Comparação de Técnicas de Redes de Petri na Descrição de um Pipeline

Cristiano Pires Martins[1], **Aleardo Manacero Jr.**[2]

[1]Centro Universitário de Jales - UNIJALES
Av. Francisco Jalles, 1851 - Centro
Jales/SP CEP 17500-000

[2]DCCE – Universidade Estadual Paulista - UNESP

cpmart@uol.com.br, aleardo@ibilce.unesp.br

Abstract. *The use of Petri nets to model digital systems is intensive, usually through several distinct extensions from Petri's original proposal. Some of these extensions are applied on systems where timed probabilistic events are observed. Here it is shown how the behaviour of these extensions can influence the analysis of digital systems, such as a pipeline. The paper presents the characteristics of a simplified pipeline model built from three Petri net extensions, one of them with especial features for timed and probabilistic events. Results achieved with the simulation of the three models are provided.*

Resumo. *O uso de redes de Petri na modelagem de sistemas digitais é intensivo, ocorrendo através de diversas extensões ao modelo originalmente proposto por Petri. Várias dessas extensões tratam de sistemas em que eventos temporizados, com ocorrências probabilísticas, são observados. Neste trabalho mostra-se como o comportamento de tais extensões influencia a análise de sistemas digitais, como um* pipeline. *Apresentam-se as características observadas nos modelos de um* pipeline *simplificado, construídos a partir de três técnicas, uma delas configurada especialmente para sistemas temporizados com características probabilísticas. Os resultados obtidos com tais modelos são apresentados.*

1. Introdução

Redes de Petri são uma ferramenta de grande uso na modelagem e especificação de sistemas em que o sincronismo entre eventos seja essencial [Cardoso and Valette 1997]. Nesse contexto surgem aplicações, como sistemas de manufatura, tempo-real, avaliação de desempenho, tolerância à falhas ou projeto de arquiteturas computacionais complexas.

Essa variedade de aplicações implica na existência de várias técnicas de redes de Petri, em que se acrescentam diferentes características ao seu modelo original, com o objetivo de resolver mais facilmente um determinado tipo de problema. Assim, surgem as redes estocásticas [Molloy 1982], as redes coloridas [Jensen 1996], temporizadas [Merlin and Farber 1976, Wang 1998], etc. Na aplicação em sistemas digitais essa variedade de técnicas se mantém, com vantagens e desvantagens inerentes a cada uma delas.

O objetivo deste trabalho é comparar algumas técnicas de rede de Petri quando usadas na descrição de um sistema digital em que seja necessário o tratamento de tempo

e eventos aleatórios. Como exemplo dessa classe adota-se o *pipeline*, dada a sua importância crescente no projeto e desenvolvimento de processadores mais velozes.

Serão tratadas aqui três das técnicas normalmente usadas, por serem representativas das principais classes de modelos. Numa das técnicas se faz apenas o tratamento de tempo. Noutra acrescenta-se a capacidade de tratamento estocástico. Finalmente, na terceira técnica, aqui denominada STERN (de **S**tochastic **T**imed **E**nvironment **R**elationship **N**et), se combina a flexibilidade das redes coloridas ao tratamento de tempo e de probabilidades.

Na próxima seção apresenta-se uma descrição das técnicas aqui utilizadas, seguida dos problemas no projeto de um *pipeline* simplificado. Seguem-se a descrição dos modelos para esse *pipeline* e os resultados, obtidos através de um simulador, com a aplicação destas técnicas.

2. Conceitos básicos

Antes de examinar a aplicação de redes de Petri na descrição de um *pipeline*, é interessante definir mais objetivamente os parâmetros utilizados no restante do texto. Para tanto, serão apresentadas nesta seção tanto o formalismo básico de redes de Petri (e suas extensões), como as características desejadas no *pipeline*.

2.1. Redes de Petri

Rede de Petri (RdP) é uma ferramenta matemática, para modelagem de sistemas, que permite tratamentos algébricos ou por simulação a partir de uma estratégia relativamente simples de compor as interrelações entre os eventos ocorrendo no sistema. Sua configuração original previa quatro elementos básicos, que são os arcos, lugares, transições e marcas. Embora esses elementos, vistos na Figura 1, possam ser interpretados de forma ampla, uma definição mais básica é a seguinte:

- **Lugar:** interpretado como um estado do sistema, sendo representado por círculos;
- **Transição:** associada com a ocorrência de eventos no sistema, sendo representada por uma barra ou retângulo;
- **Marca:** indicador de que a condição associada ao lugar em que aparece é satisfeita, representada por pontos;
- **Arcos:** fazem o mapeamento entre transições e lugares, sendo de entrada (quando vão de um lugar para uma transição) ou de saída (no sentido oposto).

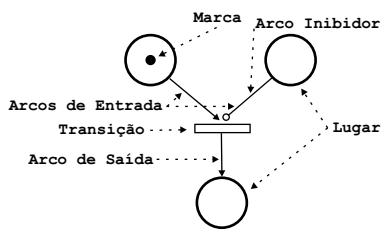

Figura 1. Rede de Petri.

Aos arcos se associam ainda duas outras características, que são o peso e inibição. O peso de um arco de entrada indica o número de marcas necessárias em seu lugar de

origem para o disparo da transição, enquanto o peso de um arco de saída indica quantas marcas serão criadas no lugar para o qual aponta. Já a inibição ocorre apenas em arcos de entrada, representada por um pequeno círculo no lugar da ponta do arco. Sua existência indica que aquela transição apenas pode disparar na ausência de marcas no lugar de que parte o arco de inibição.

De forma geral, o funcionamento de uma RdP baseia-se no disparo (execução) das transições, desde que suas condições de disparo (marcas nos lugares de entrada) sejam satisfeitas. Esses disparos consomem marcas nas entradas e geram novas marcas nos lugares de saída. As diferentes técnicas de RdP se distinguem basicamente na forma de controle das condições de disparo, como é apresentado a seguir.

