UNIVERSIDADE DE SÃO PAULO

Modelagem e Especificação Utilizando Redes de Petri e Statecharts

Carlos Renato Lisboa Francês
Renata Spolon
Marcos José Santana
Regina Helena Carlucci Santana

Nº 45

NOTAS DIDÁTICAS

Instituto de Ciências Matemáticas de São Carlos

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2585

Modelagem e Especificação Utilizando Redes de Petri e Statecharts

Carlos Renato Lisboa Francês
Renata Spolon
Marcos José Santana
Regina Helena Carlucci Santana

Nº 45

NOTAS DIDÁTICAS

São Carlos – SP
Ago./2000

Modelagem e Especificação utilizando Redes de Petri e Statecharts

Carlos Renato Lisboa Francês
Departamento de Informática
Universidade da Amazônia - Unama
Doutorando – ICMC/USP

Renata Spolon
Departamento de Computação e Estatística
Centro de Ciências Exatas e Tecnologia
Universidade Federal de Mato Grosso do Sul
Doutoranda – IFSC/USP

Marcos José Santana
Regina Helena Carlucci Santana
Departamento de Ciênicas de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

São Carlos, agosto de 2000

Os autores expressam seus agradecimentos à Capes, Fapesp, Fidesa e CNPq pelo auxílio financeiro.

Sumário

1. Introdução

No decorrer deste texto, pretende-se traçar um mapa sobre representação de sistemas (computacionais ou não), utilizando-se duas técnicas bastante difundidas na literatura: Redes de Petri e *Statecharts*. A idéia básica é disponibilizar um material que sirva de referência inicial sobre o assunto e, além disso, que possua um certo grau de profundidade a respeito do tema.

Quando se pretende representar um sistema, pode-se realizar essa tarefa sob dois pontos de vista distintos: (1) sob a ótica dos procedimentos que o sistema deve realizar (e os dados que esses procedimentos vão manipular), ou (2) sob a ótica das situações (estados) nas quais o sistema pode vir a se encontrar, com o passar do tempo. O primeiro método é, por exemplo, utilizado nos Diagramas de Fluxo de Dados (*Data Flow Diagram* - DFD), nos quais o sistema é visto como uma coleção de procedimentos que devem ser tomados para a realização de uma determinada tarefa. Nos DFD prevalece a utilização de verbos no infinitivo ou imperativo para denotar a execução de um procedimento. A figura 1.1 (Pressman, 1997) apresenta um DFD de nível 1[1] para um sistema de segurança doméstico.

[1] O incremento de nível (nível 2, nível 3, ... , nível n) significa um aumento no nível de detalhes, diminuindo-se a abstração da especificação.

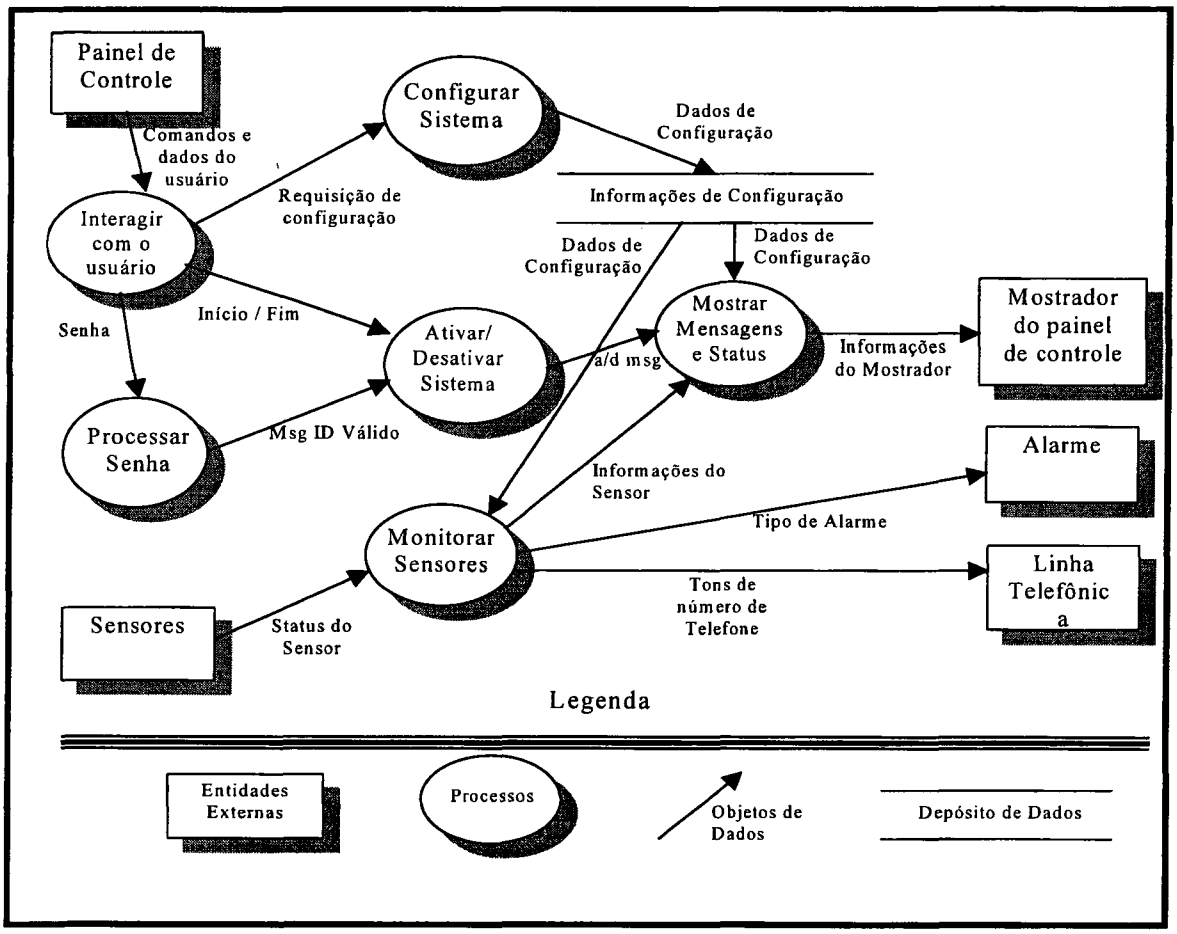

Figura 1.1. DFD de Nível 1 de um Sistema de Segurança Doméstico.

No diagrama anterior, o sistema é representado através de seus procedimentos (processos) e do fluxo de dados que esses processos geram. Alguns autores (por exemplo, (Pressman, 1997)) citam representações desse tipo como modelagem funcional, pelo fato do sistema ser todo apresentado através de suas funções e das transformações que essas funções provocam nos dados.

Uma outra maneira de se representar um sistema é através das situações nas quais esse sistema pode se encontrar com o passar do tempo. As mudanças de situações (estados) são motivadas pela ocorrência de determinados eventos. Evento, neste contexto, é um termo que denota qualquer fenômeno que faça com que o sistema transicione de um estado de origem para um estado de destino. Como exemplo de técnicas que se utilizam de estados/eventos, podem-se citar as máquinas de estados finitos, as redes de Petri e os *Statecharts* (sendo que as duas últimas são o foco central deste estudo).

O termo modelagem aqui possui um escopo mais amplo do que o atribuído ao termo especificação. Modelagem é constituída por um conjunto de fases (figura 1.2), onde a especificação é apenas a primeira. Assim, o termo especificação (e sua respectiva representação) será substituído neste texto por modelagem, que possui um caráter mais abrangente.

Além da especificação pura de um sistema, algumas técnicas (como as duas aqui abordadas) permitem que sejam feitas avaliações de desempenho do sistema modelado, extrapolando as visões semântica e sintática do sistema. Avaliar desempenho implica oferecer determinadas medidas (geralmente relacionadas ao tempo) que forneçam uma noção de quão satisfatório está o funcionamento do sistema. As técnicas para análise de desempenho utilizam métodos matemáticos como base (geralmente métodos estocásticos), mas a utilização desses métodos é, *a priori*, independente da representação do modelo. A figura 1.2 ilustra a relação (e a independência) entre os níveis de uma modelagem de um determinado sistema.

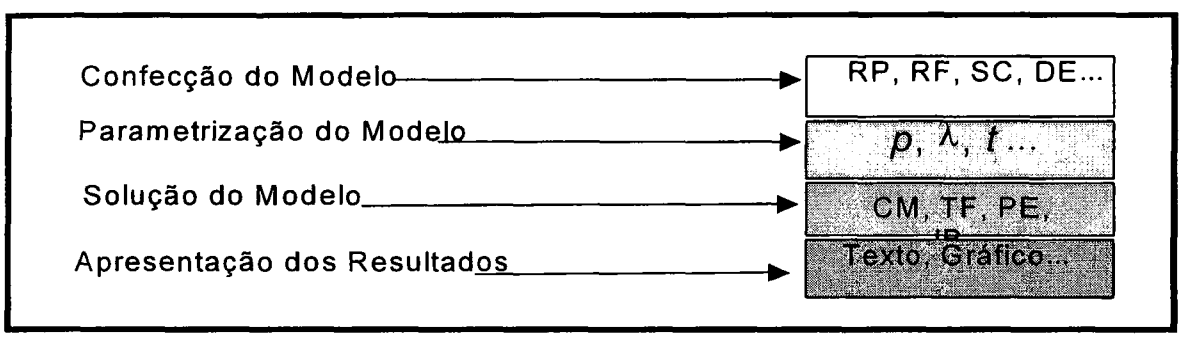

Figura 1.2. Arquitetura de uma Modelagem.

No nível mais alto da figura 1.2, estão as técnicas para confecção do modelo, como, por exemplo, Redes de Petri (RP), Redes de Filas (RF), *Statecharts* (SC) e Diagramas de Estados (DE). A escolha de uma dessas técnicas é independente dos demais níveis. No nível de parametrização, são incluídos os valores dos parâmetros relevantes para a análise (por exemplo, probabilidades (p), taxas (λ), tempos (t)). A solução do modelo é provida segundo um método matemático (probabilístico), como Cadeias de Markov (CM), Teoria de filas (TF), Processos Estocásticos (PE) e Inferência Bayesiana (IB). No nível mais baixo da modelagem está a apresentação (sob forma de texto, gráficos, entre outros) dos resultados obtidos pela solução do modelo. Durante

este texto, serão enfocados os níveis de confecção e parametrização dos modelos (exceto nos tópicos cujo teor for de avaliação de desempenho, nos quais toda a arquitetura da modelagem será efetivada).

Técnicas baseadas em estados possuem uma aplicação mais geral, pois dão uma visão mais realista de um sistema: uma visão que está relacionada ao transcorrer do tempo. Além disso, essas técnicas possuem um casamento mais natural com os métodos matemáticos estocásticos, cuja fundamentação também é baseada no tempo e em mudanças de estados.

Neste trabalho são apresentadas duas técnicas de modelagem baseadas em estados: as redes de Petri e os *Statecharts*. Ambas têm princípios semelhantes, sendo que cada uma possui vantagens e desvantagens que serão abordadas no decorrer do texto. Assim, estruturalmente o roteiro seguido é dividido em duas partes independentes (que poderiam perfeitamente ser separadas): a primeira abordando a técnica redes de Petri, e a segunda apresentando a técnica *Statecharts*. Nas duas partes, além do caráter de especificação, também sempre é abordado o enfoque de avaliação de desempenho. Ainda são apresentadas ferramentas que implementam as peculiaridades de cada técnica e, através das quais, foram testados todos os exemplos aqui apresentados.

Há ainda uma análise comparativa entre as técnicas abordadas, sugerindo-se possíveis correlações, deficiências e até melhorias futuras. Nesse ponto, o leitor já estará habilitado a fazer uma escolha entre as abordagens apresentadas, de acordo com as necessidades do sistema a ser modelado e/ou preferências pessoais.

2. Redes de Petri

Rede de Petri é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática (Maciel et al., 1996). Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos (Valette & Courvoisier, 1980).

A representação gráfica de uma rede de Petri básica é formada por dois componentes: um ativo chamado de transição (barra) e outro passivo denominado lugar (círculo). Os lugares equivalem às variáveis de estado (possíveis situações do sistema)

e as transições correspondem às ações realizadas pelo sistema. Esses dois componentes são ligados entre si através de arcos dirigidos. Os arcos podem ser únicos ou múltiplos. A figura 2.1 mostra os elementos básicos de um grafo associado às redes de Petri.

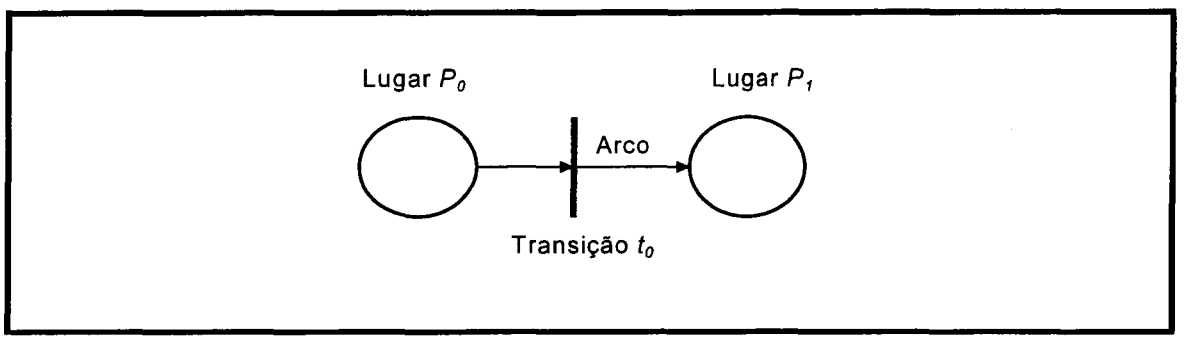

Figura 2.1. Grafo e seus Elementos Básicos.

2.1. Definições

As redes de Petri podem ser enfocadas através de três fundamentações diferentes. A primeira utiliza a teoria *bag* como suporte. A segunda usa os conceitos da álgebra matricial. A última se fundamenta na estrutura definida por relações. A seguir são apresentadas as definições formais de cada uma dessas fundamentações.

2.1.1. Fundamentações para Redes de Petri

- **Definição 1**: uma rede de Petri **R** é uma quíntupla R = (P, T, I, O, K), onde P = $\{p_1, p_2,...,p_n\}$ é um conjunto finito não-vazio de lugares, T = $\{t_1, t_2,..., t_m\}$ é um conjunto finito não-vazio de transições. I : T $\rightarrow$ P é um conjunto de *bags* [2] que representa o mapeamento de transições para lugares de entrada. O : T $\rightarrow$ P é um conjunto de *bags* que representa o mapeamento de transições para lugares de saída. K : P $\rightarrow$ N é o conjunto da capacidades associadas a cada lugar, podendo assumir um valor infinito (Peterson, 1981).

[2] *Bag* é uma generalização do conceito de conjunto que admite a repetição de elementos. Na notação de *bags*, utiliza-se [ ], enquanto que para denotar conjuntos, utiliza-se { } (Maciel et al., 1996).

Para exemplificar a definição 1, supõe-se que se deseje representar um ano letivo de uma Universidade. O ano letivo começa com o primeiro período (semestre) letivo, seguido das primeiras férias (de julho), logo após, tem-se o segundo período letivo, e finalmente as férias de final de ano. Assim, o ano letivo poderia ser representado conforme a figura 2.2.

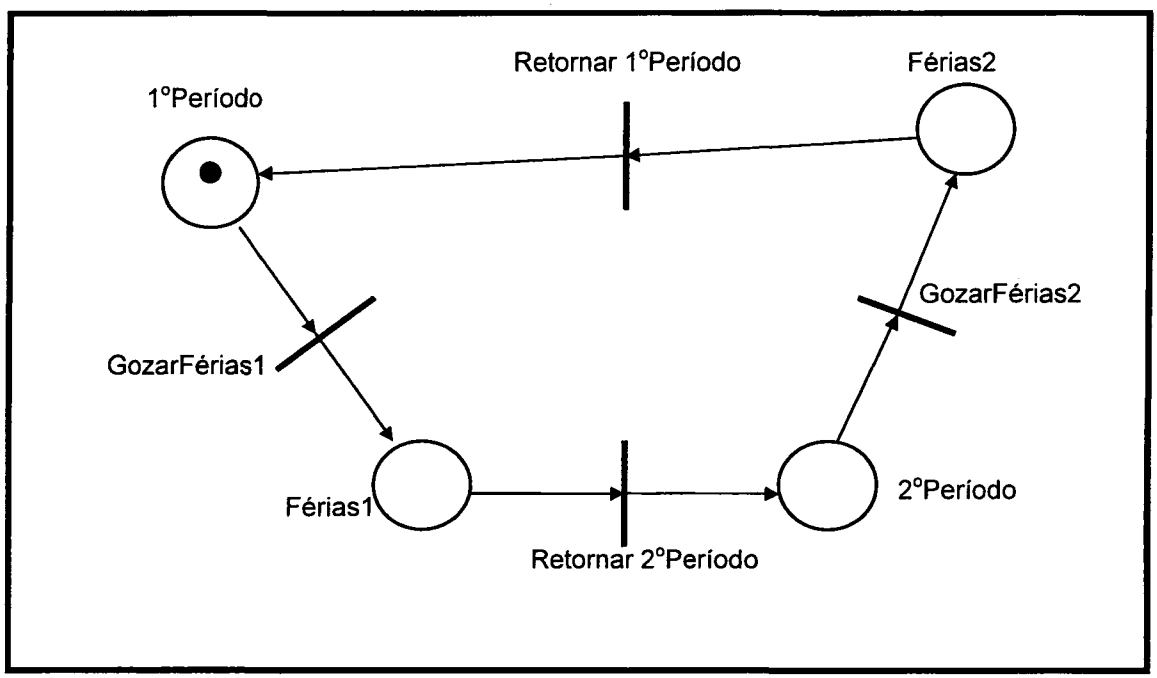

Figura 2.2. Ano Letivo Representado Graficamente em Redes de Petri.

A figura 2.2 pode ser descrita da seguinte forma, utilizando-se a definição 1:

R_{Ano_Letivo} = (P, T, I, O, K), onde

O conjunto de lugares P é

P = {1°Período, Férias1, 2°Período, Férias2};

O conjunto de transições T é

T = {GozarFérias1, Retornar2°Período, GozarFérias2, Retornar1°Período};

O conjunto de *bags* de entrada I é

I = { I (GozarFérias1) = [1°Período], I (Retornar2°Período) = [Férias1],
I (GozarFérias2) = [2°Período], I (Retornar1°Período) = [Férias2] };

O conjunto de *bags* de saída O é

O = {O (GozarFérias1) = [Férias1], O (Retornar2°Período) = [2°Período],
O (GozarFérias2) = [Férias2], O (Retornar1°Período) = [1°Período] };

e o conjunto de capacidades dos lugares é

$K = \{ K_{1oPeríodo} = 1, K_{Férias1} = 1, K_{2oPeríodo} = 1, K_{Férias2} = 1\}$.

- **Definição 2:** a estrutura de uma rede de Petri, segundo o ponto de vista matricial, é uma quíntupla R = (P, T, I, O, K), onde P é um conjunto finito de lugares. T é um conjunto finito de transições, I : P x T $\rightarrow$ N é a matriz de pré-condições. O : P x T $\rightarrow$ N é a matriz de pós-condições. K é o vetor das capacidades associados aos lugares (K : P $\rightarrow$ N) (Peterson, 1981).

Tomando-se como base novamente a figura 2.2, tem-se:

Os conjuntos de lugares e transições são idênticos àqueles vistos para a definição1.

A matriz I (pré-condições) é

I =	GozarFérias1	Retornar2°Período	GozarFérias2	Retornar1°Período	
	1	0	0	0	1°Período
	0	1	0	0	Férias1
	0	0	1	0	2°Período
	0	0	0	1	Férias2

A matriz O (pós-condições) é:

	GozarFérias1	Retornar2°Período	GozarFérias2	Retornar1°Período	
O =	0	0	0	1	1°Período
	1	0	0	0	Férias1
	0	1	0	0	2°Período
	0	0	1	0	Férias2

É importante ressaltar que as matrizes I e O representam as pré e pós-condições, respectivamente, de todas as transições da rede, isto é, as matrizes são construídas em função das transições e não dos lugares.

- **Definição 3:** a estrutura de redes de Petri, usando-se relações, é formada por uma quíntupla R = (P, T, A, V, K), onde P é o conjunto de lugares, T o de transições, A o conjunto dos arcos e V corresponde ao conjunto de valorações desses arcos. Os elementos de A são arcos que conectam transições a lugares ou lugares a transições ($A \subseteq (P \times T) \cup (T \times P)$). Assim, os elementos de A podem ser agrupados em dois subconjuntos - o conjunto das entradas às transições e o de saída às transições, I = $\{(p_i, t_j)\}$ e O = $\{(t_j, p_i)\}$, respectivamente (Murata, 1989).

Tomando-se ainda como referência a figura 2.2, tem-se que os conjuntos de lugares (P), de transições (T) e de capacidades (K) permanecem inalterados. Entretanto, na notação que utiliza relações, há o surgimento de dois novos conjuntos: o conjunto de arcos (A) e o conjunto de valores para esses arcos (V).

O conjunto de arcos A é

A = { (1°Período, GozarFérias1), (GozarFérias1, Férias1),
(Férias1, Retornar2°Período), (Retornar2°Período, 2°Período),
(2°Período, GozarFérias2), (GozarFérias2, Férias2),
(Férias2, Retornar1°Período), (Retornar1°Período, 1°Período) }

O conjunto de valores dos arcos V é

V = {1, 1, 1, 1, 1, 1, 1, 1}

2.2. Redes de Petri Marcadas

Marcas (*tokens*) são informações atribuídas aos lugares, para representar a situação (estado) da rede em um determinado momento. Define-se uma rede de Petri marcada pela dupla RM = (R, M_0), onde R é a estrutura da rede e M_0 a marcação inicial (Maciel et al., 1996). Assim, para simular o comportamento dinâmico dos sistemas, a marcação da rede de Petri é modificada a cada ação realizada (transição disparada). A figura 2.3 ilustra uma rede marcada.

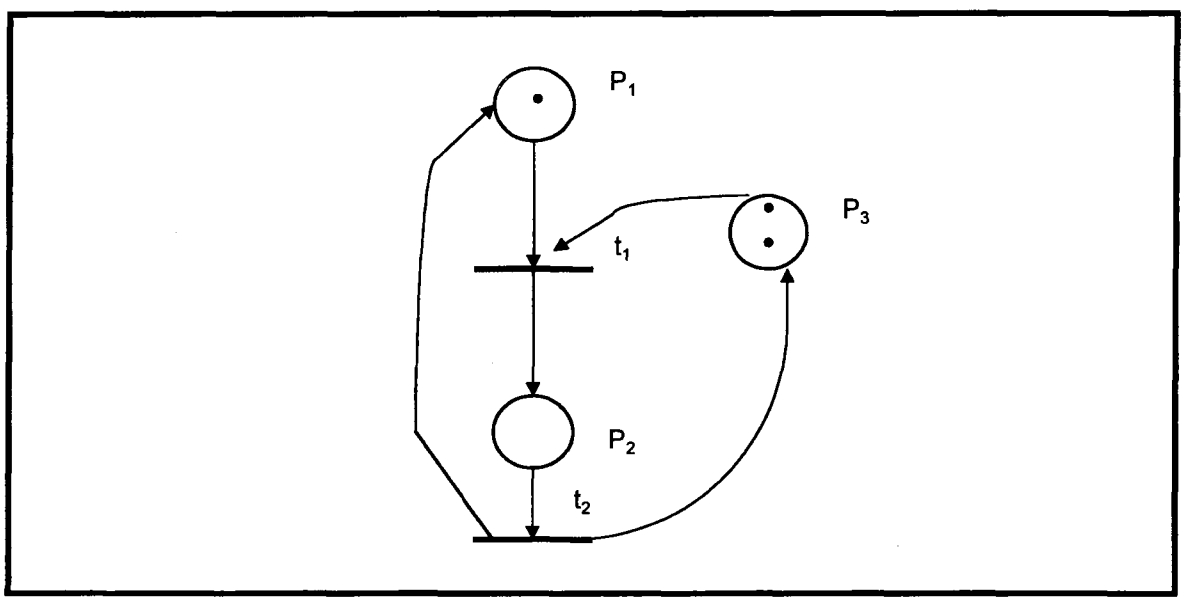

Figura 2.3. Rede Marcada.

2.3. Notações Particulares

Em alguns casos, deseja-se representar a diferença entre transições, visando melhorar a clareza do modelo. Além disso, em muitas situações, pretende-se representar a execução de uma condição externa ao sistema modelado. Para representar rótulos de transições, utiliza-se um alfabeto qualquer associado à rede (por exemplo, o alfabeto a,b,c,....,z). Para representar as condições externas, usa-se o mesmo esquema utilizado para rotular transições, entretanto, os símbolos vêm entre parênteses (conforme ilustra a figura 2.4).

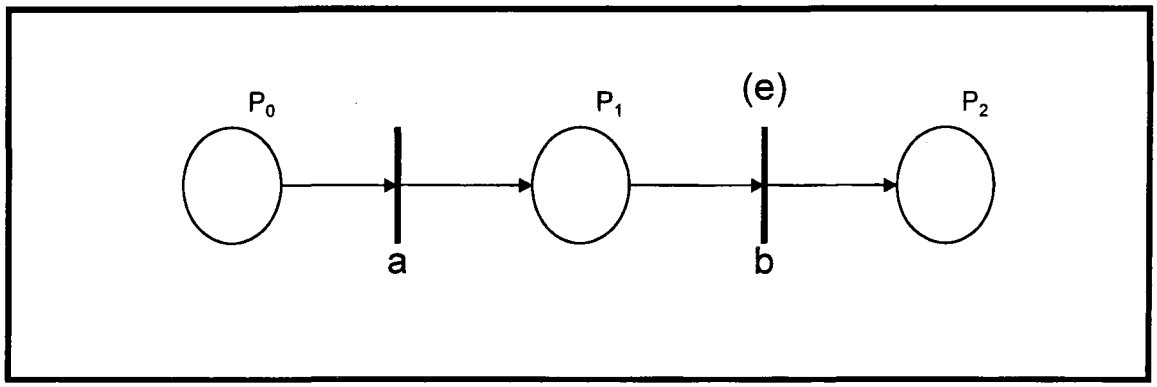

Figura 2.4. Rótulos e Condições Externas às Transições.

