Sérgio Kostin

Modelo para gerenciamento de dados em um ambiente de comando e controle

Dissertação apresentada ao Departamento de Informática da PUC/RJ como parte dos requisitos para obtenção do título de Mestre em Informática: Ciência da Computação.
Orientador: Marco Antonio Casanova

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 27 de abril de 2001

Aos meus pais

Agradecimentos

Ao professor Marco Antonio Casanova pela sua orientação, incentivo, empenho, apoio, disponibilidade, e pelas suas valiosas idéias e contribuições que enriqueceram este trabalho.

Aos professores Luis Fernando Gomes Soares e Ana Maria Carvalho de Moura pela participação na banca de avaliação.

A todos professores do Departamento de Informática da PUC pela formação acadêmica que me proporcionaram.

Ao Instituto de Pesquisa e Desenvolvimento, pelo apoio administrativo e profissional, em especial, ao Tenente Coronel José Diniz Mesquita Abrunhosa e à gerência do projeto Ruvichá.

Ao Exército Brasileiro e à PUC, pela oportunidade profissional que me ofereceu de realizar este excelente curso.

A minha família e amigos pelo estímulo e apoio.

Resumo

A criação de sistemas de informação voltados para a atividade de comando e controle é uma tendência em todas as Forças Armadas. As operações militares são significativamente móveis, sujeitas a interferências do ambiente e do inimigo, tornando as suas comunicações pouco confiáveis, somando-se, também, ao fato de se basear em sistemas rádio. Possuem uma característica de hierarquia em sua organização, bem como a alteração dinâmica de sua estrutura organizacional.

O objetivo principal da dissertação é a definição de um modelo de gerência de dados para computação móvel que atenda, em particular, aos requisitos de um sistema de comando e controle voltado para aplicações militares.

Especificamente, as principais contribuições da dissertação são: um modelo de replicação de dados; a definição de técnicas para tratamento da busca e difusão da informação no sistema, considerando as características de mobilidade, perfil dos clientes e localização das estações; e um *framework* para a implementação das técnicas propostas.

Abstract

The development of Command and Control Information Systems is a trend in worldwide Armed Forced. Military Operations are very mobile, subject to interferences of the environment and of the enemy. In short, the communications are not trustworthy. The organization of Military Operations is hierarchical and can change in time.

The goal of this dissertation is the definition of a model of data management for mobile computing that meets the requirements of a command and control system for military applications.

Specifically, the main contributions of the dissertation are: a data replication model; the definition of techniques for treatment of the search and diffusion of the system's information, considering the mobility characteristics, the customer's profile and the location of the stations; and a framework for the implementation of the proposed techniques.

Sumário

Lista de Ilustrações

Lista de Tabelas

1. Introdução

1.1 Motivação

A evolução da tecnologia da informação se faz presente nos exércitos modernos. As operações, que antes eram planejadas e controladas em papel, passam a ser controladas por meio eletrônico [Camp92], através de sistemas de comando e controle.

Isso fez surgir uma série de requisitos referentes a este tipo de aplicação, seja em infraestrutura de comunicações, seja em termos do tráfego de informações gerado, em especial aquelas referentes àárea de Gerência de Dados.

As Organizações Militares (OM) possuem um sistema de informações que atuem em várias áreas, dentro as quais podemos destacar: manobra, logística, apoio de fogo, inteligência e interoperabilidade com demais forças. Estas informações são armazenadas em SGBDs, apoiadas por uma infraestrutura de comunicações, que, em tese, deve ser adequada à característica de alta mobilidade dos componentes do sistema [BA87].

Em uma operação, as OM estão normalmente dispersas no terreno. Suas ligações com os diversos escalões podem ser através de links ponto-a-ponto ou através de uma rede multiponto. Devido à mobilidade das operações militares, as características destas redes poderiam ser sintetizadas como do tipo *ad-hoc*, com alto grau de particionamento e baixas taxas de transmissão.

Um trabalho correlato nesta área pode ser consultado em [BMM96].

O paradigma que melhor atenderia a este tipo de aplicação seria, em tese, o de computação móvel, mais especificamente de bancos de dados móveis.

Bancos de dados móveis são usualmente baseados em redes “sem fio” (*wireless*). Uma rede *wireless* consiste de um ou mais *backbones* fixos, onde as estações de controle estão conectadas entre si. Cada estação de controle coordena a comunicação de um computador móvel (também chamado de estação móvel ou unidade móvel)

dentro de sua respectiva célula para um outro computador móvel dentro da mesma célula, ou em outra célula, ou em uma estação pertencente a um *backbone* da rede. Considera-se ainda uma variação deste tipo de comunicação a situação de duas unidades comunicando-se entre si sem a intermediação de elementos da rede fixa.

Em tais ambientes, os dados podem estar localizados ou na rede fixa ou nas estações móveis. Existem diferentes tipos de estações móveis. Alguns computadores podem ser extremamente simples, com capacidade limitada. Neste caso, os dados são retirados de computadores de maior capacidade, onde as estações recuperam os dados através de procedimentos de *downloading* de acordo com as suas necessidades. Mais interessante, entretanto, é o ambiente onde as estações são mais potentes e armazenam dados nativos e, possivelmente, compartilhados por outros usuários – tecnicamente designados "*walkstations*" [IK96].

Os *links* de comunicação entre os participantes de um ambiente de comando e controle, como afirmado anteriormente, são de baixa velocidade. Existe a necessidade das OM consultarem a base de dados das outras OM as quais estejam relacionadas, seja pela sua posição geográfica, seja referente a dados de pessoal, material e atividades que foram, estejam ou serão executadas. Uma característica deste tráfego de informações é que os dados enviados pelas OM superiores podem e são normalmente difundidos através de *broadcast*, onde estas informações podem ser de interesse de uma OM específica ou de um conjunto de unidades.

A isto poderia se agregar o fato de que, em tese, o escalão superior possui uma infraestrutura de comunicações de capacidade técnica superior aos seus escalões subordinados.

Os tráfegos de informação que têm origem nas OM subordinadas possuem a característica de serem peculiares. Dizem, a priori, respeito a relatórios de situação. O fato das OM estarem normalmente desconectadas é um fator de complicação.

As OM estão organizadas dentro de um nível de hierarquia, em uma estrutura semelhante à de uma árvore. Um outro fator complicador neste tipo de ambiente é a variação dinâmica da estrutura hierárquica, onde uma OM, em virtude das características de determinada operação, pode passar ao controle de outra OM ou

ainda deslocar-se para uma área onde terá apoio de outro escalão, sendo que as atividades de comando e controle não devem sofrer solução de continuidade. Este tipo de operação é normalmente previsto na etapa de *composição de meios* de uma Ordem de Operações. Uma estrutura deve ser criada de forma que a consistência possa ser obtida através da combinação das bases de dados das OM que comandaram a unidade em questão.

As informações normalmente trafegam em diversos níveis. Entretanto, podem ocorrer situações onde a ordem é enviada diretamente a uma OM subordinada, onde este vínculo de subordinação não é direto, sendo a OM imediatamente superior à destinatária apenas informada sobre a existência desta ordem, não necessariamente naquele momento. Este tipo de situação é típico em situações emergenciais.

Determinados tipos de mensagens são transmitidos para todos os elementos, independente do nível de hierarquia. Trata-se normalmente de mensagens urgentes que devam ser de conhecimento de todos em tempo hábil, não sendo necessário que obedeçam à cadeia de comando. Um exemplo típico é a possibilidade de um ataque iminente do inimigo. Este tipo de mensagem é ligado mais a um vínculo espacial do que a um vínculo de subordinação. Para tanto, deve haver uma previsão de emissão de mensagens de acordo com a região.

As demais mensagens trafegam, em tese, no âmbito da hierarquia. A atualização da base de dados será dependente do tipo de mensagem. Determinadas mensagens, as nitidamente corriqueiras, podem atualizar a base de dados da OM superior, evitando-se assim trabalho repetitivo deste escalão. Outros tipos de mensagens são submetidos ao crivo do escalão superior, podendo ser aceitas ou não, sendo o resultado retransmitido ao escalão inferior.

Em síntese, as operações militares são significativamente móveis, sujeitas a interferências do ambiente e do inimigo, tornando as suas comunicações pouco confiáveis e de banda estreita. Possui uma característica de estrutura hierarquia em sua organização que se altera dinamicamente no tempo.

1.2 Objetivos e estrutura do trabalho

Essa dissertação propõe um modelo de gerenciamento de dados que objetive atender estes requisitos de um sistema de comando e controle, propondo um modelo abstrato de dados e um esquema de replicação de dados. Descreve, ainda, técnicas de localização de dados e estações, gerenciamento de transações e replicação de dados. As técnicas propostas são descritas e projetadas de forma integrada.

Um sistema de comando e controle (C2) deve permitir que atualizações realizadas em estações móveis reflitam no banco de dados do sistema, mesmo que tenham ocorrido muito antes da reconexão do sistema. Protocolos de controle de concorrência como pré-ordenação e bloqueio em duas fases não são adequados para sistemas de C2, pois o primeiro provavelmente rejeitará as transações que cheguem muito atrasadas e o segundo, em virtude do ambiente de fraca conectividade, poderá ocasionar bloqueios de longa duração que, ao final, provavelmente não se completarão. Assim, o método de multiversionamento [PC99] parecer ser mais adequado para o modelo.

As transações nas estações móveis, portanto, devem processar as alterações efetuadas localmente para, posteriormente, serem certificadas, ou seja, checadas se tornarão válidas ou não. Duas abordagens podem ser utilizadas neste caso. A primeira seria de transações aninhadas [Chr93]. A segunda seria através de uma distinção de tipos de transações, uma realizada nas estações móveis e outras nas estações fixas [GHOS96], [LS94], [LS95]. Ambas são válidas. Esta dissertação optou por utilizar o segundo modelo. O modelo de certificação adotado para validar as transações é o sugerido em [Bar97].

O modelo deve oferecer as funcionalidades que permitam a mudança dinâmica da hierarquia [PF97].

Deve-se também atentar quanto aos problemas de replicação de dados. Dois aspectos são fundamentais nesta escolha: o que deve ser replicado [KS92], [TLAC95], [Kue94]; e a forma como proceder esta difusão de dados em função do comportamento dos clientes [ABGM90], [BI94], [JBEA95].

O modelo proposto nesta dissertação objetiva integrar as técnicas descritas anteriormente, de forma a atender aos requisitos de um sistema de C2.

O capítulo 2 apresenta uma visão geral de um sistema de comando e controle, incluindo suas características e peculiaridades.

O capítulo 3 descreve as técnicas de gerenciamento de dados em computação móvel. São apresentadas: definições; conceitos básicos; técnicas que permitem a operação em um ambiente de rede com freqüentes desconexões ou fracamente conectado; modelos de transação; técnicas de invalidação de *cache*; e esquemas de distribuição de base de dados que manipulam a localização do usuário e as técnicas de consulta e atualização destas bases.

O capítulo 4 propõe um modelo de gerenciamento de SGBD Móveis em um ambiente de Comando e Controle, apresentando: o modelo abstrato de dados; os componentes do sistema de gerenciamento de dados; o esquema de replicação em duas camadas; e a arquitetura e funcionalidades dos gerenciadores de localização, transações e replicação.

O capítulo 5 apresenta a conclusão do trabalho e possíveis trabalhos futuros.

O anexo da dissertação apresenta os algoritmos do gerenciador de transação de 1ª classe e do *scheduler* e descreve um *framework* para a implementação das técnicas de gerenciamento de dados propostas.

2. Sistemas de Comando e Controle

Comando e Controle (C2) é o processo pelo qual comandantes militares e gerentes civis exercitam a sua autoridade e direção sobre seus recursos materiais e humanos para alcançar objetivos estratégicos e táticos [IEWG97].

C2 é alcançado através da implementação orquestrada de uma série de facilidades ligadas a área de comunicações, pessoal e equipamento e procedimentos para a monitoração, previsão, planejamento, direção, alocação de recursos e geração de opções para alcançar objetivos gerais e específicos. Em organizações empresariais este processo pode ser denominado *gerenciamento operacional de negócios* [IEWG97].

C^3I (Comando, Controle, Comunicações e Informações) é o sistema integrado, adaptado a doutrina, procedimentos, estrutura organizacional, pessoal, equipamentos e facilidades de comunicações que permitem às autoridades de todos os níveis elaborar planos, dirigir e controlar as suas atividades [IEWG97].

2.1 Visão geral de um sistema de comando e controle

Sistemas civis e militares de C^3I são similares em seus requisitos. O sucesso desses sistemas depende da sua possibilidade de produzir e executar decisões oportunamente com informação acurada e precisa. Na indústria, gerentes e líderes de corporação identificam objetivos de mercado e mobilizam os seus recursos para alcançá-los. No campo militar, chefes militares planejam e executam longas e complicadas operações para cumprirem a sua missão. Diretores da indústria mobilizam linhas de produção, gerentes, trabalhadores especializados e seus recursos naturais e manufaturados para produzir produtos superiores. Comandantes militares mobilizam armas, tropas e sofisticados meios de comunicação para defender e conquistar territórios e objetivos militares e políticos.

O levantamento de requisitos do usuário é a chave para um projeto bem sucedido de qualquer Sistema de Informação, assim como para sistemas de C^3I. Neste sentido, o levantamento de requisitos é normalmente mais fácil na área militar do que em organizações civis devido à descrição e detalhes da maior parte das atividades militares estarem bem reguladas e definidas em termos de níveis de comando, conflito, missões chaves e categorias.

Poder-se-ia listar de maneira genérica os elementos básicos de qualquer sistema de C^3I, sejam eles em nível estratégico, tático, de teatro de operações, de forças policiais, e outros similares como [Oliv99]:

- *Subsistemas de missões*, que reúnem informações sobre a localização, movimentos, atividades inimigas e posicionamento dos meios amigos;
- *Subsistemas de navegação*, que informam, às forças amigas, o posicionamento de seus meios;
- *Centros de Comando e Integração*, que reúnem, integram e mostram as atividades dos recursos amigos e inimigos, proporcionando a dinamização dos meios adequados para emprego imediato.

Em seu papel integrador, os Centros de Comando e Integração devem estar em condições de responder à complexidade e às variações rápidas das situações táticas, as quais impõem freqüentes limitações aos comandantes, para compreenderem a situação geral e particular que lhes apresentam, dando-lhe subsídios seguros para adotar uma decisão ótima e resposta apropriada.

O modo natural de implementar um sistema de C^3I é normalmente através da automatização dos procedimentos previstos em manuais e regulamentos, possibilitando a obtenção de resultados de uma maneira, hipoteticamente mais rápida e eficiente.

Um sistema de C^3I tático é iniciado com sensores que examinam o ambiente e geram as primeiras informações as quais são imediatamente avaliadas e comparadas aos recursos disponíveis, permitindo desenvolver um **P**lano de **Op**erações (POp) [Oliv99].

O POp permite visualizar as atribuições e ações necessárias que serão distribuídas, de modo a obter o máximo rendimento dos recursos disponíveis. Essas ações afetam o ambiente, gerando novos dados para os sensores. A partir daí o circuito é fechado e cada elemento é realimentado. Para que o círculo tenha o seu funcionamento normal, é de se supor que seus componentes devam ser unidos por meios de enlace de comunicações tecnicamente confiáveis. Isto nem sempre ocorre, visto que as comunicações poderiam ser rompidas por problemas técnicos ou por interferência de ações do inimigo.

A Figura 2.1 permite verificar vários pontos interessantes. A similaridade com um sistema de controle automático é aparente, com exceção das funcionalidades ligadas à inteligência do sistema. Esta não é imediatamente receptiva a uma determinada quantificação, porém sua importância é tal que se assemelha muito aos sistemas de comando e controle enlaçados pelas comunicações. Pode-se citar como exemplo o sistema de C^3I, onde a letra I significa ***I****nteligência* e, nos níveis menores de comando, ***I****nformações*.

Este melhor emprego das informações para a elaboração das decisões de comando aponta a possibilidade da utilização do sistema de C^3I como multiplicador de força ou do poder de combate. Este conceito pode ser modelado por meio da equação de *Lanchester*, que estabelece que o efetivo de uma força é proporcional ao produto da eficácia de suas armas pelo quadrado do seu quantitativo.

Numa situação hipotética, em face a uma força numericamente superior, de dois para um, é necessário contê-la com uma arma que seja quatro vezes mais eficaz do que as forças inimigas, de modo a alcançar a igualdade. Exemplos do passado são as fortificações, onde apesar de possuir efetivo consideravelmente menor do que as forças agressoras, elas conseguiam resistir ao ataque em virtude da eficácia de sua construção.

Hodiernamente, a eficácia dos sistemas de C^3I é operacionalizado por meio da concentração ágil de forças nos locais de engajamento, de modo a se obter superioridade numérica localizada e, pela assimetria, no engajamento de armas.

O caminho mais eficaz para neutralizar um sistema de C^3I é cortar os enlaces ou nodos do sistema. Quando um único enlace de um sistema de C^3I é suprimido, é necessário que enlaces redundantes, já existentes, sejam imediatamente instalados para assegurar a sobrevivência do sistema. Não somente estes fatores mas outros também devem permitir que se faça a atualização do sistema, quando da restauração das comunicações.

Os sistemas de C^3I táticos são considerados mais complexos e dinâmicos que os estratégicos. Exigem operações mais próximas ao terreno sob o controle inimigo e, por isso, devem ser totalmente móveis ou, pelo menos, transportáveis.

As operações táticas são caracterizadas pelas rápidas mudanças de posição e, em conseqüência, impõem severas limitações aos sistemas C^3I táticos. Estas limitações afetam a habilidade dos comandantes para perceber a situação geral, interagindo às atividades amigas e inimigas, avaliando as ameaças para, depois, decidir e comandar as respostas apropriadas.

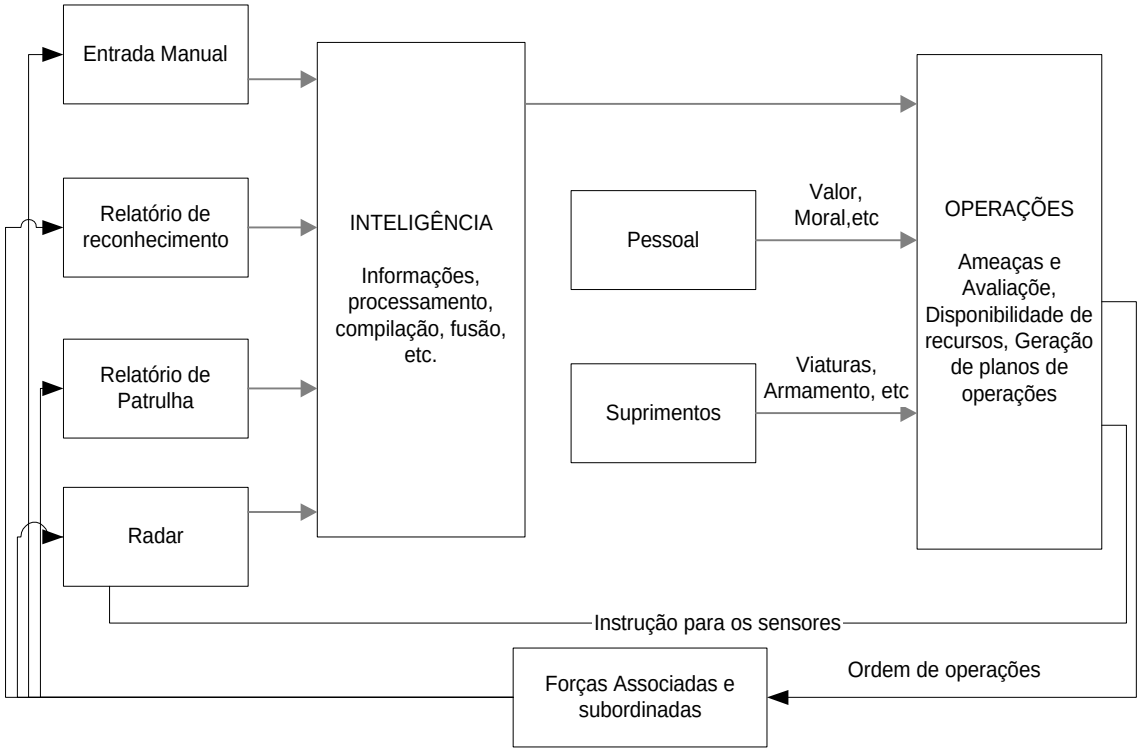

Figura 2.1 - Modelo de Comando e Controle Tático

Pode-se modelar os processos de C2 segundo a seguinte representação exposta na Figura 2.2.

As decisões são tomadas pelo comandante para conter a ameaça e retomar a situação normal, atuando de forma mais eficaz possível diante da ameaça em um dado ambiente.

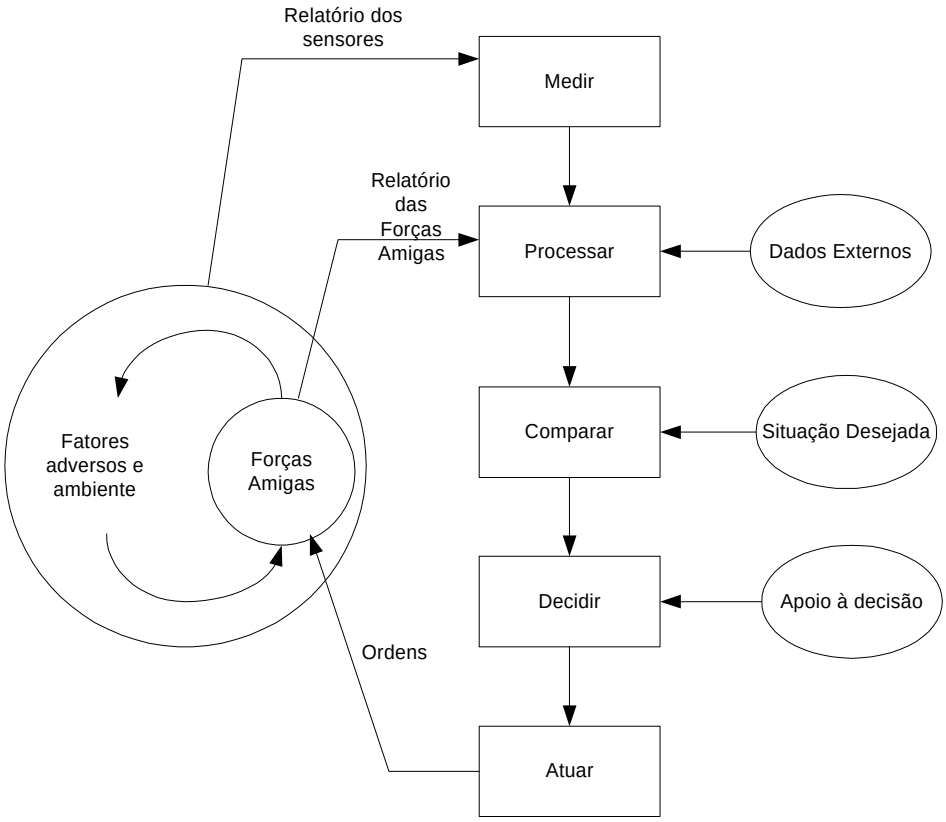

Figura 2.2 - Modelo de Processo de Comando e Controle (C2)

A informação sobre a ameaça, ambiente e os meios dsponiveis são levados à célula C2 por meio de sensores e por parte dos relatórios das forças. Esses dados avaliados são finalmente completados pelas informações oriundas de centros cooperantes. São processados, organizados e registrados para produzir uma imagem da situação do campo de operações.

Finalmente, comparando a situação desejada com a situação existente, os responsáveis pela tomada de decisões podem avaliar os objetivos a alcançar e tomar as decisões apropriadas, que são transmitidas às organizações operacionais.

Os dados gerados pelos sensores e os fornecidos pelas forças são transmitidos ao Centro de Comando e controle, onde se ordenam, processam, relacionam, armazenam e se apresentam aos responsáveis pela tomada decisão, em diferentes formatos, de acordo com as necessidades dos usuários: textos, listas, tabelas, símbolos sobre cartas ou mapas, imagens, etc. As decisões são tomadas com auxílio computacional, resultando em ordens que são preparadas e difundidas para as forças no campo de operações.

O círculo de C2, apresentado na Figura 2.2, termina quando a situação resultante das ações realizadas em cumprimento da execução dos Planos e Ordens de Operações emitidos é informada de retorno pelos sensores e pelas forças. Dependendo do tipo, valor, estrutura e necessidades operacionais da organização considerada, o sistema pode reunir vários tipos de diferentes sensores (radar, vídeo, GPS, etc.) e de redes (linha de transmissão ou infra-estrutura de microondas e rádio móvel), e inclui mais de um centro C2, nos casos em que estão envolvidas forças nacionais, regionais e locais.

A estrutura de comando pode atuar em coordenação com outras organizações que fornecem ou necessitam de informes, como, por exemplo, Forças Terrestres e Aéreas, polícia e bombeiros militares. Esta determina a capacidade de *Interoperabilidade* do sistema.

A definição, especificação, projeto, desenvolvimento e implementação bem sucedida deste sistema requerem um método de sistema integrado que garanta a operação de ponta a ponta dos sensores para os centros de C2 e destes para as forças no campo. A integração do sistema requer um conhecimento total de todas as funções do sistema e uma visão muito clara das interfaces entre os componentes, a nível físico e funcional.

2.2 Um exemplo de aplicação: Forças de Segurança e/ou Forças Armadas em operação de pacificação

Em um panorama hipotético, os sistemas de C^3I são destinados a operações que tenham por finalidade dar apoio à prevenção e a intervenção contra qualquer eventualidade que ocorra à sociedade tais como roubos, terrorismo, narcotráfico, acidentes de estrada, contrabando, etc. ou acontecimentos naturais, como incêndios, enchentes, poluição, inundações, terremotos, etc [Oliv99].

Os sistemas de C^3I proporcionam enlaces entre as forças policiais e de segurança à sociedade, assegurando a ordem pública geral e prestando auxílio aos cidadãos que necessitarem de assistência. Os sistemas de C^3I também fornecem armazenamento eficiente, classificação e acesso aos dados de qualquer informação que possam ser necessários às forças de polícia e segurança em ação no curso de um incidente.

Em se tratando de polícia e segurança, os quatro elementos de C^3I aplicam-se do seguinte modo:

- *Comando*, significa destinar forças (e.g., número de patrulhas destinadas a um bairro, número e tipo de viaturas bombas e postos de bombeiros militares), planejar missões (e.g. o itinerário das patrulhas e sua programação diária) e difundir ordens de atividades ou tarefas;
- *Controle*, refere-se a monitorar a situação (localização das forças e avaliação da condição em que se encontra a missão, incidentes recentes, etc.), gerenciar forças e recursos (e.g. número de bombeiros, posicionamento das patrulhas de polícia encarregadas de controlar uma manifestação) e seguir a correta execução das ordens.
- *Comunicações*, relaciona-se ao intercâmbio de informações, independente do tipo: voz, dados, gráficos, imagem e vídeo.
- *Inteligência* (*Informações* no nível tático), significa obter o conhecimento de qualquer dado que tenha maior ou menor importância para as operações das forças policiais e de segurança (e.g., localização de hospitais, acesso às estações de metrô, plantas dos mais importantes edifícios públicos)

As forças policiais, civis e militares, podem ser genericamente estruturadas em três níveis hierárquicos. As intervenções recentes normalmente são encaminhadas ao nível mais baixo de hierarquia, seja nos sensores ou nas forças desdobradas no terreno. Estes sensores e forças recebem missões e/ou tarefas e são coordenados a nível

intermediário, onde todas as chamadas ou pedidos de auxílio ou reforço são recebidos, monitorados e, algumas vezes, em casos especiais, controlados diretamente pelo nível mais alto da hierarquia.

Ainda que sistemas de C^3I se estruturem sobre base técnicas comuns, são operados ou funcionam em contextos muito diferentes (político, ambiental, população, etc.), em diferentes áreas, de acordo com procedimentos variados e em coordenação com distintas organizações. Todas estas características tornam cada sistema diferente, exigindo cooperação entre os usuários finais e os planejadores para determinar as principais funções do sistema.

Conseqüentemente, é obrigatório enfocar o projeto de forma aberta e flexível, possibilitando posteriores ajustes, modificações e acréscimos sobre uma base estável.

Assim sendo, a arquitetura do software é de fundamental importância, tendo que assegurar a maior independência de qualquer característica específica de cada sistema.

Várias organizações especializadas em sistemas realizaram uma análise detalhada dos requisitos de várias forças policiais e de segurança. Afirmaram que para alcançar o êxito em uma missão crítica, utilizando uma determinada arquitetura de sistema de C^3I é preciso ter: o apoio de comunicações centro a centro e entre centro e as forças móveis; um equipamento telefônico destinado à recepção e ao processamento das chamadas de assistência dos cidadãos; a possibilidade de monitorar em tempo real o estado e a localização das forças, assim como os incidentes em curso e os possíveis eventos; a capacidade de vigilância, por vídeo, de locais e interesse estratégico; o acesso aos dados de inteligência (e.g. criminalidade, documentos de identidade, carteira de motorista, etc.) e os instrumentos para as instruções iniciais e para os relatórios finais após o cumprimento das missões.

2.3 Níveis de comando

Em qualquer organização empresarial existe a presença de uma estrutura hierárquica caracterizada pelo organograma da empresa. O mesmo ocorre, em maior grau, no

ambiente militar, onde existe ainda a relação de hierarquia até entre militares não interligados funcionalmente [IEWG97].

Uma estrutura de organização genérica para uma organização militar poderia ser descrita em uma árvore de três níveis (Figura 2.3). Esta estrutura organizacional de três camadas descreve uma hierarquia cujos níveis correspondem a um comando geral, um comando subordinado e suas peças de manobra. Recursivamente podemos expandir esta estrutura para cima e para baixo. No caso de um sistema de C^3I tático, poderíamos ilustrá-lo tendo como topo da hierarquia o comando de uma divisão, suas brigadas imediatamente subordinadas, e no nível final as Organizações Militares (OM) que poderiam ser batalhões e companhias autônomas (e.g. Companhia de Comunicações). Poder-se-ia também considerar o núcleo de um sistema de C^3I tático, aquele cuja raiz fosse uma brigada, por exemplo.

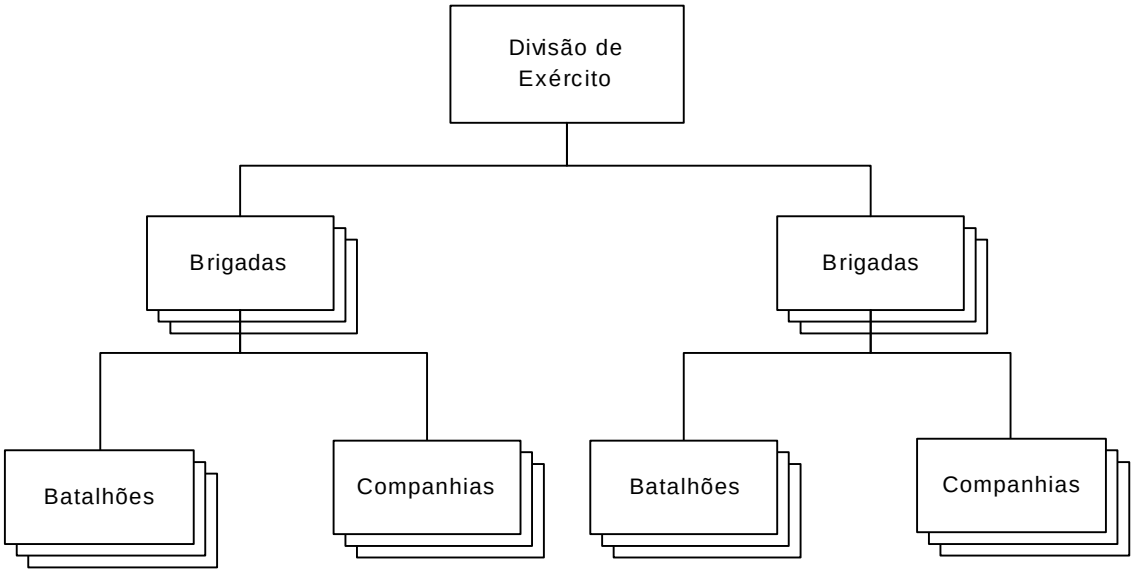

Figura 2.3 - Estrutura genérica de uma Divisão de Exército

Este Sistema Tático possui uma interface com o Sistema Estratégico, o qual também pode ser uma fonte de informação para a tomada de decisões. O Sistema Estratégico de mais alto nível é composto pelo comando conjunto das três forças (Exército, Marinha e Força Aérea), tendo no nível abaixo o comando de cada Força Singular (e.g. Comando do Exército) e no nível final o Exército de Campanha que por sua vez é composto de divisões que compõem a raiz de nosso sistema tático.

As diferenças mais importantes entre estes dois tipos de sistema se referem ao planejamento e a obtenção das informações dos “sensores”. No caso do sistema tático, o tempo de realimentação das informações dos sensores é vital para a sobrevivência em combate. No caso dos Sistemas Estratégicos, os sistemas de apoio à decisão são consideravelmente complexos, em virtude das características e conseqüências das decisões elaboradas neste escalão.

A Doutrina Militar Brasileira preconiza a existência de apenas dois tipos de sistema de Comando e Controle – Tático e Estratégico. Entretanto, a Doutrina Americana e outras incluem ainda um terceiro tipo: o Operacional, que seria a ligação entre os Sistemas Táticos e Estratégico. Ele seria o sistema que controlaria o **T**eatro de **O**perações (TO) [Oliv99].

Cada comando tem um **Q**uartel-**G**eneral (QG) em tempo de paz e de guerra. As informações necessárias requeridas por um comando superior, em tempo de paz, normalmente são obtidas do escalão inferior, seguindo uma linha de comando. Porém, em tempo de guerra, estas informações podem ser obtidas diretamente do mais baixo escalão da hierarquia, de acordo com a situação. As decisões produzidas pelos níveis hierárquicos mais altos são enviadas para as OM subordinadas e o resultado das ações correspondentes àquelas decisões são monitoradas continuamente através de relatórios de atividades remetidos pelos comandos subordinados.

3. Gerência de Dados para Computação Móvel

Comunicações sem fio permitem que computadores portáteis movimentem-se e ainda permaneçam conectados àsua rede. Este paradigma é denominado computação móvel (*mobile or nomadic computing*) [IB93]. A computação móvel adiciona uma nova dimensão à computação distribuída, que é o acesso universal à informação em qualquer tempo e lugar. Esta dimensão cria novas classes de aplicações. Entretanto, a criação destas aplicações pressupõe que mudanças em relação ao gerenciamento de dados sejam executadas.

Atualmente, a construção de sistemas de software para computação móvel tem sido uma tarefa estimulante pelo menos por duas razões. Primeiro, os parâmetros que devem ser levados em consideração não estão totalmente claros e definidos. Segundo, os desafios são enormes: variação e disponibilidade de banda, freqüentes desconexões de rede, mobilidade dos dados, larga escalabilidade em termos de distribuição e número de elementos computacionais. Computação móvel é o pior caso de computação distribuída, já que hipóteses fundamentais sobre conectividade, imobilidade e escala não são mais válidas.

3.1 Infra-estrutura de Comunicações e Características de Aplicações em Computação Móvel

3.1.1 Arquitetura física para computação móvel

Personal Communication Network (PCN), *Personal Communication System* (PCS), *Universal Mobile Telecommunication System* (UMTS) são os nomes que representam a infraestrutura de rede que objetiva fornecer serviços de comunicação sem fio de

cobertura universal. Estas arquiteturas objetivam acessar recursos de rede usando diferentes tipos de meios de comunicações, independentemente da localização do usuário e a informação que será acessada.

Tendo em vista que esta arquitetura ainda não foi plenamente desenvolvida, a estrutura de telefonia celular existente está sendo utilizada para estes serviços [IB94a] (e.g., WAP). A configuração de rede consiste de uma rede fixa estendida por um número de estações móveis, que se comunicam com transceptores estacionários denominados estações base ou *mobile support stations* (MSS). A Figura 3.1 ilustra esta arquitetura. A área de cobertura de um MSS é denominada célula. A estação móvel comunica-se com as outras unidades, fixas ou móveis, apenas através da estação base da célula onde reside.

