Análise de Controle de Concorrência e Escalonamento de Transações em Bancos de Dados em Tempo-Real Usando Redes de Petri

Pedro Fernandes Ribeiro Neto

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Angelo Perkusich
(orientador)
Maria Lígia Barbosa Perkusich
(orientadora)

Campina Grande, Paraíba, Brasil

A484a Ribeiro Neto, Pedro Fernandes
Analise de controle de concorrencia e escalonamento de transacoes em bancos de dados em tempo-real usando redes de petri / Pedro Fernandes Ribeiro Neto. - Campina Grande, 2001.
78 f.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Petri 2. Bancos de Dados em Tempo-Real 3. Escalonamento - Transacoes 4. Dissertacao - Engenharia Eletrica I. Perkusich, Angelo II. Perkusich, Maria Ligia Barbosa III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 681.3.02(043)

ANÁLISE DE CONTROLE DE CONCORRÊNCIA E ESCALONAMENTO DE TRANSAÇÕES EM BANCO DE DADOS EM TEMPO REAL USANDO REDES DE PETRI

PEDRO FERNANDES RIBEIRO NETO

Dissertação Aprovada em 31.08.2001

PROF. ANGELO PERKUSICH, D.Sc., UFPB
Orientador

PROFA. MARIA LÍGIA BARBOSA PERKUSICH, D.Sc., UNICAP
Orientadora

PROFA. MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFPB
Componente da Banca

PROF. JORGE CESAR ABRANTES DE FIGUEIREDO, D.Sc., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Agosto - 2001

Resumo

Nesse trabalho é apresentado um modelo para analisar o controle de concorrência e o escalonamento de transações para bancos de dados em tempo-real. O modelo é baseado em um método de modelagem para bancos de dados em tempo-real usando redes de Petri Coloridas (CPN).

As características da técnica de controle de concorrência semântico são suportadas pelas restrições das funções de compatibilidade, definidas pelo projetista do sistema, que são suficientes para limitar a imprecisão resultante da negociação entre a consistência lógica e temporal dos dados e das transações.

O modelo CPN para o banco de dados em tempo-real é analisado utilizando cenários e diagramas de seqüências de mensagens e através da geração do espaço de estados para os cenários. São realizados testes de desempenho para verificar se as transações atendem os seus prazos finais.

Abstract

This work presents a model to analyze the scheduling of transactions for real-time databases considering semantic concurrency control mechanisms. The model is based on a modeling method for real-time databases using Colored Petri nets (CPN).

The features of a semantic concurrency control technique are supported by the restrictions of the compatibility functions that are enough to bound the resulting imprecision of the trade-off between both the traditional logical consistency constraints of data and transactions, and the additional temporal consistency constraints of data and transactions.

The CPN Model for real-time databases is analyzed using scenarios and messages sequences charts as well as the generation of the states space for scenarios. Performance tests are executed to verify if the transactions have fulfilled their deadlines.

Agradecimentos

Agradeço ao Professor Angelo Perkusich e a Professora Lígia Perkusich pelo apoio, orientação, dedicação e confiança.

Agradeço aos meus pais, Paulo e Maria Ester, aos meus irmãos Isabelle e Aldo e ao meu cunhado Ricardo, pelo amor e segurança que sempre me proporcionaram.

Agradeço aos professores, das disciplinas do curso, pelos ensinamentos e atenção.

Agradeço a toda família Fotosensores pelo estímulo e perseverança.

Agradeço a UERN pelo incentivo e apoio.

Agradeço aos meus colegas de curso pelo companheirismo e paciência.

Agradeço as minhas sobrinhas, Ana Ester e Ana Carolina, pelos momentos de descontração e lazer.

Agradeço, especialmente, aos meus amados filhos Yasmin, Yngrid e Pedro Filho por serem a razão da minha força de vontade e do meu sucesso.

Dedico essa dissertação a minha querida esposa, Yaskara, que heroicamente soube superar as dificuldades e a minha ausência me estimulando e me dando forças para sempre continuar.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Hoje em dia, aplicações com restrições de tempo real podem ser encontradas em diversas áreas de nosso cotidiano, tais como controle de tráfego aéreo, telecomunicações, geração de energia e sistemas flexíveis de manufatura. Geralmente esses sistemas precisam interagir continuamente com o ambiente para obter dados, que podem ter um tempo de vida útil limitado (restrição temporal dos dados), e utilizá-los no processamento de atividades que devem produzir respostas em tempo-real (restrição temporal das transações).

Essas aplicações são suportadas por sistemas de gerenciamento de bancos de dados para aplicações em tempo-real, ou simplesmente sistemas de gerenciamento de banco de dados em tempo-real, que apresentam as características necessárias para expressar e manter dados e transações com restrições temporais [Ram93; OS95]. Dessa forma, os sistemas de gerenciamento de banco de dados em tempo-real podem ser definidos como sistemas de processamento de transações com restrições explícitas de tempo.

Já os sistemas de gerenciamento de banco de dados (SGBDs) tradicionais disponibilizam mecanismos para implementação de apenas algumas das funções requeridas por essas aplicações. Por exemplo, os SGBDs tradicionais não são adequados para aplicações de tempo crítico, já que não são projetados para suportar transações de tempo-real [Ram93; SSH99]. Muitas vezes, os SGBDs tradicionais produzem tempos de resposta de pior caso, desde que a consistência lógica dos dados esteja garantida [SP95].

As pesquisas na área de sistemas de gerenciamento de banco de dados em tempo-

real têm se intensificado nos últimos anos. Tanto pesquisadores da área de bancos de dados quanto pesquisadores da área de sistemas em tempo-real (STRs) estão dedicando seus estudos visando integrar os benefícios de ambas as áreas [OS95; Ram93; SSH99]. Assim, considerando os mecanismos da tecnologia de banco de dados para tratar com aplicações que envolvem grandes volumes de dados e os avanços obtidos pela tecnologia dos sistemas em tempo-real para processar atividades com restrições temporais, essa integração seria benéfica, já que as transações com restrições temporais poderiam ser exploradas.

Modelos de dados orientados a objetos têm atraído a atenção dos pesquisadores em sistemas de gerenciamento de banco de dados em tempo-real. Um banco de dados orientado a objetos possibilita manter a informação em termos de classes e instâncias dessas classes. Classes e instâncias são denominadas de objetos. Uma classe define atributos e procedimentos pelos quais as instâncias podem ser manipuladas. Os procedimentos associados a uma classe são chamados de métodos que podem invocar outros métodos de outros objetos no banco de dados.

Protocolos de controle de concorrência em bancos de dados orientados a objetos (BDOO) possibilitam o desenvolvimento de técnicas que permitam uma maior concorrência na execução de transações em objetos de dados, através do uso da informação semântica de operações definidas nos objetos. Essas técnicas são denomidadas técnicas de controle de concorrência baseadas em semântica [LS94].

Em geral, modelos de dados orientados a objetos fornecem maiores possibilidades para suportar controle de concorrência baseado em semântica do que os modelos de bancos de dados tradicionais. A possibilidade de incluir operações de complexidade arbitrária, definidas pelo usuário, em representações de objetos de dados permite considerar informação semântica sobre as operações das aplicações, que podem ser exploradas no controle de concorrência [LS94].

Considerando que os objetos de dados em BDOO são freqüentemente grandes e complexos, bloquear um objeto de dados inteiro pode ser ineficiente e desnecessário no contexto de controle de concorrência. Se a semântica das operações for considerada no controle de concorrência, a necessidade de bloquear o objeto de dado inteiro desaparece e muitas operações, que não afetam a mesma parte do objeto, podem ser executadas

concorrentemente sem violar as restrições de consistência lógica do objeto [LS94].

Devido às características citadas para BDOO, o estudo detalhado nessa dissertação foi realizado considerando modelos de sistemas de gerenciamento de banco de dados em tempo-real orientados a objetos, que serão referidos como sistemas de gerenciamento de bancos de dados em tempo-real (SGBD-TR).

1.1 Motivação

Como discutido em [Ram93; OS95; SSH99], a simples integração dos SGBDs e STRs, considerando conceitos, mecanismos e ferramentas, não é suficiente para desenvolver um sistema de gerenciamento de banco de dados em tempo-real. Apesar de muitas das técnicas usadas em STRs e SGBDs poderem ser aplicadas aos sistemas de gerenciamento de banco de dados em tempo-real, há muitas diferenças que requerem adaptações das abordagens usadas nas duas áreas, ou mesmo, o desenvolvimento de novas abordagens. Questões como modelagem conceitual, controle de concorrência, escalonamento, recuperação, entre outras, estão sendo consideradas e pesquisadas para sistemas de gerenciamento de bancos de dados em tempo-real [SSH99; OS95; BLS97; KS96; Jon98].

A necessidade de adaptações, até mesmo da definição, de novas abordagens nos motivou a desenvolver um modelo para os SGBDs-TR visando a possibilidade de analisar algumas dessas abordagens. Uma vantagem do desenvolvimento de modelos é a possibilidade de capturar conhecimentos sobre os sistemas de tal forma que o comportamento desses possa ser analisado de forma automática. No entanto, os modelos podem ser utilizados para outros propósitos diferentes, tais como:

1. documentação de subsistemas, componentes e interfaces;
2. visualização e animação de funcionalidades;
3. verificação de exigências funcionais; e
4. evolução de exigências de tempo e de desempenho.

Um formalismo de modelagem ideal deve suportar todos esses diferentes propósitos. No entanto, muitos formalismos para modelagem usados hoje incorporam apenas um ou dois desses aspectos, exigindo do desenvolvedor esforço e custo adicionais para construir modelos diferentes para o mesmo sistema.

A escolha das redes de Petri, em particular as redes de Petri Coloridas, como formalismo ocorreu devido à sua adequação à modelagem conceitual, à análise, incluindo simulação, de sistemas complexos e concorrentes. Ainda, com o suporte de uma ferramenta computacional para modelagem e análise, é possível não só analisar a estrutura do sistema, mas também seu comportamento dinâmico.

1.2 Objetivos da Dissertação

O objetivo desse trabalho é investigar e definir um modelo, para sistemas de bancos de dados em tempo-real, que permita analisar o escalonamento de transações *periódicas* e *esporádicas* com prazos *estritos* ou *suaves*, considerando mecanismos de controle de concorrência semântico para a negociação entre a consistência lógica e a temporal dos dados e das transações.

O controle de concorrência semântico suporta a consistência temporal das transações por permitir a especificação de escalonamentos mais flexíveis que aqueles permitidos por seriação. Entretanto, o relaxamento da seriação pode resultar em dados imprecisos, além de impor dificuldades para determinar um escalonamento que satisfaça as restrições tanto sobre transações como sobre dados.

Para alcançar essa meta definimos um modelo para banco de dados em tempo-real, usando redes de Petri Coloridas. Esse modelo engloba as características da técnica de controle de concorrência semântico, ao considerar as restrições das funções de compatibilidade suficientes para limitar a imprecisão resultante.

1.3 Relevância

A relevância dessa dissertação é disponibilizar um modelo que possibilite a análise de controle de concorrência e de escalonamento de transações em sistemas de gerencia-

mento de bancos de dados em tempo-real, modelados com redes de Petri Coloridas. A modelagem do SGBD-TR é baseada no método desenvolvido em [Per00].

1.4 Organização da Dissertação

Essa dissertação está organizada como apresentado a seguir. No Capítulo 2, o tema sistemas de bancos de dados em tempo-real é discutido e o trabalho é situado no escopo mais específico de modelagem de sistemas de bancos de dados em tempo-real usando redes de Petri Coloridas. No Capítulo 3 são introduzidos os conceitos necessários ao entendimento das redes de Petri.

No Capítulo 4, o método de modelagem é detalhado e alguns aspectos sobre a implementação do método e da técnica são mostrados de forma mais elaborada. No Capítulo 5 apresentamos a análise do modelo, através de um procedimento para análise de sistemas modelados em redes de Petri Coloridas. Por fim, no Capítulo 6 são apresentadas as conclusões e as propostas de trabalhos futuros.

Capítulo 2

Sistemas de Bancos de Dados em Tempo-real

Em bancos de dados que não tratam com restrições de tempo, a ênfase é na manutenção da consistência lógica dos dados. Em contraste, os sistemas de gerenciamento de banco de dados em tempo-real são direcionados para atender a consistência temporal dos dados e os prazos finais[1] das transações. Nesse capítulo, serão abordados os conceitos básicos de SGBD-TR. Na Seção 2.1 conceituamos os SGBD-TR, bem como apresentamos suas principais características. Nas Seções 2.2 e 2.3 os principais conceitos sobre uma técnica de controle de concorrência semântico e as exigências para o escalonamento são introduzidos.

2.1 Introdução

Os sistemas de gerenciamento de bancos de dados em tempo-real devem englobar além das características de um sistema de banco de dados convencional, que processa transações e garante a integridade dos dados, características de um sistema de tempo-real, que satisfaz às restrições de tempo impostas às transações [BLS97]. Um SGBD-TR possui como principais características o tratamento de dados temporais, ou seja, dados que são válidos apenas por períodos de tempo específicos, e de transações com restrições explícitas de tempo.

[1] *Deadline.*

Comparando os objetivos de projeto entre SGBD e SGBD-TR, temos que, enquanto os SGBDs são projetados objetivando satisfazer às restrições lógicas dos dados e das transações, os SGBD-TR são projetados para satisfazer às restrições temporais dos dados e das transações. Assim, enquanto o tempo de resposta das transações nos SGBDs é considerado *requisito de qualidade*, ou seja, um melhor tempo de resposta é desejável, o tempo de resposta das transações em tempo-real é considerado um *requisito de corretude*. Por outro lado, a consistência lógica é um *requisito de corretude* em bancos de dados convencionais, enquanto que para aplicações em tempo-real, é um *requisito de qualidade*, ou seja, é sempre desejável que os dados sejam consistentes [BLS97].

No entanto como em STR, a velocidade de processamento não é suficiente para o bom desempenho de SGBD-TR [SSH99], ou seja, o fato do *hardware* ser veloz não assegura que as transações serão escalonadas apropriadamente para satisfazer suas restrições temporais, nem assegura que os dados usados são temporalmente válidos. Para um SGBD-TR, uma transação que satisfaz suas restrições temporais usando dados obsoletos é considerada incorreta.

Em SGBD-TR, quando um dado se torna inconsistente ou uma restrição temporal é violada, então o gerenciador deve disponibilizar meios para detectar essa violação e ações de recuperação devem ser tomadas. Essas ações de recuperação devem ser especificadas como parte da definição do banco de dados ou implementadas pela aplicação que está usando o banco de dados [PWPD96].

Em seguida serão comentadas as características dos dados e das transações para SGBD-TR, bem como a redefinição das propriedades ACID[2] das transações.

2.1.1 Características dos Dados

A consistência temporal dos dados exige que o estado atual do ambiente da aplicação e o estado representado pelo conteúdo do banco de dados devam ser próximos o bastante para permanecer em um limite de tolerância aceitável pelas aplicações. A esse aspecto dá-se o nome de *consistência externa*. Em geral, a consistência externa dos dados pode ser: absoluta ou relativa [BLS97].

[2]É o acrônimo para *Atomicidade, Consistência, Isolamento e Durabilidade.*

A *consistência absoluta* é medida entre o estado do ambiente e como ele é refletido no banco de dados. Então, um item de dado deve ser gravado dentro de intervalos de tempo que garantam que ele reflete o estado real do ambiente. Essa medida surge da necessidade de manter a visão do sistema consistente com o estado real do ambiente. Por exemplo, em um sistema de transporte automatizado, o dado correspondente a um sensor que verifica a velocidade dos veículos deve ser atualizado periodicamente, por exemplo, a cada cinco minutos. Assim, o valor da velocidade é absolutamente consistente se ele foi gravado nos últimos cinco minutos.

Já a *consistência relativa* é medida entre os dados que são usados na computação de outros dados. Então, um conjunto de itens de dados deve ser gravado dentro de um mesmo intervalo de tempo que possa representar aproximadamente o mesmo instante de tempo. Essa medida surge da necessidade de produzir novos dados a partir de dados gravados em tempos aproximados. Por exemplo, se o sistema de transporte computa o valor dos níveis de consumo de combustível usando os valores da *velocidade* e a *posição atual* em que se encontra um avião, é importante que esses valores tenham sido gravados aproximadamente no mesmo tempo, por exemplo, nos últimos trinta segundos, de forma que eles possam refletir o mesmo instante do ambiente, ou a computação não fará sentido.

2.1.2 Características das Transações

Como já mencionado, as transações nos SGBD-TR possuem restrições temporais a serem cumpridas. Essas restrições podem determinar prazos estritos[3] e suaves[4]. Normalmente, define-se uma restrição de tempo ou prazo final *estrito* quando qualquer resultado que é produzido após seu prazo é inútil para o sistema e é considerado uma falha fatal. Um prazo *estrito* é observado em transações que, se produzirem seus resultados depois do seu prazo, ocasionarão conseqüências catastróficas. Como exemplos temos: um comando atrasado para parar um trem pode causar uma colisão; uma bomba lançada após o seu tempo pode cair sobre uma população em vez de acertar o alvo militar pretendido. Em contraste, completar uma transação com prazo *suave* após sua

[3] *Hard.*
[4] *Soft.*

restrição de tempo é indesejável. No entanto, não cumprir prazos *suaves* em transações não acarreta problemas sérios; contudo, o desempenho do sistema pode piorar a medida que mais processos com prazos *suaves* deixem de atender suas restrições temporais.

Com a crescente demanda pela utilização de bancos de dados em tempo-real as definições para os prazos das transações estão sendo reconsiderados. Por exemplo, para uma aplicação na bolsa de valores, perder o prazo para uma compra ou venda de milhares de dólares em ações pode resultar em um grande prejuízo, sem necessariamente ser uma catástrofe. Assim, pode-se definir uma transação *estrita* como sendo aquela que não pode perder seu prazo final, isto é, ela tem que atender suas restrições de tempo. Por outro lado, se o prazo pode ser perdido ocasionalmente com alguma probabilidade aceitável, então essa restrição de tempo é *suave* [Liu00].

Muitas transações em STR são para gravar as leituras dos dispositivos associados ao ambiente ou para manipular eventos do sistema. Geralmente, em bancos de dados em tempo-real, essas transações são executadas *periodicamente*. Nas transações *periódicas* cada computação é executada repetidamente em um intervalo de tempo regular para desempenhar uma função do sistema; intervalo de tempo que não existe na execução das transações *aperiódicas*. Essas possuem prazos suaves ou não possuem prazos, e terminar cada execução o mais cedo possível é o ideal, todavia a resposta atrasada pode ser tolerável. As transações *esporádicas* são iniciadas em instantes de tempo aleatórios, no entanto uma vez executadas possuem prazos *estritos*.

De modo geral, as transações em SGBD-TR também podem ser classificadas como: transações de escrita (ou sensores), transações de atualização e transações de leitura [Ram93]. As transações de escrita, tipicamente periódicas, obtém o estado do ambiente e escrevem os dados no banco de dados. As transações de atualização tanto podem ler quanto escrever no banco de dados periódica ou aperiodicamente. As transações de leitura, lêem dados do banco de dados e também podem ser periódicas ou aperiódicas.

2.1.3 Propriedades ACID

Um projeto de banco de dados convencional deve garantir a manutenção da consistência dos dados, isto é, evitar informações contraditórias para diferentes transações no mesmo instante de tempo e manter a integridade do sistema. Em banco de dados isso é definido

como *consistência interna*. Para garantir a *consistência interna*, em um banco de dados deve-se implementar corretamente todas as transações para assegurar as propriedades ACID. As propriedades ACID são definidas a seguir:

- *Atomicidade*: uma transação deve ser totalmente executada ou nenhum passo dela deve ser considerado;
- *Consistência*: a execução de uma transação deve sempre transformar o estado consistente de um banco de dados em outro estado consistente;
- *Isolamento*: as ações de uma transação não devem ser visíveis por nenhuma outra transação até que ela seja terminada;
- *Durabilidade*: as ações de uma transação devem ser permanentes.