2.4. Classes das Redes de Petri

Podem-se agrupar as redes de Petri em duas grandes classes: Ordinárias e Não-Ordinárias (de Alto nível) (Maciel et al., 1996). As redes ordinárias utilizam apenas um tipo básico de marcas, o tipo inteiro não-negativo. Fazendo-se uma analogia às linguagens de programação, as redes de alto nível podem possuir marcas mais sofisticadas, como (em linguagens de programação) os tipos de dados definidos pelo usuário ou ainda os tipos compostos que são formados por vários tipos mais elementares (por exemplo, na linguagem C, *Structs*). As redes ordinárias se subdividem em:

- Rede Binária: é a rede mais elementar dentre todas. Essa rede permite, no máximo, um *token* em cada lugar, e todos os arcos possuem valor unitário.

- Rede *Place-Transition*: é o tipo de rede que permite o acúmulo de marcas no mesmo lugar, assim como valores não unitários para os arcos.

As redes de alto nível se diferenciam das ordinárias basicamente por individualizarem os *tokens*. Assim, por exemplo, em um mesmo lugar "fruteira" pode haver os *tokens* maçã, banana e laranja (cada um de um tipo diferente). Esse tipo de rede permite a individualização de uma marca (pertencente a um grupo) em um mesmo lugar. Essa individualização pode ser realizada através de vários artifícios, como por exemplo, cor da marca ou objetos representando os *tokens*. Redes não-ordinárias não

aumentam o poder de representação de um modelo. Entretanto, elas permitem uma maior clareza e um maior (ou menor) nível de abstração ao modelo.

2.5. Redes Elementares

Nesta seção, são apresentadas algumas redes, a partir das quais derivam outras redes mais complexas. São discutidas as redes representativas de seqüenciamento, distribuição, junção, escolha não-determinística e atribuição.

- Seqüenciamento: é a rede que representa a execução de uma ação, desde que uma determinada condição seja satisfeita. Após a execução dessa ação, pode-se ter outra ação, desde que seja satisfeita uma determinada condição (figura 2.5).

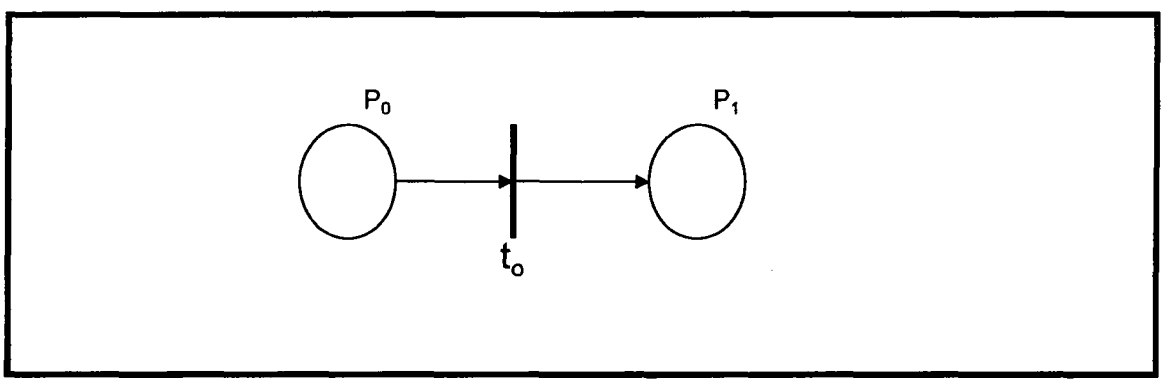

Figura 2.5. Seqüenciamento.

- Distribuição: é a rede elementar utilizada na criação de processos paralelos a partir de um processo pai. Os processos filhos são criados através da distribuição dos *tokens* encontrados no processo (lugar) pai. A distribuição é mostrada na figura 2.6. É importante observar que se houvesse um *token* em P_1, ele seria propagado (obrigatoriamente) tanto para P_2 quanto para P_3.

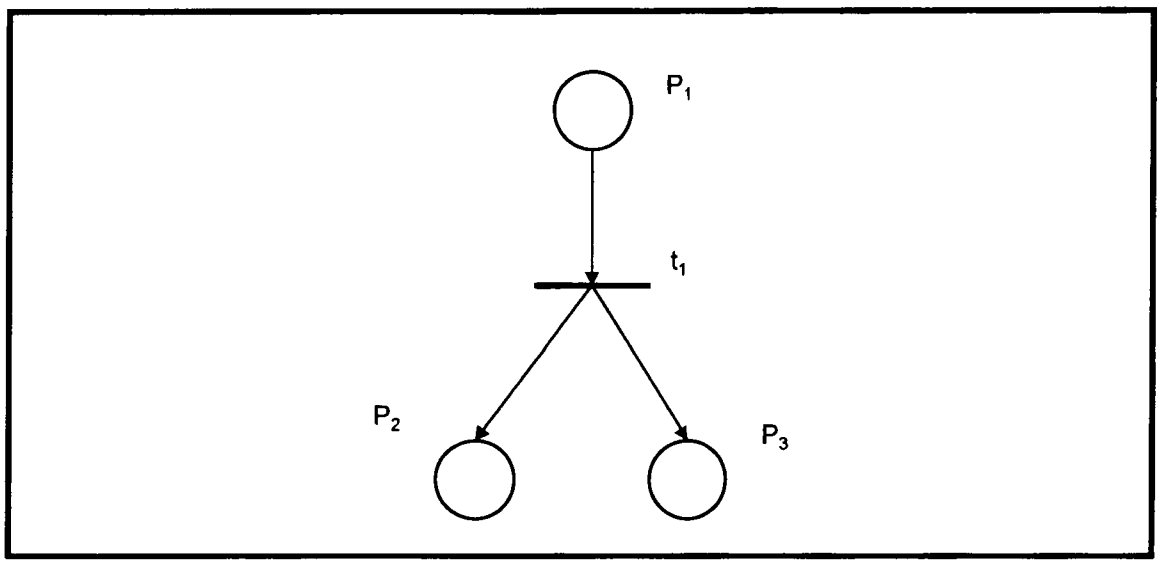

Figura 2.6. Distribuição.

- Junção: é a rede usada para sincronizar atividades concorrentes. No exemplo da figura 2.7, a transição t_1 só dispara quando existirem marcas tanto em P_1, quanto em P_2, estabelecendo, assim, o sincronismo.

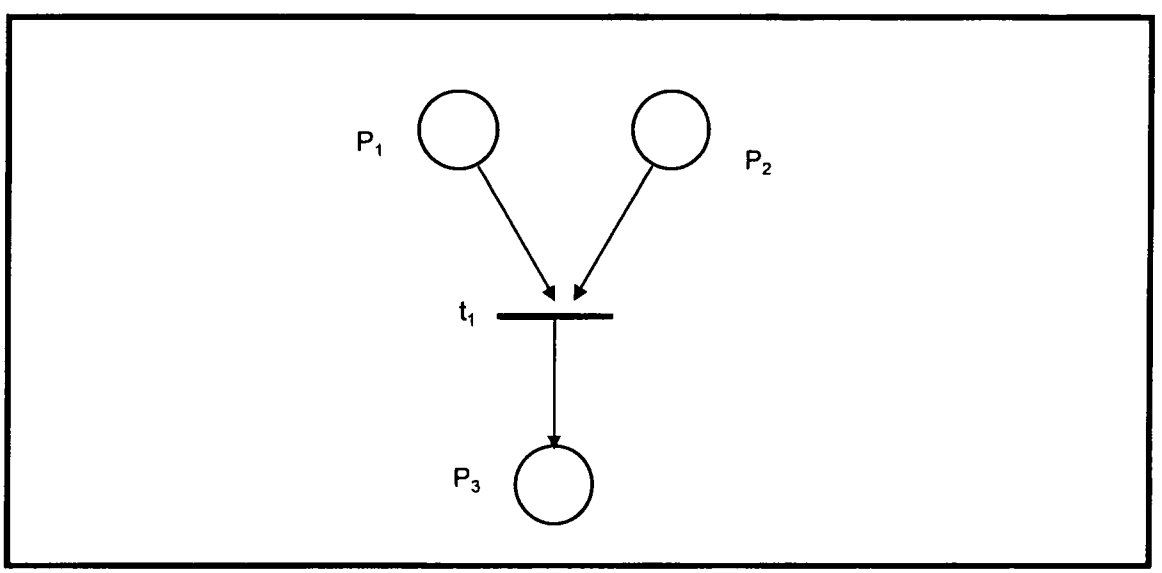

Figura 2.7. Junção.

- Escolha Não-Determinística: é uma rede que ao se disparar uma transição, inabilita-se a outra. Entretanto, não existe possibilidade de escolha (conforme figura 2.8). O fator não-determinístico dessa rede gera uma situação chamada de conflito. O conflito pode ser classificado como estrutural ou efetivo. Ambos os conflitos estão

associados ao fato de duas transições possuírem o mesmo lugar como entrada. Porém, se a rede não possuir *tokens*, o conflito é dito estrutural. Contudo, se há uma única marca no lugar comum às transições, diz-se que o conflito é efetivo. A figura 2.9 (Maciel et al., 1996) ilustra os dois tipos de conflito.

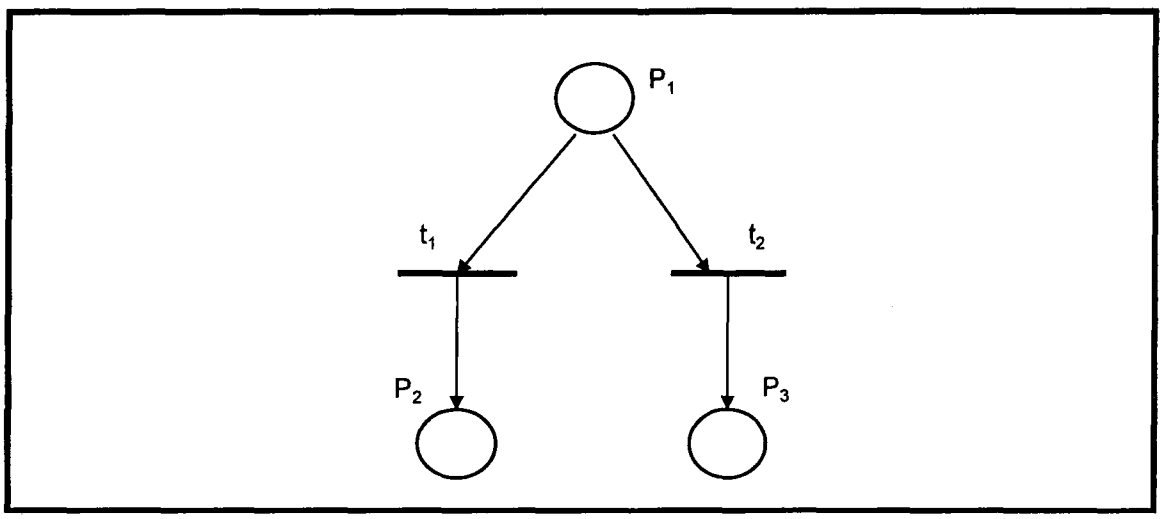

Figura 2.8. Escolha Não-Determinística.

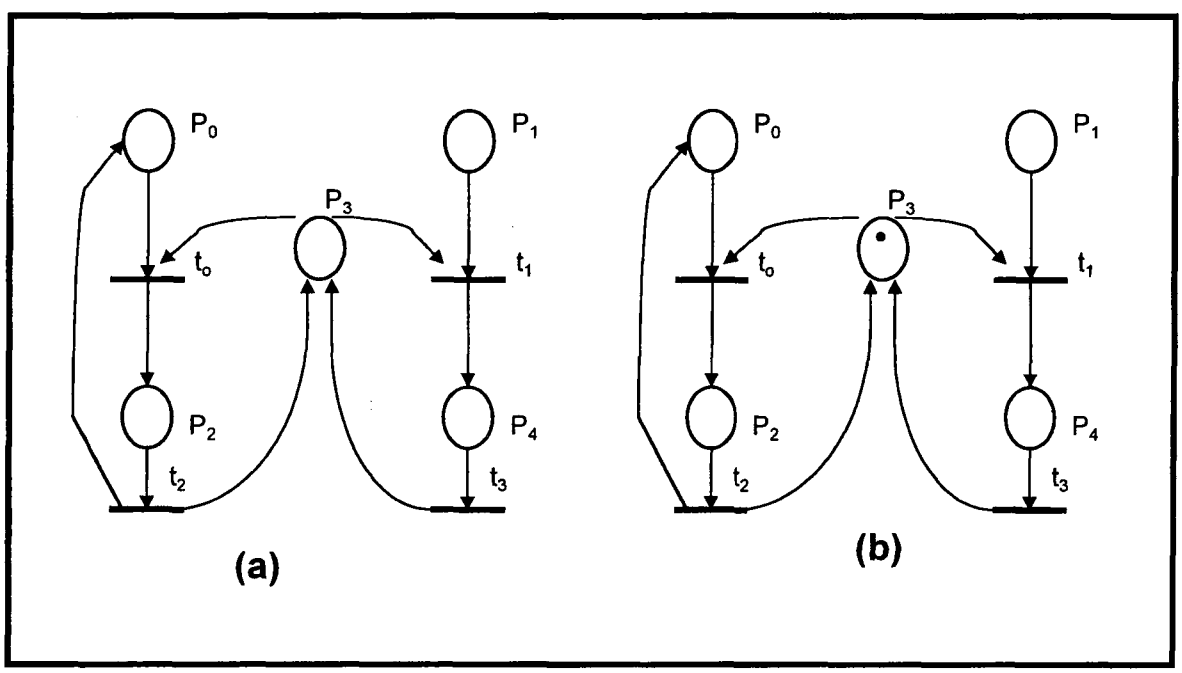

Figura 2.9. **(a)** Conflito Estrutural e **(b)** Conflito Efetivo.

A escolha determinística da transição a ser disparada não é um recurso abordado nas redes elementares. Porém, essa deficiência das redes de Petri originais é resolvida em algumas extensões propostas, abordadas em seções posteriores.

2.6. Propriedades das Redes de Petri

As redes de Petri permitem que sejam feitas certas inferências sobre as mesmas, de acordo com algumas propriedades. Essas propriedades podem ser comportamentais e estruturais, em função da utilização ou não da marcação. Se, na análise, está sendo utilizada a marcação (dependente da marcação), as propriedades são ditas comportamentais. Caso contrário (não dependente de marcação), as propriedades são ditas estruturais. Por possuir maior aplicabilidade, aqui são apresentadas apenas as propriedades comportamentais.

2.6.1. Propriedades Comportamentais

As propriedades comportamentais dizem respeito às alterações sofridas pela marcação, através do disparo de uma seqüência de transições.

- Alcançabilidade

A propriedade de alcançabilidade é aquela que permite verificar se uma determinada marcação pode ser alcançada, partindo-se da marcação inicial e sendo disparado um número finito de transições. Essa propriedade é bastante interessante para detectar possíveis impasses (*deadlocks*) em sistemas que são propícios a essas ocorrências. Se uma determinada marcação representar a ocorrência de *deadlocks*, então a verificação da alcançabilidade dessa marcação irá detectar a presença de um *deadlock*.

Um exemplo a respeito do assunto pode ser visto em um sistema onde há dois processos (1 e 2) e dois recursos (1 e 2). O processo1 detém o recurso1, e o processo2 detém o recurso2. Após isso, o processo1 deseja obter o recurso2, e o processo2 deseja obter o recurso1, conforme mostra a figura 2.10.

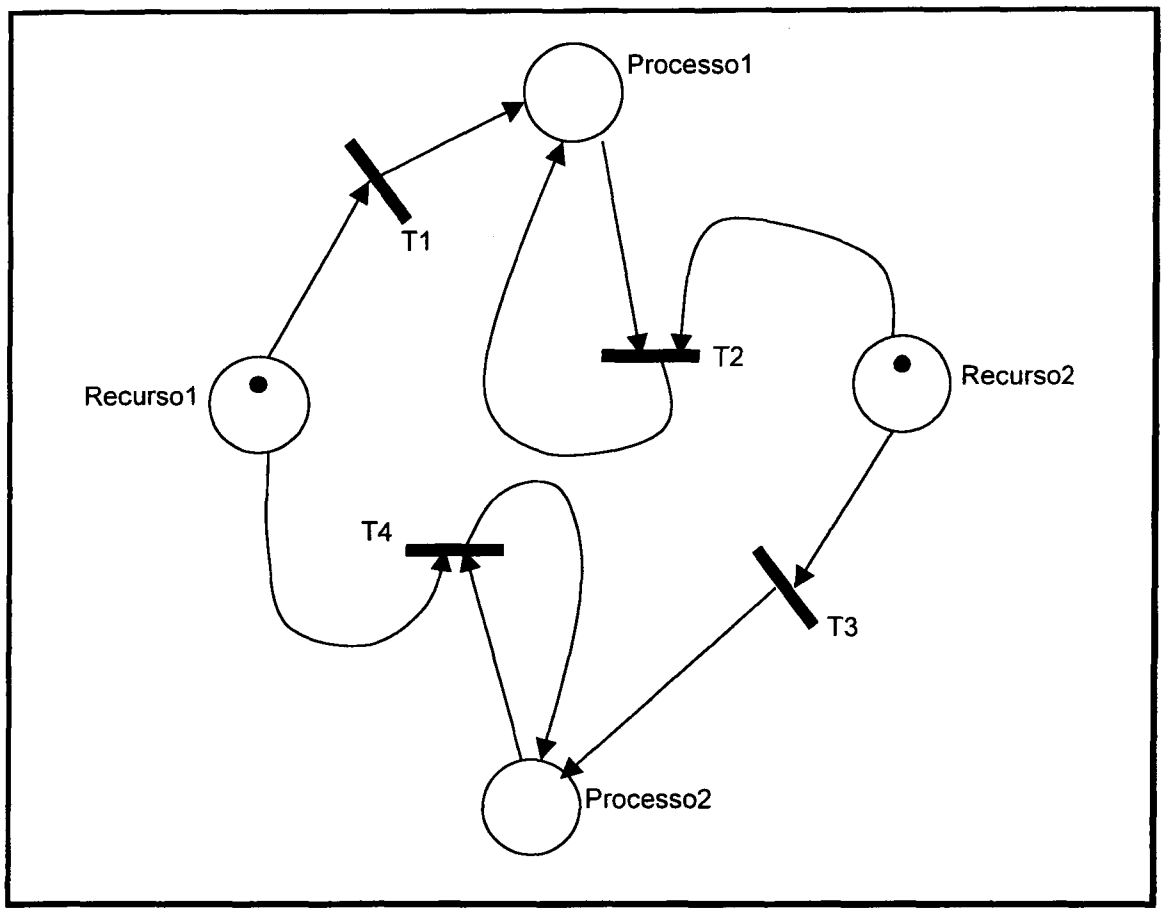

Figura 2.10. Seqüência de Disparos que leva a um *Deadlock*.

Com a marcação inicial da figura 2.10, a seqüência de disparo (T1, T3), isto é, o processo1 toma posse do recurso1, e o processo2 toma posse do recurso2, leva a rede ao *deadlock*, pois o processo1 consome o *token* do lugar recurso1, o processo2 consome o *token* do lugar recurso2, e ambos os processos ficam à espera de um *token* inexistente no recurso que não estão de posse.

A obtenção de uma marcação que leva a rede a um impasse é um processo iterativo. No exemplo trivial apresentado, a solução pode ser facilmente encontrada manualmente. Mas, à medida que o tamanho do modelo aumenta, é recomendável que seja utilizado um método de análise para verificar essa propriedade. Por exemplo, árvore de cobertura e equação fundamental das redes de Petri (EFRP), apresentadas mais adiante, são dois métodos possíveis.

- Limitação (*Boudedness*)

A propriedade limitação se refere à quantidade máxima de marcas que pode conter em um determinado lugar, em qualquer marcação possível. Assim, se um lugar pode possuir até duas marcas, em quaisquer marcações possíveis para a rede, então esse lugar é dito 2-limitado. Por extensão, se todos os lugares da rede são limitados, então a rede também o é. O exemplo apresentado a seguir (figura 2.11) possui todos os lugares limitados, possuindo, no máximo três marcas em qualquer marcação alcançável. Assim, a rede é considerada 3-limitada.

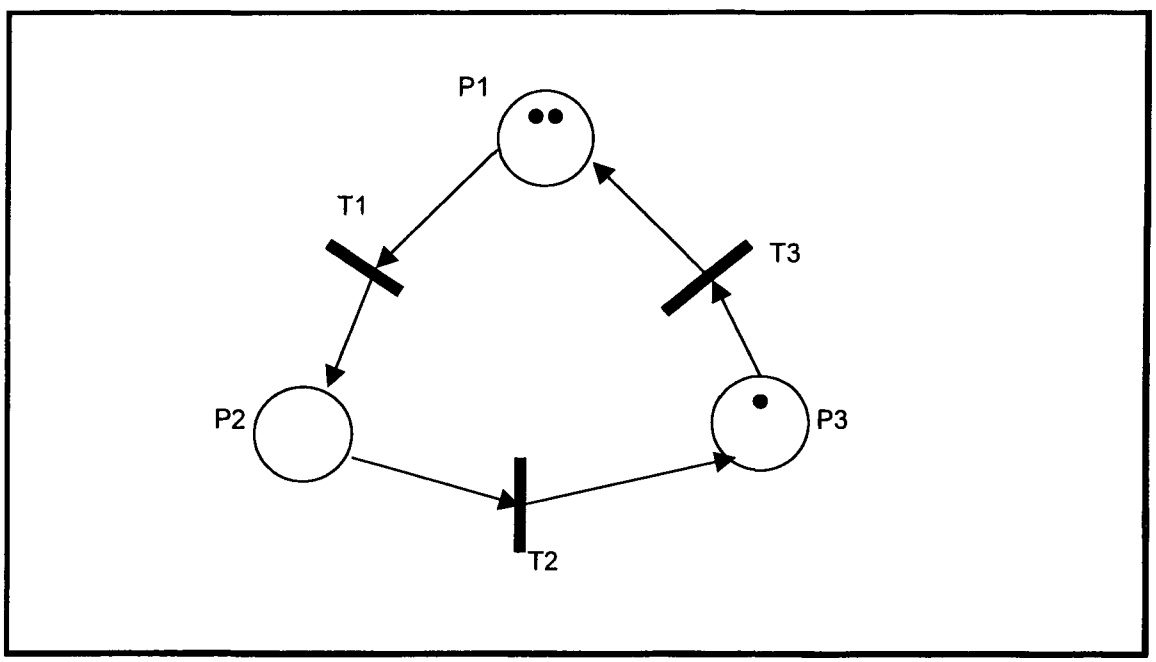

Figura 2.11. Rede 3-Limitada (3-*Bouded*)

- Segurança (*Safeness*)

A propriedade segurança é uma particularização da propriedade anterior (limitação). Se um lugar possui uma limitação de apenas um *token* (1-limitado), então ele é dito um lugar seguro (*safe*). Por extensão, se todos os lugares da rede são seguros, então a rede é considerada segura.

A segurança pode ser usada em qualquer situação binária (existe um ou nenhum *token* em um determinado lugar). Por exemplo, uma função AND poderia ser representada pela rede segura da figura 2.12. Existem dois lugares de entrada (P1 e P2) e um lugar de saída (P3). Se os lugares são seguros, ou seja, se tem uma marca

ou nenhuma, para habilitar a transição AND, é preciso ter uma marca tanto em P1 quanto em P2 (duas entradas verdadeiras fornecem uma saída verdadeira. Em qualquer outra combinação, a saída é falsa).

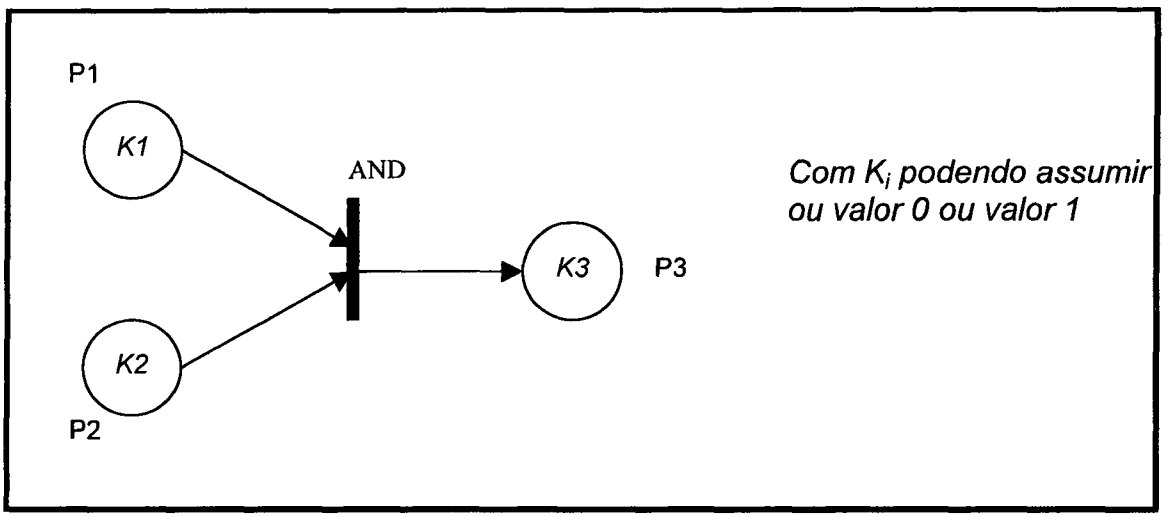

Figura 2.12. Função AND Baseada em uma Rede Segura.

2.7. Extensões às Redes de Petri

Nesta seção são vistas algumas extensões propostas com a finalidade de aumentar a aplicação das redes de Petri. São discutidas as redes de Petri com arco inibidor, coloridas, hierárquica e temporizadas determinísticas.

2.7.1. Redes de Petri com Arco Inibidor

Observando-se a figura 2.13, pode-se considerar que se deseja estabelecer uma ordem de disparo, isto é, T1 precede T2. Para essas situações pode-se utilizar o recurso do arco inibidor (arco com um pequeno círculo na extremidade), conforme mostra a figura 2.14.