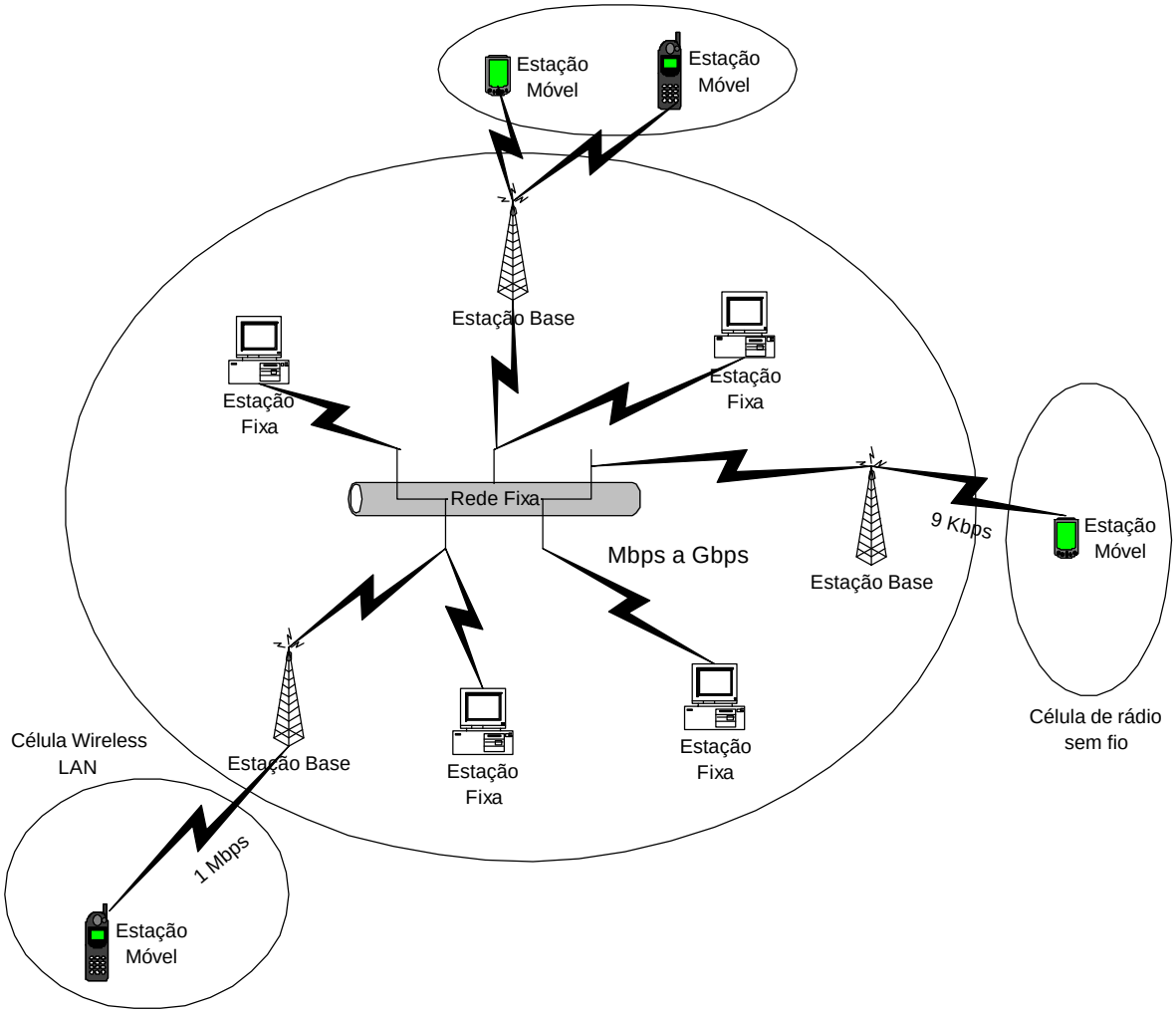

Figura 3.1 - Arquitetura de rede móvel

O tamanho da célula pode variar significativamente em tamanho [FZ94]. Por exemplo, um satélite pode cobrir uma área com mais de 700 Km de diâmetro, um transceptor de celular cobre uma área de 16 Km de diâmetro e uma WLAN cobre a área de um edifício. Com o barateamento e miniaturização das estações bases e a necessidade de aumentar a sua capacidade, espera-se que o tamanho das células diminua cada vez mais. A tendência de células cada vez menores é também justificada pela necessidade de aumentar o número de usuários através dos esquemas de reuso de freqüências.

Quando uma estação móvel movimenta-se, ela pode cruzar o limite de uma célula e entrar numa área coberta por uma estação base diferente. Este processo é denominado *handoff*. O processo de *handoff* atualiza os bancos de dados que mantém a localização dos usuários móveis. No caso de comunicação de voz, manter a comunicação ininterrupta durante os *handoffs* é relativamente trivial. Entretanto, no caso de transmissão de dados, o gerenciamento de *handoff* torna-se sofisticado, devida a pequena perda de dados que pode ser tolerada.

As características das estações móveis podem variar. A arquitetura PCS deve suportar desde pequenos e leves até os clientes mais robustos. Uma estação móvel é, normalmente, dependente de bateria. Esta restrição faz com que as estações móveis estejam freqüentemente, voluntariamente ou involuntariamente, desconectadas.

Embora o FCC já tenha alocado uma faixa de freqüência (2 GHz nos EUA), a arquitetura PCS ainda não está finalizada. Espera-se que os protocolos de comunicações e outras especificações sejam definidos de maneira a satisfazer a maioria dos requisitos da computação móvel. O gerenciamento de rede e outras funções de controle serão distribuídos pela rede, aumentando a disponibilidade e escalabilidade de vários serviços.

3.1.2 Conseqüências e limitações da arquitetura

3.1.2.1 Mobilidade

A localização dos elementos móveis e, portanto, o seu ponto de localização e ligação com a rede fixa modificam-se à medida que se movimentam.

As conseqüências desta mobilidade são numerosas. A configuração de um sistema que inclui elementos móveis não é estática. No projeto de algoritmos distribuídos, não se pode confiar em uma topologia fixa. O centro de atividade, a carga do sistema e a noção de localização mudam dinamicamente.

O custo da localização de elementos móveis é adicionado ao custo das comunicações que a envolvem. Estruturas de dados eficientes, algoritmos e planos de execução de consultas devem ser planejados para representar, gerenciar e consultar a localização de elementos móveis, com a rápida mudança de dados.

A conectividade variará significativamente em performance e confiabilidade. Por exemplo, um cliente móvel na rua baseia-se em redes de faixa estreita. Quando está, por exemplo, em um edifício, pode ter acesso a uma rede de alta velocidade. Podem existir áreas sem cobertura adequada, resultando em desconexões. O número de dispositivos dentro de uma célula varia com o tempo, variando também a carga da estação base e a largura de banda disponível. Assuntos como segurança e autenticação também são importantes.

3.1.2.2 Meio de transmissão

As redes sem fio são mais caras, possuem menor largura de banda e são menos confiáveis que as redes com fio.

As redes sem fio encontram mais obstáculos devido à interação do ambiente com o sinal de transmissão. Enquanto que o crescimento das taxas de transmissão nas redes

fixas tem sido extraordinário (*Ethernet*: 10 Mbps, FDDI: 100 Mbps, ATM 155 Mbps), os produtos de comunicação sem fio tem alcançado faixas de 9-19 Kbps, exceção feita às redes locais sem fio (*Wireless LANs*), que alcançam magnitudes de Mbps. Como o meio é ainda dividido entre vários usuários que estão na célula, a velocidade pode ser ainda menor. Para a transmissão de rádio, a taxa de erro é tão alta que a largura efetiva de banda é menor do que 10 Kbps.

Conseqüentemente, a conectividade é fraca e a freqüência é intermitente devido ao fato de que elementos móveis voluntariamente operam desconectados por longos períodos de tempo. Isto resulta em uma forma de computação distribuída, que assume conectividade drasticamente diferente do que nos sistemas distribuídos tradicionais onde as desconexões são raras.

Normalmente existe um canal mais largo para a difusão de dados da estação base para os clientes que estão dentro da sua célula, sendo isto útil para a disseminação de informação para um grupo de clientes.

3.2 Operação em um ambiente de rede com desconexões ou fracamente conectado

3.2.1 Operações em ambientes com desconexões

Quando uma operação de desconexão vai ser executada previsivelmente, os itens de dados são transmitidos para o cliente móvel para permitir que este opere autonomamente durante a desconexão. A operação de pré-carga em razão deste tipo

de desconexão é denominada de *hoarding*. A operação de desconexão pode ser descrita com sendo uma transição entre três estados [KS92].

Anteriormente àdesconexão, a estação móvel está no estado de *Data Hoarding*. Neste estado, os itens de dados necessários àoperação são carregados para a estação móvel. Os dados poderiam ser simplesmente movidos da estação fixa para a estação móvel, tornando-se inacessíveis para outras estações. Como alternativa, os itens de dados deveriam ser replicados na estação fixa.

O tipo de objetos de dados transferidos para a estação móvel depende da aplicação e do modelo abstrato de dados. Por exemplo, no caso de sistemas de arquivos, os dados podem ser arquivos, diretórios ou volumes. No caso de SGBDs, os dados podem ser relacionamentos ou visões. No caso de sistemas de gerenciamento de *workflow*, os dados podem ser tarefas de *workflow*. Os objetos de dados podem carregar informações adicionais, tais como operações executáveis (leitura, escrita, etc.). No caso de operações de desconexão previsíveis (por exemplo, quando o usuário entra numa região onde o preço das comunicações é caro), a operação de *data hoarding* pode ser executada apenas momentos antes da desconexão. No caso de operações de desconexão menos previsíveis, o *hoarding* deve ser executado periodicamente.

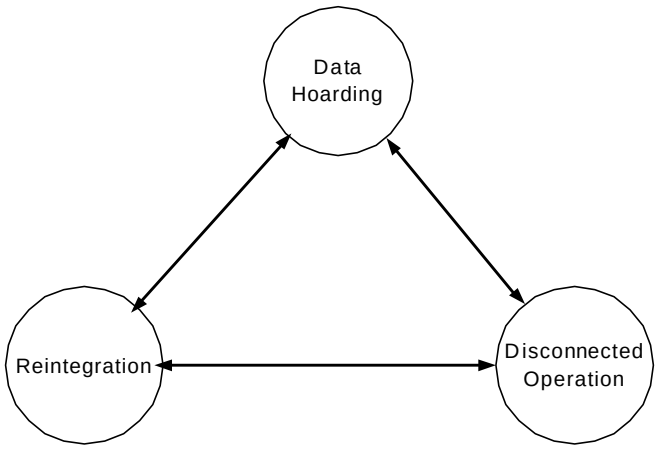

Figura 3.2 - Estados da operação

Um problema importante é como antecipar as necessidades futuras de dados das unidades móveis. Uma abordagem é permitir aos usuários especificar que itens de dados serão pré-carregados. Uma outra abordagem seria o uso do histórico de informações sobre acesso de dados realizados anteriormente para predizer as futuras necessidades de dados. Que tipos de dados serão pré-carregados dependerá do tipo de

aplicação que o sistema utilizará. Um assunto que aumenta a complexidade do *hoarding* é o caso de recursos compartilhados com outras estações que necessitam operar sobre alguns itens de dados concorrentemente com o site desconectado. Deve-se levar em consideração a probabilidade de conflito de operações para decidir quais dos itens que sofrerão *hoarding* irão aprimorar a eficiência das operações de desconexão.

Com o evento da desconexão, a unidade móvel entra no estado desconectado. Neste estado, a unidade móvel pode somente utilizar dados locais. Solicitações de dados que não estejam disponíveis não poderão ser respondidas e retornarão mensagens de erro. Estas solicitações poderão ser inseridas em uma fila, de forma que sejam respondidas quando da reconexão. Ao que se refere a atualizações de dados compartilhados, existem duas abordagens. Na abordagem pessimista, as atualizações são executadas apenas em um site, utilizando *locking* ou alguma forma de *check-in*/*check-out*. Na abordagem otimista, as atualizações não são restringidas nos sites (mais de um) proporcionando uma maior probabilidade de erro em operações conflitantes. As atualizações nas unidades móveis são registradas no SGBD local. O tipo de informação afeta a eficiência de reintegração das atualizações quando da reconexão, bem como a efetividade dos registros de atualização. A otimização do *log*, mantendo-o em tamanho pequeno é importante para reduzir o tempo para a propagação das atualizações e a reintegração durante a reconexão. A otimização do *log* pode ser executada ou durante a operação de desconexão, incrementando a cada vez que uma nova operação é inserida dentro do *log*, ou como um passo de pré-processamento antes de realizar a propagação ou aplicar o *log* na reconexão.

No evento da reconexão, a estação móvel entra no *estado de reintegração*. Neste estado, as atualizações realizadas nas estações móveis são reintegradas com as atualizações realizadas por outros sites. A reintegração de atualização é normalmente executada através da re-execução do *log* nas estações fixas. Se as operações executadas nos sites desconectados irão ser aceitas, dependerá da semântica de concorrência adotada pelo sistema. Esta semântica poderá checar pontos como a serialização das transações e resolver problemas de concorrência em um mesmo objeto.

3.2.1.1 Operações de desconexão em Sistemas de Arquivos

O principal propósito dos sistemas de arquivos em possibilitar a continuidade da operacionalidade nas operações de desconexão é estender o gerenciamento de cache, levando em consideração este novo estado de desconexão [KS92]. O gerenciamento de cache na operação de desconexão é diferente do mesmo gerenciamento quando o sistema está conectado. Inicialmente, falhas de cache não podem ser corrigidas. Assim, atualizações no sistema não poderão ser propagadas a partir do elemento desconectado. Quaisquer atualizações devem ser integradas quando da reconexão.

A operação de *hoarding* é similar àbusca prévia (*prefetching*) usada nos sistemas de banco de dados, com o intuito de melhorar a performance. A busca prévia é executada continuamente, em contraste com o *hoarding*, que busca manter o overhead de tráfego baixo. O *hoarding* é considerado mais crítico que a busca prévia, porque durante as desconexões, uma falha de *cache* não pode ser oferecida. Um parâmetro importante é a unidade de *hoarding*, variando desde um bloco de disco, um arquivo, grupos de arquivos e diretórios. Outro parâmetro importante é quando iniciar a operação de *hoarding*.

Como descrito anteriormente, a decisão de que quais arquivos poderiam ser armazenados no cache pode ser: (a) assistida por instruções explicitas fornecidas pelo usuário, ou (b) executada automaticamente pelo sistema através da utilização de informações implícitas, que é mais freqüentemente baseada no passado histórico das referências de arquivos. O *Coda File System* [KS92] combina as duas abordagens na decisão de que dados irão se submeter à operação de *hoarding*. Os dados são transmitidos com base nas prioridades de uma combinação de referência histórica recente e os arquivos de *hoard* definidos pelo usuário. Um método *tree-based* é sugerido em [TLAC95] que processa o histórico das referências de arquivos para construir uma árvore de execução. Os nós da árvore representam os programas e os arquivos de dados referenciados. Um caminho entre o nó-pai A para o nó-filho B existe ou quando o programa A chama o programa B, ou quando o programa A usa o arquivo B. Um GUI (*Guide User Interface*) auxilia o usuário na tarefa de que

arquivos irão sofrer *hoard*. *Kuening* em [Kue94] descreve um esquema de previsão de cache baseado no comportamento passado do usuário. Os arquivos são automaticamente transmitidos com base em uma medida denominada “distância semântica” que quantifica a relação entre eles. A medida escolhida é a referência local de um arquivo A para um arquivo B. Esta distância pode ser informalmente definida como o número de referências de arquivos que separam duas referências adjacentes entre A e B no histórico de referência de arquivos passados.

Enquanto está desconectada, a unidade móvel utiliza apenas os dados disponíveis em seu cache. Falhas de cache são tratadas como erros ou geradoras de exceção. A aplicação pode ou bloquear a sua operação ou continuar trabalhando. Se ao cliente é permitido executar atualizações, estas são realizadas localmente. No Coda, um *log* registra todas as chamadas de argumentos do sistema [KS92], bem como as versões de estado de todos os objetos referenciados por uma chamada do sistema. Duas otimizações são executadas antes que um registro seja adicionado ao *log*. Primeiro, qualquer operação que sobre-escreva o efeito de operações anteriores pode cancelar o registro destas no *log*. Segundo, uma operação inversa (e.g., *rmdir*) cancela a operação feita anteriormente (e.g., *mkdir*).

Atualizações executadas num site desconectado podem conflitar com operações realizadas por outros sites. Assim, atualizações de dados no cache podem ser consideradas tentativas. No Coda, o *log replay* é executado como uma transação simples, bloqueando todos os objetos referenciados dentro do *log*. São utilizadas estratégias diferentes para manipular atualizações concorrentes em arquivos e em diretórios. Isto é devido ao fato de existir conhecimento de semântica suficiente em diretórios que permitem a resolução transparente de conflitos. Por exemplo, a resolução de diretórios falha apenas se um nome criado colide com um nome existente, ou se um objeto atualizado pelo cliente ou servidor já havia sido deletado por outro, ou se os atributos dos diretórios foram modificados pelo servidor ou cliente. No Coda, a resolução de arquivos é baseado no *application-specific resolvers* (ASRs) *per file* [KS95]. Um ASR é um programa que encapsula o conhecimento necessário àresolução de arquivos, e é invocado pelo cliente quando uma divergência entre cópias é detectada. São utilizadas regras para selecionar o ASR apropriado. As mutações de ASR são executadas localmente no cache do cliente e escritas no

servidor depois que o ASR completa as suas tarefas. A execução de um ASR tem semântica transacional garantida. Se nenhum ASR é encontrado ou se a execução do ASR falha, um código de erro indicando um conflito é retornado. No caso de conflitos não resolvidos, uma ferramenta de reparação manual é oferecida ao cliente.

No caso de sistemas de arquivos, apenas os conflitos de escrita/escrita são detectados, porque produzem cópias diferentes. Conflitos leitura/escrita não são considerados. Tais conflitos ocorrem, por exemplo, quando o valor de um arquivo lido por um cliente desconectado não é o mais recente em virtude de uma atualização pelo servidor, depois da desconexão do cliente.

3.2.1.2 Operações de desconexão em SGBD

A granularidade do *hoarding* em um SGBD relacional pode variar desde tuplas, conjunto de tuplas até relações completas. Uma abordagem lógica seria executar o *hoarding* através de consultas, isto é, pela pré-carga de objetos de dados que constituirão respostas a uma determinada consulta.

Deve-se decidir também quais dados serão submetidos a *hoarding*, identificando os conjuntos de dados através das consultas. Como no *Coda*, isto pode ser implementado como uma combinação de escolhas do usuário e também baseado no histórico do sistema.

No caso de basear-se no histórico, o sistema automaticamente executa *hoarding* no conjunto mais comumente utilizado ou os últimos itens referenciados. O uso das referências históricas para deduzir dependências entre itens de banco de dados é mais difícil de identificar do que as dependências entre arquivos. A integridade e a completude também devem ser levadas em consideração.

A consistência durante operações de desconexão tem sido extensivamente detalhadas no contexto de particionamento de redes. Dentro deste contexto, uma falha na rede

particiona os sites de um sistema de banco de dados distribuído em *clusters* desconectados. Uma bibliografia que trata este assunto é [DGMS85]. Geralmente, tais abordagens podem ser classificadas ao longo de duas dimensões ortogonais. A primeira preocupa-se com a negociação entre consistência e disponibilidade. A outra dimensão preocupa-se com o nível de conhecimento semântico usado na determinação da corretude.

Existem determinadas individualidades que devem ser consideradas quando da revisão do problema do particionamento de rede no caso de computação móvel. O particionamento de rede é usualmente abordado em modelos *peer-to-peer*, onde as transações executadas em qualquer partição são de igual importância. Em computação móvel, as transações nas estações móveis são freqüentemente consideradas como de segunda classe. Outro aspecto é a freqüência das desconexões em computação móvel. Particionamento de redes são consideradas como comportamento de falha, ao passo que desconexões em computação móvel são consideradas operações previsíveis.

Uma tendência comum é tentar confirmar as transações executadas na unidade móvel e tornar os seus resultados visíveis nas transações subseqüentes dentro da mesma unidade. Quando esta unidade reconecta-se ao sistema, um processo de certificação acontece, durante o qual a execução de qualquer tentativa daquelas transações confirmadas anteriormente na estação móvel que estava desconectada é validada ou não, segundo um critério de corretude adotado pelo sistema. Se estiver de acordo com o critério, a transação é de fato confirmada. Caso contrário, a transação deve ser abortada ou compensada. Tais ações podem ter efeito em cascata sobre outras tentativas de transações confirmadas.

O conceito de IOT (*Isolation-Only Transaction*) é introduzido em [LS94],[LS95] para fornecer suporte a transações em sistemas de arquivos. O IOT é uma seqüência de operações de acesso a arquivo. A execução da transação é realizada inteiramente no cliente e nenhum resultado parcial é visível nos servidores. Uma transação T é denominada transação de primeira classe, se ela não teve nenhum acesso a arquivo particionado, caso contrário é denominada transação de segunda classe. O resultado de uma transação de primeira classe é imediatamente confirmado nos servidores, ao passo que a transação de segunda classe é armazenada no cache do cliente e visíveis apenas nos acessos subseqüentes no mesmo cliente. As transações de segunda classe

têm a garantia de serem localmente serializáveis entre si. Quando da reconexão, a transação de segunda classe é validada entre uma de duas restrições de serialização. A primeira é a serialização global, que significa que se o resultado de uma transação local pendente foi escrito no servidor sem modificações, ela seria serializável com todas as transações confirmadas anteriormente. A segunda restrição é um critério mais exigente de consistência denominado *Global Certifiability* (GC), que requer que uma transação pendente seja globalmente serializável não apenas todas as transações anteriormente confirmadas ou resolvidas, mas com as transações que foram confirmadas ou resolvidas no sistema durante o período de desconexão. Intuitivamente, o GC assegura que todos os dados acessados por uma transação pendente são imodificáveis no servidor entre o início e a validação da transação. Transações que não estejam neste critério são solucionadas incrementalmente, isto é, uma a uma, de acordo com a sua ordem local de serialização. Uma transação invalidada é abortada, re-executada ou um ASR é automaticamente chamado. Reparações manuais são consideradas como último recurso.

No esquema *two-tier replication* [GHOS96], os dados replicados tem duas versões nos nós móveis: mestre e tentativa. Uma versão mestre registra os mais recentes valores recebidos enquanto o site estava conectado. Uma versão tentativa registra todas as atualizações locais. Existem dois tipos de transações análogas às transações de primeira e segunda classe do IOTs: tentativa e base. A transação tentativa trabalha nos dados locais de tentativa e produz dados de tentativa. Uma transação base trabalha somente nos dados mestre e produz dados mestre. Transações base envolvem apenas sites conectados. Quando da reconexão, as transações de tentativa são reprocessadas como transações base. Se elas falham em adaptar-se em algum critério específico de aceitação da aplicação, elas são abortadas e uma mensagem é retornada para a estação móvel.

3.2.2 Operações em ambiente de fraca conectividade

Fraca conectividade é o equivalente ao fato do serviço de comunicações oferecido ser caro ou de baixa velocidade. Em tais redes, a conectividade é freqüentemente perdida por curtos períodos de tempo. Fraca conectividade impõe várias limitações que não estão presentes quando a conectividade é normal e força a revisão de vários sistemas de protocolo. Uma característica da fraca conectividade em computação móvel diz respeito à sua variação em intensidade. A conectividade em computação móvel varia em custo, largura de banda fornecida e confiabilidade. O objetivo das principais propostas para fraca conectividade é o uso prudente da largura de banda. Estas propostas procuram negociar "fidelidade" e redução do custo de comunicação.

3.2.2.1 Sistemas de Gerenciamento de Banco de Dados em ambientes de fraca conectividade

Uma estação móvel pode desempenhar diversos papéis num SGBD Distribuído. Ela pode, por exemplo, simplesmente submeter operações para que sejam executadas no servidor. Neste caso, ele pode determinar que as transações sejam executadas uma a uma ou que um conjunto de transações seja uma unidade atômica [JBE95].

O controle de concorrência no caso de transações distribuídas envolvendo estações fixas e móveis é complicado. Transações que acessam dados tanto em estações móveis quanto em estações fixas trazem consigo um grande *overhead*. Por exemplo, o caso de um protocolo pessimista de concorrência que requer que as transações realizem bloqueios em sites múltiplos. Neste caso, as transações poderiam ficar paradas por longos períodos se requisitassem bloqueio em sites que estivessem desconectados. Técnicas como *timestamps* podem ocasionar um grande número de transações abortadas já que as operações podem ser propagadas com retardo em redes de baixa velocidade.

Para evitar retardos impostos pela baixa velocidade da rede, modelos de *open-nested transactions* [Chr93] e esquemas de multiversão são mais apropriados.

Em [Chr93], uma transação móvel envolvendo estações fixas e móveis não são tratadas como uma unidade atômica, mas como um conjunto de componentes de transação relativamente independentes, alguns dos quais rodando unicamente nas estações móveis. Os componentes das transações podem ser confirmados sem aguardar a resposta de outras transações do conjunto. Em particular, como no caso da desconexão, as transações realizadas na estação móvel são visíveis somente para esta e seu resultado são visíveis para as transações locais subseqüentes. Estas transações são certificadas para correção nas estações fixas, posteriormente. As estações fixas podem difundir para as estações móveis informações sobre transações confirmadas antes do evento da certificação [Bar97]. Esta informação pode ser utilizada para reduzir o número de transações abortadas.

As transações que são executadas somente na estação móvel são denominadas *fracas* [PB95], [Pit96], enquanto que as transações restantes são denominadas *estritas*. As cópias fracas são apenas tentativas de confirmação e podem armazenar valores obsoletos. *Transações fracas* atualizam apenas cópias fracas, enquanto que *transações estritas* acessam cópias *estritas* localizadas em qualquer site. As cópias fracas são integradas com as cópias *estritas* quando um limite definido pela aplicação que permite a mudança entre cópias fracas e *estritas* é alcançado. As aplicações nos sites fracamente conectados podem optar pelo uso de *transações estritas* quando elas desejarem consistência *estritas*. As *transações estritas* são mais lentas que as transações fracas por envolverem o *link* de comunicação *wireless*, porém garantem consistência imediata em contrapartida.

Bayou [TDPS91] não suporta transações. Bayou é uma arquitetura *peer-to-peer* com um conjunto de servidores replicados fracamente e conectados um ao outro. Neste esquema, uma aplicação do usuário pode ler ou escrever qualquer cópia disponível. Escritas são propagadas para outros servidores durante o contato entre os pares, denominado como *anti-entropy sessions*. Como na replicação *two-tier*, cada servidor mantém duas visões do banco de dados: uma cópia que reflete apenas os dados confirmados e outra cópia que também reflete as tentativas de escrita conhecidas pelo servidor. Eventualmente, cada escrita é confirmada utilizando um esquema de *commit* primário, onde um servidor é designado como primário e toma a responsabilidade de confirmar as atualizações. Em virtude dos servidores poderem receber atualizações

vindas dos usuários e de outros servidores em ordens diferentes, os servidores podem necessitar desfazer o efeito de algumas tentativas anteriores de uma operação de escrita e reaplicá-las. Bayou possibilita checagem de dependência para detecção automática de conflitos e *procedures* de fusão para resolução. Ao invés de transações, Bayou trabalha com sessões. Uma sessão é uma abstração para uma seqüência de operações de leitura e escrita realizadas durante a execução de uma aplicação.

3.2.2.2 Commit distribuído

O processamento do *commit* em ambientes distribuídos é geralmente implementado através do protocolo *Two-Phase Commit* (2PC) e suas variações. O modelo proposto nesta dissertação aplica diretamente o *Presumed Abort Two-Phase Commit* [OV99].

Se o papel das estações móveis é limitado em apenas emitir pedidos, o *Two-Phase Commit* pode ser aplicado sem modificações, em função do controle da transação está implicitamente designado a um site na rede fixa.

Algumas modificações podem ser implementadas no caso de redes móveis.

Jung em [JBE95] propõe que o fluxo do 2PC seja utilizado para carregar algumas informações extras para suportar o esquema de controle de concorrência. O processo de *commit*, entretanto permanece inalterado. Durante a desconexão as transações nos sites móveis são apenas tentativas de *commit*. Na integração, entretanto, estas tentativas fazem parte de uma transação distribuída que necessita ser confirmada.

Se a estação móvel participa ativamente dentro de uma transação móvel como um *partner* igual com seus dados próprios, então o processamento de *commit* deve levar em consideração os novos problemas que são impostos pela fraca conectividade. Os Protocolos de *Commit* deveriam ser estudados e melhorados levando em consideração as longas desconexões, conectividade intermitente, alta latência, largura de banda baixa, comunicação cara e mobilidade.

Limitações ao 2PC clássico, tal como o bloqueio [Ske91],[SBCM95], podem ocorrer mais freqüentemente no ambiente móvel e soluções como o processamento heurístico para estes problemas são necessárias.

O processamento de *commit* pode tirar vantagem das características de broadcast para otimizar a sua performance. As otimizações de *commit* podem ser ajustadas para a computação móvel. Enquanto o cliente móvel pode iniciar o processamento de *commit*, pode ser prudente transferir sempre, através da otimização do ultimo agente de *commit*, a coordenação de *commit* a um parceiro mais confiável.

3.2.2.3 Mobilidade

A configuração de um sistema distribuído que inclui elementos móveis é extremamente dinâmica. As unidades móveis comunicam-se diretamente com diferentes estações base enquanto entram e saem de células. Esta mobilidade também compreende o deslocamento de perfil e atividades de leitura e escrita [IB93]. Em função disso, itens ou tarefas deverão estar próximos destes elementos móveis para melhorar a performance. O custo de comunicação na computação móvel é diferente da comunicação fixa. Ela é aumentada pelo custo de busca que é necessário para determinar a localização exata das estações móveis [IB92]. Segundo, ela também pode ser baseada na duração da conexão como oposição ao número de mensagens enviadas.

Em virtude das mudanças de diversos parâmetros em função do tempo num ambiente de computação móvel, a alocação estática de dados, de certo, não é apropriada.

Huang em [HW94] descreve uma distinção entre o custo de replicação em ambientes de computação fixa e móvel, baseada na hipótese de que o custo de I/O torna-se insignificante, devido às taxas de comunicação. Um algoritmo de alocação de dados tem a propriedade de fazer com que um processador ao ter uma cópia, salvá-la em seu banco de dados. Assim, o algoritmo dinâmico é superior, comparado com o esquema estático.

O caso de usuários de computação móvel acessando dados de um banco de dados localizado em uma estação estática é considerado em [HSW94]. O problema discutido é o benefício, em termos de custo de comunicação, em se manter uma cópia local de um item de dados na estação móvel. Intuitivamente, manter uma cópia local é apropriado para itens de dados que são freqüentemente lidos, comparados àtaxa com que são atualizados. Haveria então três alternativas para a alocação de cópias: no servidor, no cliente móvel e de servidores geograficamente (ou tecnicamente) localizados para o cliente [IB92]. A principal consideração é o custo de busca para localizar cópias em estações móveis. Assim, além das freqüências relativas das operações de leitura e escrita, a escolha de alocação também dependerá da mobilidade e do esquema de gerenciamento de localização.

Em [BGM94] são sugeridos algoritmos dinâmicos para alocação de dados replicados que podem ser obtidos simplesmente através de transações de atualização nos diretórios. Diretório é uma estrutura de dados que especifica os sites que contém cópias de itens de dados. Diretórios e outros itens de dados são tratados da mesma maneira. Para localizar cópias de dados, as transações primeiro executam bloqueios de leitura nos diretórios. Uma mudança na alocação das cópias é uma transação de atualização no diretório. Atualizações no diretório são serializáveis com transações regulares. Por fim, um diretório "*primary-by-row*" de gerenciamento é descrito, onde a reconfiguração é inicializada no site que mantém a cópia e deseja repassá-la para um outro site.

3.3 Difusão de dados

Nos sistemas tradicionais cliente-servidor, os dados são enviados baseados na demanda do cliente. Nesta arquitetura, o cliente solicita explicitamente dados ao servidor. Quando uma solicitação é recebida pelo servidor, este localiza a informação de interesse e a retorna para o cliente. Esta forma de entrega de dados é denominada *pull-based* ou baseados em demanda.

Em ambientes móveis, estações fixas possuem uma banda passante relativa alta, que permite difundir dados para as estações móveis. Esta facilidade é a base para o modelo de transmissão de dados denominado *push-based ou baseados em difusão.* Neste modelo, os servidores difundem repetidamente dados para os clientes, sem a necessidade de uma solicitação específica. Clientes monitoram o canal e recuperam os dados de que necessitam. A idéia de difusão de dados não é nova. Trabalhos anteriores foram realizados na área de sistemas de Teletexto e Videotexto.

3.3.1 Modelos de difusão

O modelo *push-based* é adequado nos casos em que a informação é transmitida para um grande número de clientes que possuam interesses em comum. Neste caso, o servidor poupa a emissão de diversas mensagens que seriam transmitidas individualmente nos sistemas sob demanda. Isto ainda minimiza problemas de sobrecarga no servidor em razão de solicitações de múltiplos clientes. O modelo *push-based* é escalável porque sua performance não depende do número de clientes que escutam o canal. O modelo *pull-based*, por outro lado, não é escalável, sendo limitado pela capacidade do servidor ou da rede. Uma das limitações do *push-based* é que o acesso é apenas seqüencial. Os clientes necessitam esperar até que os dados solicitados apareçam no canal. Assim, a latência de acesso degrada com o volume de dados difundido. Nos sistema *pull-based*, os clientes têm um papel mais ativo e podem explicitamente requerer dados do servidor.

Os sistemas *push-based* e *pull-based* podem ser combinados. Isto pode ser feito quando os clientes possuem um canal de comunicação no sentido do cliente para o servidor, usado para transmitir mensagens para o servidor. Este canal tem duas denominações na literatura: *uplink* ou *backchannel*. Esta abordagem é denominada entrega de dados híbrida (*hybrid data delivery*). Um fator importante na entrega híbrida é que um mesmo canal usado pelo servidor para comunicar-se com seus clientes é utilizado para a transmissão por difusão e também para as transmissões das respostas do servidor às solicitações de demanda. Neste caso, técnicas para compartilhamento eficiente do canal são importantes. Os clientes podem utilizar o

backchannel para diversas finalidades tais como *feedback* e informação de perfil para os servidores. Estes podem também utilizar o *backchannel* para requisitar dados diretamente, a exemplo de dados críticos, os quais clientes não poderiam esperar que estes (os dados críticos) aparecessem no canal.

O *backchannel* é utilizado conjuntamente com o *caching* dos clientes [AFZ97] para permitir que estes solicitem páginas que não estejam mais disponíveis ou desatualizadas em seu cache local. Para evitar sobrecarga do servidor em função das solicitações, várias técnicas foram propostas. Em uma delas, o cliente envia uma solicitação para um dado item *i* apenas se o número de itens agendados para aparecer antes de *i* no canal de difusão seja maior que o parâmetro limite. Outra técnica sugerida é ajustar apropriadamente uma banda de resposta, isto é, destinada à respostas das solicitações.

Uma maneira de realizar a entrega híbrida é particionar os itens de dados do SGBD em dois conjuntos [SRB97], [IB94]: para difusão pura (*push-based*) e o outro disponível para dados sob demanda (*pull-based*). Determinar qual subconjunto do SGBD será difundido dependerá de fatores tais como padrão de acesso dos clientes e a capacidade do servidor em atender as solicitações. A técnica descrita em [SRB97] é apresentada como o ajuste contínuo do conteúdo da difusão de dados na busca do *hot-spot* do banco de dados. O *hot-spot* é calculado pela observação de falhas do *broadcast* indicadas por pedidos de dados que não estão no canal de difusão. Estes pedidos possibilitam ao servidor ter uma estatística sobre a demanda atual. Na técnica apresentada em [IB94], o critério de particionamento do banco de dados é minimizar os pedidos vindos pelo *backchannel* através da restrição do tempo de resposta, que deve estar abaixo de um limite máximo pré-definido.

A mobilidade dos usuários é também crítica para determinar o conjunto de dados a ser difundido. Células podem diferir em seu tipo de infraestrutura de comunicação e também em sua capacidade em atender pedidos. Quando os clientes movem entre células, a distribuição dos pedidos para dados específicos em cada célula é modificada. Duas variações de um algoritmo adaptativo que leva em conta a mobilidade dos usuários entre células de uma arquitetura celular são propostas em [DCKVK97]. Este algoritmo seleciona os dados a serem difundidos baseados nos perfis dos usuários e no registro em cada célula.

3.3.2 Caching e broadcast

Clientes normalmente armazenam itens de dados para diminuir sua dependência da política do servidor em disseminar dados. Os parâmetros da política podem não ser ótimos para determinados clientes, quando o servidor distribui seus dados para uma grande população de clientes com diversas necessidades. Outro fator a se considerar é que as necessidades específicas de acesso dos clientes podem mudar no decorrer do tempo. Em qualquer caso, itens armazenados em cache diminuem o tempo de retardo necessário para se acessar a base do cliente. Num outro contexto, a difusão pode ser usada para propagar atualizações de cache em qualquer sistema cliente-servidor independentemente da abordagem adotada (*pull-based*, *push-based* ou híbrida).

3.3.2.1 Broadcast dinâmico e consistência

Em muitas aplicações, a difusão deve acomodar mudanças. Pelo menos três tipos diferentes de mudanças são possíveis [AFZ95]. Primeiro, o conteúdo da difusão pode mudar em termos de inclusão de novos itens e remoção de outros já existentes. Isto é típico dos sistemas híbridos já descritos anteriormente. Segundo, a organização dos dados da difusão pode ser modificada, por exemplo, para fornecer um diferente esquema de índices ou no caso de mudanças das freqüências dos dados a serem difundidos. E por fim, a atualização dos dados que são transmitidos.

Permitir a atualização dos dados cria a necessidade de protocolos de controle de consistência. Tais protocolos variam em torno de vários parâmetros. Primeiro, o protocolo depende das hipóteses levadas em conta sobre a entrega de dados, por exemplo, da existência de um *backchannel* para dados encomendados sob demanda, bem como se os itens de dados são armazenados no cache dos clientes, e se os clientes podem realizar atualizações. Protocolos de controle de consistência dependem do modelo de consistência em uso.

Nos SGBDs tradicionais, a consistência é baseada na serialização, que assegura que as operações serão executadas no contexto de atomicidade, consistência, isolamento e transações duráveis (ACID). Nos sistemas baseados em disseminação, entretanto, outros modelos de consistência de dados também são razoáveis [AFZ96]. Quando os clientes não podem realizar nenhuma atualização, um modelo de consistência apropriado é o *quase-caching*. Neste modelo, embora o valor do cache de dados possa ser o não mais recente, este valor é garantido estar dentro de uma variação permitida de acordo com as condições de coerência do cliente. *Quase-caching* [ABGM90] é uma escolha razoável nos casos de longas desconexões e/ou baixa conectividade.

3.3.2.2 Invalidação de cache por broadcast

Numa arquitetura cliente/servidor, o servidor pode utilizar a difusão para informar os clientes das atualizações dos itens em seu cache. Ele pode fazer isto ou sincronamente ou assincronamente [BI94].

No método assíncrono, o servidor difunde um relatório de invalidação em relação a um determinado item tão logo este item modifique o seu valor.