Em STR, monitorar o ambiente e controlar os dispositivos são funções importantes. Portanto, manter a consistência externa do banco de dados pode, as vezes, exigir que a consistência interna e o critério de corretude para a execução das transações, denominado *seriação*, sejam violados. Isso pode ser observado em situações em que as transações com inconsistência interna, tiverem que bloquear ou desfazer algumas operações. Em se tratando de operações do mundo real bloquear ou desfazê-las, pode ser impossível. Portanto, para atender as necessidades das aplicações em tempo-real, as propriedades ACID foram redefinidas para SGBD-TR, de forma a permitir melhor suporte para consistência temporal, enquanto mantém suporte para consistência lógica [DiP95; BLS97].

Para transações em tempo-real, a execução atômica é seletivamente aplicada às subtransações que necessitam tratar com dados totalmente consistentes, ao invés de aplicá-la à transação toda. Por exemplo, considere que uma transação que está atualizando dados sobre o ambiente, perde seu prazo e deve parar de executar. Geralmente, é desnecessário desfazer as operações, uma vez que os valores anteriores também estão desatualizados. Além do mais, pode ser mais interessante dispor de um banco de dados parcialmente atualizado do que de um banco de dados totalmente desatualizado.

As transações em SGBD-TR devem suportar uma negociação entre manter consistência lógica ou temporal. Essa negociação pode introduzir imprecisão no banco de

dados. No entanto, aplicações em tempo-real podem suportar inconsistências, tais como: dados que mudam freqüentemente não precisam ser atualizados a cada mudança, o que consumiria tempo de computação e recursos. Geralmente nessas aplicações as atualizações são realizadas apenas quando dados atualizados são necessários.

A propriedade de isolamento que garante que não há interdependências na execução das transações pode não ser interessante em SGBD-TR. As transações podem precisar se comunicar e sincronizar com outras transações de forma a executar funções de controle, portanto, suas ações podem ser vistas por outras transações mesmo antes delas terem terminado. Além disso, algumas transações podem ter restrições temporais e podem usar dados que precisam ser temporalmente consistentes. Assim, os dados gerados por uma transação não terminada podem ser vistos por outras transações para evitar que se tornem desatualizados ou que não satisfaçam suas restrições temporais.

Quanto à propriedade de durabilidade, em SGBD-TR, alguns dados têm validade temporal e não precisam ser gravados. Por outro lado, o banco de dados deve refletir o estado do ambiente, portanto, é fácil recriá-lo a partir da leitura dos sensores, ao invés de recriá-lo no tempo em que ocorreu uma falha, pois esses dados provavelmente já serão inválidos.

2.2 Técnica de Controle de Concorrência Semântico

Nessa seção será discutido como o controle de concorrência semântico define a negociação entre a consistência lógica e temporal dos dados e transações. O critério de corretude para a execução das transações também será discutido.

Uma técnica de controle de concorrência semântico utiliza conhecimento específico sobre uma aplicação de forma a aumentar a concorrência na execução das transações. Em [DiP95] apresenta-se uma técnica de controle de concorrência semântico orientada a objetos, denominada *técnica de bloqueio semântico*, que suporta consistência lógica e temporal dos dados e transações e permite a negociação entre elas. A técnica possibilita também expressar a imprecisão resultante dessa negociação. Essa técnica usa bloqueio semântico para determinar quais transações podem invocar métodos em um objeto. O bloqueio semântico é controlado em cada objeto individualmente por uma *função*

de compatibilidade (*FC*) que controla o acesso concorrente aos seus métodos, *FC* é apresentada com detalhes na Seção 2.2.2.

Essa técnica suporta consistência lógica de dados e transações através da definição de funções similares às técnicas de controle de concorrência para banco de dados convencionais. Ela suporta consistência temporal dos dados ao permitir a negociação entre a consistência lógica e a consistência temporal e suporta consistência temporal das transações por permitir a especificação de escalonamentos mais flexíveis que aqueles permitidos por seriação. Além disso, essa técnica pode especificar e limitar alguma imprecisão, definida no contexto de *Seriação Epsilon* [RP95], apresentada na Seção 2.2.1, que possa surgir no sistema devido ao relaxamento da seriação.

Nos últimos anos, muitas outras técnicas para controle de concorrência semântico foram propostas, algumas delas podem ser vistas em [WA92; YWLS94]. No entanto nenhuma delas suporta a consistência temporal dos dados e a negociação entre consistência lógica e temporal das transações, tal como a técnica de bloqueio semântico, apresentada em [DiP95].

2.2.1 Imprecisão Limitada

Em sistemas em tempo-real, muitas vezes resultados imprecisos podem ser aceitáveis para permitir a satisfação das restrições temporais das transações. Isso impõe às técnicas de controle de concorrência, para tais sistemas, a necessidade de controlar e limitar a imprecisão resultante. Muitas das técnicas para controle de concorrência são baseadas no critério de corretude (*Seriação Epsilon* [RP95]), apresentada a seguir.

Seriação *Epsilon (Epsilon Serializability - ES)*

ES [RP95] é um critério de corretude formal que especifica que um escalonamento para transações é correto se o resultado do escalonamento (ambos, valores dos dados e valores dos argumentos de retorno de transações) está dentro de limites especificados, ou seja, ele generaliza seriação por permitir imprecisão limitada no processamento de transações. Para acumular e limitar imprecisão, esse critério assume usar apenas dados mensuráveis.

Uma transação especifica a quantidade de imprecisão que ela pode importar e exportar em um item de dado. A quantidade máxima de imprecisão que uma transação t pode importar para um item x é definida como $lim\_imp_{t,x}$, e a quantidade máxima de imprecisão que uma transação t pode exportar para um item x é definida como $lim\_exp_{t,x}$. Para cada item de dados x no banco de dados, lim_max_x é definido como o limite máximo de imprecisão que pode ser escrito em x.

A quantidade de imprecisão importada e exportada por cada transação, assim como as imprecisões escritas nos itens de dados, devem ser acumuladas durante a execução da transação. A quantidade de imprecisão importada pela transação t para um item de dados x é definida por $imprec\_imp_{t,x}$, e a quantidade de imprecisão exportada pela transação t para um item de dados x é definida por $imprec\_exp_{t,x}$. A quantidade de imprecisão escrita em um item de dados x é definida como $quant\_imp_x$.

As propriedades de segurança definidas para as transações e os dados especificam os limites de imprecisão para transações e dados. Dado um item de dados x, a seguinte propriedade define um dado *seguro*:

$$x : quant\_imp_x \leq lim\_max_x$$

Essa propriedade indica que a imprecisão no item de dados x é aceitável, se ela não for maior que o limite especificado.

Dada uma transação t e um item de dados x, as seguintes propriedades definem uma transação *segura*:

$$(t,x)_{leitura} : imprec\_imp_{t,x} \leq lim\_imp_{t,x}$$

$$(t,x)_{escrita} : imprec\_exp_{t,x} \leq lim\_exp_{t,x}$$

Essa propriedade indica que a imprecisão nos itens de dados lidos ou escritos por uma transação é aceitável se eles não forem maiores que os limites especificados para eles. Nesse contexto, dizemos que *ES* está garantida, se e somente se, todos os dados e transações são seguros.

As definições acima são aplicáveis à sistemas baseados em transações. Essas definições foram adaptadas para sistemas orientados a objetos e serão definidas a seguir.

Objeto

Um objeto é uma quíntupla $\langle N, AT, MT, R, FC \rangle$, onde:

1. N é um identificador único para um objeto.

2. AT é um conjunto de atributos, e $\forall\ a \in AT, a = \langle N, VAL, TEMPO, AVI, QUANT\_IMP \rangle$, onde:

   N é o nome do atributo;

   VAL é o valor do atributo;

   $TEMPO$ é o rótulo de tempo, que determina quando o valor do atributo foi gravado;

   AVI é o intervalo de validade absoluta para o atributo, ou seja, determina por quanto tempo o valor do atributo é válido;

   $QUANT\_IMP$ é a imprecisão acumulada para o atributo.

3. MT é um conjunto de métodos, e $\forall\ mt_i \in MT$, mt_i possui um conjunto de argumentos Arg_e e Arg_r, onde:

   $\forall\ Arg_e = \langle (N, VAL, TEMPO, AVI, QUANT\_IMP), LIM\_EXP \rangle$, onde N, VAL, $TEMPO$, AVI, e $QUANT\_IMP$ são como definidos para AT, e LIM_EXP_e especifica o limite máximo de imprecisão que pode ser passado pelo método.

   $\forall\ Arg_r = \langle (N, VAL, TEMPO, AVI, QUANT\_IMP), LIM\_IMP \rangle$, onde N, VAL, $TEMPO$, AVI, e $QUANT\_IMP$ são como definidos para AT, e LIM_IMP_r especifica o limite máximo de imprecisão permitido no valor retornado.

4. R é um conjunto de restrições lógicas e temporais que definem o estado correto de uma instância de um objeto. Uma restrição é definida como um predicado que pode incluir os campos VAL, $TEMPO$, AVI e $QUANT\_IMP$, de um atributo. Através desses predicados declaram-se as restrições lógicas e temporais, bem como os limites de imprecisão para os atributos.

5. FC, definida em 2.2.2, é uma função de compatibilidade que utiliza informação semântica sobre os objetos e o estado do sistema para determinar quando dois métodos de um objeto podem ser executados concorrentemente.

Seriação *Epsilon* Orientada a Objetos (SEOO)

Em modelos de dados orientados a objetos, os dados são representados por objetos. Um objeto é dito *seguro* se obedece a seguinte propriedade:

$$objeto_o : \forall a \in o_A(a.quant\_imp \leq lim\_max_a)$$

onde o_A é o conjunto de atributos de o e lim_max_a é o limite máximo de imprecisão que pode ser escrito em a.

Assim, se cada atributo em um objeto satisfaz a imprecisão especificada, então o objeto é *seguro*.

Em modelos orientados a objetos, as transações operam sobre os objetos através dos métodos dos objetos. Os valores dos itens de dados são obtidos dos objetos através dos argumentos de retorno dos métodos, e são passados para os objetos através dos argumentos de entrada dos métodos. Seja t_{MI} o conjunto de métodos invocados por uma transação t e o_M o conjunto de métodos em um objeto o. Seja $t_{MI} \sqcap o_M$, os métodos de o invocados por t. Uma transação t é *segura* em relação a um objeto o se obedece as seguintes propriedades:

$$(t,o)_{leitura} : \forall m \in (t_{MI} \sqcap o_M) \forall r \in arg\_retorno(m)(r.quant\_imp \leq lim\_imp_r)$$

$$(t,o)_{escrita} : \forall m \in (t_{MI} \sqcap o_M) \forall e \in arg\_entrada(m)(e.quant\_imp \leq lim\_exp_e)$$

Essa propriedade indica que se os argumentos dos métodos invocados por t em um objeto o estão dentro de seus limites de imprecisão, então t é segura para o.

Assim, SEOO é garantida se e somente se todos os objetos e transações dos objetos são seguros.

2.2.2 Função de Compatibilidade

A função de compatibilidade (FC) para um objeto, é definida para todo par ordenado de métodos e é avaliada em tempo de execução. É definida pelo projetista do sistema, para expressar compatibilidade entre a execução concorrente dos métodos de um objeto. A função tem a forma:

$$FC(m_{ati}, m_{inv}) = Expressão\ Booleana$$

onde m_{ati} representa o método que está sendo executado e m_{inv} representa o método que foi invocado. A expressão booleana pode conter predicados envolvendo valores dos argumentos dos métodos, do banco de dados e do sistema em geral.

O conceito de *conjunto afetado* foi introduzido em [BR88] e é usado como uma base para representar o conjunto de atributos de um objeto que um método lê ou escreve. Cada método m tem um *conjunto de leitura afetado* ($CLA(m)$), isto é, o conjunto de atributos do objeto que serão lidos pelo método, e um *conjunto de escrita afetado* ($CEA(m)$), isto é, o conjunto de atributos do objeto que será escrito pelo método.

Em adição, além de expressar a compatibilidade entre a invocação de dois métodos, a técnica de bloqueio semântico possibilita que a função de compatibilidade expresse informações sobre a imprecisão máxima acumulada que pode ser introduzida com a execução concorrente dos métodos. Considerando isso, a FC, passa a ter a seguinte forma:

$$FC(m_{ati}, m_{inv}) = Expressão\ Booleana \Rightarrow IA$$

onde IA representa a imprecisão acumulada.

A FC pode expressar para as invocações dos métodos m_{ati} e m_{inv} [DiP95], três diferentes situações:

1. Imprecisão no valor de um atributo que está no conjunto de escrita afetado (CEA) de m_{ati} e m_{inv}, isto é, m_{ati} e m_{inv} escrevem no mesmo atributo.

2. Imprecisão no valor do argumento de retorno de m_{ati}, quando m_{ati} lê atributos escritos por m_{inv}.

3. Imprecisão no valor do argumento de retorno de m_{inv}, quando m_{inv} lê atributos escritos por m_{ati}.

Para ilustrar essas situações, serão apresentadas as funções de compatibilidade que expressam compatibilidade condicional para a execução concorrente dos métodos.

Exemplo 1

A função de compatibilidade 2.1 expressa uma negociação de consistência lógica por consistência temporal quando o método *LerDado* (Ler um dado qualquer) está sendo executado e o método *AtuDado* (Atualizar dado) é invocado. Sob seriação, esses métodos não poderiam ser executados concorrentemente, pois a visão do atributo *dado* pelo método *LerDado* seria prejudicada. No entanto, se a validade temporal do atributo *dado* for violada, é importante permitir a execução do método *AtuDado* para restaurar sua consistência. Porém, os dois métodos só podem ser executados concorrentemente se o valor do atributo *dado* escrito por *AtuDado* ($A.val$) é próximo do valor corrente do atributo *dado* ($dado.val$). Essa determinação é baseada no valor corrente do atributo *dado* ($dado.val$), no limite máximo de imprecisão permitido para o argumento de retorno L de *LerDado* (lim_imp_L), na quantidade de imprecisão que *AtuDado* irá escrever em *dado* através de A ($A.quant\_imp$) e na quantidade de imprecisão existente no argumento de retorno ($L.quant\_imp$). A imprecisão acumulada, resultante da execução concorrente dos dois métodos, também é mostrada. Nesse caso, o argumento de retorno L do método *LerDado* tem um aumento de imprecisão igual à diferença entre o valor do atributo *dado* antes da atualização ($dado.val$) e o novo valor do atributo após a atualização ($A.val$), mais a quantidade de imprecisão que é escrita no atributo *dado* por *AtuDado*, ($A.quant\_imp$).

$$
\begin{aligned}
FC(LerDado(L), AtuDado(A)) = {} & (agora - dado.tempo \leq dado.avi) \\
& \wedge (|dado.val - A.val| \leq (lim\_imp_L \\
& - (L.quant\_imp + A.quant\_imp))) \\
& \Rightarrow L.quant\_imp = L.quant\_imp \\
& + A.quant\_imp + |dado.val - A.val|
\end{aligned}
\tag{2.1}
$$

Exemplo 2

Na função de compatibilidade 2.2, o método *AtuDado* atualiza o valor do atributo *dado* com o valor do argumento de entrada A e o método *LerDado* lê o valor do atributo *dado*. Esses métodos só poderão ser executados concorrentemente se a diferença entre o novo valor do dado ($A.val$) e o valor original ($dado.val$) estiver dentro do limite de imprecisão aceito pelo argumento de retorno L do método *LerDado* (lim_imp_L). Essa determinação é baseada no valor atualizado por *AtuDado* ($A.val$), no limite de imprecisão permitido para o argumento de retorno de *LerDado* (lim_imp_L), e na quantidade de imprecisão acumulada no argumento de retorno de *LerDado* ($L.quant\_imp$). A imprecisão acumulada no argumento de retorno L do método *LerDado* tem um aumento igual à diferença entre o valor original ($dado.val$) e o novo valor ($A.val$), caso os métodos sejam executados concorrentemente.

$$FC(AtuDado(A), LerDado(L)) = |dado.val - A.val| \leq lim\_imp_L - L.quant\_imp) \Rightarrow L.quant\_imp = L.quant\_imp + |dado.val - A.val| \quad (2.2)$$

Exemplo 3

A função de compatibilidade 2.3 mostra como um atributo pode se tornar impreciso. Duas invocações de um mesmo método *AtuDado* podem ocorrer concorrentemente se um sensor atualiza o atributo *dado* e um operador humano também solicita a execução de uma operação de escrita nesse mesmo atributo. A função de compatibilidade indica que duas invocações do método *AtuDado* podem ser executadas concorrentemente desde que as diferenças entre os valores escritos, quando executando as duas invocações, não excedam a quantidade de imprecisão permitida para *dado*. A determinação de tal imprecisão é baseada nos valores escritos pelas duas invocações do método ($A_1.val$ e $A_2.val$), na imprecisão existente em *dado* ($dado.quant\_imp$) e no limite de imprecisão permitido para *dado* (lim_max_{dado}). A imprecisão acumulada no atributo *dado* tem um acréscimo igual a diferença entre os valores escritos pelas duas invocações, se os métodos forem executados concorrentemente.

$$
\begin{aligned}
FC(AtuDado_1(A_1), AtuDado_2(A_2)) = (&|A_1.val - A_2.val| \\
&\leqslant (lim\_max_{dado} - dado.quant\_imp)) \\
&\Rightarrow dado.quant\_imp = dado.quant\_imp \\
&+ |A_1.val - A_2.val|
\end{aligned}
\tag{2.3}
$$

2.2.3 Seriação

Suponha que dois usuários, de um sistema de reserva de passagens aéreas, submetem as transações ilustradas na Figura 2.1. Se a intercalação entre a execução das operações não é possível, há somente duas maneiras de ordenar as operações das duas transações:

```
ler_item (X);
X:=X-N;
escrever_item(X);
ler_item (Y);
Y:=Y+N;
escrever_item (Y);
```

(a)

```
ler_item (X);
X:=X+M;
escrever_item(X);
```

(b)

Figura 2.1: (a) Transação **T1**. (b) Transação **T2**.

1. Executar todas as operações de **T1** (em seqüência) seguidas por todas as operações de **T2** (em seqüência);

2. Executar todas as operações de **T2** (em seqüência) seguidas por todas as operações de **T1** (em seqüência);

Essas alternativas são ilustradas nas Figuras 2.2(a) e 2.2(b), respectivamente. Se a intercalação das operações for permitida, então, existirá muitas ordenações possíveis na qual o sistema poderá executar as operações das transações. Dois escalonamentos, dos possíveis, são mostrados nas Figuras 2.2(c) e 2.2(d).

Um aspecto importante de controle de concorrência, chamado *Teoria da Seriação* [EN94], tenta determinar quais escalonamentos são corretos e quais não são e procura desenvolver técnicas que permitam somente escalonamentos corretos.

Os escalonamentos A e B, na Figura 2.2(a) e 2.2(b), são chamados de *escalonamentos seriais* porque as operações de cada transação são executadas consecutivamente, sem qualquer intercalação das operações de outras transações. No escalonamento serial, transações inteiras são executadas em uma ordem serial: **T1** e, então, **T2** na Figura 2.2(a), e **T2** e, então, **T1** na Figura 2.2(b). Os escalonamentos C e D nas Figuras 2.2(c) e 2.2(d) são chamados de *não-seriais*, já que as operações são intercaladas, quando da execução de suas transações.

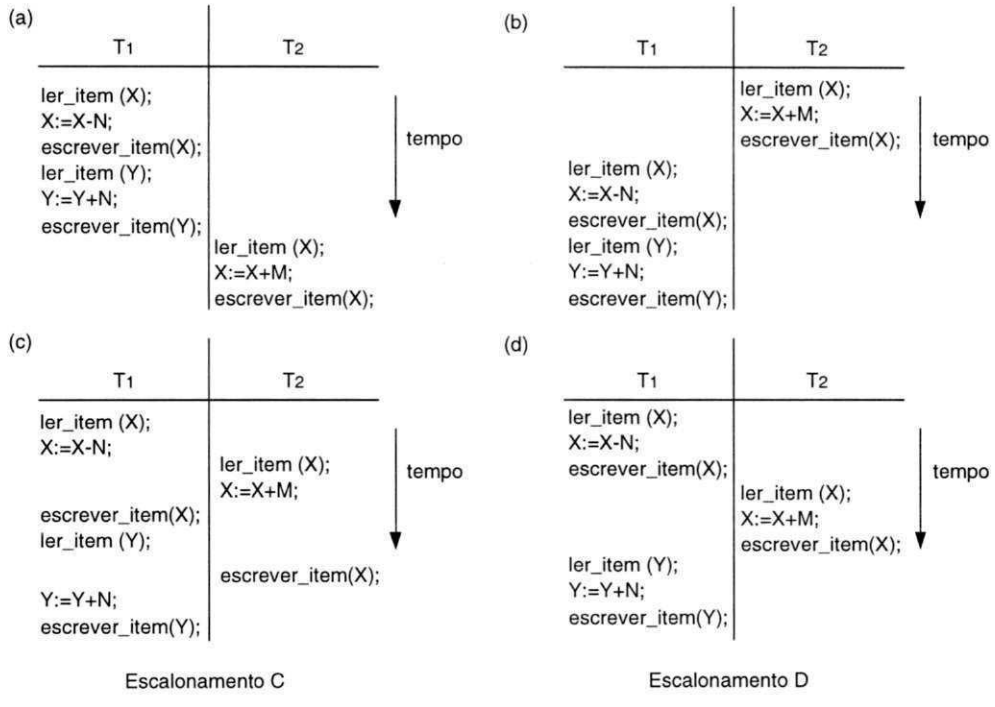

Figura 2.2: (a) Escalonamento A: **T1** seguido por **T2**. (b) Escalonamento B: **T2** seguido por **T1**. (c) e (d) Dois escalonamentos com intercalação de operações.