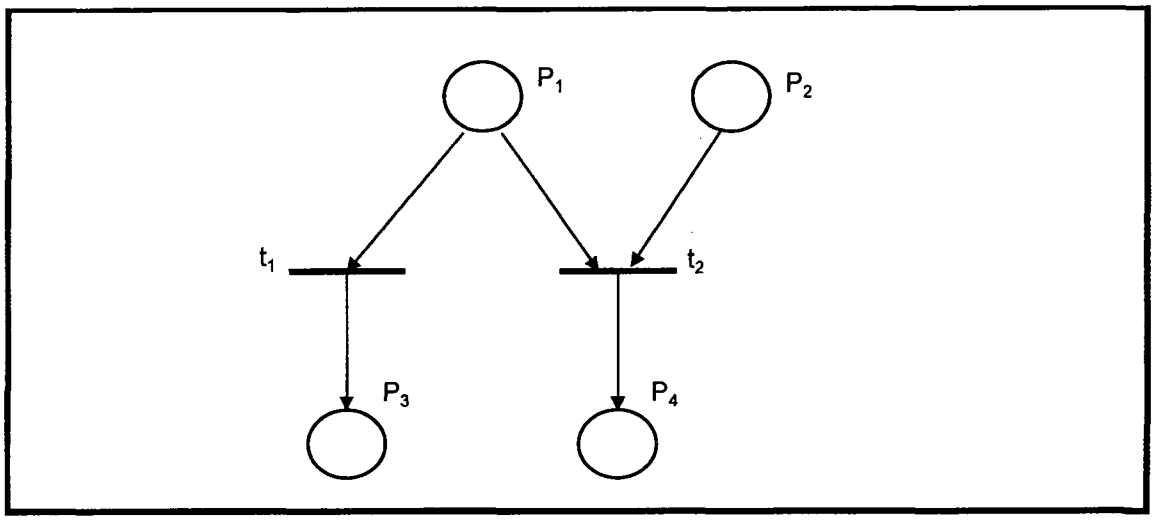

Figura 2.13. Não-determinismo no disparo de T1 e T2.

A nova regra de disparo é que as transições que têm como entrada arcos inibidores só serão habilitadas quando, em seus lugares de entrada, não houver mais *tokens*. Essa condição é chamada de *teste de zero marcas*. Assim, retomando-se o exemplo da figura 2.13, a transição T2 só será habilitada quando, no lugar P1, não houver mais nenhuma marca.

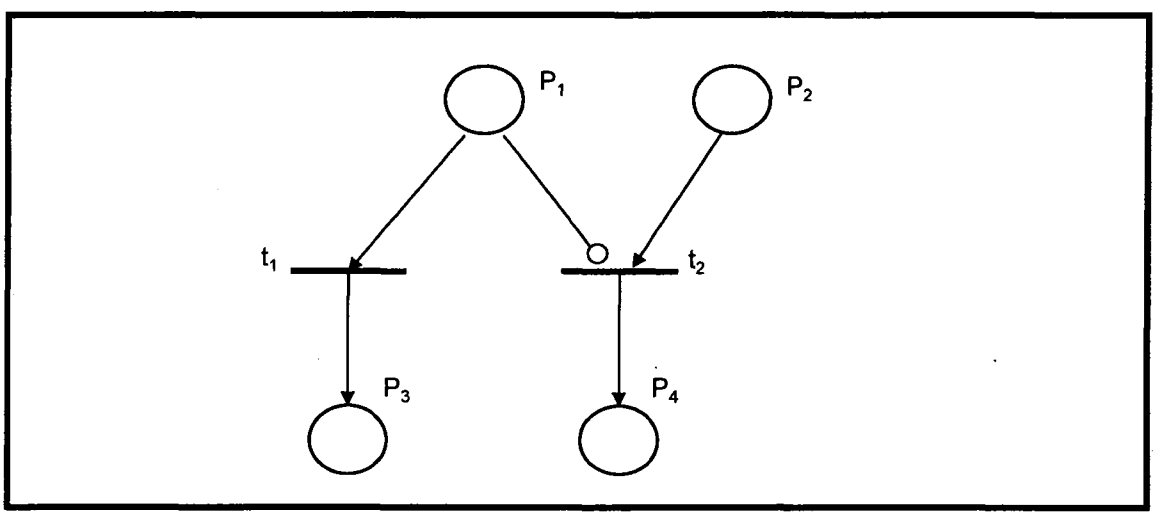

Figura 2.14. Rede com Arco Inibidor.

2.7.2. Redes de Petri Coloridas

As redes de Petri coloridas têm por objetivo reduzir o tamanho do modelo, permitindo que os *tokens* sejam individualizados, através de cores atribuídas a cada um deles. Assim, diferentes clientes, processos ou recursos podem ser representados em

uma mesma rede, o que é bastante conveniente em modelos que representam eficientemente sistemas reais. As cores não significam apenas cores ou padrões. Elas podem representar tipos de dados complexos, usando a nomenclatura de colorida apenas para referenciar a possibilidade de distinção entre os *tokens* (Jensen et al., 1990). A figura 2.15 apresenta uma rede colorida, possuindo a representação original, onde são realmente utilizadas cores para os *tokens*. Nessa figura, os arcos são rotulados com cores (DNS, Telnet, FTP, HTTP).

Nesse exemplo, há um cliente TCP/IP que pode solicitar quatro tipos de serviços a quatro servidores diferentes (o servidor DNS, o servidor Telnet, o servidor FTP e o servidor HTTP). Então, as solicitações do cliente devem ser diferenciadas, de acordo com o servidor que vai atendê-las. A associação de cores e marcas individualizadas para cada requisição, e as restrições colocadas nos arcos determinam quais marcas podem atravessar esses arcos. Na figura 2.15, utiliza-se o modo mais elementar de redes coloridas, no qual se associa ao arco uma determinada cor. Assim, o *token* se destinará ao arco cuja cor for idêntica a da marca. Desse modo, pode-se perceber que os *tokens* de "Cliente" não habilitarão as transições t_1 e t_2, pois os arcos que ligam Cliente às transições t_1 e t_2 só aceitam cores do tipo Telnet e FTP, e o lugar Cliente só possui marcas do tipo DNS e HTTP. O que significa dizer que o cliente só pode solicitar serviços de DNS e HTTP.

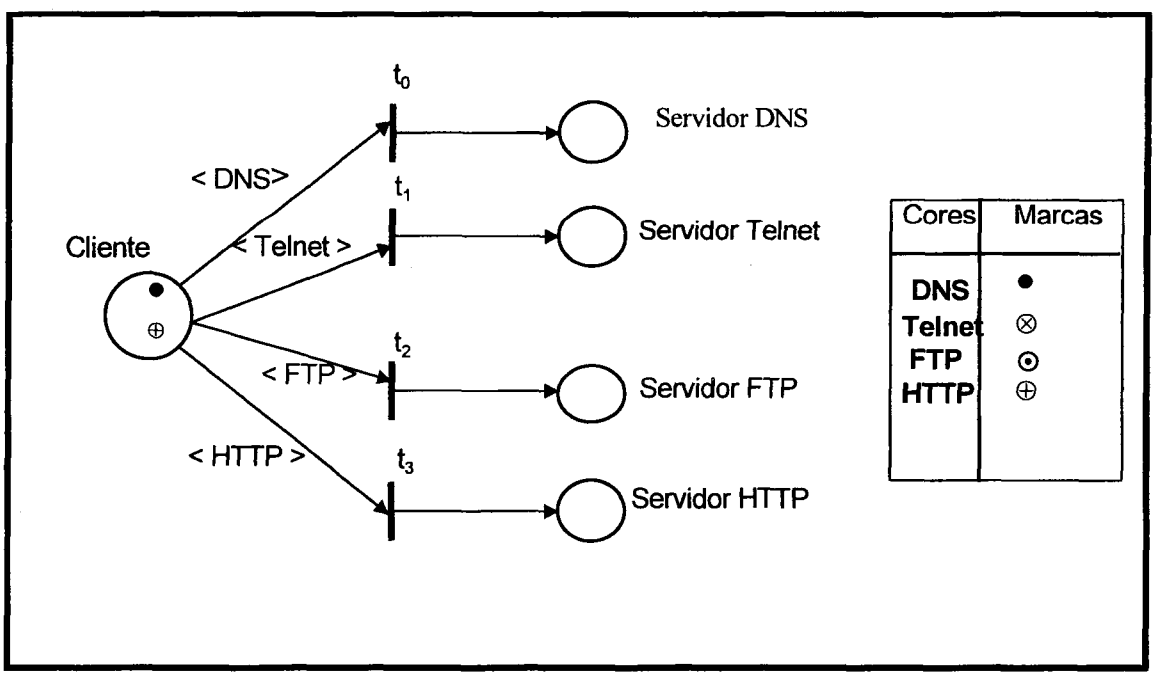

Figura 2.15. Rede de Petri Colorida.

Ainda que um tanto rudimentar, a rede colorida original provê mecanismos que possibilitam efetuar uma escolha determinística. Esse poder de escolha já significa um grande avanço em direção a uma representação mais clara de um modelo. Porém, modificações (acréscimos) posteriores vieram dar maior adequação às redes coloridas, com relação à representação das escolhas não-determinísticas. A seguir, processa-se uma discussão a respeito de algumas melhorias adicionadas às redes coloridas, tornando-as mais poderosas.

As redes de Petri coloridas são compostas por três partes (Maciel et al., 1996):

- Estrutura,
- Declarações,
- Inscrições.

A Estrutura ainda permanece a básica, porém com uma pequena modificação gráfica, mas não semântica. Aqui, os lugares, além de círculos, podem ser representados por elipses. Já as transições são representadas por retângulos. Essa representação herda a propriedade das redes coloridas originais de poder armazenar em cada lugar marcas de tipos diferentes, além de poder representar valores associados a tipos de dados mais complexos. Declarações são a especificação dos conjuntos de cores e declarações de variáveis. As inscrições variam de acordo com o componente da rede. Os lugares possuem três tipos de inscrições: nomes, conjunto de cores e expressão de iniciação (marcação inicial). As transições têm dois tipos de inscrições: nomes e expressões-guarda, e os arcos apenas um tipo de inscrição dado pela expressão. Como formas de distinguir as inscrições, nomes são escritos com letras normais, cores em itálico, expressões de iniciação sublinhadas e as expressões guarda são colocadas entre colchetes.

Nomes, quando associados a lugares, não têm significado formal, apenas facilitam a identificação. As expressões-guarda associadas às transições são expressões *booleanas* que devem ser atendidas para que seja possível o disparo das transições (Maciel et al., 1996).

Para ilustrar a aplicação de redes de Petri coloridas, será mostrada uma situação clássica de geração de *deadlock*: o jantar dos filósofos, que foi extraído na íntegra de

(Maciel et al., 1996). Essa situação consiste de três filósofos que podem estar em três estados diferentes: comendo, pensando ou com fome. Os filósofos estão à volta de uma mesa, sendo que cada um deles tem à sua frente um garfo e um prato de comida. São, no entanto, necessários dois garfos para que um filósofo possa comer, ou seja, um filósofo precisa do seu garfo e do de seu vizinho. O *deadlock* ocorrerá quando todos os filósofos pegarem o garfo da direita e aguardarem a liberação do garfo da esquerda. A figura 2.16 apresenta o jantar dos filósofos modelado em rede de Petri colorida.

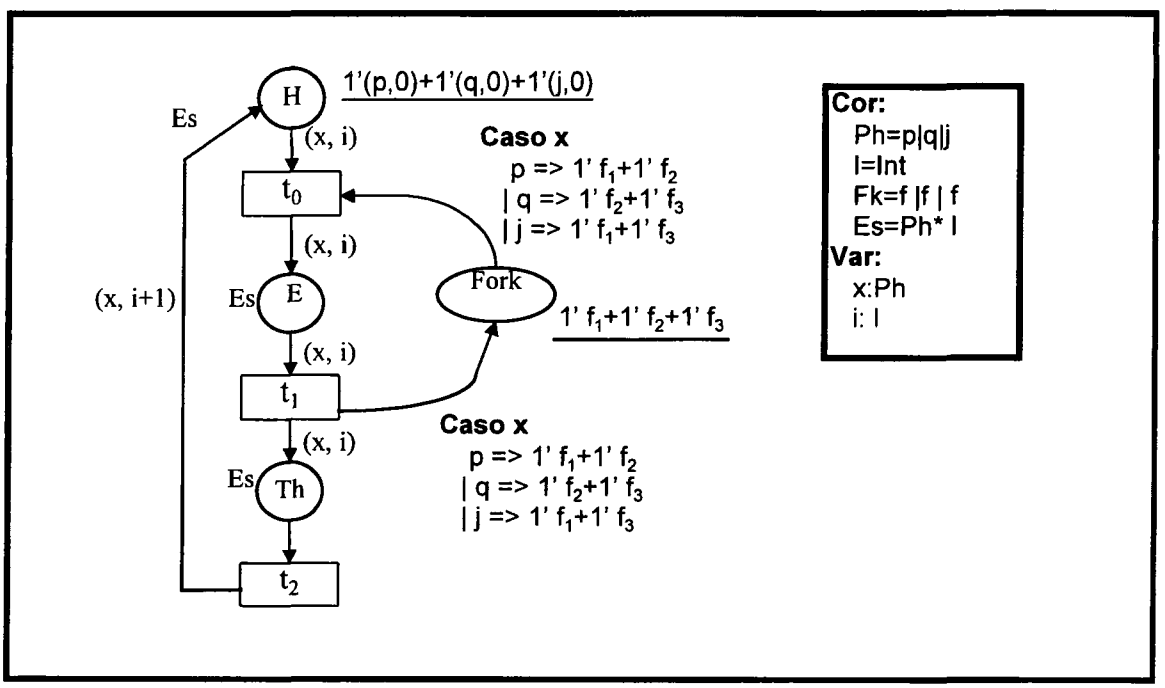

Figura 2.16. Jantar dos Filósofos Modelado em Redes Coloridas.

Na figura 2.16, os lugares representam os estados possíveis para os filósofos (H para "com fome", E para "comendo" e Th para "pensando") e os recursos do sistema (no caso, os garfos representados por Fork). A variável x indica o filósofo que irá passar para o estado "comendo" (E) e a variável i indica o número de iterações que já ocorreram. Na primeira iteração, o lugar H possui três marcas (marcação inicial) que estão sublinhadas. Dependendo da atribuição dada à variável x, uma das três expressões é avaliada no arco que liga o lugar Fork à transição t_0. Assim, se, por exemplo, x = p, então a expressão 1' f_1 + 1' f_2 é avaliada e apenas a marcação 1'(p, 0) do lugar H irá à transição t_0, significando que apenas o filósofo *p* irá passar para o estado "comendo". Desta forma, para cada atribuição de x (p, q ou j) haverá uma

situação diferente, ou seja, uma marca diferente no lugar E (estado "comendo"). A modelagem colorida, além de evitar o impasse (o que também poderia ser obtido através de uma rede não-determinística), ainda possibilita uma representação mais clara e concisa ao modelo.

Apenas para efeito de comparação, apresenta-se também a figura 2.17, ilustrando o mesmo problema da figura 18, o jantar dos filósofos. Só que na figura 2.17 (Maciel et al., 1996), a representação utilizada segue a metodologia das redes de Petri ordinárias. A comparação evidencia o poder de concisão e clareza que as redes coloridas proporcionam. Como é usada apenas para efeito ilustrativo, a rede ordinária não será explicada, mas vale ressaltar que com a extensão colorida foram necessários 4 lugares e 3 transições, enquanto que com redes de Petri ordinárias são necessários 12 lugares e 9 transições.

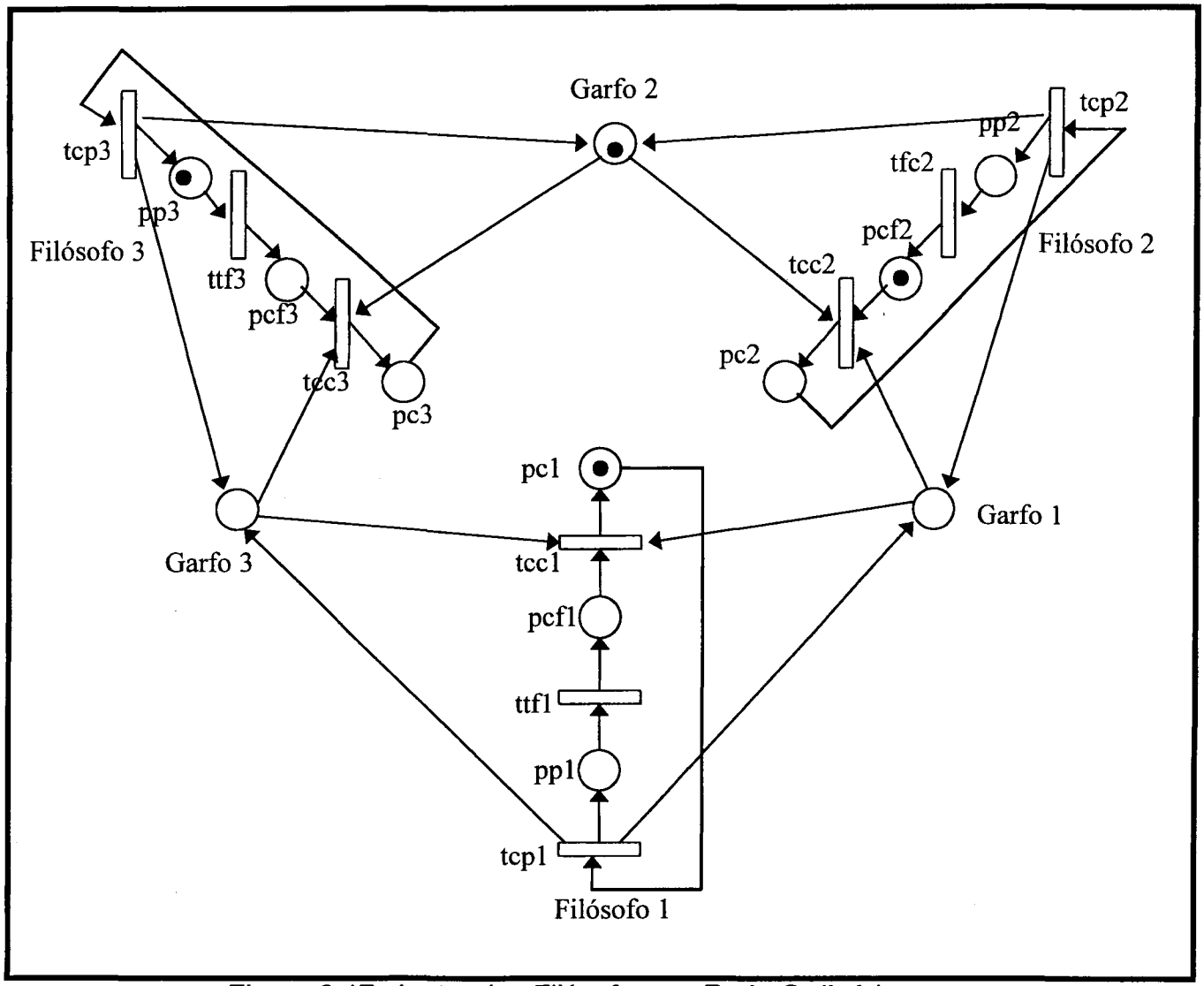

Figura 2.17. Jantar dos Filósofos em Rede Ordinária.

2.7.2. Redes Hierárquicas

Um dos problemas apresentados nas redes de Petri originais é o fato de que, à medida que o tamanho do sistema cresce, vai se tornando cada vez mais difícil manter a clareza do modelo. Esse fato se deve, em grande parte, à falta de hierarquização dos modelos originais. Por exemplo, a figura 2.18 apresenta um protocolo simples, com um transmissor e dois receptores. As representações de redes ordinárias não permitem a criação de um modelo com maior abstração, apesar de receptores e transmissor poderem ser agrupados em elementos mais gerais.

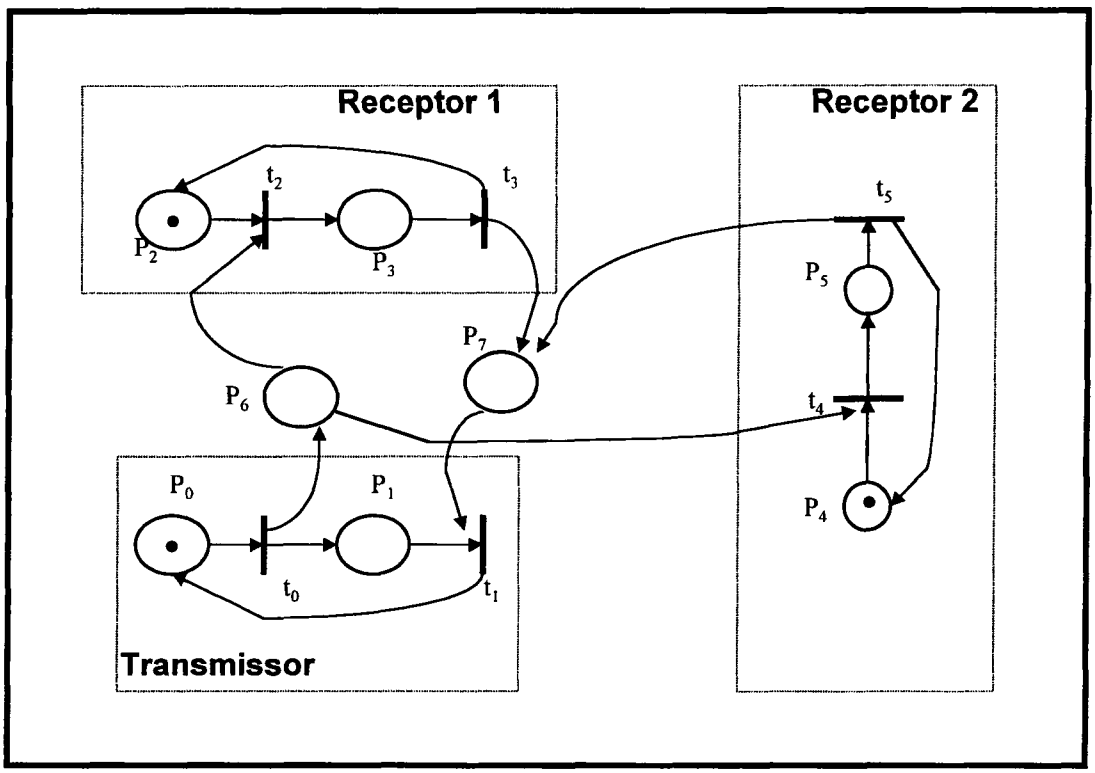

Figura 2.18. Protocolo Simples em Rede de Petri Ordinária.

Para amenizar essa limitação, foram criados mecanismos que possibilitam o agrupamento ou o refinamento de partes do modelo. Um dos problemas dessa abordagem de mais alto nível é manter a consistência com os elementos vizinhos àqueles que sofrem um agrupamento. Na abordagem hierárquica, mostrada nesta seção, lugares e transições podem ser apresentados sob uma ótica mais abstrata.

Na representação hierárquica, dois componentes são fundamentais para viabilizar uma representação em mais alto nível: a superpágina e a subpágina (Dittrich, 1995). A primeira representa um agrupamento de componentes (transições, lugares e arcos), visando à geração de um modelo mais compacto e inteligível, como se fosse uma "caixa preta". Já as subpáginas são o refinamento de uma superpágina, de forma a esclarecer alguns detalhes omitidos na representação em alto nível. A figura 2.19 apresenta uma representação em alto nível para o exemplo do protocolo de comunicação visto na figura 2.18. O exemplo mostra a representação do transmissor e dos receptores 1 e 2 em forma de superpáginas.

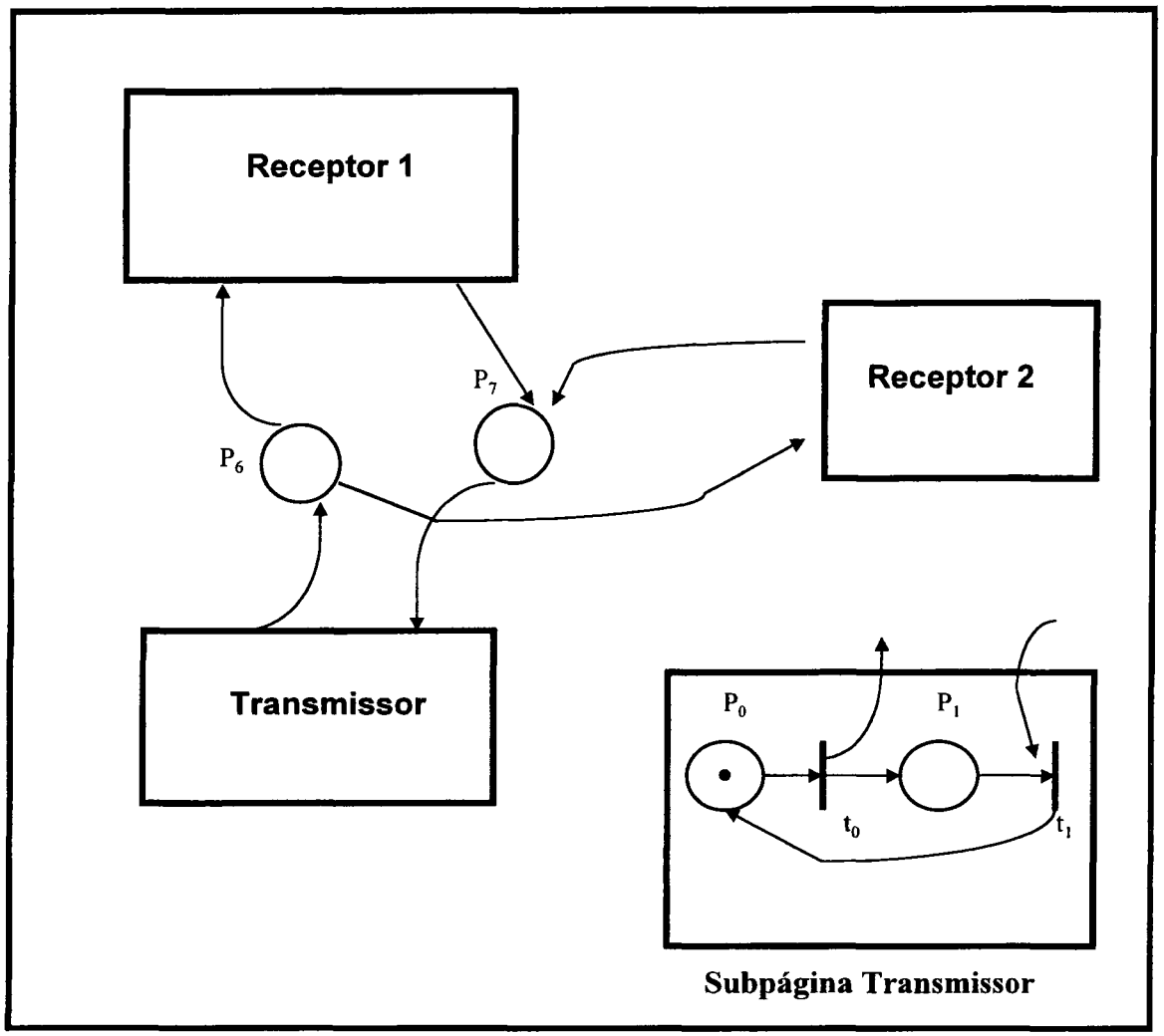

Figura 2.19. Hierarquia Utilizando Superpáginas.

