Mecanismos de Qualidade de Serviços para o Gerenciamento de Dados e Transações em Tempo-Real

Pedro Fernandes Ribeiro Neto

Tese de Doutorado submetida à Coordenação do Programa de Pós-Gradução em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Angelo Perkusich, D.Sc.
Orientador
Maria Lígia Barbosa Perkusich, D.Sc.
Orientadora

Campina Grande, Paraíba, Brasil

Mecanismos de Qualidade de Serviços para o Gerenciamento de Dados e Transações em Tempo-Real

Pedro Fernandes Ribeiro Neto

Tese de Doutorado apresentada em Março de 2006

Angelo Perkusich, D.Sc.
Orientador
Maria Lígia Barbosa Perkusich, D.Sc.
Orientadora

Ângelo Roncalli Alencar Brayner, Dr.
Componente da Banca
Fernando da Fonseca de Souza, Dr.
Componente da Banca
José Sérgio da Rocha Neto, D.Sc.
Componente da Banca
Maria de Fátima Queiroz Vieira Turnell, Ph.D.
Componente da Banca

Campina Grande, Paraíba, Brasil, Março de 2006

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

R484m 2006

Ribeiro Neto, Pedro Fernandes

Mecanismos de qualidade de serviços para o gerenciamento de dados e transações em tempo-real/ Pedro Fernandes Ribeiro Neto. — Campina Grande, 2006.

145f. il.

Referências.

Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientadores: Ângelo Perkusich, D.Sc. e Maria Lígia Barbosa Perkusich, D.Sc.

1— Redes de sensores 2— Redes de Petri 3— Qualidade de serviço 4— Bancos de dados em tempo-real I— Título

CDU 681.3.02:681.3.014

MECANISMOS DE QUALIDADE DE SERVIÇOS PARA O GERENCIAMENTO DE DADOS E TRANSAÇÕES EM TEMPO-REAL

PEDRO FERNANDES RIBEIRO NETO

Tese Aprovada em 16.03.2006

ANGELO PERKUSICH, D.Sc., UFCG
Orientador

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

FERNANDO DA FONSECA DE SOUZA, Dr., UFPE
Componente da Banca

ANGELO RONCALLI ALENCAR BRAYNER, Dr., UFC
Componente da Banca

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

JOSÉ SÉRGIO DA ROCHA NETO, D.Sc., UFCG
Componente da Banca

CAMPINA GRANDE – PB
MARÇO - 2006

Resumo

Os sistemas de gerenciamento de banco de dados em tempo-real (SGBD-TR) apresentam as características necessárias para gerenciar eficientemente aplicações onde dados e transações possuem restrições temporais, tais como: aplicações para automação industrial, aviação, redes de sensores, entre outras. Na última década, várias aplicações que precisam executar em ambientes abertos e imprevisíveis foram identificadas. Tais aplicações caracterizam-se pela grande distribuição geográfica, alta heterogeneidade, inexistência de controle global, falhas parciais e falta de segurança.

Em virtude das características supracitadas, muitas questões em SGBD-TR precisam ser reconsideradas, tais como: mecanismos de controle de concorrência, políticas de escalonamento e gerenciamento de qualidade de serviços. Por exemplo, os algoritmos de escalonamento devem considerar os diferentes tipos de prazos das transações (suave, firme ou estrito), enquanto os protocolos de controle de concorrência devem permitir que transações conflitantes executem concorrentemente, de acordo com as necessidades da aplicação, onde essas necessidades são especificadas através de funções de qualidade de serviços e métricas de desempenho. Portanto, considerando a complexidade dessas aplicações, torna-se fundamental o desenvolvimento de métodos baseados em técnicas formais de análise e simulação de sistemas para o desenvolvimento das mesmas.

Nesta tese é proposto um método baseado em uma técnica formal, para o desenvolvimento de aplicações que executam em ambientes abertos e imprevisíveis. Através desse método é possível realizar análises e simulações do sistema, buscando orientar o processo de tomada de decisão e propor soluções para a melhoria da mesma. Para validar o método, um estudo de caso considerando o domínio de aplicação de redes de sensores foi adotado.

Abstract

Real-time database management systems (RT-DBMS) have the necessary characteristics for providing efficient support to develop applications in which both, data and transactions have temporal constraints, such as industrial automation, aviation, sensors network, and so on. However, in the last decade, new applications were identified. Such applications are characterized by a large geographic distribution, high heterogeneity, lack of global control, partial failures and lack of safety. Besides of this, they need to manage large data volumes with real-time constraints.

Due to these characteristics, many questions in RT-DBMS have brought interest to researches in this area, such as: concurrence control mechanisms, scheduling policy, and quality of services management. Scheduling algorithms should consider transactions with soft deadlines and the concurrency control protocols should allow conflicting transactions to execute in parallel. The last ones should be based in their requirements, which are specified through both quality of services functions and performance metrics. However, considering the complexity of these applications, it is fundamental to develop formal analysis and simulation techniques for systems with the discussed characteristics.

In this work, a method to model and develop applications that execute in open and unpredictable environments is proposed. Based on this method this model, it is possible to perform analysis and simulations of systems, to guide the decision making process and to identify solutions for improving it. For validating the model, a case study considering the application domain of sensors network is discussed.

Sumário

Lista de Símbolos e Abreviaturas

FE	Função de Especificação
FM	Função de Mapeamento
FMo	Função de Monitoração
FN	Função de Negociação
FR	Função de Re-negociação
$Impr$	Imprecisão Máxima do Dado
Pt	Número de Transações que Perderam o Prazo
2PL-HP	Protocolo de Bloqueio de Duas Fases com Alta Prioridade
2PLP	Protocolo de Bloqueio de Duas Fases Pessimista
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ADC	Conversor analógico-digital
ANSI	*American National Standards Institute*
Ap	Aperiódicas
BD	Banco de Dados
BDS	Banco de Dados Sensores
BDW	Banco de Dados *Warehousing*
CASE	*Computer Aided Software Engineering*
CC	Controle de Concorrência
CCO	Protocolo de Controle de Concorrência Otimista
DSM	Diagrama de seqüência de mensagens
E/S	Entrada e Saída
Es	Esporádica
FC	Função de Compatibilidade
HCPN	*Hierarchical Coloured Petri Nets*

HCPN	*Hierarchical Coloured Petri Nets*
LC-BDTR	Linguagem de Consulta para Banco de Dados em Tempo-Real
OG	Grafo de Ocorrência
Pe	Período
Pr	Prazo
QoS	Qualidade de Serviços
RS	Redes de sensores
RTSQL	*Real-Time SQL*
SE	Serialização *Epsilon*
SEOO	Serialização *Epsilon* Orientada a Objetos
SEOO	Serialização *Epsilon* Orientada a Objetos
SGBD	Sistemas de Gerenciamento de Banco de Dados
SGBD-TR	Sistemas de Gerenciamento de Banco de Dados em Tempo-Real
SQL	*Structured Query Language*
STR	Sistemas em Tempo-Real
TAS	Transação de Aquisição Sensores
tc	Tempo computacional
TES	Transação de Escrita Sensores
TEW	Transação de Escrita Servidor
tl	Tempo de Liberação
TLS	Transação de Leitura Sensores
TLW	Transação de Leitura Servidor
UCP	Unidade Central de Processamento
UML	*Unified Modeling Language*

Lista de Figuras

Capítulo 1

Introdução

Os Sistemas de Gerenciamento de Banco de Dados em Tempo-Real (SGBD-TR) são apropriados para gerenciar grandes volumes de dados compartilhados em aplicações de tempo-real. Esses sistemas são projetados para, adicionadas as funcionalidades dos bancos de dados convencionais, incluir as características necessárias para prover suporte à manipulação de dados e transações satisfazendo restrições temporais.

Os SGBD-TR são úteis para Sistemas em Tempo-Real (STR) com restrições de tempo não negociáveis, tais como aplicações em aviões e automóveis, onde os prazos temporais impostos aos dados e as transações não podem ser perdidos sob a pena de gerar alguma catástrofe. Nas últimas décadas, o domínio de aplicação dos STR tem se expandido para aplicações que executam em ambientes imprevisíveis, tais como redes de sensores, comércio eletrônico e bolsa de valores *on-line*, nos quais quanto mais prazos forem satisfeitos melhor é o desempenho do sistema.

Os ambientes de execução imprevisíveis e, conseqüentemente, a necessidade de políticas de escalonamentos e de protocolos de controle de concorrência que possibilitem ao usuário especificar níveis de corretude e de desempenho aceitáveis para um SGBD-TR, são o foco desta tese.

Como solução, mecanismos de Qualidade de Serviços (QoS) são utilizados para definir os níveis de corretude e de desempenho que devem ser satisfeitos. Um modelo formal é introduzido para de validar a efetividade de mecanismos de QoS para gerenciar dados e transações com validade temporal. A modelagem matemática se justifica pela possibilidade de submeter o modelo a procedimentos automatizados de verificação e validação formal.

Ainda nesse Capítulo, a motivação e o contexto do trabalho são apresentados na Seção 1.1. As declarações dos problemas que motivaram a realização desta pesquisa são descritos na Seção 1.2. Em seguida, na Seção 1.3, são descritos os objetivos da tese, separando-os em objetivos gerais e específicos. Também é apresentada a metodologia utilizada durante a

pesquisa na Seção 1.4 e por fim, na Seção 1.5, é apresentada a estrutura deste documento.

1.1 Contexto e Motivação

Sistemas de Banco de Dados

Um sistema de banco de dados consiste de um conjunto de dados denotado Banco de Dados (BD), e um componente de software denotado Sistema de Gerenciamento de Banco de Dados (SGBD). O BD é um depósito auto-descritivo, onde os dados são armazenados permanentemente em um ou mais arquivos. SGBD é um sistema de software que gerencia o acesso concorrente ao BD, realizado por vários usuários e programas. A fim de evitar que inconsistências sejam geradas, o SGBD deve prover mecanismos para controlar a execução concorrente de diferentes programas ao BD.

O acesso ao conteúdo do BD é realizado através de *transações*. O conceito de transações foi introduzido em 1976, como parte de um modelo proposto por Eswaran et al. (1976). Neste modelo, as transações são definidas como uma abstração que representa uma seqüência de operações de banco de dados resultante da execução de um programa.

O modelo de transação tem sido consolidado como uma solução padrão para os protocolos de controle de concorrência nos sistemas de banco de dados (BRAYNER, 1999). O principal objetivo desses protocolos é promover a concorrência máxima entre as transações, assegurando a consistência do banco de dados.

Os protocolos de controle de concorrência baseiam-se no conceito de *serialização* para determinar quais transações podem executar concorrentemente. A execução intercalada das operações de diferentes transações é considerada correta, ou serializável, se produzirem o mesmo efeito de uma execução serial para as mesmas transações. Esses protocolos asseguram as propriedades ACID[1] das transações que impõem restrições na ordem de execução das transações (restrições de *consistência lógica das transações*) e como os dados podem ser acessados (restrições de *consistência lógica dos dados*) (ELMASRI; NAVATHE, 1994).

Sistemas em Tempo-Real

A corretude de muitos sistemas computacionais depende tanto dos resultados lógicos produzidos por eles, como do tempo em que esses resultados são produzidos. Esses sistemas são comumente denominados de sistemas em tempo-real.

Em um STR, as especificações de tempo são impostas às tarefas[2] na forma de tempo

[1] Acrônimo para Atomicidade, Consistência, Isolamento e Durabilidade.

[2] Unidade de execução com restrição temporal.

de liberação, tempo computacional, período e prazo.

- O Tempo de Liberação (tl) é o momento no qual a tarefa está pronta para executar, e tl pode ser calssificado em tempo de início mais cedo e tempo de início mais tarde da tarefa. O primeiro especifica um tempo absoluto antes do qual a execução da tarefa não pode iniciar e o segundo especifica um tempo absoluto antes do qual a execução da tarefa dnão deve iniciar. Essa última restrição permite detectar a violação do prazo antes da tarefa executar.

- Tempo Computacional (tc) da tarefa, também denominado de tempo de execução, é o tempo necessário para a execução da tarefa sem interrupções. Essa especificação de tempo é crucial para o sucesso do escalonamento das tarefas. Determinar o tempo que uma tarefa utilizará um recurso é geralmente muito difícil. Por exemplo, considere o caso de determinar o tempo de Unidade de Central de Processamento (UCP) que uma tarefa precisa para executar uma computação. Para estabelecer o pior caso de utilização da UCP, todas as possíveis ramificações da tarefa devem ser consideradas para o pior caso e todos os laços e recursões devem ser consideradas para ter um número limitado de instâncias. No entanto, utilização de UCP é apenas um dos recursos que uma tarefa pode precisar. Uma tarefa também pode precisar utilizar memória principal, dispositivos de Entrada e Saída (E/S), etc. Além disso, podem existir dependências no uso dos recursos e, portanto não poder ser computado de forma isolada. De modo geral, estimativas da utilização de recursos podem ser determinadas. Entretanto, tais estimativas podem ser muito imprecisas.

- Período (Pe) da tarefa é o intervalo de tempo mínimo existente entre duas execuções consecutivas da mesma tarefa. Uma tarefa também pode ser esporádica ou aperiódica. Uma tarefa esporádica (Es) pode ou não executar, uma vez executada existe um intervalo de tempo mínimo entre duas execuções consecutivas da mesma tarefa. As tarefas aperiódicas (Ap) não possuem um intervalo de tempo regular entre suas execuções. Neste documento, esta taxonomia é chamada de *padrões de chegada das tarefas*.

- Prazo (Pr), traduzido do termo em inglês *deadline*, é o tempo absoluto antes do qual a tarefa deve ser completada. O prazo das tarefas pode ser classificado em relação às conseqüências de sua violação. *Estrito*, tradução do termo em inglês *hard*, é o prazo da tarefa que não pode ser perdido; *suave*, tradução do termo em inglês *soft*, é o prazo que pode ser perdido, porém à medida que prazos suaves são perdidos o desempenho do sistema pode ser comprometido e; *firme*, tradução do termo em inglês *firm*, é o prazo que se perdido não agrega nenhum valor ao sistema.

Em sistemas sem restrições de tempo especificadas, o escalonamento ideal deve maximizar a vazão média, ou seja, completar o maior número de tarefas por unidade de tempo, e/ou minimizar o tempo médio de espera para as tarefas. Nos STR, o escalonamento ideal deve assegurar que cada tarefa complete sua execução dentro do prazo definido. Isso pode ser validado através da análise de escalonabilidade que determina se um conjunto de tarefas executando concorrentemente satisfaz as restrições de tempo e podem ser escalonadas com sucesso (CHENG, 2002; COOLING, 2003).

Contudo para a análise de escalonabilidade ser realizada é necessário que o sistema seja previsível, ou seja, as ações que serão executadas e os recursos necessários para isso devem ser conhecidas. Em Ribeiro Neto (2001), a análise de escalonabilidade de um conjunto de transações com diferentes especificações de tempo foi realizada.

Sistemas de Banco de Dados em Tempo-Real

Nas últimas duas décadas, pesquisas sobre SGBD-TR de modo a determinar métodos, técnicas e soluções para tratar com a complexidade imposta pela necessidade de garantir a integridade dos dados e satisfazer as restrições de tempo das transações têm sido realizadas. Assim, novas abordagens têm sido introduzidas no escopo de SGBD-TR com o objetivo de tratar com os diversos problemas decorrentes dos requisitos impostos, dentre os quais destacamos:

- Métodos para modelagem conceitual (RAMAMRITHAM, 1993; PECKHAM et al., 1996; PERKUSICH, 2000; RIBEIRO NETO, 2001);
- Gerenciadores de banco de dados (STANKOVIC; SON; LIEBEHERR, 1998; ANDLER et al., 1996; BUCHMANN et al., 1995; ADELBERG; KAO; GARCIA-MOLINA, 1996);
- Protocolos de controle de concorrência (NYSTRöM et al., 2004a; LAM et al., 2002; KAO et al., 1999);
- Algoritmos de escalonamento (KAO et al., 1999; KANG; SON; STANKOVIC, 2004; SANTOSHKUMAR; MANIMARAN; MURTHY, 1998; SANTOS; LIPARI; SANTOS, 2004; LU et al., 2004; HAUBERT; SADEG; AMANTON, 2004);
- Linguagens de consultas (PRICHARD, 1995; LEITE et al., 2005).

O foco definido desta tese está definido conexto de métodos para modelagem conceitual e controle de concorrência considerando qualidade de serviço.

Modelagem Conceitual

O termo *modelo conceitual* foi definido na área de banco de dados na década de 70 por um grupo de trabalho da associação norte-americana de normas (ANSI - *American National Standards Institute*). Um modelo conceitual é a representação gráfica/matemática das principais propriedades estáticas, funcionais e dinâmicas de um sistema de banco de dados. Uma linguagem de modelagem conceitual deve permitir que todas as propriedades desejáveis para o sistema sejam capturadas pela descrição do modelo, sem que propriedades indesejáveis sejam incluídas. Além disso, a linguagem de modelagem deve fornecer modelos facilmente compreensíveis pelos usuários e projetistas, e deve fornecer modelos exatos e não ambíguos, o que implica que a linguagem deve ter algum tipo de formalismo matemático que permita a análise e verificação das propriedades modeladas (PERKUSICH, 2000).

Através da análise do modelo conceitual, um sistema real pode ser estudado sem o perigo, custo ou inconveniência da manipulação de seus elementos. Quando se trata de sistemas complexos, o processo de análise do modelo precisa ser automatizado. Portanto é fundamental a utilização de modelos matemáticos, os quais possibilitam submeter os modelos a procedimentos automáticos de análise e verificação. Tais procedimentos permitem detectar uma série de deficiências que podem estar presentes no modelo conceitual, tais como: contradições, ambigüidades, redundâncias, incompletude, entre outras. No contexto desta tese, a base formal utilizada são as redes de Petri (MURATA, 1989).

Gerenciadores de Banco de Dados

Vários protótipos de SGBD-TR têm sido desenvolvidos nos últimos anos. O sistema BeeHive é um banco de dados em tempo-real orientado a objetos com ênfase em quatro dimensões: tempo-real, tolerância à faltas, segurança e qualidade de serviços. Esse sistema foi desenvolvido na Universidade da Virginia, EUA (STANKOVIC; SON; LIEBEHERR, 1998). Outro sistema é o DeeDS, que suporta transações em tempo-real com prazos suaves e estritos. Foi desenvolvido na Universidade de Skövde, Suécia (ANDLER et al., 1996, 1998). O sistema REACH (BUCHMANN et al., 1995; ZIMMERMANN; BUCHMANN, 1995), desenvolvido na Universidade Técnica de Darmstadt, Alemanha, é um banco de dados orientado a objetos ativo para sistemas com restrições de tempo-real suaves. O sistema RODAIN desenvolvido na Universidade de Helsinki, Finlândia, é um sistema de banco de dados em tempo-real para transações com prazos firmes adequado para aplicações de telecomunicações (LINDSTRöM; NIKLANDER; RAATIKAINEN, 2000). STRIP é um sistema de banco de dados em tempo-real suave distribuído desenvolvido na Universidade de Stanford, EUA (ADELBERG; KAO; GARCIA-MOLINA, 1996; ADELBERG; GARCIA-MOLINA; WIDOM, 1997).

Algoritmos de Escalonamento

Para os algoritmos de escalonamento em tempo-real as transações são definidas com prazos finais pré-definidos e não negociáveis, além de que o tempo de execução de pior caso e os padrões de chegada das transações devem ser conhecidos a *priori* para garantir que todas atendam seus prazos finais (CHENG, 2002; STEWART; BARR., 2002; STANKOVIC; RAMAMRITHAM; SPURI, 1998; GIRAULT et al., 2001).

Protocolos de Controle de Concorrência

Os protocolos de controle de concorrência mais utilizados para SGBD-TR são os Protocolos de Bloqueio de Duas Fases Pessimista (2PLP) (NYSTRöM et al., 2004b), para o qual as transações adquirem os bloqueios antes de executarem suas operações no banco de dados ou esperam pelo bloqueio se este não puder ser adquirido. Todavia, a aplicação desse protocolo pode gerar tempos indeterminados de espera pelo bloqueio. Dessa forma, a utilização desse mecanismo pode ser vantajosa por manter a consistência lógica do banco de dados, mas em contrapartida pode comprometer as restrições de tempo impostas às transações.

Outro protocolo utilizado é o Protocolo de Controle de Concorrência Otimista (CCO) (LINDSTRöM, 2002a, 2002b; RAATIKAINEN; LINDSTRöM, 2002), para o qual a resolução de conflito é atrasada até a transação estar num estado próximo de comprometer. Para este protocolo não existe a indeterminação do tempo de espera, porém a quantidade de transações que podem ser reiniciadas é grande, o que pode ser fatal em sistemas com restrições temporais.

Em DiPippo (1995) apresenta-se uma técnica de controle de concorrência semântica orientada a objetos, denominada *técnica de bloqueio semântico*. Com base nesta técnica é possível tanto garantir a consistência lógica e temporal dos dados e transações como definir critérios para a negociação entre elas. Com base nesta técnica também é possível expressar a imprecisão resultante dessa negociação utizando o conceito de bloqueio semântico para determinar quais transações podem invocar métodos de um objeto. O bloqueio semântico é controlado em cada objeto individualmente por uma *função de compatibilidade* (FC) que implementa mecanismos para controlas o acesso concorrente aos seus métodos dos objetos.

Linguagem de Consulta

Em PRICHARD (1995) *Real-Time SQL* (RTSQL) é uma linguagem de consulta definida no contexto de SGBD-TR. A RTSQL é baseada no modelo relacional de dados e estende o padrão SQL-92 para incluir extensões que especificam a consistência temporal dos dados

e das transações.

Em Leite et al. (2005) foi introduzida uma Linguagem de Consulta para Banco de Dados em Tempo-Real (LC-BDTR) que integra um SGBD Objeto-Relacional comercial ao pacote de especificação Java para tempo-real. Como resultado dessa integração, uma linguagem de consulta que permite o processamento de fluxo contínuo de dados baseada no padrão SQL-99 é obtida.

Gerenciamento de Qualidade de Serviços

Qualidade de serviços pode ser definida como o conjunto de características definidas para um sistema visando atingir uma determinada funcionalidade (AURRECOECHEA; CAMPBELL; HAUW, 1998). Dentre os mecanismos de QoS disponíveis estão as funções de QoS e as métricas de desempenho. O processamento de QoS em um sistema começa com o estabelecimento dos parâmetros exigidos pelo usuário, através das métricas. Esses parâmetros são mapeados e negociados entre os componentes do sistema, assegurando que todos podem atingir um nível de QoS aceitável. Recursos são então alocados e monitorados, havendo possibilidade de renegociação caso as condições do sistema se alterem.

O conceito de qualidade de serviços foi originalmente introduzido em redes de computadores para caracterizar principalmente o desempenho em transmissão de dados (FIROIU et al., 2002; LI; NAHRSTEDT, 1998; COCCOLI; BONDAVALLI; GIANDOMENICO, 2001). Com o sucesso da utilização do gerenciamento de QoS, pesquisas atuais visam integrar estes conceitos a outras áreas da computação. Nos bancos de dados multimídia onde os objetos multimídia são volumosos, manipulações e visualizações desses objetos podem requerer uma grande quantidade de tempo e de recursos. Através de QoS esses sistemas podem fornecer a informação pertinente com a qualidade requerida e acesso eficiente (CARDOSO, 2002; YE; KERHERVé; BOCHMANN, 1999; STAEHLI, 1996; GOEBEL et al., 1998; DONALDSON, 1994; SALAMATIAN; FDIDA, 2001; BORN; HALTEREN; KATH, 2000).

A utilização de mecanismos de QoS para o gerenciamento de dados e transações em tempo-real é motivada pela necessidade de se especificar a qualidade requerida em termos de quão impreciso pode ser o dado para ainda ser considerado válido e qual o limite de perdas de prazo das transações para o desempenho do sistema ainda ser aceitável. Isto se torna útil, principalmente devido à ineficiência dos testes de escalonabilidade em ambientes imprevisíveis. Esta ineficiência é justificada pela impossibilidade de prever as ações e os recursos necessários pelas transações, além de que valores imprecisos podem ser admitidos para os dados a fim de assegurar os prazos das transações. Em contrapartida, algumas transações podem ter que esperar que os dados se tornem válidos.

Redes de Petri

Uma rede de Petri é um grafo direcionado bipartido com um estado inicial, denominado marcação inicial (MURATA, 1989). O grafo direcionado consiste de dois tipos de nós, denominados *lugares* e *transições*. Os lugares são elementos essencialmente passivos nas redes de Petri, ou seja, lugares podem armazenar, mas não podem transformar informações. Em geral são utilizados para representar condições, recursos, informações, banco de dados, etc. As transições são os únicos elementos ativos em redes de Petri. Isso significa que transições podem transformar, mas nunca armazenar informações. São utilizadas para representar ações, atividades, transformações, eventos, etc. Os nós em uma rede de Petri são relacionados (conectados) por arcos rotulados com pesos (inteiros positivos). Um arco não pode relacionar componentes do mesmo tipo. Graficamente, lugares são representados por círculos e transições por retângulos. Um lugar p é entrada para uma transição t se existe um arco direcionado conectando o lugar à transição, nesse caso o lugar é um *lugar de entrada*. Um lugar p é saída para uma transição, se existe um arco direcionado conectando a transição ao lugar, nesse caso o lugar é um *lugar de saída*.

Sendo as redes de Petri uma linguagem com base matemática, modelos descritos com esta linguagem podem ser submetidos a procedimentos automáticos de análise. A análise de modelos de redes de Petri é principalmente baseada na construção do espaço de estados, por exemplo, o grafo de alcançabilidade. Um *grafo de alcançabilidade* representa o conjunto de estados alcançáveis e pode ser usado para verificar uma variedade de propriedades, por exemplo, se a rede é livre de impasse[3] (MURATA, 1989).

As redes de Petri têm se mostrado bastante úteis para a modelagem conceitual, análise, simulação e controle de sistemas complexos. Elas apresentam as seguintes características:

- São uma ferramenta com capacidade para modelagem hierárquica, com fundamentação matemática bem desenvolvida, que pode ser usada para análise.
- Diferentes propriedades de sistemas concorrentes, tais como conflito, concorrência, impasse, sincronismo, e exclusão mútua entre outras, podem ser verificadas;
- Têm uma representação gráfica que pode ser usada como documentação e que facilita a interação entre desenvolvedores e/ou usuários do sistema;
- Podem ser simuladas, de modo que não só a estrutura do sistema, mas também o comportamento dinâmico da especificação pode ser observado pelos projetistas e usuários do sistema com base nas sessões de simulação;

Devido às características acima mencionadas, a modelagem de sistemas através de redes de Petri suporta o desenvolvimento a nível conceitual, permitindo manter a coerência

[3]Tradução do termo *Deadline*.

com elementos do domínio do problema nos níveis mais abstratos. Sua faceta gráfica é extremamente importante quando a questão é prover mecanismos para a construção de sistemas reais e não apenas um modelo teórico sobre o qual se deseja apenas raciocinar matematicamente sobre um sistema. É principalmente essa faceta gráfica de redes de Petri que as torna atrativas como linguagem de especificação com a qual se pode interagir com os usuários de um sistema (GUERREIRO, 2002).

Redes de Sensores

As redes de sensores estão sendo utilizadas em várias aplicações, tais como: atividades de controle, ambientais, tráfego, segurança, medicina, militar e etc. O crescimento na demanda por estas redes é justificado pelo avanço tecnológico que os seus nodos têm sofrido nas últimas décadas. As redes de sensores podem conter diferentes tipos de nodos em sua estrutura, os considerados nesta pesquisa são os *sensores inteligentes*. Os sensores inteligentes, resumidamente chamados de sensores, são caracterizados por possuírem um ou mais sensores aplicados ao *chip* com capacidade de processamento de sinais e comunicação de dados (LOUREIRO et al., 2003).

Como características inerentes a estas redes estão às restrições em relação a:

- Comunicação: As redes conectando sensores provêem usualmente uma qualidade de serviço muito limitada, possuem latência com grande variância, largura de banda limitada e perda freqüente de pacotes.

- Consumo de Energia: Os sensores têm fornecimento limitado de energia e assim conservação de energia necessita ser uma das considerações principais do projeto do sistema.

- Processamento: Redes de sensores possuem poder de processamento e tamanho de memória limitados. Isso restringe os tipos de algoritmos de processamento de dados em um sensor e restringe o tamanho dos resultados intermediários que podem ser armazenados em um nó sensor.

- Incerteza em leituras de sensores: Sinais detectados em sensores físicos herdam incertezas e eles podem conter ruídos do ambiente. Funcionamentos ruins dos sensores e organizações de sensores inadequadas (tal como um sensor de temperatura próximo ao do ar condicionado) podem gerar dados imprecisos.

Uma opção para melhor utilizar estas redes, mesmo com estas restrições, é proposta em Bonnet et al. (1999), onde os sensores são capazes de armazenar e processar localmente, bem como transferir os dados produzidos por eles. Assim, uma parte do processamento

pode ser realizada no próprio nó da rede, ou em um grupo desses, reduzindo o consumo de energia e o tráfego de dados e, conseqüentemente aumentar o tempo de vida da rede. Esta abordagem é denominada de *abordagem distribuída*, onde os dados podem ser armazenados nos sensores e em um servidor de banco de dados. Três tipos de transações são definidos: (i) *consultas a dados históricos* que são consultas realizadas no servidor de banco de dados; (ii) *consultas instantâneas* que são consultas realizadas no sensor inteligente em um dado instante de tempo e; (iii) *consultas longas* que são consultas ao sensor inteligente durante um intervalo de tempo.

Arquitetura do Sistema

Os SGBD-TR podem ser aplicados em redes de sensores, a fim de garantir que dados atuais, coletados pelos nodos da rede, sejam utilizados por transações válidas temporalmente. Na Figura 1.1 é mostrada uma arquitetura simplificada do sistema modelado, considerando o domínio de aplicação de redes de sensores, que consiste de um sistema controlador e um sistema controlado.

O sistema controlador é composto pelo computador servidor, pelos sensores e pelas interfaces humanas, enquanto que o sistema controlado representa o ambiente a ser controlado, podendo ser uma fábrica, um armazém, cidades, poços de petróleo, etc. O computador servidor é o Banco de Dados *Warehousing* (BDW), onde os dados são armazenados formando um histórico da interação com o ambiente através dos dados obtidos pelos sensores. O BDW consiste de um componente de software com as funcionalidades do SGBD-TR e de um repositório de dados, na Figura representados por SGBD-TR e BDW, respectivamente. Cada sensor inteligente possui as funcionalidades de um sistema de banco de dados, sendo denominado de Banco de Dados Sensor (BDS) local, composto por um SGBD e por um BD.

O sistema controlador interage com o sistema controlado com base nos dados disponíveis sobre o ambiente, os quais são obtidos pelos vários sensores inteligentes. Os dados obtidos por esses sensores podem ser armazenados neles mesmos para posteriores consultas.

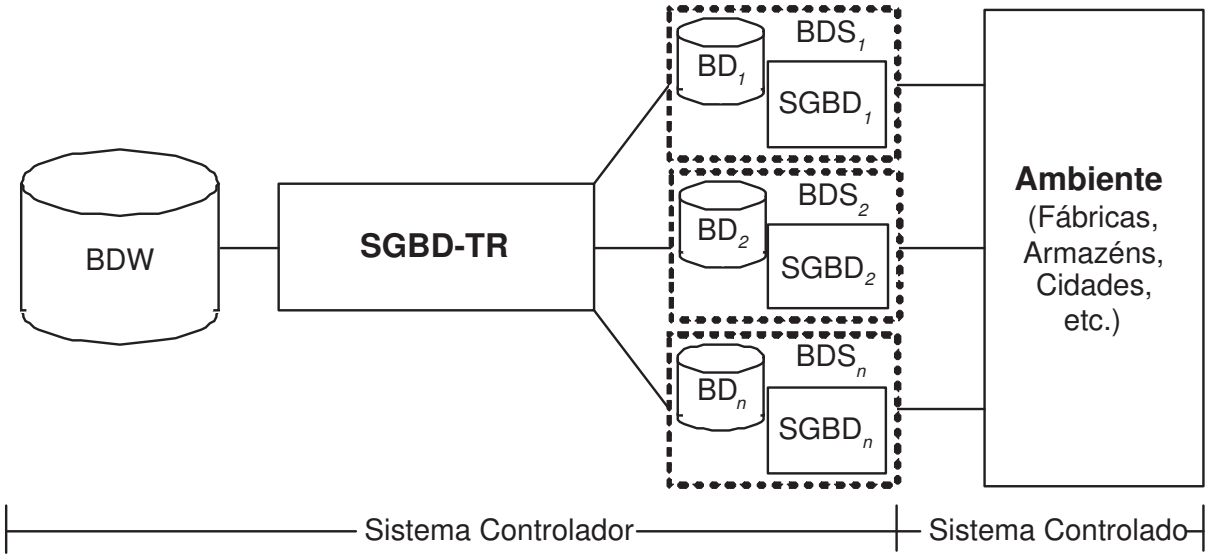

Figura 1.1: Arquitetura do Sistema no domínio de Redes de Sensores

1.2 Declaração do Problema e da Tese

Os algoritmos de escalonamento em tempo-real existentes não são eficientes para as aplicações que executam em ambientes imprevisíveis. Essa ineficiência é em decorrência da impossibilidade de definir as ações e os recursos necessários para que todas as transações atendam seus prazos. Também se deve considerar que nessas aplicações existem transações que podem perder seus prazos sem causar maiores prejuízos, em conseqüência de erros nas estimativas dos tempos de execução, por exemplo. Um outro fator relevante é o requisito de atender várias aplicações com necessidades distintas, exigindo que as políticas de escalonamentos se adaptem as necessidades da aplicação. Em suma, no contexto de aplicações que executam em ambientes imprevisíveis, os algoritmos de escalonamento deverão admitir transações com diferentes padrões de chegada (estrito, suave ou firme), ou seja, permitir o relaxamento das restrições de tempo-real quando necessário e assegurar níveis de qualidade especificados.

Como já mencionado, os protocolos de controle de concorrência baseiam-se no conceito de *serialização* para determinar quais transações podem executar concorrentemente. Em SGBD-TR, os aspectos temporais dos dados e as restrições de tempo das transações também devem ser considerados. Portanto, uma técnica de controle de concorrência para estes sistemas deve também manter a *consistência temporal dos dados e transações*. Dessa forma, a técnica deve permitir uma negociação entre consistência lógica e temporal, quando a manutenção de ambas não for possível.

O protocolo de controle de concorrência semântico é adequado para os SGBD-TR por permitir a negociação entre a consistência lógica e a consistência temporal (DIP-

IPPO, 1995). Este protocolo é baseado no critério de corretude Serialização *Epsilon* (SE) (RAMAMRITHAM; PU, 1995). SE é definido como um critério de corretude formal que especifica que a execução concorrente das transações é correta se o resultado dessa execução estiver dentro de limites de imprecisão[4] especificados. SE generaliza serialização por permitir imprecisão limitada no processamento de transações. A imprecisão é definida e controlada através da *Função de Compatibilidade*.

A Função de Compatibilidade é um componente de um objeto, definida pelo projetista do sistema, para expressar a compatibilidade entre a execução concorrente de dois métodos definidos para este objeto. Ela é avaliada considerando os valores dos parâmetros, onde alguns destes valores são definidos em tempo de execução e outros são definidos a *priori*, tal como o limite de imprecisão.

Em trabalhos recentes, a Função de Negociação (FN) QoS foi definida para expressar a execução concorrente das transações (RIBEIRO NETO; PERKUSICH; PERKUSICH, 2004). A função de negociação difere da função de compatibilidade por definir o limite de imprecisão através de uma métrica de desempenho. A vantagem disto é que os limites de imprecisão são obtidos em tempo de execução de acordo com as exigências da aplicação. Contudo, a limitação de avaliar apenas um par de métodos motivou uma investigação para estender FN, a fim de considerar n métodos executando concorrentemente para o mesmo objeto.

1.3 Objetivos da Tese

1.3.1 Objetivo Geral

O objetivo deste trabalho é disponibilizar um método para auxiliar o desenvolvimento de aplicações que executam em ambientes imprevisíveis e necessitam gerenciar dados e transações com restrições de tempo-real. Com base neste método, é possível análisar e simular um modelo para o sistema, buscando orientar o processo de tomada de decisão e propor soluções para a melhoria do mesmo. Mais especificamente, pretende-se assegurar a consitência e o desempenho dos SGBD-TR em ambientes não propícios de predizer quais os recursos e os prazos necessários para a completa execução das transações.

Mecanismos de QoS, através da definição de métricas de desempenho e da adequação de funções disponíveis, são utilizados para especificar, mapear, negociar e monitorar os requisitos de qualidade requeridos.

Para a validação do método, considera-se um estudo no contexto de aplicações em redes de sensores. As ferramentas matemáticas usadas para desenvolver o modelo são

[4]Por imprecisão entende-se a diferença entre o valor do item de dado no ambiente e como ele é refletido no banco de dados.

as Redes de Petri Coloridas Hierárquicas (HCPN) e através do pacote de ferramentas computacionais *Design/CPN* serão realizadas simulações desse.

1.3.2 Objetivos Específicos

Outras contribuições com a conclusão desta tese são:

- definição de métricas de desempenho adaptativas para garantir o desempenho desejado do sistema;
- utilização de funções de QoS para garantir a corretude do sistema mesmo em situações imprevisíveis;
- definir a função de negociação para avaliar n métodos concorrentes, onde a definição clássica permite avaliar apenas um par de métodos, e;
- disponibilizar um modelo, baseado no método introduzido, para redes de sensores considerando que os nós da rede, os sensores inteligentes, possuem as funcionalidades de um sistema de banco de dados permitindo armazenamento e consultas de dados.

1.4 Metodologia Utilizada

Revisão Bibliográfica

Inicialmente foi realizada uma revisão bibliográfica sobre sistemas de gerenciamento de banco de dados em tempo-real e sobre os mecanismos para o gerenciamento de qualidade de serviços. Essa revisão teve como objetivo estudar os principais conceitos destas áreas relacionados aos problemas abordados no trabalho.

Definição das Métricas de Desempenho e Funções de QoS

Nesta etapa, cinco funções de QoS foram definidas no contexto de SGBD-TR. Um modelo formal também foi desenvolvido considerando estas funções. A produção literária resultante desta etapa consta de um relatório técnico disponível na miniblio da COPELE (RIBEIRO NETO, 2002) que serviu de base para a publicação de três artigos completos em eventos (RIBEIRO NETO; PERKUSICH; M.L.B., 2003; RIBEIRO NETO; PERKUSICH; PERKUSICH, 2003a, 2003c).

Definição Formal de SE considerando QoS

Nesta etapa, a definição formal do critério de corretude serialização *Epsilon* foi realizada considerando mecanismos de qualidade de serviços. Mais especificamente, a função de

negociação QoS foi definida para expressar a execução concorrente das transações, onde o limite de imprecisão é obtido através de uma métrica de desempenho. Como produção literária resultante consta um relatório técnico disponível na miniblio da COPELE (RIBEIRO NETO, 2003b) e um artigo completo publicado em evento (RIBEIRO NETO; PERKUSICH; PERKUSICH, 2003b).

Estudo de Caso

Nesta fase, um estudo de caso foi contemplado considerando uma rede de sensores. A técnica de modelagem foi utilizada para descrever o que o sistema deve fazer e quais resultados ele deve gerar. Através do estudo de caso, uma descrição de como QoS pode ser eficiente para gerenciar dados e transações em tempo-real é realizada. Uma descrição detalhada da arquitetura do SGBD-TR em uma rede de sensores também é apresentada. Desta pesquisa resultou um relatório técnico disponível na miniblio da COPELE (RIBEIRO NETO, 2003a) e serviu de base para a publicação de seis artigos completos em eventos (RIBEIRO NETO; PERKUSICH; PERKUSICH, 2004; RIBEIRO NETO; PERKUSICH; M.L.B., 2004; RIBEIRO NETO; PERKUSICH; PERKUSICH, 2004b, 2004d, 2004c, 2004a).

Modelagem do Estudo de Caso

Em seguida, algumas abordagens para modelagem de processos concorrentes foram avaliadas, resultando em uma abordagem para modelagem formal, verificação e validação de SGBD-TR considerando mecanismos de QoS.

Verificação e Validação do Modelo

Por fim, uma abordagem formal foi definida para verificar e validar formalmente a utilização de métricas de desempenho e de funções de qualidade de serviços no gerenciamento de dados e transações com restrições temporais. Através da simulação do modelo resultante, diagramas foram gerados automaticamente para facilitar a validação por parte do usuário. A abordagem formal concebida nesta etapa resultou em um capítulo do livro Ribeiro Neto et al. (2005) que foi aceito para publicação.

1.5 Estrutura do Documento

Capítulo 2

Neste Capítulo, uma taxonomia é definida para os dados e transações nos sistemas de gerenciamento de banco de dados em tempo-real. Uma arquitetura do sistema considerando

esta taxonomia também é descrita. Por fim, uma visão geral do estado da arte é apresentada em relação a modelos, métodos de modelagem e projetos de gerenciadores de banco de dados em tempo-real.

Capítulo 3

Neste Capítulo, o critério de corretude serialização clássica é definido, em seguida as transações e a equivalência de escalonamentos são formalizadas para este critério. O critério de corretude serialização *Epsilon* também é definido, com a descrição das transações e a formalização das propriedades de segurança inerentes a serialização *Epsilon*. A técnica de controle de concorrência semântica também é descrita neste Capítulo. Por fim, um modelo de transações em tempo-real é mostrado, seguido da descrição da arquitetura do sistema, contemplando o que foi discutido.

Capítulo 4

Os mecanismos para o gerenciamento de qualidade de serviços são discutidos neste Capítulo. Inicialmente QoS é definida, para então seus conceitos serem contextualizados para os SGBD-TR. Nesta etapa, as funções e as métricas de desempenho são descritas. Uma especificação formal de serialização *Epsilon* considerando QoS é realizada. Por fim, uma arquitetura para o gerenciamento de transações em tempo-real é ilustrada e comentada, seguida da descrição da arquitetura do sistema, contemplando o que foi discutido.

Capítulo 5

As redes de Petri coloridas hierárquicas são utilizadas para desenvolver o modelo formal. Neste Capítulo são descritos os conceitos básicos e as extensões propostas para as redes de Petri. Também é apresentada o pacote de ferramentas *Design/CPN* utilizada para desenvolver o modelo formal, e realizar a verificação e validação deste.

Capítulo 6

Como domínio de aplicação para o modelo desenvolvido, as redes de sensores foram consideradas. Para facilitar o entendimento, esses sistemas são definidos e classificados em relação ao armazenamento dos dados.

