexes to Destiny Arcs:
            if ((m_NumDestinyArcs>0)&&(BoxSrc.mp_DestinyArcIndex!=NULL))
                        {
                        mp_DestinyArcIndex= new long[m_NumDestinyArcs];
                        for (long i=0;i<m_NumDestinyArcs;i++)
                                  {
                                  mp_DestinyArcIndex[i]=BoxSrc.mp_DestinyArcIndex[i];
                                  }
                        }
            else
                        {
                        mp_DestinyArcIndex=NULL;
                        }

            // Flag: Do I have any mark inside this box?
            m_HasAnyMark=BoxSrc.m_HasAnyMark;
            // This Box Mark:
            m_BoxMark=BoxSrc.m_BoxMark;

////////////////////////////////////////////////////////////////

 mp_ConditionsArray  = NULL;
 mp_ThenAttributions = NULL;
 mp_ElseAttributions = NULL;

 if (BoxSrc.mp_ConditionsArray!=NULL)
   {
   mp_ConditionsArray = new CCondArray;
   *(mp_ConditionsArray) = *(BoxSrc.mp_ConditionsArray);
   }

 if (BoxSrc.mp_ThenAttributions!=NULL)
   {
   mp_ThenAttributions = new CAtbnMatrix;
   *(mp_ThenAttributions) = *(BoxSrc.mp_ThenAttributions);
```

```
  }

 if (BoxSrc.mp_ElseAttributions!=NULL)
  {
  mp_ElseAttributions = new CAtbnMatrix;
  *(mp_ElseAttributions) = *(BoxSrc.mp_ElseAttributions);
  }

//Old CCapacity Box Data Members:
        // Box Capacity:
        m_Capacity= BoxSrc.m_Capacity;
        // Box Array of Marks
        if (BoxSrc.mp_BoxMarkArray!=NULL)
                {
                mp_BoxMarkArray=new CMarkArray;
  for (long i=0;i<BoxSrc.mp_BoxMarkArray->GetSize();i++)
                {
    mp_BoxMarkArray->Add(BoxSrc.mp_BoxMarkArray->GetAt(i));
    }
                }
        else
                {
                mp_BoxMarkArray=NULL;
                }
        // Picking Order Preference:
        m_Fifo= BoxSrc.m_Fifo;
 m_Deadlock = BoxSrc.m_Deadlock;
 mp_Owner=BoxSrc.mp_Owner;

m_StartTime = BoxSrc.m_StartTime;
m_TimeCte = BoxSrc.m_TimeCte;
m_TimeCount = BoxSrc.m_TimeCount;
m_IsTimerRunning = BoxSrc.m_IsTimerRunning;
m_HasTimerRunned = BoxSrc.m_HasTimerRunned;

        }

        CBox& CBox::operator = (CBox& BoxSrc)
        {
        m_Type = BoxSrc.m_Type;
        // Box Label:
        m_Label=BoxSrc.m_Label;
        // Number of Origin and Destiny Arcs:
        m_NumOriginArcs=BoxSrc.m_NumOriginArcs;
        m_NumDestinyArcs=BoxSrc.m_NumDestinyArcs;
        // Array of Indexes to Origin Arcs:
        if ((m_NumOriginArcs>0)&&(BoxSrc.mp_OriginArcIndex!=NULL))
                        {
                        if (mp_OriginArcIndex!=NULL)
                                {
                                delete [] mp_OriginArcIndex;
                                mp_OriginArcIndex=NULL;
                                }
                        mp_OriginArcIndex= new long[m_NumOriginArcs];
                        for (long i=0;i<m_NumOriginArcs;i++)
                                {
                                mp_OriginArcIndex[i]=BoxSrc.mp_OriginArcIndex[i];
                                }
                        }
        else
                        {
                        mp_OriginArcIndex = NULL;
                        }
        // Array of Indexes to Destiny Arcs:
        if ((m_NumDestinyArcs>0)&&(BoxSrc.mp_DestinyArcIndex!=NULL))
                {
                if (mp_DestinyArcIndex!=NULL)
                                {
                                delete [] mp_DestinyArcIndex;
                                mp_DestinyArcIndex = NULL;
                                }
                mp_DestinyArcIndex= new long[m_NumDestinyArcs];
```

```
                for (long i=0;i<m_NumDestinyArcs;i++)
                        {
                        mp_DestinyArcIndex[i]=BoxSrc.mp_DestinyArcIndex[i];
                        }
                }
        else
                {
                mp_DestinyArcIndex=NULL;
                }

        // Flag: Do I have any mark inside this box?
        m_HasAnyMark=BoxSrc.m_HasAnyMark;
        // This Box Mark:
        m_BoxMark=BoxSrc.m_BoxMark;


//Old CTransformatorBox Data Members:
////////////////////////////////////////////////////////////////


 if (mp_ConditionsArray!=NULL)
  {
  delete mp_ConditionsArray;
  mp_ConditionsArray = NULL;
  }

 if (mp_ThenAttributions!=NULL)
  {
  delete mp_ThenAttributions;
  mp_ThenAttributions = NULL;
  }

 if (mp_ElseAttributions!=NULL)
  {
  delete mp_ElseAttributions;
  mp_ElseAttributions = NULL;
  }

 if (BoxSrc.mp_ConditionsArray!=NULL)
  {
  mp_ConditionsArray = new CCondArray;
  *(mp_ConditionsArray) = *(BoxSrc.mp_ConditionsArray);
  }

 if (BoxSrc.mp_ThenAttributions!=NULL)
  {
  mp_ThenAttributions = new CAtbnMatrix;
  *(mp_ThenAttributions) = *(BoxSrc.mp_ThenAttributions);
  }

 if (BoxSrc.mp_ElseAttributions!=NULL)
  {
  mp_ElseAttributions = new CAtbnMatrix;
  *(mp_ElseAttributions) = *(BoxSrc.mp_ElseAttributions);
  }

////////////////////////////////////////////////////////////////


//Old CCapacity Box Data Members:
        // Box Capacity:
        m_Capacity= BoxSrc.m_Capacity;
        // Box Array of Marks
        if (BoxSrc.mp_BoxMarkArray!=NULL)
                {
                if (mp_BoxMarkArray!=NULL)
                                {
                                delete mp_BoxMarkArray;
                                mp_BoxMarkArray = NULL;
                                }
                mp_BoxMarkArray=new CMarkArray;
  for (long i=0;i<BoxSrc.mp_BoxMarkArray->GetSize();i++)
```

```
                    {
    mp_BoxMarkArray->Add(BoxSrc.mp_BoxMarkArray->GetAt(i));
    }
                    }
          else
                    {
                    mp_BoxMarkArray=NULL;
                    }
          // Picking Order Preference:
          m_Fifo= BoxSrc.m_Fifo;
 m_Deadlock = BoxSrc.m_Deadlock;
 mp_Owner=BoxSrc.mp_Owner;

m_StartTime = BoxSrc.m_StartTime;
m_TimeCte = BoxSrc.m_TimeCte;
m_TimeCount = BoxSrc.m_TimeCount;
m_IsTimerRunning = BoxSrc.m_IsTimerRunning;
m_HasTimerRunned = BoxSrc.m_HasTimerRunned;


          return *this;
          }

          IMPLEMENT_SERIAL (CBox,CObject,1);

          void CBox::Serialize( CArchive& ar )
          {
          ASSERT(FALSE); // Not being used;
          }

          void CBox::ResetMarkFlag()
          {
          ASSERT(FALSE); // Not being used;
          }
BOOL CBox::IsAReadyPreCondition()
{
BOOL ReturnValue = FALSE;
switch (GetType())
 {
 case COMMON_BOX:
  {
  ReturnValue = HasMark();
  break;
  }
 case TEMP_BOX:
  {
  ReturnValue = HasMark()&&HasTimerRunned();
  break;
  }
 case CAPACITY_BOX:
  {
  ReturnValue = HasMark();
  break;
  }
 case TRANSFORMATOR_BOX:
  {
  ReturnValue = HasMark();
  break;
  }
 case PACKING_BOX:
  {
  ReturnValue = IsFull();
  break;
  }
 case UNPACKING_BOX:
  {
  ReturnValue = HasMark();
  if (m_Deadlock)
   {
   ReturnValue = FALSE;
   }
  }
```

```
    };
  return ReturnValue;
  }

  BOOL CBox::IsAReadyPosCondition()
  {
  BOOL ReturnValue = FALSE;
  switch (GetType())
   {
   case COMMON_BOX:
    {
    ReturnValue = !HasMark();
    break;
    }
   case TEMP_BOX:
    {
    ReturnValue = !HasMark();
    break;
    }
   case CAPACITY_BOX:
    {
    ReturnValue = !IsFull();
    break;
    }
   case TRANSFORMATOR_BOX:
    {
    ReturnValue = !HasMark();
    break;
    }
   case PACKING_BOX:
    {
    ReturnValue = !IsFull();
    break;
    }
   case UNPACKING_BOX:
    {
    ReturnValue = !HasMark();
    if (m_Deadlock)
     {
     ReturnValue = FALSE;
     }
    }
   };
 return ReturnValue;
 }

void CBox::ResetDeadlock()
{
m_Deadlock=FALSE;
}

BOOL CBox::IsDeadlock()
{
return m_Deadlock;
}


void CBox::SetBoxAsTemporized(long MyTimeCte)// only for Common Box
{
ASSERT(m_Type==COMMON_BOX);
m_Type = TEMP_BOX;
m_TimeCte = MyTimeCte;
m_IsTimerRunning = FALSE;
m_HasTimerRunned = FALSE;
m_TimeCount = 0;
}

void CBox::StartTimer(CTime MyTime)
{
m_StartTime = MyTime;
ResetTimer();
}
```

```
void CBox::UpdateTimer(CTime MyTime)
{
if (IsTimeCounting())
 {
 CTimeSpan DeltaT = MyTime-m_StartTime;
 m_TimeCount = DeltaT.GetTotalSeconds();
 if (m_TimeCount>=m_TimeCte)
  {
  m_HasTimerRunned = TRUE;
  }
 else
  {
  m_HasTimerRunned = FALSE;
  }
 }
else
 {
 StartTimer(MyTime);
 }
}
BOOL CBox::IsTimeCounting()
{
return m_IsTimerRunning;
}
void CBox::ResetTimer()
{
m_IsTimerRunning= FALSE;
m_HasTimerRunned = FALSE;
m_TimeCount = 0;
}

BOOL CBox::HasTimerRunned()
{
return m_HasTimerRunned;
}

void CBox::AddBlock(class CCondArray MyConditions, class CAtbnMatrix MyThenAtbn, class CAtbnMatrix MyElseAtbn)
{
ASSERT(m_Type==TRANSFORMATOR_BOX);
if (mp_ConditionsArray!=NULL)
 {
 delete mp_ConditionsArray;
 mp_ConditionsArray = NULL;
 }

if (mp_ThenAttributions!=NULL)
 {
 delete mp_ThenAttributions;
 mp_ThenAttributions = NULL;
 }

if (mp_ElseAttributions!=NULL)
 {
 delete mp_ElseAttributions;
 mp_ElseAttributions = NULL;
 }

mp_ConditionsArray  = new CCondArray;
mp_ThenAttributions = new CAtbnMatrix;
mp_ElseAttributions = new CAtbnMatrix;

*(mp_ConditionsArray)  = MyConditions;
*(mp_ThenAttributions) = MyThenAtbn;
*(mp_ElseAttributions) = MyElseAtbn;
}
```

Files Transit.h and Transit.cpp: E-MFG Transitions

```
// transit.h eMFG Transitions Definition:
//
// version:2.0
```

```
// Definition Date: 16/12/97
// Last Modified: 08/07/98
// programmed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira

/// Objetivos: Definir a estrutura de dados dos elementos de mudança de estado
// (transições) do grafo.

class CTransition:public CObject
{
// member variables:
protected:
 CGraph* mp_Owner;
        short m_Type;
        CString m_Label;// Transition Label
        long m_NumOriginArcs;
        long m_NumDestinyArcs;
        long* mp_OriginArcIndex;// Array of Indexes to Origin Arcs
        long* mp_DestinyArcIndex;// Array of Indexes to Destiny Arcs
        long m_NumGates;
        long* mp_GateIndex;// Array of Indexes to Gates

        BOOL m_IsPossibilyFireable;
        BOOL m_IsEnabled;
        BOOL m_IsFireable;
        BOOL m_IsFlaggedForNextStateChange;

        BOOL m_HasCondition;
 long m_ConditionIndex;

        BOOL m_IsASolvedConflict;

// Temporized Transitions Data Mambers:

 CTime m_StartTime;
 long m_TimeCte;
 long m_TimeCount;
 BOOL m_IsTimerRunning;
 BOOL m_HasTimerRunned;
 BOOL m_HasTimer;


public:
        // Construction & Destruction
        CTransition();

        CTransition(const CTransition& SrcTrans);
        CTransition& operator=(CTransition& SrcTrans);

        ~CTransition();


        // Build Functions
 void Create(CGraph* MyGraph,short MyType,CString MyLabel, long MyNumOrigin, long MyNumDestiny, long*
MyOriginArray, long* MyDestinyArray, long MyNumGates, long* MyGates);

        void AddOriginArc(long MyOriginArcIndex);
        void AddDestinyArc(long MyDestinyArcIndex);

        void RemoveOriginArc(long MyOriginArcIndex);
        void RemoveDestinyArc(long MyDestinyArcIndex);

        long GetNumOriginArcs();
        long GetNumDestinyArcs();
        long GetNumGates();

        long* GetOriginArcs();
        long* GetDestinyArcs();
        long* GetGates();

        BOOL IsChangeFlagged();
```

```
        BOOL SetChangeFlag(BOOL MyStatus);

    BOOL IsFireable();
        BOOL SetFireableFlag(BOOL MyStatus);

        BOOL IsEnabled();
        BOOL SetEnableFlag(BOOL MyStatus);

        BOOL IsPossiblyFireable();
        BOOL SetPossiblyFireableFlag(BOOL MyStatus);

        BOOL CanBeFired();
        void FireTransition(long MyMarkID=0);
        void ResetFlags();
        void UpdateFlags();

        CString GetLabel();

        BOOL IsASolvedConflict();
        BOOL SetConflictFlag(BOOL Status);

    void SetCondition(long MyCondition);

    // Temporized Transitions Functions (not implemented):
  public:
        void SetAsTemporized(long MyTimeCte);
    void StartTimer(CTime MyTime);
    void UpdateTimer(CTime MyTime);
    BOOL IsTimeCounting();
    void ResetTimer();
    BOOL HasTimerRunned();

};


// transit.cpp eMFG Transitions Implementation:
//
// version:2.0
// Definition Date: 16/12/97
// Last Modified: 08/07/98
// programmed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira

//
// Objetivos: Implementar a estrutura de dados dos elementos de  mudança de estado
// (transições) do grafo.


#include "stdafx.h"                     // standard windows aplication
#include "elements.h"   // E-MFG Standard Elements

CTransition::CTransition()
{
m_Type=COMMON_TRANSITION;// Transition Type
m_Label="";// Transition Label
m_NumOriginArcs=0;
m_NumDestinyArcs=0;
mp_OriginArcIndex=NULL;// Array of Indexes to Origin Arcs
mp_DestinyArcIndex=NULL;// Array of Indexes to Destiny Arcs
m_NumGates=0;
mp_GateIndex=NULL;// Array of Indexes to Gates
// Graph Dynamics Flags:
m_IsPossibilyFireable=FALSE;
m_IsEnabled=FALSE;
m_IsFireable=FALSE;
m_IsFlaggedForNextStateChange=FALSE;

m_HasCondition=FALSE;
m_ConditionIndex=-1;
m_IsASolvedConflict = FALSE;

//CTime m_StartTime;
```

```
m_TimeCte = 0;
m_TimeCount = 0;
m_IsTimerRunning = FALSE;
m_HasTimerRunned = FALSE;
m_HasTimer = FALSE;
}

CTransition::CTransition(const CTransition& SrcTrans)
{
mp_Owner = SrcTrans.mp_Owner;
m_Type=SrcTrans.m_Type;// Transition Type
m_Label=SrcTrans.m_Label;// Transition Label
m_NumOriginArcs=SrcTrans.m_NumOriginArcs;
m_NumDestinyArcs=SrcTrans.m_NumDestinyArcs;
if ((SrcTrans.mp_OriginArcIndex!=NULL)&&(SrcTrans.m_NumOriginArcs>0))
                    {
                    mp_OriginArcIndex = new long[m_NumOriginArcs];
                    for (long index=0;index<m_NumOriginArcs;index++)
                                        {
                                        mp_OriginArcIndex[index]=SrcTrans.mp_OriginArcIndex[index];// Array of Indexes
to Origin Arcs
                                        }
                    }
else
                    {
                    mp_OriginArcIndex=NULL;
                    }

if ((SrcTrans.mp_DestinyArcIndex!=NULL)&&(SrcTrans.m_NumDestinyArcs>0))
                    {
                    mp_DestinyArcIndex = new long[m_NumDestinyArcs];
                    for (long index=0;index<m_NumDestinyArcs;index++)
                                        {
                                        mp_DestinyArcIndex[index]=SrcTrans.mp_DestinyArcIndex[index];// Array of
Indexes to Destiny Arcs
                                        }
                    }
else
                    {
                    mp_DestinyArcIndex=NULL;
                    }

m_NumGates=SrcTrans.m_NumGates;

if ((SrcTrans.mp_GateIndex!=NULL)&&(SrcTrans.m_NumGates>0))
                    {
                    mp_GateIndex = new long[m_NumGates];
                    for (long index=0;index<m_NumGates;index++)
                                        {
                                        mp_GateIndex[index]=SrcTrans.mp_GateIndex[index];// Array of Indexes to Gates
                                        }
                    }
else
                    {
                    mp_GateIndex=NULL;
                    }

// Graph Dynamics Flags:
m_IsPossibilyFireable=SrcTrans.m_IsPossibilyFireable;
m_IsEnabled=SrcTrans.m_IsEnabled;
m_IsFireable=SrcTrans.m_IsFireable;
m_IsFlaggedForNextStateChange=SrcTrans.m_IsFlaggedForNextStateChange;

m_HasCondition=SrcTrans.m_HasCondition;
m_ConditionIndex=SrcTrans.m_ConditionIndex;
m_IsASolvedConflict=SrcTrans.m_IsASolvedConflict;

m_StartTime=SrcTrans.m_StartTime;
m_TimeCte =SrcTrans.m_TimeCte;
m_TimeCount =SrcTrans.m_TimeCount;
m_IsTimerRunning =SrcTrans.m_IsTimerRunning;
m_HasTimerRunned =SrcTrans.m_HasTimerRunned;
```

```
 m_HasTimer =SrcTrans.m_HasTimer;

 }

 CTransition& CTransition::operator=(CTransition& SrcTrans)
 {
 mp_Owner = SrcTrans.mp_Owner;
 m_Type=SrcTrans.m_Type;// Transition Type
 m_Label=SrcTrans.m_Label;// Transition Label
 m_NumOriginArcs=SrcTrans.m_NumOriginArcs;
 m_NumDestinyArcs=SrcTrans.m_NumDestinyArcs;
 if (mp_OriginArcIndex!=NULL)
                        {
                        delete [] mp_OriginArcIndex;
                        mp_OriginArcIndex=NULL;// Array of Indexes to Origin Arcs
                        }
 if ((SrcTrans.mp_OriginArcIndex!=NULL)&&(SrcTrans.m_NumOriginArcs>0))
                        {
                        mp_OriginArcIndex = new long[m_NumOriginArcs];
                        for (long index=0;index<m_NumOriginArcs;index++)
                                                {
                                                mp_OriginArcIndex[index]=SrcTrans.mp_OriginArcIndex[index];// Array of Indexes
 to Origin Arcs
                                                }
                        }
 else
                        {
                        mp_OriginArcIndex=NULL;
                        }
 if (mp_DestinyArcIndex!=NULL)
                        {
                        delete [] mp_DestinyArcIndex;
                        mp_DestinyArcIndex=NULL;// Array of Indexes to Destiny Arcs
                        }
 if ((SrcTrans.mp_DestinyArcIndex!=NULL)&&(SrcTrans.m_NumDestinyArcs>0))
                        {
                        mp_DestinyArcIndex = new long[m_NumDestinyArcs];
                        for (long index=0;index<m_NumDestinyArcs;index++)
                                                {
                                                mp_DestinyArcIndex[index]=SrcTrans.mp_DestinyArcIndex[index];// Array of
 Indexes to Destiny Arcs
                                                }
                        }
 else
                        {
                        mp_DestinyArcIndex=NULL;
                        }


 m_NumGates=SrcTrans.m_NumGates;
 if (mp_GateIndex!=NULL)
                        {
                        delete [] mp_GateIndex;
                        mp_GateIndex=NULL;// Array of Indexes to Gates
                        }
 if ((SrcTrans.mp_GateIndex!=NULL)&&(SrcTrans.m_NumGates>0))
                        {
                        mp_GateIndex = new long[m_NumGates];
                        for (long index=0;index<m_NumGates;index++)
                                                {
                                                mp_GateIndex[index]=SrcTrans.mp_GateIndex[index];// Array of Indexes to Gates
                                                }
                        }
 else
                        {
                        mp_GateIndex=NULL;
                        }

 // Graph Dynamics Flags:
 m_IsPossibilyFireable=SrcTrans.m_IsPossibilyFireable;
 m_IsEnabled=SrcTrans.m_IsEnabled;
 m_IsFireable=SrcTrans.m_IsFireable;
```

```
 m_IsFlaggedForNextStateChange=SrcTrans.m_IsFlaggedForNextStateChange;

 m_HasCondition=SrcTrans.m_HasCondition;
 m_IsASolvedConflict=SrcTrans.m_IsASolvedConflict;
 m_ConditionIndex=SrcTrans.m_ConditionIndex;

 m_StartTime=SrcTrans.m_StartTime;
 m_TimeCte =SrcTrans.m_TimeCte;
 m_TimeCount =SrcTrans.m_TimeCount;
 m_IsTimerRunning =SrcTrans.m_IsTimerRunning;
 m_HasTimerRunned =SrcTrans.m_HasTimerRunned;
 m_HasTimer =SrcTrans.m_HasTimer;

 return *this;
 }


 CTransition::~CTransition()
 {
 if (mp_OriginArcIndex!=NULL)
                        {
                        delete [] mp_OriginArcIndex;
                        mp_OriginArcIndex=NULL;// Array of Indexes to Origin Arcs
                        }
 if (mp_DestinyArcIndex!=NULL)
                        {
                        delete [] mp_DestinyArcIndex;
                        mp_DestinyArcIndex=NULL;// Array of Indexes to Destiny Arcs
                        }
 if (mp_GateIndex!=NULL)
                        {
                        delete [] mp_GateIndex;
                        mp_GateIndex=NULL;// Array of Indexes to Gates
                        }
 }


 void CTransition::Create(CGraph* MyGraph,short MyType,CString MyLabel, long MyNumOrigin, long MyNumDestiny, long*
 MyOriginArray, long* MyDestinyArray, long MyNumGates, long* MyGates)
 {
 mp_Owner = MyGraph;
 m_Type=MyType;// Transition Type
 m_Label=MyLabel;// Transition Label
 m_NumOriginArcs=MyNumOrigin;
 m_NumDestinyArcs=MyNumDestiny;
 if ((MyOriginArray!=NULL)&&(MyNumOrigin>0))
                        {
                        mp_OriginArcIndex = new long[m_NumOriginArcs];
                        for (long index=0;index<m_NumOriginArcs;index++)
                                                {
                                                mp_OriginArcIndex[index]=MyOriginArray[index];// Array of Indexes to Origin Arcs
                                                }
                        }
 else
                        {
                        mp_OriginArcIndex=NULL;
                        }

 if ((MyDestinyArray!=NULL)&&(MyNumDestiny>0))
                        {
                        mp_DestinyArcIndex = new long[m_NumDestinyArcs];
                        for (long index=0;index<m_NumDestinyArcs;index++)
                                                {
                                                mp_DestinyArcIndex[index]=MyDestinyArray[index];// Array of Indexes to Destiny
 Arcs
                                                }
                        }
 else
                        {
                        mp_DestinyArcIndex=NULL;
                        }
```

```
 m_NumGates=MyNumGates;

 if ((MyGates!=NULL)&&(MyNumGates>0))
                        {
                        mp_GateIndex = new long[m_NumGates];
                        for (long index=0;index<m_NumGates;index++)
                                                {
                                                mp_GateIndex[index]=MyGates[index];// Array of Indexes to Gates
                                                }
                        }
 else
                        {
                        mp_GateIndex=NULL;
                        }

 // Graph Dynamics Flags:
 m_IsPossibilyFireable=FALSE;
 m_IsEnabled=FALSE;
 m_IsFireable=FALSE;
 m_IsFlaggedForNextStateChange=FALSE;

 m_HasCondition=FALSE;
 m_IsASolvedConflict=FALSE;
 }

 void CTransition::AddOriginArc(long MyOriginArcIndex)
 {
 ASSERT(FALSE);
 }

 void CTransition::AddDestinyArc(long MyDestinyArcIndex)
 {
 ASSERT(FALSE);
 }

 void CTransition::RemoveOriginArc(long MyOriginArcIndex)
 {
 ASSERT(FALSE);
 }

 void CTransition::RemoveDestinyArc(long MyDestinyArcIndex)
 {
 ASSERT(FALSE);
 }

 long CTransition::GetNumOriginArcs()
 {
 return m_NumOriginArcs;
 }

 long CTransition::GetNumDestinyArcs()
 {
  return m_NumDestinyArcs;
 }

 long CTransition::GetNumGates()
 {
  return m_NumGates;
 }


 long* CTransition::GetOriginArcs()
 {
 return mp_OriginArcIndex;// Array of Indexes to Origin Arcs
 }

 long* CTransition::GetDestinyArcs()
 {
 return       mp_DestinyArcIndex;// Array of Indexes to Destiny Arcs
 }

 long* CTransition::GetGates()
```

```
{
return      mp_GateIndex;// Array of Indexes to Destiny Arcs
}


BOOL CTransition::IsChangeFlagged()
{
 return m_IsFlaggedForNextStateChange;
}

BOOL CTransition::SetChangeFlag(BOOL MyStatus)
{
 m_IsFlaggedForNextStateChange = MyStatus;
 return m_IsFlaggedForNextStateChange;
}

//////////////////////////////////////////////////////////////

BOOL CTransition::IsFireable()
{
// Verifica se existem condições booleanas associadas a essa transição:
            if (m_HasCondition)
                        {
                        ASSERT(FALSE);
   // if (mp_Owner->GetConditions()->GetAt(m_ConditionIndex).Evaluate())
   // {SetFireableFlag(TRUE);}
   // else {SetFireableFlag(FALSE);}
                        }
            else
                        {
                        SetFireableFlag(TRUE);
                        }
 return m_IsFireable;
}

BOOL CTransition::SetFireableFlag(BOOL MyStatus)
{
            m_IsFireable = MyStatus;
 return m_IsFireable;
}

BOOL CTransition::IsEnabled()
{
BOOL EnableStatus = TRUE;
long NumGates = GetNumGates();
long* p_GatesIndex = GetGates();
if (NumGates>0)
                        {
                        for (long index = 0; index<NumGates;index ++)
                                    {
                                    BOOL Temp = (mp_Owner->GetGates()->GetAt(p_GatesIndex[index])).GoAhead();
                                    EnableStatus = EnableStatus & Temp;
                                    }
                        SetEnableFlag(EnableStatus);
                        }
else
                        {
                        SetEnableFlag(EnableStatus);
                        }
 return m_IsEnabled;
}

BOOL CTransition::SetEnableFlag(BOOL MyStatus)
{
 m_IsEnabled=MyStatus;
 return m_IsEnabled ;
}

BOOL CTransition::IsPossiblyFireable()
{
            BOOL PreConditions = TRUE;
            BOOL PosConditions = TRUE;
```

```
		long MyNumPreConditions=GetNumOriginArcs();
		long MyNumPosConditions=GetNumDestinyArcs();
		long* MyOriginArcs = GetOriginArcs();
		long* MyDestinyArcs = GetDestinyArcs();

		for (long index = 0; index<MyNumPreConditions; index++)
				{
				long BoxIndex = mp_Owner->GetArcs()->GetAt(MyOriginArcs[index]).GetOrigin();
				BOOL Temp = mp_Owner->GetBoxes()->GetAt(BoxIndex).IsAReadyPreCondition();
				PreConditions = PreConditions & Temp;
				}

		for (index = 0; index<MyNumPosConditions; index++)
				{
				long BoxIndex = mp_Owner->GetArcs()->GetAt(MyDestinyArcs[index]).GetDestiny();
				BOOL Temp = (mp_Owner->GetBoxes()->GetAt(BoxIndex).IsAReadyPosCondition());// if has mark =
false else =true
				PosConditions = PosConditions & Temp;
				}

return SetPossiblyFireableFlag(PosConditions & PreConditions);
}

BOOL CTransition::SetPossiblyFireableFlag(BOOL MyStatus)
{
		m_IsPossibilyFireable=MyStatus;
 return m_IsPossibilyFireable;
}

BOOL CTransition::CanBeFired()
{
return m_IsPossibilyFireable & m_IsEnabled & m_IsFireable & m_IsFlaggedForNextStateChange;
}

void CTransition::FireTransition(long MyMarkID)
{
ASSERT (CanBeFired());
CMark MyMigratingMark;
MyMigratingMark.Create(mp_Owner,mp_Owner->GetAttribTemplates(), MyMarkID);
long MyNumPreConditions=GetNumOriginArcs();
long MyNumPosConditions=GetNumDestinyArcs();
long* MyOriginArcs = GetOriginArcs();
long* MyDestinyArcs = GetDestinyArcs();
BOOL First=TRUE;
for (long index = 0; index<MyNumPreConditions; index++)
		{
		long BoxIndex = mp_Owner->GetArcs()->GetAt(MyOriginArcs[index]).GetOrigin();
		CMark Temp = mp_Owner->GetBoxes()->ElementAt(BoxIndex).RemoveMark();
		if (First)
				{
				MyMigratingMark.SetID(Temp.GetID());
				First=FALSE;
				}
		MyMigratingMark.MergeMark(mp_Owner->GetArcs()->GetAt(MyOriginArcs[index]).ApplyFilter(Temp));
		}
for (index = 0; index<MyNumPosConditions; index++)
		{
		long BoxIndex = mp_Owner->GetArcs()->GetAt(MyDestinyArcs[index]).GetDestiny();
		mp_Owner->GetBoxes()->ElementAt(BoxIndex).AddMark(mp_Owner->GetArcs()-
>GetAt(MyDestinyArcs[index]).ApplyFilter(MyMigratingMark));
		}
ResetFlags();
ResetTimer();
}

void CTransition::ResetFlags()
{
SetChangeFlag(FALSE);
SetEnableFlag(FALSE);
SetFireableFlag(FALSE);
SetPossiblyFireableFlag(FALSE);
SetConflictFlag(FALSE);
```

```
}

void CTransition::UpdateFlags()
{
SetChangeFlag(IsEnabled()&IsFireable()&IsPossiblyFireable());
if (CanBeFired()&&m_HasTimer)
 {
 if (!IsTimeCounting())
  {
  m_IsTimerRunning = TRUE;
  SetChangeFlag(FALSE);
  }
 else
  {
  if (HasTimerRunned())
   {
   SetChangeFlag(TRUE);
   }
  else
   {
   SetChangeFlag(FALSE);
   }
  }
 }
}

CString CTransition::GetLabel()
{
return m_Label;
}

BOOL CTransition::IsASolvedConflict()
{
return m_IsASolvedConflict;
}

BOOL CTransition::SetConflictFlag(BOOL Status)
{
m_IsASolvedConflict = Status;
return m_IsASolvedConflict;
}

void CTransition::SetCondition(long MyCondition)
{
ASSERT(MyCondition>=0);
m_ConditionIndex = MyCondition;
m_HasCondition = TRUE;
}


void CTransition::SetAsTemporized(long MyTimeCte)
{
ASSERT(MyTimeCte>0);
m_TimeCte = MyTimeCte;
m_IsTimerRunning = FALSE;
m_HasTimerRunned = FALSE;
m_TimeCount = 0;
m_HasTimer = TRUE;
}

void CTransition::StartTimer(CTime MyTime)
{
m_StartTime = MyTime;
ResetTimer();
}

void CTransition::UpdateTimer(CTime MyTime)
{
if (IsTimeCounting())
 {
 CTimeSpan DeltaT = MyTime-m_StartTime;
 m_TimeCount = DeltaT.GetTotalSeconds();
```

```
 if (m_TimeCount>=m_TimeCte)
  {
  m_HasTimerRunned = TRUE;
  }
 else
  {
  m_HasTimerRunned = FALSE;
  }
 }
else
 {
 StartTimer(MyTime);
 }
}

BOOL CTransition::IsTimeCounting()
{
return m_IsTimerRunning;
}

void CTransition::ResetTimer()
{
m_IsTimerRunning= FALSE;
m_HasTimerRunned = FALSE;
m_TimeCount = 0;
}

BOOL CTransition::HasTimerRunned()
{
return m_HasTimerRunned;
}
```

Files Arcs.h and Arcs.cpp: E-MFG Arcs

```
// arcs.h - eMFG Arcs Definition:
// version: 2.0
// Definition Date: 11/12/97
// Last Modified:08/07/98
// programed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira


// Objetivos: Definir a estrutura de dados dos elementos de conexão
// do grafo (arcos direcionais)

#include"arcfilt.h"


class CArc : public CObject
{
protected:
  class CGraph* mp_Owner;                          // Owner Graph
            short m_OriginType;       // Tipo de Origem
            long m_OriginIndex;       // Índice da Origem
            long m_DestinyIndex[2];    // Índices dos destinos
  CString m_Label;          // nome do arco
  CArcFilter m_Filter;  // filtro do arco

public:
            // Default Construction & Destruction
            CArc();
            ~CArc();


            // Build Functions

  void Create(class CGraph* MyOwner, CString MyLabel,short MyOriginType, long MyOriginIndex, long MyDestinyIndex);

            // Data Management
            long GetOrigin();
            long GetDestiny();
```

```
        void SetOrigin(long MyOrigin);
        void SetDestiny(long MyDestiny); //single destiny or condition = TRUE Destiny

 void SetFilter(CArcFilter MyFilter);

//      CMark AplyFilter(CMark& SrcMark);
        CMark ApplyFilter(CMark& SrcMark);

        CArc( const CArc& ArcSrc );
        CArc& operator = (const CArc& ArcSrc);
};


// arcs.cpp - eMFG Arcs Implementation:
// version: 2.0
// Definition Date: 11/12/97
// Last Modified: 08/07/98
// programed by: Daniel M. S. Ferreira
// last modified by: Daniel M. S. Ferreira

// Objetivos: Definir a estrutura de dados dos elementos de conexão
// do grafo (arcos direcionais)


#include "stdafx.h"                         // standard windows aplication
#include "elements.h"   //E-MFG Standard Elements

CArc::CArc()
{
mp_Owner=NULL;        // owner graph pointer
m_OriginType=UNKNOWN;  // Arc's origin type
m_OriginIndex = -1;    // origin index
m_DestinyIndex[0] = -1; // default destiny index
m_DestinyIndex[1] = -1; // secondary destiny index
m_Label = "";         // Arc's name (used for dump purposes)
}

CArc::~CArc()
{
// the graph pointer should never be deleted by a structural element
}


CArc::CArc( const CArc& ArcSrc )
{
// copy constructor implementation
ASSERT(ArcSrc.mp_Owner!=NULL);
mp_Owner=ArcSrc.mp_Owner;                 // owner graph pointer
m_OriginType=ArcSrc.m_OriginType;          //Arc's origin type
m_OriginIndex = ArcSrc.m_OriginIndex;      // origin index
m_DestinyIndex[0] = ArcSrc.m_DestinyIndex[0]; // default destiny index
m_DestinyIndex[1] = ArcSrc.m_DestinyIndex[1]; // secondary destiny index
m_Label = ArcSrc.m_Label;                 // Arc's name (used for dump purposes)
m_Filter = ArcSrc.m_Filter;               // Arc's Mark Filter

}

CArc& CArc::operator = (const CArc& ArcSrc)
{
// this operator makes a copy of the right side attribution element
ASSERT(ArcSrc.mp_Owner!=NULL);
mp_Owner=ArcSrc.mp_Owner;                 // owner graph pointer
m_OriginType=ArcSrc.m_OriginType;          //Arc's origin type
m_OriginIndex = ArcSrc.m_OriginIndex;      // origin index
m_DestinyIndex[0] = ArcSrc.m_DestinyIndex[0]; // default destiny index
m_DestinyIndex[1] = ArcSrc.m_DestinyIndex[1]; // secondary destiny index
m_Label = ArcSrc.m_Label;                 // Arc's name (used for dump purposes)
m_Filter = ArcSrc.m_Filter;               // Arc's Mark Filter

return *this;
}
```

```
void CArc::Create(class CGraph* MyOwner, CString MyLabel,short MyOriginType, long MyOriginIndex, long MyDestinyIndex)
{
// this function creates a fully functional arc
ASSERT(MyOwner!=NULL);
mp_Owner=MyOwner; //the owner graph
m_Label = MyLabel; // Arc's name
m_OriginType=MyOriginType;// Arc's origin Type
m_OriginIndex = MyOriginIndex;// origin index
m_DestinyIndex[0] = MyDestinyIndex;// default destiny index
m_DestinyIndex[1] = -1;// secondary destiny index
m_Filter.Create(mp_Owner,PASS_ALL,0,NULL,PASS_COMPOSITE); // The filter defaults to PASS_ALL
}

long CArc::GetOrigin()
{
// return the arc's origin index here
return m_OriginIndex;
}

long CArc::GetDestiny()
{
// return the arc's actual destiny index here
return m_DestinyIndex[0];
}

void CArc::SetOrigin(long MyOrigin)
{
// updates arc's origin (not used for the E-MFG controller program)
m_OriginIndex = MyOrigin;
}

void CArc::SetFilter(CArcFilter MyFilter)
{
// updates arc's filter
m_Filter = MyFilter;
}

void CArc::SetDestiny(long MyDestiny)
{
// updates arc's destiny (not used for the E-MFG controller program)
// any logic related to changing dinamically the arc's destiny should be added
// to this function
// the arc has a pointer to the graph so it can modify the other references in the graph
// to it
m_DestinyIndex[0] = MyDestiny;
}

CMark CArc::ApplyFilter(CMark& SrcMark)
{
return m_Filter.ApplyFilter(SrcMark);
//return SrcMark;
}
```

Files Gates.h and Gates.cpp: E-MFG Gates

```
// gates.h EMFG Gates Definition:
//
// version:2.0a
// Definition Date: 27/05/98
// Last Modified: 01/06/98
// programed by: Marco A. A. Silva

// Objetivos: Definir a estrutura de dados dos elementos de I/O
// (gates) do grafo.


class CGate:public CObject
{
// empty constructor / destructor
public:
        CGate();
        ~CGate();
```

```
        void Create (class CGraph* MyGraph, short MyType, short MyRetType, CString MyLabel, short MyBox, short
MyAttrib, short MyTransition, CString MyMethod, CString MyParameter, short myActivation);

// Copy Constructor:
        CGate( const CGate& GateSrc );
// Operator =
        CGate& operator=(const CGate& GateSrc);

// Function Members:
public:
        short           GetType();
        CString GetLabel();
        short           GetBox();
 short  GetAttrib();
        short           GetTransition();
        CString GetMethod();
        CString GetParameter();
 short  GetRetType();
 short  GetActivation();

public:
        void            SetBOOL(BOOL arg);
 BOOL           GoAhead();
        void            SetString(CString arg);
        CString GetString();
        void            SetInteger(short arg);
        long            GetInteger();

public:
        void            AddConditional(CConditional* pCond);
        void            RemoveConditional();

// Data Members:
protected:
        class CGraph* mp_Owner;                                                         //
Owner Graph
        short   m_Type;
                        //tipo do gate
        CString m_Label;
                //nome do gate
 short  m_Box;
                //box associada
 short  m_Attrib;            //atributo associado
        short   m_Transition;
        //transição associada
        CString m_Method;
                //método de comunicação
        CString m_Parameter;
        //parâmetro de comunicação
 short  m_Activation;        //tipo de ativação
        BOOL            m_boolean;
                        //boolean return value
        CString m_string;
                //string return value
        long            m_integer;
                //integer return value
 short  m_rettype;          //return type

        CConditional* mp_Cond;
        //associated conditional structure
};

// gates.cpp eMFG Gates Implementation:
//
// version:2.0
// Definition Date: 27/05/98
// Last Modified: 15/06/98
// Programmed by: Marco A. A. Silva
// Last modified by: Marco A. A. Silva
//
// Objetivos: Implementar a estrutura de dados dos elementos de I/O
// (gates) do grafo. Está prevista a comunicação com outros programas
```

```
// via DDE através dos valores registrados nesses elementos.