2.7.3. Rede de Petri Temporizada Determinística

Para suprir a necessidade de escolha determinística (assim como nas redes coloridas), foram desenvolvidas as redes de Petri temporizadas. Essas redes possibilitam a representação do comportamento dinâmico de sistemas que possuam atividades concorrentes, assíncronas e não-determinísticas, através da adição do conceito de tempo no modelo (Coolahan & Roussopoulos, 1983). O tempo também pode ser usado de maneira probabilística, ou seja, o disparo de transições está associado a distribuições de probabilidade (seção 3.8.1). Essas redes são denominadas redes de Petri estocásticas, pois seus comportamentos podem ser descritos por processos estocásticos.

A associação do tempo a componentes da rede pode se realizar de várias maneiras. As principais são (Maciel et al., 1996):

- **O tempo associado aos lugares**. Assim, os *tokens* (após o disparo de uma transição) só estarão disponíveis para disparar uma nova transição após um determinado tempo que está associado ao lugar.

- **O tempo associado às marcas**. Nesse caso o tempo indica quando a marca estará disponível para disparar uma transição.

- **O tempo associado às transições**. O objetivo desta seção é enfocar as redes de Petri temporizadas determinísticas com tempos associados às transições. Por uma questão prática, quando se fizer referência à rede de Petri temporizada, estará subentendido que se está referindo às redes de Petri temporizadas determinísticas com tempos associados às transições.

Um exemplo de rede de Petri temporizada determinística é apresentado na figura 2.20. Nela, as transições t_1 e t_2 possuem tempos associados diferentes, o que significa que uma transição será disparada antes da outra. Supondo-se que $d_1 < d_2$, então o *token* chegará primeiro ao lugar P_1. Assim, de maneira determinística, pode-se estabelecer a ordem em que os eventos devem ocorrer.

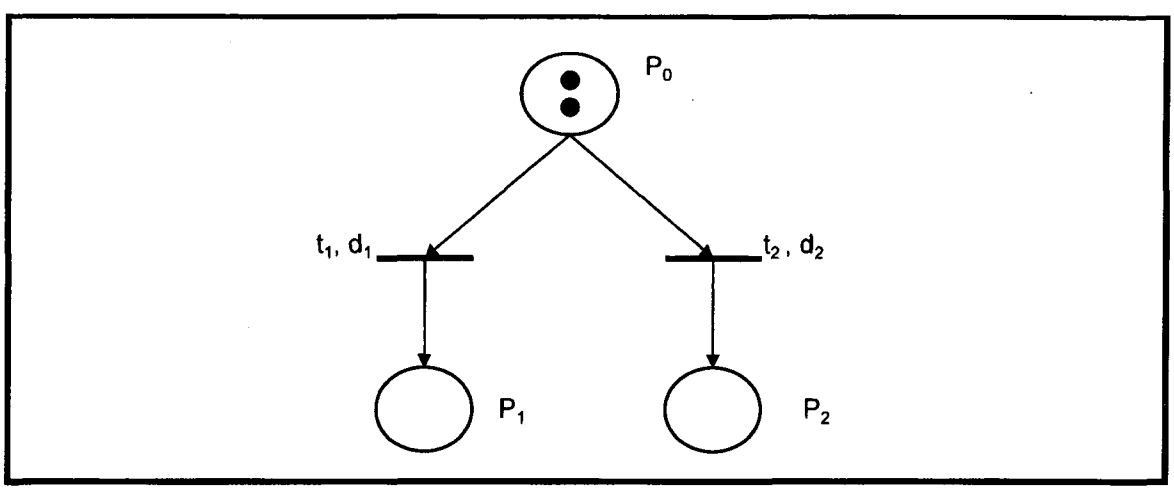

Figura 2.20. Rede de Petri Temporizada Determinística.

2. 8. Análise das Redes de Petri

Há três métodos para se analisar as redes de Petri: análise baseada na árvore de cobertura, métodos baseados na equação fundamental das redes de Petri e as técnicas de redução (Murata, 1989). Esta seção enfoca a análise baseada na equação fundamental das redes de Petri (EFRP).

2.8.1. Análise da EFRP

A equação fundamental, ou equação de estados, possibilita a análise da acessibilidade das marcações, bem como o número de vezes que cada transição deve ser disparada para que se obtenha a referida marcação.

A Equação Fundamental das Redes de Petri é:

$$M'(p) = M_0(p) + C.\, s, \forall\, p \in P$$

Onde s é o vetor característico cujos componentes s_i são naturais e representam o número de vezes que cada transição t_i foi disparada para obter-se a marcação M'(p) a partir de $M_0(p)$, e C é a matriz de incidência, dada pela diferença entre as matrizes de pós e pré-condições (C = O - I). Assim, pode-se determinar se uma marcação M_k é acessível a partir de M_0.

Retomando-se a figura 2.21, pode-se analisar a acessibilidade da marcação M' = (1,1, 1) a partir da marcação inicial M_0 = (2,0, 1).

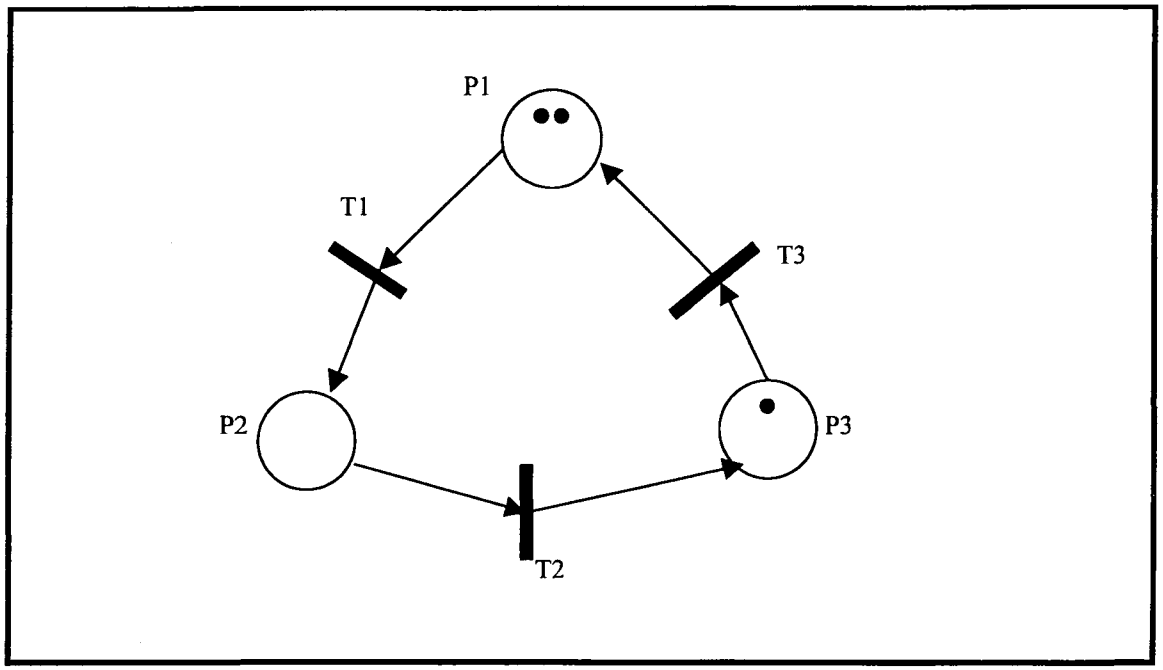

Figura 2.21. Acessibilidade de uma Marcação.

A partir da EFRP, tem-se o seguinte sistema matricial:

$$\underset{M'(p)}{\begin{bmatrix}1\\1\\1\end{bmatrix}} - \underset{M_0(p)}{\begin{bmatrix}2\\0\\1\end{bmatrix}} = \underset{C}{\begin{bmatrix}-1 & 0 & 1\\ 1 & -1 & 0\\ 1 & 1 & -1\end{bmatrix}} \times \underset{S}{\begin{bmatrix}s_1\\s_2\\s_3\end{bmatrix}}$$

A partir da solução do sistema matricial, podem-se determinar as relações entre os números de disparos de das transições. Por exemplo, tem-se que $s_1 = s_3 + 1$, isto é, o número de disparos de t_1 é uma unidade maior que o número de disparos da transição t_3, e $s_2 = s_3$, indicando que o número de disparos de t_2 deve ser igual ao número de disparos de t_3. Assim, a partir dos elementos de ***s***, podem-se obter as seguintes relações:

$$s_1 = s_3 + 1,\ s_1 = s_2 + 1,\ s_2 = s_3$$

Se for considerado que $s_1 = 1$, $s_2 = 0$ e $s_3 = 0$, tem-se como solução do problema s = (1,0,0). Significando que com o disparo da transição t_1, obtém-se a marcação M', a

partir de M_0, isto é, M' é alcançável a partir de M_0. Vale observar que a marcação M' pode ser alcançada através de outras relações de disparo de transições, como (2,1,1).

2.9 Avaliação de Desempenho com Redes de Petri

A avaliação de desempenho de um sistema computacional pode ser efetuada através de técnicas de aferição ou técnicas de modelagem. As técnicas de aferição, tais como protótipos, *benchmarks* e monitores são mais adequadas quando se dispõe do sistema para efetuar a análise. As técnicas de modelagem permitem o estudo de um sistema que está em fase de projeto ou que não existe, ou ainda nas situações em que o estudo no sistema real pode influenciar nas medidas obtidas. Um modelo é uma representação do sistema e pode ser especificado através de técnicas como por exemplo as Redes de Filas, Redes de Petri ou *Statecharts*. A solução do modelo pode ser efetuada através de resolução analítica ou de simulação (Spolon et al., 1999). As Redes de Petri constituem uma técnica poderosa devido ao fato de serem voltadas para a especificação de sistemas concorrentes. A seção 4 faz uma comparação entre Redes de Petri e *Statecharts*, ressaltando vantagens e desvantagens de cada técnica.

As Redes de Petri tradicionais e extensões apresentadas até a seção anterior permitem efetuar apenas o estudo comportamental de um sistema, verificando a sintaxe e a semântica do modelo. Para que fosse possível efetuar o estudo de desempenho de um sistema, tornou-se necessário introduzir a noção de processos estocásticos nas Redes de Petri tradicionais, criando-se novas extensões. A seção 2.9.1 apresenta os conceitos básicos necessários para a compreensão dessas extensões.

Nas seções 2.9.2 e 2.9.3 são descritas as Redes de Petri Estocásticas e as Redes de Petri Estocásticas Generalizadas e exemplos de estudos de desempenho utilizando os recursos oferecidos pelas Redes Estocásticas. É importante ressaltar que tais exemplos foram verificados e executados no software ALPHA/Sim (Alphatech Corporation, 1995), descrito na seção 2.10.

Exemplos de ferramentas que exploram as Redes de Petri como técnica de modelagem aparecem na seção 2.10. São abordadas algumas das ferramentas mais discutidas na literatura e que envolvem diferentes funcionalidades oferecidas pelas Redes de Petri e suas extensões, incluindo ferramentas que permitem apenas a análise

comportamental do modelo em estudo e também aquelas que permitem a avaliação estocástica do mesmo.

2.9.1 Considerações Iniciais

Segundo Marsan (Marsan, 1989; Marsan et al., 1996, Marsan et al. 1998) um processo estocástico consiste em um modelo matemático útil para a descrição de um fenômeno de natureza probabilística com função de um parâmetro de tempo. Existem exemplos nas mais diversas áreas, tais como variações na qualidade dos produtos de uma fábrica, jogos baseados em dados, a variação no número de mensagens em uma rede de comunicação, o tamanho da fila de um recurso em um sistema computacional.

Antes de definir-se um processo estocástico, é necessário explicar alguns conceitos importantes da Teoria de Probabilidades. Um experimento aleatório consiste em um experimento que pode ter vários resultados diferentes. O conjunto de todos os resultados possíveis constitui o espaço amostral do experimento. Um exemplo bastante simples de um experimento aleatório consiste no lançamento de um dado, cujo espaço amostral possui 6 resultados elementares. Associando-se uma medida de probabilidade a todos os possíveis resultados (os elementares e os complexos, tal como maior do que o valor 2) obtém-se um espaço de probabilidade. Uma variável aleatória consiste em uma função real definida sobre um espaço de probabilidade, e o conjunto dos valores possíveis da função forma o espaço de estado da variável aleatória. Exemplificando, pode-se definir uma variável aleatória como sendo a função iΠ onde i corresponde aos resultados elementares i, i = 1, 2, ..., 6 do experimento aleatório de lançamento de um dado (Marsan, 1989).

Define-se então processo estocástico, denotado por $\{X(t), t \in T\}$, como sendo uma família de variáveis aleatórias X(t), onde t é um parâmetro pertencente a um conjunto T (Ochi, 1990).

O parâmetro t é comumente interpretado como tempo. Um processo estocástico $\{X(t), t \in T\}$ é uma função de dois argumentos $\{x(t, w); t \in T, w \in \Omega\}$ onde Ω é o espaço amostral. Para um t fixo, x(w) é uma família de variáveis aleatórias. Para um w fixo, x(t) é a função de tempo que pode ser chamada de função amostral.

O espaço de estados do processo pode ser discreto ou contínuo. No primeiro caso denomina-se processo de espaço discreto e, no segundo, processo de espaço contínuo. Se o parâmetro t é contínuo, tem-se um processo de tempo contínuo, senão tem-se um processo estocástico de tempo discreto, referenciado como uma seqüência estocástica (Marsan et al., 1996). Neste trabalho, os objetos de estudo são os processos estocásticos de espaço e tempo discretos, mais relacionados aos problemas de avaliação de desempenho de sistemas computacionais e protocolos de sincronização de processos.

Apresentada a definição de um processo estocástico, o leitor está apto a estudar as Redes de Petri Estocásticas e as Redes de Petri Estocásticas Generalizadas.

2.9.2 Redes de Petri Estocásticas

Modelos probabilísticos de desempenho tentam representar o comportamento de sistemas determinísticos complexos através de processos estocásticos. Dessa forma, para que as Redes de Petri pudessem ser utilizadas para avaliação de desempenho foram introduzidas mudanças nas Redes de Petri Temporizadas Determinísticas.

A união das características de Redes de Petri (que são voltadas para descrever situações de concorrência e sincronismo) com um modelo estocástico permite realizar o estudo de desempenho de sistemas computacionais complexos, através das Redes de Petri Estocásticas (*Stochastic Petri Nets – SPN*) (Marsan et al., 1996).

Nessa variação das Redes de Petri, associa-se a cada transição uma variável aleatória exponencialmente distribuída, que expressa a quantidade de tempo decorrido até que a transição seja habilitada.

As *SPN* podem ser formalmente definidas como sendo (Marsan, 1989):

SPN = (P, T, A, M_0, L) onde:

P = $\{p_1, p_2, ..., p_n\}$ é o conjunto de lugares;

T = $\{t_1, t_2, ..., t_m\}$ é o conjunto de transições;

A = $(A_e \cup A_s)$ é o conjunto de arcos;

$A_e \subset (P \times T)$ é o conjunto de arcos de entrada;

$A_s \subset (T \times P)$ é o conjunto de arcos de saída;

$M_0 = \{m_{01}, m_{02}, ..., m_{0n}\}$ onde m_{0i} representa a número de *tokens* no lugar p_i, na marcação inicial M_0;

$L = \{l_1, l_2, ..., l_m\}$ é o conjunto de taxas de disparo associadas às transições.

As Redes de Petri Estocásticas permitem a obtenção de estatísticas como por exemplo o tempo médio que um *token* gasta para percorrer a rede, o número médio de vezes que uma determinada transição dispara e o número médio de *tokens* em um lugar.

Como um exemplo de uma Rede de Petri Estocástica, considera-se um estudo realizado com o protocolo de sincronização *Time Warp*, utilizado na sincronização entre os processos de uma simulação distribuída (Jefferson, 1985). Para tornar claro o conceito de simulação distribuída (o termo paralela também é utilizado para descrever tal tipo de simulação) é necessário entender o funcionamento da simulação seqüencial tradicional. Nesse caso, o programa de simulação contém um relógio que armazena o tempo simulado e uma estrutura de dados denominada lista de eventos futuros. Essa lista mantém todos os eventos da simulação organizados em ordem crescente do tempo de ocorrência e faz com que sejam executados na ordem correta. Na simulação distribuída, não existem os conceitos de relógio único e lista de eventos única, mas sim relógios locais e listas de eventos futuros distribuídas entre os vários processos. Por isso torna-se necessária a sincronização entre os processos e diversos protocolos têm sido desenvolvidos para esse fim. Basicamente, podem-se agrupá-los em protocolos otimistas e protocolos conservativos. No caso conservativo os eventos somente são executados quando se tem a certeza de que não irão provocar erros (ou seja, executar um evento cujo tempo de ocorrência é menor do que o relógio do processo, denominado erro de causa e efeito). Já os protocolos otimistas permitem que a simulação avance e quando um erro de causa e efeito ocorre, fazem uso do mecanismo de *rollback* para retornar a simulação a um estado seguro, isto é, desfazer toda a computação executada de forma errônea. O mecanismo de *rollback* consiste basicamente em restaurar o estado do processo para um estado seguro no tempo de simulação e enviar mensagens (denominadas antimensagens) para desfazer os efeitos das mensagens enviadas anteriormente.

A figura 2.22 apresenta um exemplo de uma Rede de Petri Estocástica, consistindo de parte de um modelo mais complexo que representa e analisa o comportamento do protocolo *Time Warp*, e que foi desenvolvido no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP. As estruturas de dados básicas envolvidas são a Lista de Entrada de Mensagens que armazena todos os eventos executados e a serem executados, em ordem crescente do tempo de ocorrência, e o *Buffer* que armazena as mensagens que foram recebidas de outros processos e precisam ser tratadas.

O exemplo mostra, de forma bastante simplificada, o comportamento de um processo que, a partir de um estado Pronto (lugar com mesmo nome), pode executar os eventos de sua Lista de Entrada de Mensagens (que pode envolver o envio de mensagens a outros processos), ou então tratar os eventos que chegaram e estão previamente armazenados em um *Buffer*. Nesse modelo simplificado o processo pode executar duas ações com relação às mensagens que chegam. Em um caso, armazena a mensagem na Lista de Entrada de Mensagens, para ser posteriormente executada (ou seja, assume que a mensagem carrega um evento com tempo de ocorrência maior ou igual ao valor do relógio do processo). Outra ação corresponde às mensagens que provocam *rollback*, também chamadas de antimensagens, cujos tempos de ocorrência são maiores do que o relógio local. Nesse caso, o processo deve desfazer o trabalho errôneamente efetuado. Para facilitar o entendimento o *rollback* é apresentado apenas como um lugar nesse exemplo, e mostrado com maior nível de detalhamento no exemplo da figura 2.24.

As ações efetuadas no modelo correspondem às transições temporizadas, que descrevem:

- t_0: o processo retira a primeira mensagem da Lista de Entrada de Mensagem e executa o evento correspondente, desde que o processo não esteja tratando uma mensagem que chegou e está armazenada no *Buffer* (essas duas atividades são mutuamente exclusivas);

- t_1: a execução do evento gerou um novo evento para execução local;

- t_2: a execução do evento correspondente à mensagem da Lista de Entrada de Mensagens não gerou novas mensagens de evento;

- t_3: a execução do evento gerou um novo evento, a mensagem correspondente deve ser enviada para execução em outro processo da simulação;

- t_4: atraso para permitir que o processo volte ao estado Pronto depois que as informações para envio da mensagem já estejam preparadas;

- t_5: atraso inserido no modelo para mostrar que o processo volta ao estado Pronto apenas depois que a mensagem foi inserida na Lista de Entrada de Mensagens;

- t_6: o processo retira uma mensagem do *Buffer*, desde que não esteja executando localmente uma mensagem;

- t_7: uma mensagem chegou e deve ser armazenada na Lista de Entrada de Mensagens, para posterior execução;

- t_8: uma antimensagem chegou e o mecanismo de *rollback* é ativado;

- t_9: atraso inserido no modelo para permitir que o processo volte ao estado Pronto apenas depois que a mensagem foi inserida na Lista de Entrada de Mensagens;

- t_{10}: mecanismo de *rollback*;

- t_{11}: após o *rollback*, o processo volta ao estado Pronto.

Os lugares da Rede de Petri representam estados do processo (Pronto), estruturas de armazenamento (Lista de Entrada de Mensagens, *Buffer*), início de

alguma atividade (Execução_Local, Recebe_Mensagen, Mensagem, Antimensagem) ou simplesmente simplificações de ações mais complexas (*Rollback* e Transmissão). O lugar Pronto contém um *token*, indicando que o processo ou está executando uma mensagem ou está tratando uma mensagem que foi recebida e está armazenada no *Buffer*.

O modelo foi parametrizado com uma carga sintética, apenas para mostrar aos leitores como a avaliação de desempenho pode ser efetuada utilizando a técnica Estocástica e a ferramenta ALPHA/Sim (Alphatech Corporation, 1995). Esses parâmetros não refletem uma plataforma de *hardware* real nem uma aplicação sendo resolvida pelo *Time Warp*. A Lista de Entrada de Mensagens contém inicialmente 1000 *tokens*. O *Buffer* também teve sua parametrização inicial definida como 1000 *tokens*. Os tempos associados às transições são:

- t_0: exponencial com média 1.0 milisegundo;
- t_1: exponencial com média 2.0 milisegundos;
- t_2: exponencial com média 1.0 milisegundo;
- t_3: exponencial com média 2.0 milisegundo;
- t_4: exponencial com média 1.0 milisegundo;
- t_5: exponencial com média 1.0 milisegundo;

1. t_6: exponencial com média 1.0 milisegundo;

- t_7: exponencial com média 2.0 milisegundos;
- t_8: exponencial com média 1.0 milisegundo;
- t_9: exponencial com média 1.0 milisegundo;

- t_{10}: exponencial com média 1.0 milisegundo;

- t_{11}: exponencial com média 1.0 milisegundo;

O lugar Execução_Local faz uma escolha probabilística da próxima ação a ser efetuada, com 30 % de chance de executar t_1, 30% de executar t_2 e 40% de chance de executar t_3. Considerou-se, nessa parametrização, que 20% das mensagens que chegam ao processo provocam *rollback*.

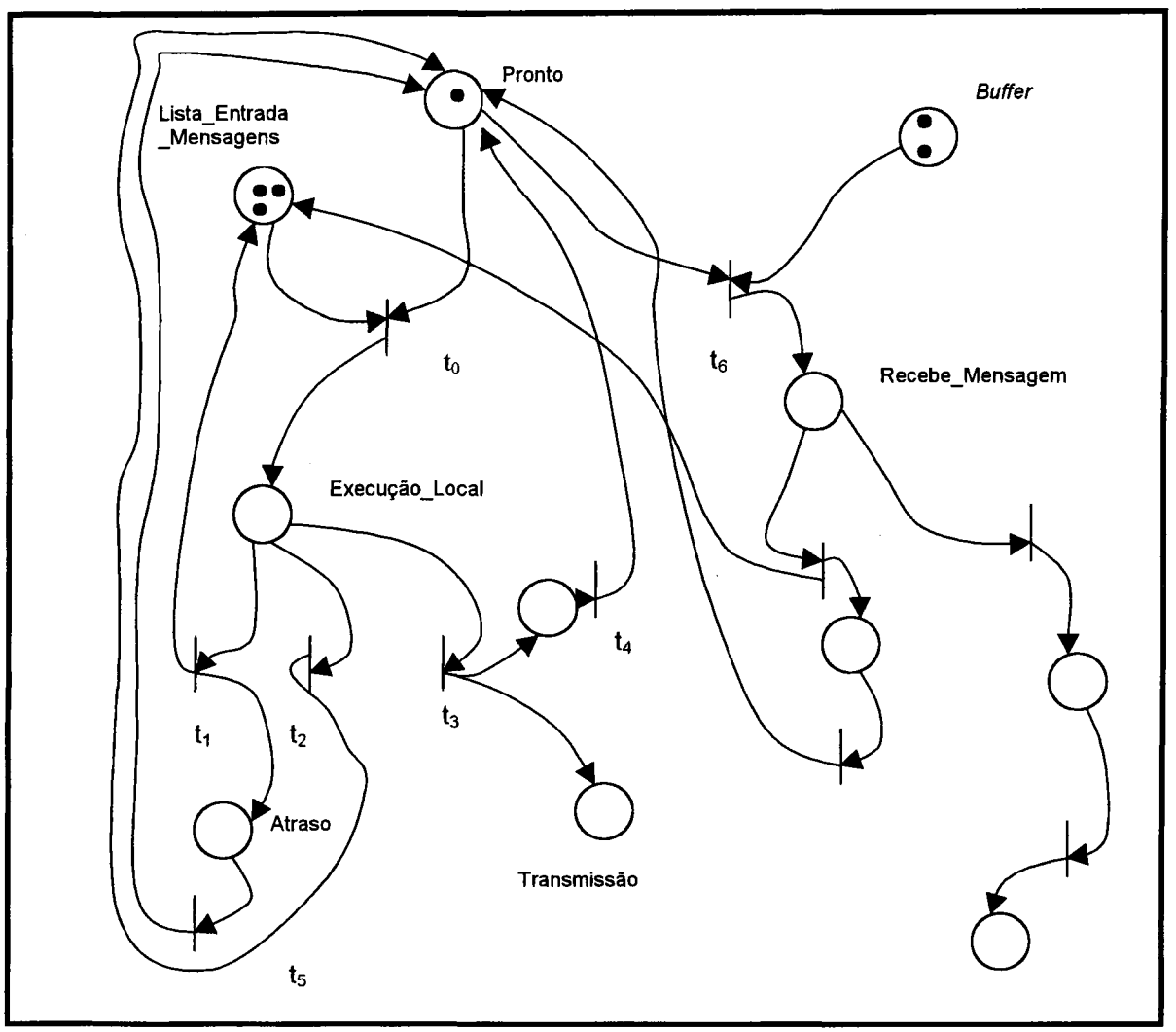

Figura 2.22. Exemplo de uma Rede de Petri Estocástica.

Como exemplo de resultado de desempenho que pode ser obtido a partir do ALPHA/Sim, considerando-se o modelo da figura 2.2 e os parâmetros apresentados, pode-se verificar o tempo médio gasto na atividade de *rollback*, que foi de 1542,61632 milisegundos (esse tempo é uma métrica importante no estudo de desempenho de mecanismos de simulação baseados no *Time Warp*). Pode-se utilizar essa informação para comparar o tempo gasto em *rollback* utilizando-se diferentes mecanismos de sincronização, por exemplo.