Um escalonamento S é *serial* se, para cada transação T participante do escalonamento, todas as operações de T são executadas consecutivamente no escalonamento; caso contrário, o escalonamento é *não serial* [EN94]. O problema com escalonamento serial são os seus limites de concorrência ou intercalação das operações. No escalonamento serial, se uma transação espera por uma operação de *entrada e saída* para terminar, não se pode mudar o processador para executar outra transação, isso desperdiça um valioso tempo de processamento e faz com que o escalonamento serial não seja aceitável.

Para ilustrar essa discussão, considere os escalonamentos na Figura 2.2, e assuma que o valor inicial dos itens do banco de dados são X = 90, Y = 90 e que N = 3 e M = 2. Após a execução das transações T1 e T2, é esperado que os valores no banco de dados fossem X = 89 e Y = 93. Certo, se os escalonamentos seriais A ou B forem considerados. Agora considere os escalonamentos não-seriais C e D. No escalonamento C tem-se X = 92 e Y = 93, onde o valor de X está errado, já no escalonamento D obtem-se o valor correto.

No escalonamento C, o valor de X está errado, porque a transação T2 lê o valor de X antes desse ser mudado pela transação T1, dessa forma somente o valor de T2, em X, é refletido no banco de dados. O efeito de T1, em X, é perdido ou sobrescrito por T2, levando o resultado incorreto para o item X. Entretanto, mesmo escalonamentos não-seriais podem produzir resultados corretos. Seria interessante determinar quais escalonamentos não-seriais dão o resultado correto e quais dão resultados errados. O conceito usado para caracterizar escalonamentos dessa maneira é o de *seriação* do escalonamento.

Um escalonamento S de n transações é seriável se for equivalente a um escalonamento serial das mesmas n transações. Note que existem $(n)!$ possíveis escalonamentos seriais para n transações e muito mais escalonamentos não-seriais possíveis. Pode-se, então, formar dois grupos diferentes de escalonamentos não-seriais: os que são equivalentes a um (ou mais) escalonamento serial, sendo seriável; e os que não são equivalentes a qualquer escalonamento serial, não sendo seriável.

2.3 Política de Escalonamento

Em se tratando de bancos de dados que têm que processar transações com restrições de tempo para os dados e transações, as políticas de escalonamento para as transações devem obedecer à duas exigências conflitantes. A primeira é a preservação da consistência lógica do banco de dados por assegurar a execução das transações em tempo-real de maneira seriável. A segunda exigência é a de satisfazer às restrições temporais das transações, ou seja, atender seus prazos. Para isso dois escalonamentos estão envolvidos, são eles: escalonamento de unidade central de processamento (UCP)

e escalonamento de dados. Em seguida discutiremos esses escalonamentos.

2.3.1 Escalonamento de UCP

Em sistemas de tempo-real, muitos dos algoritmos de escalonamento de UCP propostos na literatura são baseados em prioridades de transações [LL99]. As prioridades das transações podem ser determinadas por sua urgência e estado crítico[5]. Os escalonamentos de UCP mais utilizados são *Earliest Deadline First* (EDF), *Least Slack Time* (LST) e *Rate Monotonic* (RM). A definição dessas políticas de escalonamento pode ser encontrada com maiores detalhes em [Liu00; KS96].

2.3.2 Escalonamento de Dados

Escalonamentos de dados são, principalmente, controlados por protocolos de controle de concorrência (PCC). O objetivo desses protocolos é de manter a consistência lógica no banco de dados e prover uma maior concorrência ao mesmo tempo. Muitos dos estudos em PCC usam seriação, como critério de corretude, para manter a consistência dos dados [AM92; BLS97]. Não há um critério para manter a consistência lógica dos dados disponível, tal que seja menos rigoroso que seriação e ainda obviamente correto e implementável.

A integração de escalonamento de UCP e escalonamento de dados não é simples em SGBD-TR. PCC tradicionais não são suficientes para suportar execuções de transações em SGBD-TR porque eles não consideram prazos finais de transações ou prioridades em escalonamento de dados. Diferentes PCC têm sido propostos, tais como: *Two-Phase Locking* (2PL) [KS96] e *Optimistic Concurrency Control* (OCC) [BLS97; KS96]. O principal problema com 2PL, em SGBD-TR, é a possibilidade de ocorrer inversão de prioridade, em que uma transação de alta prioridade é bloqueada por uma de menor prioridade. Isso pode ser resolvido ao reiniciar ou elevar a prioridade da transação de baixa prioridade.

As propriedades de não bloqueio e livre de impasse[6] para OCC faz com que esse se torne muito atrativo para ser aplicado em SGBD-TR. Vários OCC de tempo-real

[5] *Criticality.*
[6] *Deadlock.*

têm sido propostos, uma vez que eles são mais satisfatórios que 2PL para os SGBD-TR [LL99]. Para OCC, a resolução de conflitos entre transações é atrasada até que uma transação esteja pronta para completar, com isso, existirão mais informações disponíveis para decidir sobre qual transação deve ser reiniciada. A resolução de conflitos baseada em reinício no protocolo OCC tem o problema de sobrecarga de reinício. Isso traz um impacto negativo no sistema, já que o tempo de execução das transações reiniciadas aumenta muito e muitos dos recursos do sistema que são consumidos por essas transações são desperdiçados. Também, devido o fato de poder afetar a continuidade das transações que compartilham recursos no sistema com as transações que reiniciam.

Alguns recentes trabalhos mostram a diminuição do número de reinícios por ajustar a ordem de seriação entre as transações [BLS97]. Trata-se de protocolos que ajustam dinamicamente a ordem de seriação, em favor das transações com alta prioridade, evitando bloqueios e reinícios, resultando em um protocolo que mescla ordem de seriação com a ordem de prioridade das transações.

Capítulo 3

Redes de Petri

Nesse capítulo são apresentados os conceitos básicos de redes de Petri e suas extensões suficientes ao bom entendimento do trabalho. O capítulo está dividido em três seções: na Seção 3.1 são apresentadas as definições de redes de Petri Lugar/Transição, sendo essas uma base para a perfeita compreensão da Seção 3.2 onde são definidas algumas extensões às redes de Petri e na Seção 3.3 comenta-se uma ferramenta para manipular redes de Petri Coloridas, já que essas são a extensão de redes de Petri utilizada na dissertação.

3.1 Introdução às Redes de Petri

Um modelo é uma representação, geralmente em termos matemáticos, das principais características de um objeto ou sistema. Através da análise do modelo, um sistema real pode ser estudado sem o perigo, custo ou inconveniência da manipulação de seus elementos. Dentre as técnicas formais para modelar e analisar sistemas, podemos citar, por exemplo, as redes de Petri.

As redes de Petri devem seu nome ao trabalho de Carl Adam Petri que em sua tese de doutorado, apresentou um tipo de grafo bipartido[1] com os estados associados, com o objetivo de estudar a comunicação entre os autômatos [Mur89]. O seu desenvolvimen-

[1]Um grafo G é denominado bipartido quando os seus nós podem ser divididos em dois conjuntos $N1$ e $N2$, tais que nenhum nó contido em $N1$ ou $N2$ se encontra ligado a outro nó contido no mesmo conjunto.

to posterior se deu pelas suas numerosas potencialidades de modelagem, dentre elas: sincronização de processos, concorrência, conflitos e compartilhamento de recursos.

Como ferramenta matemática e gráfica, as redes de Petri oferecem um ambiente uniforme para modelagem, análise formal e simulação de sistemas a eventos discretos, permitindo uma visualização simultânea de sua estrutura e comportamento. Mais especificamente, as redes de Petri modelam dois aspectos desses sistemas, eventos e condições, bem como as relações entre eles. Segundo essa caracterização, em cada estado do sistema verificam-se determinadas condições. Essas podem possibilitar a ocorrência de eventos que por sua vez podem ocasionar a mudança de estado do sistema. É possível relacionar, de forma intuitiva, condições e eventos com os dois tipos de nós da rede, respectivamente, *lugares* e *transições.*

Estrutura da Rede

Uma rede de Petri é composta de uma *estrutura de rede*, *inscrições* associadas a essa estrutura e uma *marcação.* A estrutura de rede e a marcação definem a sintaxe de uma rede de Petri. A evolução de suas marcações, segundo uma *regra de ocorrência*, estabelece a sua semântica.

A estrutura de uma rede de Petri pode ser representada por um grafo bipartido direcionado, cujos elementos são de dois tipos distintos: *lugares* e *transições.* Os lugares são graficamente representados por círculos ou elipses e as transições por retângulos.

Uma estrutura de rede de Petri é uma tripla $N = \langle P, T, F \rangle$, na qual:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;
- $F \subseteq P \times T \cup T \times P$ é uma relação de fluxo;
- $P \cap T = \varnothing$.

Na Figura 3.1, apresenta-se uma rede de Petri Lugar/Transição. Nessa pode-se observar dois tipos de inscrições: o *peso dos arcos* e a *marcação.* O peso dos arcos equivale a um inteiro associado a cada arco direcionado. Na figura, o peso do arco direcionado entre o lugar **p1** e a transição **t2** é 2. Já a marcação equivale ao estado

da rede e é representada por elementos associados aos lugares, denominados *fichas*. As fichas, pequenos círculos pretos, estabelecem a marcação da rede. A formalização para essas inscrições é definida a seguir:

Uma rede de Petri Lugar/Transição é uma tripla $PN = \langle N, W, M_0 \rangle$, na qual:

- N *é uma estrutura de rede de Petri;*
- $W : F \rightarrow \{1, 2, 3, ...\}$ *é uma função de peso;*
- $M_0 : P \rightarrow \{0, 1, 2, 3, ...\}$ *é a marcação inicial.*

A função das transições é remover e/ou adicionar fichas e é através de sua ação, denominada *ocorrência*, que a marcação da rede pode ser alterada.

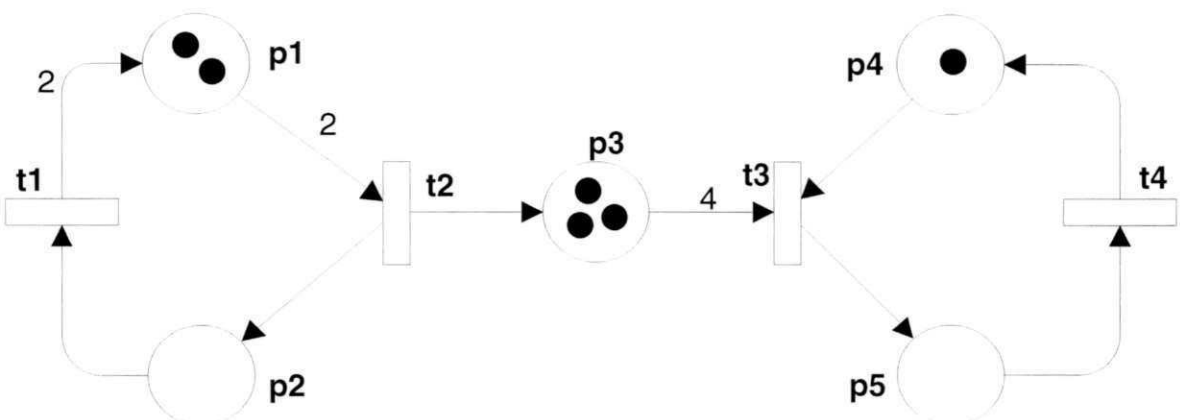

Figura 3.1: Rede de Petri Lugar/Transição

Evolução da Rede

Um arco orientado de um lugar para uma transição é chamado de *arco de entrada* e indica que essa transição pode remover, quando da sua ocorrência, fichas desse lugar. De forma simétrica, um arco orientado de uma transição para um lugar é chamado de *arco de saída* e indica que essa transição pode adicionar, quando da sua ocorrência, fichas a esse lugar.

Uma transição só pode ocorrer se em cada lugar de entrada, para essa transição, há a quantidade de fichas, pelo menos igual ao peso do arco. Quando isso ocorre, diz-se que a transição está *habilitada* a ocorrer. Ao ocorrer, uma quantidade de fichas,

igual ao peso do arco de entrada, é removida e uma quantidade de fichas, igual aos pesos dos arcos de saída, é acrescentada a cada lugar de saída. Para exemplificar o que foi discutido, ilustramos a Figura 3.2, que é um estado seguinte ao estado da rede da Figura 3.1, devido a ocorrência da transição **t2**. A transição **t2** refere-se a transição **entrega**, na Figura 3.2.

Um sistema modelado usando rede de Petri não prevê nem impõe uma ordem de ocorrência dos eventos. Isso significa que se em um determinado instante, mais que uma transição se encontra habilitada, qualquer uma pode ocorrer. Isso pode ser observado através da Figura 3.2, onde as transições **produz** e **recebe** estão habilitadas, podendo qualquer uma das duas ocorrer.

Modelagem com Redes de Petri

Com base em um modelo descrito em redes de Petri, pode-se criar uma interpretação da rede. Essa interpretação ou significado faz a ligação do modelo abstrato que uma rede de Petri representa com o sistema concreto que se pretende modelar. Por exemplo, uma possível interpretação da rede da Figura 3.2 seria a modelagem de um sistema produtor-consumidor, onde os itens são produzidos aos pares e que para cada um desses pares é entregue uma unidade. Por outro lado, o consumidor (modelado pelos *lugares pode receber* e *pode consumir* e pelas *transições recebe* e *consome*) necessita receber quatro unidades e, posteriormente, de as consumir antes de poder receber mais unidades.

As redes de Petri permitem a modelagem e visualização de diversos conceitos e relações, tais como: paralelismo, concorrência, compartilhamento de recursos e sincronização. Em [Mur89] é possível encontrar as construções que permitem esses tipos de modelagens.

3.2 Extensões de Rede de Petri

Diversas variantes ao modelo de redes de Petri surgiram devido à necessidade de adaptação dessas à especificidade da aplicação conforme seu funcionamento.

A aplicação de redes de Petri para descrever sistemas reais tende a ser complexas

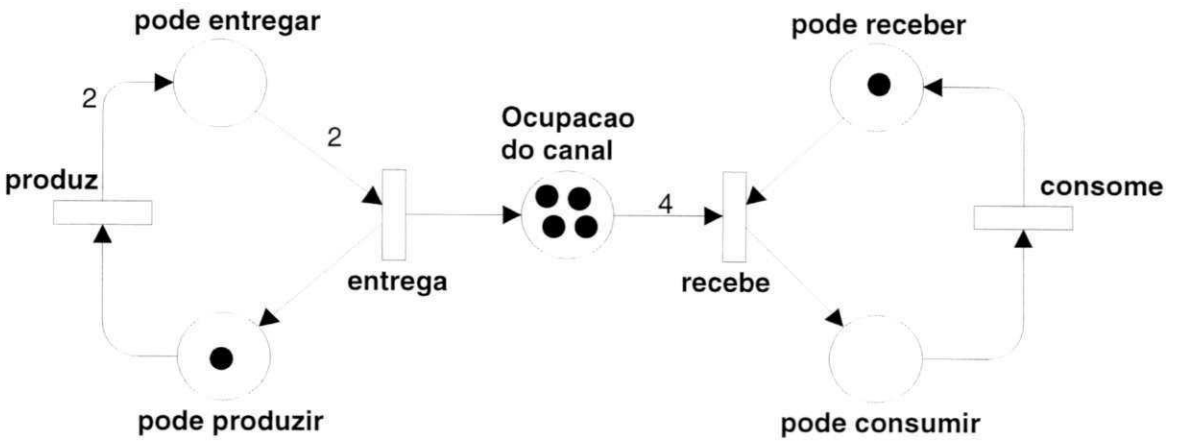

Figura 3.2: Estado da rede de Petri Lugar/Transição depois da ocorrência da transição **entrega**

e extremamente grandes. Além do mais, o modelo básico de redes de Petri não é completo o bastante para estudar o desempenho de certos sistemas, uma vez que nenhuma suposição é feita sobre a duração de suas atividades, isto é, a definição original do modelo não leva em consideração noções quantitativas dos seus aspectos temporais. Para atender essas necessidades, algumas extensões ao modelo original foram propostas. Nessa seção introduziremos informalmente duas extensões ao modelo básico de redes de Petri: as extensões relacionadas com a capacidade de modelagem funcional (redes de Petri de alto nível) e as extensões relacionadas com a caracterização de restrições temporais.

3.2.1 Redes de Petri de Alto Nível

Quando da modelagem de sistemas que se constituem de muitos componentes idênticos que interagem de alguma forma, o modelo de redes de Petri Lugar/Transição exibe grande redundância. Isso ocorre porque a única maneira de diferenciar dois componentes idênticos é especificar uma estrutura idêntica de sub-rede para cada componente. A principal causa desse problema é que todas as fichas são idênticas, não existindo nenhuma maneira de diferenciá-las entre diversas entidades. Nas extensões às redes de Petri relacionadas com a capacidade de modelagem funcional, as fichas podem ter associadas a elas diferentes tipos de informação.

Com essa extensão, pode-se modelar o componente comum apenas uma vez e asso-

ciar diferentes fichas para cada componente idêntico. As extensões relacionadas com a capacidade de modelagem funcional conduzem a uma classe de redes de Petri chamadas de *Redes de Petri de Alto Nível.* Uma abordagem de redes de Petri de alto-nível que se destaca são as redes de Petri Coloridas (CPN) [Jen98].

Redes de Petri Coloridas

As Redes de Petri Coloridas (CPN) são, provavelmente, as redes de Petri de alto nível que tem despertado o maior interesse. Nas CPN é possível representar tipos e manipulações de dados complexos. A cada ficha é associado um valor de dado denominado *cor da ficha* que pode representar tipos arbitrários de dados complexos como por exemplo; inteiros, reais, registros, etc.

Uma rede de Petri Colorida é composta por três partes distintas: *estrutura*, *declaração* e *inscrições.* A estrutura é formada por lugares, transições e arcos direcionados. As declarações definem conjunto de cores (domínios), variáveis e operações (funções) usadas nas inscrições. As inscrições, por sua vez, podem ser de quatro tipos:

1. *Cores dos Lugares*: determinam a cor associada ao lugar. Um lugar só pode comportar fichas cujos valores sejam do tipo dessa cor;
2. *Guardas*: são expressões booleanas que restringem a ocorrência das transições;
3. *Expressões dos Arcos*: servem para manipular as informações contida nas fichas;
4. *Inicializações*: associadas aos lugares, estabelecem a marcação inicial da rede.

Na Figura 3.3, ilustra-se um exemplo de uma rede de Petri Colorida. As declarações estão expressas no retângulo na parte superior da figura 3.3b. Os textos próximos aos lugares, em itálico, indicam suas cores e as expressões, do lado esquerdo dos lugares, suas inicializações. As expressões dos arcos são os textos localizados próximo aos arcos direcionados.

Uma rede Petri Lugar/Transição pode ser transformada em uma rede de Petri Colorida, isso consiste fundamentalmente na substituição de conjuntos de lugares idênticos por um só lugar, contendo o tipo da cor associado. Esses tipos são representados por

fichas que permitem a representação de cada um desses lugares através de valores distintos. Essa fusão de lugares obriga necessariamente uma fusão dos respectivos arcos. Essa é conseguida através das expressões dos arcos que permitem determinar quais fichas remover ou adicionar aos lugares. Isso pode ser visto, na Figura 3.3b, através da função **ant()**, onde essa função determina qual número inteiro deve ser removido do lugar **Garfo**. É importante salientar que a redução de complexidade resultante através da utilização da rede de Petri Colorida será tanto mais significativa, quanto maior for a quantidade de lugares idênticos.