2.1.1. Redes de Petri Baseadas em Tempo (RdPBT) -

Nessas redes cada marca é associada ao instante de disparo da transição que a criou. Associado ao registro de tempo existe uma função *time*, tal que *time(p)* é o instante de criação da marca, e portanto do disparo da transição que a criou.

Para cada transição se definem intervalos de disparo, dentro dos quais é possível disparar a transição se ela estiver habilitada naquele momento. Uma transição está habilitada se e somente se acontecer o seguinte:

1. Cada lugar de entrada contém o número necessário de marcas. Se arcos inibidores forem usados, então o lugar de entrada conectado com eles deve estar vazio. Esta condição é idêntica às RdP's comuns;
2. Cada transição, quando necessário, deve ter indicado o tempo mínimo de espera antes de se tornar habilitada com base no tempo de criação de cada marca no(s) lugar(es) de entrada, assim, como, um tempo máximo de duração para essa habilitação. Todas as marcas devem chegar aos seus lugares de entrada antes que expire o intervalo da habilitação.

A execução das transições obedece às seguintes regras:

i Somente transições habilitadas podem disparar. O disparo de uma dada transição deve ocorrer em algum instante durante o intervalo de habilitação;
ii Quando uma transição *t* dispara, as marcas necessárias são removidas dos lugares de entrada de *t*. Aqui a escolha das marcas a serem removidas, em geral, depende dos registros de tempo;
iii O disparo da transição entrega à saída um número apropriado de marcas, todas associadas com o tempo do disparo e de acordo com o peso do arco;
iv As demais regras das RdP não temporizadas permanecem aplicáveis.

Deve-se salientar que essa é apenas uma das formas para tratamento de tempo em RdP. Outras formas aparecem, p. ex., em [Buy and Sloan 1994, Ghezzi et al. 1991].

2.1.2. Redes de Petri Estocásticas Generalizadas (RdPEG) -

As redes de Petri estocásticas generalizadas são redes temporizadas nas quais classificam-se as transições em dois grupos [Molloy 1982, Granda et al. 1992], de acordo com a sua

prioridade. Assim, existem as transições imediatas, que têm prioridade de disparo, e as temporizadas, sendo que para as primeiras se define probabilidades de disparo e para as últimas atrasos aleatórios com distribuição exponencial.

O procedimento de disparo das transições e execução de uma RdPEG, segue basicamente o procedimento das RdP básicas, exceto pela forma de habilitação das transições, que é a seguinte:

1. Transições imediatas: a presença das marcas nos lugares de entrada habilita a transição com uma probabilidade de disparo *Pr*, ou não (probabilidade *1- Pr*);
2. Transições temporizadas: a presença das marcas nos lugares de entrada faz com que se habilite a transição após a espera de um atraso determinado aleatoriamente, segundo uma função de distribuição exponencial.

Uma característica adicional das RdPEG, muito útil no tratamento de sistemas em que a evolução no tempo é um aspecto importante, é de que as mesmas são isomorfas às cadeias de Markov (CM), sendo que seus estados correspondem aos estados da CM equivalente [Granda et al. 1992].

2.1.3. Rede de Petri Colorida (RdPC) -

A descrição das redes coloridas é necessária, nesse contexto, por serem a base da construção do modelo STERN, visto a seguir. A característica principal das RdPC é uma estruturação das marcas no sistema, diferenciando-se cada marca, pelos eventos e condições em que ocorreu sua geração, através de cores [Jensen 1996]. Isso implica em se associar, para cada transição, um conjunto de regras de disparo baseadas em cores.

Em um modelo de rede de Petri Colorida uma transição está habilitada se e somente se há marcas suficientes, com as cores determinadas, conforme os pesos dos arcos de entrada em cada lugar de entrada. Essa condição deve permanecer verdadeira até o disparo da transição, quando as marcas são consumidas conforme os pesos e as regras de disparo relacionadas às cores delas, enquanto novas marcas são produzidas nos lugares de saída, também conforme as regras de cores.

2.1.4. Redes STERN -

Apesar de as RdPC oferecerem um tratamento diferenciado de cada marca, permitindo mais flexibilidade no tratamento de eventos em sistemas complexos, seu funcionamento básico não contempla tempo ou eventos probabilísticos. Assim, várias extensões ao modelo básico de RdPC, com ou sem a explicitação de cores, têm sido propostas incluíndo tais parâmetros no modelo [Ghezzi et al. 1991, M. Nielsen 2001]. Nesse trabalho apresenta-se novo um modelo híbrido, compondo as RdPEG, RdPBT e RdPC, definindo-se os lugares da rede como sendo portadores de ambientes (as estruturas definidas pelas marcas coloridas) e as transições como operadores de relacionamentos entre os ambientes.

Numa STERN o funcionamento segue exatamente as restrições de cada um dos modelos que a compoem, assim, o disparo de uma transição ocorre se:

i Houver, em seus lugares de entrada, marcas em quantidades e cores determinadas, de forma a habilitar a transição;

ii O disparo pode ocorrer apenas dentro do intervalo de habilitação definido pelas condições de tempo da transição;
iii Vencer a disputa de probabilidades, se for uma transição imediata (RdPEG).

2.2. Características de *pipelines*

Pipelines são componentes de microprocessadores que, em última análise, permitem o paralelismo ao nível de instruções. Todos os processadores modernos fazem amplo uso desse tipo de componente, uma vez que a aceleração obtida no processamento é alta, sendo obtida a um custo relativamente pequeno, principalmente em processadores que usem tecnologia RISC, uma vez que a técnica funciona melhor quando as instruções possuem formato e tamanho padronizado.

