UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Especificação e implementação de um modelo assíncrono para replicação, propagação e conciliação de bases de dados distribuídas

Vinicius Fernandes de Oliveira

Orientador: Prof. Sergio Vianna Fialho, D.Sc.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Natal, RN, Julho de 2007

Divisão de Serviços Técnicos
Catalogação da Publicação na Fonte. UFRN / Biblioteca Central Zila Mamede

Oliveira, Vinicius Fernandes de.
Especificação e implementação de um modelo assíncrono para replicação, propagação e conciliação de bases de dados distribuídas / Vinicius Fernandes de Oliveira. – Natal, RN, 2007.
111 f. : il.

Orientador : Sergio Vianna Fialho.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica.

1. Base de dados – Dissertação. 2. Banco de dados distribuído – Dissertação. 3. Replicação de dados – Dissertação. I. Sergio Vianna Fialho. II. Título.

RN/UF/BCZM CDU 004.65(043.3)

À minha esposa Eloiza e aos meus filhos, Laura e Bruno, pela compreensão e colaboração durante a realização deste trabalho.

Agradecimentos

Ao meu colega de trabalho Cledson Francisco Silva, pelo apoio técnico e logístico.

Aos diretores da Bonor Indústria de Botões do Nordeste S.A., pelo apoio que sempre me deram na realização deste projeto.

À minha família, pela paciência e compreensão.

Resumo

Esta dissertação propõe um modelo para replicar de forma assíncrona bases de dados heterogêneas. A principal característica do modelo é ter reunido, de forma sistematizada e em um só projeto, diversos conceitos, técnicas e paradigmas relacionados às áreas de replicação de banco de dados e gerenciamento de bases de dados heterogêneas. Uma das principais vantagens da replicação é permitir aos aplicativos continuar processando informações durante intervalos de tempo em que se encontram desconectados da rede, garantindo o disparo de procedimentos de sincronização entre as bases de dados, tão logo a conexão venha a se restabelecer. Faz parte do modelo, portanto, um protocolo de comunicação e atualização entre bases de dados replicadas, que leva em conta o ambiente de característica assíncrona onde será utilizado. Por fim, foi desenvolvida, com o objetivo de realizar testes e validações, uma ferramenta na linguagem Java baseada nas premissas do modelo.

Abstract

This graduate thesis proposes a model to asynchronously replicate heterogeneous databases. This model singularly combines - in a systematic way and in a single project - different concepts, techniques and paradigms related to the areas of database replication and management of heterogeneous databases. One of the main advantages of the replication is to allow applications to continue to process information, during time intervals when they are off the network and to trigger the database synchronization, as soon as the network connection is reestablished. Therefore, the model introduces a communication and update protocol that takes in consideration the environment of asynchronous characteristics used. As part of the work, a tool was developed in Java language, based on the model's premises in order to process, test, simulate and validate the proposed model.

Sumário

Lista das figuras

Lista das tabelas

Lista de acrônimos

2LSR ..... *Two-level serializability*
2PC ..... *Two-phase commit protocol*
3PC ..... *Three-phase commit protocol*
ACID ..... Atomicidade, Consistência, Isolamento e Durabilidade
BD..... Banco de Dados
BDD..... Banco de Dados Distribuído
CAD..... *Computer-Aided Design*
CASE..... *Computer-Aided Software Engineering*
CPU ..... *Central Processing Unit*
CVS ..... *Control Version Software*
DARPA ..... *Defense Advanced Research Projects Agency*
DNS ..... *Domain Name Server*
DOM..... *Document Object Model*
DOS..... *Disk Operating System*
Ficus ..... *File Sharing Under Fícus*
FDBS..... *Federated Database System*
IP ..... *Internet Protocol*
LAN ..... *Local Area Network*
MANET ..... *Mobile Ad-Hoc Networks*
NIS..... *Name Information Service*
OIS ..... *Office Information Systems*
OMG..... *Object Management Group*
PDA..... *Personal Digital Assistants*
PDU ..... *Protocol Data Unit*
Roam..... *Replication System for Mobile Environments*
RPCD..... Replicação, Propagação e Conciliação de Dados
Rumor..... *Replication for Mobile Computers*
RUP..... *Rational Unified Process*
SGBD ..... Sistemas Gerenciadores de Banco de Dados
SGBDD ..... Sistemas Gerenciadores de Banco de Dados Distribuído
SQL ..... *Structured Query Language*
TCP ..... *Transmission Control Protocol*
Truffles ..... *Secure Flexible File-Sharing Over Wide-Area Networks*
UML ..... *Unified Modeling Language*
URL..... *Uniform Resource Locator*
W3C ..... *World Wide Web Consortium*
WAN ..... *Wide Area Network*
WARD ..... *Wide Área Replication Domain*
XML ..... *eXtensible Markup Language*

Glossário de termos recorrentes

Aplicado (*tentative*) .....estado da réplica onde foi aplicada uma alteração. Pode ser desfeita antes da efetivação.