#include "stdafx.h"                    //standard windows aplication
#include "elements.h"          // EMFG Elements include list


CGate::CGate()
{
        mp_Owner                = NULL;
        mp_Cond                 = NULL;
        m_Type                  = NULO;
        m_Label                 = "";
        m_Box                           = NULO;
 m_Attrib   = NULO;
        m_Transition= NULO;
        m_Method            = "";
        m_Parameter = "";
        m_boolean           = FALSE;
        m_string            = "";
        m_integer           = NULO;
 m_rettype  = NULO;
}

CGate::CGate( const CGate& GateSrc )
{                                                                       //Owner Graph
        mp_Owner   = GateSrc.mp_Owner;
        m_Type     = GateSrc.m_Type;
                //tipo do gate
        m_Label    = GateSrc.m_Label;
        //nome do gate
 m_Box      = GateSrc.m_Box;
        //box associada
 m_Attrib   = GateSrc.m_Attrib;          //atributo associado
        m_Transition= GateSrc.m_Transition;
        //transição associada
        m_Method   = GateSrc.m_Method;
                //método de comunicação
        m_Parameter = GateSrc.m_Parameter;
        //parâmetro de comunicação
 m_Activation= GateSrc.m_Activation;        //tipo de ativação
        m_boolean  = GateSrc.m_boolean;
        //boolean return value
        m_string   = GateSrc.m_string;
        //string return value
        m_integer  = GateSrc.m_integer;
        //integer return value
 m_rettype  = GateSrc.m_rettype;          //return type

 mp_Cond = NULL;

        if (GateSrc.mp_Cond!=NULL)
  {
  mp_Cond = new CConditional;
  *(mp_Cond) = *(GateSrc.mp_Cond);
  }
}

CGate& CGate::operator=(const CGate& GateSrc)
{                                                                       //Owner Graph
        mp_Owner   = GateSrc.mp_Owner;
        m_Type     = GateSrc.m_Type;
                //tipo do gate
        m_Label    = GateSrc.m_Label;
        //nome do gate
 m_Box      = GateSrc.m_Box;
        //box associada
 m_Attrib   = GateSrc.m_Attrib;          //atributo associado
        m_Transition= GateSrc.m_Transition;
        //transição associada
        m_Method   = GateSrc.m_Method;
                //método de comunicação
```

```
        m_Parameter = GateSrc.m_Parameter;
        //parâmetro de comunicação
m_Activation= GateSrc.m_Activation;          //tipo de ativação
        m_boolean   = GateSrc.m_boolean;
        //boolean return value
        m_string    = GateSrc.m_string;
        //string return value
        m_integer   = GateSrc.m_integer;
        //integer return value
m_rettype   = GateSrc.m_rettype;             //return type

if (mp_Cond!=NULL)
  {
  delete mp_Cond;
  mp_Cond = NULL;
  }

        if (GateSrc.mp_Cond!=NULL)
  {
  mp_Cond = new CConditional;
  *(mp_Cond) = *(GateSrc.mp_Cond);
  }

return *this;
}

CGate::~CGate()
{
        if (mp_Cond!=NULL)
  {
                delete mp_Cond;
  mp_Cond=NULL;
  }
}

//construtor genérico (ver usos dentro de cada tipo)
void CGate::Create (class CGraph* MyGraph, short MyType, short MyRetType, CString MyLabel, short MyBox, short MyAttrib,
short MyTransition, CString MyMethod, CString MyParameter, short myActivation)
{
        mp_Owner = MyGraph;

        m_Type  = MyType;
        m_Label = MyLabel;

        switch (m_Type)
                {
                default:
                        {
                        ASSERT (0);                          //nenhum tipo definido foi especificado
                        break;
                        }

                //exemplo.Create(INT_PERMIT,BOOLEAN,"Permit1",iBox,iTrans,"INTERNAL","");
                case INT_PERMIT:
                        {
                        m_Box = MyBox;
    m_Attrib = NULO;
                        m_Transition = MyTransition;
                        ASSERT (MyMethod=="INTERNAL");
                        m_Method = "INTERNAL";
                        m_Parameter = "";
    m_Activation = NULO;
    m_rettype = BOOLEAN;
                        break;
                        }

                //exemplo.Create(INT_NOT_PERMIT,BOOLEAN,"NotPermit1",iBox,iTrans,"INTERNAL","");
                case INT_NOT_PERMIT:
                        {
                        m_Box = MyBox;
    m_Attrib = NULO;
                        m_Transition = MyTransition;
```

```
                    ASSERT (MyMethod=="INTERNAL");
                    m_Method = "INTERNAL";
                    m_Parameter = "";
m_Activation = NULO;
m_rettype = BOOLEAN;
                    break;
                    }

            //exemplo.Create(INT_FULL_PERMIT,BOOLEAN,"Permit1",iBox,iTrans,"INTERNAL","");
            case INT_FULL_PERMIT:
                    {
                    m_Box = MyBox;
m_Attrib = NULO;
                    m_Transition = MyTransition;
                    ASSERT (MyMethod=="INTERNAL");
                    m_Method = "INTERNAL";
                    m_Parameter = "";
m_Activation = NULO;
m_rettype = BOOLEAN;
                    break;
                    }

            //exemplo.Create(INT_FULL_NOT_PERMIT,BOOELAN,"Permit1",iBox,iTrans,"INTERNAL","");
            case INT_FULL_NOT_PERMIT:
                    {
                    m_Box = MyBox;
m_Attrib = NULO;
                    m_Transition = MyTransition;
                    m_Method = "INTERNAL";
                    m_Parameter = "";
m_Activation = NULO;
m_rettype = BOOLEAN;
                    break;
                    }


    //exemplo.Create(EXTERNAL_INPUT_PERMIT,BOOLEAN,"Permit2",NULO,iTrans,"DDE","Grafo!gate");
            case EXTERNAL_INPUT_PERMIT:
                    {
                    m_Box = NULO;
m_Attrib = NULO;
                    m_Transition = MyTransition;
                    m_Method = MyMethod;
                    m_Parameter = MyParameter;
m_Activation = myActivation;
m_rettype = BOOLEAN;
                    break;
                    }


    //exemplo.Create(EXTERNAL_INPUT_NOT_PERMIT,BOOLEAN,"Permit2",NULO,iTrans,"DDE","Grafo!gate");
            case EXTERNAL_INPUT_NOT_PERMIT:
                    {
                    m_Box = NULO;
m_Attrib = NULO;
                    m_Transition = MyTransition;
                    m_Method = MyMethod;
                    m_Parameter = MyParameter;
m_Activation = myActivation;
m_rettype = BOOLEAN;
                    break;
                    }

            //exemplo.Create(EXTERNAL_OUTPUT_PERMIT,BOOLEAN,"Permit2",iBox,NULO,"DDE","");
            case EXTERNAL_OUTPUT_PERMIT:
                    {
                    m_Box = MyBox;
m_Attrib = NULO;
                    m_Transition = NULO;
                    m_Method = MyMethod;
                    m_Parameter = MyParameter;
m_Activation = myActivation;
```

```
m_rettype = BOOLEAN;
                        break;
                        }

                //exemplo.Create(EXTERNAL_OUTPUT_NOT_PERMIT,BOOLEAN,"Permit2",iBox,NULO,"DDE","");
                case EXTERNAL_OUTPUT_NOT_PERMIT:
                        {
                        m_Box = MyBox;
m_Attrib = NULO;
                        m_Transition = NULO;
                        m_Method = MyMethod;
                        m_Parameter = MyParameter;
m_Activation = myActivation;
m_rettype = BOOLEAN;
                        break;
                        }

                //exemplo.Create(EXTERNAL_OUTPUT_FULL_PERMIT,BOOLEAN,"Permit2",iBox,NULO,"DDE","");
                case EXTERNAL_OUTPUT_FULL_PERMIT:
                        {
                        m_Box = MyBox;
m_Attrib = NULO;
                        m_Transition = NULO;
                        m_Method = MyMethod;
                        m_Parameter = MyParameter;
m_Activation = myActivation;
m_rettype = BOOLEAN;
                        break;
                        }


        //exemplo.Create(EXTERNAL_OUTPUT_FULL_NOT_PERMIT,BOOLEAN,"Permit2",iBox,NULO,"DDE","");
                case EXTERNAL_OUTPUT_FULL_NOT_PERMIT:
                        {
                        m_Box = MyBox;
m_Attrib = NULO;
                        m_Transition = NULO;
                        m_Method = MyMethod;
                        m_Parameter = MyParameter;
m_Activation = myActivation;
m_rettype = BOOLEAN;
                        break;
                        }


        //exemplo.Create(INT_DATA,TEXT_ATRIB/INTEGER_ATRIB,"Data1",NULO,NULO,"INTERNAL","box!atributo);
                case INT_DATA:
                        {
//outcasted
}

                //exemplo.Create(INT_N,INTEGER_ATRIB,"Data1",iBox,NULO,"INTERNAL","");
                case INT_N:
                        {
                        m_Box = MyBox;
m_Attrib = NULO;
                        m_Transition = NULO;
                        m_Method = "INTERNAL";
                        m_Parameter = "";
m_rettype = INTEGER_ATRIB;
                        break;
                        }


        //exemplo.Create(EXTERNAL_OUTPUT_DATA,TEXT_ATRIB/INTEGER_ATRIB,"ExtData1",NULO,NULO,"DDE
","box!atributo");
                case EXTERNAL_OUTPUT_DATA:
                        {
                        m_Box = MyBox;
m_Attrib = MyAttrib;
                        m_Transition = NULO;
                        m_Method = MyMethod;
```

```
                                m_Parameter = MyParameter;
    m_Activation = myActivation;
    m_rettype = MyRetType;
                                break;
                                }

                        //exemplo.Create(EXTERNAL_OUTPUT_N,INTEGER_ATRIB,"ExtData1",iBox,NULO,"DDE",);
                        case EXTERNAL_OUTPUT_N:
                                {
                                m_Box = MyBox;
    m_Attrib = NULO;
                                m_Transition = NULO;
                                m_Method = MyMethod;
                                m_Parameter = MyParameter;
    m_Activation = myActivation;
    m_rettype = INTEGER_ATRIB;
                                break;
                                }


            //exemplo.Create(EXTERNAL_INPUT_DATA,TEXT_ATRIB/INTEGER_ATRIB,"ExtData2",NULO,NULO,"DDE","
grafo!gate");
                        case EXTERNAL_INPUT_DATA:
                                {
                                m_Box = NULO;
    m_Attrib = NULO;
                                m_Transition = NULO;
                                m_Method = MyMethod;
                                m_Parameter = MyParameter;
    m_Activation = myActivation;
    m_rettype = MyRetType;
                                break;
                                }
                        }
}


//retorna o tipo
short CGate::GetType()
{
        return m_Type;
}

//retorna o nome
CString CGate::GetLabel()
{
        return m_Label;
}

//retorna a box associada
short CGate::GetBox()
{
        return m_Box;
}

//retorna o atributo associado
short CGate::GetAttrib()
{
        return m_Attrib;
}

//retorna a transição associada
short CGate::GetTransition()
{
        return m_Transition;
}

//retorna o tipo de retorno
short CGate::GetRetType()
{
        return m_rettype;
}
```

```
//retorna o método
CString CGate::GetMethod()
{
        return m_Method;
}

//retorna os parâmetros
CString CGate::GetParameter()
{
        return m_Parameter;
}


//funções de acesso
void CGate::SetBOOL(BOOL arg)
{
        ASSERT( (m_Type==INT_PERMIT)||(m_Type==INT_NOT_PERMIT)||
    (m_Type==INT_FULL_PERMIT)||(m_Type==INT_FULL_NOT_PERMIT)||
    (m_Type==EXTERNAL_INPUT_PERMIT)||(m_Type==EXTERNAL_INPUT_NOT_PERMIT)||
    (m_Type==EXTERNAL_OUTPUT_PERMIT)||(m_Type==EXTERNAL_OUTPUT_NOT_PERMIT)||
    (m_Type==EXTERNAL_OUTPUT_FULL_PERMIT)||(m_Type==EXTERNAL_OUTPUT_FULL_NOT_PERMIT));

        m_boolean = arg;
}

BOOL CGate::GoAhead()
{
        ASSERT( (m_Type==INT_PERMIT)||(m_Type==INT_NOT_PERMIT)||
    (m_Type==INT_FULL_PERMIT)||(m_Type==INT_FULL_NOT_PERMIT)||
    (m_Type==EXTERNAL_INPUT_PERMIT)||(m_Type==EXTERNAL_INPUT_NOT_PERMIT)||
    (m_Type==EXTERNAL_OUTPUT_PERMIT)||(m_Type==EXTERNAL_OUTPUT_NOT_PERMIT)||
    (m_Type==EXTERNAL_OUTPUT_FULL_PERMIT)||(m_Type==EXTERNAL_OUTPUT_FULL_NOT_PERMIT));

        if (mp_Cond!=NULL)
                if (!mp_Cond->Evaluate())
                        return (FALSE);

        return(m_boolean);
}

short CGate::GetActivation()
{
 return m_Activation;
}

void CGate::SetString(CString arg)
{
        ASSERT((m_Type==INT_DATA)||(m_Type==INT_N)||(m_Type==EXTERNAL_INPUT_DATA)||
    (m_Type==EXTERNAL_OUTPUT_DATA)||(m_Type==EXTERNAL_OUTPUT_N));

        m_string = arg;
}

CString CGate::GetString()
{
        ASSERT((m_Type==INT_DATA)||(m_Type==INT_N)||(m_Type==EXTERNAL_INPUT_DATA)||
            (m_Type==EXTERNAL_OUTPUT_DATA)||(m_Type==EXTERNAL_OUTPUT_N));


        if (mp_Cond!=NULL)
                if (!mp_Cond->Evaluate())
                        return ("");

        return m_string;
}

void CGate::SetInteger(short arg)
{
        ASSERT((m_Type==INT_DATA)||(m_Type==INT_N)||(m_Type==EXTERNAL_INPUT_DATA)||
            (m_Type==EXTERNAL_OUTPUT_DATA)||(m_Type==EXTERNAL_OUTPUT_N));
```

```
        m_integer = arg;
}

long CGate::GetInteger()
{
        ASSERT((m_Type==INT_DATA)||(m_Type==INT_N)||(m_Type==EXTERNAL_INPUT_DATA)||
            (m_Type==EXTERNAL_OUTPUT_DATA)||(m_Type==EXTERNAL_OUTPUT_N));

        if (mp_Cond!=NULL)
                if (!mp_Cond->Evaluate())
                        return (NULO);

        return m_integer;
}

void CGate::AddConditional(CConditional* pCond)
{
        if (mp_Cond!=NULL)
  {
                delete mp_Cond;
  mp_Cond=NULL;
  }

 mp_Cond = new CConditional;
        *(mp_Cond) = *(pCond);
}

void CGate::RemoveConditional()
{
        if (mp_Cond!=NULL)
  {
                delete mp_Cond;
  mp_Cond=NULL;
  }
}
```

C - Estrutura do Grafo

Files Graph.h and Graph.cpp - E-MFG Graph

```
// graph.h - eMFG Graph header file
// version: 2.00
// Definition Date: 22/03/98
// Last Modified: 08/07/98
// programed by: Marco A. A. Silva
// last modified by: Daniel M. S. Ferreira

// Objetivos: Agrupar todas os objetos de um grafo

class CGraph : public CObject
{
public:
        CGraph();

public:
        CString m_title;
        CString m_descr;
private:

        class CMarkAttribArray* mp_MarkAttributesArray;
        class CArcArray* mp_ArcsArray;
        class CGateArray* mp_GatesArray;
        class CTransitionArray* mp_TransitionsArray;
        class CBoxArray* mp_BoxesArray;

public:
        ~CGraph();
        class CBoxArray* GetBoxes();
        class CArcArray* GetArcs();
        class CTransitionArray* GetTransitions();
        class CGateArray* GetGates();
        class CMarkAttribArray* GetAttribTemplates();
```

```
 void GetBoxConnectionsDump(class CLinesArray* p_LinesVector);
        void GetBoxesPropertiesDump(class CLinesArray* p_LinesVector);
};

// graph.cpp - eMFG Graph implementation file
// version: 2.00
// Definition Date: 22/03/98
// Last Modified: 08/07/98
// programed by: Marco A. A. Silva
// last modified by: Daniel M. S. Ferreira

// Objetivos: Agrupar todas os objetos de um grafo

#include "stdafx.h"                         // standard windows aplication

#include "definitions.h"                    //E-MFG Constant Definitions
#include "wordsarray.h"                         //Extension of String Arrays
#include "basis.h"                                  //E-MFG Basis Class
#include "graph.h"                                  //E-MFG Graph Components
#include "marks.h"                                  //E-MFG Marks Definition
#include "arcs.h"                                       //E-MFG Arcs Definition
#include "boxes.h"                                  //E-MFG Boxes Definition
#include "transit.h"                            //E-MFG Transition Definition
#include "gates.h"                                  //E-MFG Gates Definition
#include "list.h"                                       //Linked List Definition
#include "arrays.h"                                 //Arrays Extension Class

CGraph::CGraph()
{
        m_title = "";
        m_descr = "";
        mp_MarkAttributesArray = new CMarkAttribArray;
        mp_ArcsArray = new CArcArray;
        mp_GatesArray = new CGateArray;
        mp_TransitionsArray = new CTransitionArray;
        mp_BoxesArray= new CBoxArray;
}

CGraph::~CGraph()
{
        delete mp_MarkAttributesArray;
        delete mp_ArcsArray;
        delete mp_GatesArray;
        delete mp_TransitionsArray;
        delete mp_BoxesArray;

}

CMarkAttribArray* CGraph::GetAttribTemplates()
{
return mp_MarkAttributesArray;
}

class CArcArray* CGraph::GetArcs()
{
return mp_ArcsArray;
}

CBoxArray* CGraph::GetBoxes()
{
return mp_BoxesArray;
}

class CGateArray* CGraph::GetGates()
{
return mp_GatesArray;
}

class CTransitionArray* CGraph::GetTransitions()
{
return mp_TransitionsArray;
```

```
}

void CGraph::GetBoxesPropertiesDump(class CLinesArray* p_LinesVector)
{
for (long index=0;index<GetBoxes()->GetSize();index++)
                        {
                        GetBoxes()->ElementAt(index).GetDump(p_LinesVector);
                        }
}

void CGraph::GetBoxConnectionsDump(class CLinesArray* p_LinesVector)
{
CWordsArray* p_Line;

for (long index=0;index<GetBoxes()->GetSize();index++)
                        {
                        p_Line = new CWordsArray;
                        long NumOrigins = GetBoxes()->ElementAt(index).GetNumOriginArcs();
                        long* OriginIndex = GetBoxes()->ElementAt(index).GetOriginTransitionsIndexList(GetArcs());
                        CString Origins;
                        for (long k=0;k<NumOrigins;k++)
                                        {
                                        if(k==0)
                                                        {
                                                        Origins = "[ ";
                                                        }
                                        Origins = Origins + GetTransitions()->ElementAt(OriginIndex[k]).GetLabel();
                                        if (k<NumOrigins-1)
                                                        {
                                                        Origins += ", ";
                                                        }
                                        else
                                                        {
                                                        Origins += " ]";
                                                        }
                                        }
                        if (NumOrigins==0)
                                        {
                                        Origins = "[]";
                                        p_Line->Add(Origins);
                                        }
   else
     {
                        p_Line->Add(Origins);
     }
                        p_Line->Add("->");
                        CString Label;
                        if (GetBoxes()->ElementAt(index).HasMark())
                                        {
                                        Label = "** " + GetBoxes()->ElementAt(index).GetLabel() + " **";
                                        }
                        else
                                        {
                                        Label = GetBoxes()->ElementAt(index).GetLabel();
                                        }
                        p_Line->Add(Label);
                        p_Line->Add("->");
                        long NumDestiny = GetBoxes()->ElementAt(index).GetNumDestinyArcs();
                        long* DestinyIndex = GetBoxes()-
>ElementAt(index).GetDestinyTransitionsIndexList(GetArcs(),GetGates(),GetBoxes());
                        CString Destiny;
                        for (k=0;k<NumDestiny;k++)
                                        {
                                        if (k==0)
                                                        {
                                                        Destiny = "[ ";
                                                        }
                                        Destiny = Destiny + GetTransitions()->ElementAt(DestinyIndex[k]).GetLabel();
                                        if (k<NumDestiny-1)
                                                        {
                                                        Destiny += ", ";
```

```
                                        }
                                else
                                        {
                                        Destiny += " ]";
                                        }
                                }
                        if (NumDestiny==0)
                                {
                                Destiny = "[]";
                                p_Line->Add(Destiny);
                                }
  else
     {
     p_Line->Add(Destiny);
     }
                        p_LinesVector->Add(*p_Line);
     if (DestinyIndex!=NULL)
       delete []DestinyIndex;
     if (OriginIndex!=NULL)
       delete []OriginIndex;
                        delete p_Line;
                        }

}
```

D - Gerenciador de Marcas e Controlador de I/O

Files Manager.h and Manager.cpp - E-MFG Mark Manager

```
// manager.h eMFG MarkManager Definition:
// version:1.0a
// Definition Date: 07/03/98
// Daniel M. S. Ferreira
// Last Modified: 08/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira

// Objetivos: Definir a estrutura do Gerenciador de Marcas do grafo



// Mark Manager

class CMarkManager:public CObject
{
private:
class CMarkAttribArray* mp_MarkAtributesArray;
class CArcArray* mp_ArcsArray;
class CGateArray* mp_GatesArray;
class CTransitionArray* mp_TransitionsArray;
class CBoxArray* mp_BoxesArray;
class CGraph* mp_Graph;
CTime m_ActualTime;
CTimeSpan m_GraphCycleTime;
CString m_CycleTag;
BOOL m_FileLog;
BOOL m_RunnedOnce;

public:
CMarkManager();
~CMarkManager();
CMarkManager(CMarkAttribArray* p_MarkAtributesArray,CArcArray* p_ArcsArray,CGateArray*
p_GatesArray,CTransitionArray* p_TransitionsArray, CBoxArray* p_BoxesArray);
void Create(CMarkAttribArray* p_MarkAtributesArray,CArcArray* p_ArcsArray,CGateArray* p_GatesArray,CTransitionArray*
p_TransitionsArray, CBoxArray* p_BoxesArray);
void Create (CGraph* MyGraph);

void SetUpTransitions();
void FireTransitions();
void LogGraph();

// Access Functions:
```

```
void SetFileLog(BOOL Status);
BOOL IsLogging();

CString GetTimeStamp();
CString GetTimeSpanStamp();
CTime GetLastFireTime();
CTimeSpan GetLastCycleSpan();


class CBoxArray* GetBoxes();
class CBox& GetBox(CString BoxLabel);
class CBox& GetBox(long MyIndex);
class CMarkAttribute GetMarkAtribute(CString BoxLabel, CString MyLabel);
class CMarkAttribute GetMarkAtribute(long MyBoxIndex, CString MyLabel);

class CArcArray* GetArcs();
class CArc& GetArc(long MyIndex);

class CTransitionArray* GetTransitions();
class CTransition& GetTransition(long MyIndex);

class CGateArray* GetGates();
class CGate& GetGate(CString GateLabel);
class CGate& GetGate(long MyIndex);

CMarkAttribArray* GetAtribTemplates();

void SeekAndSolveConflicts();

void RunControlCycle();

void UpdateTimers(CTime Actual);

protected:

void SeekAndSolveInputConflicts();
void SeekAndSolveOutputConflicts();
void SolveInputConflict(long NumOfConflicts, long* p_MySrcTransitions);
void SolveOutputConflict(long NumOfConflicts, long* p_MyDestinyTransitions);
void LogFiredTransition(CString TransitionLabel);
};

// manager.cpp eMFG MarkManager Definition:
// version:1.0a
// Definition Date: 07/03/98
// Daniel M. S. Ferreira
// Last Modified: 08/07/98
// Programed by: Daniel M. S. Ferreira
// Last modified by: Daniel M. S. Ferreira

// Objetivos: Implementar o Gerenciador de Marcas do grafo


#include "stdafx.h"                          // standard windows aplication
#include "elements.h"   // E-MFG Standard Elements

#include "manager.h"
// The headers bellow are necessary for the random number generator
// used to solve the graph's conflicts:

#include <stdlib.h> // rand() function header
#include <time.h> // time() function header


// Mark Manager

// class CMarkManager:public CObject

CMarkManager::CMarkManager()
{
mp_MarkAtributesArray=NULL;
mp_ArcsArray=NULL;
```

```
mp_GatesArray=NULL;
mp_TransitionsArray=NULL;
mp_BoxesArray=NULL;
//CTime m_ActualTime;
//CTimeSpan m_GraphCycleTime;
m_FileLog = FALSE;
mp_Graph=NULL;
m_CycleTag = "><";
m_RunnedOnce = FALSE;
}

CMarkManager::~CMarkManager()
{
// void destructor: there is no need to free memory here
// the mark manager pointers belong to the graph
}
void CMarkManager::Create (CGraph* MyGraph)
{
mp_MarkAtributesArray=MyGraph->GetAttribTemplates();
mp_ArcsArray=MyGraph->GetArcs();
mp_GatesArray=MyGraph->GetGates();
mp_TransitionsArray=MyGraph->GetTransitions();
mp_BoxesArray=MyGraph->GetBoxes();
ASSERT((mp_MarkAtributesArray!=NULL)&&
                                (mp_ArcsArray!=NULL)&&
                                (mp_GatesArray!=NULL)&&
                                (mp_TransitionsArray!=NULL)&&
                                (mp_BoxesArray!=NULL));
//CTime m_ActualTime;
//CTimeSpan m_GraphCycleTime;
mp_Graph = MyGraph;
m_FileLog = FALSE;
}

CMarkManager::CMarkManager(CMarkAttribArray* p_MarkAtributesArray,CArcArray* p_ArcsArray,CGateArray*
p_GatesArray,CTransitionArray* p_TransitionsArray, CBoxArray* p_BoxesArray)
{
ASSERT(FALSE);
mp_MarkAtributesArray=p_MarkAtributesArray;
mp_ArcsArray=p_ArcsArray;
mp_GatesArray=p_GatesArray;
mp_TransitionsArray=p_TransitionsArray;
mp_BoxesArray=p_BoxesArray;

ASSERT((mp_MarkAtributesArray!=NULL)&&
                                (mp_ArcsArray!=NULL)&&
                                (mp_GatesArray!=NULL)&&
                                (mp_TransitionsArray!=NULL)&&
                                (mp_BoxesArray!=NULL));

//CTime m_ActualTime;
//CTimeSpan m_GraphCycleTime;
m_FileLog = FALSE;
}

void CMarkManager::Create(CMarkAttribArray* p_MarkAtributesArray,CArcArray* p_ArcsArray,CGateArray*
p_GatesArray,CTransitionArray* p_TransitionsArray, CBoxArray* p_BoxesArray)
{
ASSERT(FALSE);
mp_MarkAtributesArray=p_MarkAtributesArray;
mp_ArcsArray=p_ArcsArray;
mp_GatesArray=p_GatesArray;
mp_TransitionsArray=p_TransitionsArray;
mp_BoxesArray=p_BoxesArray;
ASSERT((mp_MarkAtributesArray!=NULL)&&
                                (mp_ArcsArray!=NULL)&&
                                (mp_GatesArray!=NULL)&&
                                (mp_TransitionsArray!=NULL)&&
                                (mp_BoxesArray!=NULL));

//CTime m_ActualTime;
//CTimeSpan m_GraphCycleTime;
```

```
m_FileLog = FALSE;
m_RunnedOnce = FALSE;
}

void CMarkManager::SeekAndSolveInputConflicts()
{
// This function searches for any pos-condition box that have
// more than one transition ready to be fired connected to it
// if there is a box in this state the function SolveInputConflict (..)
// is called for these transitions.

long* p_TransitionsIndex = NULL;
long ActualNumOfConflicts = 0;
long Count=0;
for (long index=0;index<GetBoxes()->GetSize();index++)
                {
                CBox ActualBox = (GetBoxes()->GetAt(index));
                if (ActualBox.IsAReadyPosCondition())
                                {
                                long NumArcs = ActualBox.GetNumOriginArcs();
                                Count=0;
                                if (NumArcs>1)
                                                {
                                                long* MyTransitions =
ActualBox.GetOriginTransitionsIndexList(GetArcs());
                                                p_TransitionsIndex = new long[NumArcs];
                                                for (long pointer=0;pointer<NumArcs;pointer++)
                                                                {
                                                                if ((GetTransitions()-
>GetAt(MyTransitions[pointer])).CanBeFired())
                                                                                {

        p_TransitionsIndex[Count] = MyTransitions[pointer];
                                                                                Count++;
                                                                                }
                                                                }
                                                if (Count>1)
                                                                {
                                                                ActualNumOfConflicts = Count;

        SolveInputConflict(ActualNumOfConflicts,p_TransitionsIndex);
                                                                }
                                                delete p_TransitionsIndex;
                                                delete MyTransitions;
                                                }
                                }
                }
}

void CMarkManager::SeekAndSolveOutputConflicts()
{
// This function searches for any pre-condition box that have
// more than one transition ready to be fired connected to it
// if there is a box in this state the function SolveOutputConflict (..)
// is called for these transitions.

long* p_TransitionsIndex = NULL;
long ActualNumOfConflicts = 0;
long Count=0;
for (long index=0;index<GetBoxes()->GetSize();index++)
                {
                CBox ActualBox = (GetBoxes()->GetAt(index));
                if (ActualBox.IsAReadyPreCondition())
                                {
                                long NumArcs = ActualBox.GetNumDestinyArcs();
                                Count=0;
                                if (NumArcs>1)
                                                {
                                                long* MyTransitions =
ActualBox.GetDestinyTransitionsIndexList(GetArcs(),GetGates(),GetBoxes());
                                                p_TransitionsIndex = new long[NumArcs];
                                                for (long pointer=0;pointer<NumArcs;pointer++)
```

```
                                                                {
                                                                if ((GetTransitions()-
>GetAt(MyTransitions[pointer])).CanBeFired())
                                                                        {

        p_TransitionsIndex[Count] = MyTransitions[pointer];
                                                                        Count++;
                                                                        }
                                                                }
                                                        if (Count>1)
                                                                {
                                                                ActualNumOfConflicts = Count;

        SolveOutputConflict(ActualNumOfConflicts,p_TransitionsIndex);
                                                                }
                                                        delete p_TransitionsIndex;
                                                        delete MyTransitions;
                                                        }
                                        }
                        }

}

void CMarkManager::SolveInputConflict(long NumOfConflicts, long* p_MySrcTransitions)
{
// Each transition in conflict has an equal chance to fire
// This function calculates this chance(subset) and generates a random
// percentile number to determine the winner transition index

ASSERT((NumOfConflicts>0)&&(p_MySrcTransitions!=NULL));
srand( (unsigned)time( NULL ) );
int RandNumber = rand();
double Percentile = 100.0*((double)RandNumber)/((double)RAND_MAX);
double Subset = 100.0/(double)NumOfConflicts;
ASSERT((Subset>0.0001));
long Count =0;
while (Percentile-Subset>0.0001)
                        {
                        Percentile=Percentile - Subset;
                        Count++;
                        }
ASSERT(Count<NumOfConflicts);

// Searches for a previously solved conflict:
BOOL SolvedConflict=FALSE;
for (long index=0;index<NumOfConflicts;index++)
                        {
                        SolvedConflict = SolvedConflict | GetTransition(p_MySrcTransitions[index]).IsASolvedConflict();
                        if (SolvedConflict)//If this transition is a solved conflict
                                        {
                                        Count = index; //This is the transition to be fired;
                                        break;
                                        }
                        }

// Only one transition can be fired:
for (index=0;index<NumOfConflicts;index++)
                        {
                        if (index!=Count)
                                        {
                                        GetTransition(p_MySrcTransitions[index]).SetChangeFlag(FALSE);
                                        }
                        else{
                                        // this transition will be fired:
                                        GetTransition(p_MySrcTransitions[index]).SetChangeFlag(TRUE);
                                        GetTransition(p_MySrcTransitions[index]).SetConflictFlag(TRUE);
                                        }
                        }
}

void CMarkManager::SolveOutputConflict(long NumOfConflicts, long* p_MyDestinyTransitions)
{
```

```
// Each transition in conflict has an equal chance to fire
// This function calculates this chance(subset) and generates a random
// percentile number to determine the winner transition index
ASSERT((NumOfConflicts>0)&&(p_MyDestinyTransitions!=NULL));
srand( (unsigned)time( NULL ) );
int RandNumber = rand();
double Percentile = 100.0*((double)RandNumber)/((double)RAND_MAX);
double Subset = 100.0/(double)NumOfConflicts;
ASSERT((Subset>0.0001));
long Count =0;
while (Percentile-Subset>0.0001)
                        {
                        Percentile=Percentile - Subset;
                        Count++;
                        }
ASSERT(Count<NumOfConflicts);

// Searches for a previously solved conflict:
BOOL SolvedConflict=FALSE;
for (long index=0;index<NumOfConflicts;index++)
                        {
                        SolvedConflict = SolvedConflict | GetTransition(p_MyDestinyTransitions[index]).IsASolvedConflict();
                        if (SolvedConflict)//If this transition is a solved conflict
                                                {
                                                Count = index; //This is the transition to be fired;
                                                break;
                                                }
                        }
// Only one transition can be fired:
for (index=0;index<NumOfConflicts;index++)
                        {
                        if (index!=Count)
                                                {
                                                GetTransition(p_MyDestinyTransitions[index]).SetChangeFlag(FALSE);
                                                }
                        else{
                                                // this transition will be fired:
                                                GetTransition(p_MyDestinyTransitions[index]).SetChangeFlag(TRUE);
                                                GetTransition(p_MyDestinyTransitions[index]).SetConflictFlag(TRUE);
                                                }
                        }
}

void CMarkManager::SetUpTransitions()
{
for (long index=0;index<GetTransitions()->GetSize();index++)
                        {
                        GetTransition(index).UpdateFlags();
                        }
}

void CMarkManager::FireTransitions()
{
if (m_CycleTag == "><")
 {
 m_CycleTag = "<>";
 }
else
 {
 m_CycleTag = "><";
 }
for (long index=0;index<GetTransitions()->GetSize();index++)
                        {
                        if (GetTransition(index).CanBeFired())
                                                {
                                                GetTransition(index).FireTransition();
                                                if (IsLogging())
                                                                        {
                                                                        LogFiredTransition(GetTransition(index).GetLabel());
                                                                        }
                                                }
                        else
```

```
                                        {
                                        GetTransition(index).ResetFlags();
                                        }
                        }
}

void CMarkManager::LogGraph()
{
FILE* file=NULL;                                                        //file stream
CString line;       //linha do arquivo
CTime Now;
Now=m_ActualTime; //Pega o timer atual;

CString filename = mp_Graph->m_title;
filename=filename+".ghp";
file = fopen(filename,"a+");//abre o arquivo permitindo que o MS_DOS Type
                //visualize o arquivo
if (file==NULL)
                        return;
line = "TimeStamp: [d/m/y][H:M:S] \n";
fprintf(file,"%s",line);
if (m_RunnedOnce)
  line = "TimeStamp: " + Now.Format("[%d/%m/%y][%H:%M:%S] \n");
else
  line = "TimeStamp: " + Now.GetCurrentTime().Format("[%d/%m/%y][%H:%M:%S] \n");
fprintf(file,"%s",line);