Em linhas gerais, *pipeline* é uma técnica em que múltiplas instruções são sobrepostas durante a execução. Para isso é preciso dividir sua execução em fases, que seriam executadas pelos respectivos estágios do *pipeline*. Isso implica na elaboração de algo parecido com uma linha de montagem que, apesar de não diminuir o tempo de execução individual de cada instrução, permite o aumento na velocidade de processamento pela freqüência com que as instruções saem do *pipeline*. Embora os processadores atuais possuam bastante estágios (dez ou mais), pode-se entender o conceito envolvido com essa técnica através de um *pipeline* simples, de apenas cinco estágios, descritos a seguir:

- **Busca**, encarregado da busca das instruções no *cache* de instruções, para posterior decodificação;
- **Decodificação**, em que se faz a decodificação da instrução, identificando quais serão sua operação e dados de entrada e saída;
- **Busca de Dados**, em que se faz a busca pelos dados de entrada no *cache* de dados (se necessário) ou se identifica os registradores em que os mesmos estão;
- **Execução**, encarregado da execução efetiva da instrução, operando sobre os dados dos registradores por ela indicados, disponibilizando seus resultados ao próximo estágio;
- **Escrita**, em que se armazena os resultados obtidos no estágio anterior. É a partir deste estágio que as instruções são consideradas terminadas.

Mesmo a partir desses poucos estágios de um *pipeline* é possível identificar alguns perigos em sua operacionalização. Esses perigos, ou conflitos, ocorrem por problemas **estruturais**, quando o *hardware* não suporta a combinação de instruções em execução, **de controle**, quando uma decisão deve ser tomada a partir do resultado de uma instrução ainda em execução, ou **de dados**, quando os dados de entrada de uma instrução são a saída de uma instrução ainda em execução. Um outro problema é o fato de a instrução (ou de seus dados) não estarem em *cache*, o que implicaria em retardar a execução de todo o conjunto até que essa informação estivesse disponível.

Esse último problema é de fundamental importância, uma vez que o fluxo de execução do *pipeline* apenas pode ser mantido se as instruções e seus dados estiverem facilmente disponíveis em *cache* e não na memória RAM. Uma forma de se fazer isso é através do carregamento antecipado das instruções em *cache*, técnica esta conhecida como carregamento especulativo. Apesar de importante para o funcionamento correto do *pipeline*, o carregamento especulativo depende, fundamentalmente, do trabalho eficiente do compilador e não será analisado aqui com mais detalhes.

Já os problemas de conflitos dependem de como se projetou o *pipeline*. Os conflitos estruturais, quando existentes, não têm solução, como o caso de disputa pelo acesso a um estágio do *pipeline*. Já os conflitos de controle e de dados merecem um exame mais cuidadoso, uma vez que a sua ocorrência pode retardar a execução do fluxo de instruções.

Os conflitos de dados são resultado de uma má ordenação das instruções, quando se coloca duas instruções seguidas que apresentam dependência de dados. A solução para esse tipo de conflito está no trabalho mais eficiente do compilador (reordenando as instruções para evitar que instruções com dependência de dados sejam vizinhas), ou pela inserção de uma estrutura de envio de resultados (*forwarding unit*), como ocorre em processadores MIPS [Hennessy and Patterson 2002].

Finalmente, os maiores problemas surgem com os conflitos de controle, sendo originados a partir da dificuldade no tratamento de instruções de desvio, que ocupam em geral cerca de 30% do código de um programa. O problema é que o resultado de uma instrução de desvio define se a próxima instrução a ser executada é aquela que se segue (dentro do fluxo normal de endereços) ou uma outra, determinada pelo endereço de salto. Como essa definição ocorre apenas após o estágio de execução, é impossível determinar *a priori* qual o fluxo que deve se seguir à instrução de desvio. Esse problema é agravado pela ocorrência de erros na previsão do fluxo, quando o *pipeline* deverá ser esvaziado para que o fluxo correto possa ser carregado, desperdiçando uma quantidade razoável de ciclos a cada vez que isso ocorre.

Em alguns casos é possível fazer o tratamento de desvio de forma estática, deixando a cargo do compilador identificar situações em que o salto é altamente improvável (como exceções em chamadas de funções do sistema), ou no desdobramento de laços de repetição para pequenas quantidades de iterações [Smith 1981]. Outras formas de se fazer o tratamento de desvios incluem a inserção de instruções de atraso (*delay slots*) pelo compilador, duplicação de caminhos no *pipeline* (com a duplicação de seus estágios iniciais), e previsão dinâmica de fluxo. Apenas essa última será descrita aqui por estar em uso nos modelos da próxima seção.

Na previsão dinâmica de fluxo [Evers and Yeh 2001] se faz a definição do caminho a ser seguido a partir de uma estratégia de tentativas e erros, baseada na contagem de erros e acertos nos caminhos já percorridos. Assim, um contador interno ao *pipeline* é incrementado a cada vez que se acerta o caminho e decrementado em caso contrário. Toda vez que a contagem chega a zero o controle de desvios altera a política de escolha atual (instrução seguinte ou instrução de salto). Testes com essa abordagem mostram resultados bastante bons no acerto dos desvios.

3. Modelos de *pipelines* usando redes de Petri

Após a descrição dos modelos básicos de redes de Petri e dos problemas característicos do projeto de um *pipeline* pode-se passar à descrição dos modelos de *pipeline* que podem ser gerados através de redes de Petri. Nesta seção serão examinados os modelos de *pipeline* gerados para as redes baseada em tempo (RdPBT), estocástica (RdPEG) e o modelo híbrido (STERN). Em particular, nos três casos adotou-se um *pipeline* com quatro estágios e módulos adicionais de busca em *cache* e tratamento de desvios com esvaziamento de estágios no caso de insucessos, sendo indicados, para cada rede, quais lugares representam tais módulos.

3.1. *Pipeline* usando uma RdPBT

A Figura 2 apresenta o modelo obtido utilizando-se apenas a técnica de rede de Petri Baseada em Tempo. Ele é composto por quatro módulos (que seriam futuros circuitos), que são: seqüência de estágios (p5 até p19), gerenciamento de *cache* de instruções (p1, p3) e de dados (p2, p4), tratamento de desvios (p22 até p30) e módulo de esvaziamento do *pipeline* (p20 e p21).

