USP

Campus de São Carlos

EDITOR DE MODELOS DE SISTEMAS DE EVENTOS DISCRETOS, BASEADO EM REDES DE PETRI INTERPRETADAS

JOÃO BORSOI SOARES

Orientador: Prof. Dr. Ricardo Yassushi Inamasu

UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Serviço de Pós-Gradução EESC/USP

EXEMPLAR REVISADO

Data de entrada no Serviço: 04 / 09 / 01

Ass.: Guilherme

EDITOR DE MODELOS DE SISTEMAS DE EVENTOS DISCRETOS, BASEADO EM REDES DE PETRI INTERPRETADAS.

JOÃO BORSOI SOARES

Dissertação apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Mecânica.

ORIENTADOR: Prof. Dr. Ricardo Yassushi Inamasu

DEDALUS - Acervo - EESC

3110003675 4

EESC - USP Biblioteca CAMPUS DE S. CARLOS

São Carlos

2001

Class. TESE-EESC
Cutt. 5701
Tombo T0193/01

nt 120042 7

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca - EESC/USP

S676e

Soares, João Borsoi
Editor de modelos de sistemas de eventos discretos, baseado em redes de Petri interpretadas / João Borsoi Soares. -- São Carlos, 2001.

Dissertação (Mestrado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 2001.
Área : Engenharia Mecânica.
Orientador: Prof. Dr. Ricardo Yassushi Inamasu.

1. Petri interpretada. 2. Software aberto. 3. Framework I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **JOÃO BORSOI SOARES**

Dissertação defendida e julgada em 01-06-2001 perante a Comissão Julgadora:

Prof. Doutor **RICARDO YASSUSHI INAMASU (Orientador)** — Aprovado
(EMBRAPA)

Prof. Associado **JOSÉ CARLOS MALDONADO** — APROVADO
(Instituto de Ciências Matemáticas e Computação/USP)

Prof. Doutor **PAULO EIGI MIYAGI** — APROVADO
(Escola Politécnica - USP)

Prof. Associado **ARTHUR JOSÉ VIEIRA PORTO**
Coordenador do Programa de Pós-Graduação
em Engenharia Mecânica

JOSÉ CARLOS A. CINTRA
Presidente da Comissão de Pós-Graduação da EESC

Aos meus pais, minha irmã
e à minha linda mulher, Ana.

Agradecimentos

Ao Prof. Dr. Ricardo Yassushi Inamasu pela excelente orientação;

Aos colegas do Laboratório de Simulação pelo apoio e incentivo;

À FAPESP pelo apoio financeiro concedido.

Sumário

Lista de Figuras iv

Lista de Tabelas vi

Lista de Abreviaturas e Siglas vii

Lista de Símbolos viii

Resumo x

Abstract xi

Capítulo 1 - Introdução 1

1.1. CONTEXTO E MOTIVAÇÃO 1

1.2. OBJETIVOS 5

1.3. ORGANIZAÇÃO DO TRABALHO 5

Capítulo 2 - Revisão Bibliográfica 6

2.1. CONSIDERAÇÕES INICIAIS 6

2.2. REDE DE PETRI LUGAR/TRANSIÇÃO 6

2.2.1. Elementos estruturais e suas inter-relações 7

2.2.2. Marcação e seu comportamento dinâmico 7

2.2.2.1. Marcação 7

2.2.2.2. Habilitação e disparo de transição 7

2.2.3. Definição 9

2.2.4. Propriedades comportamentais 10

2.2.4.1. Alcançabilidade 10

2.2.4.2. Limitabilidade 10

2.2.4.3. Vivacidade 10

2.2.4.4. Reversibilidade e estado origem 12

2.2.4.5. Abrangência de cobertura 12

2.2.4.6. Persistência 12

2.2.4.7. Distância sincrônica 12

2.2.5. Métodos de análise 13

2.2.5.1. Árvore de alcançabilidade 13

2.2.5.2. Matriz de incidência e equação de estado 14
2.2.5.3. Análise usando invariantes 16
2.3. MARK FLOW GRAPH (MFG) 16
2.3.1. Elementos estruturais e suas inter-relações 17
2.3.2. Marcação e seu comportamento dinâmico 20
2.3.2.1. Marcação 20
2.3.2.2. Habilitação de disparo de transição 20
2.3.2.3. Disparo de transição 21
2.3.3. Definição 22
2.3.4. Propriedades comportamentais 25
2.3.5. Métodos de análise 25
2.3.5.1. Matriz de incidência e equação de estado 25
2.3.5.2. Análise usando invariantes 26
2.3.6. Estudo de caso 27
2.4. MARK FLOW GRAPH E PRODUCTION FLOW SCHEMA 29
2.5. SEQUENTIAL FLOW CHART 30
2.5.1. Elementos estruturais e suas inter-relações 30
2.5.2. Comportamento dinâmico 31
2.5.3. SFC representado como rede de Petri Lugar/Transição 32
Capítulo 3 - Petri Net Tools 2000 34
3.1. CONSIDERAÇÕES INICIAIS 34
3.2. ARQUITETURA ABERTA 35
3.3. ASPECTOS OPERACIONAIS 36
3.3.1. Características genéricas 37
3.3.1.1. Edição do modelo 37
3.3.1.2. Controle de projeto 38
3.3.1.3. Ferramentas de Auxílio à Análise 39
3.3.2. Template para rede de Petri Lugar/Transição 39
3.3.2.1. Rede de Petri modular 39
3.3.2.2. Análise 40
3.3.3. Template para MFG/PFS 43
3.3.3.1. Análise 43
3.3.4. Template para SFC 44
3.3.4.1. Análise 45
3.4. ASPECTOS DE IMPLEMENTAÇÃO 46

3.4.1. Orientação a Objeto *47*
3.4.2. Design Patterns *47*
3.4.3. Frameworks *48*

Capítulo 4 - Conclusões **50**

4.1. TRABALHOS FUTUROS 51

Referências Bibliográficas **53**

Lista de Figuras

Figura 1. Exemplo de rede de Petri Lugar/Transição e disparo 8

Figura 2. Exemplo de rede de Petri segura, não viva, mas estritamente L1-viva (MURATA 1989). 11

Figura 3. Exemplo para verificação da distância sincrônica (MURATA 1989). 13

Figura 4. Exemplo de representação da árvore de alcançabilidade para uma rede de Petri (MURATA 1989). 14

Figura 5. Um exemplo de rede de Petri (MURATA 1989) 15

Figura 6. Tipos de boxes suas correspondências com a rede de Petri Lugar/Transição 17

Figura 7. Arcos orientados e suas correspondências com a rede de Petri Lugar/Transição 18

Figura 8. Portas e boxes comuns com suas correspondências com a rede de Petri Lugar/Transição 19

Figura 9. Portas e demais boxes com suas correspondências com a rede de Petri Lugar/Transição 19

Figura 10. Exemplo de um MFG (MIYAGI, 1993) e sua correspondência como rede de Petri. 20

Figura 11. Exemplo de disparo de uma transição (MIYAGI, 1993) 22

Figura 12. Modelo MFG para o exemplo 27

Figura 13. Exemplo de representação de um SED através do PFS 29

Figura 14. Níveis hierárquicos de uma atividade em PFS junto com elementos MFG 30

Figura 15. Conceitos básicos do SFC 31

Figura 16. Exemplo de representação de um SFC em rede de Petri Lugar/Transição 33

Figura 17. Representação das estruturas divergentes e convergentes do SFC em rede de Petri Lugar/Transição 33

Figura 18. Modelo geral de funcionamento do editor 35

Figura 19. Tela de escolha de *template* 36

Figura 20. Tela do Petri Net Tools 2000 com um template para MFG/PFS 37

Figura 21. Janela do *Object Inspector* 38

Figura 22. Janela de projeto 38

Figura 23. Exemplo de rede de Petri modular 40

Figura 24. Janela de simulação 41

Figura 25. Janela de propriedades comportamentais 41

Figura 26. Exemplo de matriz de incidência e árvore de alcançabilidade 42

Figura 27. Exemplo da análise de invariantes 43

Figura 28. Exemplo de representação de matriz de incidência para o MFG 44

Figura 29. *Template* para SFC 45

Figura 30. Árvore de alcançabilidade para um exemplo de SFC 46

Figura II-1. Estrutura do Método da Fábrica em Design Patterns II-2

Figura II-2. Estrutura da Composição em Design Patterns II-3

Figura II-3. Estrutura do Método Template em Design Patterns II-4

Lista de Tabelas

Tabela 1. Lista de operações para o exemplo..... 27

Lista de Abreviaturas e Siglas

CLP	- Contador Lógico Programável
DLL	- Dynamic Link Library
E-MFG	- Enhanced Mark Flow Graph
FMS	- Flexible Manufacturing System
MDI	- Multiple Document Interface
MFG	- Mark Flow Graph
OMG	- Object Management Group
PFS	- Production Flow Schema
SED	- Sistema de Eventos Discretos
SFC	- Sequential Flow Chart
UML	- Unified Modeling Language

Lista de Símbolos

A - a matriz de incidência

B - um conjunto finito de boxes

B_A - um conjunto finito de boxes agrupadores

B_C - um conjunto finito de boxes comuns

B_D - um conjunto finito de boxes dispersores

B_K - um conjunto finito de boxes capacidade

d_{ij} - a distância sincrônica entre duas transições (i e j)

F - um conjunto de arcos (relação de fluxo)

G_H - o conjunto de portas habilitadoras

G_I - o conjunto de portas inibidoras

K - a função capacidade

$L(M_0)$ - o conjunto de todas possíveis seqüências de disparos de transições a partir de M_0

M - o vetor marcação para uma rede de Petri ou MFG/PFS

M_0 - o vetor marcação inicial para uma rede de Petri ou MFG/PFS

$\bullet p$ - o conjunto de transições de entrada de p

$p \bullet$ - o conjunto de transições de saída de p

P - um conjunto finito de lugares

$R(M_0)$ - o conjunto de todas possíveis marcações alcançáveis a partir de M_0

σ - uma seqüência de disparos de transição

$\overline{\sigma}(t)$ - o número de disparos da transição t na seqüência σ

$\bullet t$ - o conjunto de lugares de entrada de t

$t \bullet$ - o conjunto de lugares de saída de t

T - um conjunto finito de transições

u - um vetor de controle indicando a transição disparada

ω - uma marcação que tende ao infinito

W - a função peso

Resumo

SOARES, J. B. *Editor de modelos de sistemas de eventos discretos, baseado em redes de petri interpretadas*. São Carlos, 2001. 63p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

A rede de Petri consiste numa poderosa ferramenta gráfica e matemática, que tem sido desenvolvida e aprimorada para a modelagem e análise de Sistemas de Eventos Discretos. Uma de suas limitações é a profusão de elementos gráficos para modelos relativamente grandes. Algumas redes de Petri interpretadas consistem em propostas que visam sanar este tipo de problema. Este trabalho tem por objetivo implementar um *framework* de edição e análise gráfica de redes de Petri, a partir de técnicas de reutilização de código como padrões de projeto. Assim, foi proposto um software aberto que viabiliza o processo de contínuo desenvolvimento, promovendo a inclusão, reutilização e aperfeiçoamento de funcionalidades. Como resultado foi possível incorporar a edição de outros tipos de redes de Petri, principalmente as interpretadas, através da reutilização de funcionalidades matemáticas e de edição gráfica da rede de Petri Lugar/Transição.

Palavras-chave: Petri; interpretada; software; aberto; framework.

Abstract

SOARES, J. B. *Event Discrete System model editor, based on interpreted Petri nets.* São Carlos, 2001. 63p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

The Petri net is a powerful graphical and mathematical tool, which has been developed and improved to the modeling and analysis of Discrete Event Systems. One of its limitations is the large profusion of graphic elements in complex models. Some interpreted Petri nets consist in proposals to solve this kind of limitation. This work's main goal is to implement a Petri net graphical editor and analyzer framework, using concepts of code reuse and design patterns. In this way, an open-software was proposed, which enables the continuous development process, providing the inclusion, reuse and improvement of functionalities. As a result, it was possible to incorporate the edition of some Petri net extensions, essentially the interpreted, through the reuse of mathematical and graphical edition functionalities of Place/Transition Petri net.

Keywords: Petri; interpreted; software; open; framework.

Capítulo 1 - Introdução

1.1. Contexto e Motivação

A rede de Petri e suas extensões consistem em ferramentas de modelagem gráfica e matemática, aplicáveis a uma variedade de sistemas, inclusive para a análise e controle de Sistemas de Eventos Discretos (SEDs) (MIYAGI, 1993).

Dentre as diversas aplicações relacionadas, podemos citar a utilização para modelar componentes de Sistema Flexível de Manufatura (FMS) (CECIL et al., 1992) e processos de fabricação (PORTO, 1990). Através dela é possível representar graficamente os estados dinâmicos (estados que podem ser assumidos durante o processo em questão) dos elementos de um sistema e seus eventos relacionados, como intertravamento de máquinas, coordenação de máquinas (robô, Contador Numérico e Veículo Auto-Guiado), seqüenciamento da produção, sincronismo de operações de montagem, etc.

Algumas das características mais interessantes da rede de Petri são (MURATA, 1989):

- Visualização através de representação gráfica;
- Modelagem através de vários níveis de abstração;
- Modelagem de sistemas paralelos e concorrentes;

- Sincronização de eventos;
- Verificação de propriedades de forma sistemática.

Através da rede de Petri é possível abstrair estados e ações de um sistema, e representá-los através de elementos gráficos. Porém, se o sistema possuir relativamente uma grande quantidade de estados e ações inter-relacionadas, como normalmente ocorre nos atuais sistemas de automação, um número equivalente de elementos pode ser necessário para a sua representação. Conseqüentemente, há uma dificuldade em contornar problemas relacionados à profusão de elementos gráficos no modelo.

Uma das alternativas consiste em representar uma quantidade maior de estados através de equações matemáticas discretas, conforme proposto pela rede de Petri colorida (JENSEN, 1980; JENSEN, 1992 e JENSEN, 1994). Outra alternativa é apresentada pela rede de Petri modular (INAMASU, 1995; INAMASU & PORTO, 1997; JENG, 1997; PROTH et al., 1997 e VOLGLER, 1992), que divide um modelo em módulos menores.

As redes de Petri interpretadas, como Sequential Flow Chart (SFC), Mark Flow Graph (MFG), Production Flow Schema (PFS) (BIEREL & ROUSSEL, 2001; FUJINO et al., 2000; MIYAGI et al., 1989; MIYAGI, 1993 e ZAYTOON, 1996), incorporam elementos gráficos diferentes. No entanto, podem ser interpretadas como rede de Petri convencional, possuindo sempre uma representação sistemática correspondente. Assim, elas podem ser vistas como um subconjunto da rede de Petri convencional, muitas vezes restringindo o âmbito de aplicação do problema e também incorporando simplificações gráficas.

PETRI (2001b) apresenta uma relação das principais ferramentas para redes de Petri existentes no mercado, cada qual oferecendo funcionalidades específicas para um âmbito de solução. Por exemplo, algumas dessas ferramentas trabalham com redes de Petri coloridas, outras permitem redes de Petri temporais, redes de Petri para sistemas contínuos, muitas permitem a programação e conseqüente interação com outros ambientes, e assim por diante. SIMÃO (2000) apresenta ainda uma importante contribuição através de uma ferramenta para a validação de redes de Petri baseada na análise de mutantes, que por sua vez, consiste numa inovação

em relação às usuais técnicas de análise implementadas nos demais editores existentes no mercado.

Apesar da relativa facilidade de uso das redes de Petri, implementar softwares de edição, análise e simulação de modelos representados em rede de Petri e suas extensões é uma tarefa árdua. No entanto, existe uma semelhança entre todos os tipos de rede de Petri e extensões que viabiliza esforços em se reutilizar estruturas de implementação.

Do ponto de vista matemático, as redes de Petri interpretadas podem, por exemplo, ser descritas como redes de Petri Lugar/Transição. Desta forma, ferramentas de análise e simulação implementadas para rede de Petri Lugar/Transição podem ser reaproveitadas para as redes de Petri interpretadas. Algumas características relativas aos softwares de edição também podem ser reaproveitadas, como o manuseio e visualização de objetos gráficos, controle de *zoom*, controle de linhas de grade, armazenamento do modelo em arquivos, impressão, etc.

Torna-se assim de significante importância a construção de uma ferramenta de software flexível, que promova o reaproveitamento de código e projeto, e que permita a inclusão de novas funcionalidades, como por exemplo, a implementação de novas propostas de extensões. A esse tipo de ferramenta denomina-se software aberto. Ressalta-se que o conceito de software aberto empregado aqui não está relacionado com o conceito de sistemas independentes de plataforma, denominados sistemas abertos.