No método síncrono, o servidor difunde periodicamente um relatório de invalidação. O cliente deve "escutar" o relatório antes de decidir se o seu cache é ou não válido. Assim, o cliente terá confiança de seu cache quando do último relatório de invalidação. Isto adiciona alguma latência ao processamento de consultas, porque, ao responder uma consulta, o cliente deve esperar pelo próximo relatório de invalidação. Este tempo extra na latência da consulta pode ser evitado se for utilizado um modelo de consistência menos rígido como o *quase-caching*[ABGM90].

Protocolos de invalidação de cache possuem diferentes hipóteses: se o servidor mantém ou não informação sobre os clientes que estão atualmente em suas células; qual o conteúdo de seu cache; ou quando foi a última vez que seu cache foi validado.

Os servidores que armazenam tais informações são denominados *stateful*, enquanto que os servidores que não o fazem são denominados *stateless* [BI94].

Os relatórios de invalidação variam no tipo de informação que enviam aos clientes. Por exemplo, os relatórios de informação podem conter os valores que foram atualizados, ou apenas as identidades e os *timestamps* de suas ultimas atualizações.

Incluir nos relatórios os valores atualizados pode ser um desperdício do canal de transmissão, especialmente quando os itens serão atualizados apenas em uns poucos clientes. Por outro lado, se os valores não forem incluídos, o cliente deve então descartar o item de seu cache ou comunicar com o servidor para receber o valor atualizado. Os relatórios podem fornecer informações sobre itens individuais ou agregar informações cobre um conjunto de itens. Os relatórios de invalidação devem ser tal que se um cliente conclui que seu cache é válido, este é de fato o caso. Entretanto, alarmes falsos, onde o cliente considera erradamente que seu cache é inválido, pode ser tolerado.

Três estratégias de sincronismo para servidores *stateless* são propostas em [BI94].

Na *timestamps strategy* (TS), o relatório de invalidação contém os *timestamps* das últimas mudanças para os itens que sofreram atualizações nos últimos w segundos.

Na estratégia *amnestic terminals* (AT), o servidor envia apenas os identificadores dos itens que sofreram modificações desde o último relatório de invalidação.

A última estratégia é a de assinaturas (*signatures*). Uma assinatura é um valor computado sobre um número de itens, aplicando-se técnicas de compressão de dados similares àquelas utilizadas na comparação de arquivos.

Cada uma destas estratégias é adequada para cada tipo de cliente, dependendo principalmente da conectividade do cliente. Se os clientes estão freqüentemente conectados, são denominados *workaholics*, enquanto que aqueles clientes que estão freqüentemente desconectados são denominados de *sleepers*.

Assinaturas são mais adequadas para aqueles cujo período de desconexão é longo e difícil de predizer e a taxa de atualizações é maior do que de consultas.

O método AT é melhor para os *workaholics*.

O TS é vantajoso no caso dos *sleepers* quando a taxa de consultas é maior do que a taxa de atualizações.

Jing et al em [JBEA95] apresentam um método assíncrono baseado em seqüência de bits. Neste método, o relatório de invalidação é organizado como uma seqüência de bits com um conjunto de *timestamps* associados. Cada bit dentro da seqüência representa um item de dados dentro do banco de dados. Um bit “1” indica que o item correspondente foi atualizado desde o tempo especificado pelo *timestamp* associado, enquanto que um “0” indica que o item não foi modificado.

O conjunto de bits é organizado dentro de uma estrutura hierárquica. Isto permite com que o algoritmo trabalhe consistentemente mesmo sob condições de taxas de atualização variável e desconexões dos clientes. Uma versão escalável do algoritmo também é apresentada. Nesta versão, ao invés de aumentar o número de bits do relatório para grandes SDBDs, a granularidade de cada bit é aumentada. Assim, ele representará grupos de itens que raramente modificam-se, ao invés de cada um deles.

Deve-se ter cuidado especial com os clientes desconectados. Um cliente pode perder o relatório de invalidação porque estava desconectado.

Os métodos síncronos são mais eficientes que os assíncronos, já que os clientes necessitam apenas sintonizar-se periodicamente para ler o relatório de invalidação. No método assíncrono, esta sintonia deve ser continua.

Entretanto, se o cliente permanecer inativo por um período maior que o broadcast, o cache inteiro deve ser descartado, a menos que alguma checagem especial seja empregada.

Na checagem simples, o cliente envia o seu identificador, o cliente envia o identificador de todos os objetos armazenados no cache, com seus *timestamps* para o servidor para fins de validação. Isto requer grande faixa de banda no *uplink*.

Alternativamente, o cliente pode enviar grupos de identificadores e *timestamps* e realizar a validação daquele grupo. Isto é similar àchecagem de volume do *Coda File*

System. Checagem ao nível de grupo reduz as especificações de taxas de transmissão do canal de *uplink*. Por outro lado, uma simples atualização de um objeto do grupo invalida o grupo inteiro. Como resultado, a quantidade de itens contidos no cache pode ser significativamente reduzida, descartando os itens válidos do grupo. Para consertar esta situação, o servidor identificaria um conjunto de itens com maior probabilidade de atualização e os excluiria do grupo, quando da checagem da validade do grupo.

O recurso de *broadcast* pode ser explorado em vários algoritmos para controle de concorrência como para invalidação de transações de clientes como sugerido em [Bar97].

No controle de concorrência otimista, o agendador (*scheduler*) de transações do servidor checa no momento do *commit* se a execução que inclui a transação do cliente é serializável ou não. Se ela é serializável, ele aceita a transação, caso contrário ela a aborta.

O servidor periodicamente difunde para os seus clientes um relatório de certificação (*certification report* - CR), que incluem o *readset* e o *writeset* das transações ativas que declararam a sua intenção de confirmar no servidor no intervalo anterior (entre a última difusao e o instante atual) a foram satisfatoriamente certificadas.

Os clientes móveis utilizam esta informação para abortar de seu conjunto de transações aquelas transações cujos *readset* e *writesets* tem interseção com o CR atual. Assim, parte da verificação é realizada no cliente móvel. Apenas quando o cliente móvel não pode detectar nenhum conflito, o servidor completa a verificação. Se a transação pode realizar o *commit*, o servidor atualizará os valores no banco de dados e notificará os clientes via *broadcast*.

3.4 Esquemas hierárquicos de gerenciamento de localização

3.4.1 Introdução

A localização de usuários é basicamente um problema de diretório. Existem duas operações primitivas: consulta da atual localização de um usuário ou atualização quando o usuário desloca-se para uma nova posição. O problema está presente nas diversas camadas do sistema. Não existe nenhum padrão para as aplicações no uso e aquisição de informações de localização. Outras abordagens referentes ao gerenciamento de localização são aplicáveis, quando, ao invés de busca de localizações, o objetivo é recuperar eficientemente outras informações referentes a usuários móveis, tais como perfis ou parâmetros referentes a QoS.

Apesar de não haver nenhum padrão para este tipo de problema, o modelo abordado nesta dissertação utiliza a arquitetura de referência consistente de três níveis [Wang93]: o acesso, o fixo e a rede inteligente. O fixo consiste da rede fixa. A rede de acesso é a interface entre o usuário móvel e a rede fixa. E a rede inteligente é a rede de conexão de qualquer registro de localização, isto é, registros utilizados para armazenar informações sobre a localização de usuários. Ela é usada para conduzir tráfego referente ao rastreamento de usuários móveis.

A área de registro de um usuário é a região na qual pode ser localizado e pode ser definido como a célula na qual o usuário reside. Uma vez que a área de registro é localizada, o usuário pode ser monitorado dentro dela através de uma forma de *paging* ou *broadcast*.

A informação de localização é mantida apenas para aqueles terminais que estão ativos. Nenhuma informação de localização está disponível sobre terminais que estejam desligados. Para reduzir a sobrecarga de comunicação utilizam-se áreas de serviços (*services areas*) que são as regiões onde o usuário pode ser alcançado. Quando um cliente móvel esta fora de sua área de serviço, nenhuma informação de localização deste usuário está disponível.

Duas arquiteturas são consideradas base para a solução deste problema: uma é a arquitetura *two-tier* baseada no par de registros *home*/*visitor*; a outra é baseada nos registros de localização hierárquica, conectados através de uma rede inteligente.

Os critérios de selecionar um esquema de localização incluem o custo das atualizações da localização, a capacidade máxima de serviço de cada banco de dados de localização e o tipo e freqüências de chamadas em relação às operações de movimentação. Em particular, um parâmetro que afeta a performance da maioria dos esquemas de gerenciamento de localização é o comportamento das chamadas e movimentos de um usuário, que é traduzido pela relação de chamadas em razão da mobilidade (CMR). Seja $C_{i,j}$ o número de chamadas esperadas feitas da zona j para um usuário P_i, em um período de tempo T e U_i o número de movimentos de Pi no tempo T. A relação local chamada-mobilidade LCMR $_{i,j}$ é definida como LCMR $_{i,j}$ = C $_{i,j}$/U_i.

No *two-tier scheme*, um banco de dados *home*, denominado *Home Location Register* (HLR), é associado a cada usuário. O HLR de um usuário x mantém a localização atual de x e também de seu perfil. Os procedimentos de busca e atualização são simples. Para localizar um usuário x, o HLR de x é identificado e consultado. Quando um usuário x movimenta-se para uma nova célula, o HLR de x é atualizado.

O aperfeiçoamento deste sistema é a introdução do *Visitor Location Registers* (VLR) nestas mesmas células. O VLR de uma zona armazena as cópias dos perfis dos usuários que não estão em "*home*" e atualmente localizados nesta zona. Quando uma chamada é feita dentro de uma célula i para o usuário x, o VLR da célula i é consultado primeiramente, e apenas se o usuário não é encontrado no VLR, então o HLR de i é contatado. Quando um usuário x movimenta-se de uma célula i para j, além de atualizar o HLR de x, o registro de x é removido do VLR da célula i, e uma nova entrada de x é adicionada no VLR da célula j. Este esquema é denominado *básico*. A maior parte das companhias telefônicas utiliza este esquema.

Para reduzir o custo de consulta, a localização de um usuário pode ser mantida em sites adicionais. Em contrapartida, há um aumento no custo de movimentação dos clientes, em virtude de que estes sites adicionais também devem ser atualizados. A localização de um usuário pode ser armazenada no site do chamador ou replicada nos

sites de onde a maioria das chamadas é originada. Para diminuir o custo de atualização, o uso de *forwarding pointers* é proposto [JL95].

Os esquemas de localização hierárquicos são implementados através do uso de uma hierarquia da base de dados de localização. Nesta hierarquia, estruturada em forma de árvore, os banco de dados de localização no nível mais alto contém informações de localização dos usuários que estão localizados nos níveis inferiores ao dele. Neste caso, os banco de dados de localização na extremidade da árvore servem, cada um, a uma única célula e contém entradas para todos os usuários registrados nesta célula. Um banco de dados de um nível intermediário mantém as informações dos usuários registrados dentro do conjunto de células de sua sub-árvore. Para cada cliente móvel, esta informação é um ponteiro para uma entrada de um banco de dados de nível inferior ou para a localização atual do cliente.

LCA(i,j) denota o ancestral mais próximo entre os nós i e j (Least Common Ancestor). Nos esquemas de localização hierárquica, a razão comunicação e mobilidade ($LCMR_{i,j}$) para um nó j é interpretada como $LCMR_{i,j} = \Sigma\ LCMR_{i,k}$, onde k é um filho de j. Assim, a razão comunicação/mobilidade para um usuário P_i, e um nó interno j é a razão do número de chamadas para P_i originadas de qualquer célula na sub-árvore de j e o número de células visitadas por P_i.

O tipo de informação mantida dentro dos bancos de dados de localização afeta o custo de atualizações e consultas, bem como a carga de distribuição entre os *links* e nós da hierarquia. Considere-se o caso de manter *forwarding pointers* em todos os bancos de dados locais. Por exemplo, considere na Figura 3.3, um usuário x que está na célula 18. Existe uma entrada dentro do banco de dados no nó 0 apontando para a entrada de x no banco de dados do nó 2. A entrada de x no banco de dados no nó 2 aponta para a entrada de x no banco de dados no nó 18.

Quando x se movimenta da célula x para a célula j, as entradas de x dentro dos bancos de dados existentes no caminho entre j a LCA(i,j) e de x a LCA(j,i) são atualizadas.

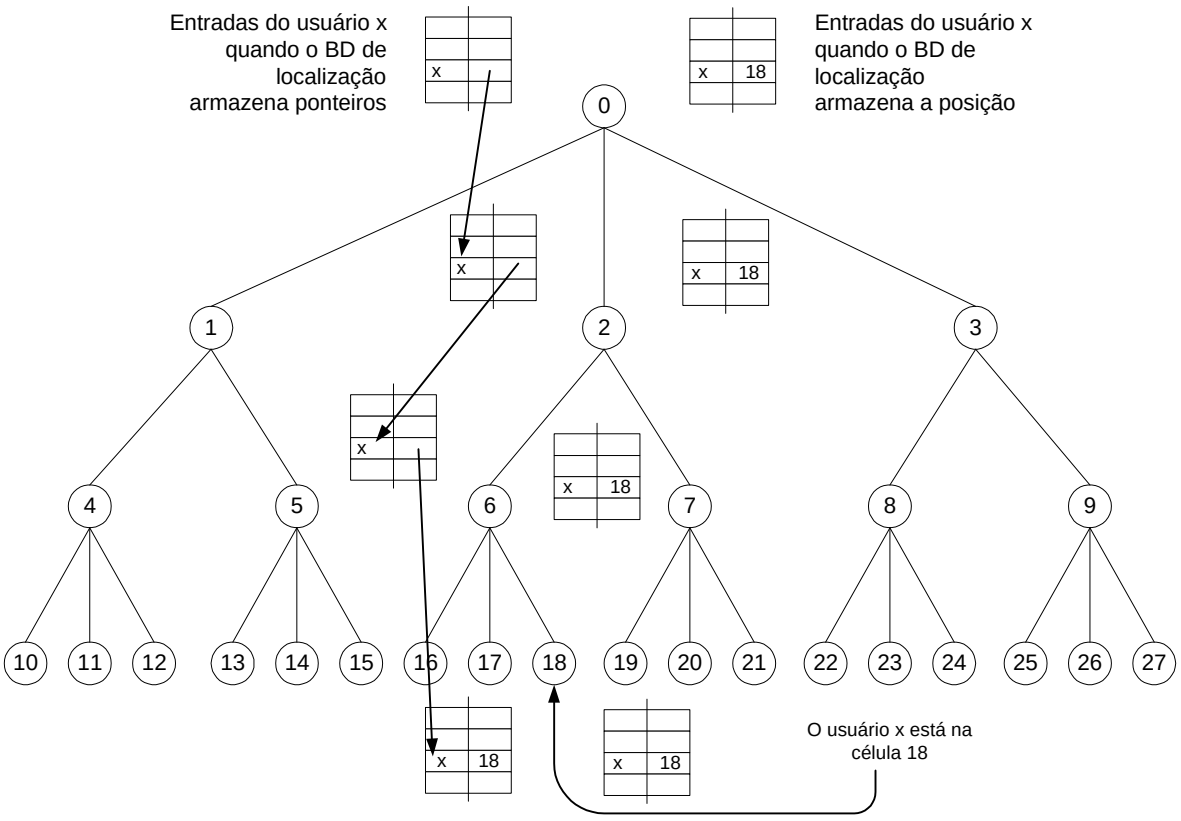

Figura 3.3 - Esquema de localização hieráquico

Quando um usuário x movimenta-se de 18 para 20, as entradas nos nós 20, 7, 2, 6 e 18 são atualizadas. Mais especificamente, as entradas de x são removidas dos bancos de dados dos nós 6 e 18 e as entradas de x são adicionadas aos bancos de dados dos nós 2, 7 e 20.

Quando um chamador localizado na célula i realiza uma chamada para um usuário y localizado na célula j, o procedimento de busca consulta os bancos de dados, desde o nó i, depois subindo a árvore até encontrar a primeira entrada de x. Isto acontece no nó LCA(i,j). Depois disso, o procedimento prossegue descendo a árvore seguindo os ponteiros para o nó j.

Por exemplo, uma chamada da célula 21 para o usuário x, consulta os bancos de dados nos nós 21, 7 e encontra a primeira entrada de x no nó 2. Então ele segue os ponteiros para os nós 6 e 18.

Considere o caso das entradas dos bancos de dados que mantém a localização atual de cada usuário. Para um usuário x registrado em 18, existem entradas nos bancos de dados nos nós 0, 2, 6 e 18, cada qual contendo um ponteiro para a posição 18. Neste caso, um movimento da célula i para a célula j ocasiona a atualização de todas as entradas no caminho que vai de j para a raiz da árvore e da raiz para o nó i.

Uma realocação do usuário x do nó 18 para o nó 20 resulta na atualização nas entradas de x em 20, 7, 0, 2, 6 e 18. Depois da atualização, as entradas de x existem nos bancos de dados localizados nos nós 0, 2, 7 e 20, cada qual contendo um ponteiro para 20, enquanto as entradas de x nos bancos de dados nos nós 6 e 18 são removidas. Em contrapartida, o custo de uma chamada de i para j é reduzida, desde que uma vez que LCA(i,j) é alcançado, não existe nenhuma necessidade de consultar os bancos de dados no caminho de descida para j. Por exemplo, uma chamada do nó 21 para o usuário x, consulta os bancos de dados dos nós 21, 7, 2 e então 18 diretamente, sem consultar o banco de dados no nó 6.

Quando os bancos de dados de localização hierárquica são usados, não existe nenhuma necessidade de existir um *Home Location Register* (HLR). O usuário pode ser localizado através de consulta ao banco de dados dentro da hierarquia. No pior caso, uma entrada para o usuário será encontrada no banco de dados da raiz. O esquema hierárquico leva a redução do custo de comunicação quando muitas chamadas e movimentos são localizados geograficamente. Nestes casos, ao invés de contatar o HLR do usuário, que pode estar distante da localização atual do usuário, um pequeno número de bancos de dados de localização na vizinhança do usuário é acessado. Entretanto, o número de bancos de dados atualizados e consultados é maior do que ocorre no esquema *two-tier*. Para reduzir o número de consultas e atualizações, ao invés de alocar bancos de dados em todos os nós dentro da hierarquia, os bancos de dados podem ser seletivamente alocados em alguns nós dentro da hierarquia. Ainda assim, ao invés de manter a informação exata de localização, poderia se manter informações aproximadas mantidas nos nós internos. Quando a raiz não contém o banco de dados de localização, ou quando mantém apenas informações aproximadas, existe a necessidade de alguma forma de restringir a difusão.

3.4.2 Otimização da localização dos bancos de dados

Manter a informação de localização em todos os nós dentro da hierarquia, embora resulte numa consulta de menor custo, aumenta o número de bancos de dados que devem ser atualizados durante uma operação de movimento. Para reduzir o custo de atualização, as entradas dos bancos de dados podem ser seletivamente escolhidas entre nós específicos dentro da estrutura hierárquica. Neste caso, os procedimentos de busca e atualização não mudam, exceto pelo fato de que apenas os nós que contém os bancos de dados são consultados ou localizados. Os outros são ignorados. Por exemplo, quando uma chamada é realizada de j para i, o procedimento de busca percorre a árvore do nó j até o ancestral mais novo de LCA(i,j) que contém o banco de dados de localização.

Uma possível solução é manter as entradas de localização apenas nos nós folhas. Neste caso, quando não existe nenhum HLR associado à um cliente móvel, as estratégias de localização devem usar alguma forma de broadcast. Neste cenário, as estratégias de localização incluem buscas *flat*, *expandida* e *híbrida* [BIV92].

Seja *home* a célula na qual o usuário se registra inicialmente.

O procedimento de busca *flat* inicia da raiz e então inicia consultas paralelamente em todos os nós do próximo nível da árvore, repetindo este ciclo até que o nível mais baixo seja alcançado.

O procedimento de busca *expandida* inicia a busca consultando o *home* do visitante i, então consulta o pai de *home*, que então consulta todos os seus filhos e assim por diante. Este tipo de busca favorece movimentos em localidades próximas.

O procedimento *híbrido* de busca inicia seu procedimento como no *expandido*, mas se a localização não é encontrada nos filhos do pai do *home* do visitante, uma busca flat é iniciada. O esquema *híbrido* pode localizar rapidamente aqueles usuários que, quando não estão em *home*, podem ser encontrados longe dali.

Se as entradas dos bancos de dados são mantidas apenas em nós selecionados, a implementação de um HLR pode mostrar-se útil. Neste caso, uma chamada originada da célula i inicia a busca dos visitantes desta célula, prosseguindo depois o caminho de i para o LCA de i e o HLR do visitante e então move nos sentindo descendente do HLR do visitante, a menos que uma entrada para o visitante seja encontrada dentro de um banco de dados deste caminho. Esta entrada é, então, acompanhada. Este é o menor caminho para o chamador [Wang93].

A alocação dos bancos de dados de localização pode ser visto como um problema de otimização. Deve-se levar em conta:

(a) O número de atualizações e acesso

(b) O custo das comunicações

(c) A soma do tráfego entre os *links*

Algumas restrições devem também ser obedecidas

(a) Um limite superior da taxa na qual cada banco de dados pode ser atualizado ou acessado

(b) A capacidade dos *links*

(c) A capacidade de armazenamento disponível

Este tipo de estudo está descrito em [AHMK94]. O objetivo é minimizar o número de atualizações e acessos por unidade de tempo, dada a capacidade máxima de serviço do banco de dados e estimativa da razão chamada/mobilidade. Nesta abordagem, as comunicações não são consideradas, e, assim, se a capacidade de serviço é suficientemente grande, um computador central na raiz é a solução ótima do problema. A questão é formulado como um problema de otimização combinatória e é resolvido usando um algoritmo de programação dinâmica.

3.4.3 Partições

Para evitar a manutenção das entradas de localização em todos os níveis da hierarquia, e ao mesmo tempo a redução do tempo de busca, é sugerido o uso de partições. As partições para cada usuário são obtidas pelo agrupamento de células entre as quais o usuário movimenta-se freqüentemente e separando as células onde eles não são encontrados com freqüência. Assim, as partições exploram a regionalidade dos movimentos do usuário (destino) ao passo que a replicação explora a localidade das chamadas (origem).

Partições são utilizadas para direcionar a árvore de busca. Para cada partição, a informação se o usuário está atualmente dentro da partição é mantida no LCA de todos os nós dentro da partição, denominado o representante da partição. Esta informação é usada durante a busca flat (isto é, a busca *top-down*, começando pela raiz) para decidir qual sub-árvore dentro da hierarquia buscar.

Assim, as partições reduzem o custo de busca total quando comparado à busca flat. Entretanto, ao que se refere o custo de atualização, este é aumentado, pois quando um usuário cruza uma partição, os representantes das suas partições anteriores e atuais devem ser informados.

Um uso ligeiramente diferente de partições denominado *redirection trees* é proposto em [CM95]. Uma partição simples, denominada região local, é definida pela inclusão de todos os nós entre os quais o usuário freqüentemente se movimenta. O representante desta região local, denominado agente de redireção (*redirection agent*) mantém a localização de todos os usuários que o elegeram como o seu agente. Os agentes redirecionam todos os pacotes que passam por ele durante qualquer tipo de busca (por exemplo, *flat* ou usando HLR) da atual localização do usuário, quando este é localizado em sua região local. Os movimentos dentro de uma região local são registrados dentro do agente de redireção e não necessariamente nos servidores de localização fora da região.

3.4.4 Caching

Como nos esquemas *two-tier*, as técnicas de *caching* podem ser implementadas dentro das arquiteturas hierárquicas para explorar a localidade das chamadas. Quando uma chamada é realizada da célula i para o usuário x localizado na célula j, o procedimento *search* atravessa a árvore no sentido ascendente de i para LCA(i,j) e então desce para j. Também deve-se considerar as mensagens de *acknowledgment* que retornam de j para i. Para suportar *caching*, durante o caminho de volta, um par de *bypass pointers*, denominado *forward* e *reverse*, é criado. Um *forward bypass pointer* é uma entrada para um ancestral de i, denominado s, que aponta para um ancestral j, dito t. O *reverse bypass pointer* é de t para s. Durante a próxima chamada da célula i para o usuário x, as mensagens de busca atravessam a árvore no sentido ascendente até que s seja alcançado. Então a mensagem vai até o banco de dados t via LCA(i,j), ou via uma rota mais curta, se esta rota está disponível na rede. Similarmente, a mensagem de *acknowledgment* pode saltar todos os ponteiros intermediários no caminho entre t e s.

Por exemplo, seja uma chamada da célula 13 para o usuário x na célula 16 (Figura 3.4). Um *forward bypass pointer* é setado no nó 1 apontando para o nó 6; o *reverse bypass pointer* é de 6 para 1. Durante a próxima chamada da célula 13 para o usuário x, a mensagem de busca atravessa a árvore do nó 13 até o nó 1 e então para o nó 6, ou através de LCA(1,6), que é o nó 0, ou via um caminho mais curto. Em qualquer caso, nenhuma consulta é realizada nos bancos de dados dos nós 0 e 2.

O nível de nós s e t onde o *bypass pointers* são de iguais ou diferentes níveis. No *simple caching*, s e t são ambos nós folhas, enquanto no *level caching*, s e t são nós pertencentes a qualquer nível e possibilitam o acesso independente de nível diferente (como no exemplo anterior). Colocar um *bypass pointer* no nó de alto-nivel, faz esta entrada disponível para todas as chamadas originadas das células na sub-árvore s. Entretanto, as chamadas devem percorrer um caminho mais longo para alcançar s. Utilizar o ponteiro para apontar um nível nó mais alto t, aumenta o custo de busca, porque para localizar um usuário, um longo caminho de t para o nó folha deve ser percorrido. Por outro lado, a entrada do cache permanece válida por todo o tempo em

que o usuário movimenta-se dentro da sub-árvore de t. Um esquema adaptativo pode ser considerado para setar os níveis de s e t dinamicamente.

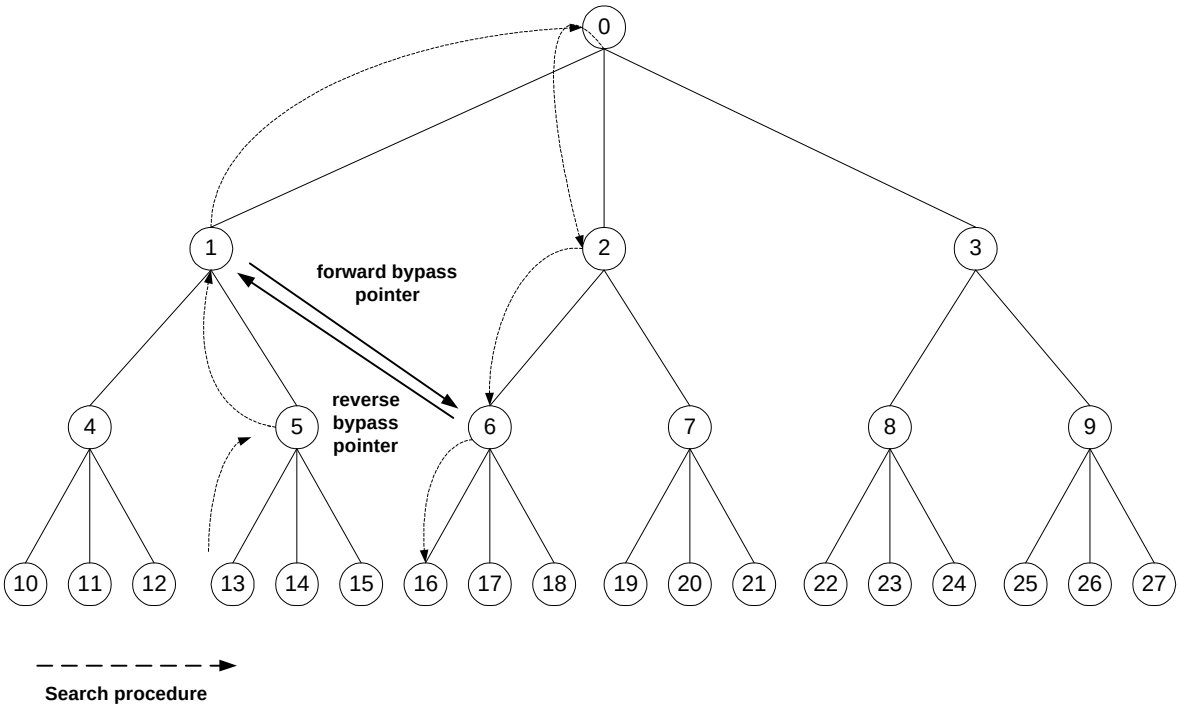

Figura 3.4 - Caching no esquema de localização hierárquica

Como no esquema de localização *two-tier*, existem muitas possibilidades de variações para realizar a invalidação de cache [Jain96]. No *lazy caching*, a operação de movimentação permanece inalterada, pois as entradas de cache são atualizadas apenas quando uma falha de cache é sinalizada. No *eager caching*, as entradas de cache são atualizadas para cada operação de movimentação. Especificamente, considere uma operação de movimentação da célula i para a célula j, onde uma mensagem de registro/desregistro propaga de j via LCA(j,i) para i. Durante este procedimento, os *bypass pointers* que não são mais válidos são deletados. Estes ponteiros incluem quaisquer *forward bypass pointers* encontrados durante o caminho ascendente da mensagem de registro, e qualquer *reverse* ou *bypass pointers* encontrados durante o caminho descendente das mensagens de cancelamento de registro [Jain96].

Os resultados preliminares de performance são reportados em [Jain96]. A analise é baseada em *Regional Call-to-Mobility Ratio* (RCMR) definido por um usuário x com respeito aos nós da árvore s e t. É definido como um número médio de chamadas da sub-árvore com raiz em s para o usuário x, enquanto o usuário x está dentro da sub-árvore com raiz em t. Isto é demonstrado, sob determinadas hipóteses, para usuário com RCMR > 5, com o *caching* podendo resultar até numa redução de 30% dentro do custo tanto de chamadas e movimentos, quando considera-se apenas o número de operação de bancos de dados.

O *caching* dentro do caso de armazenamento nos nós internos da exata localização, em oposição aos ponteiros nos bancos de dados de nível mais baixo, pode ser desenvolvido de diversas formas. Abordagens abrangem desde *caching* simples, onde a localização é armazenada apenas nos nós folhas, para o nível de *caching*, onde a localização é alocada em todos os nós em um dado nível.

3.4.5 Replicação

Para reduzir o custo de consultas, a localização de um usuário é seletivamente replicada em nós adicionais dentro da hierarquia. Os sites de replicação não são necessariamente nós folha. Especificamente, como no esquema *two-tier scheme*, a localização de um usuário é replicada em um nó somente se o custo de replicação não exceder o custo da não-replicação. No esquema de bancos de dados de localização hierárquica, existe uma consideração adicional. Se um valor alto de LCMR é o critério básico de seleção de sites de replicação, os bancos de dados nos níveis mais altos tendem a ser selecionados como sites de replicação sobre os bancos de dados nos níveis mais baixos, desde que eles processem valores de LCMR mais altos. O LCMR para um nó interno é a soma dos LCMRs de seus filhos. Isto resulta em atividades excessivas de atualização nos bancos de dados de mais alto nível.

O objetivo do algoritmo de replicação *HiPer* [JLSWC96] é minimizar o custo de comunicação. O HiPer replica a localização do usuário i no banco de dados j se isto for necessário, isto é, se os benefícios da replicação excede o seu custo.

Não é judicioso replicar, se o $LCMR_{i,j} < R_{min}$.

Caso $LCMR_{i,j} > R_{max}$, então a replicação é bem vinda.

Se $R_{min} < LCMR_{i,j} < R_{max}$, temos o caso em que a resposta não é direta, e a replicação é condicionada à topologia do banco de dados. As restrições que devam ser levadas em consideração são o número de máximos de réplicas, N_{max}, por usuário e um nível máximo L, na qual as localizações deveriam ser replicadas. Um algoritmo que roda off-line calcula os sites de replicação para cada usuário em duas fases. Na primeira fase, em um caminho *bottom-up*, ele aloca réplicas de i em todos os bancos de dados com $LCMR_{i,j} >= R_{max}$ até que o número de alocações de réplicas n não exceda Nmax. Na segunda fase, se $n = N_{max}$, o algoritmo aloca as réplicas restantes para os bancos de dados abaixo do nível L com o maior valor não negativo $LCMR_{i,j} - R_{max}$ dentro de uma abordagem *top-down*.

3.4.6 Forwarding pointers

Dentro de um esquema de localização hierárquica, quando um usuário x movimenta-se de uma célula i para uma célula j, as entradas de x são criadas em todos os bancos de dados no caminho de j para LCA(j,i), enquanto as entradas de x no caminho de LCA(j,i) são removidas. Para reduzir o custo de atualização, apenas os bancos de dados até o nível m são atualizados. Ainda assim, um ponteiro é setado do nó s para o nó t, onde s é o ancestral de i no nível m, e t é o ancestral de j no nível m. Como no esquema de *caching*, os níveis de s e t variam. No *simple forwarding*, s e t são nós folhas, enquanto no *level forwarding*, s e t podem ser nós em qualquer nível.

Por exemplo, um usuário x localizado no nó 14 que se movimenta para o nó 17 (Figura 3.5).

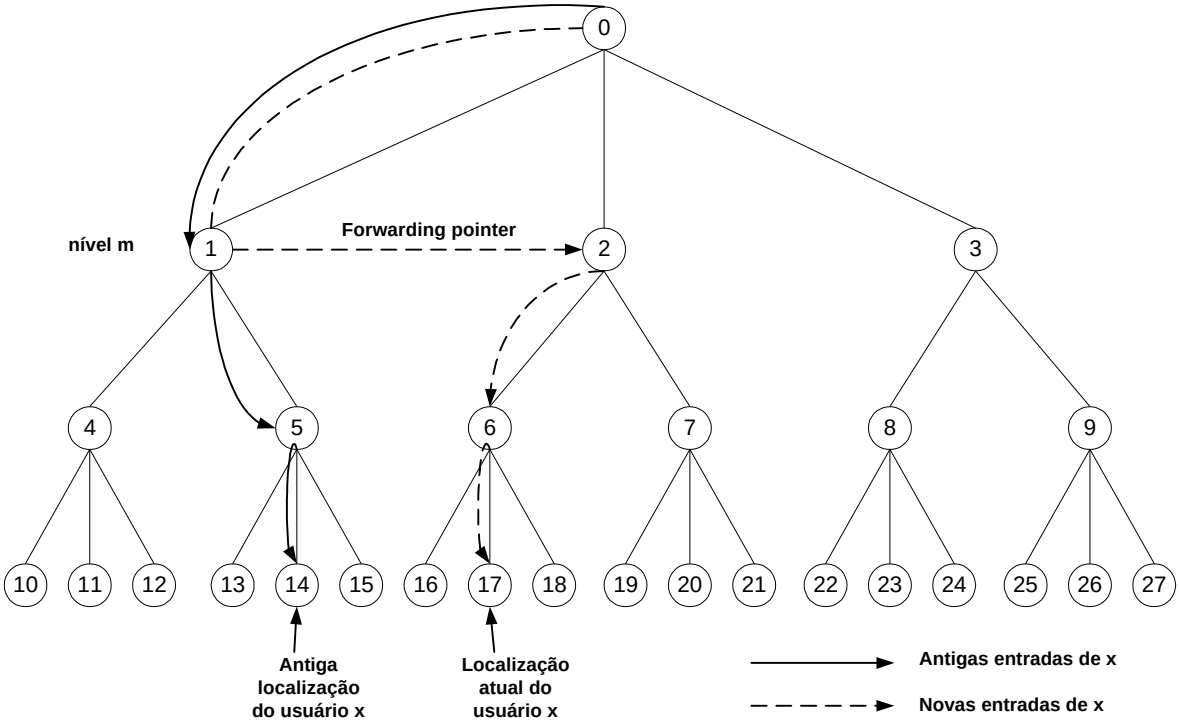

Figura 3.5 - Exemplo de forwarding pointer

Seja o nível m=2. Uma nova entrada para x é criada nos bancos de dados dos nós 17, 6 e 2, as entradas para x dentro dos bancos de dados dos nós 14 e 5 são removidos, e um ponteiro é setado na entrada de x no banco de dados do nó 1 apontando para a entrada de x no banco de dados no nó 2. A entrada de x no nó 0 não é atualizada. Quando um usuário, por exemplo, na célula 12, chama x, a mensagem de busca atravessa a árvore do nó 12 até o nó 0 na raiz, onde a primeira entrada de x é encontrada, então desce até o nó 1, seguindo o *forwarding pointer* para 2, descendo depois de 2 para 17. Por outro lado, uma chamada de um usuário na célula 18, resulta em uma rota mais curta: ela sobe até 6 e dali até 17.

As entradas obsoletas nos bancos de dados nos níveis mais altos que m (por exemplo, a entrada no nível 0) podem ser atualizadas depois de uma consulta bem sucedida. As atualizações ocorrem de várias formas. No *jump updates*, uma atualização de

localização acontece apenas nos bancos de dados da célula do chamador, enquanto que no *path compression*, todos os nós no caminho são atualizados. Outra possibilidade de atualização é que cada nó envie uma mensagem de atualização de localização para os servidores de localização no seu caminho para a raiz nas horas de fora de pico.