O módulo de estágios é composto pelos estágios de busca, decodificação, execução e gravação dos resultados. Em particular, como as transições t12 e t13 são concorrentes e com probabilidades desiguais de disparo, é bastante difícil tratar essa desigualdade apenas a partir de restrições de tempo.

O módulo de gerenciamento de *cache*, tanto de dados quanto instruções, também padece do mesmo problema para modelar a existência ou não do recurso em *cache*. Já o tratamento de atrasos em função de busca em memória é facilitado exatamente pela existência das restrições de tempo.

O módulo responsável pelo esvaziamento do *pipeline*, em caso de erro de previsão, usa também o tratamento de tempo para simular a necessidade ou não desse procedimento. Em particular, dados estatísticos indicam que isso ocorre em 11% das instruções, o que é modelado pelo disparo da transição t19 (ver Figura 2).

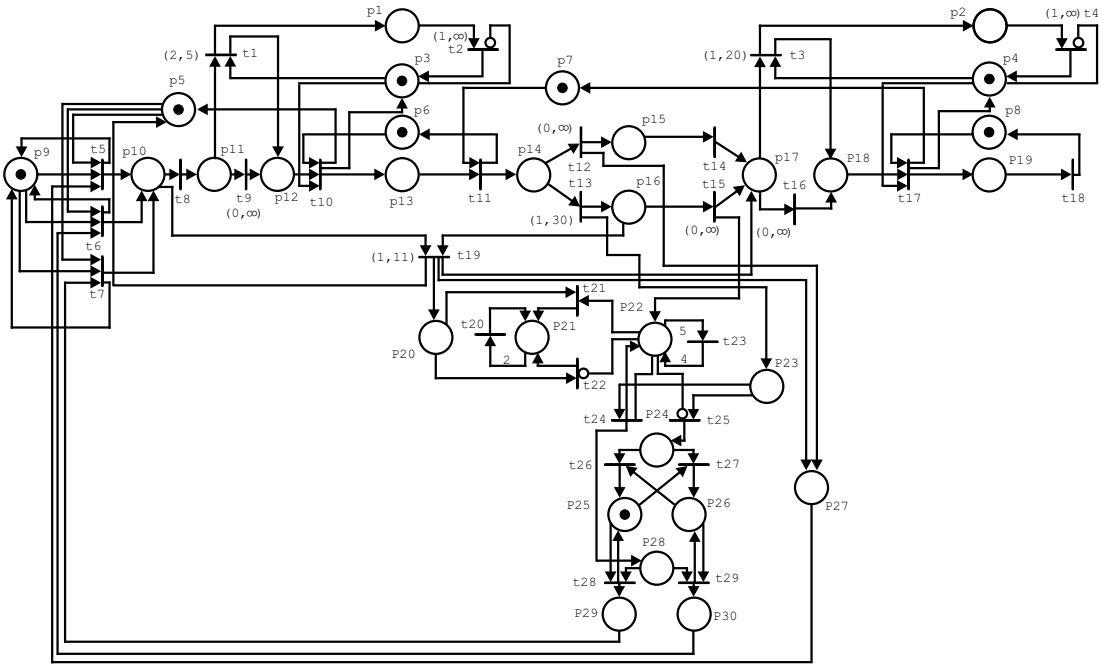

Figura 2. Modelagem do *Pipeline* utilizando rede de Petri Baseada em Tempo.

A dificuldade na geração desse modelo reside basicamente no tratamento de probabilidades em transições concorrentes. A solução é inserir restrições de tempo (intervalos de habilitação) de forma a proporcionar taxas de disparo equivalentes às probabilidades esperadas. Esse trabalho, entretanto, é empírico, dependendo, fundamentalmente, do passo de discretização de tempo do simulador e do número médio de transições habilitadas em cada passo de simulação. A única correlação existente é de que ao determinar-se

o valor ideal para uma dada probabilidade, pode-se aplicar, proporcionalmente, esse valor de intervalo aos demais pontos de concorrência probabilística.

3.2. *Pipeline* usando uma RdPEG

A Figura 3 apresenta o modelo de um *pipeline* utilizando apenas rede de Petri Estocástica Generalizada. Nessa rede tem-se os módulos de estágios (p5 até p19), gerenciamento de *cache* de instruções (p1, p3) e de dados (p2, p4), tratamento de desvios (p22 até p30) e o módulo de esvaziamento do *pipeline* (p20 e p21), que são os mesmos da RdPBT aqui apresentada.

Nesta rede a geração do modelo é mais simples e direta do que no caso da RdPBT. Isso é obtido pelo uso de probabilidades e das implicações das transições imediatas e temporizadas. Entretanto, ainda não implica numa diminuição no número de estados e transições do modelo. Essa redução apenas é possível com o tratamento diferenciado entre as marcas geradas por cada evento, como é feito no modelo híbrido (STERN).