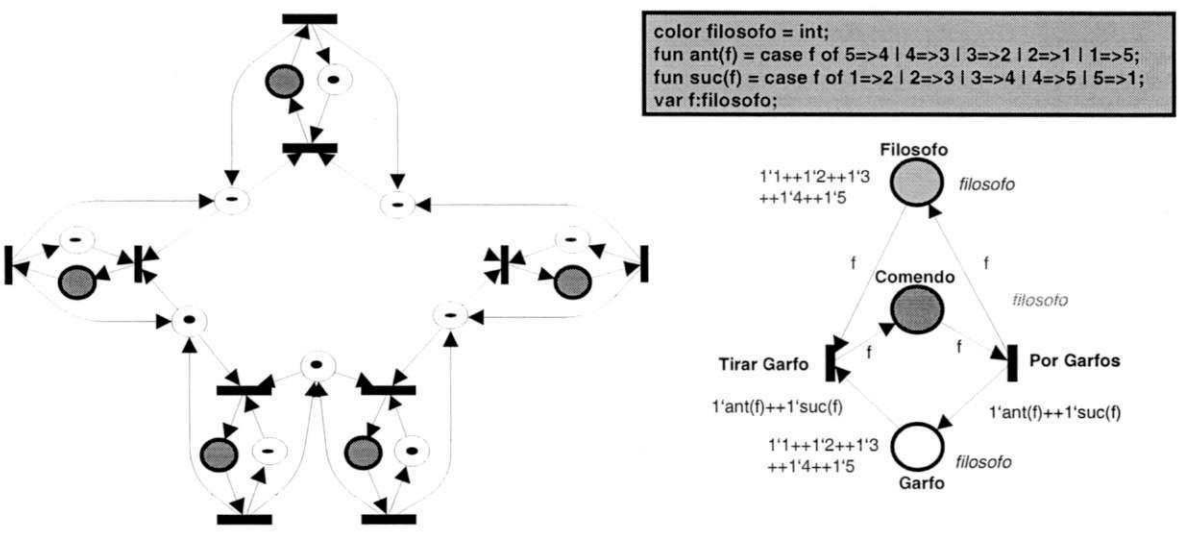

Figura 3.3: a) Rede de Petri Lugar/Transição; b) Rede de Petri Colorida correspondente.

3.2.2 Redes de Petri Hierárquicas

Nessa seção são definidas informalmente as redes de Petri Coloridas Hierárquicas (HCPN) [Jen92b]. O objetivo de definir as HCPNs é disponibilizar mecanismos para construir modelos através da combinação de um conjunto de CPNs denominadas *páginas*. Essa abordagem pode ser comparada à construção de um programa para computador a partir de um conjunto de módulos e funções.

Do ponto de vista teórico, uma HCPN possibilita descrever os mesmos tipos de sistemas que uma CPN não hierárquica e, portanto, são modelos equivalentes. É sempre possível transformar uma HCPN em uma CPN não hierárquica. Contudo, as

HCPNs disponibilizam ao projetista, mecanismos de abstração que permitem descrever de forma mais organizada modelos de sistemas complexos. Considerando que no caso desses sistemas, um dos grandes problemas enfrentados por desenvolvedores de sistemas atualmente é lidar com muitos detalhes ao mesmo tempo, as HCPNs disponibilizam mecanismos de abstração que permitem ao projetista lidar e manter o foco em uma determinada parte de um modelo de cada vez.

Modelos HCPN são implementados utilizando-se os conceitos de *lugares de fusão* e *transições de substituição*. Lugares de fusão são estruturas que permitem especificar um conjunto de lugares como funcionalmente um único lugar, isto é, se uma ficha é removida ou adicionada em um dos lugares, uma ficha idêntica é adicionada ou removida dos outros lugares pertencentes ao conjunto. Um conjunto de lugares de fusão é denominado de conjunto de fusão[2].

Uma transição de substituição pode ser vista como uma transição de mais alto nível que se relaciona a uma CPN mais complexa que descreve com mais detalhes as atividades modeladas pela transição de substituição. A página que contém a transição de substituição é denominada *superpágina* da CPN. A parte mais detalhada, correspondente, fica na página denominada *subpágina*. Cada transição de substituição é denominada de *supernó* da subpágina correspondente.

Uma transição de substituição se relaciona com sua subpágina através da utilização de um tipo de conjunto de fusão de dois membros denominados portas e *sockets*. Essas estruturas descrevem a interface entre a transição de substituição e a subpágina. *Sockets* são atribuídos aos lugares conectados à transição de substituição e portas são associadas a determinados lugares na subpágina tal que um par *socket*/porta formam um conjunto de fusão. Dessa forma, quando uma ficha é depositada num *socket*, ela aparece também na porta associada àquele *socket*, permitindo assim a conexão entre a superpágina e a subpágina. Cada *socket* pode ser associado a uma ou mais portas, e uma porta pode ser associada somente a um *socket*. Na Figura 3.4, ilustra-se uma hierarquia de duas páginas conforme descrito até o momento.

Como dito anteriormente, é sempre possível traduzir uma rede de Petri hierárquica para sua correspondente não hierárquica. Para isso, basta substituir cada transição de

[2] *Fusion set.*

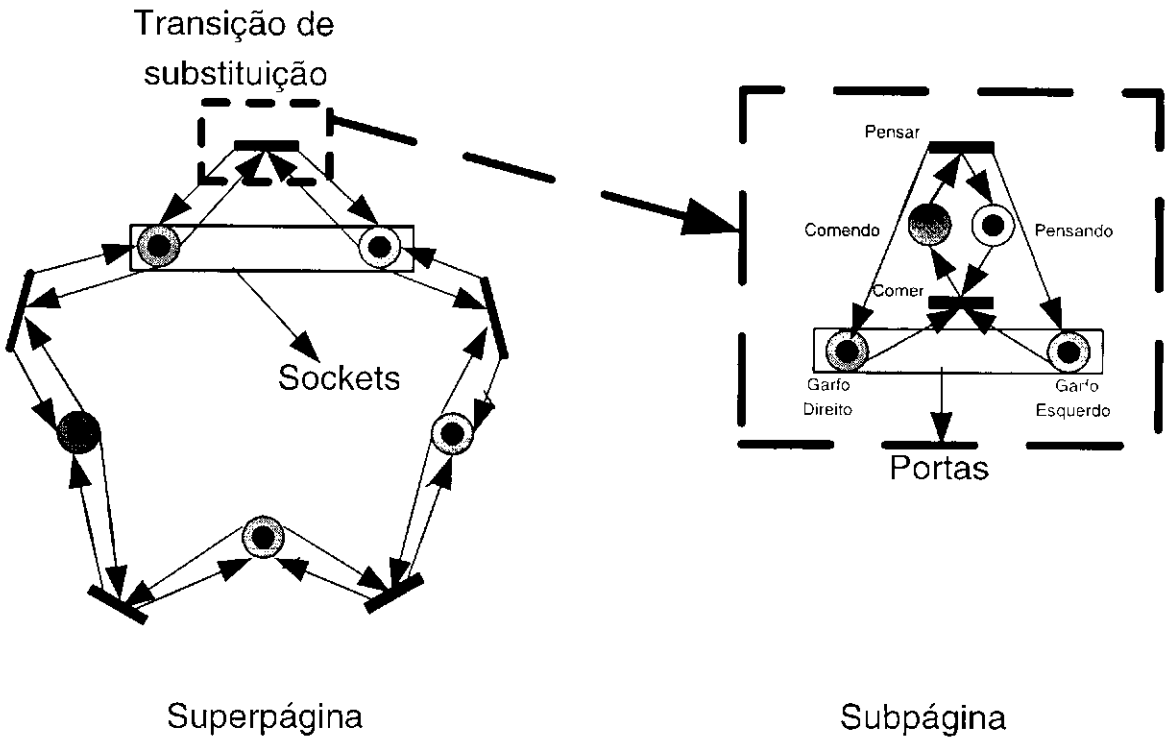

Figura 3.4: Exemplo de uma hcpn.

substituição e arcos conectados, por sua respectiva subpágina, "colando" cada *socket* à sua respectiva porta. Detalhes teóricos relacionados a HCPNs podem ser encontrados em [Jen92b].

3.2.3 Redes de Petri Temporizadas

Em muitas aplicações, as redes de Petri são utilizadas para investigar apenas aspectos relacionados às restrições lógicas. Isso, devido à caracterização das propriedades temporais de um sistema não serem possíveis de representação, isto é, podem-se representar apenas as propriedades qualitativas (não relacionadas a tempo) em um sistema usando redes de Petri. Para isso, algumas extensões de redes de Petri foram desenvolvidas possibilitando investigar aspectos relacionados a representação de propriedades quantitativas com relação ao tempo. Tais aspectos podem ser relacionados às medidas de natureza determinística, como, por exemplo, a investigação do tempo máximo utilizado para a execução de certas atividades, ou então, aspectos não determinísticos, como o tempo médio para atendimento de requisições. Essas exten-

sões utilizam basicamente dois tipos de abordagem: determinística [BR90; GMMP89; Zub91] e estocástica [HS86].

Na abordagem determinística, os requisitos de tempo são definidos como constantes e a análise, podendo ser efetuada através de um método analítico ou por simulação. Em geral, nesse tipo de abordagem, a caracterização das restrições de tempo modifica o comportamento qualitativo do modelo em relação ao modelo correspondente sem as restrições de tempo. A abordagem determinística é adequada para a modelagem de sistemas em tempo-real nos quais a caracterização de limites de tempo é essencial. Essa abordagem é limitada para a avaliação de desempenho de sistemas em que, basicamente, determina-se o tempo de ciclo mínimo, isto é, o tempo necessário para atingir um determinado estado a partir de um estado inicial. As extensões determinísticas foram usadas em diversas áreas de aplicação como por exemplo, para modelar protocolos de comunicação, e em sistemas flexíveis de manufatura, entre outras. As extensões temporais determinísticas também foram aplicadas à classe de redes de Petri de alto-nível para reduzir a complexidade na modelagem de sistemas complexos [Fig94]. As redes de alto-nível temporais, notadamente as redes de Petri coloridas temporizadas [Jen97], são adequadas para modelagem de sistemas complexos.

3.3 O Design/CPN

Para analisar um modelo CPN é fundamental dispor de um conjunto de ferramentas computacionais para edição, simulação e análise. O Design/CPN [Ja99] é um pacote de ferramentas para o desenvolvimento de modelos de redes de Petri Coloridas, e é constituído de quatro ferramentas computacionais integradas em um único pacote:

1. Editor gráfico para construir, modificar e executar análise sintática de modelos CPN;

2. Um simulador podendo operar simulação interativa ou automática, possibilitando ainda definir diferentes critérios para parada e observação da evolução da rede;

3. Uma ferramenta para gerar e analisar grafos de ocorrência de modelos CPN;

4. Uma ferramenta para análise de desempenho que possibilita a simulação e observação de dados de desempenho de modelos CPN.

O Design/CPN é um dos pacotes de ferramentas mais elaborados para construção, modificação, e análise de redes de Petri, e tem sido extensivamente utilizado com sucesso em diferentes domínios de aplicação. O leitor interessado em maiores detalhes pode consultar [CJK97].

Capítulo 4

Modelagem

Um modelo pode ser definido simplesmente como uma abstração de algum sistema, e seu propósito é permitir que se conheça propriedades desse sistema antes mesmo de construí-lo. Por exemplo, para construir sistemas complexos, o desenvolvedor deve abstrair as diversas (e diferentes) visões do sistema, construir modelos utilizando uma notação precisa, verificar se os modelos satisfazem os requisitos do sistema e acrescentar detalhes gradativamente para transformar os modelos em uma implementação.

A abstração consiste em isolar os aspectos que sejam importantes para algum propósito e suprimir os que não forem, determinando o que é ou não importante. São possíveis várias abstrações diferentes de uma mesma entidade, dependendo do propósito. Um bom modelo incorpora os aspectos fundamentais de um problema e omite os demais.

Atualmente as pesquisas na área de modelagem e análise de sistemas de informação têm despertado muito interesse por parte dos pesquisadores de banco de dados, com vários avanços na representação e manipulação dos dados, bem como pelos pesquisadores da área de engenharia de software, com metodologias de análise e projeto e métodos de especificação formal. Essas pesquisas têm se concentrado na abordagem orientada a objetos [Boo94; CY90; Dou98; RJB99]. Vários métodos orientados a objetos para modelagem de sistemas foram propostos. Esses métodos podem ser usados para a modelagem dos aspectos estruturais e comportamentais dos sistemas. No entanto, esses métodos disponibilizam mecanismos e técnicas muito limitadas para a análise formal.A análise formal é extremamente importante no projeto de sistemas grandes e comple-

xos, para reduzir o tempo de desenvolvimento e manutenção de tais sistemas [Mar95], bem como aumentar a segurança em seu funcionamento [Jal94]. As características das redes de Petri e, em particular, das CPNs introduzidas no Capítulo 3, indicam que elas são mais apropriadas para a modelagem das propriedades dinâmicas de SGBD-TR e fornecem suporte para a análise formal dos requisitos de tais sistemas.

4.1 Um Método para Modelagem de Bancos de Dados em Tempo-real

Nessa seção apresenta-se um método para modelagem de SGBD-TR [PTP99; Per00]. A aplicação do método resulta na obtenção de dois modelos: *o modelo de objetos*, usado para modelar as propriedades estruturais do sistema e, *o modelo de processos*, usado para modelar as propriedades comportamentais do sistema.

Modelo de Objetos

O método descrito em [PTP99; Per00] é baseado no modelo OMT (*Object Modelling Technique*) [BP98], e no modelo RTSORAC (*Real-Time Semantic Objects Relationships and Constraints*) [DiP95; PWPD96]. O método OMT usa três modelos complementares para modelar sistemas: *Modelo de Objetos* que caracteriza as propriedades estruturais dos objetos, tais como, atributos, operações e restrições lógicas, o *Modelo Funcional* que define as operações que os objetos executam e o *Modelo Dinâmico* que descreve as interações temporais entre os objetos e suas respostas a eventos. Ele mostra sob quais condições as operações são executadas e por quanto tempo elas devem continuar executando.

A modelagem dos objetos começa com uma análise da declaração do problema e tem os seguintes passos:

1. *Identificação dos objetos*;
2. *Identificação dos relacionamentos entre objetos*;
3. *Adição dos atributos dos objetos*;

4. *Uso de generalização para observar similaridades e diferenças*;

5. *Identificação das operações*;

6. *Identificação das operações compatíveis, ou seja, que podem ser executadas concorrentemente*;

7. *Identificação das restrições lógicas e temporais*;

8. *Refinamento do modelo*;

No escopo dessa dissertação, nos interessam os passos 6 e 7 introduzidos para definir as operações compatíveis, assim como as restrições lógicas e temporais.

Modelo de Processos (Funcional e Dinâmico)

O modelo de processos, captura as propriedades funcionais e dinâmicas dos objetos, isto é, ele pode ser visto como a combinação do modelo funcional e do modelo dinâmico em um único modelo. O modelo de processos pode ser usado nas fases de análise, projeto e implementação do sistema. Essas fases podem ser tratadas simultaneamente, usando redes de Petri Coloridas (CPN). Então, o projetista de uma aplicação pode usar as CPNs para analisar o comportamento do sistema.

No modelo de processos os objetos são descritos através de módulos em CPN. Ele é definido a partir do modelo de objetos. Então, para cada objeto (contendo operações), identificado no modelo de objetos, é criado um módulo CPN, no qual serão modeladas as operações dos objetos correspondentes.

4.2 Transações de uma Aplicação de Tempo-Real

A abordagem que introduzimos nessa seção baseia-se em [PTP99; Per00]. É importante observar que essa abordagem utiliza a extensão de redes de Petri denominada G-CPN, para o modelo de processos. As G-CPNs foram introduzidas em [Gue97; GPdF97; GdFP97]. No entanto, para o nosso trabalho serão considerados diretamente os modelos equivalentes às G-CPNs em CPN.

Como mencionado na introdução e definido no Capítulo 2, nessa dissertação transações periódicas e esporádicas com prazos estritos ou suaves serão consideradas na análise do escalonamento [PMRN01; NPLT01]. Dessa forma, considere um conjunto finito de transações em tempo-real modeladas em CPN, onde para cada transação τ_i define-se os parâmetros temporais pela quádrupla (tl_i, tc_i, pr_i, pe_i), onde:

1. tl_i: é o tempo de liberação da transação, isto é, o momento no qual todos os recursos necessários à execução da transação τ_i estão disponíveis. A partir desse momento a transação estará pronta para ser executada.

2. tc_i: representa o tempo computacional da transação, isto é, o tempo de processamento necessário para a executá-la.

3. pr_i: especifica o prazo máximo para a execução da transação τ_i.

4. pe_i: indica a periodicidade da transação.

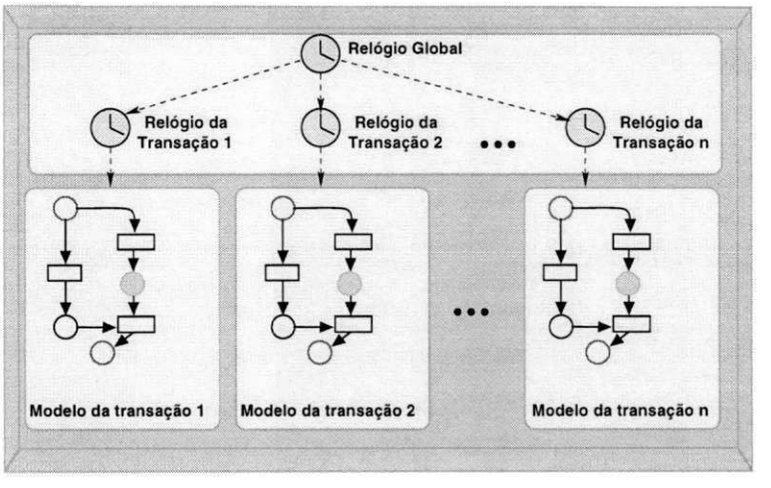

Figura 4.1: Módulo CPN para os Relógios

Na Figura 4.1 apresentamos um esquema para modelar um conjunto de transações, onde identificam-se duas camadas funcionais: relógios e transações. Observa-se que cada transação é modelada por uma página CPN, facilitando a inserção de novas transações no esquema, quando necessário. Dessa forma, o sistema é composto por um conjunto de n transações modeladas por n páginas CPN, onde para cada nova transação existe um relógio local sincronizado pelo relógio global.