Uma análise do método *forwarding* dentro de um esquema de localização hierárquica na qual as entradas são os endereços ao invés de ponteiros para bancos de dados de mais baixo nível é apresentada em [KVP96]. Todas as combinações de *forwarding* (no *forwarding* (NF), *simple forwarding* (SF) e *level forwarding* (LF)) e de atualizações em chamadas (*jump updates* (JU), *path compression updates* (PC) e no *updates* (NU)) são consideradas. Resultados de simulações preliminares são apresentados para dois tipos de ambientes: (a) movimentos e chamadas arbitrárias e (b) movimentos curtos e chamadas localizadas (isto é, a maior parte das chamadas recebidas vem de um conjunto específico de usuários). O custo agregado de busca e atualização é considerado, e a medida de custo é o número de mensagens. A simulação [KVP96] mostra a combinação LF-PC a fim de realizar todas as combinações em ambos os ambientes, exceto no caso de alta comunicação e baixa mobilidade e de baixa mobilidade e alta comunicação. Nestes casos, dentro de um segundo ambiente, a combinação LF-JU tem melhor performance devido àlocalidade das chamadas. Um esquema adaptativo por usuário é sugerido para escolher entre as combinações LF-PC e LF-JU baseada nas características de chamada e mobilidade. Para determinar estas características, cada unidade móvel mantém uma seqüência de todos os movimentos feitos e chamadas realizadas. Esta seqüência determina o grau de mobilidade da estação (alto ou baixo) e se este possui um grande número de chamadores freqüentes.

Para evitar a criação de longas cadeias de *forwarding pointers*, algumas formas de redução de ponteiros são necessárias. Existem muitas variações, dependendo do tipo de redução e quando esta é iniciada [PF97]. No *simple purge*, um *direct pointer* é adicionado do primeiro nó da cadeia para a localização atual do usuário, enquanto que todos os *forwarding pointers* intermediários são removidos. Isto resulta em uma cadeia de comprimento 1.

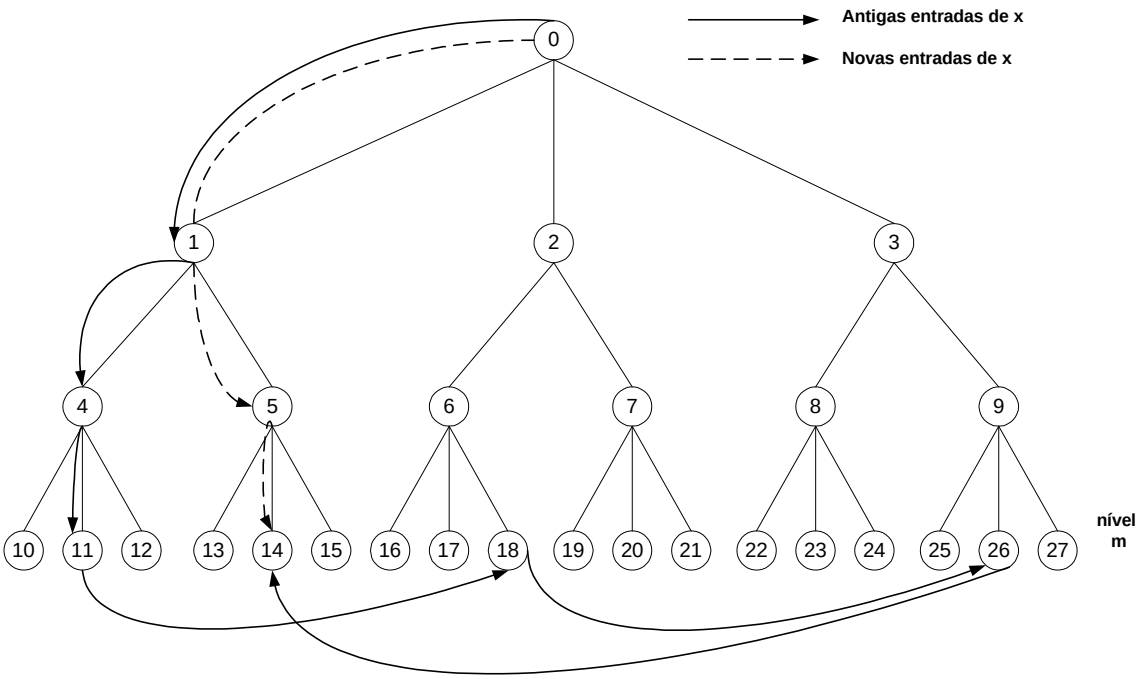

Figura 3.6 - Exemplo de pointer purging

Tome-se, por exemplo, a cadeia 11-18-26-14 que do movimento do usuário *x* do nó 11, para os nós 28, 26 e 14, nesta ordem. No *simple purging*, a entrada em 11 é apontada diretamente para 14 e as entradas nos nós 18 e 26 são apagadas. Assim a cadeia 11-14 é produzida. No *complete purging*, a entrada do primeiro nó da cadeia é também apagado, produzindo uma cadeia de comprimento 0. Isto envolve a deleção de todas as entradas de todos os bancos de dados internos no caminho originado do primeiro nó ao LCA do primeiro nó e a atual localização e a adição de entradas nos bancos de dados internos no caminho do LCA até a localização atual. Por exemplo, na cadeia 11-18-26-14, além das entradas de x em 18 e 26, a entradas de x em 11 também é apagada. Então, as entradas nos bancos de dados de mais alto nível que conduzem a 11 também são removidas. Em particular, as entradas de x em 4 são removidas e as entradas são setadas nos nós 1, 5 e 14 conduzem a 14, a nova localização (veja a Figura 3.6). *Purging* de *forwarding pointers*, simples ou completos, podem ocorrer nas chamadas e movimentos. *Purging* nas chamadas é iniciado quando, após uma consulta bem sucedida, o primeiro nó da cadeia é alcançado. *Purging* nas chamadas é iniciado quando um calor máximo definido pelo

sistema é alcançado. Alternativamente, os *forwarding pointers* podem ser removidos periodicamente.

A Tabela 3.1 resume o uso de *caching*, replicação e *forwarding pointers* nos esquemas de localização hierárquica.

Caching	Quando x na célula i é chamado por um usuário y na célula j, realiza o cache de um nó no caminho de j para LCA(i,j) um ponteiro para um nó no caminho de i para LCA(i,j) será utilizado nas chamadas subseqüentes para x originadas da célula j
Replicação	Replica seletivamente a localização de x nos nós internos e/ou bancos de dados das folhas
Forwarding Pointers	Quando x movimenta-se da célula i para a célula j, ao invés de atualizar todos os bancos de dados no caminho de i para LA(i,j) e de LCA(i,j) para j, atualizando todos os bancos de dados até o nível m e adicionar um *forwarding pointer* no nível m, ancestral de i aponta para o nível m ancestral de j

Tabela 3.1 - Uso de *caching*, replicação e *forwarding pointers* nos esquemas de localização hierárquica.

4. Um modelo de gerenciamento de dados em um ambiente de Comando e Controle

Viu-se que as operações militares são significativamente móveis, sujeitas a interferências do ambiente e do inimigo, tornando as suas comunicações pouco confiáveis, somando-se, também, ao fato de se basear em sistemas rádio. Possuem, ainda, uma característica de hierarquia em sua organização, bem como a alteração dinâmica de sua estrutura organizacional.

Um sistema de comando e controle (C2) deve permitir que atualizações realizadas em estações móveis reflitam no banco de dados do sistema, mesmo que tenham ocorrido muito antes da reconexão do sistema. Protocolos de controle de concorrência como pré-ordenação e bloqueio em duas fases não são adequados para sistemas de C2, pois o primeiro provavelmente rejeitará as transações que cheguem muito atrasadas e o segundo, em virtude do ambiente de fraca conectividade, poderá ocasionar bloqueios de longa duração que, ao final, provavelmente não se completarão. Assim, o método de multiversionamento [PC99] parecer ser mais adequado para o modelo.

As transações nas estações móveis, portanto, devem processar as alterações efetuadas localmente para, posteriormente serem certificadas, ou seja, checadas se tornarão válidas ou não. Duas abordagens podem ser utilizadas neste caso. A primeira seria de transações aninhadas [Chr93], descrita na seção 3.2.2.1. A segunda seria através de uma distinção de tipos de transações, uma realizada nas estações móveis e outras nas estações fixas [GHOS96], [LS94], [LS95]. Ambas são válidas. Esta dissertação optou por utilizar o segundo modelo. O modelo de certificação adotado é o sugerido em [Bar97]. O modelo proposto utiliza o *Presumed Abort 2-Phase Commit*. Propõe, como sugerido na seção 3.2.2.2, que, em futuros trabalhos, o coordenador possa ser o elemento participante de maior grau hierárquico.

O modelo deve oferecer as funcionalidades que permitam a mudança dinâmica da hierarquia [PF97]. O modelo desta dissertação implementa as técnicas básicas descritas em 3.4, deixando as otimizações descritas naquela seção como sugestão de trabalhos futuros.

Deve-se também atentar quanto aos problemas de replicação de dados. Dois aspectos são fundamentais nesta escolha: o que deve ser replicado [KS92], [TLAC95], [Kue94], abordado na seção 3.2.1.1; e a forma como proceder esta difusão de dados em função do comportamento dos clientes [ABGM90], [BI94], [JBEA95], descrito na seção 3.3.2.2.

O modelo proposto nesta dissertação objetiva integrar as técnicas descritas anteriormente, de forma a atender aos requisitos de um sistema de C2.

Para tal, este capítulo apresenta, nesta ordem, o modelo abstrato de dados, os componentes do sistema de gerenciamento de dados em sistemas móveis, o esquema de replicação em duas camadas, a arquitetura e as funcionalidades dos gerenciadores de localização, de transações e replicação.

4.1 Modelo abstrato de dados

Esta seção descreve o modelo abstrato dos dados num ambiente de Comando e Controle, localização de cópias primárias e réplicas. Descreve também o modelo transacional adotado neste ambiente distribuído, definindo os tipos de versões de dados existentes nos nós da rede e os tipos de transações existentes no sistema, bem como condições de consistência de dados aplicados ao sistema.

4.1.1 Introdução

O modelo na qual se baseia o sistema, a princípio, não possui limitações de energia, nem de memória, itens comumente abordados na literatura de banco de dados móveis.

Porém, reflete a característica de baixa velocidade de transmissão e fraca conectividade.

As informações nos Sistemas de Comando e Controle são produzidas tanto em escalões superiores como nos escalões inferiores.

Os *links* de comunicações entre o Escalão Superior e o Escalão Inferior são denominados de *downlinks*, normalmente com característica de *broadcast* e velocidades de transmissão superior aos *uplinks*, que são as ligações ponto-a-ponto no sentido inverso, isto é, entre o Escalão Inferior e o Escalão Superior.

As mensagens produzidas nos escalões superiores genericamente são:

- Relatórios de informações de interesse geral ou particular das OM subordinadas (difundidas independentemente de demanda).
- Ordens a serem cumpridas pelos escalões inferiores;

Os relatórios de informações de interesse geral ou interesse particular das unidades subordinadas são difundidos para os escalões inferiores. Sua principal característica é não ser, normalmente, resultado de demanda da OM destinatária. Exemplos são: Ordem de Operações sendo difundida para todos as OM daquele escalão; um ataque inimigo em determinada localidade; informações de interesse para as OM (ou uma única OM) de um determinado setor do terreno. Estas informações são difundidas independentemente de solicitações das OM subordinadas. Estes dados são incorporados à base de dados local das OM, sendo, normalmente, acessíveis apenas para leitura. O sistema tenta atualizar o sistema através da propagação das atualizações dos dados da estação que os gera para as estações que possuem cópias destes dados, objetivando alcançar a consistência lógica do sistema. Assim, poder-se-ia dizer que as cópias primárias destes dados estão nas unidades superiores e as demais são réplicas dos dados.

Existem dados que não estão não estão em consistência lógica com a base de dados que o gerou em virtude de sua importância para o sistema. Estes dados não serão considerados no contexto desta dissertação.

Outro fluxo importante de dados diz respeito ao que é gerado pelos Escalões Inferiores.

O trânsito de mensagens do Escalão Inferior para o Escalão Superior é realizado através dos *uplinks*, que, em regra, são de velocidade inferior aos *downlinks* (do escalão superior para o escalão inferior).

As mensagens geradas pelo Escalão Inferior para o Escalão Superior poderiam ser sintetizadas como:

- Solicitação de consultas;
- Relatório de dados.

As OM subordinadas necessitam, em determinadas situações, de dados específicos de outras OM, sejam de OM pertencentes ao escalão a que estiver subordinado, sejam pertencentes a outro escalão. Assim, a OM faz a solicitação de consulta ao escalão imediatamente superior, que de acordo com a solicitação responderá conforme os dados que tem em seu SGDB, ou encaminhará a solicitação a quem de direito. Determinadas consultas que se tornam freqüentes podem ser enviadas periodicamente para os escalões inferiores, objetivando antecipar operações de leitura (*prefetching*) realizadas objetivando melhorar a performance do sistema.

A OM também é responsável por enviar relatórios de seus dados ao Escalão Superior. Determinados dados são enviados periodicamente. Outros dados são enviados independente de periodicidade, em virtude de seu conteúdo. Estes dados devem estar em consistência lógica com sua cópia primária.

Semelhante ao que ocorre com os dados gerados nos escalões superiores, as cópias primárias estariam nas unidades inferiores e os dados propagados para os outros escalões são consideradas réplicas.

4.1.2 Tipos de dados, propagação e transações do Modelo

Como sugerido em [GOSH96], elaborou-se o seguinte modelo abstrato de dados, tipificados da seguinte forma:

- *Cópia Primária*
- *Cópias Secundárias* (com consistência lógica)
- *Réplicas Não-Confiáveis* (sem consistência lógica)

Réplicas Não-Confiáveis são aquelas em que o sistema não tem um compromisso com a consistência lógica das mesmas. Neste caso, os dados estão armazenados no SGDB e poderão servir de resposta a consultas no caso de desconexão, ou ainda no caso em que tempo de resposta do SGBD, que possua os dados relativamente confiáveis, for demasiadamente longo. A maior parte dos relatórios de informação de interesse geral se enquadra neste tipo de dado. Este tipo de dado, apesar de fazer parte de um Sistema de Comando e Controle, não será mais abordado no escopo desta dissertação.

4.1.2.1 Cópias primárias

Cópias Primárias se referem aos dados, em regra, localizados no SGBD que os criaram. Possuem um único detentor e são, na maioria dos casos, atualizados somente por este SGDB.

Existem, entretanto, situações em que esta cópia pode ser modificada ou criada por outros SGBDs, obedecendo às restrições de acesso. Os relatórios de informação geral, os relatórios de dados das OM inferiores e as ordens a serem cumpridas armazenadas nas OM que as geraram são um exemplo deste tipo de dado.

As cópias primárias são atualizadas por transações, que se dividem, por sua vez, em transações de 1ª classe e transações de 2ª classe, cuja definição completa está na seção 4.5.

As cópias primárias possuem múltiplas versões geradas ao longo do tempo, refletindo os valores dos dados em determinado instante. Estas versões são descartadas, periodicamente, através de um coletor de lixo.

As cópias primárias são de dois tipos:

- *Cópias primárias sem prazo de validade;*
- *Cópias primárias com prazo de validade*

Cópias primárias sem prazo de validade

As versões das cópias primárias sem prazo de validade se dividem em:

- *Versões Mestres*
- *Versões Tentativas*

As *Versões Mestres* das cópias primárias sem prazo de validade se referem ao estado atual do dado, refletindo alterações e execuções de consultas que, efetivamente, tornaram-se duráveis. Normalmente, são resultados de transações de 1ª classe ou resultado de uma certificação positiva de uma transação de 2ª classe.

As *Versões Tentativas* são as atualizações que estão em processo de certificação, ou seja, aguardando uma autorização do sistema para tornarem-se *Versões Mestres*.

Cópias primárias com prazo de validade

As *cópias primárias com prazo de validade* não possuem versão tentativa visto que são atualizadas somente pelo detentor da cópia.

4.1.2.2 Cópias secundárias

Cópias Secundárias são geradas a partir da cópia primária e mantém um nível de consistência lógica com esta última. Salienta-se, novamente, que a consistência lógica é bastante prejudicada em função da fraca conectividade do sistema. Sistemas de *quasi-caching*, determinando prazos de expiração para dados, são uma solução para este tipo de problema. Um exemplo deste tipo de dado são as rotas de deslocamento de viaturas.

Assim, podemos dividir as cópias secundárias em:

- *Cópias secundárias sem prazo de validade*
- *Cópias secundárias com prazo de validade*

Cópias secundárias sem prazo de validade

As *cópias secundárias sem prazo de validade* podem não estar em consistência lógica com a sua cópia primária em virtude da fraca conectividade do sistema. Elas objetivam refletir os valores contidos nas cópias primárias e podem ser operadas quando da desconexão ou sobrecarga do sistema.

Semelhante às cópias primárias, as cópias secundárias sem prazo de validade, possuem múltiplas versões, às quais podem ser operadas, obedecidas às restrições de acesso. Dividem-se também em:

- *Versões Mestres*
- *Versões Tentativas*

Estas cópias secundárias podem ser objeto de processo de consulta, bem como servir de apoio à operações de atualização num ambiente de desconexão, através de transações de 2ª classe.

As *Versões Mestres*, semelhantes ao seu equivalente nas cópias primárias, refletem as atualizações ocorridas nas cópias secundárias, que se tornaram duráveis. As versões mestres podem ser atualizadas por operações de replicação ou através de mecanismos de certificação de transações de 2ª classe.

Versões Tentativas são as atualizações que estão em processo de certificação, ou seja, aguardando uma autorização do sistema para tornarem-se duráveis.

Cópias secundárias com prazo de validade

As cópias secundárias com prazo de validade possuem o mesmo valor da cópia primária ao *timestamp* equivalente, sendo consideradas, para efeitos de consulta, como consistentes e, portanto, com o mesmo valor das cópias primárias. O detentor da cópia é o único autorizado a modificar o seu valor, porém esta modificação somente pode ocorrer após a expiração do prazo de validade.

Da mesma forma que o seu equivalente da cópia primária, não existe versão tentativa deste tipo de dado.

4.1.2.3 Propagação das cópias secundárias

As *Cópias Secundárias* propagam-se, via de regra, em dois sentidos:

- *Sentido Ascendente*, em direção à parte superior da estrutura hierárquica (Ex: relatórios de dados dos escalões inferiores);
- *Sentido Descendente*, em direção à parte inferior da estrutura hierárquica (Ex: ordens).

Os relatórios de dados, em regra, propagam-se normalmente no sentido ascendente, isto é, atualizando a base de dados de cada SGBD que possua nível hierárquico superior.

As informações que trafegam no sentido descendente não são, em regra, propagadas para todos os filhos do nó, ou seja, cada filho recebe do nó pai apenas as informações que lhe sejam relevantes.

Entretanto, existem algumas exceções onde esta propagação segue um padrão distinto. Um exemplo seria a réplica parcial de dados de um batalhão nas OM geograficamente adjacentes.

4.1.2.4 Exemplo

A Figura 4.1 ilustra uma possível disposição dos tipos de dados definidos.

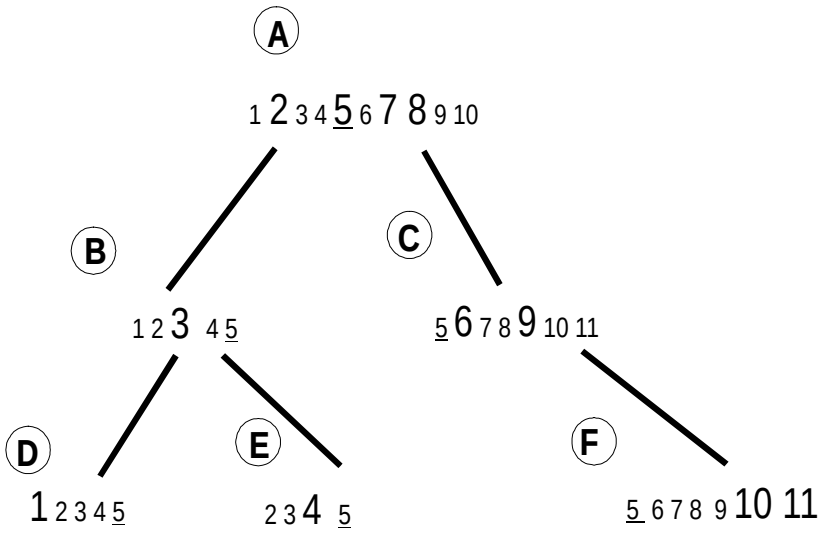

Figura 4.1 - Exemplo de disposição de dados

Os dados com tamanho de fonte maior representam as cópias primárias dos dados. As sublinhadas representam as cópias primárias com prazo de validade.

Os dados com tamanho de fonte menor e sublinhados representam cópias secundárias com prazo de validade.

Os dados com tamanho de fonte menor e não sublinhados representam as cópias secundárias sem prazo de validade.

Existem outros fatores complicadores neste tipo de ambiente. Um dos que mais se destaca é o movimento dos nós dentro estrutura da árvore, onde, por sua vez, as réplicas também deverão se movimentar e outras serão eliminadas.

No exemplo da Figura 4.2, a OM D passa a ser subordinada a OM C.

Ao modificar este vínculo de subordinação, deverão ser criadas novas réplicas na estrutura e, de acordo com a necessidade e manutenção da consistência, eliminar as réplicas anteriores.

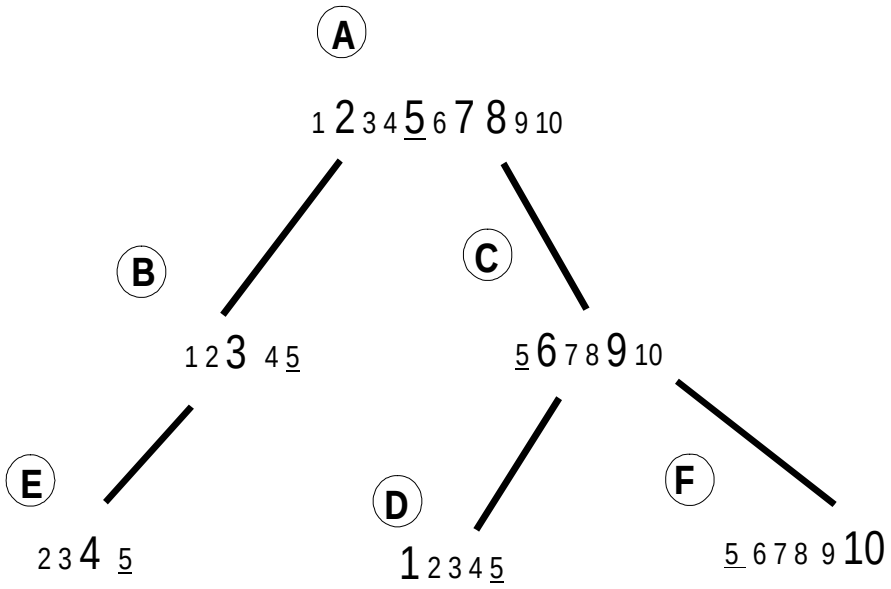

Figura 4.2 - Mudança de subordinação da unidade D

4.1.2.5 Transações

São através das transações que o usuário do sistema realiza as operações de consulta e atualização. Elas se dividem em dois grupos:

- *Transações de 1ª Classe*
- *Transações de 2ª Classe*

As *Transações de 1ª Classe* são aquelas que somente acessam, durante a sua execução, cópias primárias e/ou cópias primárias com prazo de validade (pode ocorrer o caso de uma transação operar somente em cópias primárias e não ser uma transação de 1ª classe). Produzem novas versões mestres nas cópias primárias. Não existem diferenças deste tipo de transação com o modelo tradicional de transações.

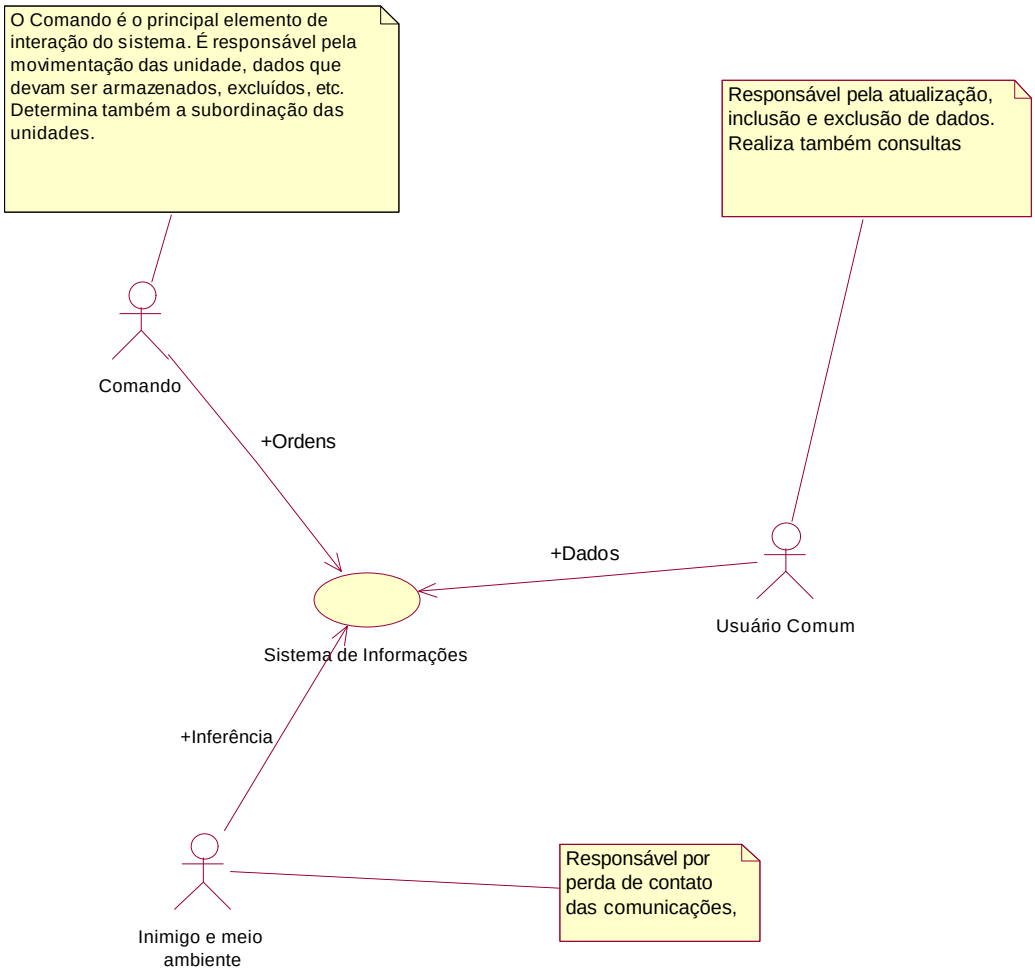

Figura 4.3 - Cenário de operação

As *Transações de 2ª Classe* produzem, via de regra, versões tentativas nos dados que operam (que podem ser cópias primárias ou secundárias) e que serão certificadas posteriormente, podendo transformar aquelas versões tentativas em versões mestres.

A compreensão completa de transações de 1ª e 2ª classe está na seção 4.5.

4.2 Componentes e mensagens do Modelo

A Figura 4.3 mostra os diversos atores que fazem parte do sistema.

O sistema de informações da Figura 4.3 pode ser decomposto conforme a Figura 4.4.

A Infraestrutura de Comunicações auxilia na difusão de informações, porém o escopo principal desta tese é o SGBDDMóvel.

4.2.1 Componentes do Modelo

Para que o sistema de informações funcione conforme o proposto, são necessários diversos pacotes, cada qual responsável por uma parte da funcionalidade. Apesar de serem pacotes separados, as funcionalidades de todos eles interagem, não podendo funcionar isoladamente.

A estrutura geral do sistema está ilustrada na Figura 4.5.

Os três principais pacotes do sistema são o *gerenciador de transação*, o *gerenciador de replicação* e o *gerenciador de localização*.

Existem outros pacotes (*Scheduler*, *Roteador de Mensagens*, *Data Processor*, etc.) agregados ao sistema, porém a função principal destes é auxiliar no funcionamento dos três *packages* anteriormente citados. Dentre estes *packages* auxiliares, destaca-se o *Scheduler*, responsável pelo controle de concorrência de dados existentes, especialmente entre o Gerenciador de Transações e o Gerenciador de Replicação.

Nota-se, pela Figura 4.5, que não existe acesso aos dados sem antes passar pelo *Scheduler*.

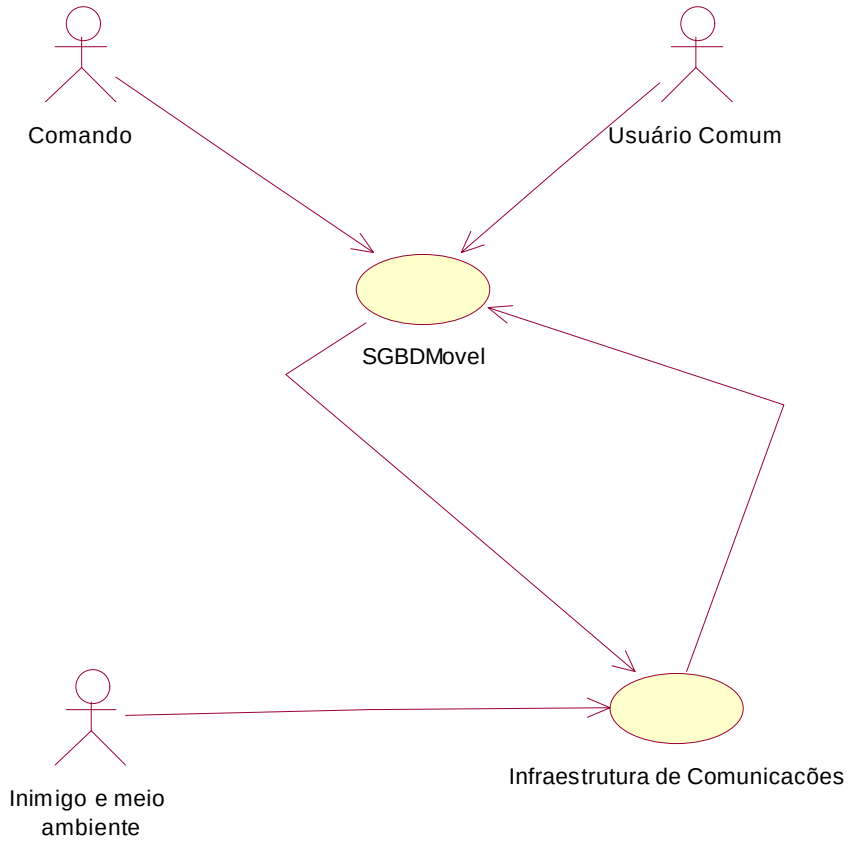

Figura 4.4 - Componentes do sistema de informações

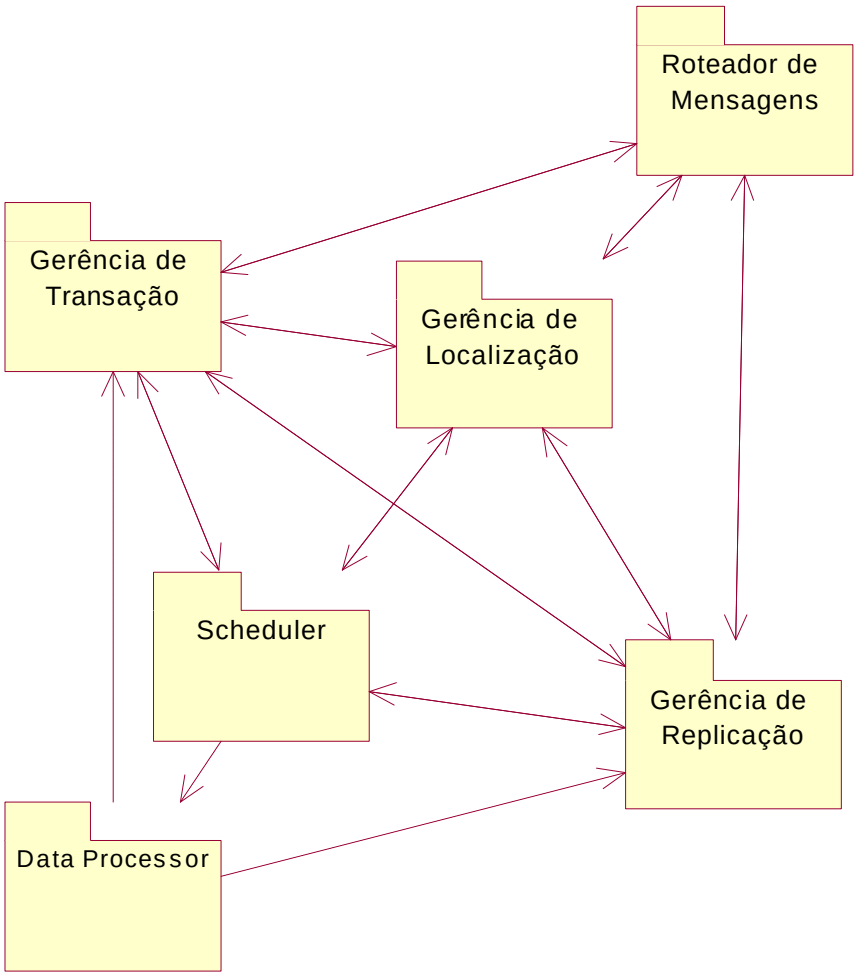

Figura 4.5 - Componentes internos do sistema

4.2.2 Mensagens do Modelo

As principais mensagens que trafegam entre os componentes do modelo são:

1. **Mapeamento de dados**: São modificações no diretório global de dados.
2. **Mudança de hierarquia**: Modificação na estrutura hierárquica do sistema.
3. **Mudança no esquema de replicação**: Modificação no sentido de que as réplicas irão movimentar-se. Pode inserir, remover ou modificar sentido de

movimentação (ascendente ou descendente). Pode prever parâmetros extras, como freqüência de replicação e duração de replicação.

4. **Operações de leitura simples**: São operações que não produzem resultado no controle de concorrência do sistema. Tenta retornar a “melhor” versão do dado.
5. **Transações de 1ª classe** : Executadas nas cópias primárias e replicadas posteriormente para os demais SGBDs.
6. **Transações de 2ª classe** : Tentativas locais de atualização. Sofrerão posterior processo de certificação para validação de suas operações.
7. **Relatórios de invalidação de *cache***: Realiza a invalidação das cópias secundárias existentes nos SGBDs.
8. **Atualização das cópias secundárias:** Envio dos valores das cópias primárias para fins de atualização das cópias secundárias existentes no SGBD.
9. **Relatórios de atividade**: Influencia a política de difusão das informações, informando os servidores dos estados dos clientes.
10. **Aviso de desconexão**: Informa que a estação pretende desconectar-se.

4.3 Esquema de Replicação em Duas Camadas

4.3.1 Modelos de replicação

Um esquema ideal de replicação deve atingir quatro objetivos:

Disponibilidade e escalonamento: Fornecer alta disponibilidade e escalonamento através de replicação, evitando-se a instabilidade do banco de dados.

Mobilidade: Permite aos nós móveis ler e atualizar o banco de dados enquanto estão desconectados da rede.

Serialização: Possibilita a execução de transações serializáveis.

Convergência: Possibilita que o sistema alcance o estado consistente.

Para realizar a replicação é necessário que se adotem estratégias de propagação em conjunto com estratégias de domínio.

4.3.2 Estratégias de propagação

Quando for executar a replicação, uma transação pode propagar suas atualizações remotamente como parte da própria transação (*eager*); ou como transações separadas (*lazy*).

Na estratégia *Eager*, as atualizações são aplicadas em todas as réplicas como parte da transação original.

Por sua vez, na estratégia *Lazy*, uma réplica é atualizada pela transação original. As atualizações das réplicas são processadas assincronamente, tipicamente como uma transação separada em cada nó.

A Figura 4.6 mostra estes duas estratégias de propagação.

A replicação *Eager* mantém todas as cópias sincronizadas em todos os nós através da atualização de todas as réplicas como parte de uma transação. A replicação *Eager* garante execução serializável, mas reduz a performance de atualização e aumenta o tempo de resposta em função das atualizações extras que são adicionadas àtransação. Utiliza esquemas de bloqueio para controlar execuções concorrentes. Entretanto, este esquema de replicação (*Eager*) não é uma opção para aplicações móveis onde a maioria dos nós está normalmente desconectada.

Aplicações móveis requerem algoritmos de replicação *Lazy* que processam assincronamente as atualizações das réplicas para os outros nós depois do commit das transações de atualização. O esquema de replicação *Lazy* pode utilizar o controle de concorrência de multiversões. Isto pode permitir que uma transação consulte um valor armazenado em um outro momento, que é diferente do valor atual.

Porém o esquema *lazy* tem algumas desvantagens, sendo a mais importante o fato de existirem versões desatualizadas no banco de dados.

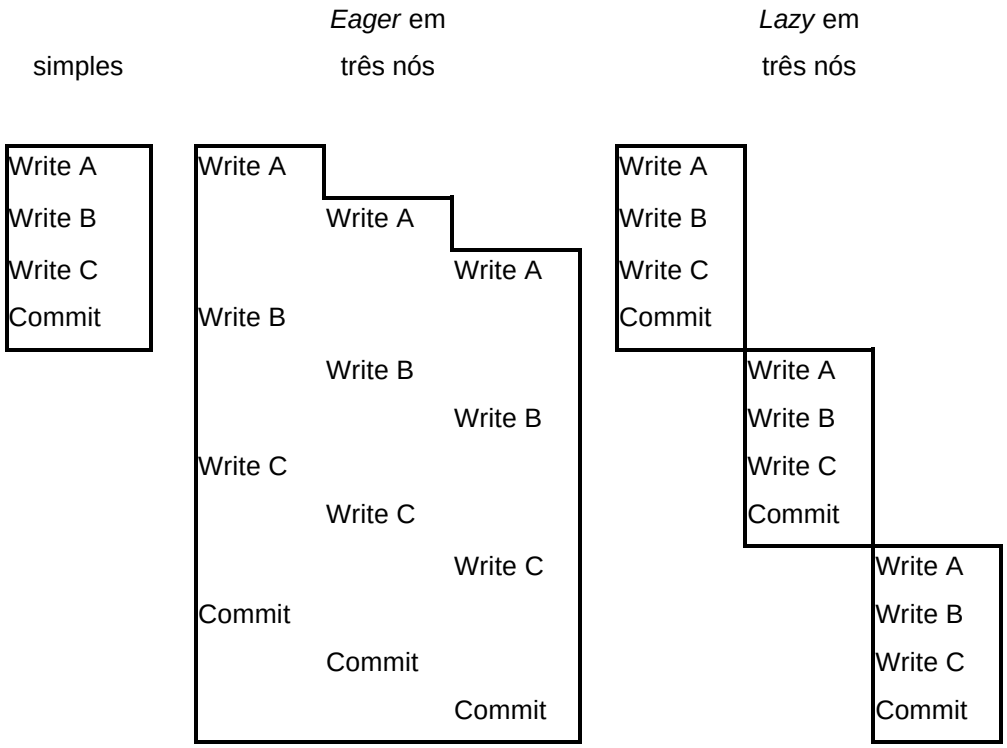

Figura 4.6 - Modo de propagação de réplicas

4.3.3 Estratégias de domínio

A Figura 4.7 mostra dois modos de regular atualizações de réplicas.