Os principais benefícios de um software aberto aplicado ao desenvolvimento de ferramentas para edição e análise de redes de Petri e suas extensões são:

- Promove o processo de contínuo desenvolvimento do software a partir da inclusão e aperfeiçoamento de funcionalidades do software;
- Reduz o tempo e custo de futuras implementações, a partir do reaproveitamento de funcionalidades teoricamente já testadas e validadas;
- Viabiliza a implementação da edição de redes de Petri interpretadas e encoraja o desenvolvimento de futuras propostas baseadas em redes de Petri;

- Viabiliza a integração com sistemas externos;

No campo da engenharia de software, a metodologia de desenvolvimento de sistemas orientados a objeto consiste numa ferramenta para análise, projeto e implementação de sistemas, muito difundida devido às seguintes características (KHOSHAFIAN, 1990):

- Modelagem do mundo real mais próxima à ótica do usuário;
- Facilidade de interação com ambiente computacional, usando metáforas familiares;
- Possibilidade de construção de componentes reutilizáveis de software e facilidade de expandir bibliotecas nos módulos do software;
- Facilidade de implementar modificações e extensões em componentes sem a necessidade de uma exaustiva re-codificação.

A representação do sistema através de objetos, por ser mais próxima à ótica humana, facilita sua interpretação e, conseqüentemente, sua modelagem. A implementação através deste paradigma fica mais simples e apurada em relação a usual técnica de programação estruturada.

A rede de Petri, por ser uma técnica de modelagem formal e bem definida, possui atributos gráficos e matemáticos que beneficiam uma implementação orientada a objeto. Além disso, a orientação a objeto tem sido vista como uma ferramenta de auxílio para promover a reutilização de código e projeto, onde técnicas de utilização de padrões de projeto (*design patterns*) e *frameworks* viabilizam a implementação de softwares abertos (DEUTSCH, 1989; FAYAD, 1999; FAYAD, 2000; GAMMA et a., 1997; GUIMARÃES, 1998; JOHNSON & FOOT, 1988; JOHNSON & RUSSO, 1991 e JOHNSON, 2001).

Para o desenvolvimento de aplicativos orientados a objetos, uma das linguagens disponíveis e mais utilizadas no mercado é o C++ (MEYERS, 1992 e STROUSTRUP, 1991). Esta pode ser considerada uma linguagem de alto nível, mas também pode trabalhar com códigos de mais baixo nível, na busca de algoritmos mais eficientes.

1.2. Objetivos

A ferramenta Petri Net Tools consiste num editor e simulador gráfico de redes de Petri Lugar/Transição, desenvolvido com a metodologia orientada a objetos. Este trabalho tem por objetivo propor uma nova estrutura ampliada sobre esta ferramenta, incorporando características de um software aberto. Viabiliza-se assim o processo de contínuo desenvolvimento, promovendo a inclusão, reutilização e aperfeiçoamento de funcionalidades. Busca-se incorporar a edição de outros tipos de redes de Petri, principalmente as interpretadas, através da reutilização de funcionalidades matemáticas e de edição gráfica da rede de Petri Lugar/Transição.

1.3. Organização do Trabalho

Neste capitulo foram apresentados o contexto, motivação e objetivos deste trabalho.

O Capítulo 2 apresenta uma revisão bibliográfica da rede de Petri Lugar/Transição e efetua uma formalização de algumas técnicas de redes de Petri interpretadas, descritas enquanto rede de Petri Lugar/Transição.

O Capítulo 3 apresenta o editor Petri Net Tools 2000, desenvolvido neste trabalho, abordando detalhes da arquitetura do software, aspectos operacionais e de implementação.

O Capítulo 4 apresenta os resultados e conclusões, obtidos na implementação do editor. Apresentam-se também as principais contribuições deste trabalho e futuras propostas de trabalho.

Capítulo 2 - Revisão Bibliográfica

2.1. Considerações Iniciais

Conforme relatado anteriormente, dentre as metas deste trabalho busca-se propor uma estrutura de software aberto que reutilize as funcionalidades de edição gráfica e de análise da rede de Petri Lugar/Transição, para a incorporar a edição de redes de Petri interpretadas.

Este capítulo inicialmente apresenta uma revisão da técnica de rede de Petri Lugar/Transição. Posteriormente efetua-se uma formalização das técnicas do Mark Flow Graph (MFG), Production Flow Schema (PFS) e Sequential Flow Chart (SFC), descritas enquanto rede de Petri Lugar/Transição.

2.2. Rede de Petri Lugar/Transição

A rede de Petri foi introduzida em 1962 por Carl Adam Petri, e consiste numa ferramenta gráfica e matemática extremamente efetiva para a modelagem e análise de Sistemas de Eventos Discretos (SEDs).

Além de permitir a modelagem e representação gráfica de uma variedade de sistemas, a rede de Petri possui uma formalização matemática bem definida e técnicas de análise já consagradas, fundamentais na validação do modelo. Dentre as técnicas existentes, pode se destacar a verificação de propriedades comportamentais do modelo, determinação da árvore de alcançabilidade, uso da

matriz de incidência e equação de estado, análise de invariantes, entre outras (AL-JAAR & DESROCHERS, 1995 e MURATA, 1989).

2.2.1. Elementos estruturais e suas inter-relações

A rede de Petri consiste num grafo direcionado, com peso e bipartido, composto por dois elementos estruturais: *lugares* e *transições*. O lugar é representado graficamente por um círculo e a transição por uma barra. Os elementos estruturais são utilizados para criar o modelo, no qual arcos orientados conectam lugares a transições e transições a lugares. Estes arcos podem ser rotulados com um valor inteiro positivo, indicando seu peso. Um arco de peso *k* pode ser interpretado como *k* arcos paralelos.

2.2.2. Marcação e seu comportamento dinâmico

2.2.2.1. Marcação

Cada lugar pode possuir marcas (fichas), indicando um estado. A marcação do sistema (estado), é denotada por um vetor M de dimensão igual ao número de lugares do modelo. O *p*-ésimo componente de M, indicado por $M(p)$ consiste no número de fichas do lugar p. O vetor M_0 consiste no vetor de marcação inicial, isto é, estado inicial do sistema.

Na modelagem, usando o conceito de condições e eventos, o lugar pode ser interpretado como uma condição e a transição como um evento. Uma transição (evento) possui um certo número de lugares de entrada e de saída, representando as pré e pós-condições, respectivamente. A presença de uma ficha num lugar indica que a condição deste é verdadeira.

O comportamento de diversos sistemas pode ser descrito em temos de seus estados e de sua respectiva mudança ou transição para outros possíveis estados. Na rede de Petri, a mudança de estado ocorre de acordo com a regra de habilitação e disparo das transições.

2.2.2.2. Habilitação e disparo de transição

A ocorrência de um evento é denominada disparo de transição. A regra de habilitação e disparo de transições é:

- Uma transição está habilitada para disparar se todo lugar de entrada, que possuir arco para a transição, possuir um número de marcas maior ou igual ao peso do arco;

- Uma transição habilitada pode disparar ou não, dependendo se o evento realmente ocorrer;

- No disparo de uma transição habilitada, todo lugar que possui um arco para a transição tem seu número de marcas reduzido pelo valor do peso deste arco, e todo lugar que possui um arco vindo da transição tem seu número de marcas acrescido do valor do peso deste arco.

Uma transição sem lugares de entrada é denominada *transição fonte*, e a sem lugares de saída é chamada de *transição sumidouro*. Uma transição fonte está sempre habilitada, e o disparo de uma transição sumidouro consome fichas, mas não produz nenhuma.

A Figura 1 esboça um exemplo de modelo feito em rede de Petri, representando a montagem de uma peça a partir de outras duas peças ("peça 1" e "peça 2"). Segundo o modelo, para montar uma peça são utilizadas duas peças do tipo "1" e uma peça do tipo "2".

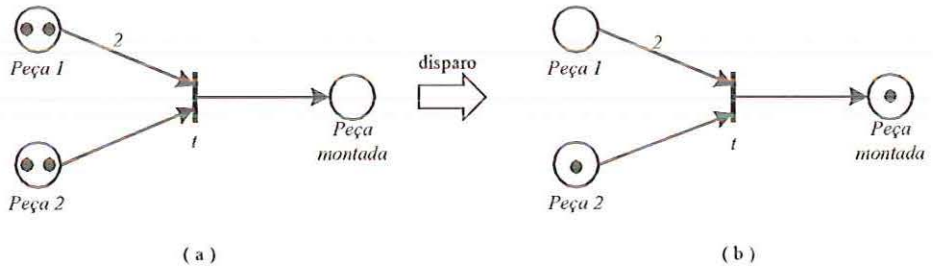

Figura 1. Exemplo de rede de Petri Lugar/Transição e disparo

Os lugares "peça 1" e "peça 2" representam, respectivamente, o número de peças do tipo "1" e do tipo "2" disponíveis. De forma análoga o lugar "peça montada" representa o número de peças montadas. A transição "t" indica o evento de montagem.

A Figura 1 mostra o modelo em dois estados distintos. A Figura 1(a) representa o estado inicial, no qual existem duas peças do tipo 1 e duas peças do tipo 2, e nenhuma peça montada. A Figura 1(b) representa o estado do sistema após a

montagem de uma peça, indicando o disparo da transição "t" a partir do estado inicial.

2.2.3. Definição

A rede de Petri é uma quíntupla, $PN = (P,T,F,W,M_0)$ onde (MURATA, 1989):

$P = \{p_1, p_2, ..., p_m\}$ é um conjunto finito de lugares,

$T = \{t_1, t_2, ..., t_n\}$ é um conjunto finito de transições,

$F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos (relação de fluxo),

$W : F \rightarrow \{1,2,3,...\}$ é a função peso,

$M_0 : P \rightarrow \{0,1,2,3,...\}$ é a marcação inicial,

$P \cap T = \phi$, $P \cup T \neq \phi$,

A estrutura da rede de Petri $N = (P,T,F,W)$ sem qualquer marcação inicial específica é denotada por N.

Uma rede de Petri com uma dada marcação inicial é denotada por (N,M_0).

Além disto:

$M(p_i)$ é a marcação $\in M$ em p_i,

$\bullet p = \{t \mid (t,p) \in F\}$ é o conjunto de transições de entrada de p,

$p\bullet = \{t \mid (p,t) \in F\}$ é o conjunto de transições de saída de p,

$\bullet t = \{p \mid (p,t) \in F\}$ é o conjunto de lugares de entrada de t,

$t\bullet = \{p \mid (t,p) \in F\}$ é o conjunto de lugares de saída de t,

$R(N,M_0)$ ou simplesmente $R(M_0)$ é o conjunto de todas possíveis marcações alcançáveis a partir de M_0 e

$L(N,M_0)$ ou simplesmente $L(M_0)$ é o conjunto de todas possíveis seqüências de disparos de transições a partir de M_0.

2.2.4. Propriedades comportamentais

As propriedades comportamentais da rede de Petri indicam como o sistema se comporta a partir da marcação inicial. Segue abaixo algumas propriedades comportamentais relacionadas em MURATA (1989).

2.2.4.1. Alcançabilidade

A alcançabilidade é base fundamental para o estudo de propriedades dinâmicas de qualquer sistema, e consiste em determinar se um dado estado M_n é alcançável a partir de uma seqüência de disparos de transições, numa rede (N,M_0). Isto é, determinar se $M_n \in R(M_0)$. Esta seqüência de disparos é denotada por $\sigma = M_0 t_1 M_1 t_2 M_2 \cdots t_n M_n$ ou simplesmente por $\sigma = t_1\ t_2 \cdots t_n$.

2.2.4.2. Limitabilidade

Uma rede de Petri (N,M_0) é dita ser *k-limitada* ou simplesmente *limitada* se o número de marcas em cada lugar não exceder um número finito *k* para qualquer marcação alcançável de M_0, isto é, $M(p) \leq k$, para cada lugar p e para cada marcação $M \in R(M_0)$. Uma rede de Petri é dita ser *segura* se é 1-limitada.

Os lugares em rede de Petri são muitas vezes usados para representar *buffers* ou registros para armazenamento de dados de intermédio. Verificar se uma rede é limitada ou segura, garante que não haverá estouro nos *buffers* ou registros (*overflows*), independentemente da seqüência sendo disparada.

2.2.4.3. Vivacidade

Uma rede de Petri (N,M_0) é tida como *viva* se para qualquer marcação $M \in R(M_0)$, existir sempre pelo menos uma transição habilitada para disparo. A vivacidade é uma propriedade desejada para muitos sistemas. No entanto, em muitos casos ela é de difícil caracterização, como por exemplo, na modelagem de

sistemas operacionais de sistemas computacionais robustos. Neste sentido são definidos diferentes níveis de vivacidade. Assim, uma transição *t* numa rede de Petri (N,M_0) é dita ser:

0) *morta* (*L0-viva*) se *t* nunca pode ser disparada em qualquer seqüência de disparo em $L(M_0)$;

1) *L1-viva* (*potencialmente disparável*) se *t* pode ser disparada pelo menos uma vez em alguma seqüência de disparo em $L(M_0)$;

2) *L2-viva* se, dado um inteiro positivo *k*, *t* pode ser disparado pelo menos *k* vezes em alguma seqüência de disparo em $L(M_0)$;

3) *L3-viva* se *t* aparece muitas vezes em alguma seqüência de disparo em $L(M_0)$;

4) *L4-viva* ou *viva* se *t* é L1-viva para cada marcação $M \in R(M_0)$.

Uma rede de Petri (N,M_0) é dita com *Lk-viva* se cada transição na rede for *Lk-viva*, para *k = 0, 1, 2 ,3 ,4*.

A Figura 2 apresenta uma rede de Petri não viva, uma vez que nenhuma transição fica habilitada para disparo se a transição t_1 for disparada inicialmente. Mas ela pode também ser classificada como estritamente L1-viva.

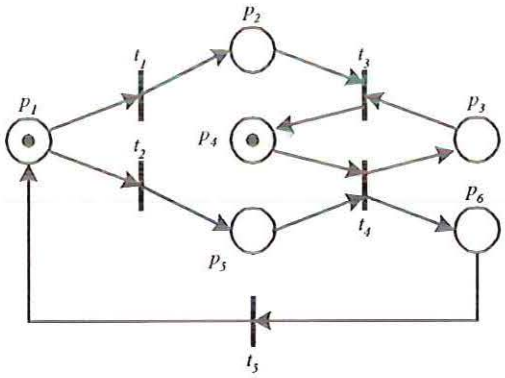

Figura 2. Exemplo de rede de Petri segura, não viva, mas estritamente L1-viva (MURATA 1989).

2.2.4.4. Reversibilidade e estado origem

Uma rede de Petri (N,M_0) é dita *reversível* se, para cada $M \in R(M_0)$, M_0 é alcançável de M. Desta forma, numa rede reversível é sempre possível retornar ao estado inicial. Em muitas aplicações não é necessário retornar ao estado inicial, sendo suficiente retornar a algum estado (origem). Uma marcação M' é dita *estado origem* se, para cada $M \in R(M_0)$, M' é alcançável de M.

2.2.4.5. Abrangência de cobertura

A marcação M numa rede de Petri (N,M_0) é dita ser *abrangível de cobertura* se existir uma marcação $M' \in R(M_0)$ tal que $M'(p) \geq M(p)$ para cada p da rede. Se uma marcação M for a marcação mínima necessária para o disparo de uma transição t, então t é morta (não L1-viva) se e somente se M não for abrangível de cobertura.

2.2.4.6. Persistência

Uma rede de Petri (N,M_0) é dita *persistente* se, para qualquer duas transições habilitadas, o disparo de uma não desabilita a outra. Uma transição numa rede persistente, uma vez habilitada, continuará habilitada até seu disparo.

2.2.4.7. Distância sincrônica

O conceito de distância sincrônica está relacionado a um grau de dependência mútua entre dois eventos em um sistema condição/evento. A distância sincrônica entre duas transições t_i e t_j numa Rede de Petri (N,M_0) é definida por:

$$d_{ij} = \max\left|\overline{\sigma}(t_i) - \overline{\sigma}(t_j)\right|,$$

onde σ é uma seqüência de disparo a partir de alguma marcação $M \in R(M_0)$ e $\overline{\sigma}(t_i)$ é o número de vezes que a transição t_i dispara em σ. Por exemplo, a Figura 3 mostra uma rede na qual $d_{12} = 1$, $d_{34} = 1$ e $d_{13} = \infty$.