2.9.3 Redes de Petri Estocásticas Generalizadas

A principal motivação para o desenvolvimento das Redes de Petri Estocásticas Generalizadas (*Generalized Stochastic Petri Nets - GSPN*) consiste no fato de que em determinadas situações não é desejável associar um tempo aleatório a todas as transições, mas sim apenas àquelas que estão relacionadas com eventos que exerçam maior impacto no desempenho do sistema em estudo (Marsan et al., 1996). Como exemplo pode-se considerar um estudo do protocolo de sincronização para simulação distribuída, o *Time Warp* (Jefferson, 1985). O tempo necessário para que um processo faça a atualização do valor do relógio local pode ser desprezível, quando comparado ao tempo necessário para manipular as estruturas de fila que armazenam os estados do processo e as mensagens de evento recebidas ou enviadas. Ou então, deseja-se inserir no modelo transições descrevendo atividades lógicas do sistema em estudo, mas que não possuem um tempo associado.

Dessa forma, as *GSPN* possuem transições imediatas, que disparam em tempo zero quando estão habilitadas, e transições temporizadas, que disparam após um tempo aleatório exponencialmente distribuído quando estão habilitadas. As transições imediatas têm prioridade sobre as transições temporizadas. Para facilitar a distinção, convenciona-se utilizar um traço para representar graficamente uma transição imediata e um retângulo para representar uma transição temporizada (figura 2.23).

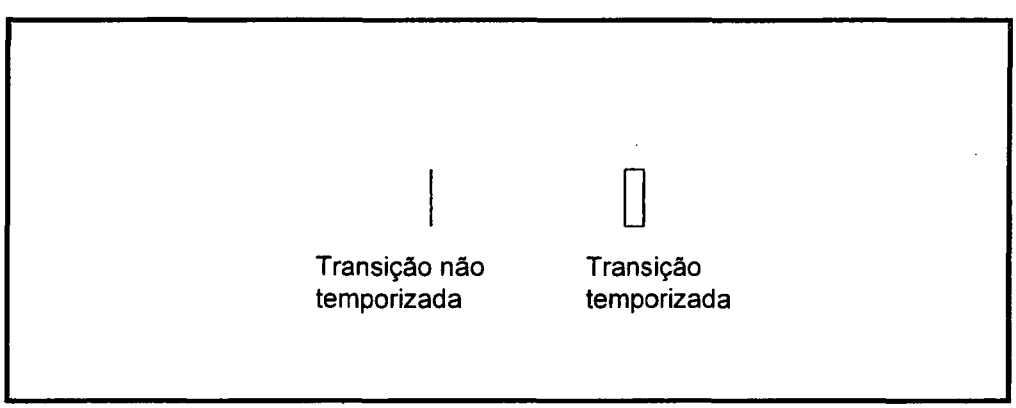

Figura 2.23. Representação Gráfica de Transições.

Formalmente, uma Rede de Petri Estocástica Generalizada pode ser representada por:

GSPN = (P, T, A, M_0, L) onde:

P = $\{p_1, p_2, ..., p_n\}$ é o conjunto de lugares;
T = $\{t_1, t_2, ..., t_m\}$ é o conjunto de transições;
A = $(A_e \cup A_s)$ é o conjunto de arcos;
$A_e \subset (P \times T)$ é o conjunto de arcos de entrada;
$A_s \subset (T \times P)$ é o conjunto de arcos de saída;
$M_0 = \{m_{01}, m_{02}, ..., m_{0n}\}$ onde m_{0i} representa a número de *tokens* no lugar p_i, na marcação inicial M_0;
$L = \{l_1, l_2, ..., l_{m'}\}$ é o conjunto de taxas de disparo associadas as transições e m' é o número de transições temporizadas.

A figura 2.24 apresenta um exemplo de um estudo simplificado modelado através de uma Rede de Petri Estocástica Generalizada, abordando o problema de se efetuar *rollback* no protocolo *Time Warp*. Os eventos da simulação são distribuídos entre os diversos processos e um *rollback* ocorre quando algum deles verifica que executou eventos fora de ordem cronológica (erro de causa e efeito). Ao efetuar *rollback*, o processo deve retroceder sua execução no tempo da simulação, restaurando seu estado para uma situação segura. As etapas envolvidas em uma situação de *rollback* consistem nas transições:

- t_0: Atualizar o valor do relógio local;
- t_1: Armazenar a mensagem que chegou na Fila de Entrada de Mensagens;
- t_2: Essa atualização da Fila de Entrada de Mensagens precisa estar encerrada antes da restauração do estado;
- t_3: Restaurar o estado do processo;

- t_4: Verificar as antimensagens que deverão ser enviadas isto é, todas as antimensagens que estão na Fila de Saída de Mensagens, com marca de tempo maior ou igual ao valor da marca de tempo da mensagem que provocou o *rollback*;

- t_5 e t_6: Enviar as antimensagens;

- t_7 e t_8: Após esses passos, o processo pode prosseguir com sua execução normal.

Os lugares representam depósitos (Lista_Entrada_de_Mensagens, Lista_Saída_Mensagens), estado do processo (Execução_Normal), simplificações (Restaurar Estado, Transmissão) ou início de atividades (Atualizar_Relógio, Armazenar_Mensagem, Atraso_Atualização, Verificar_Antimensagens, Enviar_Antimensagens, Laço_Enviar_Antimensagens.

O exemplo da figura 2.24 foi executado no *ALPHA/Sim* com parâmetros de entrada definidos de forma hipotética:

- t_0: transição imediata;

- t_1: exponencial com média 1 milisegundo;

- t_2: exponencial com média 1 milisegundo;

- t_3: exponencial com média 1 milisegundo;

- t_4: exponencial com média 1 milisegundo;

- t_5: exponencial com média 1 milisegundo;

- t_6: transição imediata;
- t_7: exponencial com média 2.0 milisegundos;
- t_8: exponencial com média 1.0 milisegundo;

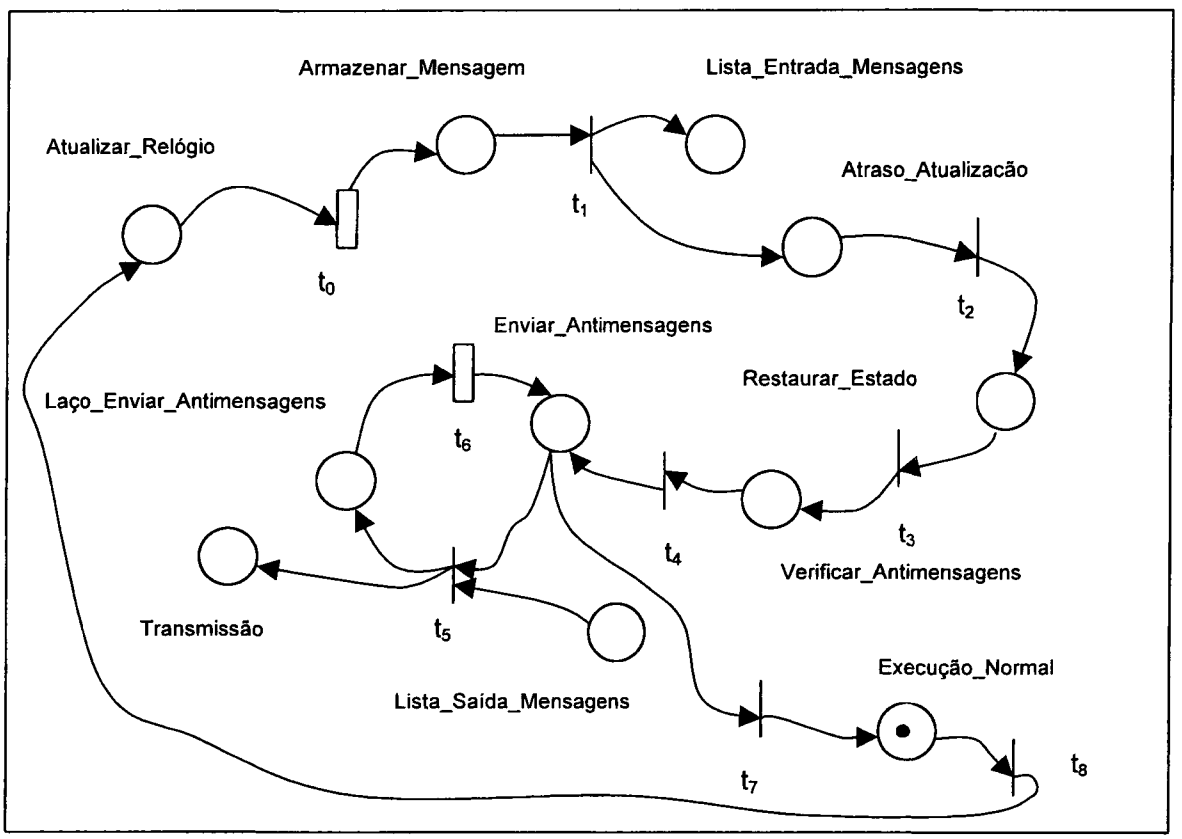

Figura 2.24. Exemplo de Rede de Petri Estocástica Generalizada.

Como exemplo de métrica que pode ser obtida a partir da simulação do modelo, tem-se: o tempo médio gasto para restaurar estados, no valor de 709,89624 milisegundos.

2.10. Ferramentas Disponíveis

O objetivo desta seção é apresentar algumas ferramentas para Redes de Petri que estão disponíveis gratuitamente ou comercialmente. O endereço http://www.daimi.au.dk/PetriNets/tools/complete_db.html é uma boa referência para a

busca de novas ferramentas, sendo constantemente atualizado. É importante ressaltar que o uso das Redes de Petri voltadas para avaliação de desempenho de sistemas complexos depende, em muito, da disponibilidade de ferramentas tanto para a construção do modelo (editores gráficos) quanto para a sua resolução (a complexidade do modelo pode aumentar bastante a complexidade da solução) (Marsan, 1989).

Verifica-se que as ferramentas abordam diferentes características das Redes de Petri e de suas extensões, estocásticas ou não. O leitor deve identificar aquela que mais se adapta às necessidades do problema em estudo. Ressalta-se que esta seção não apresenta todas as ferramentas disponíveis, apenas procura reunir exemplos que utilizem a maior variedade possível dos recursos oferecidos pelas Redes de Petri e que possam ser facilmente encontrados através de suas *homepages*.

ALPHA/Sim é uma ferramenta de simulação de propósito geral e de domínio privado, que faz uso de propriedades das Redes de Petri Estocásticas Generalizadas, das Redes de Petri Coloridas e das Redes de Petri Hierárquicas, como método para efetuar a modelagem. Através de um editor gráfico possibilita ao usuário desenhar e fornecer todos os parâmetros necessários ao modelo de simulação (Moore & Hammer, 1999). Após a definição e parametrização do modelo, o usuário pode executar a simulação sem a necessidade de escrever o código correspondente. *ALPHA/Sim* também coleta, automaticamente, as estatísticas da simulação, como por exemplo as relacionadas com filas e atrasos, e faz verificação das expressões e existência de laços infinitos (Alphatech Corporation, 1995).

Algumas características interessantes da ferramenta são a possibilidade de se utilizar hierarquia, com blocos que permitem encapsular submodelos, e a construção de *tokens* com múltiplos atributos. Como desvantagem, a ferramenta não dispõe de representações gráficas distintas para transições imediatas e transições temporizadas.

ALPHA/Sim está disponível atualmente para plataformas *IBM-PC* Compatíveis executando *Windows NT* e estações de trabalho *Sun* executando *SunOS* ou *Solaris*. A versão para *Windows NT* é a que está em uso no Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC/USP.

Outra ferramenta com funcionalidades semelhantes ao *ALPHA/Sim* é *Artifex*, que fornece ao usuário um ambiente gráfico para desenvolvimento e execução de um modelo de simulação baseado em Redes de Petri Hierárquicas e Redes de Petri Estocásticas. A execução da simulação pode ser animada e tem todo o suporte de um

editor gráfico para verificação de erros e consistência. *Artifex* está disponível comercialmente para estações de trabalho *SUN Sparc* executando *Solaris*, *Alpha Digital* com *Unix*, *IBM Risc* 6000 com *AIX* e microcomputadores IBM-PC compatíveis executando *Windows NT* ou *Linux RedHat* (Artir Software Corporation, 1999).

Diferente dos anteriores, *CodeSign* é um ambiente de modelagem com interface gráfica que utiliza as Redes de Petri Temporizadas Determinísticas e o conceito de hierarquia. Está disponível para plataformas *IBM-PC* compatíveis executando *MS Windows* e é uma ferramenta comercial, embora esteja disponível gratuitamente para instituições universitárias.

Design/CPN é uma ferramenta de domínio que público utiliza as Redes de Petri Coloridas e as Redes de Petri Hierárquicas como técnica de modelagem. Possui um editor gráfico, um simulador e uma ferramenta para construção e análise de árvores de alcançabilidade.

Outra ferramenta que vem sendo utilizada no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP é *DNANet*, ferramenta de domínio público baseada nos conceitos de Redes de Petri Coloridas, que pode ser utilizada para o projeto, análise e simulação de sistemas concorrentes e distribuídos. A extensão *DaNAMiCS (Data Network Architecture - Modelling Concurrent Systems)* explora as característcas das Redes de Petri Estocásticas Generalizadas, aumentado a funcionalidade do *DNANet.* A simulação retorna métricas como por exemplo número médio de *tokens* residentes em um lugar e o *throughput* de uma transição.

3. Statecharts

Statecharts, assim como redes de Petri, é uma técnica de representação de sistemas através da visão de seus estados, e a modificação (mudança) deles, diante da ocorrência de uma determinada ação. A partir desta seção, são apresentados conceitos, notações e exemplos de modelos utilizando a referida técnica.

3.1. Definição e Notação Básica

A definição básica de *statecharts* é alicerçada em conjuntos de estados, transições, eventos primitivos, condições primitivas e variáveis, a partir dos quais o modelador pode especificar os valores das variáveis e a situação do sistema em um determinado instante. A idéia central de um *statechart* é suprir a deficiência dos diagramas de estado em representar sistemas que possuam peculiaridades como paralelismo e interatividade. Esse tipo de sistema requer uma estrutura de representação hierárquica (com agrupamento e refinamento de estados) e de concorrência, de maneira que seja facilmente visível o movimento através dos estados do sistema no decorrer do tempo.

A técnica *statecharts* é uma extensão às máquinas de estado finito, que possibilita a representação de hierarquia, concorrência e comunicação entre os diversos estados de um determinado sistema (Harel, 1987). Os *statecharts* têm, preferencialmente, a finalidade de especificar sistemas reativos, ou seja, sistemas que devem reagir a estímulos externos e internos, normalmente sob condições críticas em relação ao tempo, o que não impede (de maneira alguma) que especificações de outros tipos de sistemas (mais elementares ou mais complexos) sejam realizadas.

Esquematicamente, tem-se:

Statecharts = Diagrama de Estados + Hierarquia + Concorrência + Mecanismos de Comunicação (Harel, 1987).

Os estados de um *statechart* (que representam os valores das variáveis do sistema em um determinado instante) podem ser classificados em dois grupos: básicos e não-básicos. Os *estados básicos* são aqueles que não possuem subestados. Já os

não-básicos são decompostos em subestados. Essa decomposição pode ser de dois tipos: OR ou AND. Se a decomposição é do tipo OR, então o sistema sempre estará em um único subestado em um certo instante. Na verdade, um XOR entre os subestados (disjuntos), pois se pode estar em apenas um estado, mas nunca em mais de um ao mesmo tempo. Já na decomposição é do tipo AND, o estado poderá estar em mais de um estado (ou subestado), simultaneamente. A figura 3.1 mostra os elementos básicos de estados na notação *statechart*.

A figura 3.1 apresenta as formas de representação de estados em *statecharts*. À esquerda, um estado X do tipo OR e seus subestados (X_1, X_2, X_3), indicando que o sistema não estará em mais de um subestado ao mesmo tempo (subestados disjuntos). À direita, um estado Y do tipo AND e seus subestados (Y_1, Y_2, Y_3, cada um deles possuindo os seus respectivos subestados Y_4, Y_5, Y_6), indicando que o sistema pode estar em mais de um subestado simultaneamente. A concorrência de estados (ou de subestados) é representada através de linhas tracejadas. Além da concorrência, a figura em questão ainda introduz a idéia de hierarquia entre os estados, ou seja, os subestados estão contidos em um estado maior denominado estado-pai (no exemplo, X e Y). Reciprocamente, um estado-pai contém estados filhos (subestados) dentro de si.

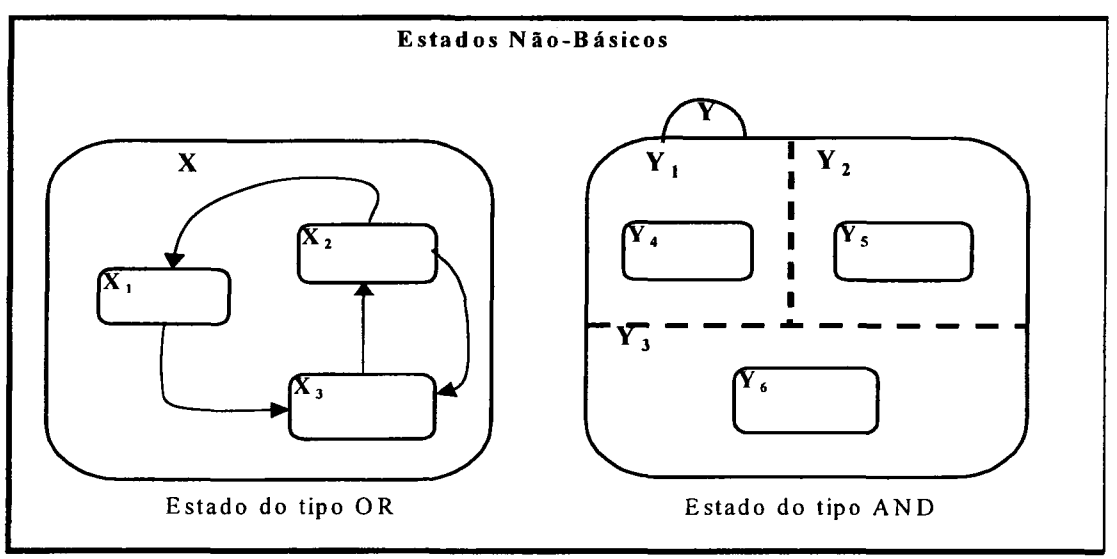

Figura 3.1. Representação de Estados em *Statecharts*.

Na verdade, a notação usada na figura 3.1 é uma simplificação da representação utilizada na figura 3.2. Na primeira, o estado Y aparece envolvido por um semicírculo,

representando que o mesmo é um superestado que contém os demais estados paralelos. Na segunda, é utilizada a representação original, com o estado Y efetivamente desenhado, contendo os demais.

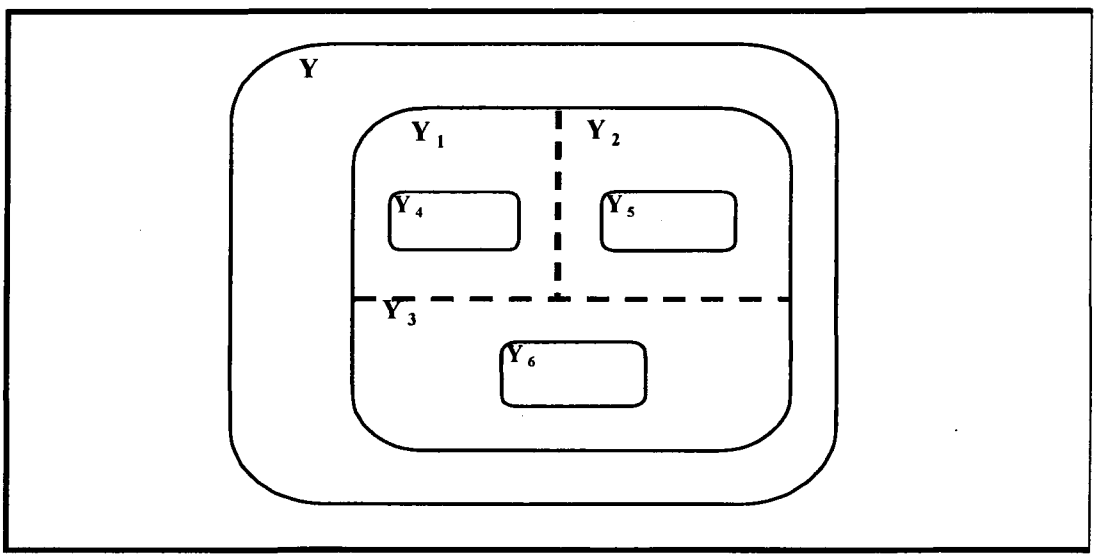

Figura 3.2 Representação Original de Estados Superestados.

Eventos (letras minúsculas) disparam transições, as quais são representadas por setas que unem estados. Os eventos podem ocorrer obedecendo a certas condições. Essas condições são representadas por letras minúsculas entre parênteses (Boaventura, 1992). Além disso, um superestado pode ter um estado inicial *default*, ou seja, toda vez que o sistema entrar nesse superestado, sempre será através do estado *default*. A figura 3.3 ilustra esses conceitos.

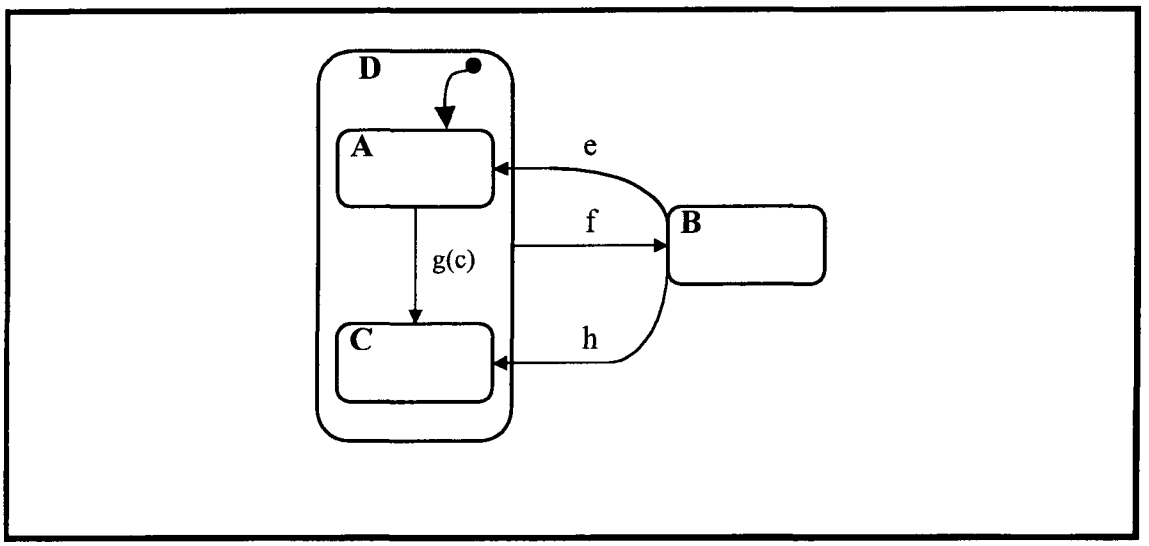

Figura 3.3. Estados, Eventos e Transições.

Na figura, A, B, C e D são estados. O sistema passa de um estado para o outro quando a ocorrência de um evento (e, f, g(c), h) dispara uma transição (setas). O evento g(c) está controlado por uma condição (no caso, condição c). O estado *default* de D (superestado) é o estado A, representado pela seta característica do diagrama. Esse fato significa que o estado D sempre começa pelo seu subestado A. Os estados *default* (figura 3.4 (b)), na verdade, substituem uma notação menos sofisticada, como a apresentada na figura 3.4 (a), sendo que esta última também pode ser usada, isto é, (a) e (b) são semanticamente equivalentes.

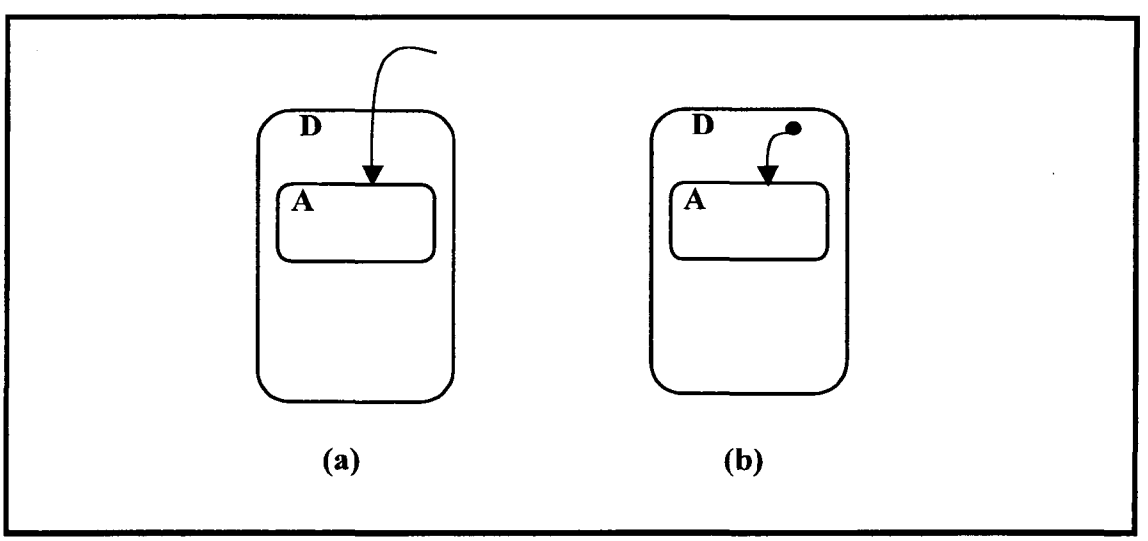

Figura 3.4 (a) Escolha sem Utilização de *Default* (b) Escolha através de Estado *Default*.

Um outro aspecto importante a ser abordado na introdução aos Statecharts é a possibilidade de representação de uma saída de um superestado, não se indicando por qual dos seus subestados se está saindo. A figura 3.5 (a) apresenta uma saída genérica de D para o estado B (provocado pelo evento β), independente se estimulada por A ou C. Já a figura 3.5 (b) apresenta uma saída explícita pelos estados A e C (eventos α e ρ) para o estado B.