A estratégia de domínio diz respeito à política de autorização de atualização de réplicas. Quanto ao critério de quem pode alterar os dados do sistema, ele pode ser dividido em:

- *Group*: qualquer nó, com uma cópia de um item de dado, pode atualizá-lo. Isto é denominado *update anywhere*. Segue o modelo ROWA (*R*ead *O*ne *W*rite *A*ll).

- *Master*: cada objeto tem um nó mestre. Apenas o nó mestre pode atualizar a cópia primária do objeto. Todas as outras réplicas são *read-only*. Os outros nós que desejarem atualizar o objeto devem realizar um pedido ao detentor da cópia.

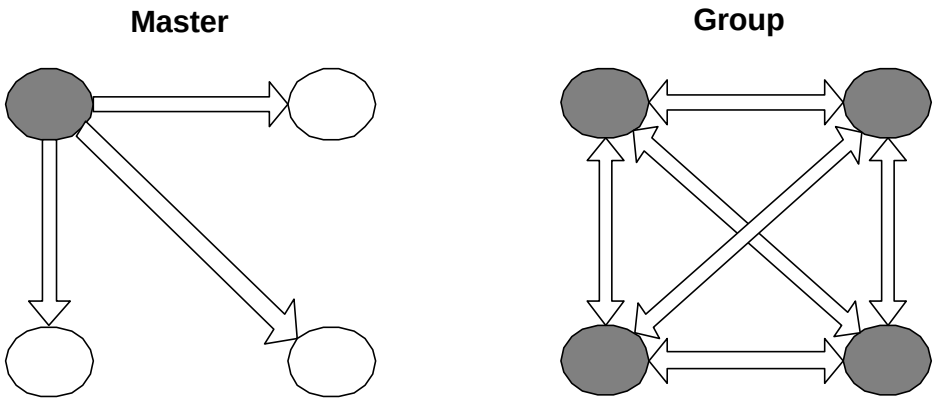

Figura 4.7 - Estratégias de domínio

4.3.4 Combinação de estratégias de propagação e domínio

Aplicando as estratégias de propagação e domínio conjuntamente, teríamos a seguinte taxonomia ilustrada na Tabela 4.1.

O modelo *Eager-Master* não possui replicação, pois a cópia permanece somente na estação onde ocorre a transação.

O modelo *Eager-Group* apresenta o problema de não poder ser executada quando houver estações desconectadas fazendo parte da transação. Ademais, a probabilidade de *deadlocks* e conseqüentemente o crescimento de transações abortadas aumentam com o tamanho das transações e com o número de nós. Um aumento de magnitude 10 no número de nós aumenta a probabilidade de falhas em transação numa magnitude de 1000 [GHOS96].

	Lazy	Eager
Group	N transações N proprietários	1 transação N proprietários
Máster	N transações 1 proprietário	1 transação 1 proprietário

Tabela 4.1 - Propagação vs Domínio

No modelo *Lazy-Group*, é possível que dois nós atualizem o mesmo objeto e concorram um contra o outro para propagar suas transações para os outros nós. O mecanismo de replicação deve detectar estas ocorrências e reconciliar estas transações de forma que estas não sejam descartadas. Ao invés de informar que uma transação falhou, cria-se um cenário mais adverso. No modelo *Eager*, tínhamos como principal ponto negativo a possibilidade do *deadlock* retardar a execução de aplicações. O *Lazy-Group* possibilita a ocorrência de um falso commit.

O modelo *Lazy-Master* também não é adequado para aplicações móveis. Um nó que deseja atualizar um objeto (cópia primária) deve estar conectado ao detentor deste e participar em uma transação da qual fazem parte aquele nó e o detentor da cópia.

A solução adequada para os problemas expostos anteriormente, no qual se enquadra o gerenciamento de dados no ambiente de comando e controle (redes móveis em um ambiente de fraca conectividade), requer um esquema de replicação modificado, alterando sutilmente as estratégias de domínio e propagação. Este esquema será denominado *Replicação em Duas Camadas* [GHOS96], [LV94], [LV95].

Para evitar a reconciliação (problema que ocorre no modelo *Lazy-Group*), cada objeto possui uma cópia primária que é administrada por um nó. Todos os nós poderiam realizar tentativas de atualização destas cópias, independentemente de estarem conectados ao nó detentor da cópia primária. As tentativas de atualização seriam posteriormente submetidas a um processo de certificação, que as tornariam válidas ou não.

4.3.5 Replicação em duas camadas

O esquema de replicação em duas camadas assume que existam dois tipos de nós:

Nós móveis que estão desconectados a maior parte do tempo. Armazenam cópias primárias e secundárias. Sobre as cópias secundárias originam-se transações de 2ª classe.

Nós base que estão normalmente conectados. Armazenam também cópias primárias e secundárias do banco de dados. Em nossa aplicação, os nós superiores podem ser considerados como os nós base, pois a maioria das atualizações que ocorrem são aquelas realizadas pelos nós inferiores para os nós superiores. Entretanto, no modelo proposto, não haverá restrição de considerar os nós superiores como nós móveis e os inferiores como nós base.

Neste modelo de replicação, temos dois tipos distintos de transações:

Transações de 1ª classe: transações de 1ª classe operam apenas sobre versões mestres das cópias primárias e cópias (primárias ou secundárias) com prazo de validade. Quando executada com sucesso, produz novas versões mestres nas cópias primárias.

Transações de 2ª classe: este tipo de transações possibilita que estações que estejam desconectadas dos nós que possuem as cópias primárias realizem transações, superando as limitações existentes nos modelos *Eager* e *Lazy-Master*. Produzem versões tentativas nas cópias secundárias (e, conforme o caso, versões tentativas em cópias primárias), para depois serem submetidas a um processo de certificação.

Quando o sistema está conectado, o comportamento deste esquema de replicação opera analogamente ao modelo *Lazy-Master*, com o detalhe de que as réplicas são transmitidas individualmente, ao invés de seguirem o modelo transacional.

Quando o sistema está desconectado, o comportamento é diferente. Suponha que uma estação esteja desconectada há um dia. Existe uma cópia das bases de dados de um dia atrás. Foram geradas transações de 2ª classe na base de dados local sobre cópias

secundárias e primárias. Isto gerará versões tentativas sobre estes dados. Se a estação realizar uma consulta sobre estes dados, ela obterá versões tentativas.

Quando uma estação móvel se conecta a um nó superior, a estação:

1. Aceita as réplicas atualizadas das OM superiores, atualizando as versões mestres de suas cópias secundárias.
2. Descarta todas as suas versões tentativas que forem incompatíveis com as atualizações dos detentores das cópias primárias.
3. Envia a atualização de todas as suas cópias primárias para as estações que possuam cópias secundárias destes dados.
4. Inicia o processo de certificação de suas transações de 2ª classe.
5. Recebem mensagens de sucesso ou falha deste processo de certificação. Certificações bem sucedidas transformarão as versões tentativas daquelas cópias secundárias em versões mestres, tornando-as duráveis. Certificações mal-sucedidas descartarão as versões tentativas, descartando, também, transações subseqüentes que operaram sobre aqueles dados.

Quando se reconecta com o nó subordinado, o nó superior:

1. Envia as atualizações de suas cópias primárias para as estações que possuam cópias secundárias destes dados.
2. Aceita as réplicas atualizadas das cópias primárias das unidades inferiores, atualizando as versões mestres destas cópias secundárias.
3. Recebe as operações oriundas de transações de 2ª classe. Envia o resultado para o gerenciador de transações de 2ª classe que está na estação do nó subordinado, que checará se o resultado é compatível ou não com a sua base de dados. Caso seja compatível, este enviará a notificação de *commit* para tornar as modificações duráveis nos nós superiores, modificando, assim, as versões mestres das cópias primárias.
4. Caso *confirmar* uma transação de 1ª classe, ele propaga a atualização não como uma transação para as outras réplicas e sim cada dado individualmente, de acordo com a importância de cada cópia primária.

5. Repassa réplicas atualizadas de outros nós, isto é, as réplicas recebidas de um nó que lhe é superior para o nó inferior. A mesma tarefa é realizada no sentido ascendente.

Quando todas as réplicas atualizadas forem transmitidas, o SGBDD móvel estará em consistência lógica.

As propriedades do esquema de replicação de duas camadas são:

1. Todos os nós podem tentar realizar tentativas de atualizações no banco de dados em cópias primárias das quais não sejam detentores.
2. Uma transação de 2ª classe torna-se durável após o processo de certificação.
3. As réplicas em todos os nós conectados convergem para o estado do sistema base.

4.4 Gerência de Localização

O gerenciador de localização armazena as três principais estruturas do sistema:

- *Diretório global de dados;*
- *Estrutura hierárquica;*
- *Esquema de replicação*

4.4.1 Diretório global de dados

O Gerenciador de Localização, na sua forma mais simples, contém todas as informações sobre o sistema e sobre os banco de dados e transações já definidos. Essas informações são armazenadas sob forma interna para serem usadas pelos outros componentes do SGBD e, naturalmente, é fundamental para o funcionamento do sistema.

Este papel básico do gerenciador de localização pode ser expandido com outras facilidades, tornando-se a principal ferramenta de administração do banco e a principal fonte de informação sobre o significado dos dados para os usuários.

O diretório, por si mesmo, é um banco de dados que contém metadados sobre os dados armazenados no SGBD. Um diretório pode ser global em relação ao sistema como um todo, ou local em cada site.

Um diretório pode ser mantido centralizado em um site, mas esta solução, obviamente, não pode ser utilizada na aplicação em questão. Porém, a sua distribuição nas estações do sistema aumenta a complexidade da manutenção dos diretórios.

O *diretório global de dados* identifica univocamente as cópias primárias dos dados, juntamente com os seus detentores. Armazena também dados referentes à permissão de escrita e leitura.

Nas estações, indica se aquele dado representa uma cópia primária, uma cópia secundária, se possui prazo de validade ou não.

Para se realizar modificações neste diretório global de dados é necessário o envio de *mensagens de mapeamento de dados* (4.2.2).

As mensagens de mapeamento de dados são também utilizadas para a criação de novas cópias secundárias nos nós.

4.4.2 Estrutura hierárquica

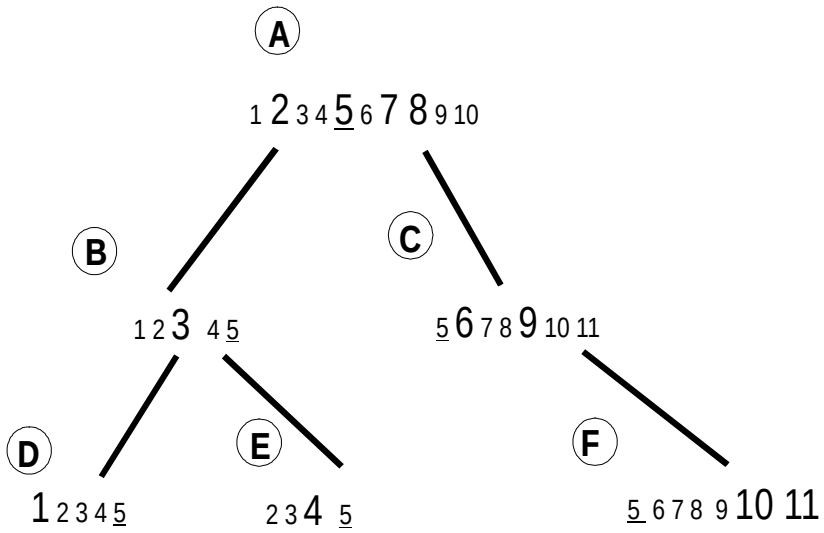

Figura 4.8- Estado original do sistema

A *estrutura hierárquica* é como as unidades (ou SGBDs) estão dispostas em relação ao vínculo de subordinação. Esta estrutura não é fixa, podendo variar no tempo em função da mudança de subordinação das unidades.

Na Figura 4.8, tem-se um exemplo de um possível estado original do sistema.

Existem várias maneiras de se implementar o mapeamento da estrutura hierárquica, dentre as quais pode-se destacar [CM95]:

- *Replicação completa da estrutura hierárquica em cada nó*
- *Replicação parcial da estrutura hierárquica*

A *replicação completa* possibilita a interação direta de componentes remotos (4.5.2). Em contrapartida, é necessário difundir por toda a rede as modificações de estrutura que, por ventura, ocorrerem. Para redes com poucos nós, este modelo não traz maiores complicações. Porém, em redes com maior número de nós e mudanças constantes, o custo de atualização da estrutura hierárquica nos nós torna-se inviável.

A *replicação parcial* segue o esquema de mapeamento de estrutura hierárquica proposto na seção 3.4.2, desta dissertação, onde o Gerenciador de Localização no nível mais alto contém informações de localização dos nós que estão localizados nos

níveis inferiores ao dele. Um gerenciador de localização de um nível intermediário mantém as informações dos nós que estão dentro do conjunto de sua sub-árvore.

O esquema de *replicação parcial* admite duas variações [CM95]:

- *Armazenagem de ponteiros*
- *Armazenagem da estrutura*

No modelo de *armazenagem de ponteiros*, a operação sobre os dados é realizada de maneira indireta, necessitando da colaboração de outros gerenciadores de localização para encontrar a informação desejada (4.5.2).

A estrutura de *armazenagem de ponteiros* tem menor custo de atualização da estrutura hierárquica nos nós nas mudanças de subordinação. Neste modelo, uma modificação na estrutura não necessitaria ser propagada até a raiz da árvore, mas, sim, até a raiz da sub-árvore em questão. Em contrapartida, o mecanismo de consulta e atualização deve percorrer uma lista de ponteiros de nós, até que se possa operar sobre o dado desejado.

A *armazenagem da estrutura* facilita o mecanismo de execução de consultas e atualizações. A desvantagem óbvia é a necessidade de se propagar a modificação da estrutura até a raiz da árvore.

Em redes móveis, a *iniciativa* do registro na estrutura hierárquica é do *usuário*. No ambiente de comando e controle, a iniciativa do processo seria do nó responsável pela mudança de hierarquia, que possui um nível hierárquico maior ou igual ao da raiz da sub-árvore, o LCA (*Least Common Ancestor*), ou seja, o ancestral mais próximo entre a posição anterior e a designada.

Um aspecto interessante na estrutura hierárquica seria a informação que cada nó possui a respeito de seus superiores hierárquicos. Pode-se adotar duas estratégias:

- *Conhecimento do superior imediato*
- *Conhecimento de toda a cadeia de comando*

O *conhecimento do superior imediato* possui a vantagem de se efetuar uma menor custo de atualização da estrutura hierárquica no Gerenciador de Localização. Em contrapartida, o mecanismo de consultas e atualizações é mais complexo, necessitando da cooperação dos outros gerenciadores de localização nas tarefas de consulta e atualização de dados.

O mecanismo de consulta e atualização é mais simples quando se tem o *conhecimento de toda cadeia de comando*. Porém, o custo de manutenção da estrutura hierárquica do Gerenciador de Localização é maior, devendo-se estender a todos os nós subordinados do nó movimentado.

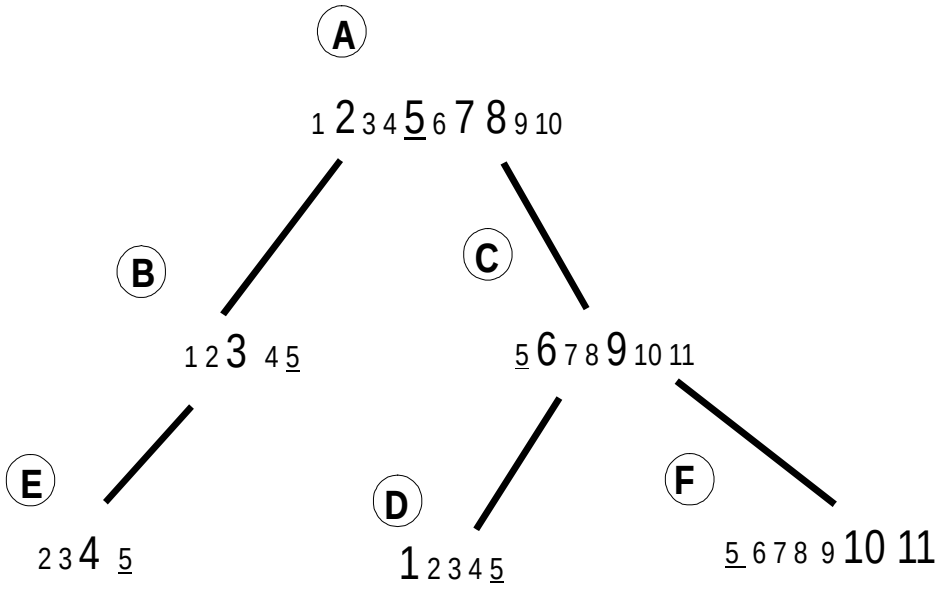

Figura 4.9 - Mudança de subordinação do nó D

A mensagem de *mudança de hierarquia* (4.2.2) será, obviamente, função do modelo adotado. Existe, porém, uma característica que é comum a todos os modelos de que a iniciativa será de um nó com nível hierárquico igual ou maior ao da sub-árvore (LCA) em questão e. ainda, dentro da cadeia de comando.

No caso da *replicação completa da estrutura hierárquica* em cada nó, a mensagem de *mudança de hierarquia* será idêntica para todos os nós.

Na replicação parcial da estrutura hierárquica, com *armazenagem de ponteiros*, o processamento de mensagens opera diferentemente. Considere o caso de mudança da estrutura hierárquica da Figura 4.8 para a Figura 4.9.

O LCA, neste caso, é o nó A, devendo partir deste a iniciativa de atualização das estruturas hierárquicas contidas nos gerenciadores de localização. O nó modifica a posição do nó D, que estava ligado ao nó B para o nó C. Envia uma mensagem para o nó B com dois comandos: apagar D de sua tabela e informar o nó D de seu novo superior hierárquico. Envia uma mensagem para o nó C informando que o nó D passa a ser seu subordinado direto.

O nó B, sendo superior imediato de D, apaga o registro do nó D de sua tabela e informa este nó de que seu superior passa a ser o nó C.

O nó C, recebendo a mensagem do nó A, insere o nó B na sua lista de filhos imediatos.

No caso de *armazenagem de estrutura*, a mensagem consistiria em informar o nó que estava mudando a sua posição e o seu novo pai. Esta mensagem seria propagada para todos os nós descritos no parágrafo anterior e aqueles que estivessem na cadeia de comando até o nó raiz.

Nos *nós subordinados com conhecimento da cadeia de comando*, seria necessária a difusão da informação para todos os nós de posição hierárquica inferior ao nó que está mudando o seu vínculo de subordinação.

4.4.3 Esquema de replicação

O *esquema de replicação* reproduz os dados segundo o modelo proposto na seção 4.1. Alguns dados têm o sentido ascendente da árvore hierárquica. Outros, o sentido descendente. Existem ainda aqueles que tem procedimento misto, ou seja, sobem e descem na árvore hierárquica. Os dados possuem informações de sua origem (se é subordinada ou superior) e sua saída (que pode ser subordinada, superior ou ambos).

Na Figura 4.8, sob o ponto de vista da estação B, o dado 4 tem sua origem como inferior (a estação recebeu o dado de um unidade subordinada, no caso a estação E) e

retransmite o dado para os dois escalões (subordinado e superior), pois tanto a estação A (que é seu superior) como a estação D (que é sua subordinada) possuem, segundo o esquema de replicação, cópias secundárias do dado 4. A Figura 4.10 ilustra uma possível estrutura de dados que represente este esquema de replicação.

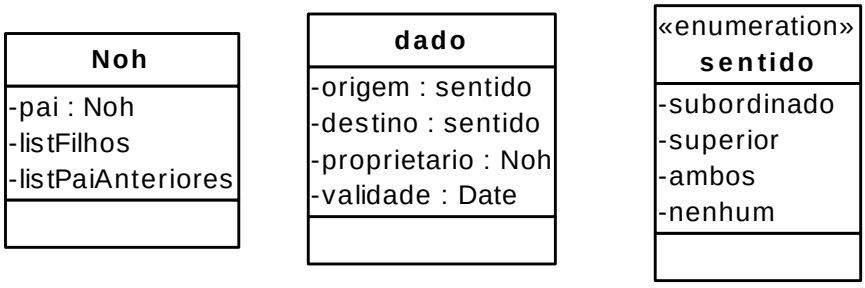

Figura 4.10 - Estruturas de dados do gerenciador de localização

O padrão normal é que relatórios de unidades subordinadas tenham sentido ascendente.

Relatórios de informações de interesse geral ou particular têm sentido descendente.

Ordens do escalão superior têm sua cópia primária na unidade que gerou a ordem e uma réplica na unidade subordinada que recebeu a ordem. O cumprimento ou não da ordem é feito através de atualizações na cópia primária do escalão superior. Esta transação pode ser uma transação de 1ª classe ou através de um processo de certificação de uma transação de 2ª classe realizada pelo escalão inferior .

Consultas freqüentes a determinados dados geram modificações no esquema de replicação, criando novos fluxos de informação.

As *mensagens do esquema de replicação* (4.2.2) são inseridas, a princípio, durante a configuração do sistema e incluídas posteriormente sob demanda. O sentido ascendente ou descendente é obtido através de uma consulta à estrutura hierárquica, que informa a posição do nó detentor da cópia primária (por exemplo, *NCP*) com o nó (por exemplo, *X*) em questão. Se o nó que possui a cópia primária não for subordinado do nó, a origem de replicação é descendente, passando obrigatoriamente pelo LCA(*NCP*, *X*).

O *esquema de replicação* é dependente da *estrutura hierárquica*. Assim, as *mensagens de mudança de hierarquia* irão disparar *mensagens de mudança do esquema de replicação*, porém a modificação automática do *esquema de replicação* baseado nas mudanças de *estrutura hierárquica* é complexa e de difícil implementação, pois poderão existir cópias secundárias de unidades que não possuem relação de subordinação (por exemplo, na Figura 4.8, a réplica do dado 4 com cópia primárias na unidade D e secundárias em E).

4.4.4 Algoritmos de busca de réplicas

No esquema de replicação proposto, a confiabilidade das cópias secundárias é função da distância hierárquica entre a cópia primária e a cópia secundária. Isto é resultado da fraca conectividade do sistema, onde o sistema de difusão de réplicas está sujeito às condições de operabilidade dos sistemas de comunicações.

No exemplo da Figura 4.8, o nó D possui a melhor (e obviamente a representação correta por ser a cópia primária) representação do dado 1. Em decorrência do esquema de replicação, o nó B possuiria a segunda melhor versão, pois é o primeiro nó a receber o conteúdo da informação contida em 1. Por sua vez, o nó B é o responsável por retransmitir esta informação para os nós A e E, e assim sucessivamente. Desta forma, a informação do dado 1 no nó A irá depender das condições de comunicação entre os nós D e B e entre a própria estação A e B, comprovando, assim, a menor confiabilidade das cópias secundárias à medida que se afastam das cópias primárias.

Os dados são localmente identificados com outras informações: nó detentor da cópia primária; direção de sua origem (superior, subordinada ou nenhuma se for a cópia primária); e a direção de seu destino (superior, subordinada, ambos ou nenhum). A Figura 4.10 modela este representação.

Um sistema de comando e controle, em virtude de sua forma piramidal, privilegia os nós que estão nos níveis mais altos da estrutura hierárquica. Para estes nós, existe um

esquema de consulta que permite a consulta de cópias secundárias sem seguir o caminho normal de propagação das mesmas, baseando-se num histórico de estruturas hierárquicas.

Por exemplo, suponha que, o nó D perca o seu vínculo de subordinação com o nó B e passe a ser subordinado ao nó C, a figura de subordinação ficaria ilustrada conforme a Figura 4.9.

Suponha também que, logo após ter mudado a estrutura de subordinação, o nó A deseje fazer uma consulta ao dado 1, cuja cópia primária está no nó D. O dado 1 não possui prazo de validade, de sorte que a cópia secundária de 1 existente na estação A não é uma réplica confiável do dado 1.

Seguindo o esquema proposto de confiabilidade, a melhor conduta seria consultar a cópia primária de 1, que está no nó D.

Mas o nó D poderia estar, por hipótese, desconectado da rede.

O nó que poderia, então, ter o valor mais confiável de 1, segundo o esquema de replicação proposto, seria o nó imediatamente superior à D. Como a mudança de hierarquia foi realizada recentemente, é provável que C ainda não tenha recebido as réplicas do nó D. Porém, as réplicas ascendentes, como afirmado anteriormente, permitem um esquema alternativo de busca.

Como C ainda não tem o valor de 1, não poderia ter enviado o seu dado para o seu superior, inviabilizando a sua consulta.

Neste caso, quem poderia ter a melhor réplica do dado 1 seria B, o antigo superior do nó D.

Para cumprir esta tarefa, a eliminação de cópias secundárias anteriores (no exemplo seria a eliminação da cópia secundária do dado 1 no nó B), em virtude de mudanças do esquema hierárquico, não deve ser realizada logo após a mudança de hierarquia e sim após um determinado período de tempo

A Figura 4.11 ilustra o algoritmo de busca de réplicas de origem ascendente. Uma variação interessante deste algoritmo seria a escolha da versão mais atualizada. Ao invés de terminar a execução do algoritmo, este iria comparando as versões até não haver mais pais anteriores, escolhendo ao final do algoritmo, a versão mais atualizada. Cabe aqui ressaltar, que este algoritmo não está levando em consideração a estratégia de armazenamento de estruturas hierárquicas, considerando apenas a execução de consulta do *Data Processor* de cada nó. Os detalhes referentes a interação do gerenciador de localização com os demais componentes do SGBD, seja local ou remoto, em função da estratégia de armazenamento de estruturas hierárquicas, será abordado na seção 4.5 desta dissertação.

A forma de execução deste algoritmo depende do modelo de estrutura hierárquica adotada. Na replicação total, não é necessária modificação alguma. Na replicação parcial, este algoritmo é adaptado para que possa ser executado distribuidamente.

Justifica-se, assim, o armazenamento de versões de estruturas hierárquicas. Funcionalidade análoga com dados históricos poderia existir em relação aos dados de sentido descendente, porém isto não é abordado em função de dois fatores:

- Os nós superiores estão normalmente conectados e os nós inferiores estão normalmente sujeitos ao ambiente de fraca conectividade
- O objetivo principal de um sistema de comando e controle é privilegiar os escalões superiores.

O algoritmo de busca de réplicas que possuam o sentido descendente é mais simples e se resume a percorrer o caminho das réplicas. Exemplificando, suponha que, na Figura 4.8, o nó D deseja saber o valor de 2. A cópia primária de 2 está no nó raiz. O nó D, de posse desta informação, tenta consultar A. Caso o nó A esteja indisponível, a consulta passa para o nó B, que é o próximo nó no sentido descendente. Segue-se este processo até que o nó em questão passa a ser o próprio nó que está realizando a consulta. Como no algoritmo de busca de réplicas ascendentes, este pode ser estendido para que recupere a versão mais atual. Analogamente ao outro algoritmo, não se está considerando o modelo de estrutura hierárquica adotado.

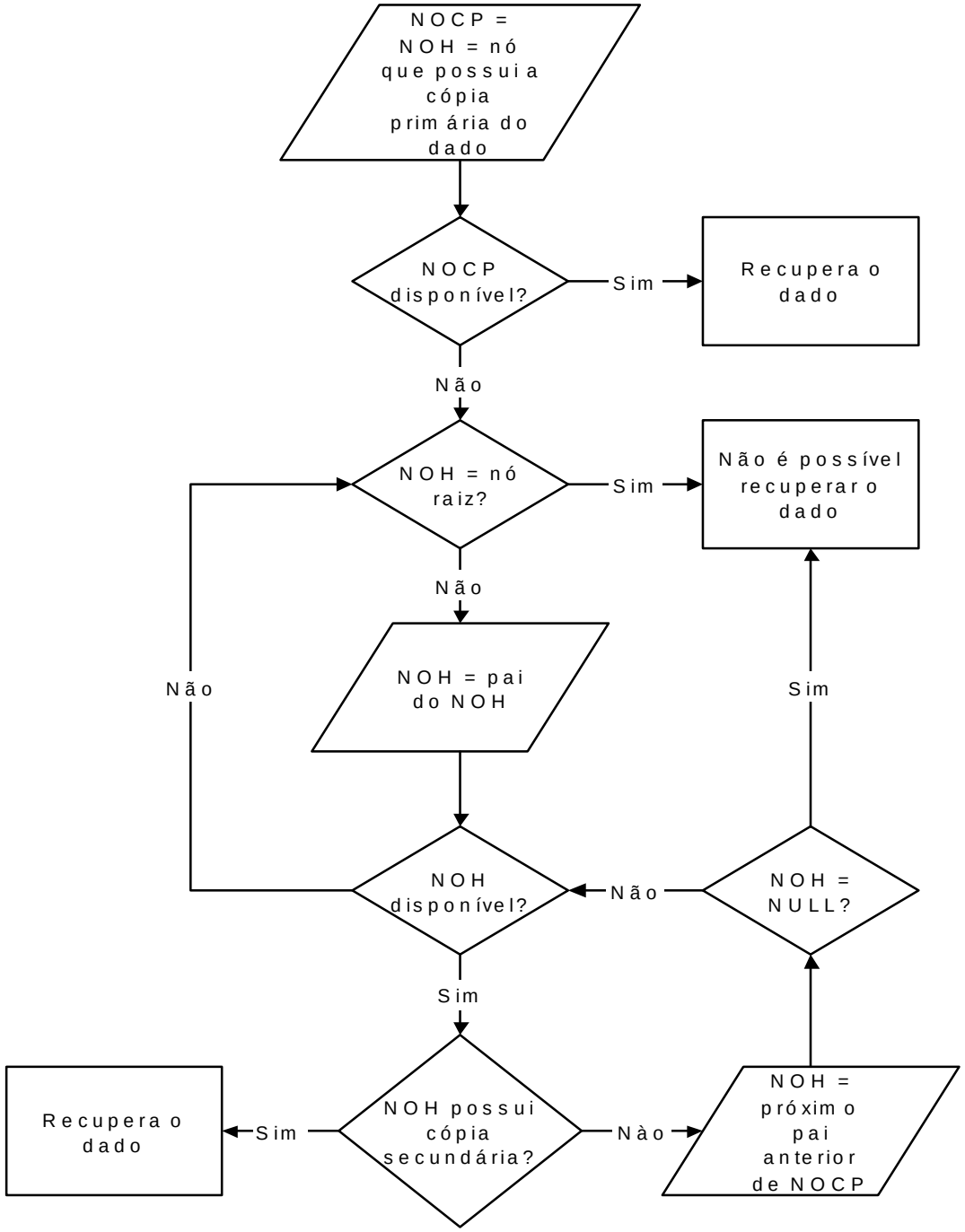

Figura 4.11 - Algoritmo de busca de réplicas de origem ascendente

A forma de execução deste algoritmo depende do modelo de estrutura hierárquica adotada. Na replicação total, não é necessária modificação alguma. Na replicação parcial, este algoritmo deve ser adaptado para que possa ser executado distribuidamente.

Caso o caminho de propagação das réplicas tenha sentido ascendente e descendente, aplicam-se os dois algoritmos conjuntamente.

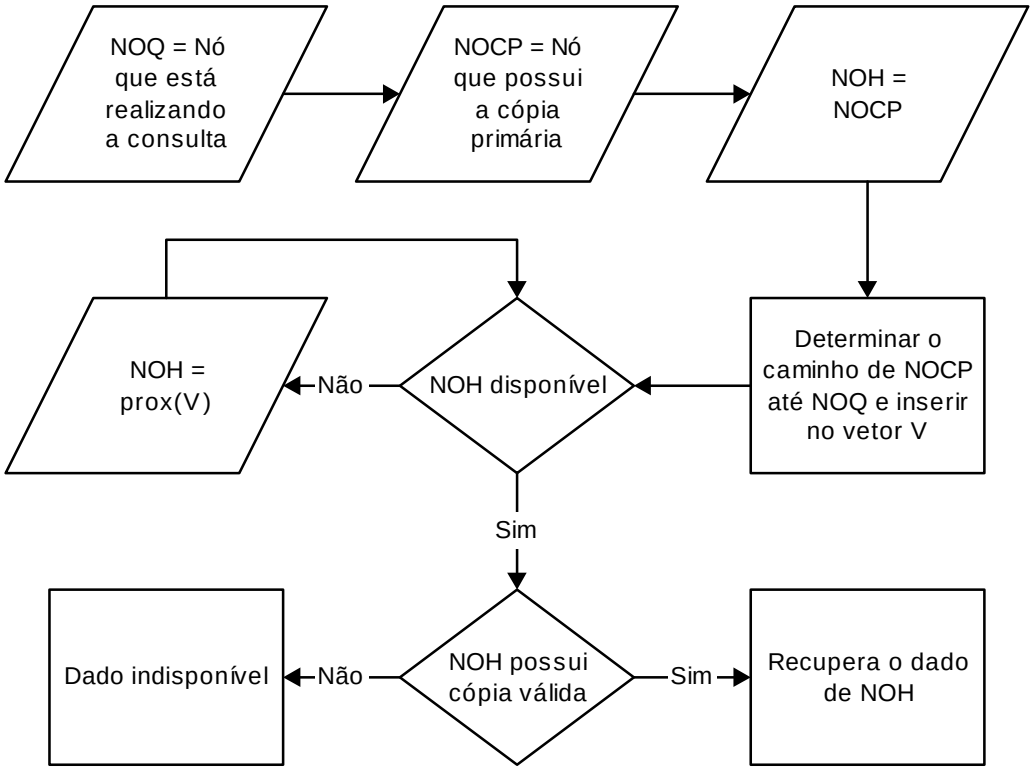

Figura 4.12 - Algoritmo de busca de réplicas descendentes

4.4.5 Movimentação de réplicas

Além deste procedimento de consulta, é necessário que as réplicas se movimentem. O nó D alimentará o nó C com réplicas de suas cópias primárias. Porém este processo deve ser executado paralelamente. Neste caso, além da alimentação vinda dos nós inferiores, esta alimentação pode vir de antigos nós superiores. Este assunto será abordado detalhadamente na seção 4.6.

Ao final da operação de réplicas poder-se-ia ter o seguinte cenário descrito na Figura 4.13.

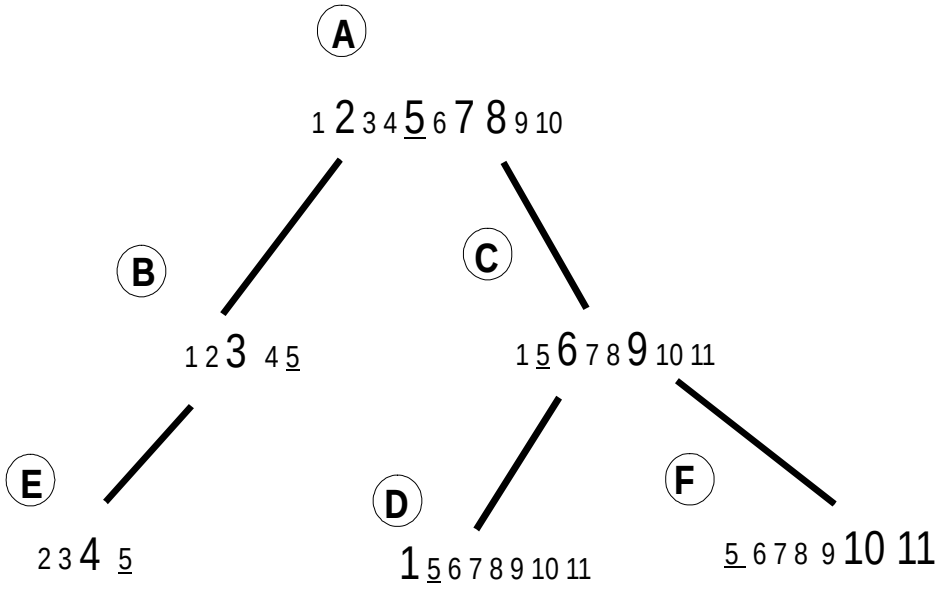

Figura 4.13 - Organização final das réplicas após mudança de subordinação

4.4.6 Interação com os outros pacotes do sistema

Como visto na seção 4.2, o gerenciador de localização interage com o *gerenciador de transações*, *gerenciador de replicação* e o *Scheduler*.

Em alguns modelos de estrutura hierárquica, o *gerenciador de localização* interage com outros *gerenciadores de localização*. Este assunto será abordado com maior profundidade na seção 4.5.

O *gerenciador de localização* interage com o *gerenciador de transação*, fornecendo-lhe informações sobre as permissões do sistema (respondendo, por exemplo, se o gerenciador de transações pode operar sobre aqueles dados), sobre a localização das cópias primárias e réplicas dos dados e informações referentes àestrutura hierárquica.

O *gerenciador de localização* interage com o gerenciador de replicação informando-o como deve propagar as suas réplicas. Mudanças na estrutura hierárquica ocasionam mudanças no esquema de replicação. São também previsíveis mudanças na forma com a qual a replicação deve funcionar durante a operação do sistema. Isto é

administrado através de mensagens de mudança de estrutura de replicação. No exemplo anterior, poderia ser solicitado ao nó B que enviasse réplicas dos dados de D para o nó C, em virtude da mudança de subordinação.

