Modelagem e Análise de Desempenho de uma Arquitetura de Fluxo de Dados

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Sérgio Roberto Pereira da Silva e aprovada pela Comissão Julgadora.

Campinas, 30 de Maio de 1991.

Prof. Dr. Arthur João Catto.
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisíto parcial para obtenção do Título de Mestre em Ciência da Computação.

Modelagem e Análise de Desempenho de uma Arquitetura de Fluxo de Dados

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da Unicamp, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Autor: Sérgio Roberto Pereira da Silva

Orientador: Prof. Dr. Arthur João Catto

Departamento de Ciência da Computação

IMECC - UNICAMP

Fevereiro de 1991

Lute sempre para conquistar

seus mais altos ideais

sem desanimar

no primeiro tombo.

Abstract

This work presents the development and analysis of a model for the Manchester data flow architecture using the General Net Theory. The model's development phases are detailed, demonstrating the applicability of such theory to the modelling of systems with concurrent tasks.

During the model's validation phase, an analytical deterministic method based on Petri nets, the Q-model, was used. The same method was used during the model's analysis phase, aiming at identifying performance bottlenecks in the architecture and at evaluating alternatives in order to correct such problems.

The work, besides developing a model for the Manchester data flow architecture, demonstrates the potential of this kind of tool for identifying problems and evaluating design alternatives for this class of machines.

Sumário

Este trabalho apresenta o desenvolvimento e a análise de um modelo para a arquitetura de fluxo de dados de Manchester empregando a Teoria Geral de Redes. As fases de desenvolvimento do modelo encontram-se detalhadas demonstrando a aplicabilidade desta teoria à modelagem de sistemas com tarefas concorrentes.

Na fase de validação do modelo foi utilizado para sua análise um método analítico determinístico baseado em redes de Petri, o modelo-Q. Este mesmo método foi empregado na fase de análise do modelo, visando a identificação de pontos de estrangulamento do desempenho da arquitetura, e na avaliação de algumas alternativas, visando a remoção destes pontos.

O trabalho, além de desenvolver um modelo para a arquitetura de fluxo de dados de Manchester, demonstra a potencialidade deste tipo de ferramenta para identificação de problemas e avaliação de alternativas de projeto para esta classe de máquinas.

Agradecimentos

Meus sinceros agradecimentos ao meu orientador, Prof. Dr. Atrhur J. Catto, pelo valiosos conselhos e, principalmente, pelo irrestrito apoio e encorajamento durante o desenrolar deste trabalho.

Um agradecimento especial ao Prof. Dr. Maurizzio Tazza pelo auxílio prestado sobre a Teoria Geral de Redes. Também, a Paulo Fernandes pelas interessantes discussões sobre modelagem de sistemas utilizando as redes de Petri e, ao Prof. Dr. Phillipe Navaux, pela atenção dispensada e a possibilidade de uso do *software* SQ1, desenvolvido por seu grupo de pesquisa na UFRGS.

Aos amigos, Fernando e Cecília, do grupo de Fluxo de Dados da Unicamp, e ao Prof. Dr. Antonio Carlos Ruggiero, pela profícua troca de idéias e apoio no decorrer deste projeto. E a todos os colegas e professores da pós-graduação da Unicamp, assim como as demais pessoas, que compartilharam estes agradáveis e difíceis anos.

Índice

Lista de Figuras

1 Introdução

Para atingir o nível de desempenho exigido na atualidade pelos usuários de supercomputadores empreendem-se muitos esforços tanto técnicos quanto econômicos. A simultaniedade e a concorrência na realização de tarefas são as chaves para estas máquinas de grande potência. Visto que os componentes de *hardware* somente podem ter sua velocidade aumentada até um limite físico, a única forma de aumentar a velocidade final além deste limite será explorando o paralelismo existente nos problemas [GAJ82].

Idéias já sedimentadas, como o emprego de *pipelines*, e outras ainda incipientes, como o multiprocessamento, vêm sendo utilizadas de várias formas e, apesar de ter-se obtido sucesso para alguns grupos específicos de aplicações, ainda não se chegou a uma solução definitiva. Dentre os vários problemas encontrados dois estão no cerne da questão.

O primeiro está relacionado com a representação dessas aplicações, ou seja, com a descrição dos algoritmos que as implementam. A questão está na necessidade de se adequar o algoritmo à arquitetura da máquina que o irá executar, causando a necessidade de partição do mesmo e o tratamento das tarefas de sincronização entre as partes pelo próprio programador.

Uma das causas deste problema é atribuída à semântica seqüencial que as linguagens de programação mais comuns apresentam. Como solução adotou-se a criação de linguagens que incorporam mecanismos de controle que têm relação quase direta com a estrutura da máquina para a qual o código será gerado. Esta solução somente mascara o problema real pois torna o algoritmo dependente da arquitetura da máquina que o irá executar, sendo necessária sua total reprogramação para a execução em outra arquitetura.

O segundo problema é devido ao fato de o modelo de computação atualmente empregado basear-se em um processo estritamente seqüencial. Este fato impossibilita que tanto as máquinas *pipeline* quanto os multiprocessadores utilizem totalmente o paralelismo provido pelo *hardware*. Nos processadores com *pipeline*, as dependências de dados entre instruções sucessivas e a relativamente freqüente transferência de controle nos programas convencionais reduzem em muito, na prática, o desempenho final [FLY72]. Nos multiprocessadores, além da partição dos programas em um número razoável de módulos, o que já irá causar uma sobrecarga de comunicação, tem-se também os problemas de sincronização causados pelas interdependências entre os espaços de endereçamento dos diferentes módulos o que irá reduzir a concorrência que poderia ser obtida [SUL77].

Uma solução parcial para estes problemas está sendo procurada através da mudança do modelo de computação empregado no desenvolvimento de novas arquiteturas. Desta busca

surgiram vários modelos alternativos dentre os quais dois se destacam: os modelos de computação dirigidos pela demanda de resultados e pela disponibilidade de dados. Na classe de modelos dirigidos por demanda, a necessidade de resultados dispara as operações que os produzem. Na classe de modelos dirigidos por disponibilidade, a presença de operandos dispara as operações que os utilizam [TRE82].

Na classe de modelos dirigidos pela disponibilidade de dados, o modelo de fluxo de dados já é suportado por algumas implementações ([HER87], [SRI86], [VEN86]), e empregado na Máquina de Fluxo de Dados de Manchester, cuja modelagem será um dos alvos deste trabalho. O modelo de fluxo de dados constitui-se em um poderoso formalismo para a representação de tarefas paralelas [KAV86]. Os programas neste modelo são melhor entendidos como grafos orientados, onde os vértices representam as instruções e as arestas as dependências de dados existentes entre essas instruções. Como parte das vantagens apresentadas por este modelo está a desnecessidade de explicitar-se a sincronização entre tarefas e a partição do algoritmo, que ficarão por conta do *hardware*.

Apesar de esses modelos apresentarem algumas soluções para os problemas eles são muito novos para mostrarem implementações com eficiência a nível das máquinas seqüenciais atuais. Além disto, algumas questões específicas destas classes de modelos necessitam ser resolvidas.

Historicamente, a avaliação de soluções alternativas para os problemas encontrados nestas novas arquiteturas tem sido buscada com o emprego de simuladores ou com a construção de protótipos. Porém, estes métodos são muito caros, tanto na fase de desenvolvimento, quando se necessita de muito tempo e detalhes sobre o sistema, quanto na fase de aplicação, onde apresentam um tempo de resposta muito elevado, provocando um alto custo de processamento.

Por outro lado, modelos analíticos apresentam menor custo de desenvolvimento, pois não exigem em primeira instância um detalhamento excessivo do sistema, nem precisam de muito processamento na análise, o que resulta em um tempo curto de resposta. Estes fatores influenciaram a escolha do estudo de modelos analíticos como um segundo alvo deste trabalho.

1.1 Modelagem de Sistemas Computacionais

Por modelo entende-se uma abstração do conhecimento que se detém sobre uma situação real. Um modelo pode ser pictórico, matemático ou simbólico, mas tem sempre como objetivo representar um ponto de vista das relações entre as entidades que compõem o sistema estudado e o meio que as envolve.

O processo de construção de um modelo apresenta duas abordagens extremas [GIL84]: a abordagem conceitual, baseada no conhecimento teórico de que se dispõe sobre o sistema; e a abordagem empírica, baseada somente nos dados obtidos sobre o sistema, normalmente na

ausência de conhecimento teórico preciso. A maioria dos modelos enquadra-se numa abordagem mista, que emprega técnicas de ambas.

Pode-se distinguir nestas abordagens entre modelos determinísticos e estocásticos. Nos primeiros, as entidades envolvidas guardam entre si relações conhecidas, que determinam valores exatos para os resultados de uma análise. Nos últimos, pelo menos parte das variáveis é de natureza aleatória, obtendo-se somente valores médios para os resultados de uma análise. Na modelagem de sistemas computacionais normalmente utilizam-se modelos estocásticos.

O processo de modelagem pode ser dividido em quatro fases [GIL84]:

a) **Fase de Identificação**, que corresponde à escolha da técnica apropriada para representar o sistema estudado, à seleção das variáveis e parâmetros relevantes e à definição da estrutura do modelo. Esta é uma fase de primordial importância, pois dela depende a definição do método de trabalho a ser empregado nas demais etapas.

b) **Fase de Adaptação**, que corresponde ao ajuste do modelo geral ao sistema específico. Para isso fazem-se suposições e simplificações quanto à estrutura e às relações representadas, e estimam-se os valores numéricos para os parâmetros do modelo, utilizando-se técnicas estatísticas ou especificações do sistema.

c) **Fase de Validação**, que consiste na verificação da adequação do modelo ao sistema. A rigor esta fase distribui-se durante todo o processo de modelagem, confrontando-se os resultados obtidos contra os dados reais. Em conseqüência, introduzem-se alterações e ajustes no modelo.

d) **Fase de Aplicação à Análise**, que consiste na avaliação do sistema visando um conhecimento mais detalhado de seu comportamento ou a previsão de seu comportamento frente a novas situações. Esta fase realiza essa tarefa, complementando as anteriores que são construtivas. De fato, o processo todo é iterativo, sendo necessário um contato íntimo entre todas as fases para não se construir um modelo impreciso ou inconsistente.

1.2 Técnicas de Construção de Modelos

Várias técnicas têm sido utilizadas na modelagem de sistemas computacionais. Esse amplo espectro engloba desde regras de bom senso, provenientes da experiência acumulada no manuseio de um sistema, até a utilização de elaborados experimentos de *benchmarking*.

Essas duas técnicas estão em extremos opostos quanto ao nível de complexidade e custo do esforço de desenvolvimento requerido, ficando os *benchmarks* no nível mais elevado. Por outro lado, ambas não se mostram adequadas à previsão do desempenho futuro de um sistema.

Nos níveis intermediários encontram-se técnicas mais úteis, que vêm tendo aceitação crescente nos últimos anos.

O emprego de técnicas empíricas, baseadas em ajustes estatísticos das variáveis e parâmetros do sistema, leva a melhores resultados de análise, ampliando a faixa de uso do modelo desenvolvido e possibilitando a previsão de resultados futuros. No entanto, estas técnicas exigem esforços de desenvolvimento relativamente grandes para a obtenção de resultados.

Por sua vez, as técnicas analíticas têm recebido maior atenção no meio de computação nos últimos anos. Estas técnicas têm um rigoroso embasamento teórico e apresentam níveis de complexidade e custo de desenvolvimento razoáveis ([ALL80], [PET77]). Dentre as técnicas analíticas mais conhecidas estão as **redes de filas** e as **redes de Petri**.

1.2.1 Redes de Filas

As redes de filas são uma extensão da teoria de filas, parte da teoria da probabilidade que estuda os fenômenos de formação de filas [KLE75]. Este fenômeno é comum nos sistemas computacionais, ocorrendo sempre que se requer o atendimento de uma tarefa por recursos do sistema já utilizados ([TRI80], [LIP77]).

A estrutura básica do modelo de redes de filas é composta de:

- usuários, que entram e saem do sistema ou de seus elementos;
- centros de serviço, que representam os recursos, as partes ativas que provocam alterações no estado do sistema e;
- filas, as partes passivas, ou listas de espera para atendimento nos centros de serviço.

O agrupamento e a interligação de centros de serviço e filas dão origem às redes de filas, que podem ser:

- abertas, quando o número de usuários é ilimitado;
- fechadas, quando o número de usuários é limitado ou;
- mistas, quando há ciclos fechados e abertos no mesmo sistema.

A teoria suporta também extensões, como o emprego de classes diferentes de usuários [LAZ84].

O desenvolvimento de modelos de redes de filas exige a especificação de itens como:

- o tipo da rede formada;
- o tipo de distribuição dos intervalos de chegada de usuários ao sistema;
- o tipo de distribuição dos tempos de serviço requeridos pelos usuários;

- o tamanho das filas utilizadas;
- o número de centros de serviço de cada tipo;
- a disciplina para atendimento dos usuários nas filas e;
- outros dados ou resultados de equações derivadas [ALL80].

Essas redes, apesar de terem sido alvo de muitos estudos e disporem de uma teoria bem fundamentada, não têm uma metodologia de desenvolvimento de fácil aplicação. Além disso, não apresentam muitos meios para a modelagem de sistemas onde há disputa por recursos ou com alto grau de concorrência: a simples introdução de uns poucos elementos do sistema, atuando simultaneamente sobre um ou mais recursos já é suficiente para tornar os modelos muito complexos de serem analisados.

Os modelos para a análise de desempenho através de redes de filas podem ser:

- estocásticos ([KOB81], [KLE76]), correspondendo à abordagem clássica ou;
- determinísticos [DEN78], correspondendo a um enfoque com fundamentação teórica na pesquisa operacional.

O processo de análise dessas redes pode ser exato [LAV83] ou aproximado ([SAU80], [CHA78]), dependendo da complexidade do sistema modelado.

1.2.2 Redes de Petri

As redes de Petri são técnicas gráficas, com um formalismo matemático associado, utilizadas na modelagem de sistemas de eventos discretos, sendo por isso muito empregadas nos meios computacionais para análise de questões referentes tanto a *hardware* quanto a *software* ([PET81], [AGE79]).

A estrutura das redes de Petri é composta de:

- locais, que representam condições existentes no sistema;
- transições, que representam eventos, responsáveis pela alteração de estado do sistema;
- marcas, que representam objetos ou recursos do sistema;
- conjuntos de entrada e saída dos locais e transições e;
- um vetor de marcas que indica o estado atual do sistema.

O desenvolvimento de modelos segundo essa técnica exige a especificação da estrutura de interligação entre os locais e as transições e do vetor inicial de marcas. Podem ser necessários, também, a especificação dos tempos associados ao disparo das transições e outros itens normalmente associados a interpretações especiais da rede, por exemplo, para análise de desempenho.

A interpretação básica das redes de Petri sofreu várias alterações para representar outros tipos de sistemas. Entre estas alterações pode-se citar o suporte para várias classes de objetos ou recursos ([HEU89], [REI85A], [GEN81], [PET80]) e a associação de tempo às transições ([HOL87], [MAR84]), ou locais [TAZ88], o que permite a análise de desempenho.

Apesar de esta técnica ser relativamente nova, muitos estudos têm sido desenvolvidos na área, que dispõe de uma forte fundamentação teórica. Este grande interesse deve-se à relativa facilidade de sua aplicação a uma variedade muito grande de sistemas. Outro fato a se destacar, é a facilidade da representação de concorrência entre eventos e de posse simultânea de recursos, que não é compartilhada pelas redes de filas.

Os fatos acima, aliados ao interesse em se dispor de uma ferramenta de fácil aprendizagem e uso para a modelagem e análise de alternativas de projeto para arquiteturas paralelas, levaram à opção pelo uso desta técnica no desenvolvimento de modelos e na análise de desempenho realizados no âmbito deste projeto.

1.3 Análise de Modelos

A construção de modelos está vinculada à análise de questões que validem suposições feitas sobre o sistema, ou façam previsão do seu comportamento futuro, frente a possíveis modificações das condições existentes. Essa análise pode ser qualitativa ou quantitativa.

A análise qualitativa normalmente desenvolve-se durante as três primeiras fases da modelagem pois está ligada à possibilidade de existirem representações para as interligações entre os elementos do sistema, as suposições feitas e a validação do modelo. Já a análise quantitativa normalmente recai no emprego de técnicas matemáticas, estatísticas ou de pesquisa operacional, sendo realizada posteriormente ao processo de especificação do modelo ou na fase final de validação.

Um fator importante a ser levado em conta na análise quantitativa é o grau de precisão dos resultados obtidos. Este item é afetado por vários fatores, entre os quais:

- a sensibilidade da técnica quanto às suposições adotadas na construção do modelo;
- os erros causados pelo processo de estimativa dos parâmetros devido a desvios estatísticos ou de especificação e;
- os erros adicionados pelas ferramentas empregadas no processo de cálculo dos resultados como, por exemplo, computadores digitais.

A análise quantitativa admite os seguintes métodos:

a) **Analíticos**, que fazem uso de técnicas algébricas para a solução de sistemas de equações que representam o modelo. Este método normalmente apresenta solução exata, porém

sua aplicação é muito restrita devido à crescente complexidade dos sistemas. Seus erros só dependem de erros de modelagem.

b) **Numéricos**, utilizados quando os modelos matemáticos criados são de solução algébrica difícil ou impossível; ou no caso de utilizarem-se técnicas estatísticas de modelagem. Neste caso empregam-se métodos computacionais para estimar resultados. Apesar de ser mais amplo que o anterior, este método tem sua precisão afetada pelos erros de truncamento e arredondamento causados pelo uso de computadores digitais, além dos erros inerentes à modelagem.

c) **De simulação**, empregados nos casos em que não é possível obter um modelo matemático adequado ao sistema, pela falta de teoria desenvolvida ou pela complexidade do sistema em si. Neste método a obtenção dos resultados está vinculada à aplicação de técnicas estatísticas. Este método também exibe problemas de representatividade e erros.

1.4 Organização do Trabalho

A arquitetura de fluxo de dados de Manchester foi uma das primeiras desta classe a ter um protótipo em funcionamento [GUR83]. Além disto, este grupo de pesquisa apresentou muitos trabalhos nesta área, demostrando soluções interessantes para os problemas encontrados durante o desenvolvimento do protótipo, e também para alternativas de implementação. Observa-se contudo que o projeto não dispunha de uma ferramenta de análise de modelos, impondo-se a necessidade de simular qualquer modificação considerada interessante, o que elevou o custo do projeto e desestimulou a busca de alternativas.

Os fatos acima demonstram existir um espaço nas técnicas de desenvolvimento que não foi explorado pelo grupo de Manchester. O desejo de evitar a repetição desse equívoco levou ao desenvolvimento deste trabalho sobre a arquitetura de Manchester, escolhida com base na existência de uma ligação histórica entre o grupo de pesquisa em fluxo de dados da Unicamp e o grupo de pesquisa de Manchester.

Os capítulos seguintes descrevem o trabalho de modelagem e análise de desempenho realizados. No capítulo 2 serão apresentados vários submodelos de computação que se enquadram no modelo abstrato de fluxo de dados. Será apresentada também uma descrição da arquitetura da Máquina de Fluxo de Dados de Manchester (**MFDM**[1]), implementada segundo o submodelo de fluxo de dados dinâmico .

---

[1] Ao final desta dissertação encontra-se um glossário de abreviações que abre-se para a direita facilitanto a sua consulta durante a leitura.

O capítulo 3 resume a Teoria Geral de Redes, empregada na modelagem e na análise de desempenho da arquitetura escolhida. Inclui-se uma descrição dos vários sistemas de interpretação mais comuns para essas redes, além da descrição de um método a ser empregado na realização de análise de desempenho de sistemas.

No capítulo 4 aplicar-se a Teoria Geral de Redes à construção dos modelos conceitual e de recursos da arquitetura em estudo. Descreve-se neste capítulo a construção dos modelos de forma a criar um conjunto implícito de regras a serem empregadas em outros casos de modelagem.

O capítulo 5 complementa o trabalho com a validação dos modelos desenvolvidos e a demonstração da aplicabilidade do modelo de recursos à análise de pontos de estrangulamento e alternativas de implementação para a arquitetura em estudo.

Por fim, o capítulo 6 apresenta um sumário do trabalho desenvolvido e uma comparação dos resultados obtidos com os de outro trabalho correlato disponível na literatura. Além disto, são propostas futuras utilizações para o modelo desenvolvido, como também possíveis desdobramentos do trabalho realizado.

2 Arquiteturas de Fluxo de Dados

A exploração de paralelismo nos algoritmos deparou-se, desde cedo, com problemas inerentes ao modelo de computação adotado nas arquiteturas tradicionais. De um lado, devido ao controle seqüencial centralizado do processamento, de outro pela forma de armazenamento e manipulação dos objetos do programa, realizado através de uma memória global endereçável e atualizável.

A adição de mais processadores, com controle individual sobre o fluxo da computação, ocupando-se das linhas de processamento concorrentes existentes no algoritmo, é uma solução freqüentemente encontrada para esses problemas. Porém, isto aumenta o tráfego de informação entre a memória global e os processadores, degradando o desempenho do sistema e agravando as conseqüências do **gargalo de von Neumann**. A partição da memória como forma de evitar o agravamento do gargalo, depara-se com a tarefa não trivial de partição do algoritmo que expressa a solução do problema, tornando-o dependente da máquina a executá-lo.

Devido a este fatores, vários grupos pesquisaram, a partir do final dos anos 60, modelos de computação alternativos que pudessem expressar o paralelismo existente nos programas de forma clara e transparente ao programador ([ROD69], [ADA68], [KAR66]). Destas pesquisas originaram-se os modelos dirigidos pela disponibilidade de dados e pela demanda de resultados, ambos tendo como vantagens um alto grau de paralelismo exibido e assincronismo na execução das operações.

Nos modelos dirigidos pela demanda de resultados, a necessidade de um valor ativa a operação que o produz e que, por sua vez, pode ativar outras operações produtoras de argumentos necessários à sua execução. Como desvantagem tem-se o fato de que, devido à **avaliação tardia** desses argumentos, pode ocorrer sobrecarga dos processadores pelo grande número de tarefas adiadas a espera de dados [TRE82].

Nos modelos dirigidos pela disponibilidade de dados, as operações são ativadas pela presença dos dados necessários à sua execução. Esse modelo tem como desvantagem o fato de que, devido à **avaliação apressada** dos resultados, alguns valores produzidos podem não ser eventualmente utilizados pelo programa, acarretando a realização de trabalho desnecessário [BUZ88].

2.1 Modelos de Fluxo de dados

Dentre os modelos dirigidos pela disponibilidade de dados, o modelo abstrato de fluxo de dados sobressaiu-se e, já nos anos 70, vários projetos buscavam sua implementação ([GOS79],

[GUR78], [ARV77], [DEN75]). Mais recentemente, forneceu-se fundamentação teórica consistente ao modelo [KAV86] e proliferaram implementações e simulações, como as apresentadas nos trabalhos de [HER87], [VEN86], [SRI82], [TRE82], entre outros.

O modelo abstrato de fluxo de dados difere do modelo abstrato de fluxo de controle de von Neumann em vários pontos fundamentais [AGE82], tais como:

- inexistência de nomes para as células de memória que armazenam valores, o modelo trabalha somente com valores;
- inexistência de um contador de instrução, as instruções são habilitadas quando os valores de entrada requeridos ficarem disponíveis;
- a execução das instruções habilitadas não apresenta uma ordem pré-determinada, consumindo seus dados de entrada e gerando resultados sem apresentarem efeitos colaterais;
- as linguagens baseadas no modelo não apresentam outras restrições de seqüencialização que não as impostas pelas dependências de dados no algoritmo.

Dessa forma, o modelo abstrato apresenta comportamento funcional, vindo as linguagens aplicativas e de atribuição única a apresentarem-se como as melhores candidatas para a programação dos sistemas que o implementam [ACK82].

Um programa de fluxo de dados em linguagem de alto-nível pode ser diretamente traduzido para um grafo orientado, como na Figura 2.1, com os vértices representado funções e as arestas orientadas representando as dependências de dados entre essas funções. Os valores processados são representados por fichas, que fluem pelas arestas, entre os vértices [DAV82].

Nesse modelo, o estado da computação pode ser visto como o conjunto das fichas que

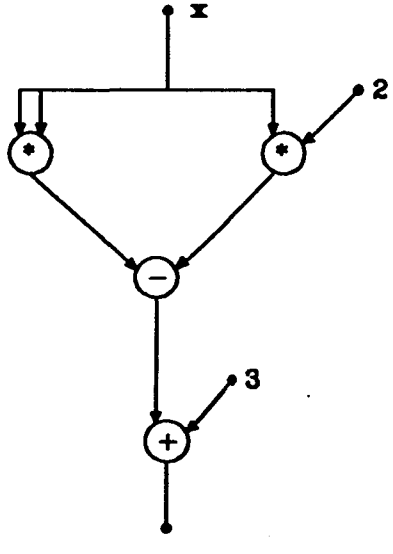

Figura 2.1: Grafo da função $x^2 - 2x + 3$.

estão presentes nas arestas do grafo a cada instante, e a computação em si, como uma seqüência desses estados, entre os quais fichas são colocadas e removidas das arestas pelo disparo de um ou mais vértices.

As regras que regem a remoção e colocação de fichas nas arestas e limitam o número de fichas que podem estar presentes em uma mesma aresta determinam os diferentes modelos de implementação do modelo abstrato de fluxo de dados.

2.1.1 Modelo Básico

A limitação de que, em toda a história de uma computação, não haja duas fichas com a mesma aresta destino caracteriza o modelo básico de fluxo de dados, que tem como regras de disparo de um vértice:

- um vértice do grafo está **habilitado** se, e somente se, existir uma ficha em cada uma de suas arestas de entrada;
- qualquer vértice habilitado pode **disparar** para gerar o próximo estado da computação;
- um vértice é disparado pela remoção das fichas que o habilitavam de suas arestas de entrada e colocação de fichas, correspondentes ao resultado da operação implementada, nas suas arestas de saída.

A programação de algoritmos requer construções capazes de expressar ações seqüenciais, paralelas, condicionais ou seletivas, repetitivas e abstrativas. A seqüencialização e paralelização de ações em grafos de fluxo de dados é direta e obedece as dependências dos dados, sendo necessários somente vértices de operações aritméticas, lógicas e de entrada e saída para a sua implementação.

Para se realizarem operações condicionais e seletivas, são necessários vértices que façam a seleção e fusão entre dois ou mais caminhos alternativos no grafo, ou seja, vértices seletores e coletores para as fichas que percorrem o grafo.

Já a implementação de operações abstrativas, representadas pela aplicação de funções, requer a existência de vértices que realizem operações complexas sobre suas arestas de entrada, normalmente implementadas através de subgrafos. O modelo básico realiza esta tarefa pela substituição estática do subgrafo no grafo principal, como mostra a Figura 2.2, de forma que é possível realizar-se a **avaliação não-estrita** de funções, caracterizadas pela possibilidade de se começar a obter resultados sem a presença de todo o conjunto de fichas de entrada [DEN85].

As limitações impostas por este modelo implicam na não reutilização de grafos. Assim, a iteração deve ser realizada de modo linear, muito oneroso, enquanto a recursão fica impossibilitada. Essas restrições tornam a utilização prática deste modelo totalmente inviável.

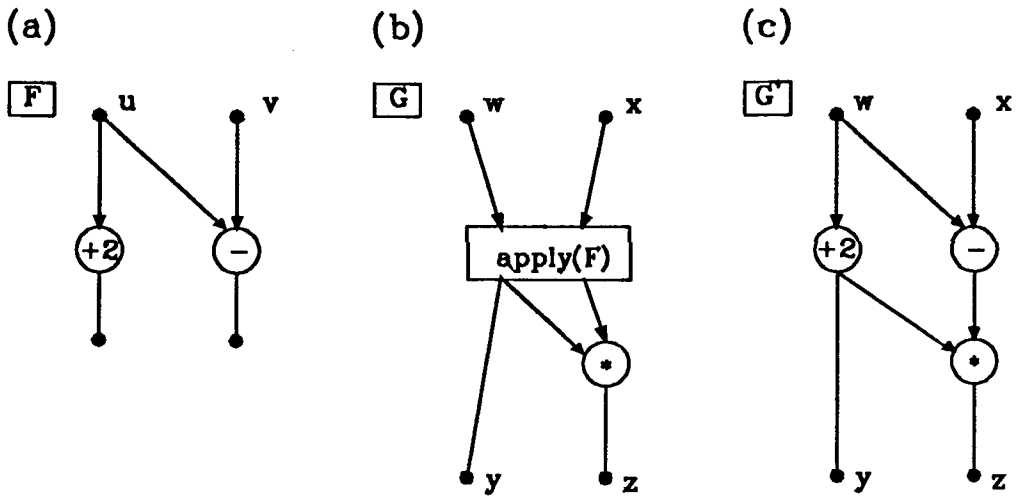

Figura 2.2: Demostração da substituição de grafo de uma função.

2.1.2 Modelo Estático

A possibilidade da utilização de uma aresta por mais de uma ficha durante a computação, porém limitada à presença de somente uma ficha por vez na aresta, caracteriza o modelo estático, que com isto minimiza os problemas identificados no modelo básico. Porém, para isso, a regra de habilitação de um vértice deve ser modificada para:

- um vértice estará habilitado se, e somente se, existirem fichas em todas as suas arestas de entrada e não houver fichas nas suas arestas de saída.

A imprevisibilidade da ordem de percurso dos subgrafos em uma seqüência de operações condicionais implica na necessidade de ordenar-se os resultados gerados. Esta ordenação é necessária para garantir que a seqüência de resultados corresponda à seqüência de dados. Para isso, pode-se transformar o vértice coletor em um vértice arbitrador, e adicionar um vértice do tipo FIFO antes dele, como visto na Figura 2.3. Este último armazenará a seqüência de fichas booleanas empregadas na seleção do subgrafo a ser percorrido e permitirá a ordenação dos resultados gerados [DEN85].

O vértice arbitrador tem regras de habilitação e disparo diferentes:

- Ele está habilitado se, e somente se: não existirem fichas nas suas arestas de saída; existir uma ficha booleana na sua aresta de controle e existir uma ficha na aresta de entrada selecionada pela ficha de controle; a outra entrada pode ou não conter ficha.
- Ele dispara removendo uma ficha de sua aresta de controle e da aresta de entrada selecionada, colocando esta última na sua aresta de saída.

A introdução deste vértice no modelo estático permite processar diferentes conjuntos de dados no mesmo grafo em *pipeline*, tornando assim eficiente a execução da iteração. Porém,

ainda não é possível implementar a recursão, devido a não haver distinção entre instâncias diferentes de um mesmo grafo.

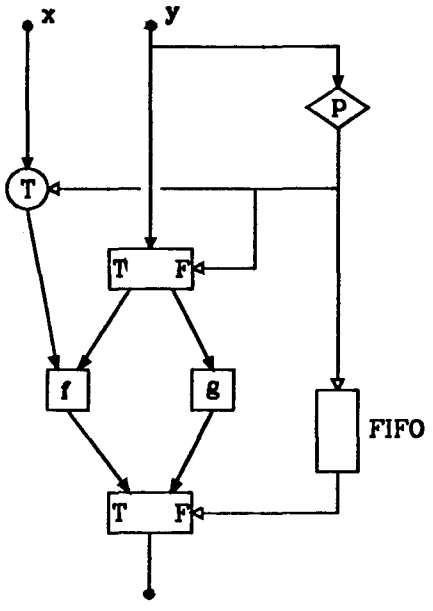

Figura 2.3: Grafo com nó FIFO.

2.1.3 Modelo Dinâmico

A eliminação da restrição ao número de fichas presentes numa aresta de um grafo, através de sua rotulação ou cópia, são os dois mecanismos empregados para a realização de recursão e determinam as duas formas do modelo de implementação dinâmico.

No mecanismo de cópia de grafo somente uma ficha estará presente em uma aresta por vez, podendo empregar-se desta forma as mesmas regras de habilitação e disparo do modelo básico. A forma de possibilitar a recursividade, mantendo-se esta restrição, encontra-se em realizar uma cópia dinâmica, sob demanda, do grafo do procedimento ativado. Desta forma, as restrições para os problemas práticos ficam reduzidas, pois ocorre uma redução no uso de memória pelo programa.