CString EMFGCompilerInfo = "/* EMFG Script Language Version 1.0 */\n";
EMFGCompilerInfo += "/* EMFG Interpreter Version 1.0 */\n";
EMFGCompilerInfo += "/* EMFG Linker Version 1.0 */\n";
EMFGCompilerInfo += "/* EMFG MarkManager Version 1.0 */\n\n";
EMFGCompilerInfo += "/* EMFG Communicator Version 1.0 */\n\n";
CString SectionHeader = "<< E-MFG Boxes Connections Dump >>\n\n";
CString DumpFormat = "Dumping Format:/n";
DumpFormat += "[Origin Transition Label]->(BoxLabel)->[Destiny Transition Label]\n";
DumpFormat += "if a box has a mark it's label will be marked as follows:\n";
DumpFormat += "[Origin Transition Label]->(** BoxLabel **)->[Destiny Transition Label]\n\n";


line = EMFGCompilerInfo;
fprintf(file,"%s",line);
line = SectionHeader;
fprintf(file,"%s",line);
line = DumpFormat;
fprintf(file,"%s",line);

for (long index=0;index<GetBoxes()->GetSize();index++)
                        {
                        long NumOrigins = GetBoxes()->ElementAt(index).GetNumOriginArcs();
                        long* OriginIndex = GetBox(index).GetOriginTransitionsIndexList(GetArcs());
                        CString Origins = "[ ";
                        for (long k=0;k<NumOrigins;k++)
                                        {
                                        Origins = Origins + GetTransition(OriginIndex[k]).GetLabel();
                                        if (k<NumOrigins-1)
                                                        {
                                                        Origins += ", ";
                                                        }
                                        }
                        Origins += " ]";

                        long NumDestiny = GetBoxes()->ElementAt(index).GetNumDestinyArcs();
                        long* DestinyIndex = GetBox(index).GetDestinyTransitionsIndexList(GetArcs(),GetGates(),GetBoxes());
                        CString Destiny = "[ ";
                        for (k=0;k<NumDestiny;k++)
                                        {
                                        Destiny = Destiny + GetTransition(DestinyIndex[k]).GetLabel();
                                        if (k<NumDestiny-1)
                                                        {
                                                        Destiny += ", ";
                                                        }
                                        }
```

```
                    Destiny += " ]";

                    CString Label;
                    if (GetBox(index).HasMark())
                                        {
                                        Label = "** " + GetBox(index).GetLabel() + " **";
                                        }
                    else
                                        {
                                        Label = GetBox(index).GetLabel();
                                        }

                    CString Output = Origins + "-> (" + Label + ") ->" + Destiny + "\n";
  line = Output;
                    fprintf(file,"%s",line);
                    }

CString EndSectionHeader = "\n<<END: E-MFG Boxes Connections Dump >>\n";
line = EndSectionHeader;
fprintf(file,"%s",line);

CLinesArray* p_LinesArray;
p_LinesArray = new CLinesArray;
mp_Graph->GetBoxesPropertiesDump(p_LinesArray);

SectionHeader = "<< E-MFG Boxes Properties Dump >>\n\n";
DumpFormat = "Dumping Format:/n";
DumpFormat += "Name , Type , # Marks: ";
DumpFormat += "Atrib1 = XXX & Atrib2 = XXX & ... & AtribN = XXX\n\n";

line = SectionHeader;
fprintf(file,"%s",line);
line = DumpFormat;
fprintf(file,"%s",line);

for (long y=0; y<p_LinesArray->GetSize();y++)
 {
 line = "";
 for (long x=0; x<p_LinesArray->GetAt(y).GetSize(); x++)
  {
  if (x<2)
   {
   line += p_LinesArray->GetAt(y).GetAt(x) + "  , ";
   }
  else
   {
   if (x==2)
    {
    if (x!=p_LinesArray->GetAt(y).GetSize()-1)
     line += p_LinesArray->GetAt(y).GetAt(x) + ": ";
    else
     line += p_LinesArray->GetAt(y).GetAt(x) + "\n";
    }
   else
    {
    if (x!=p_LinesArray->GetAt(y).GetSize()-1)
     {
     line += p_LinesArray->GetAt(y).GetAt(x) + " & ";
     }
    else
     {
     line += p_LinesArray->GetAt(y).GetAt(x) + "\n\n";
     }
    }
   }
  }
 fprintf(file,"%s",line);
 }

EndSectionHeader = "\n<<END: E-MFG Boxes Properties Dump >>\n";
line = EndSectionHeader;
fprintf(file,"%s",line);
```

```
delete p_LinesArray;

SectionHeader = "<< E-MFG Gates Values Dump >>\n\n";
DumpFormat = "Dumping Format:\n";
DumpFormat += "(GateLabel)=Value\n";
DumpFormat += "(GateLabel)='String'\n\n";

line = SectionHeader;
fprintf(file,"%s",line);

line = DumpFormat;
fprintf(file,"%s",line);

for (index=0;index<GetGates()->GetSize();index++)
                        {
                        CString OutValue;
                        switch (GetGate(index).GetType())
                                                {
                                                case TEXT_ATRIB:
                                                                        {
                                                                        OutValue = "'" + GetGate(index).GetString() + "'";
                                                                        break;
                                                                        }
                                                case INTEGER_ATRIB:
                                                                        {
                                                                        CString Value;
                                                                        Value.Format("%ld", GetGate(index).GetInteger());
                                                                        OutValue = Value;
                                                                        break;
                                                                        }
                                                break;
                                                };
                        CString Output;
                        Output = "(" + GetGate(index).GetLabel() + ")=" + OutValue +"\n";
   line = Output;
   fprintf(file,"%s",line);
                        }

EndSectionHeader = "\n<<END: E-MFG Gates Values Dump >>\n";
line = EndSectionHeader;
fprintf(file,"%s",line);


fclose(file);

}

// Access Functions:
void CMarkManager::SetFileLog(BOOL Status)
{
m_FileLog = Status;
}

BOOL CMarkManager::IsLogging()
{
return m_FileLog;
}

CString CMarkManager::GetTimeStamp()
{
return m_ActualTime.Format("%c");
}

CString CMarkManager::GetTimeSpanStamp()
{
CString Return ="Seconds: " + m_GraphCycleTime.Format("%S");
return Return;
}

CTime CMarkManager::GetLastFireTime()
{
```

```
return m_ActualTime;
}

CTimeSpan CMarkManager::GetLastCycleSpan()
{
return m_GraphCycleTime;
}


class CBoxArray* CMarkManager::GetBoxes()
{
return mp_BoxesArray;
}

class CBox& CMarkManager::GetBox(CString BoxLabel)
{
for (long index=0;index<GetBoxes()->GetSize();index++)
                        {
                        if (GetBoxes()->GetAt(index).GetLabel()==BoxLabel)
                                        break;
                        }
ASSERT(GetBoxes()->GetAt(index).GetLabel()==BoxLabel);
return GetBoxes()->ElementAt(index);
}

class CBox& CMarkManager::GetBox(long MyIndex)
{
ASSERT(MyIndex>=0);
ASSERT(MyIndex<GetBoxes()->GetSize());
return GetBoxes()->ElementAt(MyIndex);
}

class CMarkAttribute CMarkManager::GetMarkAtribute(CString BoxLabel, CString MyLabel)
{
return GetBox(BoxLabel).GetMarkAtrib(MyLabel);
}

class CMarkAttribute CMarkManager::GetMarkAtribute(long MyBoxIndex, CString MyLabel)
{
ASSERT(MyBoxIndex>=0);
ASSERT(MyBoxIndex<GetBoxes()->GetSize());
return GetBoxes()->ElementAt(MyBoxIndex).GetMarkAtrib(MyLabel);
}

class CArcArray* CMarkManager::GetArcs()
{
return mp_ArcsArray;
}

class CArc& CMarkManager::GetArc(long MyIndex)
{
ASSERT(MyIndex>=0);
ASSERT(MyIndex<GetArcs()->GetSize());
return GetArcs()->ElementAt(MyIndex);
}

class CTransitionArray* CMarkManager::GetTransitions()
{
return mp_TransitionsArray;
}

class CTransition& CMarkManager::GetTransition(long MyIndex)
{
ASSERT(MyIndex>=0);
ASSERT(MyIndex<GetTransitions()->GetSize());
return GetTransitions()->ElementAt(MyIndex);
}

class CGateArray* CMarkManager::GetGates()
{
return mp_GatesArray;
}
```

```
class CGate& CMarkManager::GetGate(CString GateLabel)
{
for (long index=0;index<GetBoxes()->GetSize();index++)
                        {
                        if (GetGates()->GetAt(index).GetLabel()==GateLabel)
                                        break;
                        }
ASSERT(GetGates()->GetAt(index).GetLabel()==GateLabel);
return GetGates()->ElementAt(index);
}

class CGate& CMarkManager::GetGate(long MyIndex)
{
ASSERT(MyIndex>=0);
ASSERT(MyIndex<GetGates()->GetSize());
return GetGates()->ElementAt(MyIndex);
}

void CMarkManager::LogFiredTransition(CString TransitionLabel)
{
 FILE* file;                                                    //file stream
 CString line;      //linha do arquivo
 CTime Now;
 Now=m_ActualTime; //Pega o timer atual;

 CString filename = mp_Graph->m_title;
          filename=filename+".dmp";
 file = fopen(filename,"a+");//abre o arquivo permitindo que o MS_DOS Type
                  //visualize o arquivo
          if (file==NULL)
                  return;
 line = m_CycleTag + " ";
 line += Now.Format("[%d/%m/%y][%H:%M:%S] ");
 line += "Fired: ";
 line += TransitionLabel + "\n"; // Daniel 01/12
 fprintf(file,"%s",line);
 fclose(file);
}

CMarkAttribArray* CMarkManager::GetAtribTemplates()
{
return mp_MarkAtributesArray;
}

void CMarkManager::SeekAndSolveConflicts()
{
SeekAndSolveInputConflicts();
SeekAndSolveOutputConflicts();
}

void CMarkManager::RunControlCycle()
{
SetUpTransitions();
SeekAndSolveConflicts();
FireTransitions();
}


void CMarkManager::UpdateTimers(CTime Actual)
{
m_ActualTime=Actual;
//Updates Transitions
for (long index=0;index<GetTransitions()->GetSize();index++)
 {
 GetTransitions()->ElementAt(index).UpdateTimer(Actual);
 }
//Update Boxes
for (index=0;index<GetBoxes()->GetSize();index++)
 {
 GetBoxes()->ElementAt(index).UpdateTimer(Actual);
 }
```

```
if (!m_RunnedOnce)
 {
 m_RunnedOnce = TRUE;
 }
}
```

Files Comm.h and Comm.cpp - Communicator

```
//EMFG Communicator Definition
//Definition Date : 20/05/98
//Last Changed : 24/07/98
//Programmed by : Marco A. A. Silva
//Last Changed by: Marco A. A. Silva

//Definir o objeto comunicador do controlador

class CCommunicator : public CObject
{
public:
        CCommunicator();
        ~CCommunicator();

public:
        BOOL  UpdateAllGates();

public:
        void  SetGraph(CGraph* MyGraph);

public:
 void  SetErrorLogging(BOOL);
 void  SetLogFile(CString);
 void  Log(CString);

public:
        CTime GetLastUpdate();
        long  GetLastControlCycleDelta();
        long  GetLastCommCycleDelta();

public:
        CDDESocket      m_DDESock;

        CGraph*                 mp_Graph;
        BOOL                            m_GraphSet;

 BOOL     m_LogErrors;
 short    m_CommError;
 BOOL     m_LogFileSet;
 CString  m_LogFile;

        CTime                           m_Now;
        CTimeSpan       m_Delta;
        CTimeSpan               m_Control;
        CTimeSpan               m_Comm;
};

//EMFG Communicator Definition
//Definition Date : 20/05/98
//Last Changed : 24/07/98
//Programmed by : Marco A. A. Silva
//Last Changed by: Marco A. A. Silva

//Definir o objeto comunicador do controlador

#include "stdafx.h"

#include <ddeml.h>                                      //DDE Management Library

#include "elements.h"                           //E-MFG Elements Definition

#include "ddesock.h"                            //DDE Socket Definition
#include "comm.h"                                       //E-MFG Communicator Definition
```

```
CCommunicator::CCommunicator()
{
        mp_Graph = NULL;

 m_GraphSet = FALSE;
 m_LogErrors = TRUE;

 m_LogFile = "";

 m_CommError = NULO;

        m_Now = CTime::GetCurrentTime();
}

BOOL CCommunicator::UpdateAllGates()
{
        short i,n;

 long myBox;
 long myAttrib;

 short type;

 short RetInt;
 BOOL RetBOOL;
 CString RetString;

        CTime m_Begin;                  //momento de início do ciclo de comunicação

 m_CommError = NULO;  //nenhum erro de comunicação DDE

 m_Begin = CTime::GetCurrentTime();

        //try's e catches para capturar erros de comunicação
                if (!m_GraphSet)
                        {
                        MessageBox(NULL,"The Graph must set EVERY time you use UpdateAllGates","Graph not
set",MB_OK);
                        AfxThrowUserException();
                        }

                //número de gates
                n = mp_Graph->GetGates()->GetSize();

                //loop de atualização
                for (i=0;i<n;i++)
                        {
                        switch (mp_Graph->GetGates()->GetAt(i).GetType())
                                {
                                case INT_PERMIT:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                        //lógica dos gates permit
                                        if (mp_Graph->GetBoxes()->GetAt(myBox).HasMark())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                        else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                        break;
                                        }
                                case INT_NOT_PERMIT:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                        //lógica dos gates not permit
                                        if (mp_Graph->GetBoxes()->GetAt(myBox).HasMark())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                        else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                        break;
```

```
                                        }
                                case INT_FULL_PERMIT:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                        //lógica dos gates permit
                                        if (mp_Graph->GetBoxes()->GetAt(myBox).IsFull())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                        else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                        break;
}
                                case INT_FULL_NOT_PERMIT:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                        //lógica dos gates not permit
                                        if (mp_Graph->GetBoxes()->GetAt(myBox).IsFull())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                        else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                        break;
}
                                case EXTERNAL_INPUT_PERMIT:
                                        {
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  m_DDESock.GetBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  mp_Graph->GetGates()->ElementAt(i).SetBOOL(RetBOOL);
  }
 }
CATCH( CUserException, e )
 {
                        m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
   case NULO:
    {
    MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
 }
        END_CATCH
break;
}
                                case EXTERNAL_INPUT_NOT_PERMIT:
                                        {
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  m_DDESock.GetBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  mp_Graph->GetGates()->ElementAt(i).SetBOOL(RetBOOL);
  }
 }
CATCH( CUserException, e )
 {
                        m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
```

```
      case NULO:
       {
       MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
       break;
       }
      };
     }
    }
          END_CATCH
break;
}
                                    case EXTERNAL_OUTPUT_PERMIT:
                                          {
                                          //box de origem
                                          myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                          //lógica dos gates permit
                                          if (mp_Graph->GetBoxes()->GetAt(myBox).HasMark())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                          else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
//tipo de ativação
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  RetBOOL = mp_Graph->GetGates()->GetAt(i).GoAhead();
  m_DDESock.PutBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  }
 }
CATCH( CUserException, e )
 {
                m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
   case NULO:
    {
    MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
 }
          END_CATCH
break;
}
                                    case EXTERNAL_OUTPUT_NOT_PERMIT:
                                          {
                                          //box de origem
                                          myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                          //lógica dos gates not permit
                                          if (mp_Graph->GetBoxes()->GetAt(myBox).HasMark())
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                          else
                                                mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
//tipo de ativação
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  RetBOOL = mp_Graph->GetGates()->GetAt(i).GoAhead();
  m_DDESock.PutBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  }
 }
CATCH( CUserException, e )
 {
                m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
```

```
  else
   {
   switch(m_CommError)
    {
    case NULO:
     {
     MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
     break;
     }
    };
   }
  }
          END_CATCH
break;
}
                                        case EXTERNAL_OUTPUT_FULL_PERMIT:
                                                {
                                                //box de origem
                                                myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                                //lógica dos gates permit
                                                if (mp_Graph->GetBoxes()->GetAt(myBox).IsFull())
                                                        mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
                                                else
                                                        mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
//tipo de ativação
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  RetBOOL = mp_Graph->GetGates()->GetAt(i).GoAhead();
  m_DDESock.PutBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  }
 }
CATCH( CUserException, e )
 {
                m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
   case NULO:
    {
    MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
 }
          END_CATCH
break;
}
                                        case EXTERNAL_OUTPUT_FULL_NOT_PERMIT:
                                                {
                                                //box de origem
                                                myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                                //lógica dos gates not permit
                                                if (mp_Graph->GetBoxes()->GetAt(myBox).IsFull())
                                                        mp_Graph->GetGates()->ElementAt(i).SetBOOL(FALSE);
                                                else
                                                        mp_Graph->GetGates()->ElementAt(i).SetBOOL(TRUE);
//tipo de ativação
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  RetBOOL = mp_Graph->GetGates()->GetAt(i).GoAhead();
  m_DDESock.PutBOOL(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetBOOL);
  }
 }
CATCH( CUserException, e )
```

```
 {
                    m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
   case NULO:
    {
    MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
 }
        END_CATCH
break;
}
                                case INT_N:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
//number of marks
RetInt = (short)mp_Graph->GetBoxes()->ElementAt(myBox).GetNumMarks();
//sets the gate value
                                        mp_Graph->GetGates()->ElementAt(i).SetInteger(RetInt);
break;
}
                                case EXTERNAL_OUTPUT_DATA:
                                        {
                                        //box de origem
                                        myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
                                        //atributo da marca
                                        myAttrib = mp_Graph->GetGates()->GetAt(i).GetAttrib();
//o default é nulo (caso não haja marca)
RetInt   = 0;
RetString = "";
//determina o tipo
type = mp_Graph->GetGates()->GetAt(i).GetRetType();
//verifica presença de marca e recupera o valor
                                        if (mp_Graph->GetBoxes()->GetAt(myBox).HasMark())
 {
 CMarkAttribute myAttribute;
 myAttribute = mp_Graph->GetBoxes()->GetAt(myBox).GetMarkAtrib(myAttrib);
 switch (myAttribute.GetType())
  {
  case INTEGER_ATRIB:
   {
   RetInt = (short)myAttribute.GetIntegerAtrib();
   break;
   }
  case TEXT_ATRIB:
   {
   RetString = RetString = myAttribute.GetTextAtrib();
   break;
   }
  };
 }
//dependendo do tipo executa a atualização e operação DDE no caso ativo
TRY
 {
 switch (type)
  {
  case INTEGER_ATRIB:
   {
   mp_Graph->GetGates()->ElementAt(i).SetInteger(RetInt);
   if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
    m_DDESock.PutInteger(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetInt);
   break;
   }
  case TEXT_ATRIB:
```

```
      {
      mp_Graph->GetGates()->ElementAt(i).SetString(RetString);
      if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
       m_DDESock.PutString(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetString);
      break;
      }
     };
    }
   CATCH( CUserException, e )
    {
                        m_GraphSet = FALSE;
    if (m_LogErrors)
     Log (m_DDESock.m_AppName+".dde");
    else
     {
     switch(m_CommError)
      {
      case NULO:
       {
       MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
       break;
       }
      };
     }
    }
          END_CATCH
break;
}
                                    case EXTERNAL_OUTPUT_N:
                                                {
                                                //box de origem
                                                myBox = mp_Graph->GetGates()->GetAt(i).GetBox();
//number of marks
RetInt = (short)mp_Graph->GetBoxes()->ElementAt(myBox).GetNumMarks();
//sets the gate value
                                                mp_Graph->GetGates()->ElementAt(i).SetInteger(RetInt);
TRY
 {
 if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
  {
  m_DDESock.PutInteger(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetInt);
  }
 }
CATCH( CUserException, e )
 {
                        m_GraphSet = FALSE;
 if (m_LogErrors)
  Log (m_DDESock.m_AppName+".dde");
 else
  {
  switch(m_CommError)
   {
   case NULO:
    {
    MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
 }
          END_CATCH
break;
}
                                    case EXTERNAL_INPUT_DATA:
                                                {
if (mp_Graph->GetGates()->GetAt(i).GetActivation()==ACTIVE)
 {
 TRY
  {
  if (mp_Graph->GetGates()->GetAt(i).GetRetType()==INTEGER_ATRIB)
   {
   m_DDESock.GetInteger(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetInt);
```

```
            mp_Graph->GetGates()->ElementAt(i).SetInteger(RetInt);
            }
           else
            {
            m_DDESock.GetString(mp_Graph->GetGates()->GetAt(i).GetParameter(),RetString);
            mp_Graph->GetGates()->ElementAt(i).SetString(RetString);
            }
           }
          CATCH( CUserException, e )
           {
                              m_GraphSet = FALSE;
           if (m_LogErrors)
            Log (m_DDESock.m_AppName+".dde");
           else
            {
            switch(m_CommError)
             {
             case NULO:
              {
              MessageBox(NULL,"Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
              break;
              }
             };
           }
          }
                 END_CATCH
        break;
        }
       }
                                          };
                                  }

            //este é o fim do ciclo de comunicação
            //calculam-se os deltas aqui
            CTime temp = CTime::GetCurrentTime();                    //momento de início do ciclo de comunicação
            m_Delta  = temp -  m_Now;
  m_Control = m_Begin - m_Now;
            m_Comm   = temp - m_Begin;
  m_Now    = temp;
            //força a posterior chamada de SetGraph
            m_GraphSet = FALSE;

            //tudo ok
            return TRUE;
}

CTime CCommunicator::GetLastUpdate()
{
            return (m_Now);
}

long CCommunicator::GetLastCommCycleDelta()
{
            return(m_Comm.GetTotalSeconds());
}

long CCommunicator::GetLastControlCycleDelta()
{
            return(m_Control.GetTotalSeconds());
}

void CCommunicator::SetGraph(CGraph* MyGraph)
{
            mp_Graph = MyGraph;
            m_GraphSet = TRUE;
}

void CCommunicator::SetErrorLogging(BOOL arg)
{
 m_LogErrors = arg;
}
```

```
void CCommunicator::SetLogFile(CString arg)
{
 m_LogFileSet = TRUE;
 m_LogFile = arg;
}

void CCommunicator::Log(CString filename)
{
 FILE* file;                                                    //file stream
 CString line;      //linha do arquivo

          file = fopen(filename,"a");

          if (file==NULL)
                  return;

 line = m_Now.Format("[%d/%m][%H:%M:%S] ");

 line += m_DDESock.m_AppName;
 line += " - ";
 line += "App ";
 line += m_DDESock.m_App;
 line += " - Topic ";
 line += m_DDESock.m_Topic;
 line += " - Item ";
 line += m_DDESock.m_Item;
 line += " - erro de comunicação : ";

 switch (m_CommError)
  {
  case NULO:
   {
   line += "não definido\n";
   break;
   }
  };

 fprintf(file,"%s",line);

 fclose(file);
}

CCommunicator::~CCommunicator()
{
}
```

Files Ddesock.h and Ddesock.cpp - DDE Socket Implementation

```
//DDEML.H Encapsulating Object
//Definition Date: 20/06/98
//Programmed by : Marco A. A. Silva
//Last Change : 21/06/98
//Changed by : Marco A. A. Silva

//Define an encapsulating object for DDE Applications

class CDDESocket : public CObject
{
public:
          CDDESocket();
          ~CDDESocket();

public:
          void      RegisterApp(CString Name);

public:
 void GetString  (CString Param, CString& arg);
 void GetInteger (CString Param, short& arg);
 void GetBOOL    (CString Param, BOOL& arg);

public:
```

```
  void PutString  (CString Param, CString arg);
  void PutInteger (CString Param, short arg);
  void PutBOOL    (CString Param, BOOL arg);

public:
  void SetTimeOut(DWORD arg);

private:
        void Parsing(CString& Param);

public:
        CString m_AppName;

        CString m_App;
        CString m_Topic;
        CString m_Item;

        BOOL   m_Registered;
        BOOL   m_Connected;

 DWORD  m_timeout;

 UINT   m_error;

public:
        DWORD  m_idInst;
        HCONV  m_hConv;                                  // our handle to conversation
};


//DDEML.H Encapsulating Object
//Definition Date: 20/06/98
//Programmed by : Marco A. A. Silva
//Last Change : 21/06/98
//Changed by : Marco A. A. Silva

//Define an encapsulating object for DDE Applications

#include "stdafx.h"
#include <ddeml.h>
#include "ddesock.h"

#include "elements.h"

#include "stdlib.h"
#include "string.h"


HDDEDATA CALLBACK DdeCallback(WORD wType, WORD wFmt, HCONV hConv, HSZ hszTopic,

                              HSZ hszItem, HDDEDATA hData, DWORD lData1, DWORD lData2);

CDDESocket::CDDESocket()
{
        m_AppName="";

        m_Registered  = FALSE;
        m_Connected   = FALSE;

 m_error = 0;

 m_timeout = 200;

 m_App  = "";
 m_Item  = "";
 m_Topic = "";

 m_idInst = 0;
}

void CDDESocket::RegisterApp(CString Name)
{
```

```
        HSZ hszAppName = 0;

 if (m_Registered)
  {
  DdeNameService(m_idInst, 0, 0, DNS_UNREGISTER); // unregister all services
                DdeUninitialize (m_idInst);
  }

 m_Registered = FALSE;

        //DDE Initialization, observe it is a client_only application
        DdeInitialize(&m_idInst, (PFNCALLBACK)DdeCallback, APPCLASS_STANDARD, 0);

 hszAppName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        Name,                 /* application name                */
                        CP_WINANSI);  /* Windows ANSI code page */

 DdeNameService(m_idInst, hszAppName, 0, DNS_REGISTER);

 m_error = DdeGetLastError(m_idInst);
 if (m_error)
                {
                AfxThrowUserException();
                return;
                }

 m_AppName = Name;

        m_Registered = TRUE;
}

void CDDESocket::GetString (CString Param, CString& arg)
{
        HSZ hszServName;                                        //handle to service name
        HSZ hszSysTopic;                                        //handle to topic name
        HSZ hszItem;                                                    //handle to topic item

 HDDEDATA hData;                                                //handle to DDE data

 char* psz;                                     //result string
 DWORD cb;                                                              //size of result
string
 DWORD i;
 char ch;

        //Parsing the parameters from parameter string
        Parsing(Param);

        //Obtaining handles to service, topic and item strings
hszServName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_App,                /* application name                */
                        CP_WINANSI);  /* Windows ANSI code page */

        hszSysTopic = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Topic,                              /* System topic        */
                        CP_WINANSI);  /* Windows ANSI code page */

        hszItem = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Item,                                       /* Topic item          */
                        CP_WINANSI);  /* Windows ANSI code page */

        if (!m_Connected)
                {
                //DDE Connection
                m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                m_error = DdeGetLastError(m_idInst);
                if (m_error)
                        {
```

```
                                AfxThrowUserException();
                                return;
                                }
                        }

  //DDE Client Transaction with item specified at Cell string
            hData = DdeClientTransaction(NULL, 0xFFFF, m_hConv, hszItem, CF_TEXT, XTYP_REQUEST, m_timeout,
 NULL);
  m_error = DdeGetLastError(m_idInst);

            DdeDisconnect(m_hConv);
  m_Connected = FALSE;

  if (m_error)
                        {
                        AfxThrowUserException();
                        return;
                        }

            //convert data from handle to string format
  cb = DdeGetData(hData, NULL, 0, 0);

            psz = new char[cb];

  m_error = DdeGetData(hData, (LPBYTE)psz, cb, 0L);

  arg="";

  for (i=0;i<cb;i++)
   {
   ch = psz[i];
   if ((ch!=13)||(ch!='\x00'))
     arg+=ch;
   else
     break;
   }

            //Freeing data handle
            DdeUnaccessData(hData);

            DdeFreeDataHandle(hData);

 //Freeing string handles
            DdeFreeStringHandle(m_idInst,hszServName);
            DdeFreeStringHandle(m_idInst,hszSysTopic);
            DdeFreeStringHandle(m_idInst,hszItem);

            delete (psz);

            return;
}


void CDDESocket::GetInteger (CString Param, short& arg)
{
            HSZ hszServName;                                    //handle to service name
            HSZ hszSysTopic;                                    //handle to topic name
            HSZ hszItem;                                                    //handle to topic item

 HDDEDATA hData;                                                    //handle to DDE data

 char* psz;                                 //result string
 DWORD cb;                                                                  //size of result
string

 //Parsing the parameters from parameter string
            Parsing(Param);

            //Obtaining handles to service, topic and item strings
 hszServName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_App,                  /* application name                  */
```

```
                                CP_WINANSI);   /* Windows ANSI code page */

        hszSysTopic = DdeCreateStringHandle(
                                m_idInst,     /* instance identifier   */
                                m_Topic,                                /* System topic        */
                                CP_WINANSI);   /* Windows ANSI code page */

        hszItem = DdeCreateStringHandle(
                                m_idInst,     /* instance identifier   */
                                m_Item,                                                 /* Topic item          */
                                CP_WINANSI);   /* Windows ANSI code page */

        if (!m_Connected)
                {
                //DDE Connection
                m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                m_error = DdeGetLastError(m_idInst);
                if (m_error)
                        {
                        AfxThrowUserException();
                        return;
                        }
                }

 //DDE Client Transaction with item specified at Cell string
        hData = DdeClientTransaction(NULL, 0xFFFF, m_hConv, hszItem, 1, XTYP_REQUEST, m_timeout, NULL);

 m_error = DdeGetLastError(m_idInst);

        DdeDisconnect(m_hConv);
 m_Connected = FALSE;

 if (m_error)
                {
                AfxThrowUserException();
                return;
                }

        //convert data from handle to string format
 cb = DdeGetData(hData, NULL, 0, 0);

        psz = new char[cb];

 m_error = DdeGetData(hData, (LPBYTE)psz, cb, 0L);

 CString string="";

 for (DWORD i=0;i<cb;i++)
  {
  char ch = psz[i];
  if ((ch!=13)&&(ch!='\x00'))
    string+=ch;
  else
    break;
  }

 delete psz;

 arg = (short)atoi(string);

        //Freeing data handle
        DdeUnaccessData(hData);
        DdeFreeDataHandle(hData);

//Freeing string handles
        DdeFreeStringHandle(m_idInst,hszServName);
        DdeFreeStringHandle(m_idInst,hszSysTopic);
        DdeFreeStringHandle(m_idInst,hszItem);

        return;
}
```

```
void CDDESocket::GetBOOL (CString Param, BOOL& arg)
{
        HSZ hszServName;                                    //handle to service name
        HSZ hszSysTopic;                                    //handle to topic name
        HSZ hszItem;                                                    //handle to topic item

 HDDEDATA hData;                                                    //handle to DDE data

 char* psz;                                 //result string
 DWORD cb;                                                                  //size of result
string

 //Parsing the parameters from parameter string
        Parsing(Param);

        //Obtaining handles to service, topic and item strings
 hszServName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_App,                  /* application name                  */
                        CP_WINANSI);   /* Windows ANSI code page */

        hszSysTopic = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Topic,                                /* System topic        */
                        CP_WINANSI);   /* Windows ANSI code page */

        hszItem = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Item,                                         /* Topic item          */
                        CP_WINANSI);   /* Windows ANSI code page */

        if (!m_Connected)
                {
                //DDE Connection
                m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                m_error = DdeGetLastError(m_idInst);
                if (m_error)
                        {
                        AfxThrowUserException();
                        return;
                        }
                }

//DDE Client Transaction with item specified at Cell string
//DDE Client Transaction with item specified at Cell string
        hData = DdeClientTransaction(NULL, 0xFFFF, m_hConv, hszItem, 1, XTYP_REQUEST, m_timeout, NULL);

m_error = DdeGetLastError(m_idInst);

        DdeDisconnect(m_hConv);
m_Connected = FALSE;

if (m_error)
                {
                AfxThrowUserException();
                return;
                }

        //convert data from handle to string format
cb = DdeGetData(hData, NULL, 0, 0);

        psz = new char[cb];

m_error = DdeGetData(hData, (LPBYTE)psz, cb, 0L);

CString string="";

for (DWORD i=0;i<cb;i++)
 {
 char ch = psz[i];
 if ((ch!=13)&&(ch!='\x00'))
  string+=ch;
```

```
    else
     break;
    }

  delete psz;

  arg = (BOOL)atoi(string);

  if (arg)
   arg = TRUE;
  else
   arg = FALSE;

            //Freeing data handle
            DdeUnaccessData(hData);

            DdeFreeDataHandle(hData);

 //Freeing string handles
            DdeFreeStringHandle(m_idInst,hszServName);
            DdeFreeStringHandle(m_idInst,hszSysTopic);
            DdeFreeStringHandle(m_idInst,hszItem);

            return;
}

void CDDESocket::PutString (CString Param, CString arg)
{
 short n;
 char* psz;

            HSZ hszServName;                                    //handle to service name
            HSZ hszSysTopic;                                    //handle to topic name
            HSZ hszItem;                                                    //handle to topic item

 HDDEDATA hData;                                                    //handle to DDE data

            //Parsing the parameters from parameter string
            Parsing(Param);

            //Obtaining handles to service, topic and item strings
hszServName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_App,                  /* application name                    */
                        CP_WINANSI);   /* Windows ANSI code page */

          hszSysTopic = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Topic,                                /* System topic        */
                        CP_WINANSI);   /* Windows ANSI code page */

          hszItem = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Item,                                 /* Topic item          */
                        CP_WINANSI);   /* Windows ANSI code page */

          if (!m_Connected)
                    {
                    //DDE Connection
                    m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                    m_error = DdeGetLastError(m_idInst);
                    if (m_error)
                              {
                              AfxThrowUserException();
                              return;
                              }
                    }

n = arg.GetLength();
psz = new char[n+1];
for (short i=0;i<n;i++)
 psz[i] = arg.GetAt(i);
```

```
 psz[i] = '\x00';


 //DDE Client Transaction with item specified at Cell string
        hData = DdeClientTransaction((LPBYTE)psz, n+1, m_hConv, hszItem, CF_TEXT, XTYP_POKE, m_timeout,
NULL);
 m_error = DdeGetLastError(m_idInst);

        DdeDisconnect(m_hConv);
 m_Connected = FALSE;

 if (m_error)
                {
                AfxThrowUserException();
                return;
                }

 //freeing char vector
 delete []psz;

        //Freeing data handle
        DdeFreeDataHandle(hData);

 //Freeing string handles
        DdeFreeStringHandle(m_idInst,hszServName);
        DdeFreeStringHandle(m_idInst,hszSysTopic);
        DdeFreeStringHandle(m_idInst,hszItem);

        return;
}

void CDDESocket::PutInteger (CString Param, short arg)
{
        HSZ hszServName;                                        //handle to service name
        HSZ hszSysTopic;                                        //handle to topic name
        HSZ hszItem;                                                    //handle to topic item

 HDDEDATA hData;                                                //handle to DDE data

        //Parsing the parameters from parameter string
        Parsing(Param);

        //Obtaining handles to service, topic and item strings
hszServName = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_App,                  /* application name                     */
                        CP_WINANSI);   /* Windows ANSI code page */

        hszSysTopic = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Topic,                                /* System topic         */
                        CP_WINANSI);   /* Windows ANSI code page */

        hszItem = DdeCreateStringHandle(
                        m_idInst,     /* instance identifier   */
                        m_Item,                                 /* Topic item           */
                        CP_WINANSI);   /* Windows ANSI code page */

        if (!m_Connected)
                {
                //DDE Connection
                m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                m_error = DdeGetLastError(m_idInst);
                if (m_error)
                        {
                        AfxThrowUserException();
                        return;
                        }
                }

CString string;
```

```
  sprintf(string.GetBuffer(20),"%d",arg);
  string.ReleaseBuffer();

  DWORD cb = string.GetLength();
  cb++;

  char* psz = new char[cb];

  for (DWORD i=0; i<(cb-1); i++)
   psz[i] = string.GetAt(i);

  psz[i] = '\x00';

  //DDE Client Transaction with item specified at Cell string
          hData = DdeClientTransaction((LPBYTE)psz, cb, m_hConv, hszItem, CF_TEXT, XTYP_POKE, m_timeout, NULL);

  m_error = DdeGetLastError(m_idInst);

          DdeDisconnect(m_hConv);
  m_Connected = FALSE;

  if (m_error)
                  {
                  AfxThrowUserException();
                  return;
                  }

  delete psz;

          //Freeing data handle
          DdeFreeDataHandle(hData);

 //Freeing string handles
          DdeFreeStringHandle(m_idInst,hszServName);
          DdeFreeStringHandle(m_idInst,hszSysTopic);
          DdeFreeStringHandle(m_idInst,hszItem);

          return;
}

void CDDESocket::PutBOOL (CString Param, BOOL arg)
{
          HSZ hszServName = 0;                                    //handle to service name
          HSZ hszSysTopic = 0;                                    //handle to topic name
          HSZ hszAppName = 0;
          HSZ hszItem = 0;                                                //handle to topic item

 HDDEDATA hData;                                                  //handle to DDE data

          //Parsing the parameters from parameter string
          Parsing(Param);

          //Obtaining handles to service, topic and item strings
hszServName = DdeCreateStringHandle(
                          m_idInst,     /* instance identifier   */
                          m_App,                  /* application name                     */
                          CP_WINANSI);  /* Windows ANSI code page */

          hszSysTopic = DdeCreateStringHandle(
                          m_idInst,     /* instance identifier   */
                                                  /* System topic         */
                          CP_WINANSI);  /* Windows ANSI code page */

          hszItem = DdeCreateStringHandle(
                          m_idInst,     /* instance identifier   */
                                                  /* Topic item           */
                          CP_WINANSI);  /* Windows ANSI code page */

          //DDE Initialization, observe it is a client_only application
          DdeInitialize(&m_idInst, (PFNCALLBACK)DdeCallback, APPCLASS_STANDARD, 0);

hszAppName = DdeCreateStringHandle(
```

```
                                m_idInst,     /* instance identifier   */
                                m_AppName,                  /* application name                */
                                CP_WINANSI);   /* Windows ANSI code page */

    DdeNameService(m_idInst, hszAppName, 0, DNS_REGISTER);

    m_error = DdeGetLastError(m_idInst);
            if (m_error)
                    {
                    AfxThrowUserException();
                    return;
                    }

            if (!m_Connected)
                    {
                    //DDE Connection
                    m_hConv = DdeConnect (m_idInst, hszServName, hszSysTopic, NULL);
                    m_error = DdeGetLastError(m_idInst);
                    if (m_error)
                            {
                            AfxThrowUserException();
                            return;
                            }
                    }

    CString string;
    sprintf(string.GetBuffer(20),"%d",arg);
    string.ReleaseBuffer();

    DWORD cb = string.GetLength();
    cb++;

    char* psz = new char[cb];

    for (DWORD i=0; i<(cb-1); i++)
      psz[i] = string.GetAt(i);

    psz[i] = '\x00';

    //DDE Client Transaction with item specified at Cell string
            hData = DdeClientTransaction((LPBYTE)psz, cb, m_hConv, hszItem, CF_TEXT, XTYP_POKE, m_timeout, NULL);

    m_error = DdeGetLastError(m_idInst);

            DdeDisconnect(m_hConv);
    m_Connected = FALSE;

    if (m_error)
                    {
                    AfxThrowUserException();
                    return;
                    }


    delete psz;