A interação com o *Scheduler* refere-se, principalmente, ao mapeamento de dados, como a criação e eliminação de cópias primárias e secundárias.

4.5 Gerência de transações

O gerenciador de transações é o componente responsável pela integridade do sistema. O gerenciador de transações interage com o *Scheduler*, enviando as operações de leitura e escrita que deverão ser executadas pelo gerente de dados. As operações executadas pelo SGBD são empacotadas em transações.

Para se modelar o gerenciamento de transações deve-se levar em consideração o modelo adotado, as operações do sistema e o protocolo de concorrência e terminação.

Supõe-se, ainda, que o banco de dados é modelado por N objetos nomeados x_1, x_2, ..., x_n, e que estes objetos terão cópias e versões conforme descrito nas seções posteriores. As operações de consulta e atualização são simplificadas para operações de leitura e escrita. Esta abstração visa facilitar a definição e análise de algoritmos.

O modelo é baseado numa adaptação do esquema de duas camadas proposto por [GHOS96], [LS94] e [LS95], baseado em *transações de primeira classe e transações de segunda classe*.

Em função do resultado de suas operações, dividem-se as transações em dois tipos [BHG88]:

- Transações que possuem apenas operações de leitura (Ex: $T_1 = r[x]r[y]r[z]c_1$) e;
- Transações que possuem operações de escrita. (Ex: $T_2 = r[x]w[y]r[z]c_2$)

O protocolo de controle de concorrência a ser utilizado é o de pré-ordenação com suporte a multiversão. O protocolo de terminação será o *Presumed Abort 2 Phase Commit* (PA2PC) [BHG88].

4.5.1 Multiversões de dados

A utilização de versões múltiplas no controle de concorrência possibilita que o *Scheduler* não rejeite operações de leitura e diminua a rejeição das operações de atualização [CM84].

Neste tipo de ambiente, em função da fraca conectividade, é alta a probabilidade que as operações cheguem atrasadas. Assim, o valor do dado que supostamente seria lido caso não houvesse problemas com as comunicações, foi substituído por um mais novo em virtude de uma operação de atualização executada por um outro nó, durante o período de desconexão.

Uma forma de minimizar esta rejeição seria manter uma série histórica com todas as versões do valor de cada dado. Quanto maior a série, maior é a chance do valor de uma ação de leitura estar disponível.

No algoritmo de controle de concorrência de multiversão, cada operação de escrita do dado *x* (W(*x*)), produz uma nova cópia (ou versão) de *x*. O *Data Processor* que gerencia *x* mantém uma lista de versões deste dado, que é a história de valores que o *Data Processor* designou para *x*. Para cada *Read*(*x*), o *Scheduler* não apenas decide quando enviar o *Read* para o *Data Processor*, mas também diz ao *Data Processor* quais das versões de *x* é para ser lida.

No método de controle de concorrência de pré-ordenação multiversionado, as atualizações não modificam o banco de dados. Cada operação de escrita cria uma nova versão daquele item de dado. Cada versão é marcada pelo *timestamp* da transação que o criou. Assim, o algoritmo de pré-ordenação multiversionado troca

espaço em disco por tempo. Desta maneira, ele processa cada transação no estado em que o banco de dados se encontrava.

A existência de versões é transparente para os usuários que executam transações. Estas fazem referências a itens de dados, não a uma versão específica. O gerenciador de transações designa um *timestamp* para cada transação que também é armazenado. As operações são processadas pelo *Scheduler* da seguinte forma [OV99]

1. Suponha que a ação é uma leitura de $R_i(x)$ com *timestamp i*. Deve-se retornar a versão de x (digamos, x_v), tal que $ts(x_v)$ é o maior *timestamp* das versões existentes de x menores que $ts(T_i)$. A operação $R_i(x_v)$ é então enviada para o *Data Processor*.
2. Suponha que a ação é uma atualização $W_i(x)$ com *timestamp i*. Representa-se esta operação como $W_i(x_w)$, com $ts(x_w) = ts(T_i)$. A operação é enviada para o *Data Processor* se, e somente se, não existe nenhuma transação com um timestamp maior do que $ts(T_i)$ que já leu uma versão de x (por exemplo, x_r, tal que $ts(x_r) > ts(x_w)$). Em outras palavras, se o *Scheduler* já processou uma operação de leitura $R_j(x_r)$ tal que $ts(T_i)<ts(x_r)<ts(T_j)$, fazendo com que $W_i(x)$ seja rejeitada

A leitura de dados no *Data Processor* é realizada somente sobre dados *confirmados*.

Quando uma operação é enviada para o *Data Processor*, o *Scheduler* necessita interromper temporariamente o envio de outra operação incompatível, porém dentro dos parâmetros de aceitação para o *Data Processor*, até que o primeiro seja inteiramente processado e aceito. Um exemplo é uma operação de leitura que é posterior a uma operação de escrita do mesmo dado.

Cabe ressaltar que os problemas de terminação persistem nesta implementação e que são necessárias operações periódicas de descarte de versões excedentes.

Em virtude do esquema de multiversões, as operações de leitura sempre retornam um valor. Entretanto, a utilização do esquema de multiversões não consegue superar a limitação da consistência das cópias secundárias com prazo de validade. Assim, mesmo com o uso de versões, é possível, em virtude da fraca conectividade do

sistema, que nem todas as versões de um determinado dado sejam replicadas para as cópias secundárias.

Este método produz consistência nas operações de leitura, conforme abordagem semelhante proposta em [PC99] e provada através de teorema (“*O método de .difusão de dados por multiversão produz transações de somente leitura corretas”*).

4.5.2 Arquitetura do SGBDD móvel para execução de transações

Os componentes do sistema que participam do gerenciamento de transações em um banco de dados distribuído tradicional são:

- *Gerenciador de Transações*
- *Scheduler*
- *Data Processor*
- *Diretório Global*

A forma tradicional de interação entre estes componentes pode ser visualizada na Figura 4.14.

Num SGBDD Móvel, esta arquitetura somente pode ser utilizada se houver a *replicação completa da estrutura hierárquica em cada nó*. Neste modelo, substitui-se o item diretório global pelo gerenciador de localização, responsável em auxiliar o *Gerenciador de Transações* em acessar os *Schedulers* e os *Data Processors* dos vários SGBDs participantes do sistema

Entretanto, este modelo apresenta a desvantagem, como exposto na seção 4.4, da obrigatoriedade da difusão da configuração da rede para todos os gerenciadores de localização, aumentando o custo de manutenção dos dados que representam a estrutura hierárquica nos gerenciadores de localização.

A arquitetura com os mesmos componentes, porém utilizando o modelo de *replicação parcial da estrutura hierárquica com armazenagem de ponteiros* está ilustrada na Figura 4.15.

.

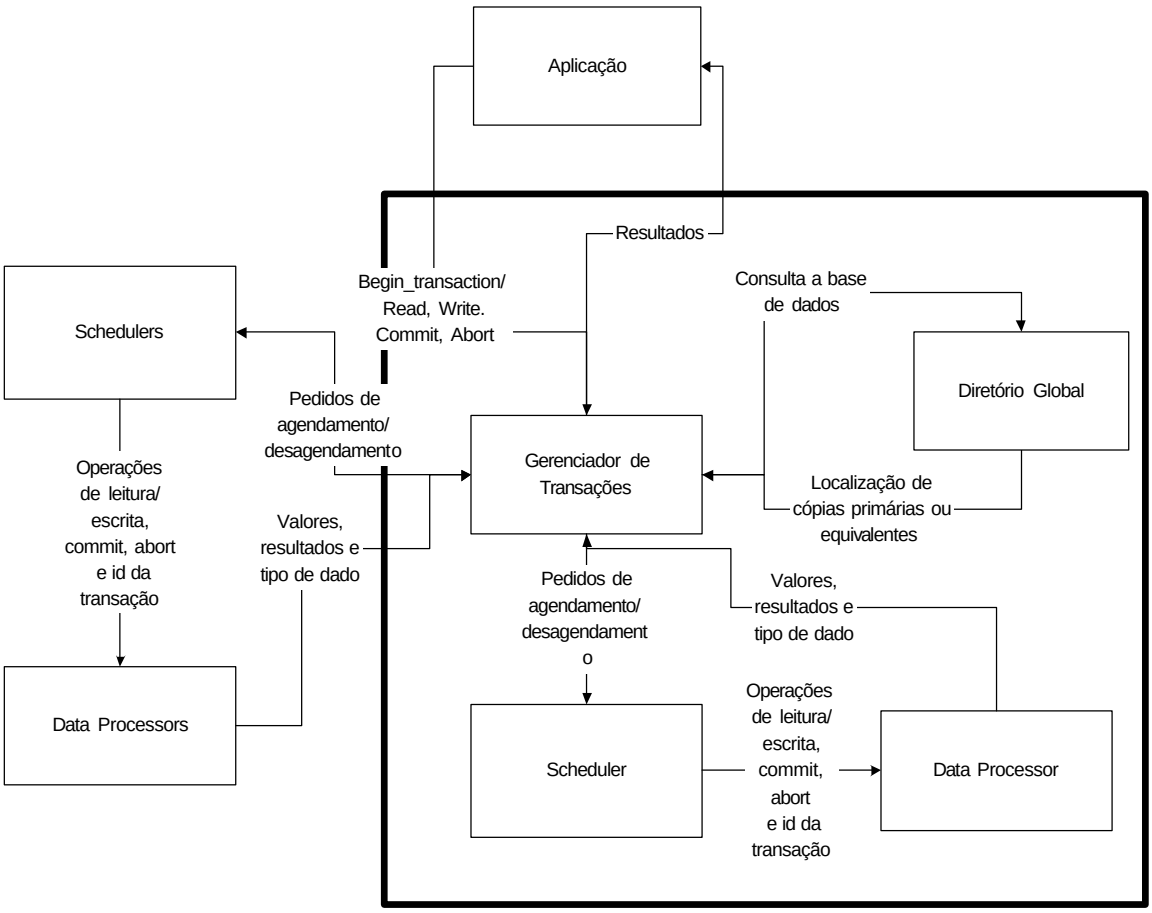

Figura 4.14 – Arquitetura de um modelo de execução distribuída tradicional e do SGBDD Móvel com replicação completa da estrutura hierárquica

Neste modelo, o gerenciador de localização do SGBD local não possui informações dos outros bancos de dados, devendo solicitar a colaboração dos demais gerenciadores de localização para que possa acessar os *Schedulers* e *Data Processors* remotos.

O sistema modifica-se sutilmente ser for utilizado a arquitetura de replicação parcial da estrutura hierárquica com armazenagem da estrutura (Figura 4.16).

Neste modelo, algumas estações não têm o conhecimento da estrutura hierárquica, devendo operar segundo o modelo de armazenagem de ponteiros. Outras estações

conhecem a estrutura hierárquica, podendo acessar diretamente os componentes remotos. A forma de retorno (direta ou indireta) dos valores, resultados e tipos de dados é dependente também do modelo de estrutura hierárquica adotada.

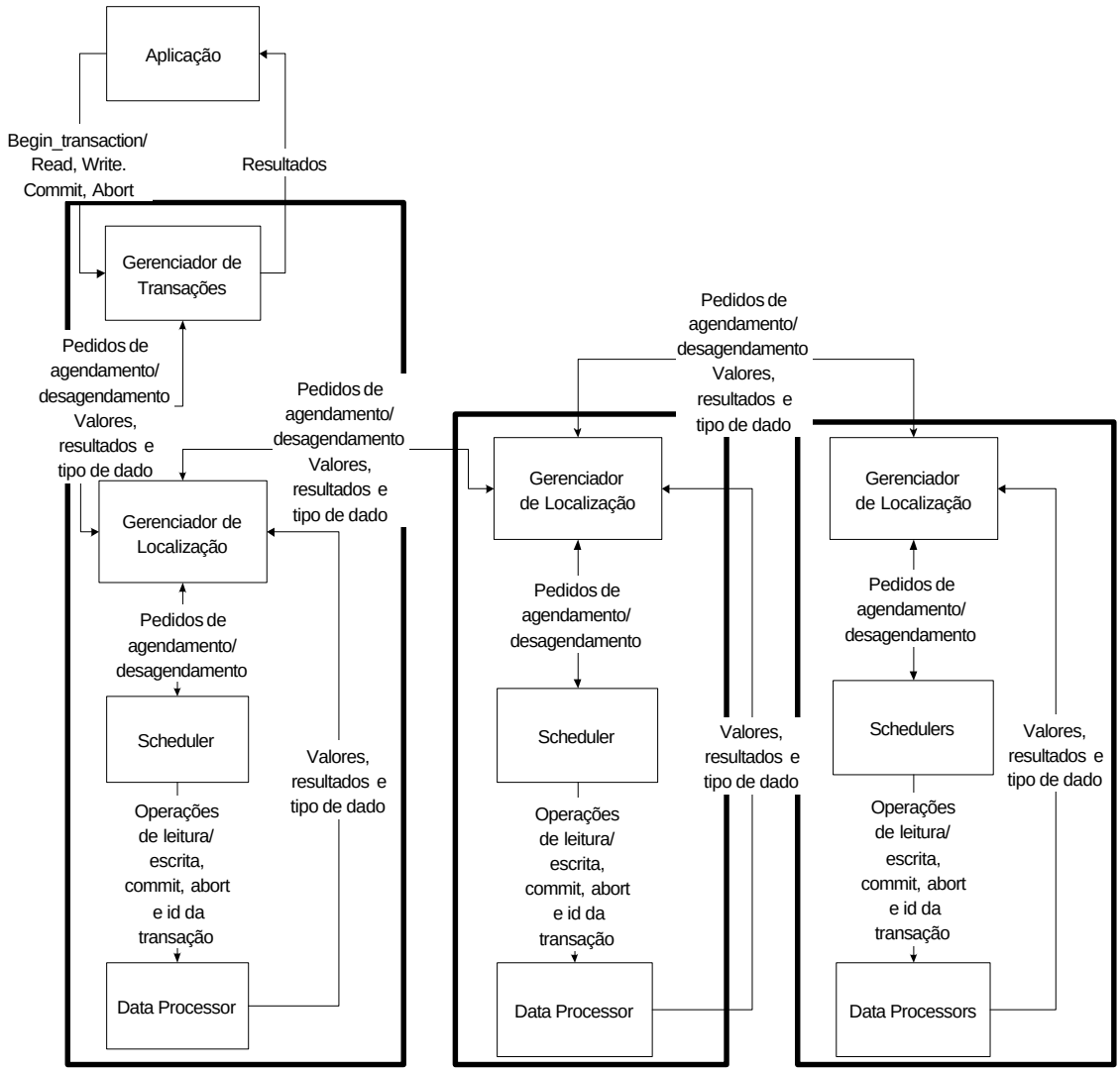

Figura 4.15 - Arquitetura de um SGBD Móvel Hierárquico com armazenagem de ponteiros

Por seu aspecto modular e recursivo, a arquitetura com armazenagem de ponteiros parece um modelo mais fácil de ser implementado que os demais. Porém, não parece ser, a primeira vista, o mais eficiente, em virtude de seu *overhead*.

4.5.3 Transações que possuem apenas operações de leitura

As operações de leitura podem ser de dois tipos:

- Operações que interferem no esquema de multiversões
- Operações que não interferem no esquema de multiversões (*operações simples*)

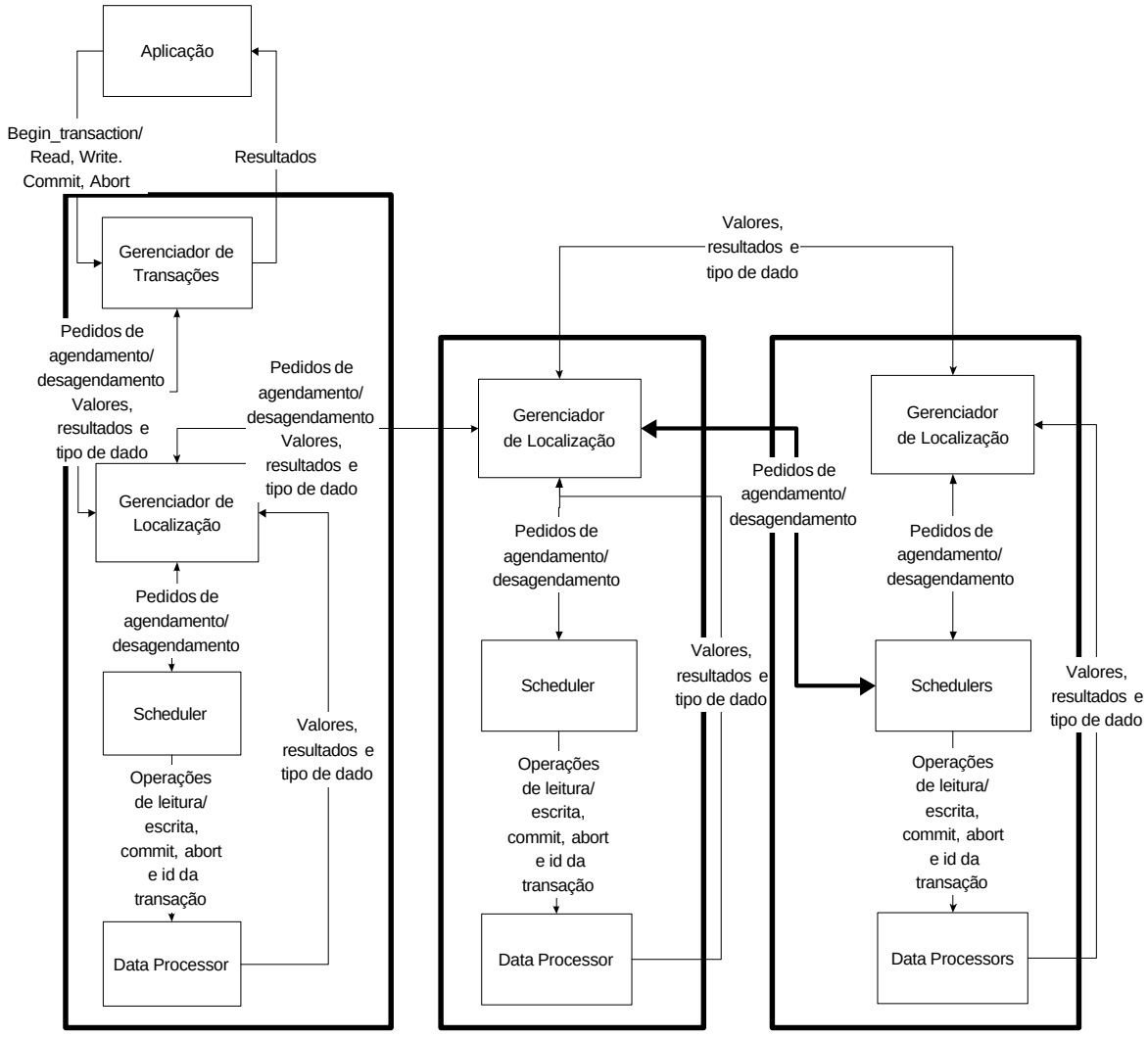

Figura 4.16 - Arquitetura de um SGBD Móvel Hierárquico com armazenagem de estruturas

As operações que interferem no esquema de multiversões são aquelas que afetam o algoritmo de controle de concorrência. São operações duráveis. É o equivalente a uma certidão. São processadas de maneira equivalente a uma transação de primeira classe.

As operações que não interferem no esquema de multiversões (*operações simples*) são puramente informativas.

Segundo os modelos de dados expostos na seção 4.1, as operações de leitura podem retornar vários tipos de dados. São eles:

- *Cópias primárias e secundárias com prazo de validade*
- *Versões mestres de cópias primárias*
- *Versões tentativas de cópias primárias*
- *Versões mestres de cópias secundárias sem prazo de validade*
- *Versões tentativas de cópias secundárias sem prazo de validade*

O *Data Processor* é o responsável por retornar o valor do dado ao *gerenciador de transações*, identificando também o *tipo de dado retornado*, a *estação* que retornou o dado e se o *dado é válido ou não*.

Este fluxo de informação entre o *Data Processor* e o *gerenciador de transações* pode ser direto ou indireto, dependendo do modelo de estrutura hierárquica utilizado no gerenciador de localização, conforme foi verificado na seção 4.5.2.

O *tipo de dado retornado* traz a informação adicional de saber se aquele dado está em consistência lógica ou não.

Dentre os tipos de dados retornados, apenas as *versões mestres de cópias primárias* e as *cópias secundárias com prazo de validade* retornarão dados que estão, seguramente, em consistência lógica com o sistema.

4.5.3.1 Atividade do *gerenciador de localização* nas operações de leitura

Replicação completa da estrutura hierárquica em cada nó

O *gerenciador de transações* consulta o *gerenciador de localização* a respeito do sentido do dado que deseja consultar, bem como das informações necessárias (estações a percorrer, vínculos de subordinação, etc.) para executar *internamente* (no gerenciador de transação), o algoritmo de busca de réplicas (ascendente ou descendente), conforme o sentido de movimentação do dado desejado.

A partir destas informações recebidas do *gerenciador de localização*, o *gerenciador de transações* acessa diretamente os *Schedulers* (local ou remoto) para a execução das operações.

O *gerenciador de localização* tem papel simplesmente informativo neste tipo de configuração, limitando-se a informar o *gerenciador de transações* sobre configuração da rede, localização de *cópias primárias*, etc.

O *Scheduler*, por sua vez, encaminha a operação (e outras informações necessárias, como o *id* da transação, que possui o endereço da estação solicitante) desejada para o *Data Processor*, que executa a operação, transferindo o resultado diretamente para o *gerenciador de transações* .

Replicação parcial da estrutura hierárquica com armazenamento de ponteiros

Neste modelo, o gerenciador de localização não tem conhecimento da configuração da rede, sendo necessário interagir com outros *gerenciadores de localização* para poder executar o algoritmo de busca de réplicas (ascendente ou descendente).

Os algoritmos de busca de réplicas são executados distribuidamente no *gerenciador de localização de cada nó*.

Por exemplo, o nó D deseja realizar uma operação de leitura do dado 2, segundo a configuração da Figura 4.17.

O *gerenciador de transações* do nó D envia um pedido de agendamento da operação de leitura do dado 2 para o *gerenciador de localização* local.

O *gerenciador de localização* então verifica que o dado 2 tem sua *cópia primária* no nó A e a origem desta cópia é descendente. Aplicar-se-á, portanto, o algoritmo de busca de réplicas descendentes.

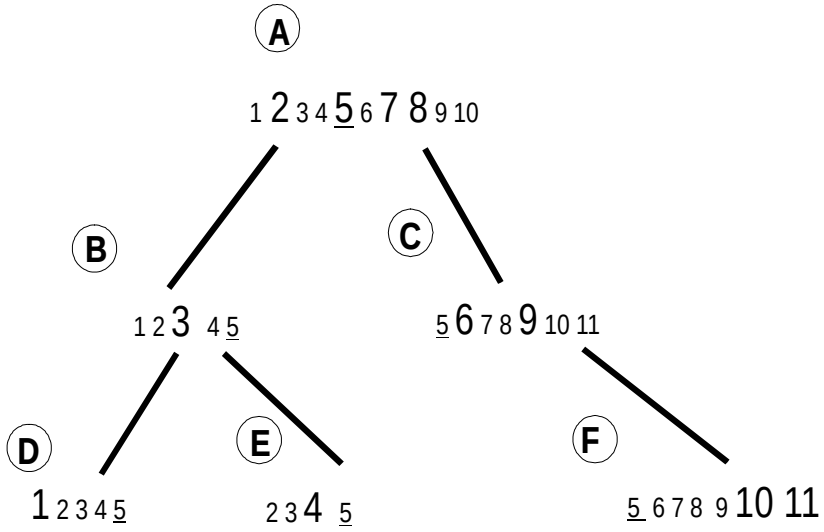

Figura 4.17 – Possível configuração do sistema

De posse desta informação, ele encaminha este pedido de agendamento desta operação, identificada univocamente, para o *gerenciador de localização* do nó B e inicia uma operação de *timeout*.

O *gerenciador de localização* do nó B repete o procedimento do *gerenciador de localização* do nó D e verifica que a origem da cópia é descendente, encaminha, por sua vez, a operação para o nó A, iniciando também uma operação de *timeout*.

O *gerenciador de localização* do nó A recebe o pedido do *gerenciador de localização* do nó B, e consultando a sua base, verifica que possui a cópia primária do dado 2. Envia o pedido para o *Scheduler* local, enquanto inicia uma operação de *timeout* referente à operação do *Scheduler*. O *Scheduler*, por sua vez, encaminha a operação de leitura para o *Data Processor*. Este componente, então, retorna o resultado para o *gerenciador de localização* do nó A.

O *gerenciador de localização* do nó A verifica o número da transação associada à operação e encaminha o resultado a quem a solicitou, no caso o *gerenciador de localização* do nó D. Para acessar este gerenciador, deve encaminhar o resultado para o *gerenciador de localização* do nó B.

O *gerenciador de localização* do nó B recebe o resultado da operação do *gerenciador de localização* do nó A, suspende o seu *timeout* associado à operação, e envia o resultado para o *gerenciador de localização* que lhe havia solicitado execução da operação, no caso o *gerenciador de localização* do nó D.

O *gerenciador de localização* do nó D recebe o resultado da operação do *gerenciador de localização* do nó B, suspende o seu *timeout* associado à operação, e envia o resultado para o *gerenciador de transações* do nó D.

No caso de ter ocorrido *timeout* em alguma das estações, estas realizariam um pedido de agendamento no *Scheduler* local, que encaminharia o pedido para o *Data Processor*, que, por sua vez, retornaria o resultado para este *gerenciador de localização*, retornando para o *gerenciador de transações* que originou a transação um valor correspondente a uma *cópia secundária*.

Replicação parcial da estrutura hierárquica com armazenagem de estrutura

O procedimento segue os passos da replicação com armazenagem de ponteiros até alcançar um nó que possua a armazenagem da estrutura, executando a partir deste nó o algoritmo de replicação completa da estrutura hierárquica.

4.5.3.2 Atividade do *scheduler* e do *data processor* nas operações de leitura

É necessária a identificação do tipo de operação de leitura que se está realizando.

No caso das *operações que não interferem no esquema de multiversões*, não existe o controle de concorrência e a informação é simplesmente enviada pelo *Scheduler* ao *Data Processor*, que retorna o valor para o Gerenciador de Transações.

O *Data Processor* armazena tanto as versões mestre quanto as versões tentativas. No processamento de consultas oriundas de estações remotas, o *Data Processor* deve retornar o valor da versão mestre.

No caso de *operações que interferem no esquema de multiversões*, ou seja, as equivalentes a transações de 1ª ou 2ª classe, este segue o processamento da implementação tradicional do esquema de multiversões específico para cada tipo de transação. A implementação do *Scheduler* para estes tipos de operações está descrita na seção 4.5.4.2.

Quando o *Scheduler* atua sobre cópia secundárias, o mesmo não necessita executar o mecanismo de controle de concorrência, exceção feita às *operações locais*, oriundas de transações de 2ª classe (4.5.5).

4.5.4 Transações de primeira classe

Esta seção descreve transações que possuem pelo menos uma operação de atualização (*W* – escrita), ou quando o usuário deseja registrar suas operações de leitura no banco de dados.

Este tipo de transação é o caso “normal”, ou seja, o sistema está operando com suas estações conectadas. Deve operar sobre dados confiáveis, ou seja, *versões mestres de cópias primárias* ou *cópias primárias ou secundárias com prazo de validade*.

O protocolo de terminação a se aplicar neste caso, como afirmado anteriormente, é o *Presumed Abort Two-Phase Commit*. Opta-se por este protocolo em virtude da fraca conectividade da rede. Para aumentar a eficiência e confiabilidade do sistema, poder-se-ia deslocar a coordenação do processo de commit para o nó participante de maior hierarquia. Isto tem conseqüências quanto ao trâmite de mensagens e controle do processo, que não serão abordados nesta dissertação.

Observa-se que não é possível projetar protocolos de terminação não bloqueantes quando existe a falha de mais de um site [SS83].

O algoritmo do *gerenciador de transações*, adaptação do algoritmo de pré-ordenação existente em [OV99], com replicação completa da estrutura hierárquica é descrito na próxima seção.

O algoritmo do gerenciador de transações de 1ª classe encontra-se no apêndice desta seção.

O *gerenciador de transações* no modelo com replicação parcial da estrutura hierárquica com armazenagem de ponteiros é sutilmente modificado. Neste modelo, ele não tem acesso direto aos *Schedulers*, devendo encaminhar a transação para o *gerenciador de localização*. A execução de uma transação de 1ª classe somente pode ser executada sobre dados que estejam em consistência lógica, obrigando neste tipo de transação a atuar somente sobre determinados sites, não aceitando valores de cópias secundárias.

O controle dos sites participantes da transação continua o mesmo, sem modificações, ou seja, o gerenciador de localização possui a informação, *a priori*, dos sites participantes da transação de 1ª classe.

A única tarefa do gerenciador de localização nas transações de 1ª classe, neste caso, é de roteamento da solicitação da operação e resultado da mesma.

Nas operações de leitura não integrantes de uma transação de 1ª classe, ele possui um papel mais ativo, possibilitando retornar o valor existente no *Data Processor* local, no caso de indisponibilidade dos SGBDs que, em tese, possuiriam valores mais confiáveis.

No caso das transações de 1ª classe, o *gerenciador de localização* não pode retornar este valor, a não ser que seja uma cópia que esteja em consistência lógica com o sistema.

O *Scheduler* é o responsável pelo controle de concorrência aos dados do *Data Processor*. Seu algoritmo no controle de multiversões, adaptado de [OV99], está descrito a seguir.

Nas operações oriundas das transações de 1ª classe, além de executar o procedimento referente a estas operações, deve ainda realizar o controle das versões tentativas oriundas de transações de 2ª classe.

4.5.5 Transações de segunda classe

Transações de 2ª classe são aquelas que ocorrem quando não é possível acessar todos os sites participantes de uma transação de 1ª classe.

No modelo proposto por [GHOS96], as transações de 2ª classe seriam executadas quando as estações móveis estivessem desconectadas das estações base. O modelo desta dissertação, além desta hipótese, considera que se pode executar transações de 2ª classe quando não se pode acessar todos os sites de uma transação de 1ª classe.

Um usuário deve decidir se vai realizar uma transação de 1ª classe ou de 2ª classe. Para decidir que tipo de transação vai executar, um dos parâmetros a ser levado em consideração são as conexões ativas do sistema.

O usuário pode, por exemplo, tentar executar uma transação de 1ª classe e obter o erro de falha na conexão. Diante deste erro, optar por realizar uma transação de 2ª classe.

A falha de execução de uma transação de 1ª classe não significa que será transformada em uma transação de 2ª classe. Esta deve ser escolhida expressamente.

4.5.5.1 Coexistência de versões mestres e versões tentativas

Uma transação de 2ª classe é processada integralmente no SGBD local, operando sobre versões tentativas ou versões mestres.

Caso execute uma operação de escrita integrante de uma transação de 2ª classe, esta, se estiver dentro dos critérios de aceitação, irá gerar uma nova versão tentativa.

Caso execute uma operação de leitura integrante de uma transação de 2ª classe e a atualização anterior mais próxima foi resultado de uma transação de 2ª classe, lerá esta versão, caso contrário lerá a versão mestre. A Figura 4.18 ilustra possíveis cenários.

Transações de 1ª classe têm precedência sobre transações de 2ª classe. Isto significa que se o usuário decidir realizar uma transação de 1ª classe, ele pode cancelar automaticamente uma ou mais transações de 2ª classe que estavam aguardando o processo de certificação.

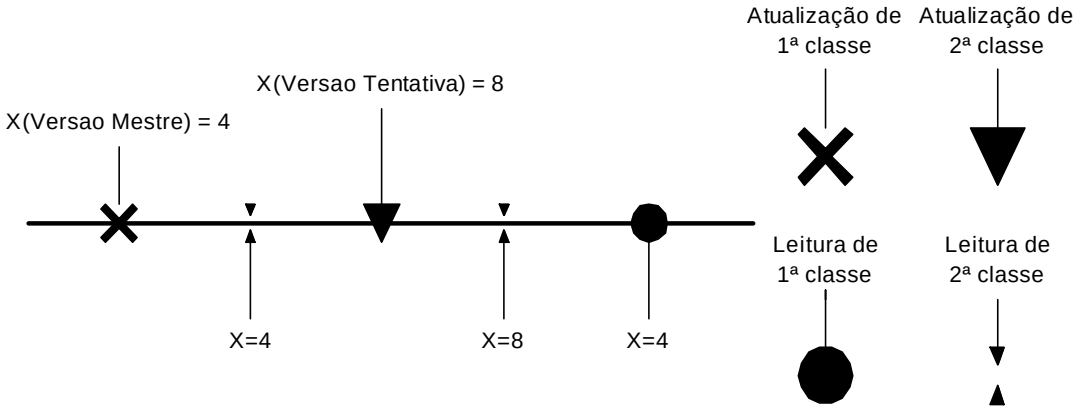

Figura 4.18 – Cenário possível entre transações de 1ª e 2ª classe

A Figura 4.19 ilustra um exemplo de um cenário inicial contendo versões mestres e tentativas.

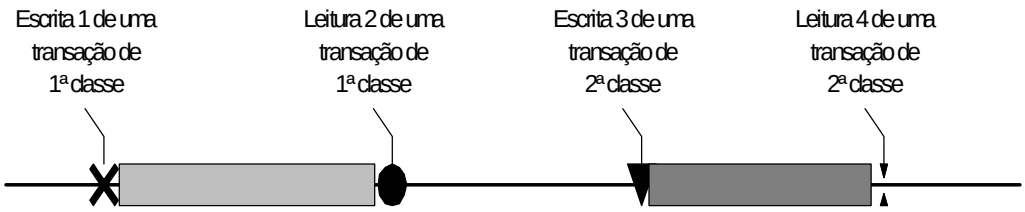

Figura 4.19 - Versões tentativas e mestres de um dado

Suponhamos que uma transação de 1ª classe, digamos W(escrita)5, escreva no intervalo de tempo entre W3 e R4, ilustrado na Figura 4.20.

Esta operação resultaria no cancelamento da leitura 4 da transação de 2ª classe, deixando porém a escrita 3 ainda pendente do processo de certificação.

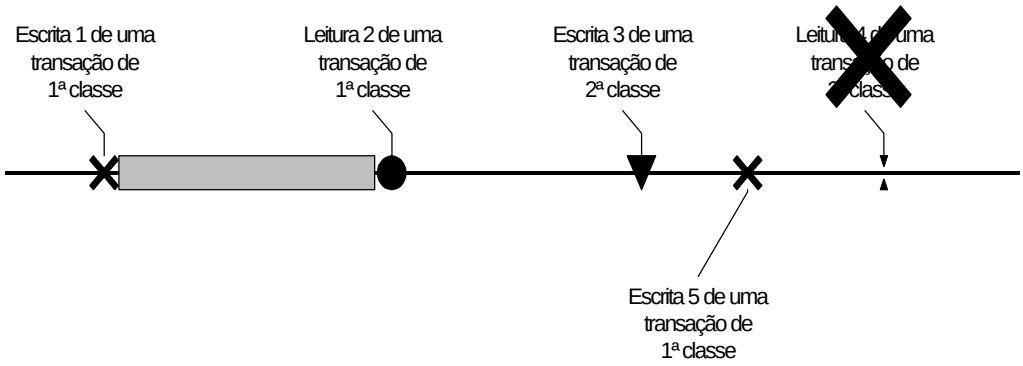

Figura 4.20 - Cancelamento de uma operação de leitura de 2ª classe

Porém, se fosse uma operação de leitura de uma transação de 1ª classe, resultaria no cancelamento da escrita 3 e da leitura 4, pois a leitura 4 é dependente da escrita 3. Este cenário está descrito na Figura 4.21.

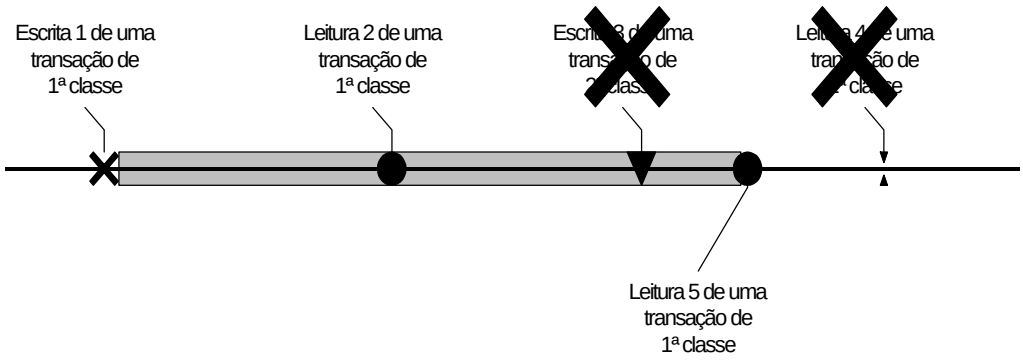

Figura 4.21 - Cancelamento da operação de escrita de 2ª classe

As leituras oriundas de transações de 2ª classe, cuja origem do resultado é uma transação de 1ª classe sofrem tratamento análogo. Na Figura 4.22, a leitura 3 de uma transação de 2ª classe lê o valor da escrita 1.