Nesse mecanismo fica impossibilitada a iteração como descrita no modelo estático, pois os grafos deverão ser orientados e acíclicos visto que não poderá haver mais de uma ficha por vez em uma aresta. Não há, neste caso, vértice arbitrador ou FIFO. Porém, é possível implementá-la na forma de recursão de cauda, fazendo-se uma cópia do subgrafo do bloco de comandos para cada re-entrada no mesmo [DEN85].

As fichas nos modelos básico e estático têm que carregar somente as seguintes informações:

- valor do dado;

- endereço do vértice de destino e;
- posição de entrada no vértice;

podendo ser representada da seguinte forma:

<valor,destino,posição>.

No modelo dinâmico com cópia de grafo será necessário inserir mais um campo na ficha, para identificar a ativação do grafo à qual a ficha pertence: este campo será denominado **nome de ativação** e, juntamente com o campo de destino, formará o **contexto** a que a ficha pertence.

<valor,<nome de ativação,destino>,posição> ou,

<valor,contexto,posição>.

A liberação do número de fichas presentes em uma aresta permite utilizar o mesmo grafo para realizar ciclos sucessivos de uma iteração, atender chamadas recursivas a um procedimento e implementar estruturas de dados. Porém, o contexto da ficha deverá ser ampliado para incluir um campo identificador do **nível de iteração** em que essa se encontra ao entrar em um subgrafo e outro de **índice** que corresponde à posição de uma ficha num vetor.

Ao conjunto de campos formado pelo nome de ativação, nível de iteração e índice, dá-se o nome de rótulo da ficha que, juntamente como o campo destino, compõe o novo contexto da ficha:

<valor,<<nome de ativação,nível de iteração,índice>,destino>,posição> ou,

<valor,<rótulo,destino>,posição> ou,

<valor,contexto,posição>.

Esta nova composição da ficha implica em que, para o emprego do mecanismo de rotulação de fichas, é necessário modificar as regras de habilitação e disparo dos vértices para:

- um vértice está habilitado se, e somente se, existirem fichas de mesmo rótulo em todas as suas arestas de entrada;
- qualquer vértice habilitado pode ser escolhido para disparo, gerando um novo estado da computação;
- disparar um vértice implica em remover fichas que o habilitam das suas arestas de entrada, colocando fichas nas suas arestas de saída com valor e contexto determinados pelo tipo do vértice.

Neste mecanismo também não há necessidade dos vértices arbitrador e FIFO, porém serão necessários vértices especiais para a manipulação dos campos dos rótulos, de modo a realizar corretamente a iteração, a chamada de procedimentos e a recursão [GOS82].

O mecanismo de rotulação de fichas é o mais genérico entre as implementações do modelo de fluxo de dados, embora também apresente problemas, tais como:

- o custo de carregar um rótulo, que não é pequeno, juntamente com o valor da ficha;
- maior complexidade para identificação de vértices ativados;
- maior complexidade para o tratamento de estruturas, o que é comum a todos os modelos de implementação, devido à funcionalidade apresentada pela sua semântica que impõe a necessidade de realizar-se uma cópia da estrutura para qualquer modificação feita a mesma.

A busca de soluções para esses problemas levou ao desenvolvimento de várias arquiteturas que implementam o modelo dinâmico de fluxo de dados por rotulação de fichas. A seção seguinte apresenta a descrição de uma destas arquiteturas, a dizer, a máquina de fluxo de dados de Manchester..

2.2 Arquitetura da MFDM

O modelo dinâmico de rotulação de fichas exige a realização de três operações básicas para computar um grafo:

- a transferência de fichas do início de uma aresta para o seu final;
- o agrupamento de fichas, de mesmo rótulo, que se encontram no final de arestas que apontam para um mesmo vértice, e o disparo do vértice quando todas suas arestas dispuserem de fichas;
- a execução da operação especificada pelo vértice, sobre os dados de entrada, produzindo fichas no início de suas arestas de saída.

Além dessas operações básicas, é necessário que a máquina possa receber o grafo a ser processado e realizar a saída de resultados para o meio exterior.

A **MFDM** satisfaz estes requisitos através de um sistema composto por cinco módulos ligados em *pipeline*, formando um anel, como mostra a Figura 2.4 [GUR80].

Os módulos que compõem o anel têm seu funcionamento interno implementado de forma síncrona, com relógios independentes. Porém, a comunicação intermódulos foi implementada de forma assíncrona e, juntamente com a adição de *buffers* de entrada e saída para cada módulo, permite uma melhor exploração do paralelismo da máquina.

No anel formado, a ***matching store unit*** (**MSU**) é responsável por reunir as fichas que se dirigem para um mesmo vértice e dispará-lo quando todas estiverem presentes, formando um pacote chamado **grupo de fichas**. Esse pacote dirige-se para a ***node store unit*** (**NSU**), onde

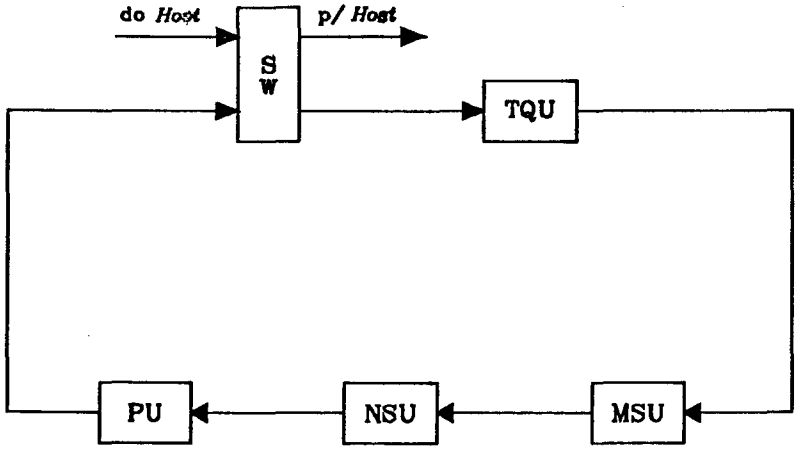

Figura 2.4: Esquema do *pipeline* da **MFDM**.

compõe com a instrução endereçada um **pacote executável**. Ao chegar à ***processing unit*** (**PU**), o pacote é executado e os resultados são gerados nas arestas de saída do vértice. Assim, o ciclo de agrupamento e execução se completa, restando à ***switch unit*** (**SW**) realizar a tarefa de levar a ficha do início da aresta de saída para o seu final, na **MSU**. A **SW** serve também como meio de comunicação com um computador hospedeiro (***Host***), empregado para realizar a entrada dos grafos de programas e dos dados e a saída dos resultados.

A ***token queue unit*** (**TQU**) foi adicionada ao anel para permitir a uniformização do fluxo de fichas, de modo a compensar eventuais irregularidades no grau de paralelismo exibido pelos programas. Esta unidade serve para evitar a formação de bolhas no *pipeline*, que podem resultar em queda no desempenho da máquina.

As fichas circulantes no anel são compostas de três campos: um de dados (37 bits), um de rótulo (36 bits) e um de destino (22 bits), além de um bit extra para diferenciar as mensagens especiais do sistema, totalizando 96 bits. Um grupo de fichas possui, ainda um campo de dados (37 bits) adicional totalizando 133 bits de largura. Um pacote executável possui um campo para a instrução (10 bits) e um segundo destino (22 bits) opcional, para a geração de uma cópia extra do resultado da operação, totalizando 166 bits de largura [GUR85A].

Os vértices dos grafos de programa tiveram o número máximo de suas arestas de entrada limitados a duas. Isto foi necessário devido à natureza associativa da **MSU**. O mesmo limite foi imposto ao número de arestas de saída, procurando simplificar a **NSU** e, também, devido à grande largura do barramento paralelo requerido pelo pacote executável formado. Isto não causa problemas à computação, pois sempre é possível simular vértices complexos com estes vértices restritos, embora provoque uma queda no desempenho de algumas instruções da máquina.

2.2.1 *Matching Store Unit*

A indisponibilidade de *chips* de memória associativa de grande capacidade levou a implementação desta unidade a ser realizada de forma pseudoassociativa, empregando-se técnicas de *hashing* [DAS82] e dividindo-a em duas subunidades: a de **cálculo de *hashing*** e a **memória de fichas** desemparelhadas, conforme mostra a Figura 2.5.

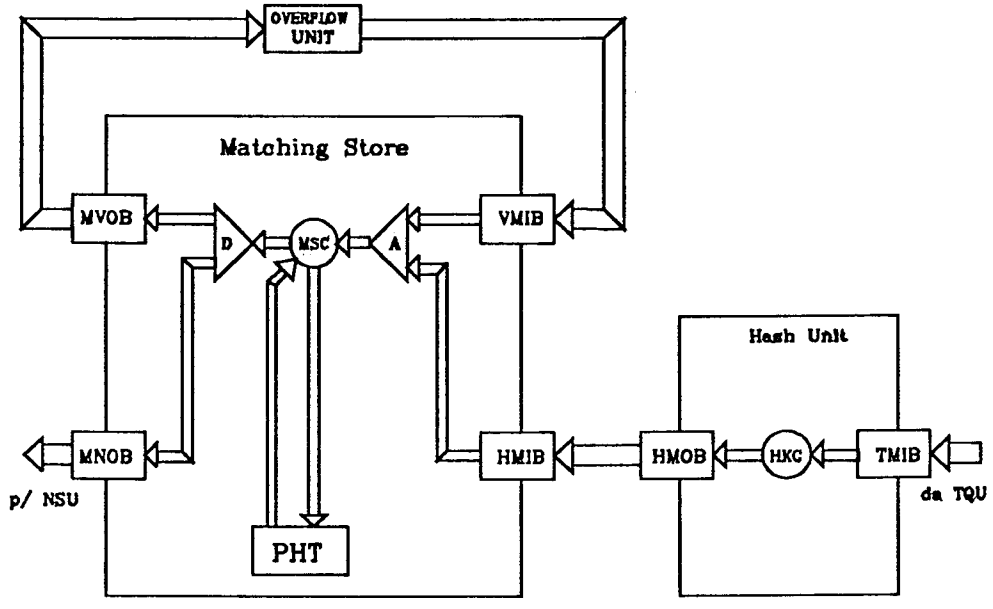

Figura 2.5: Esquema detalhado da **MSU**.

A subunidade de cálculo de *hashing* gera o endereço empregado na consulta aos bancos de memória. Esta subunidade é composta de um *buffer* de entrada, um de saída e um registro de cálculo do endereço de *hash*, empregando para tal os campos de rótulo e destino da ficha que dá entrada na **MSU**.

A subunidade de memória de fichas realiza o agrupamento das fichas que se dirigem para o mesmo vértice e efetiva o disparo deste quando suas fichas estão presentes. Ela é composta de um *buffer* de entrada e um de saída, assim como de uma memória de armazenamento de fichas à espera de suas parceiras, com 8 bancos paralelos de 2K fichas cada.

As fichas que não necessitam de agrupamento passam direto para o *buffer* de saída da subunidade de memória de fichas. As fichas que necessitam de agrupamento podem ter, basicamente, dois comportamentos, dependendo da presença ou não da ficha parceira na memória de fichas. Quando presente, ela é retirada do banco de memória e empacotada com a ficha de entrada, formando um grupo de fichas que é enviado ao *buffer* de saída. Na ausência da pareceira a ficha de entrada será depositada em um banco de memória livre no endereço de *hash* especificado, à espera de sua parceira.

Nesse último caso, pode ocorrer o fato de não haver lugar nos bancos de memória para mais uma ficha. Isto causará o deslocamento da ficha de entrada para uma subunidade de **tratamento de *overflow* de memória** e a marcação do endereço de memória para busca posterior. Esse fato não causa problemas à computação, devido à natureza assíncrona do protótipo, o que permite inclusive a realização de buscas simultâneas na subunidade de *overflow* e na memória de fichas. No entanto, o acesso à subunidade de *overflow* causa uma queda no desempenho da máquina [GUR80].

O agrupamento de fichas causa uma diminuição do fluxo de fichas após a **MSU**, provocando o aparecimento de bolhas no *pipeline*. Assim, a percentagem de fichas que necessitam de emparelhamento afeta de forma considerável o desempenho da máquina.

2.2.2 *Node Store Unit*

Esta unidade é responsável pela composição do pacote executável, combinando cada grupo de fichas recebido com uma cópia da instrução endereçada. A unidade foi originalmente projetada para ser composta de um *buffer* de entrada, um de saída e um intermediário; uma tabela de segmentos, um registro gerador de endereço e uma memória de instruções. Porém, na sua implementação final foi eliminado o *buffer* intermediário, e fundido o gerador de endereços com a pesquisa a tabela de segmentos, ficando a sua estrutura final conforme mostra a Figura 2.6.

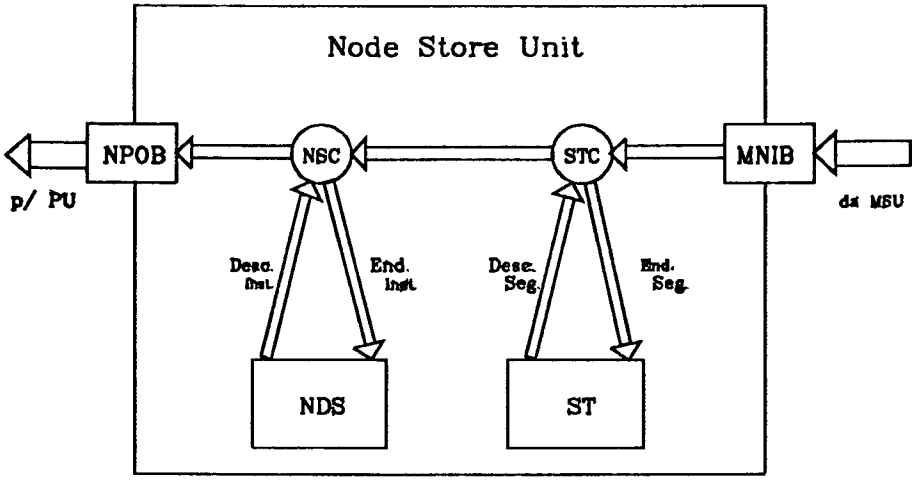

Figura 2.6: Esquema detalhado da **NSU**.

O acesso à tabela de segmentos é realizado empregando-se os 6 bits mais significativos do campo destino. O endereço final na memória de instruções é composto pelo endereço base do segmento, retirado da tabela, aposto ao valor dos 12 bits menos significativos do campo destino, que correspondem ao seu valor de *offset*.

2.2.3 *Processing Unit*

Nesta unidade é realizada a execução da instrução com a conseqüente produção de **fichas-resultado**. Para tal, ela dispõe de dois estágios: o primeiro executa algumas operações de alta velocidade que não podem ser realizadas de forma distribuída; o segundo é composto de um conjunto paralelo de ***function units*** (**FUs**) que executam a maior parte das instruções. Estas subunidades são microprogramadas de forma a facilitar eventuais mudanças do conjunto de instruções [DAS81].

O primeiro estágio da unidade é composto de: um *buffer* de entrada, um de saída e um pré-processador. O segundo estágio é composto de: um *buffer* distribuidor para a entrada do conjunto de FUs, um arbitrador para a saída das FUs, um para a saída da unidade e um conjunto de 1 a 20 FUs, conforme mostra a Figura 2.7.

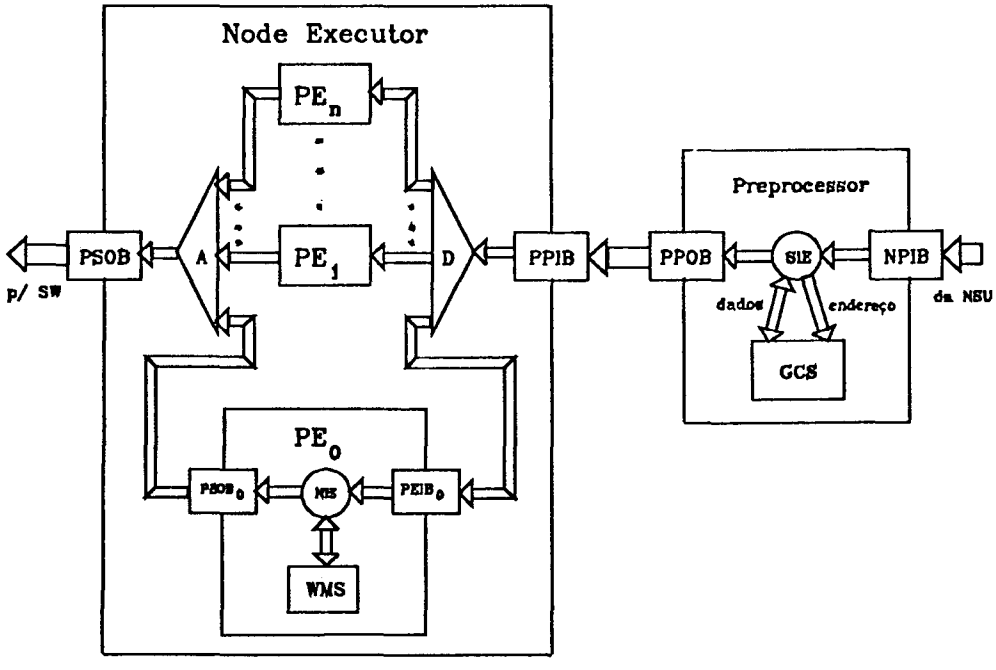

Figura 2.7: Esquema detalhado da **PU**.

O número de resultados gerados pode ser zero, um ou dois, conforme a instrução e o número de endereços carregados pelo pacote executável [GUR80]. A porcentagem de instruções de cada tipo executadas afeta de forma considerável o desempenho da máquina.

2.2.4 *Switch Unit*

Devido ao anel ser somente um mecanismo de execução de grafos de fluxo de dados, foi necessária a inclusão desta unidade para possibilitar a comunicação com o meio exterior.

A unidade é composta de um *buffer* de entrada, um de saída e uma rede de comunicação 2 X 2, como mostra a Figura 2.8. Esta rede deve arbitrar entra duas fontes de dados distintas: o

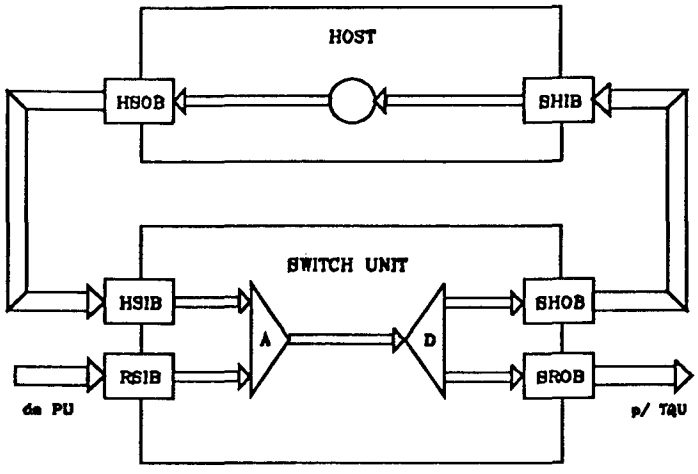

Figura 2.8: Esquema detalhado da **SW**.

computador hospedeiro e o anel. A natureza síncrona da unidade facilita esta tarefa e, sempre que dois pacotes chegam simultaneamente, é dada prioridade ao proveniente do anel. Os pacotes de resultados são reconhecidos na entrada da unidade e são enviados ao computador hospedeiro.

2.2.5 *Token Queue Unit*

A irregularidade do grau de paralelismo gerado pelos programas durante a sua execução tornou necessária esta unidade, atuando como uniformizadora do fluxo e reduzindo assim as bolhas que, de outro modo, poderiam formar-se no *pipeline*, comprometendo seu desempenho.

A Figura 2.9 mostra a estrutura da unidade, que é composta de um *buffer* de entrada, um de saída e uma memória com capacidade para 32k fichas, organizada na forma de uma fila circular FIFO. Para possibilitar a sobreposição de ações na unidade com a transferência de dados, foi incluído um *buffer* intermediário de leitura.

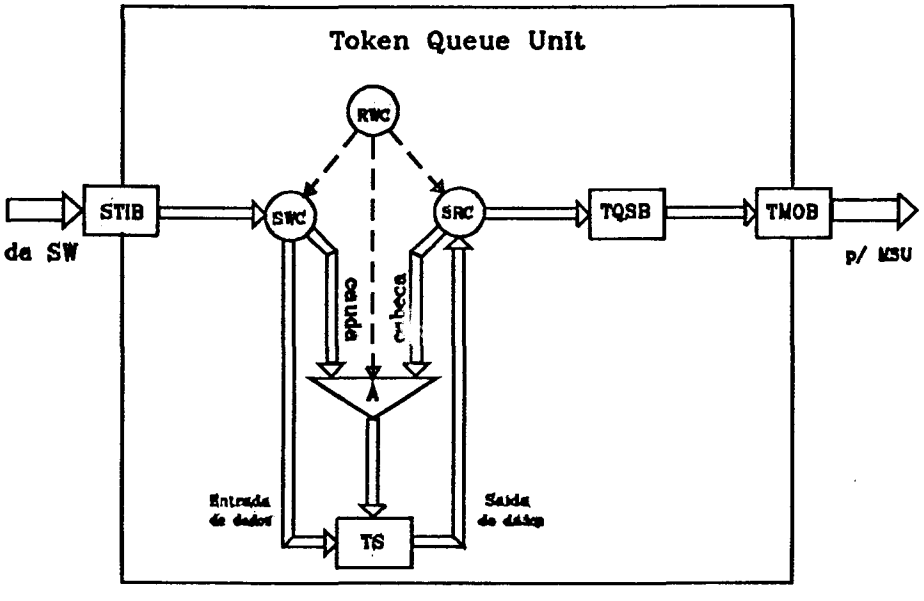

Figura 2.9: Esquema detalhado da **TQU**.

2.3 Problemas e Extensões da Arquitetura

Dentre os problemas mais sérios, inerentes ao modelo de fluxo de dados, está o tratamento de estruturas, pois, devido à semântica funcional implementada, é necessária uma cópia completa da estrutura a cada referência a ela. Além do custo proibitivo, tanto de tempo de execução, como de espaço de memória, tem-se também um aumento drástico do custo de exploração do paralelismo do algoritmo por trabalhar-se com a estrutura completa em vez de seus elementos. A primeira solução adotada em Manchester surgiu como subproduto de uma pesquisa sobre a programação de algoritmos não-determinísticos, através de ***matching functions*** **(mfs)**, que são carregadas pelas fichas e ditam o seu comportamento na **MSU** [CAT81]. As *matching functions* possibilitam implementar o armazenamento de estruturas na memória da **MSU**, de forma que é possível ter acesso a seus elementos individualmente, sem afetar a semântica do modelo.

Apesar de somente duas *matching functions* serem necessárias para as fichas normais, seis mais estão disponíveis. Exceção feita à *matching function* denominada ***bypass*** **(BY)**, empregada pelas fichas que não necessitam de agrupamento, as demais seguem um padrão para seus nomes e mnemônicos. Este padrão deriva das ações tomadas sobre a ficha de entrada, em função de estar ou não presente na memória da **MSU** sua ficha parceira. Os mnemônicos são formados pela justaposição das iniciais das ações, na presença da ficha parceira e na sua ausência, sendo as ações possíveis apresentadas abaixo ([GUR86], [DAS82], [CAT81]):

Ações na presença da ficha parceira:

"***Extract***" - remove a ficha parceira da memória;

"***Preserve***" - deixa uma cópia da ficha parceira na memória;

"***Increment***" - deixa uma cópia da ficha parceira incrementada de um na memória;

"***Decrement***" - deixa uma cópia da ficha parceira decrementada de um na memória.

Ações na ausência da ficha parceira:

"***Wait***" - insere a ficha de entrada na memória à espera da parceira;

"***Defer***" - acrescenta à ficha um parceiro do tipo "*null*" para formar o grupo de fichas e tenta novo agrupamento;

"***Abort***" - acrescenta à ficha um parceiro do tipo "*empty*" para formar o grupo de fichas;

"***Generate***" - como na anterior, porém também armazena a ficha de entrada na memória com sua aresta de entrada invertida.

Somente sete combinações destas ações foram consideradas utilizáveis, sendo apresentadas abaixo juntamente com seu possível emprego:

Extract-Wait - a *matching function* normal para os vértices com dois operandos;

Extract-Defer - empregada na entrada de regiões críticas, como uma forma de *busy waiting*; por exemplo, em situações não-determinísticas que não admitem o armazenamento de um dos operandos;

Preserve-Defer - empregada para armazenar valores, tais como elementos de um *array*, em um vértice de grafo;

Increment-Defer - empregada na entrada de monitores, para gerar uma seqüência de pedidos;

Decrement-Defer - empregada nos contadores de referências, de forma a que objetos não mais utilizados possam ser descartados;

Extract-Abort - empregada em situações não-determinísticas associadas à implementação de comandos com guardas;

Preserve-Generate - empregada na avaliação tardia de grafos.

Procurando uma solução definitiva para o problema do tratamento de estruturas, e seguindo a idéia de ***I-structures*** elaborada por Arvind [ARV80], desenvolveu-se uma unidade de tratamento de estruturas independente para a **MFDM**, a ***structure store unit*** **(SSU)**, ligada ao anel através da **SW** ([KAW86], [SAR86]).

A Figura 2.10 mostra a estrutura desta unidade, que é composta das seguintes subunidades:

- **de alocação (AU)**, responsável pela administração do espaço de memória da unidade;
- **de limpeza (CU)**, responsável pela execução de *garbage collection* automática na memória;
- **de memória de estruturas (SU)**, responsável pela armazenagem dos dados estruturados, permitindo acesso assíncrono aos mesmos e explorando melhor o paralelismo com o tratamento de elementos independentes da estrutura, porém mantendo a semântica funcional do modelo;
- **de tratamento de acessos indeferidos (DU)**, causados pela tentativa de realizar leitura a elementos não presentes na memória.

Dos problemas inerentes às arquiteturas que exploram o paralelismo de granularidade fina, o mais criticado é o uso excessivo de recursos, principalmente no que diz respeito à memória. Esse problema foi abordado no projeto da **MFDM**, resultando na implementação de um mecanismo **supressor/regulador** de paralelismo excessivo, que exerce controle sobre o grau máximo de paralelismo presente em determinado momento na máquina [RUG87].

Esse mecanismo faz uso tanto de *hardware* como de *sotfware* para realizar a sua tarefa, reduzindo o uso total de memória aos níveis das máquinas convencionais. Além disto, o

mecanismo pode levar ao reaproveitamento de nomes de ativação, campo da ficha criado a cada chamada de procedimento ou entrada de iteração, sem o que poderia ocorrer *overflow* do campo de rótulo durante o processamento de programas de maior porte.

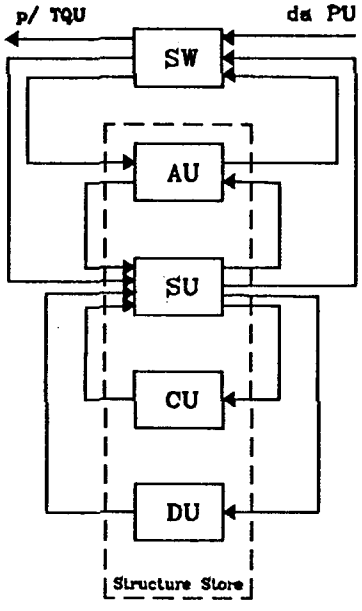

Figura 2.10: Esquema detalhado da SSU.

Após análise do código gerado, observou-se que o número de instruções empregadas na tradução de alguns construções de alto-nível era muito grande, quando comparado com as máquinas convencionais. Dessa forma, foram incluídas instruções mais complexas que possibilitaram a redução da árvore de duplicação, causada pela restrição imposta ao número de arestas de saída, e a redução do número de vértices empregados na tradução de iterações, entre outros comandos, melhorando o desempenho final da máquina [GUR86]. Outras alternativas consideradas preliminarmente pelo grupo de Manchester incluem tornar a máquina extensível através de uma estrutura com multianéis homogêneos [BAR84].

3 A Teoria Geral de Redes

Buscando descrever as relações de causa entre eventos C.A.Petri apresentou, em sua tese de doutorado [PET62], a base teórica para a comunicação entre componentes assíncronos de um sistema de computação. De seu trabalho, juntamente com os de Holt ([HOL70], [HOL68]), e Dennis *et al* [DEN70], desenvolveu-se uma teoria matemática abrangente sobre sistemas de informação baseada em conceitos como: comunicação, sincronização, fluxo de informação, escolha e concorrência.

Noções como **estado** e **transições** (mudanças de estado), que desempenham papel central na teoria dos sistemas de informação distribuídos, foram conceituadas e formalizadas na teoria geral de redes. Estas noções são empregadas na caracterização e análise do comportamento destes sistemas. Como princípios deste processo de conceituação e formalização tem-se que:

- estados e transições são noções inter-relacionadas, porém disjuntas;
- ambos, estados e transições, são entidades distribuídas.

O segundo princípio enunciado leva ao postulado de um conjunto de estados atômicos denominados **condições** e de um conjunto de transições atômicas denominadas **eventos**. Assim, um estado será visto como um conjunto de condições existentes concorrentemente e uma transição, ou seja, a passagem de um estado para outro, como um conjunto de eventos ocorrendo concorrentemente.

A relação entre estados e transições normalmente é representada através de um grafo que liga condições e eventos. As redes assim formadas descrevem a estrutura dos sistemas de informação distribuídos e podem ser vistas como a sintaxe de uma linguagem de representação desses sistemas. A semântica dessa linguagem, que descreve o comportamento do sistema, é normalmente especificada por regras de simulação para o funcionamento dessas redes.

A formalização da Teoria Geral de Redes está fortemente documentada ([REI85A], [ROZ80]), e não será abordada diretamente neste trabalho. A seguir apresentar-se-ão informalmente os principais conceitos e sistemas de interpretação dessa teoria, empregados no estudo dos possíveis comportamentos e das características dos sistemas de informação distribuídos.

3.1 Conceitos Básicos

A Teoria Geral de Redes baseia-se na relação entre duas classes de objetos. Esses objetos, enquanto não forem interpretados de alguma forma especial, como condições e eventos por

exemplo, serão denominados genericamente **elementos-S** e **elementos-T**. A relação existente entre esses objetos será denominada **relação de fluxo**, e a estrutura final formada por esse conjunto de entidades será denominada **rede**.

A forma mais comum de representação destas redes é através de um grafo bipartido, com vértices compostos de elementos-S, representados por **círculos**, e de elementos-T, representados por **retângulos**. Esses vértices estão ligados por arestas dirigidas dos elementos-S para os elementos-T, caso em que os primeiros compõem o **pré-conjunto** dos elementos-T; e dos elementos-T para os elementos-S, caso em que os últimos compõem o **pós-conjunto** dos elementos-T, conforme mostra a Figura 3.1. A assinalação da existência de um elemento-S ativo é realizada através de **marcas**, representadas por pontos pretos na Figura abaixo, adicionados aos elementos-S da rede que estão nesta situação.

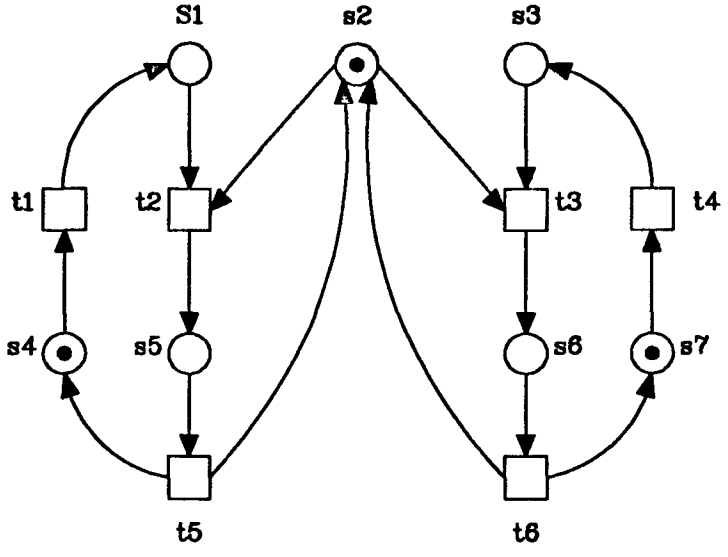

Figura 3.1: Rede básica formada de estados atômicos (elementos-S) e transições atômicas (elementos-T).