    //Freeing data handle
            DdeFreeDataHandle(hData);

    //Freeing string handles
            DdeFreeStringHandle(m_idInst,hszServName);
            DdeFreeStringHandle(m_idInst,hszSysTopic);
            DdeFreeStringHandle(m_idInst,hszItem);
            DdeFreeStringHandle(m_idInst,hszAppName);

            return;
}

void CDDESocket::SetTimeOut(DWORD arg)
{
 m_timeout = arg;
}
```

```
CDDESocket::~CDDESocket()
{
        if (m_Connected)
                {
                DdeDisconnect(m_hConv);
                }
        if (m_Registered)
                {
  DdeNameService(m_idInst, 0, 0, DNS_UNREGISTER); // unregister all services
                DdeUninitialize (m_idInst);
                }
}

void CDDESocket::Parsing(CString& Param)
{
        CString Parsing;
        short i,n;

        //parses parameters
        Parsing = Param;

        //Client Application Name
        n = Parsing.GetLength();
        i = Parsing.Find('!');
        if (i==-1)
                {
 m_error = 0;
                AfxThrowUserException();
                return;
                }

        m_App = Parsing.Left(i);
        Parsing = Parsing.Right(n-i-1);

        //Client Application Subject
        n = Parsing.GetLength();
        i = Parsing.Find('!');
        if (i==-1)
                {
 m_error = 0;
                AfxThrowUserException();
                return;
                }
        m_Topic = Parsing.Left(i);
        Parsing = Parsing.Right(n-i-1);

        //Client Application Item
        m_Item = Parsing;
}
```

E - Interpretador

Files Interp.h and Interp.cpp - E-MFG Graph Interpreter and Compiler

```
// interp.h - eMFG Interpreter header File
// version: 1.05a
// Definition Date: 06/01/98
// Last Modified: 26/06/98
// programed by: Marco A. A. Silva
// last modified by: Marco A. A. Silva

// Objetivos: Definir a estrutura do Interpretador eMFG

class CInterpreter : public CObject
{
public:
        CInterpreter();

public:
        void SetGraph(CGraph* MyGraph);
        void Read(CString filename);            //reads file and dumps into table
```

```
            void BuildTables();                                          //build literal tables
            void CrossRef();                                                    //performs table
 cross reference

 //literal tables building
 private:
            void InitialTags(BOOL bSave);
            void Compile(CString Filename, CString prefix);
            void IncludeTags(CString prefix);
            void MarkAttribTags();
            void BoxTags(CString prefix);
            void TransitTags(CString prefix);
            void ArcTags(CString prefix);
            void GateTags(CString prefix);
            void InitMarkTags();

 //object creation
 private:
            void InitVectors();
            void MarkAttribsCreate();
            void ArcsCreate();
  void FiltersCreate();
            void GatesCreate();
            void BoxesCreate();
  void TransitsCreate();
            void InitMarksCreate();
            void AttributionsCreate();
            void ConditionalsCreate();

 private:
            BOOL Search(CString Line, short start, char find, CString &Word, short &stop, char &found, BOOL trim, BOOL
 warn = TRUE);
            BOOL SeekTable(CLinesArray &Table, short col, CString word, short &index);
            BOOL SeekOrderedTable(CLinesArray &Table, short col, CString word, short &index, BOOL warn = TRUE);
            BOOL Next(CString &Line);

            void Duplicate (CLinesArray &Source);
  void CrossDuplicate (CLinesArray &Source, CLinesArray &Target);

 //associadas as condicionais
  void BuildCond  (CConditional& myCond, CString expr,short local);
  void CrossRefLOR(CString expr, BOOL &negative, short &LBox, short &LAttribute, short &Operator, short &RType, short
 &RParam1, short &RParam2, CString &RParam3, short Local);
  BOOL MatchingBrackets(CString& expr);

 //endereçamento
  void ParseAddress (CString Address, CString& First, CString& Second, short& Type);


 public:
            //tables
            CWordsArray m_FileTable;                                          //File Table

            //literal
            CLinesArray m_IncludeTable;                                  //Include Object table
            CLinesArray m_AttribTable;                                   //Marks Attributes table
            CLinesArray m_BoxesTable;                                           //Boxes table
            CLinesArray m_TransitsTable;                          //Transitions table
            CLinesArray m_ArcsTable;                                            //Arcs table
            CLinesArray m_GatesTable;                                           //Gates table
            CLinesArray m_MarksTable;                                           //Initial Marks table
            CLinesArray m_FilterTable;                                   //Arc Filter table
 CLinesArray m_TBCondTable;      //Transformator Conditionals
 CLinesArray m_TBThenTable;      //Transformator THEN Attributions
 CLinesArray m_TBElseTable;      //Transformator ELSE Attributions
 CLinesArray m_TransCondTable;    //Transitions Conditionals
 CLinesArray m_GatesCondTable;    //Gates Conditionals

           //ID
           CString m_title;
           //graph's title
```

```
        CString m_descr;
        //graph's description

  //Script
  CString m_path;
private:
        //graph
        class CGraph*       mp_Owner;                                   //my owner graph
 BOOL  m_GraphSet;


        //table building
        short               m_cursor;
        //table cursor
        short               m_length;
        //table length
        short               m_level;
        //include level
        CString m_line;
        //table line
        CString m_word;
        //line word
        BOOL                m_supported;
        //supported command
        CWordsArray m_param[2];                                         //auxiliary string arrays

        //object creation
 short *mp_nBOr;
        //number of arches to boxes
        short *mp_nBDs;
        //number of arches from boxes
        short *mp_nTOr;
        //number of arches to transitions
        short *mp_nTDs;
        //number of arches from transitions
        short *mp_nGts;         //number of gates at transition

 long (*mp_BoxOrigin)[NMAX];
 long (*mp_BoxDestin)[NMAX];
 long (*mp_TrsOrigin)[NMAX];
 long (*mp_TrsDestin)[NMAX];
 long (*mp_GtsDestin)[NMAX];

 short m_nA, m_nB, m_nT;                                                //table lengths
        short m_nTBC, m_nGC, m_nTC;     //table lengths
 short m_nAtbn, m_nF;                                   //table lengths
        short m_nG, m_nAtb, m_nM;                                       //table lengths
        short m_index[6];                                                                       //index
parameters
        CString m_par[5];                                                                       //string
parameters
};

// interp.cpp - eMFG Interpreter implementation file
// version: 1.05a
// Definition Date: 06/01/99
// Last Modified: 26/06/98
// programed by: Marco A. A. Silva
// last modified by: Marco A. A. Silva

// Objetivos: Implementar a estrutura do Interpretador eMFG

#include "stdafx.h"

#include <stdio.h>
#include <stdlib.h>

#include "definitions.h"                //E-MFG Constant Definitions
#include "wordsarray.h"                         //Extension of String Arrays
#include "basis.h"                                      //E-MFG Basis Class
#include "graph.h"                                      //E-MFG Graph Components
#include "marks.h"                                      //E-MFG Marks Definition
```

```
#include "arcs.h"                                   //E-MFG Arcs Definition
#include "boxes.h"                                  //E-MFG Boxes Definition
#include "transit.h"                                //E-MFG Transition Definition
#include "gates.h"                                  //E-MFG Gates Definition
#include "list.h"                                   //Linked List Definition
#include "arrays.h"                                 //Arrays Extension Class

#include "interp.h"                                 // E-MFG Interpreter


CInterpreter::CInterpreter()
{
        mp_Owner = NULL;

 m_GraphSet = FALSE;

        m_title = "";
        m_descr = "";
  m_path  = "";
}

void CInterpreter::SetGraph(CGraph* MyGraph)
{
        mp_Owner = MyGraph;
 m_GraphSet = TRUE;
}

//////////////////////////////////////////////////////////////////////////
//
//Read() lê o arquivo para a tabela
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::Read(CString filename)
{
        FILE* file;                                 //file stream
        BOOL stop;                                  //stop flag
        char psz[10];                               //input buffer
        CString text;                               //line text buffer
        short i;                                    //tag number
        short j;

if (!m_GraphSet)
  {
  MessageBox(NULL,"O interpretador não tem um grafo associado","Erro de Compilação",MB_OK);
  AfxThrowUserException();
  return;
  }

m_FileTable.RemoveAll();

        file = fopen(m_path+filename,"r");

        if (file==NULL)
                {
                MessageBox(NULL,"Não foi possível abrir arquivo.","File Open Error",MB_OK);
                AfxThrowUserException();
                return;
                }

        stop = FALSE;
        i = 0;

        while (!stop)
                {
                j = 0;
                text = "";
                fscanf(file,"%c",psz);
                if (feof(file))
                        stop = TRUE;

                while ((psz[0]!=';')&&(psz[0]!='\n')&&(!stop))
```

```
                                {
                                text+=psz[0];
                                j++;
                                if ( j>255 )
                                        {
        MessageBox(NULL,"Linha comprida demais:\n"+text,"File Read Error",MB_OK);
                                        AfxThrowUserException();
                                        return;
                                        }
                                fscanf(file,"%c",psz);
                                if (feof(file))
                                        stop = TRUE;
                                }

                        j = text.Find('%');
                        if (j!=-1)
                                text = text.Left(j);

                        text.TrimLeft();
                        text.TrimRight();

                        text.MakeUpper();

                        if (text!="")
                         {
                                i++;
                                sprintf (psz,"%d",i);
                                m_FileTable.Add(text);
                                }

                        }

                fclose (file);
}

//////////////////////////////////////////////////////////////////////////
//
//BuildTables() cria as tabelas literais
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::BuildTables()
{
        short i,n;
        //auxiliary integers

        CWordsArray Parameters;                         //linha de parâmetros
        CString                 file;
        //arquivo de include
        CString                 prefix;                                 //prefixo dos
elementos
        CString                 level;                                  //nível de include

        //limpa todas as tabelas antes de começar a montá-las
        m_IncludeTable.RemoveAll();
        m_AttribTable.RemoveAll();
        m_BoxesTable.RemoveAll();
        m_TransitsTable.RemoveAll();
        m_ArcsTable.RemoveAll();
        m_GatesTable.RemoveAll();
        m_MarksTable.RemoveAll();
m_FilterTable.RemoveAll();
m_TBCondTable.RemoveAll();
m_TBThenTable.RemoveAll();
m_TBElseTable.RemoveAll();
m_TransCondTable.RemoveAll();
m_GatesCondTable.RemoveAll();

        //coloca o cursor no zero e atualiza o tamanho da tabela
        m_cursor = 0;
        m_length = m_FileTable.GetSize();

        //controle das tags iniciais, salvando o título e descrição
```

```
		InitialTags(TRUE);
		//primeiro nível de include
		m_level = 0;

		//este próprio arquivo e prefixo nulo
		Parameters.Add("this");
		Parameters.Add("");
		Parameters.Add("0");
		m_IncludeTable.Add(Parameters);

		//primeiro arquivo da lista (this)
		i = 0;

		do
			{
			file   = m_IncludeTable.GetAt(i).GetAt(0);
			prefix = m_IncludeTable.GetAt(i).GetAt(1);
			level  = m_IncludeTable.GetAt(i).GetAt(2);
			if (atoi(level)!=m_level)
				m_level++;
			Compile (file, prefix);
			i++;
			n = m_IncludeTable.GetSize();
			}while (i<n);

		//nesse ponto as tabelas literais contém os elementos devidamente prefixados
}

//////////////////////////////////////////////////////////////////////////////
//
//Monta um arquivo de include
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::Compile(CString Filename, CString Prefix)
{
		short i,n;

		//arquivo de include é carregado aqui se necessário
		if (Filename!="this")
			{
			//teste contra níveis de recursão
			n = m_IncludeTable.GetSize();
			for (i=0;i<n;i++)
				{
				m_par[0] = m_IncludeTable.GetAt(i).GetAt(0);
				if (m_par[0]==Filename)
					{
					m_index[0] = atoi (m_IncludeTable.GetAt(i).GetAt(2));
					if (m_index[0]!=m_level)
						{
						MessageBox(NULL,m_par[0]+"\nInclude com diferentes níveis de
recursão.","Multiplas Recursões",MB_OK);
						AfxThrowUserException();
						}
					}
				}
			Read(Filename);
			m_cursor = 0;
			m_length = m_FileTable.GetSize();
			InitialTags(FALSE);
			}

		//montando a tabela de includes
		IncludeTags(Prefix);

		//checando a tag <EMFG>
		Next(m_line);
		if (m_line!="<EMFG>")
			{
			MessageBox(NULL,"Faltando tag <EMFG> inicial","Erro na Construção das Tabelas",MB_OK);
			AfxThrowUserException();
			}
```

```
            //montando a tabela de atributos de marcas
            MarkAttribTags();

  //montando a tabela de boxes
            BoxTags(Prefix);

  //montando a tabela de transições
            TransitTags(Prefix);

  //montando a tabela de arcos
            ArcTags(Prefix);

  //montando a tabela de gates
            GateTags(Prefix);

  //montando a tabela de boxes
            InitMarkTags();

            //checando a tag </EMFG>
            if (m_line!="</EMFG>")
                        {
                        MessageBox(NULL,"Faltando tag </EMFG> final","Erro na Construção das Tabelas",MB_OK);
                        AfxThrowUserException();
                        }
}

////////////////////////////////////////////////////////////////////////////
//
//faz a referência cruzada
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::CrossRef()
{
  ////////////////////////
  //                    //
  // Testes de Duplicação //
  //                    //
  ////////////////////////
            //testes de duplicação dentro das mesmas tabelas
//          Duplicate(m_AttribTable);
//          Duplicate(m_BoxesTable);
//          Duplicate(m_TransitsTable);
//          Duplicate(m_GatesTable);
//          Duplicate(m_ArcsTable);
  //testes de duplicação entre tabelas distintas
  //atributos de marca e demais
// CrossDuplicate(m_AttribTable,m_BoxesTable);
// CrossDuplicate(m_AttribTable,m_TransitsTable);
// CrossDuplicate(m_AttribTable,m_GatesTable);
// CrossDuplicate(m_AttribTable,m_ArcsTable);
  //boxes e demais
// CrossDuplicate(m_BoxesTable,m_TransitsTable);
// CrossDuplicate(m_BoxesTable,m_GatesTable);
// CrossDuplicate(m_BoxesTable,m_ArcsTable);
  //transits e demais
// CrossDuplicate(m_TransitsTable,m_GatesTable);
// CrossDuplicate(m_TransitsTable,m_ArcsTable);
  //gates e demais
// CrossDuplicate(m_GatesTable,m_ArcsTable);

            //inicializa os vetores de referência cruzada
            InitVectors();

            //criação da lista de atributos da marca (template)
            MarkAttribsCreate();

            //cross-reference & criação dos arcos
            ArcsCreate();

//cross-reference & criação dos filtros
FiltersCreate();
```

```
            //cross-reference & criação dos gates
            GatesCreate();

            //o que eu sei até agora:
            //número de arcos que saem e entram em cada box
            //número de arcos que saem e entram em cada transição
            //número de gates em cada transição

            //criação das boxes
            BoxesCreate();

            //criação das transits
            TransitsCreate();

            //com o grafo já montado
            //adicionam-se as marcas
            InitMarksCreate();

            //adicionam-se as condições
            ConditionalsCreate();

  //forçando chamada de SetGraph
  m_GraphSet = FALSE;

            //liberando memória
  delete []mp_nBOr;
  delete []mp_nBDs;
  delete []mp_nTOr;
  delete []mp_nTDs;
  delete []mp_nGts;
  delete []mp_BoxOrigin;
  delete []mp_BoxDestin;
  delete []mp_TrsOrigin;
  delete []mp_TrsDestin;
  delete []mp_GtsDestin;
}


//////////////////////////////////////////////////////////////////////////
//
// Controle das tags iniciais
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::InitialTags(BOOL bSave)
{
        short i;

        //Versão do Script
        Next(m_line);
        //checando a presença da tag
        i = m_line.Find("EMFG SCRIPT VERSION");
        if (i==-1)
                {
                MessageBox(NULL,"Faltando tag com a versão do script.","Erro na Construção das Tabelas",MB_OK);
                AfxThrowUserException();
                }
        //checando a versão do script
        i = m_line.Find(':');
        m_word = m_line.Right(m_line.GetLength()-i-1);
        m_word.TrimLeft();
        if (m_word!=SCRIPT_VERSION)
                {
                MessageBox(NULL,"Versão incompatível do script","Erro",MB_OK);
                AfxThrowUserException();
                }

        //Título do Grafo
        Next(m_line);
        //checando a presença da tag
        i = m_line.Find("PROGRAM TITLE");
```

```
        if (i==-1)
                {
                MessageBox(NULL,"Faltando tag com título do programa","Erro na Construção das Tabelas",MB_OK);
                AfxThrowUserException();
                }
        //recuperando o nome do grafo (somente 1a palavra será usada para DDE)
        i = m_line.Find(':');
        m_word = m_line.Right(m_line.GetLength()-i-1);
        m_word.TrimLeft();
        if (bSave)
                m_title = m_word;

        //Descrição do Grafo
        Next(m_line);
        //checando a presença da tag
        i = m_line.Find("DESCRIPTION");
        if (i==-1)
                {
                MessageBox(NULL,"Faltando tag com descrição do programa","Erro na Construção das Tabelas",MB_OK);
                AfxThrowUserException();
                }
        //recuperando o nome do grafo (somente 1a palavra será usada para DDE)
        i = m_line.Find(':');
        m_word = m_line.Right(m_line.GetLength()-i-1);
        m_word.TrimLeft();
        if (bSave)
                m_descr = m_word;
}

////////////////////////////////////////////////////////////////////////////
// Controle das tags de include
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::IncludeTags(CString prefix)
{
        short i;                                        //auxiliary integer
        char found;                                     //auxiliary character
        char psz[10];                           //auxiliary string

        //avança a linha
        Next(m_line);

        //procura a tag de include
        if (m_line!="<INCLUDE>")
                {
                MessageBox(NULL,m_line+" encontrado.\nenquanto se esperava <INCLUDE> tag.","Erro na construção
das tabelas",MB_OK);
                AfxThrowUserException();
                }

        //avança linha
        Next(m_line);

        //montagem da tabela de includes
        while (m_line!="</INCLUDE>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();
                        if (m_word ==           "ADD")
                                {
                                m_supported = TRUE;
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(prefix+m_word);
                                sprintf(psz,"%d",m_level+1);
                                m_param[0].Add(psz);
                                }
                        if (!m_supported)
                                {
```

```
                                        MessageBox(NULL,m_word+"/nComando não identificado","Erro na Construção das
Tabelas",MB_OK);
                                        AfxThrowUserException();
                                        }
                                m_IncludeTable.Add(m_param[0]);
                                Next(m_line);
                                }
                        }

        //nesse ponto já estão presentes na tabela de include os arquivos com os prefixos apropriados
}

////////////////////////////////////////////////////////////////////////////
// Controle das tags de atributos de marca
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::MarkAttribTags()
{
        short i;                                        //auxiliary integer
        char found;                                     //auxiliary character

        //Tag de marcas
        Next(m_line);
        if (m_line!="<MARK>")
                {
                MessageBox(NULL,m_line+" encontrado\nenquanto se esperava <MARK> tag.","Erro na construção das
tabelas",MB_OK);
                AfxThrowUserException();
                }

        Next(m_line);

        //montagem da tabela dos atributos de marca
//////////////////////////////
//                      //
//      Attributes      //
//    Literal Tables    //
//                      //
//////////////////////////////
        while (m_line!="</MARK>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();
                        if (m_word ==           "INTEGER")
                                {
                                m_supported = TRUE;
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add("INTEGER");
                                m_param[0].Add(m_word);
                                }
                        if (m_word == "STRING")
                                {
                                m_supported = TRUE;
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add("STRING");
                                m_param[0].Add(m_word);
                                }
                        if (!m_supported)
                                {
                                MessageBox(NULL,m_word+"/nComando não identificado","Erro de
Compile",MB_OK);
                                AfxThrowUserException();
                                }
                        m_AttribTable.AddOrdered(m_param[0],1);
                        Next(m_line);
                        }
                }
        //aqui não ocorre prefixação, mas as redundâncias tem que ser checadas na CrossRef()
}
```

```
////////////////////////////////////////////////////////////////////////////
//
// Controle das tags de boxes
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::BoxTags(CString prefix)
{
        short i,n;                              //auxiliary integers
 short blocknum;  //auxiliary integer
        char  found;                            //auxiliary character
 char  str[10];   //auxiliary string

 BOOL block;     //IF-THEN-ELSE block

 CString mem;    //memory box label

        //tag de boxes
        Next(m_line);
        if (m_line!="<BOXES>")
                {
                MessageBox(NULL,m_line+" encontrado.\nenquanto se esperava <BOXES> tag.","Erro na Construção das
Tabelas",MB_OK);
                AfxThrowUserException();
                }

        Next(m_line);
 ////////////////////////////
 //                        //
 //        Boxes           //
 //    Literal Tables      //
 //                        //
 ////////////////////////////

        while (m_line!="</BOXES>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();

                        if (m_word ==           "COMMON")
                                {
                                m_supported = TRUE;
                                //common box
                                m_param[0].Add("COMMON");
                                //name
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(prefix+m_word);
                                }

                        if (m_word ==           "TEMPORIZED")
                                {
                                m_supported = TRUE;
                                //common box
                                m_param[0].Add("TEMPORIZED");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //time cte
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
 long Number = atol(m_word);
 if (Number<=0)
  {
                                MessageBox(NULL,m_word+" is NULL or not a Number.\nSintaxe is
TEMPORIZED(NAME,CTE)","Erro na Construção das Tabelas",MB_OK);
                                AfxThrowUserException();
  }
                                }

                        if (m_word ==           "CAPACITY")// Sintaxe CAPACITY(NAME,N,FIFO|LIFO)
                                {
```

```
                                        m_supported = TRUE;
                                        //capacity box
                                        m_param[0].Add("CAPACITY");
                                        //name
                                        Search (m_line,i,',',m_word,i,found,TRUE);
    m_param[0].Add(m_word);
    // N
                                        Search (m_line,i,',',m_word,i,found,TRUE);
    long Number = atol(m_word);
    if (Number<=0)
     {
                                        MessageBox(NULL,m_word + " is NULL or not a Number.\nSintaxe is
CAPACITY(NAME,N)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
     }
    m_param[0].Add(m_word);
    // FIFO/LIFO
                                        Search (m_line,i,')',m_word,i,found,TRUE);
    if (m_word!="LIFO"&&m_word!="FIFO")
     {
                                        MessageBox(NULL,m_word + " is not FIFO or LIFO.\nSintaxe is
CAPACITY(NAME,N)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
     }
                                        m_param[0].Add(m_word);
                                        }

                                if (m_word ==       "PACKING")// Sintaxe PACKING(NAME,N,FIFO||LIFO)
                                        {
                                        m_supported = TRUE;
                                        //packing box
                                        m_param[0].Add("PACKING");
                                        //name
                                        Search (m_line,i,',',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
    // N
                                        Search (m_line,i,',',m_word,i,found,TRUE);
    long Number = atol(m_word);
    if (Number<=0)
     {
                                        MessageBox(NULL,m_word + " is NULL or not a Number.\nSintaxe is
PACKING(NAME,N,FIFO||LIFO)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
     }
                                        m_param[0].Add(m_word);
    // FIFO/LIFO
                                        Search (m_line,i,')',m_word,i,found,TRUE);
    if (m_word!="LIFO"&&m_word!="FIFO")
     {
                                        MessageBox(NULL,m_word + " is not FIFO or LIFO.\nSintaxe is
PACKING(NAME,N)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
     }
                                        m_param[0].Add(m_word);
                                        }

                                if (m_word ==       "UNPACKING")// Sintaxe UNPACKING(NAME,N,FIFO||LIFO)
                                        {
                                        m_supported = TRUE;
                                        //packing box
                                        m_param[0].Add("UNPACKING");
                                        //name
                                        Search (m_line,i,',',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
    // N
                                        Search (m_line,i,',',m_word,i,found,TRUE);
    long Number = atol(m_word);
    if (Number<=0)
     {
                                        MessageBox(NULL,m_word + " is NULL or not a Number.\nSintaxe is
UNPACKING(NAME,N,FIFO||LIFO)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
```

```
    }
                                        m_param[0].Add(m_word);
    // FIFO/LIFO
                                        Search (m_line,i,')',m_word,i,found,TRUE);
    if (m_word!="LIFO"&&m_word!="FIFO")
     {
                                        MessageBox(NULL,m_word + " is not FIFO or LIFO.\nSintaxe is
UNPACKING(NAME,N)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
    }
                                        m_param[0].Add(m_word);
                                        }

                                if (m_word ==       "TRANSFORMATOR")
                                        {
                                        m_supported = TRUE;
                                        //common box
                                        m_param[0].Add("TRANSFORMATOR");
                                        //name
                                        Search (m_line,i,')',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
    //guarda o nome para uso nas demais tabelas
    mem = m_word;
    //avança na tabela
    Next (m_line);

    if (m_line!="{")
     {
     MessageBox(NULL,"Sinal de { esperado após declaração de\nTRANSFORMATOR("+mem+").","Erro de
Compilação",MB_OK);
     AfxThrowUserException();
     }

    //avança na tabela
    Next(m_line);

    //inicializa p/ primeiro bloco
    blocknum = -1;

    do
     {
     //bloco ainda não encontrado
     block = FALSE;

     //encontra próxima keyword
                                Search (m_line,0,'(',m_word,i,found,TRUE);

     if (m_word=="IF")
      {
      //limpa linha do parâmetro 2
      m_param[1].RemoveAll();
      //mais um bloco
      blocknum++;
      //nome da box
      m_param[1].Add(mem);
      //expressão condicional
      n = m_line.GetLength();
      m_word = m_line.Mid(i-1,n-1);
      if (!MatchingBrackets(m_word))
       {
       MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                AfxThrowUserException();
      }
                        m_param[1].Add(m_word);
     //já adiciona na tabela
     m_TBCondTable.AddOrdered(m_param[1],0);

     ////////////////////////
     //                    //
     //    Bloco THEN      //
     //                    //
     ////////////////////////
```

```
//avança na tabela
Next(m_line);
if (m_line=="THEN")
 {
 //avança na tabela
 Next(m_line);

 if (m_line=="{")
  {
  //avança na tabela
  Next(m_line);

  //limpa linha do parâmetro 2
  m_param[1].RemoveAll();
  //adiciona nome da box
  m_param[1].Add(mem);
  //adiciona o número do bloco
  sprintf(str,"%d",blocknum);
  m_param[1].Add(str);

  while(m_line!="}")
   {
   //identificando tipos de atribuição
   m_supported = FALSE;
                                    //encontra próxima keyword
   Search (m_line,0,'(',m_word,i,found,TRUE);

   if (m_word ==          "TO_NUM")
                                                    {
                                                    m_supported = TRUE;
                                                    //to numerical constant
                                                    m_param[1].Add("TO_NUM");
                                                    //attribute
                                                    Search(m_line,i,',',m_word,i,found,TRUE);
                                                    m_param[1].Add(m_word);
                                                    //constant
                                                    Search(m_line,i,')',m_word,i,found,TRUE);
                                                    m_param[1].Add(m_word);
                                                    }

                                            if (m_word ==     "TO_STR")
                                                    {
                                                    m_supported = TRUE;
                                                    //to string constant
                                                    m_param[1].Add("TO_STR");
                                                    //attribute
                                                    Search(m_line,i,',',m_word,i,found,TRUE);
                                                    m_param[1].Add(m_word);
                                                    //constant
                                                    Search(m_line,i,')',m_word,i,found,TRUE);
                                                    m_param[1].Add(m_word);
                                                    }

   if (!m_supported)
    {
    MessageBox(NULL,"Tipo de atribuição não definido:\n"+m_word,"Erro de Compilação",MB_OK);
    AfxThrowUserException();
    }

   //avança uma linha
   Next(m_line);
   }

//adiciona na lista de atribuições associadas ao then
m_TBThenTable.AddOrdered(m_param[1],0);

//////////////////////
//                  //
//    Bloco ELSE    //
//                  //
```

```
//////////////////////
//avança na tabela
Next(m_line);
if (m_line=="ELSE")
 {
 //avança na tabela
 Next(m_line);

 if (m_line=="{")
  {
  //avança na tabela
  Next(m_line);

  //limpa linha do parâmetro 2
  m_param[1].RemoveAll();
  //adiciona nome da box
  m_param[1].Add(mem);
  //adiciona o número do bloco
  sprintf(str,"%d",blocknum);
  m_param[1].Add(str);

  while(m_line!="}")
   {
   //identificando tipos de atribuição
   m_supported = FALSE;
                                    //encontra próxima keyword
   Search (m_line,0,'(',m_word,i,found,TRUE);

   if (m_word ==      "TO_NUM")
                                                          {
                                                          m_supported = TRUE;
                                                          //to numerical constant
                                                          m_param[1].Add("TO_NUM");
                                                          //attribute
                                                          Search(m_line,i,',',m_word,i,found,TRUE);
                                                          m_param[1].Add(m_word);
                                                          //constant
                                                          Search(m_line,i,')',m_word,i,found,TRUE);
                                                          m_param[1].Add(m_word);
                                                          }

                                                 if (m_word ==  "TO_STR")
                                                          {
                                                          m_supported = TRUE;
                                                          //to string constant
                                                          m_param[1].Add("TO_STR");
                                                          //attribute
                                                          Search(m_line,i,',',m_word,i,found,TRUE);
                                                          m_param[1].Add(m_word);
                                                          //constant
                                                          Search(m_line,i,')',m_word,i,found,TRUE);
                                                          m_param[1].Add(m_word);
                                                          }

   if (!m_supported)
    {
    MessageBox(NULL,"Tipo de atribuição não definido:\n"+m_word,"Erro de Compilação",MB_OK);
    AfxThrowUserException();
    }

   Next(m_line);
   }

  //adiciona na lista de atribuições associadas ao then
  m_TBElseTable.AddOrdered(m_param[1],0);

  //chegando nesse ponto sabemos que o bloco está ok
  block = TRUE;
  }
}
```

```
          }
         }
        }

       if (!block)
        {
        MessageBox(NULL,"Erro no bloco IF - THEN - ELSE no\nTRANSFORMATOR("+mem+").","Erro de
Compilação",MB_OK);
        AfxThrowUserException();
        }

       //avança na tabela
       Next(m_line);
       }while (m_line!="}");
                                        }

    if (!m_supported)
                                        {
                                        MessageBox(NULL,m_word+"\nComando não identificado","Erro de
Compile",MB_OK);
                                        AfxThrowUserException();
                                        }

                                m_BoxesTable.AddOrdered(m_param[0],1);
                                Next(m_line);
                                }
                        }
}

//////////////////////////////////////////////////////////////////////////////
//
// Controle das tags de transições
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::TransitTags(CString Prefix)
{
        short i,n;                              //auxiliary integers
        char found;                                     //auxiliary character

 BOOL cond;     //conditional boolean

        Next(m_line);

        if (m_line!="<TRANSITIONS>")
                {
                MessageBox(NULL,m_line+" encontrado.\nEsperando uma tag <TRANSITIONS>.","Erro de
Compile",MB_OK);
                AfxThrowUserException();
                }

        Next(m_line);

 ////////////////////////////////
 //                      //
 //     Transitions      //
 //    Literal Tables    //
 //                      //
 ////////////////////////////////
        while (m_line!="</TRANSITIONS>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();
   cond = FALSE;

                        if (m_word ==           "COMMON")
                                {
                                m_supported = TRUE;
                                //common box
                                m_param[0].Add("COMMON");
                                //name
```

```
                                        Search (m_line,i,')',m_word,i,found,TRUE);
                                        m_param[0].Add(Prefix+m_word);
                                        }

    if (m_word ==     "TEMPORIZED")
                                        {
     m_supported = TRUE;
                                        //common type
                                        m_param[0].Add("TEMPORIZED");
                                        //name
                                        Search (m_line,i,',',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
                                        //time cte
                                        Search (m_line,i,')',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
     long Number = atol(m_word);
     if (Number<=0)
      {
                                        MessageBox(NULL,m_word + " is NULL or not a Number.\nSintaxe is
TEMPORIZED(NAME,CTE)","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
      }
                                        }

                                if (m_word ==          "ADD_CONDITION")
                                        {
                                        m_supported = TRUE;
     cond = TRUE;
                                        //associated transition
                                        Search (m_line,i,',',m_word,i,found,TRUE);
                                        m_param[0].Add(m_word);
                                        //conditional expression
     n = m_line.GetLength();
     m_word = m_line.Mid(i,n-1);
     if (!MatchingBrackets(m_word))
      {
      MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                         AfxThrowUserException();
      }
                                        m_param[0].Add(m_word);
                                        }

                                if (!m_supported)
                                        {
                                        MessageBox(NULL,m_word+"\nComando não identificado","Erro na Construção das
Tabelas",MB_OK);
                                        AfxThrowUserException();
                                        }


    if (!cond)
      m_TransitsTable.AddOrdered(m_param[0],1);
    else
      m_TransCondTable.AddOrdered(m_param[0],0);

    Next(m_line);
                                }
                        }
}

//////////////////////////////////////////////////////////////////////////////
//
// Controle das tags de arcos
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::ArcTags(CString Prefix)
{
        short i;                                        //auxiliary integer
        char found;                                     //auxiliary character

 BOOL filter;     //filter boolean
```

```
        Next(m_line);

        if (m_line!="<ARCS>")
                {
                MessageBox(NULL,m_line+" encontrado.\nenquanto se esperava <ARCS> tag.","Erro na Construção das
Tabelas",MB_OK);
                AfxThrowUserException();
                }

        Next(m_line);


////////////////////////////////
//                            //
//        Arcs                //
//    Literal Tables          //
//                            //
////////////////////////////////
while (m_line!="</ARCS>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();
  filter = FALSE;

                        if (m_word ==           "FROM_BOX")
                                {
                                m_supported = TRUE;
                                //common box
                                m_param[0].Add("FROM_BOX");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                //box
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                //transition
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                }

                        if (m_word ==           "FROM_TRANSITION")
                                {
                                m_supported = TRUE;
                                //common box
                                m_param[0].Add("FROM_TRANSITION");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                //box
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                //transition
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(Prefix+m_word);
                                }

  if (m_word ==     "ADD_FILTER")//Sintaxe: ADD_FILTER(name,composition,type);{ param1,param2,...}
                                {
                                m_supported = TRUE;
   filter = TRUE;
   m_param[0].RemoveAll();
                                //filter keyword
                                m_param[0].Add("FILTER");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //composite mark flag (PASS_COMPOSITE|NOT_PASS_COMPOSITE)
                                Search (m_line,i,',',m_word,i,found,TRUE);
   if (m_word!="PASS_COMPOSITE"&&m_word!="NOT_PASS_COMPOSITE")
    {
```

```
                                        MessageBox(NULL,"Composite mark filter flag required. Found "+ m_word+"
instead.","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
        }
                                        m_param[0].Add(m_word);
                                        //type
                                        Search (m_line,i,')',m_word,i,found,TRUE);
      if (m_word!="PASS_ALL"&&m_word!="NOT_PASS_ALL"&&m_word!="NOT_PASS"&&m_word!="PASS")
      {
                                        MessageBox(NULL,"Filter type required. Found "+ m_word+" instead.","Erro na
Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
        }
                                        m_param[0].Add(m_word);
                                        Next(m_line);

                                        if (m_line!="{")
                                                {
                                                MessageBox(NULL,m_line+"\nUnexpected Tag","Erro na Construção das
Tabelas",MB_OK);
                                                AfxThrowUserException();
                                                }

                                        Next(m_line);

                                        while (m_line!="}")
                                                {
                                                if (Search(m_line,0,',',m_word,i,found,TRUE,FALSE)) // Daniel 01/12
                                                        {

        // Attribute
                                                        m_param[0].Add(m_word);
                                                        }

       Next(m_line);
                                                }
      m_FilterTable.Add(m_param[0]);
                                        }

                                if (!m_supported)
                                        {
                                        MessageBox(NULL,m_word+"\nUnsupported Command","Erro na Construção das
Tabelas",MB_OK);
                                        AfxThrowUserException();
                                        }

    if (!filter)
                                 m_ArcsTable.AddOrdered(m_param[0],1);

                                Next(m_line);
                                }
                        }
}

//////////////////////////////////////////////////////////////////////////
//
// Controle das tags de gates
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::GateTags(CString Prefix)
{
        short i,j,n;                    //auxiliary integers
        char found;                                     //auxiliary character
 BOOL cond;     //conditional boolean


 Next(m_line);

        if (m_line!="<GATES>")
                {
                MessageBox(NULL,m_line+" encontrado.\nenquanto se esperava <GATES> tag.","Erro na Construção das
Tabelas",MB_OK);
                AfxThrowUserException();
                }
```

```
        Next(m_line);

////////////////////////////////
//                            //
//        Gates               //
//    Literal Tables          //
//                            //
////////////////////////////////
        while (m_line!="</GATES>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE))
                        {
                        m_supported = FALSE;
    cond = FALSE;
    m_param[0].RemoveAll();

                        if (m_word ==       "INTERNAL_PERMIT")
                                {
                                m_supported = TRUE;
                                //internal
                                m_param[0].Add("INTERNAL_PERMIT");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //origin box
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //destination transition
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                }

                        if (m_word ==       "INTERNAL_NOT_PERMIT")
                                {
                                m_supported = TRUE;
                                //internal not-permit
                                m_param[0].Add("INTERNAL_NOT_PERMIT");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //origin box
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //destination transition
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                }

                        if (m_word ==       "INTERNAL_FULL_PERMIT")
                                {
                                m_supported = TRUE;
                                //internal full permit
                                m_param[0].Add("INTERNAL_FULL_PERMIT");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //origin box
                                Search (m_line,i,',',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                //destination transition
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                }

                        if (m_word ==       "INTERNAL_FULL_NOT_PERMIT")
                                {
                                m_supported = TRUE;
                                //internal full not permit
                                m_param[0].Add("INTERNAL_FULL_NOT_PERMIT");
                                //name
                                Search (m_line,i,',',m_word,i,found,TRUE);
```

```
                                    m_param[0].Add(m_word);
                                    //origin box
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //destination transition
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    }

                            if (m_word ==       "EXTERNAL_INPUT_PERMIT")
                                    {
                                    m_supported = TRUE;
                                    //external input permit permit
                                    m_param[0].Add("EXTERNAL_INPUT_PERMIT");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //destination transition
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
j = i;
n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
 {
 while ((found!=',')&&(i<n))
  Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
 if (found!=',')
  {
  MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                AfxThrowUserException();
  }
 m_word = m_line.Mid(j,i-j-1);
 if (!MatchingBrackets(m_word))
  {
  MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                    AfxThrowUserException();
  }
 }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    }