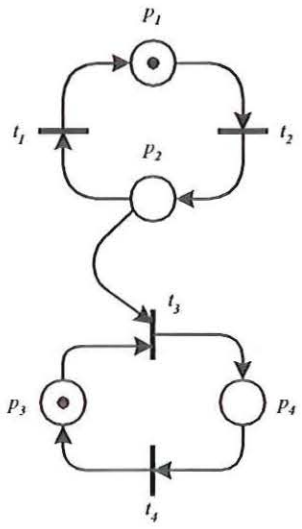

Figura 3. Exemplo para verificação da distância sincrônica (MURATA 1989).

2.2.5. Métodos de análise

A partir de uma análise do modelo pode-se determinar todos os estados que o sistema pode assumir, e como estes estados alternam-se entre si de acordo com eventos relacionados ao sistema. Desta forma, consistem em ferramentas fundamentais na validação do modelo.

2.2.5.1. Árvore de alcançabilidade

A árvore de alcançabilidade descreve todos os possíveis estados do sistema e a seqüência de disparos para alcançá-los. Dada uma Rede de Petri (N, M_0), a partir da marcação inicial M_0 pode-se obter outras marcações de acordo com o disparo das transições habilitadas. Esse processo pode ser interpretado como uma árvore de marcações. Cada nó representa uma marcação a partir de M_0 (nó raiz), e cada arco (galho) representa o disparo de uma transição.

Em redes onde o número de marcas em um lugar excede um número finito, isto é, redes não limitadas, essa representação em árvore cresceria infinitamente. Assim, introduz-se o símbolo ω, representando uma marcação que tende ao infinito. Dessa forma, para cada inteiro n, $\omega > n$, $\omega \pm n = \omega$ e $\omega \geq \omega$.

A Figura 4 exibe um exemplo de representação da árvore de alcançabilidade para uma rede de Petri. Nota-se que é introduzido o conceito de *dead-end*, indicando

uma marcação no qual não há transições habilitadas, e *old*, indicando que a marcação já existe em algum nó de nível inferior da árvore (mais próximo da raiz).

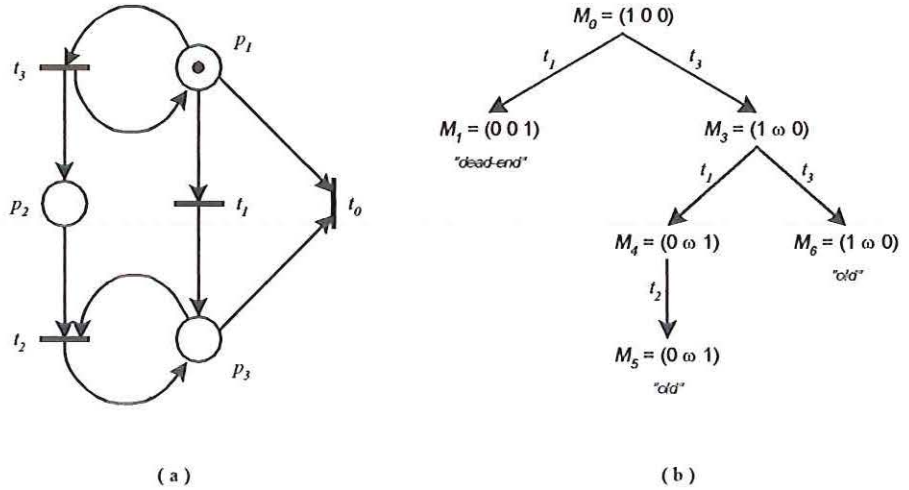

Figura 4. Exemplo de representação da árvore de alcançabilidade para uma rede de Petri (MURATA 1989).

2.2.5.2. Matriz de incidência e equação de estado

A *matriz de incidência* e a *equação de estado* possibilitam a análise algébrica do comportamento dinâmico das redes de Petri.

Para uma Rede de Petri (N, M_0) composta por n transições e m lugares, a matriz de incidência $A = [a_{ij}]$ é uma matriz $n \times m$ de inteiros onde,

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-},$$

$a_{ij}^{+} = W(i, j)$ é o peso do arco da transição i para o lugar j e

$a_{ij}^{-} = W(j, i)$ é o peso do arco do lugar j para a transição i.

Observa-se que a_{ij}^{+}, a_{ij}^{-} e a_{ij} representam, respectivamente, o número de marcas adicionadas, removidas e alteradas no lugar j a partir do disparo da transição i. Uma transição i está habilitada para disparo numa marcação M se e somente se:

$$a_{ij}^{-} \leq M(j),\ j = 1, 2, \ldots, m.$$

A equação de estado indica a alteração de estado (marcação) numa rede de Petri, após o disparo de uma transição. Para uma Rede de Petri (N, M_0) composta por n transições, m lugares e uma matriz de incidência A, define-se a equação de estado como:

$M' = M + A^T u$ onde,

M e M' representam, respectivamente, a marcação antes e depois do disparo;

A é a matriz de incidência;

u é um vetor coluna $n \times 1$ de controle, de $n-1$ posições de valor zero e uma entrada de valor um, indicando a transição disparada.

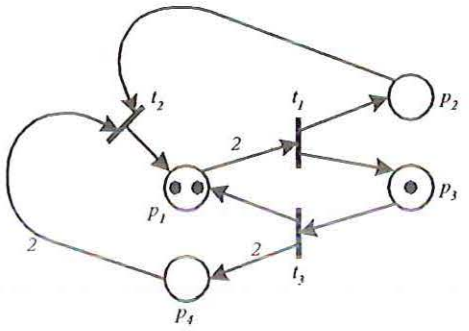

Figura 5. Um exemplo de rede de Petri (MURATA 1989).

Por exemplo, considerando a rede de Petri da Figura 5, com marcação inicial $M_0 = (2 \quad 0 \quad 1 \quad 0)^T$, a equação de estado pode ser utilizada para determinar qual a marcação da rede, a partir da e do disparo da transição t_3:

$$M' = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Considerando a equação de estado e uma seqüência de d disparos a partir da marcação inicial M_0, pode-se escrever a seguinte equação:

$$M_f = M_0 + A^T \sum_{k=1}^{d} u_k$$

2.2.5.3. Análise usando invariantes

A análise de rede de Petri usando invariantes consiste em determinar os *S-invariantes* e *T-invariantes* (AL-JAAR & DESROCHERS, 1995 e MURATA, 1989). Considerando uma Rede de Petri (N, M_0) composta por n transições, m lugares e uma matriz de incidência A, definem-se:

- Um *S-invariante* consiste num vetor y $(m \times 1)$, de inteiros positivos, de forma que $Ay = 0$, e assim, $M^T y = M_0^T y$, para qualquer $M \in R(M_0)$. Esta equação implica que o total de marcas iniciais em M_0, pesado pelo *S-invariante*, é constante para qualquer seqüência de disparo.
- Um *T-invariante* consiste num vetor x $(n \times 1)$, de inteiros positivos, de forma que $A^T x = 0$, e assim, $M_0 = M_f$ para x sendo a seqüência de disparos. Esta equação implica que, se o vetor de disparo for um *T-invariante*, então o sistema retorna para o estado inicial após os disparos.

Um exemplo de aplicação da análise de invariantes é feito posteriormente num estudo de caso para o MFG na Seção 2.3.6.

2.3. Mark Flow Graph (MFG)

O MFG (MIYAGI et al., 1989; MIYAGI, 1993), desenvolvido no *Tokyo Institute of Technology* (Japão), consiste numa versão da rede de Petri própria para aplicação em modelagem, análise e controle de SEDs. GUSTIN (1999) e JUNQUEIRA (2001) apresentam exemplos e aplicações bem detalhadas do MFG, abrangendo algumas outras derivações interessantes como o Enhanced Mark Flow Graph (E-MFG), E-MFG orientado a objetos e MFG Auto-Modificável.

O MFG pode ser visto como uma interpretação de rede de Petri, possibilitando assim que as técnicas de análise já consagradas possam ser abordadas sobre ele, como por exemplo, a verificação de propriedades comportamentais do modelo,

determinação da árvore de alcançabilidade, uso da matriz de incidência e equação de estado, análise de invariantes, entre outras.

Nesta seção é introduzido o MFG descrito enquanto uma rede de Petri Lugar/Transição, visando sua implementação no editor Petri Net Tools. Neste contexto são apresentados alguns métodos de análise (matriz de incidência, equação de estado e análise usando invariantes) e um estudo de caso a fim de comprovar a reutilização das técnicas de análise da rede de Petri.

2.3.1. Elementos estruturais e suas inter-relações

O MFG é um grafo bipartido direcionado e composto dos seguintes elementos estruturais:

- *Box*: indica uma condição. É representado por um par de lugares, de forma a controlar o número de marcas do box. A Figura 6 exibe os quatro tipos de boxes existentes, e suas respectivas correspondências com a rede de Petri Lugar/Transição.

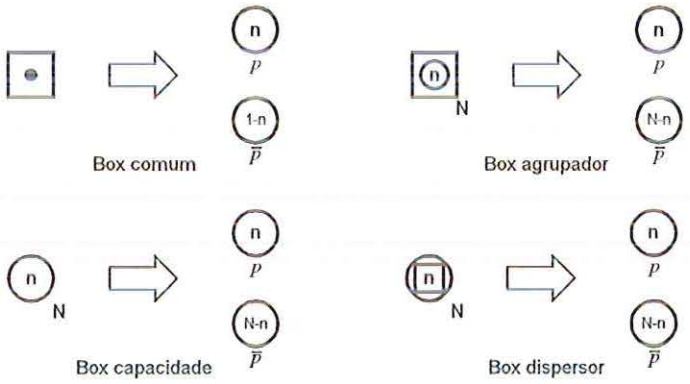

Figura 6. Tipos de boxes suas correspondências com a rede de Petri Lugar/Transição.

O box comum pode possuir apenas no máximo uma marca, diferentemente dos demais, que possuem capacidade *N*. A diferença entre o box capacidade e os boxes agrupador e dispersor é respectiva à dinâmica do MFG e será descrita posteriormente;

- *Transição:* indica um evento e é representado por uma barra vertical. Possui correspondência direta com a transição existente na rede de Petri Lugar/Transição;

- *Arco orientado:* conecta boxes e transições para indicar a relação entre uma condição e os pré e pós-eventos que o definem. Arcos que possuem como origem um determinado box ou transição consistem em seus arcos de saída. De forma análoga, arcos que possuem como destino um determinado box ou transição consistem em seus arcos de entrada. A Figura 7 mostra os possíveis arcos entre boxes e transições, e suas correspondência com a rede de Petri Lugar/Transição;

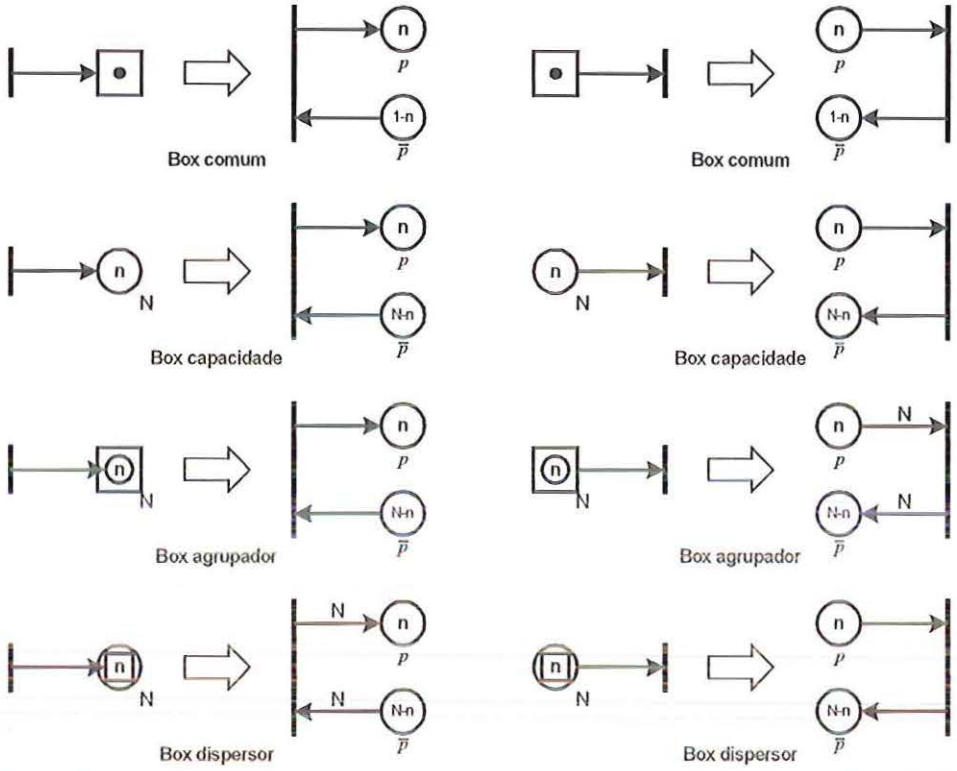

Figura 7. Arcos orientados e suas correspondências com a rede de Petri Lugar/Transição.

- *Porta:* habilita ou inibe a ocorrência dos eventos correspondentes às transições de acordo com sua natureza. Podem ser sub-classificadas em porta externa ou porta interna quando a origem do sinal for um box ou um dispositivo externo, respectivamente. Quando a origem do sinal for um box comum, a porta habilita ou inibe uma transição, se existir marca no box de origem. A Figura 8 exibe as portas e boxes comuns com suas correspondências com a rede de Petri Lugar/Transição

Figura 8. Portas e boxes comuns com suas correspondências com a rede de Petri Lugar/Transição.

Para os demais tipos de boxes, a condição para que uma porta habilite ou iniba uma transição pode se dar pela existência de alguma marca no box (indicado por *n*) ou quando o box possui o número máximo de marcas (indicado por *N*). A Figura 9 detalha o funcionamento das portas para os demais tipos de boxes, através de sua correspondência com a rede de Petri Lugar/Transição.

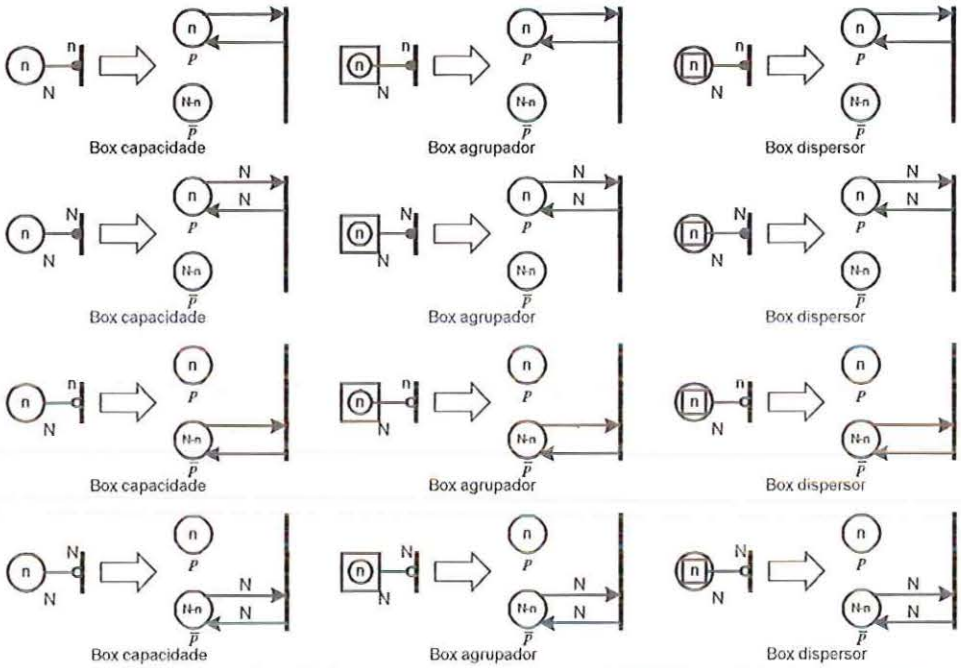

Figura 9. Portas e demais boxes com suas correspondências com a rede de Petri Lugar/Transição.

Uma porta externa habilita ou inibe a transição quando o sinal de origem do dispositivo externo for "1". Para a análise, considera-se que os elementos externos não fazem parte do modelo, e assim, nunca influenciam de forma a desabilitar o disparo de uma transição. Os casos em que uma porta externa desabilita uma transição fazem parte da árvore de alcançabilidade do modelo proposto.