As redes formadas pela relação dos elementos S e T podem ser divididas em classes, como, por exemplo, redes finitas, puras e simples. Uma rede será dita **finita** quando o conjunto de seus elementos S e T for finito; uma rede será **pura** se não contiver nenhum **laço**, ou seja, se nenhum de seus elementos-S pertencer tanto ao pré quanto ao pós-conjunto de algum elemento-T, e uma rede será **simples** se, e somente se, elementos distintos na rede não tiverem o mesmo pré e pós-conjuntos. A rede da Figura 3.1 é finita, pura e simples e também não contém elementos isolados. Nada dito em contrário, as redes mencionadas neste trabalho serão finitas, puras e simples.

Duas redes N e N' são ditas **isomorfas** se apresentarem um **morfismo** entre si, ou seja, se apresentarem um mapeamento de um conjunto de elementos da rede N em um elemento da rede N'. A rede N será a **fonte** do morfismo, e será dita ser um **refinamento** de N', a rede **objetivo** do

morfismo, como pode ser visto na Figura 3.2. Morfismos podem ser representados também por inscrições especiais atribuídas à rede fonte ou à rede objetivo [GEN80].

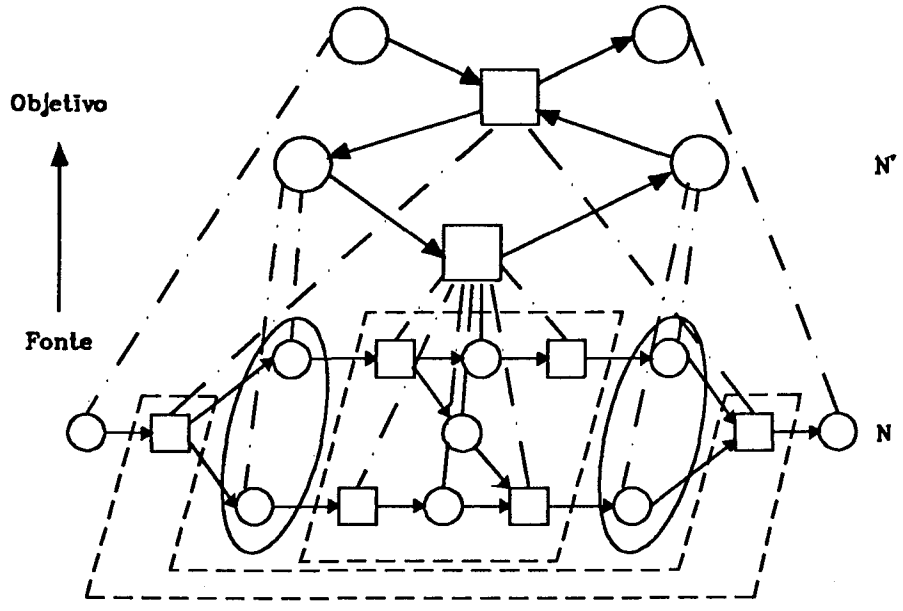

Figura 3.2: Isomorfismo entre redes.

Os morfismos são muito importantes no desenvolvimento hierárquico de modelos, pois através dos mesmos é possível definir a rede que representará o sistema estudado em vários níveis de abstração. Uma vez definidos os modelos nos vários níveis pode-se mover de um nível para o outro conforme a necessidade de ampliação ou simplificação do grau de informação necessário ao estudo realizado.

3.2 Interpretações da Teoria das Redes

A estrutura das redes geradas por esta teoria aceita vários sistemas de interpretação semântica que visam caracterizar os diferentes tipos de redes e sistemas estudados. A interpretação dos elementos da rede como condições e eventos, gera o sistema de condições/eventos que corresponde à interpretação básica da teoria. A possibilidade de um elemento-S ser ocupado por várias marcas proposto por Petri, originou o sistema de lugar/transição normalmente conhecido como redes de Petri. Mais recentemente, o tratamento de marcas como indivíduos originou o sistema de redes de alto-nível, muito úteis na descrição de sistemas complexos. A seguir será apresentada a conceituação desses sistemas e suas principais características.

3.2.1 Sistema de Condições/Eventos

No sistema de condições/eventos (**sistemas C/E**) os elementos-S representam as **condições** e os elementos-T os **eventos**. Nesse sistema condições são satisfeitas ou não, ou seja, um elemento-S

poderá ter no máximo uma marca, e a ocorrência de um evento muda o conjunto de condições existentes.

A ocorrência de um evento no sistema será regida por regras de simulação para o funcionamento das redes. Na sua forma mais elementar, essas regras exigem que um evento esteja **habilitado** para que possa **disparar**, gerando outro estado. A habilitação de um evento é dada pela existência das suas pré-condições e da não existência das suas pós-condições. Na ocorrência de um evento suas pré-condições deixam de vigorar e suas pós-condições passam a serem válidas. Desta forma, a ocorrência de eventos ou um conjuntos destes, seqüencial ou concorrentemente, altera o estado do sistema, definindo seu comportamento. A Figura 3.1 pode ser interpretada como um sistema C/E que modela a exclusão mútua entre duas tarefas, S5 e S6.

Ao conjunto de condições existentes num determinado estado do sistema denominar-se-á **caso**. Quando um sistema passa de um caso para outro, novos eventos podem ocorrer, gerando eventualmente outros casos. Um conjunto de eventos habilitados de um caso é dito **desconexo** se, e somente se, seus conjuntos de pré e pós-condições forem disjuntos. Eventos desconexos podem ocorrer concorrentemente e ser combinados para formar um **passo**, ou seja, uma transição de estado de um caso para outro no sistema.

Para um sistema C/E estar totalmente descrito é necessário que sejam especificadas sua estrutura, ou seja, a sua rede, e sua dinâmica, ou seja, todos os casos a serem considerados pelo sistema. Esse conjunto de casos deve ser especificado de tal modo que:

- um passo possível num caso do sistema derive outro caso que pertença ao conjunto;
- inversamente, se um caso resulta de um passo, o estado anterior do sistema deve pertencer ao conjunto de casos;
- todos os casos do conjunto possam ser transformados reciprocamente;
- o conjunto deve ser grande o suficiente para que cada evento tenha um caso em que ele tenha concessão, e cada condição pertença a pelo menos um caso, porém não pertença a todos.

Todos estes requisitos restringem a classe de sistemas modelados, tornando possível demonstrar que a classe de casos de uma rede pode ser definida por um único caso da rede [REI85A]. Dessa forma, é comum definir-se um sistema C/E pela sua estrutura e pelo seu caso inicial, descrito através de marcas sobre as condições da rede.

A exigência de que, para um evento ocorrer, todas suas pré-condições e nenhuma de suas pós-condições vigorem possibilita a ocorrência de situações em que um evento seja inabilitado pela existência de alguma pós-condição. Tais situações são denominadas **situações de contato**. Sempre é possível, e interessante, trabalhar com sistemas livres de contato, o que não reduz o

poder de representação do sistema [REI85A]. Nada dito em contrário, trabalhar-se-á somente com sistemas livres de contato.

Durante a modelagem do comportamento de sistemas, dois eventos *e1* e *e2* podem relacionar-se em um caso de pelo menos três maneiras distintas:

i) Dois eventos são ditos ocorrer em **seqüência** se o primeiro puder ocorrer em um caso e o segundo não. Porém, após o primeiro ocorrer o segundo também poderá, como indicado na Figura 3.3 para *e1* e *e2*.

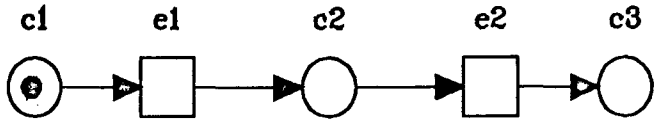

Figura 3.3: Sistema C/E com eventos seqüenciais.

ii) Dois eventos são ditos estar em **conflito** se ambos puderem ocorrer individualmente em um caso, mas não poderem ocorrer conjuntamente neste mesmo caso, como demonstrado a Figura 3.4 para *e1* e *e2*.

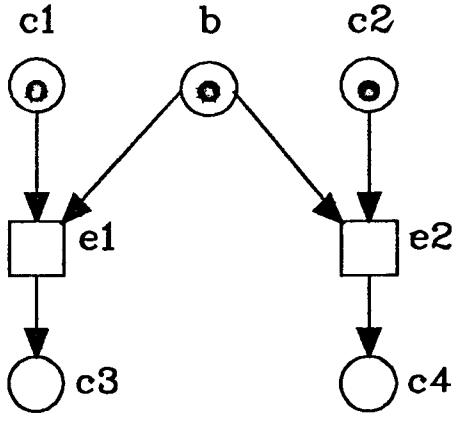

Figura 3.4: Sistema C/E com eventos em conflito.

Na Figura 3.4 a condição compartilhada *b* impede a ocorrência de ambos os eventos, apesar de cada um poder ocorrer individualmente. Esta é uma forma muito útil para modelar exclusão mútua entre eventos de um sistema.

iii) Dois eventos são ditos **concorrentes** quando ambos podem ocorrer em um caso sem ordem específica sobre suas ocorrências, como demonstrado à Figura 3.5 para *e1* e *e2*.

Na Figura 3.5 ambos os eventos podem ocorrer sem interferir mutuamente e sem uma ordem específica para suas ocorrências, de forma que é possível modelar o comportamento não-seqüencial em sistemas.

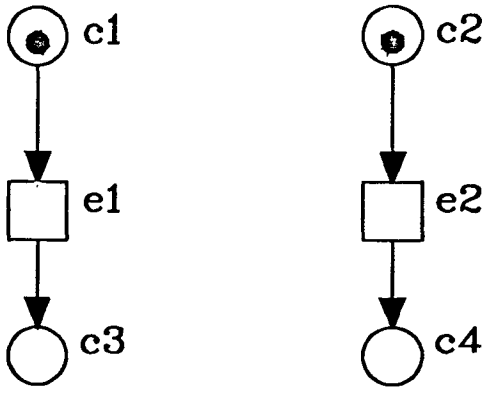

Figura 3.5: Sistema C/E com eventos concorrentes.

A existência de conflitos numa rede implica em que a seqüência de ocorrência de eventos não será totalmente determinística, ou seja, para se obter a resolução de um conflito existe a necessidade de maior informação sobre o modelo, o que normalmente provem de um observador externo. Dessa forma, um conflito modela o ganho ou perda de informações pelo sistema.

A combinação das três situações acima, juntamente com o conceito de isomorfismo entre redes, permite modelar sistemas reais de forma hierárquica, com base nas suas condições e nos eventos que as transformam. Porém, algumas combinações, onde ocorre sobreposição de concorrência e conflito, geram problemas na determinação da seqüência de ocorrência de eventos. Tais situações são denominadas de **confusão**, como demonstra a Figura 3.6.

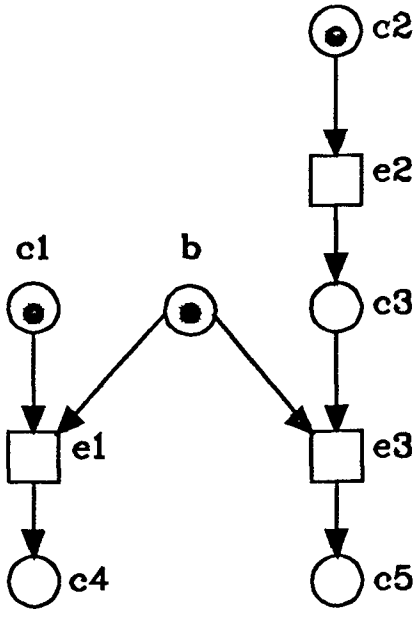

Figura 3.6: Sistema C/E com situação de confusão.

Na Figura acima se *e1* ocorrer primeiro não haverá conflito, porém a ocorrência de *e2* primeiro provocará conflito entre *e1* e *e3*, gerando a confusão entre as possíveis seqüências de

eventos. As situações de confusão tornam a análise do sistema muito complexa e devem sempre ser evitadas.

A descrição do comportamento dinâmico de um sistema C/E, gerada pela execução da rede associada sobre o caso inicial a ela atribuído, é melhor compreendida pelo conceito de **processos**. Para caracterizar tais processos foram desenvolvidas **redes de ocorrência**, que não serão abordadas neste trabalho mas estão bem descritas em [REI85A].

3.2.2 Sistema de Lugar/Transição

O sistema C/E apresenta grande importância para a caracterização do comportamento dos sistemas de informação distribuídos, principalmente no que concerne à descrição de concorrência entre os eventos do sistema modelado. No entanto, a limitação de que somente uma marca possa ocupar um elemento-S restringe seu escopo de aplicação à representação da existência ou não de uma condição. A eliminação desta restrição e a adição da possibilidade de um elemento-T consumir ou gerar mais de uma marca, de um ou para um mesmo elemento-S, amplia seu escopo de aplicação permitindo também a representação da existência de uma quantidade variável de entidades em um elemento-S, dando lugar ao sistema de Lugar/Transição (**sistema L/T**).

O sistema L/T é muito empregado quando a movimentação e a distribuição de informação são importantes. Em tais áreas o que se procura é o estudo da organização do fluxo de informação, de forma a avaliar as possíveis variações a que o sistema pode ser submetido, sempre sujeitas a limitações a serem respeitadas. Dentro deste escopo, os elementos passivos (*buffers*, armazéns, etc.) do sistema serão representados por elementos-S, e denominados **lugares**; os elementos ativos (processadores, máquinas, etc.) serão representados por elementos-T, e denominados **transições**. A informação que estará fluindo será representada por marcas inseridas na rede.

A definição de uma rede no sistema L/T irá requerer que se estipule, além de sua estrutura, a **capacidade** de marcas dos lugares e o **peso** a ser inscrito nas arestas da rede, representando o número de marcas que deverão fluir por esta aresta. A capacidade dos lugares e o peso das arestas deverá ser maior que zero, podendo a capacidade ser ilimitada, caso que se representará com um **w** ao lado do círculo que representa o lugar na rede. Desta forma, o estado atual do sistema será representado pela **marcação** da rede, ou seja, pela distribuição de marcas sobre os lugares da rede, tal que o número de marcas seja maior ou igual a zero e menor que a capacidade dos lugares da rede.

No sistema L/T as regras de simulação para o funcionamento das redes permitirão que uma transição dispare se, e somente se, todos os lugares de seu pré-conjunto contiverem marcas em número maior ou igual ao peso das arestas que os ligam à transição e se cada lugar de seu pós-conjunto apresentar espaço para receber as novas marcas que serão geradas, ou seja, o

número de marcas atuais destes lugares mais o peso das arestas que fazem sua ligação com a transição correspondente deverá ser menor ou igual à capacidade do lugar, conforme demonstra a Figura 3.7. Observe-se que, na representação de um sistema C/E, emprega-se o caso inicial para representar a classe de casos válidos, enquanto nos sistemas L/T emprega-se uma **marcação inicial**, de forma que os comportamentos dos sistemas serão diferentes.

À nova marcação gerada pela ocorrência de uma transição habilitada, dentro de uma marcação pré-existente, dá-se o nome de **marcação diretamente seguinte** à marcação atual, e diz-se que a marcação produzida é **acessível** a partir da marcação pré-existente. O conhecimento do conjunto de todas as marcações acessíveis a partir de uma dada marcação é muito útil para a compreensão do comportamento do sistema. Normalmente, o **conjunto de marcações acessíveis** para uma dada marcação é infinito, porém existe uma forma de representá-lo através de um grafo finito denominado **árvore de cobertura** da rede ([REI85A], [ROZ80]).

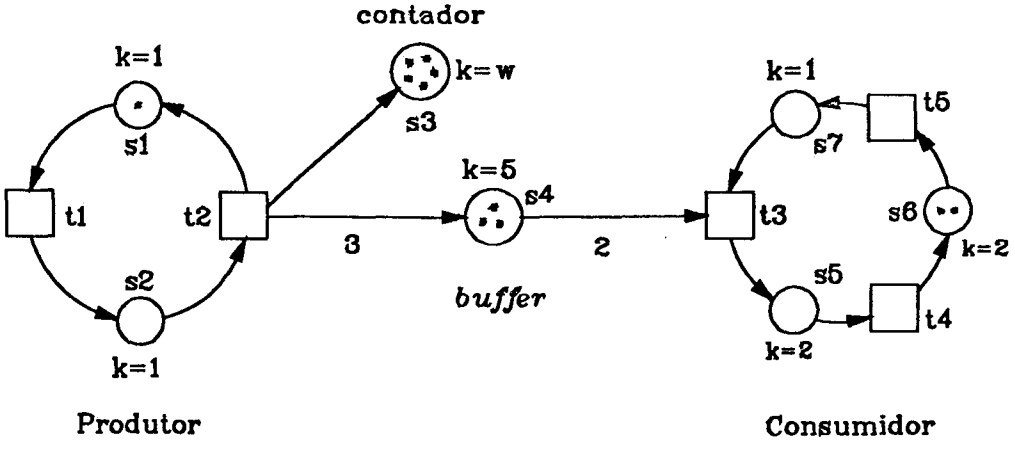

Figura 3.7: Sistema L/T com transições habilitadas (t1,t3,t5) e desabilitadas (t2,t4).

Dentre as propriedades dos sistemas modelados que se procura verificar estão as situações em que há bloqueio, parcial ou total, no fluxo da informação e as situações em que a capacidade limite de um lugar pode ser superada.

A primeira situação pode ser verificada identificando-se as transições que podem alcançar uma marcação a partir da qual não venham mais a ser disparadas, caso em que são ditas transições **mortas**. Essas transições tem relação direta com o conceito complementar a este, o de **vida** de uma transição. O conceito de **vivacidade** para uma transição pode aceitar várias definições dependendo das regras de simulação definidas para a rede. Porém, independentemente do conceito adotado, uma rede é dita **viva** se, e somente se, para toda marcação seguinte acessível a partir da marcação inicial todas suas transições são vivas.

A segunda propriedade pode ser verificada através da análise dos estados que a rede pode atingir. Assim, uma rede é dita **limitada** se, e somente se, independentemente da capacidade dos

lugares, existir para todo lugar da rede um **limite superior** para o número de marcas que poderá ser atingido pelo lugar, dentro do conjunto de marcações acessíveis. Esta propriedade é muito importante na modelagem de sistemas reais, pois estes sempre têm um limite superior para a capacidade de seus agentes passivos.

Os conceitos acima são mais facilmente verificados através da árvore de cobertura da rede, pois através desta poder-se-á determinar tanto o limite de marcas para os lugares quanto a vivacidade das transições presentes na rede [REI85A]. Além destes conceitos, é possível estender os conceitos de redes com contato, com laço, puras e simples, anteriormente definidas, para os sistemas L/T. Desta forma, continuar-se-á a tratar neste trabalho somente de redes livres de contato, puras, simples e finitas.

Outra forma de representar essas redes é através de uma **matriz de incidência.** Para as redes que são puras, simples e finitas esta matriz é única sendo muito útil para a análise de propriedades do sistema modelado. Tal matriz corresponde a um mapeamento do produto cartesiano dos conjuntos de lugares e transições sobre o conjunto dos números naturais. Esse mapeamento é resultado da atribuição aos elementos da matriz de uma função sobre o peso das arestas [REI85A].

O comportamento dinâmico de um sistema L/T depende diretamente da estrutura de sua rede e de sua marcação inicial. Do ponto de vista de propriedades a serem verificadas, é muito importante poder garantir que um conjunto de marcas de uma determinada rede não será perdido, pelo menos não de forma descontrolada. Essa propriedade, assim como a determinação da seqüência de transições necessárias para que uma marcação se repita, são passíveis de determinar-se através da obtenção dos **invariantes S** e **T** da rede em estudo.

Um invariante-S é um conjunto de lugares pertencentes a uma rede cujo número de marcas é constante, independentemente da ocorrência de transições. A Figura 3.8 mostra um invariante-S. Na rede apresentada, o número de marcas do conjunto de lugares {s1,s3,s4} não será alterado pela ocorrência de transições. Um invariante-T indica a freqüência, partindo-se de uma marcação, com que um determinado conjunto de transições deve disparar para reproduzir a marcação de partida. Ambos conceitos são de grande valia na análise de propriedades como vivacidade, limite, reprodutividade de marcações e verificação de fatos associados a um sistema L/T ou C/E, este último visto como um caso particular do primeiro. Além disto, o conceito de invariantes-S será de grande importância no desenvolvimento de um modelo para aplicação em análise de desempenho de sistemas, como será demonstrado posteriormente.

As técnicas para o cálculo dos invariantes-S e T utilizam-se da matriz de incidência da rede e dos métodos da álgebra linear. Essas técnicas, assim como suas várias aplicações, encontram-se bem descritas em vários trabalhos ([LAU87], [MEM87], [REI85A]) e não serão abordadas diretamente neste trabalho.

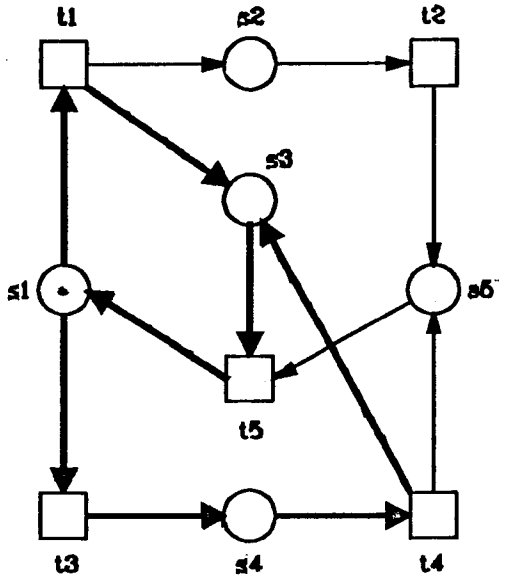

Figura 3.8: Sistema L/T com um invariante-S.

3.2.3 Sistema de Redes de Alto-Nível

Os sistemas C/E e L/T não fazem distinção entre as marcas da rede. Esta abordagem, apesar de útil na análise de uma vasta gama de propriedades de sistemas, apresenta problemas quando se deseja verificar fatos sobre elementos individuais do mesmo. Os problemas advêm principalmente do fato de as redes se tornarem muito grandes, inviabilizando as técnicas de análise anteriormente desenvolvidas. A consideração das marcas como indivíduos, ou seja, como entidades que carregam consigo uma identidade, faz com que as redes necessárias para a representação desses sistemas sejam bastante simplificadas. Esta simplificação vem ao custo do aumento da complexidade da semântica do modelo e da necessidade de estenderem-se as técnicas de análise disponíveis.

O novo sistema gerado recebe o nome de **sistema de rede de alto-nível**. Nesse sistema as marcas representam entidades que carregam consigo uma identidade própria. Os elementos-S representam elementos passivos do sistema, ou seja, lugares que armazenam indivíduos de um grupo de classes de entidades. Os elementos-T representam elementos ativos do sistema, ou seja, transições que atuam como movimentadoras ou transformadoras de marcas.

O sistema de redes de alto-nível comporta uma grande variedade de tipos de redes, entre as quais as mais comuns são: **redes de predicado/transição (redes Pr/T)** ([GEN87], [GEN81]); a **redes coloridas** ([JEN87], [JEN81]); e as **redes relacionais** ([REI85A], [REI85B]).

Essas redes têm em comum as seguintes características:

a) necessitam da especificação de um **universo de discurso (UD)** para o modelo, visto que as entidades presentes neste são diferentes entre sí;

b) podem ter **incrições** inseridas nas suas transições que, normalmente, serão compostas de assertivas lógicas, funções ou relações entre entidades do UD do modelo;

c) podem ter incrições sobre suas arestas que determinarão o tipo de entidades do UD do modelo, ou seja, quais tipos de marcas, poderão fluir pelas mesmas.

Desta forma, a definição de um modelo no sistema de redes de alto-nível exige que se especifique: a rede que definirá sua estrutura; as incrições que irão sobre as arestas e internamente as transições; assim como o UD do modelo, os dois últimos itens definirão a semântica do modelo.

As regras de simulação para o funcionamento das redes nos sistemas de redes de alto-nível, normalmente, exigem para a habilitação de uma transição que:

- existam marcas satisfazendo as incrições das arestas que unem as transições e seus pré-lugares;
- a inscrição interna à transição, se existir, seja satisfeita;
- dependendo do tipo de rede, não exista colisão entre as marcas geradas e outras já existentes nos pós-lugares da transição considerada.

Toda transição que satisfizer a estas exigências está apta a ocorrer gerando uma nova marcação e um novo estado na rede.

Procurando aproveitar o que havia de positivo nas redes de Pr/T, que derivam do sistema C/E, e nas redes coloridas, Heuser desenvolveu um tipo de rede que emprega uma linguagem da lógica de predicados de primeira ordem na descrição das inscrições a serem inseridas nas arestas e transições ([HEU89A], [HEU89B]). Esta linguagem atua sobre o UD do sistema modelado, complementando a rede com incrições que especificarão sua semântica. Estas redes receberão o nome de ***Condition Event Modelling nets*(redes CEM)**.

A linguagem desenvolvida para as redes CEM compactas especifica operações com entidades dentro do UD do modelo. Estas operações podem definir:

- constantes e variáveis dentro do UD;
- funções entre entidades do UD;
- operações relacionais e existenciais entre entidades do UD;
- e operações sobre conjuntos de entidades do UD.

A Figura 3.9 apresenta um exemplo de uma rede CEM modelando um sistema de transações num mercado de trabalho.

Os conceitos vistos nesta parte do trabalho não têm a função de apresentar uma descrição detalhada das redes CEM. Estes buscam colocar o leitor a par da situação em que se encontra a

teoria nesta área. A sintaxe e semântica das redes CEM estão bem descritas nos trabalhos de Heuser, anteriormente citados e, apesar de terem sido aplicadas neste trabalho, não serão objeto de análise mais detalhada.

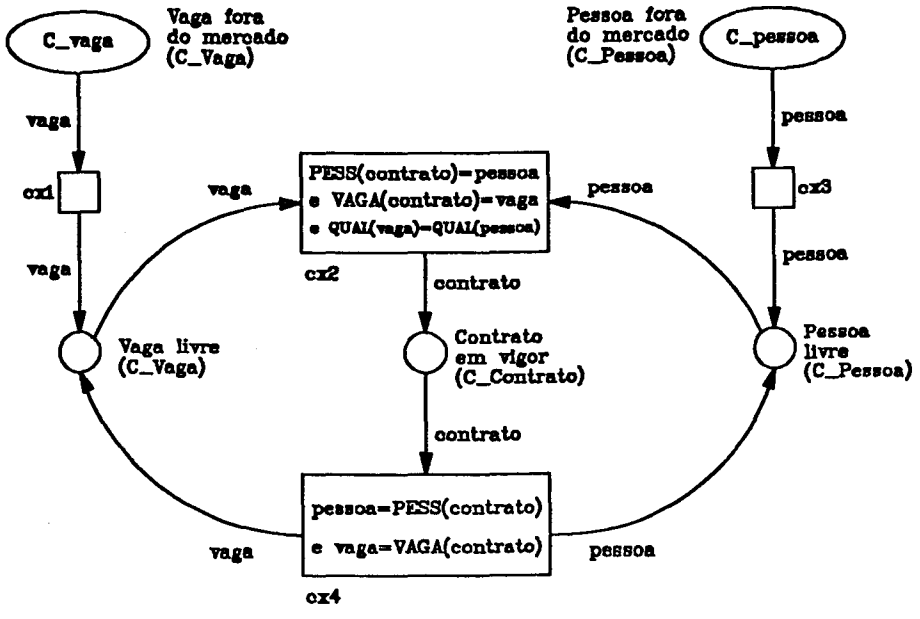

Figura 3.9: Rede CEM compacta representando um sistema de transações num mercado de trabalho.

3.3 Um Método para a Análise de Desempenho

Normalmente modela-se um sistema computadorizado visando obter algum conhecimento em relação ao seu comportamento, ou como forma de estabelecer uma linguagem comum de trabalho. A análise de desempenho de um sistema, frente a certas situações de carga de trabalho, tem sido uma preocupação constante das pessoas que trabalham nesta área, por implicar em perda ou ganho de lucratividade dos empreendimentos.

Para se obter uma avaliação do desempenho de um sistema é necessário realizar uma análise quantitativa do mesmo, obtendo-se assim valores para parâmetros como: seus tempos de resposta a um pedido de serviço; sua taxa de utilização de recursos sobre um tipo de carga de serviço, etc.

O uso das redes de Petri na análise de desempenho requer que se introduza o conceito de tempo nos sistema desenvolvidos. Este conceito pode ser associado tanto aos lugares quanto às transições da rede, sendo a escolha indiferente do ponto de vista de análise, conforme demostrado em [SIF80]. Vários trabalhos têm sido desenvolvidos nesta área, introduzindo o conceito de tempo tanto de forma estocástica ([MAR84], [MOL82]) quanto determinística ([TAZ88], [HOL87]).

Em sua tese de doutorado [TAZ85], Tazza desenvolveu um método de análise determinístico denominado **Modelo-Q**. Seu trabalho faz uso do sistema L/T para a definição do padrão de utilização de recursos do sistema, e introduz o conceito de tempo associado aos lugares da rede. Dessa forma, uma marca que entra em um lugar temporizado permanecerá indisponível para o disparo das transições da rede durante o tempo atribuído ao respectivo lugar.

O método desenvolvido corresponde a um conjunto de equações para a determinação de parâmetros do sistema, e é composto de quatro camadas, diferenciadas pelo padrão de utilização dos recursos do sistema. As camadas estão organizadas de forma incremental, ou seja, a classe de problemas analisáveis por uma camada mais externa inclui a das camadas mais internas.

O conceito básico no desenvolvimento das redes no modelo-Q é o de **cadeia principal**, ou cadeia aberta, de um determinado processo. Esse processo será o alvo da avaliação de desempenho e é composto de um grupo de subtarefas representadas por lugares da rede. A cadeia aberta de um modelo contém subtarefas do processo modelado, e pontos de espera por sincronização ou recursos que são utilizados pelos processos na realização de suas tarefas, estes recursos são modelados por marcas inseridas em lugares específicos, a margem da cadeia principal.

Esses conceitos estão exemplificados na rede da Figura 3.10, que modela um sistema composto de um processo com duas subtarefas. A primeira emprega somente o recurso r_1, alocado pela transição ts_1, e é modelada pelo lugar de utilização u_1. A segunda emprega ambos os recursos, r_1 e r_2, e é modelada pelo lugar de utilização u_2. A alocação do recurso r_2 é realizada pela transição ts_2. Após a conclusão destas tarefas ambos os recursos são liberados pela transição tr_1.

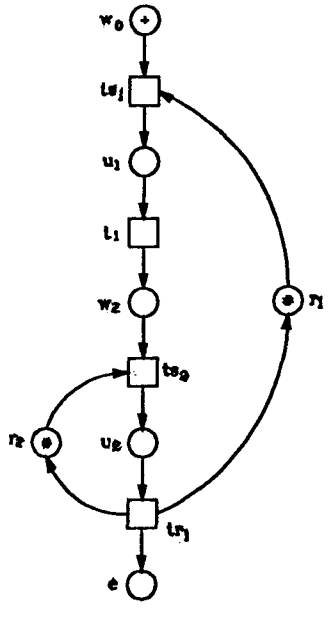

Figura 3.10: Rede modelando um sistema com dois recursos r_1 e r_2 empregando o modelo-Q.

Nessa rede a cadeia principal é constituída pelos lugares w_0, u_1, w_2, u_2 e e. O lugar w_0 modela um número infinito de solicitações à espera de atendimento pelo sistema. O lugar w_2 modela a espera pela disponibilidade do recurso r_2 para a realização da segunda subtarefa. O lugar e modela um depósito de solicitações atendidas que terão tempo de espera indefinido.

Desta forma, pode-se verificar que o pressuposto básico do modelo-Q é que o sistema trabalhe em sua capacidade máxima, ou seja, tantas requisições da tarefa-alvo quantas forem possíveis serão atendidas.

A definição da classe de problemas analisáveis por uma camada do modelo-Q é realizada através da definição do seu escopo de aplicação. A especificação do escopo de aplicação de uma camada requer que se defina:

a) os **aspectos estruturais** do padrão de utilização de recursos, através de um conjunto de restrições sobre as relações entre os lugares e transições de uma rede genérica;

b) os **aspectos quantitativos** do padrão de utilização, através de restrições impostas sobre a capacidade dos lugares, sua marcação inicial e o peso das arestas;

c) os **aspectos temporais** do padrão de utilização, através da definição de um tempo de permanência das marcas nos lugares da rede.

O escopo de aplicação de uma camada será dado pelo conjunto de todas as redes temporizadas que sigam as restrições acima especificadas.