                            if (m_word ==       "EXTERNAL_INPUT_NOT_PERMIT")
                                    {
                                    m_supported = TRUE;
                                    //external input not permit
                                    m_param[0].Add("EXTERNAL_INPUT_NOT_PERMIT");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //destination transition
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
j = i;
n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
 {
 while ((found!=',')&&(i<n))
  Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
 if (found!=',')
  {
  MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
```

```
                    AfxThrowUserException();
    }
    m_word = m_line.Mid(j,i-j-1);
    if (!MatchingBrackets(m_word))
    {
    MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                    AfxThrowUserException();
    }
    }
                    m_param[0].Add(m_word);
                    //activation
                    Search (m_line,i,')',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    }

                if (m_word ==       "EXTERNAL_OUTPUT_PERMIT")
                    {
                    m_supported = TRUE;
                    //external output permit
                    m_param[0].Add("EXTERNAL_OUTPUT_PERMIT");
                    //name
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //origin box
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //method
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //parameter
    j = i;
    n = m_line.GetLength();
                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
    {
    while ((found!=',')&&(i<n))
     Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
    if (found!=',')
     {
     MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                AfxThrowUserException();
     }
    m_word = m_line.Mid(j,i-j-1);
    if (!MatchingBrackets(m_word))
     {
     MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                    AfxThrowUserException();
     }
    }
                    m_param[0].Add(m_word);
                    //activation
                    Search (m_line,i,')',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    }

if (m_word ==    "EXTERNAL_OUTPUT_NOT_PERMIT")
                    {
                    m_supported = TRUE;
                    //external output not permit
                    m_param[0].Add("EXTERNAL_OUTPUT_NOT_PERMIT");
                    //name
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //origin box
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //method
                    Search (m_line,i,',',m_word,i,found,TRUE);
                    m_param[0].Add(m_word);
                    //parameter
    j = i;
    n = m_line.GetLength();
                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
```

```
        {
        while ((found!=',')&&(i<n))
          Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
        if (found!=',')
          {
          MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                    AfxThrowUserException();
          }
        m_word = m_line.Mid(j,i-j-1);
        if (!MatchingBrackets(m_word))
          {
          MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                    AfxThrowUserException();
          }
        }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
 }

if (m_word ==    "EXTERNAL_OUTPUT_FULL_PERMIT")
                                    {
                                    m_supported = TRUE;
                                    //external output full permit
                                    m_param[0].Add("EXTERNAL_OUTPUT_FULL_PERMIT");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //origin box
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
      j = i;
      n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
        {
        while ((found!=',')&&(i<n))
          Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
        if (found!=',')
          {
          MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                    AfxThrowUserException();
          }
        m_word = m_line.Mid(j,i-j-1);
        if (!MatchingBrackets(m_word))
          {
          MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                    AfxThrowUserException();
          }
        }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
 }

if (m_word ==    "EXTERNAL_OUTPUT_FULL_NOT_PERMIT")
                                    {
                                    m_supported = TRUE;
                                    //external output full permit
                                    m_param[0].Add("EXTERNAL_OUTPUT_FULL_NOT_PERMIT");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //origin box
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
```

```
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
        j = i;
        n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
         {
         while ((found!=',')&&(i<n))
          Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
         if (found!=',')
          {
          MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                    AfxThrowUserException();
          }
         m_word = m_line.Mid(j,i-j-1);
         if (!MatchingBrackets(m_word))
          {
          MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                      AfxThrowUserException();
          }
         }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
 }

if (m_word ==    "INTERNAL_N")
                                    {
                                    m_supported = TRUE;
                                    //internal n
                                    m_param[0].Add("INTERNAL_N");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //origin box
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
 }

if (m_word ==    "EXTERNAL_OUTPUT_DATA")
                                    {
                                    m_supported = TRUE;
                                    //external output data
                                    m_param[0].Add("EXTERNAL_OUTPUT_DATA");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //box
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //atributo
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
        j = i;
        n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
         {
         while ((found!=',')&&(i<n))
          Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
         if (found!=',')
          {
          MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                    AfxThrowUserException();
          }
         m_word = m_line.Mid(j,i-j-1);
         if (!MatchingBrackets(m_word))
          {
```

```
        MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                    AfxThrowUserException();
        }
       }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
  }

 if (m_word ==   "EXTERNAL_OUTPUT_N")
                                    {
                                    m_supported = TRUE;
                                    //external output n
                                    m_param[0].Add("EXTERNAL_OUTPUT_N");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //origin box
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
     j = i;
     n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
      {
      while ((found!=',')&&(i<n))
       Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
      if (found!=',')
       {
       MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                    AfxThrowUserException();
       }
      m_word = m_line.Mid(j,i-j-1);
      if (!MatchingBrackets(m_word))
       {
       MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                    AfxThrowUserException();
       }
      }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
 }

if (m_word ==   "EXTERNAL_INPUT_DATA")
                                    {
                                    m_supported = TRUE;
                                    //internal full permit
                                    m_param[0].Add("EXTERNAL_INPUT_DATA");
                                    //name
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //return type
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //method
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //parameter
     j = i;
     n = m_line.GetLength();
                                    if (!Search (m_line,i,',',m_word,i,found,TRUE,FALSE))
      {
      while ((found!=',')&&(i<n))
       Search (m_line,i,',',m_word,i,found,TRUE,FALSE);
      if (found!=',')
       {
```

```
            MessageBox(NULL,"Caracter ',' não foi encontrado\nna expressão : "+m_line,"Erro de busca",MB_OK);
                        AfxThrowUserException();
            }
           m_word = m_line.Mid(j,i-j-1);
           if (!MatchingBrackets(m_word))
            {
            MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                    AfxThrowUserException();
            }
           }
                                    m_param[0].Add(m_word);
                                    //activation
                                    Search (m_line,i,')',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
    }

                            if (m_word ==       "ADD_CONDITION")
                                    {
                                    m_supported = TRUE;
    cond = TRUE;
                                    //associated gate
                                    Search (m_line,i,',',m_word,i,found,TRUE);
                                    m_param[0].Add(m_word);
                                    //conditional expression
    n = m_line.GetLength();
    m_word = m_line.Mid(i,n-i-1);
    if (!MatchingBrackets(m_word))
     {
     MessageBox(NULL,m_word+"\nCheque a ordem de abertura e fechamento.","Erro de Parênteses",MB_OK);
                                     AfxThrowUserException();
     }
                                    m_param[0].Add(m_word);
                                    }

   if (!m_supported)
                                    {
                                    MessageBox(NULL,m_word+"\nComando não identificado","Erro na Construção das
Tabelas",MB_OK);
                                    AfxThrowUserException();
                                    }

   if (!cond)
     m_GatesTable.AddOrdered(m_param[0],1);
   else
     m_GatesCondTable.AddOrdered(m_param[0],0);

   Next(m_line);
                            }
                    }
}

//////////////////////////////////////////////////////////////////////////////
//
// Controle das tags de marcação inicial
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::InitMarkTags()
{
        short i;                                        //auxiliary integer
        char found;                                     //auxiliary character

 CString mem;    //memorized box

 Next(m_line);

        if (m_line!="<INITIAL>")
                {
                MessageBox(NULL,m_line+" encontrado.\nenquanto se esperava <INITIAL> tag.","Erro na Construção das
Tabelas",MB_OK);
                AfxThrowUserException();
                }
```

```
        Next(m_line);

////////////////////////////////
//                            //
//    Initial Marks           //
//    Literal Tables          //
//                            //
////////////////////////////////
        while (m_line!="</INITIAL>")
                {
                if (Search(m_line,0,'(',m_word,i,found,TRUE,FALSE))
                        {
                        m_supported = FALSE;
                        m_param[0].RemoveAll();

                        if (m_word ==           "ADD_MARK")
                                {
                                m_supported = TRUE;
                                //initial
                                m_param[0].Add("SINGLE");
                                //box
                                Search (m_line,i,')',m_word,i,found,TRUE);
                                m_param[0].Add(m_word);
                                mem = m_word;

                                Next(m_line);

                                if (m_line!="{")
                                        {
                MessageBox(NULL,m_line+"\nTag Inesperada","Erro na Construção das Tabelas",MB_OK);
                                        AfxThrowUserException();
                                        }

                                Next(m_line);

                                m_param[1].RemoveAll();

                                while (m_line!="}")
                                        {
                                        if (Search(m_line,0,'(',m_word,i,found,TRUE,FALSE))
                                                {
                                                m_supported = FALSE;

                                                if (m_word ==           "TO_NUM")
                                                        {
                                                        m_supported = TRUE;
                                                        //belongs to the current box
                                                        m_param[1].Add(mem);
                                                        //to numerical constant
                                                        m_param[1].Add("TO_NUM");
                                                        //attribute
                                                        Search(m_line,i,',',m_word,i,found,TRUE);
                                                        m_param[1].Add(m_word);
                                                        //constant
                                                        Search(m_line,i,')',m_word,i,found,TRUE);
                                                        m_param[1].Add(m_word);
                                                        }


                                                if (m_word ==           "TO_STR")
                                                        {
                                                        m_supported = TRUE;
                                                        //belongs to the current box
                                                        m_param[1].Add(mem);
                                                        //to string constant
                                                        m_param[1].Add("TO_STR");
                                                        //attribute
                                                        Search(m_line,i,',',m_word,i,found,TRUE);
                                                        m_param[1].Add(m_word);
                                                        //constant
                                                        Search(m_line,i,')',m_word,i,found,TRUE);
                                                        m_param[1].Add(m_word);
```

```
                                                                    }


                                                                    if (!m_supported)
                                                                            {
                                                            MessageBox(NULL,m_word+"\nComando não identificado","Erro na Construção das
Tabelas",MB_OK);
                                                                            AfxThrowUserException();
                                                                            }
        Next(m_line);
                                                            }
                                                            else
        {
                        MessageBox(NULL,m_line+"\nErro nessa linha!","Erro na Construção das Tabelas",MB_OK);
                        AfxThrowUserException();
        }
       }
                                                    }

                                    if (!m_supported)
                                                    {
                                                    MessageBox(NULL,m_word+"\nComando não identificado","Erro na Construção das
Tabelas",MB_OK);
                                                    AfxThrowUserException();
                                                    }
    if (m_param[1].GetSize()==0)
       m_param[1].Add(mem);
    m_MarksTable.Add(m_param[1]);
                                    Next(m_line);
                                    }
   else
     {
    MessageBox(NULL,m_line+"\nErro nessa linha!","Erro na Construção das Tabelas",MB_OK);
                        AfxThrowUserException();
    }
                        }
            Next(m_line);
}

//////////////////////////////////////////////////////////////////////////////
// inicialização dos vetores de referência cruzada
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::InitVectors()
{
            short i,j;

 ////////////////////////
 //                    //
 //    Inicialização   //
 //                    //
 ////////////////////////

            //inicializa vetores dos boxes
            m_nB = m_BoxesTable.GetSize();
 mp_nBOr = new short[m_nB];
            mp_nBDs = new short[m_nB];

 mp_BoxOrigin = new long[m_nB][NMAX];
 mp_BoxDestin = new long[m_nB][NMAX];

 for (i=0;i<m_nB;i++)
                        {
                        mp_nBOr[i]=-1;
                        mp_nBDs[i]=-1;
    for (j=0;j<NMAX;j++)
      {
      mp_BoxOrigin[i][j]=-1;
      mp_BoxDestin[i][j]=-1;
      }
                        }
```

```
            //inicializa vetores das transitions
            m_nT = m_TransitsTable.GetSize();
            mp_nTOr = new short[m_nT];
            mp_nTDs = new short[m_nT];

  mp_TrsOrigin = new long[m_nT][NMAX];
  mp_TrsDestin = new long[m_nT][NMAX];

  for (i=0;i<m_nT;i++)
                        {
                        mp_nTOr[i]=-1;
                        mp_nTDs[i]=-1;
     for (j=0;j<NMAX;j++)
       {
      mp_TrsOrigin[i][j]=-1;
      mp_TrsDestin[i][j]=-1;
      }
                        }

            //inicializa vetor de contagem dos gates
            mp_nGts = new short[m_nT];
  mp_GtsDestin = new long[m_nT][NMAX];

            for (i=0;i<m_nT; i++)
   {
                        mp_nGts[i] = -1;
   for (j=0;j<NMAX;j++)
     mp_GtsDestin[i][j]=-1;
   }

  //inicializa número de filtros
  m_nA   = m_ArcsTable.GetSize();

  //inicializa número de filtros
  m_nF   = m_FilterTable.GetSize();

  //inicializa número de condições associadas a gates
  m_nGC  = m_GatesCondTable.GetSize();

  //inicializa número de condições associadas a transições
  m_nTC  = m_TransCondTable.GetSize();

  //inicializa número de blocos nos TBs
  m_nTBC = m_TBCondTable.GetSize();
}

////////////////////////////////////////////////////////////////////////////
// criação do template de atributos de marca
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::MarkAttribsCreate()
{
            short i,j,n;
            BOOL  exist;
            CMarkAttribute* myAttrib;

            //criação da lista de atributos da marca (template)
            m_nAtb = m_AttribTable.GetSize();

  for (i=0;i<m_nAtb;i++)
                        {
   exist  = FALSE;

   m_word = m_AttribTable.GetAt(i).GetAt(0);
                        if (m_word == "INTEGER")
                                    m_index[0] = INTEGER_ATRIB;

                        if (m_word == "STRING")
                                    m_index[0] = TEXT_ATRIB;

                        m_word = m_AttribTable.GetAt(i).GetAt(1);
                        n = m_AttribTable.GetSize();
```

```
                    j = i+1;

                    //caso haja duplicação de atributo devido ao include!
                    while ((j<n)&&(!exist))
                            {
                            m_par[0] = m_AttribTable.GetAt(j).GetAt(1);
                            if (m_word == m_par[0])
                                    exist=TRUE;
                            j++;
                            }

                    if (!exist)
                            {
                            myAttrib = new CMarkAttribute;
                            myAttrib->Create(m_word, m_index[0]);
                            mp_Owner->GetAttribTemplates()->Add(*myAttrib);
                            delete myAttrib;
                            }
                    }
}

//////////////////////////////////////////////////////////////////////////
//
// criação dos arcos
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::ArcsCreate()
{
        short i;

        CArc* myArc;

 ////////////////////////////
 //                        //
 //        Arcos           //
 //    Cross - Reference   //
 //                        //
 ////////////////////////////

 //cross-reference & criação dos arcos
        for (i=0; i<m_nA; i++)
                {
                //pega linha da tabela
                m_param[0].RemoveAll();
                m_param[0] = m_ArcsTable.GetAt(i);

                //pega 1a coluna da linha
                m_par[0] = m_param[0].GetAt(0);

                //cria arco e faz cross-reference
                if (m_par[0]=="FROM_BOX")
                        {
                        m_index[0] = FROM_BOX;

                        m_par[0] = m_param[0].GetAt(2);
                        SeekOrderedTable(m_BoxesTable,1,m_par[0],m_index[1]);
   mp_nBOr[m_index[1]]++;
   mp_BoxOrigin[m_index[1]][mp_nBOr[m_index[1]]]=i;

                        m_par[0] = m_param[0].GetAt(3);
                        SeekOrderedTable(m_TransitsTable,1,m_par[0],m_index[2]);
   mp_nTDs[m_index[2]]++;
   mp_TrsDestin[m_index[2]][mp_nTDs[m_index[2]]]=i;

                        m_par[0] = m_param[0].GetAt(1);

                        myArc = new CArc;
                        myArc->Create(mp_Owner,m_par[0],m_index[0],m_index[1],m_index[2]);
                        }

                //cria arco e faz cross-reference
                if (m_par[0] == "FROM_TRANSITION")
```

```
                                {
                                m_index[0] = FROM_TRANSITION;

                                m_par[0] = m_param[0].GetAt(2);
                                SeekOrderedTable(m_TransitsTable,1,m_par[0], m_index[1]);
    mp_nTOr[m_index[1]]++;
    mp_TrsOrigin[m_index[1]][mp_nTOr[m_index[1]]]=i;

                                m_par[0] = m_param[0].GetAt(3);
                                SeekOrderedTable(m_BoxesTable,1,m_par[0],m_index[2]);
    mp_nBDs[m_index[2]]++;
    mp_BoxDestin[m_index[2]][mp_nBDs[m_index[2]]]=i;

                                m_par[0] = m_param[0].GetAt(1);

                                myArc = new CArc;
                                myArc->Create(mp_Owner,m_par[0],m_index[0],m_index[1],m_index[2]);
                                }

                        //insere arco na lista de arcos
                        mp_Owner->GetArcs()->Add(*myArc);
                        delete myArc;
                        }
}

//////////////////////////////////////////////////////////////////////////////
//
// criação dos filtros
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::FiltersCreate()
{
            short i;

 for (i=0;i<m_nF;i++)
  {
  long* AttrRef=NULL;
  long nRef=0;
  m_param[0].RemoveAll();
  m_param[0] = m_FilterTable.GetAt(i);
  //Qual Arco?
  m_word = m_param[0].GetAt(1);
  SeekTable(m_ArcsTable,1,m_word,m_index[0]);
  //Composite Flag
  m_word = m_param[0].GetAt(2);
  if (m_word=="PASS_COMPOSITE")
   {
   m_index[1] = PASS_COMPOSITE;
   }
  else
   {
   m_index[1] = NOT_PASS_COMPOSITE;
   }
  //Type
  m_word = m_param[0].GetAt(3);
  if (m_word=="PASS_ALL")
   {
   m_index[2] = PASS_ALL;
   }
  if (m_word=="NOT_PASS_ALL")
   {
   m_index[2] = NOT_PASS_ALL;// Daniel 01/12
   }
  if (m_word=="NOT_PASS")
   {
   m_index[2] = NOT_PASS;
   nRef = m_param[0].GetSize() - 4;
   if (nRef>0)
    {
    AttrRef=new long[nRef];
    }
   for (long AttIndex=4;AttIndex<m_param[0].GetSize();AttIndex++)
```

```
    {
    m_word = m_param[0].GetAt(AttIndex);
    SeekTable(m_AttribTable,1,m_word,m_index[3]);
    AttrRef[AttIndex-4] = m_index[3];
    }
   }
  if (m_word=="PASS")
   {
   m_index[2] = PASS;
   nRef = m_param[0].GetSize() - 4;
   if (nRef>0)
    {
    AttrRef=new long[nRef];
    }
   for (long AttIndex=4;AttIndex<m_param[0].GetSize();AttIndex++)
    {
    m_word = m_param[0].GetAt(AttIndex);
    SeekTable(m_AttribTable,1,m_word,m_index[3]);
    AttrRef[AttIndex-4] = m_index[3];
    }
   }
  CArcFilter MyFilter;
  MyFilter.Create(mp_Owner,m_index[2],nRef,AttrRef,m_index[1]);
  mp_Owner->GetArcs()->ElementAt(m_index[0]).SetFilter(MyFilter);
  if (AttrRef!=NULL)
   delete [] AttrRef;
  }
}

////////////////////////////////////////////////////////////////////////////
// criação dos gates
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::GatesCreate()
{
        short i;

        CGate* myGate;

 CWordsArray line;

 CString param[6];

        //cross-reference & criação dos gates
 ////////////////////////////////
 //                            //
 //        Gates               //
 //    Cross - Reference       //
 //                            //
 ////////////////////////////////
        m_nG = m_GatesTable.GetSize();
        for (i=0;i<m_nG; i++)
                {
                //pega linha da tabela
                line.RemoveAll();
                line = m_GatesTable.GetAt(i);

                //pega 1a coluna da tabela
                m_word = line.GetAt(0);

                //cria gate e faz cross-reference
                if (m_word=="INTERNAL_PERMIT")
                        {
                        m_index[0] = INT_PERMIT;
                        //name
                        param[0] = line.GetAt(1);
                        //box
                        param[1] = line.GetAt(2);
                        //transit
                        param[2] = line.GetAt(3);
                        //transformando os nomes em índices
                        SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
```

```
                                SeekOrderedTable(m_TransitsTable, 1, param[2], m_index[2]);
                                //cross-reference
    mp_nGts[m_index[2]]++;
    mp_GtsDestin[m_index[2]][mp_nGts[m_index[2]]]=i;
    //criação do gate
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
m_index[2],"INTERNAL","",NULO);
                                }

                        //cria gate e faz cross-reference
                        if (m_word=="INTERNAL_NOT_PERMIT")
                                {
                                m_index[0] = INT_NOT_PERMIT;
                                //name
                                param[0] = line.GetAt(1);
                                //box
                                param[1] = line.GetAt(2);
                                //transit
                                param[2] = line.GetAt(3);
                                //transformando os nomes em índices
                                SeekTable(m_BoxesTable, 1, param[1], m_index[1]);
                                SeekTable(m_TransitsTable, 1, param[2], m_index[2]);
                                //cross-reference
    mp_nGts[m_index[2]]++;
    mp_GtsDestin[m_index[2]][mp_nGts[m_index[2]]]=i;
    //criação do gate
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
m_index[2],"INTERNAL","",NULO);
                                }

                        //cria gate e faz cross-reference
                        if (m_word=="INTERNAL_FULL_PERMIT")
                                {
                                m_index[0] = INT_FULL_PERMIT;
                                //name
                                param[0] = line.GetAt(1);
                                //box
                                param[1] = line.GetAt(2);
                                //transit
                                param[2] = line.GetAt(3);
                                //transformando os nomes em índices
                                SeekTable(m_BoxesTable, 1, param[1], m_index[1]);
                                SeekTable(m_TransitsTable, 1, param[2], m_index[2]);
                                //cross-reference
    mp_nGts[m_index[2]]++;
    mp_GtsDestin[m_index[2]][mp_nGts[m_index[2]]]=i;
    //criação do gate
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
m_index[2],"INTERNAL","",NULO);
                                }

                        //cria gate e faz cross-reference
                        if (m_word=="INTERNAL_FULL_NOT_PERMIT")
                                {
                                m_index[0] = INT_FULL_NOT_PERMIT;
                                //name
                                param[0] = line.GetAt(1);
                                //box
                                param[1] = line.GetAt(2);
                                //transit
                                param[2] = line.GetAt(3);
                                //transformando os nomes em índices
                                SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                SeekOrderedTable(m_TransitsTable, 1, param[2], m_index[2]);
                                //cross-reference
    mp_nGts[m_index[2]]++;
    mp_GtsDestin[m_index[2]][mp_nGts[m_index[2]]]=i;
    //criação do gate
                                myGate = new CGate;
```

```
                                    myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
m_index[2],"INTERNAL","",NULO);
                                    }

                          //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_INPUT_PERMIT")
                                    {
                                    m_index[0] = EXTERNAL_INPUT_PERMIT;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //transit
                                    param[1] = line.GetAt(2);
                                    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                    //transformando os nomes em índices
                                    SeekOrderedTable(m_TransitsTable, 1, param[1], m_index[1]);
                                    //cross-reference
    mp_nGts[m_index[1]]++;
    mp_GtsDestin[m_index[1]][mp_nGts[m_index[1]]]=i;
    //criação do gate
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], NULO, NULO,
m_index[1],"DDE",param[3],m_index[2]);
                                    }

                          //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_INPUT_NOT_PERMIT")
                                    {
                                    m_index[0] = EXTERNAL_INPUT_NOT_PERMIT;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //transit
                                    param[1] = line.GetAt(2);
                                    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                    //transformando os nomes em índices
                                    SeekOrderedTable(m_TransitsTable, 1, param[1], m_index[1]);
                                    //cross-reference
                                    myGate = new CGate;
                                    //cross-reference
    mp_nGts[m_index[1]]++;
    mp_GtsDestin[m_index[1]][mp_nGts[m_index[1]]]=i;
    //criação do gate
```

```
                                    myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], NULO, NULO,
m_index[1],"DDE",param[3],m_index[2]);
                                    }

                          //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_OUTPUT_PERMIT")
                                    {
                                    m_index[0] = EXTERNAL_OUTPUT_PERMIT;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //box
                                    param[1] = line.GetAt(2);
                                    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                    //transformando os nomes em índices
                                    SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
NULO,"DDE",param[3],m_index[2]);
                                    }

                          //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_OUTPUT_NOT_PERMIT")
                                    {
                                    m_index[0] = EXTERNAL_OUTPUT_NOT_PERMIT;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //box
                                    param[1] = line.GetAt(2);
                                    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                    //transformando os nomes em índices
                                    SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
NULO,"DDE",param[3],m_index[2]);
                                    }

                          //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_OUTPUT_FULL_PERMIT")
                                    {
                                    m_index[0] = EXTERNAL_OUTPUT_FULL_PERMIT;
                                    //name
                                    param[0] = line.GetAt(1);
```

```
                                //box
                                param[1] = line.GetAt(2);
                                //method
                                param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                param[3] = line.GetAt(4);
                                //activation
                                param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                //transformando os nomes em índices
                                SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
NULO,"DDE",param[3],m_index[2]);
                                }

                        //cria gate e faz cross-reference
  if (m_word=="EXTERNAL_OUTPUT_FULL_NOT_PERMIT")
                                {
                                m_index[0] = EXTERNAL_OUTPUT_FULL_NOT_PERMIT;
                                //name
                                param[0] = line.GetAt(1);
                                //box
                                param[1] = line.GetAt(2);
                                //method
                                param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                param[3] = line.GetAt(4);
                                //activation
                                param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                //transformando os nomes em índices
                                SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], BOOLEAN, param[0], m_index[1], NULO,
NULO,"DDE",param[3],m_index[2]);
                                }

                        //cria gate e faz cross-reference
  if (m_word=="INTERNAL_N")
                                {
                                m_index[0] = INT_N;
                                //name
                                param[0] = line.GetAt(1);
                                //box
                                param[1] = line.GetAt(2);
                                //transformando os nomes em índices
                                SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                myGate = new CGate;
                                myGate->Create(mp_Owner,m_index[0], INTEGER_ATRIB, param[0], m_index[1], NULO,
NULO,"INTERNAL","",NULO);
                                }

                        //cria gate e faz cross-reference
```

```
    if (m_word=="EXTERNAL_OUTPUT_DATA")
                                    {
                                    m_index[0] = EXTERNAL_OUTPUT_DATA;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //box
                                    param[1] = line.GetAt(2);
                                    //atributo
                                    param[2] = line.GetAt(3);
                                    //method
                                    param[3] = line.GetAt(4);
     if (param[3]!="DDE")
      {
      MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
      AfxThrowUserException();
      return;
      }
                                    //parameter
                                    param[4] = line.GetAt(5);
     //transformando os nomes em índices
     SeekOrderedTable(m_BoxesTable,1,param[1],m_index[1]);
     SeekOrderedTable(m_AttribTable,1,param[2],m_index[2]);
     m_index[3]=mp_Owner->GetAttribTemplates()->GetAt(m_index[2]).GetType();
                                    //activation
                                    param[5] = line.GetAt(6);
     if (param[5]=="ACTIVE")
      m_index[4] = ACTIVE;
     else
      m_index[4] = PASSIVE;
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], m_index[3], param[0], m_index[1], m_index[2],
NULO, param[3], param[4], m_index[4]);
                                    }

   //cria gate e faz cross-reference
   if (m_word=="EXTERNAL_OUTPUT_N")
                                    {
                                    m_index[0] = EXTERNAL_OUTPUT_N;
                                    //name
                                    param[0] = line.GetAt(1);
                                    //box
                                    param[1] = line.GetAt(2);
                                    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
   //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
   if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
   else
     m_index[2] = PASSIVE;

                                    //transformando os nomes em índices
                                    SeekOrderedTable(m_BoxesTable, 1, param[1], m_index[1]);
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], INTEGER_ATRIB, param[0], m_index[1], NULO,
NULO,param[1],param[2],m_index[2]);
                                    }


  //cria gate e faz cross-reference
  if (m_word=="EXTERNAL_INPUT_DATA")
                                    {
                                    m_index[0] = EXTERNAL_INPUT_DATA;
                                    //name
```

```
                                    param[0] = line.GetAt(1);
                                    //return type
                                    param[1] = line.GetAt(2);
    if (param[1]=="INTEGER")
     m_index[1]=INTEGER_ATRIB;
    else
     m_index[1]=TEXT_ATRIB;
    //method
                                    param[2] = line.GetAt(3);
    if (param[2]!="DDE")
     {
     MessageBox(NULL,param[2]+"não é um método identificado.","Erro na construção das tabelas",MB_OK);
     AfxThrowUserException();
     return;
     }
    //parameter
                                    param[3] = line.GetAt(4);
                                    //activation
                                    param[4] = line.GetAt(5);
    if (param[4]=="ACTIVE")
     m_index[2] = ACTIVE;
    else
     m_index[2] = PASSIVE;
                                    //transformando os nomes em índices
                                    myGate = new CGate;
                                    myGate->Create(mp_Owner,m_index[0], m_index[1], param[0], NULO, NULO,
NULO,param[2],param[3],m_index[2]);
                                    }

   mp_Owner->GetGates()->Add(*myGate);
                        delete myGate;
                        }
}

////////////////////////////////////////////////////////////////////////////
// criação das boxes
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::BoxesCreate()
{
 short i,j,n,m,p;

 CCondArray   Conditions;     //TB Blocks
 CAtbnMatrix  ThenAttribs;    //TB THEN
 CAtbnMatrix  ElseAttribs;    //TB ELSE
 CAttrbnArray Attributions;   //lista de atribuições


 CWordsArray line;

 CString param[5];

 CString RName[3];

 short type;

 CBox* myBox;

 char ch;

 //cria a box baseado no cross reference já feito
 ////////////////////////////////
 //                            //
 //         Boxes              //
 //     Cross - Reference      //
 //                            //
 ////////////////////////////////
          for (i=0;i<m_nB;i++)
                        {
                        //nesse ponto os vetores já contém
                        //os índices para os arcos
```

```
                    line.RemoveAll();
                    line = m_BoxesTable.GetAt(i);
                    m_word = line.GetAt(0);

                    if (m_word=="COMMON")
                                {
                                m_index[0] = COMMON_BOX;
                                //name
                                param[0] = line.GetAt(1);
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0], mp_nBDs[i]+1, mp_nBOr[i]+1,
mp_BoxDestin[i], mp_BoxOrigin[i]);
                                }

   if (m_word=="TEMPORIZED")
                                {
                                m_index[0] = COMMON_BOX; // somente o box comum pode ser temporizado
                                //name
                                param[0] = line.GetAt(1);
    //Time Cte
    param[1] = line.GetAt(2);
    long TimeCte = atol(param[1]);
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0],mp_nBDs[i]+1, mp_nBOr[i]+1,
mp_BoxDestin[i], mp_BoxOrigin[i]);
                                myBox->SetBoxAsTemporized(TimeCte); // Box m_Type data member receives TEMP_BOX
value
    }

                    if (m_word=="CAPACITY")
                                {
    // type
                                m_index[0] = CAPACITY_BOX;
                                //name
                                param[0] = line.GetAt(1);
    //N
    param[1] = line.GetAt(2);
    long N = atol(param[1]);
    //LIFO or FIFO
    param[2] = line.GetAt(3);
    if (param[2]=="FIFO")
     {
     m_index[1] = TRUE;
     }
    else
     {
     m_index[1] = FALSE;
     }
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0], mp_nBDs[i]+1, mp_nBOr[i]+1,
mp_BoxDestin[i], mp_BoxOrigin[i],N,m_index[1]);
   }

                    if (m_word=="PACKING")
                                {
   // type
                                m_index[0] = PACKING_BOX;
                                //name
                                param[0] = line.GetAt(1);
   //N
   param[1] = line.GetAt(2);
   long N = atol(param[1]);
   //LIFO or FIFO
   param[2] = line.GetAt(3);
   if (param[2]=="FIFO")
    {
    m_index[1] = TRUE;
    }
   else
```

```
    {
    m_index[1] = FALSE;
    }
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0], mp_nBDs[i]+1, mp_nBOr[i]+1,
mp_BoxDestin[i], mp_BoxOrigin[i],N,m_index[1]);
    }


                if (m_word=="UNPACKING")
                                {
    // type
                                m_index[0] = UNPACKING_BOX;
                                //name
                                param[0] = line.GetAt(1);
    //N
    param[1] = line.GetAt(2);
    long N = atol(param[1]);
    //LIFO or FIFO
    param[2] = line.GetAt(3);
    if (param[2]=="FIFO")
     {
     m_index[1] = TRUE;
     }
    else
     {
     m_index[1] = FALSE;
     }
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0],  mp_nBOr[i]+1, mp_nBDs[i]+1,
mp_BoxOrigin[i], mp_BoxDestin[i],N,m_index[1]);
    }


                if (m_word=="TRANSFORMATOR")
                                {
                                m_index[0] = TRANSFORMATOR_BOX;
                                //name
                                param[0] = line.GetAt(1);
                                //cria novo box
                                myBox = new CBox;
                                myBox->Create(mp_Owner, m_index[0], param[0], mp_nBDs[i]+1, mp_nBOr[i]+1,
mp_BoxDestin[i], mp_BoxOrigin[i]);
    //agora faz referência cruzada dos blocos
    //declara vetor de condições e matriz de atribuições
    Conditions.RemoveAll();
    ThenAttribs.RemoveAll();
    ElseAttribs.RemoveAll();

    //encontra 1o bloco do box
    SeekOrderedTable(m_TBCondTable,0,param[0],m_index[0]);
    m_word = m_TBCondTable.GetAt(m_index[0]).GetAt(0);
    p = m_TBCondTable.GetSize();

    //enquanto estivermos na mesma box e não for final da lista
    while ((m_word == param[0])&&(m_index[0]<p))
     {
     //monta o bloco

     //monta o condicional
     param[1]=m_TBCondTable.GetAt(m_index[0]).GetAt(1);
     CConditional* myCond;
     myCond = new CConditional;
     //cria a condição com box local dada por i
     BuildCond(*myCond,param[1],i);
     //adiciona a condicional na lista
     Conditions.Add(*myCond);
     //desaloca memória
     delete myCond;
```

```
//monta as atribuições

//limpa a lista de atribuições atual
Attributions.RemoveAll();

//bloco THEN
//encontra 1a atribuição then da box
SeekOrderedTable(m_TBThenTable,0,param[0],m_index[1]);
m_word = m_TBThenTable.GetAt(m_index[1]).GetAt(0);

m = m_TBThenTable.GetSize();

while ((m_word == param[0])&&(m_index[1]<m))
 {
 m_index[2] = atoi(m_TBThenTable.GetAt(m_index[1]).GetAt(1));
 if (m_index[2]==m_index[0])
  {
  n = m_TBThenTable.GetAt(m_index[1]).GetSize();
  for (j=2;j<n;)
   {
   param[1] = m_TBThenTable.GetAt(m_index[1]).GetAt(j);
   if (param[1]=="TO_NUM")
    {
    //monta atribuição
    CAttribution* myAtbn;
    myAtbn = new CAttribution;
    //atributo
    param[2] = m_TBThenTable.GetAt(m_index[1]).GetAt(j+1);
    SeekOrderedTable(m_AttribTable,1,param[2],m_index[2]);
    //define se é cte numérica, atributo ou gate
    param[2] = m_TBThenTable.GetAt(m_index[1]).GetAt(j+2);
    ParseAddress (param[2],RName[0],RName[1],type);
    if (type == SINGLE)                    //tipo simples -> cte string, cte numérica, box local
     {
     m_index[3] = atoi (RName[0]);
     if ((m_index[3]==0)&&(RName[0]!='0'))          //não é numérico, pois não foi convertido com sucesso
      {
      //endereço pode estar se referindo à um gate externo de entrada de dados ou atributo local
      if (SeekOrderedTable(m_AttribTable,1,RName[0],m_index[3],FALSE))
       {
       //atributo local
       if (m_AttribTable.GetAt(m_index[3]).GetAt(0)=="INTEGER")
        {
        myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],i,m_index[3]);
        }
       else
        {
        MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nAtributo não é INTEGER","Erro
Encontrado",MB_OK);
        AfxThrowUserException();
        }
       }
      else
       {
       //gate externo de entrada de dados
       if (SeekOrderedTable(m_GatesTable,1,RName[0],m_index[3]))
        {
        if (m_GatesTable.GetAt(m_index[3]).GetAt(2)=="INTEGER")
         {
         myAtbn->Create(EQUAL_TO_GATE,i,m_index[2],m_index[3]);
         }
        else
         {
         MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate não compatível com operação de atribuição
INTEGER","Erro Encontrado",MB_OK);
         AfxThrowUserException();
         }
        }
       else
        {
        MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate ou atributo não está declarado","Erro
Encontrado",MB_OK);
```

```
                AfxThrowUserException();
                }
               }
              }
             else
              {
              //atribuição numérica cte
              //checagem de tipo com LValue é necessária
              if (m_AttribTable.GetAt(m_index[2]).GetAt(0)!="INTEGER")
               {
               MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[2]).GetAt(1)+"\nAtributo não é do tipo
INTEGER","Erro Encontrado",MB_OK);
               AfxThrowUserException();
               return;
               }
              //cria a atribuição simples
              myAtbn->Create(EQUAL_TO_NUM_CTE,i,m_index[2],m_index[3]);
              }
             }
            else
             {
             //endereço deve ser um box!atributo
             if (SeekOrderedTable(m_BoxesTable,1,RName[0],m_index[3]))
              {
              if (SeekOrderedTable(m_AttribTable,1,RName[1],m_index[4]))
               {
               if (m_AttribTable.GetAt(m_index[4]).GetAt(0)=="INTEGER")
                {
                myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],m_index[3],m_index[4]);
                }
               else
                {
                MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[4]).GetAt(1)+"\nAtributo não é do tipo
INTEGER","Erro Encontrado",MB_OK);
                AfxThrowUserException();
                }
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+RName[1]+"\nAtributo não está declarado","Erro
Encontrado",MB_OK);
               AfxThrowUserException();
               }
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nBox não está declarada","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            //adiciona nessa linha de atribuições
            Attributions.Add(*myAtbn);
            //desaloca memória
            delete myAtbn;
            //próxima coluna
            j += 3;
            }

           if (param[1]=="TO_STR")
            {
            //monta atribuição
            CAttribution* myAtbn;
            myAtbn = new CAttribution;
            //atributo
            param[2] = m_TBThenTable.GetAt(m_index[1]).GetAt(j+1);
            SeekOrderedTable(m_AttribTable,1,param[2],m_index[2]);
            //define se é cte numérica, atributo ou gate
            param[2] = m_TBThenTable.GetAt(m_index[1]).GetAt(j+2);
            ParseAddress (param[2],RName[0],RName[1],type);
            if (type == SINGLE)                    //tipo simples -> cte string, cte numérica, atributo local
             {
             ch = RName[0].GetAt(0);
```