- *Arco de sinal de saída:* este arco envia um sinal binário do box para os dispositivos externos do grafo. Quando houver marca no box de origem, o sinal de saída é "1", quando não houver marca, é "0". Não há uma representação em rede de Petri para o sinal de saída. Desta forma o próprio box de origem representa a saída do sistema.

Os boxes e as transições são necessariamente conectados de forma intercalada através de arcos orientados. Não existe limite para o número de arcos que entram ou saem dos boxes e das transições, mas num par transição-box, ou num par transição-origem do sinal externo, pode existir no máximo apenas um arco entre estes elementos.

Um exemplo de um MFG e sua correspondência com rede de Petri pode ser visto abaixo na Figura 10.

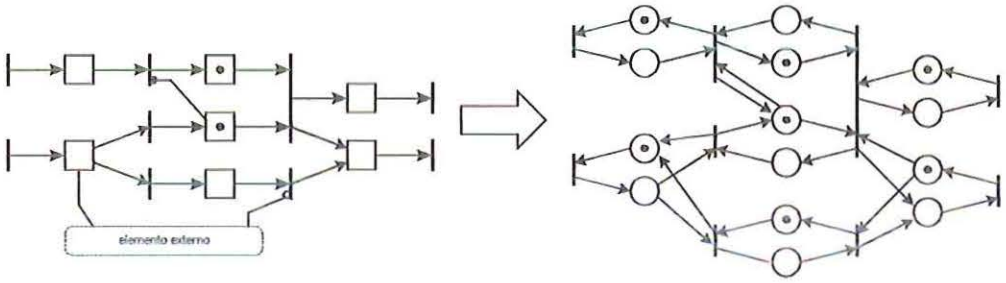

Figura 10. Exemplo de um MFG (MIYAGI, 1993) e sua correspondência como rede de Petri.

2.3.2. Marcação e seu comportamento dinâmico

2.3.2.1. Marcação

Assim como na rede de Petri, o estado de um sistema pode ser representado pelo arranjo das marcas no grafo (marcação).

A partir de uma marcação inicial, o comportamento dinâmico do sistema é representado pela alteração dos estados causada pela ocorrência de eventos. Os eventos consistem no disparo das transições.

2.3.2.2. Habilitação de disparo de transição

Uma transição está habilitada para disparo se e somente se:

EESC - USP Biblioteca São Carlos

- Não existe box comum ou box dispersor com marcas no lado de saída da transição;
- Não existe box agrupador com o número de marcas igual a sua capacidade ($n = N$) no lado de saída da transição;
- Não existe box comum ou box dispersor sem marcas no lado de entrada da transição;
- Não existe box agrupador com o número de marcas diferente ao de sua capacidade ($n \neq N$) no lado de entrada da transição;
- Não existe arco habilitador interno que esteja no estado de desabilitação;
- Não existe arco inibidor interno que esteja no estado de inibição.

2.3.2.3. Disparo de transição

Uma transição é denominada "disparável" se ela está habilitada e não possui:

- Nenhuma porta habilitadora externa no estado de desabilitação, e também;
- Nenhuma porta inibidora externa no estado de inibição.

Se uma transição é disparável, ela dispara imediatamente, com exceção de certos casos que envolvem conflito e atrasos de tempo (MIYAGI, 1993).

No disparo, no lado de entrada da transição, as marcas dos boxes comuns e agrupadores desaparecem, e nos boxes dispersores são reduzidas de uma unidade. Imediatamente, na saída da transição, os boxes comuns e agrupadores são acrescidos de uma marca, e o box dispersor é acrescido de N marcas, onde N é a sua capacidade. A Figura 11 esboça um exemplo de disparo de uma transição.

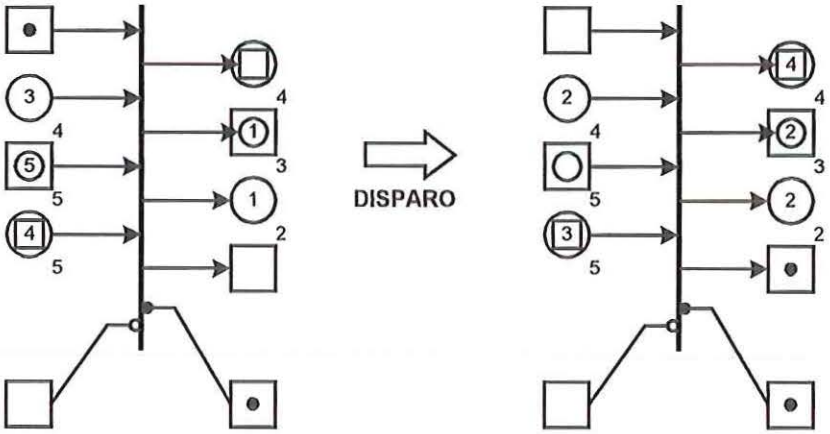

Figura 11. Exemplo de disparo de uma transição (MIYAGI, 1993)

No MFG os disparos são discretos no tempo, existindo uma relação de precedência entre eles. Desta forma, é impossível que duas transições sejam disparadas simultaneamente. Esta ordem de precedência é denominada seqüência de disparo, e está baseada na seqüência temporal.

2.3.3. Definição

O MFG é representado como um caso particular da rede de Petri, caracterizado também como uma quíntupla:

$$MFG = (B, T, F, W, M_0) \subseteq PN = (P, T, F, W, M_0)$$

onde,

$B = P_N \cup P_L \subseteq P$ é um conjunto finito de $2i$ lugares onde,

$P_N = \{p_1, p_2, ..., p_i\}$ é um conjunto finito de i lugares,

$P_L = \{\overline{p}_1, \overline{p}_2, ..., \overline{p}_i\}$ é um conjunto finito de lugares limitadores correspondentes para cada $p_i \in P_N$,

$b_x = (p_x, \overline{p}_x)$ é um box para $p_x \in P_N$, $\overline{p}_x \in P_L$ e $0 \leq x \leq i$;

$T = \{t_1, t_2, ..., t_j\}$ é um conjunto finito de j transições;

$F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos orientados (relação de fluxo), no qual

$F = F_A \cup G_H \cup G_I$ e $F_A \cap G_H \cap G_I = \phi$;

$F_A \subseteq (B \times T) \cup (T \times B)$ é um conjunto de arcos de box ou simplesmente arcos onde

$f_{(t_m,b)} = (f_{(t_m,p)}, f_{(\overline{p},t_n)})$ é o arco de entrada do box $b = (p, \overline{p})$ e

$f_{(b,t_n)} = (f_{(p,t_n)}, f_{(t_n,\overline{p})})$ é o arco de saída do box $b = (p, \overline{p})$,

sendo $t_m \in p\bullet$ e $t_n \in \bullet p$;

$G_H \subseteq (B \times T)$ é o conjunto de portas habilitadoras onde

$g_{H(t_m,b)} = (f_{(t_m,p)}, f_{(p,t_n)})$ é a porta habilitadora do box $b = (p, \overline{p})$,

sendo $t_m \in p\bullet$ e $t_m \in \bullet p$;

$G_I \subseteq (B \times T)$ é o conjunto de portas inibidoras onde

$g_{I(t_m,b)} = (f_{(t_m,\overline{p})}, f_{(\overline{p},t_n)})$ é a porta inibidora do box $b = (p, \overline{p})$,

sendo $t_m \in \overline{p}\bullet$ e $t_m \in \bullet\overline{p}$;

$W : F \rightarrow \{1,2,3,...\}$ é a função peso onde,

$W(b,t) = W(p,t) = W(t,\overline{p})$ para $b = (p,\overline{p})$ e $(b,t) \in F_A$,

$W(t,b) = W(t,p) = W(\overline{p},t)$ para $b = (p,\overline{p})$ e $(t,b) \in F_A$,

$W(b,t) = W(p,t) = W(t,p)$ para $b = (p,\overline{p})$ e $(b,t) \in G_H$,

$W(b,t) = W(\overline{p},t) = W(t,\overline{p})$ para $b = (p,\overline{p})$ e $(b,t) \in G_I$;

$M_0 : B \rightarrow \{0,1,2,...\}$ é a marcação inicial onde

$M(b) = M(p)$, para $b = (p,\overline{p})$.

Neste contexto definem-se:

$K : B \rightarrow \{1,2,3,...\}$ é a função capacidade dos boxes, sendo que

$K(b) = M(p) + M(\overline{p})$, para $b \in B$;

B_C é um conjunto finito de boxes comuns no qual, para $b \in B_C$

$K(b) = 1$,

$W(b,t) = 1$ para $\forall t \in b\bullet$,

$W(t,b) = 1$ para $\forall t \in \bullet b$;

B_K é um conjunto finito de boxes capacidade no qual, para $b \in B_K$

$K(b) = n \mid n \in \mathrm{N}^+$,

$W(b,t) = 1$ para $\forall t \in b\bullet$ e $(b,t) \in F_A$,

$W(t,b) = 1$ para $\forall t \in \bullet b$ e $(t,b) \in F_A$,

$W(b,t) = \{1, K(b)\}$ para $\forall t \in b\bullet$ e $(b,t) \in G_H$,

$W(t,b) = \{1, K(b)\}$ para $\forall t \in \bullet b$ e $(t,b) \in G_I$;

B_D é um conjunto finito de boxes dispersores no qual, para $b \in B_D$

$K(b) = n \mid n \in \mathrm{N}^+$,

$W(b,t) = 1$ para $\forall t \in b\bullet$ e $(b,t) \in F_A$,

$W(t,b) = K(b)$ para $\forall t \in \bullet b$ e $(t,b) \in F_A$,

$W(b,t) = \{1, K(b)\}$ para $\forall t \in b\bullet$ e $(b,t) \in G_H$,

$W(t,b) = \{1, K(b)\}$ para $\forall t \in \bullet b$ e $(t,b) \in G_I$;

B_A é um conjunto finito de boxes agrupadores no qual, para $b \in B_A$

$K(b) = n \mid n \in \mathrm{N}^+$,

$W(b,t) = K(b)$ para $\forall t \in b\bullet$ e $(b,t) \in F_A$,

$W(t,b) = 1$ para $\forall t \in \bullet b$ e $(t,b) \in F_A$,

$W(b,t) = \{1, K(b)\}$ para $\forall t \in b\bullet$ e $(b,t) \in G_H$,

$W(t,b) = \{1, K(b)\}$ para $\forall t \in \bullet b$ e $(t,b) \in G_I$;

2.3.4. Propriedades comportamentais

Assim como para a rede de Petri, as propriedades comportamentais de modelos em MFG indicam como o sistema se comporta a partir da marcação inicial. Conforme apresentado, o MFG pode ser considerado num subconjunto da rede de Petri e, desta forma, as mesmas propriedades comportamentais apresentadas na Seção 2.2.4 se aplicam para o MFG, com uma única observação em relação a limitabilidade, uma vez que os modelos em MFG sempre são limitados.

2.3.5. Métodos de análise

Os métodos de análise para rede de Petri vistos na Seção 2.2.5 podem ser reaproveitados para o MFG, sendo este considerado como um subconjunto da rede de Petri. Segue abaixo a descrição da matriz de incidência e equação de estado para o MFG, considerando a representação adotada.

2.3.5.1. Matriz de incidência e equação de estado

A matriz de incidência é definida para o MFG como uma matriz $A\ (n \times 2m)$ de inteiros, no qual n é o número de transições e m é o número de boxes. Sendo $A = [a_{ij}]$, para $1 \leq j \leq m$, tem-se:

- $a_{ij} = 1$ se existir um arco da transição t_i para o box b_j, sendo este um box comum, box capacidade ou box agrupador;

- $a_{ij} = -1$ se existir um arco do box b_j para a transição t_i, sendo b_j um box comum, box capacidade ou box dispersor;
- $a_{ij} = K(b_j)$ se existir um arco da transição t_i para o box b_j, sendo este um box dispersor;
- $a_{ij} = -K(b_i)$ se existir um arco do box b_j para a transição t_i, sendo b_j um box agrupador.

Para $m \leq j \leq 2m$, $a_{ij} = -a_{ik}$ onde $k = j - m$.

A mudança de estado do MFG pode ser descrita pelas mesmas equações da rede de Petri apresentadas na Seção 2.2.5.2. Desta forma, para um modelo em MFG com m boxes e n transições, tem-se:

$M' = M + A^T u$ onde,

M e M' representam, respectivamente, a marcação antes e depois do disparo;

A é a matriz de incidência;

u é um vetor coluna $n \times 1$ de controle, de $n-1$ posições de valor zero e uma entrada de valor um, indicando a transição disparada.

Considerando a equação de estado e uma seqüência de d disparos a partir da marcação inicial M_0, pode-se escrever a seguinte equação:

$$M_f = M_0 + A^T \sum_{k=1}^{d} u_k$$

2.3.5.2. Análise usando invariantes

A análise de MFG usando invariantes consiste em determinar os *B-invariantes* e *T-invariantes*. Considerando um MFG com marcação inicial M_0, composto por n transições, m boxes e uma matriz de incidência A, definem-se:

- Um *B-invariante* consiste num vetor y $(2m \times 1)$, de inteiros positivos, de forma que $Ay = 0$, e assim, $M^T y = M_0^T y$, para qualquer $M \in R(M_0)$. Esta equação implica que o total de marcas iniciais em M_0, pesado pelo *B-invariante*, para qualquer seqüência de disparo.

- Um *T-invariante* consiste num vetor x $(n \times 1)$, de inteiros positivos, de forma que $A^T x = 0$, e assim, $M_0 = M_f$ para x sendo a seqüência de disparos. Esta equação implica que, se o vetor de disparo for um *T-invariante*, então o sistema retorna para o estado inicial após os disparos.

2.3.6. Estudo de caso

Suponha um FMS com duas máquinas capazes de produzir dois tipos de peças, chamados P_1 e P_2. O tipo de peça P_1 deve ter sua primeira operação realizada na máquina M_1, e a segunda operação pode ser realizada na máquina M_1 ou M_2. Já o tipo de peça P_2 só necessita de uma operação a ser realizada na máquina M_1. A Tabela 1 esboça esta lista de operações.

Tabela 1. Lista de operações para o exemplo

	Operação 1	Operação 2
P_1	M_1	M_1 ou M_2
P_2	M_1	-

Deseja-se produzir n_1 peças de P_1 e n_2 peças de P_2. A Figura 12 exibe o modelo MFG para este exemplo.

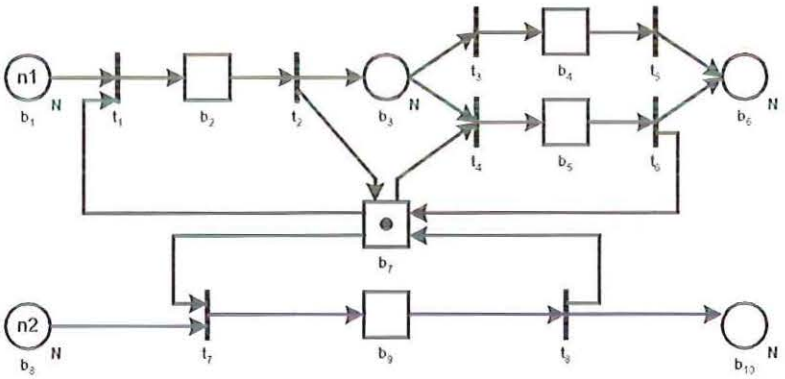

Figura 12. Modelo MFG para o exemplo

Neste modelo o box b_7 indica a disponibilidade da máquina M_1; b_1 (b_8) representa as peças P_1 (P_2) disponíveis para a primeira operação; b_2 (b_9) representa a primeira

operação da peça P_1 (P_2), na máquina M_1; b_3 representa a peça P_1 disponível para a segunda operação; b_4 representa a segunda operação da peça P_1 na máquina M_2, e b_5 representa a mesma operação só que na máquina M_1. Finalmente o box b_6 (b_{10}) representa as peças P_1 (P_2) finalizadas.

De acordo com a descrição de MFG elaborada neste trabalho, a matriz incidente correspondente para o modelo fica da seguinte forma:

$$A=\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & -1 \end{bmatrix}$$

A marcação inicial é dada por:

$$M_0^T=(n_1,0,0,0,0,0,1,n_2,0,0,N-n_1,1,N,1,1,N,0,N-n_2,1,N)$$

Como estes dados, pode-se verificar as seguintes soluções *P-invariantes*:

$$y_1^T=(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

$$y_2^T=(0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0)$$

$$y_3^T=(0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$$

Uma vez que $M^T y=M_0^T y$ e que $M(b_x)=M(p_x)$, tem-se:

$$M(b_1)+M(b_2)+M(b_3)+M(b_4)+M(b_5)+M(b_6)=n_1,$$

$$M(b_8)+M(b_9)+M(b_{10})=n_2 \text{ e}$$

$$M(b_2)+M(b_5)+M(b_7)+M(b_9)=1$$

As duas primeiras equações implicam que o total de peças P_1 no sistema é constante e igual a n_1, em qualquer momento. Isto também é verdadeiro para P_2. Mesmo se tratando de conclusões triviais, estas são características importantes que um modelo deve possuir.