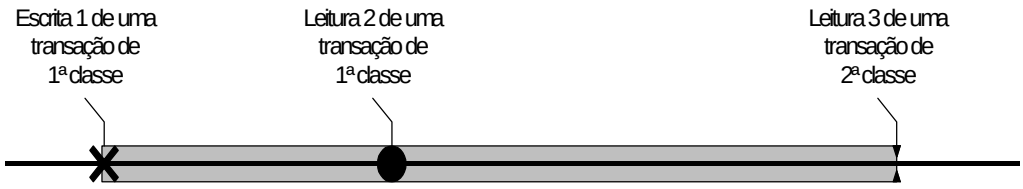

Figura 4.22 - Leitura de 2ª classe sobre um dado de 1ª classe

Caso ocorra uma escrita de 1ª classe neste intervalo entre a escrita 1 e a leitura 3, isto cancelará a leitura 3. O exemplo está na Figura 4.23.

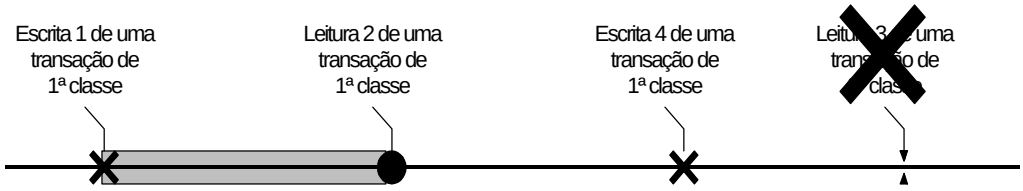

Figura 4.23 - Cancelamento de uma leitura de 2ª classe

Desta forma, como no controle de multiversão existente nas transações de 1ª classe, visualiza-se uma relação entre as leituras realizadas com as atualizações das versões tentativas existentes no banco de dados.

Quando for feita a atualização ou leitura de uma operação participante de uma transação de 1ª classe, deve-se checar as atualizações e leituras das versões tentativas existentes e excluí-las do processo de certificação, caso sejam incompatíveis com as primeiras.

O algoritmo do *Scheduler*, portanto, deve atentar não somente para as *operações de 1ª classe* existentes no sistema. Deve realizar a preparação das *operações participantes de transações de 2ª classe* e ainda auxiliar no *processo de certificação das transações de 2ª classe*.

No *processo de certificação das transações de 2ª classe* da estação local deve atentar, ainda, que ele está promovendo o status da versão tentativa para versão mestre, seja na cópia primária ou secundária existente na base de dados local, devendo, portanto, não cancelar as operações subseqüentes, em virtude desta transformação.

Diante do exposto, verifica-se que transações de 1ª classe e transações de 2ª classe podem ser executadas concorrentemente em uma estação. O *Scheduler* é o componente responsável pela concorrência de acesso entre as *transações de 1ª classe* e as *transações de 2ª classe* sobre as cópias primárias e secundárias existentes na base de dados local. O *Scheduler* é o responsável, ainda, pelo processo de certificação das transações de 2a classe.

4.5.5.2 Processo de certificação e serialização das transações de 2ª classe

As transações de 2ª classe, localmente, são *serializáveis* entre si. Mais do que isso, elas geram dependência, onde a falha de uma transação acarreta na falha de outra que lhe é superveniente.

Suponha que tenhamos as seguintes transações [BHG88]:

$$T_0 = w_0[x]\ w_0[y]\ w_0[z]\ c_0$$

$$T_1 = r_1[x]\ w_1[y]\ c_1$$

$$T_2 = r_2[x]\ r_2[z]\ w_2[x]\ c_2$$

$$T_3 = w_3[z]\ w_3[y]\ w_3[z]\ c_3$$

$$T_4 = r_4[x]\ r_4[y]\ r_4[z]\ c_4$$

Onde as transações foram executadas na ordem de seus índices na estação local.

A dependência ocorre quando uma operação de escrita for superveniente a uma operação de escrita de uma transação de 2ª classe. Este é o caso das transações T_1 e T_2 em relação a T_0 e de T_4 em relação a T_3.

Tome, por exemplo, as transações T_0 e T_3. Caso se utilizasse o esquema de pré-ordenação, a transação T_3 seria dependente do resultado de T_0. Um erro no processo de certificação de T_0 acarretaria no cancelamento das transações T_3 e T_4.

Em virtude do protocolo de concorrência de multiversões, tem-se, na Figura 4.24, o seguinte grafo de serialização:

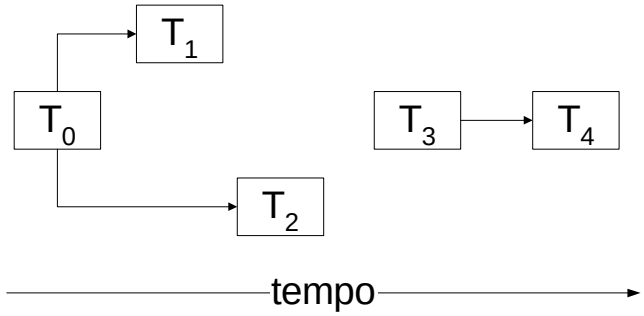

Figura 4.24 - Grafo de serialização

Assim, as transações T_1 e T_2 são dependentes do resultado de T_0, assim como T_4 é dependente de T_3. T_0 e T_3 são independentes entre si.

Posteriormente, na reintegração ao sistema, as transações de 2ª classe serão submetidas a um processo de certificação.

Neste processo de certificação, semelhantemente as transações de 1ª classe, elas são enviadas para as estações detentoras de cópias primárias, ou cópias (primárias ou

secundárias) com prazo de validade e seus resultados enviados de volta para a estação que gerou a transação de 2ª classe.

O resultado de certificação positiva para as operações de leitura, é quando o *writeset* (versão do dado) da cópia secundária coincide com o *writeset* da cópia primária[PC99], [Bar97]. Isto faz com que os dados sejam idênticos e a consistência garantida.

O resultado de certificação positiva para as operações de escrita é quando o resultado é a própria autorização para a operação de escrita solicitada.

Além destas operações, deve-se realizar outras duas tarefas.

A primeira é verificar se os nós que participam do processo de certificação da transação de 2ª classe estão conectados ao nó que está gerenciando a transação de 2ª classe.

A segunda é manter uma estrutura de dados que permita o controle de dependência das transações.

Na seção anterior foi abordado o assunto de que operações oriundas de transações de 1ª classe podem cancelar as transações de 2ª classe.

Em função deste cancelamento, deve-se checar o grafo de serialização, pois o cancelamento proveniente de um dado pode afetar uma transação que não operava sobre aquele dado, mas era dependente da transação que foi cancelada.

A execução do processo de certificação de transações de 2ª classe pode ser concorrente. No exemplo desta seção, T_0 e T_3 não são dependentes entre si. Isto possibilita que sejam executados simultaneamente.

4.5.5.3 Algoritmo do gerenciador de transações de 2ª classe

O procedimento do gerenciador de transações de 2ª classe, além de executar as transações de 2ª classe, realiza o procedimento de certificação das mesmas, quando da reintegração do sistema.

Para tanto, deve verificar, inicialmente, quais transações de 2ª classe são independentes das demais. Após este passo, deve verificar quais as estações que participam destas transações, devendo monitorar se as mesmas estão conectadas com a estação local.

Paralelamente, as outras estações irão atualizar as cópias secundárias da estação local, cancelando várias transações de 2ª classe. Estas transações canceladas cancelarão outras transações de 2ª classe, que lhe eram dependentes.

As transações de 2ª classe, à medida que vão sendo certificadas, atualizam os grafos de dependência, gerando novas transações independentes. No exemplo da seção anterior, depois que T_0 é certificada, T_1 e T_2 podem realizar o seu processo de certificação independentemente. O gerenciador de transações de 2ª classe passa, então, a monitorar as estações que participam destas novas transações independentes.

4.6 Gerência de replicação

4.6.1 Modelo de difusão

O gerenciador de replicação é o responsável pela convergência do sistema de gerenciamento de dados.

Conforme o exposto na seção 3.3.1 existem três modelos de difusão de dados: *push-based*; *pull-based* e híbrido. O modelo mais apropriado para o problema em questão é

obviamente o híbrido, resultado da política de replicação dos dados e dos dados obtidos por demanda.

O módulo de replicação é o responsável pelo balanceamento dos dados *pull* e *pushed*. Isto, porém, sendo um problema de otimização, não será abordado nesta dissertação.

4.6.2 Modelo de dados em função da freqüência de replicação

Um parâmetro importante na replicação de dados é a freqüência de replicação de dados.

Dividem-se os dados em dois grupos em razão da freqüência de replicação:

- *Dados com prazo de validade*
- *Dados sem prazo de validade*

Dados com prazo de validade utilizam a técnica de *quasi-caching* [ABMG90]. Sua periodicidade de difusão está relacionada diretamente com este prazo, sendo o processo de difusão disparado pela estação detentora da cópia primária. Um exemplo deste tipo de dado seria as rotas de deslocamento de viaturas. Durante o prazo de validade, o valor do dado não pode ser modificado pela cópia primária, sob pena de inconsistência da base de dados.

Dados sem prazo de validade são difundidos periodicamente em virtude da importância que possuem para o sistema (ex: localização de unidades, situação logística, etc.). A periodicidade desta difusão pode ser definida antecipadamente pelo administrador ou usuário do sistema ou ainda ser resultado da demanda de consulta e/ou atualização dos dados. Esta demanda é calculada através de um componente do gerenciador de replicação, o *gerenciador de estatística*, que, de acordo com a freqüência de acesso ao dado e capacidade de comunicação na direção de transmissão

(*uplink* ou *downlink*), irá modificar a taxa de periodicidade de difusão do dado, atualizando a *tabela de replicação*.

A freqüência de difusão como resultado de consultas do sistema pelo usuário, pode ser obtido através das operações realizadas no *Scheduler*, que informa ao *gerenciador de estatística* sobre cada operação de acesso àbase de dados do sistema.

4.6.3 Organização da difusão de dados

O componente do gerenciador de replicação que organiza a difusão dados é o *despachante de replicação*. Sua implementação é semelhante aos esquemas de distribuição de tarefas em sistemas operacionais, onde a periodicidade de difusão está associada àprioridade da tarefa.

Este modelo é dinâmico, podendo variar em função da demanda dos clientes por determinados dados, que alteram a tabela de prioridades de difusão de dados.

Assim, o conteúdo da difusão modifica-se dinamicamente, incluindo novos itens e removendo outros existentes. A organização dos dados da difusão pode ser modificada também ao que se refere às freqüências de difusão.

A prioridade da tarefa é determinada pelo *gerenciador de estatística*, que modifica dinamicamente a prioridade e também pelo usuário também pode determinar a freqüência de difusão.

Estes fatores estão, obviamente, relacionados com a capacidade de carga do sistema. Deve-se seguir, sempre que possível, o tempo estipulado para cada item, porém, isto será limitado em função da capacidade de comunicações do sistema.

Os dados a serem transmitidos são apenas os duráveis. Isto significa que apenas dados confirmados serão transmitidos.

As *mensagens de atualização das cópias secundárias* (4.2.2) possuem as seguintes informações:

- Identificação do dado
- Valor do dado
- Timestamp (*writeset*) associado àúltima versão do dado
- Timestamp (*readset*) da ultima leitura realizada no dado

Estes dados irão influenciar no processo de certificação de transações de 2ª classe ainda pendentes, conforme o exposto na seção anterior.

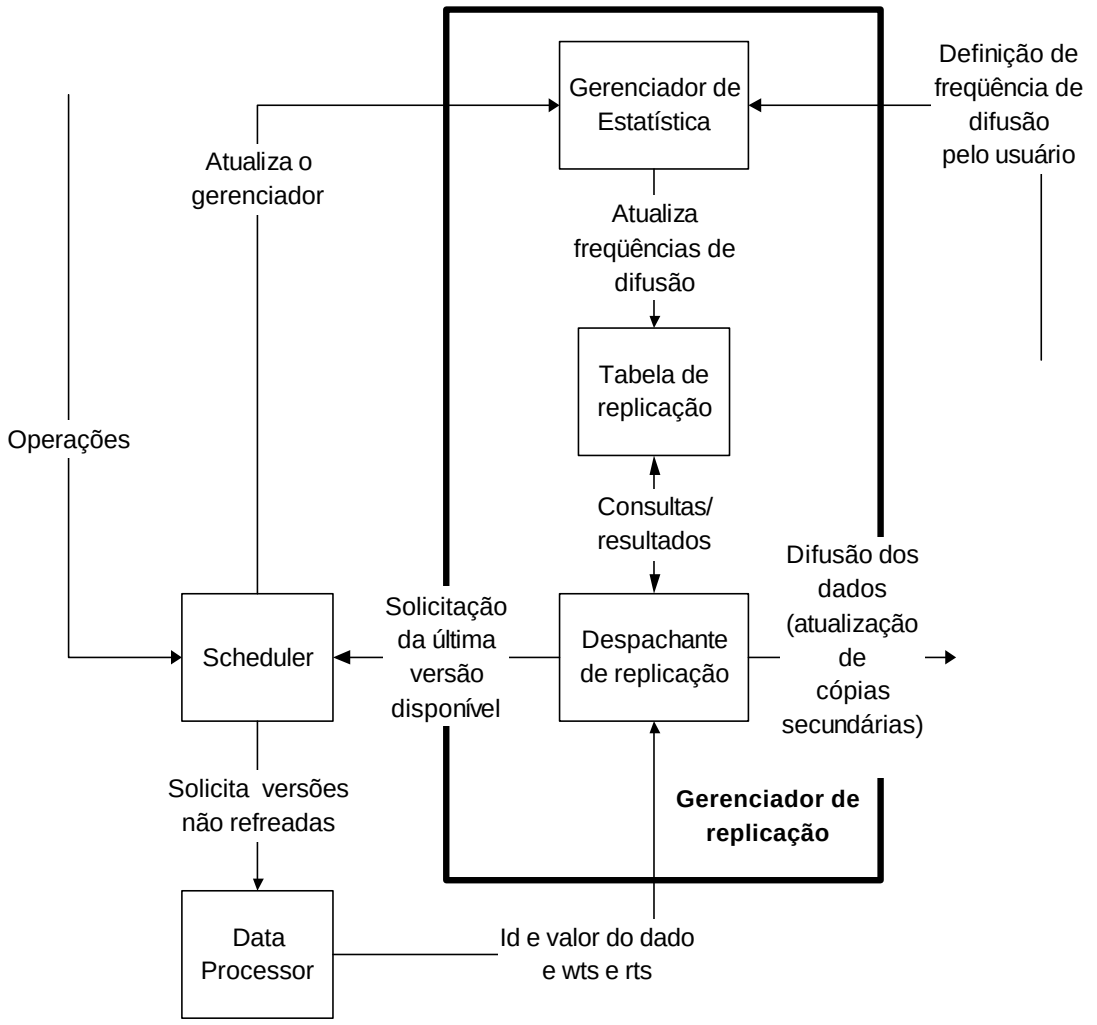

Figura 4.25 - Despachante de replicação e gerenciador de estatística

O *despachante de replicação* envia os dados baseados nos dados existentes no *Data Processor* local. Quando da proximidade de enviar determinado dado, o *despachante*

de replicação envia uma mensagem para o *Scheduler* solicitando a ultima versão do dado.

Caso a operação tenha que ser retida em virtude de uma operação de escrita, o *Scheduler* solicita a versão disponível mais atualizada, que não necessite ser suspensa, enviando estes dados para o *Data Processor*, que repassará ao despachante de replicação.

4.6.4 Caching, broadcast, consistência e invalidação de réplicas

4.6.4.1 Caching, broadcast, consistência

Tradicionalmente, a consistência é baseada na serialização, através da execução do *log* das transações, assegurando que as operações serão executadas no contexto de *A*tomicidade, *C*onsistência, *I*solamento e transações *D*uráveis (ACID).

A consistência, neste modelo, é garantida apenas em determinados tipos de dados (cópias primárias e cópias com prazo de validade), não sendo necessário, portanto a execução dos *log* das transações. Adota-se o modelo da replicação individual de cada dado.

Exemplificando, suponha que uma transação na estação A execute a seguinte transação T_1: $w_1(x)$, $w_1(y)$.

A difusão da atualização de x e y não será realizada através da execução do *log* de T_1 nas estações onde existam réplicas de x_1 e x_2 e sim através de mensagens que atualizam x e y separadamente, ocasionado inconsistências temporárias nas estações remotas. Isto é mais evidente quando x possui uma freqüência de atualização maior do que y.

Um outro fator que recomenda este tipo de modelo de replicação é o fato da não obrigatoriedade da existência dos dados *x* e *y*, conjuntamente, em todos os bancos de dados.

O modelo de replicação exposto na seção 4.3 (*replicação em duas camadas*), pretende que o banco de dados alcance a sua consistência através da convergência das operações de replicação.

As atualizações são recebidas por um componente do Gerenciador de Replicação, o receptor de replicação, que as envia para o *Scheduler*. Este atualiza as cópias secundárias (versões mestres) dos dados, cancelando, por ventura, transações de 2ª classe ainda pendentes.

Estas atualizações não concorrem, no *Scheduler*, com operações participantes de transações de 1ª classe, visto que estas últimas operam sobre cópias primárias. Estas operações também não são computadas no *gerenciador de estatística*.

No exemplo anterior, quando a estação A enviar tanto a atualização de *x* como de *y*, o banco encontrará a convergência. Outro padrão, já exposto nesta seção, que procura assegurar a consistência dos bancos de dados é o uso de *quasi-caching*, que são dados que possuem prazos de validade.

Porém, existem outros métodos que buscam alcançar a consistência do banco de dados, como, por exemplo, os relatórios de invalidação de réplicas.

4.6.4.2 Relatórios de invalidação de réplicas

Nem sempre será possível enviar a modificação de dados para as suas réplicas. Um cenário possível para esta hipótese é quando o tamanho dos dados a serem transmitidos for incompatível com a capacidade de transmissão dos meios de comunicações, sendo, porém, importante notificar as outras estações do sistema da invalidação de suas réplicas.

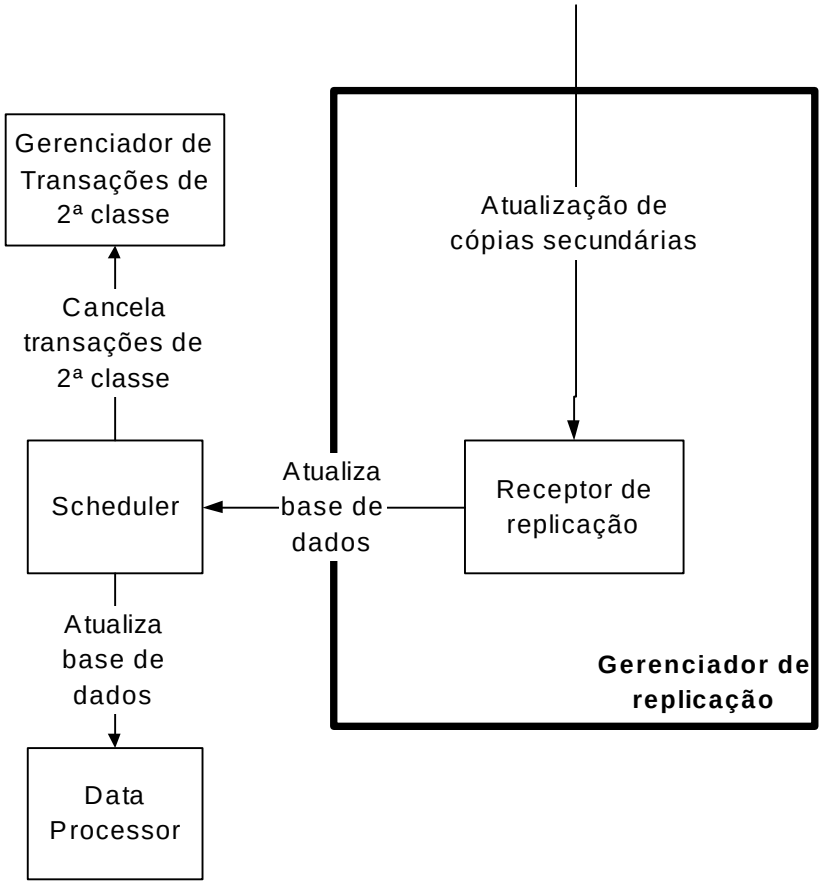

Figura 4.26 - Atualização de cópias secundárias

Um componente do gerenciador de replicação, *o gerenciador de invalidação de cache*, trabalha em paralelo com a difusão de dados, notificando os SGBDs remotos da invalidação de suas réplicas.

Esta informação pode ser útil, por exemplo, para evitar a execução de transações de 2ª classe pendentes baseadas em leituras de versões mestres já existentes, não impedindo, entretanto, tentativas de escritas. A Figura 4.27 ilustra um exemplo desta situação.

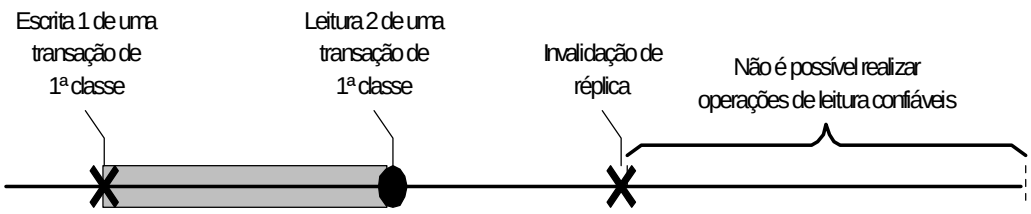

Figura 4.27 - Invalidação de réplicas

Os relatórios de invalidação de cache (4.2.2) contém as seguintes informações:

- Identificação do dado;
- Timestamp da invalidação do dado

Na ausência da data, considera a data invalidação como a data da mensagem.

A invalidação pode ser realizada *sincronamente* ou *assincronamente*.

No método *assíncrono*, o servidor difunde um relatório de invalidação em relação a um determinado item tão logo este modifique o seu valor.

No método *síncrono*, o servidor difunde periodicamente um relatório de invalidação.

Outro componente do gerenciador de replicação, o *Observador de clientes*, é o responsável por verificar o estado dos clientes que, segundo os quais, serão enviados estes relatórios de invalidação de réplicas.

As três estratégias de sincronismo para estes servidores são:

- *Timestamps Strategy* (TS), no qual o relatório de invalidação contém os timestamps das últimas mudanças para os itens que sofreram atualizações nos últimos w segundos.
- *Amnestic Terminals* (AT), onde o servidor envia os identificadores dos itens que sofreram modificações desde o ultimo relatório de invalidação.
- *Assinaturas* (*Signatures*), onde um valor é computado sobre um número de itens, aplicando-se técnicas de compressão de dados similares àquelas utilizadas na comparação de arquivos e *shadow pages*.

De acordo com o perfil dos clientes, um tipo de estratégia é mais adequado.

Para clientes freqüentemente conectados (*workalcoholics*), o método adequado é o AT.

Para cliente freqüentemente desconectados (*sleepers*), quando as taxas de leituras são maiores do que as taxas de atualizações, é adequado que se utilize o TS. Caso contrário o de assinaturas será o mais adequado. As taxas de leituras e atualizações são obtidas através da informação enviada pelo *Scheduler* ao gerenciador de replicação. No caso, as informações podem ser referentes tanto às cópias primárias como em relação às cópias secundárias.

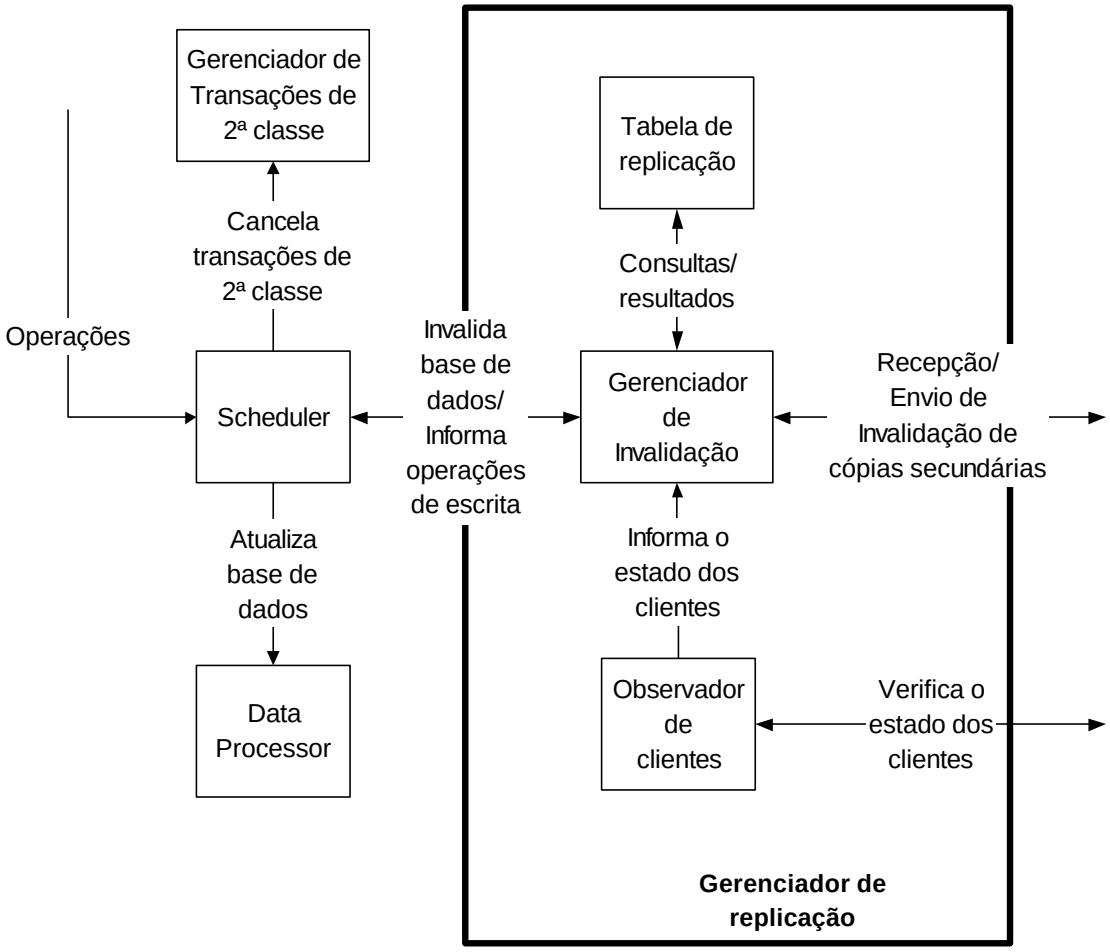

Figura 4.28 - Gerenciador de invalidação

4.6.5 Mapeamento de replicação

Os dados são mapeados para seguir um esquema de replicação. Este mapeamento está definido no *Gerenciador de Localização* e possui determinados padrões, sejam eles de subida, de descida ou ainda para determinadas estações.

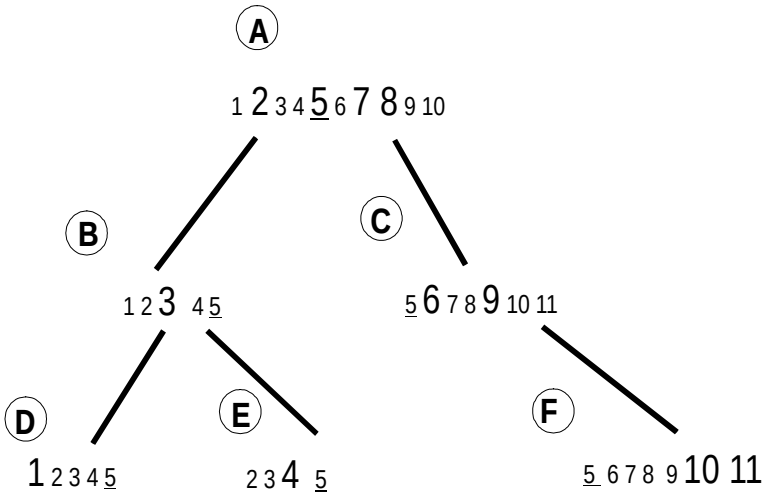

Figura 4.29 - Exemplo de disposição de dados

O mapeamento do gerenciador de localização é individual. O mapeamento do gerenciador de replicação deve levar em consideração os dados e os canais de comunicações (*Uplink* e *Downlink*). Assim, deve-se fazer uma transformação do mapeamento individual existente no gerenciador de localização para um mapeamento por canal existente no gerenciador de replicação.

Na Figura 4.29, tem-se um exemplo de disposição de dados. Considere o mapeamento de replicação do nó B.

A freqüência de replicação de cada um dos dados é definida pelo usuário ou administrador ou ser ainda função da dinâmica de consulta aos mesmos, controlados pelo *gerenciador de estatística*.

Para cumprir o seu papel de replicador, B armazena uma estrutura relacionando os canais de comunicações com os dados a serem difundidos. A Tabela 4.2 ilustra esta relação.

Uplink	1; 3; 4
Downlink	2; 4; 5

Tabela 4.2 - Relação entre os canais de difusão e dados de B

O dado 4 é difundido pelo canal de *downlink* para D como para E. O *Scheduler* da estação D recebe a mensagem de atualização de cópias secundárias (4.2.2). Porém, a estação D é a detentora da cópia primária do dado 1. Sendo assim, o seu *Scheduler* não aceita esta atualização.

Cada um destes dados armazenados possui ainda outras informações, tais como a freqüência de replicação e duração do fluxo.

A freqüência de replicação não será obrigatoriamente idêntica para o mesmo dado. O dado 4 tem tanto o sentido de subida quanto de descida. Ele pode ter uma freqüência f_1 para a subida e uma freqüência f_2, diferente de f_1, para o nó F.

A duração do fluxo está relacionada ao tempo que determinado dado irá seguir aquele fluxo. Suponhamos que o dado 1 seja importante para o nó até o dia *d*. Neste caso a duração do fluxo seria até o dia *d*. Na ausência deste dado, o fluxo segue indefinidamente.

4.6.6 Mudança de hierarquia

Na Figura 4.30 tem-se a seguinte mudança de hierarquia:

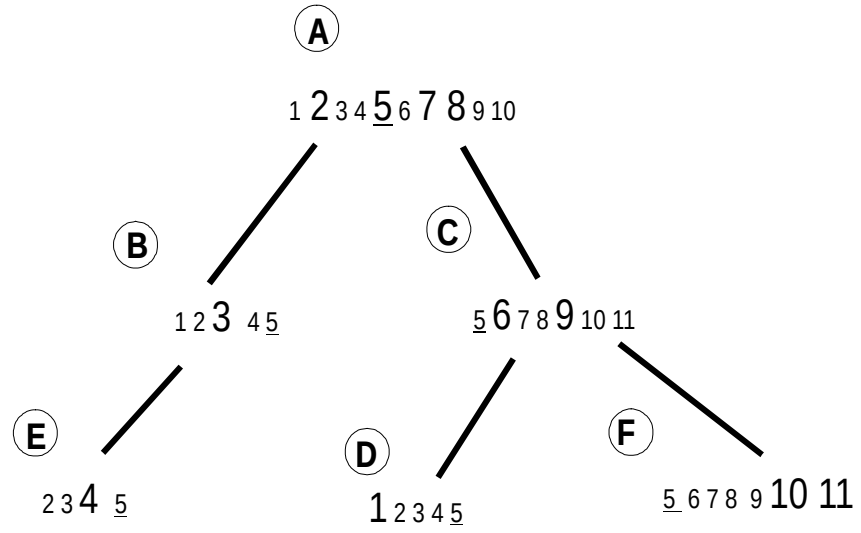

Figura 4.30 - Mudança de subordinação do nó D

Em função desta mudança de subordinação, o banco de dados ficará com o seguinte esquema de replicação, ilustrado na Figura 4.31:

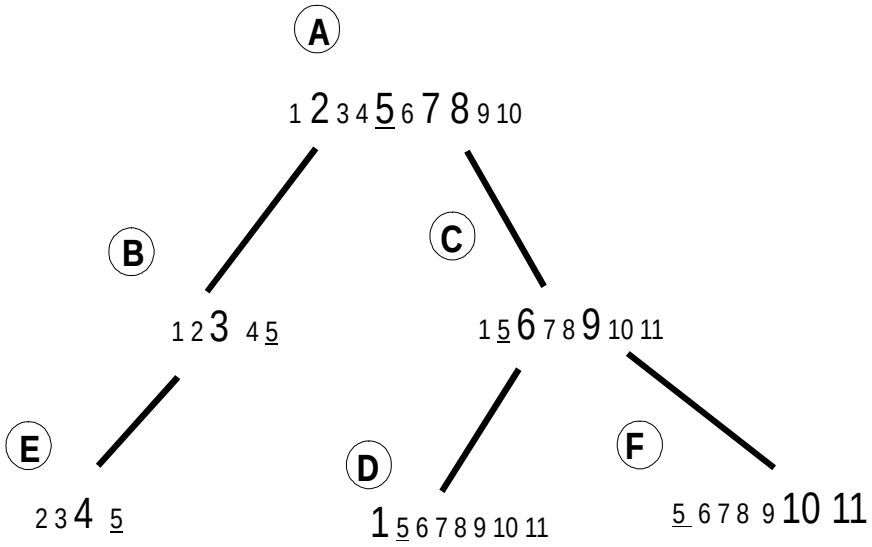

Figura 4.31 - Organização final das réplicas após mudança de subordinação

Os nó A, B, C e D alterarão sua tabela de replicação conforme a Tabela 4.3.

Canal / Nó	A	B	C	D
Uplink		3; 4	1; 6; 9;10	1
Downlink	2; 5; 7; 8	2; 3; 5	5;6;7;8;9;10;11	

Tabela 4.3 - Mudança dos canais de difusão

As mudanças de hierarquia podem gerar, segundo a conveniência do sistema, um fluxo temporário de dados, que alimente as cópias secundárias dos dados referentes às estações que mudaram de hierarquia.

Este fluxo temporário compreenderia o caminho do antigo superior até o LCA, e deste até o novo superior designado.

Por exemplo, o nó D pode estar desconectado por um determinado tempo durante a mudança de hierarquia, ou a transferência ser lenta em virtude das características do canal de comunicação.

A solução deste problema seria a criação de um fluxo temporário de dados. No caso seria adicionado na tabela de replicação do nó B, no índice de subida, o dado 1, com um tempo pré-determinado de *uplink* e no nó A, no índice (nó C) o valor 1.

Em virtude deste fluxo temporário, pode ocorrer o caso da atualização da cópia secundária de 1 chegar ao nó C oriundo de A ou de D, que possui a cópia primária..

O esquema de multiversões permite que isto ocorra sem conflito, uma vez que cada atualização corresponde a uma versão distinta.

5. Conclusão

Neste capítulo será apresentada uma breve revisão do material contido nesta dissertação, com destaque para as contribuições obtidas. Ao final serão descritos alguns trabalhos futuros que podem dar seqüência a este trabalho.

5.1 Resumo do trabalho

O objetivo principal da dissertação é a definição de um modelo de gerência de dados para computação móvel que atenda, em particular, aos requisitos de um sistema de comando e controle voltado para aplicações militares.

No início da dissertação foi apresentado o que seria um sistema de comando e controle, suas características e finalidades, bem como as diversas áreas da computação que podem ser aplicadas nestes sistemas.

Verificou-se na bibliografia pesquisada que não havia assuntos referentes a modelos de arquitetura de banco de dados móveis que fosse aplicada em sistemas táticos de comando e controle militares.

Procurou-se, então, integrar as principais áreas de conhecimento da computação móvel e elaborar uma arquitetura que integrasse estas tecnologias e atendesse, no possível, aos requisitos do sistema tático de comando e controle para um projeto ainda em desenvolvimento no Exército Brasileiro, em particular, no Instituto de Pesquisa e Desenvolvimento (IPD), órgão pertencente àSecretaria de Ciência e Tecnologia.

Assim, dever-se-iam descrever as principais técnicas de operações em ambientes de desconexão ou fraca conectividade, modelos de difusão de dados e gerenciamento de localização em um ambiente de estrutura hierárquica.

O próximo passo seria a integração das diversas técnicas, integrando um único modelo. A descrição deste modelo compõe-se de um modelo abstrato de dados, pacotes integrantes do sistema, modelo de replicação e as gerências de localização, transações e replicação.

Foram feitos ainda dois anexos, um com os algoritmos de controle de concorrência de multiversionamento e outro com o *framework* das técnicas de gerenciamento de dados propostas.

5.2 Contribuições

Como principais contribuições destacam-se:

- Um modelo de replicação de dados;
- Definição de técnicas para tratamento da busca e difusão da informação no sistema, considerando as características de mobilidade, perfil dos clientes e localização das estações; e
- Um *framework* para a implementação das técnicas propostas.

5.3 Trabalhos futuros

Esta dissertação pode ter continuidade de diversas maneiras, incluindo:

- Implementação do modelo de gerenciamento de dados aqui proposto, que não foi possível de ser realizada devido à sua complexidade e ao curto tempo disponível;
- A aplicação da tecnologia de agentes no Gerenciador de Localização;
- Modificação automática do Gerenciador de Replicação quando da mudança hierárquica;
- Comparação quantitativa e qualitativa com outros métodos de controle de concorrência, como pré-ordenação, *Serialization-Graph Testing* e *Two-Phase Locking*, considerando a distribuição e fraca conectividade da rede, baseado na arquitetura proposta; e
- Deslocamento do coordenador de gerência de transações para uma estação participante mais confiável.