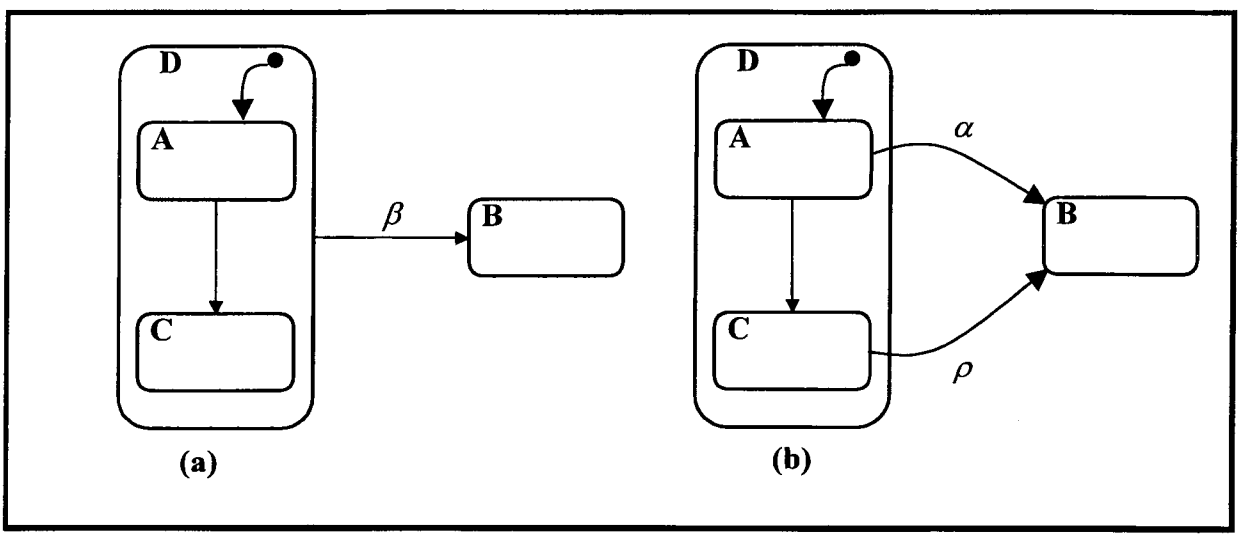

Figura 3.5. (a) Saída Genérica de D (b) Saída Direta de Subestados A e C para B.

Uma situação interessante para determinados problemas é poder guardar uma visita passada recente a um determinado estado. Os Statecharts possuem a propriedade denominada de Entrada por História, cuja função é armazenar a última visita a um determinado estado (ou conjunto de estado).

3.2. Entrada por História (H)

A figura 3.6 apresenta a representação de entrada por história.

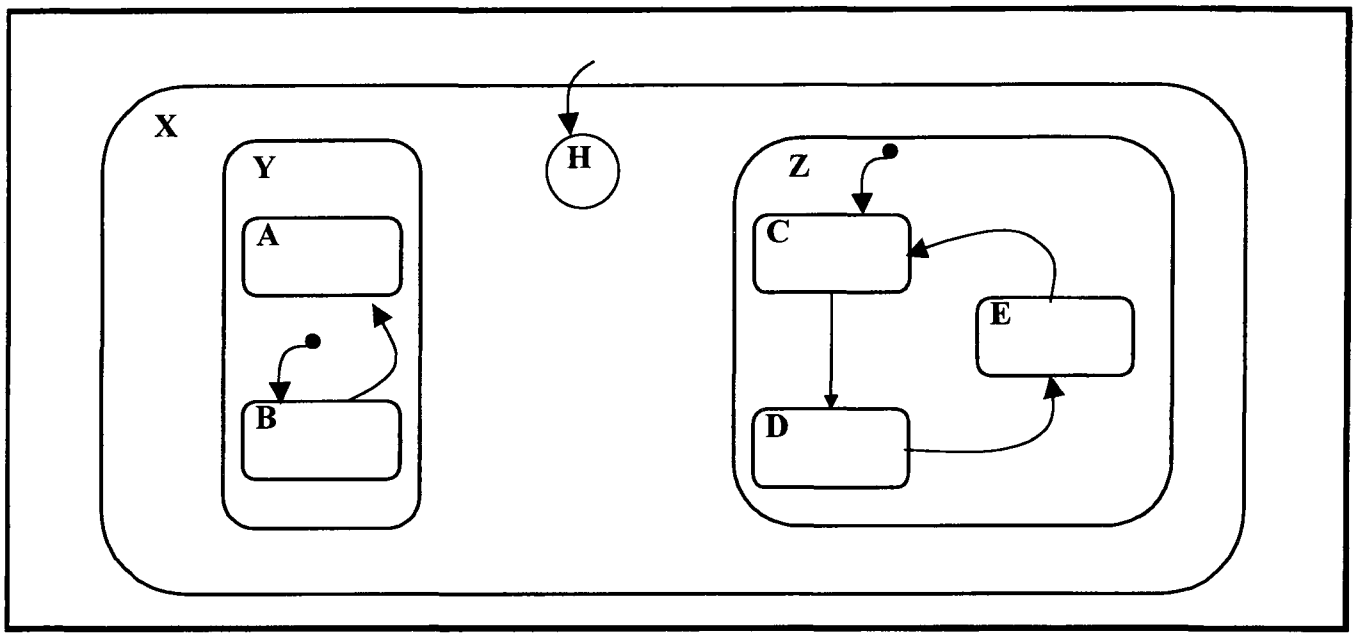

Figura 3.6. Entrada por História H.

A semântica do mecanismo de história da figura anterior estabelece que o sistema deverá escolher entre Y e Z, dependendo da última visita, isto é, se a última visita foi em A ou B, a nova entrada será em Y (pelo estado *default* B). Se, por outro lado, a última visita foi em C, D e E, então a nova entrada será realizada através de Z (pelo seu estado *default* C). Nesse tipo de construção, são beneficiados apenas os superestados, pois, a partir de um de seus estados componentes, a nova visita será feita em um superestado e não em um de seus componentes. Uma alternativa a esse problema é a notação do círculo com o H, acrescida de um asterisco. A figura 3.7 apresenta essa representação alternativa.

O acréscimo feito pela semântica da entrada por história H* é a possibilidade de se entrar pelo último estado visitado, independente se ele é um superestado ou um subestado. Na figura 3.7, supondo-se que o último estado visitado tenha sido o estado E. Então, utilizando-se o mecanismo H*, na próxima visita a entrada seria realizada pelo estado E, e não pelo seu superestado Z. É importante ressaltar que, no caso da figura 3.7, não há estados *default*, mas se houvesse, eles deveriam ser respeitados, isto é, eles teriam precedência sobre a entrada por história H*.

A grande contribuição do mecanismo de história H* é o fato de permitir que se caminhe através de diferentes níveis de complexidade do Statechart. Assim, níveis

diferentes de complexidade (superestados e seus subestados) podem ser tratados de maneira indiferente, como se todos pertencessem a um mesmo nível.

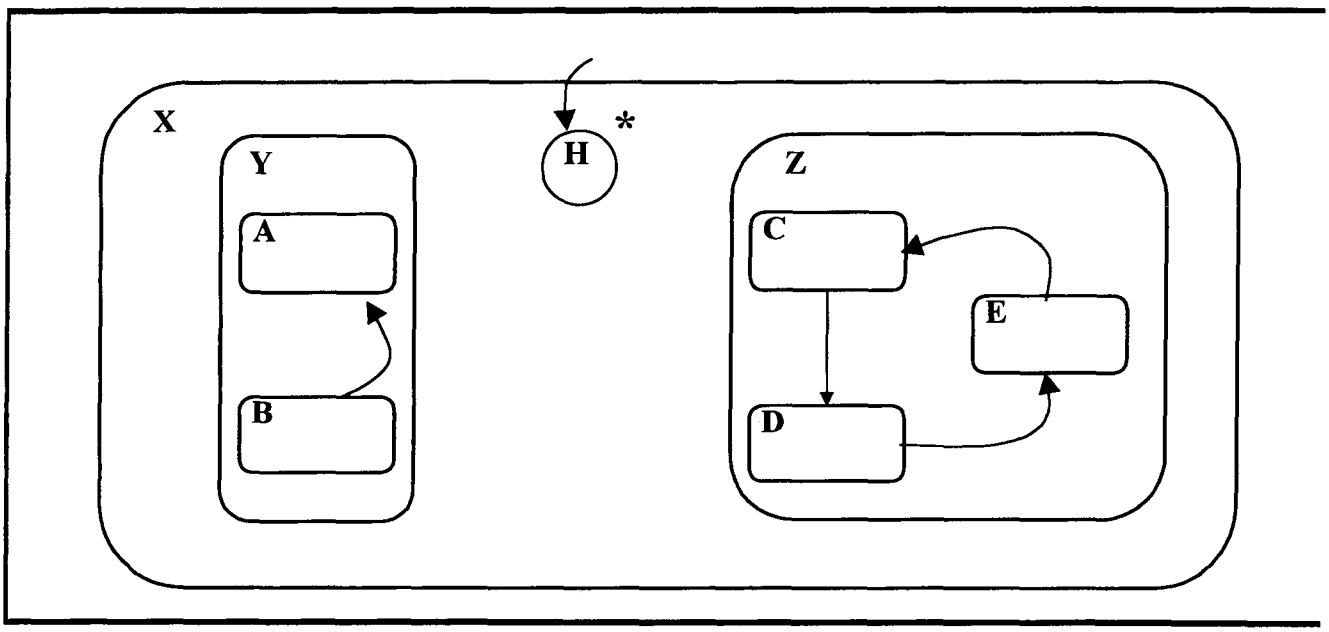

Figura 3.7. Entrada por História H*.

3.3. Entrada por Condição

Em certas situações, pode haver várias entradas para subestados diferentes de um mesmo superestados. Além disso, pode-se querer estabelecer uma condição para cada entrada. A figura 3.8 apresenta a situação descrita anteriormente, onde Q, P e R são condições para B, C e D, respectivamente.

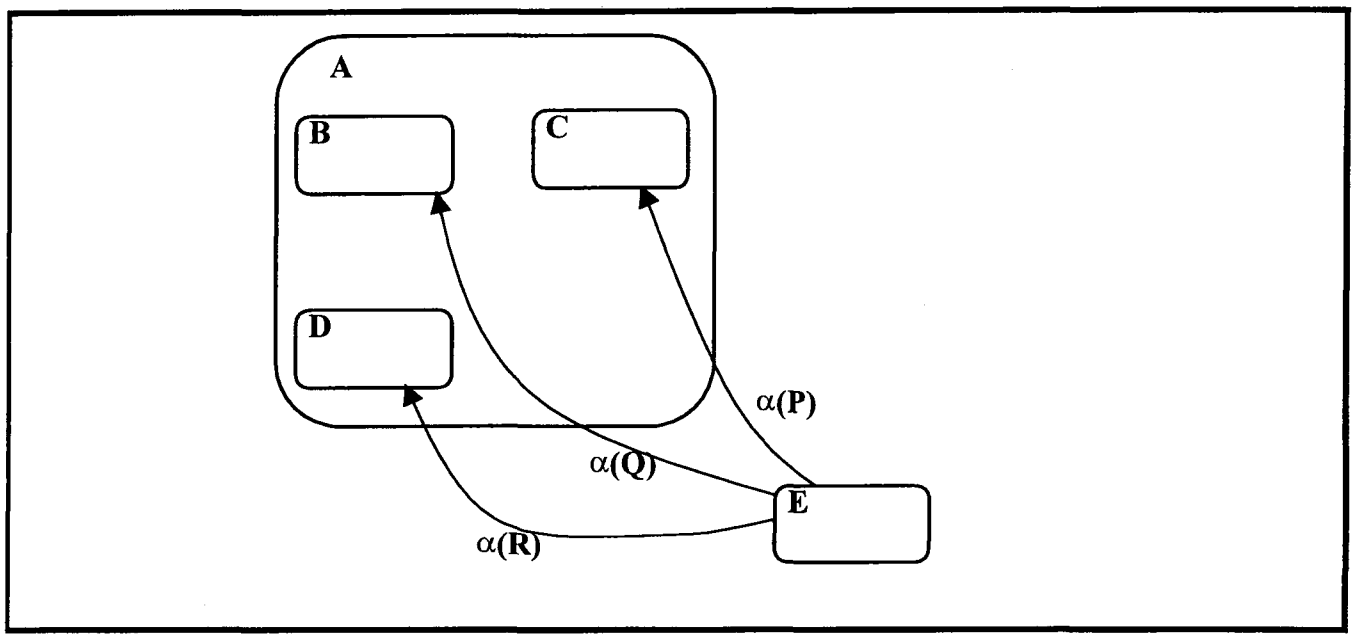

Figura 3.8. Três Entradas cada uma com suas condições associadas.

Uma forma alternativa, disponível nos Statecharts, é a Entrada por Condição, representada pela letra C envolta em um círculo, de maneira que haja apenas a condição em cada arco, a partir da entrada por condição. A figura 3.9 apresenta essa alternativa.

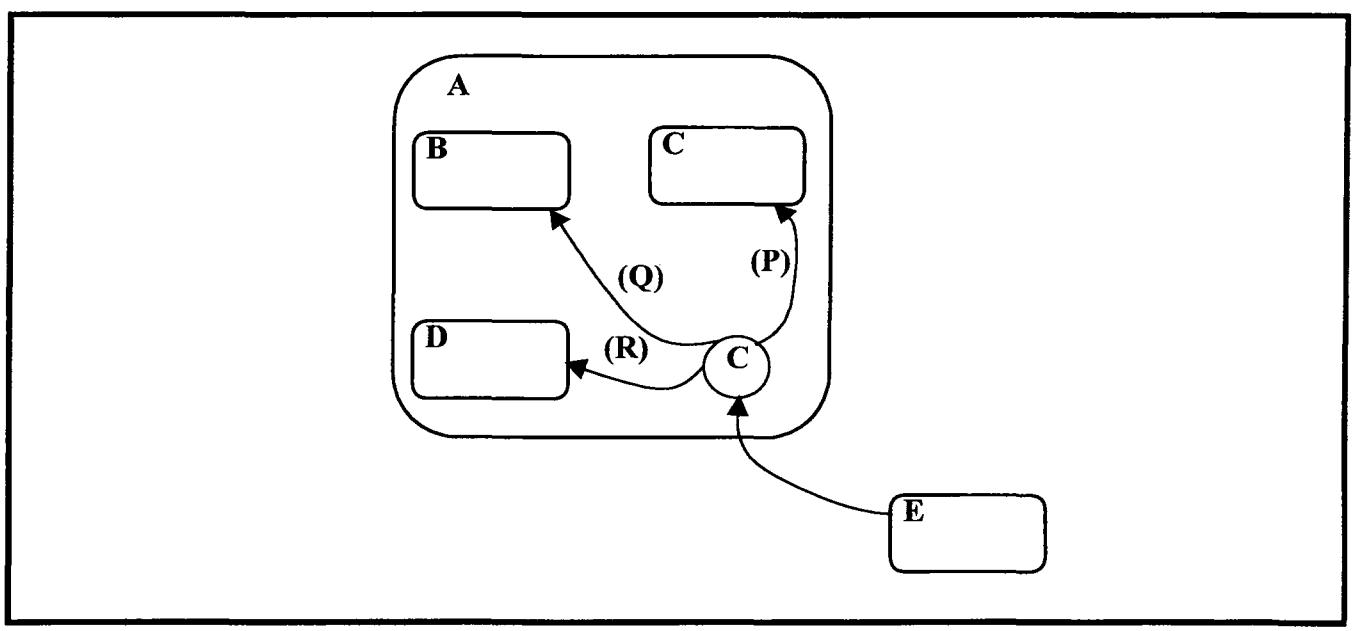

Figura 3.9. Entrada por condição.

3.4. *Delays* e *Timeouts*

Uma possibilidade que pode ser bastante aplicável a aplicações reais é o fato de poder se estabelecer um tempo de permanência em um determinado estado, provocando um retardo (*delay*) no sistema. Além disso, muitas atividades reais precisam estabelecer prazos para começar ou terminar alguma atividade. Esse recurso pode ser introduzido através da notação de *timeout* dos Statecharts. Nessa notação, pode-se estabelecer limites mínimos e máximos de permanência em um estado. A figura 3.10 apresenta a sintaxe usada para *timeout*.

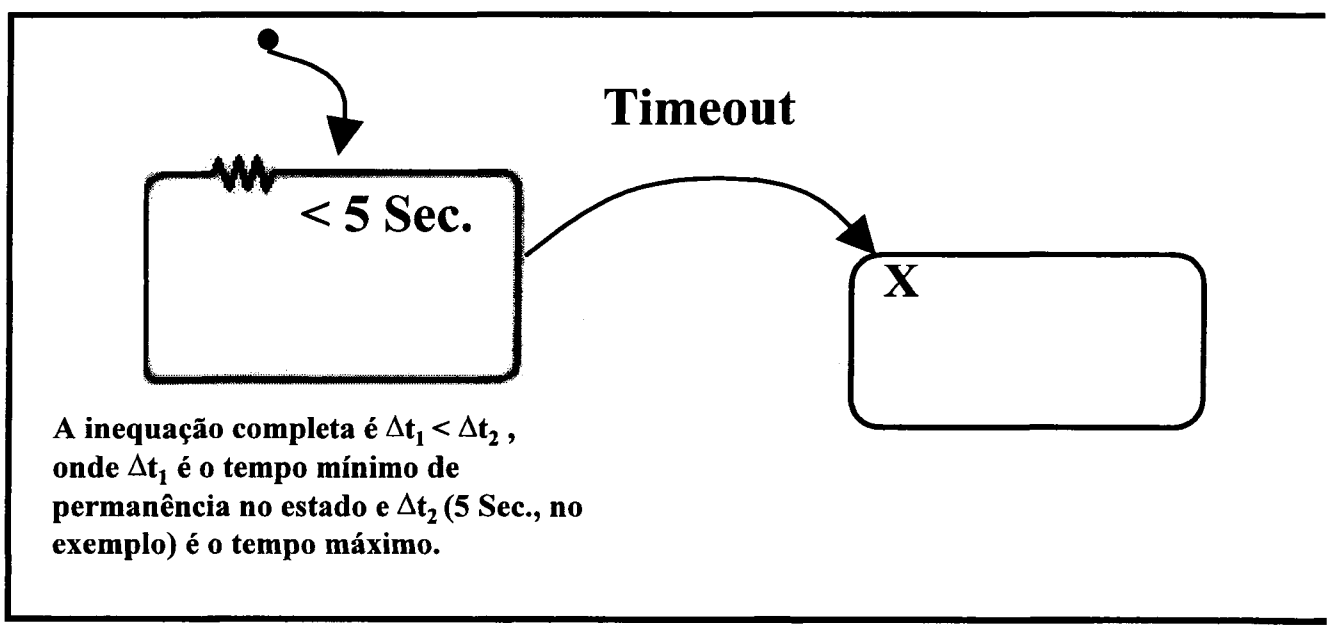

Figura 3.10. Entrada por condição.

No exemplo, o tempo mínimo foi omitido (considerado nulo). O tempo de permanência pode ser encarado como o tempo máximo de espera, que é exatamente como o *timeout* é encarado na maioria dos problemas computacionais. Além disso, o *timeout* introduz uma nova idéia bastante importante para avaliações de desempenho: as transições temporizadas entre estados diferentes. Nas GSPN, esse mesmo processo é realizado nas transições, sendo que lá, os retardos são exponencialmente distribuídos.

Após a introdução realizada até aqui, o leitor já está apto a entender a sintaxe através da qual os Statecharts são definidos. Além disso, serão apresentados conceitos que necessitam da base abordada até aqui.

3.5. Sintaxe do *Statechart*

- **Estados**

O conjunto de estados, ***S***, é função da hierarquia, do tipo, da história e do estado *default*. A função hierárquica define o número de subestados de um estado, ou seja, é uma função do tipo $S \rightarrow 2^n$ ($n \in N$). A função *tipo* define qual o tipo do estado (AND ou OR), isto é, é uma função do tipo $S \rightarrow$ {AND, OR}. A função história representa a capacidade de "lembrar" uma visita anteriormente feita a um estado, ou seja, é do tipo $H \rightarrow S$, onde ***H*** é um subconjunto do conjunto de estados. A função *default* define, para cada superestado, o seu estado inicial.

- **Transições**

Uma transição é definida por $t = (X, I, Y)$, onde X é um conjunto origem, Y é um conjunto destino, e I é um rótulo (Boaventura, 1992). Informalmente, se ***I = e / a***, o sistema está em X e ***e*** ocorre, então ***t*** está habilitada e pode ser tomada. Se ***t*** é tomada, ***a*** é executado e o sistema transiciona então para X.

Além desses dois elementos básicos (estados e transições motivadas por eventos), há mais dois componentes da sintaxe Statecharts: as *Ações* e as *Atividades*. Uma ação é um evento associado a uma transição, também tomado de maneira imediata, denotado por "α / A", onde α representa um evento e A uma ação tomada a partir da execução do evento α. Via de regra, as ações são usadas em estados concorrentes. Por exemplo, na figura 3.11, no estado A_1, mais precisamente na transição entre os subestados C e D, ocorre o evento e_1, que provoca a ação S, como um evento de saída do estado C. Além disso, S será um evento de entrada em A_2 (mais precisamente entre B e E). Assim, a ação é gerada em A_1, mas reflete na configuração de A_2.

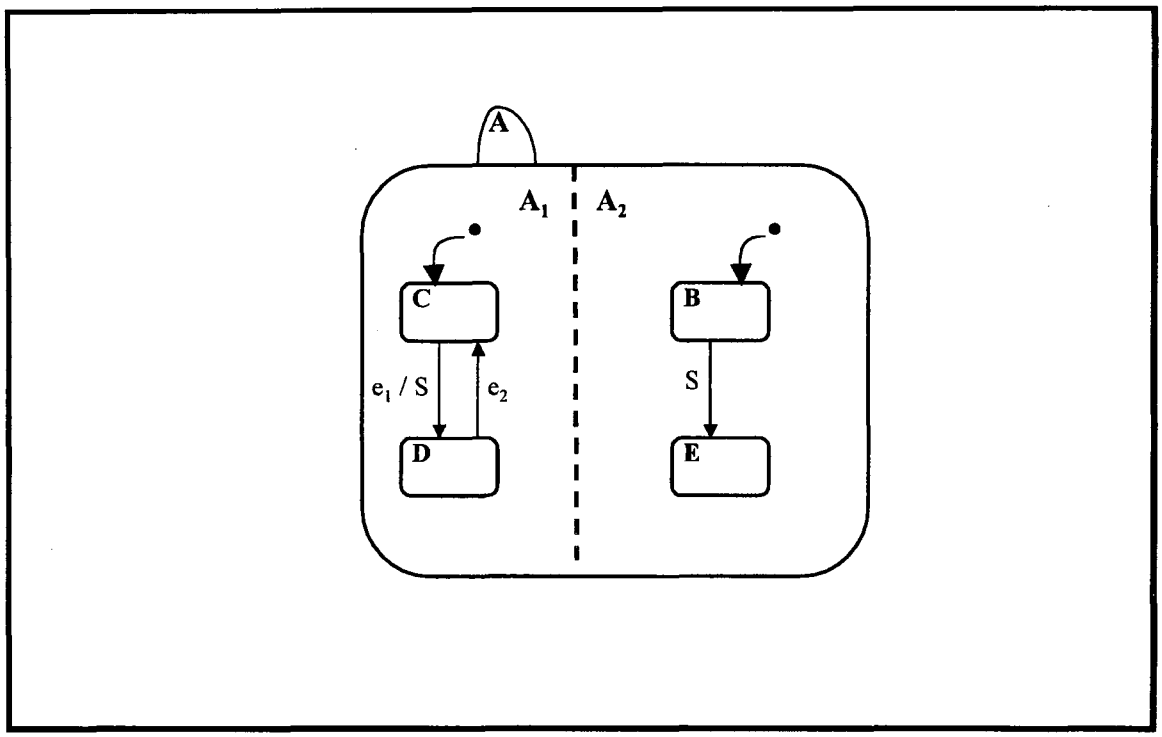

Figura 3.11. Ações em Statecharts.

Vale ressaltar que eventos e ações possuem o mesmo sentido, algo que acontece imediatamente, e que vai levar a uma mudança de estado. A diferença é apenas conceitual, pois, como observado no exemplo anterior, ambos têm uma utilização ligeiramente diferente: eventos são executados no mesmo estado concorrente, e as ações são executadas em outro estado concorrente.

Por fim, as atividades podem ser encaradas como se fossem condições para se realizar determinados eventos, ou seja, as atividades são ações cuja condição é que elas durem um tempo diferente de zero. Atividades são duráveis e não instantâneas (como as ações e os eventos), e por isso devem ter duas ações associadas a uma determinada atividade X: start (X) e stop (X), com significados óbvios. Por exemplo, a atividade de transmissão de pacotes em uma rede de comunicação poderia possui as ações *start* (transmissão) e *stop* (transmissão) associadas, logicamente, considerando que uma transmissão consome um determinado tempo.

3.6. Semântica do *Statechart*

A semântica de execução dos *statecharts* é baseada em um modelo discreto de tempo, segundo o qual os estados de um sistema são duráveis, enquanto que as

transições são instantâneas. O termo passo de execução ou de simulação tem origem nesse modelo de tempo. Diz-se, então, que em passo de execução, o *statechart* assume uma configuração de estados, a qual corresponde ao conjunto de estados básicos ativos naquele passo. Dessa forma, a cada passo, obtém-se uma nova configuração do *statechart*, a qual pode ou não ser igual à configuração anterior, dependendo de quais transições disparem. A figura 4.26 ilustra a configuração inicial de um *statechart*, a partir de seus estados *default*.

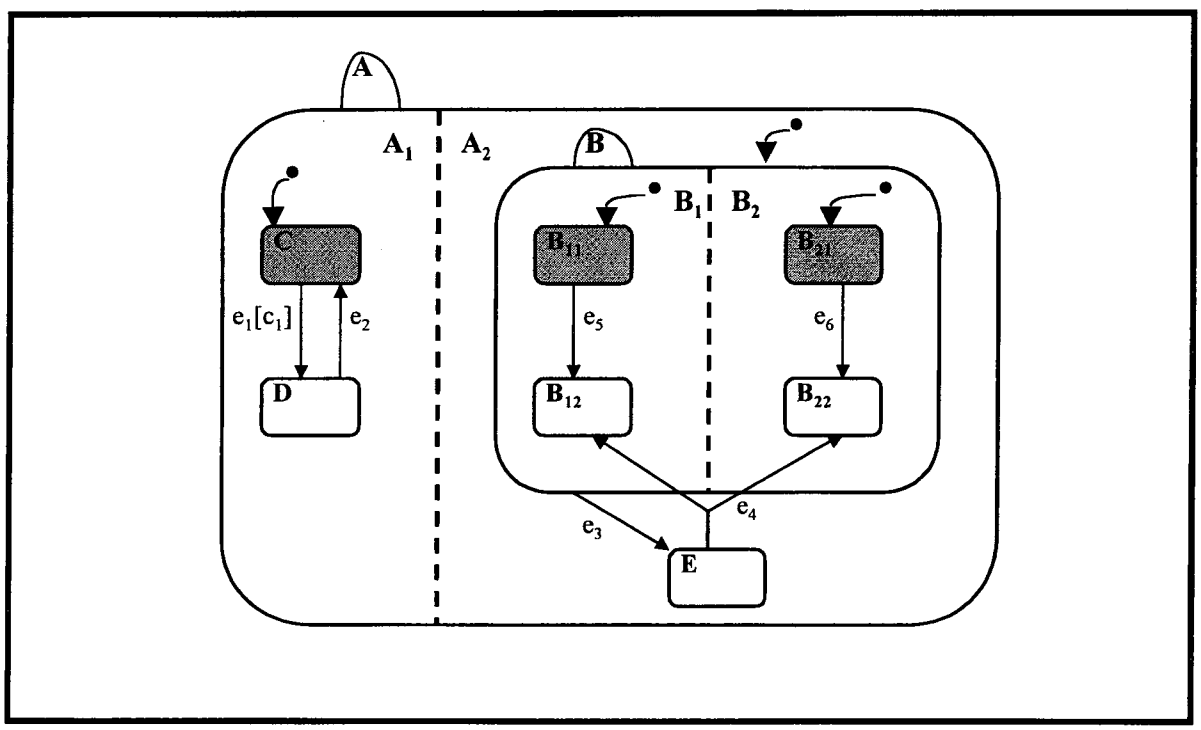

Figura 3.12. Configuração Inicial de um *Statechart*.