A ultima equação mostra que a máquina M_1 pode estar disponível ou em uso nas operações um e dois de peças P_1 , ou então nas operações um para peças P_2.

A tentativa de se encontrar as *T-invariantes* mostra que a única solução é a trivial. Desta forma, como esperado, conclui-se que o sistema não é reversível e também que se trata de um modelo não vivo (*dead*), isto é, chega a um estado no qual não são possíveis novos disparos.

2.4. Mark Flow Graph e Production Flow Schema

A técnica do Production Flow Schema (PFS), apresentada por MIYAGI (1993), decompõe um processo produtivo em elementos ativos (atividade), passivos (distribuidores) e arcos de relacionamento. A Figura 13 exibe um exemplo de representação de um SED através do PFS.

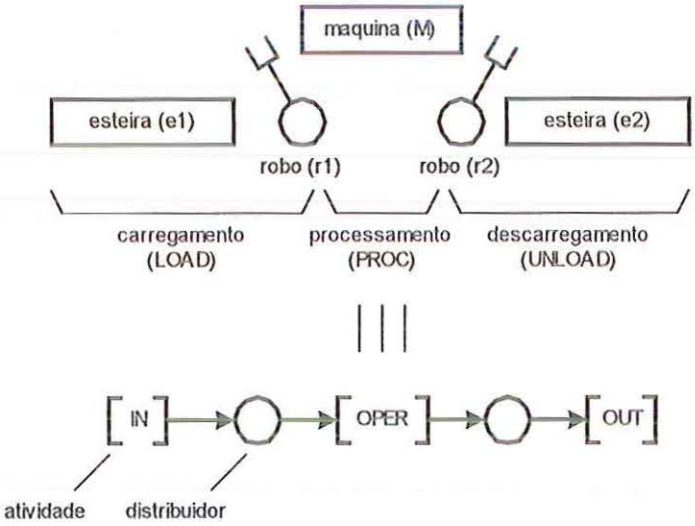

Figura 13. Exemplo de representação de um SED através do PFS

O PFS pode ser convertido num MFG interpretado, no qual as atividades e distribuidores são interpretados como transições e boxes respectivamente. Neste contexto, o MFG/PFS possui uma abordagem *top-down*, no qual a atividade pode incluir vários outros eventos e estados organizados de forma hierárquica. A

atividade corresponde assim a um macro evento que, por sua vez, representa a realização de operações (sub atividades). A Figura 14 exibe os níveis hierárquicos de uma atividade.

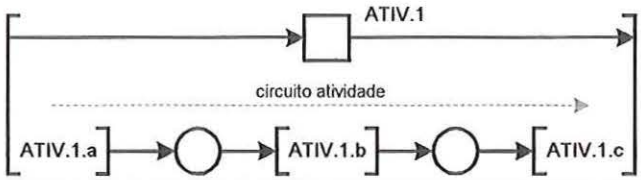

Figura 14. Níveis hierárquicos de uma atividade em PFS junto com elementos MFG

Desta forma a representação do PFS em rede de Petri Lugar/Transição é análoga à representação apresentada para o MFG. GUSTIN (1999), JUNQUEIRA (2001) e VILLANI (2000) apresentam exemplos e aplicações práticas do uso do MFG e PFS em conjunto.

2.5. Sequential Flow Chart

O Sequential Flow Chart (SFC) ou Grafcet (BIEREL & ROUSSEL, 2001; FUJINO et al., 2000; MIYAGI, 1993 e ZAYTOON, 1996) é um padrão internacional utilizado para especificação e implementação de controle seqüencial em sistemas de manufatura, e também pode ser considerado como uma rede de Petri interpretada. Assim como no MFG, as técnicas de análise já consagradas da rede de Petri podem ser abordadas sobre ele.

Nesta seção é introduzido o SFC descrito enquanto uma rede de Petri Lugar/Transição, visando sua implementação no editor.

2.5.1. Elementos estruturais e suas inter-relações

O SFC é composto dos seguintes elementos básicos estruturais: o passo, a ação, a transição e condições associadas às transições ou ações. A Figura 15 mostra os conceitos básicos do SFC.

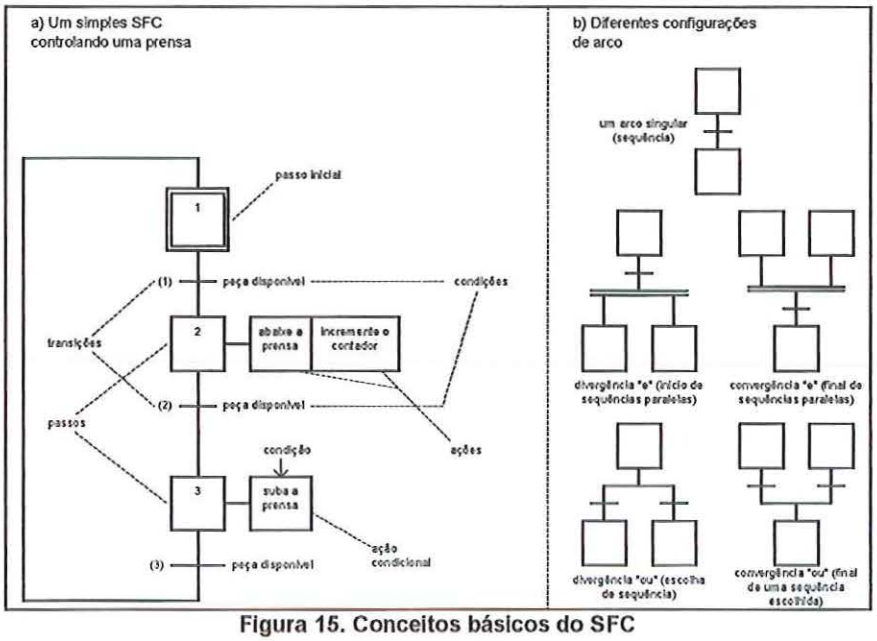

Figura 15. Conceitos básicos do SFC

2.5.2. Comportamento dinâmico

O passo representa um estado do sistema no qual ações podem estar associadas. Um passo pode estar ativo ou inativo. As ações associadas podem ser efetuadas quando o passo fica ativo. O estado de um SFC é dado pelo conjunto de passos ativos existentes. A transição conecta um (ou mais) passo(s) anteriores a um (ou mais) passo(s) posteriores, determinando a decisão de mudança de estado do sistema.

O comportamento dinâmico do SFC é dado por cinco regras de evolução:

R1: O estado inicial do SFC é dado por todos os passos ativos inicialmente (passos iniciais);

R2: Uma transição é válida se todos os passos anteriores estão ativos. Uma transição válida é disparada se a condição associada for verdadeira;

R3: O disparo de uma transição resulta na desativação de seus passos anteriores, e simultânea ativação de seus passos posteriores;

R4: Transições simultaneamente válidas são simultaneamente disparadas;

R5: Se um passo está para ser simultaneamente ativado e desativado, ele

permanece ativo.

2.5.3. SFC representado como rede de Petri Lugar/Transição

Conforme relatado, o SFC pode ser representado como uma rede de Petri interpretada. FUJINO et al. (2000) apresenta uma proposta de representação, considerando os passos como lugares na rede de Petri. No entanto, apresenta-se aqui uma proposta diferente, no qual foram estabelecidas as seguintes regras:

- As condições existentes no SFC são consideradas como estruturas externas ao modelo e, portanto, não fazem parte da representação;
- O SFC possui uma estrutura cíclica, que deve ser mantida pelo modelo em rede de Petri;
- A transição entre passos e conseqüente execução de ação, no SFC, estão associados ao disparo de uma transição na rede de Petri, uma vez que esta representa um evento;
- O lugar, na rede de Petri, representa o fim da execução das ações de um passo, indicando que uma transição, do SFC, pode disparar.
- A marcação existente no modelo está associada aos passos executados. Desta forma, a quantidade de marcas em cada estado do sistema, deve ser igual ao número de seqüências paralelas executadas. Em sistemas não paralelos, a quantidade de marcas deve sempre ser igual a um.

Um exemplo de representação de um modelo simples em SFC para rede de Petri Lugar/Transição pode ser visto na Figura 16.

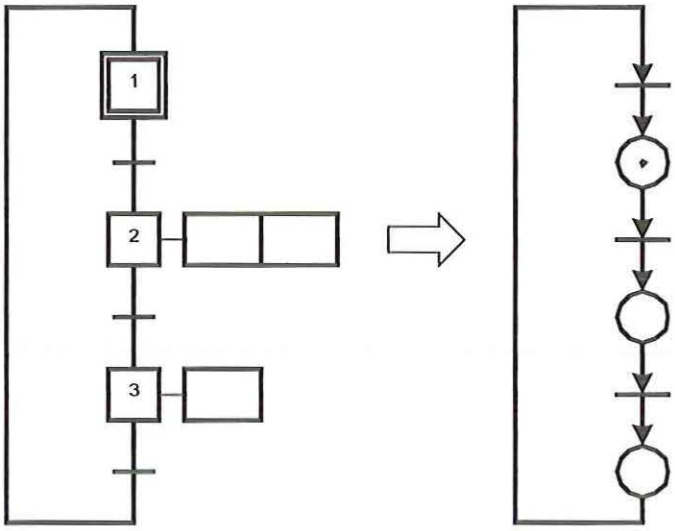

Figura 16. Exemplo de representação de um SFC em rede de Petri Lugar/Transição.

A Figura 17 exibe como as estruturas divergentes e convergentes do SFC são representadas. Observa-se que, na modelagem, uma estrutura divergente deve necessariamente ser finalizada com uma estrutura convergente do mesmo tipo (paralela ou de escolha).

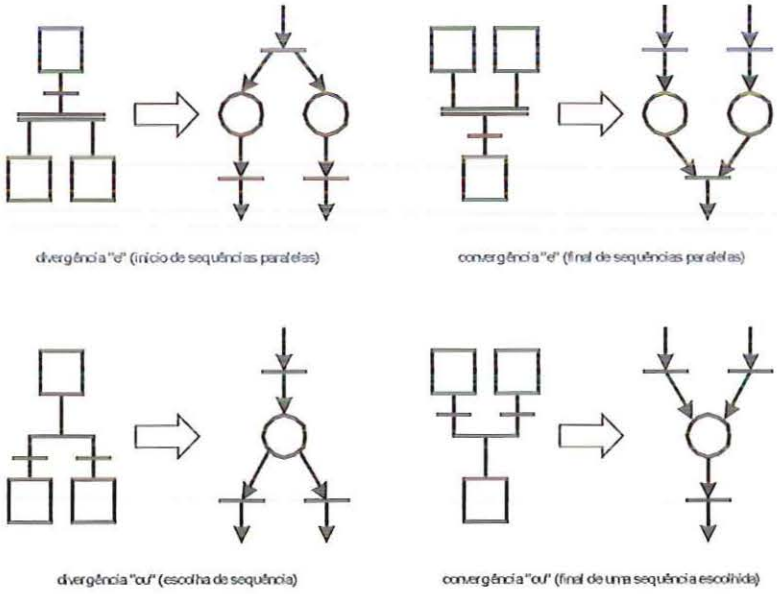

Figura 17. Representação das estruturas divergentes e convergentes do SFC em rede de Petri Lugar/Transição.

Capítulo 3 - Petri Net Tools 2000

3.1. Considerações Iniciais

A ferramenta Petri Net Tools (PETRI, 2001a), desenvolvida pelo grupo de pesquisa do laboratório de Simulação do Departamento de Engenharia Mecânica da Escola de Engenharia de São Carlos USP, consiste num editor e simulador gráfico de redes de Petri Lugar/Transição, desenvolvido com a metodologia orientada a objetos (KHOSHAFIAN, 1990).

O Petri Net Tools 2000, desenvolvido neste trabalho, consiste numa nova versão do Petri Net Tools que incorpora características de um software aberto, viabilizando o processo de contínuo desenvolvimento, promovendo a inclusão, reutilização e aperfeiçoamento de funcionalidades. A partir dessas características, torna-se possível incorporar a edição de outros tipos de redes de Petri, inclusive as interpretadas, através da reutilização de funcionalidades matemáticas e de edição gráfica da rede de Petri Lugar/Transição.

Este capítulo apresenta o editor Petri Net Tools 2000. Na Seção 3.2 são inicialmente abordados detalhes da arquitetura aberta do software, que consiste numa visualização do funcionamento geral do editor, apresentando a estrutura de *templates*, responsável pela característica de software aberto. A Seção 3.3 descreve os aspectos operacionais do software e, finalizando, a Seção 3.4 apresenta os aspectos de implementação, abrangendo detalhes do paradigma de desenvolvimento utilizado.

3.2. Arquitetura Aberta

De uma forma geral, o funcionamento do editor é composto de dois módulos, conforme pode ser visualizado na Figura 18.

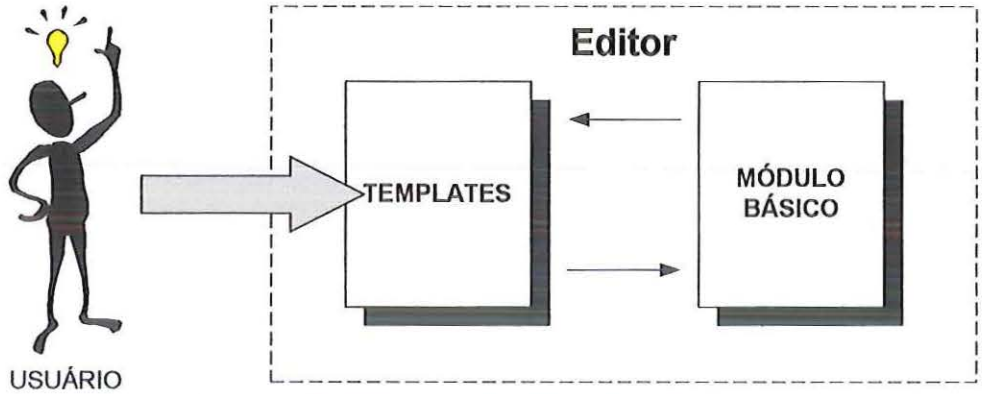

Figura 18. Modelo geral de funcionamento do editor

O módulo básico é composto por um conjunto de ferramentas genéricas de edição de modelos gráficos, e por ferramentas de análise de redes de Petri Lugar/Transição. Estas ferramentas são utilizadas pelo usuário indiretamente através dos *templates.*

Cada *template* consiste numa forma diferente e personalizada de uso destas ferramentas, isto é, os *templates* consistem em diferentes modos de funcionamento do editor. O sistema possui a característica de sistema aberto, uma vez que o usuário pode implementar, alterar, adicionar e remover *templates* do editor.

Os *templates* podem ser utilizados em diversos contextos. Podem ser utilizados para fornecer novas funcionalidades ao editor, como por exemplo, novos recursos gráficos, ou novas metodologias de análise. Mas eles também para incorporar novas ferramentas de modelagem, que podem utilizar elementos gráficos diferentes e as ferramentas de análise e simulação de redes de Petri já implementadas.

Neste contexto, os *templates* podem ser utilizados para fornecer ao editor a possibilidade de trabalhar com redes de Petri interpretadas, como o MFG/PFS e SFC.

O Petri Net Tools 2000, por se tratar de um sistema aberto, possui dois tipos distintos de usuários. Um deles é o *usuário* comum, que utiliza os *templates* já implementados do editor para elaborar um modelo com finalidades de análise e

simulação. O outro consiste num *programador* que utiliza o editor para implementar, modificar ou reaproveitar funcionalidades, através da geração de novos *templates*.

Maiores detalhes sobre os módulos do software podem ser visto no Apêndice I.

3.3. Aspectos Operacionais

Os *templates*, como relatado anteriormente, consistem em modos de funcionamento do editor. Ao iniciar o editor, o usuário deve escolher com qual *template* deseja trabalhar (Figura 19).