A primeira camada, ou seja, o núcleo do método, é denominada **modelo-Q0** e terá como classe de problemas analisáveis os sistemas em que:

a) os processos são uniformes com respeito à quantidade e tempo de utilização do recurso;

b) um único tipo de recurso está disponível;

c) todas as unidades de recurso necessárias ao atendimento de uma solicitação são alocadas

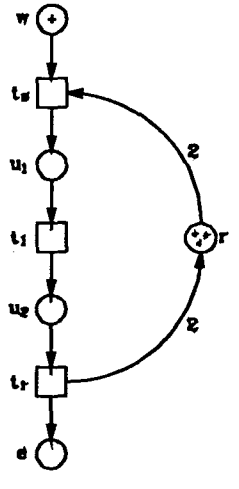

Figura 3.11: Sistema modelado empregando-se o modelo-Q0.

e liberadas posteriormente em um único passo.

A Figura 3.11 apresenta um sistema modelado pela aplicação do modelo-Q0.

Para a segunda camada, denominada **modelo-Q1**, a classe de problemas analisáveis será composta dos sistemas em que:

a) os processos são uniformes com respeito à quantidade e tempo de utilização do recurso;

b) uma solicitação pode cobrir concorrentemente diversos tipos de recursos;

c) todas as unidades de um tipo de recurso são alocadas e posteriormente liberadas em um único passo;

d) uma solicitação abandona o sistema após liberar todas as unidades cobertas por ela.

A Figura 3.10 apresenta um sistema modelado pela aplicação do modelo-Q, satisfazendo as restrições do modelo-Q1.

Na terceira camada, denominada **modelo-Q2,** a classe de problemas analisáveis será ampliada para os sistemas em que:

a) os processos são uniformes com respeito à quantidade e tempo de utilização do recurso;

b) a quantidade de recursos solicitada é conhecida a cada passo;

c) para continuar em atividade as solicitações cobrem recursos de um depósito central;

d) as unidades de recurso liberadas retornam ao depósito central;

e) os recursos são cobertos e liberados concorrentemente e em qualquer ordem pelas solicitações;

f) após liberar todas as unidades cobertas por ela, uma solicitação abandona o sistema.

A Figura 3.12 apresenta um sistema modelado pela aplicação do modelo-Q2.

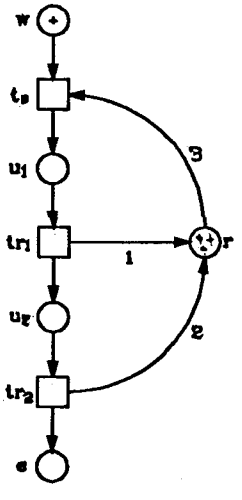

Figura 3.12: Sistema modelado empregando-se o modelo-Q2.

A quarta camada é denominada **modelo-Q3**, e sua classe de problemas analisáveis englobará situações em que a necessidade de recursos pelos processos é diferenciada. Esta diferença pode manifestar-se através de:

a) diferentes tempos de utilização para os recursos envolvidos;

b) diferentes quantidades de recursos necessários;

c) diferentes tipos de recursos necessários.

A Figura 3.13 apresenta um sistema modelado empregando-se o modelo-Q3.

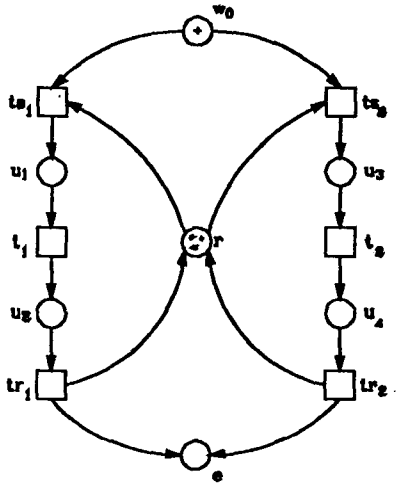

Figura 3.13: Sistema modelado empregando-se o modelo-Q3.

Uma vez definido o escopo de aplicação de um modelo, é possível desenvolver as equações de fluxo que servirão para a análise quantitativa do sistema modelado.

O tempo decorrido entre a alocação de um recurso em resposta a uma solicitação e sua liberação é denominado de **período básico** do recurso. O número máximo de solicitações que podem ser atendidas concorrentemente corresponderá ao maior inteiro menor que a razão entre a quantidade inicial de recursos e o número de recursos alocados por vez a cada solicitação. A razão entre este valor e o período básico estabelece o número de solicitações atendidas por unidade de tempo e é denominada de **taxa de fluxo básica** do recurso.

Uma vez determinada a taxa de fluxo básica de todos os recursos da rede, é possível obter a taxa de fluxo da rede como um todo, ou seja, seu **desempenho**. O valor do desempenho da rede será igual ao valor mínimo dos desempenhos dos recursos individuais. Este fato está baseado na hipótese de que o recurso de menor desempenho determinará a taxa de fluxo do sistema, vindo os demais recursos a trabalharem neste valor.

Dado que os recursos com valores de desempenho maiores que a taxa de fluxo da rede serão atrasados ocorrerá a existência de um **tempo induzido de espera** para sua alocação provocando sua **subutilização**. É possível, para sistemas analisados em camadas acima do

modelo-Q1, que também ocorra a existência de um tempo induzido de espera na alocação de um recurso a uma solicitação.

Além destes parâmetros, é possível determinar também a **população média** de um determinado lugar da rede, que será dada pelo produto do **período corrigido** de um recurso, calculado como o período básico do recurso mais os tempos induzidos de espera a que o mesmo está sujeito, pela taxa de fluxo da rede.

A formalização dos modelos desenvolvidos por Tazza, assim como as equações utilizadas na sua análise são apresentados em [TAZ88] e não serão abordadas diretamente neste trabalho.

3.3.1 Métodos Aproximativos de Análise

Apesar do amplo escopo de aplicação, o modelo-Q não apresenta bons resultados na análise de sistemas que apresentam tempo de utilização ou quantidade de alocação não-uniformes de recursos. Para cobrir esta falha, foram desenvolvidos dois métodos aproximativos [FER90]. Estes métodos baseiam-se na obtenção de diversos valores intermediários para a taxa de fluxo da rede, que são então ponderados segundo especificações complementares sobre o sistema que está sendo modelado e vão gerar um novo valor para a taxa de fluxo do modelo final. A partir deste novo valor é que serão calculados todos os demais parâmetros do sistema.

O primeiro método aproximativo procura melhorar a resposta apresentada para os sistemas com tempo de utilização não-uniforme. Esta classe de problemas não é aceita no modelo-Q, sendo que pelo mesmo a única solução possível seria a adoção de um tempo médio de utilização, o que apresenta uma resposta pouco satisfatória.

Este método aproximativo consiste na definição de um conjunto de valores para os tempos de utilização possíveis, com as respectivas probabilidades de ocorrência. A partir desses valores, associa-se a cada tempo uma rede diferente e realiza-se o cálculo dos desempenhos para cada uma delas. Ao final, ponderam-se os valores do desempenho das diversas redes através das probabilidades de ocorrência de cada tempo, sendo o valor final adotado como a nova taxa da rede original.

Casos mais complexos, como a existência de vários lugares com tempos de utilização variáveis, também podem ser analisados por este método. Nestes casos, será necessária a análise de uma rede para cada possível combinação de valores entre os conjuntos de tempos dos lugares da rede original.

O segundo método aproximativo procura tratar os casos em que existe, ou é necessário considerar, uma variação no número de recursos disponíveis. Essas variação ocorrerá, por exemplo, quando existir a necessidade de um recurso específico do sistema para a realização de uma tarefa, ou quando o número de recursos de um determinado tipo variar com o tempo.

Este método, como o anterior, considera diversas redes na análise. Estas redes correspondem aos diversos valores de disponibilidade de recursos que possam existir no sistema, com suas respectivas probabilidades de ocorrência. Esta variação é modelada com valores diferentes para a marcação inicial do lugar do recurso correspondente. Determinado o desempenho para estas redes, pode-se obter o valor do desempenho do sistema através da ponderação dos resultados correspondentes, e, a partir deste, os parâmetros restantes. Casos mais complexos, como a variação da quantidade de mais de um recurso, são tratados da mesma forma que no método anterior.

Ambos os métodos apresentam resultados mais próximos da realidade. No primeiro, isto ocorre devido à consideração das flutuações na taxa de fluxo do recurso em questão, variação esta que poderá em alguns casos levar o recurso a ser o recurso crítico do sistema, caso já não o seja. No segundo, isto ocorre devido à consideração da disputa existente pelo recurso ou pela ausência de recurso em um determinado instante. Ambos os casos poderão causar um bloqueio do sistema à espera do recurso em falta, podendo esse bloqueio resultar na transferência do recurso crítico do sistema para o recurso em questão.

A teoria geral de redes apresentada neste capítulo será utilizada na construção do modelo conceitual do sistema, que será útil como uma linguagem comum de trabalho e também na descrição dos recursos do sistema e suas formas de utilização. Esta também será empregada na construção do modelo de recursos do sistema a ser utilizado na sua análise, que será realizada empregando-se o modelo-Q, conforme será monstrado a seguir.

4 Modelagem da Arquitetura da MFDM

Na construção de modelos de sistemas de informação deve-se definir inicialmente quais os aspectos a serem modelados. Assim, desejando-se obter a análise de desempenho de um sistema, faz-se necessário estudar o seu índice de utilização de recursos.

Para estudar o índice de utilização de recursos de um sistema é necessário obter primeiro o seu modelo conceitual, ou seja, uma descrição do comportamento da informação dentro do sistema. A partir deste construir-se-á um modelo de utilização dos recursos disponíveis, que será então avaliado para o estudo do desempenho do sistema.

4.1 Modelagem Conceitual

Na elaboração de um modelo conceitual, deve-se considerar os padrões de comportamento da informação no sistema, ou seja, suas características estruturais e dinâmicas. Na modelagem estrutural considerar-se-á quais caminhos físicos a informação dispõe para poder fluir. Na modelagem das características dinâmicas observar-se-á quais transformações a informação irá sofrer no seu percurso, assim como as influências dessas transformações na escolha do caminho pelo qual a mesma irá fluir.

Várias são as técnicas utilizadas para a criação de modelos conceituais de sistemas. Entre estas, a teoria geral de redes mostra-se muito interessante à modelagem de sistemas que apresentam comportamento concorrente entre seus componentes. Além disto, ressalte-se que a fundamentação matemática de que dispõe esta técnica facilita em muito o seu emprego na análise automática de modelos.

As técnicas da teoria geral de redes também se destacam pela possibilidade de desenvolvimento incremental do modelo, sendo possível desta forma a separação da modelagem das características estruturais e dinâmicas, facilitando a compreensão e a construção de modelos de forma hierárquica.

Não existe uma metodologia bem definida a ser empregada na criação de modelos de sistemas pela teoria geral de redes. Portanto, neste trabalho procurar-se-á empregar os sistemas formados pelas interpretações dadas a esta teoria, visando atingir um conjunto de regras implícitas que esbocem uma metodologia para a construção de modelos.

4.1.1 Modelagem Estrutural

Normalmente, iniciar-se-á pela modelagem estrutural de um sistema, passando-se depois à modelagem das suas características dinâmicas, que poderão ser incorporadas como um refinamento do modelo até então obtido.

Na modelagem estrutural é comum empregar-se, inicialmente, a interpretação de sistemas de condição/evento (sistemas C/E) atribuída a teoria geral de redes, considerando-se a informação como não diferenciável na modelagem dos caminhos, ligações entre eventos e condições, pelas quais a mesma fluirá no sistema. Nesta interpretação, as condições indicam a ocupação ou o acesso a uma unidade do sistema pela informação circulante, e os eventos indicam a passagem de uma unidade de informação, representada por uma marca, de uma ou mais unidades para as outras.

Os sistemas C/E possibilitam a descrição de seqüência e concorrência entre eventos no sistema. No entanto, devido à não diferenciação das marcas, os modelos resultantes geralmente apresentam conflitos tornando-se não-determinísticos. Esses conflitos correspondem a perdas ou ganhos de informação, que não são tratados pelo modelo, e sua solução normalmente depende de um observador externo. Para obter-se um modelo totalmente determinístico, necessita-se de uma interpretação com maior poder representativo, como os sistemas de lugar/transição ou de redes de alto-nível.

A informação perdida ou ganha nesses casos pertence às características dinâmicas do sistema. Por esse motivo, na modelagem estrutural do sistema muitos conflitos poderão deixar de ser tratados, o que virá a ocorrer no desenvolvimento do modelo de características dinâmicas.

O sistema a ser modelado, a máquina de fluxo de dados de Manchester, é composta de duas partes que operam em paralelo: um computador hospedeiro (***Host***), servindo de interface com o usuário, e um anel de fluxo de dados, funcionando como um *number cruncher* (**MFDM**), ambos conectados pela unidade de comunicação do anel (**SW**) que lhes permite a troca de mensagens. Neste trabalho será adotada a nomenclatura empregada no capítulo 2 e em [GUR80].

Modelo C/E

O interesse central deste trabalho está no anel de fluxo de dados. Assim sendo, tem-se como modelo inicial do sistema a Figura 4.1[1], onde são visíveis dois conflitos, ambos na condição que representa o processamento na **SW**: um na sua saída, pois uma vez processada uma ficha não é possível determinar se o resultado será dirigido para o ***Host*** ou para a **MFDM**; outro, na sua entrada, pois não é possível determinar qual informação dará entrada na **SW** em primeiro

[1] No desenvolvimento do modelo conceitual as transições que não contiverem incrições internas serão representadas por um retângulo fino, e totalmente preenchido.

lugar, caso exista processamento no *Host* e na **MFDM**. Em ambos os casos , há falta de informação.

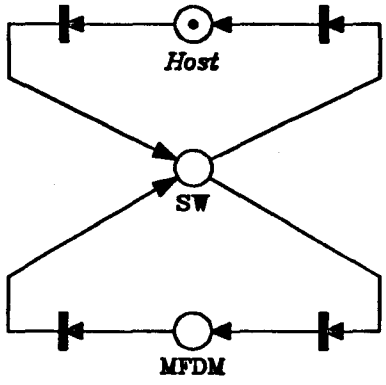

Figura 4.1: Modelo C/E inicial do sistema da **MFDM**.

Uma melhor compreensão do sistema requer o detalhamento dos módulos do anel da **MFDM**, que estão um nível hierárquico abaixo do descrito. O sistema neste novo nível apresenta um esquema com quatro novas unidades, arranjadas em *pipeline*. Este tipo de arranjo é caracterizado pelo encadeamento de ações, podendo ser modelado como uma seqüência de eventos conectados pelas condições de processamento nas unidades que o compõem, resultando no modelo da Figura 4.2.

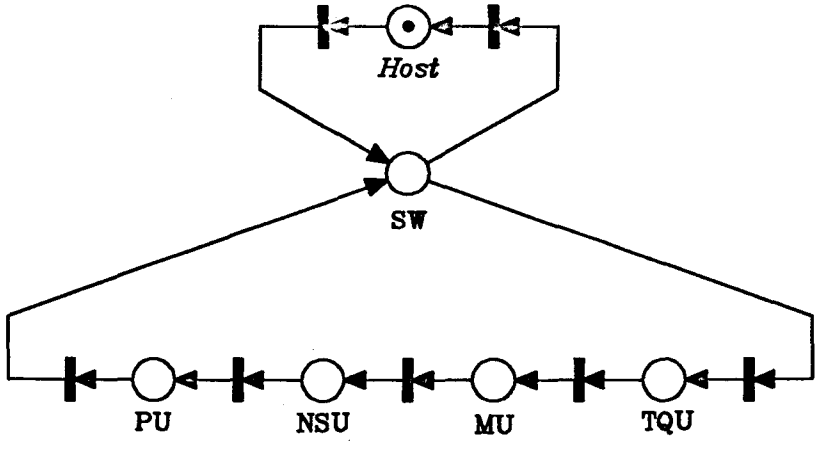

Figura 4.2: Modelo C/E detalhando o *pipeline* da **MFDM**.

Para que seja possível avaliar o comportamento estático da informação, dentro das unidades da **MFDM**, necessita-se de outro refinamento do modelo, atingindo-se um novo nível na sua hierarquia. Este refinamento procurará identificar caminhos obrigatórios, ou alternativos, que a informação deverá, ou poderá, seguir dentro destas unidades.

A primeira unidade da **MFDM** a ser detalhada será a **TQU**. Esta unidade é composta de quatro módulos ligados em *pipeline*, conforme o modelo da Figura 4.3. Um fato a ser observado é que, como o acesso à **TS** requer exclusão mútua, adicionou-se ao modelo um lugar de controle, **RWC**, que contém a marcação inicial de uma ficha. Como, pelas regras do sistema C/E somente

uma ficha pode ocupar um lugar por vez, obtém-se a requerida exclusão, pois somente um pedido de leitura ou de escrita na memória será aceito. Os lugares **SWC** e **SRC** modelam a diferença entre os locais de escrita e leitura, que ocorrem na cauda e cabeça da fila respectivamente. Esta diferença torna mais complexa a modelagem da exclusão mútua, que, de outra forma, iria requerer somente um lugar a mais.

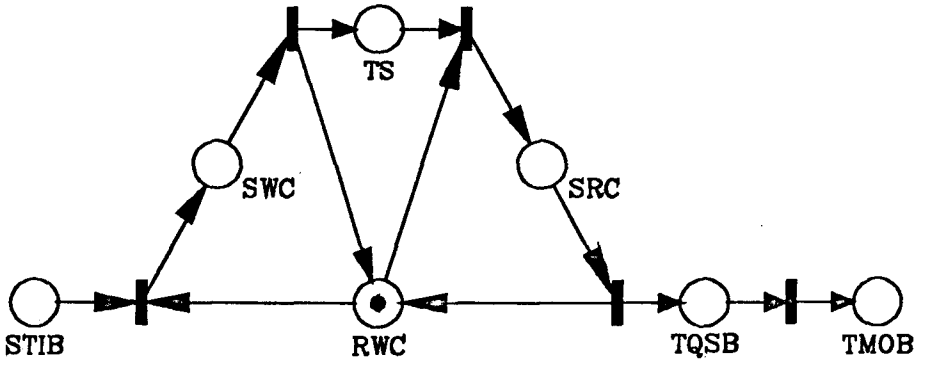

Figura 4.3: Modelo C/E detalhando a **TQU**.

É necessário dizer que o fato de a memória funcionar como um fila **FIFO** não será modelado realmente, devido ao detalhamento dessa característica encontrar-se em um nível de hierarquia diferente.

A **MSU** é composta de duas partes, a subunidade de *hashing* e a de memória, totalizando seis módulos arranjados em *pipeline*. Além destes módulos existe também a subunidade de tratamento de *overflow* de memória (**VU**) que opera em paralelo ao *pipeline*. O modelo final para este sistema encontra-se na Figura 4.4.

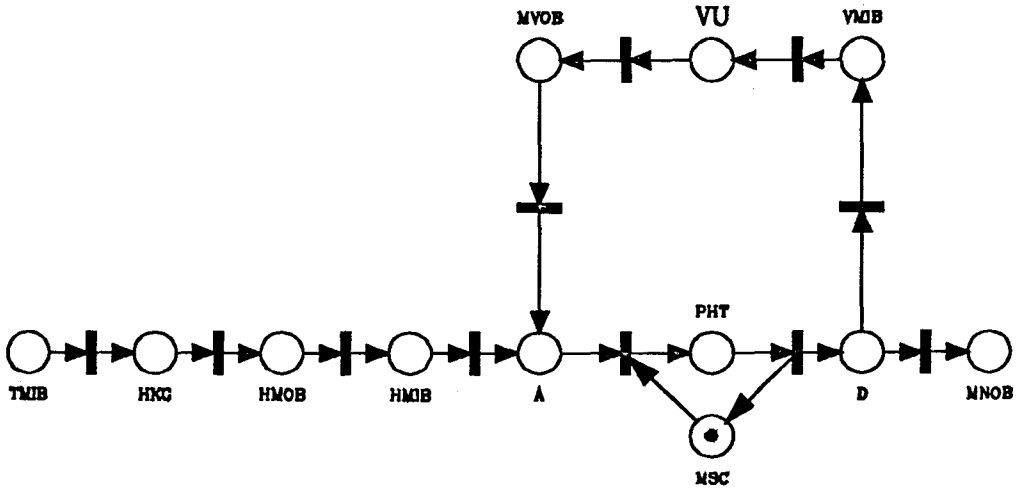

Figura 4.4: Modelo C/E detalhando a **MSU**.

Este modelo apresenta dois conflitos. O primeiro, pela necessidade de um arbitrador (**A**) entre a subunidade de *hashing* e a de *overflow* de memória para determinar qual informação entrará na memória primeiro: a que provém da subunidade de *hashing* ou a da **VU**. O segundo,

pela necessidade de um distribuidor na saída da subunidade de memória, que determinará para onde a informação se dirigirá: a **VU** ou a **NSU**.

Observa-se, também, que a modelagem de exclusão mútua nos acessos à **PHT** ocorre de uma forma mais simples que na **TS**, pois neste caso a ficha que faz acesso à memória carrega o respectivo endereço. Este fato não está aparente no modelo, pois o sistema C/E não diferencia as fichas que circulam na rede, e ficará mais claro quando da modelagem dinâmica do sistema. De qualquer modo, isto simplifica a modelagem da exclusão mútua, que não necessita de uma representação da armazenagem do endereço, como ocorre na **TQU**.

A unidade **NSU** é composta de quatro módulos arranjados em *pipeline*. Tanto a **ST** quanto a **NDS** requerem exclusão mútua em seu acesso, que neste caso também é facilmente modelado pois o endereço também será carregado pelo dado. O modelo desta unidade é mostrado na Figura 4.5.

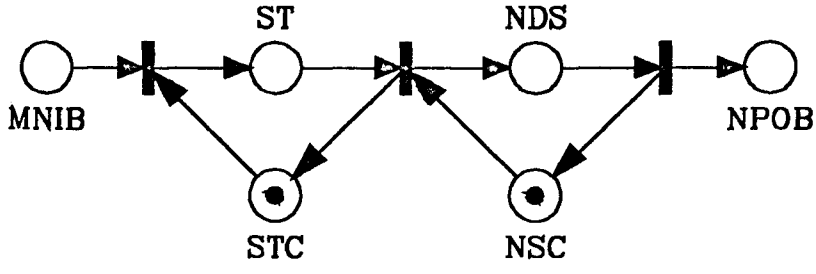

Figura 4.5: Modelo C/E detalhando a **NSU**.

A unidade **PU** difere das demais unidades em um aspecto: um de seus módulos apresenta submódulos que operam em paralelo. Seus seis módulos estão arranjados em *pipeline*, sendo que o módulo que realiza o processamento das instruções gerais é composto de várias **FUs** operando

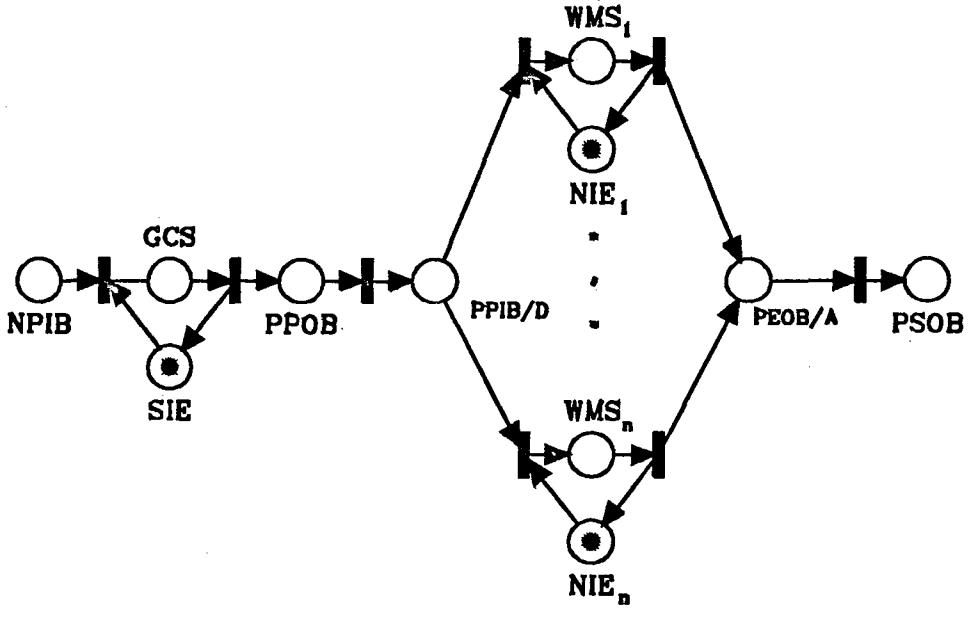

Figura 4.6: Modelo C/E detalhando a **PU**.

em paralelo. A Figura 4.6 mostra o modelo para esta unidade.

Novamente a existência de conflitos pode ser notada. Estes ocorrem na condição que modela o *buffer* de entrada para as **FUs**, que contém um distribuidor responsável por determinar para qual **FU** a informação será dirigida; e na condição que modela o *buffer* de saída das **FUs**, que contém um arbitrador responsável por determinar de qual **FU** sairá a informação que se destinará à **SW**. Note-se, também, que foi necessário modelar a exclusão mútua no acesso aos processadores, imposta pela sua memória de trabalho.

Como pode ser verificado, a partir dos casos apresentados, normalmente, a existência de conflitos modela a concorrência da informação por um recurso ou de um recurso por informação, e os conflitos estão ligados a regiões que operam em paralelo dentro do sistema. Esses conflitos, geralmente requerem, para sua resolução, informações que pertencem ao comportamento dinâmico do sistema.

Para que não existam diferentes níveis de hierarquia no modelo, faz-se também o detalhamento da parte do sistema formado pela **SW** e o ***Host***, conforme mostra a Figura 4.7, onde ficam evidentes os conflitos devidos ao arbitrador e distribuidor necessários à **SW**.

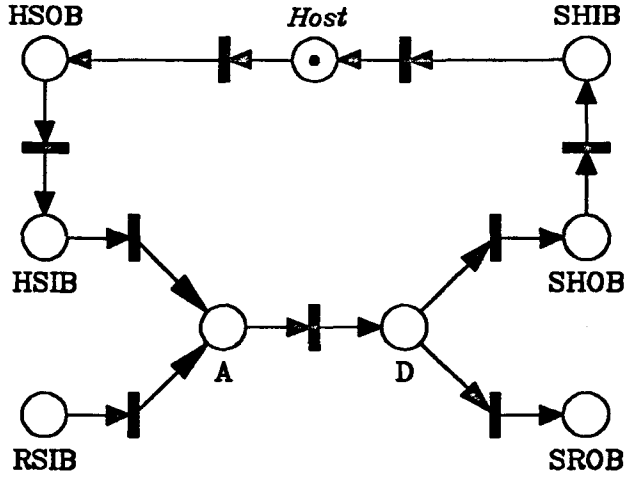

Figura 4.7: Modelo C/E detalhando a **SW**.

A ligação das sub-redes anteriormente descritas resulta em um primeiro modelo conceitual da **MFDM**, conforme mostrado na Figura 4.8. Apesar de ser um modelo completo, ele tem uma séria restrição imposta pelo sistema de interpretação empregado na modelagem. Ocorre que o sistema C/E não permite a ocupação de um lugar por mais de uma ficha. Desta forma, somente será possível a colocação de uma ficha inicial no ***Host***, o que foge à realidade. O mesmo ocorrerá com as memórias que somente poderão ter uma ficha por vez. Em suma, ter-se-á somente uma ficha circulando no sistema e, portanto, este não será representativo da realidade, ou seja, não será possível a ocupação concorrente de mais de uma unidade.

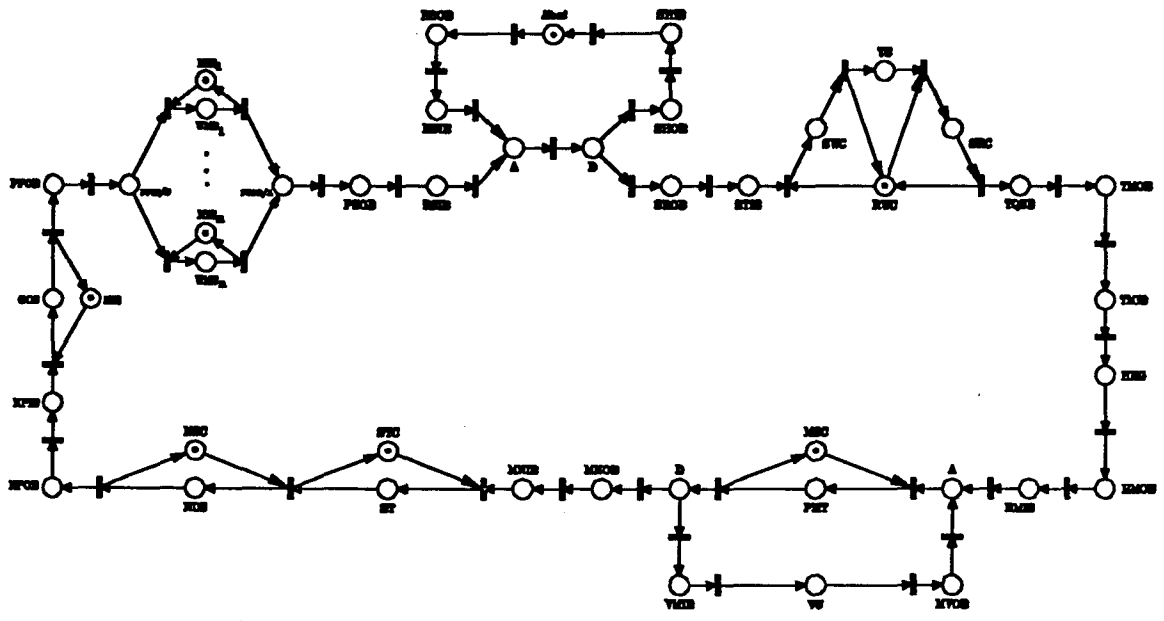

Figura 4.8: Modelo C/E do sistema completo da **MFDM**.

Modelo L/T

A solução para esse problema é a passagem para um sistema de interpretação mais poderoso. A mudança para o sistema de Lugar/Transição eliminará o problema, pois neste é permitido a um lugar conter mais de uma ficha. Nesta interpretação, um lugar também indicará a utilização de uma unidade do sistema, porém neste caso mais de uma requisição pode estar sendo atendida ao mesmo tempo. As transições terão o mesmo significado dos eventos nos sistema C/E. Estas modificações requerem algumas alterações no modelo anterior.

Primeiramente faz-se necessário definir a notação adotada para a indicação de capacidade dos lugares:

- lugares que não apresentarem indicadores de capacidade poderão conter, no máximo uma marca;
- lugares com indicadores de capacidade numéricos poderão conter, no máximo, um número de marcas igual ao valor dos respectivos indicadores;
- lugares com indicadores de capacidade w poderão conter um número indefinido de marcas.

Usando a notação acima têm-se as seguintes alterações necessárias para o modelo desenvolvido anteriormente:

a) O lugar denominado ***Host*** conterá inicialmente um número de fichas igual ao paralelismo médio do programa, simbolizado por **Pi**.

b) Os lugares **TS, PHT** e **VU** passarão a ter um indicador **w**, determinando que os mesmos são depósitos ilimitado, de marcas.

Além disto, deve ser ressaltado que:

- A utilização de indicadores **w** para os lugares **PHT, TS** e **VU** não invalida o modelo, mas simplesmente o simplifica. Poderiam também ser empregados indicadores com os valores reais dos tamanhos das respectivas memórias. No entanto, haveria o problema de alterá-los a cada modificação desses valores.
- O fato de os lugares **ST** e **NDS** não receberem um marcador **w** advém de que estes lugares modelam somente o acesso a estas memória e não a memória em si.