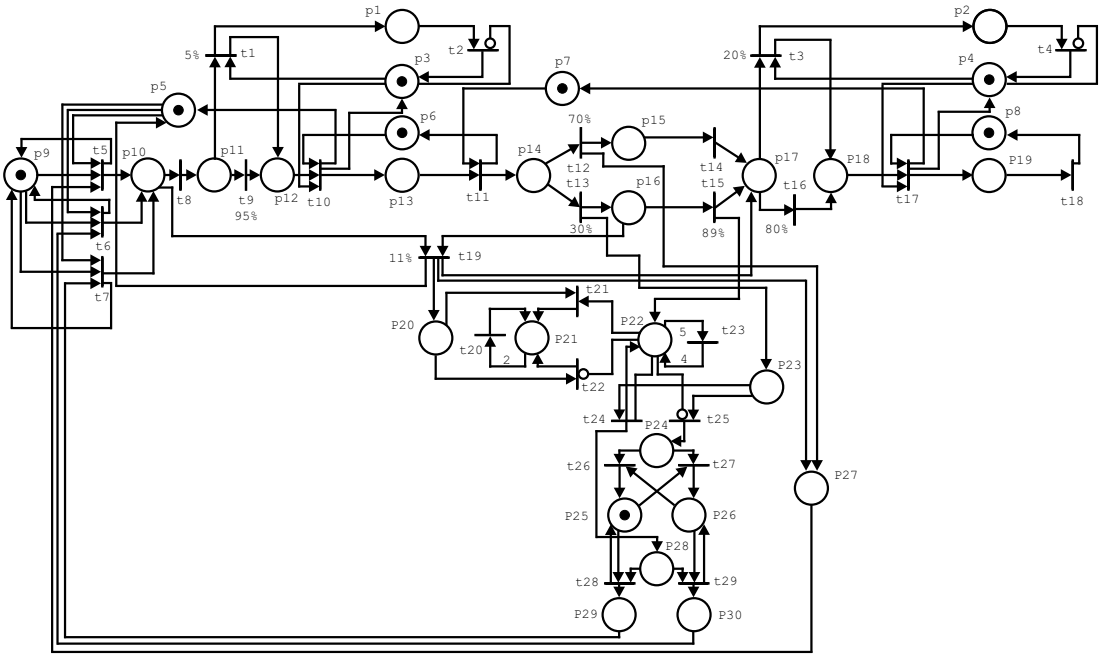

Figura 3. Modelagem do *Pipeline* utilizando rede de Petri Estocástica.

3.3. *Pipeline* usando rede STERN

Como indicado na seção 2.1.4, uma rede STERN é a composição das redes Baseada em Tempo, Estocástica Generalizada e Colorida. A razão para se utilizar essas técnicas em conjunto é a simplificação do processo de modelagem de um sistema, em especial daqueles que envolvam tempo e indeterminismo, utilizando para tanto as melhores características de cada uma dessas técnicas.

A Figura 4 apresenta o modelo STERN do *pipeline* já examinado. Ele é composto pelos módulos de estágios (p5 até p14), gerenciamento de *cache* de instruções (p1, p3) e de dados (p2, p4), tratamento de desvios (p17 até p20) e o módulo de esvaziamento do *pipeline* (p15, p16).

Os problemas no tratamento de probabilidades e tempo são resolvidos através das técnicas correspondentes (RdPEG e RdPBT, respectivamente). Isso facilita o tratamento de situações como a concorrência entre transições (t12 e t13, por exemplo).

O diferencial nas redes STERN é a possibilidade do uso de cores na diferenciação de marcas geradas por eventos distintos. Isso simplifica bastante os módulos de estágios e de tratamento de desvios. No primeiro caso, as diferenças aparecem na manipulação das informações vindas de *cache*, pela não necessidade de diferenciar informações já trazidas de RAM através de um novo lugar, e na diferenciação das instruções de desvio das demais, que agora é feito apenas por cores distintas. Já a simplificação do tratamento de desvios é feita pelos lugares que controlavam a inversão de política de caminho (lugares p24, p25, p26, p28, p29 e p30, substituídos pelo lugar p19).

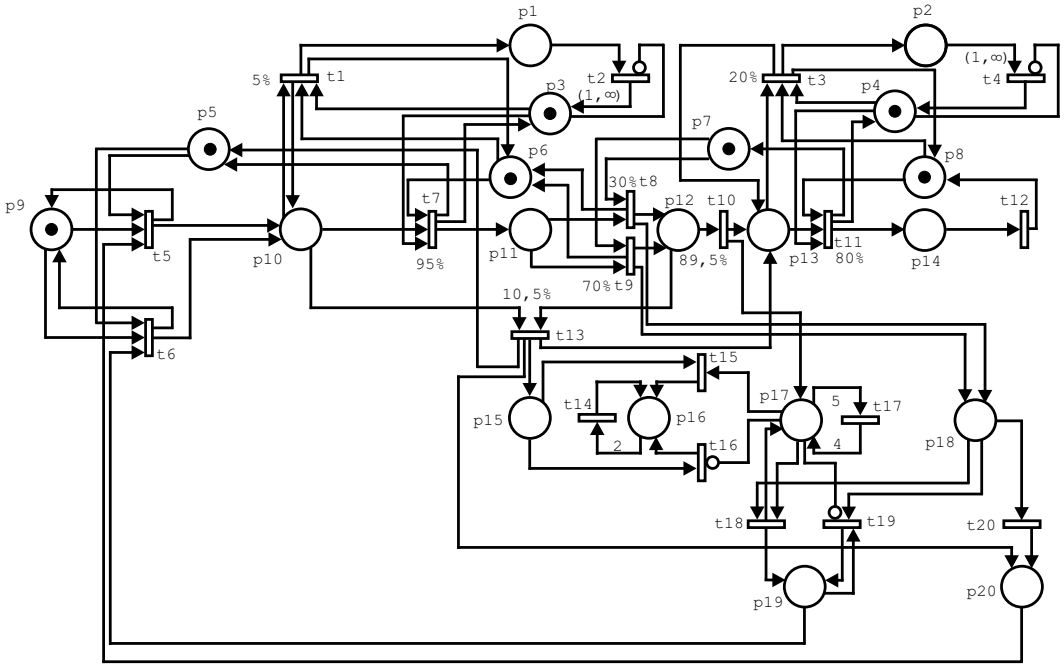

Figura 4. Modelagem do *Pipeline* utilizando rede STERN.

4. Análise quantitativa dos modelos

Apresentam-se agora os principais resultados obtidos com a simulação dos modelos descritos na seção anterior. Tais resultados foram obtidos com um simulador especialmente implementado para o tratamento das técnicas aqui apresentadas. Essa implementação foi necessária pois não existem simuladores padronizados para a rede STERN, o que impossibilitaria sua comparação em situação de igualdade com as demais técnicas.