Capítulo 7

O gerenciamento dos dados e o escalonamento das transações, ambos com restrições temporais, são investigados através de um modelo desenvolvido em redes de Petri colori-

das hierárquicas (HCPN). Também é ilustrado um esquema para orientar a modelagem. Com base nesse esquema e no método de modelagem de bancos de dados em tempo-real, definido em (PERKUSICH, 2000), são obtidos os modelos de objetos e de processos. Alguns componentes do modelo são descritos e ilustrados.

Capítulo 8

Neste Capítulo é definida uma abordagem de verificação e validação formal para bancos de dados em tempo-real, a fim de assegurar que o modelo, definido no Capítulo 7, está correto e que este atende as necessidades do usuário.

Capítulo 9

Neste Capítulo são apresentadas as contribuições esperadas com a conclusão desta tese. Um cronograma de pesquisa também é definido a fim de esclarecer quais serão os próximos passos.

Capítulo 2

Sistemas de Gerenciamento de Banco de Dados em Tempo-Real

2.1 Introdução

Os sistemas de gerenciamento de banco de dados em tempo-real devem satisfazer as restrições impostas a um banco de dados convencional, além de garantir as restrições de tempo-real impostas aos dados e transações. Essas restrições são especificadas para aplicações onde as transações devem satisfazer seus prazos finais, manipulando itens de dados que podem possuir validade temporal.

Os SGBD-TR podem ser utilizados com sucesso em aplicações com restrições de tempo que precisam armazenar, modificar e recuperar grandes volumes de dados compartilhados. Dentre estas aplicações estão comércio eletrônico, bolsa de valores *on-line* e várias aplicações de redes de sensores. Por exemplo, considere um ambiente monitorado por sensores, onde esses adquirem mudanças de estado ndo ambiente. Os valores adquiridos possuem validade temporal e as transações possuem prazos. No Algoritmo 1, uma transação de leitura deverá ler o item de dado capturado pelo *sensor1* em um prazo máximo de dez unidades de tempo. Todavia, esta transação só será executada se o item de dado estiver válido temporalmente.

Algoritmo 1 Exemplo de uma Transação acessando um Dado, ambos com Restrições Temporais

```
1: Inicie a Transação
2: Ler Item de Dado sensor1 no prazo de 10 unidades de tempo;
3: Verificar se item de dado sensor1 é válido;
4: Se Item de Dado for Velho Então
5:     Interrompa a Transação;
6:     Espere o Item de Dado ser Atualizado;
7:     Leia Item de Dado
8: Senão
9:     Leia Item de Dado
10: Fim Se
11: Commit
12: Fim da Transação
```

Um outro exemplo que possibilita veficar quais políticas de escalonamentos e protocolos de controle de concorrência com referências a tempo são indispensáveis para processamento de transações em tempo-real é ilustrado na Figura 2.1. Neste exemplo, considere a transação A com o tempo de liberação de uma unidade de tempo (u.t.), um prazo de vinte u.t. e um tempo computacional de dez u.t.. No instante de tempo um a transação começa e bloqueia o item de dado X no instante de tempo três. Uma outra transação B possui um tempo de liberação de três u.t., prazo de sete u.t. e tempo computacional de duas u.t.. No instante de tempo três a transação B tenta ler o item de dado X. Este item está bloqueado pela transação A, ocasionando a perda do prazo da transação B. Observe que ambas as transações poderiam atender seus prazos se a transação A no instante três fosse interrompida e a transação B, com prazo menor, executasse primeiro.

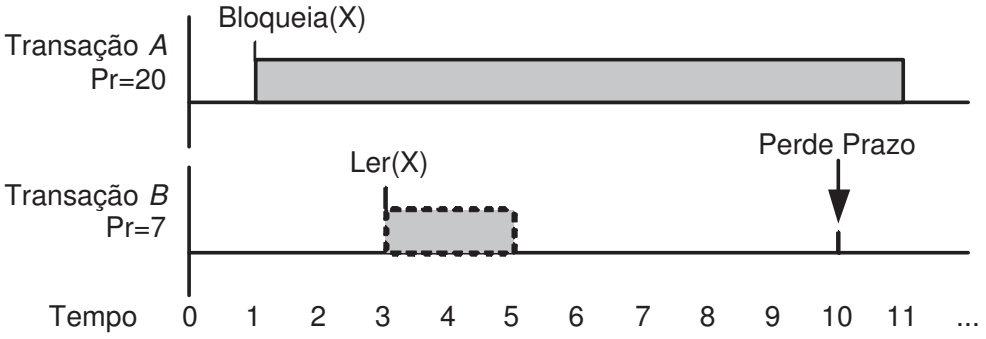

Figura 2.1: Transações Executando com Restrições de Tempo-Real

Neste Capítulo também é apresentada uma taxonomia dos SGBD-TR em relação aos dados e transações. Em seguida, uma arquitetura para um SGBD-TR é apresentada. Por

fim, o estado da arte dos SGBD-TR é descrito.

2.2 Taxonomia

Nesta Seção, os principais conceitos sobre os dados e transações que possuem restrições de tempo-real são definidos.

2.2.1 Dados com Restrições de Tempo-Real

A corretude dos dados em um SGBD-TR é garantida pela consistência lógica e temporal. Contudo, nem todos os dados possuem validade temporal, sendo possível classificá-los em dados sem restrições de tempo-real, ou simplesmente *dados atemporais* e dados com restrições de tempo-real ou *dados temporais* (KANG, 2001, 2003).

A corretude dos *dados atemporais* é garantida somente pela sua consistência lógica, uma vez que não possuem restrições temporais. Os *dados temporais* surgem da necessidade de refletir o estado do ambiente que está sendo controlado no banco de dados. Na categoria de *dados temporais* estão a temperatura corrente, a posição atual de um avião e o preço da ação na bolsa de valores. Cada *dado temporal* possui um rótulo de tempo associado, referente à sua última atualização. Os *dados temporais* podem ser categorizados em *dados base* e *dados derivados*. Esse último é derivado de vários dados base. Por exemplo, a posição corrente do avião, é um *dado derivado* da leitura de vários sensores, onde cada valor lido é um *dado base*.

Consistência Externa dos Dados com Restrições de Tempo-Real

A *consistência externa* dos *dados temporais* é definida para garantir a consistência entre o estado representado pelo conteúdo do banco de dados e o estado atual do ambiente (KANG, 2001, 2003). Intervalos de validade são utilizados para definir consistência externa e podem ser de dois tipos como descritos a seguir:

- *Intervalo de Validade Absoluta* é definido entre o estado do ambiente e o valor refletido no banco de dados. Um objeto de dado x é considerado temporalmente consistente se $(tempocorrente - rotulodetempo(x) \leq avi(x))$, onde *tempocorrente* é o tempo atual do sistema, *rotulodetempo* é o tempo da última atualização do dado e *avi* é o intervalo de validade absoluta do objeto x. Em outras palavras, *avi* significa por quanto tempo o dado é válido depois de escrito no banco de dados. Esta medida surge da necessidade de manter a visão do sistema consistente com o estado real do ambiente.

- *Intervalo de Validade Relativa* é definido entre os dados base usados para derivar outros dados. Considere um item de dado y derivado de um conjunto de dados R, onde $R = \{x_1, x_2, ..., x_k\}$. y é consistente temporalmente se os dados que o compõem, pertencentes a R, são válidos temporalmente e o $\mid rotulodetempo(x_i \in R) - rotulodetempo(x_j \in R) \mid \leq rvi(y)$, onde $rvi(y)$ é o intervalo de validade relativa de y. Esta medida surge da necessidade de produzir dados derivados a partir de dados gravados em tempos aproximados.

Representação dos Dados com Restrições de Tempo-Real

Os *dados temporais* são representados por $x : (valor, avi, rotulodetempo)$ e são consistentes temporalmente se os intervalos de validade absoluta e relativa forem satisfeitos. Considere o exemplo onde um item de dado t, com $avi(t)$ igual a cinco, reflete o valor da temperatura do ambiente e um outro item de dado p que representa a pressão possui $avi(p)$ igual a dez. O objeto de dado y derivado do conjunto de dados $R = \{t, p\}$ possui o intervalo de validade relativa $rvi(y)$ igual a dois. Se o tempo atual é igual a cinqüenta, então (a) $t : (25, 5, 45)$ e $p : (40, 10, 47)$ são consistentes temporalmente porque tanto o intervalo de validade absoluta como o relativo é válido. Este último sendo representado por $\mid rotulodetempo_t - rotulodetempo_p \mid \leq rvi(y)$. Porém, (b) $t : (25, 5, 45)$ e $p : (40, 10, 42)$ não são consistentes temporalmente, uma vez que somente o intervalo de validade absoluta é atendido.

2.2.2 Transações com Restrições de Tempo-Real

A transação é uma unidade de programa que acessa e, possivelmente, atualiza vários itens de dados. Assim, intuitivamente, uma transação executa uma ação sobre um objeto, gerando um novo estado no BD. A mudança de estado em um BD é para representar com fidedignidade as mudanças que ocorrem no mundo real. Porém, as entidades do mundo real possuem algumas restrições, tais como: a velocidade de um veículo não pode ser negativa e o preço de uma ação no mercado da bolsa de valores não pode ser inferior à zero. Desta forma, estas restrições devem ser atendidas a fim de manter a consistência do banco de dados. Como premissa básica para se manter a consistência do BD, em Özsu e Valduriez (2001), Silberschatz, Korth e Sudarshan (1999) tem-se:

Se o banco de dados era consistente antes da execução da transação, ele deve continuar consistente após a execução dessa, independente do fato de que a transação tenha sido executada de forma concorrente com outras e de que possa ter ocorrido falhas durante sua execução.

É importante observar que o banco de dados pode ficar temporariamente inconsistente

durante a execução de uma transação, como ilustrado na Figura 2.2. O importante é que o sistema fique consistente quando a transação terminar.

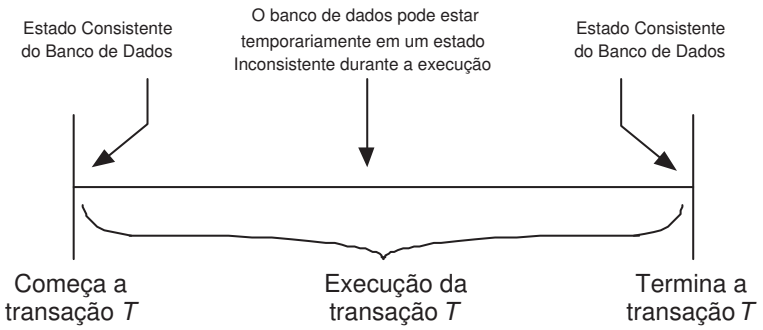

Figura 2.2: Representação de uma Execução da Transação

Propriedades ACID para Transações com Restrições de Tempo-Real

Para evitar que informações contraditórias sejam processadas por diferentes transações no mesmo instante de tempo a *consistência interna* do banco de dados deve ser garantida. Para isso, as propriedades ACID devem ser implementadas corretamente, a fim de que possam assegurar a integridade dos dados. 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. Para atender esta propriedade, transações concorrentes devem ser serilizáveis[1], isto é, a execução concorrente das transações deve produzir o mesmo resultado da execução serial das mesmas transações;
- *Durabilidade*: uma vez a transação é completada com sucesso, seus efeitos devem ser permanentes.

Contudo, as definições supracitadas não asseguram a *consistência externa* definida para os SGBD-TR. Como conseqüência, as propriedades ACID foram estendidas, onde:

- *Atomicidade*: se a transação for composta por subtransações, esta propriedade se aplica somente as subtransações;

[1]No Capítulo 3 o conceito de serialização será amplamente discutido.

- *Consistência*: o estado consistente de um banco de dados é ampliado por considerar que valores próximos dos reais estejam corretos. A diferença entre os valores reais e os próximos desse é denominada de *imprecisão*;

- *Isolamento*: a execução concorrente das transações é considerada correta se as transações atenderem seus prazos usando dados novos. Mesmo que para isso uma execução concorrente de transações não atenda o critério de corretude serialização;

- *Durabilidade*: o SGBD-TR deve refletir o estado do ambiente, sendo fácil recriá-lo a partir da leitura dos sensores, ao invés de recriá-lo no tempo em que ocorreu uma falha.

Os protocolos de controle de concorrência são implementados para manter a propriedade de isolamento. Protocolos de bloqueio de duas fases pessimista (2PLP) (NYSTRöM et al., 2004b) são bastante utilizados para controlar a execução concorrente das transações nos SGBD-TR. Várias extensões deste protocolo são propostas, onde o principal foco é evitar o problema de inversão de prioridade, tal como o Protocolo de Bloqueio de Duas Fases com Alta Prioridade (2PL-HP) (GRUENWALD; LIU, 1993). Em 2PL-HP uma transação com baixa prioridade é abortada e reiniciada em uma situação de conflito. Uma desvantagem deste protocolo é a espera pelo bloqueio por tempo indeterminado, se este não puder ser adquirido. Dessa forma, as restrições de tempo impostas às transações podem ser comprometidas. Em Lindström (2002b) é proposto um protocolo de controle de concorrência otimista onde a resolução de conflito é atrasada até a transação está próxima de terminar. O problema neste protocolo é a quantidade de transações que podem ser reiniciadas.

No Capítulo 3, uma discussão mais detalhada será apresentada sobre controle de concorrência e critérios de corretude. Também será mostrado como é definida a seqüência de execução correta das operações das transações. A principal motivação para se buscar outros critérios de corretude, que não seja o de serialização, é a necessidade de atender as restrições lógicas e temporais impostas aos dados e transações.

2.2.3 Classificação das Transações com Tempo-Real

As transações nos SGBD-TR podem ser categorizadas quanto às restrições de tempo-real, quanto aos padrões de chegada e quanto ao tipo de acesso aos dados. Esta classificação é muito importante para o gerenciamento dos dados e das transações, uma vez que a prioridade, periodicidade e conflitos podem ser definidos.

1. *Restrições de tempo-real*

As restrições de tempo-real das transações podem ser estritas, suaves e firmes. Um prazo estrito é atribuído às transações que devem atender seus prazos, caso contrário, podem provocar conseqüências catastróficas. Uma transação com prazo suave pode completar após o seu prazo. No entanto, não cumprir prazos suaves pode comprometer o desempenho do sistema. Já as transações com prazos firmes serão abortadas se as restrições temporais definidas não forem atendidas.

2. *Padrões de chegada das transações*

   Os padrões de chegada das transações se referem ao intervalo de tempo em que elas podem iniciar suas execuções. As transações podem ter padrões de chegada periódicos onde existe um intervalo de tempo regular entre duas execuções consecutivas da mesma transação. As transações também podem possuir padrões de chegada aperiódicos, onde não existe um intervalo de tempo regular entre suas execuções. Por fim, as transações podem ser esporádicas que são iniciadas em instantes de tempo aleatórios. No entanto, existe um intervalo mínimo entre duas execuções consecutivas da mesma transação.

3. *Tipo de acesso aos dados*

   Quanto ao tipo de acesso aos dados, as transações podem ser classificadas como: transações de escrita (ou sensores) e transações de leitura. As transações de escrita obtêm o estado do ambiente e escrevem os dados no banco de dados. Essas são tipicamente periódicas. As transações de leitura lêem dados do banco de dados, podendo ser periódicas ou aperiódicas.

2.3 Arquitetura do Sistema

Na Figura 2.3 é apresentada uma visão mais detalhada da Figura 1.1. Nesta arquitetura, os dados com suas restrições temporais, bem como as transações com suas restrições e categorizações são ilustrados. Assim, a compreensão sobre o mecanismo de gerenciamento de dados e transações com restrições de tempo-real pode ser facilitada.

O módulo SGBD-TR está de acordo com o descrito em (HOLANDA; BRAYNER; FIALHO, 2004) composto pelos seguintes componentes: gerenciador de transações, escalonador, gerenciador de recuperação e gerenciador de *cache*. As transações são recebidas e encaminhadas para o escalonador através do gerenciador de transações. No escalonador é feito o controle da execução concorrente das transações. O gerenciador de *cache* tem como finalidade manter o *cache*, movendo dados da unidade de armazenamento volátil para a estável e; o gerenciador de recuperação é responsável por garantir que o banco de

dados mantenha todos os efeitos de uma transação confirmada e nenhum dos efeitos de uma transação abortada (HOLANDA; BRAYNER; FIALHO, 2004).

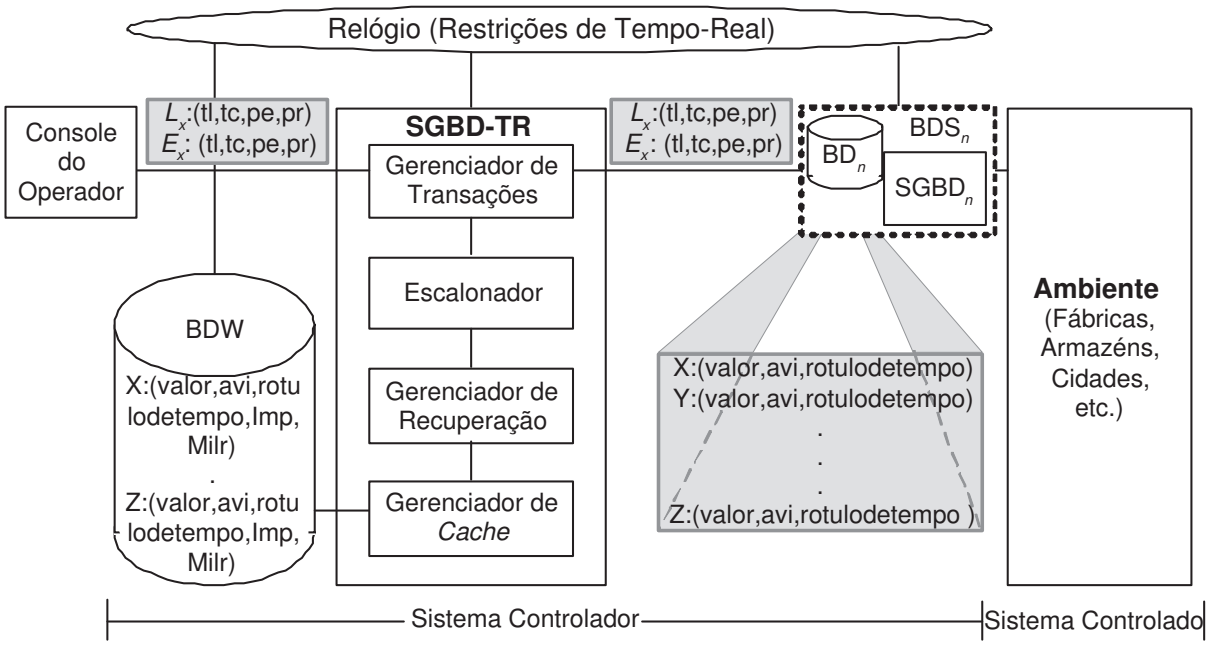

Figura 2.3: Arquitetura do Sistema com Dados e Transações com Restrições Temporais

As transações de escrita são representadas por E_x, onde uma transação vai escrever o item de dado X e as transações de leitura são representadas por L_x, onde uma transação vai ler o item de dado X. Para cada transação τ_i define-se os parâmetros temporais pela quádrupla (tl_i, tc_i, pr_i, pe_i), onde: 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; tc_i: representa o tempo computacional da transação, isto é, o tempo de processamento necessário para executá-la; pr_i: especifica o prazo máximo para execução da transação τ_i, e; pe_i: indica a periodicidade da transação.

Transações de escrita e de leitura são definidas para o BDW e para os BDS. No BDW, as transações de escrita podem ser iniciadas pelos operadores e/ou pelos sensores. As transações de leitura são executadas pelos operadores para consultar dados no servidor e nos sensores.

Os dados armazenados nos BDS possuem os atributos *valor*, *avi* e *rotulodetempo*, definidos na Seção 2.2.1. No BDW, os dados possuem mais dois atributos, além dos já citados, que são Imp e $Milr$. O primeiro especifica a imprecisão aceita no valor do item de dado para ele ainda ser considerado válido e o segundo refere-se ao limite máximo de imprecisão permitido.

2.4 Estado da Arte de Bancos de Dados em Tempo-Real

Nesta Seção será apresentada uma visão geral do estado da arte em relação às pesquisas direcionadas para o desenvolvimento de sistemas de gerenciamento de bancos de dados em tempo-real. O desenvolvimento destas pesquisas tem produzido modelos, métodos de modelagem e softwares com as propriedades de bancos de dados tradicionais e vários aspectos de consistência temporal.

2.4.1 Modelos de Bancos de Dados em Tempo-Real

Vários modelos têm sido propostos para expressar as características dos SGBD-TR. Um resumo do modelo de Ramamritham (RAMAMRITHAM, 1993) e do modelo RTSORAC (PECKHAM et al., 1996) serão apresentados nesta Seção.

Modelo de Ramamritham

Em Ramamritham (1993) é apresentado um modelo de banco de dados em tempo-real relacional, onde os dados podem apresentar restrições temporais absolutas e relativas. Neste modelo, os dados têm o seguinte formato: $d : (valor, avi, timestamp)$, onde d_{valor} denota o estado corrente de um dado d, d_{avi} denota o intervalo de validade absoluta do dado d e $d_{timestamp}$ denota o tempo em que a observação relativa ao dado d foi feita. A *consistência temporal absoluta* do dado d é mantida no tempo corrente t se $(t - d_{timestamp}) \leq d_{avi}$. As transações em tempo-real são classificadas de três formas: de acordo com seu uso por uma transação, pela natureza das restrições temporais e de acordo com o efeito de perder seu prazo. As definições apresentadas no início deste Capítulo para os dados e transações estão de acordo com este modelo.

Modelo RTSORAC

O modelo RTSORAC (*Real-Time Semantic Objects Relationships and Constraints*) (PECKHAM et al., 1996) incorpora características que suportam os requisitos de um banco de dados em tempo-real em um modelo orientado a objetos. Três componentes são utilizados para modelar as propriedades de um banco de dados orientado a objetos em tempo-real: *objetos, relacionamentos* e *métodos*. Os objetos representam as entidades do sistema, os relacionamentos representam as associações entre os objetos e definem restrições interobjetos dentro do banco de dados. Os métodos são as entidades executáveis que acessam os objetos e relacionamentos no banco de dados.

Um conjunto de restrições é definido que permite especificar corretamente o objeto. As restrições são usadas para expressar requisitos de consistência lógica e requisitos de consistência temporal.

Os modelos de Ramamritham e o modelo RTSORAC podem ser encontrados com mais detalhes em (RAMAMRITHAM, 1993) e (PECKHAM et al., 1996), respectivamente.

2.4.2 Método de Modelagem de Banco de Dados em Tempo-Real

Um método orientado a objetos para modelagem de bancos de dados em tempo-real é definido por Perkusich (2000). A aplicação do método resulta na obtenção de um *modelo de objetos* que captura os aspectos estáticos do sistema e um *modelo de processos* que captura os aspectos dinâmicos do sistema. O *modelo de processos* compreende 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.

O método é usado para modelar os três aspectos básicos de um sistema: dados, funções e dinâmica. A aplicação do método resulta na obtenção de um *modelo de objetos* e de um *modelo de processos*:

Modelo de Objetos: é usado para modelar as propriedades estáticas dos objetos, tais como: atributos, operações, restrições lógicas e restrições temporais,

Modelo de Processos: é usado para modelar as propriedades funcionais e dinâmicas dos objetos, isto é, modela as operações identificadas no modelo de objetos (métodos).

O modelo de objetos é representado através de um diagrama de classes que considera os aspectos de orientação a objetos (SILBERSCHATZ; KORTH; SUDARSHAN, 1999), tais como classes, agregação, associação e generalização. Associado a este diagrama estão às restrições temporais dos dados e uma função cuja finalidade é especificar a compatibilidade entre dois métodos de um objeto para permitir ou não a execução concorrente desses.

No modelo de processos, os objetos são descritos através de módulos HCPN que são definidos a partir do modelo de objetos. Então, para cada objeto que contém operações identificadas no modelo de objetos, é criado um módulo HCPN no qual serão modeladas as operações correspondentes.

A importância de um método para modelagem de SGBD-TR é devido a não compatibilidade dos métodos disponíveis para a descrição do modelo conceitual dos sistemas de bancos de dados convencionais, já que os mesmos não oferecem mecanismos para a representação das restrições temporais presentes nestes sistemas. Além disso, a maioria

dos modelos existentes concentra-se na representação das propriedades estáticas dos dados. No entanto, sistemas complexos, tais como bancos de dados em tempo-real também requerem a modelagem das propriedades dinâmicas dos dados.

2.4.3 Software de Gerenciamento de Bancos de Dados em Tempo-Real

Nesta Seção serão descritos alguns softwares para o gerenciamento de banco de dados em tempo-real resultantes de pesquisas realizadas em Instituições de Ensino Superior. O propósito desta descrição é para reforçar a utilização dos SGBD-TR, onde já existem várias tentativas iniciais, contudo ainda há muito que se pesquisar em relação a estes softwares. Cada sistema será sucintamente definido com suas principais características.

BeeHive

O sistema BeeHive (STANKOVIC; SON; LIEBEHERR, 1998), desenvolvido na Universidade da Virginia, EUA, é um banco de dados em tempo-real orientado a objetos.

Os conceitos de *prazo do dado*, *espera forçada* e *algoritmo de escalonamento baseado no prazo do dado* são definidos para expressar as restrições das transações. Prazo do dado é definido para atribuir um prazo mais estrito para uma transação, baseado no tempo de validade absoluta do item de dado, ou seja, o prazo do dado é atribuído ao prazo da transação se este for menor. O conceito de espera forçada é aplicado quando uma transação não puder ser completada antes do prazo do dado. Assim, o item de dado deve ser atualizado, atribuindo à transação o prazo do dado atualizado. Desta forma, define-se o algoritmo de escalonamento prazo do dado mais cedo (*earliest data-deadline first (EDDF)*) e o algoritmo de escalonamento prazo do dado mais tarde (*data deadline based least slack first (DDLSF)*).

Uma hierarquia de memória é utilizada visando alcançar uma maior previsibilidade em relação aos banco de dados de discos, uma vez que o comportamento desse pode ser comparado ao de um banco de dados de memória principal. A hierarquia de memória consiste de: memória principal, memória principal não volátil, memória em disco e armazenamento em fita. A memória principal é usada para armazenar dados que estão correntemente em uso pelo sistema. A memória principal não volátil é usada como um *cachê* de disco para dados que ainda não foram escritos no disco. A memória em disco é usada para armazenar o banco de dados, assim como qualquer banco de dados baseado em disco. Finalmente, o armazenamento em fita é usado para fazer cópias do sistema.

DeeDS

Um sistema de banco de dados em tempo-real, ativo e distribuído é proposto no projeto DeeDS (DeeDS - **[D]istributed activ[e], real-tim[e] [D]atabase [S]ystem**) (ANDLER et al., 1996, 1998) desenvolvido na Universidade de Skövde, Suécia. DeeDS utiliza regras ECA[2] estendidas para alcançar um comportamento ativo com propriedades de tempo-real em ambientes distribuídos.

O sistema DeeDS é um banco de dados de memória principal e consiste de duas partes, uma parte que trata com serviços de sistemas não críticos e outra que manipulam serviços críticos. Todos os serviços críticos são executados em um processador dedicado para diminuir o tempo de processamento desperdiçado com tarefas secundárias (*Overhead*).

Transações com padrões de chegada esporádicas (disparada por eventos) e periódicas (disparadas pelo tempo) são suportadas. As transações também podem ser classificadas como críticas, com prazos estritos e não críticas, com prazos suaves. Para assegurar que um prazo de uma transação crítica seja atendido, a monitoração de *milestone* e um esquema de contingência (*contingency plans*) são usados. Um *milestone* é definido como o prazo de uma parte da transação. Se um *milestone* for alcançado a transação não atenderá seu prazo, então um esquema de contingência deve ser ativado e a transação será abortada. O esquema de contingência deve negociar com as conseqüências resultantes do aborto da transação, ou seja, deve garantir as restrições de tempo da transação muito embora o resultado retornado possa ser impreciso.

O escalonamento das transações é feito *on-line* em dois passos. Primeiro, um escalonamento é produzido visando assegurar que todos os prazos estritos sejam atendidos, mesmo que alguns prazos suaves sejam perdidos. Segundo, o tempo de execução de pior caso das transações monitoradas é subtraído do tempo restante permitido pelo escalonamento durante este intervalo de tempo, e essa diferença é usada para otimizar o escalonamento das transações.

REACH

O sistema REACH (BUCHMANN et al., 1995; ZIMMERMANN; BUCHMANN, 1995), desenvolvido na Universidade Técnica de Darmstadt, Alemanha, é um banco de dados orientado a objetos ativo para sistemas com restrições de tempo-real suaves.

Para garantir à previsibilidade do sistema em relação às restrições de tempo-real, *milestones*, esquemas de contingência, uma ferramenta de marca de referência (*Benchmarking*) e uma ferramenta de rastreio são definidos.

Os *milestones* são usados para monitorar o progresso das transações. Se um *milestone*

[2]Acrônimo para Evento-Condição-Ação.

é perdido, a transação pode ser abortada e um esquema de contingência é fornecido. O esquema de contingência deverá ser implementado para manipular uma situação de perda de prazo ou por degradar o sistema de uma forma segura ou produzir um resultado suficientemente bom antes do fim do prazo original das transações. Pode-se dizer que um esquema de contingência em cooperação com o *milestone* trabalham como um alarme que é ativado quando um prazo esta próximo de for perdido.

A ferramenta de marca de referência pode ser usada para medir o tempo de execução de uma invocação de método ou disparo de eventos. A ferramenta de rastreio pode rastrear a ordem de execução no sistema. Se a ferramenta de marca de referência for usada junto com a ferramenta de rastreio, o comportamento do sistema e as restrições de tempo-real poderão ser determinados.

RODAIN

O sistema RODAIN é um sistema de banco de dados em tempo-real desenvolvido na Universidade de Helsinki, Finlândia, (LINDSTRöM; NIKLANDER; RAATIKAINEN, 2000).

A arquitetura do banco de dados RODAIN é distribuída e suporta particionamento de dados usando múltiplos nodos. As transações podem acessar dados no banco de dados distribuído inteiro, embora que possam somente atualizar objetos no nó de origem. Esta abordagem é adequada para muitas aplicações de telecomunicações, onde as atualizações podem ser direcionadas para um banco de dados local.

As transações com restrições de tempo-real suaves e firmes são suportadas. O conceito de disponibilidade (*Availability*) é definido para garantir o escalonamento das transaações com restrições temporais. Por disponibilidade entende-se a habilidade de um componente ou serviço em desempenhar a função requerida durante determinado período de tempo. É expresso geralmente como a *relação de disponibilidade* que significa a porção de tempo que o serviço está realmente disponível para ser usado pelos clientes dentro de um intervalo de tempo.

STRIP

O STRIP (STRIP - **[ST]anford [R]eal-time [I]nformation [P]rocessor**) é um sistema de banco de dados em tempo-real distribuído desenvolvido na Universidade de Stanford, EUA (ADELBERG; KAO; GARCIA-MOLINA, 1996; ADELBERG; GARCIA-MOLINA; WIDOM, 1997).

As transações são classificadas em *transações de baixo valor* e *transações de alto valor*. Assim, o escalonamento das transações é determinado pela classe a qual ela pertence, pela *densidade do valor* e pelo *algoritmo de escalonamento*. A densidade do valor é a taxa entre o valor da transação e o tempo de processamento restante.

Quatro algoritmos de escalonamento são definidos: primeiro atualizações (*Updates First (UF)*); primeiro transações (*Transactions First (TF)*); diferentes atualizações (*Split Updates (SU)*); e atualizações em demanda (*Apply Updates on Demand (OD)*). A distinção entre atualizações e transações do usuário é definida pelo tipo de operações que cada uma dessas pode conter. As atualizações são consideradas unicamente operações de escrita iniciadas por outro nó do banco de dados em um sistema distribuído. As transações podem conter operações de escrita e leitura iniciadas pelo usuário ou pela aplicação. As atualizações possuem as mesmas propriedades das transações em relação às classes e valor de densidade.

- A política de escalonamento UF escalona todas as atualizações antes das transações, indiferente do valor de densidade das transações. Esse algoritmo é aplicável se uma transação que está executando, e puder completar antes da atualização ser executada, for retomada pela atualização. Para um sistema com uma alta carga de atualizações esta política pode causar longos e imprevisíveis tempos de execução para as transações. No entanto, para sistemas onde a consistência lógica dos dados é mais importante que a consistência temporal das transações, esse pode ser um algoritmo adequado. Além disso, este algoritmo é apropriado para sistemas onde as atualizações são priorizadas em relação às transações. Considere um banco de dados executando transações para troca de estoque. Uma mudança no preço de um estoque deve ser realizada antes de qualquer transação pendente, a fim de obter o valor correto para a transação.

- No algoritmo de escalonamento TF as transações são sempre escalonadas antes das atualizações. Entretanto, uma transação não pode retomar uma atualização executando, isso porque atualizações, via de regra, possuem o tempo de execução menor que os das transações. Este algoritmo é adequado para sistemas onde a quantidade de dados enviados em um determinado período é grande, tais como em redes de sensores e sistema de controle industrial.

- O algoritmo SU considera diferentes classes de atualizações e escalona atualizações e transações de acordo com a seguinte ordem: atualizações de alto valor, transações e atualizações de baixo valor. Este algoritmo combina os dois algoritmos já descritos, sendo adequado em sistemas onde uma parte dos itens de dados deve ser mantida consistente logicamente, ao mesmo tempo em que algumas atualizações devem atender seus prazos finais.

- Atualizações em demanda executa transações antes das atualizações, com a diferença de que quando uma transação encontra dados antigos, a fila de atualizações é varrida

na procura de uma atualização que está esperando para atualizar este dado. Esta abordagem permitirá uma alta quantidade de transações executando com um alto grau de consistência temporal. Todavia, calcular o tempo de execução para uma transação é mais difícil do que para as atualizações, desde que o pior caso é que todos os elementos de dados usados nas transações têm atualizações pendentes.

Quadro Comparativo

Na Tabela 2.1 apresenta-se um resumo comparativo entre os SGBD-TR descritos nesta Seção.

SGBD-TR	Funcionalidade	Escalonamento	Restrição de Tempo-Real
BeeHive	*prazo do dado* e *espera forçada*	Baseado no prazo do dado	*suave*
DeeDs	*milestone* e *esquema de contingência*	Prioriza transações estritas	*estrita* e *suave*
REACH	*milestones*, *esquema de contingência*, *ferramenta de marca de referência* e *ferramenta de rastreio*	Baseado nas funcionalidades	*suave*
RODAIN	*disponibilidade*	Baseado na *relação de disponibilidade*	*suave* ou $firme$
STRIP	*classe da transação* e *densidade do valor*	Baseado nas funcionalidades	$firme$ ou *estrita*

Tabela 2.1: Resumo das Propriedades dos SGBD-TR

Capítulo 3

Critério de Corretude

Neste Capítulo é definido o critério de corretude serialização clássica e uma extensão desse, denominado serialização *Epsilon* (SE). Em seguida um comparativo entre ambos os critérios é feito. Também é apresentada uma técnica de controle de concorrência baseada na informação semântica da aplicação. Um modelo de transações em tempo-real é definido e finalmente é apresentada a arquitetura de um SGBD-TR no domínio de aplicação de redes de sensores considerando o que foi definido.

3.1 Serialização Clássica

Durante a execução das transações em um SGBD é possível: (i) *aumentar* a vazão do sistema por permitir que atividades de E/S e o uso da UCP sejam feitos em paralelo por várias transações, ocasionando em um maior número de transações executando em um determinado tempo; e (ii) *diminuir* o tempo médio de resposta para uma transação completar após ser submetida, por permitir que as transações curtas não precisem esperar que as transações longas terminem para serem iniciadas.

Para se conseguir as funcionalidades supracitadas, os sistemas de processamento de transações devem possibilitar que diversas transações sejam executadas de modo concorrente. A execução concorrente se dá pelo entrelaçamento das operações que compõem as transações. No entanto, quando múltiplas transações são executadas simultaneamente, a consistência do banco de dados pode ser destruída, mesmo que cada transação individual seja executada corretamente (WONG; AGRAWAL, 1992; YU et al., 1994; SILBERSCHATZ; KORTH; SUDARSHAN, 1999). Para assegurar a consistência, o sistema de BD deve controlar a interação entre as transações concorrentes. Este controle é feito através de protocolos de controle de concorrência.

Os protocolos de CC são implementados para garantir que cada transação, executando concorrentemente, atenda as propriedades ACID definidas no Capítulo 2. Assim, uma vez

a transação é executada essa deve ser atômica, ou seja, executada sem a interferência de outras transações; deve manter o BD consistente; deve executar de forma isolada; e ser persistente.

O princípio para garantir execuções entrelaçadas de operações das transações se fundamenta no conceito de *serialização* clássica, comumente chamada de *serialização* (ELMASRI; NAVATHE, 1994).

3.1.1 Definição

Para definir o conceito de serialização, será considerado um exemplo de um sistema bancário simplificado que consiste de várias contas e um conjunto de transações que acessam e atualizam estas contas. De acordo com a Figura 3.1, T1 e T2 são duas transações, onde T1 transfere fundos da conta X para a conta Y e a transação T2 realiza um depósito na conta X. Restrições em relação às transações, tal como, *o saldo não pode ser negativo*, serão abstraídos do exemplo por não serem necessários para a definição do conceito.

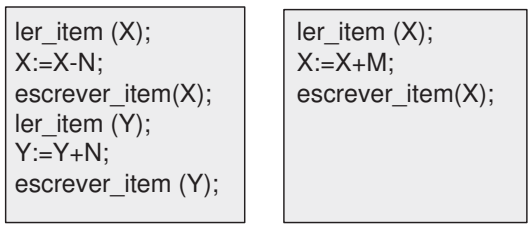

Figura 3.1: (a) Transação T1. (b) Transação T2.

Se a intercalação entre a execução das operações não for possível, haverá somente duas maneiras de ordenar as operações das duas transações:

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).

Estas alternativas são ilustradas nas Figuras 3.2(a) e 3.2(b), respectivamente. Se a intercalação das operações for permitida, então existirão muitos escalonamentos possíveis no qual o sistema poderá executar as operações das transações. Dois escalonamentos, dos possíveis, são mostrados nas Figuras 3.2(c) e 3.2(d).

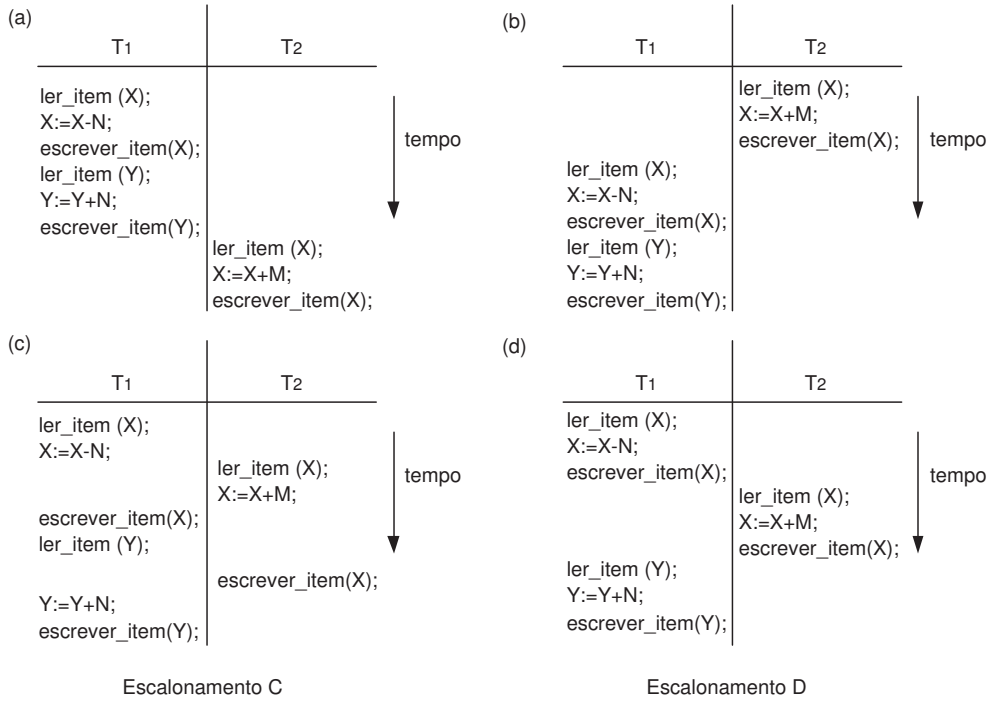

Figura 3.2: (a) Escalonamento A: T1 seguido por T2. (b) Escalonamento B: T2 seguido por T1. (c) e (d) Dois escalonamentos com operações intercaladas.

Assim, dois tipos de seqüências de execução das operações podem ser definidos: seqüência *serial* e seqüência *não serial*. Um escalonamento S possui uma seqüência *serial* se, para cada transação T participante do escalonamento, todas as operações de T são executadas consecutivamente. De outra maneira, o escalonamento possui uma seqüência *não serial* (ELMASRI; NAVATHE, 1994). Os escalonamentos A (Figura 3.2(a)) e B (Figura 3.2(b)) são *seriais* e os escalonamentos C (Figura 3.2(c)) e D (Figura 3.2(d)) são *não seriais*.

Porém, os escalonamentos *não seriais* podem gerar inconsistência se nenhum controle for exercido pelo sistema de gerenciamento de banco de dados. Para ilustrar, considere os escalonamentos na Figura 3.2 e assuma que os valores iniciais 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, os valores no banco de dados devem ser $X = 89$ e $Y = 93$. O resultado correto para os itens de dados sempre é garantido nos escalonamentos *seriais* A e B, o que não acontece nos escalonamentos *não seriais* C e D. Após a execução das transações no escalonamento C, os valores dos itens de dados são $X = 92$ e $Y = 93$. O valor de X está errado, porque a transação T2 lê o valor de X antes de esse 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.

Para evitar inconsistências, o critério de corretude serialização disponibiliza mecanismos para identificar os escalonamentos *não seriais* que são corretos. Assim, um escalona-

mento S de n transações é serializável se for equivalente a um escalonamento *serial* das mesmas n transações. 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 serializável; e os que não são equivalentes a qualquer escalonamento *serial*, não sendo serializável.

Nas próximas Seções os conceitos de transações, escalonamentos e equivalência de escalonamentos são formalmente definidos.

3.1.2 Transações-Clássica de Leitura e Escrita

Uma transação é uma abstração que representa uma seqüência de operações de leitura e escrita sobre o banco de dados resultante da execução de um programa aplicativo (BRAYNER, 1999). A notação $r_i(x)$ representa uma operação de leitura executada pela transação T_i sobre um objeto x. Analogamente, a notação $w_i(x)$ representa uma operação de escrita executada pela transação T_i sobre um objeto x.

Em Monteiro Filho (2001) uma transação é definida formalmente como uma relação de ordem $<_i$.