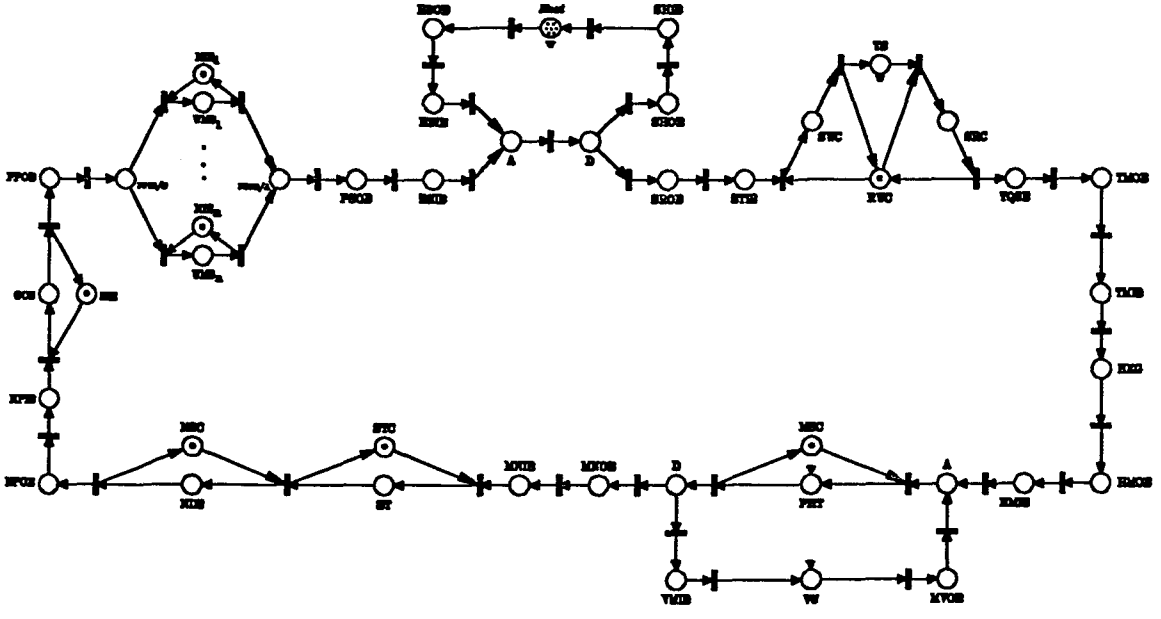

Figura 4.9: Modelo L/T do sistema completo da **MFDM**.

O modelo assim obtido permite uma maior aproximação com a realidade pois já é possível visualizar, quando da simulação de seu funcionamento, a ocupação concorrente das unidades e a variação do fluxo de fichas devida a sua retenção nas memórias. O modelo final pode ser visto na Figura 4.9, e terá, ao todo, considerando somente uma **FU**, 46 lugares e 43 transições, sendo acrescido de 2 lugares e 2 transições para cada **FU** adicionada. Este modelo, apesar de representar melhor a realidade do funcionamento da **MFDM**, ainda contém conflitos que só poderão ser solucionados de forma definitiva considerando-se as propriedades dinâmicas do sistema.

4.1.2 Modelagem do Comportamento Dinâmico

Nesta fase de modelagem do sistema considerar-se-ão as transformações que a informação poderá sofrer ao passar pelas unidades. Portanto, será necessário que diferencie as representações da informação, ou seja, será necessário levar em conta as diferentes formas que a informação será representada durante seu caminho pelo sistema.

A forma básica a ser observada na modelagem dinâmica do sistema será a ficha comum, representada por uma tupla <v,r,d>, onde: **v** corresponde ao valor da informação; **r** ao rótulo da informação e **d** ao destino da ficha (na **NSU**).

Com as fichas compostas de vários atributos, e sofrendo transformações nestes atributos nas unidades pelas quais irão passar, não será mais possível manter-se o sistema de interpretação L/T, pois neste as fichas não são diferenciadas. Assim sendo, será necessário empregar-se um sistema interpretativo para redes de alto-nível, tendo sido adotado neste trabalho o sistema **CEM** (*Condition/Event Modelling nets*) descrito no capítulo 3.

No sistema **CEM**, a interpretação dada aos lugares e transições será a mesma que no sistema L/T. Não se impedirá a ocupação de um lugar por mais de uma ficha, uma vez que não há colisões devido às fichas terem atributos diferentes,. porém, as regras de disparo de uma transição irão diferir, em parte, daquelas definidas no sistema L/T. A alteração mais significativa dá-se pelo fato de que, neste sistema, as transições poderão conter predicados internos que deverão ser satisfeitos para que ocorra a sua habilitação. Este fato apresenta duas grandes vantagens: primeiro, leva a redes mais compactas, o que facilita a descrição e compreensão dos sistemas modelados; segundo, possibilita a resolução de conflitos, pois é possível inscrever-se predicados mutuamente excludentes nas transições que estão envolvidas no conflito. Além disto, neste sistema, as arestas contêm inscrições que indicam o tipo das marcas que nelas podem fluir.

Para simplificar e ampliar a compreensão do modelo desenvolvido, recomenda-se indicar os atributos das marcas da forma mais resumida possível, utilizando somente o nível de detalhes necessário à descrição do modelo da unidade em questão. Assim, partir-se-á da notação <v,r,d> para a ficha comum, e se acrescentarão ou retirarão detalhes da mesma, conforme a necessidade.

Deve-se salientar, ainda, que será mantida a capacidade unitária nos lugares, pois eles normalmente representam *buffers* ou módulos com capacidade para uma única ficha. Também será mantida a notação adotada anteriormente para os lugares de maior capacidade ou ilimitada.

Modelo CEM

Antes de iniciar-se a modelagem dinâmica da **MFDM** algumas simplificações serão realizadas. São elas:

a) Não será considerada a unidade de *overflow* de memória da **MSU**, incorporada à **MFDM** para evitar o estouro da memória real. Em um modelo no sistema **CEM**, a capacidade de

um lugar pode ser considerada ilimitada, não se mostrando, portanto, necessária a modelagem desta subunidade num primeiro momento.

b) Serão consideradas somente duas das oito **mfs** possíveis para as fichas que entram na **MSU - EW** e **BY** -, que são as mais empregadas na construção de programas. Devido à proporção dos outros tipos de **mfs** ser muito pequena, essa decisão simplificará o modelo, sem acarretar perda de representatividade.

c) Considerar-se-á que as **FUs** geram somente resultados com zero, uma ou duas cópias. Apesar da geração de resultados múltiplos com endereços consecutivos, ser muito vantajosa, como foi descrito no capítulo 2, sua retirada numa primeira instância simplifica bastante o modelo e não causa grande prejuízo a sua representatividade devido à ocorrência de instruções deste tipo ser rara. Além disso, a maior parte dos dados a serem empregados na validação do modelo provêm de ensaios realizados quando esta facilidade ainda não estava disponível no protótipo ([GUR85A], [GUR83]).

d) Não será modelado neste trabalho o tratamento de programas que operem com estruturas de dados. Apesar de esta ser uma simplificação mais forte, ela acarreta em muito maior simplicidade para a construção do modelo, representando um risco aceitável inicialmente. Como conseqüência desta simplificação, também não será considerada a modelagem da **SSU**.

A eliminação das simplificações mais restritivas introduzidas no modelo é um dos possíveis temas para projetos subseqüentes, sugeridos no capítulo 6.

Iniciar-se-á a remodelagem das unidades pela **TQU**. Esta unidade não causa nenhuma transformação à informação sendo insensível aos novos atributos adicionais da ficha. Porém, é necessário adaptar o modelo ao uso da nova forma da ficha, ficando o modelo conforme a Figura 4.10.

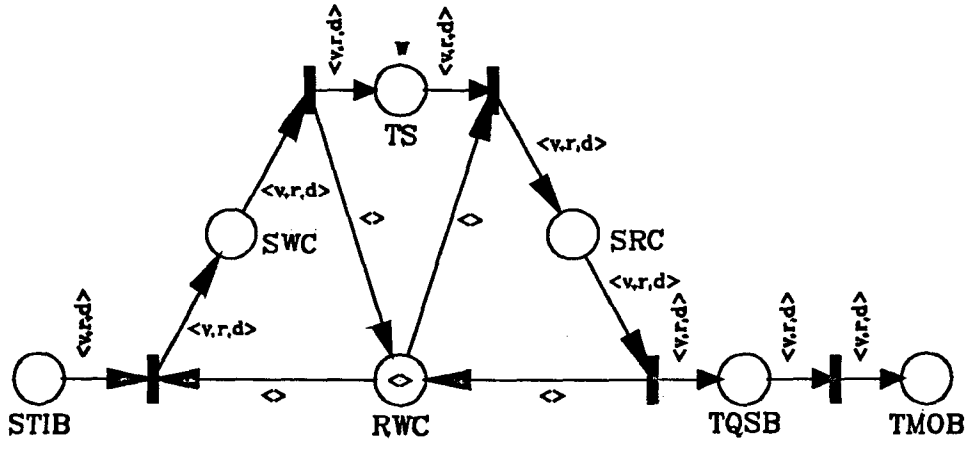

Figura 4.10: Modelo CEM detalhando a **TQU**.

Observe-se que a ficha da marcação inicial do lugar **RWC**, denotada pela marca <>, não conterá atributos indicando ser a mesma somente uma ficha de controle. O número de marcas iniciais deste tipo, no lugar **RWC**, limita o número de acessos simultâneos à memória desta unidade, pois elas são consumidas nas transições de acesso à memória e produzidas somente nas transições de liberação. Desta forma, manter-se-á o nível de exclusão mútua exigido. É importante observar ainda que, apesar de não alterar o caminho nem transformar o conteúdo das fichas, a exclusão mútua entre os acessos aos *buffers* de leitura e escrita nesta unidade provoca um atraso extra no processamento da informação.

A **MSU** transforma o pacote recebido, de modo a formar um outro mais complexo, denominado **grupo de fichas**. Esta transformação ocorre pelo acréscimo à ficha de um segundo valor a ser operado pela instrução na **PU**, representado por um novo atributo. Este acréscimo ocorre pela adição à ficha que chega de um valor padrão (***null***) ou pelo agrupamento de duas fichas destinadas a um mesmo endereço na **NSU**.

Dividir-se-á o modelo desta unidade basicamente em duas partes: o da subunidade de *hashing* e o da subunidade de memória. O modelo da subunidade de *hashing* pode ser visualizado na Figura 4.11 que apresenta o modelo simplificado para a **MSU**.

Um fato a ser observado é que a geração do endereço de *hashing* foi modelada através da combinação do lugar **HKG**, que denota o *buffer* gerador do endereço, com a inscrição de uma fórmula na transição que utiliza sua ficha. Nesse caso, a avaliação da fórmula representa uma ação da transição. Note-se também que, a partir desta transição, a ficha passa a carregar mais informação devido ao atributo nela inserido, correspondente ao resultado do *hashing*.

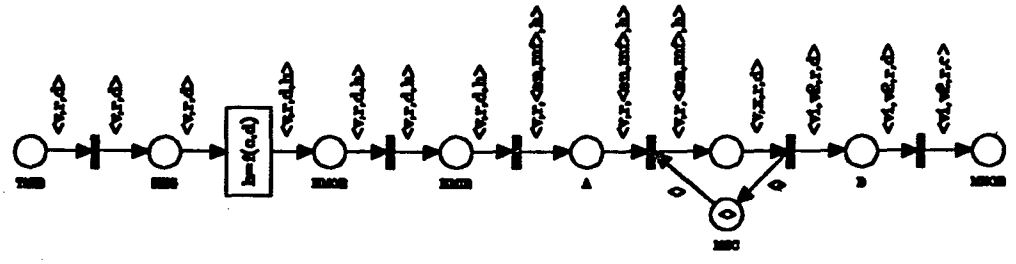

Figura 4.11: Modelo CEM detalhando a **MSU**.

A aplicação de uma fórmula, na forma de um predicado inscrito em uma transição, aos atributos das fichas que passam por esta transição, é a forma de modelar-se a geração de valores dependentes dos atributos de uma ficha. Estes valores podem ser empregados para alterar os valores de atributos já existentes, ou para criar novos atributos.

O aumento ou redução do número de atributos da ficha é modelada modificando-se o número de atributos associados à aresta de saída da transição em relação à de entrada.

O comportamento da subunidade de memória pode ser classificado inicialmente em função da necessidade ou não de acesso à **PHT** e, então, em caso de necessidade, da presença ou

não de uma ficha armazenada na mesma, com mesmo contexto daquela que chegou. É possível verificar a necessidade de acesso à **PHT** simplesmente avaliando a **mf** da ficha que chega a esta subunidade. Desta forma, primeiramente será necessário detalhar os atributos da ficha até o nível em que seja possível a avaliação da **mf**

O atributo que contém a **mf** corresponde ao destino da ficha, e é composto por: **en**, o endereço da instrução para o qual a ficha se dirige; e **mf**, a *matching function* da ficha. Nesse nível de detalhe a ficha é representada por: **<v,r,<en,mf>>**.

Portanto, o modelo desta subunidade de memória será dividido, inicialmente, em duas partes: uma concernente às fichas que não necessitam de acesso à **PHT**, e outra concernente às fichas que precisam deste acesso. O modelo da subunidade de *hashing* por ser comum a ambas as partes, não será apresentado visando a simplificação da sub-rede resultante.

Iniciar-se-á a modelagem da subunidade de memória pela parte referente ao tratamento de fichas que não necessitam de acesso à **PHT**. Estas fichas tem **mf** do tipo ***BYpass***, e, com a adição de um operando padrão do tipo ***null***, compõem o grupo de fichas como demonstrado na Figura 4.12. Não há acesso à memória e, portanto, o lugar que representa a **PHT** não aparece ligado a transição alguma.

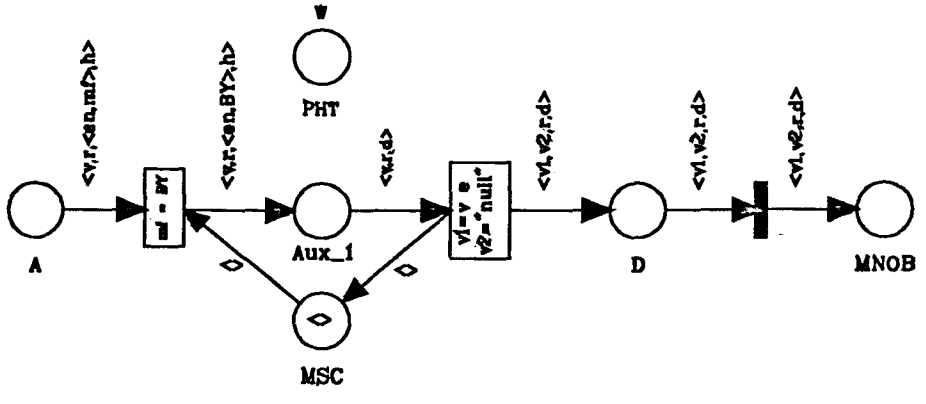

Figura 4.12: Modelo CEM detalhando o acesso à memória da **MSU** para as fichas que sofrem *BYpass*.

Observe-se que, a transição de entrada nesta sub-rede contém como inscrição o predicado lógico **mf = BY**, indicando que somente fichas com esse atributo poderão habilitá-la e dispará-la. Desta forma, apesar da existência da outra sub-rede, iniciada no lugar que modela o arbitrador, não haverá conflitos e será possível escolher entre estas através dos predicados inscritos em suas transições de entrada. O conflito que antes existia no arbitrador foi eliminado pela exclusão da modelagem da **VU**. O lugar auxiliar **MSC** desta sub-rede terá a mesma função anteriormente descrita para o sistema L/T.

A habilitação condicional de cada transição, sujeita à satisfação do predicado correspondente, é a forma utilizada pelas redes de alto-nível para evitar conflitos dentro do

modelo. Portanto, é desnecessário enfatizar a importância da seleção adequada do predicado a ser inserido em cada caso, para a corretude do modelo.

De acordo com a restrição aceita neste trabalho, todas as fichas que necessitam de acesso à **PHT** tem **mf** do tipo *Extract-Wait*. Essas fichas apresentam comportamento dependente da presença ou não de sua parceira na **PHT**, ficando o seu tratamento dividido em duas classes:

a) se a parceira estiver presente, ela será extraída e as fichas formarão um grupo de fichas, conforme a Figura 4.13;

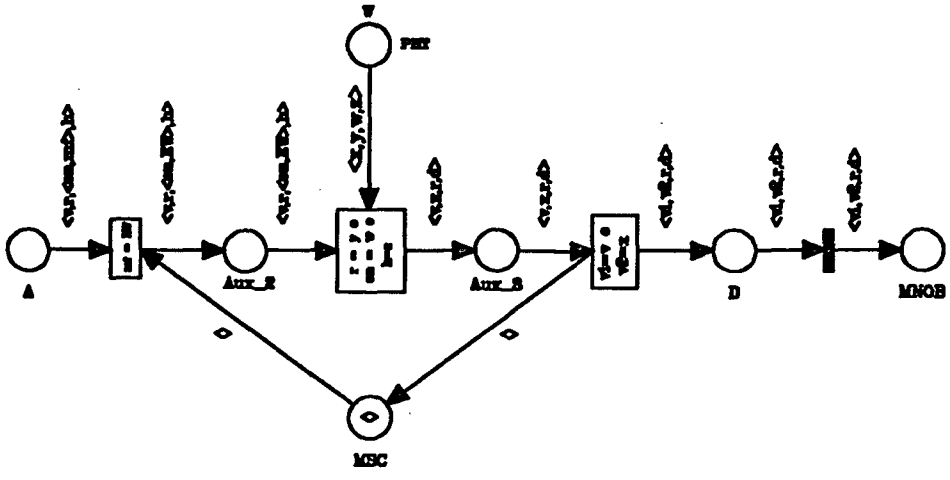

Figura 4.13: Modelo CEM detalhando o acesso à memória da **MSU** para fichas que sofrem *Extract*.

b) se a parceira não estiver presente, a ficha de entrada será escrita na memória, não sendo produzido qualquer resultado conforme a Figura 4.14.

A consulta à memória é modelada pelos predicados presentes nas transições ligadas ao lugar **PHT**, e pelas inscrições contidas nas arestas que realizam essa conexão.

No modelo desenvolvido esta consulta ocorre em duas ocasiões:

a) Quando o teste da existência de ficha parceira na **PHT** é satisfeito. Neste caso a satisfação do predicado implicará no disparo da transição e a retirada da única ficha que o satisfez da **PHT**.

b) Quando o teste de não existência de ficha parceira na **PHT** é satisfeito. Neste caso a satisfação do predicado também implicará no disparo da transição; porém, agora, todas as fichas selecionadas pela inscrição da aresta serão devolvidas à **PHT**. A devolução decorre do uso de uma aresta restadoradora, que não destrói as fichas do lugar quando de sua consulta. Além disso, a ficha que está na outra aresta de entrada da transição será depositada na **PHT**.

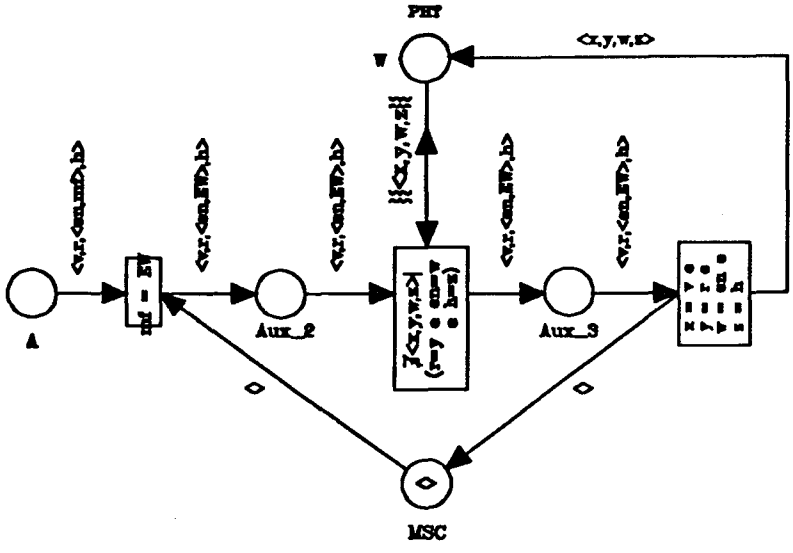

Figura 4.14: Modelo CEM detalhando o acesso à memória da **MSU** para as fichas que sofrem *Wait*.

Essa representação torna visível outra maneira de descrever características do sistema nestas redes: a possibilidade de, através de uma inscrição associada a uma aresta, selecionar um número arbitrário de fichas, dentre um conjunto presente em um lugar.

Um fato a ser notado é que, apesar da pesquisa aos diversos bancos da **PHT** ser realizada simultaneamente, este fato não pode ser representado de forma direta neste modelo, por estar em outro nível hierárquico.

A **NSU** transforma a informação adicionando à mesma os seguintes atributos: **i**, correspondente ao código da instrução; **d1** correspondente ao endereço do resultado 1; e **d2** correspondente ao endereço do resultado 2, que é opcional. Estes atributos indicarão à **PU** como processar a informação e quantos resultados gerar. A adição dos atributos à ficha é realizada por um *pipeline* de três fases :

a) a primeira fase busca o valor de **base** do endereço real da instrução na **ST**;

b) a segunda compõe o endereço real, adicionando ao valor base o valor do deslocamento (*offset*) e o envia ao *buffer* intermediário;

c) a terceira faz a busca da instrução na **NDS**, empregando o endereço gerado na fase anterior e compõe um **pacote executável**.

A descrição acima representa a idéia original da **NSU**. Na implementação atual da **MFDM** a fase de busca do valor de base e a composição do endereço estão incorporadas numa única fase, como mostra a Figura 4.15. Observa-se, novamente, o emprego de arestas restauradoras. Isto ocorre devido ao fato de os lugares a **ST** e a **NDS** serem apenas consultados, não ocorrendo, portanto, consumo de suas marcas.

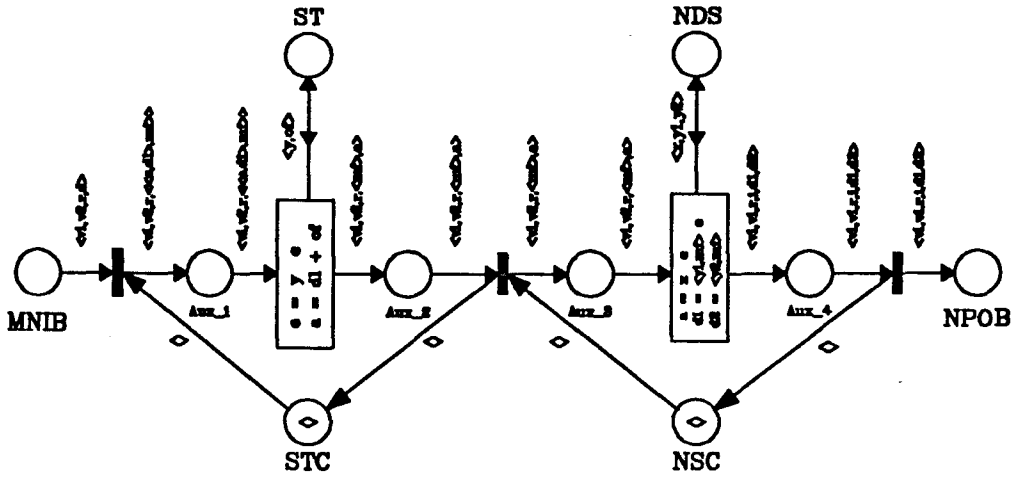

Figura 4.15: Modelo CEM detalhando a **NSU**.

A **PU** recebe um pacote executável e produz fichas comuns, fazendo com isto a transformação reversa das unidades anteriores. Desta forma, esta unidade transforma a informação, enquanto também provoca uma alteração no fluxo produzindo, em função da instrução executada, mais ou menos fichas que as recebidas

De uma forma geral, esta unidade pode gerar zero, uma ou duas fichas de saída para cada pacote processado. A determinação do número de resultados gerados depende diretamente:

- do número de endereços válidos que o pacote executável contém;
- da instrução a ser executada e;
- dos valores que o pacote carrega.

Desta forma, a modelagem exata da determinação do número de resultados gerados levaria a um modelo extremamente complexo, necessitando de níveis hierárquicos adicionais.

Visto que a principal diretriz deste trabalho é a obtenção de um modelo de fácil compreensão e aplicação, optou-se por considerar possível separar as instruções em conjuntos, diferenciados pelo número de resultados gerados, sem adicionar outros níveis à hierarquia da modelagem. Apesar de estes conjuntos não serem necessariamente disjuntos, a simplificação não irá acarretar perda de representatividade ao modelo, uma vez que na construção do modelo de recursos, onde este modelo conceitual será utilizado, trabalhar-se-á com valores percentuais em vez de absolutos. Desta forma, há três tipos de comportamento possíveis para esta unidade:

a) Instruções classificadas como geradoras de zero **fichas-resultado** são modeladas conforme a Figura 4.16, onde os pacotes executáveis são processados de forma mutuamente exclusiva. Somente é produzida a marca controladora da exclusão mútua, indicando que os valores do pacote executável foram consumidos.

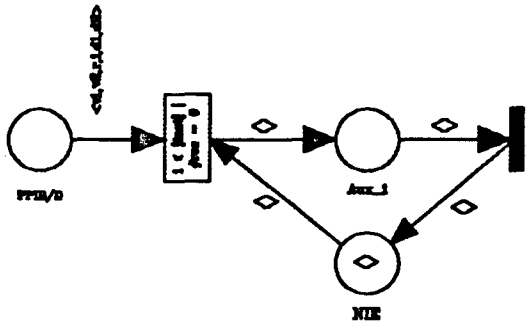

Figura 4.16: Modelo CEM detalhando a geração de zero fichas-resultado pela **FU**.

b) Instruções classificadas como geradoras de uma ficha-resultado utilizam-se somente do primeiro endereço do pacote executável, conforme a Figura 4.17. A ficha-resultado e a ficha controladora de exclusão mútua são produzidas simultaneamente.

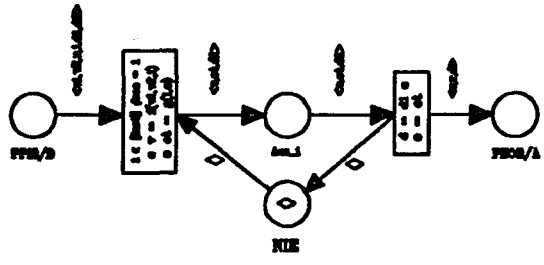

Figura 4.17: Modelo CEM detalhando a geração de uma ficha-resultado pela **FU**.

c) Instruções classificadas como geradoras de duas fichas-resultado com endereços diferentes, somente liberam a **FU** da exclusão mútua após terem produzido seus dois resultados, conforme a Figura 4.18.

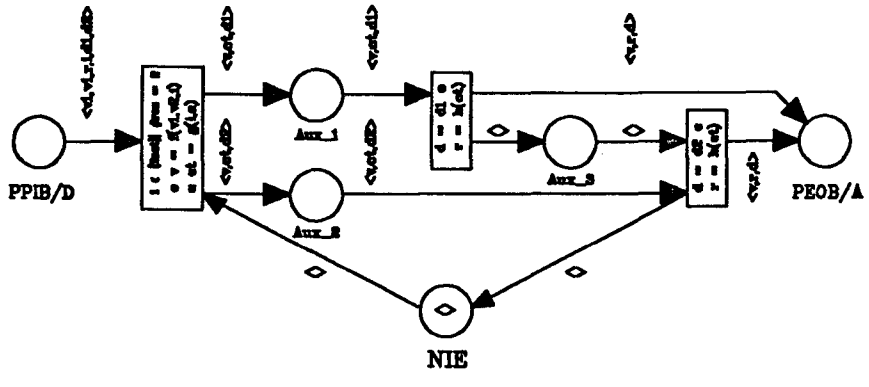

Figura 4.18: Modelo CEM detalhando a geração de duas fichas-resultado pela **FU**.

O modelo final para a **PU** poderá ser visualizado na Figura 4.19. A única peculiaridade do modelo são as **FUs**. Porém, como no sistema **CEM** as fichas são diferenciadas pelos seus

atributos, não há necessidade de ter-se uma sub-rede para cada **FU**. Desta forma, o número de **FUs** será representado pela marcação inicial do lugar **NIE**, responsável pela exclusão mútua de suas memória internas de trabalho. Assim, a mesma rede será utilizada por um número fixo, igual ao número de **FUs** existentes, de pacotes executáveis diferentes tornando o modelo mais compacto. Cita-se também que o módulo do **pré-processador** não será detalhado pois seus detalhes encontram-se em outro nível de hierarquia.

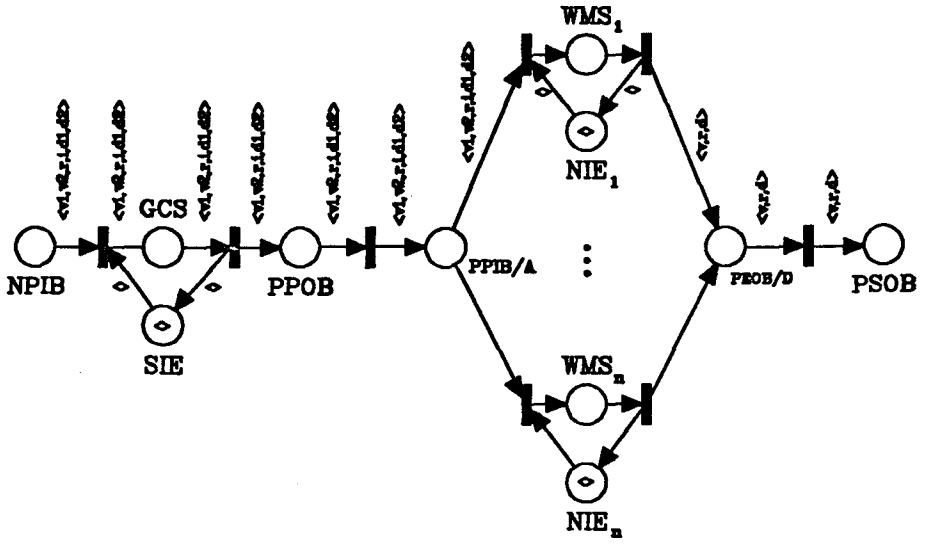

Figura 4.19: Modelo CEM detalhando a **PU**.

Observa-se que o conflito inicial na entrada das **FUs** está parcialmente resolvido, pois somente poderão disparar as transições de entrada correspondentes às **FUs** habilitadas para processamento. A escolha de qual **FU** irá disparar não precisa ser modelada, visto que a capacidade de processamento é sempre a mesma. O conflito na saída das **FUs** não tem solução, uma vez que o lugar que receberá as fichas comuns é um *buffer* de tamanho um, que desta forma somente poderá atender a uma **FU** por vez. Isto também indica que este pode vir a ser um futuro ponto de gargalo do sistema.

Para finalizar a modelagem dinâmica da **MFDM**, realizar-se-á a adaptação do modelo da **SW**, conforme mostra a Figura 4.20. Esta unidade, assim como a **TQU**, não provoca transformações na informação, ela simplesmente se utiliza do atributo **d** para determinar para onde a informação será dirigida, ao ***Host*** ou de volta ao anel, resolvendo assim o conflito do distribuidor. O conflito do arbitrador é resolvido atribuindo prioridade às fichas vindas do anel, um fato que não será diretamente modelado.

Isto conclui a modelagem dos aspectos dinâmicos da **MFDM**. O modelo apresentado descreve de forma detalhada o fluxo e as transformações que a informação sofre quando da sua passagem pelas subunidades do sistema.

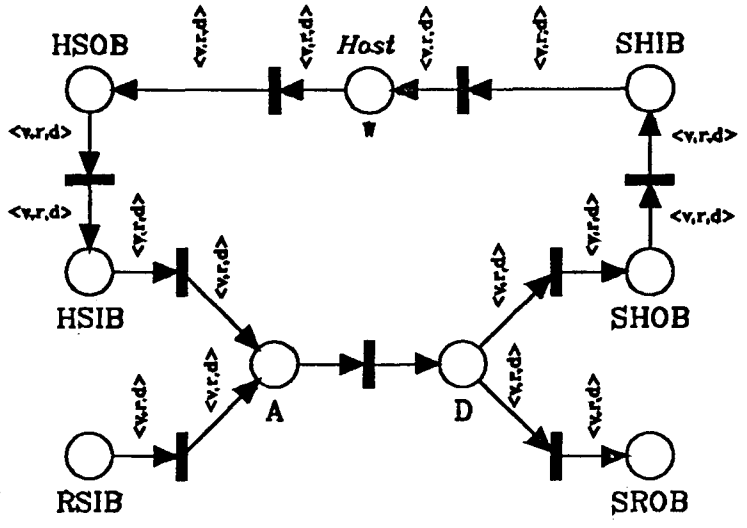

Figura 4.20: Modelo CEM detalhando a SW

Devido ao nível de detalhes alcançado por esse modelo conceitual, torna-se muito complexa sua utilização direta para análise de desempenho do sistema. Assim sendo, faz-se necessário o desenvolvimento de um modelo com nível de abstração mais elevado, que será construído com base nas informações contidas no modelo conceitual. Esse novo modelo considerará somente os padrões de utilização da informação e os caminhos pelos quais ela pode fluir, deixando de lado suas transformações estruturais.