```
            if (ch!='\')                    //começa com aspas simples -> é string cte
             {
             //endereço pode estar se referindo à um gate externo de entrada de dados
             if (SeekOrderedTable(m_AttribTable,1,RName[0],m_index[3],FALSE))
              {
              //atributo local
              if (m_AttribTable.GetAt(m_index[3]).GetAt(0)=="STRING")
               {
               myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],i,m_index[3]);
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nAtributo não é STRING","Erro Encontrado",MB_OK);
               AfxThrowUserException();
               }
              }
             else
              {
              //gate externo de entrada de dados
              if (SeekOrderedTable(m_GatesTable,1,RName[0],m_index[3]))
               {
               if (m_GatesTable.GetAt(m_index[3]).GetAt(2)=="STRING")
                {
                myAtbn->Create(EQUAL_TO_GATE,i,m_index[2],m_index[3]);
                }
               else
                {
                MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate não compatível com operação de atribuição
STRING","Erro Encontrado",MB_OK);
                AfxThrowUserException();
                }
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate ou atributo não está declarado","Erro
Encontrado",MB_OK);
               AfxThrowUserException();
               }
              }
             }
            else
             {
             //atribuição string cte
             //checagem de tipo com LValue é necessária
             if (m_AttribTable.GetAt(m_index[2]).GetAt(0)!="STRING")
              {
              MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[2]).GetAt(1)+"\nAtributo não é do tipo
STRING","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              return;
              }
             //checagem p/ fecha aspas
             if (!Search (RName[0],1,"\",param[2],i,ch,TRUE,FALSE))  //se não tiver fecha aspas
              {
              MessageBox (NULL,"Expressão com erro : "+RName[0]+"\nString constante não termina com \'","Expressão com
erro",MB_OK);
              AfxThrowUserException();
              return;
              }
             //cria a atribuição simples
             myAtbn->Create(EQUAL_TO_STRING_CTE,i,m_index[2],param[2]);
             }
            }
           else
            {
            //endereço deve ser um box!atributo
            if (SeekOrderedTable(m_BoxesTable,1,RName[0],m_index[3]))
             {
             if (SeekOrderedTable(m_AttribTable,1,RName[1],m_index[4]))
              {
              if (m_AttribTable.GetAt(m_index[4]).GetAt(0)=="STRING")
               {
```

```
                myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],m_index[3],m_index[4]);
                }
               else
                {
                MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[4]).GetAt(1)+"\nAtributo não é do tipo
STRING","Erro Encontrado",MB_OK);
                AfxThrowUserException();
                }
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+RName[1]+"\nAtributo não está declarado","Erro
Encontrado",MB_OK);
               AfxThrowUserException();
               }
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nBox não está declarada","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            //adiciona nessa linha de atribuições
            Attributions.Add(*myAtbn);
            //desaloca memória
            delete myAtbn;
            //próxima coluna
            j += 3;
            }
           }
          }
         m_index[1]++;
         if (m_index[1]<m)
          m_word = m_TBThenTable.GetAt(m_index[1]).GetAt(0);
         }

        //adiciona linha de atribuições na matriz THEN
        ThenAttribs.Add(Attributions);

        //bloco ELSE
        Attributions.RemoveAll();

        //encontra 1a atribuição then da box
        SeekOrderedTable(m_TBElseTable,0,param[0],m_index[1]);
        m_word = m_TBElseTable.GetAt(m_index[1]).GetAt(0);

        m = m_TBElseTable.GetSize();

        while ((m_word == param[0])&&(m_index[1]<m))
         {
         m_index[2] = atoi(m_TBElseTable.GetAt(m_index[1]).GetAt(1));
         if (m_index[2]==m_index[0])
          {
          n = m_TBElseTable.GetAt(m_index[1]).GetSize();
          for (j=2;j<n;)
           {
           param[1] = m_TBElseTable.GetAt(m_index[1]).GetAt(j);
           if (param[1]=="TO_NUM")
            {
            //monta atribuição
            CAttribution* myAtbn;
            myAtbn = new CAttribution;
            //atributo
            param[2] = m_TBElseTable.GetAt(m_index[1]).GetAt(j+1);
            SeekOrderedTable(m_AttribTable,1,param[2],m_index[2]);
            //define se é cte numérica, atributo ou gate
            param[2] = m_TBElseTable.GetAt(m_index[1]).GetAt(j+2);
            ParseAddress (param[2],RName[0],RName[1],type);
            if (type == SINGLE)                    //tipo simples -> cte string, cte numérica, box local
             {
             m_index[3] = atoi (RName[0]);
             if ((m_index[3]==0)&&(RName[0]!='0'))          //não é numérico, pois não foi convertido com sucesso
```

```
            {
            //endereço pode estar se referindo à um gate externo de entrada de dados
            if (SeekOrderedTable(m_AttribTable,1,RName[0],m_index[3],FALSE))
             {
             //atributo local
             if (m_AttribTable.GetAt(m_index[3]).GetAt(0)=="INTEGER")
              {
              myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],i,m_index[3]);
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nAtributo não é INTEGER","Erro
Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            else
             {
             //gate externo de entrada de dados
             if (SeekOrderedTable(m_GatesTable,1,RName[0],m_index[3]))
              {
              if (m_GatesTable.GetAt(m_index[3]).GetAt(2)=="INTEGER")
               {
               myAtbn->Create(EQUAL_TO_GATE,i,m_index[2],m_index[3]);
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate não compatível com operação de atribuição
INTEGER","Erro Encontrado",MB_OK);
               AfxThrowUserException();
               }
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate ou atributo não está declarado","Erro
Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            else
             {
             //atribuição numérica cte
             //checagem de tipo com LValue é necessária
             if (m_AttribTable.GetAt(m_index[2]).GetAt(0)!="INTEGER")
              {
              MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[2]).GetAt(1)+"\nAtributo não é do tipo
INTEGER","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              return;
              }
             //cria a atribuição simples
             myAtbn->Create(EQUAL_TO_NUM_CTE,i,m_index[2],m_index[3]);
             }
            }
           else
            {
            //endereço deve ser um box!atributo
            if (SeekOrderedTable(m_BoxesTable,1,RName[0],m_index[3]))
             {
             if (SeekOrderedTable(m_AttribTable,1,RName[1],m_index[4])
              {
              if (m_AttribTable.GetAt(m_index[4]).GetAt(0)=="INTEGER")
               {
               myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],m_index[3],m_index[4]);
               }
              else
               {
               MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[4]).GetAt(1)+"\nAtributo não é do tipo
INTEGER","Erro Encontrado",MB_OK);
               AfxThrowUserException();
               }
```

```
            }
           else
            {
            MessageBox(NULL,"Expressão com erro: "+RName[1]+"\nAtributo não está declarado","Erro
Encontrado",MB_OK);
            AfxThrowUserException();
            }
           }
          else
           {
           MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nBox não está declarada","Erro Encontrado",MB_OK);
           AfxThrowUserException();
           }
          }
         //adiciona nessa linha de atribuições
         Attributions.Add(*myAtbn);
         //desaloca memória
         delete myAtbn;
         //próxima coluna
         j += 3;
         }

        if (param[1]=="TO_STR")
         {
         //monta atribuição
         CAttribution* myAtbn;
         myAtbn = new CAttribution;
         //atributo
         param[2] = m_TBElseTable.GetAt(m_index[1]).GetAt(j+1);
         ParseAddress (param[2],RName[0],RName[1],type);
         if (type == SINGLE)                    //tipo simples -> cte string, cte numérica, atributo local
          {
          ch = RName[0].GetAt(0);
          if (ch!='\")                          //começa com aspas simples -> é string cte
           {
           //endereço pode estar se referindo à um gate externo de entrada de dados
           if (SeekOrderedTable(m_AttribTable,1,RName[0],m_index[3],FALSE))
            {
            //atributo local
            if (m_AttribTable.GetAt(m_index[3]).GetAt(0)=="STRING")
             {
             myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],i,m_index[3]);
             }
            else
             {
             MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nAtributo não é STRING","Erro Encontrado",MB_OK);
             AfxThrowUserException();
             }
            }
           else
            {
            //gate externo de entrada de dados
            if (SeekOrderedTable(m_GatesTable,1,RName[0],m_index[3]))
             {
             if (m_GatesTable.GetAt(m_index[3]).GetAt(2)=="STRING")
              {
              myAtbn->Create(EQUAL_TO_GATE,i,m_index[2],m_index[3]);
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate não compatível com operação de atribuição
STRING","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            else
             {
             MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nGate ou atributo não está declarado","Erro
Encontrado",MB_OK);
             AfxThrowUserException();
             }
            }
```

```
            }
           else
            {
            //atribuição string cte
            //checagem de tipo com LValue é necessária
            if (m_AttribTable.GetAt(m_index[2]).GetAt(0)!="STRING")
             {
             MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[2]).GetAt(1)+"\nAtributo não é do tipo
STRING","Erro Encontrado",MB_OK);
             AfxThrowUserException();
             return;
             }
            //checagem p/ fecha aspas
            if (!Search (RName[0],1,\",param[2],i,ch,TRUE,FALSE))   //se não tiver fecha aspas
             {
             MessageBox (NULL,"Expressão com erro : "+RName[0]+"\nString constante não termina com \"","Expressão com
erro",MB_OK);
             AfxThrowUserException();
             return;
             }
            //cria a atribuição simples
            myAtbn->Create(EQUAL_TO_STRING_CTE,i,m_index[2],param[2]);
            }
           }
          else
           {
           //endereço deve ser um box!atributo
           if (SeekOrderedTable(m_BoxesTable,1,RName[0],m_index[3]))
            {
            if (SeekOrderedTable(m_AttribTable,1,RName[1],m_index[4]))
             {
             if (m_AttribTable.GetAt(m_index[4]).GetAt(0)=="STRING")
              {
              myAtbn->Create(EQUAL_TO_MARKATRIB,i,m_index[2],m_index[3],m_index[4]);
              }
             else
              {
              MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[4]).GetAt(1)+"\nAtributo não é do tipo
STRING","Erro Encontrado",MB_OK);
              AfxThrowUserException();
              }
             }
            else
             {
             MessageBox(NULL,"Expressão com erro: "+RName[1]+"\nAtributo não está declarado","Erro
Encontrado",MB_OK);
             AfxThrowUserException();
             }
            }
           else
            {
            MessageBox(NULL,"Expressão com erro: "+RName[0]+"\nBox não está declarada","Erro Encontrado",MB_OK);
            AfxThrowUserException();
            }
           }
          //adiciona nessa linha de atribuições
          Attributions.Add(*myAtbn);
          //desaloca memória
          delete myAtbn;
          //próxima coluna
          j += 3;
          }
         }
        }
       m_index[1]++;
       if (m_index[1]<m)
        m_word = m_TBElseTable.GetAt(m_index[1]).GetAt(0);
       }

      //adiciona linha de atribuições na matriz ELSE
      ElseAttribs.Add(Attributions);
```

```
      //veja a box do próximo bloco
     m_index[0]++;

      if (m_index[0]<p)
        m_word = m_TBCondTable.GetAt(m_index[0]).GetAt(0);
       }
     //adiciona bloco no box
     myBox->AddBlock(Conditions,ThenAttribs,ElseAttribs);
                        }

                        //adiciona box na lista
                        mp_Owner->GetBoxes()->Add(*myBox);
                        delete myBox;
                        }
}

//////////////////////////////////////////////////////////////////////////////
//
// criação das transições
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::TransitsCreate()
{
 short i;

 CWordsArray line;

 CString param[5];

 CTransition* myTransit;


          //cross-reference para as transits
 ////////////////////////////////
 //                          //
 //        Transits          //
 //     Cross - Reference    //
 //                          //
 ////////////////////////////////
          for(i=0;i<m_nT;i++)
                        {
                        //todos os vetores construídos
                        //podemos criar a transição

                        param[0] = m_TransitsTable.GetAt(i).GetAt(0);

                        if (param[0]=="COMMON")
                                {
                                m_index [0] = COMMON_TRANSITION;
                                //nome
                                param[1] = m_TransitsTable.GetAt(i).GetAt(1);
                                //cria transição
                                myTransit = new CTransition;
                                myTransit->Create(mp_Owner, m_index[0], param[1], mp_nTDs[i]+1, mp_nTOr[i]+1,
mp_TrsDestin[i], mp_TrsOrigin[i], mp_nGts[i]+1, mp_GtsDestin[i]);
                                }

                        if (param[0]=="TEMPORIZED")
                                {
    m_index [0] = TEMPORIZED_TRANSITION;
                                //nome
                                param[1] = m_TransitsTable.GetAt(i).GetAt(1);
    //
    param[2] = m_TransitsTable.GetAt(i).GetAt(2);
    long TimeCte = atol(param[2]);
                                //cria transição
                                myTransit = new CTransition;
                                myTransit->Create(mp_Owner, m_index[0], param[1], mp_nTDs[i]+1, mp_nTOr[i]+1,
mp_TrsDestin[i], mp_TrsOrigin[i], mp_nGts[i], mp_GtsDestin[i]);
    myTransit->SetAsTemporized(TimeCte);
                                }
```

```
                    //adiciona transição na lista
                    mp_Owner->GetTransitions()->Add(*myTransit);
                    delete myTransit;
                    }
}

////////////////////////////////////////////////////////////////////////////
// criação das marcas iniciais
//
////////////////////////////////////////////////////////////////////////////
void CInterpreter::InitMarksCreate()
{
        short i;

 CString param[5];

 CMark* myMark;

        //com o grafo já montado
        //adicionam-se as marcas com os valores iniciais
 ////////////////////////////////
 //                  //
 //    Marcas Iniciais    //
 //   Cross - Reference    //
 //                  //
 ////////////////////////////////

        m_nM = m_MarksTable.GetSize();

        for (i=0;i<m_nM;i++)
                {
                param[0] = m_MarksTable.GetAt(i).GetAt(0);

                SeekOrderedTable(m_BoxesTable,1,param[0],m_index[0]);

                if(!(mp_Owner->GetBoxes()->ElementAt(m_index[0]).IsFull()))//Daniel trocou de HasMark para IsFull
                        {
                        myMark = new CMark;
                        myMark->Create(mp_Owner,mp_Owner->GetAttribTemplates(),0);
                  long nLine = m_MarksTable.GetAt(i).GetSize();
   for (long Mm_index=0;Mm_index<=m_MarksTable.GetAt(i).GetSize()-4;Mm_index=Mm_index+4)
    {
    param[0] = m_MarksTable.GetAt(i).GetAt(Mm_index+1);

                if (param[0]=="TO_STR")
                        {
                        param[1] = m_MarksTable.GetAt(i).GetAt(Mm_index+2);
                        param[2] = m_MarksTable.GetAt(i).GetAt(Mm_index+3);
      SeekOrderedTable(m_AttribTable,1,param[1],m_index[1]);
      if (m_AttribTable.GetAt(m_index[1]).GetAt(0)=="STRING")
       {
                        myMark->SetAtrib(m_index[1],param[2]);
       }
      else
       {
       MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[1]).GetAt(1)+"\nAtributo não é
INTEGER","Erro Encontrado",MB_OK);
       AfxThrowUserException();
       }
                  }

                if (param[0]=="TO_NUM")//Daniel (TO_INT anteriormente)
                        {
                        param[1] = m_MarksTable.GetAt(i).GetAt(Mm_index+2);
                        param[2] = m_MarksTable.GetAt(i).GetAt(Mm_index+3);
                        SeekOrderedTable(m_AttribTable,1,param[1],m_index[1]);
      long value = (long)atol((const char*)param[2].GetBuffer(param[2].GetLength()));
      if (m_AttribTable.GetAt(m_index[1]).GetAt(0)=="INTEGER")
       {
                        myMark->SetAtrib(m_index[1],value);
       }
      else
```

```
        {
        MessageBox(NULL,"Expressão com erro: "+m_AttribTable.GetAt(m_index[1]).GetAt(1)+"\nAtributo não é
INTEGER","Erro Encontrado",MB_OK);
        AfxThrowUserException();
        }
                                }
    }

                    mp_Owner->GetBoxes()->ElementAt(m_index[0]).AddMark(*myMark);
                            delete myMark;
                            }
                    }
}

//////////////////////////////////////////////////////////////////////////////
// criação das atribuições e valores iniciais
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::ConditionalsCreate()
{
 ////////////////////////////////
 //                          //
 //       Condicionais       //
 //     Cross - Reference    //
 //                          //
 ////////////////////////////////

 CConditional* myCond;
 short i;

 //Condicionais dos Gates
 m_nGC = m_GatesCondTable.GetSize();

 for (i=0;i<m_nGC;i++)
  {
  myCond = new CConditional;
  //gate
  m_par[0] = m_GatesCondTable.GetAt(i).GetAt(0);
  //expressão
  m_par[1] = m_GatesCondTable.GetAt(i).GetAt(1);
  //transforma o nome em índice
  SeekOrderedTable(m_GatesTable,1,m_par[0],m_index[0]);
  //cria o condicional
  BuildCond(*myCond,m_par[1],NULO);
  //adiciona o condicional ao gate
  mp_Owner->GetGates()->ElementAt(m_index[0]).AddConditional(myCond);
  delete myCond;
  }

}

//////////////////////////////////////////////////////////////////////////////
//
//
// BuildCond é a função que
// cria um elemento condicional independente dele ser
// SIMPLES ou COMPOSTO
// qualquer erro incorre em jogar exceção
//
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::BuildCond (CConditional& myCond, CString expr, short local)
{
 //parâmetros de construção dos condicionais
 BOOL   not;
 short  LBox, LAtb, OP;
 short  type;
 short  RParam1, RParam2;
 CString RParam3;

 //listas auxiliares
 CCondArray  CondList;
```

```
CCondArray  LevelList;
CIntArray   OPList;

//outras variáveis auxiliares
CString word;
CConditional* Cond;
short i,n,level;
char  ch;


//inicializando variáveis
n = expr.GetLength();
i = 0;
level = 0;
word = "";

//início da interpretação
ch = expr.GetAt(i);

//////////////////////
//                  //
// Condicional SIMPLES //
//                  //
//////////////////////

if (ch!='(')
  {
  //checa p/ tipos especiais
  if (expr=="TRUE")
    {
    myCond.Create(ALWAYS_TRUE,mp_Owner,FALSE,NULO,NULO,NULO,NULO,NULO,NULO,"");
    return;
    }

  if (expr=="FALSE")
    {
    myCond.Create(ALWAYS_FALSE,mp_Owner,FALSE,NULO,NULO,NULO,NULO,NULO,NULO,"");
    return;
    }


  //realiza cross-reference
  CrossRefLOR(expr,not,LBox,LAtb,OP,type,RParam1,RParam2,RParam3,local);

  //inicializa condicional
  switch (type)
    {
    case INT_CTE:
      {
      myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, INT_CTE, RParam1);
      break;
      }
    case STR_CTE:
      {
      myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, STR_CTE, NULO, NULO, RParam3);
      break;
      }
    case INT_BOX_ATB:
      {
      myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, INT_BOX_ATB, RParam1, RParam2);
      break;
      }
    case STR_BOX_ATB:
      {
      myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, STR_BOX_ATB, RParam1, RParam2);
      break;
      }
    case INT_GATE_VAL:
      {
      myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, INT_GATE_VAL, RParam1);
      break;
      }
```

```
    case STR_GATE_VAL:
     {
     myCond.Create(NULO, mp_Owner, not, LBox, LAtb, OP, STR_GATE_VAL, RParam1);
     break;
     }
    };
   return;
   }

//////////////////////////
//                      //
// Condicional COMPOSTO //
//                      //
//////////////////////////

 while (i<n)                  //enquanto houver caracteres
  {
  ch = expr.GetAt(i);
  switch (ch)
   {
   default:                   //adiciona à expressão
    {
    word+=ch;
    break;
    }
   case '(':                  //sobe o nível
    {
    if (level!=0)             //não adiciona o primeiro parênteses
     word+=ch;
    level++;
    break;
    }
   case ')':                  //desce o nível
    {
    level--;
    if(level==0)              //caso seja zero adiciona a lista, chamando recursivamente a própria função
     {
     CConditional *LocalCond;
     LocalCond = new CConditional;
     BuildCond(*LocalCond,word,local);
     CondList.Add(*LocalCond);
     delete LocalCond;
     i++;
     if (i<n)                 //expressões compostas com operadores && ou ||
      {
      ch = expr.GetAt(i);
      switch(ch)
       {
       default:
        {
        MessageBox(NULL,"Expressão com erro : \n"+expr+"Operador não definido","Expressão com Erro",MB_OK);
        AfxThrowUserException();
        break;
        }
       case '&':
        {
        i++;
        ch = expr.GetAt(i);
        if (ch=='&')
         {
         word="";
         OPList.Add(AND);
         }
        else
         {
         MessageBox(NULL,"Expressão com erro : \n"+expr+"Operador não definido","Expressão com Erro",MB_OK);
         AfxThrowUserException();
         }
        break;
        }
       case '|':
        {
```

```
            i++;
            ch = expr.GetAt(i);
            if (ch=='|')
              {
              word="";
              OPList.Add(OR);
              }
            else
              {
              MessageBox(NULL,"Expressão com erro : \n"+expr+"Operador não definido","Expressão com Erro",MB_OK);
              AfxThrowUserException();
              }
            break;
            }
          };
        }
      }
     else
       word+=ch;
     break;
     }
   };
  i++;
  }
 n = OPList.GetSize();
 if (n!=0)
  {
  Cond = new CConditional;
  Cond->Create(OPList.GetAt(0));
  Cond->AddRight(CondList.ElementAt(0));
  Cond->AddDown (CondList.ElementAt(1));
  LevelList.Add(*Cond);
  delete Cond;
  i = 1;
  while (i<n)
   {
   Cond = new CConditional;
   Cond->Create(OPList.GetAt(i));
   Cond->AddDown (LevelList.ElementAt(i-1));
   Cond->AddRight(CondList.ElementAt(i+1));
   LevelList.Add(*(Cond));
   delete Cond;
   i++;
   }

  //retorna o topo da lista de níveis (nivel mais elevado)==(pai de todos)
  myCond = LevelList.ElementAt(i-1);

  //liberando memória antes de retornar
  CondList.RemoveAll();
  LevelList.RemoveAll();
  OPList.RemoveAll();

  return;
  }

 //nunca espero ter que usar essa exceção, algo muito errado pode estar acontecendo
 if (CondList.GetSize()==0)
  {
  MessageBox(NULL,"Expressão com erro : \n"+expr,"Expressão com erro",MB_OK);
  AfxThrowUserException();
  }

 //só topo da lista é levado em conta
 myCond = CondList.ElementAt(0);

 //liberando memória antes de retornar
 CondList.RemoveAll();

 return;
}
```

```
////////////////////////////////////////////////////////////////////////////
//
//
// CrossRefLOR é a função que
// faz a referência cruzada de APENAS UMA expressão condicional
// já executa checagem de tipo onde necessário
// qualquer erro incorre em jogar exceção
//
//
////////////////////////////////////////////////////////////////////////////

void CInterpreter::CrossRefLOR ( CString expr,
                    BOOL &negative,
                    short &LBox, short &LAttribute,
                    short &Operator,

                                short &RType, short &RParam1, short &RParam2, CString &RParam3,
                    short LocalBox)
{
 CString LName1, LName2;
 CString OP;
 CString RName1, RName2;

 char ch;
 short i,n;
 BOOL left;

 short type;

 //incializando
 i = 0;
 n = expr.GetLength();
 ch = expr.GetAt(i);          //primeiro caractere

 left = TRUE;                 //começa com LValue

 LName1 == "";
 LName2 == "";
 RName1 == "";
 RName2 == "";
 OP == "";

 type = NULO;

 //determina se a expressão está negada
 if (ch=='~')
  {
  negative = TRUE;
  i+=2;
  ch = expr.GetAt(i);
  }
 else
  {
  negative = FALSE;
  }

 //dividindo em LValue, Operador e, RValue
 while (i<n)
  {
  switch(ch)
   {
   default:
    {
    if (left)
     LName1+=ch;              //quando for LValue
    else
     RName1+=ch;              //quando for RValue
    break;
    }
   case '!':                  //operador !=
    {
```

```
      i++;
      ch = expr.GetAt(i);
      if (ch=='=')              //! faz parte do operador !=
       {
       OP = "!=";
       left = FALSE;
       }
      else
       {
       if (left)              //mas quando ! é usado como operador de endereçamento
        {
        LName1+='!';          //prossegue-se adicionando ao LValue
        LName1+=ch;
        }
       else
        {
        RName1+='!';          //ou ao RValue
        RName1+=ch;
        }
       }
      break;
      }
     case '=':                //operador ==
      {
      OP+=ch;
      i++;
      ch = expr.GetAt(i);
      OP+=ch;
      left = FALSE;
      break;
      }
     case '>':                //operador >
      {
      OP+=ch;
      ch = expr.GetAt(i);
      left = FALSE;
      break;
      }
     case '<':                //operador <
      {
      OP+=ch;
      ch = expr.GetAt(i);
      left = FALSE;
      break;
      }
     };
   i++;
   if (i<n)
    ch = expr.GetAt(i);
   }

////////////////////////////
//                        //
//       LValue           //
//                        //
////////////////////////////

ParseAddress(LName1,LName1,LName2,type);

if (type==SINGLE)                        // endereço é local (caso do box transformator)
 {
 if (LocalBox==NULO)                     // tem que ser especificado no momento da chamada dessa função
  {
  MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não pode ser local","Erro Encontrado",MB_OK);
  AfxThrowUserException();
  return;
  }
 else
  {
  LBox = LocalBox;
  if (!SeekOrderedTable(m_AttribTable,1,LName1,LAttribute))  // caso não atributo não seja encontrado
   {
```

```
    MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue especifica atributo local inválido","Erro Encontrado",MB_OK);
    AfxThrowUserException();
    return;
    }
   }
  }
 else                                  // endereço não é local (caso genérico)
  {
  if (!SeekOrderedTable(m_BoxesTable,1,LName1,LBox))        //caso box não seja encontrado
   {
   MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue especifica box inválido","Erro Encontrado",MB_OK);
   AfxThrowUserException();
   return;
   }
  if (!SeekOrderedTable(m_AttribTable,1,LName2,LAttribute))  //caso atributo não seja encontrado
   {
   MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue especifica atributo inválido","Erro Encontrado",MB_OK);
   AfxThrowUserException();
   return;
   }
  }

//////////////////////////
//                      //
//      Operador        //
//                      //
//////////////////////////

if (OP=="==")
  Operator = EQUAL;
else
  {
  if (OP=="!=")
   {
   Operator = EQUAL;
   negative = !negative;
   }
  else
   {
   if (OP==">")
    {
    Operator = GREATER;
    }
   else
    {
    if (OP=="<")
     {
     Operator = LESSER;
     }
    else
     {
     MessageBox (NULL,"Operador não suportado : "+OP,"Operador não suportado",MB_OK);
     AfxThrowUserException();
     return;
     }
    }
   }
  }

//////////////////////////
//                      //
//       RValue         //
//                      //
//////////////////////////

ParseAddress (RName1,RName1,RName2,type);

if (type == SINGLE)                   //tipo simples -> cte string, cte numérica, box local
 {
 RParam1 = atoi (RName1);
 if ((RParam1==0)&&(RName1!='0'))          //não é numérico, pois não foi convertido com sucesso
  {
```

```
i = 0;
n = RName1.GetLength();
ch = RName1.GetAt(i);
if (ch=='\'')                          //começa com aspas simples -> é string cte
 {
 RType = STR_CTE;
 RParam1 = NULO;
 RParam2 = NULO;
 if (!Search (RName1,1,'\'',RParam3,i,ch,TRUE,FALSE))   //se não tiver fecha aspas
  {
  MessageBox (NULL,"Expressão com erro : "+expr+"\nString constante não termina com \'","Expressão com erro",MB_OK);
  AfxThrowUserException();
  return;
  }
 else
  {
  //checagem de tipo com LValue é necessária
  if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="STRING")
   {
   MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo STRING","Erro Encontrado",MB_OK);
   AfxThrowUserException();
   return;
   }
  //checagem de tipo do operador
  if ((OP!="==")&&(OP!="!="))
   {
   MessageBox(NULL,"Expressão com erro: "+expr+"\nOperador não é definido para STRING","Erro Encontrado",MB_OK);
   AfxThrowUserException();
   return;
   }
  }
 }
else
 {
 //endereço pode estar se referindo à um gate
 if (SeekOrderedTable(m_GatesTable,1,RName1,RParam1))
  {
  if (m_GatesTable.GetAt(RParam1).GetAt(2)=="INTEGER")
   {
   RType = INT_GATE_VAL;
   //checagem de tipo com LValue é necessária
   if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="INTEGER")
    {
    MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo INTEGER","Erro Encontrado",MB_OK);
    AfxThrowUserException();
    return;
    }
   }
  else
   {
   RType = STR_GATE_VAL;
   //checagem de tipo com LValue é necessária
   if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="STRING")
    {
    MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo STRING","Erro Encontrado",MB_OK);
    AfxThrowUserException();
    return;
    }
   //checagem do operador
   if ((OP!="==")&&(OP!="!="))
    {
    MessageBox(NULL,"Expressão com erro: "+expr+"\nOperador não é definido para STRING","Erro
Encontrado",MB_OK);
    AfxThrowUserException();
    return;
    }
   }
  }
 else
  {
  //endereço é local (caso especial do box transformator)
  if (LocalBox==NULO)                  // tem que ser especificado no momento da chamada dessa função
```

```
        {
        MessageBox(NULL,"Expressão com erro: "+expr+"\nRValue não pode ser local","Erro Encontrado",MB_OK);
        AfxThrowUserException();
        return;
        }
       else
        {
        RParam1 = LocalBox;
        if (!SeekOrderedTable(m_AttribTable,1,RName1,RParam2))  // caso não atributo não seja encontrado
         {
         MessageBox(NULL,"Expressão com erro: "+expr+"\nRValue especifica atributo local inválido","Erro
Encontrado",MB_OK);
         AfxThrowUserException();
         return;
         }
        if (m_AttribTable.GetAt(RParam2).GetAt(0)!="STRING")
         {
         RType = STR_BOX_ATB;
         //checagem de tipo com LValue é necessária;
         if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="STRING")
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo STRING","Erro Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
         //checagem do operador
         if ((OP!="==")&&(OP!="!="))
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nOperador não é definido para STRING","Erro
Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
         }
        else
         {
         RType = INT_BOX_ATB;
         //checagem de tipo com LValue é necessária;
         if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="INTEGER")
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo INTEGER","Erro Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
         }
        }
       }
      }
     }
    else
     {
     //Rname1 convertida com sucesso
     RType = INT_CTE;
     RParam1 = atoi (RName1);
     //checagem de tipo com LValue é necessária
     if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="INTEGER")
      {
      MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo INTEGER","Erro Encontrado",MB_OK);
      AfxThrowUserException();
      return;
      }
     }
    }
   else     //RValue é do tipo DOUBLE (box!atributo)
    {
    if (SeekOrderedTable(m_BoxesTable,1,RName1,RParam1))
     {
     if (SeekOrderedTable(m_AttribTable,1,RName2,RParam2))
      {
      if (m_AttribTable.GetAt(RParam2).GetAt(0)=="INTEGER")
       {
       RType = INT_BOX_ATB;
```

```
        //checagem de tipo com LValue é necessária
        if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="INTEGER")
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo INTEGER","Erro Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
        }
       else
        {
        RType = STR_BOX_ATB;
        //checagem de tipo com LValue é necessária
        if (m_AttribTable.GetAt(LAttribute).GetAt(0)!="STRING")
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nLValue não é do tipo STRING","Erro Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
        //checagem do operador
        if ((OP!="==")&&(OP!="!="))
          {
          MessageBox(NULL,"Expressão com erro: "+expr+"\nOperador não é definido para STRING","Erro Encontrado",MB_OK);
          AfxThrowUserException();
          return;
          }
        }
       }
      else
       {
       MessageBox(NULL,"Atributo do RValue não foi encontrado :\n"+expr,"Atributo não econtrado",MB_OK);
       AfxThrowUserException();
       return;
       }
      }
     else
      {
      MessageBox(NULL,"Box do RValue não foi encontrado :\n"+expr,"Box não econtrado",MB_OK);
      AfxThrowUserException();
      return;
      }
     }
}

//////////////////////////////////////////////////////////////////////////////
//
// MatchingBrackets() resolve problemas de nível nos condicionais
//
//////////////////////////////////////////////////////////////////////////////
BOOL CInterpreter::MatchingBrackets(CString& expr)
{
 short i,n;
 short level;
 char  ch;

 //incializa
 n = expr.GetLength();
 i = 0;
 level = 0;
 ch = expr.GetAt(i);

 //teste de match
 while (i<n)
  {
  ch = expr.GetAt(i);
  if (ch=='(')
    level++;
  if (ch==')')
    level--;
  i++;
  }

//retorna condição
```

```
  if (level!=0)
    return FALSE;
  else
    return TRUE;
}
////////////////////////////////////////////////////////////////////////////
//
// Next() avança uma linha na tabela
//
////////////////////////////////////////////////////////////////////////////

BOOL CInterpreter::Next(CString &Line)
{
        if (m_cursor>=m_length)
                {
                MessageBox(NULL,Line+"\nFinal do arquivo encontrado inesperadamente.","Erro na Construção das
Tabelas",MB_OK);
                AfxThrowUserException();
                return FALSE;
                }

        Line = m_FileTable[m_cursor];

        m_cursor ++;

        return TRUE;
}


////////////////////////////////////////////////////////////////////////////
//
// Search() busca por um caracter numa linha da tabela
//
////////////////////////////////////////////////////////////////////////////
BOOL CInterpreter::Search(CString Line, short start, char find, CString &Word, short &stop, char &found, BOOL trim, BOOL
warn)
{
        short i;
        short length;
        BOOL  success;

        char  pos;

        length = Line.GetLength();

        success = FALSE;
        i = start;

        Word  = "";
        found = ' ';

        while (i!=length)
                {
                pos = Line.GetAt(i);

                if ((pos!=' ')||(!trim))
                        {
                        if (pos==find)
                                {
                                stop  = i+1;
                                found = find;
                                return (TRUE);
                                }
                        else
                                {
                                if ((pos=='(')||(pos==')')||(pos=='<')||(pos=='>')||(pos==',')||(pos==';')||(pos==':'))
                                        {
                                   stop  = i+1;
                                        found = pos;
                                        break;
                                        }
                                else
```

```
                                        Word += pos;
                                    }
                                }
                        i++;
                        }

  if (!warn)
   {
            //special case for end of variable-parameter functions
            if ((find==',')&&(found==')'))
                    return FALSE;

   //special case for exclamation mark in box!attribute case
   if (find=='!')
    return FALSE;

   return (FALSE);
   }
  else
   {
   CString message;
   message = "Caracter ' ";
   message += find;
   message += " ' não foi encontrado\nna expressão : "+Line;
   MessageBox(NULL,message,"Erro de busca",MB_OK);
   AfxThrowUserException();
   }

  return (FALSE);
}


//////////////////////////////////////////////////////////////////////////////
//
// ParseAddres() divide um endereço em uma ou duas partes (SINGLE ou DOUBLE)
//
//////////////////////////////////////////////////////////////////////////////
void CInterpreter::ParseAddress(CString Address, CString& First, CString& Second, short& Type)
{
 short i,n;

 char ch;

 //inicializando valores
 i = 0;
 n = Address.GetLength();

 //definindo tipo de endereço
 if (Search (Address,0,'!',First,i,ch,TRUE,FALSE))
  {
  Type = DOUBLE;
  Second = Address.Right(n-i);
  }
 else
  {
  Type  = SINGLE;
  First  = Address;
  Second = "";
  }
}

//////////////////////////////////////////////////////////////////////////////
//
// SeekOrderedTable() busca binariamente por uma label numa coluna de tabela ordenada
//
//////////////////////////////////////////////////////////////////////////////
BOOL CInterpreter::SeekOrderedTable (CLinesArray &Table, short col, CString word, short &index, BOOL warn)
{
        short low, mid, high;
  short old;
```

```
            CString cell;

    low  = 0;
            high = Table.GetSize();

    mid  = high/2;

    do
    {
    cell = Table.GetAt(mid).GetAt(col);

    old = mid;

    if (cell > word)
     {
     high = mid;
     mid  = low + (high-low)/2;
     }

    if (cell < word)
     {
     low = mid;
     mid  = low + (high-low)/2;
     }

   if (cell == word)
     {
     index = mid;
     return TRUE;
     }

    if (old == mid)
     {
     if (warn)
      {
      MessageBox(NULL,"Não foi possível encontrar a label\n"+word,"Erro de Busca",MB_OK);
      AfxThrowUserException();
      }
     return FALSE;
     }
    }while (TRUE);

   return FALSE;
}

//////////////////////////////////////////////////////////////////////////////
//
// SeekTable() busca por uma label numa coluna de tabela
//
//////////////////////////////////////////////////////////////////////////////
BOOL CInterpreter::SeekTable (CLinesArray &Table, short col, CString word, short &index)
{
            short i;
            short length;

            CString cell;

            length = Table.GetSize();

            i = 0;

            index = 0;

            while (i<length)
                        {
                        cell = Table.GetAt(i).GetAt(col);
                        if (word == cell)
                                    {
                                    index = i;
                                    return TRUE;
                                    }
                        i++;
```

```
                }
        return FALSE;
}


//////////////////////////////////////////////////////////////////////////
//
// Duplicate() checa por duplicação dentro de uma mesma tabela
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::Duplicate (CLinesArray &Source)
{
        short i, j, n;

        CString word, comp;

        n = Source.GetSize();

        for(i=0;i<n;i++)
                {
                word = Source.GetAt(i).GetAt(1);
                for (j=i+1;j<n;j++)
                        {
                        comp = Source.GetAt(j).GetAt(1);
                        if (word==comp)
                                {
                                MessageBox(NULL,word+"\nLabel duplicada.\nCheque nomes dos elementos.","Erro
de Referência Cruzada",MB_OK);
                                AfxThrowUserException();
                                }
                        }
                }
}

//////////////////////////////////////////////////////////////////////////
//
// CrossDuplicate() checa por duplicação dentro de duas tabelas
//
//////////////////////////////////////////////////////////////////////////
void CInterpreter::CrossDuplicate (CLinesArray &Source, CLinesArray &Target)
{
        short i, j, n, m;