1. $OP(T_i) \subseteq \{r_i(x), w_i(x) \mid x \text{ é um item de dado}\} \cup \{a_i, c_i\}$
2. $a_i \in OP(T_i)$ se e somente se $c_i \notin OP(T_i)$
3. Seja $t \in \{a_i, c_i\}$, para qualquer operação $p \in OP(T_i), p <_i t$ e
4. Se $r_i(x), w_i(x) \in OP(T_i)$, então $r_i(x) <_i w_i(x)$ ou $w_i(x) <_i r_i(x)$

onde $OP(T_i)$ representa o conjunto de todas as operações executadas pela transação T_i. Após as operações de leitura e/ou escrita de uma transação uma operação de comprometimento (*commit*) (c_i) ou de aborto (a_i) deverá aparecer. A relação de ordem $<_i$ representa a precedência entre duas operações de uma mesma transação.

3.1.3 Equivalência de Escalonamentos

Um escalonamento de transações especifica a ordem em que às operações de diferentes transações podem ser executadas. Como já mencionado, um escalonamento S de n transações é correto se for equivalente a um escalonamento *serial* das mesmas n transações.

A equivalência entre escalonamentos pode ser obtida de três formas (SILBERSCHATZ; KORTH; SUDARSHAN, 1999; BRAYNER, 1999): equivalência por estado final, equivalência por visão e equivalência por conflito, onde:

- Dois escalonamentos executados com o mesmo conjunto de transações ST são ditos equivalentes por estado final, se e somente se:

  1. possuem as mesmas operações pertencentes às transações em ST; e
  2. produzem o mesmo estado do BD, se eles são executados a partir do mesmo estado inicial.

- Dois escalonamentos executados com o mesmo conjunto de transações ST são ditos equivalentes por visão, se e somente se:

  1. possuem as mesmas operações pertencentes às transações em ST;
  2. $\forall r_i(x) \in T_i$, onde $T_i \in ST$, o valor lido por r_i deve ser o mesmo em ambos escalonamentos; e
  3. se $w_i(x)$ é a última operação de escrita sobre um objeto x em um escalonamento, então $w_i(x)$ é também a última operação de escrita no escalonamento equivalente.

- Dois escalonamentos executados com o mesmo conjunto de transações ST são ditos equivalentes por conflito, se e somente se:

  1. possuem as mesmas operações pertencentes às transações em ST;
  2. $\forall p_i \in OP(T_i)$ e $\forall q_j \in OP(T_j)$, com $i \neq j$. Se p_i precede q_j, então p_i precede q_j no escalonamento equivalente.

A noção de equivalência por estado final é baseada nos conceitos de estado inicial e estado final de um BD, os quais não são capturados pelo conceito de escalonamento. Este problema pode ser resolvido por utilizar equivalência por visão. Contudo, o problema de verificar se um escalonamento é equivalente por visão não pode ser solucionado em tempo polinomial, isto é, ente é um problema *NP*-Completo. Já a equivalência por conflito pode ser verificada em tempo polinomial, isto é, existe um método eficiente para verificar se um escalonamento é equivalente por conflito. Por tal motivo, a maioria dos protocolos de CC implementa o critério de corretude serialização baseado em equivalência por conflitos (BRAYNER, 1999).

Em equivalência por conflitos define-se o conceito de conflitos de operações, onde: duas operações quaisquer de transações diferentes estão em conflito, se elas acessam o mesmo item de dados e pelo menos uma das operações é de escrita.

Um teste para saber se um escalonamento é serializável é obtido através da construção do grafo de precedência para o escalonamento. Os nós do grafo de precedência correspondem a transações e existe um arco $T \rightarrow U$, se alguma operação de T no escalonamento

entrar em conflito com uma operação posterior de U. Um escalonamento é serializável se e somente se o grafo de precedência é acíclico.

Contudo, serialização é bastante restritivo para aplicações de tempo-real, já que só consideram aspectos de consistência lógica de dados e transações desconsiderando qualquer aspecto temporal para essas.

3.2 Serialização *Epsilon*

3.2.1 Definição

O critério de corretude denominado serialização *Epsilon* SE, proposto em (RAMAMRITHAM; PU, 1995), é uma generalização de serialização clássica, sendo definido como um critério de corretude formal que especifica que um escalonamento para transações é correto se o resultado do escalonamento está dentro de limites especificados. SE generaliza serialização por permitir uma imprecisão limitada resultante da execução concorrente de transações.

O conceito de imprecisão limitada é introduzido a fim de permitir que os valores dos itens de dados do mundo real não, necessariamente, sejam idênticos em relação aos seus respectivos valores representados no BD. Contudo, a diferença existente entre o valor do mundo real e o representado no BD deve ser limitada. Dessa forma, transações conflitantes podem executar concorrentemente, ou seja, uma operação de leitura pertencente à transação T_i pode acessar um item de dado que está sendo atualizado por uma operação de escrita pertencente à transação T_j.

Considerando a imprecisão limitada, um BD é consistente, se:

$$\forall x \wedge y, \text{ o BD é consistente se } x \in [x - y, x + y]$$

onde x é o item de dado e y é a imprecisão permitida para x.

A grande motivação para se permitir uma imprecisão no valor do item de dado é a de atender as restrições de tempo-real impostas aos dados e transações nos SGBD-TR. Uma vez que, em SGBD-TR é mais importante obter o valor aproximado da posição corrente de um avião em um determinado instante de tempo que o valor exato desta posição tarde demais. SE já foi utilizado com êxito no contexto de bancos de dados em tempo-real, como descrito em (RIBEIRO NETO, 2001; RIBEIRO NETO; PERKUSICH; PERKUSICH, 2004).

3.2.2 Transações-*Epsilon* de Leitura e Escrita

A noção de corretude em SE é baseada no limite da quantidade de imprecisão importada pela transação-*Epsilon* de leitura e no limite de imprecisão exportada pela transação-

Epsilon de escrita (RAMAMRITHAM; PU, 1995). A notação $r\epsilon_i(x)$ representa uma operação de leitura executada pela transação-*Epsilon* ($T\epsilon_i$) sobre um objeto x. Analogamente, a notação $w\epsilon_i(x)$ representa uma operação de escrita executada pela $T\epsilon_i$ sobre um objeto x.

Conceitualmente, uma transação-*Epsilon* é uma transação que suporta inconsistência, ou seja, especifica a quantidade de imprecisão que pode ser importada ou exportada em um item de dado. Uma transação-*Epsilon* de leitura possui um limite-importado que especifica a quantidade máxima de imprecisão que pode ser importada. Similarmente, uma transação-*Epsilon* de escrita tem um limite-exportado que especifica a quantidade máxima de imprecisão que pode ser exportada.

O limite de imprecisão definido para cada item de dado é especificado e o sistema deve assegurar que esses limites não sejam excedidos durante a execução concorrente das transações-*Epsilon*. Como exemplo, assuma que o gerente do banco financeiro quer saber o valor total referente a todas as contas correntes de uma determinada agência bancária. Considerando a definição de conflitos, esta consulta não poderia ser executada durante o horário bancário, uma vez que várias operações bancárias estão sendo executadas. No entanto, assumindo que várias contas correntes possuem pequenos valores e que algumas operações não alterarão o resultado final, por se tratarem de pequenas quantias se considerado o saldo total da agência, a consulta poderá ser realizada. Em outras palavras, se for considerado o limite-importado y para a $T\epsilon$ de leitura, para essa consulta o resultado é correto se estiver dentro deste limite tanto para mais quanto para menos, ou seja, se o valor preciso da consulta for de z então ele estará correto se estiver no intervalo $\mathsf{z} \pm \mathsf{y}$.

3.2.3 Propriedade de Segurança

Durante a execução concorrente das $T\epsilon$, o sistema necessita controlar a quantidade de imprecisão resultante. A quantidade máxima de imprecisão que uma $T\epsilon$ de leitura pode importar para um item de dado x é definida como $lim\_imp_{T\epsilon,x}$ e a quantidade máxima de imprecisão que uma $T\epsilon$ de escrita pode exportar para um item x é definida como $lim\_exp_{T\epsilon,x}$. Para cada item de dado 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 pela $T\epsilon$ de leitura para um item de dado x é definida por $imprec\_imp_{T\epsilon,x}$ e a quantidade de imprecisão exportada pela $T\epsilon$ de escrita para um item de dado x é definida por $imprec\_exp_{T\epsilon,x}$. A quantidade de imprecisão escrita em um item de dado x é definida como $quant\_imp_x$.

Estes limites de imprecisão são garantidos através das propriedades de segurança que são definidas para as transações e para os dados. Um item de dado x é *seguro*, se e

somente se:

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

onde a imprecisão no item de dado x é aceitável, se ela não for maior que o limite de imprecisão especificado para o item de dado.

Dada uma $T\epsilon$ e um item de dado x, as seguintes propriedades definem uma transação *segura*:

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

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

onde a imprecisão no item de dado lido (ou escrito) por uma $T\epsilon$ de leitura (ou escrita) é aceitável, se ela não for maior que o limite de imprecisão especificado.

Assim, serialização *Epsilon* é garantido se e somente se todos os dados e transações são seguros.

3.3 Serialização *Epsilon versus* Serialização Clássica

As propriedades de serialização *Epsilon* e as propriedades de serialização clássica podem ser comparadas de acordo com as definições:

- Quando todos os limites-importados e limites-exportados são iguais à zero, serialização *Epsilon* é igual à serialização clássica;
- Um conjunto de transações pode não satisfazer a serialização clássica, mas podem satisfazer serialização *Epsilon*; e
- Quando os limites-importados e limites-exportados são maiores que zero, isto é, serialização *Epsilon* $\supseteq$ serialização clássica. Significa que serialização *Epsilon* permite mais escalonamentos de transações que serialização.

3.4 Técnica de Controle de Concorrência Semântico

Em DiPippo (1995) 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. Através da técnica também é possível expressar a imprecisão resultante desta negociação. Esta 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.

Esta 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 por garantir os intervalos de validade absoluta e relativa, definidos no Capítulo 2 e suporta consistência temporal das transações por considerar a especificação de escalonamentos permitidos por serialização *Epsilon*, definido na Seção anterior.

Nos últimos anos, muitas outras técnicas para controle de concorrência semântico foram propostas, algumas delas podem ser vistas em (WONG; AGRAWAL, 1992; YU et al., 1994). 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 (DIPIPPO, 1995).

Nas Seções seguintes, os objetos, o critério de corretude serialização *Epsilon* orientada a objetos e as funções de compatibilidade serão definidos formalmente.

3.4.1 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 definidos como 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 destes predicados declaram-se as restrições lógicas e temporais, bem como os limites de imprecisão para os atributos.

5. FC 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.

3.4.2 Serialização *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*.

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.

3.4.3 Função de Compatibilidade

No modelo RTSORAC (PECKHAM et al., 1996) o conceito de *função de compatibilidade* (FC) é introduzido para expressar a compatibilidade entre duas transações executando concorrentemente, além de definir e controlar a imprecisão permitida para os dados e transações pertencentes à esta execução.

FC foi adotada na técnica de bloqueio semântico, onde é um componente de um objeto do BD, definido pelo projetista do sistema. Ela é avaliada em tempo de execução e definida sobre cada par ordenado de métodos de um objeto.

Uma FC tem a forma:

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

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.

Em adição, além de expressar a compatibilidade entre a invocação de dois métodos, a função de compatibilidade expressa informações sobre a imprecisão máxima acumulada que pode ser introduzida com a execução concorrente dos métodos.

Considerando isso, FC passa a ter a seguinte forma:

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

onde IA representa a imprecisão acumulada.

A FC pode expressar para as invocações dos métodos m_{ati} e m_{inv} (DIPIPPO, 1995), 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}.

O conceito de *conjunto afetado* foi introduzido em (BRADINATH; RAMAMRITHMAN, 1988) 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.

Para ilustrar essas situações, as funções de compatibilidade serão definidas.

Função de Compatibilidade(Leitura,Escrita)

A função de compatibilidade 3.1 expressa a negociação entre uma $T\epsilon_i$ de leitura ($r\epsilon_i(x)$) que está sendo executada e uma $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) que é invocada para executar sobre o mesmo item de dado x. Considerando a técnica de bloqueio (SILBERSCHATZ; KORTH; SUDARSHAN, 1999), estas operações são conflitantes, portanto não poderiam executar concorrentemente, uma vez que a operação invocada necessitaria de um bloqueio exclusivo.

No entanto, na Seção 3.4 é definida a técnica de bloqueio semântico onde o bloqueio semântico é controlado em cada objeto individualmente por uma FC que controla o acesso concorrente aos seus métodos. Dessa forma, os parâmetros da transação invocada, da transação executando e do sistema são passados para a FC, a fim de evitar que a validade temporal de x e da $T\epsilon_j$ de escrita seja violada.

FC é avaliada em verdadeira se o valor novo do item de dado x escrito por $w\epsilon_j(x)$ ($x_{novo}.val$) for próximo do valor corrente do item de dado x ($x.val$) armazenado no BD. Essa determinação é baseada em $x.val$, no limite máximo de imprecisão permitido por $r\epsilon_i(x)$ para x ($lim\_imp_{T\epsilon,x}$), na quantidade de imprecisão que $w\epsilon_j(x)$ irá escrever em x através de $x_{novo}.val$ ($x_{novo}.quant\_imp$) e na quantidade de imprecisão existente em x ($x.quant\_imp$).

A imprecisão acumulada, resultante da execução concorrente das duas transações, também é mostrada. Neste caso, x em $r\epsilon_i(x)$ tem um aumento de imprecisão igual à diferença entre o valor do item de dado x antes da atualização ($x.val$) e o novo valor do atributo após a atualização ($X_{novo}.val$), mais a quantidade de imprecisão que é escrita em x por $w\epsilon_j(x)$ ($x_{novo}.quant\_imp$).

$$\begin{aligned} FC_1(r\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\ &\wedge (|x.val - x_{novo}.val| \leq (lim\_imp_{T\epsilon,x} \\ &- (x.quant\_imp + x_{novo}.quant\_imp))) \\ &\Rightarrow x.quant\_imp = x.quant\_imp \\ &+ x_{novo}.quant\_imp + |x.val - x_{novo}.val| \end{aligned} \tag{3.1}$$

onde $x.tempo$ é o rótulo de tempo do item de dado x, ou seja, o tempo em que este valor foi gravado e $x.avi$ é o tempo de validade absoluta do item de dado x.

Função de Compatibilidade(Escrita,Leitura)

Na função de compatibilidade 3.2, a $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) está atualizando o valor do item de dado x com o valor x_{novo} e a $T\epsilon_i$ de leitura ($r\epsilon_i(x)$) é invocada para ler o valor do item de dado x. Neste caso, a transação invocada requer um bloqueio compartilhado, o

que não é possível devido ao bloqueio exclusivo que o item de dado possui (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Para avaliar a FC neste exemplo, a diferença entre o novo valor do dado ($x_{novo}.val$) e o valor atual ($x.val$) é considerada e deve está dentro do limite de imprecisão aceito pela $T\epsilon_i$ de leitura ($lim\_imp_{T\epsilon,x}$). Essa determinação é baseada no valor atualizado por $w\epsilon_j(x)$ ($x_{novo}.val$), no limite de imprecisão permitido por x (lim_imp_x) e na quantidade de imprecisão acumulada em x ($x.quant\_imp$). A imprecisão acumulada em x tem um aumento igual à diferença entre o valor atual ($x.val$) e o novo valor ($x_{novo}.val$), caso as transações sejam executadas concorrentemente.

$$\begin{aligned} FC_2(w\epsilon_j(x), r\epsilon_i(x)) = (&agora - x.tempo \leq x.avi) \\ &\wedge (|x.val - x_{novo}.val| \leq \\ &(lim\_imp_{T\epsilon,x} - x.quant\_imp) \\ &\Rightarrow x.quant\_imp = \\ &x.quant\_imp + |x.val - x_{novo}.val| \end{aligned} \tag{3.2}$$

onde $x.tempo$ é o rótulo de tempo do item de dado x, ou seja, o tempo em que este valor foi gravado e $x.avi$ é o tempo de validade absoluta do item de dado x.

Função de Compatibilidade(Escrita,Escrita)

Através da função de compatibilidade 3.3 é mostrada a situação onde um bloqueio exclusivo é concedido a uma $T\epsilon_i$ de escrita ($w\epsilon_i(x)$) e uma outra $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) solicita este mesmo bloqueio. De acordo com a técnica de bloqueio semântico, estas transações podem ser executadas concorrentemente desde que a diferença entre o valor atual ($x.val$) e o novo valor ($x_{novo}.val$) não excedam a quantidade de imprecisão permitida para x. A determinação de tal imprecisão é baseada nos valores escritos pela $T\epsilon_i$ que está executando ($x_1.val$) e pela $T\epsilon_j$ que é invocada ($x_2.val$), na imprecisão existente em x ($x.quant\_imp$) e no limite de imprecisão permitido (lim_max_x).

$$\begin{aligned} FC_3(w\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\ &\wedge (|x.val - x_{novo,k}.val| \leq \\ &(lim\_max_{T\epsilon,x} - x.quant\_imp) \\ &\Rightarrow x.quant\_imp = \\ &x.quant\_imp + |x.val - x_{novo,k}.val| \end{aligned} \tag{3.3}$$

onde $x.tempo$ é o rótulo de tempo do item de dado x, ou seja, quando esse valor foi gravado

e $x.avi$ é o tempo de validade absoluta do item de dado x. Em $x_{novo,k}$, k identifica o item de dado em relação a transação de escrita.

3.5 Modelo de Transações em Tempo-Real

Um modelo para o processamento de transações em tempo-real, que podem acessar dados com validade temporal, será definido nesta Seção. Para isso, a arquitetura proposta para SGBD-TR, definida no Capítulo 1, será considerada. Desta forma, o sistema é composto por um servidor de banco de dados *warehousing* onde os dados são armazenados formando um histórico da interação com o ambiente através dos dados obtidos pelos sensores. O servidor pode ser consultado e/ou alterado por programas aplicativos e alterado por transações enviadas a partir dos sensores inteligentes. Os sensores inteligentes também fazem parte do sistema, onde cada um possui funcionalidades de um sistema de banco de dados, sendo denominado de Banco de Dados Sensor local. Os dados nos sensores podem ser consultados por programas aplicativos.

Em seguida, uma classificação das transações no modelo é apresentada:

1. Transações Sensores

   - *Transação de Aquisição Sensores* (TAS): é a ação de sensoriamento, ou seja, transações que representam uma nova aquisição do item de dado que será armazenado no banco de dados sensor. Estas transações são de escrita, esporádicas e firmes;
   - *Transação de Leitura Sensores* (TLS): são as transações enviadas pelos programas aplicativos para consultar os bancos de dados locais. Tais transações são aperiódicas e estritas com operações de leitura.

2. Transações do Servidor

   - *Transação de Escrita Sensores* (TES): transações enviadas pelos sensores para atualizar o servidor. São transações periódicas, suaves ou estritas, contendo somente operações de escrita;
   - *Transação de Escrita Servidor* (TEW): são transações enviadas pelos programas aplicativos com operações de escrita para o servidor. Essas são esporádicas, podendo possuir prazo suave ou firme.
   - *Transação de Leitura Servidor* (TLW): transações enviadas pelos programas aplicativos para consultar o servidor, contendo operações de leitura, padrões de chegada esporádicos com prazo firme e/ou estrito.

Transações	*Propriedades das Transações*		
	Tipo de Acesso	Padrões de Chegada	Restrição de Tempo-Real
TAS	*escrita*	*esporadica*	*firme*
TLS	*leitura*	*aperiodica*	*estrita*
TES	*escrita*	*periodica*	*suave* ou *estrita*
TEW	*escrita*	*esporadica*	*suave* ou *firme*
TLW	*leitura*	*esporadica*	*firme* ou *estrita*

Tabela 3.1: Transações do Modelo

Definição 1 *Um sistema de banco de dados em tempo-real no domínio de aplicação de redes de sensores consiste de:*

1. *Um conjunto* $BDS = \{BDS_1, BDS_2, BDS_3, ..., BDS_m\}$ *de sensores inteligentes.*
2. *Um BDW localizado em um computador fixo.*
3. *Um conjunto* $S = \{S_1, S_2, S_3, ..., S_n\}$ *de transações sensores, onde cada* S_i *representa o conjunto de operações executadas nos BDS.*
4. *Um conjunto* $W = \{W_1, W_2, W_3, ..., S_n\}$ *de transações do servidor, onde cada* W_i *representa o conjunto de operações executadas no BDW.*

Definição 2 *Um escalonador EW modela o entrelaçamento das operações pertencentes às transações sensores e transações do servidor. Dessa forma, o escalonador EW é definido sobre o conjunto de transações* $T_i = S_i \cup W_i$*. Nesse caso, são definidas três funções de compatbilidade:*

1. FC_1 *para avaliar a compatibilidade quando uma transação de leitura do servidor* (TLW) *está executando e uma transação sensor* (TES) *é invocada para executar.*
2. FC_2 *para avaliar a compatibilidade quando uma transação sensor* (TES) *está executando e uma transação de leitura do servidor* (TLW) *é invocada para executar.*
3. FC_3 *para avaliar a compatibilidade quando uma transação de escrita do servidor* (TEW) *está executando e uma transação sensor* (TES) *é invocada para executar.*

Definição 3 *Um escalonador ES modela o entrelaçamento das operações pertencentes às transações sensores. Dessa forma, o escalonador ES é definido sobre o conjunto de transações* S_i*. Nesse caso, são definidas duas funções de compatbilidade:*

1. FC_1 *para avaliar a compatibilidade quando uma transação de aquisição sensor* (TAS) *está executando e uma transação de leitura do servidor* (TLS) *é invocada para executar.*

2. *FC_2 para avaliar a compatibilidade quando uma transação de leitura do servidor (TLS) está executando e uma transação de aquisição sensor (TAS) é invocada para executar.*

3.6 Arquitetura do Sistema

Na Figura 3.3 é ilustrada a arquitetura do sistema proposto, onde é possível identificar as transações sensores, transações do servidor, além dos escalonadores EW e ES.

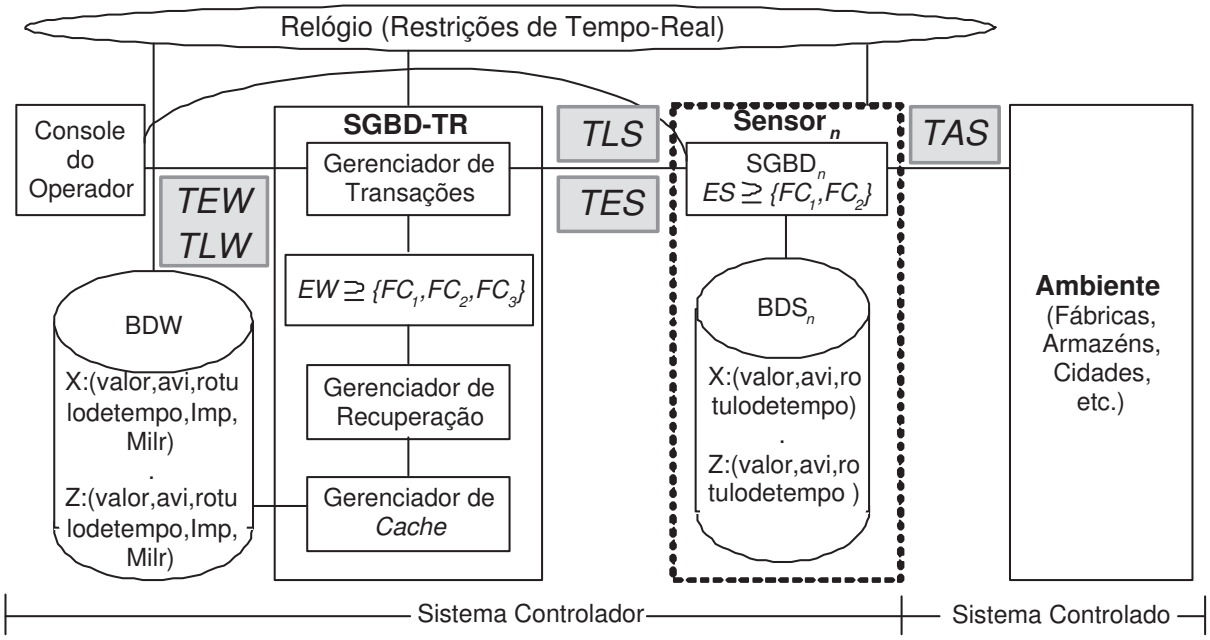

Figura 3.3: Esquema da arquitetura do sistema, ilustrando os escalonadores EW e ES

O conjunto de transações do servidor é representado por W_i e essas são iniciadas pelo *console do operador* que são os programas aplicativos e pelos sensores inteligentes. Quando enviadas para o servidor, estas transações são colocadas no gerenciador de transações, para então serem enviadas para o escalonador EW. Em caso de execução concorrente são definidas três funções de compatibilidade para avaliar se duas operações de transações diferentes conflitantes podem ser executadas concorrentemente.

O conjunto de transações sensores é representado por S_i e essas possuem operações executadas nos sensores. O escalonador ES definido para os sensores possuem duas funções de compatibilidade. As situações de concorrência nos sensores são: (i) quando o item de dado está sendo adquirido do sistema controlado e uma transação do servidor de consulta está tentando acessar o sensor e; (ii) quando uma transação do servidor de consulta está acessando o sensor, no instante em que um item de dado está sendo adquirido do sistema controlado.

Capítulo 4

Qualidade de Serviços

4.1 Introdução

A execução concorrente das transações em um SGBD convencional deve atender a algum critério de corretude para gerar escalonamentos considerados corretos, como descrito no Capítulo 3. Estes critérios devem atender as propriedades ACID das transações.

Nos SGBD-TR, os escalonamentos gerados devem levar em consideração, além da consistência lógica, as restrições de tempo impostas às transações e os intervalos de validade temporal atribuídos para alguns dados. O escalonamento ideal nestes sistemas deve assegurar que cada tarefa complete sua execução dentro do prazo previsto. Isso pode ser validado através da análise de escalonabilidade, definido para STR (LIU, 2000), que determina se um conjunto de tarefas executando concorrentemente satisfaz as restrições de tempo e podem ser escalonadas com sucesso (CHENG, 2002; COOLING, 2003). Para a análise de escalonabilidade ser realizada é necessário que o sistema seja previsível, ou seja, os recursos necessários e as ações que serão executadas para isso devem ser conhecidos.

Nos últimos anos, a demanda por SGBD-TR tem se estendido para executar em ambientes onde é difícil prever quais as ações que serão executadas e que recursos serão necessários. Uma outra característica destas aplicações é a presença de transações com restrições de tempo-real suaves e/ou firmes, ou seja, transações que é desejável atender suas restrições de tempo.

Como conseqüência, aparece à necessidade de gerenciar o *desempenho* do sistema em relação à quantidade de transações que podem perder seus prazos, e a *corretude* desse considerando o quão impreciso o item de dado pode ser para ainda ser considerado válido. Estas medidas podem variar de aplicação para aplicação, visto que é permitido que diferentes aplicações acessem o repositório de dados com diferentes exigências.

Por estas razões, mecanismos de gerenciamento de QoS estão sendo propostos, através desta pesquisa, para garantir o desempenho e a corretude das aplicações utilizando SGBD-

TR.

O restante desse Capítulo está organizado da seguinte forma: na Seção 4.2 são definidos o mecanismo de gerenciamento de QoS e as funções propostas para este mecanismo. Na Seção 4.3, QoS é descrita no contexto dos SGBD-TR, definindo funções e métricas de desempenho. Uma extensão do conceito de dados e transações seguros, bem como da FC, para permitir um maior número de escalonamentos aceitáveis é apresentada na Seção 4.4. Na Seção 4.5 é ilustrada uma visão funcional para o acesso ao SGBD-TR considerando as funções e as métricas apresentadas. Por fim, a arquitetura do SGBD-TR proposta nesta pesquisa é mostrada.

4.2 Qualidade de Serviços

4.2.1 Definição

Qualidade de serviços pode ser definida como o conjunto de características de um sistema necessário para atingir uma determinada funcionalidade (AURRECOECHEA; CAMPBELL; HAUW, 1998). O processamento de QoS em um sistema começa com a definição dos parâmetros exigidos pelo usuário. Estes parâmetros são mapeados e negociados entre os componentes do sistema, assegurando que todos podem atingir um nível de qualidade aceitável. Recursos são então alocados e monitorados, havendo possibilidade de renegociação caso as condições do sistema se alterem.

O conceito de qualidade de serviços foi originalmente introduzido em redes de computadores para caracterizar principalmente o desempenho em transmissão de dados (FIROIU et al., 2002; LI; NAHRSTEDT, 1998; COCCOLI; BONDAVALLI; GIANDOMENICO, 2001). Com o sucesso da utilização do gerenciamento de QoS, pesquisas atuais visam integrar estes conceitos a outras áreas da computação. Nos bancos de dados multimídia onde os objetos multimídia são volumosos, manipulações e visualizações destes objetos podem requerer uma grande quantidade de tempo e de recursos. Através de QoS estes sistemas podem fornecer a informação pertinente com a qualidade requerida e acesso eficiente (CARDOSO, 2002; YE; KERHERVé; BOCHMANN, 1999; STAEHLI, 1996; GOEBEL et al., 1998; DONALDSON, 1994; SALAMATIAN; FDIDA, 2001; BORN; HALTEREN; KATH, 2000).

Algumas pesquisas recentes têm focado a utilização de mecanismos de QoS para SGBD-TR. Dentre elas algumas merecem destaque, tal como a Tese de Doutorado de Kang (2003) onde é apresentada uma abordagem baseada em QoS para o gerenciamento de dados em tempo-real. No entanto, nenhuma função de QoS é definida no contexto dos SGBD-TR, além de não ficar explícito qual critério de corretude utilizado para definir a execução concorrente das transações. Também não é desenvolvido nenhum modelo formal

para análises e simulações, a fim de orientar o processo de tomada de decisões e propor soluções para a melhoria do sistema.

Amirijoo, Hansson e Son (2003) é outra pesquisa interessante para o assunto ora abordado, onde a especificação e o gerenciamento de QoS são utilizados durante o escalonamento das transações em bancos de dados em tempo-real imprecisos. Contudo, o gerenciamento de QoS é considerado apenas durante o escalonamento das transações. Em Ribeiro Neto, Perkusich e M.L.B. (2003) são definidas funções de QoS para especificar a qualidade do sistema requerida e assegurar que estas especificações sejam atendidas. Através destas funções uma funcionalidade fim a fim é fornecida ao sistema. Isso é especialmente importante quando ambientes imprevisíveis são considerados durante a execução das transações, tais como a internet e redes de sensores.

4.2.2 Funções de QoS

A garantia de qualidade fim a fim é usualmente oferecida através de funções de gerenciamento de QoS utilizadas para lidar com diferentes situações que porventura possam ocorrer durante o tempo de vida dos sistemas (AURRECOECHEA; CAMPBELL; HAUW, 1998; YE; KERHERVé; BOCHMANN, 1999; BOCHMANN et al., 1996). Na Figura 4.1 tem-se a ilustração das funções de QoS e sua abrangência em relação aos componentes necessários para o funcionamento do sistema.

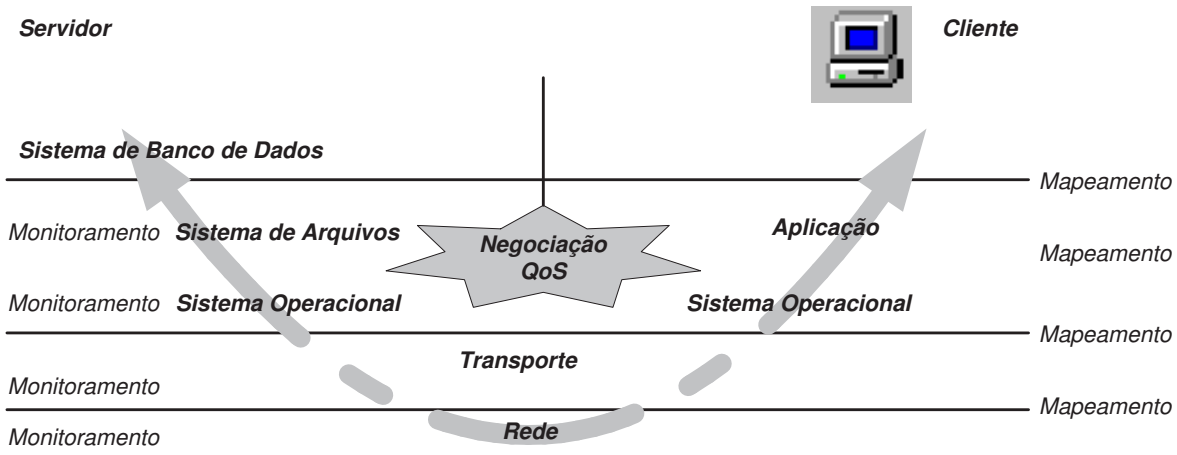

Figura 4.1: Camadas do SGBD-TR e Gerenciamento de QoS

São cinco as funções de QoS definidas nesta pesquisa.

- *Função de Especificação*: função básica disponível em qualquer sistema de gerenciamento de QoS. Através dessa são identificados os parâmetros disponíveis, bem como a sintaxe e a semântica desses.

- *Função de Mapeamento*: traduz as exigências de qualidade definidas em um nível do sistema para todos os outros. Como exemplo, considere o pedido de um usuário para receber um filme em alta qualidade, tamanho grande e colorido. Tais parâmetros devem ser traduzidos em termos de vazão, atraso e tremulação (*Throughput, delay e jitter*, respectivamente) na camada de rede (FISCHER; MEER, 1998).

- *Função de Negociação*: representa a negociação entre as diferentes camadas e componentes do sistema a fim de satisfazer o que foi solicitado. Considerando uma tele-conferência, o protocolo de negociação é executado para verificar se o sistema tem recursos disponíveis para acomodar um novo usuário que quiser entrar numa sessão, sem comprometer as garantias dadas aos participantes já existentes. Outros mecanismos podem ser definidos para realizar a negociação, tais como:

  - *controle de admissão*: usado para determinar se o novo usuário pode ser admitido e,
  - *reserva de recurso*: executada a fim de garantir o que foi pedido, quando o novo usuário for admitido.

- *Função de Monitoração*: examina o nível atual de QoS provido pelos componentes do sistema e compara esse com as exigências iniciais. Quando as exigências iniciais não são satisfeitas, o usuário ou a aplicação é notificado para uma eventual decisão de re-negociação.

- *Função de Re-negociação*: pode ser iniciada pelo usuário ou pela aplicação. Uma re-negociação iniciada pelo usuário permite a esse pedir uma melhor qualidade. Por exemplo, o usuário solicita vídeo colorido enquanto é fornecido preto e branco para reduzir o custo da sessão corrente. Esse tipo de re-negociação é crucial para aplicações onde QoS não pode ser especificada durante o serviço inteiro. Exemplos são aplicações médicas, sistemas de tele-conferência e trabalhos cooperativos suportados pelo computador onde não se podem prever corretamente as situações a serem executadas durante a fase ativa da sessão. Uma re-negociação iniciada pela aplicação é executada sempre que o sistema não suportar uma negociação mais longa.

4.3 Qualidade de Serviços para SGBD-TR

4.3.1 Funções de QoS para SGBD-TR

No contexto de bancos de dados em tempo-real, as funções utilizadas para o gerenciamento de QoS são definidas como segue:

- *Função de Especificação*: através desta função são identificados os parâmetros de tempo e os parâmetros lógicos necessários para garantir o perfeito funcionamento do sistema. Uma vez identificados, alguns destes parâmetros são especificados a *priori* e outros em tempo de execução de acordo com a situação atual do sistema. Dentre os parâmetros especificados inicialmente tem-se a periodicidade, o prazo e o tempo de liberação de uma transação. Em relação aos valores especificados em tempo execução está a imprecisão admitida no item de dado.

- *Função de Mapeamento*: traduz as exigências de qualidade definidas nas transações sensores e transações do servidor para os sistemas de gerenciamento de dados, mais especificamente para os escalonadores. Por exemplo, para a função de compatibilidade ser avaliada é necessário que o rótulo de tempo e o intervalo de validade absoluta definida para o item de dado sejam conhecidos.

- *Função de Negociação*: esta função substitui a função de compatibilidade. Por conseguinte, o papel da função de negociação é gerenciar a execução concorrente das transações. Essa deve, além de permitir e limitar valores imprecisos, considerar valores aceitáveis para a negociação a fim de evitar que uma das transações seja abortada ou perca seu prazo. As principais vantagens de FN em relação à FC é a possibilidade de avaliar mais de duas transações executando concorrentemente e a definição do limite de imprecisão permitida para o item de dado ser obtida por uma métrica de desempenho.

  - *controle de admissão*: usado para determinar se uma transação pode ser admitida considerando a situação atual do sistema e,
  - *reserva de recurso*: se a transação for admitida deve existir um escalonamento correto.

- *Função de Monitoração*: examina se os prazos especificados para as transações estão sendo obedecidos e, se os limites de imprecisão definidos para os itens de dados não foram excedidos. Também, controla a quantidade de transações que podem perder o prazo em um determinado intervalo de tempo.

- *Função de Re-negociação*: caso uma transação não atenda seus prazos é verificado se ela pode ser reiniciada. Esta função é adequada para transações com prazos firmes ou suaves.

Execução das Transações com QoS

Na Figura 4.2 é ilustrado um esquema simplificado onde as funções de QoS são utilizadas durante a execução das transações nos SGBD-TR. As transações de leitura e de escrita

possuem os parâmetros temporais que são especificados através da FE. Uma vez especificados, estes parâmetros serão mapeados (FM) durante toda a execução da transação. No caso de execução concorrente de transações conflitantes, uma negociação deve ser realizada onde uma FN é avaliada. Antes das transações completarem a FMo é executada para verificar se a qualidade requerida está sendo atendida.

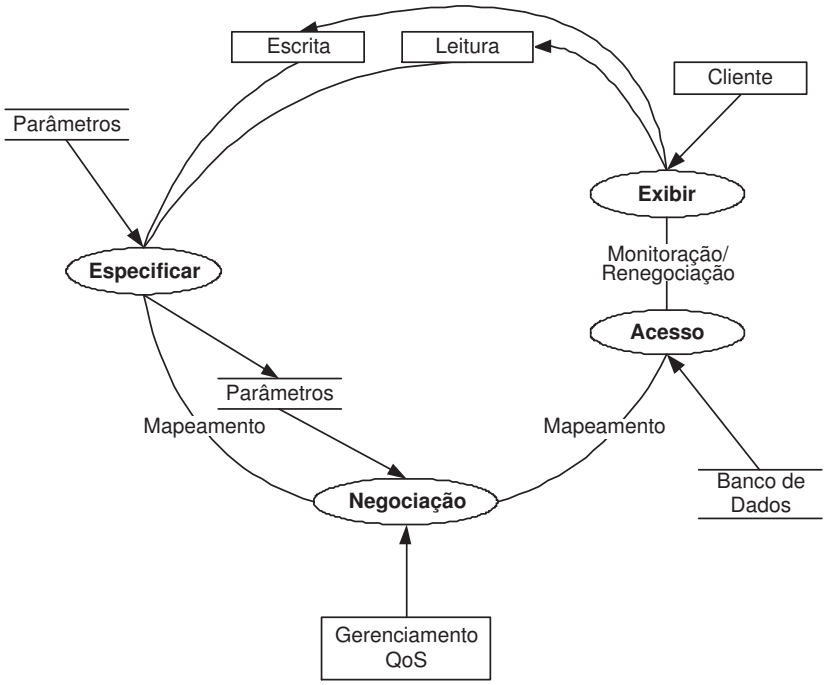

Figura 4.2: Execução das Transações com QoS

4.3.2 Métricas de Desempenho para SGBD-TR

Algumas métricas de desempenho são definidas para garantir a corretude dos SGBD-TR. Estas métricas são especificadas, mapeadas e, se necessário, negociadas entre os componentes do sistema. Duas métricas são definidas nesta pesquisa, como mostradas a seguir:

- Número de transações que perderam o prazo (Pt): esta métrica estabelece a taxa de perda de prazos das transações que pode ser tolerada durante um determinado intervalo de tempo.

  A métrica é definida como,

$$Pt = 100 \times \frac{TransacoesPerderamPrazo}{TransacoesConcluidas} \tag{4.1}$$

  onde Pt é a quantidade de transações que perderam o prazo final ($Transacoes$

$PerderamPrazo$) em relação a quantidade de transações que concluíram seus prazos com sucesso ($TransacoesConcluidas$).

- Imprecisão máxima do dado ($Impr$): é a imprecisão máxima admitida no item de dado para ele ainda ser considerado válido logicamente.

  $Impr$ é definida como,

$$Impr = \frac{Imp}{100} \times ValorCorrente \tag{4.2}$$

  onde $ValorCorrente$ é o valor do item de dado que está armazenado no banco de dados e Imp é o índice de quanto o item de dado pode ser impreciso.

4.4 Especificação Formal de Serialização *Epsilon* considerando QoS

Nesta Seção, o critério de corretude serialização *Epsilon* é redefinido considerando conceitos de qualidade de serviços. O objetivo é garantir um maior número de escalonamentos corretos que os permitidos por SE, por considerar valores em tempo de execução para atender a necessidade da aplicação.

Para isso, as propriedades de segurança para as transações e para os dados são redefinidas, bem como são definidas as *funções de negociação*.

4.4.1 Propriedades de Segurança para Transações e Dados

No Capítulo 3 é definido que serialização *Epsilon* está garantido se e somente se todos os dados e transações são seguros. A redefinição das propriedades de segurança consiste em considerar a métrica $Impr$, a fim de permitir que parâmetros sejam limitados a valores aceitáveis visando atender o propósito da aplicação.

Assim, um item de dado x é seguro se e somente se,

$$x : quant\_imp_x \leq Impr$$

onde a imprecisão no item de dado x é aceitável, se ela não for maior que o limite de imprecisão definido em tempo de execução pela métrica $Impr$.

Da mesma forma, uma $T\epsilon$ de leitura é considerada segura se e somente se

$$(T\epsilon, x)_{leitura} : imprec\_imp_{T\epsilon,x} \leq Impr_{T\epsilon,x}$$

onde a imprecisão importada no item de dado x não pode exceder o limite de imprecisão importado pela $T\epsilon$ para o item de dado x, definido pela métrica $Impr$.

A mesma regra é aplicada a uma transação-*Epsilon* de escrita.

$$((T\epsilon, x)_{escrita} : imprec\_exp_{T\epsilon,x} \leq Impr_{T\epsilon,x})$$

onde a imprecisão exportada no item de dado x não pode exceder o limite de imprecisão exportada pela $T\epsilon$ para o item de dado x, definido pela métrica $Impr$.