As análises aqui realizadas se concentram em alguns parâmetros quantitativos, como tempo médio para fluxo no *pipeline*, freqüência de trocas de política de previsão de desvio, freqüência de faltas de *cache*, freqüência de esvaziamento de *pipeline* e intervalo entre disparos de um mesmo estágio. Uma análise subjetiva sobre as capacidades de modelagem de cada uma dessas redes já apareceu ao longo da seção anterior, não se repetindo aqui.

Para efeito de comparação, nas simulações executadas em cada modelo, o controle de parada se deu pela variação no número de transições disparadas, ficando entre 1000 e

50000 disparos. A seguir apresenta-se os resultados obtidos e a relevância dos parâmetros descritos no parágrafo anterior.

Tempo de percurso no *pipeline*

Esse parâmetro é importante para identificar se o *pipeline* tem um funcionamento regular ou não. O gráfico da Figura 5 mostra os resultados obtidos para os três modelos. Como se pode ver, em todos os casos o valor de tempo médio foi constante, independente do número de transições disparadas. Em particular, o modelo híbrido (STERN) apresentou menor valor médio, não muito distante dos modelos RdPEG e RdPBT. Isso indica um funcionamento mais eficiente do *pipeline* segundo esse modelo.

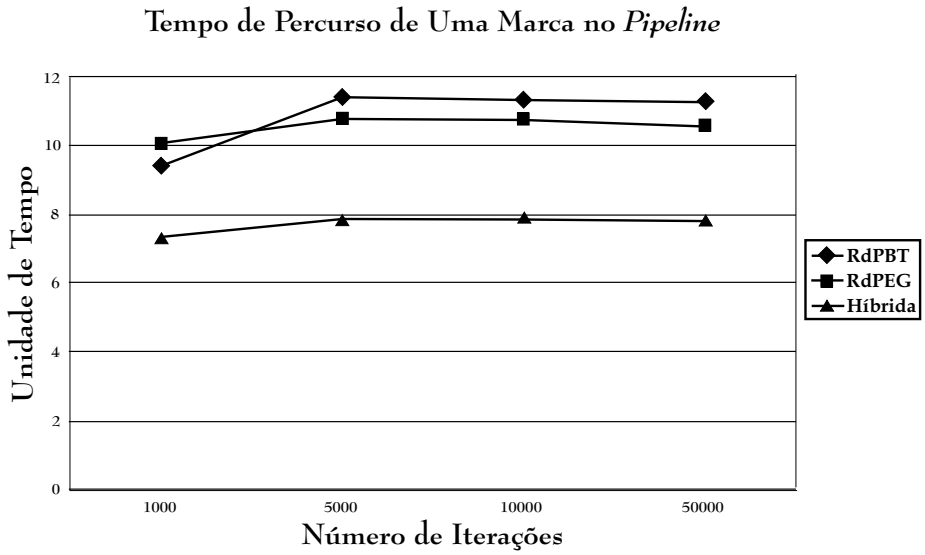

Figura 5. Tempo de percurso da marca no *pipeline*.

Tempo entre disparos de um mesmo estágio

O tempo entre disparos de um mesmo estágio indica se existe uma homogeneidade entre os estágios (desejável para evitar o surgimento de gargalos). Observe-se que esse tempo não pode ser imediatamente obtido a partir do valor de percurso, uma vez que esse último ainda é afetado pelos esvaziamentos e pelas buscas em memória RAM. O gráfico da Figura 6 mostra os resultados para os três modelos.

Com base no gráfico, em todos os casos o valor de tempo médio foi constante, independentemente do número de transições disparadas. Em particular, o modelo STERN continuou apresentando menor valor médio, não muito distante dos demais modelos. Com relação às diferenças entre tempo de percurso e de disparo, pode-se notar que para a rede STERN a relação entre tempo de percurso e de estágios ficou em 2,67 (o ideal seria 4,0 pois são quatro estágios), enquanto que para a RdPEG esse valor ficou em 2,44 e para a RdPBT ficou em 1,62. Portanto mais uma vez o melhor desempenho foi apresentado pela STERN.

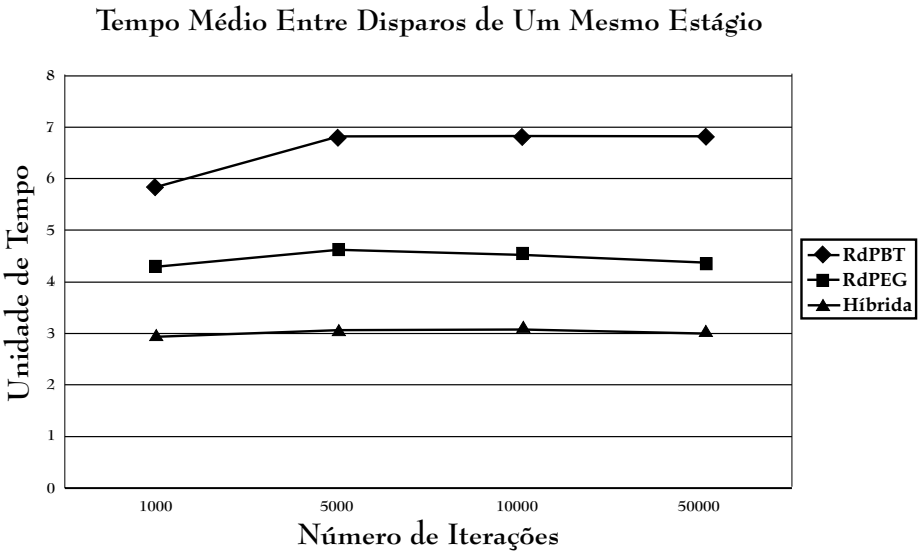

Figura 6. Tempo entre disparos de um mesmo estágio.