        CString word, comp;

        n = Source.GetSize();
 m = Target.GetSize();


        for(i=0;i<n;i++)
                {
                word = Source.GetAt(i).GetAt(1);
                for (j=0;j<m;j++)
                        {
                        comp = Target.GetAt(j).GetAt(1);
                        if (word==comp)
                                {
                                MessageBox(NULL,word+"\nLabel Duplicada.\bCheque nomes dos
elementos.","Erro de Referência Cruzada",MB_OK);
                                AfxThrowUserException();
                                }
                        }
                }
}
```

F - Aplicação

Files Etna.mak - E-MFG Controller Application Makefile

```
# Microsoft Developer Studio Generated NMAKE File, Format Version 40001
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Application" 0x0101

!IF "$(CFG)" == ""
CFG=Etna - Win32 Debug
!MESSAGE No configuration specified.  Defaulting to Etna - Win32 Debug.
!ENDIF

!IF "$(CFG)" != "Etna - Win32 Release" && "$(CFG)" != "Etna - Win32 Debug"
!MESSAGE Invalid configuration "$(CFG)" specified.
!MESSAGE You can specify a configuration when running NMAKE on this makefile
!MESSAGE by defining the macro CFG on the command line.  For example:
!MESSAGE
!MESSAGE NMAKE /f "Etna.mak" CFG="Etna - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Etna - Win32 Release" (based on "Win32 (x86) Application")
!MESSAGE "Etna - Win32 Debug" (based on "Win32 (x86) Application")
!MESSAGE
!ERROR An invalid configuration is specified.
!ENDIF

!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
################################################################################
# Begin Project
# PROP Target_Last_Scanned "Etna - Win32 Debug"
CPP=cl.exe
RSC=rc.exe
MTL=mktyplib.exe

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

# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 5
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
OUTDIR=.\Release
INTDIR=.\Release

ALL : "$(OUTDIR)\Etna.exe"

CLEAN :
	-@erase ".\Release\Etna.exe"
	-@erase ".\Release\font.obj"
	-@erase ".\Release\Etna.pch"
	-@erase ".\Release\ControleDlg.obj"
	-@erase ".\Release\Transit.obj"
	-@erase ".\Release\comm.obj"
	-@erase ".\Release\Interp.obj"
	-@erase ".\Release\manager.obj"
	-@erase ".\Release\arcfilt.obj"
	-@erase ".\Release\ddesock.obj"
	-@erase ".\Release\Arcs.obj"
```

```
	-@erase ".\Release\EtnaDoc.obj"
	-@erase ".\Release\Gates.obj"
	-@erase ".\Release\basis.obj"
	-@erase ".\Release\list.obj"
	-@erase ".\Release\MainFrm.obj"
	-@erase ".\Release\StdAfx.obj"
	-@erase ".\Release\gridctrl.obj"
	-@erase ".\Release\graph.obj"
	-@erase ".\Release\arrays.obj"
	-@erase ".\Release\boxes.obj"
	-@erase ".\Release\marks.obj"
	-@erase ".\Release\picture.obj"
	-@erase ".\Release\EtnaView.obj"
	-@erase ".\Release\wordsarray.obj"
	-@erase ".\Release\DDEOptions.obj"
	-@erase ".\Release\Etna.obj"
	-@erase ".\Release\status.obj"
	-@erase ".\Release\Etna.res"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_AFXDLL" /D "_MBCS"
/Yu"stdafx.h" /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS" /Yu"stdafx.h" /c
CPP_PROJ=/nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_MBCS" /Fp"$(INTDIR)/Etna.pch" /Yu"stdafx.h" /Fo"$(INTDIR)/" /c
CPP_OBJS=.\Release/
CPP_SBRS=
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ=/nologo /D "NDEBUG" /win32
# ADD BASE RSC /l 0x416 /d "NDEBUG" /d "_AFXDLL"
# ADD RSC /l 0x416 /d "NDEBUG"
RSC_PROJ=/l 0x416 /fo"$(INTDIR)/Etna.res" /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/Etna.bsc"
BSC32_SBRS=
LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /machine:I386
# ADD LINK32 /nologo /subsystem:windows /machine:I386
LINK32_FLAGS=/nologo /subsystem:windows /incremental:no\
 /pdb:"$(OUTDIR)/Etna.pdb" /machine:I386 /out:"$(OUTDIR)/Etna.exe"
LINK32_OBJS= \
	"$(INTDIR)/font.obj" \
	"$(INTDIR)/ControleDlg.obj" \
	"$(INTDIR)/Transit.obj" \
	"$(INTDIR)/comm.obj" \
	"$(INTDIR)/Interp.obj" \
	"$(INTDIR)/manager.obj" \
	"$(INTDIR)/arcfilt.obj" \
	"$(INTDIR)/ddesock.obj" \
	"$(INTDIR)/Arcs.obj" \
	"$(INTDIR)/EtnaDoc.obj" \
	"$(INTDIR)/Gates.obj" \
	"$(INTDIR)/basis.obj" \
	"$(INTDIR)/list.obj" \
	"$(INTDIR)/MainFrm.obj" \
	"$(INTDIR)/StdAfx.obj" \
	"$(INTDIR)/gridctrl.obj" \
	"$(INTDIR)/graph.obj" \
	"$(INTDIR)/arrays.obj" \
	"$(INTDIR)/boxes.obj" \
	"$(INTDIR)/marks.obj" \
	"$(INTDIR)/picture.obj" \
	"$(INTDIR)/EtnaView.obj" \
	"$(INTDIR)/wordsarray.obj" \
	"$(INTDIR)/DDEOptions.obj" \
	"$(INTDIR)/Etna.obj" \
	"$(INTDIR)/status.obj" \
```

```
            "$(INTDIR)/Etna.res"

"$(OUTDIR)\Etna.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

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

# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 6
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
OUTDIR=.\Debug
INTDIR=.\Debug

ALL : "$(OUTDIR)\Etna.exe"

CLEAN :
            -@erase ".\Debug\vc40.pdb"
            -@erase ".\Debug\Etna.pch"
            -@erase ".\Debug\vc40.idb"
            -@erase ".\Debug\Etna.exe"
            -@erase ".\Debug\list.obj"
            -@erase ".\Debug\graph.obj"
            -@erase ".\Debug\picture.obj"
            -@erase ".\Debug\boxes.obj"
            -@erase ".\Debug\marks.obj"
            -@erase ".\Debug\status.obj"
            -@erase ".\Debug\wordsarray.obj"
            -@erase ".\Debug\DDEOptions.obj"
            -@erase ".\Debug\gridctrl.obj"
            -@erase ".\Debug\ControleDlg.obj"
            -@erase ".\Debug\font.obj"
            -@erase ".\Debug\manager.obj"
            -@erase ".\Debug\EtnaView.obj"
            -@erase ".\Debug\arcfilt.obj"
            -@erase ".\Debug\ddesock.obj"
            -@erase ".\Debug\comm.obj"
            -@erase ".\Debug\StdAfx.obj"
            -@erase ".\Debug\EtnaDoc.obj"
            -@erase ".\Debug\Arcs.obj"
            -@erase ".\Debug\MainFrm.obj"
            -@erase ".\Debug\arrays.obj"
            -@erase ".\Debug\Interp.obj"
            -@erase ".\Debug\Etna.obj"
            -@erase ".\Debug\Transit.obj"
            -@erase ".\Debug\Gates.obj"
            -@erase ".\Debug\basis.obj"
            -@erase ".\Debug\Etna.res"
            -@erase ".\Debug\Etna.ilk"
            -@erase ".\Debug\Etna.pdb"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_AFXDLL" /D
"_MBCS" /Yu"stdafx.h" /c
# ADD CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_AFXDLL" /D "_MBCS"
/Yu"stdafx.h" /c
CPP_PROJ=/nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_AFXDLL" /D "_MBCS" /Fp"$(INTDIR)/Etna.pch" /Yu"stdafx.h" /Fo"$(INTDIR)/"\
 /Fd"$(INTDIR)/" /c
CPP_OBJS=.\Debug/
CPP_SBRS=
# ADD BASE MTL /nologo /D "_DEBUG" /win32
```

```
# ADD MTL /nologo /D "_DEBUG" /win32
MTL_PROJ=/nologo /D "_DEBUG" /win32
# ADD BASE RSC /l 0x416 /d "_DEBUG" /d "_AFXDLL"
# ADD RSC /l 0x416 /d "_DEBUG" /d "_AFXDLL"
RSC_PROJ=/l 0x416 /fo"$(INTDIR)/Etna.res" /d "_DEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/Etna.bsc"
BSC32_SBRS=
LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /debug /machine:I386
# ADD LINK32 /nologo /subsystem:windows /debug /machine:I386
LINK32_FLAGS=/nologo /subsystem:windows /incremental:yes\
 /pdb:"$(OUTDIR)/Etna.pdb" /debug /machine:I386 /out:"$(OUTDIR)/Etna.exe"
LINK32_OBJS= \
        "$(INTDIR)/list.obj" \
        "$(INTDIR)/graph.obj" \
        "$(INTDIR)/picture.obj" \
        "$(INTDIR)/boxes.obj" \
        "$(INTDIR)/marks.obj" \
        "$(INTDIR)/status.obj" \
        "$(INTDIR)/wordsarray.obj" \
        "$(INTDIR)/DDEOptions.obj" \
        "$(INTDIR)/gridctrl.obj" \
        "$(INTDIR)/ControleDlg.obj" \
        "$(INTDIR)/font.obj" \
        "$(INTDIR)/manager.obj" \
        "$(INTDIR)/EtnaView.obj" \
        "$(INTDIR)/arcfilt.obj" \
        "$(INTDIR)/ddesock.obj" \
        "$(INTDIR)/comm.obj" \
        "$(INTDIR)/StdAfx.obj" \
        "$(INTDIR)/EtnaDoc.obj" \
        "$(INTDIR)/Arcs.obj" \
        "$(INTDIR)/MainFrm.obj" \
        "$(INTDIR)/arrays.obj" \
        "$(INTDIR)/Interp.obj" \
        "$(INTDIR)/Etna.obj" \
        "$(INTDIR)/Transit.obj" \
        "$(INTDIR)/Gates.obj" \
        "$(INTDIR)/basis.obj" \
        "$(INTDIR)/Etna.res"

"$(OUTDIR)\Etna.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ENDIF

.c{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.c{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

################################################################################
# Begin Target
```

```
# Name "Etna - Win32 Release"
# Name "Etna - Win32 Debug"

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

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

!ENDIF

################################################################################
# Begin Source File

SOURCE=.\ReadMe.txt

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

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

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Etna.cpp
DEP_CPP_ETNA_=\
        ".\StdAfx.h"\
        ".\Etna.h"\
        ".\elements.h"\
        ".\Interp.h"\
        ".\ddesock.h"\
        ".\comm.h"\
        ".\manager.h"\
        ".\MainFrm.h"\
        ".\EtnaDoc.h"\
        ".\ControleDlg.h"\
        ".\status.h"\
        ".\EtnaView.h"\
        ".\Definitions.h"\
        ".\wordsarray.h"\
        ".\Basis.h"\
        ".\graph.h"\
        ".\Marks.h"\
        ".\arcfilt.h"\
        ".\Arcs.h"\
        ".\Boxes.h"\
        ".\Transit.h"\
        ".\Gates.h"\
        ".\list.h"\
        ".\arrays.h"\
        ".\gridctrl.h"\


"$(INTDIR)\Etna.obj" : $(SOURCE) $(DEP_CPP_ETNA_) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\StdAfx.cpp
DEP_CPP_STDAF=\
        ".\StdAfx.h"\


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

# ADD CPP /Yc"stdafx.h"

BuildCmds= \
        $(CPP) /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"\
```

```
 /Fp"$(INTDIR)/Etna.pch" /Yc"stdafx.h" /Fo"$(INTDIR)/" /c $(SOURCE) \


"$(INTDIR)\StdAfx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

"$(INTDIR)\Etna.pch" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

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

# ADD CPP /Yc"stdafx.h"

BuildCmds= \
            $(CPP) /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_AFXDLL" /D "_MBCS" /Fp"$(INTDIR)/Etna.pch" /Yc"stdafx.h" /Fo"$(INTDIR)/"\
 /Fd"$(INTDIR)/" /c $(SOURCE) \


"$(INTDIR)\StdAfx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

"$(INTDIR)\Etna.pch" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\MainFrm.cpp
DEP_CPP_MAINF=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\elements.h"\
            ".\status.h"\
            ".\ControleDlg.h"\
            ".\EtnaView.h"\
            ".\MainFrm.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
            ".\gridctrl.h"\


"$(INTDIR)\MainFrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\EtnaDoc.cpp
DEP_CPP_ETNAD=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\elements.h"\
            ".\Interp.h"\
            ".\ddesock.h"\
            ".\comm.h"\
            ".\manager.h"\
```

```
            ".\EtnaDoc.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\EtnaDoc.obj" : $(SOURCE) $(DEP_CPP_ETNAD) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Etna.rc
DEP_RSC_ETNA_R=\
            ".\res\EtnaDoc.ico"\
            ".\res\icon1.ico"\
            ".\res\wait1.ico"\
            ".\res\go1.ico"\
            ".\res\Toolbar.bmp"\
            ".\res\bitmap1.bmp"\
            ".\res\Etna.ico"\
            ".\res\Etna.rc2"\


"$(INTDIR)\Etna.res" : $(SOURCE) $(DEP_RSC_ETNA_R) "$(INTDIR)"
  $(RSC) $(RSC_PROJ) $(SOURCE)


# End Source File
################################################################################
# Begin Source File

SOURCE=.\EtnaView.cpp

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

DEP_CPP_ETNAV=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\elements.h"\
            ".\Interp.h"\
            ".\ddesock.h"\
            ".\comm.h"\
            ".\manager.h"\
            ".\status.h"\
            ".\DDEOptions.h"\
            ".\ControleDlg.h"\
            ".\EtnaView.h"\
            ".\EtnaDoc.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
```

```
            ".\gridctrl.h"\


"$(INTDIR)\EtnaView.obj" : $(SOURCE) $(DEP_CPP_ETNAV) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


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

DEP_CPP_ETNAV=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\status.h"\
            ".\elements.h"\
            ".\Interp.h"\
            ".\ddesock.h"\
            ".\comm.h"\
            ".\manager.h"\
            ".\DDEOptions.h"\
            ".\ControleDlg.h"\
            ".\EtnaView.h"\
            ".\EtnaDoc.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
            ".\gridctrl.h"\


"$(INTDIR)\EtnaView.obj" : $(SOURCE) $(DEP_CPP_ETNAV) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\wordsarray.cpp
DEP_CPP_WORDS=\
            ".\StdAfx.h"\
            ".\wordsarray.h"\


"$(INTDIR)\wordsarray.obj" : $(SOURCE) $(DEP_CPP_WORDS) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\list.cpp
DEP_CPP_LIST_=\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
```

```
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\list.obj" : $(SOURCE) $(DEP_CPP_LIST_) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\basis.cpp
DEP_CPP_BASIS=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\basis.obj" : $(SOURCE) $(DEP_CPP_BASIS) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Arcs.cpp
DEP_CPP_ARCS_=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\Arcs.obj" : $(SOURCE) $(DEP_CPP_ARCS_) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\boxes.cpp
DEP_CPP_BOXES=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
```

```
                ".\graph.h"\
                ".\Marks.h"\
                ".\arcfilt.h"\
                ".\Arcs.h"\
                ".\Boxes.h"\
                ".\Transit.h"\
                ".\Gates.h"\
                ".\list.h"\
                ".\arrays.h"\


"$(INTDIR)\boxes.obj" : $(SOURCE) $(DEP_CPP_BOXES) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Gates.cpp
DEP_CPP_GATES=\
                ".\StdAfx.h"\
                ".\elements.h"\
                ".\Definitions.h"\
                ".\wordsarray.h"\
                ".\Basis.h"\
                ".\graph.h"\
                ".\Marks.h"\
                ".\arcfilt.h"\
                ".\Arcs.h"\
                ".\Boxes.h"\
                ".\Transit.h"\
                ".\Gates.h"\
                ".\list.h"\
                ".\arrays.h"\


"$(INTDIR)\Gates.obj" : $(SOURCE) $(DEP_CPP_GATES) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\marks.cpp
DEP_CPP_MARKS=\
                ".\StdAfx.h"\
                ".\elements.h"\
                ".\Definitions.h"\
                ".\wordsarray.h"\
                ".\Basis.h"\
                ".\graph.h"\
                ".\Marks.h"\
                ".\arcfilt.h"\
                ".\Arcs.h"\
                ".\Boxes.h"\
                ".\Transit.h"\
                ".\Gates.h"\
                ".\list.h"\
                ".\arrays.h"\


"$(INTDIR)\marks.obj" : $(SOURCE) $(DEP_CPP_MARKS) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Transit.cpp
```

```
DEP_CPP_TRANS=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\Transit.obj" : $(SOURCE) $(DEP_CPP_TRANS) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\status.cpp
DEP_CPP_STATU=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\status.h"\


"$(INTDIR)\status.obj" : $(SOURCE) $(DEP_CPP_STATU) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\Interp.cpp
DEP_CPP_INTER=\
            ".\StdAfx.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
            ".\Interp.h"\
            ".\arcfilt.h"\


"$(INTDIR)\Interp.obj" : $(SOURCE) $(DEP_CPP_INTER) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\graph.cpp
DEP_CPP_GRAPH=\
            ".\StdAfx.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
```

```
            ".\graph.h"\
            ".\Marks.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
            ".\arcfilt.h"\


"$(INTDIR)\graph.obj" : $(SOURCE) $(DEP_CPP_GRAPH) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\font.cpp
DEP_CPP_FONT_=\
            ".\StdAfx.h"\
            ".\font.h"\


"$(INTDIR)\font.obj" : $(SOURCE) $(DEP_CPP_FONT_) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\gridctrl.cpp
DEP_CPP_GRIDC=\
            ".\StdAfx.h"\
            ".\gridctrl.h"\
            ".\picture.h"\
            ".\font.h"\


"$(INTDIR)\gridctrl.obj" : $(SOURCE) $(DEP_CPP_GRIDC) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\picture.cpp
DEP_CPP_PICTU=\
            ".\StdAfx.h"\
            ".\picture.h"\


"$(INTDIR)\picture.obj" : $(SOURCE) $(DEP_CPP_PICTU) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\arrays.cpp
DEP_CPP_ARRAY=\
            ".\StdAfx.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\Arcs.h"\
```

```
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\
            ".\arcfilt.h"\


"$(INTDIR)\arrays.obj" : $(SOURCE) $(DEP_CPP_ARRAY) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\comm.cpp
DEP_CPP_COMM_=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\ddesock.h"\
            ".\comm.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\comm.obj" : $(SOURCE) $(DEP_CPP_COMM_) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\ddesock.cpp
DEP_CPP_DDESO=\
            ".\StdAfx.h"\
            ".\ddesock.h"\
            ".\elements.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\ddesock.obj" : $(SOURCE) $(DEP_CPP_DDESO) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\arcfilt.cpp
```

```
DEP_CPP_ARCFI=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\arcfilt.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\arcfilt.obj" : $(SOURCE) $(DEP_CPP_ARCFI) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\manager.cpp
DEP_CPP_MANAG=\
            ".\StdAfx.h"\
            ".\elements.h"\
            ".\manager.h"\
            ".\Definitions.h"\
            ".\wordsarray.h"\
            ".\Basis.h"\
            ".\graph.h"\
            ".\Marks.h"\
            ".\arcfilt.h"\
            ".\Arcs.h"\
            ".\Boxes.h"\
            ".\Transit.h"\
            ".\Gates.h"\
            ".\list.h"\
            ".\arrays.h"\


"$(INTDIR)\manager.obj" : $(SOURCE) $(DEP_CPP_MANAG) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\DDEOptions.cpp
DEP_CPP_DDEOP=\
            ".\StdAfx.h"\
            ".\Etna.h"\
            ".\ddesock.h"\
            ".\DDEOptions.h"\


"$(INTDIR)\DDEOptions.obj" : $(SOURCE) $(DEP_CPP_DDEOP) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
################################################################################
# Begin Source File

SOURCE=.\ControleDlg.cpp
DEP_CPP_CONTR=\
            ".\StdAfx.h"\
            ".\Etna.h"\
```

```
            ".\Definitions.h"\
            ".\ControleDlg.h"\


"$(INTDIR)\ControleDlg.obj" : $(SOURCE) $(DEP_CPP_CONTR) "$(INTDIR)"\
 "$(INTDIR)\Etna.pch"


# End Source File
# End Target
# End Project
################################################################################
```

Files Etna.h and Etna.cpp - E-MFG Controller Application

```
// Etna.h : main header file for the ETNA application
//

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CEtnaApp:
// See Etna.cpp for the implementation of this class
//

class CEtnaApp : public CWinApp
{
public:
        CEtnaApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CEtnaApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CEtnaApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
// Etna.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"

#include <ddeml.h>

#include "Etna.h"

#include "elements.h"       //E-MFG Standard Elements
#include "interp.h"                                         //E-MFG Interpreter Definition
#include "ddesock.h"                                    //DDE Socket Definition
#include "comm.h"                                               //E-MFG Communicator Definition
#include "manager.h"        //E-MFG Mark Manager Definition


#include "MainFrm.h"
#include "EtnaDoc.h"
```

```
#include "controledlg.h"     //Control Dialog
#include "status.h"          //Interpreter Status
#include "EtnaView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEtnaApp

BEGIN_MESSAGE_MAP(CEtnaApp, CWinApp)
        //{{AFX_MSG_MAP(CEtnaApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEtnaApp construction

CEtnaApp::CEtnaApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CEtnaApp object

CEtnaApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CEtnaApp initialization

BOOL CEtnaApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();       // Call this when linking to MFC statically
#endif

        LoadStdProfileSettings(2);  // Load standard INI file options (including MRU)

        // Enable OLE Controls
        AfxEnableControlContainer();

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CEtnaDoc),
                RUNTIME_CLASS(CMainFrame),      // main SDI frame window
                RUNTIME_CLASS(CEtnaView));
        AddDocTemplate(pDocTemplate);

        // Enable DDE Execute open
        EnableShellOpen();
```

```
        RegisterShellFileTypes(TRUE);

        // Parse command line for standard shell commands, DDE, file open
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);

        // Dispatch commands specified on the command line
        if (!ProcessShellCommand(cmdInfo))
                return FALSE;

        // Enable drag/drop open
        m_pMainWnd->DragAcceptFiles();

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CEtnaApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
/*
* This is the main DDEML callback proc.  It handles all interaction with
```

```
 * DDEML that is DDEML originated.
 */

HDDEDATA CALLBACK DdeCallback(
WORD wType,
WORD wFmt,
HCONV hConv,
HSZ hszTopic,
HSZ hszItem,
HDDEDATA hData,
DWORD lData1,
DWORD lData2)
{
        switch (wType)
                {
                default:
                        {
                        break;
                        }
  case XTYP_REGISTER:
                        {
                        return ((HDDEDATA)TRUE);
                        break;
                        }
  case XTYP_UNREGISTER:
   {
                        return ((HDDEDATA)TRUE);
                        break;
                        }
  case XTYP_DISCONNECT:
                        {
                        return ((HDDEDATA)TRUE);
                        break;
                        }
                case XTYP_REQUEST:
                        {
  CMainFrame* pFrame = (CMainFrame*)AfxGetMainWnd();
  if (pFrame!=NULL)
   {
   CEtnaDoc*  pDoc = (CEtnaDoc*)pFrame->GetActiveDocument();
   if (pDoc->m_Comunicador.m_DDESock.m_Registered)
    {
    DWORD cb;
    char* psz;

    cb  = DdeQueryString(pDoc->m_Comunicador.m_DDESock.m_idInst,hszItem,NULL,0,CP_WINANSI);
    cb++;
    psz = new char[cb];
    DdeQueryString(pDoc->m_Comunicador.m_DDESock.m_idInst,hszItem,psz,cb,CP_WINANSI);

    CString ItemGate (psz);
    ItemGate.MakeUpper();

    delete psz;

    short i = 0;
    short n = pDoc->m_Grafo.GetGates()->GetSize();

    BOOL stop = FALSE;

    while ((i<n)&&(stop==FALSE))
     {
     if(pDoc->m_Grafo.GetGates()->GetAt(i).GetLabel()==ItemGate)
      {
      long dummy = pDoc->m_Grafo.GetGates()->GetAt(i).GetActivation();
      if(dummy==PASSIVE)
       {
       switch (pDoc->m_Grafo.GetGates()->GetAt(i).GetRetType())
        {
        case INTEGER_ATRIB:
         {
         CString string;
```

```
                sprintf(string.GetBuffer(20),"%d",pDoc->m_Grafo.GetGates()->GetAt(i).GetInteger());
                string.ReleaseBuffer(-1);
                DWORD cb = string.GetLength();
                cb++;
                char* psz;
                psz = new char[cb];
                for (DWORD i=0;i<(cb-1);i++)
                  psz[i]=string.GetAt(i);
                psz[i]=13;
                HDDEDATA myData = DdeCreateDataHandle(pDoc->m_Comunicador.m_DDESock.m_idInst, (LPBYTE)psz, cb,
0, hszItem, CF_TEXT, 0);
                delete psz;
                return myData;
                break;
                };
               case TEXT_ATRIB:
                {
                CString string = pDoc->m_Grafo.GetGates()->GetAt(i).GetString();
                DWORD cb = string.GetLength();
                cb++;
                char* psz;
                psz = new char[cb];
                for (DWORD i=0;i<(cb-1);i++)
                  psz[i]=string.GetAt(i);
                psz[i]=13;
                HDDEDATA myData = DdeCreateDataHandle(pDoc->m_Comunicador.m_DDESock.m_idInst, (LPBYTE)psz, cb,
0, hszItem, CF_TEXT, 0);
                delete psz;
                return myData;
                break;
                }
               case BOOLEAN:
                {
                CString string;
                sprintf(string.GetBuffer(20),"%d",pDoc->m_Grafo.GetGates()->GetAt(i).GoAhead());
                string.ReleaseBuffer(-1);
                DWORD cb = string.GetLength();
                cb++;
                char* psz;
                psz = new char[cb];
                for (DWORD i=0;i<(cb-1);i++)
                  psz[i]=string.GetAt(i);
                psz[i]=13;
                HDDEDATA myData = DdeCreateDataHandle(pDoc->m_Comunicador.m_DDESock.m_idInst, (LPBYTE)psz, cb,
0, hszItem, CF_TEXT, 0);
                delete psz;
                return myData;
                break;
                }
              };
             }
            stop = TRUE;
            }
           i++;
           }
          }
         }
                                return ((HDDEDATA)FALSE);
                                break;
                                }
                        case XTYP_POKE:
                                {
  CMainFrame* pFrame = (CMainFrame*)AfxGetMainWnd();
  if (pFrame!=NULL)
   {
   CEtnaDoc*  pDoc = (CEtnaDoc*)pFrame->GetActiveDocument();
   if (pDoc->m_Comunicador.m_DDESock.m_Registered)
    {
    DWORD cb;
    char* psz;

    cb  = DdeQueryString(pDoc->m_Comunicador.m_DDESock.m_idInst,hszItem,NULL,0,CP_WINANSI);
```

```
cb++;
psz = new char[cb];
DdeQueryString(pDoc->m_Comunicador.m_DDESock.m_idInst,hszItem,psz,cb,CP_WINANSI);

CString ItemGate (psz);
ItemGate.MakeUpper();

delete psz;

short i = 0;
short n = pDoc->m_Grafo.GetGates()->GetSize();

BOOL stop = FALSE;

while ((i<n)&&(stop==FALSE))
 {
 if(pDoc->m_Grafo.GetGates()->GetAt(i).GetLabel()==ItemGate)
  {
  if(pDoc->m_Grafo.GetGates()->GetAt(i).GetActivation()==PASSIVE)
   {
   switch (pDoc->m_Grafo.GetGates()->GetAt(i).GetRetType())
    {
    case INTEGER_ATRIB:
     {
     short inteiro;
                //convert data from handle to string format
     cb = DdeGetData(hData, NULL, 0, 0);
                psz = new char[cb];
     DdeGetData(hData, (LPBYTE)psz, cb, 0L);
     inteiro = atoi((const char*)psz);
     pDoc->m_Grafo.GetGates()->ElementAt(i).SetInteger(inteiro);
   return((HDDEDATA)DDE_FACK);
     break;
     };
    case TEXT_ATRIB:
     {
     CString string;
                //convert data from handle to string format
     cb = DdeGetData(hData, NULL, 0, 0);
              psz = new char[cb];
     DdeGetData(hData, (LPBYTE)psz, cb, 0L);
     char ch;
     for (DWORD j=0;j<cb;j++)
      {
      ch = psz[j];
      if (ch!=13)
       string+=ch;
      else
       break;
      }
     pDoc->m_Grafo.GetGates()->ElementAt(i).SetString(string);

   return((HDDEDATA)DDE_FACK);
     break;
     }
    case BOOLEAN:
     {
     BOOL boolean;
                //convert data from handle to string format
     cb = DdeGetData(hData, NULL, 0, 0);
                psz = new char[cb];
     DdeGetData(hData, (LPBYTE)psz, cb, 0L);
     boolean = atoi((const char*)psz);
     pDoc->m_Grafo.GetGates()->ElementAt(i).SetBOOL(boolean);
                        return ((HDDEDATA)TRUE);
     break;
     }
    };
   }
  stop = TRUE;
  }
 i++;
```

```
        }
       }
      }
                                return ((HDDEDATA)FALSE);
                                break;
                                }
                        case XTYP_CONNECT:
                                {
    CMainFrame* pFrame = (CMainFrame*)AfxGetMainWnd();
    if (pFrame!=NULL)
     {
     CEtnaDoc*  pDoc = (CEtnaDoc*)pFrame->GetActiveDocument();
     if (pDoc->m_Comunicador.m_DDESock.m_Registered)
      {
      HSZ hszMyTopic;                                                   //handle to my topic
              hszMyTopic = DdeCreateStringHandle(
                                pDoc->m_Comunicador.m_DDESock.m_idInst,     /* instance identifier   */
                                "data",                                                         /* System topic
*/
                                CP_WINANSI);                    /* Windows ANSI code page */
      if(!DdeCmpStringHandles(hszMyTopic,hszTopic))
       {
                                return ((HDDEDATA)TRUE);
       }
      }
     }
                                return ((HDDEDATA)FALSE);
                                break;
                                }
  };
 return((HDDEDATA)FALSE);
}
```

Files Mainfrm.h and Mainfrm.cpp - E-MFG Controller Main Frame Window

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CDialogBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
```

```
        afx_msg void OnClose();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Etna.h"

#include "elements.h"       //E-MFG Elements
#include "status.h"         //Interpreter Status
#include "controledlg.h"    //Control Dialog
#include "EtnaView.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_WM_CLOSE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this,IDD_TOOLBAR,CBRS_TOP,1000))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }


        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        short cx,cy;
 long  dbunits, dbx, dby;


        cx = GetSystemMetrics(SM_CXSCREEN);
```

```
            cy = GetSystemMetrics(SM_CYSCREEN);

   dbunits = GetDialogBaseUnits();

   dbx = dbunits/65536;
   dby = dbunits%65536;


            cs.cx = 3700*dbx/100;
            cs.cy = 5400*dby/100;

            cs.x = (cx-cs.cx)/2;
            cs.y = (cy-cs.cy)/2;

            return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
            CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
            CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnClose()
{
            CEtnaView* pView = (CEtnaView*)GetActiveView();

 if (pView->mp_myControle==NULL)
            CFrameWnd::OnClose();
 else
   MessageBox("Não é possível fechar a aplicação\n  com a janela de controle aberta","Não Permitido",MB_OK);
}
```

Files Etnadoc.h and Etnadoc.cpp - E-MFG Controller Document

```
// EtnaDoc.h : interface of the CEtnaDoc class
//
/////////////////////////////////////////////////////////////////////////

class CEtnaDoc : public CDocument
{
protected: // create from serialization only
            CEtnaDoc();
            DECLARE_DYNCREATE(CEtnaDoc)

// Attributes
public:
   CString m_ScriptFile;                                                    //EMFG Script File
   CString m_ScriptPath;
            BOOL    m_ScriptFileSet;                                        //EMFG Script File is
specified

            BOOL                m_TablesBuilt;
 BOOL    m_LogTransitions;

            CGraph                                  m_Grafo;
            //EMFG Graph
            CInterpreter                m_Interpretador;        //EMFG Interpreter
```

```
        CCommunicator        m_Comunicador;        //EMFG Communicator
  CMarkManager  m_MarkManager;  //EMFG Mark Manager


// Operations
public:
  void RunControl();
  void RunCommunication();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CEtnaDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CEtnaDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CEtnaDoc)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
// EtnaDoc.cpp : implementation of the CEtnaDoc class
//

#include "stdafx.h"

#include <ddeml.h>                                      //DDE Management Library

#include "Etna.h"

#include "elements.h"     //E-MFG Graph Elements

#include "interp.h"                                     //E-MFG Interpreter Definition
#include "ddesock.h"                                    //DDE Socket Definition
#include "comm.h"                                       //E-MFG Communicator Definition
#include "manager.h"      //E-MFG Mark Manager Definition


#include "EtnaDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEtnaDoc

IMPLEMENT_DYNCREATE(CEtnaDoc, CDocument)

BEGIN_MESSAGE_MAP(CEtnaDoc, CDocument)
        //{{AFX_MSG_MAP(CEtnaDoc)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEtnaDoc construction/destruction
```

```
CEtnaDoc::CEtnaDoc()
{
        m_ScriptFile = "não definido ainda";
  m_ScriptPath = "";
        m_ScriptFileSet = FALSE;

        m_TablesBuilt = FALSE;

 m_LogTransitions = TRUE;

 m_MarkManager.Create(&m_Grafo);
 m_MarkManager.SetFileLog(m_LogTransitions);
}

CEtnaDoc::~CEtnaDoc()
{
m_Grafo.GetAttribTemplates()->RemoveAll();
m_Grafo.GetGates()->RemoveAll();
m_Grafo.GetTransitions()->RemoveAll();
m_Grafo.GetBoxes()->RemoveAll();
m_Grafo.GetArcs()->RemoveAll();

}

BOOL CEtnaDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CEtnaDoc serialization

void CEtnaDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                ar<<m_ScriptFile;
                ar<<m_ScriptPath;
                ar.Write(&m_ScriptFileSet,2);
                ar.Write(&m_LogTransitions,2);
        }
        else
        {
                ar>>m_ScriptFile;
                ar>>m_ScriptPath;
                ar.Read(&m_ScriptFileSet,2);
                ar.Read(&m_LogTransitions,2);
        }
}

/////////////////////////////////////////////////////////////////////////////
// CEtnaDoc diagnostics

#ifdef _DEBUG
void CEtnaDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CEtnaDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG
```

```
////////////////////////////////////////////////////////////////////
// CEtnaDoc commands

void CEtnaDoc::RunCommunication()
{
 // Sets I/O Controller Graph
 m_Comunicador.SetGraph(&m_Grafo);
 // Calls I/O Controller to update the data highway
 m_Comunicador.UpdateAllGates();
}

void CEtnaDoc::RunControl()
{
 // Calls Mark Manager to update temporized structural elements
 m_MarkManager.UpdateTimers(m_Comunicador.GetLastUpdate());
 // Calls Mark Manager to verify fireable transitions
 m_MarkManager.SetUpTransitions();
 // Calls Mark Manager to solve any input or output conflicts that may have ocurred
 m_MarkManager.SeekAndSolveConflicts();
 // Calls Mark Manager to fire the fireable transitions
 m_MarkManager.FireTransitions();// no transition fired will be logged
}
```