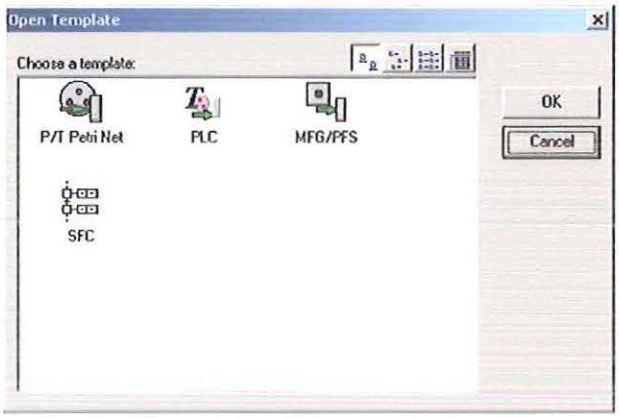

Figura 19. Tela de escolha de *template*

O funcionamento do editor varia de acordo com o *template* escolhido. De uma forma geral, o editor consiste num aplicativo do tipo *Multiple Document Interface* (MDI), capaz de trabalhar com diversos documentos simultaneamente. Cada documento constitui num modelo representável por uma rede de Petri Lugar/Transição. A Figura 20 mostra a tela do Petri Net Tools 2000 executando um *template* para edição de modelos de MFG/PFS.

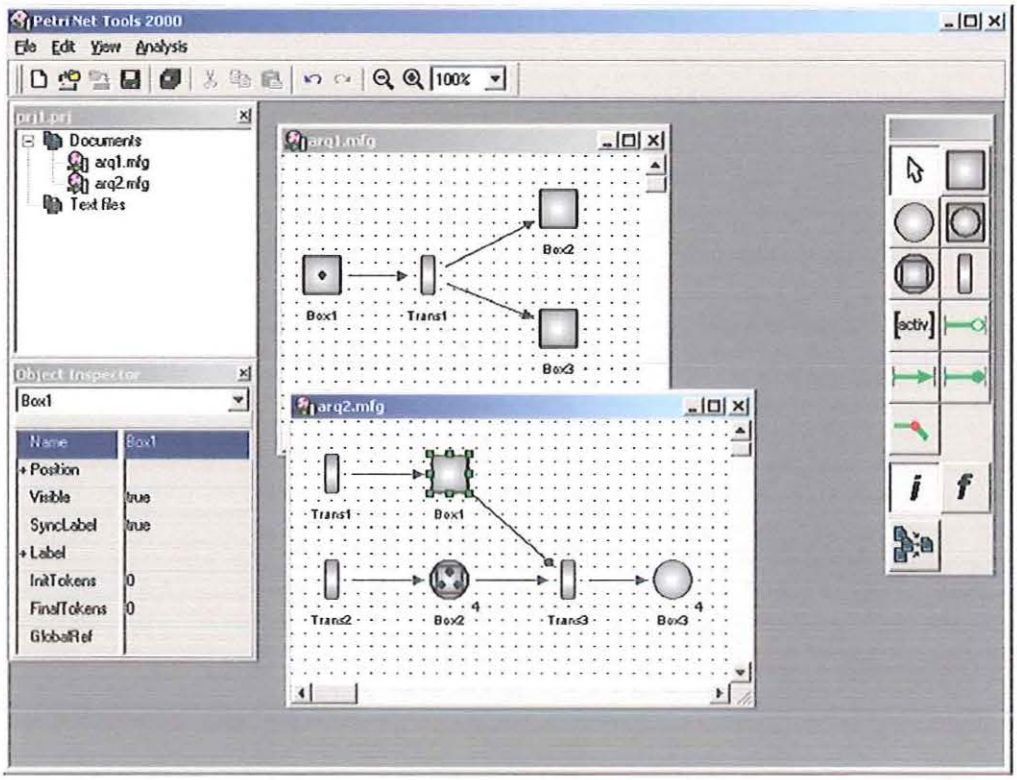

Figura 20. Tela do Petri Net Tools 2000 com um template para MFG/PFS

Os aspectos operacionais de funcionamento do editor são formados por um grupo de características genéricas, compartilhadas ou não pelos *templates*, e pelas características específicas de cada *template*.

3.3.1. Características genéricas

3.3.1.1. Edição do modelo

A edição do modelo é feita através da janela de edição (vide Figura 20), na qual a manipulação dos objetos normalmente é realizada com auxílio de uma barra de ferramentas. O funcionamento da barra de ferramentas esta geralmente associado com a criação, posicionamento e seleção dos objetos na janela de edição. No entanto os botões e seus respectivos funcionamentos podem ser personalizados pelos *templates*.

Outra ferramenta de manipulação de objetos é o *Object Inspector* que permite acesso aos atributos dos objetos, como por exemplo, o nome, a posição ou o número de marcas iniciais. A Figura 21 exibe a janela do *Object Inspector* para um Box de um modelo em MFG/PFS.

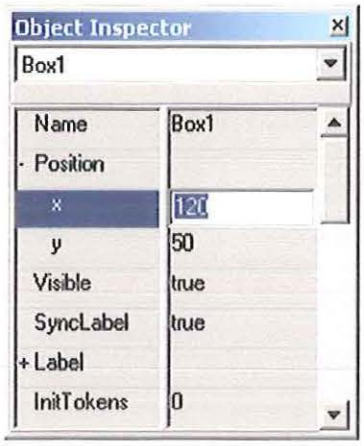

Figura 21. Janela do *Object Inspector*

Através da barra de menus do editor é possível acessar outras funcionalidades úteis na edição dos modelos. São estas o controle e exibição de pontos de grade (*grid*); controle de *zoom*; operações de fazer/refazer (*undo/redo*); funcionalidades para recortar, copiar e colar objetos (*cut/copy/paste*); e funcionalidades de arquivamento e recuperação de modelos para arquivos.

3.3.1.2. Controle de projeto

O controle de projeto consiste em permitir que documentos relacionados possam ser organizados na forma de um projeto. A janela de projeto (Figura 22) permite a visualização e manipulação destes documentos.

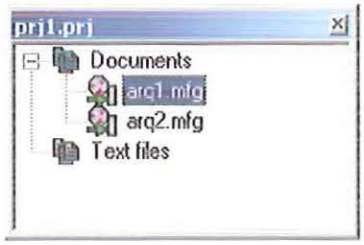

Figura 22. Janela de projeto

Um exemplo de uso desta funcionalidade pode ser visto no *template* para rede de Petri Lugar/Transição, onde o conceito de rede de Petri modular é implementado unindo todos os arquivos do projeto num único modelo.

3.3.1.3. Ferramentas de Auxílio à Análise

As ferramentas de auxílio à análise também consistem em características genéricas compartilhadas pelos *templates*. No entanto, elas não possuem uma visualização gráfica padrão. Cada *template* é responsável por implementar a visualização gráfica específica dessas ferramentas, ou então reaproveitar essa funcionalidade de outro *template*.

As ferramentas de análise, disponíveis no módulo básico são:

- Geração da matriz de incidência
- Geração de árvore de alcançabilidade;
- Análise de propriedades comportamentais (alcançabilidade, limitabilidade, e vivacidade);
- Análise usando invariantes (S-invariante e T-invariante).

3.3.2. Template para rede de Petri Lugar/Transição

Este *template* permite a edição, modelagem e análise de redes de Petri Lugar/Transição, no qual o usuário pode criar, através da barra de ferramentas, objetos que representam lugares, transições e arcos orientados, cada qual com suas respectivas propriedades, como número de marcas dos lugares e peso dos arcos.

3.3.2.1. Rede de Petri modular

Uma das funcionalidades do *template* para rede de Petri Lugar/Transição consiste no uso da técnica de rede de Petri modular (INAMASU, 1995; INAMASU & PORTO, 1997; JENG, 1997; PROTH et al., 1997 e VOLGLER, 1992), que tem como objetivo a simplificação do modelo em rede de Petri. O usuário pode criar lugares e transições comuns em mais de um arquivo do projeto, denominados lugares e transições globais. Todos os arquivos do projeto podem então ser integrados, formando um único modelo.

A Figura 23 exibe um exemplo de rede de Petri modular editado no Petri Net Tools 2000, com três arquivos e dois lugares globais, denominados "Entrada", de capacidade 1, e "Saída".

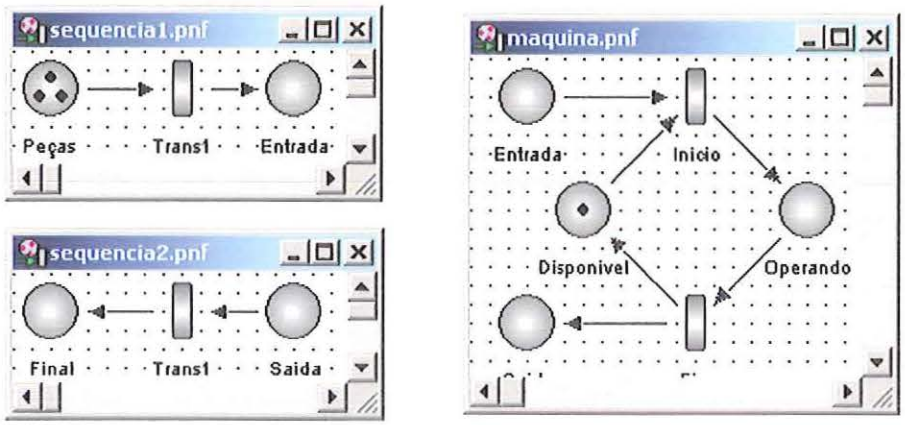

Figura 23. Exemplo de rede de Petri modular

Neste exemplo, o arquivo "maquina.pnf" modela a operação de uma máquina sobre uma peça, e os arquivos "seqüência1.pnf" e "seqüência2.pnf" modelam a disponibilidade e saída das peças na máquina.

3.3.2.2. Análise

As possíveis formas de análise disponíveis para este *template* consistem na simulação, identificação de propriedades comportamentais do modelo, geração da matriz de incidência, geração da árvore de alcançabilidade, e análise usando invariantes. Todas essas funcionalidades estão disponíveis no menu *Analysis*.

A simulação (Figura 24) fornece mecanismos ao usuário para escolha do caminho desejado (soluções da árvore de alcançabilidade) e de controle da simulação (passo a passo ou continuamente).

EESC - USP Biblioteca Campus de S. Carlos

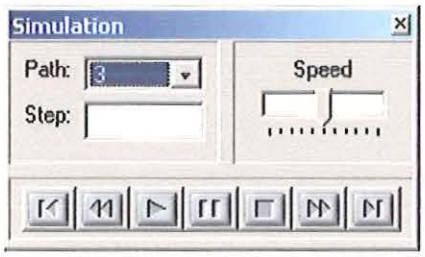

Figura 24. Janela de simulação

As propriedades comportamentais do modelo são simplesmente disponibilizadas através da janela de propriedades do modelo (Figura 25).

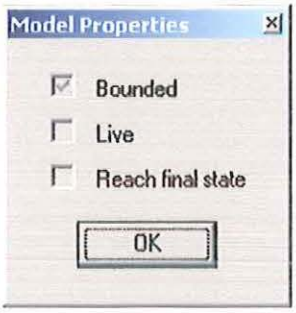

Figura 25. Janela de propriedades comportamentais

A matriz de incidência e a árvore de alcançabilidade são disponibilizadas em formato texto. Considerando o exemplo da Figura 23, as saídas geradas pelo editor podem ser visualizadas na Figura 26.

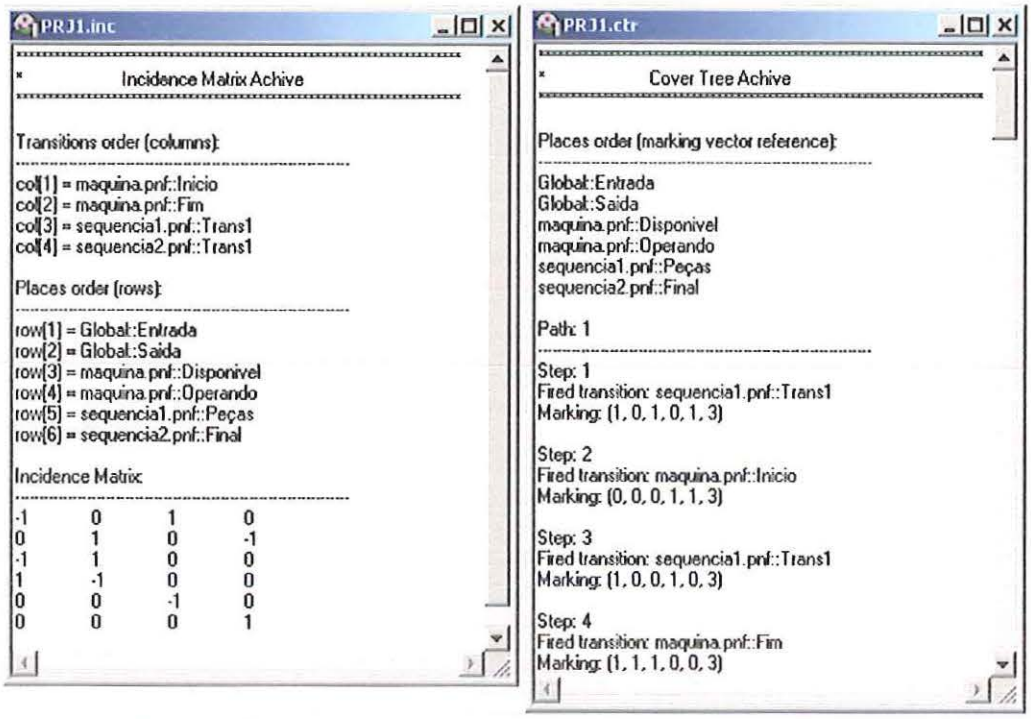

Figura 26. Exemplo de matriz de incidência e árvore de alcançabilidade

Por fim, a análise de invariantes, implementada para os S-invariantes, consiste em identificar se a soma de marcas de um conjunto de lugares é constante em qualquer marcação alcançável. Assim o usuário seleciona um conjunto de lugares e o sistema retorna se eles são ou não invariantes, conforme pode ser visualizado na Figura 27.

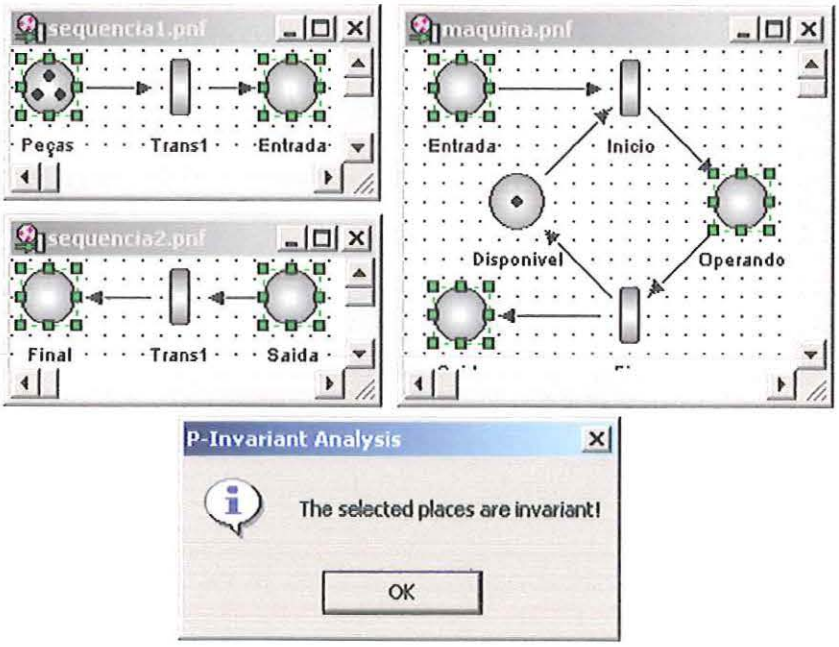

Figura 27. Exemplo da análise de invariantes

3.3.3. Template para MFG/PFS

Este *template* permite a edição, modelagem e análise de modelos em MFG/PFS, no qual o usuário pode criar, através da barra de ferramentas, objetos que representam boxes, boxes agrupadores, boxes dispersores, boxes capacidade, transições, arcos orientados, portas e atividades (veja Figura 20).

De forma análoga ao conceito de rede de Petri modular, este *template* implementa o conceito de MFG/PFS modular, no qual boxes e transições podem ter escopo global, integrando os diversos modelos de um projeto.

3.3.3.1. Análise

As mesmas análises aplicadas para a rede de Petri Lugar/Transição (simulação, identificação de propriedades comportamentais do modelo, geração da matriz de incidência, geração da árvore de alcançabilidade, e análise usando invariantes) são reaproveitadas e implementadas para o MFG/PFS. Neste contexto, o modelo em MFG/PFS é representado como rede de Petri Lugar/Transição, conforme formalizado na Seção 2.3 e Seção 2.4.