Freqüência de buscas por instruções

O valor esperado, e usado no modelo, para a freqüência de busca de instruções em memória (*cache misses*) é de 5%. Os resultados obtidos (Figura 7) reafirmam a estabilidade dos modelos, em particular para o modelo híbrido, que continuou apresentando o valor médio mais próximo do esperado.

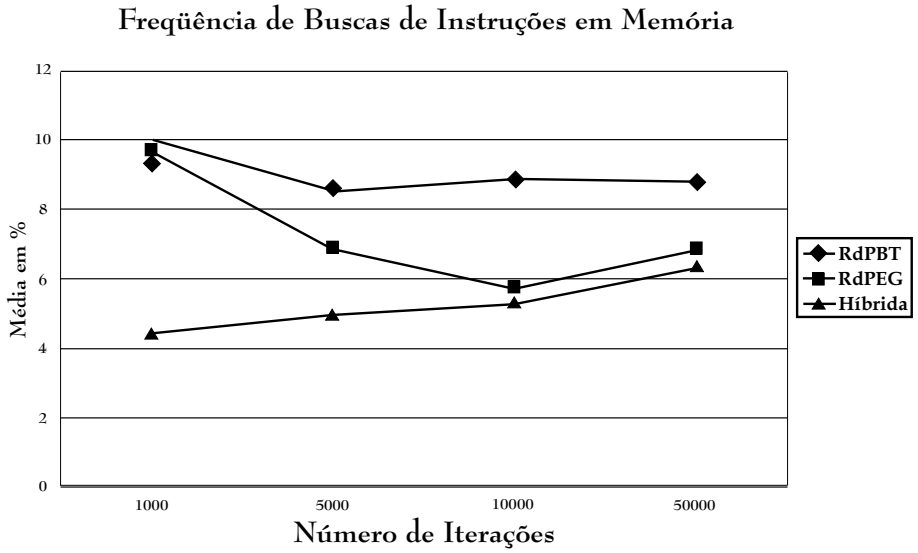

Figura 7. Freqüência de buscas de instruções em memória.

Freqüência de buscas por dados em memória

O gráfico da Figura 8 mostra a diferença entre os três modelos simulados quanto à ocorrência de faltas de dados em *cache*. Também aqui se percebe que as três redes apresentam resultados semelhantes e próximos ao esperado, em torno de 20%. Em particular a rede híbrida (STERN) apresenta resultados melhores do que as demais.

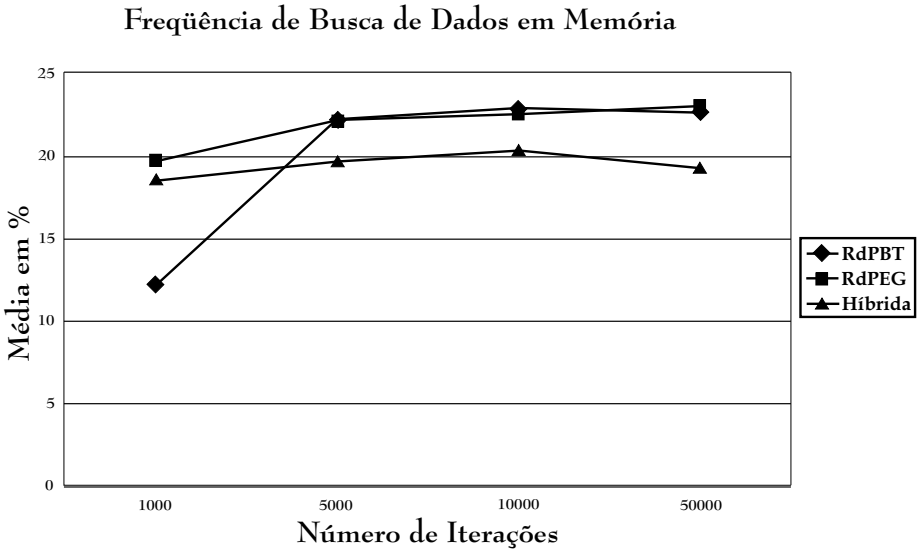

Figura 8. Freqüência de buscas de dados em memória.

Freqüência de esvaziamento do *pipeline*

A taxa prevista de insucessos na previsão do caminho de desvio era de 11%, que deveria ser portanto igual à freqüência de esvaziamento obtida na simulação. O gráfico da Figura 9 mostra os resultados obtidos, percebendo-se mais uma vez que os resultados do modelo STERN são melhores que as demais. Os resultados para a RdPEG ficaram relativamente fora do esperado, apresentando raros esvaziamentos do *pipeline*.

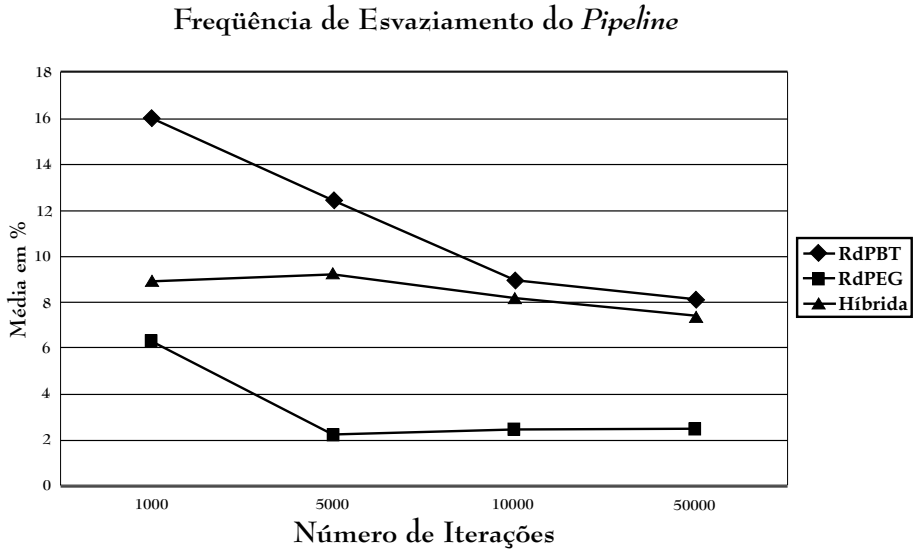

Figura 9. Freqüência de esvaziamento do *pipeline*.