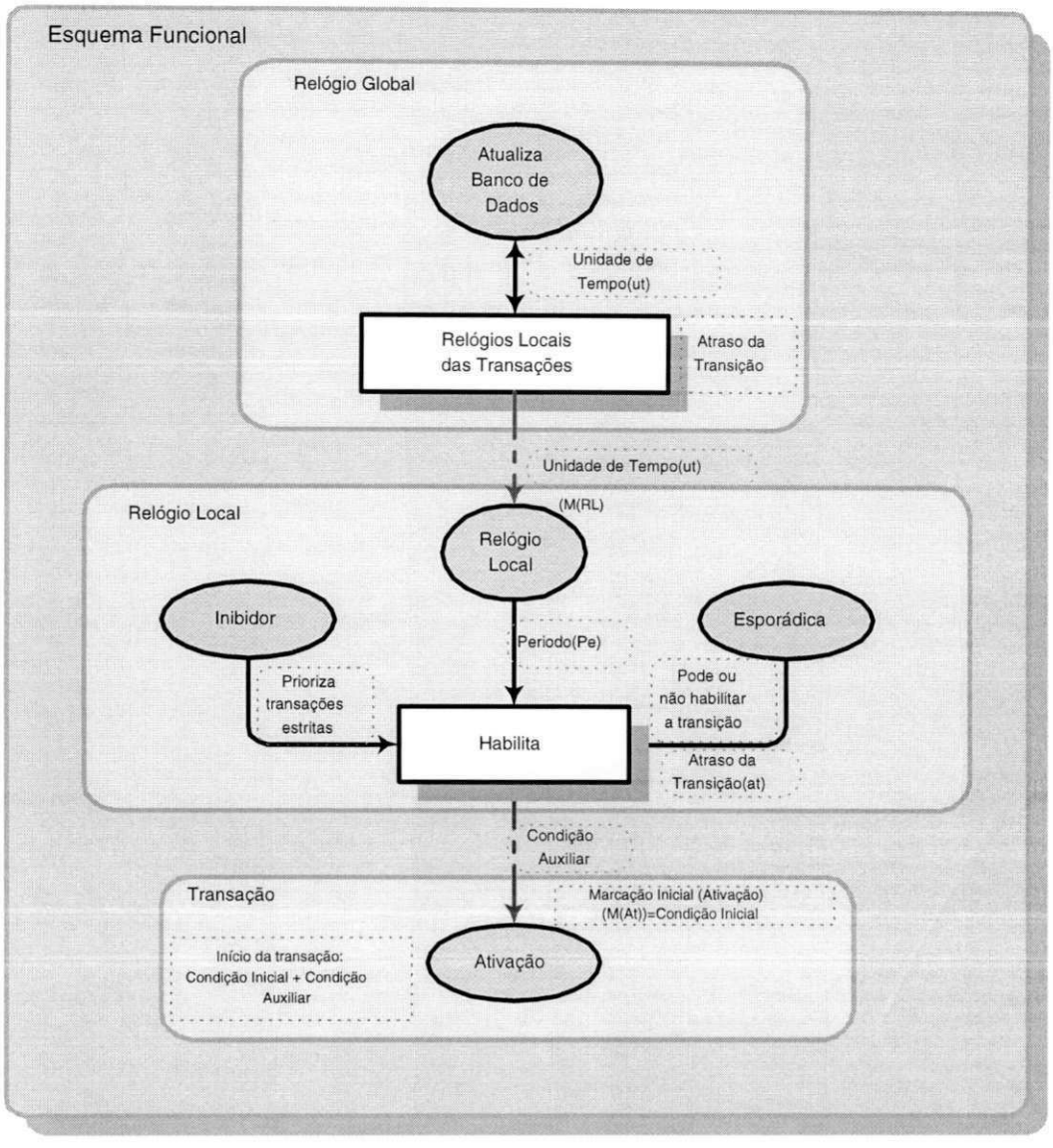

Figura 4.2: Esquema Funcional

As características temporais para o sistema, são modeladas da seguinte forma:

- Os relógios são representados por lugares e transições como ilustrado na Figura 4.2, então:
  - o par (**Relogio Local**, **Habilita**), para o relógio local da transação.
  - o par (**Atualiza Banco de Dados**, **Relógios Locais das Transações**), para o relógio global.
- Os tempos computacionais de cada transação são definidos pelos tempos de exe-

cução dos objetos invocados.

- Os prazos de execução são identificados através de marcações terminais, indicando se uma tarefa cumpriu ou não seu prazo para execução.

- O tempo de liberação é representado pela marcação inicial do relógio local da transação, e é definido da seguinte maneira:

$$M_0(\text{Relogio Local}) = \begin{cases} 1'e & \text{se } tl = 0 \\ (pe - tl + 1)'e & \text{se } tl > 0 \end{cases}$$

$$M_0(\text{Ativacao}) = \begin{cases} Condicao\_Inicial + Condicao\_Auxiliar & \text{se } tl = 0 \\ Condicao\_Inicial & \text{se } tl > 0 \end{cases}$$

Na Figura 4.2, temos que: o tempo de liberação é obtido pela expressão **Pe-M(RL)+at**, onde **Pe** é o período, **M(RL)** é a marcação inicial para o **Relogio Local** e **at** é o atraso na transição **Habilita**.

- O período é representado pela quantidade de fichas removidas dos lugares modelando os relógios locais. Na Figura 4.2, o período é definido pela inscrição do arco entre o lugar **Relogio Local** e a transição **Habilita**.

Os lugares **Inibidor** e **Esporádica**, no esquema da Figura 4.2, determinam as seguintes situações: O lugar **Inibidor** representa a situação em que a execução das transações com prazos estritos tem prioridade, evitando o problema de *inversão de prioridade*, isto é, que transações com prazos estritos (maior prioridade) esperem por transações com prazos suaves (menor prioridade) para executar. O lugar **Esporádica** representa a situação na qual mesmo a transação estando pronta para executar (instante de tempo atual igual ou maior que o tempo de liberação para essa transação), ela pode ou não ocorrer, caracterizando as transações esporádicas.

Uma transação pode ser iniciada quando no lugar **Ativação** tiver associado a **Condição Inicial** acrescida da **Condição Auxiliar**. A **Condição Auxiliar** é adicionada ao lugar **Ativação** quando do tempo de liberação da respectiva transação. O fim da execução da transação e representada quando a **Condição Inicial** for novamente adicionada a esse lugar. Contudo, se a **Condição Auxiliar** for adicionada ao lugar **Ativação** antes da **Condição Inicial** representa que transação perdeu o prazo de execução. Observe que

o estado **Condição Inicial + Condição Auxiliar**, precedido pelo estado **Condiçaõ Inicial**, indica uma marcação terminal.

Esse esquema de modelagem para transações foi adotado devido a facilidade de análisar o escalonamento das transações, visto que para saber se uma transação cumpriu o prazo ou não basta apenas, através da geração do espaço de estados, verificar as marcações do lugar **Ativação**.

4.3 Estudo de Caso

Nessa seção detalha-se como obter um modelo conceitual de um sistema de gerenciamento de banco de dados em tempo-real através de um método para modelagem de bancos de dados em tempo-ral. O SGBD-TR é composto por um objeto de banco de dados. Foram modeladas três funções de compatibilidade para esse objeto, essas funções representam as situações descritas no Capítulo 2. Quatro diferentes tipos de transações invocam os métodos do objeto de banco de dados:

1. A transação 1 é uma transação de atualização periódica com prazo estrito;
2. A transação 2 é também uma transação de atualização porém é esporádica com prazo estrito;
3. A transação 3 é uma transação de leitura periódica com prazo estrito e
4. A transação 4 é uma transação de leitura esporádica com prazo suave.

Cada uma dessas transações possue um relógio local sincronizado por um relógio global.

4.3.1 Modelando a Aplicação

Serão descritas nessa seção as páginas que compõem o modelo. Primeiro apresentaremos uma visão hierárquica do modelo e em seguida o nó de declarações. Depois detalharemos o objeto de banco de dados, bem como as transações que compõem o modelo.

Visão Hierárquica do Modelo

Na Figura 4.3 apresenta-se a visão hierárquica para a aplicação, através de uma página CPN. Podemos identificar as páginas para as transações que estão sincronizadas pela página **Relogio**. As páginas **Trans1**, **Trans2**, **Trans3** e **Trans4** modelam as transações. O objeto de banco de dados é modelado pela página **ObjetoBD**. As páginas para o **Sincronizador** e para as funções de compatibilidade definidas para o objeto **ObjetoBD** completam o modelo. O **Sincronizador** tem a função de implementar a semântica associada ao sistema. Em se tratando da modelagem de sistemas distribuídos, o sincronizador pode ser visto como a abstração da rede de comunicação, tendo a função de implementar a semântica de transferência de mensagens entre os módulos. Para a nossa aplicação as páginas CPN representam os módulos. Quando um módulo requisita um serviço de outro módulo, o sincronizador implementa o transporte das mensagens de requisição e de resposta. Dessa forma, do ponto de vista do modelo, a transferência pode ser abstraída, bastando indicar qual o serviço desejado (método) e o identificador necessário. Além dessas, há mais as páginas: **Declaracoes**, **Ativar**, e **Desempenho**. A página **Declaracoes** contém as declarações de cores (tipos de dados), variáveis, e funções utilizadas no modelo. Na página **Ativar** há funções utilizadas para simulação e definição de critérios de parada para a construção do grafo de ocorrência, e a página **Desempenho** contém as definições e funções utilizadas pela ferramenta para análise de desempenho do modelo em redes de Petri Coloridas.

Nó de Declarações

Para simplificar a apresentação do modelo, descreveremos a seguir as cores, variáveis e funções utilizadas no modelo.

```
(*---------------------- Conjunto de Cores ------------------------*)
color INT = int timed;
color MIL = INT;
color IMP = INT;

(* Cor para o(s) tipo(s) de item(ns) de dado(s)*)
color PRODS =  with t1 timed;

(*Cores para os objetos do banco de dados*)
```

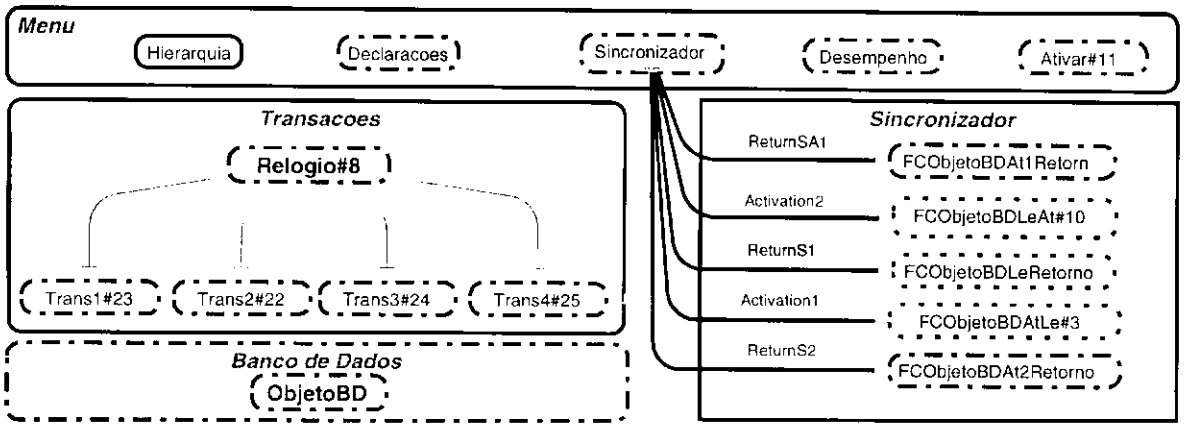

Figura 4.3: Hierarquia para o modelo CPN

```
color DBOCELL = record nr:PRODS*vrr:INT*avir:INT*tsr:INT*impr:INT*milr:MIL;
color PRODSIMPMIL = record nrl:PRODS*impl:IMP*mill:MIL;
color PRODSSOFT = record nrs:PRODS*impl:IMP;

(*Cores para os estados dos objetos*)
color ESTADO = with executando | livre;
color ESTADOPRODS = product ESTADO*PRODS;
color EVALFC = with fcfalso | fcverd;
color PRODSIMPMILXEVALFC = product PRODSIMPMIL*EVALFC;
color DBOCELLXEVALFC = product DBOCELL*EVALFC;

(* Cores para os  relógio*)
color HABILITA = with pronta | prontaS | terminada timed;
color RELOGIO = with e timed;
color DISPARO = record be:INT*cc:INT;

(* Cores definidas com o Gerador de Contexto *)
color OCC = int;
color INV = with bdcelle1 | bdcelat1| bdcelat2 | bdcelsoft1;
color INVOCC2 = product INV*OCC*OCC;
color OBJ = with bdcelle | bdcelat | bdcelsoft;
color RETURN = union dboP:DBOCELL + dboR:DBOCELL + dboQ:DBOCELL;
color RETURNS_ = product RETURN*OCC;
color ACTIVATE = union dboS:DBOCELL + psimpmilA:PRODSIMPMIL + pssoftA:PRODSSOFT;
color ACTIVATIONS_ = product OBJ*ACTIVATE*OCC;
color DBOCELL_ = product DBOCELL*OCC;
color DBOCELLX_ = product INV*DBOCELL*OCC;
color PRODSIMPMIL_ = product PRODSIMPMIL*OCC;
color PRODSIMPMIL_XEVALFC = product PRODSIMPMILXEVALFC*OCC;
```

```
color DBOCELL_XEVALFC = product DBOCELLXEVALFC*OCC;
color PRODSSOFT_=product PRODSSOFT*OCC;

(*Cores para as transações esporádicas*)
color Int_0_10 = int with 0..10;
color Int_1_10 = int with 1..10;

(*---------------------- Variáveis-----------------------*)
var s: Int_0_10;
var r: Int_1_10;
var ps,ps1: PRODS;
var oi,or,oc,ox,pq : OCC;
var inv : INV;
var m,n,on,os,i : INT;
var nts,ts : INT;
var obj: OBJ;
var dbnew,dbo,dboup,dbocell : DBOCELL;
var psimpmil : PRODSIMPMIL;
var pssoft : PRODSSOFT;
var mil,mille : MIL; var estado : ESTADO;
var avalfc : EVALFC; var estadoprods : ESTADOPRODS;
var tp1,tp12,tp2,tp3: DISPARO;

(*------------------------ Funções ----------------------*)
fun Ok(s:Int_0_10, r:Int_1_10) = (r<=s); (* Habilita a transação esporádica ou nao*)

fun atribui(tp1:DISPARO):DISPARO=
        {be = agora(),        (*Atribui tempo atual ao atributo*)
         cc = #cc(tp1) + 1    (*Incrementa sempre que a transação  ocorrer*)
        };

fun tpDisparo(tp2:DISPARO):INT = #be(tp2); (*Retorna o valor do atributo*)

fun atudbo(dbo:DBOCELL,dboup:DBOCELL):DBOCELL=
        {nr  = #nr(dboup),
         vrr = #vrr(dboup) + #vrr(dbo),(*Adiciona quantidade (x) ao registro*)
         avir= #avir(dboup),
         tsr = agora(),               (*Atualiza o rótulo de tempo*)
         impr= #impr(dboup),
         milr= #milr(dboup)
        };

fun restart(dbocell:DBOCELL,dboup:DBOCELL):DBOCELL=
        {nr  = #nr(dboup),
         vrr = #vrr(dbocell)-#vrr(dboup), (*Subtrai quantidade (x) do registro*)
         avir= #avir(dboup),
         tsr = agora(),                   (*Atualiza o rótulo de tempo*)
         impr= #impr(dboup),
         milr= #milr(dboup)
```

```
        };

fun restartle(dbo:DBOCELL,psimpmil:PRODSIMPMIL):DBOCELL =
        {nr  = #nr(dbo),
         vrr = 0,                          (*Zera o valor do registro*)
         avir= #avir(dbo),
         tsr = #tsr(dbo),
         impr= #impr(dbo),
         milr= #milr(dbo)
        };

fun ajustaimp(dbocell:DBOCELL,psimpmil:PRODSIMPMIL):DBOCELL=
        {nr  = #nr(dbocell),
         vrr = #vrr(dbocell),
         avir= #avir(dbocell),
         tsr = #tsr(dbocell),
         impr= #impr(dbocell)+#impl(psimpmil), (* Ajusta a imprecisão do registro*)
         milr= #milr(dbocell)
        };

fun ajustaimp1(dbocell:DBOCELL,dboup:DBOCELL):DBOCELL=
        {nr  = #nr(dbocell),
         vrr = #vrr(dbocell),
         avir= #avir(dbocell),
         tsr = #tsr(dbocell),
         impr= #impr(dbocell)+#impr(dboup), (* Ajusta a imprecisão do registro*)
         milr= #milr(dbocell)
        };

(*Atribui valores ao registro -  Valores dos atributos iguais ao inicial *)
fun recria(dbocell:DBOCELL):DBOCELL=
        {nr  = #nr(dbocell),
         vrr = 1,
         avir= 10,
         tsr = 0,
         impr= 1,
         milr= 8
        };

(*********************************************************
        Estado inicial das variáveis usadas no modelo
*********************************************************)
val milt1L        = 5 ;   (*Limite de imprecisao para ler tipo t1*)
val objetobdvrr1  = 2;    (*Tempo para atualização 1 no ObjetoDB*)
val objetobdvrr2  = 4;    (*Tempo para atualização 2 no ObjetoDB*)


(*Tempo de execução  diferente para as atualizações*)
fun tempoexec(dboup:DBOCELL):INT  =
```

```
if (#vrr(dboup)=1)
   then objetobdvrr1
   else objetobdvrr2;
```

Comportamento do Objeto de Banco de Dados

Nessa seção detalhamos o comportamento do objeto de banco de dados. Para tal, definimos os métodos do objeto, bem como sua funcionalidade.

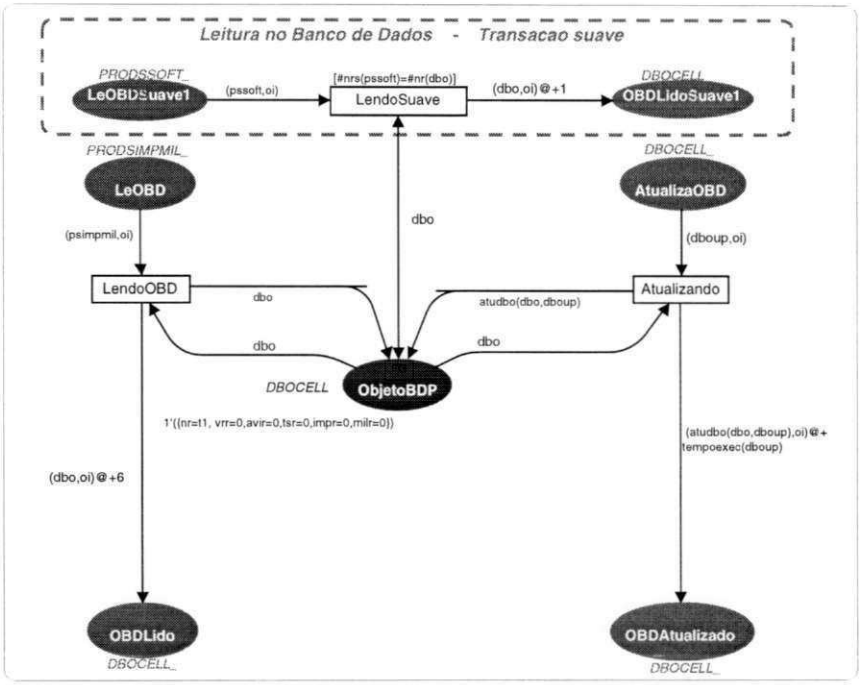

Figura 4.4: CPN para o objeto de banco de dados

Para o objeto banco de dados **ObjetoBD**, mostrado na Figura 4.4, são definidos os métodos **Atualiza**, **LeOBD**, **LeOBDSuave** e o atributo **ObjetoBDP** que corresponde ao banco de dados onde são armazenados os dados. Os métodos serão definidos a seguir:

1. Método **LeOBD**: esse método implementa a leitura com prazo estrito no banco de dados. Ele recebe como argumento de entrada uma ficha da cor **PRODSIMPMIL_**, que indica o tipo do dado que deve ser lido, a imprecisão acumulada, e o limite máximo de imprecisão aceita. Para esse método o lugar inicial é o lugar **LeOBD**.

2. Método **LeOBDSuave**: esse método implementa a leitura com prazo suave no banco de dados. Ele recebe como argumento de entrada uma ficha da cor **PRODSSOFT_**, que indica o tipo do dado que deve ser lido e a imprecisão acumulada. Para esse método o lugar inicial é o lugar **LeOBDSuave1**.

3. Método **Atualiza**: esse método implementa a atualização no banco de dados. Ele recebe como argumento de entrada uma ficha da cor **DBOCELL_** que define um registro indicando o tipo e quantidade do item de dado que deve ser atualizado. Para esse método o lugar inicial é o lugar **AtualizaOBD**.

Quando o método **Atualiza** é invocado, uma ficha definindo um registro para o tipo do item de dado é depositada no lugar **AtualizaOBD**. Em seguida, a transição **Atualizando** ocorre, o banco de dados **ObjetoBDP** é atualizado e o método termina sua invocação depositando uma ficha no lugar **OBDAtualizado**. A função **atudbo()** é definida no nó de declarações. Para a atualização do banco de dados definimos um atraso associado à inscrição do arco entre a transição **Atualizando** e o lugar de terminação **OBDAtualizado**. Esse atraso é determinado pela função **tempoexec(dboup)**. A necessidade de utilizar uma função foi devido o fato de especificarmos atrasos para as atualizações com valores diferentes.