A Figura 28 exibe um exemplo de geração da matriz incidente para um modelo simples em MFG/PFS, no qual é possível visualizar a representação de um box e uma transição conectados por um arco orientado.

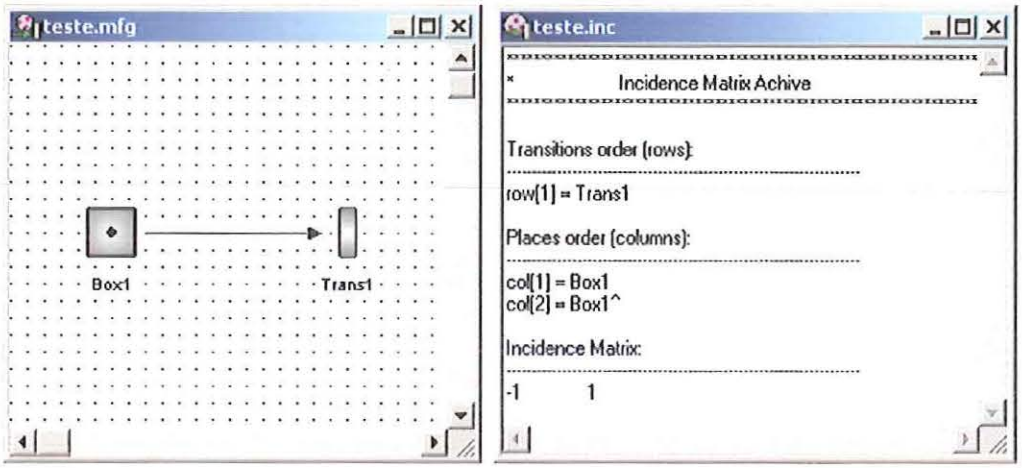

Figura 28. Exemplo de representação de matriz de incidência para o MFG

Pode-se observar que a matriz de incidência consiste num vetor coluna de duas dimensões, que representam dois places, conforme o modelo proposto neste trabalho. Assim, "Box1" representa o número de marcas no box e "Box1^" representa o lugar limitador de "Box1".

3.3.4. Template para SFC

Este *template* permite a edição, modelagem e análise de modelos em SFC, no qual o usuário pode criar através da barra de ferramentas passos, estruturas simultâneas ("e") e de seleção ("ou"), *loops*, ações e condições. A Figura 29 mostra o editor para o *template* de SFC.

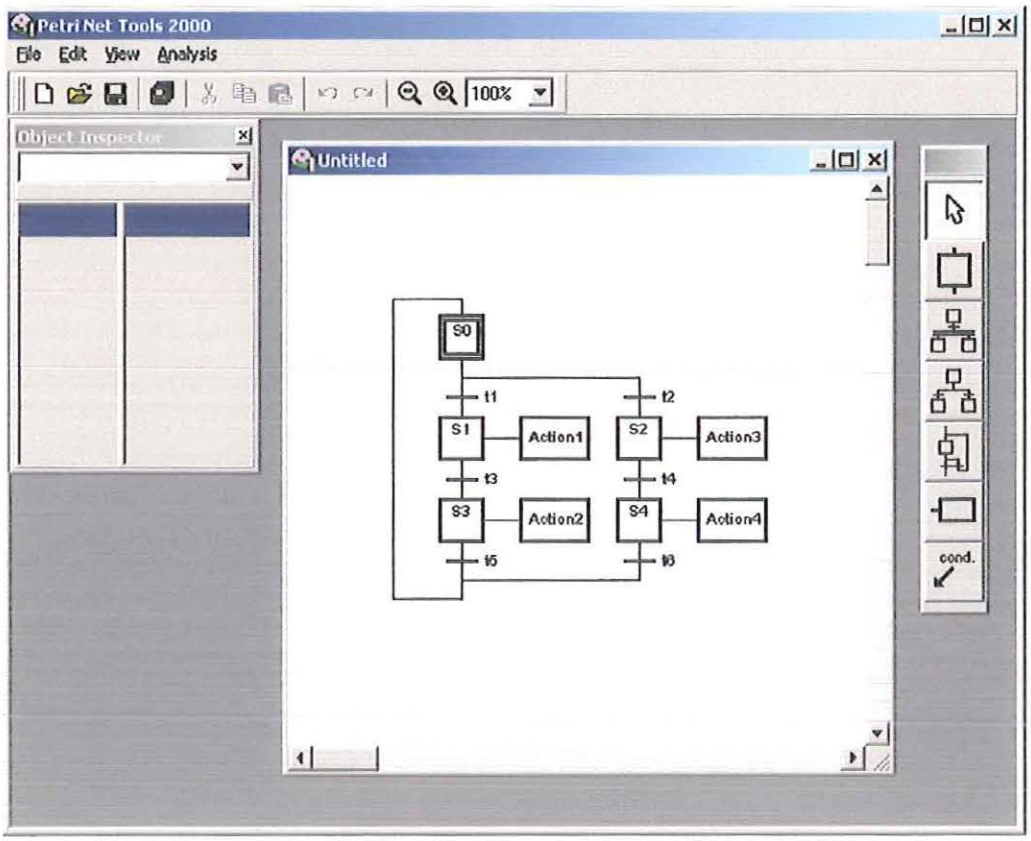

Figura 29. *Template* para SFC

3.3.4.1. Análise

As mesmas análises aplicadas para a rede de Petri Lugar/Transição (simulação, identificação de propriedades comportamentais do modelo, geração da matriz de incidência, geração da árvore de alcançabilidade, e análise usando invariantes) são reaproveitadas e implementadas para o SFC. Vale lembrar que os modelos em SFC consistem num conjunto restrito da rede de Petri sempre reversível e 1-limititado.

Neste contexto, o modelo em SFC é representado como rede de Petri Lugar/Transição, conforme formalizado na Seção 2.5. A Figura 30 mostra um exemplo de geração da árvore de alcançabilidade para o exemplo apresentado na Figura 29.

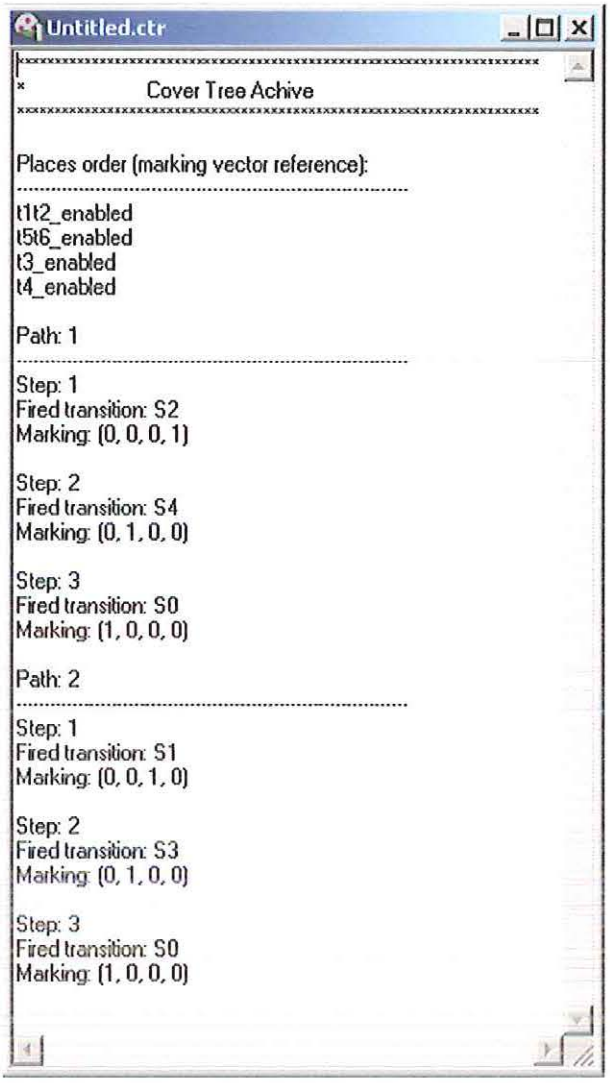

Figura 30. Árvore de alcançabilidade para um exemplo de SFC

Nota-se que o disparo das transições está associado à execução de um passo (S0, S1, S2, S3 ou S4), e que a marcação indica qual transição do SFC pode ser disparada.

3.4. Aspectos de Implementação

Conforme relatado, o módulo básico é composto por um conjunto de ferramentas genéricas de edição de modelos gráficos e por ferramentas de análise de redes de Petri Lugar/Transição, que são utilizadas pelo usuário indiretamente através dos *templates*. Desta forma, os *templates* reutilizam e personalizam o código e o projeto de software desenvolvidos no módulo básico, de acordo com a necessidade. A

orientação a objetos e as metodologias de reutilização de código e projeto, *design patterns* e *frameworks* consistem nas principais técnicas utilizadas na implementação do sistema com essas características.

3.4.1. Orientação a Objeto

A orientação a objeto (GUIMARÃES, 1998) tem sido vista como uma ferramenta de auxilio para promover a reutilização de código (JOHNSON & FOOT, 1988). Isto é possível através da combinação de suas principais características, como abstração de dados, herança e o polimorfismo.

A abstração dos dados é obtida uma vez que a modelagem do sistema é baseada no uso de objetos, que por sua vez encapsulam dados e operações, encorajando a modularização do sistema e facilitando o seu entendimento. A herança possibilita que objetos compartilhem operações, e também permite a programação por diferença, na qual, ao reutilizar o código, o programador só necessita implementar as diferenças desejadas. Por fim, o polimorfismo facilita com que componentes trabalhem corretamente numa maior variedade de contextos.

No entanto, a simples utilização destes recursos não é o suficiente. Os componentes de um programa devem ser projetados para serem reutilizados, no qual as palavras chaves são a generalização e a flexibilidade.

Neste contexto a utilização de padrões de projeto (*design patterns*) consiste numa ferramenta muito útil no auxílio à análise e projeto de módulos visando generalização e flexibilidade para reutilização, e foram amplamente utilizadas no desenvolvimento do editor.

3.4.2. Design Patterns

O projeto de software orientado a objeto já consiste numa tarefa difícil, e, pensando-se em reutilização, a tarefa pode se tornar mais difícil ainda. Devido a estas constantes dificuldades, programadores experientes organizaram um conjunto padrão de soluções genéricas e flexíveis, para os problemas que normalmente são enfrentados na análise e projeto de software orientado a objeto. A este conjunto de soluções é atribuído o nome de *design patterns* (padrões de projeto) (GAMMA et a., 1997).

Dentre os padrões de projeto relacionados em GAMMA et al. (1997), o *Método da Fábrica*, a *Composição* e o *Método Template* foram os padrões mais utilizados na implementação do editor. Eles fornecem estruturas que viabilizam a implementação de um sistema aberto de acordo com as características desejadas (vide Apêndice II).

3.4.3. Frameworks

A reutilização de código e o uso de padrões de projeto consistem nas metodologias que viabilizam a implementação de um sistema genérico e flexível como desejado. No entanto, o módulo básico, além de permitir a reutilização de código, implementa todo a estrutura e o fluxo de execução do programa. Isto é, o módulo básico contém já definidos a análise e projeto do editor, cabendo aos *templates* identificarem apenas características específicas do modelo proposto. Neste contexto o módulo básico consiste num esqueleto geral do programa, chamado de *framework* (FAYAD, 1999; FAYAD, 2000; GUIMARÃES, 1998 e JOHNSON, 2001).

Segundo JOHNSON & FOOT (1988), um *framework* consiste num conjunto de classes (abstratas ou não) que incorpora um projeto abstrato para a solução de problemas relacionados. Em outras palavras, um *framework* consiste num esqueleto de programa construído para um domínio específico, no qual um conjunto de classes implementa uma solução padrão para a construção de um aplicativo. Um *framework* deve também possibilitar ao programador a flexibilidade de definir as características específicas para cada aplicação, através da reutilização do código implementado por suas classes (criação dos *templates*).

O uso de *frameworks* tem como objetivo principal à construção de aplicativos (no caso a construção dos *templates*) com um reduzido uso de componentes e com o menor código possível, de forma a facilitar a manutenção. Num *framework* existe um intercâmbio de chamadas de métodos entre o *framework* (módulo básico) e o código do programador (*template*), no qual caracteriza-se uma inversão de controle. O *framework* é responsável por controlar o fluxo de execução do programa, e utiliza o código do programador para as implementações específicas da aplicação.

Decorrente da dificuldade já relatada em se projetar classes para serem reutilizadas, surge um dos principais problemas dos *frameworks*, isto é, a grande dificuldade de sua construção. Segundo JOHNSON (2001), esta dificuldade surge a

partir do fato de que pessoas pensam de forma concreta, não abstratamente. Abstrações são obtidas de uma análise *bottom-up*, no qual é realizada uma análise de domínio a partir de exemplos concretos.

Os exemplos concretos utilizados na análise de domínio para a construção do editor foram os *templates* para rede de Petri Lugar/Transição, MFG/PFS e SFC.

Capítulo 4 - Conclusões

A rede de Petri, por se tratar de uma poderosa ferramenta de modelagem e análise, é utilizada numa grande variedade de sistemas. Conforme apresentado na Seção 1.1, existem diversas ferramentas de redes de Petri no mercado, cada qual oferecendo funcionalidades específicas para um âmbito de solução. O Petri Net Tools 2000 apresenta uma grande contribuição uma vez que consiste numa ferramenta aberta que permite a inclusão e aperfeiçoamento de funcionalidades. Através da reutilização de código e projeto de software, simplifica-se a implementação de soluções para diversos âmbitos de aplicação. Além disso, a estrutura de *templates* permite trabalhar de diversas formas no mesmo editor. O grande potencial desta ferramenta não está nas funcionalidades que ela apresenta, mas sim no processo de contínuo desenvolvimento, que possibilita a cada tipo de usuário personalizar as funcionalidades existentes de acordo com o domínio de aplicação desejado.

Os resultados obtidos da revisão das técnicas do MFG/PFS e SFC mostram que estas podem ser interpretadas como redes de Petri Lugar/Transição. No entanto, deve-se lembrar que, para o MFG, não foram considerados os elementos externos (portas externas e arcos de sinal de saída). O estudo de caso elaborado para a técnica do MFG/PFS comprova que as metodologias de análise já consagradas podem ser aplicadas e reaproveitadas para essas interpretações, como por exemplo, o uso da matriz de incidência, a determinação e análise da árvore de alcançabilidade, análise usando invariantes entre outras. De certo, os resultados

obtidos com a implementação dos *templates* (MFG/PFS e SFC) confirmam o sucesso na implementação da ferramenta.

Apesar do potencial desta ferramenta, algumas considerações são necessárias. A implementação de um *template* necessariamente requer o domínio da metodologia de desenvolvimento de sistema orientado a objetos, e também das técnicas de reutilização de código, padrões de projeto e *frameworks*. Além disso, o conhecimento do *framework* implementado e de sua estrutura interna também são necessários. O módulo básico consiste no módulo de maior complexidade do editor, conforme o esperado, simplificando a estrutura dos *templates*. No entanto, o programador deve conhecer o funcionamento do módulo básico para conseguir implementar as funcionalidades desejadas no *template*. Desta forma, a fim de garantir o sucesso da ferramenta, torna-se indispensável uma documentação bem detalhada e planejada do sistema por completo, incluindo um livro de receitas com explicações passo a passo de como desenvolver as funcionalidades normalmente desejadas.

Outro aspecto importante está relacionado aos testes e validação do *framework*, indispensáveis em futuros trabalhos. Conforme apresentado, o *framework* simplifica novas implementações através da reutilização. Um dos aspectos mais importantes relacionado à reutilização consiste na redução de custos, assumindo que o código reutilizado já foi testado e validado.

Diante dos resultados apresentados, conclui-se que o sistema proposto atingiu os objetivos traçados, além de fornecer uma contribuição efetiva para área, possibilitando que novas propostas de redes de Petri interpretadas possam facilmente ser testadas e validadas junto ao editor. No entanto, futuros trabalhos relacionados com uma documentação para o desenvolvimento de *templates* e com testes e validação do *framework* são necessários.

4.1. Trabalhos Futuros

Devido às características do editor, existem diversos futuros trabalhos a serem implementados, muitos na forma de novos *templates*. Desta forma seguem abaixo algumas sugestões:

- Desenvolvimento de uma documentação interativa voltada para o desenvolvimento de *templates;*
- Realização de testes e validação do *framework* a fim de garantir o sucesso na reutilização de funcionalidades implementadas;
- Desenvolvimento de *template* para programação de Contadores Lógicos Programáveis (CLPs) através de redes de Petri Lugar/Transição em conjunto com elementos de SFC;
- Ampliação do *template* de MFG/PFS para incorporar elementos de E-MFG orientado a objetos;
- Implementar extensões e métodos de análise mais consagradas da rede de Petri podendo incorporar também a análise de mutantes;
- Desenvolver interface para entrada e análise de dados apropriada para uso em planejamento e análise de sistemas.