Caso os eventos $e_1(c_1)$ e e_5 ocorram, a nova configuração do statechart será igual à ilustrada na figura 4.27. Nota-se que, como os estados A_1 e A_2 são paralelos (ortogonais), os eventos $e_1(c_1)$ e e_5 acontecem simultaneamente.

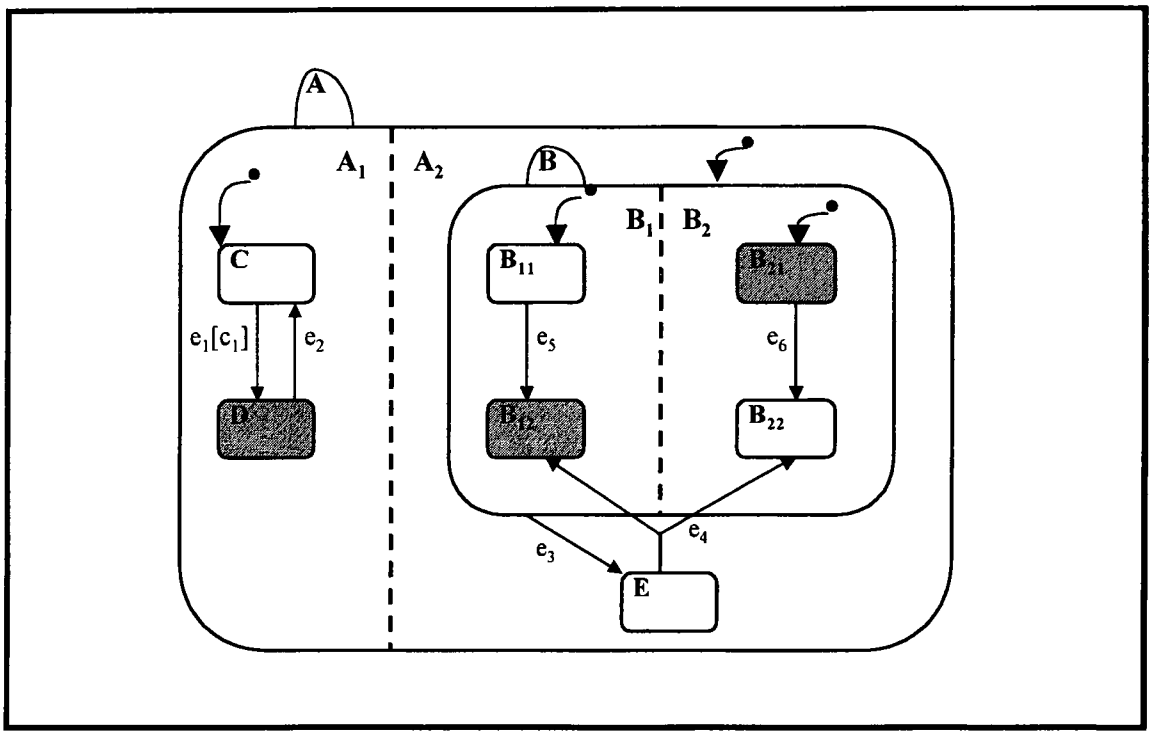

Figura 3.13. Configuração do *Statechart* após a Ocorrência de $e_1(c_1)$ e e_5.

3.7. *Statecharts* Temporais

Os *statecharts* foram estendidos para uma versão voltada à especificação de sistemas de tempo real, dando origem aos "Timed Statecharts" [PNU92]. Nessa extensão, as expressões de eventos são acrescidas de um intervalo [1, u] que determina as fronteiras temporais mínima e máxima do disparo da transição. Os parâmetros temporais associados à transição determinam o tempo durante o qual o sistema permanecerá no conjunto de estados de origem, bem como o atraso permitido para que a transição dispare após a fronteira mínima ter sido alcançada.

Informalmente, pode-se dizer que a regra de disparo de uma transição temporal é a seguinte: uma transição temporal com uma expressão de evento "*t*: [*l*, *u*][*c*]/*a*" associada, irá disparar se a condição ***c*** resultar em *true*. Além disso, a transição só dispara depois de decorridas ***l*** unidades de tempo, podendo levar *u* - *l* unidades de tempo para que isso ocorra. Quando a transição finalmente disparar, a ação ***a*** será executada e o sistema passará a estar no conjunto de estados-destino da transição. A figura 4.28 ilustra um protocolo simples (anteriormente modelado em redes de Petri, figura 4.17), agora modelado em *statecharts* temporais.

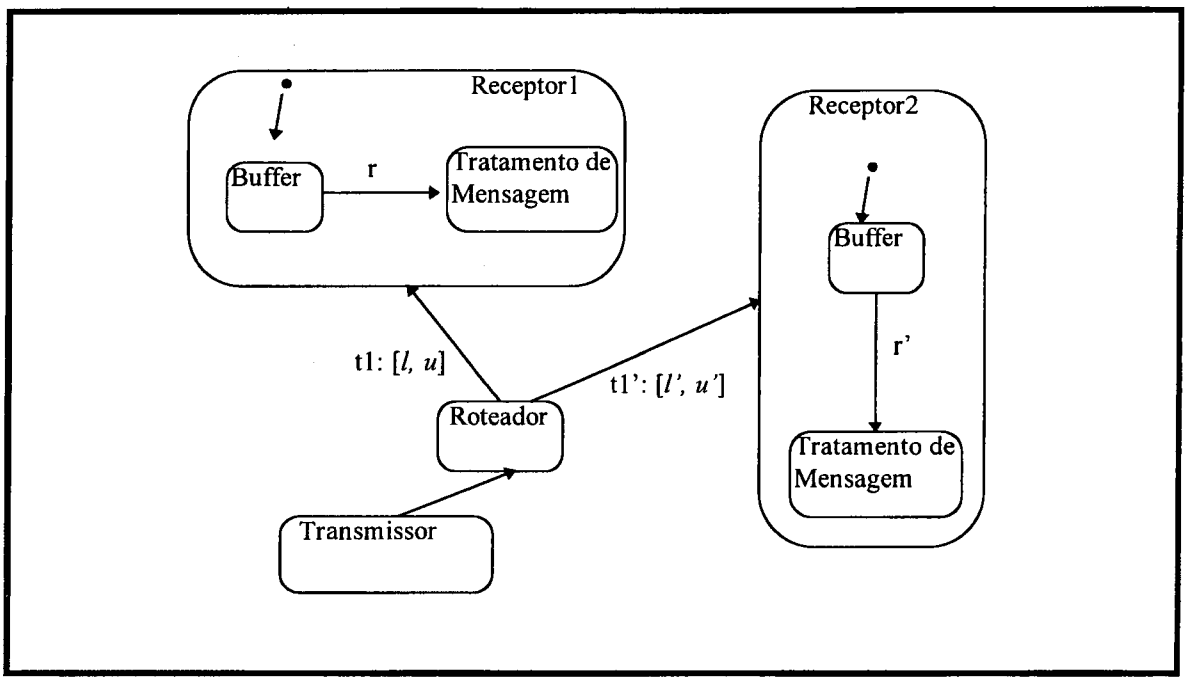

Figura 3.14. *Statechart* Temporal Representando um Protocolo Simples.

Na figura 4.28, a representação temporal evita a possibilidade de existência de escolha não-determinística no estado "Roteador", ou seja, a transição disparada será aquela que possuir menor valor de limite máximo (u ou u'). Assim a mensagem seguirá para o receptor que primeiro conseguir passar do estado "Roteador" para o estado Receptor (1 ou 2).

3.8. Avaliação de Desempenho

Das técnicas abordadas neste trabalho, em sua concepção original, nem redes de Petri e nem Statecharts eram voltados à avaliação de desempenho, e sim para a especificação de sistemas. Entretanto, a necessidade de se representar mais que apenas a abordagem comportamental fez com que fossem criadas extensões a essas técnicas formais, cuja finalidade primordial girava em torno da representação dos sistemas de maneira mais real, ou seja, também em termos estocásticos. Dessa preocupação, surgiram algumas extensões para redes de Petri com esse propósito. Dentre elas, duas se tornaram notórias: *Stochastic Petri Nets* (Molloy, 82) e *Generalized Stochastic Petri Nets* (Chiola, 93), vistas anteriormente neste texto.

Apesar dessa latente necessidade, os *Statecharts* ainda não possuem uma extensão com esse propósito. O próprio idealizador dos *Statecharts*, David Harel (Harel,

87), já sugeria para trabalhos futuros a possibilidade da descrição de processos estocásticos, batizado à época de "Markov-Charts". Uma associação Cadeias de Markov / *Statecharts*, denominada de *Stochastic Feature Charts* (SFC), é apresentada em (Francês, 1998), que visa a determinar de maneira probabilística (e, portanto, mais realista) as mudanças de estados ocorridas em um sistema com o decorrer do tempo. SFC se baseia na teoria de Cadeias de Markov a Tempo Discreto, o que, de certa forma, estabelece certas restrições à análise. A segunda parte deste item apresenta uma abordagem de Statecharts baseados em Cadeias de Markov a Tempo Contínuo, propiciando uma possibilidade de comparação entre as abordagens.

3.8.1. Statecharts e Cadeias de Markov a Tempo Discreto

A partir deste ponto, será dado enfoque a um caso particular de processos estocásticos, com as seguintes propriedades:

(i) Sendo cada resultado pertencente a um conjunto finito de resultados (a_1, a_2, ..., a_m), chamado o espaço de estados do sistema; se o resultado da n-ésima tentativa é a_i, diz-se que o sistema se encontra no estado a_i no instante n;

(ii) O resultado de qualquer ensaio depende no máximo do resultado imediatamente anterior e não de qualquer outro dos estados precedentes; a cada par de estados (a_i, a_j) está associada a probabilidade p_{ij} de que a_j ocorre imediatamente após ter ocorrido a_i;

Então, esse processo estocástico é uma Cadeia de Markov Finita e os números p_{ij} são probabilidades de transição entre dois estados, sendo que cada linha da matriz denominada de matriz de transição apresenta os valores das probabilidades p_{ij} de um estado origem transicionar para um estado destino (Clarke & Disney, 1979).

Na verdade, uma cadeia de Markov é um processo estocástico que possui uma propriedade especial denominada *Memoryless*, ou seja, o futuro não depende da forma em que se chega a um estado s_{n-1}; depende só do fato que no tempo t_{n-1} está-se no estado s_{n-1} (Kovacs, 1996). Partindo-se dessa propriedade, pode-se definir um *Statechart* através de uma analogia às cadeias de Markov, vendo esse *Statechart* através de matrizes de transição e suas propriedades. A título de esclarecimento, é apresentado um exemplo de um servidor de arquivos, conforme a figura 3.15.

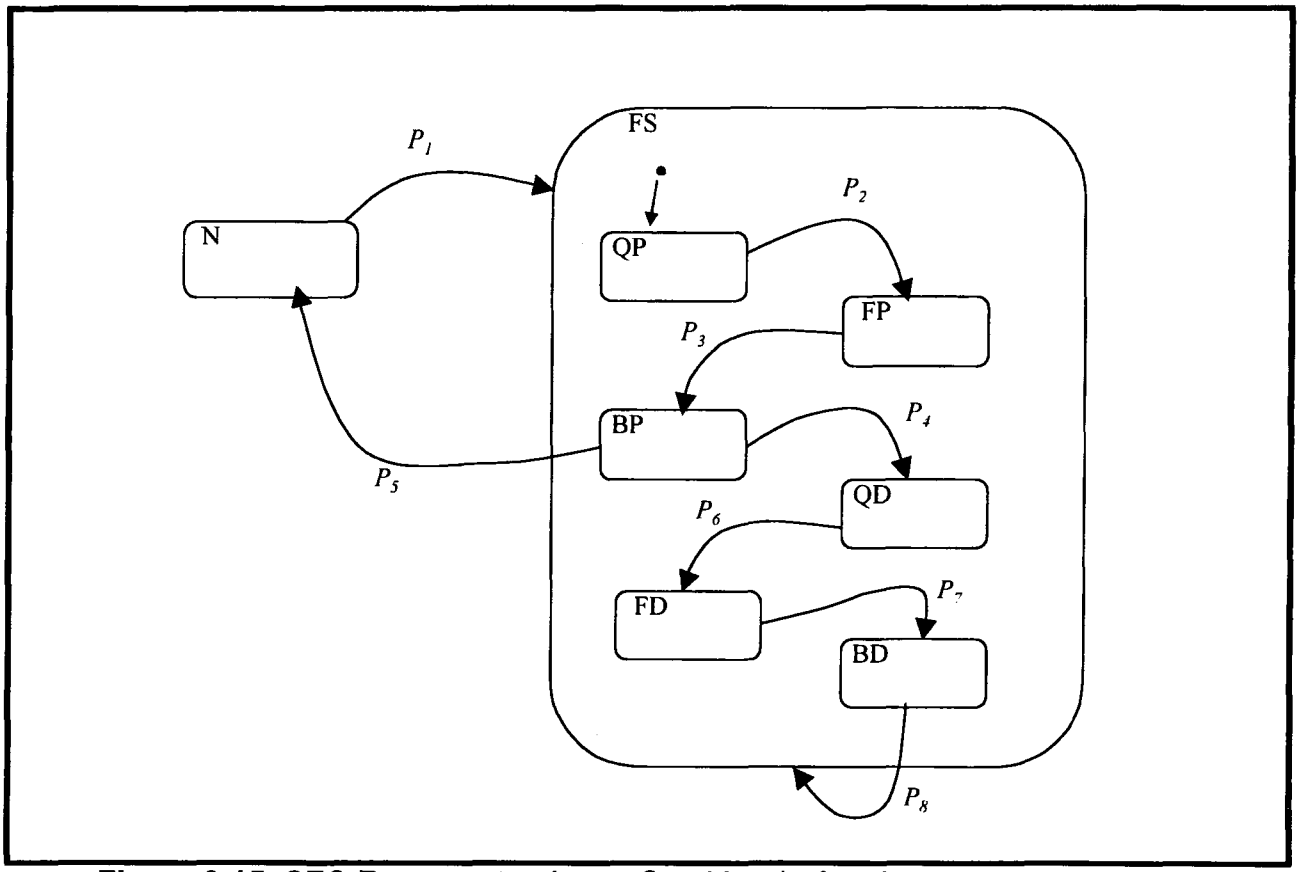

Figura 3.15. SFC Representando um Servidor de Arquivos.

A figura 3.15 apresenta a seguinte situação: se ocorrer uma requisição proveniente de um cliente na rede, o sistema passa do estado N (*Network*) para um estado FS (*File Server*), com uma probabilidade p_1. Esse estado é um superestado, possuindo os subestados PQ (*Processor Queue*), FP (*Free Processor*), BP (*Busy Processor*), DQ (*Disc Queue*), FD (*Free Disc*) e BD (*Busy Disc*) . O estado *default* de FS é QP, pois uma requisição vinda da rede, obrigatoriamente, deve entrar na fila do processador, mesmo que essa fila esteja vazia. Uma vez na fila, a requisição só muda de estado se o processador estiver disponível (estado FP). Se a requisição ocupa o processador, ela o leva a um estado de ocupado (BP). Após o atendimento no processador, a requisição pode retornar à rede (com uma probabilidade p_5) ou ir para a fila do disco (DQ), com probabilidade p_4. A segunda possibilidade é efetuada quando se desejar realizar uma operação de entrada/saída no disco. No caso da segunda hipótese, a requisição vai ao disco se ele estiver livre (FD), e o ocupa (BD). Vale ressaltar que p_4 e p_5 são equiprováveis, isto é, possuem a mesma probabilidade.

A base da análise provida pelas Cadeias de Markov está no *Teorema das Probabilidades-Limite*, que diz, de maneira bastante simplificada, que em certas condições, pode-se achar uma matriz (que é um produto da matriz da transição original), cujas linhas tendem a um vetor, chamado vetor de probabilidades-limite. O significado físico desse vetor é que cada componente dele representa o tempo médio que o sistema passou no estado correspondente àquela componente.

	N	FS	PQ	FP	BP	DQ	FD	BD
N	0	1	0	0	0	0	0	0
FS	0	0	1	0	0	0	0	0
PQ	0	0	0	1	0	0	0	0
FP	0	0	0	0	1	0	0	0
BP	1/2	0	0	0	0	1/2	0	0
DQ	0	0	0	0	0	0	1	0
FD	0	0	0	0	0	0	0	1
BD	0	1	0	0	0	0	0	0

Figura 3.16. Matriz de Transição do Servidor de Arquivos.

Para a matriz do servidor de arquivos (figura 3.16), vetor encontrado está especificado a seguir (figura 3.17). Pelas componentes desse vetor, pode-se obter o tempo médio de permanência de cada estado. Assim, por exemplo, uma requisição pode ter passado 1/4 do seu tempo em filas (1/6 do tempo na fila do processador (PQ) e 1/6 na fila do disco (DQ)), caso ela tenha precisado de uma operação de entrada/saída no disco; ou o processador passou 1/6 do tempo ocioso (FP) e o disco passou 1/12 do tempo do experimento ocupado (BD).

N	FS	PQ	FP	BP	DQ	FD	BD
(1/12	1/6	1/6	1/6	1/6	1/12	1/12	1/12)

Figura 3.17. Vetor de Probabilidades-limite do Servidor de Arquivos.

Com a abordagem utilizada até aqui, foi feita uma "discretização" do tempo. Uma outra opção é tratar o tempo como um fenômeno contínuo, como ele realmente é. O próximo tópico aborda essa outra possibilidade de análise.

3.8.2. Statecharts e Cadeias de Markov a Tempo Contínuo.

O estudo realizado neste item foi extraído na íntegra de (Vijaykumar, 1999) e apresenta um modelo que descreve um sistema de manufatura com duas máquinas Ma e Mb, um depósito S, um robô Rb e um operador Op. As máquinas produzem o mesmo tipo de produto, que após confeccionado, é sempre colocado no depósito para que a máquina fique livre. A ação de carregar produto nas máquinas, retirá-los depois de prontos, colocá-los no depósito é realizada pelo robô. O operador entra em ação na ocorrência de falhas ou nas máquinas ou no robô. Os estados em que as máquinas podem se encontrar são W, P, WU e B, para "esperando para ser carregada", "processando um produto", "esperando para ser descarregada" e "falha", respectivamente. Todos os tempos envolvidos no processo são considerados exponencialmente distribuídos (tempo para falhar máquina ou robô, tempo para reparar as falhas, tempo para processar cada produto e tempos de carga e descarga de produtos).

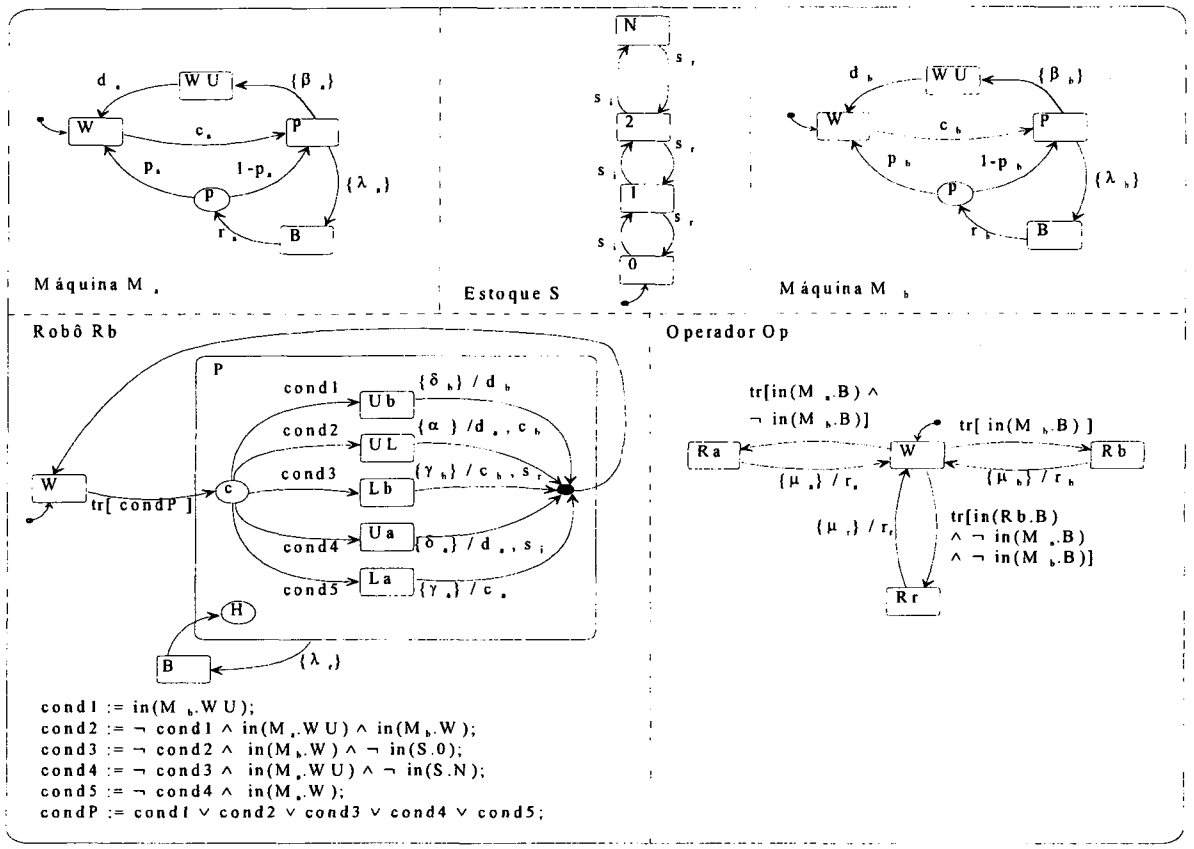

Figura 3.18. *Statechart* Representando um Sistema de Manufatura.

O estado inicial de cada máquina é W. Um evento ca (cb, respectivamente) gerado pelo robô corresponde a carregar um produto na máquina Ma (Mb, respectivamente) e transicionar o modelo de W para P. A partir de P, as máquinas podem ir ou para "aguardar descarga" ou para "falha". O tempo para processar um produto também é considerado exponencialmente distribuído com parâmetros βa e βb para Ma e Mb, respectivamente, assim como o tempo de falha com parâmetros λa e λb para Ma e Mb, respectivamente. Se uma falha ocorre, a máquina é transicionada de P para B e, nesse caso, um produto pode ser perdido com probabilidade pa para Ma ou com probabilidade pb para Mb. Os eventos ra e rb indicam o término do reparo feito pelo operador quando uma máquina falha.

Cada estado do depósito representa o número de produtos que ele contém. Os eventos si e sr representam a colocação e a retirada de um produto do depósito, respectivamente.

Os estados do robô são W, P e B, para "aguardando", "processando" e "falha", respectivamente. Os tempos de carga para Ma, descarga de Ma para S, carga de S

para Mb, e de descarga de Mb são também considerados exponencialmente distribuídos, com parâmetros γa, δa, γb, δb, respectivamente. Os tempos para descarga de Ma e carga de Mb são considerados exponencialmente distribuídos, com parâmetro α.

O operador Op pode se encontrar nos estados W, Ra, Rb e Rr que correspondem a "esperando", "reparando Ma", "reparando Mb" e "reparando o robô". Os tempos de reparo são exponencialmente distribuídos com parâmetros μa, μb, e μr para Ma, Mb e o robô, respectivamente.

- Resultados Numéricos

Os valores numéricos usados são mostrados na tabela 2. A partir deles, são obtidas as medidas de desempenho da tabela 3 para as máquinas Ma e Mb, além da disponibilidade do robô.

	Máquina Ma	Máquina Mb	Robô
Taxa de Produção	βa = 8, 10 or 12	βb = 10	
Taxa de Falha	λa = 1	λb = 0,5	λr = 1
Taxa de Reparo	μa = 10	μb = 15	μr = 10
Probabilidade de Perda de Produto	Pa = 0,5	pb = 0,3	
Taxa de Carga de Ma			γa = 100
Taxa de Descarga de Ma			δa = 100
Taxa de Carga de Mb			γb = 100
Taxa de Descarga de Mb			δb = 100
Taxa de Movimento de Ma para Mb			α = 70

Tabela 3.1. Valores Numéricos Utilizados.

	Máquina Ma	Máquina Mb	Robô
Taxa Média de Produção (produtos/unidades de tempo)	6,91	6,81	
Taxa Média de Perda de Produtos (produtos/unidades de tempo)	0,35	0,10	
Disponibilidade	92,9%	97,5%	97,5%

Tabela 3.2. Medidas de Desempenho.

3.9. Ferramentas que Utilizam Notação *Statecharts*

O artigo (Harel et al, 1990) descreve o sistema STATEMATE com um conjunto de ferramentas, com uma forte orientação gráfica, destinado à especificação, análise, projeto e documentação de sistema reativos complexos, tais como sistemas de tempo real, sistemas de comunicação e controle, além de *hardware* ou *software* interativos. A especificação é feita sob três pontos de vista interrelacionados: estrutural, funcional e comportamental. Esse último ponto de vista - o comportamental - é baseado na utilização de *statecharts*.

(Meira & Masiero, 1991) apresentam um gerador de aplicação, o StatSim, que se baseia em uma interface gráfica, onde podem ser construídos modelos baseados em *statecharts*. Nesse ambiente, está implementada toda sintaxe permitida pelos *statecharts*. A idéia é criar o *statechart* e depois disparar suas transições, observando o comportamento que o mesmo toma a partir desses disparos, através de uma animação.