4.4.2 Definição Formal de Função de Negociação

Função de Negociação(Leitura,Escrita)

Em 4.3 é apresentada a função de compatibilidade que expressa a negociação entre uma $T\epsilon_i$ de leitura ($r\epsilon_i(x)$) que está sendo executada e uma $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) que é invocada para executar sobre o mesmo item de dado x. Esta FC foi descrita com mais detalhes no Capítulo 3.

$$\begin{aligned} FC_1(r\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\ &\wedge (|x.val - x_{novo}.val| \leq (lim\_imp_{T\epsilon,x} \\ &- (x.quant\_imp + x_{novo}.quant\_imp))) \\ &\Rightarrow x.quant\_imp = x.quant\_imp \\ &+ x_{novo}.quant\_imp + |x.val - x_{novo}.val| \end{aligned} \tag{4.3}$$

onde $x.tempo$ é o rótulo de tempo do item de dado x, ou seja, quando esse valor foi gravado e $x.avi$ é o tempo de validade absoluta do item de dado x.

A função de negociação definida para expressar a negociação descrita na Equação 4.3 é apresentada na Equação 4.4.

$$\begin{aligned} FN_1(r\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\ &\wedge (|x.val - x_{novo}.val| \leq (Impr_{T\epsilon,x} \\ &- (x.quant\_imp + x_{novo}.quant\_imp))) \\ &\Rightarrow x.quant\_imp = x.quant\_imp \\ &+ x_{novo}.quant\_imp + |x.val - x_{novo}.val| \end{aligned} \tag{4.4}$$

Função de Negociação(Escrita,Leitura)

Na função de compatibilidade 4.5, a $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) está atualizando o valor do item de dado x com o valor x_{novo} e a $T\epsilon_i$ de leitura ($r\epsilon_i(x)$) é invocada para ler o valor do item de dado x. $x.tempo$ é o rótulo de tempo do item de dado x, ou seja, quando esse valor foi gravado e $x.avi$ é o tempo de validade absoluta do item de dado x.

$$
\begin{aligned}
FC_2(w\epsilon_j(x), r\epsilon_i(x)) = (&agora - x.tempo \leq x.avi) \\
&\wedge (|x.val - x_{novo}.val| \leq \\
&(lim\_imp_{T\epsilon,x} - x.quant\_imp) \\
&\Rightarrow x.quant\_imp = \\
&x.quant\_imp + |x.val - x_{novo}.val|
\end{aligned} \tag{4.5}
$$

A função de negociação 4.6 expressa a negociação descrita na Equação 4.5, considerando a propriedade de segurança definida nesta Seção.

$$
\begin{aligned}
FN_2(w\epsilon_j(x), r\epsilon_i(x)) = (&agora - x.tempo \leq x.avi) \\
&\wedge (|x.val - x_{novo}.val| \leq \\
&(Impr_{T\epsilon,x} - x.quant\_imp) \\
&\Rightarrow x.quant\_imp = \\
&x.quant\_imp + |x.val - x_{novo}.val|
\end{aligned} \tag{4.6}
$$

Função de Negociação(Escrita,Escrita)

Através da função de compatibilidade 4.7 é mostrada a situação onde um bloqueio exclusivo é concedido a uma $T\epsilon_i$ de escrita ($w\epsilon_i(x)$) e uma outra $T\epsilon_j$ de escrita ($w\epsilon_j(x)$) solicita esse bloqueio.

$$
\begin{aligned}
FC_3(w\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\
&\wedge (|x.val - x_{novo,k}.val| \leq \\
&(lim\_max_{T\epsilon,x} - x.quant\_imp) \\
&\Rightarrow x.quant\_imp = \\
&x.quant\_imp + |x.val - x_{novo,k}.val|
\end{aligned} \tag{4.7}
$$

A função de negociação 4.8 expressa a execução concorrente de duas $T\epsilon$ de escrita ($w\epsilon_i(x)$ e $w\epsilon_j(x)$), considerando a métrica de desempenho $Impr$.

$$
\begin{aligned}
FN_3(w\epsilon_i(x), w\epsilon_j(x)) = (&agora - x.tempo \leq x.avi) \\
&\wedge (|x.val - x_{novo,k}.val| \leq \\
&(Impr_{T\epsilon,x} - x.quant\_imp) \\
&\Rightarrow x.quant\_imp = \\
&x.quant\_imp + |x.val - x_{novo,k}.val|
\end{aligned} \tag{4.8}
$$

4.4.3 Representação da Função de Negociação

Uma representação para a execução da função de negociação é ilustrado na Figura 4.3. A situação representa uma $T\epsilon_i$ de escrita executando e várias $T\epsilon_j$ são invocadas, com $j = 2, 3, 4, ..., n$. De acordo com a figura a transação T_1 é a $T\epsilon_i$ de escrita e $T_2, T_3, ..., T_n$ representam as $T\epsilon_j$ invocadas. A função de negociação é representada pelos retângulos rotulados por Parâmetros de tempo, Parâmetros Lógicos e Gerenciamento QoS. O repositório dos dados é representado na Figura por BDW.

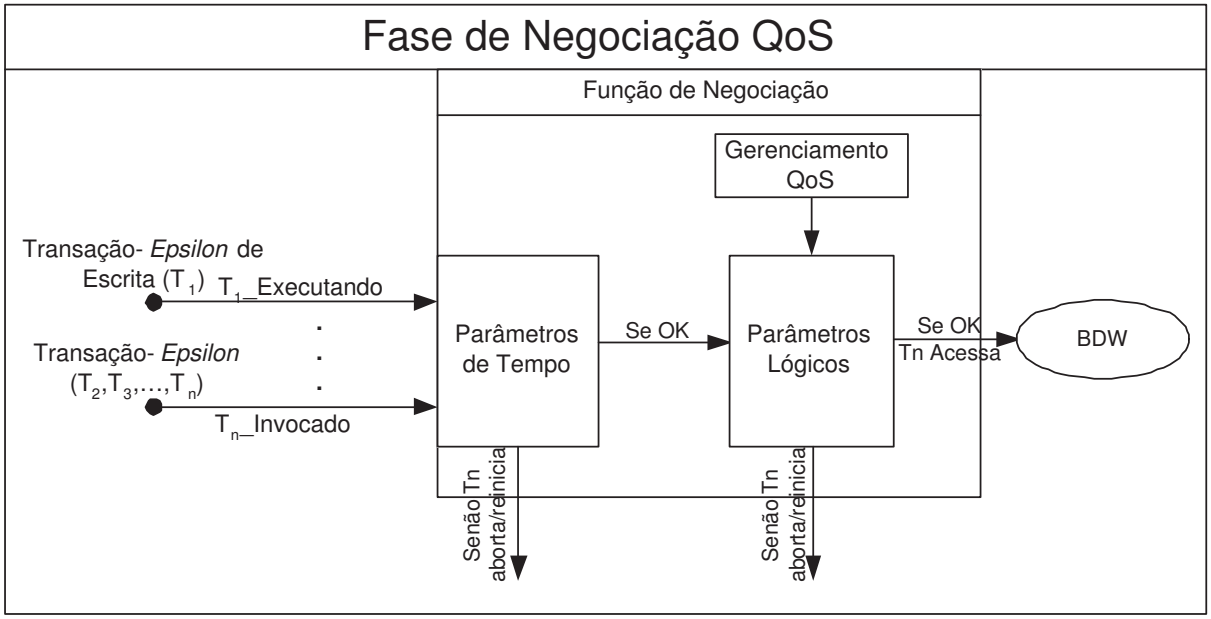

Figura 4.3: Representação da Função de Negociação

Na situação em que a função de negociação é avaliada quando T_1 está executando e T_2 é invocada para acessar o banco de dados. FN pode ser avaliada em verdadeira ou avaliada em falsa, onde primeiro os parâmetros de tempo são avaliados para então verificar os parâmetros lógicos. Se os parâmetros de tempo da FN forem avaliados como falso, então T_2 não poderá executar concorrentemente, podendo reiniciar caso exista tempo suficiente para esperar T_1 terminar. Se estes parâmetros forem avaliados como verdadeiro, então os parâmetros lógicos serão avaliados.

Os parâmetros lógicos são avaliados considerando a métrica de desempenho $Impr$, onde a diferença entre o valor do item de dado no repositório e o valor que será escrito pela transação T_1 deve ser inferior ao limite máximo de imprecisão permitido por T_2 definido pela métrica $Impr$.

4.5 Arquitetura do SGBD-TR

A função de negociação é executada pelo componente escalonador definido no módulo SGBD-TR, como descrito no Capítulo 2. Outros componentes formam este módulo, tais como o gerenciador de transações, gerenciador de recuperação e gerenciador de *cache* (HOLANDA; BRAYNER; FIALHO, 2004).

Nesta Seção, o gerenciador de transações e o escalonador serão descritos de forma a contemplar os conceitos apresentados. Na Figura 4.4, um esquema é ilustrado enfocando o funcionamento destes dois componentes. O gerenciador de transações é responsável por receber e encaminhar as transações para o escalonador e é representado na Figura por *Fila de Pronto*. As transações do servidor (TES, TEW, TLW) são enfileiradas de forma que as TES (transações de escrita sensores) possuem prioridades em relação às TEW (transações de escrita servidor) e TLW (transações de leitura servidor).

As TEW e TLW são escalonadas se nenhuma TES estiver pronta para executar e podem ser retomadas se uma nova TES for enfileirada. Este controle é realizado por *Controle de Admissão* ilustrado na Figura.

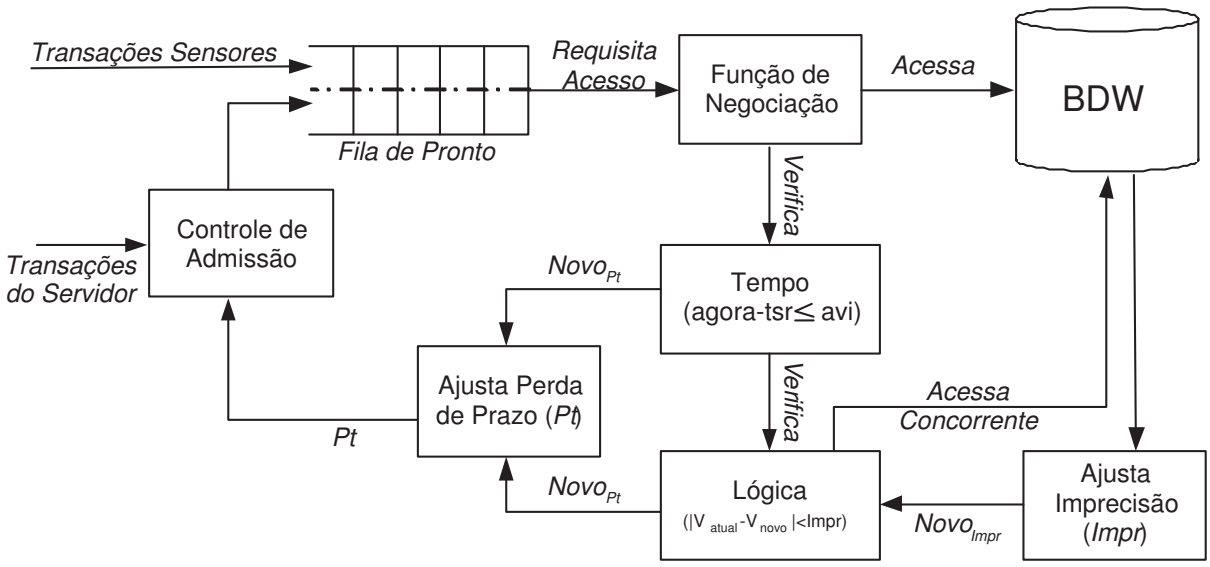

Figura 4.4: Esquema para ilustrar o Gerenciador de Transações e Escalonador do SGBD-TR

No escalonador é feito o controle da execução concorrente das transações. A execução concorrente das transações conflitantes é gerenciada pela função de negociação. Os *Parâmetros de Tempo*, os *Parâmetros Lógicos* e *Gerenciamento QoS*, ilustrados na Figura 4.3, são mostrados com maiores detalhes, onde:

- *Parâmetros de Tempo*: é verificado o intervalo de validade absoluta do item de dado e ilustrado na Figura pelo caixa *Tempo*;

- *Parâmetros Lógicos*: é verificada a propriedade de segurança definida para o item de dado e ilustrado na Figura pela caixa *Lógica*;

- *Gerenciamento QoS*: representado pelas métricas de desempenho que são monitoradas e alimentadas em *Ajusta Perda de Prazo (Pt)* e *Ajusta Imprecisão (Impr)* durante um determinado intervalo de tempo.

Em *Ajusta Imprecisão (Impr)* o limite de imprecisão permitido para o item de dado é obtido em relação ao valor atual deste item representado no BDW. A imprecisão é obtida através da diferença entre o valor atual do item de dado armazenado em BDW (V_{atual}) e o valor que será atualizado (V_{novo}), onde esta diferença deve ser menor que $Impr$ ($\mid V_{atual} - V_{novo} \mid < Impr$)).

Para *Ajusta Perda de Prazo (Pt)* é definida a taxa de perda de prazos das transações que pode ser tolerada durante um determinado intervalo de tempo. Por exemplo, considere que o valor especificado para Pt é igual a quinze por cento, ou seja, a cada cem transações quinze podem perder seus respectivos prazos. Se o valor atual da métrica Pt estiver acima do especificado, nenhuma transação do servidor será admitida, essa filtragem é realizada pelo *Controle de Admissão*. O valor de Pt é alterado se a função de negociação for avaliada em falsa. Isso acontece se os *Parâmetros de Tempo* ou os *Parâmetros Lógicos* não forem satisfeitos, incrementando a quantidade de transações abortadas.

4.6 Arquitetura do Sistema

Na Figura 4.5 é ilustrada a arquitetura do sistema, contemplando as funções de QoS e as métricas de desempenho adotadas.

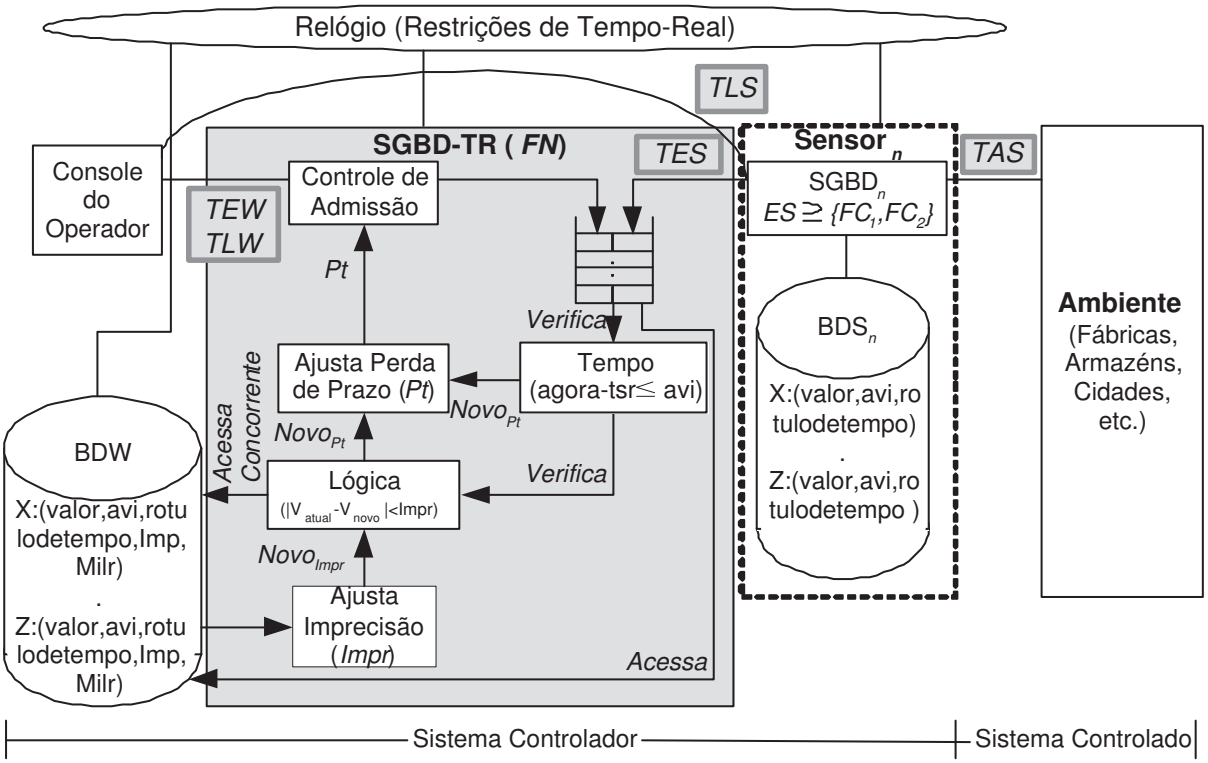

Figura 4.5: Arquitetura do Sistema com as Funções de QoS e Métricas de Desempenho

As transações são enfileiradas quando submetidas ao SGBD-TR. A fila ilustrada na figura representa o gerenciador de transações. Se o item de dado a ser utilizado por uma transação não estiver sendo acessado por outra transação, então o acesso ao item no BDW é cedido. Na situação em que o item de dado estiver sendo manipulado por outra transação conflitante, então a FN é avaliada como descrito na Seção 4.5.

Os parâmetros das transações são especificados (FE) sempre que elas são submetidas. Estes parâmetros são mapeados (FM) e se necessário negociados (FN). Antes da transação completar é realizada uma monitoração (FMo) para verificar se a qualidade especificada está sendo atendida.

Capítulo 5

Redes de Petri

Neste Capítulo os conceitos básicos relativos as redes de Petri coloridas hierárquicas são apresentados. HCPN são utilizadas como base formal no desenvolvimento do modelo proposto nesta pesquisa.

5.1 Definição

As redes de Petri são originadas da pesquisa de doutorado de Carl Adam Petri que apresentou um tipo de grafo bipartido[1] com os estados associados, tendo como objetivo estudar a comunicação entre os autômatos (MURATA, 1989). Desde então, surgiram várias extensões das redes de Petri que são usadas para modelar e analisar uma variedade de sistemas, devido às suas potencialidades de representar 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, permitindo uma visualização simultânea de sua estrutura e comportamento.

5.1.1 Estrutura da Rede de Petri

A estrutura de uma rede de Petri, também chamada de rede de Petri Lugar/Transição, é representada por um grafo bipartido direcionado, cujos elementos são de dois tipos distintos: *lugares* e *transições*. A estrutura é representada pela tupla $N = \{P, T, F\}$, na qual:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;

[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.

- $F \subseteq P \times T \cup T \times P$ é uma relação de fluxo;
- $P \cap T = \varnothing$.

Na Figura 5.1 é ilustrado um exemplo de uma rede de Petri Lugar/Transição, onde:

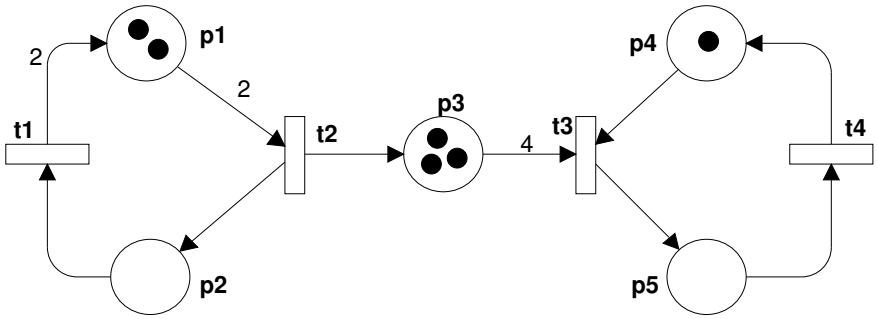

Figura 5.1: Rede de Petri Lugar/Transição

- $P = \{p1, p2, p3, p4, p5\}$;
- $T = \{t1, t2, t3, t4\}$;
- $F = \{(t1, p1), (p1, t2), (t2, p2), (p2, t1), (t2, p3), (p3, t3), (t3, p5),$ $(p5, t4), (t4, p4), (p4, t3)\}$;
- $P \cap T = \varnothing$.

Por padronização, os lugares são representados por círculos e as transições são representadas por retângulos. Esses elementos compõem a *estrutura de rede*, que possuem *inscrições* associadas. As inscrições podem ser de dois tipos: o *peso dos arcos* e a *marcação*. O peso dos arcos equivale a um inteiro associado a cada arco direcionado. Na Figura 5.1, o peso do arco direcionado entre o lugar `p1` e a transição `t2` é dois. Já a marcação equivale ao estado da rede e é representada por elementos associados aos lugares, denominados *fichas*. As fichas, representadas por pequenos círculos pretos, estabelecem a marcação da rede. Através da ação das transições, denominada de *ocorrência*, a marcação da rede pode ser alterada. As inscrições são representadas pela tupla $PN = \{N, W, M_0\}$, 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 rede de Petri ilustrada na Figura 5.1 pode ser formalmente definida como:

- $N = \{P, T, F, W, M_0\}$;
- $P = \{p1, p2, p3, p4, p5\}$;
- $T = \{t1, t2, t3, t4\}$;
- $F = \{(t1, p1), (p1, t2), (t2, p2), (p2, t1), (t2, p3), (p3, t3), (t3, p5), (p5, t4), (t4, p4), (p4, t3)\}$;
- $W = \{[(t1, p1), 2], [(p1, t2), 2], [(t2, p2), 1], [(p2, t1), 1], [(t2, p3), 1], [(p3, t3), 4], [(t3, p5), 1], [(p5, t4), 1], [(t4, p4), 1], [(p4, t3), 1]\}$;
- $M_0 = \{(p1, 2), (p2, 0), (p3, 3), (p4, 1), (p5, 0)\}$

5.1.2 Comportamento da Rede de Petri

Um *arco de entrada* é um arco orientado para um lugar ou para uma transição, denominados de lugar de entrada e transição de entrada, respectivamente. De forma simétrica, um *arco de saída* é um arco orientado de uma transição ou de um lugar, denominados de transição de saída e lugar de saída, respectivamente.

Uma transição só pode ocorrer se em cada lugar de saída, há uma quantidade de fichas pelo menos igual ao peso do arco. Quando isso acontece, diz-se que a transição está *habilitada* a ocorrer. Ao ocorrer, uma quantidade de fichas igual ao peso do arco de entrada da transição, é removida do lugar de saída. Da mesma forma, uma quantidade de fichas, igual ao peso do arco de saída da transição, é acrescentada a cada lugar de entrada. Para esclarecer o que foi descrito a Figura 5.2 é ilustrada. Essa Figura é um estado seguinte ao estado da rede da Figura 5.1, devido à ocorrência da transição t2. A transição t2 refere-se à transição entrega na Figura 5.2.

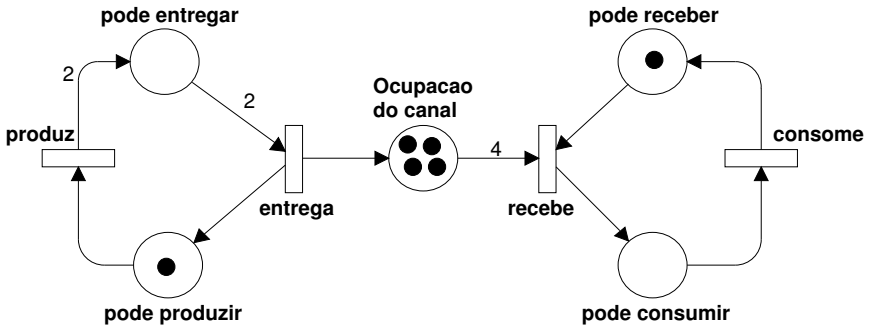

Figura 5.2: Estado da rede de Petri Lugar/Transição depois da ocorrência da transição entrega

Um sistema modelado usando redes de Petri não prevê nem impõe uma ordem de ocorrência das transições. Isso significa que se em um determinado instante, mais de

uma transição se encontrar habilitada, qualquer uma poderá ocorrer. Isso pode ser observado através da Figura 5.2, onde as transições produz e recebe estão habilitadas, podendo qualquer uma das duas ocorrer.

5.1.3 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 5.2 seria a modelagem de um sistema produtor-consumidor. O produtor é modelado pelos lugares *pode produzir* e *pode entregar* e pelas transições *produz* e *entrega*. Para se produzir uma unidade são necessários dois itens. Por outro lado, o consumidor é modelado pelos lugares *pode receber* e *pode consumir* e pelas transições *recebe* e *consome*. Esse só pode receber novas unidades se já tiver consumido a que tinha. São necessários quatro elementos para o consumidor receber um.

5.2 Extensões de Redes de Petri

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

Um modelo em redes de Petri para descrever sistemas reais tende a ser complexo e extremamente grande. 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. Dentre elas estão às extensões relacionadas com a capacidade de modelagem funcional e as extensões relacionadas com a caracterização de restrições temporais.

5.2.1 Redes de Petri Coloridas

Para a modelagem de sistemas constituídos de muitos componentes idênticos que interagem entre si 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.

As redes de Petri coloridas (CPN - *Coloured Petri Nets*) (JENSEN, 1990, 1992, 1997) são extensões às redes de Petri Lugar/Transição, onde as fichas podem ter diferentes tipos de informações associadas. Dessa forma, pode-se modelar o componente comum apenas uma vez e associar diferentes fichas a esse componente, tornando possível representar as diferentes estruturas através de fichas. Para isso, a cada ficha é associado um valor 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 o conjunto de cores (domínios), as variáveis e as 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 contidas nas fichas;

4. *Inicializações*: associadas aos lugares, estabelecem a marcação inicial da rede.

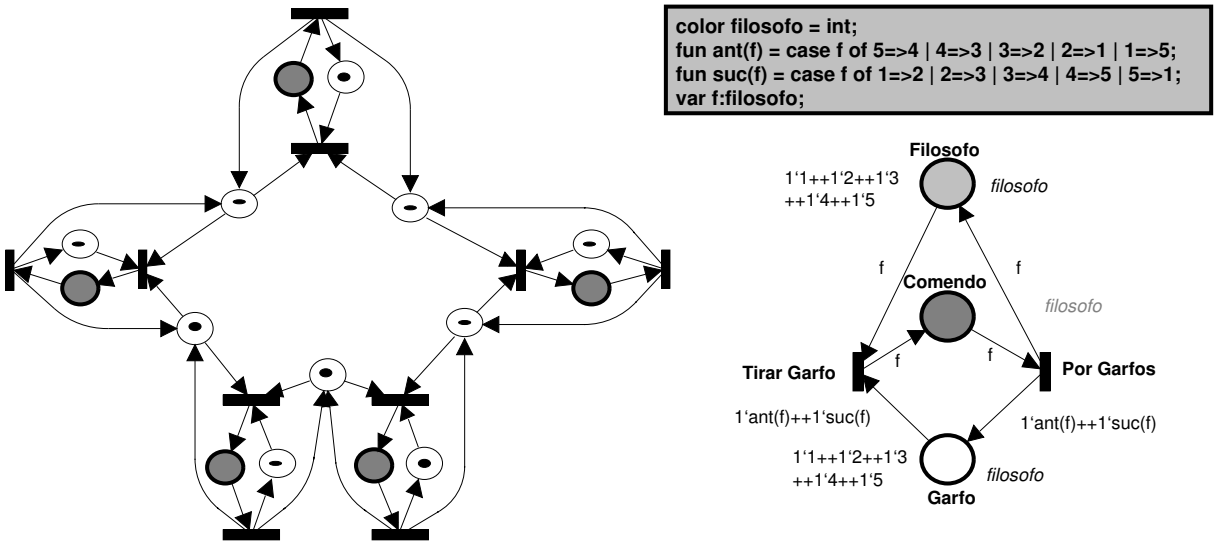

Figura 5.3: a) Rede de Petri Lugar/Transição; b) Rede de Petri colorida correspondente.

Na Figura 5.3, ilustra-se um exemplo de uma rede de Petri Lugar/Transição (Figura 5.3(a)) e uma rede de Petri colorida correspondente (Figura 5.3(b)), onde: o nó de declarações é expresso no retângulo na parte superior da Figura; as cores dos lugares são os textos em itálico próximos aos lugares (por exemplo, *filosofo*) e; as expressões dos arcos são os textos localizados próximo aos arcos direcionados (por exemplo, *1‘ant(f)++1‘suc(f)*).

Uma rede Petri Lugar/Transição sempre 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 e é conseguida através das expressões dos arcos que permitem determinar quais fichas remover ou adicionar aos lugares. Isso pode ser visto, na Figura 5.3(b), através da função ant(), onde essa função determina qual número inteiro deve ser removido do lugar Garfo. É importante salientar que a redução de tamanho da estrutura do modelo, resultante da utilização da rede de Petri colorida, será tanto mais significativa quanto maior for a quantidade de lugares idênticos.

5.2.2 Redes de Petri Coloridas Hierárquicas

As redes de Petri coloridas hierárquicas (HCPN) (JENSEN, 1998) disponibilizam mecanismos para construir modelos através da combinação de um conjunto de CPN denominadas *páginas*. Essa abordagem pode ser comparada à construção de um software a partir de um conjunto de módulos e funções. Do ponto de vista teórico, uma HCPN possibilita descrever os tipos de sistemas equivalentes a uma CPN, tornando sempre possível transformar uma HCPN em uma CPN. Considerando que um dos grandes problemas enfrentados por desenvolvedores de sistemas é lidar com muitos detalhes ao mesmo tempo, as HCPN podem ser bastante úteis.

Dois construtores de linguagens foram introduzidos em HCPN para possibilitar a construção hierárquica de CPN. Os dois construtores para hierarquia são chamados *transições de substituição* (*substitution transitions*) e *lugares de fusão* (*fusion place*).

- 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 removida ou adicionada dos outros lugares pertencentes ao conjunto. Um conjunto de lugares de fusão é denominado de conjunto de fusão (*fusion set*).

- Uma transição de substituição representa uma CPN mais complexa que descreve com mais detalhes as atividades modeladas por ela. A página que contém a transição de substituição é denominada *superpágina* da HCPN. A parte mais detalhada fica na página denominada *subpágina*. Cada transição de substituição é denominada de *supernó* da subpágina correspondente.

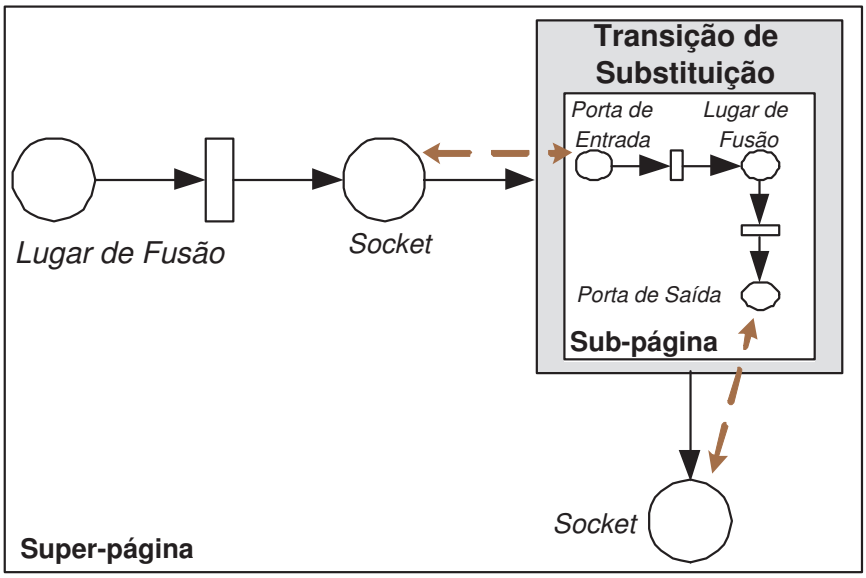

Figura 5.4: Exemplo de uma HCPN

Na Figura 5.4, uma HCPN é ilustrada. A transição de substituição se relaciona com sua subpágina através da utilização de dois membros denominados portas e *sockets*. *Sockets* são atribuídos aos lugares conectados à transição de substituição na superpágina e portas são associadas a determinados lugares na subpágina, tal que um par *socket*/porta forma 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*.

Como dito anteriormente, é sempre possível traduzir uma rede de Petri colorida hierárquica para sua correspondente não hierárquica. Para isso, basta substituir cada transição de substituição e arcos conectados, por sua respectiva subpágina, substituindo cada *socket* à sua respectiva porta.

Com HCPN é possível modelar aspectos relacionados ao tempo. Tais aspectos podem ser medidos 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.

5.3 Design/CPN

Design/CPN é uma ferramenta CASE (*Computer Aided Software Engineering*) usada para desenhar e simular redes de Petri coloridas hirárquicas (JENSEN; AL, 1999). Para analisar um modelo HCPN é fundamental dispor de um conjunto de ferramentas computacionais para edição, simulação e análise. O Design/CPN (CHRISTENSEN; JOERGENSEN; KRIS-

TENSEN, 1997; DAIMI, 2004a) é constituído de quatro ferramentas computacionais para o desenvolvimento de modelos de redes de Petri coloridas hierárquicas. Essas ferramentas são definidas a seguir:

1. Editor gráfico para construir, modificar e executar análise sintática de modelos HCPN;
2. Um simulador podendo operar de forma 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 HCPN;
4. Uma ferramenta para análise de desempenho que possibilita a simulação e observação de dados de desempenho de modelos HCPN.

Capítulo 6

Redes de Sensores

Neste Capítulo, os conceitos sobre as Redes de Sensores (RS), mais especificamente os nodos que compõem estas redes, as abordagens propostas para o armazenamento dos dados, além dos tipos de consultas que podem ser realizadas serão definidos.

Tais definições sobre RS são importantes nesta pesquisa, uma vez que uma arquitetura para os sistemas de bancos de dados em tempo-real é proposta considerando tais redes como o domínio de aplicação.

6.1 Definição

As redes de sensores são compostas por centenas ou milhares de dispositivos autônomos, denominados sensores, e são utilizadas para monitorar e eventualmente, controlar o ambiente. Nos últimos anos, as redes de sensores têm sido bastante utilizadas em aplicações nas áreas de turismo, transporte, educação, saúde entre outras (YAO; GEHRKE, 2002; CHEN; GEHRKE; KORN, 2001; BONNET; SESHADRI, 2000). No escopo deste trabalho, o principal objetivo de uma rede de sensores é coletar dados de uma região de observação específica, processar estes dados, permitindo que consultas sejam realizadas, e transmiti-los para um ou mais pontos de acesso à rede, no caso as estações base. Este procedimento resulta em informações que podem ser úteis para várias aplicações. Na Figura 6.1 tem a ilustração de uma rede de sensores sem fio com uma única estação base.

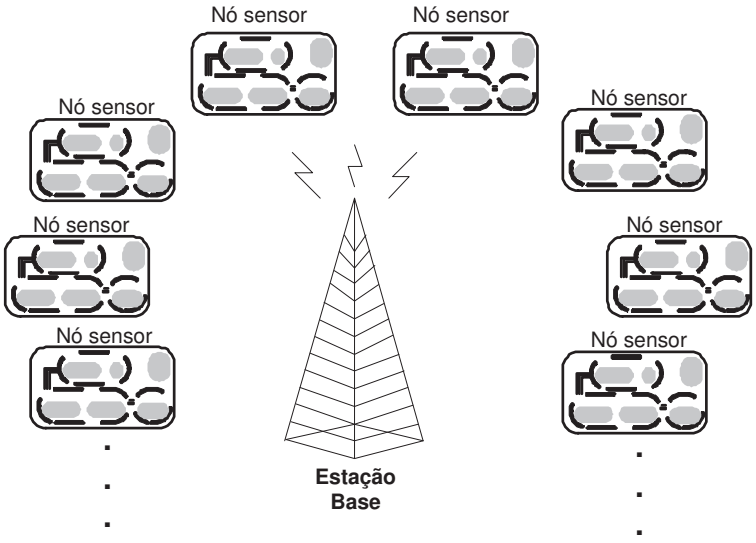

Figura 6.1: Rede de sensores sem fio

A coleta dos dados em uma rede de sensores pode ser realizada sob diferentes enfoques. Segundo Malladi e Agrawal (2002), uma rede de sensores é uma rede sem fio formada por um grande número de dispositivos usados para detectar e transmitir alguma característica física do ambiente. Os dados coletados nos sensores são agregados numa base central de dados para então serem consultados. Em outras palavras, os sensores são pré-programados para enviarem os dados coletados para um servidor de banco de dados central, permitindo que consultas *off-line* sejam realizadas. Porém, o fluxo contínuo de uma grande quantidade de dados incorre em um custo muito elevado impossibilitando, dentre outras coisas, que mais dispositivos sejam conectados a rede.

Outro enfoque sobre a coleta de dados é definido em (YAO; GEHRKE, 2003), onde uma rede de sensores é um conjunto de nós individuais que operam sozinhos, mas que podem formar uma rede com o objetivo de juntar as informações individuais de cada sensor para monitorar algum fenômeno. Consultas podem ser realizadas no próprio dispositivo, formando uma classe particular de sistemas distribuídos. Tal funcionalidade prover a redução na quantidade do fluxo de dados na rede, em conseqüência do aumento de processamento local (BONNET; SESHADRI, 2000).

As Seções seguintes estão organizadas da seguinte maneira: os nós sensores são definidos na Seção 6.2, em seguida na Seção 6.3 são apresentados os tipos de consulta permitidos na rede. Na Seção 6.4, duas abordagens distintas sobre o armazenamento e consulta de dados são descritas.

6.2 Sensor Inteligente

O interesse na utilização de redes de sensores foi impulsionado com o surgimento dos sensores inteligentes. O termo sensor inteligente é aplicado ao dispositivo que contém um ou mais sensores com capacidade de processamento de sinais e comunicação de dados (RUIZ et al., 2004). Tais dispositivos são resultantes do avanço tecnológico na área de microprocessadores, do desenvolvimento de novos materiais de sensoriamento e da possibilidade de comunicação de dados sem fio.

Na Figura 6.2 tem a ilustração de um nó sensor sem fio genérico com os seus componentes básicos, que são: *fonte de energia*, *sensor*, *ADC*, *memória*, *processador* e *transceptor*. As baterias são utilizadas como fontes de energia. Estas diferem bastante entre si quanto o consumo de energia, volume, condições de temperatura e capacidade inicial. A unidade de sensoriamento é formada pelo sensor e pelo ADC[1]. O sensor produz uma resposta para uma mudança na condição física. Esta resposta do sensor é convertida para um dado digital que pode ser processado pelo processador e armazenado na memória. A conversão é realizada pelo ADC. O processador e a memória compõem a unidade de processamento.

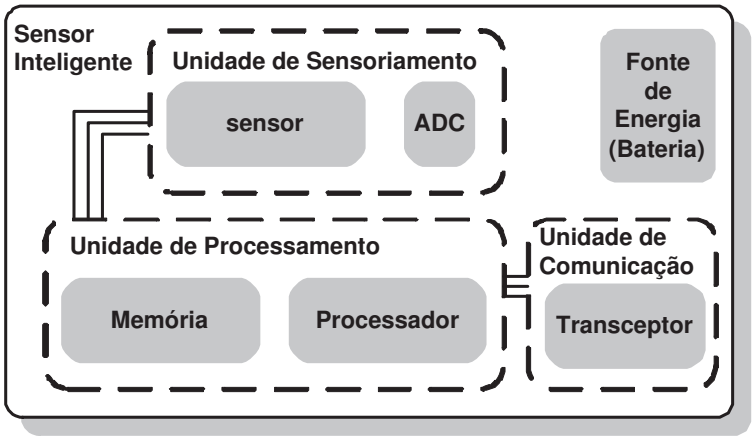

Figura 6.2: Nó sensor genérico

A comunicação de dados nas redes de sensores pode ser realizada com ou sem fio. Dentre os mecanismos para comunicação sem fio utilizados, à transmissão por radio freqüência tem sido bastante utilizada (ADELSTEIN et al., 2005). O nó sensor inteiro é encapsulado em um repositório apropriado para o ambiente no qual o nó sensor será hospedado.

As redes de sensores também podem ser formadas pelos atuadores. Estes dispositivos desempenham a função de modificar valores do meio, a fim de corrigirem falhas e controlar o objeto monitorado. Um nó sensor que possui a função de sensor e de atuador é denominado de transdutor.

[1]Acrônimo para *Analog-to-Digital Converter*.

6.3 Transações de Sensores

As transações nos nós sensores podem ser de leitura ou de escrita. Desta forma, os dados armazenados no nó podem ser lidos e estes dados podem ser transmitidos para outros nós da rede. As transações de sensores possuem rótulos de tempo e seus resultados devem ser atualizados periodicamente, já que dados de sensores se tornam rapidamente inválidos devido às restrições temporais.

Como descrito no Capítulo 3, as transações de leitura nos sensores são denominadas de *transações de leitura sensores* e são executadas pelos programas aplicativos para consultar os dados armazenados na unidade de processamento dos sensores inteligentes. As transações de escrita nos sensores, quando executadas, inserem um novo dado no sensor inteligente. Nesta pesquisa estas transações são chamadas de *transações de aquisição sensores*.

As TLS são divididas em dois tipos, tal divisão segue a classificação atribuída às transações de sensores descrita em (YAO; GEHRKE, 2003).

- *consultas instantâneas*: são consultas realizadas no próprio dispositivo em um dado instante de tempo, e;
- *consultas longas*: são consultas no próprio dispositivo durante um intervalo de tempo.

No Algoritmo 2 tem o exemplo de uma consulta instantânea. Considerando um sistema de monitoração de tráfego em uma cidade, a consulta retorna a quantidade de veículos que passaram em uma rodovia até este momento. Um registro é armazenado na tabela *veic_sensor* sempre que um veículo passa pelo sensor.

Algoritmo 2 Exemplo de uma Consulta Instantânea

```
1: SELECIONE quantidade de veículos
2: DE veic_sensor
3: DURANTE (agora)
```

Considerando o mesmo sistema de monitoração de tráfego em uma cidade, no Algoritmo 3 tem o exemplo de uma consulta longa. Durante uma hora, ou seja, três mil e seiscentos segundos, a consulta retorna a cada cem segundos à quantidade de veículos que está armazenada na tabela *veic_sensor*.

Algoritmo 3 Exemplo de uma Consulta Longa

```
1: SELECIONE quantidade de veículos
2: DE veic_sensor
3: DURANTE (agora) até (agora + 3600 segundos)
4: A CADA 100 segundos
```

Em Yao e Gehrke (2003) são definidas as cláusulas que permitem este tipo de consulta. Tais consultas são realizadas utilizando SQL, onde: *SELECIONE* equivale à *SELECT*, *DE* equivale à *FROM*, *DURANTE* à *DURATION* e *A CADA* equivale à *EVERY*.

Na arquitetura do sistema proposta nesta Tese, os dados armazenados nos sensores são enviados para o servidor de banco de dados (*transações de escrita sensores*), onde as *consultas a dados históricos* (YAO; GEHRKE, 2003) podem ser executadas. Tais consultas retornam o histórico dos dados que foram adquiridos e transmitidos pelos sensores. A necessidade de armazenar este histórico no servidor é devido à limitação de memória que existem nos sensores inteligentes.