Files Etnaview.h and Etnaview.cpp - E-MFG Controller Document View

```
// EtnaView.h : header file
//

////////////////////////////////////////////////////////////////////
// CEtnaView form view
//{{AFX_INCLUDES()
#include "gridctrl.h"
//}}AFX_INCLUDES

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CEtnaView : public CFormView
{
protected:
        CEtnaView();        // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CEtnaView)

// Form Data
public:
        //{{AFX_DATA(CEtnaView)
        enum { IDD = IDD_NEWVIEW };
        CString         m_script;
 CString      m_title;
        CGridCtrl      m_grid;
 CComboBox     m_dumpbox;
 CComboBox     m_literalbox;
 BOOL          m_controle;
 CControleDlg* mp_myControle;
 CIstatus*     mp_myStatus;
        //}}AFX_DATA

 CLinesArray   m_dummy;
// Attributes
public:

// Operations
public:
        virtual void OnDraw(CDC* pDC);
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CEtnaView)
        public:
        virtual void OnInitialUpdate();
        protected:
```

```
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

        void ShowTable(CLinesArray &table);
        void ShowTable(CWordsArray &table);

// Implementation
protected:
        virtual ~CEtnaView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
        //{{AFX_MSG(CEtnaView)
        afx_msg void OnActionsCompile();
        afx_msg void OnDumpGraphdump();
        afx_msg void OnUpdateDumpGraphdump(CCmdUI* pCmdUI);
        afx_msg void OnFileSettextfile();
        afx_msg void OnTablesScriptfile();
        afx_msg void OnUpdateTablesScriptfile(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralArcs(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralAtribuiestbelse(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralAtribuiesthen(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralBoxes(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralCondicionaisdastransies(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralCondicionaisdosgates(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralCondicionaistb(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralFilters(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralGates(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralIncludes(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralInitialmarks(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralMarkattributes(CCmdUI* pCmdUI);
        afx_msg void OnUpdateLiteralTransitions(CCmdUI* pCmdUI);
        afx_msg void OnLiteralArcs();
        afx_msg void OnLiteralAtribuiestbelse();
        afx_msg void OnLiteralAtribuiesthen();
        afx_msg void OnLiteralBoxes();
        afx_msg void OnLiteralCondicionaisdastransies();
        afx_msg void OnLiteralCondicionaisdosgates();
        afx_msg void OnLiteralCondicionaistb();
        afx_msg void OnLiteralFilters();
        afx_msg void OnLiteralGates();
        afx_msg void OnLiteralIncludes();
        afx_msg void OnLiteralInitialmarks();
        afx_msg void OnLiteralMarkattributes();
        afx_msg void OnLiteralTransitions();
        afx_msg void OnSelchangeDump();
        afx_msg void OnSelchangeLiteral();
        afx_msg void OnOpesComunicaodde();
        afx_msg void OnUpdateDumpValoresdosgates(CCmdUI* pCmdUI);
        afx_msg void OnDumpValoresdosgates();
        afx_msg void OnAesAcionarcontrole();
        afx_msg void OnUpdateOpesRegistrardisparodetransies(CCmdUI* pCmdUI);
        afx_msg void OnOpesRegistrardisparodetransies();
        afx_msg void OnBoxpropertiesdump();
        afx_msg void OnUpdateBoxpropertiesdump(CCmdUI* pCmdUI);
        afx_msg void OnActionLogGraph();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
// EtnaView.cpp : implementation file
//

#include "stdafx.h"

#include <ddeml.h>                                    //DDE Management Library
#include <stdio.h>
```

```
#include "etna.h"

#include "elements.h"       //E-MFG Elements

#include "interp.h"                                          //E-MFG Interpreter Definition
#include "ddesock.h"                                    //DDE Socket Definition
#include "comm.h"                                                    //E-MFG Communicator Definition
#include "manager.h"        //E-MFG Mark Manager Definition

#include "status.h"         //Interpreter Status
#include "ddeoptions.h"     //DDE Options Configuration Dialog
#include "controledlg.h"    //Control Dialog

#include "EtnaView.h"
#include "EtnaDoc.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEtnaView

IMPLEMENT_DYNCREATE(CEtnaView, CFormView)
CEtnaView::CEtnaView()
        : CFormView(CEtnaView::IDD)
{
        //{{AFX_DATA_INIT(CEtnaView)
        m_script = _T("não definido ainda");
 m_title = _T("não definido ainda");
 m_controle = FALSE;
 mp_myControle = NULL;
 mp_myStatus = NULL;
        //}}AFX_DATA_INIT
}

CEtnaView::~CEtnaView()
{
}

void CEtnaView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CEtnaView)
        DDX_Text(pDX, IDC_SCRIPT, m_script);
 DDX_Text(pDX, IDC_TITLE, m_title);
        DDX_Control(pDX, IDC_GRID, m_grid);
        DDX_Control(pDX, IDC_LITERAL, m_literalbox);
        DDX_Control(pDX, IDC_DUMP, m_dumpbox);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CEtnaView, CFormView)
        //{{AFX_MSG_MAP(CEtnaView)
        ON_COMMAND(ID_ACTIONS_COMPILE, OnActionsCompile)
        ON_COMMAND(ID_DUMP_GRAPHDUMP, OnDumpGraphdump)
        ON_UPDATE_COMMAND_UI(ID_DUMP_GRAPHDUMP, OnUpdateDumpGraphdump)
        ON_COMMAND(ID_FILE_SETTEXTFILE, OnFileSettextfile)
        ON_COMMAND(ID_TABLES_SCRIPTFILE, OnTablesScriptfile)
        ON_UPDATE_COMMAND_UI(ID_TABLES_SCRIPTFILE, OnUpdateTablesScriptfile)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_ARCS, OnUpdateLiteralArcs)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_ATRIBUIESTBELSE, OnUpdateLiteralAtribuiestbelse)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_ATRIBUIESTHEN, OnUpdateLiteralAtribuiesthen)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_BOXES, OnUpdateLiteralBoxes)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_CONDICIONAISDASTRANSIES,
OnUpdateLiteralCondicionaisdastransies)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_CONDICIONAISDOSGATES,
OnUpdateLiteralCondicionaisdosgates)
```

```
        ON_UPDATE_COMMAND_UI(ID_LITERAL_CONDICIONAISTB, OnUpdateLiteralCondicionaistb)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_FILTERS, OnUpdateLiteralFilters)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_GATES, OnUpdateLiteralGates)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_INCLUDES, OnUpdateLiteralIncludes)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_INITIALMARKS, OnUpdateLiteralInitialmarks)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_MARKATTRIBUTES, OnUpdateLiteralMarkattributes)
        ON_UPDATE_COMMAND_UI(ID_LITERAL_TRANSITIONS, OnUpdateLiteralTransitions)
        ON_COMMAND(ID_LITERAL_ARCS, OnLiteralArcs)
        ON_COMMAND(ID_LITERAL_ATRIBUIESTBELSE, OnLiteralAtribuiestbelse)
        ON_COMMAND(ID_LITERAL_ATRIBUIESTHEN, OnLiteralAtribuiesthen)
        ON_COMMAND(ID_LITERAL_BOXES, OnLiteralBoxes)
        ON_COMMAND(ID_LITERAL_CONDICIONAISDASTRANSIES, OnLiteralCondicionaisdastransies)
        ON_COMMAND(ID_LITERAL_CONDICIONAISDOSGATES, OnLiteralCondicionaisdosgates)
        ON_COMMAND(ID_LITERAL_CONDICIONAISTB, OnLiteralCondicionaistb)
        ON_COMMAND(ID_LITERAL_FILTERS, OnLiteralFilters)
        ON_COMMAND(ID_LITERAL_GATES, OnLiteralGates)
        ON_COMMAND(ID_LITERAL_INCLUDES, OnLiteralIncludes)
        ON_COMMAND(ID_LITERAL_INITIALMARKS, OnLiteralInitialmarks)
        ON_COMMAND(ID_LITERAL_MARKATTRIBUTES, OnLiteralMarkattributes)
        ON_COMMAND(ID_LITERAL_TRANSITIONS, OnLiteralTransitions)
        ON_CBN_SELCHANGE(IDC_DUMP, OnSelchangeDump)
        ON_CBN_SELCHANGE(IDC_LITERAL, OnSelchangeLiteral)
        ON_COMMAND(ID_OPES_COMUNICAODDE, OnOpesComunicaodde)
        ON_UPDATE_COMMAND_UI(ID_DUMP_VALORESDOSGATES, OnUpdateDumpValoresdosgates)
        ON_COMMAND(ID_DUMP_VALORESDOSGATES, OnDumpValoresdosgates)
        ON_COMMAND(ID_AES_ACIONARCONTROLE, OnAesAcionarcontrole)
        ON_UPDATE_COMMAND_UI(ID_OPES_REGISTRARDISPARODETRANSIES,
OnUpdateOpesRegistrardisparodetransies)
        ON_COMMAND(ID_OPES_REGISTRARDISPARODETRANSIES, OnOpesRegistrardisparodetransies)
        ON_COMMAND(ID_BOXPROPERTIESDUMP, OnBoxpropertiesdump)
        ON_UPDATE_COMMAND_UI(ID_BOXPROPERTIESDUMP, OnUpdateBoxpropertiesdump)
        ON_BN_CLICKED(ID_ACTIONS_COMPILE, OnActionsCompile)
        ON_COMMAND(ID_ACTION_LOG_GRAPH, OnActionLogGraph)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
// CEtnaView diagnostics

#ifdef _DEBUG
void CEtnaView::AssertValid() const
{
        CFormView::AssertValid();
}

void CEtnaView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}
#endif //_DEBUG

void CEtnaView::ShowTable(CLinesArray &table)
{
        short lines,cols,max,max2;
        short i,j,n;
        char psz[4];

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);

        lines = table.GetSize();

        if (lines==0)
                {
                m_grid.SetFixedRows(0);
                m_grid.SetFixedCols(0);
                m_grid.SetRows(1);
                m_grid.SetCols(1);
                m_grid.SetColWidth(0,3500);
                m_grid.SetRow(0);
                m_grid.SetCol(0);
                m_grid.SetText("A tabela está vazia");
```

```
                }
        else
                {
                max = 0;
                for (i=0;i<lines;i++)
                        {
                        cols = table.GetAt(i).GetSize();
                        if (cols>max)
                                max = cols;
                        }
                m_grid.SetRows (lines);
                m_grid.SetCols (max+1);
                m_grid.SetFixedCols(1);
        m_grid.SetColWidth(0,400);
                for (i=0;i<max;i++)
    {
    max2=0;
    for (j=0;j<lines;j++)
     {
     cols = table.GetAt(j).GetSize();
     if(i<cols)
      {
      CString dummy = table.GetAt(j).GetAt(i);
      n = dummy.GetLength();
      if (n>max2)
       max2=n;
      }
     }
    if (max2==0)
                m_grid.SetColWidth(i+1,300);
    else
                m_grid.SetColWidth(i+1,max2*150);
    }
                for (i=0;i<lines;i++)
                        {
                        m_grid.SetRow(i);
                        m_grid.SetCol(0);
                        sprintf(psz,"%d",i);
                        m_grid.SetText(psz);
                        cols = table.GetAt(i).GetSize();
                        for (j=1;j<=cols;j++)
                                {
                                m_grid.SetCol(j);
                                m_grid.SetText(table.GetAt(i).GetAt(j-1));
                                }
                        }
                }
  m_grid.Refresh();
}

void CEtnaView::ShowTable(CWordsArray &table)
{
        short lines;
        short i,n,max;

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);

        lines = table.GetSize();

        if (lines==0)
                {
                m_grid.SetFixedRows(0);
                m_grid.SetFixedCols(0);
                m_grid.SetRows(1);
                m_grid.SetCols(1);
                m_grid.SetColWidth(0,3500);
                m_grid.SetRow(0);
                m_grid.SetCol(0);
                m_grid.SetText("A tabela está vazia");
                }
        else
```

```
                    {
   max = 0;
                    m_grid.SetRows (lines);
                    m_grid.SetCols (1);
                    m_grid.SetFixedRows(0);
                    m_grid.SetFixedCols(0);
   for (i=0;i<lines;i++)
    {
    n = table.GetAt(i).GetLength();
    if (n>max)
     max=n;
    }
                    m_grid.SetColWidth(0,max*95);
                    m_grid.SetCol(0);
                    for (i=0;i<lines;i++)
                              {
                              m_grid.SetRow(i);
                              m_grid.SetText(table.GetAt(i));
                              }
                    }
}

////////////////////////////////////////////////////////////////
// CEtnaView message handlers
void CEtnaView::OnDraw(CDC* pDC)
{
          CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
          ASSERT_VALID(pDoc);

          m_script = pDoc->m_ScriptFile;
          if (pDoc!=NULL)
                    if (pDoc->m_TablesBuilt)
    m_title = pDoc->m_Grafo.m_title;

 UpdateData(FALSE);
}

void CEtnaView::OnActionsCompile()
{
 long  dbunits,dbx,dby;
 long  cx,cy;
 RECT  Client;
 POINT Pos;

          CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
          ASSERT_VALID(pDoc),

 if (!pDoc->m_ScriptFileSet)
  {
  MessageBox("O arquivo do EMFG Script ainda não foi definido","Script não definido",MB_OK);
  return;
  }

 MSG msg;

 //calcula coordenadas da janela de controle
 dbunits = GetDialogBaseUnits();
 dbx = dbunits/65536;
 dby = dbunits%65536;

 cx = 1200*dbx/100;
 cy = 2800*dby/100;

 GetClientRect(&Client);
 Pos.x = (Client.right-cx)/2;
 Pos.y = (Client.bottom-cy)/2;

 ClientToScreen(&Pos);

 if (mp_myStatus!=NULL)
  {
  mp_myStatus->DestroyWindow();
```

```
  delete mp_myControle;
  mp_myStatus = NULL;
  }

mp_myStatus = new CIstatus;
mp_myStatus->Create(IDD_INTERP,this);
        mp_myStatus->ShowWindow(SW_SHOW);
mp_myStatus->SetWindowPos(&wndTop,Pos.x,Pos.y,cx,cy,SWP_SHOWWINDOW);

        TRY
                {
                //status window update
                mp_myStatus->m_10 = "Script File :";
                mp_myStatus->m_20 = "Literal Tables :";
                mp_myStatus->m_30 = "Cross Reference :";
    mp_myStatus->m_11.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_GO)));
    mp_myStatus->m_21.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));
    mp_myStatus->m_31.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));
                mp_myStatus->UpdateData(FALSE);
                mp_myStatus->UpdateWindow();

 //Graph clean-up
 pDoc->m_Grafo.GetAttribTemplates()->RemoveAll();
 pDoc->m_Grafo.GetBoxes()->RemoveAll();
 pDoc->m_Grafo.GetArcs()->RemoveAll();
 pDoc->m_Grafo.GetTransitions()->RemoveAll();
 pDoc->m_Grafo.GetGates()->RemoveAll();

 //Sets Graph
                pDoc->m_Interpretador.SetGraph(&(pDoc->m_Grafo));

 //Sets Script Path
   pDoc->m_Interpretador.m_path = pDoc->m_ScriptPath;

                //reads file
                pDoc->m_Interpretador.Read(pDoc->m_ScriptFile);

                //cancel routine
                if (PeekMessage(&msg,NULL,0,0,PM_REMOVE));
                        {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                        }

                if (mp_myStatus->cancel)
                        AfxThrowUserException();

                //updating status window
mp_myStatus->m_11.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_DONE)));
mp_myStatus->m_21.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_GO)));
mp_myStatus->m_31.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));
                mp_myStatus->UpdateWindow();

                //builds literal tables
                pDoc->m_Interpretador.BuildTables();

                //cancel routine
                if (PeekMessage(&msg,NULL,0,0,PM_REMOVE));
                        {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                        }

                if (mp_myStatus->cancel)
                        AfxThrowUserException();

                //updating status window
mp_myStatus->m_21.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_DONE)));
mp_myStatus->m_31.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_GO)));
                mp_myStatus->UpdateWindow();

//performs cross-reference
```

```
                pDoc->m_Interpretador.CrossRef();
                        }
    CATCH( CUserException, e )
     {
     //clean-up!
     //destrói janela de status
     mp_myStatus->DestroyWindow();
     delete mp_myStatus;
     mp_myStatus = NULL;
     //
     // limpando as tabelas do Interpretador
     //
                pDoc->m_Interpretador.m_FileTable.RemoveAll();
                pDoc->m_Interpretador.m_AttribTable.RemoveAll();
                pDoc->m_Interpretador.m_BoxesTable.RemoveAll();
                pDoc->m_Interpretador.m_TransitsTable.RemoveAll();
                pDoc->m_Interpretador.m_ArcsTable.RemoveAll();
                pDoc->m_Interpretador.m_GatesTable.RemoveAll();
                pDoc->m_Interpretador.m_MarksTable.RemoveAll();
                pDoc->m_Interpretador.m_FilterTable.RemoveAll();
                pDoc->m_Interpretador.m_TBCondTable.RemoveAll();
                pDoc->m_Interpretador.m_TBThenTable.RemoveAll();
                pDoc->m_Interpretador.m_TBElseTable.RemoveAll();
                pDoc->m_Interpretador.m_TransCondTable.RemoveAll();
                pDoc->m_Interpretador.m_GatesCondTable.RemoveAll();
     //
     // limpando o grafo em si
     //
                pDoc->m_Grafo.GetAttribTemplates()->RemoveAll();
                pDoc->m_Grafo.GetArcs()->RemoveAll();
                pDoc->m_Grafo.GetGates()->RemoveAll();
                pDoc->m_Grafo.GetTransitions()->RemoveAll();
                pDoc->m_Grafo.GetBoxes()->RemoveAll();
     //
     // reseta a interface
     //
               pDoc->m_TablesBuilt = FALSE;
     m_dumpbox.SetCurSel(-1);
     m_literalbox.SetCurSel(-1);
     ShowTable(m_dummy);
     pDoc->UpdateAllViews(NULL,0,NULL);
                        return;                                                    // User already
notified.
     }
    CATCH( CMemoryException, e )
     {
                        MessageBox("Not enough memory!!","Memory Allocation Error",MB_OK);
                        return;                                                    // Memory
exception must be notified.
     }
    END_CATCH

    //destrói janela de status
    mp_myStatus->DestroyWindow();
    delete mp_myStatus;
    mp_myStatus = NULL;

    //com o grafo já construído
    pDoc->m_TablesBuilt = TRUE;
    pDoc->m_Grafo.m_title = pDoc->m_Interpretador.m_title;
    pDoc->m_Comunicador.m_DDESock.RegisterApp(pDoc->m_Grafo.m_title);
    pDoc->m_Comunicador.SetGraph(&pDoc->m_Grafo);
    pDoc->m_Comunicador.UpdateAllGates();

    //mostra o arquivo
    m_dumpbox.SetCurSel(-1);
    m_literalbox.SetCurSel(0);
    ShowTable(pDoc->m_Interpretador.m_FileTable);
    pDoc->UpdateAllViews(NULL,0,NULL);
}

void CEtnaView::OnDumpGraphdump()
```

```
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);

        CLinesArray* BoxLinksDump;
  CLinesArray Refresh;

  BoxLinksDump = new CLinesArray;

        ShowTable(Refresh);
  pDoc->m_Grafo.GetBoxConnectionsDump(BoxLinksDump);
        ShowTable(*BoxLinksDump);

  delete BoxLinksDump;

 m_dumpbox.SetCurSel(0);
 m_literalbox.SetCurSel(-1);

 UpdateData(FALSE);

        pDoc->UpdateAllViews(NULL,0,NULL);
}

void CEtnaView::OnUpdateDumpGraphdump(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnFileSettextfile()
{
 CString    Filters = "Text Files (*.txt) | *.txt||";
 CFileDialog OpenDialogBox(TRUE,NULL,NULL,OFN_HIDEREADONLY,Filters);

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);

        int success;

        CString SelectedFile;

 CString filename;

        success=OpenDialogBox.DoModal();

if(success==IDOK)
 {
 SelectedFile  = OpenDialogBox.GetPathName();
 filename     = OpenDialogBox.GetFileTitle();
 pDoc->m_ScriptFile = filename+".txt";
 pDoc->m_ScriptPath = SelectedFile.Left(SelectedFile.Find(filename));
                pDoc->m_ScriptFileSet = TRUE;
        UpdateData(FALSE);

 //
 // limpando as tabelas do Interpretador
 //
        pDoc->m_Interpretador.m_FileTable.RemoveAll();
        pDoc->m_Interpretador.m_AttribTable.RemoveAll();
        pDoc->m_Interpretador.m_BoxesTable.RemoveAll();
        pDoc->m_Interpretador.m_TransitsTable.RemoveAll();
        pDoc->m_Interpretador.m_ArcsTable.RemoveAll();
        pDoc->m_Interpretador.m_GatesTable.RemoveAll();
        pDoc->m_Interpretador.m_MarksTable.RemoveAll();
        pDoc->m_Interpretador.m_FilterTable.RemoveAll();
        pDoc->m_Interpretador.m_TBCondTable.RemoveAll();
        pDoc->m_Interpretador.m_TBThenTable.RemoveAll();
```

```
            pDoc->m_Interpretador.m_TBElseTable.RemoveAll();
            pDoc->m_Interpretador.m_TransCondTable.RemoveAll();
            pDoc->m_Interpretador.m_GatesCondTable.RemoveAll();
    //
    // limpando o grafo em si
    //
            pDoc->m_Grafo.GetAttribTemplates()->RemoveAll();
            pDoc->m_Grafo.GetArcs()->RemoveAll();
            pDoc->m_Grafo.GetGates()->RemoveAll();
            pDoc->m_Grafo.GetTransitions()->RemoveAll();
            pDoc->m_Grafo.GetBoxes()->RemoveAll();
    //
    // reseta a interface
    //
        pDoc->m_TablesBuilt = FALSE;
    m_dumpbox.SetCurSel(-1);
    m_literalbox.SetCurSel(-1);
    ShowTable(m_dummy);
    }

        pDoc->UpdateAllViews(NULL,0,NULL);
}

void CEtnaView::OnTablesScriptfile()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
            ShowTable(pDoc->m_Interpretador.m_FileTable);
    m_literalbox.SetCurSel(0);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnUpdateTablesScriptfile(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnInitialUpdate()
{
        CFormView::OnInitialUpdate();

 m_dumpbox.ResetContent();
 m_dumpbox.AddString("Geral");
 m_dumpbox.AddString("Valores de Gate");

 m_literalbox.ResetContent();
 m_literalbox.AddString("Arquivo");
 m_literalbox.AddString("Include");
 m_literalbox.AddString("Atributos");
 m_literalbox.AddString("Arcos");
 m_literalbox.AddString("Filtros");
 m_literalbox.AddString("Boxes");
 m_literalbox.AddString("Transições");
 m_literalbox.AddString("Gates");
 m_literalbox.AddString("Marcas Iniciais");
 m_literalbox.AddString("Condicionais - TB");
 m_literalbox.AddString("Atribuições TB THEN");
 m_literalbox.AddString("Atribuições TB ELSE");
 m_literalbox.AddString("Condicionais - Gates");
 m_literalbox.AddString("Condicionais - Transições");
}
```

```
void CEtnaView::OnUpdateLiteralArcs(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralAtribuiestbelse(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralAtribuiesthen(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralBoxes(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralCondicionaisdastransies(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralCondicionaisdosgates(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralCondicionaistb(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralFilters(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}
```

```
void CEtnaView::OnUpdateLiteralGates(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralIncludes(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralInitialmarks(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralMarkattributes(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnUpdateLiteralTransitions(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(FALSE);
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}


void CEtnaView::OnLiteralArcs()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_ArcsTable);
    m_literalbox.SetCurSel(3);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralAtribuiestbelse()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_TBElseTable);
    m_literalbox.SetCurSel(11);
    m_dumpbox.SetCurSel(-1);
```

```
                pDoc->UpdateAllViews(NULL,0,NULL);
        }
}

void CEtnaView::OnLiteralAtribuiesthen()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_TBThenTable);
    m_literalbox.SetCurSel(10);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralBoxes()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_BoxesTable);
    m_literalbox.SetCurSel(5);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralCondicionaisdastransics()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_TransCondTable);
    m_literalbox.SetCurSel(13);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralCondicionaisdosgates()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_GatesCondTable);
    m_literalbox.SetCurSel(12);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralCondicionaistb()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
```

```
                ShowTable(pDoc->m_Interpretador.m_TBCondTable);
        m_literalbox.SetCurSel(9);
        m_dumpbox.SetCurSel(-1);
                pDoc->UpdateAllViews(NULL,0,NULL);
        }
}

void CEtnaView::OnLiteralFilters()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_FilterTable);
    m_literalbox.SetCurSel(4);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }

}

void CEtnaView::OnLiteralGates()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_GatesTable);
    m_literalbox.SetCurSel(7);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralIncludes()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_IncludeTable);
    m_literalbox.SetCurSel(1);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralInitialmarks()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
        if (pDoc!=NULL)
                if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
            ShowTable(pDoc->m_Interpretador.m_MarksTable);
    m_literalbox.SetCurSel(8);
    m_dumpbox.SetCurSel(-1);
            pDoc->UpdateAllViews(NULL,0,NULL);
    }
}


void CEtnaView::OnLiteralMarkattributes()
{
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
```

```
            ASSERT_VALID(pDoc);
            if (pDoc!=NULL)
                    if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
                ShowTable(pDoc->m_Interpretador.m_AttribTable);
    m_literalbox.SetCurSel(2);
    m_dumpbox.SetCurSel(-1);
                pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnLiteralTransitions()
{
            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
            ASSERT_VALID(pDoc);
            if (pDoc!=NULL)
                    if (pDoc->m_TablesBuilt)
    {
    ShowTable(m_dummy);
                ShowTable(pDoc->m_Interpretador.m_TransitsTable);
    m_literalbox.SetCurSel(6);
    m_dumpbox.SetCurSel(-1);
                pDoc->UpdateAllViews(NULL,0,NULL);
    }
}

void CEtnaView::OnSelchangeDump()
{
            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

 short item;

 if (pDoc!=NULL)
                    if (pDoc->m_TablesBuilt)
   {
   item = m_dumpbox.GetCurSel();
   switch(item)
    {
    case 0:
     {
     OnDumpGraphdump();
     break;
     }
    case 1:
     {
     OnDumpValoresdosgates();
     break;
     }
    }
   }
 else
   {
                            MessageBox("As tabelas ainda não foram montadas.","Tabelas não montadas",MB_OK);
   m_dumpbox.SetCurSel(-1);
   }
}

void CEtnaView::OnSelchangeLiteral()
{
 short item;
          CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
 if (pDoc!=NULL)
                    if (pDoc->m_TablesBuilt)
   {
   item = m_literalbox.GetCurSel();
   switch(item)
    {
    case 0:
     {
     OnTablesScriptfile();
     break;
```

```
    }
   case 1:
    {
    OnLiteralIncludes();
    break;
    }
   case 2:
    {
    OnLiteralMarkattributes();
    break;
    }
   case 3:
    {
    OnLiteralArcs();
    break;
    }
   case 4:
    {
    OnLiteralFilters();
    break;
    }
   case 5:
    {
    OnLiteralBoxes();
    break;
    }
   case 6:
    {
    OnLiteralTransitions();
    break;
    }
   case 7:
    {
    OnLiteralGates();
    break;
    }
   case 8:
    {
    OnLiteralInitialmarks();
    break;
    }
   case 9:
    {
    OnLiteralCondicionaistb();
    break;
    }
   case 10:
    {
    OnLiteralAtribuiesthen();
    break;
    }
   case 11:
    {
    OnLiteralAtribuiestbelse();
    break;
    }
   case 12:
    {
    OnLiteralCondicionaisdosgates();
    break;
    }
   case 13:
    {
    OnLiteralCondicionaisdastransies();
    break;
    }
  }
 }
else
 {
                        MessageBox("As tabelas ainda não foram montadas.","Tabelas não montadas",MB_OK);
 m_literalbox.SetCurSel(-1);
```

```
    }
  }

  void CEtnaView::OnOpesComunicaodde()
  {
            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
            ASSERT_VALID(pDoc);

   CDDEOptions myDDEOptions;

   if (!pDoc->m_Comunicador.m_LogErrors)
    {
    myDDEOptions.m_warn = TRUE;
    myDDEOptions.m_log = FALSE;
    }

   myDDEOptions.m_timeout  = (short)pDoc->m_Comunicador.m_DDESock.m_timeout;
   myDDEOptions.mp_DDESock = &(pDoc->m_Comunicador.m_DDESock);

   if(myDDEOptions.DoModal()==IDOK)
    {
    if (myDDEOptions.m_warn)
     pDoc->m_Comunicador.m_LogErrors=FALSE;
    else
     pDoc->m_Comunicador.m_LogErrors=TRUE;
    pDoc->m_Comunicador.m_DDESock.m_timeout = myDDEOptions.m_timeout;
    }
}

void CEtnaView::OnUpdateDumpValoresdosgates(CCmdUI* pCmdUI)
{
            pCmdUI->Enable(FALSE);

            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

            if (pDoc!=NULL)
                        if ((pDoc->m_TablesBuilt)&&(!m_controle))
                                    pCmdUI->Enable(TRUE);
}

void CEtnaView::OnDumpValoresdosgates()
{
 CLinesArray GateValuesTable;
 CWordsArray GateLine;

 short i, n, rettype;

 char psz[10];

 CString dummy;

 CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

 n = pDoc->m_Grafo.GetGates()->GetSize();

 for (i=0;i<n;i++)
  {
  GateLine.RemoveAll();
  dummy = pDoc->m_Grafo.GetGates()->ElementAt(i).GetLabel();
  GateLine.Add(dummy);
  rettype = pDoc->m_Grafo.GetGates()->ElementAt(i).GetRetType();
  switch (rettype)
   {
   case INTEGER_ATRIB:
    {
    GateLine.Add("INTEGER");
    sprintf(psz,"%d",pDoc->m_Grafo.GetGates()->GetAt(i).GetInteger());
    GateLine.Add(psz);
    break;
    }
   case TEXT_ATRIB:
    {
```

```
        GateLine.Add("STRING");
        GateLine.Add(pDoc->m_Grafo.GetGates()->GetAt(i).GetString());
        break;
        }
       case BOOLEAN:
        {
        GateLine.Add("BOOLEAN");
        if(pDoc->m_Grafo.GetGates()->GetAt(i).GoAhead())
          GateLine.Add("TRUE");
        else
          GateLine.Add("FALSE");
        break;
        }
       };
     GateValuesTable.Add(GateLine);
     }
   ShowTable(GateValuesTable);

   m_dumpbox.SetCurSel(1);
   m_literalbox.SetCurSel(-1);

   pDoc->UpdateAllViews(NULL,0,NULL);
}

void CEtnaView::OnAesAcionarcontrole()
{
 long  dbunits, dbx, dby;
 long  cx, cy;
 RECT  Client;
 POINT Pos;

        MSG msg;

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

 if (!pDoc->m_TablesBuilt)
  {
  MessageBox("O Script ainda não foi interpretado","Script não interpretado",MB_OK);
  return;
  }

 //calcula coordenadas da janela de controle
 dbunits = GetDialogBaseUnits();
 dbx = dbunits/65536;
 dby = dbunits%65536;

 cx = 30*dbx;
 cy = 3375*dby/100;

 GetClientRect(&Client);
 Pos.x = (Client.right-cx)/2;
 Pos.y = (Client.bottom-cy)/2;

 ClientToScreen(&Pos);

 if (mp_myControle!=NULL)
  {
  mp_myControle->DestroyWindow();
  delete mp_myControle;
  mp_myControle = NULL;
  }

m_controle = TRUE;

//eria janela de controle
mp_myControle = new CControleDlg;
mp_myControle->Create(IDD_CONTROLE,this);
mp_myControle->ShowWindow(SW_SHOW);
mp_myControle->SetWindowPos(&wndTop,Pos.x,Pos.y,cx,cy,SWP_SHOWWINDOW);


while (mp_myControle->m_action!=LEAVE)
```

```
        {
        //message routine
                if (PeekMessage(&msg,NULL,0,0,PM_REMOVE));
                        {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                        }
        if ((mp_myControle->m_action!=NONE)&&(mp_myControle->m_action!=LEAVE))
         {
         //communication loop
         mp_myControle->m_control.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));
         mp_myControle->m_comm.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_GO)));
         mp_myControle->UpdateWindow();
         pDoc->RunCommunication();

         mp_myControle->m_Statual.SetWindowText(pDoc->m_Comunicador.m_Now.Format("%H:%M:%S"));
         mp_myControle->m_Stdelta.SetWindowText(pDoc->m_Comunicador.m_Delta.Format("%S")+" s");
         mp_myControle->m_Stcomm.SetWindowText(pDoc->m_Comunicador.m_Comm.Format("%S")+" s");
         mp_myControle->m_Stcont.SetWindowText(pDoc->m_Comunicador.m_Control.Format("%S")+" s");
         mp_myControle->UpdateWindow();

        //message routine
                if (PeekMessage(&msg,NULL,0,0,PM_REMOVE));
                        {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                        }
        //control loop
        mp_myControle->m_comm.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_DONE)));
        mp_myControle->m_control.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_GO)));
        mp_myControle->UpdateWindow();
        pDoc->RunControl();
        mp_myControle->m_control.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_DONE)));
        mp_myControle->UpdateWindow();

        if (mp_myControle->m_action==STEP)
          mp_myControle->m_action = NONE;
        }
       }

    //destrói janela de controle
    mp_myControle->DestroyWindow();
    delete mp_myControle;
    mp_myControle = NULL;

    OnDumpGraphdump();

    m_controle = FALSE;
  }



  void CEtnaView::OnUpdateOpesRegistrardisparodetransies(CCmdUI* pCmdUI)
  {
            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

            if (pDoc!=NULL)
                    pCmdUI->SetCheck(pDoc->m_LogTransitions);
  }

  void CEtnaView::OnOpesRegistrardisparodetransies()
  {
            CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

            if (pDoc!=NULL)
    {
    pDoc->m_LogTransitions = !pDoc->m_LogTransitions;
    pDoc->m_MarkManager.SetFileLog(pDoc->m_LogTransitions);
    }
  }

  void CEtnaView::OnBoxpropertiesdump()
```

```
{
        // Daniel 01/12
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);

        CLinesArray BoxLinksDump,Refresh;
        ShowTable(Refresh);
  pDoc->m_Grafo.GetBoxesPropertiesDump(&BoxLinksDump);
        ShowTable(BoxLinksDump);

  m_dumpbox.SetCurSel(0);
  m_literalbox.SetCurSel(-1);

  UpdateData(FALSE);

        pDoc->UpdateAllViews(NULL,0,NULL);
}

void CEtnaView::OnUpdateBoxpropertiesdump(CCmdUI* pCmdUI)
{
        // Daniel 01/12
        pCmdUI->Enable(FALSE);

        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();

        if (pDoc!=NULL)
                if ((pDoc->m_TablesBuilt)&&(!m_controle))
                        pCmdUI->Enable(TRUE);
}

void CEtnaView::OnActionLogGraph()
{
        // TODO: Add your command handler code here
        CEtnaDoc* pDoc = (CEtnaDoc*)GetDocument();
        ASSERT_VALID(pDoc);
 pDoc->m_MarkManager.LogGraph();
}
```

Files ControleDlg.h and ControleDlg.cpp - E-MFG Controller Control Dialog Box

```
// ControleDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CControleDlg dialog

class CControleDlg : public CDialog
{
// Construction
public:
        CControleDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CControleDlg)
        enum { IDD = IDD_CONTROLE };
        CStatic     m_Stcont;
        CStatic     m_Stcomm;
        CStatic     m_Stdelta;
        CStatic     m_Statual;
        CButton     m_bcont;
        CStatic     m_control;
        CStatic     m_comm;
        //}}AFX_DATA

short m_action;

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CControleDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
        //}}AFX_VIRTUAL
```

```
// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CControleDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnLeave();
        afx_msg void OnSingle();
        afx_msg void OnContinuous();
        afx_msg void OnClose();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
// ControleDlg.cpp : implementation file
//

#include "stdafx.h"
#include "etna.h"
#include "definitions.h"
#include "ControleDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CControleDlg dialog


CControleDlg::CControleDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CControleDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CControleDlg)
                m_action = NONE;
        //}}AFX_DATA_INIT
}


void CControleDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CControleDlg)
        DDX_Control(pDX, IDC_DCONT, m_Stcont);
        DDX_Control(pDX, IDC_DCOMM, m_Stcomm);
        DDX_Control(pDX, IDC_DELTA, m_Stdelta);
        DDX_Control(pDX, IDC_ATUAL, m_Statual);
        DDX_Control(pDX, ID_CONTINUOUS, m_bcont);
        DDX_Control(pDX, IDC_CONTROL, m_control);
        DDX_Control(pDX, IDC_COMM, m_comm);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CControleDlg, CDialog)
        //{{AFX_MSG_MAP(CControleDlg)
        ON_BN_CLICKED(ID_LEAVE, OnLeave)
        ON_BN_CLICKED(ID_SINGLE, OnSingle)
        ON_BN_CLICKED(ID_CONTINUOUS, OnContinuous)
        ON_WM_CLOSE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CControleDlg message handlers

BOOL CControleDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
```

```
    m_comm.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));
    m_control.SetIcon(LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_WAIT)));

            return TRUE;  // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
  }

  void CControleDlg::OnLeave()
  {
            m_action = LEAVE;
  }

  void CControleDlg::OnSingle()
  {
   m_action = STEP;
  }

  void CControleDlg::OnContinuous()
  {
            if (m_action==NONE)
    {
    m_action = CONT;
    m_bcont.SetWindowText("Interromper");
    }
   else
    {
    m_action = NONE;
    m_bcont.SetWindowText("Contínuo");
    }

  }

  void CControleDlg::OnClose()
  {
            MessageBox("Use o botão de SAIR para fechar essa janela","Ação não permitida",MB_OK);
  }
```

Files DDEOptions.h and DDEOptions.cpp - E-MFG Controller DDE Options Dialog Box

```
// DDEOptions.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDDEOptions dialog

class CDDEOptions : public CDialog
{
// Construction
public:
            CDDEOptions(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
            //{{AFX_DATA(CDDEOptions)
            enum { IDD = IDD_DDE };
            CString    m_app;
            CString    m_item;
            short      m_timeout;
            CString    m_topic;
 BOOL m_peek;
 BOOL m_poke;
 BOOL m_warn;
 BOOL m_log;
            CString    m_arg;
            //}}AFX_DATA

 class CDDESocket* mp_DDESock;

// Overrides
            // ClassWizard generated virtual function overrides
            //{{AFX_VIRTUAL(CDDEOptions)
```

```
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDDEOptions)
        afx_msg void OnTest();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

// DDEOptions.cpp : implementation file
//

#include "stdafx.h"
#include "etna.h"

#include <ddeml.h>                                          //DDE Management Library
#include "ddesock.h"                                    //DDE Socket Definition

#include "DDEOptions.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDDEOptions dialog


CDDEOptions::CDDEOptions(CWnd* pParent /*=NULL*/)
        : CDialog(CDDEOptions::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDDEOptions)
        m_app = _T("");
        m_item = _T("");
        m_timeout = 0;
        m_topic = _T("");
m_peek = TRUE;
m_poke = FALSE;
m_warn = FALSE;
m_log = TRUE;
mp_DDESock = NULL;
        m_arg = _T("");
        //}}AFX_DATA_INIT
}


void CDDEOptions::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDDEOptions)
        DDX_Text(pDX, IDC_APP, m_app);
        DDX_Text(pDX, IDC_ITEM, m_item);
        DDX_Text(pDX, IDC_TIMEOUT, m_timeout);
        DDX_Text(pDX, IDC_TOPIC, m_topic);
        DDX_Check(pDX, IDC_PEEK,m_peek);
        DDX_Check(pDX, IDC_POKE,m_poke);
        DDX_Check(pDX, IDC_WARN,m_warn);
        DDX_Check(pDX, IDC_LOG,m_log);
        DDX_Text(pDX, IDC_ARG, m_arg);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDDEOptions, CDialog)
        //{{AFX_MSG_MAP(CDDEOptions)
```

```
        ON_BN_CLICKED(IDC_TEST, OnTest)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////
// CDDEOptions message handlers

void CDDEOptions::OnTest()
{
 CString param;

 TRY
  {
  UpdateData(TRUE);

  mp_DDESock->m_timeout= (DWORD)m_timeout;

  param = m_app+'!'+m_topic+'!'+m_item;
  if (m_peek)
   {
   if (!mp_DDESock->m_Registered)
    mp_DDESock->RegisterApp("teste");
   mp_DDESock->GetString(param,m_arg);
   }
  else
   {
   if (!mp_DDESock->m_Registered)
    mp_DDESock->RegisterApp("teste");
   mp_DDESock->PutString(param,m_arg);
   }
  }
  CATCH( CUserException, e )
  {
  switch(mp_DDESock->m_error)
   {
   default:
    {
    MessageBox("Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   case 0:
    {
    MessageBox("Ocorreu um erro de comunicação.","Erro de DDE",MB_OK);
    break;
    }
   };
  }
        END_CATCH

 UpdateData(FALSE);
}
```