Desta forma, o modelo final terá seu tamanho e complexidade bastante reduzidos, tornando possível um estudo analítico. A construção do modelo de recursos do sistema é descrita na seção seguinte, apresentando-se as simplificações e considerações realizadas na sua elaboração.

4.2 Modelagem da Utilização de Recursos

Na elaboração do modelo de utilização de recursos de um sistema, faz-se necessária a correta seleção dos recursos em cada unidade, assim como dos parâmetros a serem medidos e analisados. Por correta seleção entende-se que a escolha destes elementos deve levar à criação de um modelo que tenha bom equilíbrio entre as capacidades de representação e análise, fatores normalmente antagônicos.

Portanto, analisar-se-á o modelo conceitual do sistema buscando determinar primeiramente quais parâmetros são conhecidos e quais irão ser avaliados. Neste sistema, tem-se conhecimento dos seguintes parâmetros:

- número de unidades funcionais e sua estrutura interna;
- tempo médio de permanência das fichas nas unidades;
- tempo médio de execução de cada grupo de instruções;

- tamanho máximo das memórias das unidades;
- grau de paralelismo médio do programa;
- máximo e mínimo paralelismo possível no programa;
- percentagem de fichas portando cada **mf** no programa;
- tamanho médio do código de um programa;

através dos quais pretende-se obter os seguintes resultados:

- subutilização da **PU** para *n* **FUs** ativas;
- subutilização das subunidades **MSU** e **NSU**;
- desempenho final do sistema;
- obtenção dos pontos de gargalo do sistema.

Definido o conjunto de parâmetros, iniciar-se-á a determinação dos recursos da **MFDM**. Na elaboração de um modelo de recursos, considera-se inicialmente como recurso do sistema todo o objeto ao qual pode ser alocada uma tarefa dentro do sistema. Neste caso, numa primeira instância, ter-se-á todos os módulos que compõem as unidades da **MFDM** como recursos deste sistema.

Definido o conjunto de recursos, resta a determinação dos parâmetros que poderão ser empregados na análise. Portanto, tomando como primeira unidade a **TQU**, ter-se-á como parâmetros: o número de módulos da unidade e sua composição; o tempo de permanência médio de uma ficha em cada módulo, que neste caso será independente de composição da ficha, e o tamanho da sua memória, a **TS**.

A **MSU**, como mostra o modelo conceitual, realiza uma transformação na ficha requerendo que se considere a distinção existente entre as duas fases pelas quais ela passa nesta unidade, a saber: a fase de cálculo do número de *hashing* e fase a de acesso à memória. Isto se faz necessário devido a que, na fase de cálculo do número de *hashing*, o comportamento da unidade é independente da composição da ficha, o mesmo não ocorrendo na fase de acesso à memória. Ainda é necessário considerar que a unidade provoca uma diminuição no fluxo de informação, causada pelas fichas que ficam retidas na sua memória por terem **mf** igual a **EW** e não encontrarem ficha parceira à sua espera.

Assim, na primeira fase serão considerados como parâmetros: o número de módulos da fase, sua composição e o tempo médio de permanência da ficha nos módulos. Já na segunda fase, faz-se necessário considerar as diversas formas de posse da memória que ocorrem devido às ações das diferentes **mfs** das fichas que dão entrada na **MSU**. Desta forma, ter-se-á como parâmetros: o número de módulos da fase e sua composição; a percentagem de fichas de entrada

para cada mf considerada; o tempo de permanência nos módulos da fase para cada **mf** considerada; e o tamanho da sua memória, a **PHT**.

Apesar de a **NSU** transformar a informação, esta transformação independe do grupo de fichas que dá entrada na unidade, de forma que isto não acarreta tratamento especial para sua modelagem. Considerar-se-a para esta unidade somente a sua implementação atual, que é composta de fase única. Portanto, tem-se como parâmetros para a unidade: o número de módulos da unidade e sua composição; o tempo médio de permanência do grupo de fichas nos módulos; e o tamanho de suas memórias, a **ST** e a **NDS**.

A **PU** também transforma a informação e, como essa transformação depende do tipo de pacote executável recebido, faz-se necessário considerar em separado as duas fases da passagem do pacote pela unidade, a saber: pré-processamento e passagem pela **FU**, e a percentagem de fichas de cada tipo de instrução que são processadas.

Na primeira fase ter-se-á como parâmetro: o número de módulos da fase, sua composição e o tempo médio de permanência do pacote executável. Já na segunda fase deve-se levar em conta os diferentes grupos de instruções que são processados, pois os mesmos, além de terem tempos de processamento diferentes, geram volume de resultados diferentes. Desta forma, considerar-se-ão como parâmetros: o número de módulos e sua composição, e o tempo de permanência dos pacotes nos módulos.

Observa-se ainda que deve ser levada em consideração itens como:

- o número de **FUs**, pois estas são as unidades que efetivamente transformam a informação e, portanto, são o principal alvo de estudo;
- os tempos individuais para cada tipo de pacote processado nas **FUs**;
- as alterações no fluxo de informação causado por esta unidade, alterações estas que podem provocar: a diminuição do fluxo, devido às instruções que não produzem fichas-resultado; ou o aumento do fluxo, devido às instruções que produzem duas fichas-resultado.

Isto completa o estudo do modelo conceitual, na busca de uma definição dos candidatos a recursos e parâmetros no modelo de recursos. Resta definir a técnica de análise a ser empregada, verificando-se então a facilidade ou dificuldade apresentada pelo modelo resultante. Dentre as possíveis técnicas de estudo foi escolhida para este trabalho o emprego da Teoria Geral de Redes através do emprego de uma metodologia analítica. Esta escolha foi certamente influenciada pela facilidade de análise que estas redes apresentam, mas também visa manter a homogeneidade das ferramentas empregadas.

Dentre as propostas consideradas para este tipo de análise ([TAZ88], [HOL87], [MAR84]) escolheu-se a metodologia desenvolvida por Tazza [TAZ88]. Essa metodologia está

baseada na avaliação de um modelo de redes de Lugar/Transição restrito. As restrições impostas são necessárias para que se evitem problemas que poderiam impossibilitaram a análise, como, por exemplo, bloqueios perpétuos. A técnica foi aprimorada por Fernandes [FER90] para o tratamento de problemas onde ocorrem bloqueios na posse de um recurso, levando a uma metodologia aproximativa na resolução da análise.

Na definição desse último modelo, observa-se que o número de recursos e de parâmetros obtidos é muito grande, 30 recursos e 70 parâmetros sem levar em conta os diferentes tempos de execução de cada instrução, o que torna uma implementação direta intratável analiticamente. Para resolver este problema, é necessário que se façam algumas simplificações, procurando diminuir o número de recursos e parâmetros, porém sem grande prejuízo à representatividade do sistema.

Adotar-se-ão desta forma as seguintes simplificações:

a) Considerar-se-á como recursos a unidade como um todo e não seus módulos individuais. Isto não causa perda de representatividade; somente aumenta o nível de hierarquia para o qual se conseguirá obter resultados. No entanto, o número de recursos diminui para no máximo 5, tornando viável a análise de modelo.

b) Considerar-se-á que as instruções têm o mesmo tempo médio de execução. Isto provoca uma pequena perda de representatividade, que é compensada pelo ganho no tempo de análise resultante.

c) Tomar-se-ão como parâmetros, basicamente: o tempo médio de permanência das fichas nas unidades; a percentagem de fichas com **mf** igual a **BY**, considerando que o restante das fichas têm **mf** igual a **EW**, com ações de *Extract* e *Wait* igualmente distibuídas; o número de **FUs**; e o paralelismo médio do programa. A hipótese de que o paralelismo médio do programa é representativo da influência do programa no desempenho do sistema é aceitável, como foi demostrado por Gurd [GUR83].

d) Considerar-se-á que o sistema se encontra em regime estacionário de trabalho, com paralelismo médio e constante igual a **PI**, ou seja, a carga de trabalho do sistema não sofrerá as flutuações associadas ao início ou finalização de um programa. Este fato não causa grandes perdas de representatividade ao modelo, pois é comum considerar-se somente o estágio em que o processamento mostra-se estável quando se trata de análise de desempenho.

e) Considerar-se-á também, numa primeira instância, que a variação do fluxo na **MSU** e na **PU** compensam-se mutuamente, ou seja, o total de fichas consumidas será igual ao de fichas produzidas no sistema, não havendo, portanto, variação de fluxo. Apesar desse fato acarretar uma pequena perda de representatividade para o modelo, a hipótese é válida como uma forma de simplificação inicial para acelerar a análise, podendo ser retirada posteriormente, caso seja necessário.

Uma vez definidas as simplificações e pesadas as suas influências na representatividade do modelo, pode-se passar a construção do modelo final que agora terá apenas 5 recursos e 7 parâmetros. Estes números são bem menores que os encontrados anteriormente, tornado possível a aplicação de uma solução analítica.

A metodologia adotada na construção do modelo requer que se defina primeiramente uma **cadeia aberta**, que será composta de lugares de utilização e de espera, ligados por transições intermediárias. Os primeiros são responsáveis pela modelagem do tempo de execução de uma tarefa, que requer um certo número de recursos; os últimos estão sempre associados aos recursos existentes, ou seja, para cada recurso haverá um local de espera antecedendo sua entrada, modelando assim a espera pela liberação do recurso. Dessa forma, teremos um modelo com 5 lugares de utilização representando as tarefas executadas em cada unidade da **MFDM**, e com 5 lugares de espera, um para cada recurso, conforme pode ser visto na Figura 4.21.

Cabe citar que o primeiro e último lugares da cadeia aberta serão sempre lugares de espera onde o primeiro representa um número ilimitado de solicitações de tarefas a serem atendidas e o último um lugar de espera indefinida para as tarefas já realizadas.

Definida a cadeia aberta, é necessário definir as subcadeias associadas aos recursos do sistema. As subcadeias são compostas da transição de alocação do recurso a que a mesma está associada; dos lugares que representam a execução das subtarefas que requerem aquele recurso,

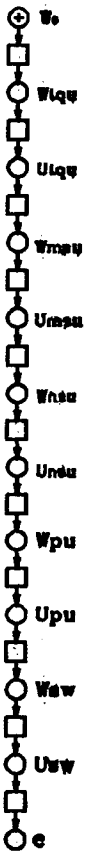

Figura 4.21: Cadeia aberta composta pelas unidades da **MFDM**.

assim com o das transições que os separam; e da transição de liberação do recurso. Esta definição delimitará o uso de recursos pelas tarefas, sendo que a cada recurso estará associada uma subcadeia.

Os recursos serão modelados como lugares que estarão ligados às transições de alocação e liberação, e que conterão como marcação inicial marcas em número igual ao número de recursos disponíveis no sistema. Assim, a estrutura do modelo da **MFDM** ficará como demonstrado na Figura 4.22.

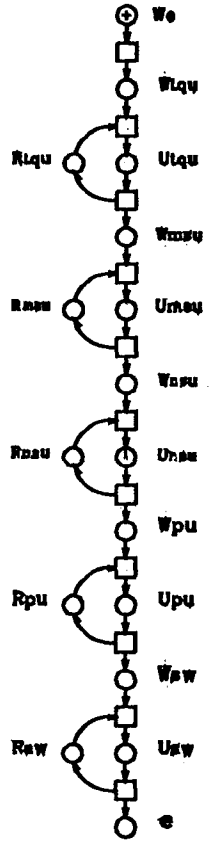

Figura 4.22: Cadeia aberta com os recursos correspondentes às unidade da **MFDM.**

Dado que o sistema tem o seu desempenho dependente do grau de paralelismo que o programa que está sendo processado apresenta, será necessária a inclusão de um recurso que represente esse comportamento. Este recurso será definido como a subcadeia principal do sistema, e englobará todas as tarefas menores executadas nas unidades. Ele deverá ter sua transição de alocação conectada ao primeiro lugar de espera da cadeia aberta e sua transição de liberação conectada ao último lugar desta cadeia, sendo sua marcação inicial igual **PI**; o paralelismo médio do sistema. O desempenho do sistema será limitado por este parâmetro, conforme ocorre na realidade, ficando a estrutura final do modelo do sistema como na Figura 4.23.

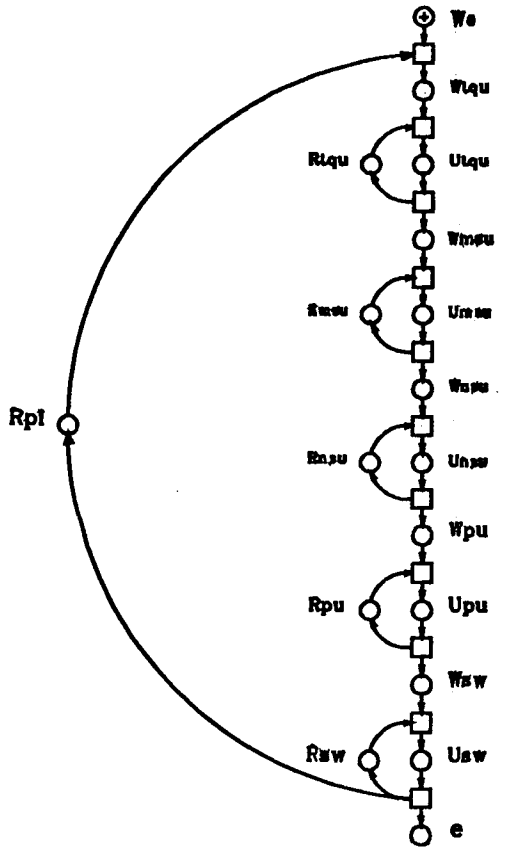

Figura 4.23: Modelo-Q1 com todos os recursos correspondentes às unidades da **MFDM**.

Antes de se dar o modelo por concluído, deve-se considerar o fato de que é necessário modelar a existência de diferentes tempos de acesso à **MSU**. Isto é necessário, mesmo com suas duas fases sendo fundidas, pois a segunda fase apresenta tempos diferentes para sua execução, que devem ser refletidos pelo modelo. Este fato pode ser inserido no modelo através da ponderação do tempo de permanência no lugar de utilização correspondente à tarefa de passagem pela **MSU**. Isto irá gerar três redes distintas que serão combinadas conforme a metodologia descrita por Fernandes [FER90].

Além dos diferentes tempos de acesso à **MSU**, também deve-se considerar a diminuição do fluxo que esta unidade causa na **NSU** e na **PU**. A modelagem deste fato pode ser realizada da mesma forma que o anterior gerando duas redes a mais para cada unidade. Na primeira rede será considerado o tempo de permanência como nulo e a segunda será considerado o tempo normal. A combinação destas redes (três para a **MSU**, duas para a **NSU** e duas para a **PU**) irá gerar 12 redes distintas que comporão o modelo de recursos da **MFDM**. Como cada análise deste modelo irá requerer doze avaliações separadas da rede, e a posterior combinação das mesmas para a obtenção do resultado final, é interessante simplificar ainda mais o modelo, procurando com isto facilitar a análise.

Uma simplificação do modelo, que a priori não trará perda de representatividade, será considerar alguns recursos como subtarefas do sistema. Dessa forma, estes recursos não terão

subcadeias nem lugar de espera., reduzindo o volume de dados gerados em cada análise, pois não serão computados para os mesmos parâmetros como desempenho, tempo de espera para atendimento, etc. Por outro lado, com esta simplificação poder-se-á perder a visualização de recursos que estejam bloqueando o desempenho do sistema, um fato que poderá facilmente ser verificado com a reinclusão do recurso. A retirada e a inclusão de recursos serão realizadas conforme a necessidade de detalhe da análise.

Observando-se a arquitetura da **MFDM** pode-se concluir que a transformação da **TQU** e da **SW** em subtarefas não trará prejuizo ao modelo, pois a primeira unidade não é primordial ao funcionamento da máquina e a segunda pode facilmente ter sua taxa de processamento alterada. Desta forma o modelo final, simplificado, ficará conforme o descrito na Figura 4.24.

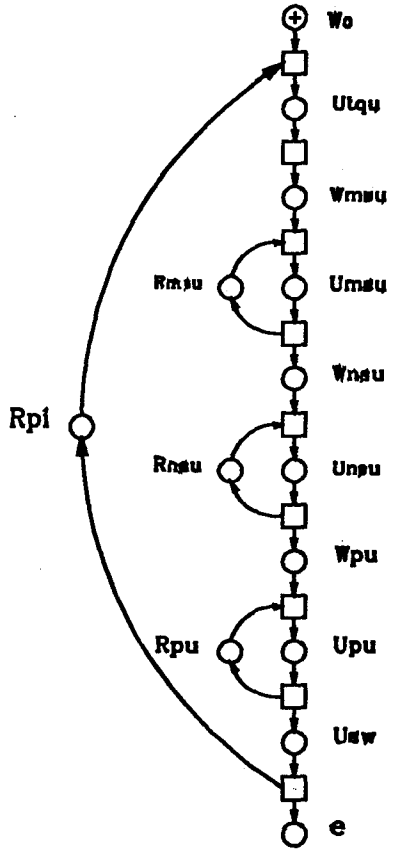

Figura 4.24: Modelo-Q1 com o conjunto de recursos correspondentes às unidades da **MFDM** simplificado.

Assim fica concluído o modelo de recursos da MFDM. Porém, antes de empregá-lo para a análise do sistema, é necessário que se realize sua validação, para garantir sua representatividade. A fase validação, juntamente com a avaliação do poder de análise, do modelo serão demonstradas a seguir.

5 Validação e Análise dos Modelos

A utilidade de um modelo pode ser sentida sob dois aspectos: primeiro como uma maneira de formalizar o conhecimento de que se dispõe sobre um sistema, criando uma linguagem padrão de trabalho; e, segundo, como uma maneira de se obter novas informações sobre o sistema, através da alteração do modelo original e avaliação de seu novo comportamento. No entanto, é necessário que o modelo tenha representatividade sobre o sistema que foi modelado, ou seja, é preciso que o modelo possa descrever as características e o comportamento conhecidos do sistema, antes que o mesmo possa ser empregado para qualquer dos fins acima citados.

A fase responsável pela verificação da adequação do modelo à realidade modelada é conhecida como fase de validação do modelo. Normalmente, esta fase ocorre após o desenvolvimento do modelo, porém isto não é regra geral; nos modelos conceituais, a validação acompanha sua fase de construção. A fase de validação é sempre essencial antes da análise para verificação de novas características, pois sem ela nenhuma afirmação poderá ser feita sobre a validade e precisão dos resultados obtidos.

Foram desenvolvidos neste trabalho dois modelos para a **MFDM**: um conceitual e outro de recursos. A validação do modelo conceitual já foi realizada durante o seu desenvolvimento, visto que a técnica empregada na sua construção - a teoria geral de redes - requer o emprego de conhecimentos sobre a estrutura interna e o comportamento dinâmico do sistema, sem os quais fica impossibilitada a sua utilização. Desta forma, neste capítulo detalhar-se-á somente a validação do modelo de recursos da **MFDM**, que requer alguns cuidados extras devido às simplificações adotadas.

5.1 Validação do Modelo de Recursos

A validação do modelo de recursos de um sistema requer a identificação de valores para seus parâmetros. Os resultados das análises do modelo realizadas com esses valores devem tornar possível ao mesmo refletir situações conhecidas do comportamento do sistema. Portanto, para realizar-se a validação de um sistema de recursos é necessário:

a) obter dados sobre situações reais de funcionamento;

b) definir valores para os parâmetros do modelo nas situações anteriores;

c) realizar a análise do modelo matemático associado a estas situações;

d) avaliar a representatividade do modelo, através da comparação dos dados obtidos com dados coletados.

As duas primeiras fases costumam representar os maiores problemas do dia-a-dia de um analista de desempenho. Isto ocorre devido às dificuldades encontradas em se obter dados de um sistema que está em regime de trabalho, sem que ele seja perturbado, e, também, sem que as medidas dos dados representem altos custos de produção.

Essas dificuldades são normalmente muito reduzidas quando se trata de sistemas em desenvolvimento, pois nestes as alterações e medidas geralmente não representam grandes custos, o que em geral se estende também para a coleta de dados experimentais, definindo-se situações de cargas típicas e obtendo-se parâmetros para elas.

Para o estudo da **MFDM**, adotou-se dois conjuntos de situações descritoras do comportamento do sistema, criados, independentemente, a partir dos os dados apresentados em ([GUR83], [GUR85A]). A escolha desses conjuntos justifica-se devido ao fato de os dados corresponderem à execução de programas no protótipo, sendo, portanto, mais representativos de seu comportamento do que outros obtidos de simuladores, que também são modelos da máquina.

É necessário observar que não existe coerência absoluta entre os valores registrados para os parâmetros nas referências consultadas. Essas diferenças podem ser provenientes do fato de que os valores em [GUR83] são valores de projeto, enquanto os valores apresentados em [GUR85A] são uma mistura de valores de projeto e experimentais. Realizou-se a análise para os dois conjuntos de parâmetros, procurando desta forma verificar qual o conjunto que melhor se adapta à realidade para o modelo proposto. Dessa forma tem-se os seguintes valores:

a) Tempo de permanência na **SW** = 200 ns.

b) Tempo de permanência na **TQU** = 400 ns [GUR83] ou 375 ns [GUR85A].

c) O tempo de permanência na **MSU** é variável e dependente da **mf** que cada ficha recebida carrega. Os valores para estes parâmetros representam a maior divergência encontrada entre as referências citadas. Foram adotados os tempos de permanência na **MSU**, para os dois conjuntos, especificados no projeto original [DAS82]:

   - fichas com **mf** = **BY**, 180 ns;
   - fichas com **mf** = **EW** realizando *extract*, 360 ns;
   - fichas com **mf** = **EW** realizando *wait*, 540 ns.

d) Tempo de permanência na **NSU** = 481 ns [GUR83] ou 500 ns [GUR85A].

e) O tempo de permanência na **PU** é dependente do número máximo de **FUs** que serão empregadas. Neste parâmetro encontra-se outra séria divergência entre os dados das referências citadas. Em [GUR83] a taxa de processamento de uma **FU** é de 0.31 MIPs e em [GUR85A] é de 0.27 MIPs. Considerando-se 12 **FUs** em funcionamento o tempo de permanência na **PU** ficará em 267 ns ou 307 ns respectivamente, resultando em um tempo de permanência na **FU** que será de 3200 ns ou de 3704 ns.

Os demais parâmetros necessários à avaliação do modelo dependem dos programas executados, e serão obtidos do mesmo conjunto abrangente de programas de características conhecidas usados para avaliar o protótipo ([GUR85A], [GUR83]). Para uma primeira análise serão utilizados os dados referentes ao programa LAPLACE. Esta escolha advém do fato deste programa apresentar uma distribuição uniforme do paralelismo durante todo o seu processamento o que é coerente com as simplificações realizadas quando da construção do modelo. Recorde-se que a principal hipótese era de que o sistema estaria em regime estacionário, ou seja, que existiria uma carga de trabalho constante, de valor igual ao paralelismo médio, PI. Os parâmetros neste caso são:

- Percentagem de fichas **BY**, Pby = 0.70;
- Paralelismo médio, PI = 50;

Para aceitação do modelo, considerou-se satisfatória, uma divergência de •15% entre a curva do modelo e a curva experimental. A escolha deste valor pode ser explicada pela função esperada de um modelo analítico, que é, a de realizar previsões de baixo custo sobre possíveis modificações da arquitetura. Entenda-se aqui por baixo custo, um pequeno tempo de desenvolvimento e processamento. Além disto, a precisão esperada do simulador da **MFDM** era de 10%, ficando o modelo pouco distante a um custo bem inferior. Assim sendo, se consideradas satisfatórias as previsões do modelo realizar-se-ão novas análises a partir de simuladores ou protótipos, que, neste caso, poderiam proporcionar maior precisão sobre os resultados.

O gráfico da Figura 5.1 apresenta os resultados de análise para o programa LAPLACE (Pby = 0.70 e PI = 50), com os dois conjuntos de parâmetros. Observa-se que para um número

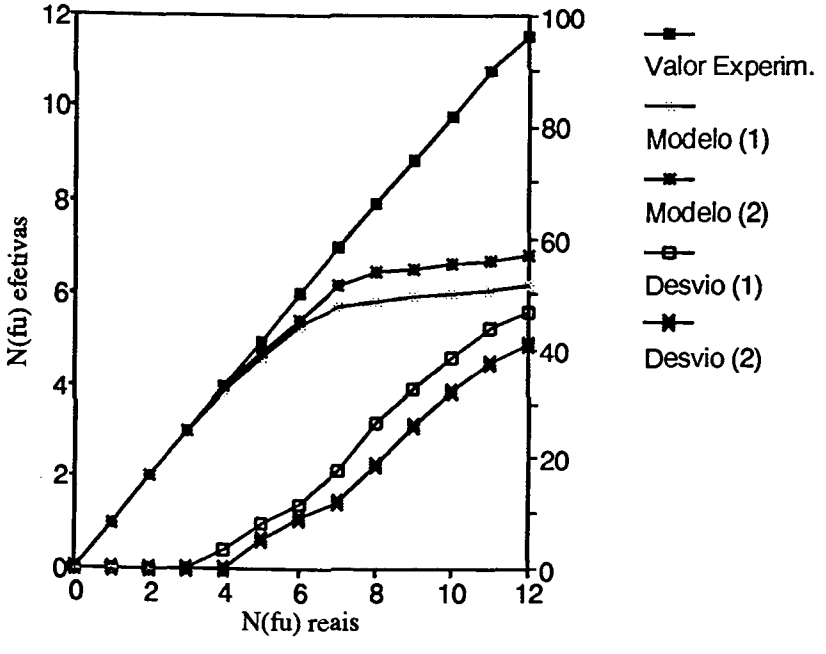

Figura 5.1: Gráfico de desempenho e desvio percentual do modelo para o programa LAPLACE com m0(fu) = 3200 ns (**1**) e m0(fu) = 3704 ns (**2**).

pequeno de **FUs** o desvio percentual apresenta-se dentro da faixa de aceitação em ambos os casos. Porém, com o crescimento do número de **FUs**, o desvio aumenta de forma não linear, tornando o modelo inaceitável.

Dos resultados anteriores é possível inferir que são necessárias, primeiramente, uma reavaliação dos valores dos parâmetros definidos, procurando verificar se é deles que decorre o erro da modelagem e, posteriormente, se isso não for confirmado, uma reavaliação da estrutura do modelo. Um dos problemas mais sérios encontrados na reavaliação de valores de parâmetros é determinar quais parâmetros estão com seus valores incorretos, visto que não é aconselhável alterar diversos parâmetros de uma só vez.

Voltando-se à Figura 5.1, verifica-se que, para o segundo conjunto de parâmetros, a resposta do modelo apresentou menores desvios, apesar de ser pequena a diferença entre eles, e, portanto, apresenta melhor representatividade. Analisando-se os valores dos parâmetros utilizados, observa-se que a taxa de processamento na **FU** é o único parâmetro que apresenta diferença de valor significativa entre os dois conjuntos. Este fato induz ao questionamento da influência deste parâmetro sobre os resultados de análise. Dessa forma, voltando-se para os valores obtidos para a taxa de processamento das **FUs** e apresentados em [GUR85A], torna-se claro que o valor escolhido para este parâmetro está muito além dos valores reais. Uma possível razão para esta diferença está no fato de que a distribuição de paralelismo durante o processamento da maioria destes programas é muito variável, conforme demonstrado em [GUR83].

A alta variação do grau de paralelismo pode ocasionar o aparecimento de um grande número de bolhas no *pipeline*, em conseqüência do grande número de fichas que não sofrem *matching* ao darem entrada na **MSU**, pela ausência de suas parceiras. Isto resulta numa queda de desempenho das **FUs** e provoca uma diminuição da taxa de processamento medida. Essas flutuações no valor do paralelismo durante o processamento não foram consideradas quando da construção do modelo. Pelo contrário, elas foram eliminadas pela principal simplificação realizada no modelo: a hipótese de que o sistema estaria em regime estacionário com taxa de trabalho igual ao paralelismo médio do programa. A diminuição dos valores da taxa de processamento medida pode ser um indicador de que esta simplificação é incorreta, mostrando ser necessária uma forma mais efetiva de se considerar as influências do programa no modelo.

Por outro lado, antes de se pensar em alterar o modelo, convém realizar uma nova análise, alterando-se o valor do taxa de processamento da **FU** (que atualmente corresponde a um valor de projeto) para a média dos valores medidos apresentados em [GUR85A] (que são valores experimentais e portanto mais representativos) e verificando-se o seu novo comportamento. Esse caminho leva a uma taxa de processamento de 0.14 MIPs, resultando em um tempo de permanência na **FU** de 7108 ns e um tempo de permanência na **PU** de 592 ns. Os valores dos demais parâmetros serão os mesmos do conjunto 2.

Como pode ser observado no gráfico da Figura 5.2, construído apenas com os dados relativos ao programa LAPLACE (Pby = 0.70 e PI = 50), o desvio da curva experimental diminuiu de forma considerável, ficando abaixo dos 10%. Assim , o novo valor escolhido para a taxa de processamento resolve o problema de representatividade para o modelo, trazendo os desvios dos valores reais para dentro da faixa de aceitação. Desta forma, a estrutura do modelo não necessitará ser redefinida para acomodar o fenômeno das flutuações do paralelismo durante o processamento. Além disto, este fato indica que a simplificação adotada pode ser considerada aceitável.

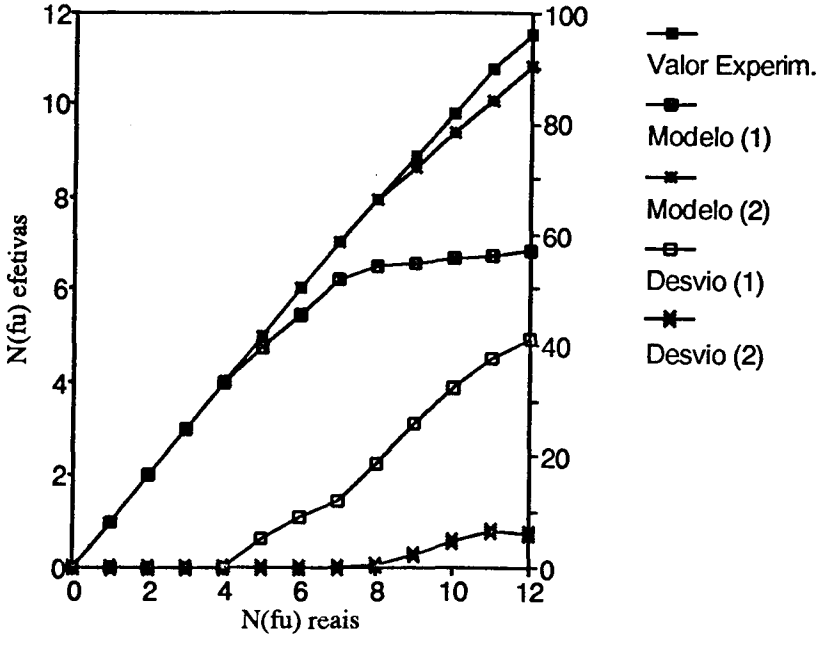

Figura 5.2: Gráfico do desempenho e desvio percentual do modelo para o programa LAPLACE com m0(fu) = 3704 ns (**1**) e m0(fu) = 7108 ns (**2**).

Apesar de esses resultados apresentarem baixos desvios em relação ao valor experimental, eles devem ser considerados com muito cuidado, uma vez que eles correspondem à análise do modelo para um único programa. Além disso, este programa tem um comportamento muito próximo ao imposto pelas simplificações adotadas quando da construção do modelo. Por isso, pode ter ocorrido um mascaramento de algumas deficiências do modelo que tenham relação direta com características do programa que está sendo executado, como, por exemplo, a distribuição do paralelismo exibido ao longo da execução.

Para uma melhor validação do modelo, será realizada uma análise mais rigorosa, verificando-se o seu comportamento para um conjunto maior de programas, exibindo uma gama mais ampla de valores de paralelismo médio, visto que Pby não sofre grande variação dentro do conjunto de programas apresentados oscilando entre 0.50 e 0.70. Para esta análise considerar-se-á o programa SUM com o Pby igual 0.61 e PI assumindo os seguintes valores: 1, 5, 10, 20, 50 e 150.