6.4 Armazenamento dos Dados Sensores

Há duas abordagens para o armazenamento e consulta dos dados de sensores na rede. Tais abordagens são denominadas: abordagem *warehousing* e abordagem distribuída e serão descritas a seguir:

6.4.1 Abordagem *Warehousing*

A abordagem *warehousing* ainda representa o estado da arte (BONNET; GEHRKE; SESHADRI, 2001). Nela, o processamento das consultas sobre os dados extraídos dos sensores e o acesso a rede são duas etapas distintas. A rede de sensores é simplesmente usada para coletar dados. O mecanismo de consulta procede em dois passos. Primeiro, os dados são extraídos dos dispositivos de uma forma predefinida e armazenados em um servidor de banco de dados. Segundo, o processamento das consultas são realizadas no servidor. Esta abordagem é bem apropriada para execução de consultas predefinidas sobre dados históricos.

A justificativa por esta abordagem ainda ser bastante adotada é devido à capacidade limitada que os sensores tinham de apenas detectar um evento e enviar um sinal. Como as RS são compostas por centenas e até milhares de nós, a substituição destes nós por sensores inteligentes acarreta em um custo muito elevado. Porém, com o avanço no desenvolvimento dos dispositivos que, atualmente, possuem capacidade de processamento e comunicação pode-se não mais admitir o procedimento adotado.

Algumas desvantagens são visíveis nesta abordagem, tais como:

1. desperdício de valiosos recursos por transferir grandes quantidades de dados para o servidor, onde muitos destes dados são irrelevantes e

2. não é adequada para responder a consultas instantâneas e longas.

6.4.2 Abordagem Distribuída

Considerando as desvantagens citadas na abordagem *warehousing* e o aumento das funcionalidades suportadas pelos sensores, uma nova abordagem está sendo proposta para RS. Denominada de abordagem distribuída (BONNET et al., 1999) e ilustrada na Figura 6.3, esta permite que algumas consultas possam ser processadas no próprio dispositivo e, também, que o banco de dados controle os recursos que estão sendo usados. Consultas instantâneas e longas são suportadas e podem ser realizadas em alguns dispositivos em vez de terem que acessar o servidor.

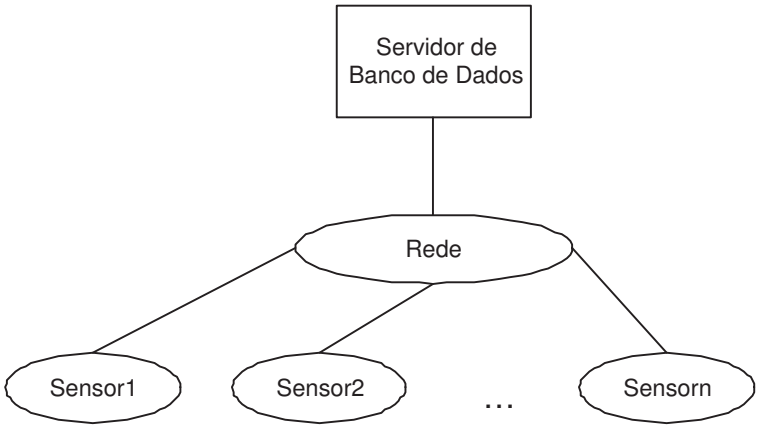

Figura 6.3: Abordagem Distribuída

Na abordagem distribuída, os nós da rede são sensores inteligentes e fazem parte do banco de dados. Cada um é considerado um banco de dados local que suporta um conjunto de funções e permite que uma certa quantidade de processamento possa ser realizada neles. Uma função suportada ou (a) adquire, armazena e processa dados ou (b) dispara uma ação no mundo físico.

As funções podem ser síncronas ou assíncronas. As primeiras são funções que retornam os resultados imediatamente após sua execução e são usadas para monitorar fenômenos contínuos, tal como uma função que retorna o nível da chuva. Nas funções assíncronas os resultados são retornados após um período arbitrário de tempo e são usadas para monitorar situações anormais, tal como função que detecta um nível elevado de chuva. Contudo, há dispositivos que podem suportar ambas as funções (síncronas e assíncronas).

Como exemplo, considere um sensor de temperatura que deve prover a funcionalidade de obter a temperatura corrente (síncrona) e disparar um alarme se esta for maior que um certo limite (assíncrona).

A abordagem distribuída é adotada neste trabalho. Porém, esta requer novas definições e técnicas, tal como o tratamento da execução concorrente das transações e o gerenciamento dos meta-dados. Este último é importante, uma vez que estes dispositivos possuem dados com semânticas idênticas, porém com sintaxes diferentes entre si.

Capítulo 7

Modelo Formal

7.1 Introdução

A finalidade de construir um modelo formal para um sistema real é definir e entender o problema, formular alternativas de solução, prever comportamentos, avaliar impactos de decisões, documentar o processo decisório, treinar não especialistas e realizar controles operacionais (SAGE, 1991).

Neste Capítulo é descrito um método formal para auxiliar na modelagem e desenvolvimento de aplicações que executam em ambientes abertos e imprevisíveis e necessitam gerenciar dados e transações com restrições de tempo-real. O modelo resultante consiste em uma representação gráfica e matemática das principais propriedades estáticas, funcionais e dinâmicas do sistema. Através do modelo, o sistema é estudado sem o perigo, custo ou inconveniência da manipulação de seus elementos. Neste caso, por se tratar de sistemas complexos, a utilização de formalismos matemáticos torna-se fundamental, pois através deles é possível submeter os modelos a procedimentos automáticos de análise, verificação e validação. Tais procedimentos têm o propósito de detectar deficiências que podem está presentes nos modelos, tais como: contradições, ambigüidades, redundância, incompletude, entre outras.

Para a validação do método, um estudo de caso considerando o domínio de aplicações em redes de sensores é realizado. Redes de Petri Coloridas Hierárquicas (HCPN) é o formalismo matemático usado para desenvolver o método. Os procedimentos automáticos de análise, verificação e validação são realizados utilizando o pacote de ferramentas computacionais e gráficas *Design/CPN*.

O restante deste Capítulo está organizado da seguinte forma: na Seção 7.2 uma arquitetura para um sistema de banco de dados em tempo-real é proposta, assim como estudo de caso considerando o domínio de aplicações em redes de sensores. Na Seção 7.3 é definido o método que é baseado na definição de dois modelos: o modelo de objetos,

apresentado na Seção 7.4, e o modelo de processos, apresentado na Seção 7.5.

7.2 Arquitetura do Sistema de Banco de Dados em Tempo-Real

A construção do método formal foi baseado na arquitetura do sistema apresentado no Capítulo 4, que tem como finalidade gerenciar dados e transações com restrições de tempo-real. Esta arquitetura é ilustrada na Figura 7.1.

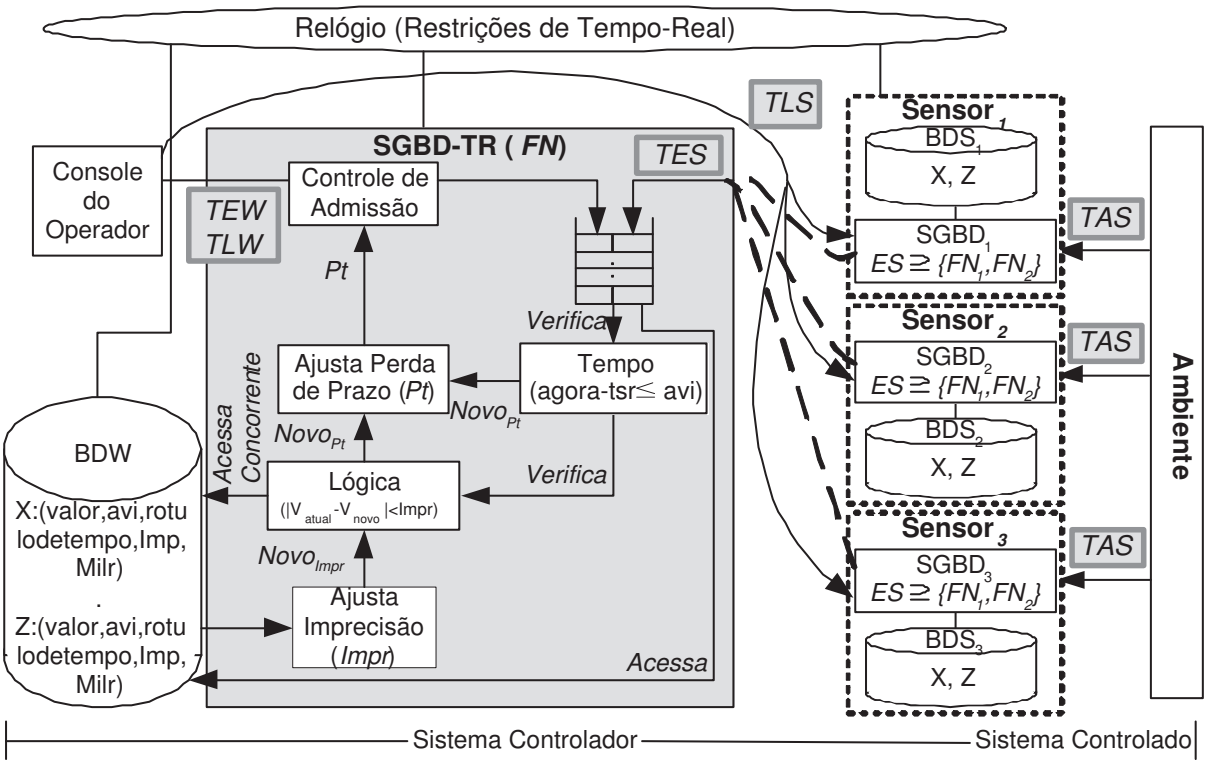

Figura 7.1: Arquitetura do Sistema de Banco de Dados em Tempo-Real

A arquitetura contempla um sistema controlador e um sistema controlado. O sistema controlador é composto pelo computador servidor, pelos sensores e pelo console do operador, enquanto que o sistema controlado representa o ambiente a ser monitorado. No computador servidor está residente o sistema de gerenciamento de banco de dados em tempo-real, onde os dados são armazenados formando um histórico da interação com o ambiente através das leituras dos sensores. O SGBD-TR consiste de um componente de software e de um repositório de dados, representados na Figura por SGBD-TR (FN) e BDW, respectivamente.

Os sensores possuem as funcionalidades de um sistema de banco de dados, sendo composto por um componente de software, representado na Figura por SGBD, e por um

repositório de dados representado por BDS. Devido a estas características os sensores são denominados de Banco de Dados Sensor local, ou sensores inteligentes. Os conteúdos do BDW e dos BDS podem ser consultados através de transações executadas a partir do console do operador.

O cenário usado como estudo de caso é um ambiente monitorado através de três sensores de temperatura. A temperatura possui um limite superior e inferior permitido. Caso este limite seja atingido ou ultrapassado, um alarme deve ser disparado. Para monitorar o ambiente os sensores adquirem os dados do ambiente e os armazenam nos sensores. Periodicamente os dados dos sensores são enviados para o SGBD-TR através das transações de sensores. O SGBD-TR é atualizado a fim de permitir que consultas históricas sejam realizadas, uma vez que não é possível armazenar todos os dados no sensor por um longo período de tempo. Os dados dos sensores são sempre mais atuais que os dados no SGBD-TR e esta diferença é devido ao atraso que existe entre a aquisição do dado pelo sensor e a atualização deste dado no servidor. Operações de leitura e escrita são permitidas tanto nos sensores como no SGBD-TR.

A consistência temporal absoluta dos dados é garantida por assegurar que o valor do dado no ambiente é próximo do seu valor armazenado nos sensores. Já a consistência temporal relativa é garantida por assegurar que os dados armazenados no SGBD-TR são consistentes em relação aos armazenados nos sensores.

7.3 Modelagem do Sistema

Para a modelagem do sistema todos os componentes da arquitetura são inicialmente identificados. O método usado para a modelagem do sistema é baseado no método definido em (PERKUSICH, 2000), e é baseado em redes de Petri. O método contempla dois modelos: o *modelo de objetos* e o *modelo de processos* que serão descritos a seguir.

7.4 Modelo de Objetos

As estruturas estáticas dos dados, as métricas de desempenho e as funções de QoS são definidas no modelo de objetos para o sistema. Cada objeto identificado é uma entidade única. Os objetos que possuem a mesma estrutura de dados (atributos) e o mesmo comportamento (operações) são agrupados em uma classe de objetos. As classes podem ter atributos, operações e métodos. Os atributos são propriedades estruturais das classes que podem ter restrições lógicas e/ou temporais. As operações são funções ou procedimentos aplicáveis aos atributos das classes e os métodos são as implementações das operações. As métricas de desempenho são definidas para possibilitar a especificação de diferentes

níveis de exigências. As funções de QoS devem garantir que as especificações das métricas de desempenho sejam atendidas.

O modelo de objetos começa com a elicitação dos requisitos e tem as seguintes etapas:

1. Identificação dos objetos e classes: os objetos identificados no modelo são os sensores representados na Figura 7.1 por $Sensor_1$, $Sensor_2$ e $Sensor_3$. Os três sensores formam a *classe sensores*. Outro objeto identificado é o SGBD-TR, representado na mesma Figura por SGBD-TR(FN) e BDW. Este objeto é comumente referido como servidor de banco de dados *warehousing*, ou simplesmente BDW.

2. Identificação dos relacionamentos entre os objetos: um sensor pode atualizar um BDW e um BDW pode ser atualizado por n sensores. Os sensores atualizam o servidor através das transações de escrita servidor (TES). Um sensor pode ser consultado por n aplicações e n aplicações podem consultar n sensores. Os sensores são consultados por transações do console do operador, denominadas de transações de leitura sensores (TLS). Os objetos sensores se relacionam ainda com o ambiente monitorado, onde os sensores são atualizados pelas transações de aquisição sensores (TAS). O servidor de banco de dados *warehousing* pode ser atualizado e consultado por n aplicações e n aplicações podem atualizar e consultar um BDW. Tais operações são executadas pelas transações de escrita servidor (TEW) e transações de leitura servidor (TLW), respectivamente.

3. Adição dos atributos das classes: o registro definido para os sensores possuem os seguintes atributos: *valor*, *avi*, *rotulodetempo* e *sensor*, onde *valor* denota o valor corrente do dado, *avi* é o intervalo de validade absoluta, *rotulodetempo* é o tempo em que o dado foi adquirido e *sensor* identifica o sensor que está executando. No BDW, o registro possui os atributos *valor*, *avi*, *rotulodetempo*, Imp e $Milr$, onde *valor* denota o valor corrente do dado, *avi* é o intervalo de validade absoluta, *rotulodetempo* é o tempo em que o dado foi adquirido, Imp especifica a imprecisão aceita no valor do item de dado e $Milr$ refere-se ao limite máximo de imprecisão permitido.

4. Uso de generalização para observar similaridades e diferenças: os objetos são identificados separadamente neste modelo, apesar das similaridades existentes.

5. Identificação das operações: nos banco de dados sensores e no banco de dados servidor são identificadas as operações de escrita e de leitura. No sensor, as operações de escrita são implementadas pelo método ATS e as operações de leitura podem ser de duas maneiras, tais como: consultas longas, implementadas pelo método CTL e consultas instantâneas, implementadas pelo método CTI. No BDW, as operações de escrita são implementadas pelo método AT e as operações de consultas históricas

são implementadas pelo método CTH. Para este objeto também é implementado o método CT com operações de escrita e leitura.

6. Identificação das operações concorrentes: as operações das transações concorrentes são definidas pelas funções de negociação. Essas possibilitam a negociação dos parâmetros lógicos pelos parâmetros temporais das transações e dos dados. Para a *classe sensores*, as funções de negociação identificadas são: FN_1 é executada quando uma transação de aquisição sensor (TAS) está executando e uma transação de leitura do servidor (TLS) é invocada para executar e; FN_2 é avaliada quando uma transação de leitura do servidor (TLS) está executando e uma transação de aquisição sensor (TAS) é invocada para executar. Para o objeto BDW são identificadas três funções de negociação. A FN_1 é usada para negociar a concorrência entre uma transação de leitura do servidor (TLW) que está executando e uma transação sensor (TES) que é invocada para executar. A FN_2 é usada para negociar quando uma transação sensor (TES) está executando e uma transação de leitura do servidor (TLW) é invocada para executar. Já a FN_3 quando executada, a negociação é realizada entre os parâmetros de uma transação de escrita do servidor (TEW) que está executando e de uma transação sensor (TES) que é invocada para executar.

7. Identificação das restrições lógicas e temporais: as restrições identificadas nos dados armazenados nos sensores estão relacionadas ao tipo do item de dado e ao intervalo de validade absoluta, dado pela expressão ($tempocorrente - rotulodetempo(x) \leq avi(x)$). As restrições definidas para o BDW são as métricas de desempenho Pt e $Impr$, além das definidas para os sensores.

A representação diagramática para o modelo de objetos é ilustrada na Figura 7.2 considerando as propriedades estáticas dos objetos, tais como: atributos, métodos, funções de negociação, restrições lógicas e restrições temporais. As métricas de desempenho e as funções de *especificação, mapeamento* e *monitoração* também são identificadas no modelo de objetos.

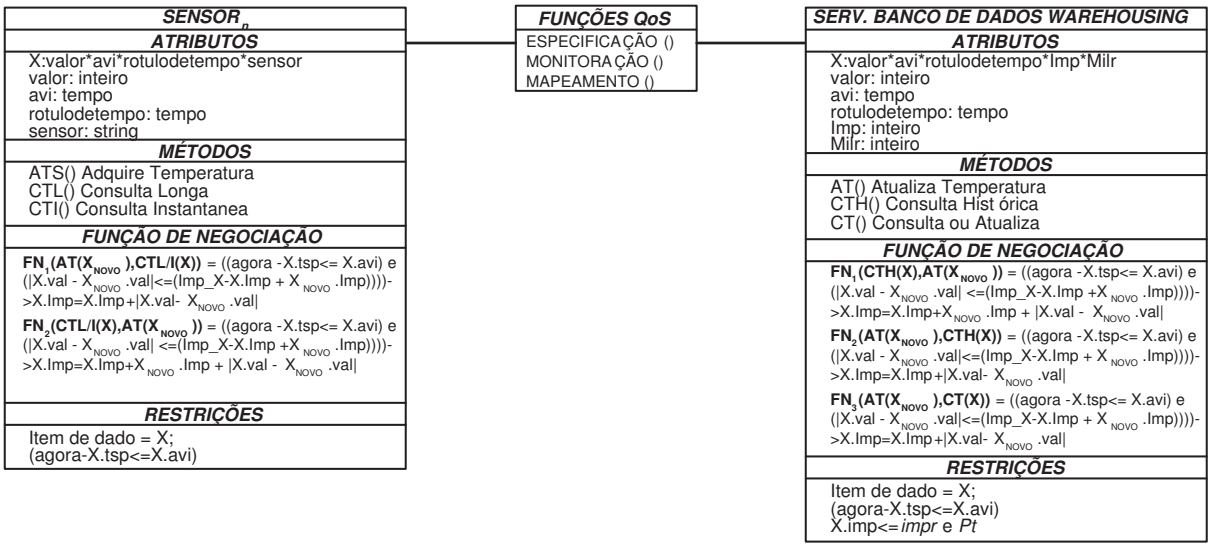

Figura 7.2: Modelo de Objetos

7.5 Modelo de Processos

No modelo de processos são identificadas as propriedades funcionais e dinâmicas dos objetos identificados no modelo de objetos. As propriedades funcionais são representadas pela modelagem das operações definidas para os objetos. As propriedades dinâmicas definem em quais condições as operações modeladas são executadas e por quanto tempo elas devem continuar executando.

7.5.1 Sistema HCPN para o Modelo de Processos

O modelo de processos é um sistema HCPN obtido através de quatro passos, como descrito a seguir:

1. Modelagem dos objetos: neste passo é construído um módulo HCPN para cada objeto identificado no modelo de objetos. Os atributos de cada objeto são considerados na construção dos módulos;

2. Modelagem das operações de cada objeto: as operações definidas para os objetos são implementadas no sistema HCPN;

3. Definição da interface de cada objeto: as interfaces dos módulos HCPN são definidas. Cada interface deve contemplar os métodos com os seus respectivos argumentos de entrada e saída e as restrições definidas para cada objeto.

4. Definição dos mecanismos de QoS: neste passo, as métricas de desempenho e as funções de QoS são implementadas em HCPN.

Nesta pesquisa, o quarto passo do modelo de processos foi modificado de forma a contemplar os mecanismos de qualidade de serviços utilizados no sistema. A definição da estrutura interna de cada objeto, quarto passo do método original, foi incorporada ao terceiro passo.

7.5.2 Tipos (Cores), variáveis e Funções do Modelo

Para simplificar a apresentação do modelo, as cores, variáveis e funções definidas para o modelo são descritas e comentadas.

```
(*******************Cor Inicializacao*******************************)
color INT = int timed; var tsr,tsr1,c:INT;

(*******************Criar Arquivo Texto*****************************)
globref outfile = TextIO.stdOut;

(*******************Cor do Relogio**********************************)
color Relogio = with e timed; fun agora():INT =
IntInf.toInt(time());

(*******************Cores, Funcoes e Variaveis dos Sensores********)
color Sensor = with S1 | S2 | S3 | L1; var s,sn,oi,or:Sensor;

(*******************Propriedades da Transacao***********************)
color Atomica = with comeca | efetiva;

(***Define o intervalo para os valores adquiridos pelos sensores***)
color ItemData = int with 5..35 declare output_col; var
item,item1:ItemData;

(****************Lista de valores adquiridos pelos sensores********)
color Lista = list ItemData; var lista:Lista;

(*******************Define o registro armazenado no sensor, bem como
uma lista destes registros. A funcao output_col'DadoTempo escreve o
valor de uma expressao especifica para um arquivo. A funcao
input_ms'DadoTempo ler um multi-set em DadoTempo de um arquivo
especifico. Se for ler um unico valor do tipo DadoTempo, deve-se
usar a funcao input_col'DadoTempo. Nao considera o sensor que leu o
item de dado*************************************************)
color DadoTempo = product ItemData * INT declare output_col;

(*******************Considera o sensor que leu o item de dado******)
color DadoTempoS = product ItemData * INT * Sensor declare
output_col; color ListDadoTempo   = list DadoTempo declare
output_col; color ListDadoTempoS = list DadoTempoS declare
output_col; var dadotempo:DadoTempo ms; var dadotempoS:DadoTempoS
ms; var listdadotempo1,listdadotempo2,listdadotempo3:ListDadoTempo;
```

```
var listdadotempoS1,listdadotempoS2,listdadotempoS3:ListDadoTempoS;

(*Funcao OK. Definir a esporadicidade da leitura de um sensor******)
color Int1 = int with 0..20; color Int2 = int with 0..1; var
d1:Int1; var d2:Int2; fun Ok(d1:Int1,d2:Int2) = (d1=d2);

(******************Funcao adiciona um item na lista, ListDadoTempo*)
fun adic(item,[]) = [item]
  | adic(item,item1::lista) =
       [item]^^([item1]^^lista);

(*****************Funcao adiciona um item na lista, ListDadoTempoS*)
fun adic1((item,tsr,s),[]) = [(item,agora(),s)]
    | adic1((item,tsr,s),(item1,tsr1,sn)::ListdadotempoS) =
       [(item,agora(),s)]^^([(item1,tsr1,sn)]^^ListdadotempoS);

(*******************Funcao retira um item da lista, ListDadoTempoS*)
fun retira([]) = 0 |
     retira((S1,item1,tsr)::lis) =
         item1;

fun retira1([]) = 0 |
     retira1((item1,tsr,s)::lis) =
         tsr;

(*Funcao que define a capacidade maxima de armazenamento do sensor.
A lista eh invertida (rev), retirado o 1o elemento dela (tl) e
depois ela eh invertida mais uma vez pra voltar a ordem original***)
fun memoria(list:ListDadoTempoS):ListDadoTempoS =
  if length(list) > 8 (*Capacidade de armazenamento*)
     then rev(tl(rev(list)))
     else list;

(******************Definicao dos Periodos dos Sensores*************)
val pe1 = 5; val pe2 = 5; val pe3 = 5;

(******************Registros dos Banco de Dados(Sensor) e Servidor*)
color ITEM = with x timed; var ps,ps1: ITEM;

(******************Define o status da transacao*******************)
color PRIOR = with critico | nulo;

(******************Define o registro armazenado no servidor********)
color DBOCELL = record
nr:ITEM*vrr:INT*vrn:INT*avir:INT*rtm:INT*impr:INT*milr:INT; var
dbnew,dbo,dboup,dbocell : DBOCELL;

(***Define o registro armazenado no servidor, composto pelo sensor*)
color DBOCELL_ = product DBOCELL*Sensor;

(*Define o registro da transacao de leitura, composto pelo item de
```

```
dado a ser lido(nrl), a imprecisao importada(impl), o limite de
imprecisao importado(mill) e o status da transacao***************)
color PRODSIMPMIL = record
nrl:ITEM*impl:INT*mill:INT*stat:PRIOR*avir:INT*rtm:INT; var
psimpmil: PRODSIMPMIL;

(*****************Define o registro da transacao de leitura,
indicando em qual sensor sera realizada a leitura*****************)
color PRODSIMPMIL_ = product PRODSIMPMIL*Sensor;

(*****************Tipos que identificam o inicio da transacao no
servidor(activate) e o fim(return) ******************************)
color OBJ = with bdcelle | bdcelat; color ACTIVATE = union
dboS:DBOCELL + psimpmilA:PRODSIMPMIL; color ACTIVATIONS_ = product
OBJ*ACTIVATE*Sensor; color RETURN = union dboP:DBOCELL +
dboR:DBOCELL + dboQ:DBOCELL; color RETURNS_ = product RETURN*Sensor;
var obj: OBJ; var ativo:ACTIVATIONS_;

(*****************Cores da Funcao de Negociacao*****************)
color ESTADO = with executando | livre; color ESTADOPRODS = product
ESTADO*ITEM; color EVALFC = with fcfalso | fcverd; color
PRODSIMPMILXEVALFC = product PRODSIMPMIL*EVALFC; color
DBOCELLXEVALFC = product DBOCELL*EVALFC; color PRODSIMPMIL_XEVALFC =
product PRODSIMPMILXEVALFC*Sensor; color DBOCELL_XEVALFC = product
DBOCELLXEVALFC*Sensor; var estado : ESTADO; var avalfc : EVALFC; var
estadoprods : ESTADOPRODS;

(*************Definicao dos Periodos. Os periodos tambem os prazos*)
val peT1 = 12; val peT2 = 12; val peT3 = 12; val peT4 = 12;

(***********Definicao dos Tempos Computacionais para as Transacoes*)
val tcT1 = 4; val tcT2 = 4; val tcT3 = 4; val tcT4 = 6;

(*****************FUNCAO DE ESPECIFICACAO***********************)
(*****************Inicializa o registro do Banco de Dados a ser
lido no sensor e atualizado no servidor**************************)
fun iniciareg(dbo:DBOCELL):DBOCELL=
    {nr=x,
    vrr = 0,
    vrn = 0,
    avir= 0,
    rtm=0,
    impr=0,
    milr=0
};

(***Inicializa o registro do Banco de Dados a ser lido no servidor*)
fun iniciaregLeit(psimpmil:PRODSIMPMIL):PRODSIMPMIL=
    {nrl=x,
    impl=0,
    mill=30,
```

```
    stat=nulo,
    avir= peT4,
    rtm=agora()
  };

(*****Retorna os valores do registro do sensor de forma individual*)
fun vlrsensor([]) = 0 |
     vlrsensor((item,tsr,s)::lis) = item;

fun rtmsensor([]) = 0 |
     rtmsensor((item,tsr,s)::lis) = tsr;

fun sensensor([]) = 0 |
     sensensor((item,tsr,s)::lis) = s;

(*************Atribui os valores atualizados armazenados no sensor*)
fun sensordb1(lis:ListDadoTempoS,dbo:DBOCELL):DBOCELL=
    {nr=x,
    vrr = vlrsensor(lis),
    vrn = 0,
    avir= peT1,
    rtm=rtmsensor(lis),
    impr=0,
    milr=10
};

(*************Atribui os valores atualizados armazenados no sensor*)
fun sensordb2(lis:ListDadoTempoS,dbo:DBOCELL):DBOCELL=
    {nr=x,
    vrr = vlrsensor(lis),
    vrn = 0,
    avir= peT2,
    rtm=rtmsensor(lis),
    impr=0,
    milr=10
};

(*************Atribui os valores atualizados armazenados no sensor*)
fun sensordb3(lis:ListDadoTempoS,dbo:DBOCELL):DBOCELL=
    {nr=x,
    vrr = vlrsensor(lis),
    vrn = 0,
    avir= peT3,
    rtm=rtmsensor(lis),
    impr=0,
    milr=10
};

(******************ARMAZENA TODOS OS REGISTROS NO SERVIDOR*********)
color LISTDBOCELL = list DBOCELL_; var listdbocell : LISTDBOCELL;
```

```
(********Adiciona uma escrita na lista de registros do servidor*)
fun adicServer((dboup,oi):DBOCELL_,[]:LISTDBOCELL):LISTDBOCELL =
     [(dboup,oi)]
     | adicServer((dbocell,or),(dboup,oi)::listdbocell) =
       [(dbocell,or)]^^([(dboup,oi)]^^listdbocell);

(******************Atualiza o rotulo de tempo (tsr) e adiciona
quantidade (m) ao registro do objeto banco de dados (dbo)**********)
fun atudbo(dbo:DBOCELL,dboup:DBOCELL):DBOCELL=
    {nr=#nr(dboup),
    vrr = #vrr(dboup),
    vrn = #vrr(dbo), (*Pega o valor antigo do dbo para um provavel restart*)
    avir= #avir(dboup),
    rtm=#rtm(dboup),
    impr=#impr(dboup),
    milr=#milr(dboup)
};

(******************Funcao q recebe o registro da atualizacao e o
registro do servidor e retorna o registro da atualizacao mantendo
seus dados temporais************************************************)
fun respdbo(dbo:DBOCELL,dboup:DBOCELL):DBOCELL=
    {nr=#nr(dboup),
    vrr = #vrr(dbo),
    vrn = #vrr(dboup),
    avir= #avir(dboup),
    rtm=#rtm(dboup),
    impr=#impr(dboup),
    milr=#milr(dboup)
};

(******************Funcao q recebe o registro de leitura e o do
servidor e retorna a leitura com os dados temporais da leitura e os
logicos do servidor*************************************************)
fun leitdbo(dbo:DBOCELL,psimpmil:PRODSIMPMIL):DBOCELL=
    {nr=#nr(dbo),
    vrr = #vrr(dbo),
    vrn = 0,
    avir= #avir(psimpmil),
    rtm=#rtm(psimpmil),
    impr=#impl(psimpmil),
    milr=#mill(psimpmil)
};

(******************IDENTIFICA QUAL TRANSACAO ESTA EXECUTANDO*******)
(******************IDENTIFICA O DEADLINE E O TEMPO COMPUTACIONAL DA
TRANSACAO QUE ESTA EXECUTANDO**************************************)
color LISTSENSOR = list Sensor; color SensorDlTc = product
Sensor*INT*INT; color LISTSensorDL = list SensorDlTc; color
ESTADOITEMSENSOR = product ESTADO*ITEM*LISTSENSOR; color
ESTADOITEMSENSOR1 = product ESTADO*ITEM*LISTSensorDL; var
```

```
sensordl:SensorDlTc; var listsensordl : LISTSensorDL; var listsensor
: LISTSENSOR;

(********Funcao adiciona um item na lista de transacoes executando*)
fun adicS(oi:Sensor,[]:LISTSENSOR):LISTSENSOR = [oi]
     | adicS(or,oi::listsensor) =
       [or]^^([oi]^^listsensor);

(********Funcao retira um item na lista de transacoes executando***)
fun retS(or:Sensor,[]:LISTSENSOR):LISTSENSOR = []
     | retS(or,oi::listsensor) =
       if oi=or
          then listsensor
          else retS(or,(listsensor^^[oi]));

(********tl(listsensor) - Retira o 1o elemento da lista************)

(********FUNCAO DEADLINE*****************************************)
fun dlAtu(dboup:DBOCELL):INT = #avir(dboup)+#rtm(dboup); fun
dlLeit(psimpmil:PRODSIMPMIL):INT =(#avir(psimpmil)+#rtm(psimpmil));

(********Retorna o o tempo computacional do sensor****************)
fun tc(oi:Sensor):INT =
    if oi=S1 then tcT1 else
    if oi=S2 then tcT2 else
    if oi=S3 then tcT3 else tcT4;

(********Funcao para adicionar um item na lista de transacoes
esxecutando com o tempo computacional e o seu deadline, o menor
deadline eh colocado na cabeca da lista**************************)
fun adicDl(oi:Sensor,dboup:DBOCELL,[]:LISTSensorDL):LISTSensorDL =
       [(oi,dlAtu(dboup),tc(oi))] |
       adicDl(oi,dboup,((oi1,dl1,tc1)::listsensordl)) =
                if dlAtu(dboup) < dl1 then
                   [(oi,dlAtu(dboup),tc(oi))]^^([(oi1,dl1,tc1)]^^listsensordl)
                   else [(oi1,dl1,tc1)]^^adicDl(oi,dboup,listsensordl);

fun
adicDl1(oi:Sensor,psimpmil:PRODSIMPMIL,[]:LISTSensorDL):LISTSensorDL
= [(oi,dlLeit(psimpmil),tc(oi))] |
       adicDl1(oi,psimpmil,((oi1,dl1,tc1)::listsensordl)) =
                if dlLeit(psimpmil) < dl1 then
                   [(oi,dlLeit(psimpmil),tc(oi))]^^([(oi1,dl1,tc1)]^^listsensordl)
                   else [(oi1,dl1,tc1)]^^adicDl1(oi,psimpmil,listsensordl);

(********Funcao para retornar somente o deadline da cabeca de
listsensordl. Como os elementos da lista sao armazenados em ordem
crescente, entao retorna o menor deadline************************)
fun dlRet([]:LISTSensorDL):INT = 0
     | dlRet((oi1,dl1,tc1)::listsensordl) = dl1;
```

```
fun dlMenor(dboup:DBOCELL,listsensordl:LISTSensorDL):INT =
     if (#avir(dboup)+#rtm(dboup))<=dlRet(listsensordl)
        then (#avir(dboup)+#rtm(dboup))
        else dlRet(listsensordl);

fun dlMenor1(psimpmil:PRODSIMPMIL,listsensordl:LISTSensorDL):INT =
     if (#avir(psimpmil)+#rtm(psimpmil))<=dlRet(listsensordl)
        then (#avir(psimpmil)+#rtm(psimpmil))
        else dlRet(listsensordl);

(********FUNCAO TEMPO COMPUTACIONAL*******************************)
fun tcSoma([]:LISTSensorDL):INT = 0+agora()
   | tcSoma((oi1,dl1,tc1)::listsensordl) =
     if listsensordl=[] then tc1+agora()
       else tc1+tcSoma(listsensordl);

(*********Funcao retira um item na lista de transacoes executando**)
fun retDl(oi:Sensor,[]:LISTSensorDL):LISTSensorDL = []
     | retDl(or,(oi,dl,tc)::listsensordl) =
       if or=oi
          then listsensordl
          else retDl(or,(listsensordl^^[(oi,dl,tc)]));

(********FUNCAO DE MONITORACAO************************************)
(********desfaz a atualizacao o rotulo de tempo (tsr) e adiciona
quantidade (m) ao registro do objeto banco de dados (dbo)**********)
fun restart(dbocell:DBOCELL,dboup:DBOCELL):DBOCELL=
    {nr=#nr(dboup),
    vrr = #vrr(dboup),
    vrn = #vrr(dboup),
    avir= #avir(dboup),
    rtm=agora(),
    impr=#impr(dboup),
    milr=#milr(dboup)
};

(****Restarta reinicializacao do valor do objeto de banco de dados*)
fun restartle(dbo:DBOCELL,psimpmil:PRODSIMPMIL):DBOCELL =
    {nr =#nr(dbo),
    vrr = 0,
    vrn = 0,
    avir= 0,
    rtm=agora(),
    impr=0,
    milr=0
};

(********METRICAS DE DESEMPENHO***********************************)
fun millpsimpmil(psimpmil:PRODSIMPMIL):int = #mill(psimpmil); fun
milrdbo(dbo:DBOCELL):int =  #milr(dbo); fun vrrdbo(dbo:DBOCELL):int
=  #vrr(dbo);
```

```
(*************FUNCAO DE NEGOCIACAO - IMP**************************)
fun metimp1(dbo:DBOCELL,dbocell:DBOCELL):int =
 if  vrrdbo(dbo)=0 then 0
   else floor(real(vrrdbo(dbo))*(real(milrdbo(dbocell))/100.0));

fun metimp2(dbo:DBOCELL,psimpmil:PRODSIMPMIL):int =
 if  vrrdbo(dbo)=0 then 0
   else floor(real(vrrdbo(dbo))*(real(millpsimpmil(psimpmil))/100.0));

val somaTc=0; fun somatTc(oi:Sensor):INT =
    if oi=S1 then tcT1 else
    if oi=S2 then tcT2 else
    if oi=S3 then tcT3 else tcT4;

(********Ajusta a imprecisao do registro da transacao de leitura***)
fun
ImprecImport(dboup:DBOCELL,dbo:DBOCELL,psimpmil:PRODSIMPMIL):PRODSIMPMIL=
    {nrl=#nrl(psimpmil),
    impl=abs(#vrr(dboup)-#vrr(dbo)), (*Retorna o valor absoluta da diferenca*)
    mill = #mill(psimpmil),
    stat = #stat(psimpmil),
    avir = # avir(psimpmil),
    rtm = #rtm(psimpmil)
};

(********Ajusta a imprecisao do registro da transacao de Escrita***)
fun
ImprecExport(dboup:DBOCELL,dbo:DBOCELL,psimpmil:PRODSIMPMIL):DBOCELL=
    {nr=#nr(dboup),
    vrr=#vrr(dboup),
    vrn = #vrn(dboup),
    avir=#avir(dboup),
    rtm=#rtm(dboup),
    impr=abs(#vrr(dboup)-#vrr(dbo)), (*Retorna o valor absoluta da diferenca*)
    milr=#milr(dboup)
};

(**********CONTROLE DE ADMISSAO**********************************)
(**********TODOS OS VALORES DEFINIDOS COMO REAL*******************)
(********O 1o inteiro eh o numero de transacoes que perderam o prazo
e o 2o eh o numero de transacoes concluidas***********************)
var transconcluidas, transperderamprazo:INT; color CORPT = product
INT*INT; var corpt : CORPT; val PtOk = 10.0; fun
inc(transconcluidas) = transconcluidas +1; fun
Pt(transperderamprazo,transconcluidas) =
     if transconcluidas = 0 then 0.0
     else 100.0*(real transperderamprazo/real transconcluidas);

(********DEFINE O INTERVALO DE QUANTO EH PARA Pt*****************)
val inter = 12; val varinter = 1;
```

```
fun intervalo(tempo,varinter)=
   if agora()<=inter*varinter andalso agora()>inter*(varinter-1)
     then inter*varinter
     else intervalo(tempo,inc(varinter));
```

7.5.3 Passo 1: Modelagem dos objetos

Os objetos identificados e modelados são: *sensor* e o *servidor de banco de dados*. Para o estudo de caso, três objetos *sensor* são modelados, resultando nos módulos HCPN smartsensor1, smartsensor2 e smartsensor3. O objeto servidor possui somente uma instância e é representado pelo módulo HCPN BDWarehousing. Os módulos HCPN são descritos a seguir.

Objeto Servidor de Banco de Dados (*BDWarehousing*)

O módulo HCPN para o servidor de banco de dados *warehousing* é ilustrado na Figura 7.3. O módulo está de acordo com o SGBD-TR mostrado na arquitetura do sistema, ilustrada na Figura 7.1.

As propriedades funcionais e dinâmicas deste módulo são descritas a seguir: no lugar *BDWarehousing* são armazenados todos os registros com todas as operações executadas, formando um histórico das operações executadas. A inserção de um novo registro no histórico ocorre sempre que a transição Atualizando dispara e é implementada pela função $adicServer((dboup, oi), listdbocell)$, onde o registro *dboup* é armazenado na lista de operações (*listdbocell*) e o parâmetro *oi* identifica o sensor. Uma leitura do histórico acontece quando a transição LendoOBD dispara.

O lugar *ServidorBD* possui o registro com os valores mais atualizados dos parâmetros. Quando uma operação de escrita está executando, então a função *atudbo(dbo,dboup)* é executada. Os argumentos de entrada referem-se ao registro com os valores correntes no servidor (*dbo*) e ao registro com os valores novos para serem atualizados (*dboup*). Esta função, como as demais descritas, estão detalhadas na Seção 7.5.2. O tempo computacional para uma operação de escrita é definido pela variável *tcT1*. Para uma operação de leitura, o tempo computacional é definido pela variável *tcT4*.

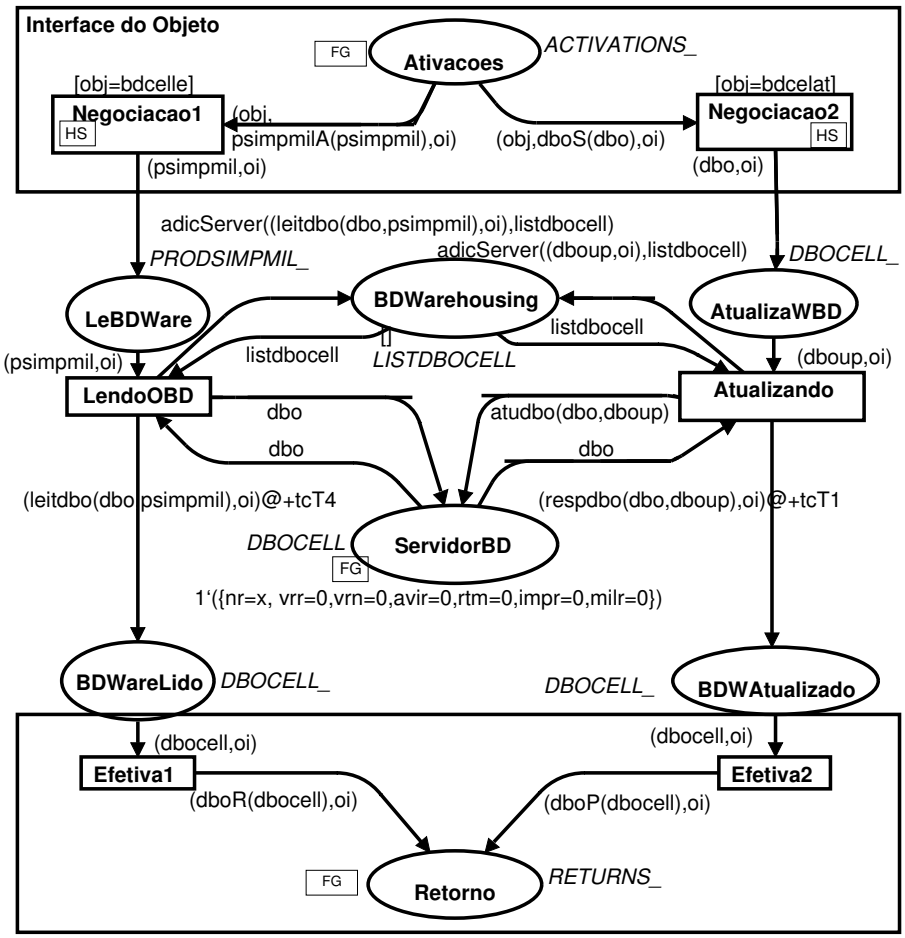

Figura 7.3: Modelo HCPN para o Servidor de Banco de Dados *Warehousing*

Os atributos definidos para os lugares *BDWarehousing* e *ServidorBD* formam o registro com os seguintes campos: *nr* é o item de dado, *vrr* é o valor do item, *vrn* é o valor antigo do item, caso a transação seja abortada este valor é atribuído ao parâmetro *vrr*, *avi* é o intervalo de validade absoluta do item de dado, *rtm* é o rótulo de tempo da operação, *impr* é a imprecisão no item de dado e, *milr* é a imprecisão acumulada para o time de dado.