Outro trabalho relevante é a proposição de uma arquitetura de classes que foi desenvolvida para projetar todas as entidades que formam a sintaxe *statechart* (Vijaykumar et al, 1997). A implementação dessa arquitetura é proposta levando-se em consideração duas fases: uma fase de especificação e outra fase da dinâmica do *statechart*. A especificação é responsável por representar o comportamento do modelo. A fase da dinâmica é responsável pela reação do sistema quando estimulado pelos eventos que ocorrem no decorrer do tempo. A especificação contempla as seguintes classes: *Node* (que encapsula as classes State, EntryByHistory e EntryByCondition), *State, EntryByHistory, EntryByCondition, Condition, Action*, *Event* e *Transition*. Já a dinâmica congrega as seguintes classes: *Configuration, Stimuli, Reactor, GraphBase* e *GraphGenerator*. Apesar da maioria das classes ser auto-explicativa, a descrição completa de cada uma delas pode ser encontrada em (Vijaykumar et al, 1997).

3.10. Propriedades Dinâmicas

Para explicar as propriedades dos *statecharts* torna-se necessária a compreensão da construção de uma árvore de alcançabilidade, semelhantes às árvores definidas para as Redes de Petri. Essa definição e o algoritmo sua construção são apresentados integralmente por Boaventura (Boaventura, 1992).

A árvore é construída levando-se em conta todas as seqüências possíveis de eventos do sistema em estudo. A cada passo considera-se que os eventos são verdadeiros, o que habilita todas as transições e leva a uma nova configuração. Cada nó da árvore é associado a uma configuração de estados do sistema em estudo, e podem ser classificados como:

- Configuração nova: configuração gerada e não inserida na árvore;

- Configuração velha: configuração já tratada e inserida na árvore;

- Configuração terminal: não apresenta transição habilitada;

- Configuração histórica: representa todas as configurações alcançáveis a partir de transições ou estados associados ao símbolo de história.

A árvore de alcançabilidade é construída a partir da configuração inicial, que se torna a sua raiz. Essa raiz é uma configuração nova a ser tratada, e são considerados todos os eventos que podem ocorrer. Cada evento produz uma nova configuração que é inserida na árvore. O processo é repetido até que não existam mais configurações novas a serem tratadas. Isso significa que todas as configurações alcançáveis foram tratadas e tornaram-se configurações velhas ou terminais.

A notação utilizada para a representação de um nó da árvore compreende parênteses, que representam os superestados do tipo XOR, e colchetes, que representam os superestados do tipo AND. Os estados átomos são representados por 1 ou 0, ou também pelo seu nome.

Como um exemplo de construção de uma árvore de alcançabilidade pode-se considerar o *statechart* da figura x.x, também extraído de (Boaventura, 1992). Esse contém duas transições, f e d, associadas a símbolos de história, e o estado destino é D. Esse estado é visitado por *default* e essa configuração é inserida como a raiz da árvore de alcançabilidade. Cria-se então o nó história.

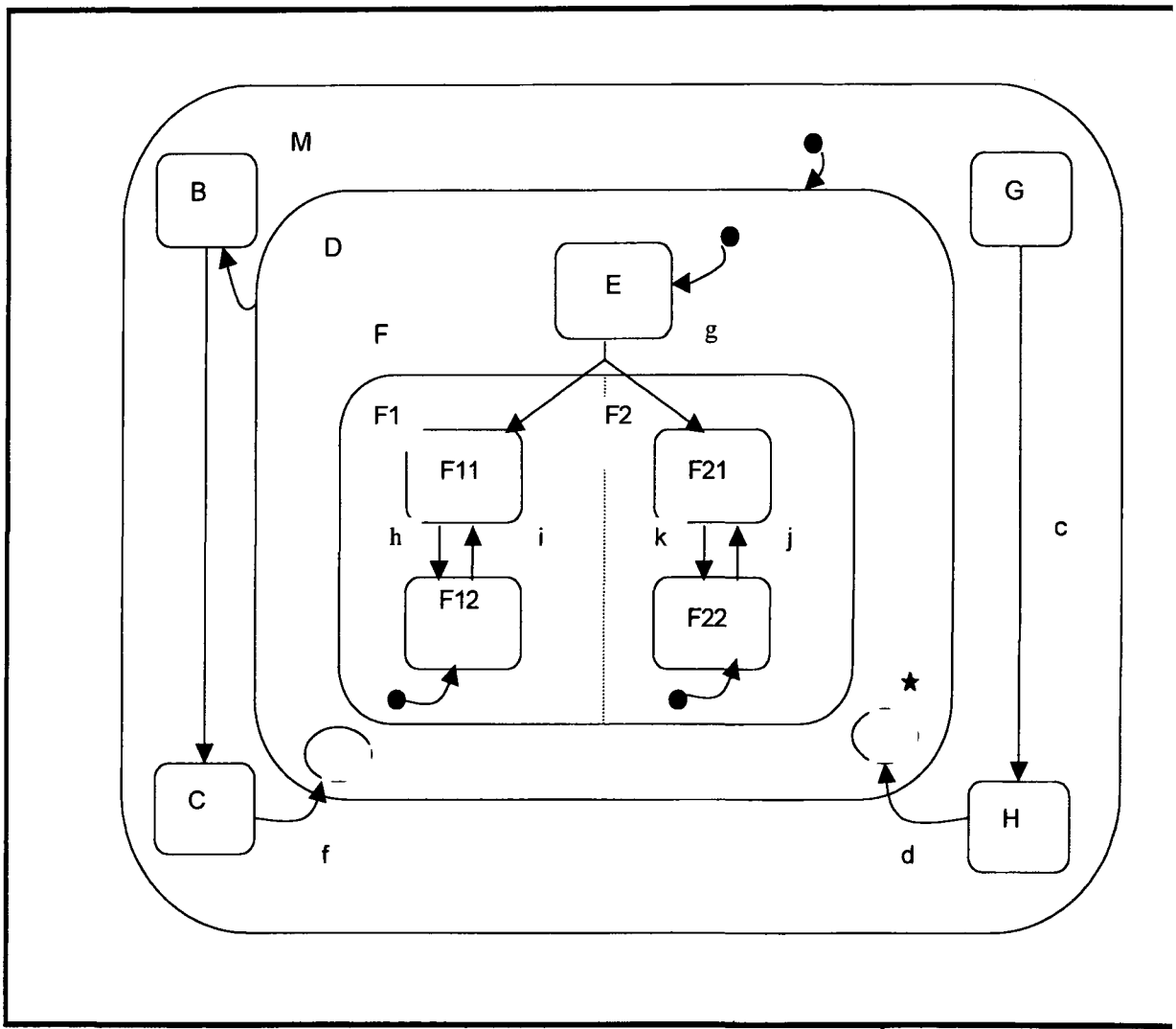

Figura 3.19. Um *Statechart* para exemplificar a construção da Árvore de Alncançabilidade.

A árvore de alcançabilidade para esse *statechart* é apresentada na figura x.x. Um nó da árvore tem a seguinte descrição:

(H, G, C, B, (E, [F12, F11),(F22,F21)]))

onde os estados não foram representados com 0's e 1's e sim através de seus nomes, para facilitar a compreensão.

Para organizar a árvore, definiu-se que para toda configuração de estados, onde são representados os subestados de D ligados, consideram-se as transições a e d. Outra solução poderia considerar essas transições apenas no primeiro nó da árvore

onde o estado D estivesse ligado, mas essa solução poderia dificultar o processo de percorrer a árvore.

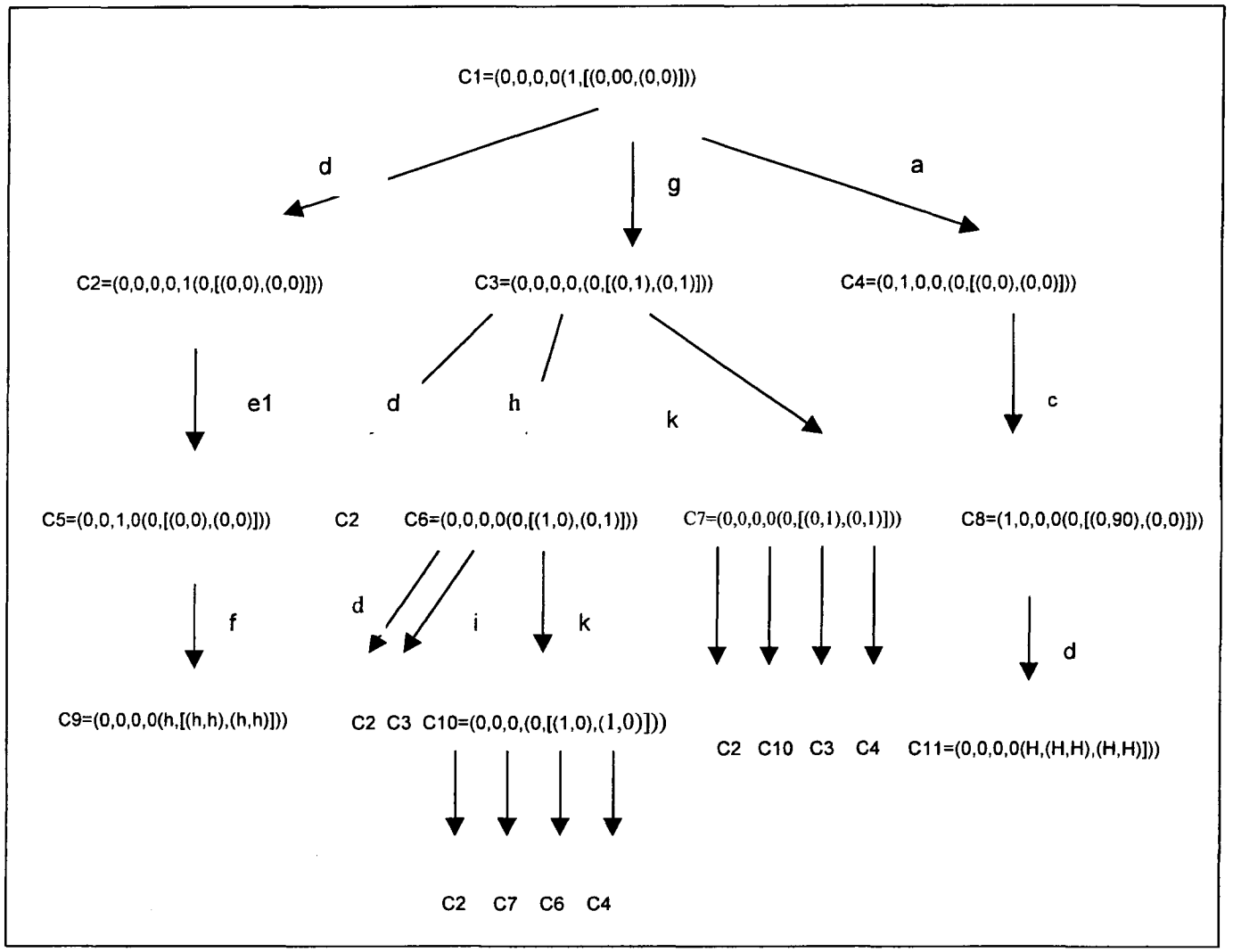

Figura 3.20. Árvore de Alcançabilidade para o *Statechart* da figura 3.19.

As propriedades dos *Statecharts* são parecidas com as propriedades definidas para as Redes de Petri. Boaventura (Boaventura, 1992) define e faz um estudo sobre os *statecharts*, suas propriedades e como verificar essas propriedades através da Árvore de Alcançabilidade:

- Vivacidade (Impasse ou *Deadlock*): um modelo em *statechart* poderá apresentar *deadlocks* se alcançar alguma configuração de estados nas quais não existam transições habilitadas para disparar. Em sua árvore de alcançabilidade aparecerão nós terminais (mas existem situações onde as

atividades possuem início e fim de processamento bem definidos e portanto a árvore apresenta nós terminais, que não são terminais);

- Seqüência Válida de Eventos: uma seqüência de eventos é valida se cada um de seus eventos provocar uma alteração na configuração de estados do modelo;

- Alcançabilidade: garante se a partir de uma configuração de estados inicial pode-se alcançar uma outra determinada configuração;

- Reiniciabilidade: diz respeito à possibilidade de retornar à configuração inicial do *statechart*, a partir de qualquer configuração de estados alcançável;

- Uso de Transição: relacionado com o conceito de vivacidade, isto é, uma transição está viva se pode ser disparada. Garante que o sistema foi especificado corretamente, não existem transições que não podem ser disparadas.

4. Análise Comparativa

Este tópico pretende apresentar uma comparação entre representações das duas técnicas abordadas no decorrer deste texto. Ambas, apesar de baseadas em estados, possuem peculiaridades, que podem ser colocadas em termos de vantagens, desvantagens e semelhanças. A seguir são apresentadas essas peculiaridades.

4.1. Vantagens da Representação de Redes de Petri (Ordinárias e de Alto Nível)

- As redes de Petri Ordinárias (ou de alto nível) permitem que os lugares sejam genéricos, isto é, eles podem ser tanto recursos de um sistema (por exemplo, um processador), quano situações abstratas (como "recurso ocupado");
- É possível representar a situação inicial do sistema, através da marcação inicial;

- É possível representar a situação corrente do sistema, através dos *tokens*;
- *Tokens* podem ser individualizados, através de, por exemplo, cores das redes de Petri coloridas;
- As extensões hierárquicas permitem uma maior compactação do modelo, mesmo que em detrimento da clareza do modelo.

4.2. Vantagens da Representação de Redes de Petri Estocásticas (SPN e GSPN)

- As GSPN possuem tanto representações para transições imediatas (aquelas que são disparadas com tempo zero) quanto para transições temporizadas (timed), chamadas de *Firing Delay* ;
- Utilizam-se de mecanismos para resolver conflitos estruturais, como arcos inibidores;
- Além de possuírem as vantagens da funcionalidade das redes de Petri ordinárias.

4.3. Desvantagens da Representação de Redes de Petri

- As redes de Petri não possuem uma representação para tratar filas, pois um lugar com vários *tokens* não pode ser encarado como uma fila (uma fila deve ter um algoritmo de escalonamento, uma taxa de chegada e, possivelmente, prioridades para clientes);
- A representação de paralelismo não é explícita;
- Um lugar não pode ser subdividido em sublugares, o que pode levar à explosão do número de lugares e transições do modelo;
- Poucas ferramentas implementam extensões hierárquicas, que possibilitam uma maior compactação do modelo.

4.4. Vantagens da Representação de Statecharts

- Os estados dos Statecharts também são de caráter genérico, podendo representar tanto recursos, quanto situações abstratas;
- Podem representar a situação inicial do sistema, através dos estados *default*;

- Podem representar a situação corrente do sistema, através do recurso chamado *passos* dos statecharts;
- A representação dos statecharts é originalmente hierárquica, através da utilização dos superestados e seus componentes (subestados);
- Possuem mecanismos para representar paralelismo de maneira explícita, através dos estados do tipo AND.

4.5. Desvantagens da Representação de Statecharts

- Não possui representação para tratar filas;
- Não individualiza clientes, como as redes de Petri fazem com os *tokens*, o que não permite que clientes sejam acompanhados dentro do sistema;
- As transições (que são representadas por arcos) são sempre tratadas como imediatas, o que nem sempre é interessante, como foi apresentado nas GSPN (*firing delay*).

4.6. Uma Representação Híbrida Deveria Ter

- Representação de filas, com (algoritmos de escalonamento, taxas entre chegadas, prioridades, e outras características relevantes);
- Individualização de clientes, possibilitando fazer inferências sobre o comportamento de um cliente em particular, como tempo total no sistema, tempo de fila e de serviço;
- Transições imediatas e temporizadas, para poder associar, por exemplo, tempos exponenciais às transições;
- Alguma forma de paralelismo explícito;
- Mecanismos para resolver situações não-determinísticas, como por exemplo, probabilidades, tempos, prioridades, etc.

4.7. Comparação das Propriedades:

Uma ótima comparação das propriedades das Redes de Petri e *Statecharts* é apresentada por Boaventura (Boaventura, 1992) e resumida na tabela 4.1. Não são definidas as propriedades limitabilidade, segurança e conservação para um *statechart*.

Propriedades	Redes de Petri	*Statecharts*
Limitabilidade	Uma Rede de Petri é k-limitada se em cada um de seus lugares não existam mais de k *tokens*	
Segurança	Ocorre quando k =1, portanto é um caso especial de limitabiilidade	
Conservação	Uma Rede de Petri é conservativa se a soma dos *tokens* em cada marcação não se alterar	
Vivacidade	Garante que o sistema não irá ficar permanentemente inativo	Garante que não existe *deadlock* no *Statechart*, ou seja, dada uma configuração inicial não existe configuração alcançável na qual nenhuma transição está habilitada
Seqüência de Disparos	Pode-se verificar a possibilidade de disparo de uma seqüência específica de transições ou de um subconjunto dessa seqüência	A definição é semelhante à definição em Redes de Petri, aqui denominada Seqüência Válida de Eventos
Alcançabilidade	Indica se a partir de uma marcação inicial pode-se atingir uma determinada marcação	A definição é a mesma que nas Redes de Petri, mas fala-se em alcançabilidade de configurações de estados
Reiniciabilidade	Permite ao sistema modelado retornar ao seu estado inicial após a execução	Um *Statechart* é definido como reiniciável se a partir de qualquer configuração pode-se retornar à sua configuração inicial
Uso de Transição	Essa propriedade está relacionada com vivacidade, ou seja, uma transição é considerada vida se pode ser disparada	A definição é a mesma que nas Redes de Petri

Tabela 4.1. Comparação entre as Propriedades de Redes de Petri e Statecharts

4.8. Comparação da Avaliação de Desempenho

Esta seção é baseada em um estudo apresentado em (Francês et al., 2000). Esse estudo faz uma comparação entre três modelos que representam um servidor de arquivos, com taxas estipuladas, e com a geração das medidas de desempenho através de um modelo em redes de filas, um em redes de Petri (GSPN) e outro em Statecharts. A figura 3.8 apresenta o modelo do servidor (e sua parametrização) através da representação de redes de filas.

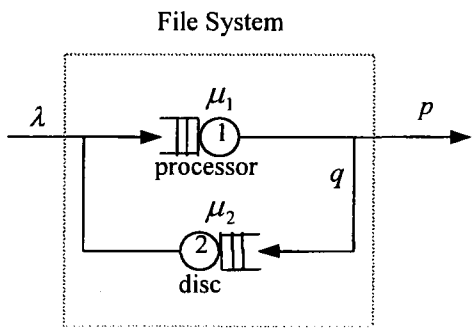

Figura 4.1. Servidor de Arquivos em Redes de Filas.

Em 4.1, as requisições chegam a uma taxa de 12, e são atendidas a uma taxa de serviço de 25. Após serem atendidas, as requisições podem sair do sistema com uma probabilidade *p* de 60%, e com uma probabilidade *q* de 40% para ir para a fila do disco, o qual atende a uma taxa de 20. Os resultados obtidos para a rede de filas do exemplo foram retirados de (Silva, 2000). As figuras 4.2 e 4.3 apresentam a mesma situação em rede de Petri (mais precisamente em GSPN) e Statecharts, respectivamente.

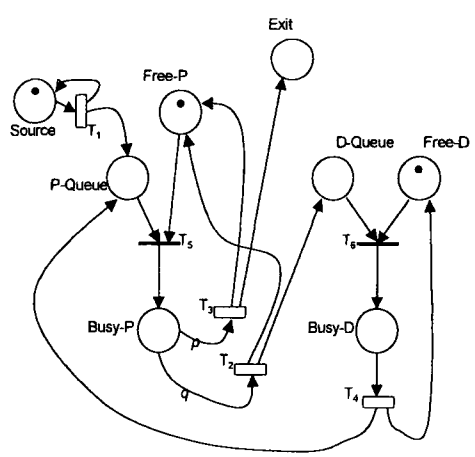

Figura 4.2. Servidor de Arquivos em GSPN.

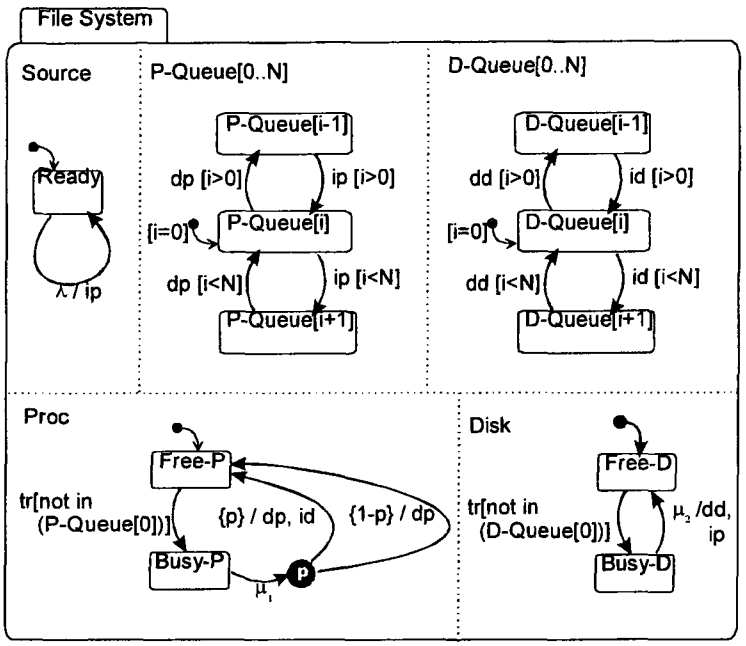

Figura 4.3. Servidor de Arquivos em Statecharts.

Os resultados obtidos para o modelo em rede de filas utilizaram a solução analítica rede de Jackson. Os resultados obtidos para o modelo em rede de Petri utilizaram simulação, através da ferramenta DNANet (Attieh et al.,1995). Já os resultados calculados para os Statecharts foram obtidos através da solução analítica cadeias de Markov a tempo contínuo, utilizando o método SOR, implementada em um ambiente para análise de desempenho de modelos markovianos (Andrade et al., 1997). A tabela 4.2 apresenta os valores de entrada.

Taxa de chegada λ	12
Taxa de Processamento μ_1	25
Taxa de atendimento do disco μ_2	20
Probabilidade de deixar o sistema p	0.6
Probabilidade de usar o disco q = 1-p	0.4

Tabela 4.2. Valores de Entrada para o Servidor de Arquivos.

A tabela 4.3 apresenta as medidas de desempenho obtidas para cada técnica.

Estados	Redes de Filas	Redes de Petri	Statecharts
Processador Ocupado	0.800	0.799	0.800
Processador Ocioso	0.200	0.200	0.199
Disco Ocupado	0.400	0.401	0.400
Disco Ocioso	0.600	0.598	0.599

Tabela 4.3. Medidas de Desempenho para as três técnicas.

Alguns aspectos são importantes de serem observados no experimento. Os modelos de redes de filas e de Petri utilizam filas de tamanho ilimitado. Essa simplificação facilita o tratamento matemático do problema, apesar de não ser muito realística, pois os lugares físicos que servem de filas são finitos (*buffers*, por exemplo). A avaliação Statecharts levou essa restrição em consideração, experimentando vários tamanhos de filas tanto para o processador quanto para o disco, para verificar o impacto que essa mudança causaria. A tabela 4.4 apresenta a comparação dos valores para diferentes tamanhos de *buffers*.

Fila do Processador (posições)	**5**	**10**	**20**	**30**
Fila do Disco (posições)	**3**	**5**	**10**	**15**
Processador Ocupado	0.728	0.781	0.798	0.800
Disco Ocupado	0.384	0.397	0.400	0.400
Processador Ocioso	0.271	0.218	0.201	0.199
Disco Ocioso	0.615	0.602	0.599	0.599

Tabela 4.4. Comparação entre Diferentes Tamanhos de *Buffers*.

Pela tabela 4.4, há uma diferença de utilização que aumenta à medida em que aumentam as posições das filas do processador e do disco, até o limite de trinta posições. A partir desse valor, há uma tendência dos valores se estacionarem, quando, provavelmente, o sistema começa a entrar em equilíbrio.

5. Referências Bibliográficas

Alphatech Corporation. ALPHA/Sim User's Guide. 1995.

ANDRADE, V.M.B., CARVALHO, S.V., VIJAYKUMAR, N.L Desenvolvimento de um Software para análise de desempenho de sistemas através de modelos markovianos. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 29, Salvador, 1997. *Anais* . Salvador, SOBRAPO.

Artifex Getting Started. ARTIS Software Corporation. 1999.

ATTIEH, A., BRADY, M.C., KNOTTENBELT, W.J., KRITZINGER. (1995, May). Functional and Temporal Analysis of Concurrent Systems. http://www.cs.uct.ac.za/~william/DNAnet.html.

BOAVENTURA, I. A. G.. Propriedade Dinâmica de *Statecharts*. Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação. 1992.

FRANCÊS, C.R.L. *Stochastic Feature Charts – uma Extensão Estocástica para os Statecharts*. Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação. 1998.

FRANCÊS, C.R.L, VIJAYKUMAR, N.L., SANTANA, M.J., SOLON, V. C., SANTANA, R.H.C. Stochastic Extension to Statecharts for Representing Performance Models: an Application to a File System. *SBAC-PAD'2000 - Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho*, São Pedro, 2000. Artigo aceito.

JEFFERSON, D. R.. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7, no. 3 (July): 404-425.

MARSAN, M. A.; BALBO. G.; CONTE, G.. *Performance Models of Multiprocessor Systems*. MIT Series in Computer Systems. 1986.

MARSAN, M. A.. Stochastic Petri Nets: An Elementary Introduction. Lecture Notes in Computer Science, n. 424, p. 1-30. 1989.

MARSAN,

MARSAN, M. A.; BOBBIO, A.; DONATELLI, S.. Petri Nets in Performance Analysis: An Introduction. Lecture Notes in Computer Science, n. 1491, p. 211-256. 1998.

MOORE, K. E.; HAMMER, S. D.. ALPHA/Sim Simulation Software Tutorial. *Proceedings of the 1998 Winter Simulation Conference*, p. 289-296.

OCHI, M. K.. *Applied Probability and Stochastic Process in Engineering and Physical Siences.* John Wiley & Sons. 1990.

SILVA, A R. F. *Modelos de redes de filas para Sistemas Computacionais Distribuídos-Simulação X Métodos Analíticos.* São Carlos, 2000. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo (USP).

SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C.. Distributed Simulation, Time Warp and Its Variants: Taxonomy and Performance Evaluation Issues. *Proceedings of the 13th European Simulation Multiconference*, p. 220-227. 1999.

VIJAYKUMAR, N. L.; CARVALHO, S. V.; ANDRADE, V. M. B. A Statecharts Tool for Performance Models: An Object Oriented Approach. *Accepted for presentation at APORS'97-Asian-Pacific Operations Research Societies*, Melbourne, Australia, November-December 1997

VIJAYKUMAR, N.L., CARVALHO, S.V., ABDURAHIMAN, V. *Statecharts: Their Use in Specifying and Dealing with Performance Models.* São José dos Campos, 1999. Tese (Doutorado) – Instituto Tecnológico da Aeronáutica (ITA).