Referências Bibliográficas

[ABGM90] Alonso, R., Barbara, D., Garcia-Molina H.; Data Caching Issues in a Information Retrieval System; ACM Transaction on Database Systems, 1990

[AFZ95] Acharya, S., Franklin, M., Zdonik, S.; Dissemination-based Data Delivery Using Broadcast Disks; IEEE Personal Comunications, 1995

[AFZ96] Acharya, S., Franklin, M., Zdonik, S.; Dissemination Updates on Broadcast Disks; Proceedings of the 22nd VLDB Conference, 1996

[AFZ97] Acharya, S., Franklin, M., Zdonik, S.; Balancing Push and Pull for Data Broadcast; Proceedings of the ACM Sigmod Conference, 1997

[AHMK94] Anantharam, V., Honig, M.L., Madhow, Kei, V.K. Optimization of a Database Hierarchy for Mobility Tracking in a Personal Communication Networks; Performance Eavaluation; 1994

[BA87] Boyes, J.L., Andriole, S.L.; Principles of command & control; AFCEA; 1987

[Bar97] Barbará, D.; Certification Reports: Supporting Transactions in Wireless Systems; Proceedings of the IEEE International Conference on Distributed Computing Systems; 1997

[BGM94] Barbará, D., Garcia-Molina, H.; Replicated Data Management in Mobile Environment: Anything New Under the Sun?; Proceedings of IFIP Conference on Applications in Parallel and Distributed Computing; 1994

[BHG88] Bernstein, P.; Hadzilacos, Goodman, N.; Concurrency Control and Recovery in Databases Systems; Addison-Wesley; 1988

[BI94] Barbara, D., Imielinski , T.; Sleepers and Workaholics: Caching Strategies in Mobile Environment; Proceedings of the ACM Sigmod Conference, 1994

[BIV92] Badrinath, B.R.; Imielinki, T.; Virmani, A.; Location Strategies for Personal Communications Networks; Proceedings of 1992 International Conference on Networks for Personal Communications; 1992

[Bla99] Black, U.; "Second generation Mobile & Wireless Networks"; Prentice Hall; 1999

[BMM96] Burhres, O., Morton, S., Mossman, M.; Mobile Computing Architecture for a Battlefield Environment; In International Symposium on Cooperative Database Systems for Advanced Applications, 1996

[Camp92] Campen, A. D.; The first information war; AFCEA; 1992

[Chr93] Chryssanthis,P.K.; Transactions Processing in Mobile Computing Environment; Proceedings of the IEEE Workshop on Advanced in Parallel and Distributed Sytems; 1993

[CM84] Casanova, M.A., Moura, A.V.; Princípios de Sistema de Gerência de Banco de Dados Distribuídos; IV Escola de Computação; 1984

[CM95] Cho, G., Marshall, L.F.; An Efficient Location and Routing Schema for Mobile Computing Environments; IEEE Journal on Selected Areas in Communications; June 1995

[DCKVK97] Datta, A., Celik, A., Kim, J., VanderMeer, D., Kumar, V.; Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environment; Proceedings of the 13th IEEE International Conference on Data Engineering, 1997

[DGMS85] Davidson, S.B., Garcia-Molina, H., Skeen, D.; Consistency in Partitioned Networks; ACM Computing Surveys; 1985

[FZ94] Forman, G.H.; Zahorjan, J.; The Challenges of Mobile Computing; IEEE Computer; 1994

[GHOS96] Gray, J., Helland, P., O'Neil, P., Sasha, D.; The Dangers of Replication and a Solution; Proceedings of the ACM SIGMOD; 1996

[HJM94] Harjono, H., Jain, R., Mohan, S.; Analysis and Simulation of a Cache-Based Auxiliary User Location Strategy for PCS; Proceedings of 1994 International Conference on Networks for Personal Communications; March 1994

[HSW94] Huang, Y., Sistla, P., Wolfson, O.; Data Replication for Mobile Computers; Proceedings of the 1994 SIGMOD Conference; 1994

[HW94] Huang, Y., Wolfson, O.; Object Allocation in Distributed Databases and Mobile Computing; Proceedings of the 10th International Conference on Data Engineering; 1994

[IB92] Imielinski, T., Badrinath, B.R.; Replication and Mobility; Proceedings of the 2nd IEEE Workshop on the Management of Replicated Data; 1992

[IB93] Imielinski, T., Badrinath, B.R.; Data Management for Mobile Computing; SIGMOD Record; 1993

[IB94] Imielinski , T., Badrinath, B. R.; Adaptive Wireless Information Systems; Proceedings of the SIGDBS Conference, 1994

[IB94a] Imielinki, T.; Bradinath, B.R.; Wireless Mobile Computing: Challenges in Data Management; Communications of the ACM, 37(10); 1994

[IEWG97] Ince, A.N., Evrendilek, C., Wilhelmsen, D., Gezer, F.; planning and Architectural Design of Modern Command Control Communications and Information System; Kluwer; 1997

[IK96] Imielinski, T., Korth, H.F.; Mobile Computing; Kluwer; 1996

[Jain96] Jain, R.; Reducing Traffic Impact of PCS Using Hierarchical User Location Databases; Proceedings of the IEEE International Conference on Communications; 1996

[JBE95] Jing, J., Bukhres, O., Elmagarmid, A.; Distributed Lock Management for Mobile Transactions; Proceedings of the 15th IEEE International Conference on Distributed Computing Systems; 1995

[JBEA95] Jing, J., Bukhers, O., Elmagarmid, A., Alonso, R.; Bit-Sequence: A New Cache Invalidation Method Environment, Department of Computer Science, Purdue University, 1995

[JL94] Jain, R., Ling, Y-B; A Caching Strategy to Reduce Networks Impacts of PCS; IEEE Journal on Selected Areas in Communication; October 1994

[JL95] Jain, R., Ling, Y-B; A Auxiliary User Location Strategy Employment to Reduce Networks Impacts of PCS; Wireless Networks; 1995

[JLSWC96] Jannink, J.; Lam, D.; Shivakumar, N., Widom, J., Cox, D. C.; Efficient and Flexible Location Management Techniques for Wireless Communication System; Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (Mobicom'96); 1996

[KS92] Kistler, J.J.; Satyanarayanan, M.; Disconnected Operation in the Coda File System; ACM Transaction on Computer Systems; 1992

[KS95] Kumar, P., Satyanarayanan, M.; Flexible and Safe Resolution of File Conflicts; Proceedings of the USENIX; 1995

[Kue94] Kuenning, G.H.; The Design of the Seer Predictive Caching System; Proceedings of the IEEE Workshop Computing Systems and Applications; 1994

[KVP96] Krishna, P., Vaidya, N.H., Pradhan, D.K.; Static and Dynamic Location Management in Mobile Networks; Journal of computer Communications; March 1996

[LS94] Lu, Q., Satyanarayanan, M.; Isolation-Only Transations for Mobile Computing; Operating Systems Review; 1994

[LS95] Lu, Q., Satyanarayanan, M.; Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions; Proceedings of the Fifth Workshop on Hot Topics in Operating Systems; 1995

[Oliv99] Oliveira, H.J.C.; Comando, controle, comunicação e inteligência (C^3I) nas operações de segurança; A Defesa Nacional; 4^o trim 1999

[OV9] Özsu, M.T., Valduriez, P.; Principles of Distributed Database Systems 2^{nd} edition; Prentice Hall; 1999

[PB95] Pitoura, E., Bhargava, B.; A Framework for Providing Consistency of Data in Distributed Environments; Proceedings of the 15^{th} IEEE International Conference on Distributed Computing Systems; 1995

[PC99] Pitoura, E., Chrysanthis, P.;Scalable Processing of Read-Only Transactions in Broadcast Push, IEEE International Conference on Distributed Computing Systems, Austin, 1999

[PF97] Pitoura, E., Fudos, I.; Tracking Mobile Users Using Hierarchical Location Databases; Technical Report DCS-97-10; Department of Computer Science; University of Ioannina; 1997

[Pit96] Pitoura, E.; A Replication Schema to Support Weak Connectivity in Mobile Information Systems; Proceedings of the 7^{th} International Conference on Database and Expert systems Applications; 1996

[SBCM95] Samaras, G., Britton, K., Ctiron, A., Mohan, C.; Two-Phase Commit Optimizations in a Commercial Distributed Environment; Distributed and Parallel Databases; 1995

[Ske91] Skeen, D.; Nonblocking Commit Protocols; Proceedings of the ACM SIGMOD; 1991

[SRB97] Stathatos, K., Roussopoulus, N., Baras, J. S.; Adaptive Data Broadcast in Hybrid Networks; Proceedings of the 23rd VLDB Conference, 1997

[SS83] Skeen, D., Stonebraker, M.; A Formal Model of Crash Recovery in a Distributed System; IEEE Trans. Software Eng, pg 219-228; 1983

[TDPS91] Terry, D., Demers, A., Petersen, K. Spreitzer, M., Theimer, M., Welch, B.; Session Guarantees for Weakly Consistent Replicated Data; Proceedings of the First International Conference on Parallel and Distributed Information Systems; 1991

[TLAC95] Tait, C., Lei, H., Acharya, S.; Chang, H.; Intelligent File Hoarding for Mobile Computers; Proceedings of the 1st ACM International Conference on Mobile Computing and Networking (Mobicom'95); 1995

[Wang93] Wang, J. Z.; A Fully Distributed Location Registration for Universal Personal Communication Systems; IEEE Journal on Selected Areas in Communication; August 1993

Anexos

Anexo 1 - Algoritmo do gerenciador de transação de 1ª classe

Declare-type

Operation: Pode ser Begin_transaction, Read, Write, Abort ou Commit

DataItem: um item de dado dentro do banco de dados

TransactionId: identificador único designado para cada transação

DataVal: primitiva de tipo de dado (ex: inteiro, real, etc.)

TipoRepl: tipo de réplica retornado (cópia primária ou secundária, versão mestre ou tentativa, com ou sem prazo de validade)

SiteId: identificador único de site

Dbop: uma tupla tripla de {operação de banco de dados oriunda da aplicação}

opn: Operation

data: DataItem

tid: TransactionId

Dpmsg: uma tupla quádrupla de {resultado da operação do *Data Processor*}

opn: Operation

tid: TransactionId

retTpRepl: TipoRepl

res: DataVal

Scmsg: uma tupla tripla de {mensagem originada do *Scheduler*}

opn: Operation

tid: TransactionId

res: DataVal

Transaction

tid: TransactionId

ts: timestamp da transação

Body

Vector(operation) {vetores das operações participantes da transação}

Message: um string de caracteres a serem transmitidos

OpSet: um conjunto de DbOp's

SiteSet: um conjunto de SiteId's

WAIT (msg: Message)

begin

{aguarda até a mensagem chegar}

end

Declare-var

T: Transaction

Op: Operation

x: DataItem

msg: Message

O: Dbop

pm: Dpmsg

res: DataVal

S: Site

begin

repeat

WAIT (msg)

case of msg

Dbop: (operação de bds do programa de aplicação)

begin

case of Op

Begin_Transaction:

begin

Designar um tid (identificador único de transação)

nr_op[tid] = 0

nr_op_OK[tid] = 0 {número de operações recebidas dos *Schedulers*}

S[tid] = ∅

designar um timestamp para T[*ts(T)*]

end

Read:

Tipo = GerLoc.TipoDado(dataItem)

nr_op[tid] = nr_op[tid] + 1

case of Tipo

begin

Cópia Secundária com prazo de validade:

Cópia Primária:

send O para o scheduler local

S[tid] = S[tid] ∪ sitelocal

Cópia Secundária sem prazo de validade:

Site=GerLoc.SiteCopiaPrimaria(dataItem)

send O para o Scheduler de Site

S[tid] = S[tid] ∪ Site

end

Write:

nr_op[tid] = nr_op[tid] + 1

Tipo = GerLoc.TipoDado(dataItem)

case of Tipo

begin

Cópia Primária:

send O para o scheduler local

S = S ∪ sitelocal

Cópia Secundária sem prazo de validade:

Site=GerLoc.SiteCopiaPrimaria(dataItem)

send O para o Scheduler de Site

S[tid] = S[tid] ∪ Site

end

Abort:

begin

send O para todos os *Scheduler* em S[tid]

end

Abort:

begin

send O para todos os *Scheduler* em S[tid]

end

Commit:

begin

if (nr_op_OK[tid] = nr_operação[tid]) {o número das operações enviadas deve ser igual ao número das responsta}

send O para todos os *Scheduler* em S[tid]

else

Avisar a aplicação para esperar todas as respostas

end

end

Scmsg: (a operação foi rejeitada pelo *Scheduler*)

begin

msg = “Abort T”

send *msg* to *Schedulers* in S[tid]

end

Dpmsg: (mensagem vinda do *Data Processor*)

begin

Op = pm.opn; res = pm.result; T= pm.id

case of Op

Read:

begin

retornar o resultado e o tipo de dado para a aplicação que está controlando a transação

nr_op_OK[tid] = nr_op_OK[tid] + 1

end

Write:

begin

retornar para a aplicação que está controlando a transação que a operação de escrita foi realizada com sucesso

nr_op_OK[tid] = nr_op_OK[tid] + 1

end

Commit:

begin

Informar a aplicação que a operação de commit foi realizada com sucesso

end

Abort:

begin

Informar a aplicacao que a operacao de abort foi realizada com sucesso

end

end

end-case (Op – Dpmsg)

end (Dpmsg)

until *forever*

end

Anexo 2 - Algoritmo do Scheduler

Declare-type

Versao: uma tupla tripla de {operação de banco de dados oriunda da aplicação}

Timestamp: Timestamp da versão

Status: Pode ser *pendente* ou *confirmado*

SetRead: conjunto de timestamp das operações de leitura já realizadas

Declare-var

msg: Message

dop: Singleop

Op: Operation

x: DataItem

T: TransactionId

SOP: OpSet

Vrs: Versão

writedoneMinTS: timestamp {timestamp de uma versão escrita já confirmada}

writedoneMaxTS: timestamp {timestamp de uma versão escrita já confirmada}

readTS

writeVersao: Versao

setWriteVersao: conjunto de Versao contendo as versões escritas

setRead: SetRead

begin

repeat

WAIT (msg)

case of msg

Dbop: {mensagem passada pelo Gerenciador de Transações}

begin

Op = dop.opn

x = dop.data

T = dop.tid

case of Op

Read:

begin

Vrs = GetVersao(x, ts(T))

if (Vrs.Status = pendente) **then**

Refrear a operacao e colocá-la de volta na fila de entrada

else begin

send Op to the *Data Processor*

Inserir (x , ts(T)) no vetor de leituras de x

end

end {of Read case}

Write:

begin

writedoneMaxTS = setWriteVersao.GetMenorMaiorWrite(x, ts(T))

readTS = setRead.getReadMaiorMenor(x, writedoneMaxTS)

if (ts(T) < readTS) **then**

begin

msg = “Reject T”

send msg para o Gerenciador de Transacao

end

else begin

send dop para o *Data Processor*

writeVersao.timestamp = ts(T)

writeVersao.status = pendente

setWriteVersao.insert(x, writeVersao)

end

end (Write)

Commit:

begin

setWriteVersao.Confirmar(x, ts(T)) {Muda o status da versao}

send *dop* para o *Data Processor*

end (Commit)

Abort:

begin

excluir todos os dados dados acessador pela transacao T

send dop para o *Data Processor*

end (Abort)

end-case (Op)

end-case (msg)

until *forever*

end

Anexo 3 -Framework para implementação das técnicas

Este anexo descreve um *framework* para implementação do modelo proposto na dissertação. A visão geral do sistema pode ser vista na seção 4.2.3. A idéia principal é mostrar as principais funcionalidades que o sistema deve oferecer.

Dividiu-se este anexo em quatro seções, seguindo a linha para a compreensão lógica do sistema: Gerenciador de Localização; *Scheduler* e *Data Processor*; Gerenciador de Transações; e Gerenciador de Replicação.

Gerenciador de Localização

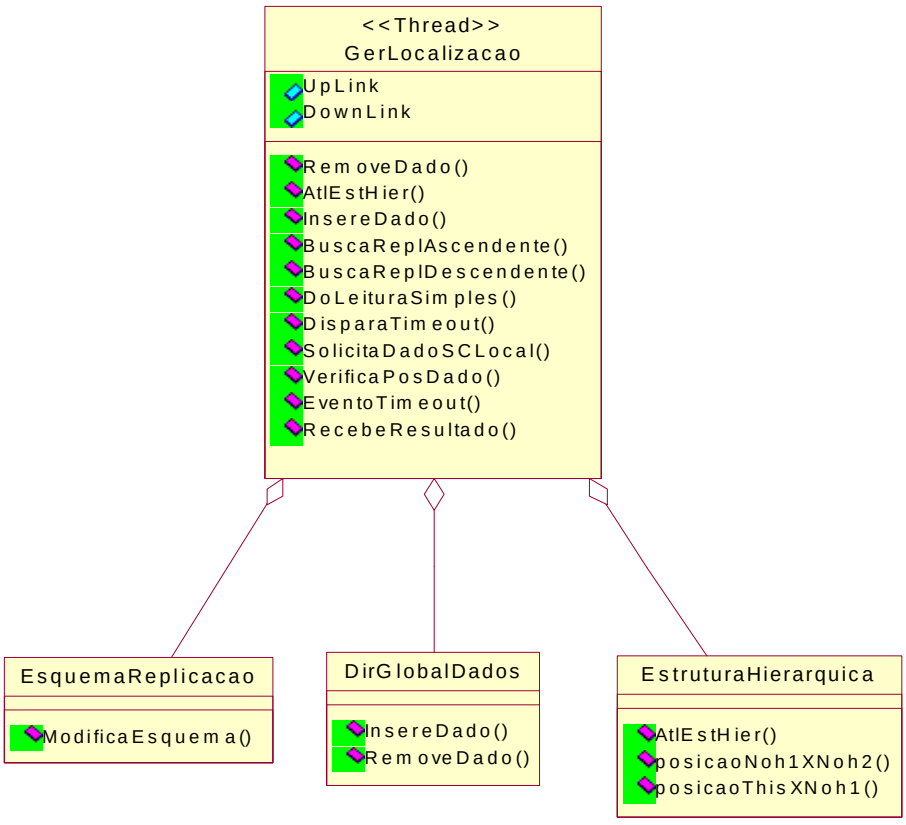

Figura 1 - Componentes principais do Gerenciador de Localização

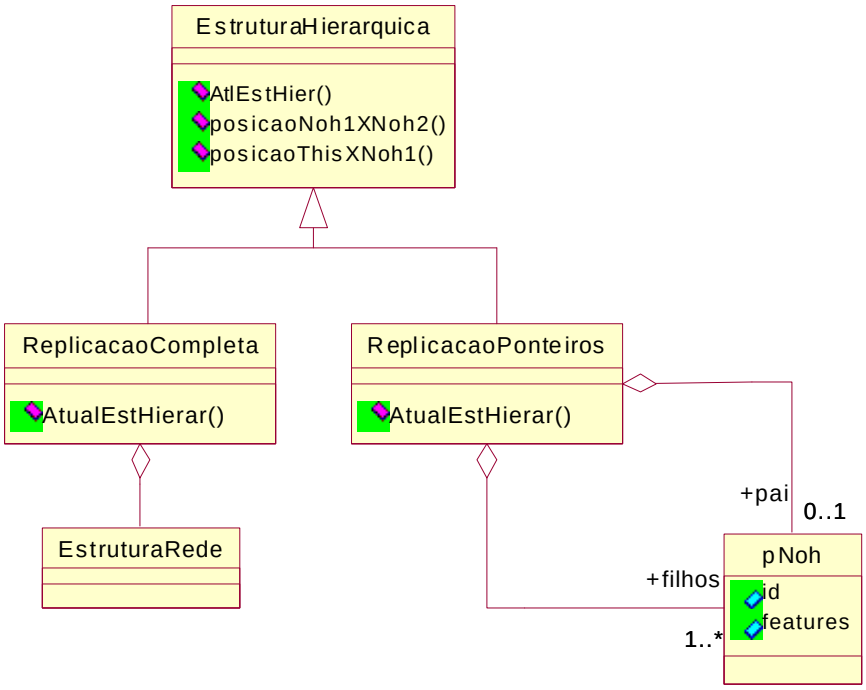

Figura 2 - Diagrama de classes da estrutura hierárquica

O nomes das operações componentes das classes estão definidas de forma a compreender a sua funcionalidade.

A figura 3 ilustra o diagrama de colaboração entre classes referentes à atualização da estrutura hierárquica no caso de uma arquitetura de replicação completa. O diagrama de colaboração no caso de uma arquitetura de replicação parcial seria semelhante, com exceção do número de participantes da operação, conforme abordado na seção 4.4.2.

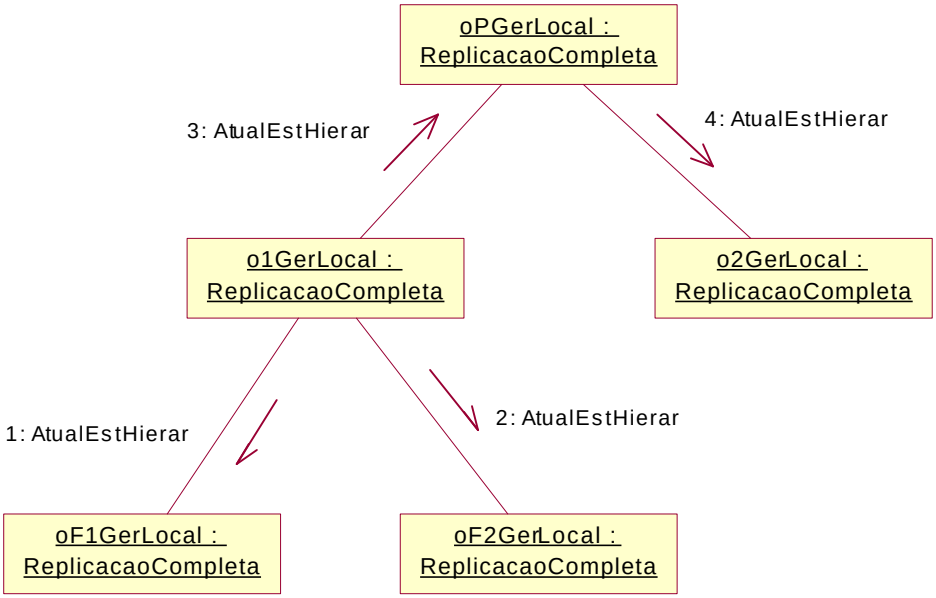

Figura 3 - Diagrama de colaboração de atualização hierárquica

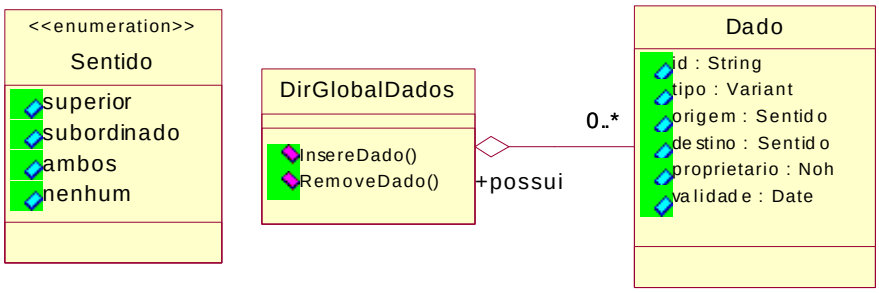

Figura 4 - Diagrama de classes do diretório global

Scheduler e Data Processor

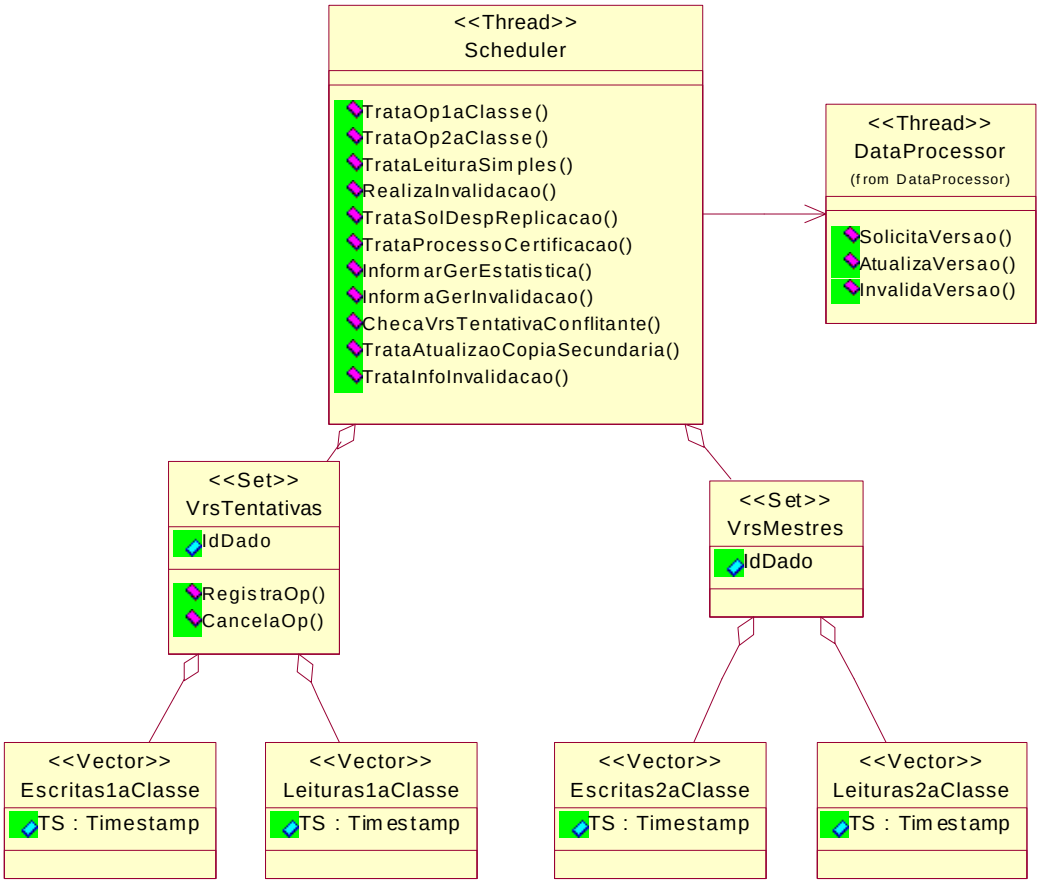

Figura 5 - Diagrama de classes do Scheduler e Data Processor

Gerenciador de Transações

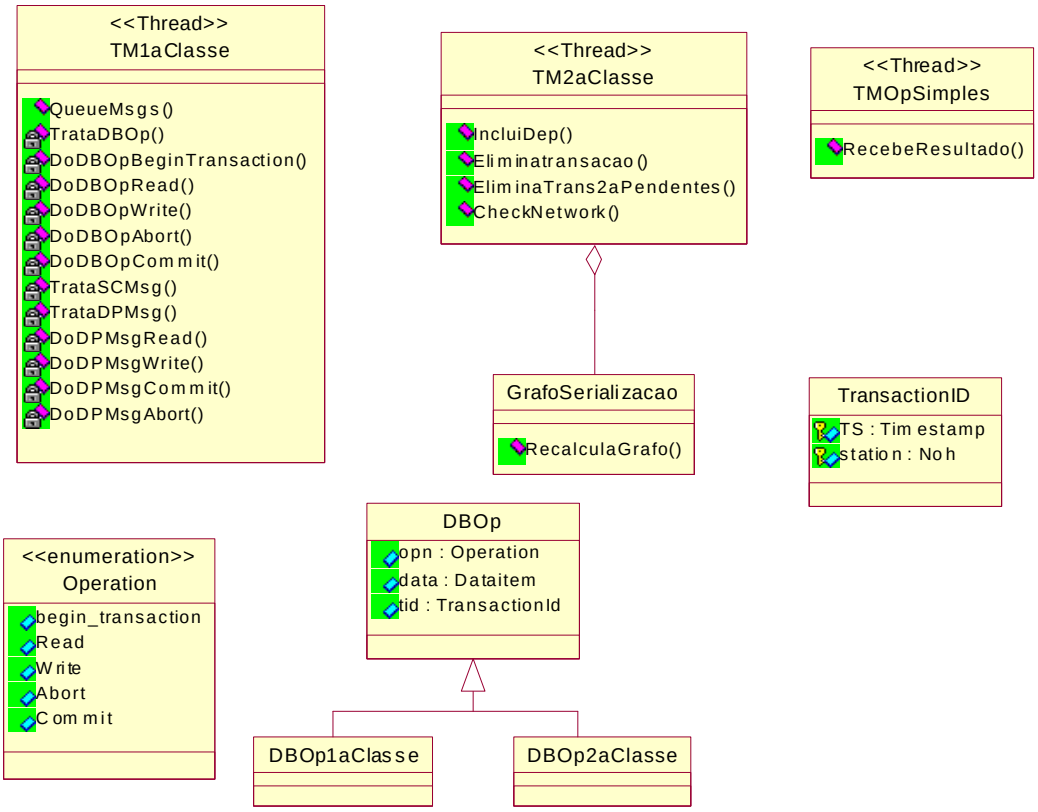

Figura 6 - Diagrama de classes do Gerenciador de Transações

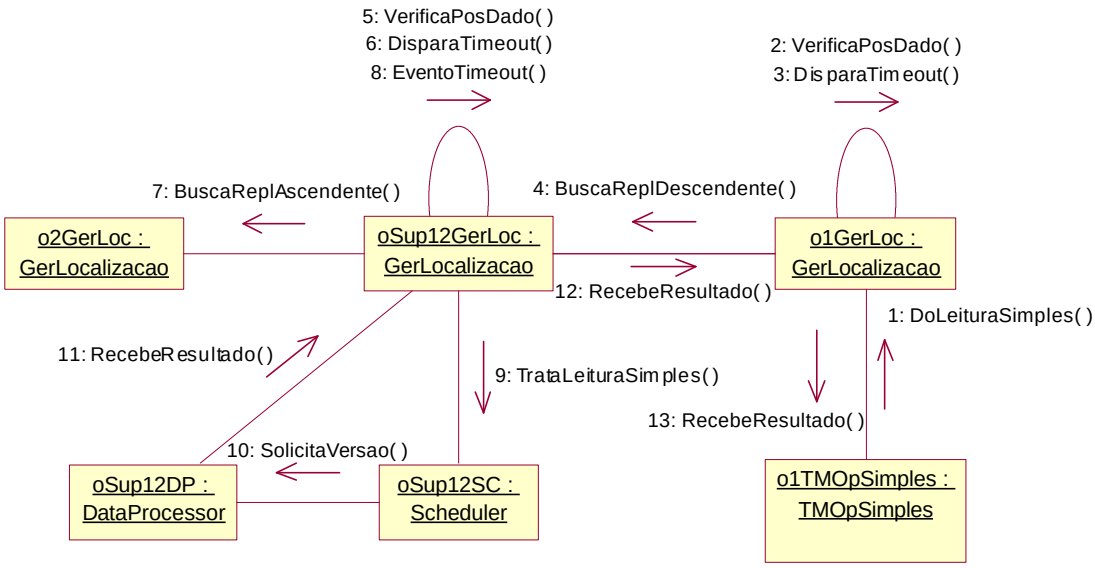

Figura 7 - Diagrama de colaboração na execução de uma leitura simples

A figura 7 ilustra a implementação da execução de uma leitura simples, que não interfere no controle de concorrência.

A figura 8 ilustra a execução de transações de 1ª classe e de 2ª classe concorrendo em um mesmo *Scheduler*.

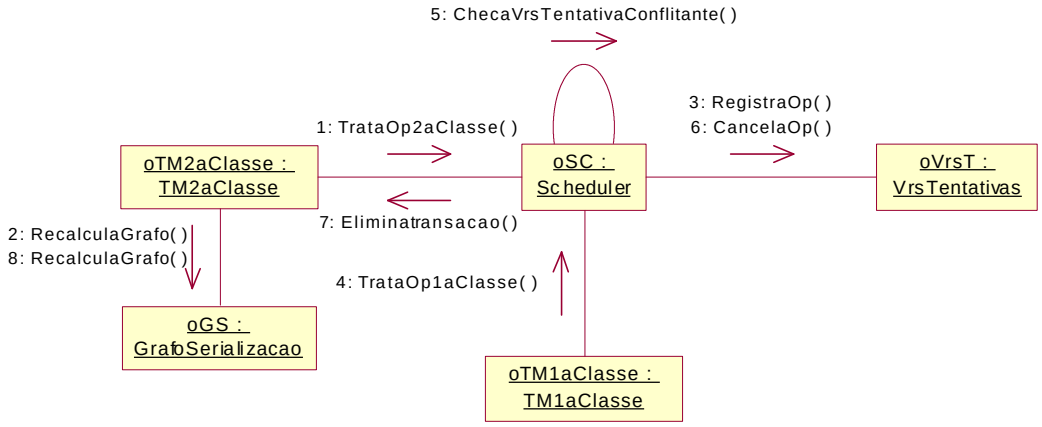

Figura 8 - Diagrama de colaboração entre transações de 1ª e 2ª classe

Gerenciador de Replicação

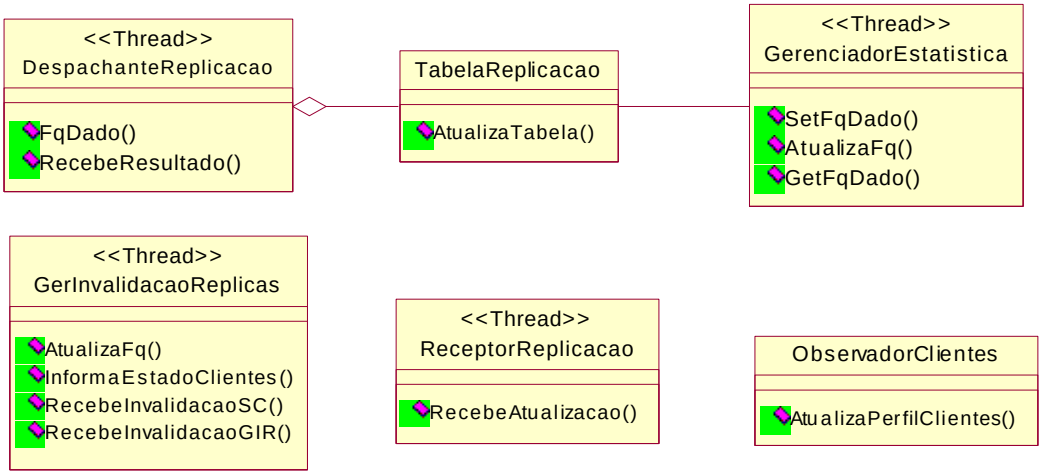

Figura 9- Diagrama de classes do gerenciador de replicação

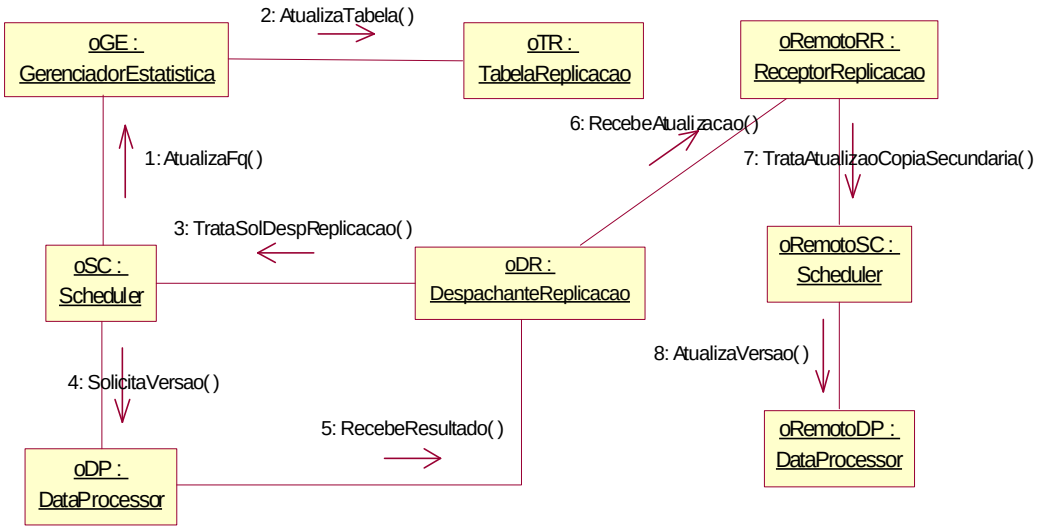

Figura 10- Diagrama de colaboração numa atualização de cópias secundárias

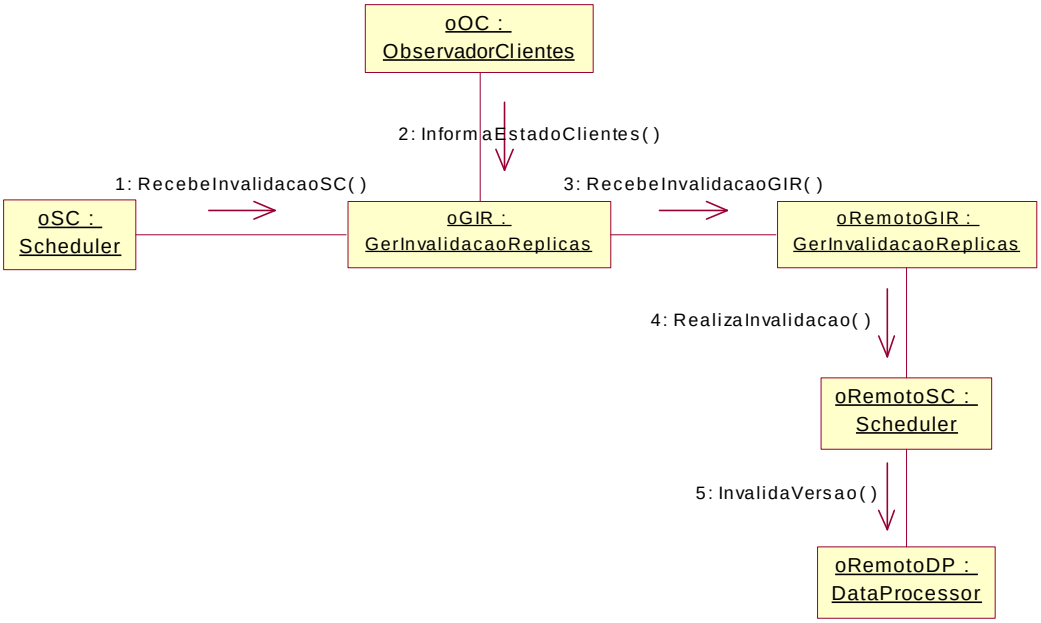

Figura 11 - Diagrama de colaboração de invalidação de réplicas