Antes de acessar o banco de dados, modelado pelos lugares *BDWarehousing* e *ServidorBD*, as transações são postas em uma fila de pronto, modelada pelo lugar *Ativacoes*. Se o item de dado estiver sendo usado por uma transação conflitante, então uma função de negociação é avaliada. As funções de negociação são representadas pelas transições de substituição *Negociacao1* e *Negociacao2*. O acesso ao repositório de dados é concluído quando as transições *Efetiva1*, para uma transação de leitura, e *Efetiva2*, para uma transação de escrita, são disparadas. Contudo, as transações só são completadas quando suas respectivas funções de monitoração forem avaliadas. As funções de QoS

serão detalhadas no **Passo 4**.

Classe Sensores (Instância *smartsensor1*)

Os módulos HCPN para a classe sensores são concebidos a partir da arquitetura de um sensor inteligente, ilustrada no Capítulo 6. De agora em diante, as descrições são relacionadas à instância *smartsensor1* da classe sensores. Todas as definições são válidas para as demais instâncias.

O elemento de sensoriamento é modelado pelo lugar *Sensor1*, considerando valores aleatórios entre cinco e trinta e cinco ([5,35]). Este intervalo pode ser facilmente alterado no nó de declarações. O conversor analógico-digital é implementado pela transição *conversor*. Ao disparar, esta transição converte o valor adquirido no registro com os campos *valor*, *avi*, *rotulodetempo* e *sensor*. O tempo de conversão do sensor, isto é, o tempo que a medida obtida pelo sensor é convertida em uma medida válida para o sistema, é modelado pelo retardo na transição *conversor* de uma unidade de tempo ($@+1$). O lugar habilita11 é o relógio interno do sensor e o lugar fluxo1 define os argumentos de entrada para a função OK. Esta função implementa a propriedade de esporadicidade para aquisição de um dado pelo sensor.

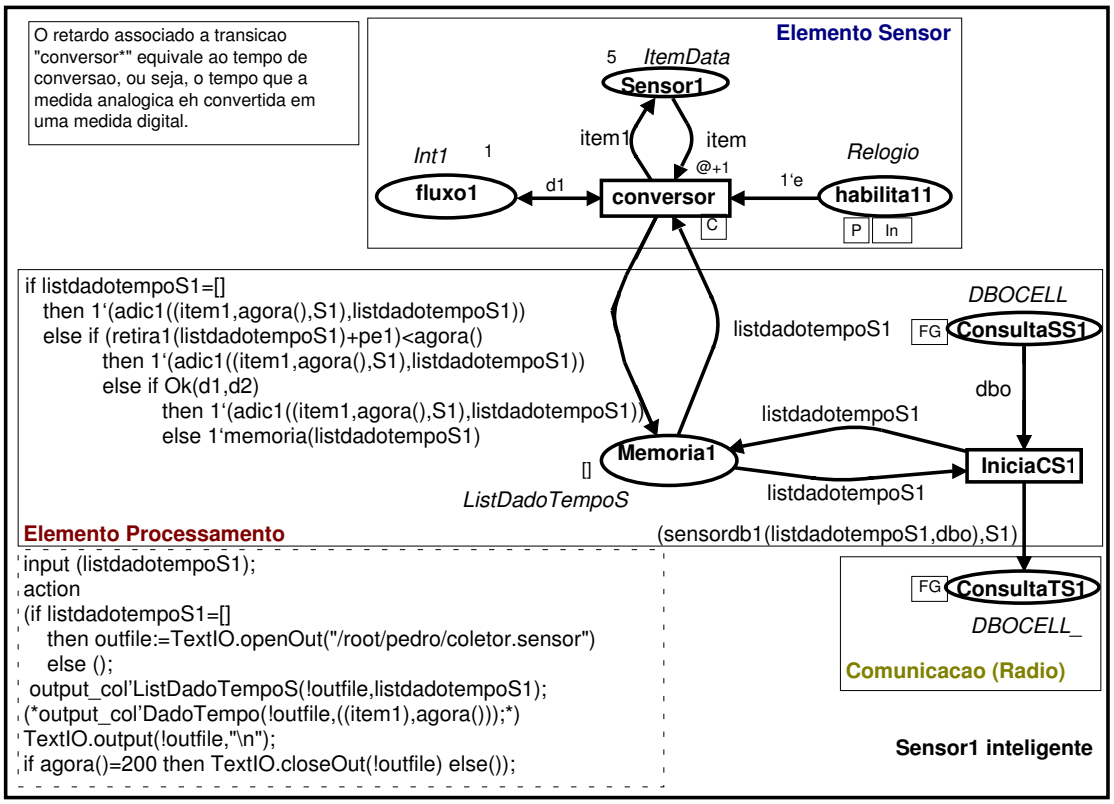

Figura 7.4: Modelo HCPN para os Sensores (Instância smartsensor1)

Um registro é inserido na memória (lugar *Memoria1*) de acordo com a inscrição do arco de entrada do lugar *Memoria1* e de saída da transição *conversor*. A inscrição do arco é obtida do algoritmo descrito a seguir.

A transição *iniciaCS1*, juntamente com os lugares *ConsultaSS1* e *ConsultaTS1*, modelam a operação de leitura do repositório de dados dos sensores. Quando esta transição dispara, a função *sensordb*1 é executada retornando o registro mais atualizado. Em seguida, estes valores são enviados através do elemento de comunicação do sensor. As linhas de código localizadas na parte de baixo da Figura 7.4, ao lado do elemento de comunicação, geram um arquivo texto com os dados armazenados no sensor. O nome do arquivo texto é *coletor.sensor* e pode ser consultado por outras aplicações. Este arquivo texto contém todos os registros armazenados no lugar *Memoria1*.

7.5.4 Passo 2: Modelagem das operações de cada objeto

Na Figura 7.5 podem ser inferidos todos os módulos HCPN necessários para a completa execução das operações definidas para os objetos.

Algoritmo 4 Inserção de um novo registro no sensor

Se Lista está vazia, ou seja, não tem nenhum registro armazenado **Então**
  Verifique o espaço disponível com a função memória e
  Adicione registro com a execução da função adic1
Senão
  Se O rótulo de tempo, obtido da função retira1, adicionado ao intervalo de validade absoluta do dado, obtido pela variável *pe*1, for menor que o tempo atual **Então**
    Verifique o espaço disponível com a função memória e
    Adicione registro com a execução da função adic1
  Senão
    Se Função Ok retornar verdadeiro **Então**
      Verifique o espaço disponível com a função memória e
      Adicione registro com a execução da função adic1
    Senão
      A função memória verifica a quantidade de registros armazenados e caso este limite seja excedido, o registro com o menor rótulo de tempo é excluído
    Fim Se
  Fim Se
Fim Se

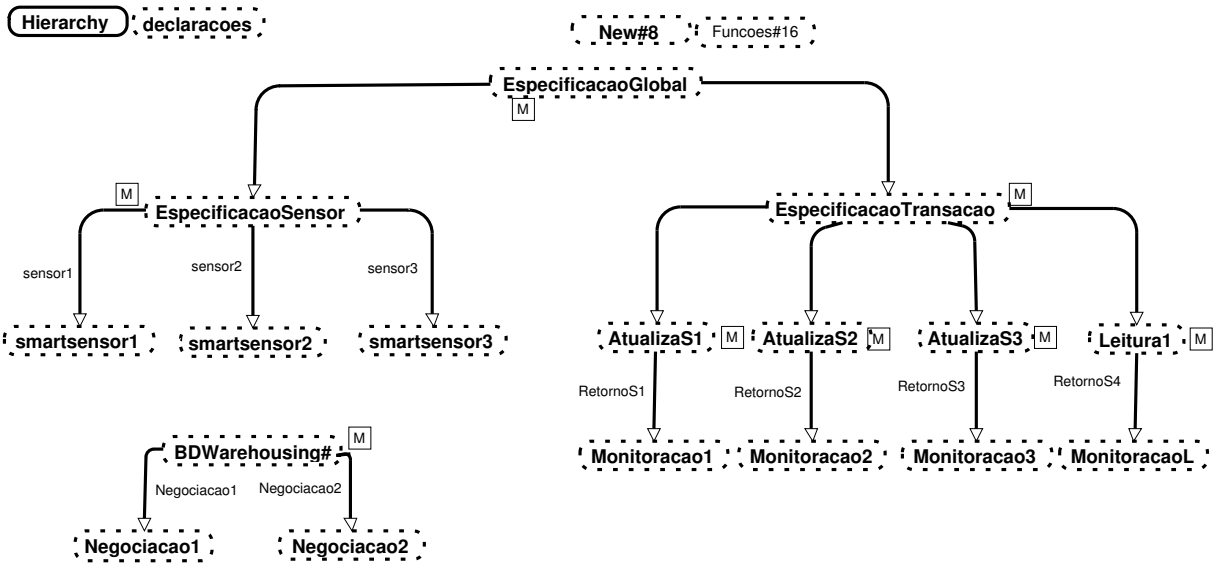

Figura 7.5: Hierarquia do Modelo

A Figura representa a estrutura hierárquica do modelo composta pelos seguintes módulos (ou páginas) HCPN:

- *declaracao*: representa o nó de declarações;
- *BDWarehousing*: representa o objeto servidor de banco de dados *warehousing*;
- *Negociacao1* e *Negociacao2*: representam as funções de negociação;

- *EspecificacaoGlobal*, *EspecificacaoSensor* e *EspecificacaoTransacao*: representam as funções de especificação;
- *smartsensor1*, *smartsensor2* e *smartsensor3*: modelam os objetos sensores;
- *AtualizaS1*, *AtualizaS2* e *AtualizaS3*: modelam as transações de sensores que atualizam o servidor de banco de dados;
- *MonitoracaoS1*, *MonitoracaoS2* e *MonitoracaoS3*: representam as funções de monitoração;
- *Leitura1* e *MonitoracaoL*: representam uma transação de atualização e a função de monitoração definida para ela;
- *Funções*: página de controle.

O objeto servidor de banco de dados é apresentado na Figura 7.3 e representado pelo módulo HCPN *BDWarehousing*. O conteúdo do repositório de dados é formado pelos lugares *ServidorBD*, que possui o registro atual, e *BDWarehousing*, que tem o histórico das operações executadas. A operação de escrita é realizada pelo método AT e a operação de leitura é implementada pelo método CTH. O método AT é invocado pela execução dos módulos *AtualizaS1*, *AtualizaS2* e *AtualizaS3* e o método CTH é invocado pelo módulo *Leitura1*. Para cada um destes módulos é desenvolvida uma função de monitoração QoS (*MonitoracaoS1*, *MonitoracaoS2*, *MonitoracaoS3* e *MonitoracaoL*) para verificar se o método foi executado atendendo todas as restrições temporais e lógicas. Tais restrições são especificadas pela função de especificação QoS, representada pelos módulos HCPN *EspecificacaoGlobal*, *EspecificacaoSensor* e *EspecificacaoTransacao*. Para o objeto servidor de banco de dados também são definidas as funções de negociação QoS para avaliar a execução concorrente dos métodos. Estas funções são representadas pelos módulos *Negociacao1* e *Negociacao2*.

O objeto sensor, representado pelos módulos HCPN *smartsensor1*, *smartsensor2* e *smartsensor3* e ilustrado na Figura 7.4, adquire um valor do ambiente, converte este valor em um tipo registro e armazena-o no repositório de dados local (lugar *Memoria1*). A operação de escrita é implementada pelo método ATS. A invocação deste método é realizada automática e periodicamente de acordo com as especificações do sensor. A operação de leitura no objeto sensor é implementada pelos métodos CTL e CTI, onde

consultas longas e consultas instantâneas são realizadas, respectivamente. A invocação destes métodos é realizada pela execução dos módulos *AtualizaS1*, *AtualizaS2* e *AtualizaS3*.

7.5.5 Passo 3: Definição da interface de cada objeto

No módulo HCPN do servidor de banco de dados *warehousing*, representado na Figura 7.3, o método AT é modelado pelos lugares *AtualizaWBD* e *BDWAtualizado* e pela transição *Atualizando*. O método CTH é representado pelos lugares *LerBDWare*, *BDWareLido* e pela transição *LendoOBD*. Por fim, o método CT que pode efetuar uma operação de escrita (método AT) ou de leitura (método CTH) no repositório de dados. Um item de dado é considerado válido se somente se o seu intervalo de validade absoluta somado do tempo em que este dado foi gravado for menor que o tempo atual do sistema.

Os métodos definidos para o módulo HCPN da classe sensores, representado na Figura 7.4, são: ATS, CTI e CTL. O método ATS é modelado pelos lugares *Sensor1* e *Memoria1*, e pela transição *conversor*. Quando este método é invocado, uma operação de escrita é executada no repositório de dados local (lugar *Memoria1*). Os métodos CTI e CTL são implementados pelos lugares *ConsultaSS1* e *ConsultaTS1*, e pela transição *IniciaCS*. A diferença entre os métodos é que o método CTI implementa uma consulta instantânea, ou seja, em um determinado momento. O método CTL implementa uma consulta longa, onde é especificado um intervalo de tempo e durante este intervalo várias consultas periódicas são realizadas. A restrição temporal para o item de dado é semelhante a descrita no objeto servidor de banco de dados.

7.5.6 Passo 4: Definição dos mecanismos de QoS

Função de Especificação

Como definido no Capítulo 4, através da *Função de Especificação* são identificados os parâmetros de tempo e os parâmetros lógicos necessários para garantir o perfeito funcionamento do sistema. Dentre os parâmetros temporais identificados tem-se a periodicidade, o prazo e o tempo de liberação de uma transação. Entre os parâmetros lógicos estão o valor, a imprecisão admitida e a imprecisão máxima permitida para o item de dado. Alguns destes parâmetros são obtidos pelas métricas de desempenho.

A função de especificação é dividida no sistema em três módulos HCPN. Um módulo refere-se ao relógio do sistema (*EspecificacaoGlobal*), o outro se refere à especificação das transações (*EspecificacaoTransacao*) e o último se refere à especificação dos sensores (*EspecificacaoSensor*). Nas Seções seguintes os dois últimos módulos serão descritos.

Módulo HCPN para Especificação das Transações

Os parâmetros temporais para as transações do sistema são modelados pelo módulo HCPN para especificação das transações, ilustrado na Figura 7.6. O tempo para as transações é obtido pelo lugar Relogio das Transacoes e pela transição RelogioT Local das Transacoes que formam o relógio das transações. Este relógio é sincronizado ao relógio do sistema, modelado pelo módulo HCPN *EspecificacaoGlobal*. O tempo para as transações é alimentado em um sempre que a transição do relógio é disparada.

Os parâmetros de tempo das transações serão descritos a seguir considerando apenas o módulo *AtualizaS1*. Tais definições são idênticas para os módulos *AtualizaS2*, *AtualizaS3* e *Leitura1*. Os tempos para as transações são obtidos pelos lugares *ConsultaS1* e *ConsultaS1I* e pela transição AcessaS1. A periodicidade da transação é dada pela variável peT1 e o prazo para a transação ser concluída é igual ao período. O tempo de liberação é a diferença entre o período e a marcação inicial do lugar ConsultaS1 mais um, ou seja, $(peT1 - marcacaoinicial + 1)$. O um é o retardo atribuído à transição. Por exemplo, considerando que a marcação inicial é igual a nove e que o período é igual a doze, então o tempo de liberação da transação será $(12 - 9) + 1 = 4$.

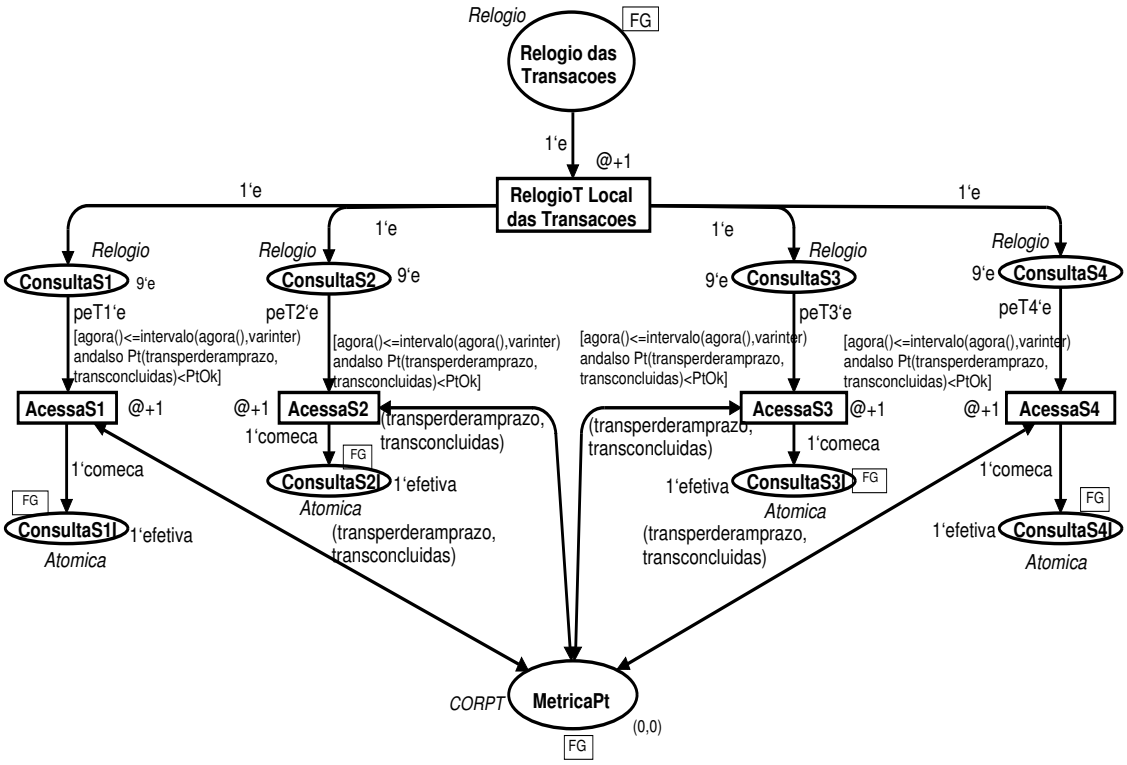

Figura 7.6: Módulo HCPN para Especificação das Transações

As guardas das transições AcessaS1, AcessaS2, AcessaS3 e AcessaS4 e o lugar MetricaPt serão explicados quando as métricas de desempenho forem descritas.

Quando o tempo de liberação de uma transação é encontrado, então uma ficha é adicionada ao lugar *ConsultaS1I* que é um lugar de fusão com *ConsultaS11*, no módulo HCPN *AtualizaS1*, ilustrado na Figura 7.7. A marcação inicial para este lugar é a ficha 1‘$efetiva$ e quando a ficha 1‘*comeca* é adicionada, a transição *IniciaConsS1* fica habilitada para disparar. Esta dinâmica modela o início da execução de uma transação e de acordo com a sua operação, invocará um dos métodos definidos para os objetos. A função iniciareg(dbo), na inscrição do arco de entrada do lugar *ConsultaRS1*, inicializa o registro do banco de dados a ser lido no sensor e atualizado no servidor. Na inicialização, os parâmetros lógicos das transações são especificados.

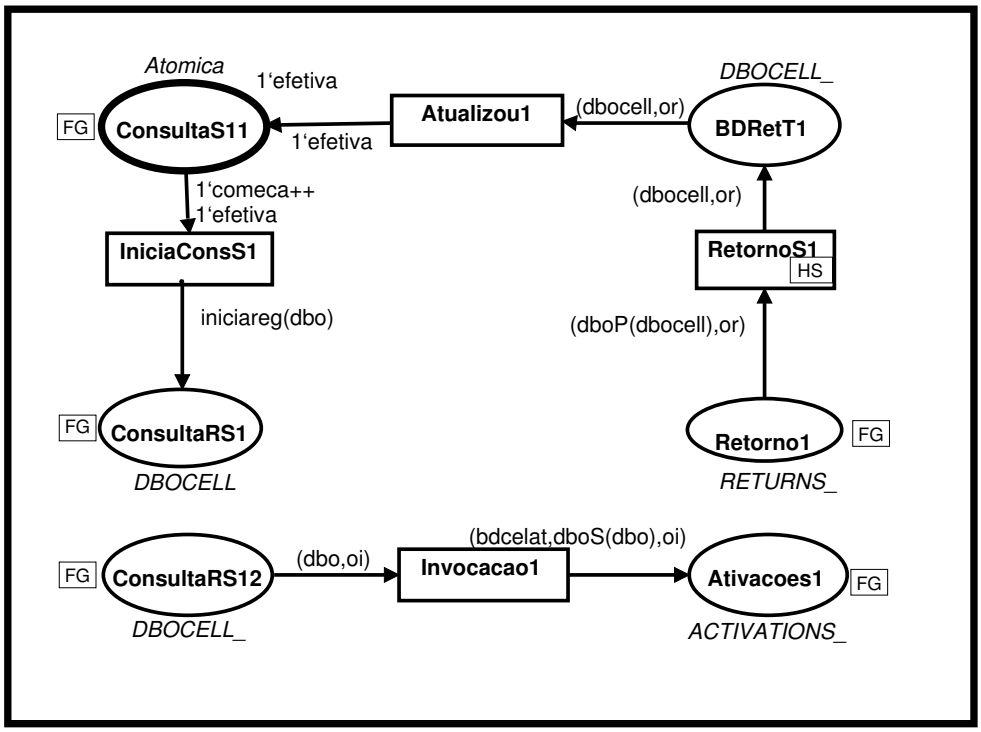

Figura 7.7: Modelo HCPN para as Transações

Como ilustrado na Figura 7.8, o lugar *ConsultaRS1* é um lugar de fusão com o lugar *ConsultaSS1* ilustrado no módulo HCPN *smartsensor1*. Quando uma ficha é adicionada a *ConsultaSS1*, a transição *iniciaCS1* dispara atribuindo ao registro os valores armazenados no sensor. Isso é feito pela função *sensordb1* que é a inscrição do arco de entrada do lugar *ConsultaTS1*, ver Figura 7.4. O lugar *ConsultaTS1* é um lugar de fusão com *ConsultaRS12*. Uma vez o registro atualizado, a transição *Invocacao1* pode disparar. Ao ocorrer, uma ficha é adicionada ao lugar *Ativacoes1* que é um lugar de fusão com o lugar *Ativacoes* no módulo HCPN BDWarehousing, definido para o objeto servidor de banco de dados.

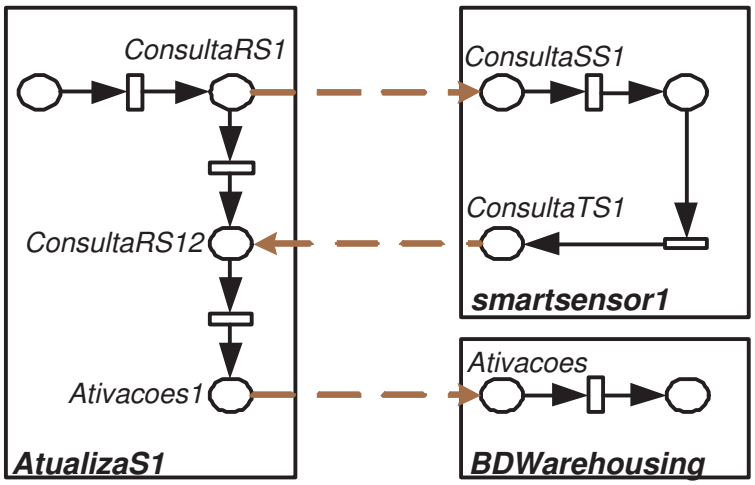

Figura 7.8: Lugares de Fusão entre os Módulos HCPN

Uma transação completa quando a transição *Atualizou1* retornar a ficha $1`efetiva$ ao lugar *ConsultaS11*. Contudo, antes disso, uma ficha deve ser colocada no lugar *Retorno1*, habilitando a transição de substituição *RetornoS1*. Esta modela a função de monitoração, onde é verificada se a especificação inicial para a transação foi satisfeita.

Módulo HCPN para Especificação dos Sensores

Os parâmetros temporais para os sensores são modelados no módulo HCPN para especificação dos sensores, ilustrado na Figura 7.9. O tempo para os sensores é obtido pelo lugar Relogio dos Sensores e pela transição Relogio Local dos Sensores que constituem o relógio dos sensores. Este relógio é sincronizado ao relógio do sistema, modelado pelo módulo HCPN *EspecificacaoGlobal*.

O lugar habilitaS1 e a transição de substituição sensor1 são incluídos para cada sensor do sistema, neste caso específico para o sensor um. Cada transição de substituição equivale a um módulo HCPN para os sensores.

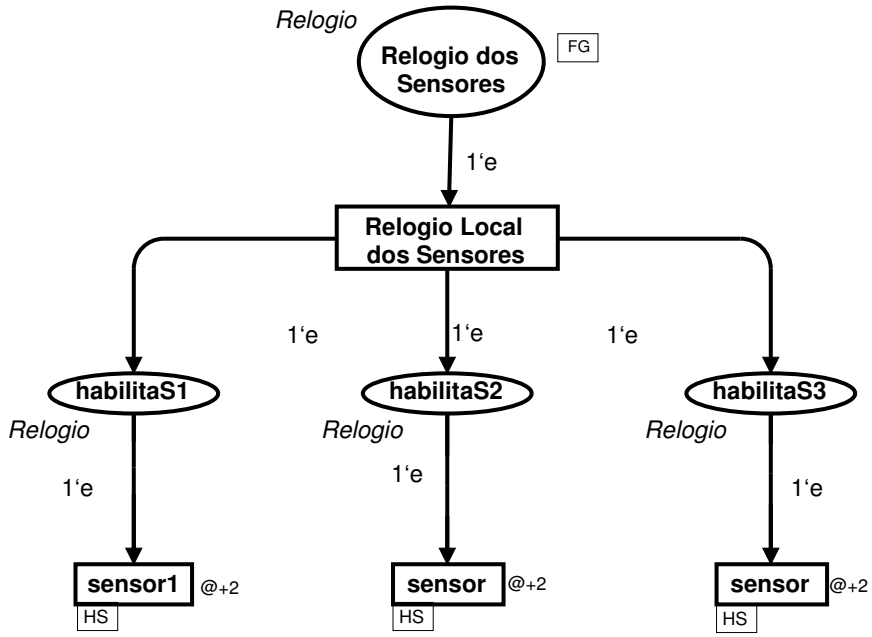

Figura 7.9: Módulo HCPN para Especificação dos Sensores

O tempo associado a cada transição de substituição, representado por $@ + 2$, modela o tempo de resposta do sensor. O tempo de resposta do sensor é o tempo que o sensor necessita para se adequar às condições do ambiente.

Função de mapeamento

Para as transações do modelo, os parâmetros temporais e lógicos são mapeados através do tipo (ou cor) DBOCELL, onde o item de dado, o valor, a validade absoluta, o rótulo de tempo, a imprecisão permitida e a imprecisão máxima acumulada são especificados.

Funções de Negociação

Como identificado no modelo de objetos, para o objeto servidor de banco de dados são identificadas três funções de negociação. A FN_1 é usada para negociar entre os parâmetros lógicos e temporais quando uma transação de leitura do servidor (TLW) está executando e uma transação sensor (TES) é invocada para executar. A FN_2 é usada para negociar os parâmetros quando uma transação sensor (TES) está executando e uma transação de leitura do servidor (TLW) é invocada para executar. A FN_3 quando executada, negocia os parâmetros temporais e lógicos de uma transação de escrita do servidor (TEW) que está executando e de uma transação sensor (TES) que é invocada para executar. Para a classe sensores, as duas primeiras funções de negociação são utilizadas.

Módulo HCPN *Negociação1*

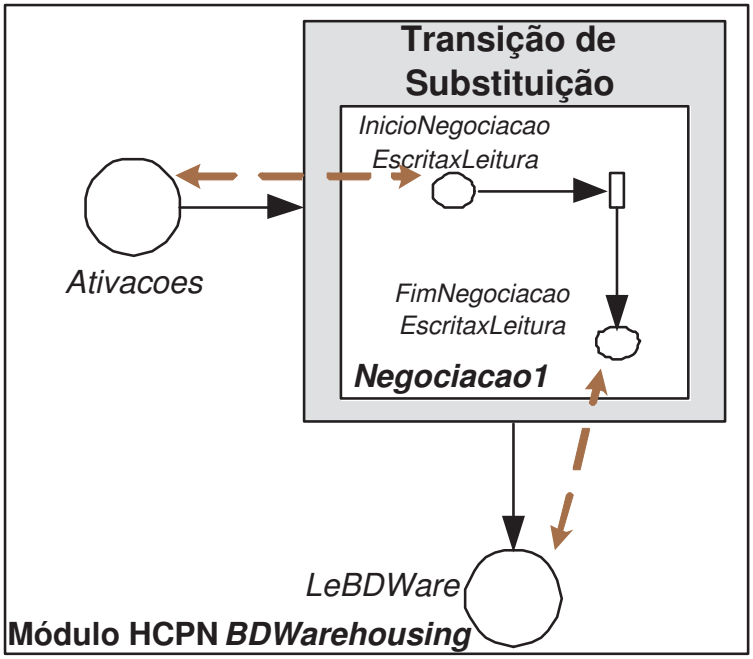

Figura 7.10: Transição de Substituição *Negociação1*

O módulo HCPN *Negociacao1* é representado por uma transição de substituição no módulo HCPN BDWarehousing, onde os lugares Ativacoes e LeBDWare são os *sockets* na superpágina e os lugares InicioNegociacaoEscritaxLeitura e FimNegociacaoEscritaxLeitura são as portas de entrada e de saída na subpágina, respectivamente. Tal como ilustrado na Figura 7.10.

Na Figura 7.11 é ilustrado o modelo para a transição de substituição *Negociacao1*. Existem dois fluxos possíveis de execução na subpágina, como definido a seguir:

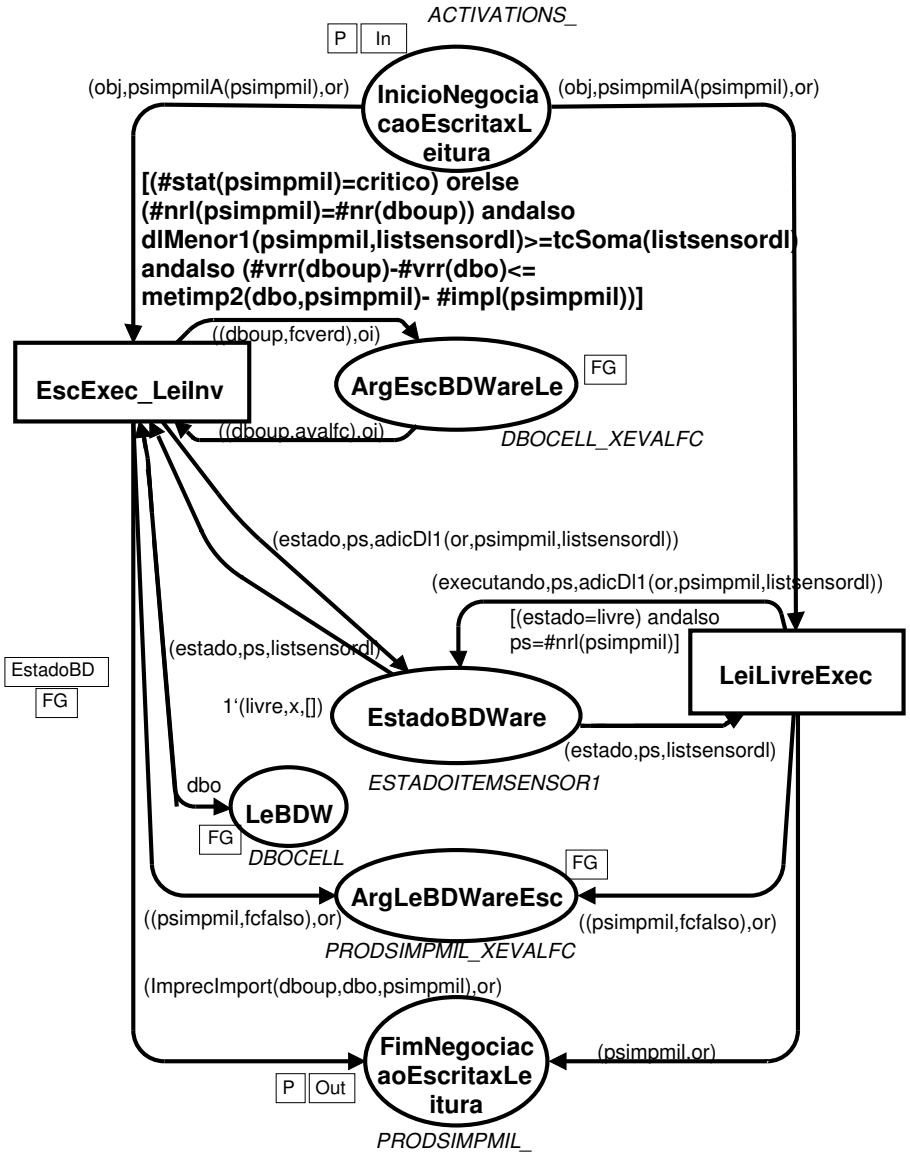

Figura 7.11: Função de Negociação - TLW é invocada e existe pelo menos uma TES executando

- TLW é invocada e não existe nenhuma outra transação executando, conseqüentemente não existe concorrência. Nesse caso, quando uma TLW é invocada, uma ficha é adicionada ao lugar InicioNegociacaoEscritaxLeitura. Se nenhuma outra transação conflitante estiver executando para o mesmo item de dado, então a transição LeiLivreExec fica habilitada. Ao disparar, a ficha do lugar EstadoBDWare, inicializada em $1‘(livre, x, [])$, é modificada para $1‘(executando, x, [sensor, (avi+rotulodetempo), tc])$, onde o primeiro parâmetro especifica que tem transação executando, o segundo especifica qual item de dado a transação está manipulando e o terceiro é uma lista que identifica qual sensor está executando, com seus respectivos prazo e tempo com-

putacional. Quando a transação for completada a ficha será reinicializada, a menos que exista uma outra transação executando. Em seguida, uma ficha é adicionada ao lugar FimNegociacaoEscritaxLeitura. Sempre que uma transação de leitura é executada o estado do lugar ArgLeBDWareEsc é alterado.

- TLW é invocada e existe pelo menos uma TES executando (FN_2). Nesse caso, quando uma TLW é invocada, uma ficha é adicionada ao lugar InicioNegociacaoEscritaxLeitura. Se pelo menos uma TES estiver executando, então à transição LeiLivreExec não fica habilitada, devido à guarda $[(estado = livre)]$. A transição EscExec_LeiInv, cuja guarda representa a FN_2, pode ser avaliada. Considerando os valores dos parâmetros da TES, da TLW, do objeto servidor de banco de dados e do sistema, a função pode ser avaliada em verdadeira ou falsa. Se for avaliada em verdadeira a transição poderá ocorrer, permitindo a execução concorrente. Caso contrário, o acesso concorrente não é permitido. FN_2 possibilita avaliar o acesso concorrente de mais de duas transações, isto é, quando uma TES estiver executando, várias TLW podem executar concorrentemente. Para isso ser possível é necessário que o menor prazo para conclusão das transações seja maior ou igual à soma dos tempos computacionais das transações. Tal funcionalidade é implementada na FN_2 pela expressão $dlMenor1(psimpmil, listsensordl) >= tcSoma(listSensordl)$, onde a função $dlMenor1(psimpmil, listsensordl)$ seleciona o menor prazo de uma lista e a função $tcSoma(listSensordl)$ soma os tempos computacionais. Estes dados são obtidos do lugar EstadoBDWare. Os parâmetros do objeto servidor de banco de dados são obtidos pelo o lugar LeBDW e os parâmetros da TES são obtidos pelo lugar ArgEscBDWareLe.

Módulo HCPN *Negociação2*

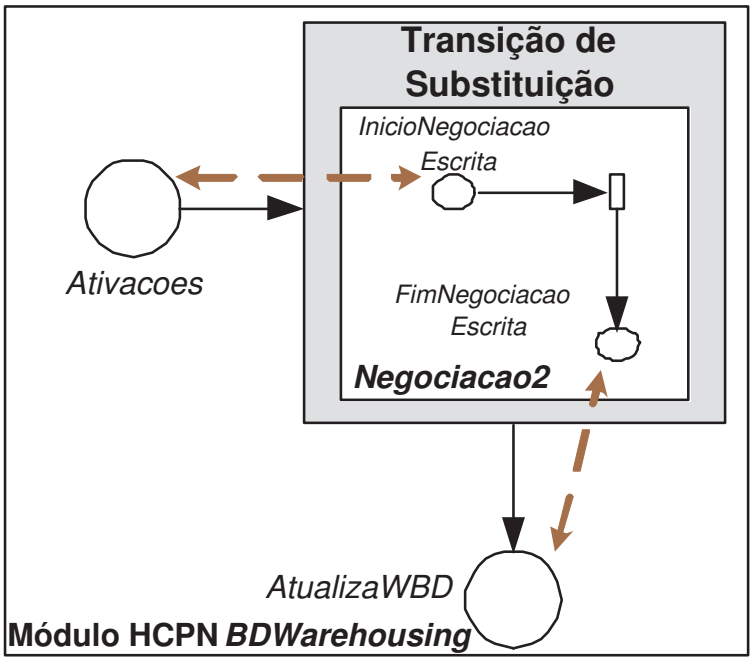

Figura 7.12: Transição de Substituição *Negociação2*

O módulo HCPN *Negociacao2* é representado por uma transição de substituição no módulo HCPN BDWarehousing, onde os lugares Ativacoes e AtualizaWBD são os *sockets* na superpágina e os lugares InicioNegociacaoEscrita e FimNegociacaoEscrita são as portas de entrada e de saída na subpágina, respectivamente. Tal como ilustrado na Figura 7.12.

Na Figura 7.13 é ilustrado o modelo para a transição de substituição *Negociacao2*. Existem três fluxos possíveis de execução na subpágina, como definido a seguir:

- *TES* é invocada e não existe nenhuma outra transação executando, conseqüentemente não existirá concorrência. Nesse caso, a transição EscLivreExec fica habilitada quando a ficha é adicionada ao lugar InicioNegociacaoEscrita. Ao disparar, a ficha do lugar EstadoBDWare, inicializada em 1‘(livre,x,[]), é modificada para 1‘(executando,x,[sensor,(avi+rotulodetempo),tc]), onde o primeiro parâmetro especifica que tem transação executando, o segundo especifica qual item de dado a transação está manipulando e o terceiro é uma lista que identifica qual sensor está executando, com seus respectivos prazo e tempo computacional. Quando a transação for completada a ficha será reinicializada, a menos que exista uma outra transação executando. Em seguida, uma ficha é adicionada ao lugar FimNegociacaoEscrita. Sempre que uma transação de escrita é executada o estado do lugar Arg0EscBDWareLe é alterado.

- TES é invocada e existe pelo menos uma TLW executando (FN_1). Nesse caso, se pelo menos uma TLW está executando, então a transição EscLivreExec não fica habilitada, devido à guarda $[(estado = livre)]$. A transição LeiExec_EscInv, cuja guarda representa a FN_1, pode ser avaliada. Considerando os valores dos parâmetros da TES, da TLW, do objeto servidor de banco de dados e do sistema, a função pode ser avaliada em verdadeira ou falsa. Se for avaliada em verdadeira a transição poderá ocorrer, permitindo a execução concorrente das transações. Caso contrário, o acesso concorrente não é permitido. FN_1 possibilita avaliar o acesso concorrente de mais de duas transações, isto é, quando uma TLW estiver executando, várias TES podem executar concorrentemente. Para isso ser possível é necessário que o menor prazo para conclusão das transações executando seja maior ou igual a soma dos tempos computacionais das transações. Tal funcionalidade é implementada na FN_1 pela expressão $dlMenor(dboup, listsensordl) >= tcSoma(listSensordl) + tc(or)$, onde a função $dlMenor(dboup, listsensordl)$ seleciona o menor prazo de uma lista e a função $tcSoma(listSensordl) + tc(or)$ soma os tempos computacionais. Estes dados são obtidos do lugar Estado0BDWare. Os parâmetros do objeto servidor de banco de dados são obtidos pelo o lugar LeBDW1 e os parâmetros da TLW são obtidos pelo lugar Arg0LeBDWareEsc.

- TES é invocada e existe pelo menos uma TEW executando, ou uma TEW é invocada e existe pelo menos uma TES executando (FN_3). Nesse caso, se pelo menos uma transação de escrita está executando, então a transição EscLivreExec não fica habilitada, devido à guarda [(estado=livre)]. A transição EscExec_EscInv, cuja guarda representa a FN_3, pode ser avaliada. Considerando os valores dos parâmetros da TES, da TEW, do servidor de banco de dados e do sistema, a função pode ser avaliada em verdadeira ou falsa. As propriedades dinâmicas para FN_3 são semelhantes as propriedades descritas no item anterior para a FN_1. Os parâmetros do objeto servidor de banco de dados são obtidos pelo o lugar LeBDW1 e os parâmetros da outra transação de escrita são obtidos pelo lugar Arg0EscBDWareLe.