Quando o método **LeOBD** é invocado, uma ficha contendo o tipo do item de dado a ser lido é colocada no lugar **LeOBD**, a transição **LendoOBD** ocorre, a leitura no banco de dados é efetivada. Em seguida, o método termina sua invocação depositando uma ficha no lugar **OBDLido**. Para a leitura do banco de dados definimos um atraso associado à inscrição do arco entre a transição **LendoOBD** e o lugar de terminação **OBDLido**, esse atraso é constante igual a 6 unidades de tempo.

Quando o método **LeOBDSuave** é invocado, uma ficha contendo o tipo do item de dado a ser lido é colocada no lugar **LeOBDSuave1**, a transição **LendoSuave** ocorre, a leitura no banco de dados é efetivada. Em seguida, o método termina sua invocação depositando uma ficha no lugar **OBDLidoSuave1**. Para a leitura, com prazo suave, do banco de dados definimos um atraso associado à inscrição do arco entre a transição **LendoSuave** e o lugar de terminação **OBDLidoSuave1**, esse atraso é constante igual a 1 unidade de tempo.

Para o objeto de banco de dados define-se também as funções de compatibilidade, que serão detalhadas na Seção 4.3.2.

Inicialmente o atributo **ObjetoBDP** possui uma marcação inicial definindo uma ficha com o tipo do item de dado, que no caso é **t1**, como mostrado a seguir:

1'{nr=t1,vrr=0,avir=0,tsr=0,impr=0,milr=0}

Para essa marcação **nr** identifica o tipo do dado, **vrr** é o valor gravado no banco de dados, **avir** é o tempo de validade absoluta, **tsr** é o rótulo de tempo, **impr** é a imprecisão no atributo e **milr** é a imprecisão total acumulada. Observe que **avir** identifica por quanto tempo além do valor definido pelo rótulo de tempo **tsr** o dado é válido.

Características das Transações do Banco de Dados

Nessa seção apresentamos os modelos CPN para os relógios e os modelos para cada transação.

Modelo dos Relógios para as Transações

Como apresentado no modelo da Figura 4.5, quando invocado, o relógio global para as transações é iniciado e uma ficha **1'e** é depositada no lugar **Atualiza Banco de Dados** e nos lugares dos relógios locais. Para essa ação é definido um atraso de 1 unidade de tempo, ilustrado na transição **Relogios Locais das Transacoes**. O lugar **Criterio de Parada** especifica o tempo máximo para o relógio global. Pode-se observar que para cada transação tem-se um relógio local.

Em seguida os relógios locais e seus parâmetros são definidos, sendo referidos pelas respectivas transações:

1. *Atualização 1*: para essa transação temos: tempo de liberação $tl = 3$ unidades de tempo e periodicidade $pe = 12$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar **Atualiza1** temos 10 fichas (**10'e**), a marcação inicial para o *lugar de Ativação* (**Inicia1**) é **1'terminada**, e o peso do arco de entrada da transição de ativação (**Acessa1**) é **12'e**.

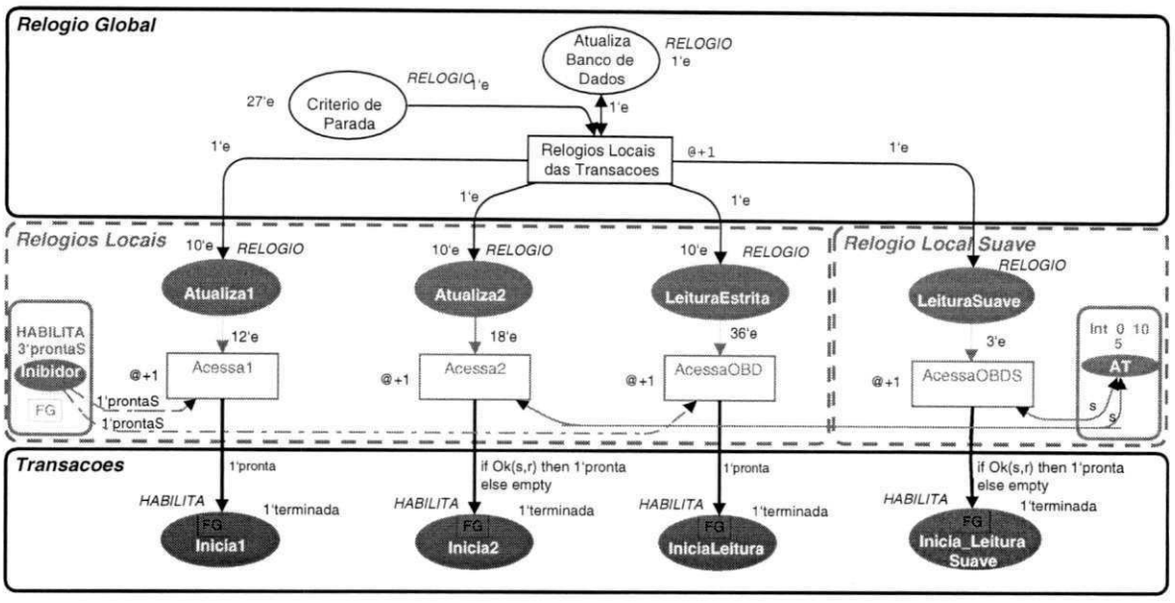

Figura 4.5: Modelo CPN para os Relógios

2. *Atualização 2*: para essa transação temos: tempo de liberação $tl = 9$ unidades de tempo e periodicidade $pe = 18$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar **Atualiza2** temos 10 fichas (**10'e**), a marcação inicial para o *lugar de Ativação* (**Inicia2**) é **1'terminada**, e o peso do arco de entrada da transição de ativação (**Acessa2**) é **18'e**.

3. *Leitura com prazo estrito*: para essa transação temos: tempo de liberação $tl = 27$ unidades de tempo, e; periodicidade $pe = 36$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar **LeituraEstrita** temos 1 ficha (**1'e**), a marcação inicial para o *lugar de Ativação* (**IniciaLeitura**) é **1'terminada**, e o peso do arco de entrada da transição de ativação (**AcessaOBD**) é **36'e**.

4. *Leitura com prazo suave*: para essa transação temos: tempo de liberação $tl = 0$, e; periodicidade $pe = 3$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar **LeituraSuave** não possui ficha, a marcação inicial para o *lugar de Ativação* (**Inicia_Leitura Suave**) é **1'terminada**, e o peso do arco de entrada da transição de ativação (**AcessaOBDS**) é **3'e**.

No lugar **AT**, referente ao lugar **Esporadica** do esquema da Figura 4.2, a marcação inicial é **1'5**. Uma vez a transição **Acessa2**, por exemplo, tenha ocorrido, a função **Ok**,

verifica se o valor de s é maior ou igual ao valor de r, se for a transição adiciona a ficha 1'pronta ao lugar de ativação, caso contrário não será adicionada nenhuma ficha a esse lugar. O valor de s é a marcação inicial do lugar AT e o valor de r é obtido aleatoriamente entre 1 e 10, como definido no nó de declarações. Através dessa função, podemos aumentar ou diminuir a possibilidade das transações esporádicas executarem, visto que é só alterar a marcação inicial do lugar AT.

Para o lugar Inibidor a marcação inicial é 3'prontaS. Essa marcação inicial representa o número de transações com prazos estritos. Quando uma transação com prazo estrito está pronta para executar, uma ficha 1'prontaS é removida e só será adicionada quando a transação terminar. Não há arcos direcionados para as transições Acessa2 e AcessaOBDS por serem transações esporádicas. Isso devido a possibilidade de que quando elas estiverem prontas para executar se a função Ok for avaliada em falso, então uma ficha será retirada do lugar Inibidor e não será mais colocada. Para evitar isso, foram modelados lugares de fusão, para esse lugar em cada página das transações. Para as transações com prazos suaves, veremos que para elas executarem uma quantidade de fichas idêntica a quantidade de fichas da marcação inicial do lugar Inibidor é necessária. Dessa forma, enquanto uma transação com prazo estrito estiver executando, as transações com prazo suave não podem executar, com isso modela-se a situação na qual prioriza-se a execução das transações com prazos estritos, evitando o problema de *inversão de prioridade.*

Descrição dos Modelos CPN das Transações

Como dito anteriormente, as transações possuem lugares e transições adicionais que representam os relógios locais que são sincronizados por um relógio global e para uma transação ser iniciada no lugar de ativação deve ter duas fichas, ou seja: 1'pronta++1'terminada. A execução da transação é iniciada pela ocorrência da transição de iniciação associada a esse lugar, que remove as duas fichas do lugar de ativação. Ao fim da execução da transação, uma ficha 1'terminada é depositada no lugar de ativação. Como já mencionado a transação 1 é uma transação de atualização periódica com prazo estrito, a transação 2 é também uma transação de atualização porém é esporádica com prazo estrito, a transação 3 é uma transação de leitura periódica com prazo estrito e a

transação 4 é uma transação de leitura esporádica com prazo suave.

Transação 1

A transação (**Atualiza1**, **Acessa1**) é uma transação de atualização periódica com prazo estrito. Essa é composta por duas subtransações mostradas no modelo da Figura 4.6, onde cada uma possui seu prazo. As subtransações foram modeladas para executar seqüencialmente. Assim, uma subtransação só pode iniciar quando a anterior terminar. De acordo com a Figura 4.6 a *subtransação 1* possui um período de 6 unidades de tempo ($pe = 6$). O tempo de liberação é de 3 unidades de tempo ($tl = 3$). Pode-se observar que o tempo de liberação da primeira subtransação na seqüência é sempre igual ao tempo de liberação da transação como um todo. A marcação inicial para o *Relógio_Local*, no lugar **Atualiza1** tem 10 fichas (**10'e**) e a marcação inicial para o *lugar de Ativação* (**IniciaAS**) é **1'terminada**. Já a *subtransação 2* possui um período de 5 unidades de tempo ($pe = 5$). Para essa, o tempo de liberação é igual ao tempo de liberação mais o tempo de execução da *subtransação 1*. Se a *subtransação 1* perder seu prazo, esse pode ser compensado na execução da *subtransação 2*. As subtransações em tempo-real, executando seqüencialmente, só serão abortadas se a soma dos tempos de execução de todas as subtransações for maior que o prazo da transação. Na inscrição do arco de saída da transição **OBDAtualizaS1** temos a ficha **1'{nr=t1,vrr=1,avir=10,tsr=0,impr=1,milr=8}**.

Ao lugar **inibP23** é adicionada uma ficha **1'prontaS** quando a transação for terminada. Esse é um lugar de fusão do lugar **Inibidor** da página CPN para os relógios. Inicialmente o lugar **Td1** possui uma marcação inicial igual a **1'{be=0,cc=0}**. Para essa marcação **be** identifica o tempo que a transação iniciou sua execução e **cc** um contador identificando quantas vezes a transação executou em um determinado intervalo de tempo.

Como essa transação possui prazo estrito, se o prazo for perdido as operações da transação devem ser desfeitas (abortadas). Isso só é interessante se a transação ainda for válida temporalmente. Na Figura 4.7, entre a transição **TFCBDOleatRP** e o lugar **ObjetoBDatLI1SP1** temos a expressão condicional, mostrada a seguir que garante essa propriedade.

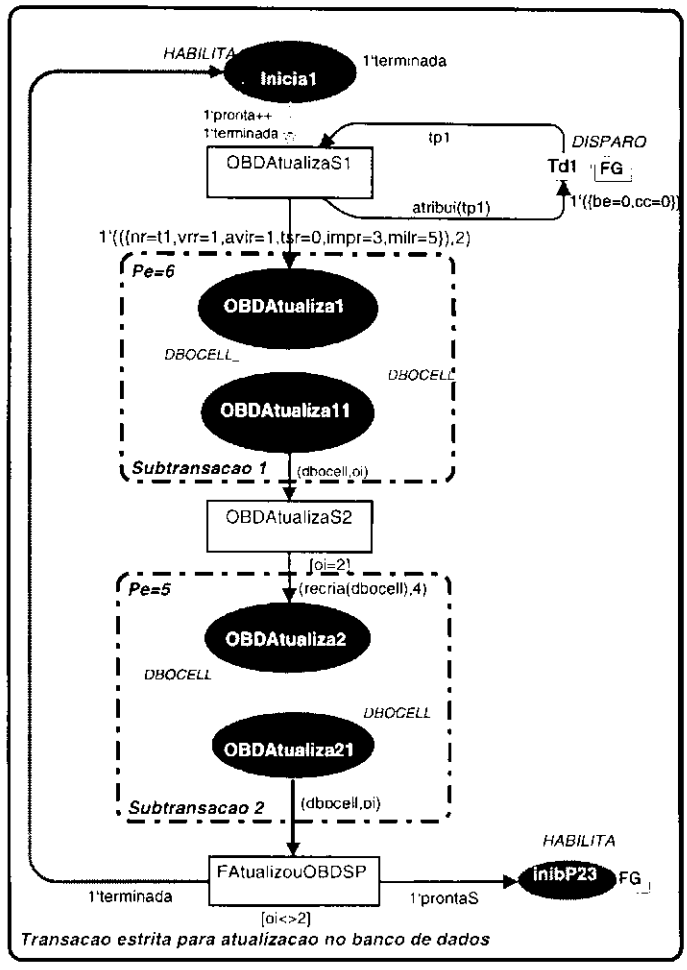

Figura 4.6: CPN para a **transação 1**

```
if (agora - #tsr(dbocell) <=  #avir(dbocell))
   then if (agora() > (tpDisparo(tp1)+12))
           then (restart(dbocell,dboup),oi)
           else if avalfc=fcfalso then (dbocell,oi)
                   else (ajustaimp1(dbocell,dboup),oi)
   else (dbocell,oi)
```

Se o instante atual for maior que o tempo de liberação mais o período, então a função *restart* é executada. Essa função, desfaz a atualização no banco de dados de acordo com seus parâmetros que são: o objeto **dbocell** que a transação está manipulando e o objeto do banco de dados **dbo**. Se a função **ajustaimp1** for executada o atributo para a imprecisão acumulada será atualizado. Essas funções estão definidas no nó de declarações.

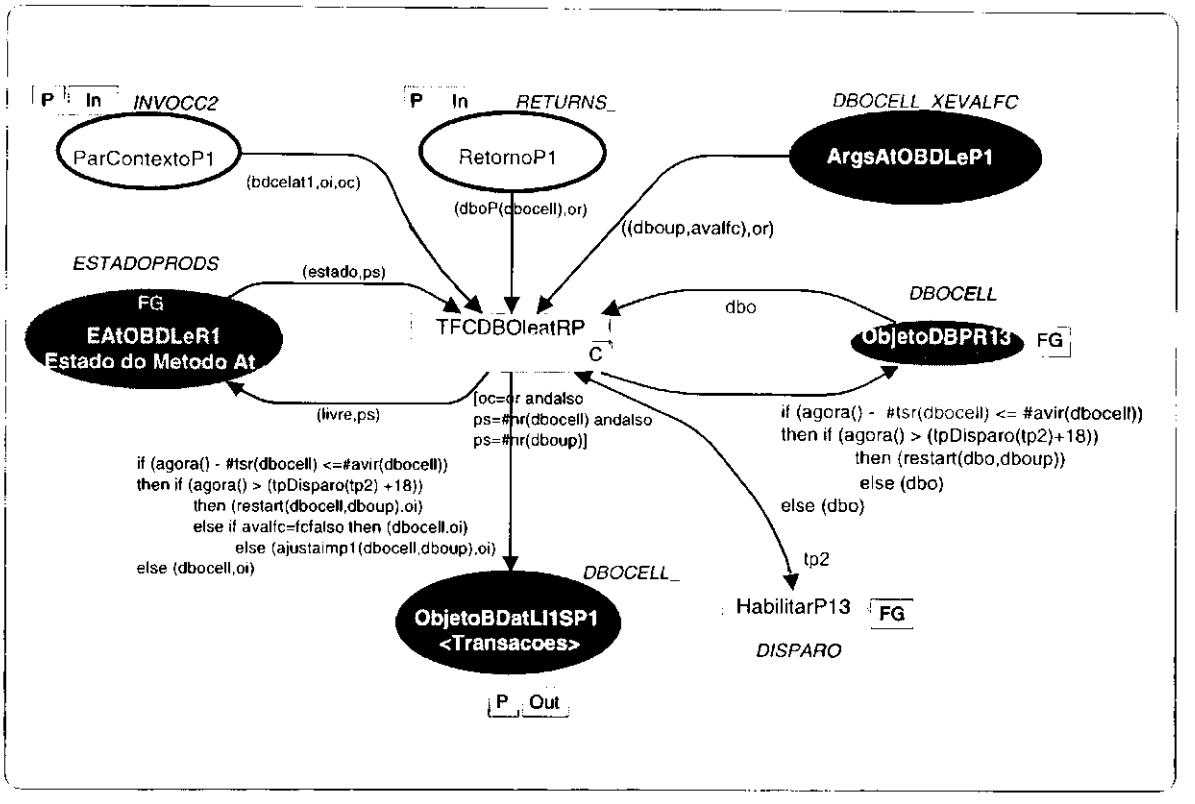

Figura 4.7: CPN para o retorno das **transações**

Transação 2

O modelo para a transação (**Atualiza2**. **Acessa2**) é mostrado na Figura 4.8. Como dito essa é uma transação de atualização esporádica com prazo estrito. As transações esporádicas, apesar de possuírem um período elas podem ser executadas ou não, uma vez executadas devem atender as restrições de tempo já que possuem prazos estritos. Para essa página, o lugar **Td2** é idêntico ao lugar **Td1** e o lugar **inibP22** é idêntico ao lugar **inibP23**, descrito para o caso da*transação 1*. Pode-se observar que para o lugar **inibP22** existe um arco orientado para a transição **OBDAtualizaT2**, isso ocorre porque trata-se de uma transação esporádica. Na inscrição do arco de saída da transição **OBDAtualizaT2** temos a ficha 1'{**nr=t1,vrr=2,avir=10,tsr=0,impr=1,milr=8**}, que é o objeto que será manipulado. As restrições de tempo para essa transação são tratadas da mesma maneira que as restrições da *transação 1*.

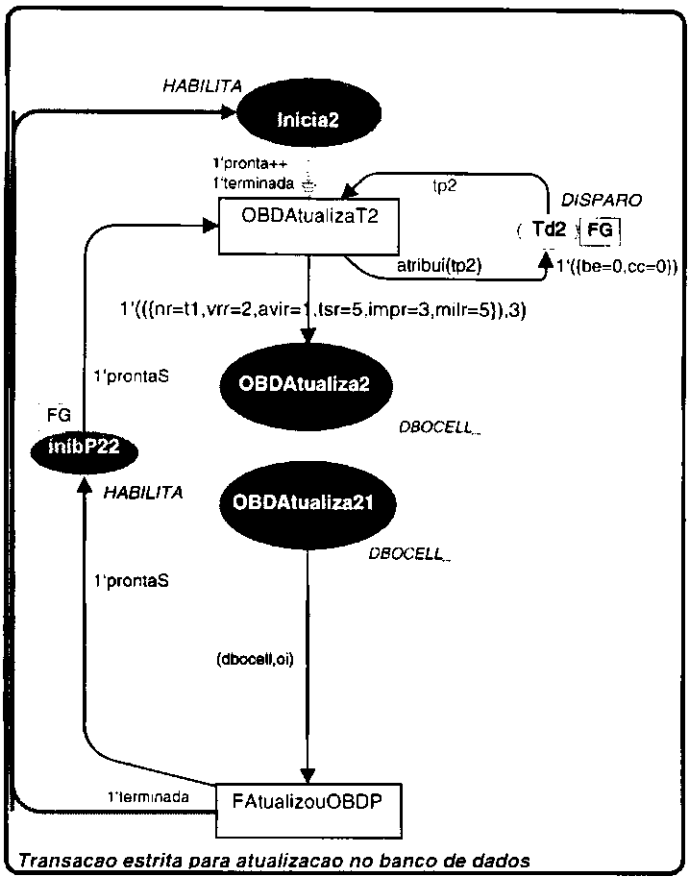

Figura 4.8: CPN para a **transação 2**

Transação 3

O modelo para a transação (**LeituraEstrita**, **AcessaOBD**) é uma transação de leitura periódica com prazo estrito. Seremos sucintos na descrição dos detalhes para essa transação, visto que suas características de modelagem são semelhantes as características das demais como podemos observar através da Figura 4.9. Na inscrição do arco de saída da transição **OBDLeituraE1** temos a ficha **1'{nrl=t1,impl=0,mill=milt1L}**. Para esse objeto temos que **nrl** identifica o tipo de dado, **impl** a imprecisão no atributo e **mill** a imprecisão total acumulada que é definida pelo valor da constante **milt1L**.