Referências Bibliográficas

AL-JAAR, R. Y.; DESROCHERS, A. (1995) Aplications of petri nets in manufacturing systems: modeling, control, and performance analysis. *IEEE*, New York.

BIEREL, E.; ROUSSEL, J. M. *Home of grafcet* (2001) http://www.lurpa.ens-cachan.fr/grafcet.html (20 Mar. 2001)

CECIL, J. A.; SRIHARI, K.; EMERSON, C. R. (1992) A review of petri-net applications in manufacturing. *The International Journal of Advanced Manufacturing Technology*, v.7, p.168-177.

DEUTSCH, L. P. (1989) Design reuse and frameworks in the Smalltalk-80 programming system. New York, *ACM - Software Reusability*, v.2, p. 55–71.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. (1999) *Building Applications Frameworks:* object-oriented foundations of framework design. New York, John Wiley & Sons.

FAYAD, M. E.; JOHNSON, R. E. (2000) *Domain-Specific Application Frameworks:* frameworks experience by industry. New York, John Wiley & Sons.

FUJINO, K.; IMAFUKU, K.; YAMASHITA, Y.; NISHITANI, H. (2000) Design and verification of the SFC program for sequential Control. *Computers and Chemical Engineering*, v24, p.303-308.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. (1997) *Design patterns: elements of reusable object-oriented software*. MA, Addison-Wesley.

GUIMARÃES, J. O. (1998) *Sistemas orientados a objeto.* São Carlos. http://www.dc.ufscar.br/~jose/courses/soo/soo.zip (20 Mar. 2001)

GUSTIN, G. D. B. (1999) *Aplicação de redes de petri interpretadas na modelagem de sistemas de elevadores em edifícios inteligentes*. São Paulo. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo.

INAMASU, R. Y. (1995) *Modelo de FMS:* uma plataforma para simulação e planejamento. São Carlos. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

INAMASU, R. Y.; PORTO, A. J. V. (1997) Plataforma para sequenciamento de processo alternativo de fabricação baseado em rede de Petri. In: CONGRESO IBEROAMERICANO DE INGENIERÍA MECÁNICA, Instituto Superior Politécnico José Antonio Echeverria, La Habana, Cuba.

JENG, M. D. (1997) Petri nets for modeling automated manufacturing systems with error recovery. *IEEE Transactions on Robotics and Automation*, v.13, n.5, p.752-760.

JENSEN, K. (1980) Coloured petri nets and the invariant methods. *Theoretical Computer Science*, v.14, p.317-336, Amsterdam.

JENSEN, K. (1992) *Coloured petri nets*: basic concepts, analysis methods and practical use. Berlin, Springer-Verlag. v.1.

JENSEN, K. (1994) *Coloured petri nets*: basic concepts, analysis methods and practical use. Berlin, Springer-Verlag. v.2.

JOHNSON, R.; FOOTE, B. (1988) Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, p. 22-35.

JOHNSON, R.; RUSSO, V. (1991) *Reusing object-oriented designs*. United States, Urbana, Department of Computer Science, University of Illinois. (Tech Report UIUCDCS 91-1696)

JOHNSON, R. (2001) *Frameworks home page*. http://st-www.cs.uiuc.edu/users/johnson/frameworks.html (20 Mar. 2001)

JUNQUEIRA, F. (2001) *Modelagem de sistemas flexíveis de movimentação de materiais através de redes de petri interpretadas*. São Paulo. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo.

KHOSHAFIAN, A. R. (1990) *Object orientation*: concepts, languages, databases, user interfaces. New York, John Wiley & Sons.

MEYERS, S. (1992) *Effective C++*. MA, Addison-Wesley.

MIYAGI, P. E.; HASEGAWA, K.; TAKAHASHI, K. (1989) *Mark flow graph (mfg) para modelamento e controle de sistemas de eventos discretos*. São Paulo, PMC/PEL/EPUSP, v.1, n.1, p.1-10. (Monografias em Automação e Inteligência Artificial)

MIYAGI, P. E. (1993) *Controle de sistemas de eventos discretos*: aplicação em automação industrial. São Paulo. Tese (Livre-Docência) – Escola Politécnica da Universidade de São Paulo.

MURATA, T. (1989) Petri net: properties, analysis and applications. *Proceedings of the IEEE*, v.77, n.4, p.541-579.

OBJECT MANAGEMENT GROUP. (2000) *OMG unified modeling language specification*. Needham, OMG. ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf (10 Apr. 2001)

PETRI *Net Tools* (2001a) http://www.petrinet.sc.usp.br (20 Mar. 2001)

PETRI *Nets Tool Database* (2001b) http://www.daimi.aau.dk/PetriNets/tools/db.html (20 Mar. 2001)

PORTO, A. J. V. (1990) *Desenvolvimento de um método de integração do planejamento do processo de fabricação e do planejamento e controle da produção, baseado na flexibilidade do processo de fabricação*. São Carlos. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

PROTH, J. M.; WANG, L.; XIE, X. (1997) A class of petri nets for manufacturing system integration. *IEEE Transaction on Robotics and Automation*, v.13, n.3, p.317-326.

SIMÃO, A. S. (2000) *Proteum - RS/PN:* uma ferramenta para a validação de redes de petri baseada na análise de mutantes. São Carlos. Tese (Mestrado) – Instituto de Ciências Matemática e de Computação, Universidade de São Paulo.

STROUSTRUP, B. (1991) *The C++ Programming Language*. MA, Addison–Wesley.

VILLANI, E. (2000) *Abordagem híbrida para modelagem de sistemas de ar condicionado em edifícios inteligentes*. São Paulo. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo.

VOLGLER, W. (1992) *Modular construction and partial order semantics of petri nets*. Lecture Notes in Computer Science, Springer-Verlag.

ZAYTOON, JANAN. (1996) Specification and design of logic controllers for automated manufacturing systems. *Robotics & Computer-Integrated Manufacturing*, v.12, n.4, p.353-366.

Apêndice I – Petri Net Tools 2000: Módulos

A arquitetura do Petri Net Tools 2000, apresentada na Seção 3.2, compõe o funcionamento do editor em dois módulos, básico e *templates*. Considerando aspectos de implementação, estes módulos são compostos por diversos sub-módulos, de forma a prover flexibilidade para a arquitetura aberta proposta. Cada sub-módulo consiste numa biblioteca de ligação dinâmica (*Dynamic Link Library*) do Microsoft Windows.

Segue a baixo a descrição das funcionalidades de cada sub-módulo pertencente ao módulo básico:

- **pntapp.dll:** Esse módulo é responsável por iniciar o aplicativo e gerenciar o carregamento dos *templates*. Ele fornece o ambiente básico do editor, isto é, a janela principal, as funcionalidades de escolha de *templates*;

- **genlex.dll:** Consiste num analisador léxico genérico, utilizado constante mente na abertura e validação dos arquivos de dados manipulados, e também nas operações de leitura da área de armazenamento do Microsoft Windows (*clipboard*);

- **pntobjects.dll:** Consiste numa biblioteca para modelagem de objetos utilizados e manipulados pelos documentos do editor. Estão implementadas todas as funcionalidades genéricas de manipulação de objetos gráficos e controle de suas respectivas propriedades. Fornece uma base para a

modelagem dos elementos gráficos dos *templates*, como por exemplo, lugares, transições e arcos da rede de Petri Lugar/Transição.

- **pnttpl.dll:** Este módulo fornece toda a estrutura funcional dos *templates*. Dentre as funcionalidades implementadas constam o controle dos documentos e das janelas de edição (funcionalidades de manipulação dos objetos gráficos, controle de *undo/redo*, funcionalidades de armazenamento em arquivos, copia e colagem da área de armazenamento, controle de grade de edição, controle de zoom), ferramentas de edição (usadas na barra de ferramentas e janela do *Object Inspector*) e funcionalidades de manipulação de projeto;

- **pntsim.dll:** Consiste no módulo de simulação, no qual estão implementadas funcionalidades de modelagem (não gráfica) e análise da rede de Petri Lugar/Transição.

Cada *template* implementado também consiste num arquivo DLL, que reutiliza as funcionalidades de todos sub-módulos listados acima, e que implementa novas funcionalidades especificas. Foram implementados os seguintes templates:

- **ptpn.dll:** *Template* para rede de Petri Lugar/Transição modular;

- **mfg.dll:** *Template* para MFG/PFS;

- **sfc.dll:** *Template* para SFC (Grafcet).

Os *templates* implementados também fazem parte do framework uma vez que também podem ser reutilizados na implementação de novos *templates*. Por exemplo, o *template* para MFG/PFS reutiliza-se das funcionalidades implementadas no *template* para rede de Petri Lugar/Transição devido a semelhança entre eles.

Apêndice II – Design Patterns

Conforme apresentado na Seção 3.4.2, *design patterns* consiste num conjunto padrão de soluções genéricas e flexíveis, para os problemas que normalmente são enfrentados na análise e projeto de software orientado a objeto.

Dentre os padrões de projeto relacionados em GAMMA et al. (1997), o *Método da Fábrica*, a *Composição* e o *Método Template* foram os padrões mais utilizados na implementação do editor. Eles fornecem estruturas que viabilizam a implementação de um sistema aberto de acordo com as características desejadas. Segue abaixo a descrição desses padrões, com o objetivo de ilustrar parte das estruturas que promovem a reutilização de código. Cada padrão é composto de uma estrutura de classes, e cada classe é denominada *participante* do padrão. A notação utilizada é a *Unified Modeling Language* (UML) especificada pela OBJECT MANAGEMENT GROUP, 2000.

Método da Fábrica

Este padrão define uma interface para criar um objeto, mas deixa que subclasses decidam qual classe a ser instanciada. O Método da Fábrica permite que uma classe transfira a instanciação para subclasses. A Figura II-1 mostra a estrutura básica deste padrão.

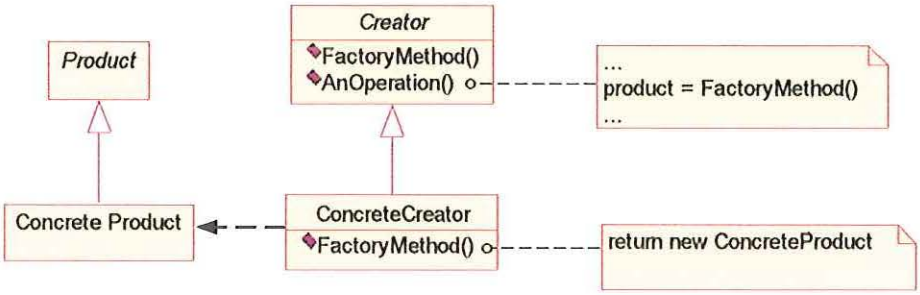

Figura 31. Estrutura do Método da Fábrica em Design Patterns

Para este padrão, os participantes são:

- ***Product***
  - Define a interface dos objetos que o método da fábrica cria.
- ***ConcreteProduct***
  - Implementa a interface de *Product*.
- ***Creator***
  - Declara o método da fábrica, o qual será responsável por criar e retornar um objeto do tipo *Product*. O *Creator* pode definir uma implementação padrão do método da fábrica, que crie um objeto *ConcreteProduct* padrão;
  - Pode chamar o método da fábrica para um objeto do tipo *Product*.
- ***ConcreteCreator***
  - Implementa o método da fábrica especificando as instâncias a serem criadas

O Método da Fábrica foi amplamente utilizado na implementação do editor. Em diversas ocasiões, o programador que implementa o *template* necessita fazer uso da herança para criar subclasses das classes definidas no módulo básico. Por exemplo, esta estrutura é aplicada para a implementação e respectiva criação dos objetos gráficos manipulados pelo *template*.

Composição

Este padrão compõe objetos em estruturas de árvore para representar hierarquias do tipo parte-total. A Composição permite que clientes tratem objetos individuais e composições de objetos uniformemente. A Figura II-2 mostra a estrutura básica deste padrão.

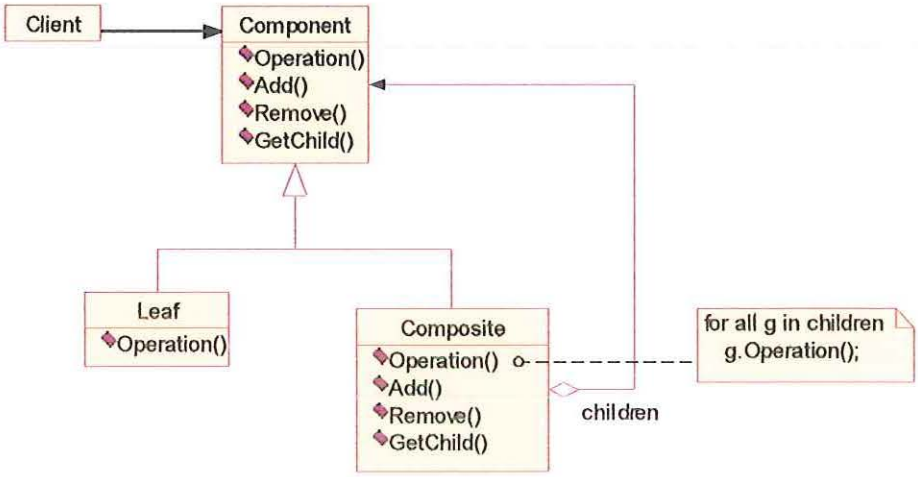

Figura 32. Estrutura da Composição em Design Patterns

Para este padrão, os participantes são:

- ***Component***
  - Declara a interface para os objetos na composição;
  - Implementa um comportamento padrão para a interface, comum a todas as classes, e de forma apropriada;
  - Declara a interface para acesso e gerenciamento dos componentes filhos;
  - Define uma interface para acesso ao pai dos componentes na estrutura recursiva, e a implementa se isto for apropriado (opcional).
- ***Leaf***
  - Representa objetos folha na composição. Uma folha não tem filhos;
  - Define o comportamento para objetos primitivos na composição.

- ***Composite***
  - Define um comportamento padrão para os componentes que tem filhos;
  - Armazena componentes filhos;
  - Implementa operações relacionadas aos filhos na interface de *Component*.
- ***Client***
  - Manipula objetos na composição através da interface de *Component*.

No editor, a Composição foi utilizada principalmente na modelagem dos objetos manipulados, como objetos gráficos e suas respectivas propriedades visualizadas pelo *Object Inspector*.

Método Template

Este padrão define o esqueleto de um algoritmo numa operação, deixando algumas etapas para subclasses. O Método *Template* permite que subclasses redefinam alguns passos do algoritmo sem alterar sua estrutura. A Figura II-3 mostra a estrutura básica deste padrão.

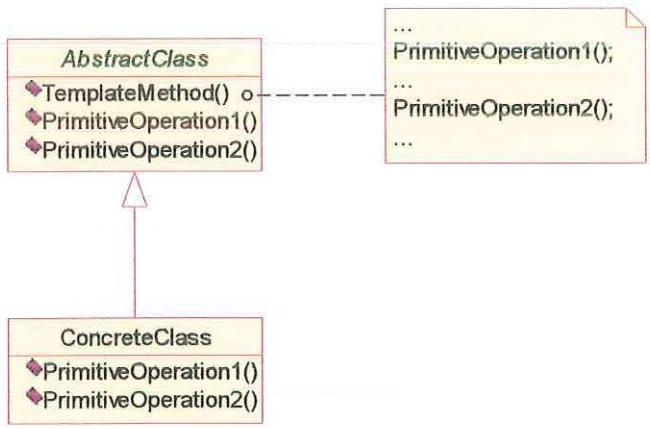

Figura 33. Estrutura do Método Template em Design Patterns

Para este padrão, os participantes são:

- ***AbstractClass***

- Define as operações primitivas para implementar passos de um algoritmo. Podem ser abstratas e implementadas por subclasses concretas, ou então possuir implementação padrão, que pode ser refeita pelas subclasses;

- Implementa o método *template* definindo o esqueleto de um algoritmo. O método *template* chama operações primitivas assim como operações definidas na *AbstractClass* ou de outros objetos.

- ***ConcreteClass***

  - Implementa as operações primitivas para definir passos específicos do algoritmo para a subclasse.

Esse padrão foi amplamente utilizado na implementação do editor, principalmente visando à programação por diferença.