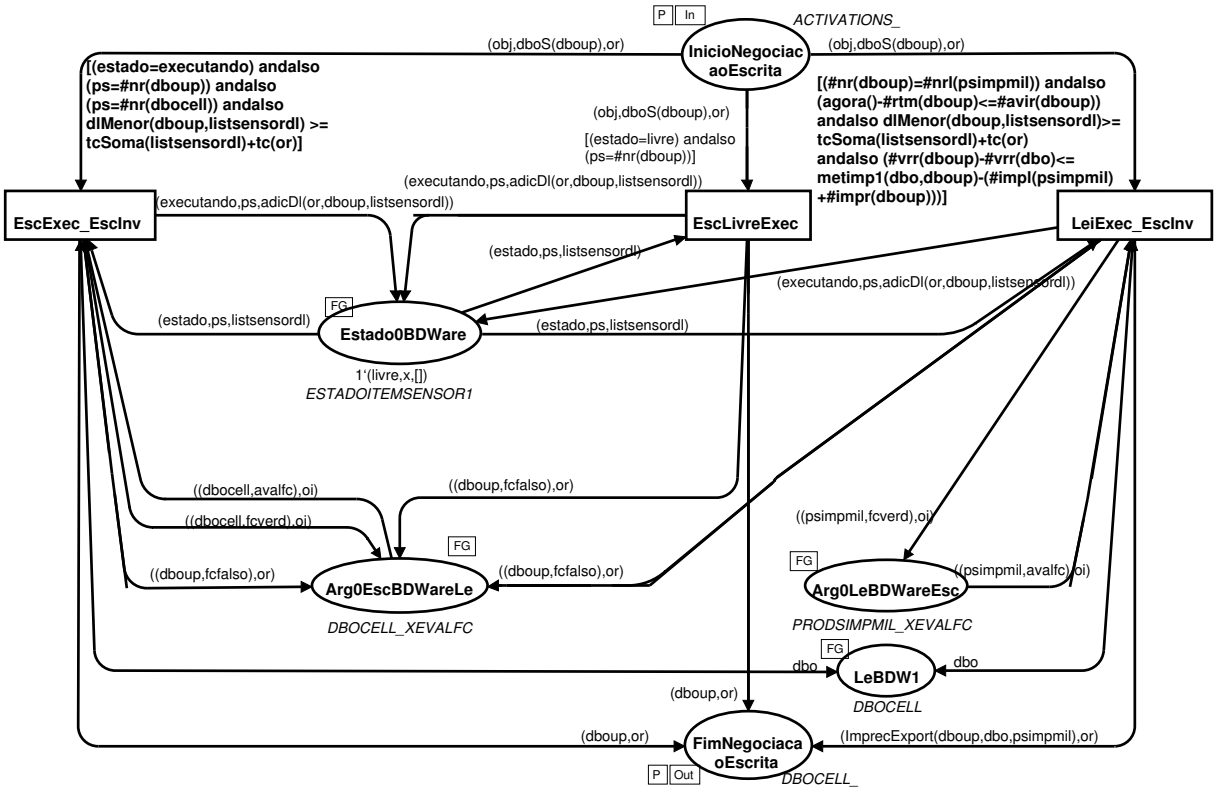

Figura 7.13: Função de Negociação - TES é invocada e existe pelo menos uma TLW ou uma TEW executando.

Função de Monitoração

Os módulos HCPN *MonitoracaoS1*, *MonitoracaoS2*, *MonitoracaoS3* e *MonitoracaoL* representam a função de monitoração QoS para o sistema. Nestes módulos são verificados se os parâmetros de tempo especificados para as transações e para os dados são atendidos. Um dos módulos, no caso *MonitoracaoS1*, é ilustrado Figura 7.14.

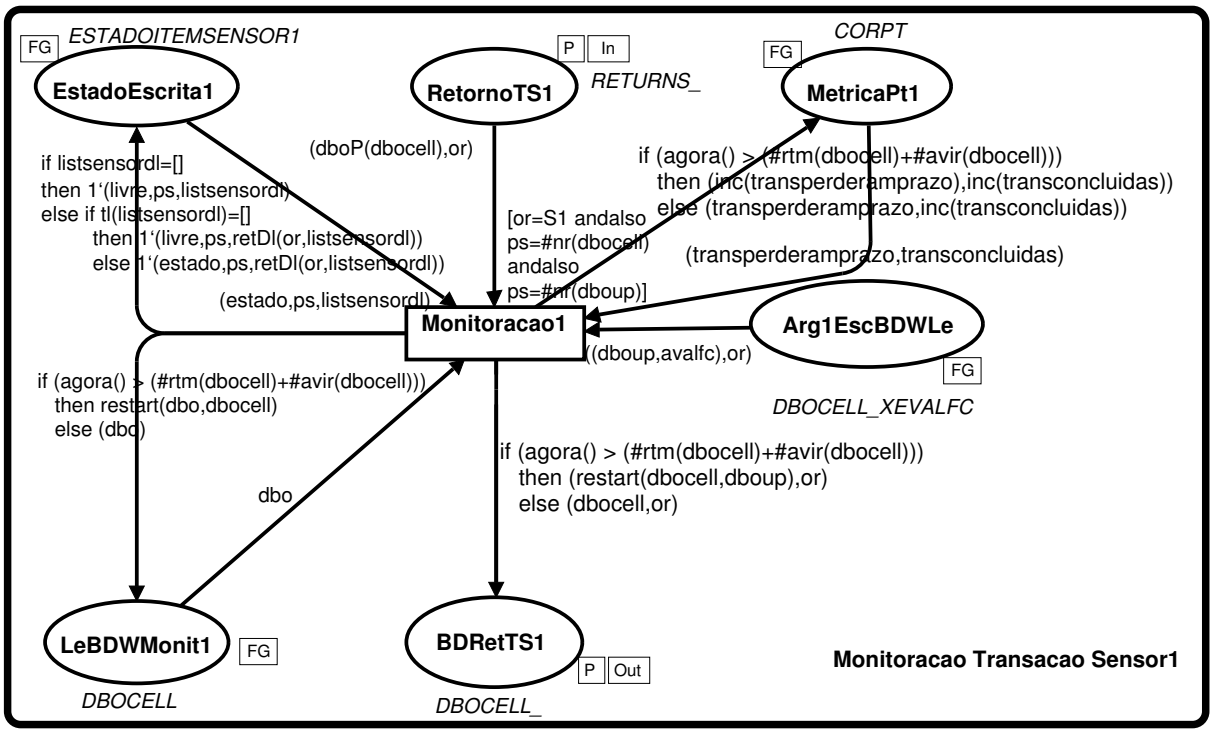

Figura 7.14: Modelo HCPN para a Função de Monitoração

O módulo é uma subpágina da transição de substituição *RetornoS1* do módulo HCPN *AtualizaS1*, tal como ilustrado na Figura 7.15. Os lugares *Retorno1* e *BDRet1* são os *sockets* na superpágina e os lugares *RetornoTS1* e *BDRetTS1* são as portas de entrada e de saída na subpágina, respectivamente.

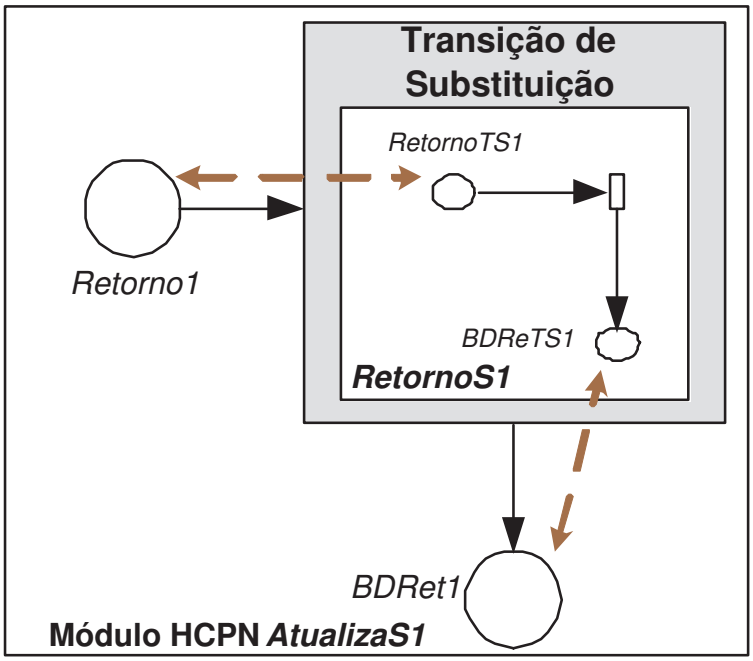

Figura 7.15: Transição de Substituição *RetornoS1*

O lugar EstadoEscrita1 é um lugar de fusão com o lugar EstadoBDWare, descrito para as funções de negociação, contendo a lista das transações que estão executando. Quando uma transação é completada, suas referências são retiradas da lista de acordo com o algoritmo a seguir.

Algoritmo 5 Retira uma transação da lista de transações executando

```
1: Se Lista está vazia, ou seja, não tem nenhum registro armazenado Então
2:    Retorne a ficha 1‘(livre,x,[])
3: Senão
4:    Se Lista só tem um elemento. A função tl retira o primeiro elemento da lista Então
5:       Retorne a ficha 1‘(livre,x,[])
6:    Senão
7:       Retire a transação da lista e retorne a ficha com a referência das demais transações
         executando
8:    Fim Se
9: Fim Se
```

O lugar LeBDWMonit1 é um lugar de fusão com o lugar ServidorBD, descrito no módulo HCPN *BDWarehousing*, onde a operação da transação é desfeita, caso ela seja abortada. O lugar Arg1EscBDWLe é um lugar de fusão com Arg0EscBDWLe. O lugar MetricaPt1 será descrito mais adiante. A inscrição de arco entre a transição Monitoracao1 e o lugar BDRetTS1, é uma expressão condicional para verificar se as restrições de tempo especificadas foram atendidas. Quando a transação é completada ou abortada uma ficha é adicionada ao lugar BDRetTS1.

Métricas de Desempenho

As métricas de desempenho definidas no Capítulo 4 são consideradas no modelo e descritas a seguir.

1. Métrica *Pt*: equivale a quantidade de transações que perderam o prazo final em relação à quantidade de transações que concluíram seus prazos com sucesso.

   No modelo formal esta métrica é implementada nos módulos HCPN *Especificacao-Transacao*, *MonitoracaoS1*, *MonitoracaoS2*, *MonitoracaoS3* e *MonitoracaoL*. Nas páginas de monitoração é modelado o lugar denominado MetricaPt1, por exemplo, em *MonitoracaoS1*, que incrementa a variável transconcluidas sempre que uma transação é concluída com sucesso. Caso a transação seja abortada, então a variável transperderamprazo também é incrementada. O incremento é realizado por executar a função

inc. O lugar MetricaPt1 é um lugar de fusão com MetricaPt no módulo *Especificacao-Transacao*.

Considerando os valores das variáveis transconcluidas e transperderamprazo, o controle de admissão é realizado, onde uma nova transação só é admitida se a quantidade de transações que perderam o prazo final em relação à quantidade de transações que concluíram estiver dentro do aceitável, durante um determinado intervalo de tempo. O controle de admissão é implementado na guarda das transições *AcessaS1*, *AcessaS2*, *AcessaS3* e *AcessaS4*, no módulo HCPN *EspecificacaoTransacao*. A expressão $agora() <= intervalo(agora(), varinter)$ define o intervalo de tempo e $Pt(transperderamprazo, transconcluidas) < PtOk$ define a relação entre as transações abortadas e concluídas.

2. Métrica $Impr$: equivale à imprecisão máxima admitida no item de dado para ele ainda ser considerado válido logicamente.

   A métrica é implementada no modelo formal pelos módulos HCPN *Negociacao1* e *Negociacao2*. Na guarda da transição *LeiExec_EscInv* no módulo *Negociacao2*, a expressão $metimp1(dbo, dboup)$ é identificada. Esta função recebe como argumentos, o valor do item de dado armazenado no servidor, dado pela ficha *dbo*, e a porcentagem de imprecisão permitida pela transação de leitura do servidor (TLW), dada por *dboup*. Por exemplo, é definido que é permitida uma imprecisão de vinte por cento no valor do dado. Então, considerando o valor do dado no servidor igual a trinta, se o valor a ser atualizado pela transação sensor (TES) estiver dentro do intervalo fechado de vinte e quatro a trinta e seis ($[24, 36]$), TES poderá executar.

   Na guarda da transição *EscExec_LeiInv*, no módulo *Negociacao1*, tem a expressão $metimp2(dbo, psimpmil)$. Esta função recebe como argumentos o valor do item de dado armazenado no servidor, dado pela ficha *dbo*, e a porcentagem de imprecisão permitida pela transação de leitura do servidor (TLW), dada por *psimpmil*. Então, especificada a porcentagem de imprecisão, é verificado se as transações podem executar concorrentemente.

   Para a FN_3, na guarda da transição *EscExec_EscInv* no módulo *Negociacao2*, esta métrica não é implementada. Esta decisão foi tomada considerando a existência de apenas transações com operações de escrita, onde em nenhum momento será

lido algum valor para posteriores tomadas de decisões. Neste caso, a execução concorrente será decidida pela avaliação das restrições temporais.

Capítulo 8

Uma Abordagem de Verificação e Validação Formal

Neste Capítulo é definida uma abordagem para verificação e validação formal de bancos de dados em tempo-real. Tal abordagem visa assegurar que o método, definido no Capítulo 7, está correto e atende as necessidades do usuário.

A abordagem formal está de acordo com a Figura 8.1 e consiste na aplicação de cinco etapas:

1. Construir um modelo de objetos;
2. Construir um modelo de processos;
3. Gerar o espaço de estados do modelo, através do grafo de ocorrência, a fim de verificar se as escalas geradas atendem ao critério de corretude serialização *Epsilon* considerando os mecanismos de qualidade de serviços;
4. Gerar o diagrama de seqüência de mensagens para validar o *workflow* definido para um determinado cenário, e;
5. Gerar o diagrama de tempo para validar as restrições temporais atribuídas às transações do servidor e dos sensores.

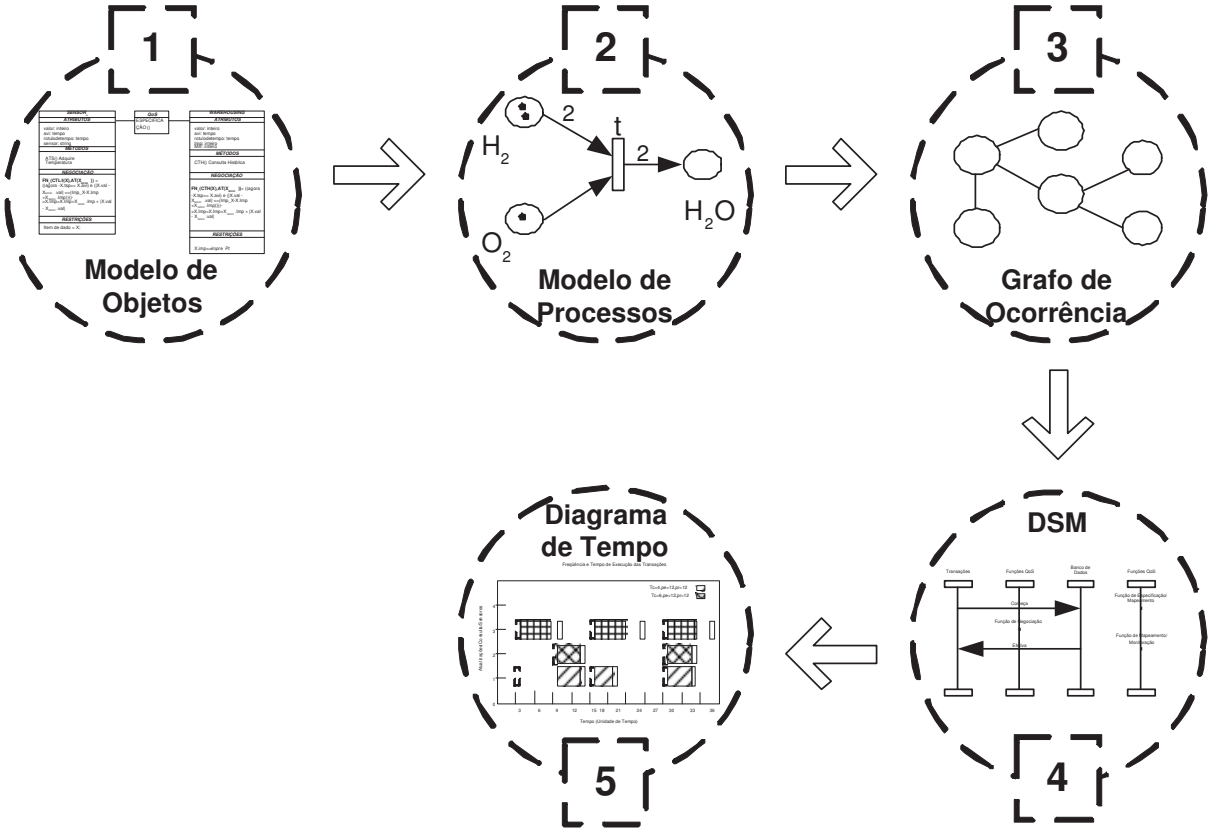

Figura 8.1: Abordagem de Verificação e Validação Formal

8.1 Introdução

As aplicações que utilizam bancos de dados em tempo-real são consideradas corretas se as transações são executadas em tempo hábil e a consistência lógica dos dados é mantida. Para isso, técnicas de controle de concorrência e políticas de escalonamento são utilizadas. As primeiras devem manter a corretude lógica do sistema e as políticas de escalonamento devem gerenciar as escalas de execução das operações das transações, a fim de garantir que as restrições temporais sejam atendidas. Técnicas de computação imprecisa (AMIRIJOO, 2003) também podem ser utilizadas para estes sistemas, onde resultados imprecisos podem ser permitidos em conseqüência da necessidade de atender o tempo de resposta das tarefas.

Exigências não funcionais, tais como desempenho, qualidade e precisão, são definidas para os STR e devem ser consideradas durante o desenvolvimento destes softwares. Por conseguinte, para assegurar a qualidade dos STR é necessário garantir que tais propriedades sejam corretamente especificadas. A correta concepção e especificação das exigências não funcionais inerentes aos STR podem ser obtidas através do desenvolvi-

mento de modelos formais. Uma vez que com tais modelos é possível definir e entender o problema, de formular alternativas de solução, de prever comportamentos e de avaliar impactos de decisões.

Contudo, análises estáticas de modelos formais, definidas para sistemas de bancos de dados em tempo-real, não são suficientes para verificar o comportamento temporal destes. A verificação e validação de tais sistemas podem ser obtidas eficientemente por técnicas de simulação do modelo. Com a simulação do modelo, uma amostra aleatória[1] será selecionada de um domínio de entrada[2] do objeto a ser testado e então simulada com os valores de entrada selecionados. Após isto, os resultados obtidos desta execução são comparados com os valores esperados. Assim, simulação do modelo é uma técnica dinâmica, isto é, uma técnica que contempla a execução dos objetos a serem testados e permite verificar e validar o comportamento do sistema.

Ainda é pequena a quantidade de trabalhos voltados para modelagem e simulação de sistemas de bancos de dados em tempo-real. Tal escassez é devido à limitação das ferramentas de modelagem de processos existentes em não suportar simulação. A linguagem de modelagem unificada (UML) apresenta várias características para modelagem de sistemas de tempo-real, como descrito em (DOUGLASS, 2004), no entanto, não possuem ferramentas com capacidade de simulação.

Então como já mencionado anteriormente, neste Capítulo é definida uma abordagem de verificação e validação formal para bancos de dados em tempo-real. A aplicação desta abordagem começa com a elicitação dos requisitos e a definição do modelo de objetos. Em seguida, com base no modelo de objetos é definido o modelo de processos. Na primeira etapa ocorre a identificação dos objetos e das classes, de seus relacionamentos, adição dos atributos das classes, identificação das operações, bem como das operações concorrentes e a identificação das restrições lógicas e temporais os objetos. Na segunda etapa ocorre a modelagem dos objetos, modelagem das operações de cada objeto, definição da interface de cada objeto e a definição dos mecanismos de QoS. Estas etapas já foram descritas no Capítulo anterior.

Nas Seções seguintes serão definidas as demais etapas que compõem a abordagem, tais como a geração do espaço de estados, dos diagramas de seqüência de mensagens e dos diagramas de tempo. Estas três etapas compõem a análise do modelo e propicia a

[1]Tal amostra pode ser um cenário.

[2]O domínio de entrada pode ser o conjunto de todos os cenários.

verificação e validação deste. A análise do modelo é realizada considerando o domínio de aplicações redes de sensores e os procedimentos para execução destas três etapas são realizados automaticamente utilizando o pacote de ferramentas computacionais e gráficas *Design/CPN*, descrito no Capítulo 5.

8.2 Grafo de Ocorrência

Grafos de ocorrência (OG)[3] são grafos direcionados que possuem um nó para cada marcação alcançável e um arco para cada elemento de ligação (*binding element*). Um arco liga o nó da marcação, na qual o elemento de ligação associado ocorre, ao nó da marcação resultante da ocorrência (JENSEN, 1997). A ferramenta grafo de ocorrência é totalmente integrada com o *Design/CPN*. A idéia principal do grafo de ocorrência é mostrar todos os estados existentes no modelo e as ligações existentes entre estes. Cada estado é uma marcação alcançada e os relacionamentos entre estas marcações são chamados de elementos de ligação.

Várias consultas padrões são permitidas no OG, tais como:

- *Reachable*: permite determinar se há uma seqüência de ocorrências entre duas marcações específicas.
- *DeadMarking*: determina se a marcação de um nó específico é morta, isto é, não tem mais elementos de ligação habilitado.
- *ListDeadMarking*: retorna uma lista com todos os nós que são marcações mortas.

Estas consultas podem ser usadas para investigar todas as propriedades padrões de uma HCPN. Em adição as consultas padrões, existe a possibilidade da formulação de outras consultas. Tais consultas são implementadas em código ML.

Através do grafo de ocorrência é possível verificar propriedades particulares do modelo. A ferramenta de grafo de ocorrência do *Design/CPN* possibilita a emissão de um relatório com as propriedades gerais do modelo. Este relatório contém informações sobre o grafo e sobre metapropriedades que são úteis para compreensão do comportamento do modelo em HCPN. Como exemplo de metapropriedades, pode-se citar:

[3]OG é derivado do termo em inglês *Occurrence Graph*.

- Propriedades de limite (*boundedness properties*) que fornecem os limites máximo e mínimo de fichas que cada lugar da rede pode conter, além da maior e da menor marcação para cada lugar;

- propriedades de vivacidade (*liveness properties*) que dizem quais marcações e transições são mortas (finais), ou seja, que não levam a nenhuma outra marcação e quais transições são vivas, ou seja, que aparecem em alguma seqüência de ocorrência iniciada da marcação inicial da rede.

A geração do grafo de ocorrência termina quando todas marcações são geradas e a partir delas ou as marcações já foram alcançadas ou não tem mais marcações alcançáveis. O OG pode ser construído considerando segmentos de código e tempo. No caso de ser temporizado, cada nó representa um estado do sistema que contém um valor de tempo associado e uma marcação temporizada, desta forma nenhuma marcação será igual a outra. Devido à isto e ao comportamento periódico de uma aplicação de tempo-real, o grafo de ocorrência precisa ser limitado através de um critério de parada determinado. O critério de parada, nesta pesquisa, é definido pelo algoritmo de escalonamento *Rate Monotonic* (LIU, 2000), onde o escalonamento é considerado válido se o número de vezes que a transação executar é menor ou igual ao mínimo múltiplo comum (MMC) dos valores para os períodos dividido pelo período da transação, ou seja:

$$N \leq \frac{X * MMCPer}{Período}$$

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.

Uma outra importância fundamental no uso desses grafos está na possibilidade de verificar todas as possíveis seqüências de execução das transações e determinar quais atendem as especificações lógicas e temporais impostas às transações e aos dados.

8.3 Diagrama de seqüência de Mensagens

Diagrama de seqüência de mensagens (DSM) é uma forma gráfica para representar as interações entre os componentes do sistema.

Na Figura 8.2, a representação do diagrama de seqüência de mensagens utilizada nesta pesquisa é apresentada. Esse contempla as funções de qualidade de serviços, as transações do servidor e dos sensores, o conteúdo do servidor de banco de dados e os conteúdos dos sensores. Também é possível verificar a execução das transações.

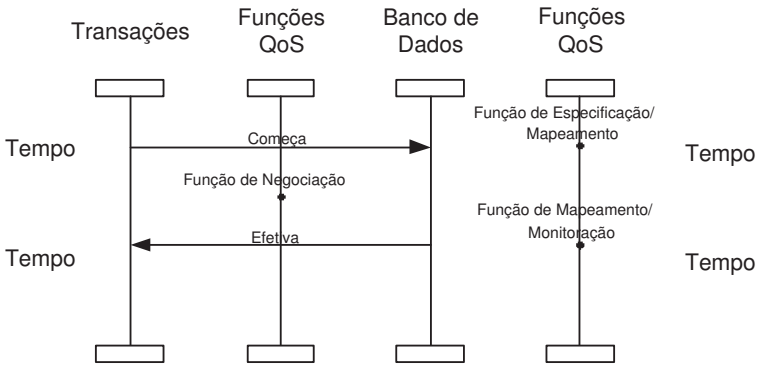

Figura 8.2: Diagrama de Seqüência de Mensagens

Nesta abordagem, o uso do DSM é fundamental, uma vez que através dele é possível verificar a execução das transações, a negociação entre elas no caso de uma execução concorrente, além do conteúdo dos repositórios de dados. Também é possível validar o comportamento dos objeto e seus relacionamentos. Para gerar o DSM, a biblioteca *smc.sml*, disponível em (DAIMI, 2004b), do *Design/CPN* é utilizada.

8.4 Diagrama de Tempo

A análise de desempenho é baseada em análises de dados extraídos do modelo HCPN durante a simulação. Estes dados podem ser usados para investigar o desempenho do sistema, como por exemplo, o tempo máximo permitido para a execução de uma transação ou a periodicidade com que o sensor adquiriu e armazenou um dado.

Na Figura 8.3, um diagrama de tempo é ilustrado. No eixo vertical, as transações dos sensores e de atualizações são representadas. No eixo horizontal são ilustrados os tempos de liberação e os períodos para as transações e para os sensores, além dos tempos computacionais e dos prazos para as transações. Uma transação de atualização/sensores possui três estados: início, processamento e efetivação. O início da transação é indicado por um retângulo com linhas pontilhadas. O processamento é indicado por um retângulo sombreado e a efetivação por um retângulo com linhas contínuas. A aquisição dos dados

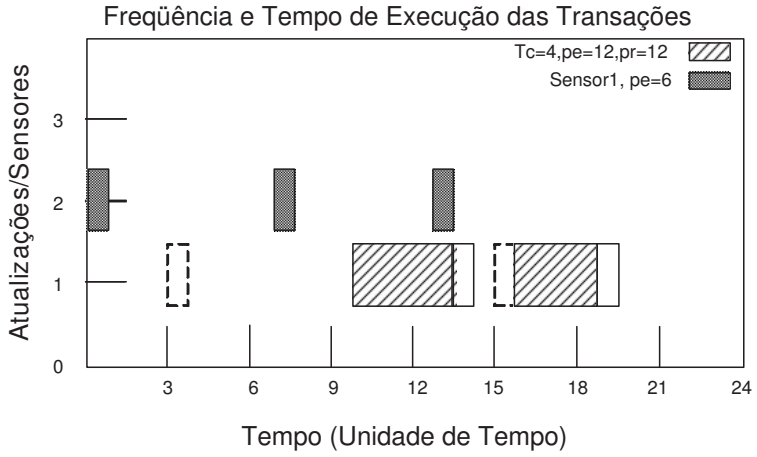

Figura 8.3: Diagrama de Tempo

por sensores é mostrada por um retângulo preenchido identificando o instante de tempo dessa ação.

8.5 Análise do Modelo

Alguns cenários são definidos para a análise do modelo. Através dos cenários, várias situações para utilização do sistema de banco de dados em tempo-real proposto nesta Tese são consideradas. A definição dos cenários é baseada na arquitetura do sistema apresentada no Capítulo 4, que tem como finalidade gerenciar dados e transações com restrições de tempo-real. Esta arquitetura é ilustrada na Figura 8.4.

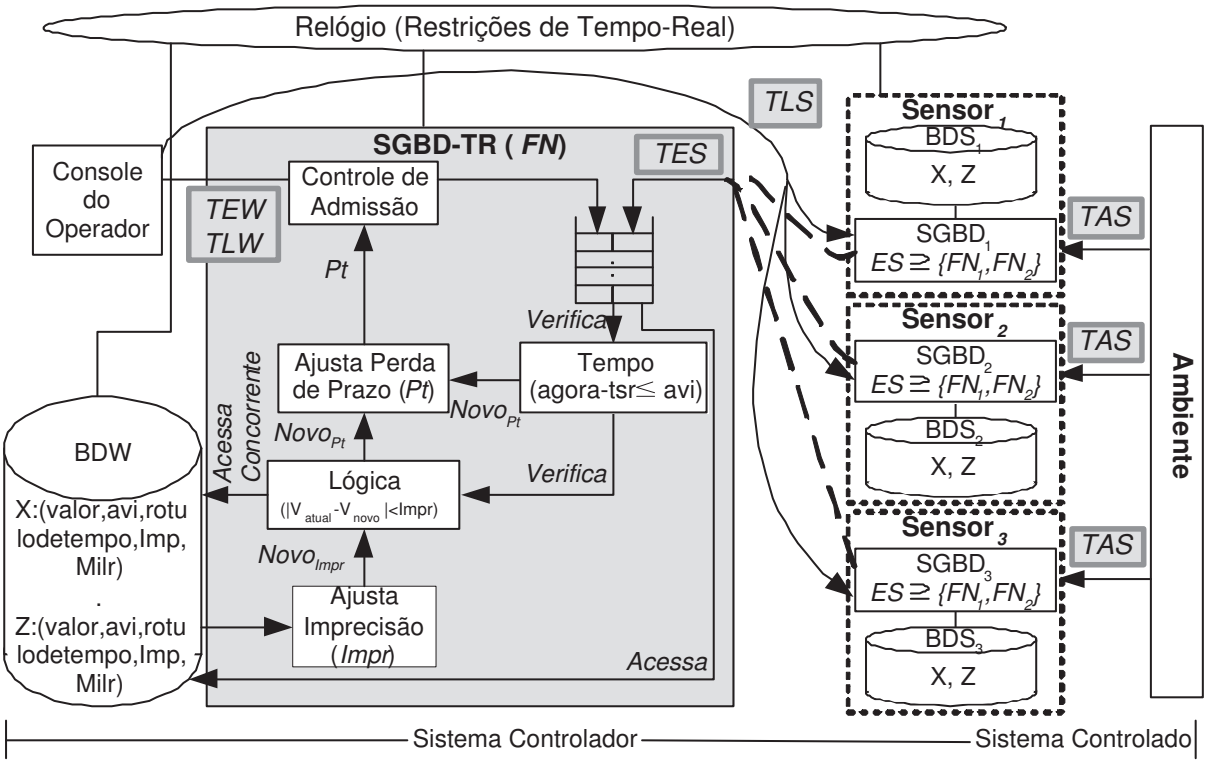

Figura 8.4: Arquitetura do Sistema de Banco de Dados em Tempo-Real

Na arquitetura é contemplado um sistema controlador e um sistema controlado. O SGBD-TR consiste de um componente de software e de um repositório de dados, representados na Figura por SGBD-TR (FN) e BDW, respectivamente. Os sensores possuem as funcionalidades de um sistema de banco de dados, sendo composto por um componente de software, representado na Figura por SGBD, e por um repositório de dados representado por BDS.

O cenário definido no Capítulo 7 para o estudo de caso é composto por três sensores de temperatura para monitorar o ambiente. O *workflow* para o cenário é como segue: os sensores adquirem os dados do ambiente e os armazenam. Periodicamente os dados dos sensores são enviados para o SGBD-TR através das transações de sensores. O SGBD-TR é atualizado a fim de permitir que consultas históricas sejam realizadas, uma vez que não é possível armazenar todos os dados no sensor por um longo período de tempo. Os dados dos sensores são sempre mais atuais que os dados no SGBD-TR e esta diferença é devido ao atraso que existe entre a aquisição do dado pelo sensor e a atualização deste dado no servidor. Operações de leitura e escrita são permitidas tanto nos sensores como no SGBD-TR.

Para facilitar a validação do modelo, o cenário supracitado é dividido em cenários

menores. Os cenários menores considerados são:

1. Operações de Leitura e Escrita nos sensores: as transações definidas para os sensores (TAS e TLS), bem como o conteúdo local dos repositórios de dados são ilustrados.

2. Operações de Leitura e Escrita no servidor: as transações definidas para o servidor (TES, TEW e TLS), bem como o conteúdo do repositório de dados são ilustrados.

3. Funções de qualidade de serviços: neste cenário, as funções de QoS definidas para o modelo são mostradas.

8.5.1 Geração do Grafo de Ocorrência

O relatório obtido com a geração do grafo de ocorrência para o banco de dados em tempo-real modelado é gerado completamente em 47 (quarenta e sete) segundos, com 6713 (seis mil, setecentos e treze) nós e 22867 (vinte e dois mil, oitocentos e sessenta e sete) arcos.

```
Estatísticas
------------------------------------------------------------------------

  Occurrence Graph
    Nodes:  6713
    Arcs:   22867
    Secs:   47
    Status: Full
```

As propriedades de limite dos lugares são mostradas a seguir. Para o lugar *ObjetoBD'ObjetoBDP*, que representa o repositório de dados, o limite máximo e mínimo de fichas é igual a 1 (um).

```
 Propriedades de Limite
------------------------------------------------------------------------

  Best Integers                 Bounds    Upper      Lower
  ObjetoBD'ObjetoBDP              1       1          1

Best Upper Multi-set Bounds
```

```
ObjetoBD'ObjetoBDP          1   1‘{nr = t1,vrr = 10,avir = 10,tsr = 33,
                                    impr = 1,milr = 8}++ 1‘{nr = t1,
                                    vrr = 10,avir = 10,tsr = 39,impr = 1,
                                    milr = 8}
```

Para a propriedade de vivacidade, 24 (vinte e quatro) marcações mortas são obtidas com a geração do grafo de occorrência. Em outras palavras, existem 24 (vinte e quatro) maneiras diferentes para concluir a simulação da rede de Petri. Também é possível observar, através da transição morta *FCObjetoBDLeAt'AvaliaFCLeAt*, que não houve nenhum conflito, ou seja, nenhuma situação de concorrência quando uma leitura está executando e uma escrita é invocada.

```
Propriedades de Vivacidade
--------------------------------------------------------------------------
  Dead Markings:  24 [6713,6712,6711,6710,6703,...]
  Dead Transitions Instances:  FCObjetoBDLeAt'AvaliaFCLeAt 1
```

8.5.2 Geração do Diagrama de Seqüência de Mensagens

Cenário 1 - Operações de Leitura e Escrita nos Sensores

As operações de escrita e de leitura nos bancos de dados sensores são executadas pelas transações de aquisição sensores (TAS) e transações de leitura sensores (TLS), respectivamente. O funcionamento dos sensores é como segue: a cada unidade de tempo, os sensores inteligentes podem adquirir um novo dado do ambiente, através da unidade de sensoriamento. Cada dado adquirido pode ser armazenado no sensor, através da unidade de processamento, ou não. Contudo, existe um intervalo de validade absoluta para o item de dado no sensor, isto é, cada item de dado armazenado é válido durante uma certa quantidade de tempo. Por conseguinte, quando a validade do dado estiver próxima de ser atingida, o novo valor do item de dado adquirido tem que ser armazenado.

Este cenário é ilustrado na Figura 8.5, onde os dados adquiridos do ambiente são exportados para os bancos de dados sensor, através das TAS. São consideradas três TAS

no modelo, uma para cada sensor, onde: $TAS1$ para o $sensor_1$, $TAS2$ para o $sensor_2$ e $TAS3$ para o $sensor_3$. Na linha de execução Banco de Dados Sensor é mostrado o conteúdo de cada repositório local de dados. O arquivo armazenado é uma lista composta por vários registros, onde cada registro possui os seguintes campos: valor do item de dado, rótulo de tempo e identificação do sensor. O tamanho do arquivo é limitado devido as restrições de armazenamento inerentes aos sensores. Os conteúdos dos bancos de dados sensores podem ser importados, através da execução das TLS. Uma TLS é definida para cada sensor e são executadas pelas aplicações residentes no console do operador.

Algumas características são definidas para as transações e para os bancos de dados sensor, onde:

- As TAS possuem somente operações de escrita, com padrões de chegada esporádica e têm restrições de tempo-real firmes. O tempo de liberação e o período destas transações são imprevisíveis, contudo o tempo mínimo entre uma execução e outra da mesma TAS é de uma unidade de tempo. O valor adquirido do ambiente tem uma validade de cinco unidades de tempo, caso este valor tenha perdido sua validade antes de ser escrito no repositório, a TAS é abortada e nenhum valor é escrito.

- Os bancos de dados sensor possuem uma capacidade limitada de armazenamento. No modelo, este limite é definido em quatro registros. Dessa forma, se um repositório estiver com quatro registros, a inserção de um novo registro se dá pela exclusão do registro mais antigo da lista. O novo registro é inserido na cabeça da lista. O intervalo de validade absoluta para o item de dado nos sensores é definido em cinco unidades de tempo, isto é, a cada cinco unidades de tempo pelo menos um registro deve ser inserido na lista.

- As TLS possuem somente operações de leitura, com padrões de chegada aperiódicos e têm restrições de tempo-real estritas. O tempo de liberação e o período destas transações são imprevisíveis, e não existe tempo mínimo entre uma execução e outra da mesma TLS. Uma TES tem que concluir sua execução, de outra forma um prejuízo é gerado.

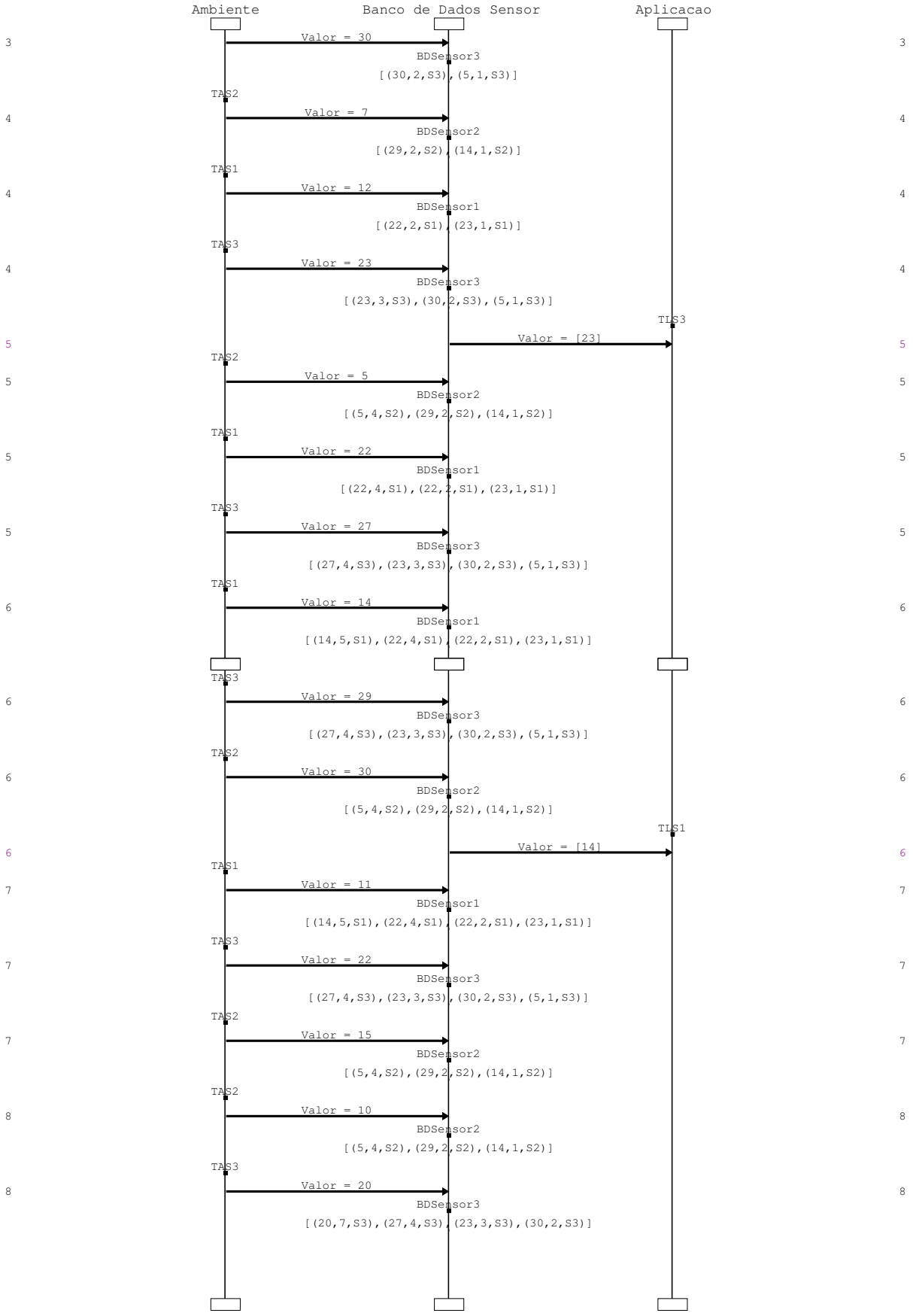

Figura 8.5: Cenário 1 - Diagrama de Seqüência de Mensagens

Na Figura 8.5, a $TAS2$ executou no instante de tempo quatro e por algum motivo este valor não foi inserido no banco de dados. O mesmo aconteceu com a $TAS1$. Já a $TAS3$ inseriu o novo valor para o item de dado que é vinte e três, com rótulo de tempo igual a três. O mesmo aconteceu no instante cinco, onde o valor para o item de dado é igual a vinte e sete, no $sensor_3$. Este valor foi importado pela $TLS3$, no mesmo instante de tempo. No instante de tempo oito, o registro $(20, 7, S3)$ é inserido no $sensor_3$. Como a capacidade de armazenamento deste estava no seu limite, então o registro mais antigo $(5, 1, S3)$, como visto no instante de tempo sete, é excluído para então o registro mais atual ser inserido.

Cenário 2 - Operações de Leitura e Escrita no Servidor

As operações de escrita no servidor de banco de dados são executadas pelas transação de escrita sensores (TES) que são transações enviadas pelos sensores para atualizar o servidor e, pelas transações de escrita servidor (TEW) que são transações enviadas pelos programas aplicativos com operações de escrita para o servidor. Já as operações de leitura são executadas pelas transações de leitura servidor (TLW) que são transações enviadas pelos programas aplicativos para consultar o servidor.