Transação 4

A transação (**LeituraSuave**, **AcessaOBDS**) é uma transação de leitura esporádica com prazo suave. Na inscrição do arco de saída da transição **OBDLeituraS** temos a ficha **1'{nrs=t1,impl=0}**. Para esse objeto temos que **nrs** identifica o tipo de dado, **impl** a

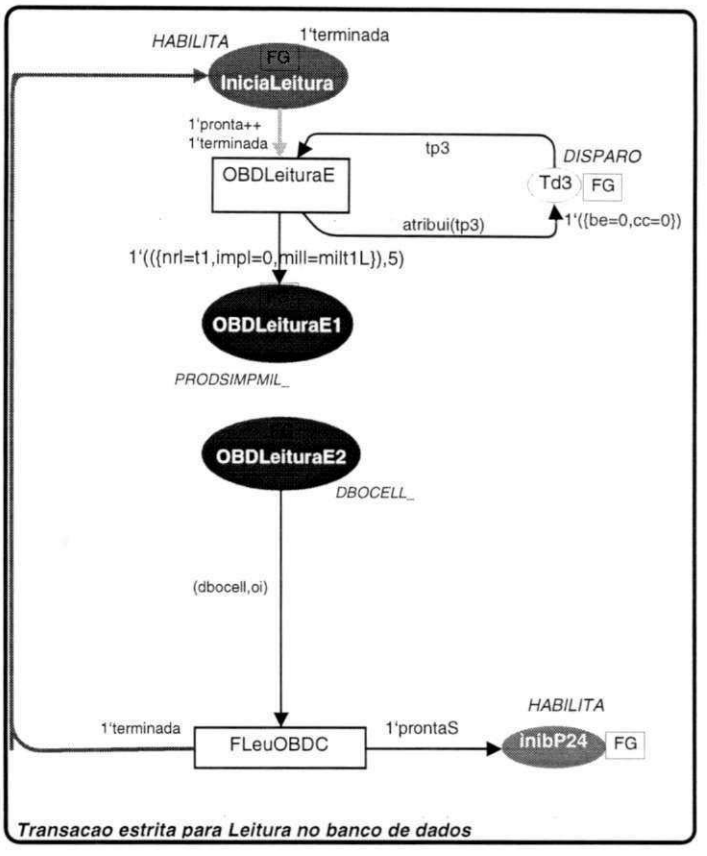

Figura 4.9: CPN para a **transação 3**

imprecisão no atributo. Note que como essa transação não será executada concorrentemente com nenhuma outra não haverá imprecisão acumulada.

Por ser uma transação suave, ela pode perder o prazo. No entanto, quanto mais transações suaves perderem seus prazos pior é o desempenho do sistema como um todo. Pode-se observar que as expressões dos arcos entre a transição **OBDLeituraS** e o lugar **InibidorP25** são **3'prontaS**. Assim essa transição só ficará habilitada se nenhuma transação com prazo estrito estiver executando, já que para a execução de uma transação com prazo estrito, uma ficha é removida do lugar **InibidorP25**.

4.3.2 Função de Compatibilidade

Tendo identificado as principais operações que serão executadas pelo objeto, o projetista deve analisar o sistema para descobrir quais as operações precisam ser executadas concorrentemente e em que condições elas podem ser executadas. Em seguida, ele define as funções de compatibilidade que descrevem detalhadamente as situações nas

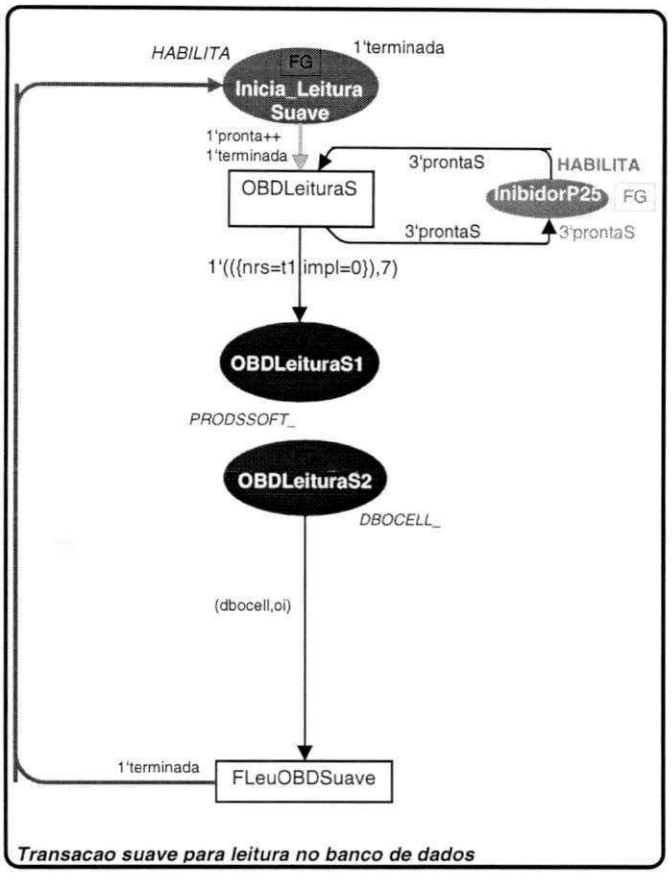

Figura 4.10: CPN para a **transação 4**

quais as operações podem ser executadas concorrentemente. Portanto, é importante que o método usado para modelar o sistema seja fundamentado em métodos formais para se verificar se essas funções estão sendo definidas corretamente, ou seja, se os limites definidos pelo projetista estão dentro de padrões aceitáveis. Em seguida, serão apresentadas três diferentes situações que a *FC* pode expressar.

Modelo CPN - Método Leitura Invocado e o Método Escrita Executando

Na Figura 4.11 mostra-se o modelo CPN para a situação quando o método *LeOBD* é invocado, e está ocorrendo a execução do método *Atualiza* para o mesmo item de dado **t1**. Na figura há o lugar **ELeOBDAtS**, que é onde verifica se já tem um outro método executando ou não. Se não tiver a ficha para esse lugar será **1'(livre,t1)**, igual sua marcação inicial, assim a transição **maOBDexecLe**, lado direito da figura, poderá ocorrer. Essa restrição é obedecida devido a guarda colocada na transição. Note que ao

ocorrer, a transição adiciona uma ficha no lugar **ArgsLeOBDAtS** e essa só será retirada desse lugar no momento que a transação estiver completada. Isso é observado pelo lugar de fusão correspondente ao lugar **EAtOBDLeR2** da Figura 4.7, adiciona também, uma ficha no lugar **LeOBDST**. Caso já exista um método sendo executado, a marcação do lugar **ELeOBDAtS** será: **1'(executando,t1)**. Assim, a transição **maOBDexecLe** não ficará habilitada. No entanto a transição **AvaliaFCAtLe**, lado esquerdo da figura, será avaliada de acordo com os valores dos atributos dos objetos, podendo ocorrer ou não. Sua guarda representa a função de compatibilidade para esse tipo de implicação. Se ocorrer a transição, a ficha do lugar **ArgsAtOBDLeS** contendo o objeto que está executando será alterada. A transição tem uma visão do conteúdo do banco de dados através do lugar de fusão **LeOBD**. A transição também adiciona uma ficha no lugar **ArgsLeOBDAtS** e uma ficha no lugar **LeOBDST**, já definidos.

Modelo CPN - Método Escrita Invocado e o Método Leitura Executando

Na Figura 4.12 apresenta-se o modelo CPN para a situação de concorrência quando o método *Atualiza* é invocado, e o método *LeOBD* está executando no mesmo item de dado, no caso **t1**. Na Figura há o lugar **ELeOBDAt**, que é onde se verifica se já tem um outro método executando ou não. Se não tiver a ficha para esse lugar será **1'(livre,t1)**, igual sua marcação inicial, assim a transição **maOBDexecAt**, no centro da figura, poderá ocorrer. Essa restrição é obedecida devido a guarda colocada na transição. Note que ao ocorrer a transição adiciona uma ficha no lugar **ArgsAtOBDLeS** e essa só será retirada desse lugar no momento que a transação estiver completada. Isso é observado pelo lugar de fusão correspondente ao lugar **EAtOBDLeR2** da Figura 4.7, adiciona também uma ficha no lugar **FCOBDSaiLe**. Caso já exista um método sendo executado, a marcação do lugar **ELeOBDAtS** será: **1'(executando,t1)**. Assim, a transição **maOBDexecAt** não ficará habilitada. No entanto, se o método que está executando é o de leitura, a transição **AvaliaFCAtle**, lado direito da figura, será avaliada, podendo ocorrer. Sua guarda é representada pela função de compatibilidade definida para esse tipo de implicação. Se ocorrer a transição, a ficha do lugar **ArgsLeOBDAt** contendo o objeto que está executando, será alterada. A transição tem uma visão do conteúdo do banco de dados através do lugar de fusão **LeOBD**. A transição, também

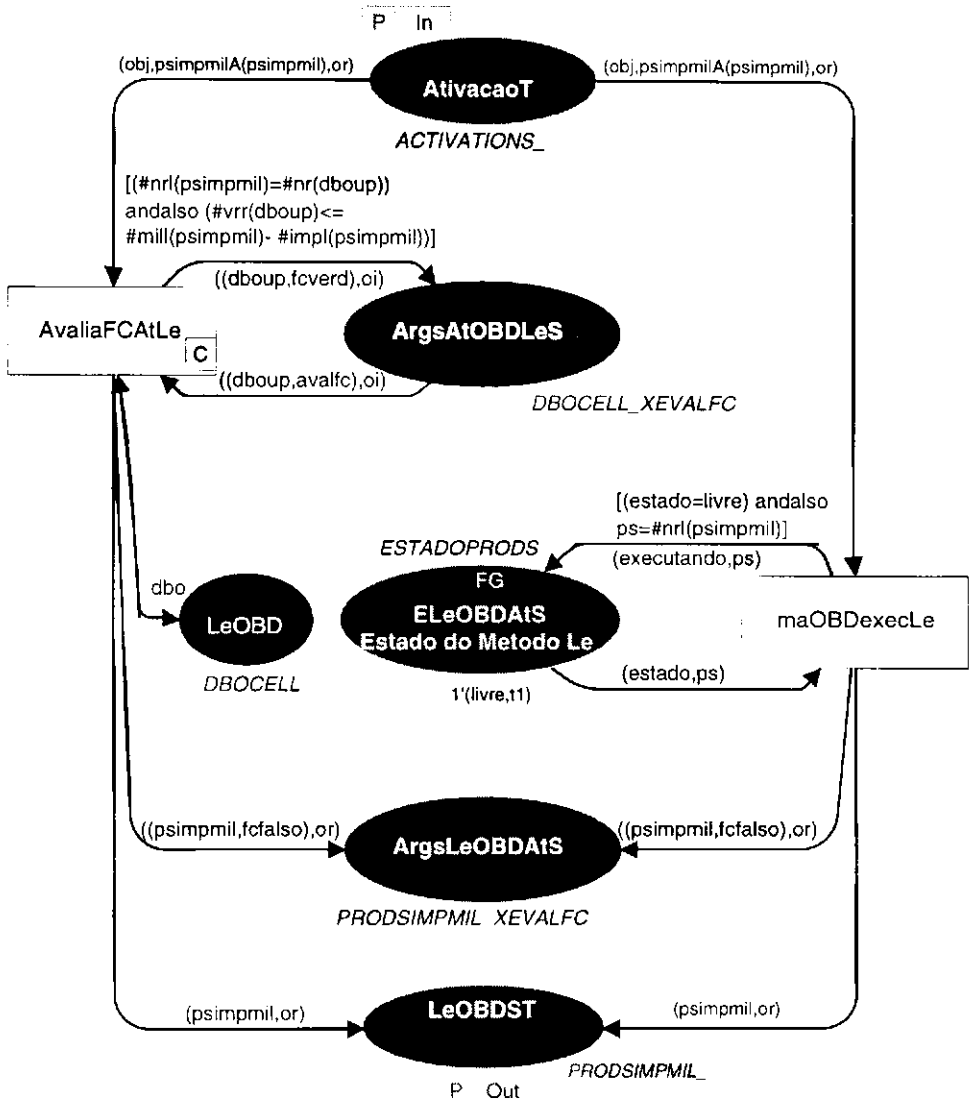

Figura 4.11: LeOBD é invocado enquanto Atualiza está executando

adiciona uma ficha no lugar **ArgsAtOBDLeS** e uma ficha no lugar **FCOBDSaiLe**, já definidos.

Modelo CPN - Método Escrita Invocado e o Método Escrita Executando

Na Figura 4.12 apresenta-se o modelo CPN para a situação de concorrência quando o método *Atualiza* é invocado, e esse mesmo método invocado por uma outra transação está executando no mesmo item de dado, no caso **t1**. Quando uma ficha é colocada no lugar **FCOBDEntAt**, a transição **AvaliaFCAtat** poderá ficar habilitada. Isso ocorre se no lugar **ArgsAtOBDLeS** tiver uma ficha, onde essa representa o objeto que está no mo-

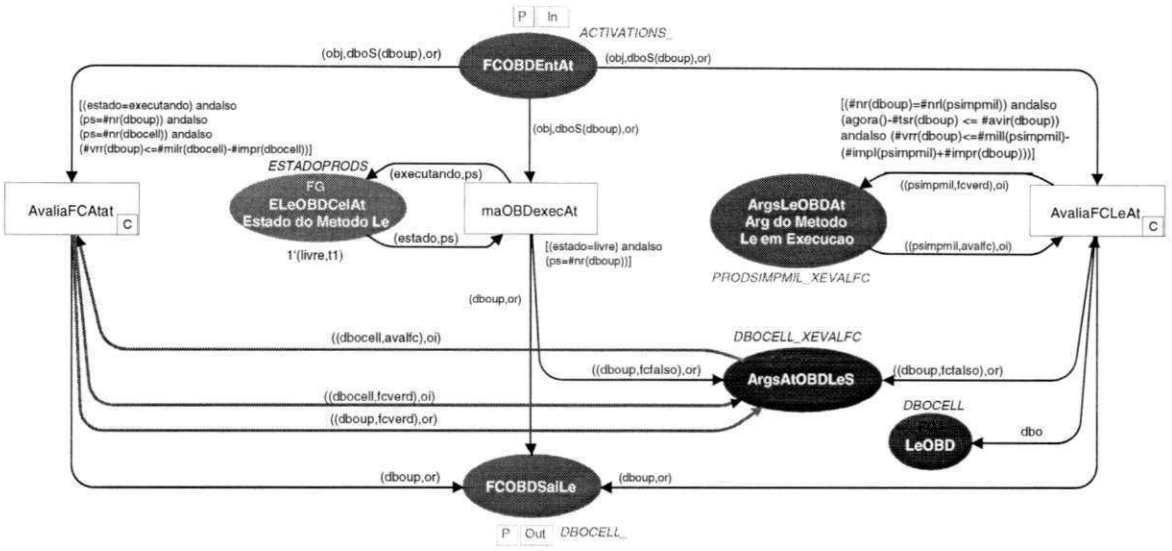

Figura 4.12: Atualiza é invocado enquanto LeOBD/Atualiza está executando

mento sendo executado pelo método de atualização. Assim, de acordo com a guarda da transição, lado esquerdo da figura, que representa a função de compatibilidade os atributos dos objetos serão considerados podendo a transição ocorrer ou não. Se ocorrer, a ficha do lugar **ArgsAtOBDLeS** contendo o objeto que está executando será alterada. A transição tem uma visão do conteúdo do banco de dados através do lugar de fusão **LeOBD**. A transição também adiciona uma ficha no lugar **ArgsAtOBDLeS**, informando que nesse momento esse objeto está sendo executado pelo método de atualização e uma ficha no lugar **FCOBDSaiLe**, já definido.

Capítulo 5

Análise do Modelo

Nesse capítulo aplicamos um procedimento para análise de modelos CPN descrito em [Per00]. O procedimento é exemplificado com base no modelo para o sistema de gerenciamento de bancos de dados em tempo-real apresentado no Capítulo 4. O modelo é analisado utilizando cenários e diagramas de seqüências de mensagens e através da geração e da análise do espaço de estados para os cenários. O pacote de ferramentas Design/CPN [Ja96; Ja99] é utilizado em todo o processo de construção e análise do modelo.

O processo de análise do modelo da aplicação de tempo-real tem como objetivo a verificação e, possivelmente, a correção das especificações e das restrições de tempo, a fim de garantir a escalonabilidade das transações. Nesse trabalho, em particular, o processo de análise tem que certificar que toda transação seja completada dentro de seu prazo de execução. Para alcançar essa meta, o modelo para banco de dados em tempo-real engloba as características da técnica de controle de concorrência semântico, que permite a especificação de escalonamentos mais flexíveis do que aqueles permitidos por seriação, através da definição das restrições das funções de compatibilidade suficientes para limitar a imprecisão resultante.

A análise formal consiste em definir as propriedades gerais do modelo, que refletem o comportamento desejado do sistema especificado, e utilizar a especificação para provar formalmente tais propriedades. Para isso pode-se utilizar o grafo de ocorrência [JCK96], que é uma representação do espaço dos estados possíveis para um modelo de redes de Petri Coloridas.

5.1 Procedimento para Análise do Modelo

Como ilustrado na Figura 5.1, a análise do modelo é realizada em dois passos: análise de cenários e análise de escalonamento.

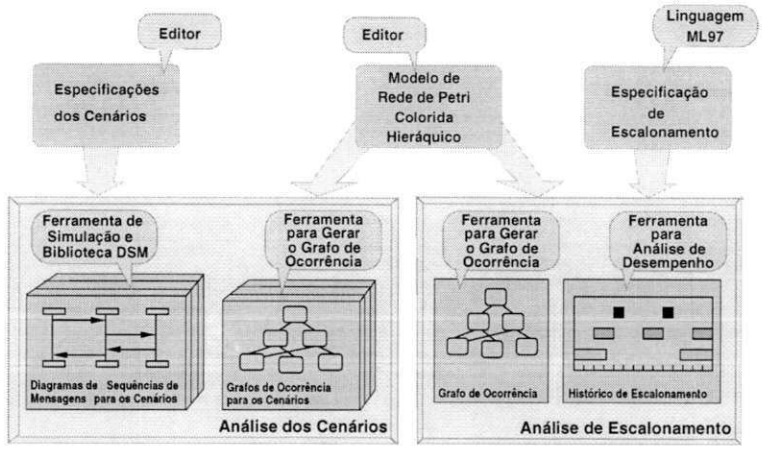

Figura 5.1: Análise do modelo

No primeiro passo são definidos os cenários. Para os cenários geramos os *diagramas de seqüências de mensagens*. Para a análise do modelo foram considerados diversos cenários. Para cada um desses obteve-se, através de simulação do modelo CPN, o diagrama de seqüência de mensagens entre os objetos. Uma vez que esse diagrama é obtido considerando apenas uma possível seqüência de execução, o espaço de estados é então construído, no caso o grafo de ocorrência, e verificado se os resultados particulares para uma das seqüências de execução são válidos para todas as seqüências de execução.

A notação utilizada para os diagramas de seqüências de mensagens apresentados nas seções seguintes, é apresentada na Figura 5.2.

No segundo passo, executa-se a análise de escalonamento para as transações periódicas ou esporádicas com prazos estritos ou suaves.