O gráfico da Figura 5.3 apresenta um resumo dos resultados obtidos para o programa SUM (Pby = 0.61), e demonstra que para valores altos do paralelismo médio, PI maior que 50, o modelo apresenta boa representatividade ficando seus desvios dentro da faixa de aceitação. O mesmo ocorre para PI igual a 1, para o qual o modelo tem bom comportamento. Porém, para valores intermediários do paralelismo médio, PI entre 5 e 20, a resposta não é boa e os desvios excedem a faixa de aceitação, inclusive para um pequeno número de FUs ativas.

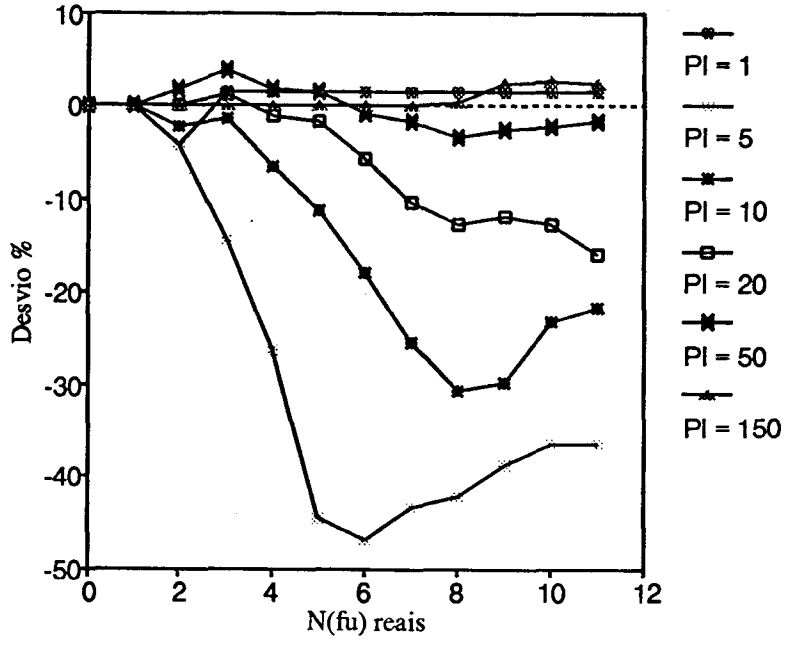

Figura 5.3: Desvio percentual do valor real para o programa SUM tomando como parâmetro a variação do paralelismo médio do programa.

As observações anteriores devem ser analisadas com cuidado para que não sejam tiradas conclusões errôneas. Os altos desvios percentuais encontrados para os valores calculados, para os casos de PI igual a 5, 10 e até mesmo 20, podem ser explicados considerando-se o tipo da distribuição de paralelismo durante o processamento do programa SUM.

O programa SUM corresponde a um algoritmo de soma de inteiros que contém uma dupla recursão. Este fato lhe proporciona uma distribuição de paralelismo bastante peculiar [GUR83], fazendo com que o paralelismo máximo do programa esteja concentrado em aproximadamente 15% do seu tempo de processamento. Devido a esta forma peculiar de distribuição, o fenômeno do surgimento de bolhas é muito acentuado quando o paralelismo médio é menor que 20. Como o paralelismo efetivo nos 85% restantes do tempo de processamento é muito baixo, em torno de 30% do valor médio, as FUs ficarão com seu desempenho muito prejudicado. Este fenômeno não será sentido com tanta intensidade para altos valores de paralelismo pois, apesar do grande número de bolhas, o paralelismo efetivo ainda será maior que o número de FUs ativas.

A Figura 5.4 mostra os desvios para um outro grupo de programas com paralelismo médio entre 10 e 20: INTEGRATE (Pby = 0.64 e PI = 12), RSIM/1 (Pby = 0.61 e PI = 15), PLUMBLINE/2 (Pby = 0.56 e PI = 20) e LEETEST/2 (Pby = 0.63 e PI = 20). Estes programas, exceto LEETEST/2, também apresentam comportamentos peculiares para a distribuição do paralelismo durante o processamento [GUR83], porém de modo menos excêntrico.

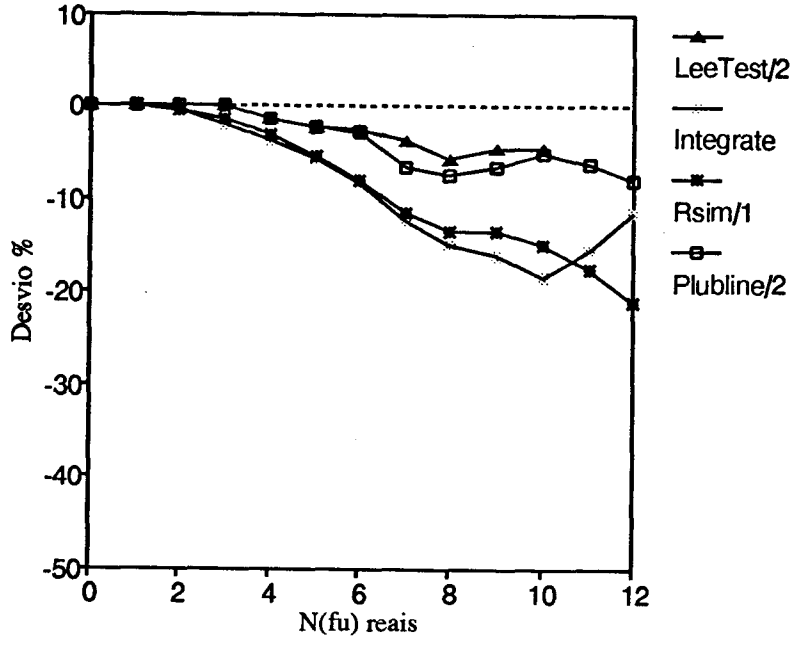

Figura 5.4: Desvio percentual do valor real para programas com PI entre 10 e 20.

Apesar de todas as curvas corresponderem a valores de PI acima de 10, os resultados apresentados vêm, de certo modo, confirmar as suposições anteriores, com desvios altos, que em certos pontos ultrapassam a faixa de aceitação.

Este fato demonstra que a faixa de baixa representatividade identificada no programa SUM está restrita a casos patológicos, e, também, que o modelo pode ser empregado com confiança fora dos limites dessa faixa, ou em casos normais, em que a distribuição de paralelismo durante o processamento aproxima-se da uniforme.

Cabe ainda uma explicação para o bom comportamento do modelo para o valor de PI igual a 1. Ocorre que, neste caso, o paralelismo médio será igual ao efetivo, pois há somente uma única linha de processamento. Desta forma, haverá apenas uma ficha circulante no anel, o que não permite a formação de grupos de bolhas.

As falhas na representatividade do modelo para baixos valores de PI dão uma demonstração clara de que é interessante sua ampliação, para suportar a especificação da distribuição do paralelismo durante o processamento.

A Figura 5.5 corresponde aos resultados obtidos para os programas: FFT (Pby = 0.70 e PI = 50), PLUMBLINE/1 (Pby = 0.61 e PI = 50), LOGICSIM/1 e /2 (Pby = 0.63 e PI = 50 e 70

respectivamente) e LEETEST/1 (Pby = 0.62 e PI = 40). O comportamento observado em relação à realidade, para PI maior que 20, indica boa representatividade para esta faixa de valores.

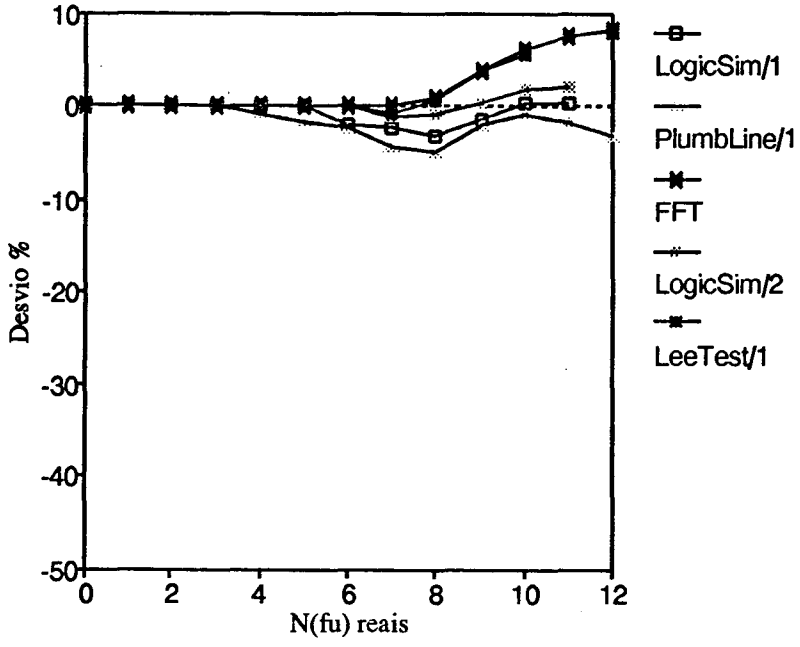

Figura 5.5: Desvio percentual do valor real para os programas com valores de PI > 20.

Dos resultados acima apresentados, e considerando a utilidade a que se propõe este tipo de modelo analítico, aceitar-se-á que o modelo formado pela rede descrita no capítulo 4 com os parâmetros aqui definidos é representativo da **MFDM**, podendo ser empregado para avaliar seu desempenho. A ampliação do modelo, com a inclusão da possibilidade de especificação da distribuição do paralelismo durante o processamento viria a aumentar a sua representatividade, porém foi descartada por estender demasiadamente o escopo do trabalho. Abrem-se, assim, várias possibilidades de emprego do modelo desenvolvido, entre elas a de localizar pontos de estrangulamento da arquitetura conforme será demonstrado a seguir.

5.2 Análise de Desempenho

Uma vez validado um modelo, este poderá ser empregado com segurança na determinação do comportamento atual e futuro do sistema. Dentre as propriedades de maior interesse a serem analisadas em arquiteturas de computadores estão: pontos de estrangulamento no desempenho atual do sistema e avaliação do seu comportamento frente a variações na sua estrutura. A determinação dos pontos de estrangulamento leva à determinação dos pontos fracos no projeto.

O estudo de desempenho e das possíveis fraquezas estruturais de um sistema pode ser realizado tanto por um modelo de recursos quanto por um conceitual. No presente trabalho foram desenvolvidos modelos de ambos os tipos para a **MFDM**. O sistema foi analisado, primeiramente, através do modelo de recursos, buscando identificar pontos de estrangulamento

de seu desempenho. A partir dessa análise, com auxílio do modelo conceitual, foram identificadas algumas das razões destes estrangulamentos e sugeridas alternativas para sua superação. Esse processo está descrito no restante desta dissertação.

5.2.1 Análise do Desempenho Atual

Na avaliação do comportamento de um sistema seria ideal analisar o máximo de informações possíveis sobre o mesmo. Porém, na busca do equilíbrio, no mínimo é necessário avaliar a curva de desempenho e o gráfico de utilização de recursos do sistema. Destes, pode-se retirar informações sobre a existência de pontos de estrangulamento, que se tornam visíveis através da variação da inclinação da curva de desempenho e do desvio das curvas de utilização dos recursos dos respectivos valores máximos.

A descoberta da existência de pontos de estrangulamento é a base para a identificação dos recursos que são os responsáveis pela queda no desempenho do sistema. Esta determinação será auxiliada pela análise das curvas do tempo médio de espera para o atendimento em um recurso, ou das curvas da população média de fichas à espera de atendimento nos recursos.

Considerando-se que o objeto de interesse deste trabalho é a análise do desempenho da **MFDM**, será necessário definir primeiramente uma configuração padrão de trabalho para o sistema. Isto implica na definição de parâmetros para o modelo analítico dentro da faixa de trabalho que se pretende atuar. Neste caso foram tomados:

a) Tempo de permanência na **SW** - 200 ns.

b) Tempo de permanência na **TQU** - 375 ns.

c) Tempo de permanência na **MSU**

- fichas com **mf** = **BY** - 180 ns;
- fichas com **mf** = **EW** com ação de *wait* - 360 ns;
- fichas com **mf** = **EW** com ação de *extract*- 540 ns.

d) Tempo de permanência na **NSU** - 500 ns.

e) Tempo de permanência na **PU** - 592 ns, resultando num tempo de permanência na **FU** - 7108 ns.

f) Paralelismo médio PI = 100.

g) Percentagem de fichas com **mf** = **BY** - Pby = 0.60.

A escolha dos tempos de permanência está associada aos valores obtidos quando da validação do modelo descrita na seção anterior. O valor de PI igual a 100 tem sua justificativa no fato de o interesse maior residir no desempenho da arquitetura para o processamento de

programas com alto grau de paralelismo. O valor de Pby foi tomado como uma aproximação da média dos valores apresentados pelos programas incluídos no pacote de avaliação [GUR85A].

Devido ao maior interesse na análise de desempenho da **MFDM** estar associado à avaliação do seu comportamento mediante a adição de **FUs** à **PU**, todas as curvas para a análise serão tomadas com base no número de **FUs** ativas. Além disto, analisar-se-á a curva da eficiência de uso das **FUs**, em lugar da curva de desempenho, para identificação dos pontos de estrangulamento. O número máximo de **FUs** ativas foi arbitrariamente fixado em 25.

O gráfico da eficiência de uso das **FUs** no sistema da **MFDM**, apresentado na Figura 5.6, demonstra claramente a existência de dois pontos de inflexão na curva a partir de 7 e 14 **FUs** ativas, respectivamente.

A identificação da unidade responsável pela redução na taxa de crescimento do desempenho requer a análise do gráfico do tempo médio de espera para o atendimento em um recurso, que se encontra na Figura 5.7. Deste gráfico podem ser tiradas as seguintes conclusões:

a) as unidades não estão operando no mesmo regime de trabalho, visto suas curvas de tempo médio de espera serem distintas;

b) até o número de 14 **FUs** ativas o gargalo do sistema está na **PU**, como fica claro devido ao maior tempo médio de espera apresentado pela sua curva;

c) com 15 **FUs** ativas haverá um salto positivo no tempo médio de espera na **MSU** e um salto negativo na **PU**, esta inversão desloca o gargalo para a **MSU**;

d) com 16 **FUS** ativas ocorre um salto positivo no tempo médio de espera na **NSU**

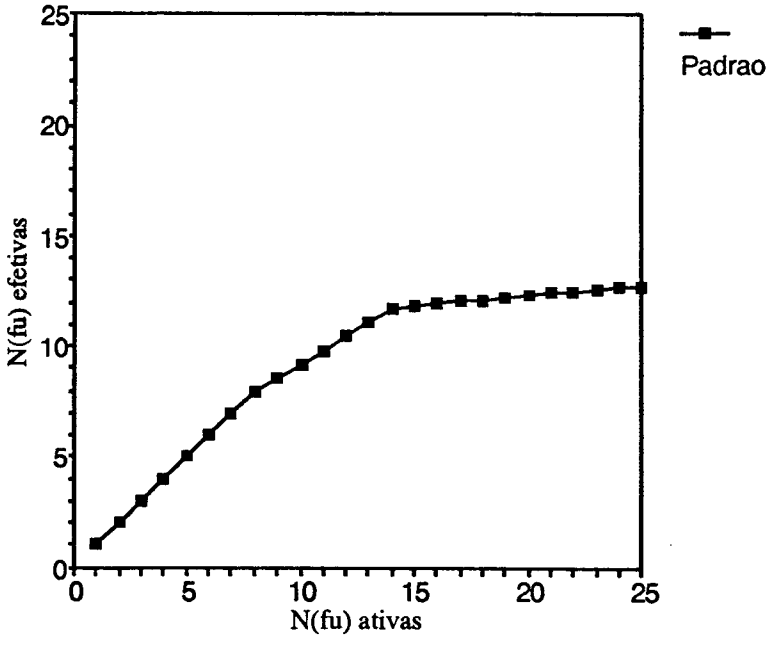

Figura 5.6: Gráfico de desempenho do sistema da **MFDM** para PI = 100 e Pby = 0.60.

deslocando o gargal[illegible]ra esta unidade.

Para estimar as razões dos problemas acima descritos é necessário observar que:

a) A taxa de processamento da **MSU** depende da fração de fichas que realizam *bypass* (Pby) e também da fração de fichas que encontram sua parceira na memória (**mf = EW**, realizando *extract*).

b) A taxa de processamento na **NSU** é constante.

c) O volume de fichas que chega à **NSU** e à **PU** é menor que o que chega à **MSU** devido ao agrupamento de fichas que esta realiza. Além disto, este volume depende do número de fichas que realizam *bypass*.

d) O aumento progressivo do número de **FUs** ativas na **MFDM** causa um aumento na velocidade com que novas fichas são produzidas na **PU** e introduzidas no anel, ou seja, causa um aumento na taxa de processamento efetivo da **PU**.

e) O modelo-Q desenvolvido é determinístico e fornece uma avaliação do melhor caso na análise do desempenho do sistema.

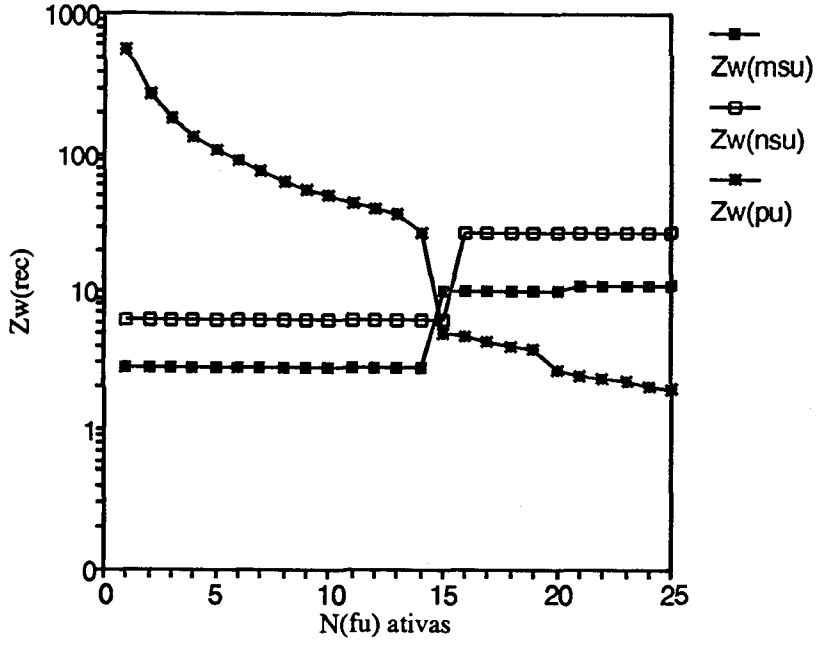

Figura 5.7: Gráfico do tempo médio de espera para atendimento nas unidades do sistema da **MFDM** para PI = 100 e Pby = 0.60.

Assim, os saltos do gráfico da Figura 5.7 decorrem, primeiramente, da forma como o modelo está construído, de modo determinístico e aproximativo, incorporando as variações dos tempos de processamento na forma de redes alternativas, cujos resultados são ponderados pelas respectivas probabilidades associadas. Além disso, mesmo as curvas ponderadas estão representadas em função do número de **FUs** ativas, não sendo possível representar frações de **FUs**.

Assim, o crescimento do número de **FUs** ativas leva a taxa de processamento nominal da **PU** a atingir a das demais unidades, fazendo com que o gargalo em uma rede específica mude para outra unidade. No primeiro salto, com 14 **FUs** ativas, atinge-se o valor da taxa da **MSU** para a rede em que somente ocorrem *waits*, e que somente afeta as redes em que a **MSU** é o gargalo. Ao acrescentar-se mais uma **FU** a taxa de processamento da **PU** atingirá o valor da taxa da **NSU** provocando um novo salto, desta vez na curva da **NSU**. Observa-se ainda que ocorrerá outro salto nas curvas da **MSU** e da **PU** ao atinge-se 20 **FUs** ativas, idêntico ao que ocorreu anteriormente com estas curvas. Este salto está relacionado com o fato de a taxa de processamento da **PU** atingir a da **MSU**, para a rede em que somente ocorrem *extracts*.

Adicionalmente, o gráfico de utilização dos recursos do sistema, que se encontra na Figura 5.8, fornece outras informações, que não estão totalmente visíveis nos demais gráficos. Através dele é possível verificar que, para até 7 **FUs** ativas, a utilização da **PU** é máxima, decaindo a partir daí. A redução da taxa de utilização da **PU** indica ociosidade de **FUs**. Esta ociosidade decorre do aparecimento de bolhas no *pipeline*, que está diretamente relacionado com número de *waits* executados na **MSU**.

Apesar de a partir de 7 **FUs**, a **PU** não trabalhar mais e sua capacidade nominal, o acréscimo de novas **FUs** continuará a melhorar o desempenho do sistema, até o limite de 14 **FUs** (Figura 5.6 e 5.8). Ao atingir 15 **FUs** ativas, a taxa de processamento da **PU** supera a da **MSU** na rede de *waits*, e o efeito das bolhas será sentido com maior intensidade, provocando a passagem temporária do gargalo para **MSU**. A partir de 16 **FUs** ativas, mesmo com o fenômeno de bolhas mais acentuado, a **NSU**, cuja a taxa efetiva já é menor que a **MSU**, passará a ser o gargalo, ao ser superada também pela **PU**.

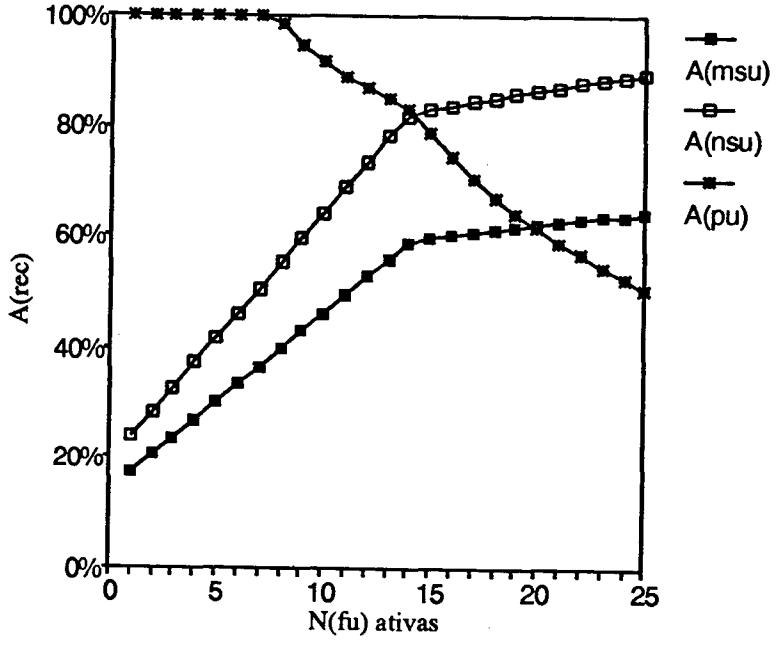

Figura 5.8: Gráfico de utilização dos recursos do sistema da MFDM para PI = 100 e Pby = 0.60.

Essa conclusão diverge parcialmente dos resultados reportados por Ghosal ([GHO90], [GHO87]) a partir de um modelo estocástico de redes de filas.

5.2.2 Análise de Alternativas de Implementação

Identificadas as unidades responsáveis pela queda do desempenho do sistema, é possível avaliar soluções alternativas para estes problemas, através da alteração do modelo e sua posterior análise. Neste caso específico podem ser sugeridas duas alternativas: a redefinição do tempo de processamento da **NSU**, ou a alteração do número de unidades disponíveis. Ambas podem ser examinadas pela alteração dos parâmetros do modelo de recursos desenvolvido.

Para examinar a primeira alternativa, arbitrou-se o valor de 360 ns, que resulta num tempo efetivo igual ao da **MSU**, equalizando, desta forma, as taxas de processamento dessas unidades. Um bom procedimento será o de equalizar também o valor da taxa de processamento da **TQU**, passando-o para 288 ns, eliminando pequenos atrasos resultantes de diferenças nestas taxas, e possibilitando uma melhor avaliação da alternativa.

O gráfico da Figura 5.9 demonstra a existência de uma melhora na eficiência do uso das **FUs**, embora ainda existam os mesmos dois pontos de deflexão na curva. Estas deflexões são indicadoras da existência de pontos de estrangulamento no desempenho do sistema, que necessitam da análise do gráfico do tempo médio de espera para sua identificação.

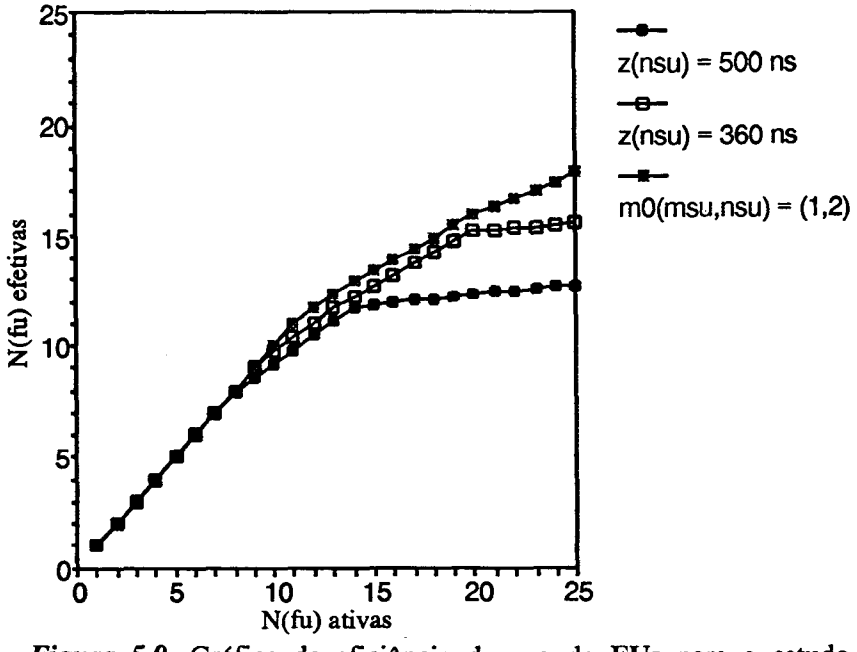

Figura 5.9: Gráfico da eficiência de uso da **FUs** para o estudo de eliminação dos gargalos do sistema.

Pode-se visualizar no gráfico da Figura 5.10, a razão básica de uma das deflexões. Ao serem atingidas 14 **FUs** ativas, ocorre um salto no tempo de espera na **MSU**, causado pela igualdade entre as taxas de processamento da **PU** e da **MSU** na rede de *waits*. Este salto provocará um acúmulo de fichas a serem atendidas na **MSU**, acarretando um aumento do número

de bolhas no *pipeline*, que fará com que a **PU** perca eficiência, embora continue a ser o gargalo do sistema. Ao se atingirem 20 **FUs** ativas, ocorrerá, além de um segundo salto na **MSU**, um salto na **NSU**, pois a taxa efetiva de processamento da **PU** alcançará a da **NSU** e a da **MSU** na rede de *extracts*. Este salto desloca o gargalo para a **MSU**, pois o aumento no tempo médio de espera na **NSU** fará com que diminua a população média à espera de atendimento na **PU**.

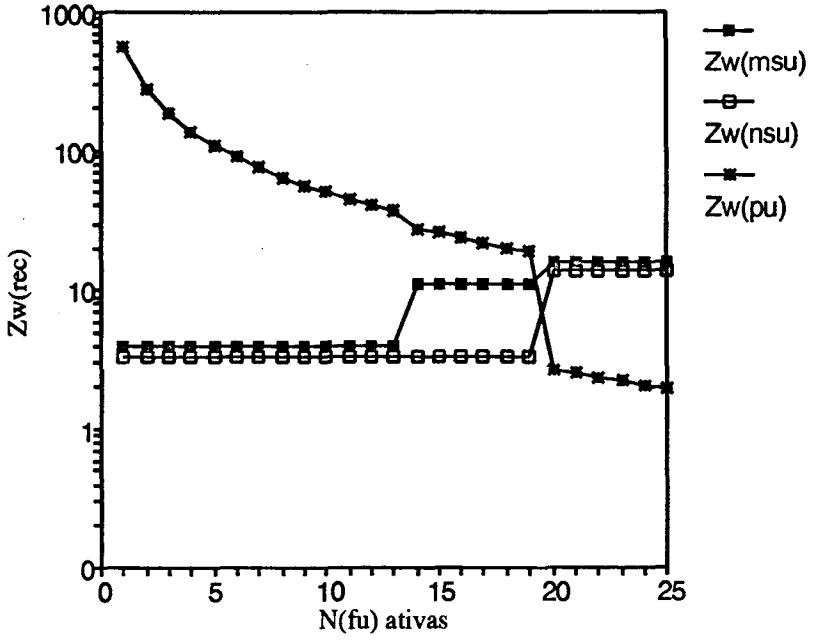

Figura 5.10: Gráfico do tempo médio de espera para atendimento com as taxas de processamento da **MSU**, **NSU** e **TQU** equalizadas.

A segunda alternativa proposta requer que se dobre o número de centros de serviço desta unidade. Neste caso poderá ser mantido o tempo de processamento empregado nas análises de identificação do gargalo (500 ns), ou alterá-lo mantendo-se o último tempo utilizado (360 ns), que equaliza as taxas de processamento. Será empregada a segunda alternativa, por ser mais eficiente no uso das **FUs**. Essa escolha também tornará possível a comparação das duas alternativas examinadas para a eliminação do gargalo.

No gráfico da Figura 5.11, pode-se verificar que as unidades responsáveis pelo estrangulamento do desempenho serão a **PU** e a **MSU**, conforme o número de **FUs** ativas, visto que o tempo médio de espera da **NSU** será constante e menor que o das outras unidades. Neste caso, as deflexões observadas nas curvas de eficiência de uso das **FUs** (Figura 5.9) também estarão relacionadas com os saltos, ou seja, com o fato de a taxa efetiva de processamento da **PU** atingir o valor das taxas de **MSU** para os diferentes casos que esta comporta.

A primeira deflexão encontrada, em ambas as alternativas, não é visível nas curvas de tempo médio de espera. Este fato decorre da deflexão ser causada pelo início do fenômeno de formação de bolhas no *pipeline*, que provocará uma queda no desempenho individual da **PU** desviando-o dos 100%.

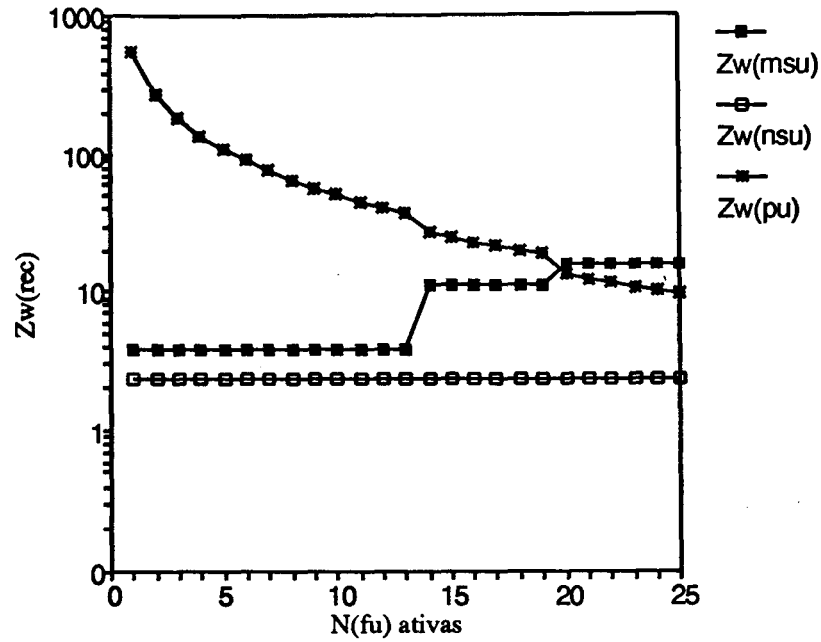

Figura 5.11: Gráfico do tempo médio de espera para atendimento com as taxas de processamento da **MSU**, **NSU** e **TQU** equalizadas e m0(nsu) = 2.

A eliminação do estrangulamento que existia na **NSU** simplesmente deslocou o gargalo para a **MSU** sem produzir um grande aumento da eficiência no uso das **FUs**. Para atacar este problema, pode-se variar o número de centros de serviço na **MSU**, e acompanhar seu comportamento. Esta possibilidade será examinada para um ou dois centros de serviço na **NSU**, mantendo-se equalizadas as taxas de processamento. As curvas de eficiência obtidas aparecem na Figura 5.12, na qual se repete, para efeito de comparação, os melhores resultados anteriores.