Este cenário é ilustrado pela Figura 8.6 e contém as TES, TEW e TLW. O conteúdo do repositório de dados do servidor também é ilustrado. As transações e o servidor de banco de dados possuem as seguintes características:

- As TES possuem somente operações de escrita, com padrões de chegada periódicos e têm restrições de tempo-real suaves. Para estas transações são definidos os tempos de liberação, tempo computacional, prazo e período. A finalidade das TES é garantir que o conteúdo do servidor de banco de dados esteja o mais próximo possível dos conteúdos dos sensores.

- O banco de dados servidor não possui capacidade limitada de armazenamento. Sempre que um novo valor é inserido no servidor, os parâmetros são atualizados. Tais parâmetros identificam o item de dado, o valor mais atual, o intervalo de validade absoluta, o rótulo de tempo, a imprecisão permitida e o limite de imprecisão.

- As TLW possuem somente operações de leitura, com padrões de chegada esporádicos e têm restrições de tempo-real firmes. Para estas transações são definidos os

tempos de liberação, tempo computacional, prazo e período.

- As TEW possuem somente operações de escrita, com padrões de chegada esporádicos e têm restrições de tempo-real suave. Para estas transações são definidos os tempos de liberação, tempo computacional, prazo e período.

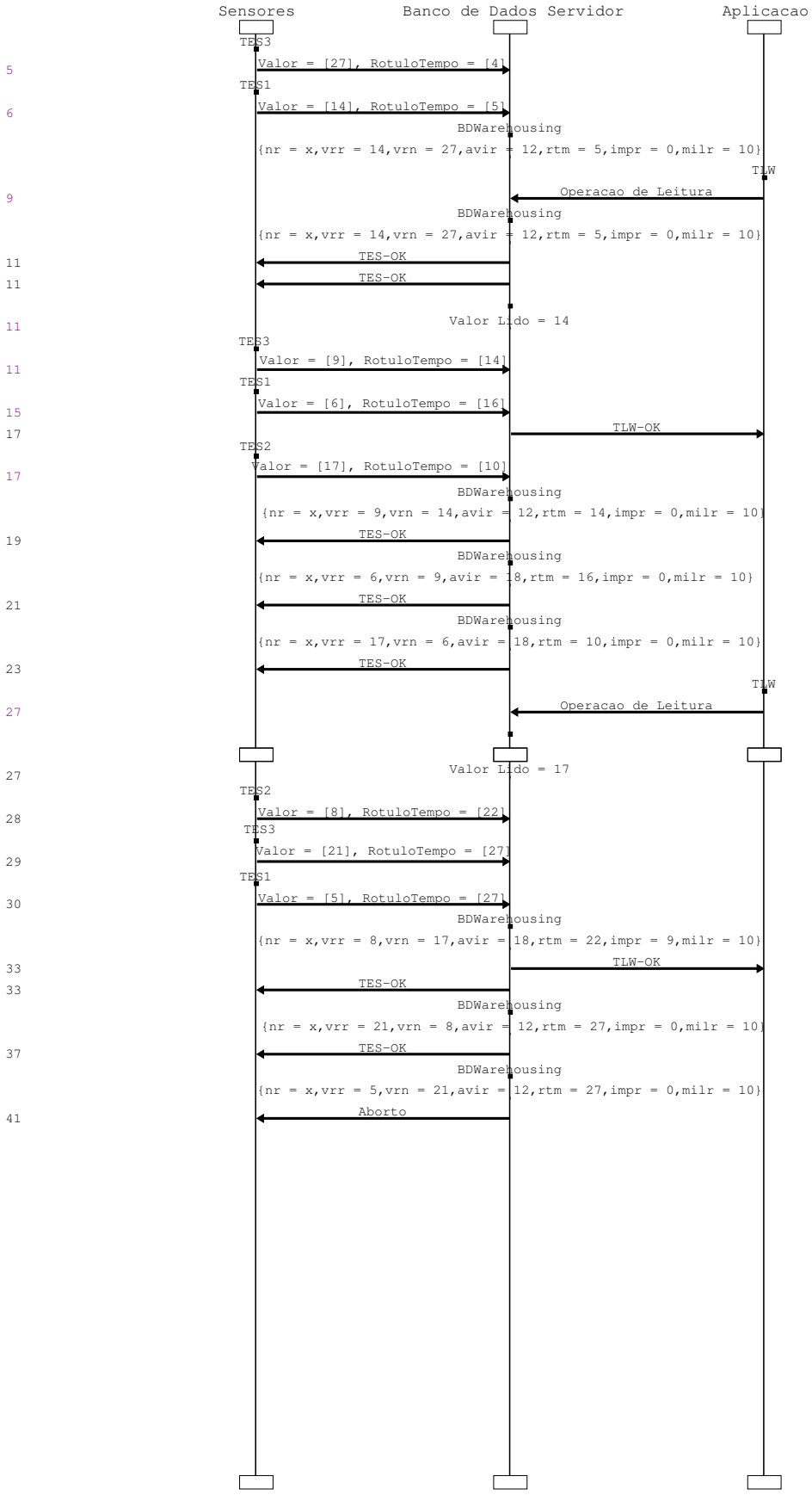

Figura 8.6: Cenário 2 - Diagrama de Seqüência de Mensagens

De acordo com a Figura 8.6, no instante de tempo cinco a $TES3$ é executada, onde o valor do item de dado é igual a vinte e sete e o rótulo de tempo é igual a quatro. No instante de tempo seis, a $TES1$ é executada escrevendo o valor do item de dado igual a quatorze e rótulo de tempo é igual a cinco. Na marca $BDWarehousing$ tais valores são armazenados em: $vrr = 14$ e $vrn = 27$. Ambas as transações são concluídas no instante de tempo onze. No instante de tempo nove uma TLW é executada a partir da aplicação. O acesso ao repositório só foi realizado no instante de tempo onze e a transação concluiu sua execução no instante dezessete. No instante de tempo vinte e oito a $TES2$ iniciou sua execução para inserir o valor para o item de dado igual a oito. Todavia, o prazo para esta transação é de doze unidades de tempo e como $TES2$ só concluiu no instante quarenta e um, então ela abortou como visto na Figura.

Cenário 3 - Funções de qualidade de serviços

As funções de QoS, definidas no Capítulo 4, são ilustradas neste cenário. No contexto de bancos de dados em tempo-real, as funções utilizadas para o gerenciamento de QoS são definidas como segue:

- *Função de Especificação*: através desta função são identificados os parâmetros de tempo e os parâmetros lógicos necessários para garantir o perfeito funcionamento do sistema. Uma vez identificados, alguns destes parâmetros são especificados a *priori* e outros em tempo de execução de acordo com a situação atual do sistema. Dentre os parâmetros especificados inicialmente tem-se a periodicidade, o prazo e o tempo de liberação de uma transação. Em relação aos valores especificados em tempo execução está a imprecisão admitida no item de dado. A FE pode ser visualizada na Figura 8.7, na linha de execução Especif./Mapeam..

- *Função de Mapeamento*: traduz as exigências de qualidade definidas nas transações sensores e transações do servidor para os sistemas de gerenciamento de dados, mais especificamente para os escalonadores. Na linha de execução Especif./Mapeam. os parâmetros especificados são mapeados durante toda a execução da transação.

- *Função de Negociação*: esta função substitui a função de compatibilidade. Por conseguinte, o papel da função de negociação é gerenciar a execução concorrente das transações. Essa deve, além de permitir e limitar valores imprecisos, considerar

valores aceitáveis para a negociação a fim de evitar que uma das transações seja abortada ou perca seu prazo. As principais vantagens de FN em relação à FC é a possibilidade de avaliar mais de duas transações executando concorrentemente e a definição do limite de imprecisão permitida para o item de dado ser obtida por uma métrica de desempenho. Na linha de execução Negociação é mostrado os valores dos parâmetros da FN sendo avaliados.

- *Função de Monitoração*: examina se os prazos especificados para as transações estão sendo obedecidos e, se os limites de imprecisão definidos para os ítens de dados não foram excedidos. Também, controla a quantidade de transações que podem perder o prazo em um determinado intervalo de tempo. Na Figura 8.7, a FM é obtida pela seta que tem início na linha de execução Banco de Dados Servidor e termina na linha Transacoes. Se a FM foi avaliada em verdadeira então o rótulo da seta é $AtualizacaoOK$, em outro caso o rótulo é *Aborto*.

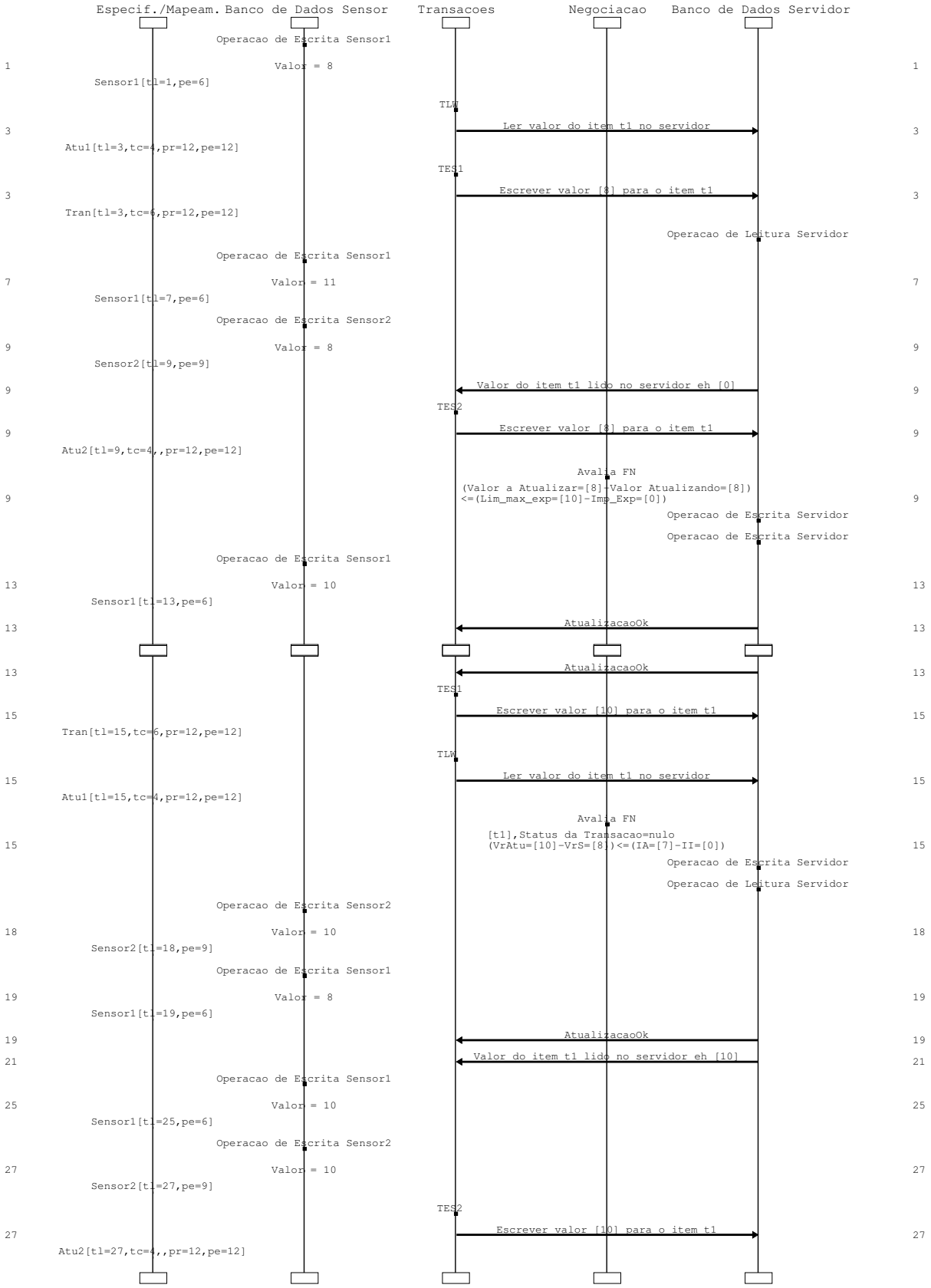

Figura 8.7: Cenário 3 - Diagrama de Seqüência de Mensagens

Para o cenário, a seguinte especificação é definida:

- $TES1$: atualiza periodicamente o banco de dados servidor em relação ao sensor1. O item de dado atualizado é o cabeça da lista no sensor1. O tempo de liberação é 3 u.t., o tempo computacional é 2 u.t., o prazo é 12 u.t. e o seu período é de 12 u.t..

- $TES2$: atualiza periodicamente o banco de dados servidor em relação ao sensor2. O item de dado atualizado é o cabeça da lista no sensor2. O tempo de liberação é 9 u.t., o tempo computacional é 4 u.t., o prazo é 18 u.t. e o seu período é de 18 u.t..

- TLW: ler periodicamente o conteúdo do objeto de banco de dados servidor. O tempo de liberação é 3 u.t., o tempo computacional é 6 u.t., o prazo é 12 u.t. e o seu período é de 12 u.t..

No instante de tempo um, ocorreu uma escrita no *sensor1*, como indicado na haste rotulada por Banco de Dados Sensor. Cada execução deste é identificada pelo rótulo Operacao de Escrita Sensor1 e o valor adicionado. Na haste rotulada por Especif./Mapeam. é possível observar as propriedades de tempo para as transações dos sensores e do servidor. A primeira execução do *sensor2* é no tempo nove, como visto na Figura 8.7.

No tempo três, tanto a TLW quanto a $TES1$ iniciam sua execução. Como observado nos rótulos das setas duas operações conflitantes tentarão acessar o *Banco de Dados Servidor*. A primeira é uma leitura e a segunda uma escrita (atualizar para oito o valor do item de dado x).

A TLW efetivou no tempo nove, de acordo com o seu tempo computacional. Já a $TES1$ teve que esperar a TLW para então acessar o servidor. Isto pode ser observado com o seu tempo computacional, onde a efetivação foi no instante treze, que era para ser no instante sete. Assim, as operações não puderam ser executadas concorrentemente, ou seja, a função de negociação foi avaliada em falsa.

Antes da efetivação da $TES1$, a $TES2$ iniciou sua execução. Mais uma vez, operações conflitantes tentam acessar o servidor ao mesmo tempo. Todavia, para esta execução concorrente, a função de negociação foi avaliada em verdadeira. Os parâmetros avaliados podem ser vistos na Figura. A haste rotulada por Negociacao apresenta uma marca com a descrição Avalia FN seguida dos parâmetros e seus respectivos valores em tempo de execução. Como conseqüência, a $TES2$ pôde executar concorrentemente.

No tempo quinze, mais uma vez operações conflitantes tentam acessar o servidor. Desta vez temos a situação em que uma atualização está executando e uma consulta é invocada. De acordo com a função de negociação, estas puderam executar concorrentemente.

8.5.3 Geração do Diagrama de Tempo

Para a geração do diagrama de tempo, duas especificações QoS são consideradas para o escalonamento das transações. Na primeira à especificação QoS é dada em termos da corretude lógica do item de dado. Transações conflitantes não podem executar concorrentemente no banco de dados. Por exemplo, se uma leitura estiver sendo processada, o sensor não poderá atualizar o banco de dados servidor. A outra especificação QoS é dada em termos da corretude temporal do sistema. Para o exemplo anterior, esta permite que o sensor atualize o servidor concorrentemente a leitura. Porém, o dado resultante pode herdar uma inconsistência. Através da métrica de desempenho $Impr$, é parametrizado o quão inconsistente o dado pode ser para ainda ser considerado válido.

Os parâmetros definidos devem ser satisfeitos durante a execução das transações de acordo com o nível de qualidade exigido. Para isso a política de escalonamento das transações deve considerar diferentes protocolos de controle de concorrência que consistem basicamente em priorizar as restrições de tempo ou as restrições lógicas. Uma vez que garantir os dados sempre atualizados é uma exigência conflitante com garantir os parâmetros temporais impostos às transações.

Para a geração do diagrama de tempo, os parâmetros das transações foram especificados em:

- $TAS1$: escreve periodicamente no banco de dados local. O tempo de liberação é de 1 unidade de tempo (u.t.) e o seu período é de 6 u.t.. Consideramos que os dados obtidos possuem valores entre 8 e 11.

- $TAS2$: escreve periodicamente no seu banco de dados local. O tempo de liberação é de 9 u.t. e o seu período é de 9 u.t.. Consideramos que os dados obtidos possuem valores entre 8 e 11.

- $TES1$: atualiza periodicamente o objeto de banco de dados servidor em relação ao sensor1. O item de dado atualizado é o cabeça da lista no sensor1. O tempo de

liberação é 3 u.t., o prazo é 12 u.t. e o seu período é de 12 u.t..

- $TES2$: atualiza periodicamente o objeto de banco de dados servidor em relação ao sensor2. O item de dado atualizado é o cabeça da lista no sensor2. O tempo de liberação é 9 u.t., o prazo é 18 u.t. e o seu período é de 18 u.t..

- TLW: ler periodicamente o conteúdo do objeto de banco de dados servidor. O tempo de liberação é 3 u.t., o prazo é 12 u.t. e o seu período é de 12 u.t..

Nas Figuras 8.8 e 8.9 têm-se os diagramas de tempo gerado através da simulação do modelo. No eixo das ordenadas, as transações são representadas, enquanto que no eixo das abcissas os tempos de liberação e os períodos para as transações do servidor e dos sensores, além dos prazos finais para as transações do servidor podem ser analisados. Para cada transação três estados são considerados: início, processamento e efetivação. O início da transação é indicado por um retângulo com linhas pontilhadas. O processamento é indicado por um retângulo sombreado e a efetivação por um retângulo com linhas contínuas.

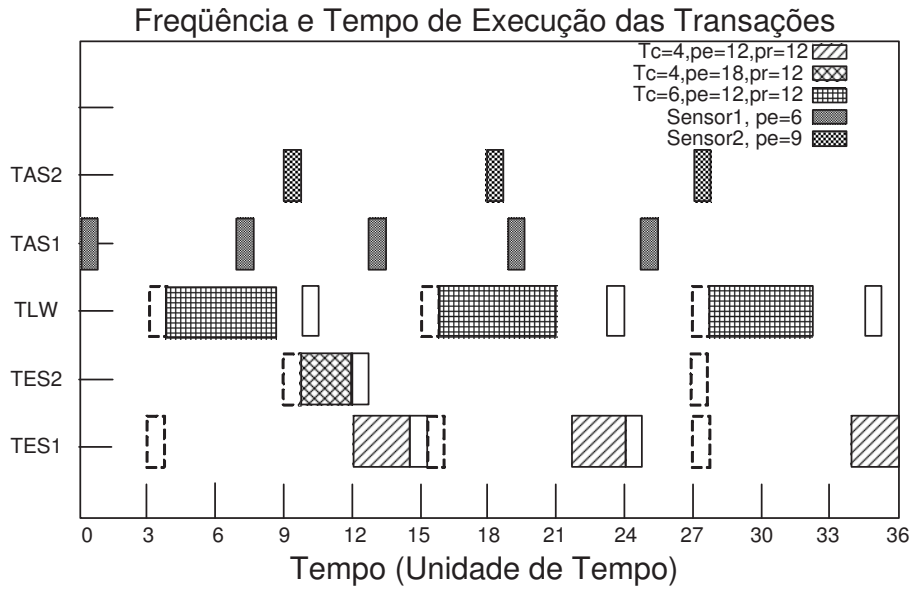

Figura 8.8: Especificação QoS - Corretude Lógica

Na Figura 8.8, o protocolo de controle de concorrência adotado é baseado no critério de corretude lógica, onde as restrições temporais não são consideradas. A transação de $TES1$ iniciou sua execução no instante de tempo três, porém só concluiu no instante quatorze e seu prazo final era no instante quinze. Isto ocorreu devido ao conflito existente

com a TLW. A $TES2$ só executou no instante de tempo nove. No instante vinte e sete ela iniciou, porém não conseguiu processar tendo que ser abortada. A TLW conseguiu processar normalmente durante o intervalo de tempo analisado.

Na Figura 8.9, a técnica de controle de concorrência semântica é adotada. Neste protocolo a $TES2$ executou duas vezes no intervalo de tempo considerado. Esta solução é válida no domínio de redes de sensores, já que o maior número de transações são executadas atendendo suas restrições temporais.

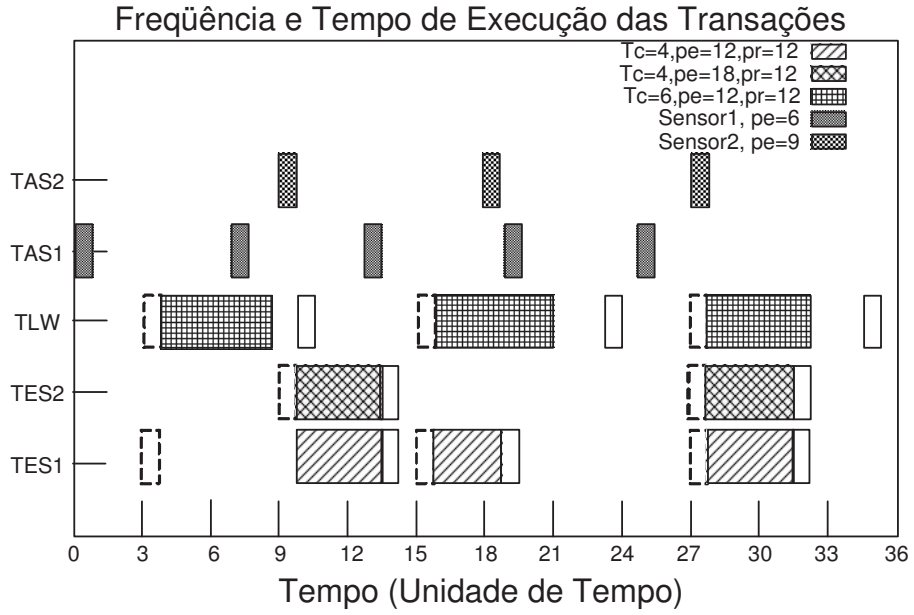

Figura 8.9: Especificação QoS - Corretude Temporal

Capítulo 9

Conclusões e Trabalhos Futuros

Os sistemas computacionais estão em constante evolução e cada vez mais presentes em todas as áreas de conhecimento. Além disso, seus usuários estão cada vez mais exigentes quanto a qualidade de seus serviços, considerando tanto a consistência lógica quanto temporal.

Por exemplo, as redes de sensores, que executam em ambientes abertos e imprevisíveis, têm chamando à atenção de vários pesquisadores. Como os sensores atualmente disponibilizam um maior poder de armazenamento e processamento, capacidade de comunicação sem fio, maior precisão na aquisição dos dados e maior tempo de vida útil, eles estão sendo utilizados para os mais diversos fins.

Neste tese, um método para auxiliar o desenvolvimento de aplicações que executam em ambientes abertos e imprevisíveis, tais como as aplicações de redes de sensores foi apresentado.

O método adota o critério de corretude seriação *Epsilon* considerando QoS para garantir quais escalas de execuções das operações de transações conflitantes são consideradas corretas. O método é uma extensão do o método desenvolvido em (PERKUSICH, 2000) e está de acordo com o esquema ilustrado na Figura 4.4.

As contribuições desta tese são resumidas a seguir:

- Garantir a previsibilidade em SGBD-TR, ou seja, garantir as restrições de tempo impostas aos dados e transações;

- Utilizar funções de qualidade de serviços para garantir o desempenho do sistema. A função de especificação reflete o que se quer; a função de negociação define como

se pode atender a esse pedido e quanto vai lhe custar e a função de monitoração verifica se o que foi pedido está sendo atendido. Se necessário, ou possível, uma função de renegociação é avaliada;

- Definição de métricas de desempenho adaptativas para garantir o comportamento desejado do sistema. Duas métricas, definidas no Capítulo 4, forão consideradas.

- Disponibilizar um método para o desenvolvimento de aplicações que executam em ambientes abertos e imprevisíveis tais como as aplicações de redes de sensores.

9.1 Trabalhos Futuros

No desenvolvimento desta Tese há algumas questões que ainda não foram discutidas ou pouco elaboradas. Desta forma, a partir desta pesquisa, abre-se um caminho promissor de desenvolvimentos a ser explorado. Tais desenvolvimentos são tanto num nível conceitual e teórico, no sentido de enriquecer o modelo; quanto num nível de implementação, de modo a disponibilizar um ferramental de análise e verificação.

Portanto, alguns direcionamentos já considerados com o objetivo de enriquecer o modelo são considerados, e incluem os seguintes:

- Implementar ferramentas integradas para automatizar o processo de desenvolvimento do modelo;

- Geração de código, a partir do modelo, para implementação do sistema.

Referências Bibliográficas

ADELBERG, B.; GARCIA-MOLINA, H.; WIDOM, J. The strip rule system for efficiently maintaining derived data. *SIGMOD Rec.*, ACM Press, v. 26, n. 2, p. 147–158, 1997. ISSN 0163-5808.

ADELBERG, B.; KAO, B.; GARCIA-MOLINA, H. Overview of the stanford real-time information processor (strip). *SIGMOD Record*, v. 25, n. 1, p. 34–37, 1996.

ADELSTEIN, F. et al. *Fundamentals of Mobile and Pervasive Computing*. [S.l.]: McGraw-Hill Companies, Inc., 2005. ISBN 0-07-141237-9.

AMIRIJOO, M. *Algorithms for Managing Real-time Data Services Using Imprecise Computation*. May 2003.

AMIRIJOO, M.; HANSSON, J.; SON, S. Specification and Management of QoS in Imprecise Real-Time Databases. *International Database Engineering and Applications Symposium (IDEAS)*, Julho 2003. Disponível em: <citeseer.ist.psu.edu/586342.html>.

ANDLER, S. et al. Deeds: Towards a distributed and active real-time database systems. *ACM SIGMOD Record*, v. 15, n. 1, p. 38–40, Março 1996. Disponível em: <citeseer.ist.psu.edu/andler96deeds.html>.

ANDLER, S. et al. An overview of the deeds real-time database architecture. *Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*, 1998. Disponível em: <citeseer.ist.psu.edu/355725.html>.

AURRECOECHEA, C.; CAMPBELL, A. T.; HAUW, L. A survey of qos architectures. *Multimedia Systems*, v. 6, n. 3, p. 138–151, 1998. Disponível em: <citeseer.ist.psu.edu/article/aurrecoechea96survey.html>.

BOCHMANN, G. et al. Architectural Design of Adaptive Distributed Multimedia Systems. In: *International Workshop on Multimedia Software Development*. Berlin (Germany): [s.n.], 1996.

BONNET, P. et al. *Query Processing in a Device Database Systems*. [S.l.], Outubro 1999.

BONNET, P.; GEHRKE, J.; SESHADRI, P. Towards Sensor Database Systems. *2nd International Conference on Mobile Data Management*, p. 3–14, Janeiro 2001. Hong Kong.

BONNET, P.; SESHADRI, P. Device Database Systems. *Proceedings of the International Conference on Data Engineering ICDE'99*, Março 2000. San Diego, CA.

BORN, M.; HALTEREN, M. V.; KATH, M. Modeling and Runtime Support for Quality of Service in Distributed Component Platforms. *11th Annual IFIP/IEEE International Workshop on "Distributed Systems: Operations and Management*, 2000.

BRADINATH, B.; RAMAMRITHMAN, K. Syncrhoning transactions on objects. *IEEE Transactions on Computers*, v. 37, n. 5, p. 541–547, 1988.

BRAYNER, A. *Transaction Management in Multidatabase Systems*. [S.l.]: Shaker Verlag, 1999. ISBN 3-8265-6142-2.

BUCHMANN, A. P. et al. The reach active oodbms. In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. [S.l.]: ACM Press, 1995. p. 476. ISBN 0-89791-731-6.

CARDOSO, J. *Quality of Service and Semantic Composition of Workflows*. Tese (Doutorado) — University of Georgia, Department of Computer Science, Athens, GA, ago. 2002.

CHEN, Z.; GEHRKE, J.; KORN, F. Query Optimization In Compressed Database Systems. *In Proceedings of the 2001 ACM Sigmod International Conference on Management of Data Santa Barbara, California*, Maio 2001.

CHENG, A. M. K. *Real-Time Systems: Scheduling, Analysis, and Verification*. [S.l.]: John Wiley & Sons, Inc., 2002. ISBN 0-471-18406-3.

CHRISTENSEN, S.; JOERGENSEN, J. B.; KRISTENSEN, L. M. Design/CPN — A computer tool for coloured Petri nets. *Lecture Notes in Computer Science*, v. 1217, p. 209–221, 1997. ISSN 0302-9743.

COCCOLI, A.; BONDAVALLI, A.; GIANDOMENICO, F. D. Analysis and Estimation of the Quality of Service of Group Communication Protocols. *ISORC'01 - 4th IEEE International Symposium on Object-oriented Real-time distributed Computing*, 2001.

COOLING, J. *Software Engineering for Real-Time Systems*. [S.l.]: Addison Wesley, 2003. ISBN 0-201-59620-2.

DAIMI. *Design/CPN - Computer Tool for Coloured Petri Nets*. 2004. DAIMI Web Site. Disponível em: <http://www.daimi.au.dk/designCPN/>.

DAIMI. *Design/CPN - Computer Tool for Coloured Petri Nets*. 2004. DAIMI Web Site. Disponível em: <http://www.daimi.au.dk/designCPN/libs/mscharts/>.

DIPIPPO, L. Semantic real-time object-based concurrency control. *Tese de Doutorado, Department of Computer Science and Statistics, University of Rhode Island*, 1995.

DONALDSON, A. J. M. Formal Specification of QoS Properties. *Workshop on Multimedia Applications and Quality of Service Verification*, 1994.

DOUGLASS, B. P. *Real Time UML: Advances in the UML for Real-Time Systems*. [S.l.]: 3rd Edition, Addison-Wesley, 2004.

ELMASRI, R.; NAVATHE, S. *Fundamentals of Database Systems, 2nd Edition*. New York: Addison–Wesley Pub. Co., 1994.

ESWARAN, K. P. et al. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, v. 19, n. 11, p. 624–633, November 1976.

FIROIU, V. et al. Theories and models for internet quality of service. *Proceedings of the IEEE (To appear)*, 2002.

FISCHER, S.; MEER, H. QoS Management: A Model-Based Approach. *6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Julho 1998.

GIRAULT, A. et al. Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems. *21st International Conference on Distributed Computing Systems(ICDCS)*, April 2001.

GOEBEL, V. et al. Towards QoS Support in Multimedia Database Management Systems (Poster). *ACM Multimedia 98*, September 1998.

GRUENWALD, L.; LIU, S. A performance study of concurrency control in a real-time main memory database system. *SIGMOD Rec.*, ACM Press, New York, NY, USA, v. 22, n. 4, p. 38–44, 1993. ISSN 0163-5808.

GUERREIRO, D. D. S. *Redes de Petri Orientadas a Objetos*. Tese (Doutorado) — Curso de Doutorado em Engenharia Elétrica, CCT/UFPB, Universidade Federal de Campina Grande, Campina Grande, PB, Abril 2002.

HAUBERT, J.; SADEG, B.; AMANTON, L. Relaxing the real-time constraints in distributed real-time database management systems. In: *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Godenburg, Suécia: Lecture Notes Series by Springer-Verlag, 2004.

HOLANDA, M.; BRAYNER, A.; FIALHO, S. V. Controle de concorrência em bancos de dados para ambientes de computação móvel. *VI Workshop de Comunicação sem Fio e Computação Móvel*, p. 161–170, 2004.

JENSEN, K. Coloured Petri Nets: A High Level Language for System Design and Analysis. In: *Advances in Petri Nets 1990*. [S.l.]: Springer-Verlag, 1990, (Lecture Notes in Computer Science, v. 483).

JENSEN, K. *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use*. [S.l.]: Springer-Verlag, 1992. (EACTS – Monographs on Theoretical Computer Science).

JENSEN, K. *Coloured petri nets-Basic Concepts, Analysis Methods and Practical Use*. [S.l.]: Springer-Verlag, 1997.

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

JENSEN, K.; AL, e. *"Design/CPN"4.0*. [S.l.], 1999. On-line version:http://www.daimi.aau.dk/designCPN/.

KANG, K. *QoS-Aware Real-Time Data Management*. Tese (Doutorado) — Department of Computer Science, University of Virginia, Maio 2003.

KANG, K.; SON, S. H.; STANKOVIC, J. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Transactions on Knowledge and Data Engineering*, v. 16, n. 07, July 2004.

KANG, K.-D. *qRTDB: QoS-Sensitive Real-Time Database*. University of Virginia, Dezembro 2001. Proposta de Tese (Doutorado).

KANG, K.-D. *QoS-Aware Real-Time Data Management*. Tese (Doutorado) — Faculty of the School of Engineering and Applied Science, University of Virginia,, Maio 2003.

KAO, B. et al. Updates and view maintenance in soft real-time database systems. In: *8th international conference on Information and knowledge management*. [S.l.]: ACM Press, 1999. p. 300–307. ISBN 1-58113-146-1.

LAM, K.-Y. et al. Evaluation of concurrency control strategies for mixed soft real-time database systems. *Inf. Syst.*, Elsevier Science Ltd., v. 27, n. 2, p. 123–149, 2002. ISSN 0306-4379.

LEITE, C. R. M. et al. QL-RTDB: Query Language for Real-Time Database. *7th International Conference on Enterprise Information Systems*, , p. , 2005.

LI, B.; NAHRSTEDT, K. A Control Theoretical Model for Quality of Service Adaptations. *Proceedings of 6th International Workshop on Quality of Service*, 1998. Napa, CA, May 1998.

LINDSTRöM, J. Integrated and adaptive optimistic concurrency control method for real-time databases. *8th International Conference on Real-Time Computing Systems and Applications (RTCSA'2002)*, University of Helsinki Finland, p. 143–151, 2002.

LINDSTRöM, J. *Optimistic Concurrency Control Methods for Real-Time Database*. Tese (Doutorado) — Department of Computer Science, University of Helsinki Finland, Novembro 2002.

LINDSTRöM, J.; NIKLANDER, T.; RAATIKAINEN, K. *Evaluation of RODAIN-2000 Prototype Real-Time Database System*. 2000. Disponível em: <citeseer.ist.psu.edu/500358.html>.

LIU, S. *Real-Time Systems*. [S.l.]: Prentice-Hall, New Jersey, 2000.

LOUREIRO, A. A. F. et al. Wireless sensors networks (in portuguese). In: *Proceedings of the 21st Brazilian Symposium on Computer Networks (SBRC'03)*. Natal, RN, Brazil: [s.n.], 2003. p. 179–226. Tutorial.

LU, W. C. et al. Online scheduling for imprecise computations with deferred optional tasks. In: *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Godenburg, Suécia: Lecture Notes Series by Springer-Verlag, 2004.

MALLADI, R.; AGRAWAL, D. P. Current and future applications of mobile and wireless networks. *Commun. ACM*, ACM Press, v. 45, n. 10, p. 144–146, 2002. ISSN 0001-0782.

MONTEIRO FILHO, J. M. S. *Teste do Grafo de Serialização Temporal: Uma Estratégia para o Controle de Concorrência em Ambientes de Broadcast*. Dissertação de Mestrado, Universidade de Fortaleza, Fortaleza, CE: [s.n.], Outubro 2001.

MURATA, T. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, abr. 1989.

NYSTRöM, D. et al. Pessimistic concurrency-control and versioning to support database pointers in real-time databases. In: *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS04)*. Sicily, Italy: IEEE Computer Society Press, 2004.

NYSTRöM, D. et al. Pessimistic concurrency control and versioning to support database pointers in real-time databases. In: *Proceedings of Euromicro conference on Real-Time Systems (ECRTS)*. [S.l.: s.n.], 2004.

PECKHAM, J. et al. *Design of a Real-Time Object-Oriented Database System*. [S.l.], 1996.

PERKUSICH, M. *Um Método Baseado em Redes de Petri para a Modelagem de Bancos de Dados para Aplicações em Tempo-Real*. Tese (Doutorado) — Curso de Doutorado em

Engenharia Elétrica, CCT/UFPB, Universidade Federal da Paraíba, Campina Grande, PB, Abril 2000.

PRICHARD, J. *"RTSQL": Extending The "SQL"Standard to Support Real-Time Databases*. Tese (Doutorado) — Department of Computer Science and Statistics, University of Rhode Island, 1995.

RAATIKAINEN, K.; LINDSTRöM, J. Using Real-Time Serializability and Optimistic Concurrency Control in Firm Real-Time Databases. University of Helsinki Finland, Março 2002.

RAMAMRITHAM, K. Real-time databases. *Distributed and Parallel Databases*, v. 1, n. 2, p. 199–226, 1993. Disponível em: <citeseer.ist.psu.edu/ramamritham93realtime.html>.

RAMAMRITHAM, K.; PU, C. A formal characterization of epsilon serializability. *IEEE Transactions on Data and Knowledge Engineering*, v. 6, n. 7, p. 997–1007, 1995.

RIBEIRO NETO, P. *Análise de Controle de Concorrência e Escalonamento em Bancos de Dados em Tempo-Real Usando Redes de Petri Coloridas*. Dissertação de Mestrado, Universidade Federal da Paraíba, Campina Grande, PB: [s.n.], Agosto 2001.

RIBEIRO NETO, P. *Qualidade de Serviços em Bancos de Dados em Tempo-Real*. Universidade Federal de Campina Grande, PB, Setembro 2002.

RIBEIRO NETO, P. *Modelagem de Banco de Dados em Tempo-Real considerando QoS para Sistemas Embarcados*. Universidade Federal de Campina Grande, PB, Agosto 2003.

RIBEIRO NETO, P. *Uma Especificação Formal de Seriação Epsilon considerando propriedades de QoS*. Universidade Federal de Campina Grande, PB, Maio 2003.

RIBEIRO NETO, P. et al. Verification, Validation and Testing in Software Engineering. In: _____. [S.l.]: Idea Group, Inc., 2005. cap. A Formal Verification and Validation Approach for Real-Time Database.

RIBEIRO NETO, P.; PERKUSICH, A.; M.L.B., P. Modelagem e Análise de Qualidade de Serviços para Bancos de Dados em Tempo-Real. *V Workshop de Tempo Real*, p. 63–70, 2003.

RIBEIRO NETO, P.; PERKUSICH, A.; M.L.B., P. Uma Aplicação de Bancos de Dados em Tempo-Real para Redes de Sensores. *VI Workshop de Tempo Real*, p. 45–52, 2004.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. A Model in Petri Nets to Analyze Quality of Service in Real-Time Databases. *IEEE Systems, Man and Cybernetics Conference*, 1, p. 300–305, 2003.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Formal Specification of the Epsilon Seriazibility Considering Quality of Service. *IEEE Systems, Man and Cybernetics Conference*, 4, p. 4027 – 4032 , 2003.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Real-Time Database Modeling Considering Quality of Service. *5th International Conference on Enterprise Information Systems*, 3, p. 403–410, 2003.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Escalonamento de Transações em Sistemas de Gerenciamento de Banco de Dados em Tempo-Real com Aplicações para Redes de Sensores. *I Workshop da Rede de Instrumentação e Controle*, 2004.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Modelling and Analysis of Real-Time Databases for Sensor Networks Using Coloured Petri Nets. *11th IFAC Symposium on Information Control Problems in Manufacturing*, p. 125 – 130, 2004.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Real-Time Database Modeling for Sensor Networks. *6th International Conference on Enterprise Information Systems*, 1, p. 599 – 603, 2004.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Real-Time Databases Application for Sensor Networks. *IEEE Systems, Man and Cybernetics Conference*, p. Aceito para publicação, 2004.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Scheduling real-time transactions for sensor networks applications. In: *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Godenburg, Suécia: Lecture Notes Series by Springer-Verlag, 2004. p. 181 – 200.

RUIZ, L. B. et al. Architectures for wireless sensor networks (in portuguese). In: *Proceedings of the 22nd Brazilian Symposium on Computer Networks SBRC'04)*. Gramado, RS, Brazil: [s.n.], 2004. p. 167–218. Tutorial. ISBN: 85-88442-82-5.

SAGE, A. P. *Decision support systems engineering*. [S.l.]: John Wiley & Sons, Inc., 1991. ISBN 0-471-53000-X.

SALAMATIAN, K.; FDIDA, S. Measurement based modelling of Quality of Service in the Internet: a methodological approach. *IWDC 2001 - Evolutionary Trends of the Internet*, 2001.

SANTOS, R.; LIPARI, G.; SANTOS, J. Scheduling open dynamic systems: The clearing fund algorithm. In: *Proceedings of the 10th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. Godenburg, Suécia: Lecture Notes Series by Springer-Verlag, 2004.

SANTOSHKUMAR, I.; MANIMARAN, G.; MURTHY, C. S. R. Static scheduling of object-based real-time tasks with probabilistic conditional branches in distributed systems. *6th IEEE International Workshop on Parallel and Distributed Real-time Systems*, Março 30-Abril 3 1998.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de Banco de Dados*. [S.l.]: MAKRON Books, 1999. ISBN 85.346.1073-8.

STAEHLI, R. A. *Quality of Service Specification for Resource Management in Multimidia Systems*. Tese (Doutorado) — The Evergreen State College, January 1996.

STANKOVIC, J. A.; RAMAMRITHAM, K.; SPURI, M. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. [S.l.]: Kluwer Academic Publishers, 1998. ISBN 0792382692.

STANKOVIC, J. A.; SON, S. H.; LIEBEHERR, J. BeeHive: Global multimedia database support for dependable, real-time applications. *Lecture Notes in Computer Science*, v. 1553, p. 51–72, 1998. Disponível em: <citeseer.ist.psu.edu/stankovic97beehive.html>.

STEWART, D.; BARR., M. Beginner's corner: Rate monotonic scheduling. *Embedded Systems Programming*, v. 15, n. 03, p. 79–80, Março 2002.

WONG, M.; AGRAWAL, D. Tolerating bounded inconsistency for increasing concurrency in database systems. In: *Proc. of 11th Principles of Databases Systems*. Tucson, AZ: [s.n.], 1992. p. 236–245.

YAO, Y.; GEHRKE, J. E. The Cougar Approach to In-Network Query Processing in Sensor Networks. *Sigmod Record*, 31, n. 3, Setembro 2002.

YAO, Y.; GEHRKE, J. E. Query Processing for Sensor Networks. *To appear in the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Janeiro 2003. Asilomar, California.

YE, H.; KERHERVé, B.; BOCHMANN, G. V. QoS - Aware Distributed Query Processing. *10th International Workshop on Database and Expert Systems Applications (DEXA)*, p. 923–927, Setembro 1999.

YU, P. et al. On real-time databases: Concurrency control and schedulling. *Proceedings of IEEE*, p. 140–157, jan. 1994.

ZIMMERMANN, J.; BUCHMANN, A. *Benchmarking active database systems: A requirements analysis*. 1995. Disponível em: <citeseer.ist.psu.edu/zimmermann95benchmarking.html>.

ÖZSU, M. T.; VALDURIEZ, P. *Princípios de Sistemas de Bancos de Dados Distribuídos*. [S.l.]: Editora Campus, 2001. ISBN 85-352-0713-9.