Freqüência de trocas de abordagem de desvios

O gráfico da Figura 10 apresenta os resultados sobre a política de troca de abordagem no tratamento de desvios. Observe-se que não há muita variação entre os três modelos, com o gráfico apresentando sinais claros de estabilidade na porcentagem de

troca de abordagem para instruções de desvio com o aumento no número de transições disparadas. Outra observação é de que o simulador se comporta de modo conservativo, ou seja, considera-se que uma vez adotada uma política teremos a mesma mantida por muitas iterações.

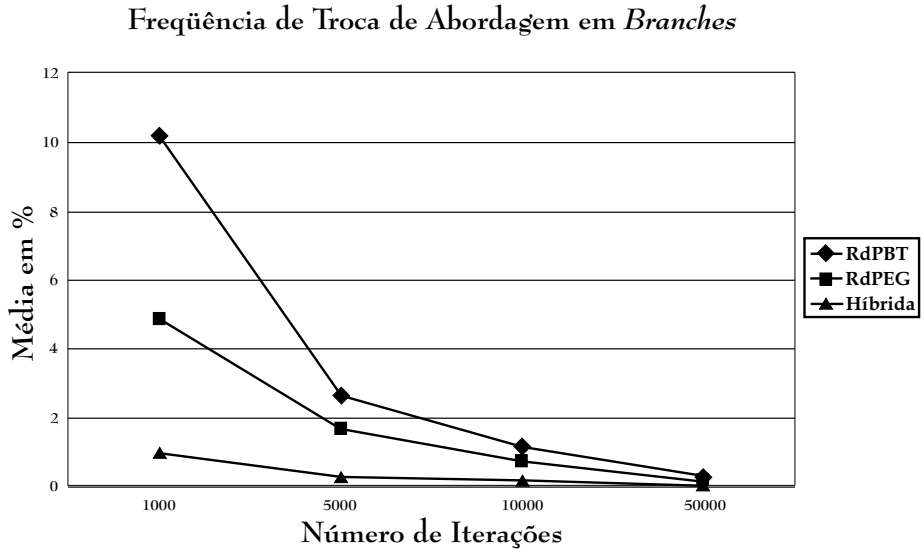

Figura 10. Freqüência de trocas de abordagem de *branch*.

5. Conclusões

Com base nos resultados apresentados na seção anterior, pode-se dizer que a rede que apresentou resultados mais próximos daqueles esperados para o modelo geral de *pipeline* proposto é a rede STERN (modelo híbrido). Os resultados obtidos com ela foram mais estáveis com a variação no número de iterações da simulação e, nos casos em que existiam valores pré-definidos de comportamento, foram também mais precisos. Já os modelos de RdPEG e RdPBT tiveram resultados razoáveis e aproximadamente equivalentes, tanto do ponto de vista de estabilidade quanto de precisão.

Do ponto de vista de simplicidade na geração do modelo, também foi a rede STERN que apresentou melhor comportamento, ao permitir tanto o tratamento de tempo e probabilidades, quanto o de estruturas de dados. Observe-se que nesse quesito o pior modelo foi exatamente o da técnica mais simples, que tratava apenas tempo (RdPBT).

Assim, ficou evidente a maior capacidade em produzir modelos elegantes (menos lugares e transições) das redes coloridas (aplicadas na técnica híbrida). O modelo criado com essa técnica não apenas é mais simples como também permite a combinação das melhores características dos demais modelos.

Uma conclusão final desse trabalho é de que a combinação de características intrínsecas de modelos mais simples de redes de Petri permite construir modelos que simulam mais precisamente os diversos sistemas. Quando combinadas com as características de redes coloridas, ganha-se também uma maior facilidade na geração do modelo, como é o caso das redes STERN. O desenvolvimento de tais modelos é uma continuidade possível a esse trabalho.

Referências

Buy, U. A. and Sloan, R. H. (1994). Analysis of real-time programs with simple time petri nets. In *International Symposium on Software Testing and Analysis*, pages 228–239.

Cardoso, J. and Valette, R. (1997). *Redes de Petri*. Editora da UFSC.

Evers, M. and Yeh, T. (2001). Understanding branches and designing branch predictors for high-performance microprocessors. *Proceedings of IEEE*, 89(11):1610–1620.

Ghezzi, C., Mandrioli, D., Morasca, S., and Pezze, M. (1991). A unified high-level petri net formalism for time-critical systems. *IEEE Trans. on Soft. Engineering*, 17(2):160–172.

Granda, M., Drake, J., and Gregorio, J. (1992). Performance evaluation of parallel systems by using unbounded generalized stochastic petri nets. *IEEE Trans. on Soft. Engineering*, 18(1):55–71.

Hennessy, J. and Patterson, D. (2002). *Computer Architecture: A Quantitative Approach, 3rd Edition*. Morgan Kaufmann Publishers.

Jensen, K. (1996). *Coloured Petri Nets: basic concepts, analysis methods and practical use*. Springer-Verlag, 2nd edition.

M. Nielsen, D. S. e. (2001). Application and theory of petri nets 2000. In *21st International Conference, ICATPN 2000*, LNCS 1825.

Merlin, P. and Farber, D. (1976). Recoverability of communications protocols: Implications of a theoretical study. *IEEE Trans. on Communications*, 24(9):1036–1043.

Molloy, M. (1982). Performance analysis using stochastic petri nets. *IEEE Trans. on Computers*, 31:913–917.

Smith, J. (1981). A study of branch prediction strategies. In *Proceedings of the 8th Annual Intl. Symposium on Computer Architecture*, pages 135–148.

Wang, J. (1998). *Timed Petri Nets: Theory and Applications*. Kluwer Academic Publishers.