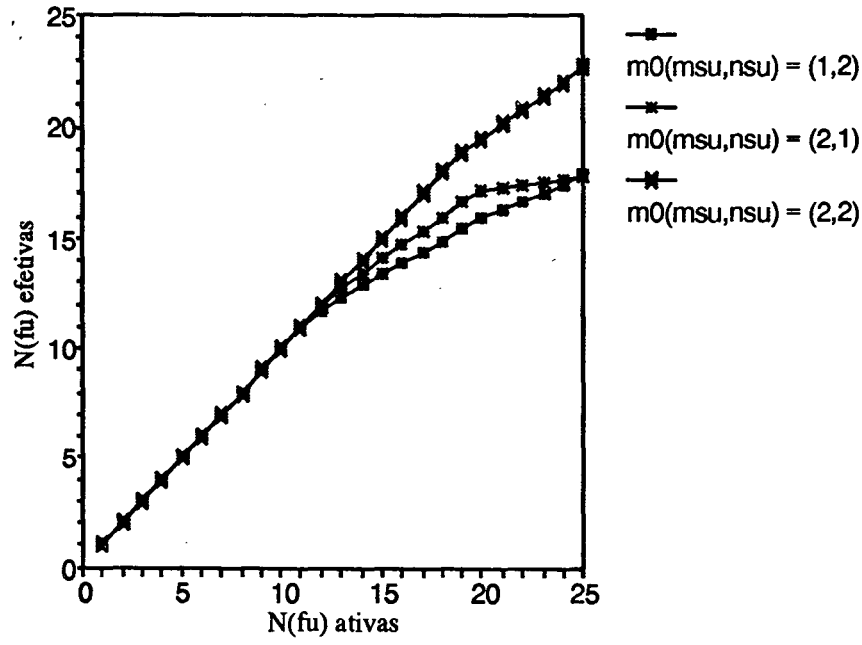

Figura 5.12: Gráfico da eficiência de uso da **FUs** para o estudo de eliminação dos gargalos do sistema.

A Figura 5.13 mostra que, dobrando-se o número de centros de serviço na MSU, o tempo médio de espera para atendimento na unidade torna-se constante e menor que nas demais unidades. Desta forma o gargalo retorna para a **PU** e a **NSU**, conforme o número de **FUs** ativas.

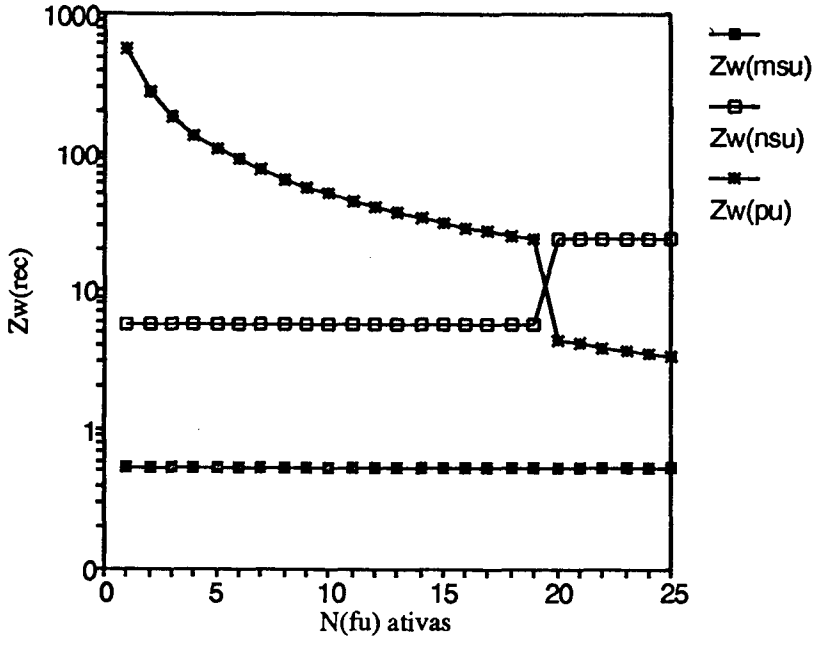

Figura 5.13: Gráfico do tempo médio de espera para atendimento com as taxas de processamento da **MSU**, **NSU** e **TQU** equalizadas e m0(msu) = 2.

Na segundo hipótese, da passagem para dois centros de serviço tanto na **MSU** quanto na **NSU**, mantidas equalizadas as taxas de processamento, o gargalo é causado exclusivamente pela **PU** (Figura 5.14). Este arranjo fornece a melhor alternativa em termos de eficiência do uso de

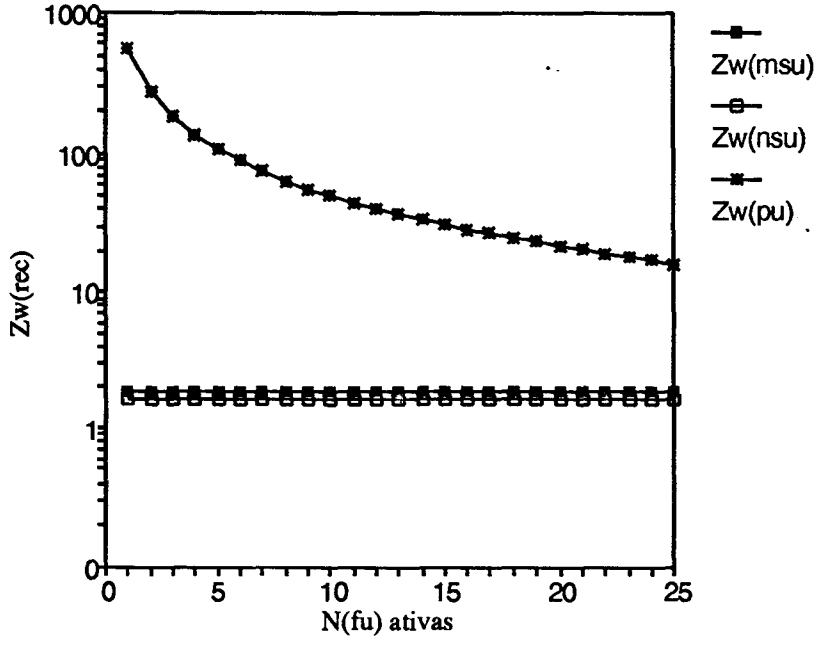

Figura 5.14: Gráfico do tempo médio de espera para atendimento com as taxas de processamento da **MSU**, **NSU** e **TQU** equalizadas e m0(msu) = 2.

FUs, mantida igual a 100% até 18 **FUs** ativas. Ainda assim, ocorrerá um desvio devido à formação de bolhas no *pipeline*, que deverá ser estudado em trabalhos posteriores.

As análises até agora desenvolvidas demonstram a capacidade de predição e avaliação de alternativas que o modelo proposto exibe. A análise de outros arranjos é também possível, com ou sem modificação do modelo de recursos desenvolvido. Outras características que podem ser incorporadas ao modelo incluem: o tratamento da variação no grau de paralelismo médio, e a avaliação do desempenho em função da percentagem de fichas que sofrem *bypass* ou do grau de paralelismo médio do programa. Recomenda-se que essas possibilidades sejam examinadas em futuros trabalhos do grupo de fluxo de dados da Unicamp.

6 Conclusões e Recomendações

A demanda por processamento de dados de alto-desempenho levou ao desenvolvimento de modelos alternativos de computação e destes a novas arquiteturas de computadores. Em muitos casos, arquiteturas foram propostas, modificadas e protótipos construídos sem uma avaliação adequada dos determinantes fatores de sua viabilidade técnica ou econômica.

É extremamente importante que a análise desses fatores seja realizada o mais cedo possível de modo a evitar que possíveis ineficiências atinjam estágios mais avançados do projeto. Uma forma consagrada para se realizar esta análise é através da criação e avaliação de modelos para a arquitetura proposta, dada sua eficácia e rapidez de desenvolvimento.

Este trabalho foi desenvolvido visando criar um modelo da arquitetura da **MFDM** e realizar a sua análise buscando identificar pontos falhos na arquitetura. No desenvolvimento e análise deste modelo foram empregadas, como opção à abordagem clássica por redes de filas, as redes de Petri, pertencentes à Teoria Geral de redes.

A seguir, descreve-se o caminho seguido durante o trabalho de desenvolvimento, validação e análise do modelo da **MFDM**, propõem-se soluções que poderão ser consideradas em futuros projetos, e discutem-se as conclusões a que se pôde chegar.

O projeto do grupo de fluxo de dados da Unicamp tomou, como ponto de partida, o projeto da Máquina de Manchester, a cujo desenvolvimento estava intimamente ligado. Esse desenvolvimento apoiou-se em um simulador e um protótipo, duas ferramentas de custo de produção, utilização e modificação bastante elevados, que oferecem limitada flexibilidade para a avaliação de alternativas de projeto. Esta pouca flexibilidade levou o grupo a fixar decisões de projeto em estágios primários do desenvolvimento, sem amparo analítico ou experimental adequado, resultando em deficiências na arquitetura final.

Para evitar a repetição desses problemas, o grupo de fluxo de dados da Unicamp decidiu desenvolver uma ferramenta com a qual pudesse avaliar uma grande variedade de alternativas de implementação, com custo razoável.

O próximo passo foi a escolha da metodologia para a modelagem e análise da arquitetura. Neste caso optou-se pela redes de Petri em contrapartida à abordagem clássica de redes de filas. Esta escolha justificou-se devido à adequação das redes de Petri para a modelagem de sistemas que apresentam tarefas paralelas, ao fato de não exigirem o conhecimento de parâmetros que nem sempre são facilmente obtidos no estágio de projeto e à simplicidade apresentada para seu aprendizado e análise.

Os resultados obtidos foram comparados não só com os oferecidos pelo protótipo ([GUR85A], [GUR83]), mas também com os do trabalho desenvolvido para a **MFDM** por Ghosal ([GHO90], [GHO87]), utilizando um modelo estocástico de redes de filas.

Em seus artigos, Ghosal adotou simplificações que reduzem os problemas enfrentados pelas redes de filas no tratamento de problemas com tarefas paralelas. Os resultados apresentados por Ghosal demonstram que, apesar das simplificações, o modelo obtido representou de forma satisfatória a arquitetura e permitiu a identificação de alguns pontos de estrangulamento.

A Figura 6.1 exemplifica um caso extremo das diferenças entre os resultados dos dois trabalhos. Observa-se que, para programas com baixos valores de paralelismo médio, no caso **PI** = 15, o modelo-Q utilizado aqui apresenta um desvio do valor real significativamente mais alto, para um grande número de **FUs** ativas.

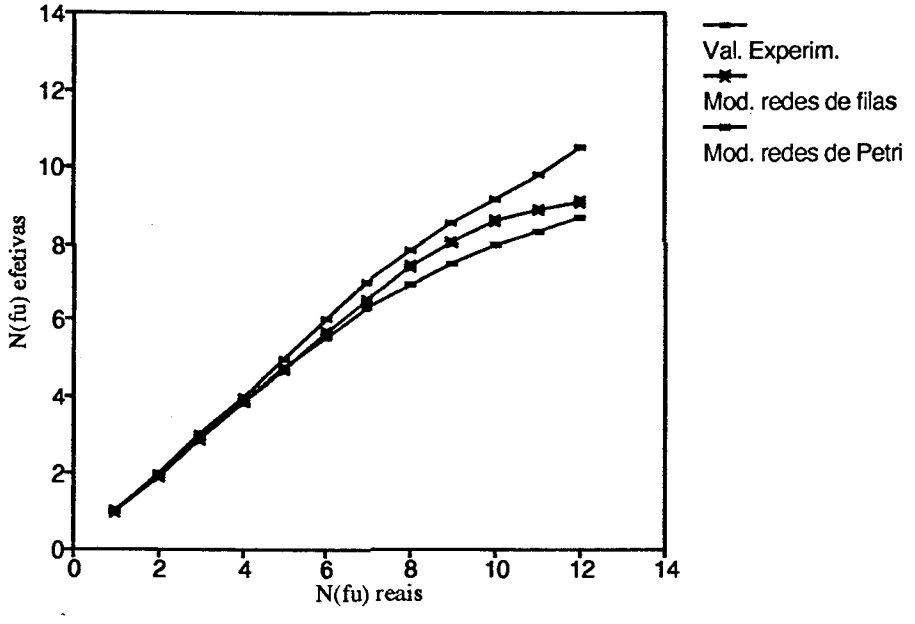

Figura 6.1: Comparação entre os modelo de redes filas e da Teoria Geral de Redes.

Este fato pode ser justificado pelo fato de que, na análise do modelo de redes de filas, o valor do paralelismo médio é tomado como a média de uma distribuição normal, pois sua análise é estocástica. Por outro lado, no modelo de redes de Petri adotado, o valor é tomado como exato e determinístico. Dessa forma, a análise do modelo de redes de filas consegue uma melhor aproximação para os resultados port levar em conta a existência de variação no paralelismo de um programa, o que não ocorre na análise desenvolvida no atual estágio do modelo de redes de Petri.

Como se discutiu no capítulo 3, é possível modelar a variação do paralelismo em um programa utilizando-se de métodos aproximativos de análise. Neste caso considera-se um certo conjunto de possíveis valores para o paralelismo médio e as respectivas probabilidades de

ocorrência e ponderam-se os resultados para determinar o desempenho final da rede. Esse método, apesar de ser determinístico, leva em consideração a variação real do paralelismo do programa e deverá aproximar-se melhor da curva experimental. Recomenda-se que esse método seja considerado como o primeiro passo em um trabalho futuro que vise ampliar o escopo da análise realizada pelo modelo.

O trabalho atingiu seus objetivos, tornando possível uma metodologia de modelagem para a arquitetura da **MFDM**, como discutido no capítulo 4, e realizando a validação e análise de um modelo de recursos, assim como a identificação de pontos de estrangulamento no sistema, conforme descrito no capítulo 5.

Tanto a modelagem como a análise foram beneficiadas pela grande adequação da técnica adotada ao problema a ser resolvido. A validação do modelo demonstrou a necessidade de aperfeiçoamentos para o caso em que o paralelismo médio é baixo, tendo este problema sido estudado e uma solução preliminar proposta, embora não implementada.

Entre as vantagens reconhecidas no emprego das técnicas de redes de Petri como ferramenta de modelagem e análise estão:

- A possibilidade do emprego de uma mesma ferramenta no desenvolvimento dos modelos conceitual e de recursos. Isto facilita não só o entendimento do sistema modelado, por utilizar uma única linguagem de descrição, mas também, a construção do modelo de recursos.
- A facilidade de modelagem de situações que apresentam concorrência entre suas tarefas, que não é compartilhada pelas redes de filas.
- A existência de métodos simples para análise de modelos de recursos baseados em redes de Petri, além do curto tempo de resposta necessário à análise e alteração dos mesmos.

Foram ainda identificados, como resultado colateral, pontos de estrangulamento da arquitetura e apresentados algumas alternativas para sua eliminação. Destes estudos foi possível concluir que é necessário realizar inicialmente um aprimoramento tanto da **NSU** quanto da **MSU** para que a arquitetura possa vir a empregar todo o seu potencial teórico.

Em princípio, o problema encontrado na **NSU** é de ordem puramente tecnológica, ou seja, o emprego de memórias de tecnologia mais recente, com um tempo de acesso menor, poderão resolver a questão. Porém, esta solução simplesmente deslocará o problema do gargalo para a **MSU**, conforme demonstrado em análise realizada no capítulo 5.

Já o problema da **MSU** é um pouco mais complexo, visto que ela requer uma unidade de memória associativa real, que apresenta um alto custo, mesmo nos dias atuais. Apresentar uma solução para este problema, sem um estudo mais profundo, é temerário, porém é possível indicar uma metodologia para que se possa acessar o problema de forma coerente.

Em síntese, recomenda-se, como linhas de continuação deste trabalho, o aperfeiçoamento do modelo com o tratamento do paralelismo variável e a representação da unidade de estruturas e do mecanismo supressor/regulador de paralelismo, além da investigação de melhorias do projeto original como a otimização do funcionamento da **MSU** e do sistema de comunicação. Somente após trabalhos como esses é que será possível realizar uma avaliação mais precisa do verdadeiro potencial desta classe de arquiteturas.

Bibliografia

[ACK82] ACKERMAM, W.B.. *Data Flow Languages.* IEEE Computer, v. 15, n. 2, p. 15-25, Feb. 1982.

[ADA68] ADAMS, D.A.. *A Computational Model with Data Flow Sequencing.* Technical Report CS 117, School of Humanities and Sience, Standford University. Standford, California, Dec, 1968.

[ALL80] ALLEN, A.O.. *Queueing Models of Computer Systems.* IEEE Computer, v. 13, n. 4, p. 13-24, Apr. 1980

[AGE79] AGERWALA, T.. *Putting Petri Nets to Work.* IEEE Computer, v. 12, n. 12, p. 85-94, Dec. 1979.

[AGE82] AGERWALA, T., ARVIND. *Data Flow Systems.* IEEE Computer, v. 15, n. 2, p. 10-13, Feb. 1982.

[ARV78] ARVIND, GOSTELOW, K.P., PLOUFFE, W.. *An Asynchronous Programming Language and Computing Machine.* Technical Report 114a, Departament of Information and Computer Science. Irvine, California : University of California, Dec. 1978.

[ARV80] ARVIND, THOMAS, R.E.. *I-Strucures: An Efficient Data Type for Functional Languages.* Technical Memo. 178, Laboratory for Computer Science, MIT. Cambridge, Massachusetts : MIT, June 1980.

[BAR84] BARAHONA, P.C.C.. *Performance Evaluation of a Multi-Ring Data Flow Machine.* Ph.D. Thesis, Departament of Computer Science, Manchester : University of Manchester, Oct. 1984.

[BUZ88] BUZATO, L.E., CALSAVARA, C.M.F.R., CATTO, A.J.. *Modelos Computacionais de Fluxo de Dados.* SEMISH 88. Águas de Lindóia, São Paulo, p. 1.1.1-1.1.7 , Set. 1988.

[CHA78] CHANDY, K.M., SAUER, C.H.. *Aproximative Methods for Analysis of Queueing Nework Models of Computer Systems.* ACM Computing Surveys, v. 10, n. 3, p. 281-317, Sept. 1978.

[CAT81] CATTO, A.J.. *Nondeterministic Programming in a Dataflow Enviroment.* Ph.D. Thesis, Departament of Computer Science. Manchester : University of Manchester, June 1981.

[DAS81] DA SILVA, J. C.D., WOODS, J.V.. *Design of a Processing Subsystem for the Manchester Data-Flow Computer.* IEE Proceedings, v. 128, Pt.E, n. 5, p. 218-224, Sept. 1981.

[DAS82] DA SILVA, J.G.D.. *The Matching Unit of the Manchester Data-Flow Computer: A Pseudo Associative Store with Hardware Hashing.* Ph.D. Thesis, Departament of Computer Science. Manchester : University of Manchester, Jan. 1982.

[DAV82] DAVIS, A.L., KELLER, R.M.. *Data Flow Program Graphs.* IEEE Computer, v. 15, n. 2, p. 26-41, Feb. 1982.

[DEN78] DENNING, P.J., BUZEN, J.P.. *The Operational Analysis of Queueing Network Models.* ACM Computing Surveys, v. 10, n. 3, p. 225-261, Sept. 1983.

[DEN70] DENNIS, J.B. (Editor). *Records of the Project MAC Conference on Concurrent Systems and Parallel Computation.* New York : ACM, June 1970.

[DEN75] DENNIS, J.B., MISSUNAS, D.P.. *A Preliminary Architecture for a Basic Data Flow Processor.* In: Proceedings 2nd International Symposium on Computer Architecture, Huston, Texas. New York : IEEE, p. 126-132, Jan. 1975.

[DEN85] DENNIS, J.B.. *Data Flow Computation.* In: Control and Data Flow: Concepts of Distributed Programming. Berlin : Spring-Verlag, 1985 (NATO ASI Series, v. F14). p. 345-398.

[FER90] FERNADES, P.H.L.. *Avaliação de desempenho de Intercoxões de Processadores - Modelos Analíticos para Alocação Simultanêa de Recursos.* Dissertação de mestrado, CPGCC-II. Porto Alegre, BR : UFRGS, Dez. 1990.

[FLY72] FLYNN, M.J.. *Some Computer Organization and Their Effectiviness.* IEEE Tansactions on Computers, v. C-21, n. 9, p. 948-960, Sept. 1972.

[GAJ82] GAJSKI, D.D., PADUA, D.A., KUCK, D.J.. *A Second Opnion on Data Flow Machines and Languages.* IEEE Computer, v. 15, n. 2, p. 58-69, Feb. 1982.

[GEN80] GENRICH, H.J., STANKIEWICZ-WIECHNO, E.. *A Dictionary of some Basic Notion of Net Theory.* In: Net Theory and Applications. Editada por W. Brauer. Berlin : Spring-Verlag, 1980. (Lecture Notes in Computer Science, 84), p. 518-531.

[GEN81] GENRICH, H.J., LAUTENBACH, K.. *System Modelling with High-Level Petri Nets.* Theoretical Computer Science, v. 13, n. 1, p. 109-136, 1981.

[GEN87] GENRICH, H.J.. *Predicate/Transaction Nets.* In: Petri Nets: Central Models and Their Properties. Editada por W. Brauer, W. Reisig and G. Rozenberg. Berlin : Springer-Verlag, 1987 (Lecture Notes in Computer Science, 254), p. 207-247.

[GHO87] GHOSAL, D., BHUYAN, L.N.. *Analytical Modeling and Architecure Modifications of a Data Flow Computer.* Computer Architectures News, v. 15, n. 2, p. 81-89, Feb. 1987.

[GHO90] GHOSAL, D., BHUYAN, L.N.. *Performance Evaluation of a Dataflow Architecture.* IEEE Transaction on Computers, v. 39, n. 5, May 1990.

[GIL84] GILCHRIST, W.. *Statistical Modelling.* Chichester : John Wiley & Sons, 1984. ISBN 0-471-90391-4.

[GOS79] GOSTELOW, K.P., THOMAS, R.E.. *A View of Dataflow.* In: Proceedings of National Computer Conference. New York : IFIPS Press, v. 48, p. 629-636, 1979.

[GOS82] GOSTELOW, K.P.. *The U-Interpreter.* IEEE Computer, v. 15, n. 2, p. 42-49, Feb. 1982.

[GUR78] GURD, J.R., WATSON, I., GLAUERT, J.. *A Multilayered Data Flow Computer Architecture.* Internal Report, Department of Computer Science. Manchester : University of Manchester, July 1978.

[GUR80] GURD, J.R., WATSON, I.. *Data Driven System for High Speed Parallel Computing - Part 2: hardware design.* Computer Design, v. 19, n. 7, p. 97-106, June 1980.

[GUR83] GURD, J.R., WATSON, I.. *Preliminay Evaluation of a Prototype Dataflow Computer.* In: Proceedings 9th IFIP World Computer Congress, [S.l.], p. 545-551, Sept. 1983.

[GUR85A] GURD, J.R., KIRKHAM, C.C., WATSON, I.. *The Manchester Prototype Dataflow Computer.* Communications of the ACM, v. 28, n. 1, p. 34-52, Jan. 1985.

[GUR85B] GURD, J.R.. *The Manchester Dataflow Machine.* In: Future Generation Computer Systems. [S.l.] : North-Holland, p. 201-212, 1985

[GUR86] GURD, J.R., BARAHONA, P.C.C., BÖHM, et al.. *Fine-Grain Parallel Computing: The dataflow Approach.* In: Berlin : Spring-Verlag, 1986 (Lecture Notes in Computer Science), p. 1-71.

[HER87] HERATH, J., YUBA, T., SAITO, N.. *Dataflow Computing.* In: Parallel Algorithms and Architectures. Berlin : Spring-Verlag, 1987 (Lecture Notes in Computer Science), p. 25-36.

[HEU89A] HEUSER, C. A.. *Modelagem Conceitual de Sistemas.* IV Escuela Brasileño-Argentina de Informática, Santiago del Estero. Buenos Aires : Argentina : Kapeluz, 1989.

[HEU89B] HEUSER, C.A., PERES, E.M.. *Rumo a um Modelo Conceitual Completo: Redes de Petri a Diagramas E/R.* Revista de Informática Teórica e Aplicada, v. 1, n. 1, p. 45-77, 1989.

[HOL68] HOLT, A.W., SAINT, H., SHAFIRO, S. et al.. *Final Report of the Information System Theory Project.* Technical Report RADC-TR-68-305, Rome Air Development Center. Griffis Air Force Base, New York, Sept. 1968. (Distibuido por Clearing House for Federal Scientific and Technical Information, US Departament of Commerce).

[HOLT70] HOLT,A.W., COMMONER, F.. *Events & Conditions.* Applied Data Research, New York, 1970.

[HOL87] HOLLIDAY, M.A., VERNON, M.K.. *A Generalized Timed Petri Net Model for Performance Analysis.* IEEE Transaction on Software Engeneering, v. SE-13, n. 12, p. 1297-1310, Dec. 1987.

[JEN81] JENSEN, K.. *Coloured Petri Nets and the Invariant-Method.* Theoretical Computer Science, v. 14, n. 3, p. 317-336, 1981.

[JEN87] JENSEN, K.. *Coloured Petri Nets.* In: Petri Nets: Central Models and Their Properties. Editada por W. Brauer, W. Reisig and G. Rozenberg. Berlin : Spring-Verlag, 1987 (Lecture Notes in Compter Science, 254), p. 248-299.

[KAR66] KARP, R., MILLER, R.. *Properties of a Model for Parallel Computation: Determinancy, Termination, Queueing.* SIAM Journal of Applied Mathematics, v. 14, n. 6, p. 1390-1411, Nov. 1966.

[KAV86] KAVI, K.M., BUCKLES, B.P., BHAT, U.N.. *A Formal Definition of Data Flow Graph Models*. IEEE Transactions On Computers, v. C-35, n. 11, p. 940-948, Nov. 1986.

[KAW86] KAWAKAMI, K., GURD, J.R.. *A Scalable Dataflow Strucute Store.* In: Proceedings 13th Annual Symposium on Computer Architecture, Tokyo. New York : ACM, p. 243-250, June 1986.

[KLE75] KLEINROCK, L.. *Queuing Systems: Volume 1 - Theory*. New York : Wiley, 1975.

[KLE76] KLEINROCK, L.. *Queueing Systems: Volume 2 - Computer Applications.* New York : Wiley, 1976.

[KOB81] KOBAYASHI, H.. *Modelling and Analysis: An Introduction to System Performance Evaluation Methodology.* Reading, Massachusetts : Addison-Wesley, 1981. ISBN 0-201-14457-3.

[LAV83] LAVENBERG, S.S.. *Computer Performance Modelling Handbook.* New York : Academic Press, 1983. ISBN 0-12-438720-9.

[LAU87] LAUTENBACH, K.. *Linear Algebric Techniques for Place/Transation Nets.* In: Petri Nets: Central Models and Their properties. Editada por W. Brauer, W. Reisig and G. Rozenberg. Berlin : Spring-verlag, 1987 (Lecture Notes in Computer Science, 254), p. 142-167.

[LAZ84] LAZOWSKA, E.D., ZAHORJAN, J., GRAHAM, G.S., et al. *Quantitative System Performance: Compter System Analysis Using Queuing Network Models.* Englewood Cliffs, New Jersey : Pentice Hall, 1984. ISBN 0-13-746975-6.

[LIP77] LIPSKI, L., CHURCH, J.D.. *Applications of a Queueing Network Model for a Computer Systems.* ACM Computing Surveys, v. 9, n. 3, p. 205-222 . Sept. 1977.

[MAR84] MARSAN, M. A., CONTE, G., BALBO G.. *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems.* ACM Tansaction on Computer Systems, v. 2, n. 2, p.93-122, May 1984.

[MEM87] MEMMI, G., VAUTHERIN, J.. *Analysing Nets by the Invariant Method.* In: Petri Nets: Central Models and Their Properties, Editada por W. Brauer, W. Reisig and G. Rozenberg. Berlin : Spring-Verlag, 1987 (Lecture Notes in Computer Science, 254), p. 300-336.

[MOL82] MOLLOY, M.. *Performance Analysis Using Stochastic Petri Nets.* IEEE Transaction on Computers, v. C-31, n. 9, p. 913-917, Sept. 1982.

[PET62] PETRI, C.A.. *Kommunikation mit Automaten.* Shriften des Institutes für Instrumentelle Mathematik, Bonn : West Germany, 1962. (In German).

[PET77] PETERSON, J.L.. *Petri Nets.* ACM Computing Surveys, v.9, n. 3, p. 223-252, Sept. 1977.

[PET80] PETERSON, J.L.. *A Note on Colored Petri Nets.* Information Processing Letters, v. 11, n. 1, p.40-43, Aug. 1980.

[PET81] PETERSON, J.L.. *Petri Nets Theory and the Modeling of Systems.* Englewood Cliffs, New Jersey : Prentice Hall, 1981. ISBN 0-13-661983-5.

[REI85A] REISIG, Wolfgang.. *Petri Nets: An introduction.* Berlin : Spring-Verlag, 1985 (EATCS Monographs in Theoretical Computer Science, 4). 160 p. ISBN 3-540-13723-8.

[REI85B] REISIG, W.. *Petri Nets with Induvidual Tokens.* Theoretical Computer Science, v. 41, n. 2-3, p. 185-213, 1985.

[ROD69] RODRIGUEZ, J.E.. *A Graph Model for Parallel Computation.* MIT Technical Report TR-64, Laboratory for Computer Science, MIT. Cambridge, Massachusetts, Set. 1969.

[ROZ80] ROZENBERG, G., THIAGARAJAN, P.S.. *Petri Nets: Basic Notions, Structure, Behavior.* Current Trends In Concurrency. Berlin : Spring-Verlag, 1980 (Lecture Notes in Computer Science, 224), p. 585-668.

[RUG87] RUGGIERO, C.A.. *Throtlle Mechanisms for the Manchester Dataflow Machine.* Ph.D. Thesis, Departament of Computer Science. Manchester : University of Manchester, July 1987.

[SAR86] SARGEANT, J., KIRKHAM, C.C.. *Stored Data Structures on the Manchester Dataflow Machine.* In: Proceedings 13th Annual Symposium on Computer Architecture, Tokyo. New York : ACM, p. 235-242, June 1986.

[SAU80] SAUER, C.H., CHANDY, K.M.. *Aproximative Soltion of Queueing Models.* IEEE Computer, v. 13, n. 4, p. 25-32, Apr. 1980

[SIF80] SIFAKIS, J.. *Performance Evaluation of Systems Using Nets.* In: Net Theory and Applications. Editada por W. Brauer. Berlin : Spring-Verlag, 1980 (Lecture Notes in Computer Science, 84), p.307-319.

[SRI86] SRINI, Vason P.. *An Architectural Comparison of Dataflow Systems.* IEEE Computer, p.68-87, Mar 1986.

[SUL77] SULLIVAN, H. et al.. *Resource Management in a Self-Organizing Parallel Processor.* Poceeding of the IEEE Intenational Conference on Parallel Processing. New York : IEEE, p. 157-162, Aug. 1977.

[TAZ85] TAZZA, M.. *Ein netztheoretisches Modell zur quantitativen Analyse von Systemen. (Q-Modell).* Berichte der GMD, Bericht Nr. 149. R. Oldenbourg Verlag, München, 1985.

[TAZ88] TAZZA, M.. *Análise Quantitativa de Sistemas.* III Escola Brasileiro-Argentina de Informática. Curitiba : Brasil, 1988.

[TRE82] TRELEAVEN, Philip C., BROWNBRIDGE, David R., and HOPKINS, Richard P.. *Data-Driven and Demand-Driven Computer Architecture*. ACM Computing Surveys, v. 14, n. 1, p. 93-143, Mar. 1982.

[TRI80] TRIVEDI, K.S., KINICKI, R.E.. *A Model for Computer Configuration Design.* IEEE Computer, v. 3, n.4, p. 47-54, Apr. 1980.

[VEN86] VENN, Arthur H.. *Dataflow Machine Architecture.* ACM Computing Surveys, v. 18, n. 4, p. 365-396, Dec. 1986.

[WAT82] WATSON, Ian, GURD, John.. *A Practical Dataflow Computer.* IEEE Computer, v. 15, n. 2, p. 51-57, Feb. 1982.