5.1.1 Análise de Cenário para Controle de Concorrência

O seguinte cenário foi utilizado para análise do modelo:

Atualização da quantidade do item de dado **t1** no objeto de banco de dados **ObjetoBD**, executando as transações representadas pelos pares **Atualiza1,Acessa1** e **Atualiza2,Acessa2**, ilustradas na Figura 4.5, bem como execução da leitura do objeto de banco

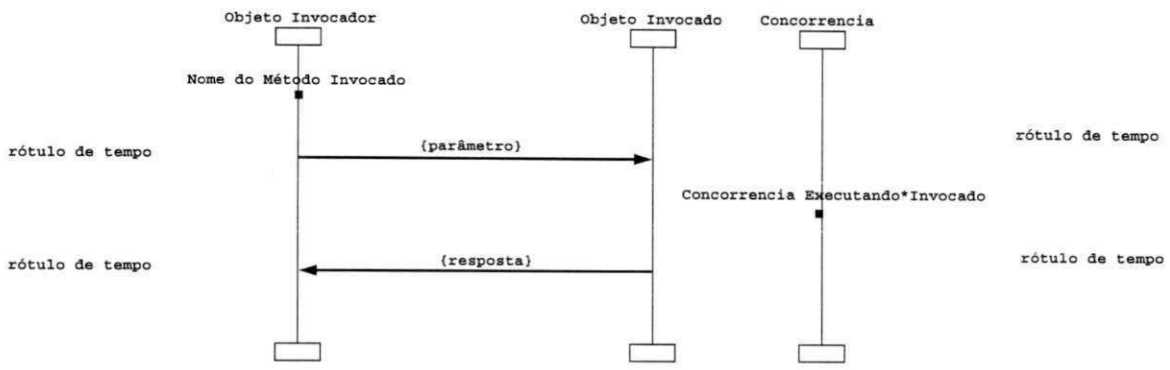

Figura 5.2: Notação para diagramas de seqüência de mensagens entre os objetos

de dados executando as transações representadas pelos pares **LeituraEstrirta,AcessaOBD** e **LeituraSuave,AcessaOBDS**, na mesma figura.

Atualização e Leitura no objeto de banco de dados

Para analisar a atualização e a leitura no objeto de banco de dados, o seguinte cenário foi utilizado:

- marcação inicial do lugar **Inicia1**, Figura 4.5, indicando que a transação para atualização de ítens de dados no banco de dados **ObjetoBD** está pronta para ser executada, ou seja: **1'terminada++1'pronta**. Essa transação atualizará uma unidade do item **t1**, como descrito na inscrição do arco de entrada do lugar **OBDAtualizaS1**, ou seja: **1'{nr=t1,vrr=1,avir=10,tsr=0,impr=1,milr=8}**.

- marcação inicial do lugar **Inicia2**, Figura 4.5, indicando que a transação para atualização no banco de dados **ObjetoBD** está pronta para ser executada, ou seja: **1'terminada++1'pronta**. Essa transação atualizará duas unidades, como descrito na inscrição do arco de entrada do lugar **OBDAtualizaT2**, ou seja: **1'{nr=t1,vrr=2,avir=10,tsr=0,impr=1,milr=8}**.

- marcação inicial do lugar **IniciaLeitura**, Figura 4.5, indicando que a transação para leitura da quantidade de itens de dados no banco de dados **ObjetoBD** está pronta para ser executada, ou seja: **1'terminada++1'pronta**.

- marcação inicial do lugar **Inicia_Leitura Suave**, Figura 4.5, indicando que a transação para leitura da quantidade de itens de dados no banco de dados **ObjetoBD**

onde N é o número de vezes que a transação executou, X é um número natural diferente de zero, $MMCPer$ é o mínimo múltiplo comum dos períodos e *Período* é o período de uma dada transação.

O grafo de ocorrência não foi obtido simplesmente por usar a função *EqualsUntimed()* como critério de parada, porque para a página **Sincronizador** há um lugar denominado de **Gerador de Contexto** que atribui um número inteiro diferente para cada objeto manipulado pela execução da transação. Dessa forma, não existirão marcações equivalentes, mesmo não considerando os rótulos de tempo dos arcos no grafo de ocorrência. A página **Sincronizador** foi definida no Capítulo 4 e pode ser encontrada em [Per00].

Análise das Propriedades do Modelo

Uma vez que o grafo de ocorrência foi construído, pode-se analisar e/ou verificar as propriedades do sistema consideradas necessárias. A ferramenta que permite a construção do grafo de ocorrência fornece um relatório com as propriedades gerais do modelo. Esse relatório contém informações sobre o grafo de ocorrência e sobre as propriedades dinâmicas que podem ser deduzidas dele.

O relatório, gerado automaticamente, contém diversas informações úteis sobre o comportamento do modelo CPN. Entretanto, pode ser necessário verificar algumas propriedades que são particulares ao modelo, como por exemplo, se as tarefas são executadas dentro de seus prazos de execução.

Uma vez obtido o grafo de ocorrência, o que interessa é obter todas as seqüências de escalonamento possíveis. As funções, na Figura 5.4, mostram a marcação terminal e o menor caminho existente entre a marcação inicial e a marcação final.

Para obtenção desses resultados, inicialmente constrói-se o grafo de ocorrência para o modelo. A seguir, realizamos uma pesquisa para verificar se a marcação terminal é alcançada ou se as marcações terminais são alcançáveis. Uma marcação terminal é definida pela existência de uma ficha **1'terminada** em uma dada marcação, seguida, em algum estado anterior, de uma marcação **1'terminada++1'pronta**. Isso significa que uma determinada transação foi executada dentro do seu prazo e alcançou um estado em que pode ser novamente executada. Determinadas as marcações terminais, encontra-se

```
SearchNodes (
  EntireGraph,
  fn n => (length(OutArcs(n)) = 0),
  100,
  fn n => n,
  [ ],
  op ::)
```

```
val it = [62] : Node list
```

```
Reachable' (1,62)
```

```
A path from node 1 to node 62 is: [1, 2, 3, 5,
9, 10, 12, 14, 16, 18, 19, 21, 22, 23, 24, 26,
28, 30, 32, 34, 36, 37, 39, 42, 46, 49, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62]
val it = true : bool
```

Figura 5.4: Funções para encontrar a marcação terminal e o menor caminho

a menor seqüência de ocorrência de transições, no caso de existir mais de uma que transforma a marcação inicial em marcações terminais para todas as transações. Essa seqüência define um escalonamento realizável para o conjunto de transações.

A função **AllPath** [Sou99] retorna todos os caminhos entre duas marcações. Os parâmetros são as marcações inicial e final para pesquisa e o tamanho do caminho a ser analisado. O valor zero faz com que sejam retornados caminhos de qualquer comprimento. Inicialmente são verificados todos os nós de saída adjacentes ao nó correspondente a marcação inicial da pesquisa (função **OutNodes** da ferramenta Design/CPN). A partir do exame desses nós de saída, inicia-se a formação de uma lista contendo o caminho em direção ao nó correspondente a marcação final da pesquisa, respeitando-se o comprimento especificado e a restrição de que um nó, que já se encontre previamente na lista, não seja repetido.

Na Figura 5.5, é mostrado o resultado da função **AllPath**, considerando a marcação terminal encontrada, mostrada na Figura 5.4. A função retornou todos os caminhos possíveis, independentemente do comprimento, entre a marcação inicial e terminal.

```
AllPath(1,62,0);
```

```
val it =
 [[1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  [1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  [1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  [1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  [1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  [1,2,3,6,9,10,12,14,16,18,20,21,...],[1,2,3,6,9,10,12,14,16,18,20,21,...],
  ...] : Node list list
```

Figura 5.5: Todos os caminhos da marcação inicial a terminal

Na Figura 5.6, apresentamos o diagrama de tempo para a execução dos méto-

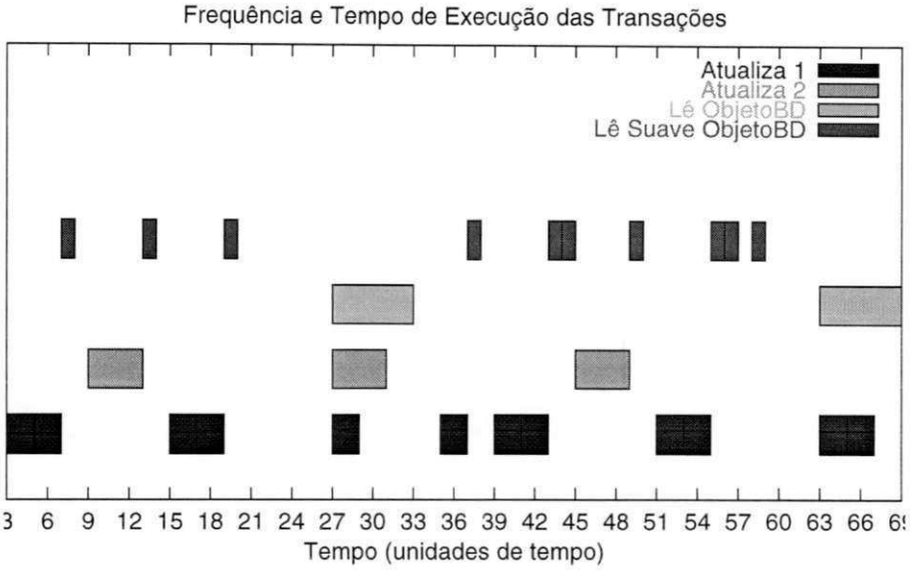

Figura 5.6: Diagrama de tempo para as transações

dos. Esse diagrama foi obtido utilizando a ferramenta para análise de desempenho do Design/CPN. Nessa figura tem-se a representação dos métodos do modelo, vistos na legenda. Os dados foram obtidos através de simulação sendo possível verificar que as restrições temporais foram satisfeitas. Observe que a **transação 1** iniciou sua execução no instante de tempo 3 e executou a **subtransação 2** no instante 5, executando novamente no instante de tempo 15. Já a **transação 2** executou no instante de tempo 9, no entanto, deveria ter executado no instante de tempo 63, o que não aconteceu.

Capítulo 6

Conclusão

Nesse trabalho discutimos e apresentamos um modelo para analisar escalonamento de transações para bancos dados em tempo-real, considerando mecanismos de controle de concorrência semântico com suporte a negociação entre a consistência lógica e a temporal dos dados e das transações. As contribuições incluem a definição de um modelo para banco de dados em tempo-real, usando redes de Petri Coloridas, a modelagem da técnica de controle de concorrência semântico, bem como as restrições da função de compatibilidade suficientes para manter a imprecisão limitada. Testes para analisar se as restrições temporais foram executados a fim de verificarmos como o modelo atende ao objetivo indicado.

6.1 Contribuições

A definição do modelo para o banco de dados em tempo-real teve como base o método desenvolvido em [Per00], definido no Capítulo 4. Esse método baseia-se em um modelo de dados em tempo-real orientado a objetos e em redes de Petri Coloridas.

A escolha de redes de Petri Coloridas como formalismo ocorreu devido à sua adequação à modelagem de sistemas complexos e concorrentes, bem como à existência de uma ferramenta eficiente para suporte à modelagem e análise de tais sistemas.

A técnica de controle de concorrência semântico modelada visa manter a consistência lógica e a consistência temporal dos dados e das transações. A consistência lógica das transações é mantida com base na consistência semântica definida pelo projetista.

Bibliografia

[AM92] R. K. Abbot and H. G. Molina. Schedulling real-time transactions: A performance evaluation. *ACM Transactions on Database Systems*, 17(3):513–561, September 1992.

[BLS97] A. Bestravos, K.-J. Lin, and Son. S.H. *Real-Time Database Systems: Issues and Applications.* Kluwer Academic Publishers, Boston, 1997.

[Boo94] G. Booch. *Object Oriented Design: With Applications.* Menlo Park, CA: Benjamin/Cummings, 1994.

[BP98] M. Blaha and W. Premerlani. *Object-Oriented Modeling and Design for Database Applications.* Prentice Hall, 1998.

[BR88] B.R. Bradinath and K. Ramamrithman. Syncrhoning transactions on objects. *IEEE Transactions on Computers*, 37(5):541–547, 1988.

[BR90] I.I. Bestuzheva and V.V. Rudnev. Timed petri nets: Classification and comparative analysis. *Automation and Remote Control*, 51(10):1303 – 1318, October 1990.

[CGdFP98] S.A.D. Costa, D.D.S. Guererro, J.C.A. de Figueiredo, and A. Perkusich. Aspectos de herança em uma ferramenta de modelagem de sistemas baseada em redes de petri. In *Anais do SBES'98, Simpósio Brasileiro de Engenharia de Software*, pages 297–312, Londrina, PR, October 1998.

[CJK97] S. Christensen, J. B. Joergensen, and L. M. Kristensen. Design/CPN — A computer tool for coloured Petri nets. *Lecture Notes in Computer Science*, 1217, 1997.

[Cos99] Sandro Alex Damasceno Costa. Aspectos de herança em uma notação orientada a objetos baseada em redes de petri. Dissertação de mestrado, COPIN - Universidade Federal da Paraíba, 1999.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[DiP95] L.C. DiPippo. *Semantic Real-Time Object-based Concurrency Control.* PhD thesis, Department of Computer Science and Statistics, University of Rhode Island, 1995.

[dMGdFP98] A.K.A. de Medeiros, D.D.S. Guererro, J.C.A. de Figueiredo, and A. Perkusich. An object-oriented petri-net modeling tool and abstraction mechanisms for cooperative systems. In *Proc. of IEEE Int. Conf. on Systems Man and Cybernetics*, pages 172–177, San Diego, USA, October 1998.

[Dou98] B.P. Douglass. *Real-Time "UML": Developing Efficient Objects for Embedded Systems.* Addison Wesley, Reading, Massachusets, 1998.

[EN94] R.A. Elmasri and S.B. Navathe. *Fundamentals of Database Systems, 2nd Edition.* Addison–Wesley Pub. Co., New York, 1994.

[Fig94] J.C.A. de Figueiredo. *Redes de Petri com Temporização Nebulosa.* PhD thesis, Curso de Doutorado em Engenharia Elétrica, Universidade Federal da Paraíba, Campina Grande, PB, August 1994.

[GdFP97] D.D.S. Guerrero, J.C.A. de Figueiredo, and A. Perkusich. Object-based high-level petri nets as a formal approach to distributed information systems. In *Proc. of IEEE Int. Conf. on Systems Man and Cybernetics*, pages 3383–3388, Orlando, USA, October 1997.

[GMMP89] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A general way to put time in petri net. In *5th International Workshop on Software Specifications and Design*, pages 60–66, May 1989.

[GPdF97] D.D.S. Guerrero, A. Perkusich, and J.C.A. de Figueiredo. Modeling a cooperative environment based on an object-based modular petri net. In *Proc. of The 9th International Conference on Software Engineering and Knowledge Engineering*, pages 240–247, Madrid, Spain, June 1997.

[Gue97] D.D.S. Guerrero. Sistemas de redes de petri modulares baseadas em objetos. Dissertação de mestrado, Curso de Mestrado em Informática - Universidade Federal da Paraíba, Campina Grande, Paraíba, 1997.

[Gue98a] Dalton D. S. Guerrero. Especificação e análise de sistemas concorrentes utilizando redes de petri e orientação a objetos. Exame de qualificação, Coordenação de Pós-graduação em Engenharia Elétrica - COPELE/UFPB, Campina Grande, PB, July 1998.

[Gue98b] Dalton D. S. Guerrero. Orientação a objetos e modelos de redes de petri. Technical report, Coordenação de Pós-graduação em Engenharia Elétrica - COPELE/UFPB, Campina Grande, PB, April 1998.

[HS86] P.J. Haas and G.S. Shedler. Regenerative stochastic petri nets. *Performance Evaluation*, 6(3):189–204, September 1986.

[Ja96] K. Jensen and et. al. *"Design/CPN"Manuals.* Meta Software Corporation and Department of Computer Science, University of Aarhus, Denmark, 1996. On-line version:http://www.daimi.aau.dk/designCPN/.

[Ja99] K. Jensen and et. al. *"Design/CPN"4.0.* Meta Software Corporation and Department of Computer Science, University of Aarhus, Denmark, 1999. On-line version:http://www.daimi.aau.dk/designCPN/.

[Jal94] P. Jalote. *Fault Tolerance in Distributed Systems.* Prentice Hall, New Jersey, 1994.

[JCK96] K. Jensen, S. Christensen, and L.M. Kristensen. *Design/CPN Occurence Graph Manual.* University of Aarhus, Aarhus, Denmark, 1996.

[Jen87] K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Lecture Notes in Computer Science, Advances in Petri Nets: Petri Nets, Central Models and Their Properties*, volume 254, pages 248–299. Springer-Verlag, 1987.

[Jen92a] K. Jensen. *Coloured Petri Nets - basic concepts, analysis methods and practical use - vol. 1 : basic concepts, Ver. 4.* Springer-Verlag, Paris, 1992.

[Jen92b] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use.* EACTS – Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[Jen97] K. Jensen. *Coloured petri nets-Basic Concepts, Analysis Methods and Practical Use*, volume 2. Springer-Verlag, 1997.

[Jen98] K. Jensen. An introduction to the practical use of coloured petri nets. In *Lectures on Petri Nets II: Applications.* Springer, 1998.

[Jon98] Greg Jones. *Scheduling with Temporal and Logical Constraints.* PhD thesis, Department of Computer Science and Statistics, University of Rhode Island, 1998.

[KS96] C. Krishna and K. G. Shin. *Real-Time Systems.* MacGraw Hill Series in Computer Science, 1996.

[Liu00] S.W.J. Liu. Real-Time Systems. In *Prentice-Hall,New Jersey*, 2000.

[LL99] V.C.S. Lee and K.W. Lam. Conflict free transaction scheduling using serialization graph for real-time databases. *Systems and Software*, (55):57–65, 1999.

[LS94] J. Lee and S. H. Son. Semantic-Based Concurrency Control for Object-Oriented Database Systems Supporting Real-Time Applications. *6th IEEE Euromicro Workshop on Real-Time Systems*, pages 156–161, July 1994.

[Mar95] T. Marx. Netcase - a petri net based method for database application design and generation. Technical report, University of Koblenz, Germany, Research Report 11-95, 1995.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[NPLT01] Pedro Fernandes Ribeiro Neto, Angelo Perkusich, Maria L.B.Perkusich, and Maria F.Q.V. Turnell. Analysis of Periodic Transactions and Semantic Concurrency Control for Real-Time Databases using Colored Petri Nets. *IEEE Systems, Man and Cybernetics Conference*, 2001.

[OS95] G. Ozsoyoglu and R.T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Transactions on Data and Knowledge Engineering*, 7(4):47–56, August 1995.

[Per00] M.L.B. Perkusich. *Um Método Baseado em Redes de Petri para a Modelagem de Bancos de Dados para Aplicações em Tempo-Real.* Tese de doutorado, Curso de Doutorado em Engenharia Elétrica, CCT/UFPB, April 2000.

[PMRN01] A. Perkusich, Perkusich M.L.B., and P.F. Ribeiro Neto. Escalonamento de Transações para Bancos de Dados de Tempo-real Usando Redes de Petri Coloridas. *III Worshop de Tempo Real*, (3), 2001.

[PTP99] M.L.B. Perkusich, M.F.Q.V. Turnell, and A. Perkusich. Modelagem de bancos de dados em tempo-real. In *Anais IV Simpósio Brasileiro de Banco de Dados*, pages 253–267, Florianópolis, SC, October 1999.

[PWPD96] J. Peckham, V.F. Wolfe, J.J. Prichard, and L.C. DiPippo. Design of a real-time object-oriented database system. Technical report, University of Rhode Island, TR94-231, 1996.

[Ram93] K. Ramamrithman. Real-time databases. *International Journal of Distributed and Parallel Databases*, 1993.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual 1st edition.* Addison Wesley, Reading, Massachusets, 1999.

[RP95] K. Ramamritham and C. Pu. A formal characterization of epsilon serializability. *IEEE Transactions on Data and Knowledge Engineering*, 6(7):997–1007, 1995.

[Sou99] M.R.F. de Souza. *Avaliação Iterativa da Especificação de Interfaces com Ênfase na Navegação.* PhD thesis, Curso de Doutorado em Engenharia Elétrica, Universidade Federal da Paraíba, Campina Grande, PB, December 1999.

[SP95] S.H. Son and S. Park. A Priority-Based Scheduling Algorithm for Real-Time Databases. *Information Science and Engineering*, 11(2), 1995.

[SSH99] J. Stankovic, S. Son, and J. Hansson. Misconceptions about real-time databases. *IEEE Computer*, 32(6):29–36, 1999.

[WA92] M. Wong and D. Agrawal. Tolerating bounded inconsistency for increasing concurrency in database systems. In *Proc. of 11th Principles of Databases Systems*, pages 236–245, Tucson, AZ, 1992.

[YWLS94] P.S. Yu, K.-L. Wu, K.-J. Lin, and S.H. Son. On real-time databases: Concurrency control and schedulling. *Proceedings of IEEE*, pages 140–157, January 1994.

[Zub91] W.M. Zuberek. Timed petri nets: Definitions, properties, and applications. *Microelectronics and Reliability*, 31(4):627 – 644, 1991.