Arquivo de transações (*log*) .....registro local das operações ou tentativas de efetivá-las.

Conciliação (*resolve*) .....processo de identificar e resolver conflitos entre alterações.

Conciliador (*resolver*).....procedimento disponibilizado por uma aplicação para resolver conflitos.

Conflito .....conjunto de alterações de diferentes *sites* que juntos violam a consistência dos dados.

Consistência.....garantia de que as informações estão corretas, dentro de um mesmo objeto ou entre objetos.

Domínio da replicação .....conjunto de *sites* que adotam o mesmo modelo global, ou seja, compartilham um ou mais objetos replicados.

Efetivação (*commit*) .....processo de acatar um estado comum dos dados ou uma ordem final na fila de operações.

Escalonamento de operações (*schedule*) a ordem em que se aplicam as operações.

Mestre (*master*) .....o *site* ou sistema que disponibiliza as alterações do objeto replicado.

Mestre-simples.....(*single-master*)sistema que possui apenas um mestre por objeto.

Multi-Mestre (*multi-master*) .....sistema que possui diversos mestres por objeto.

Modelo global .....modelo comum adotado entre réplicas ou base de dados, que possibilita a utilização de bases heterogêneas.

Objeto.....qualquer tipo de dado ou fragmento de dado compartilhado. Também são considerados dados: imagens, sons, textos, objetos multimídia e sistemas de arquivos.

Operação .....descrição semântica de uma alteração em um objeto.

Propagação.....transferência de alterações de um *site* para outros *sites.*

Pull .....técnica onde um *site* tem a iniciativa de procurar outros *sites* para receber novas alterações

Push.....técnica onde um site tem a iniciativa de enviar para outros sites suas novas alterações

Qualidade de conteúdo .....medida do quanto o conteúdo das informações entre réplicas é idêntica.

Regra de Thomas (*Thomas write rule*) algor

Relógio físico (*physical clock*).....relógio do *hardware* de cada computador que marca a passagem do tempo

Relógio lógico (*logical clock*).....técnica para identificar as precedências entre alterações.

Réplica.....uma cópia de um objeto para acesso local.

Replicação epidêmica .....modo de propagação que possibilita qualquer par de *sites* trocar quaisquer atualizações entre si.

Site .....no contexto da replicação é o lugar onde um conjunto de réplicas está disponível para acesso local.

Transferência de estado .....técnica que propaga alterações utilizando apenas dados ou todo o objeto.

Transferência de operação.....técnica que propaga alterações utilizando operações.

Transformação de operações.....técnica que troca os parâmetros de uma operação por outros, visando à consistência das informações.

Uniformidade.....propriedade do conteúdo da réplica para a qual eventualmente converge.

Vetor de *timestamps*.....estrutura de dados que registra a ordem distribuída das operações entre *sites*

Vetor de versão (*version vector*)....par de vetores *timestamp* de um objeto utilizado para identificar conflitos.

Observação: a breve explicação da maioria desses conceitos tem como base [Saito,2002-p.5]

Capítulo 1

Introdução

A proposta de distribuir bases de dados é tão antiga quanto as próprias bases de dados. Os principais motivadores para isto sempre foram *disponibilidade* e *desempenho.* Maior disponibilidade porque, se a base de dados estiver próxima da aplicação, ela estará disponível mais tempo. Melhor desempenho porque o acesso a dados locais é mais rápido do que através da infra-estrutura de comunicação de uma rede de computadores. Todas as propostas, projetos e implementações que desde então começaram a surgir, sempre enfrentaram grandes desafios relacionados às questões de segurança, consistência e confiabilidade no gerenciamento dos dados.

Para analisar os diversos modelos existentes, é importante levar em conta a realidade tecnológica da época em que foram propostos. Na década de 60 e 70, por exemplo, o custo da tecnologia de informação era maior, predominavam os *mainframes,* a maioria das arquiteturas de *hardware*, *software* e comunicação eram proprietárias e havia limitações no armazenamento, processamento e comunicação de dados. Atualmente existem a Internet, redes locais, redes com grande abrangência geográfica, arquiteturas e padrões abertos em diversas áreas de aplicação, um grande número de computadores pessoais interconectados e a computação móvel – *notebooks, palm-tops* e celulares – cada vez mais presente. A tecnologia da informação disseminou-se nas mais diversas áreas, trazendo novas possibilidades de aplicação.

Paralelamente à consolidação de tais tecnologias, continuam surgindo propostas relacionadas à distribuição de bases de dados. Só que, atualmente, o conceito de "dado" é bastante extenso. Representa, além de caracteres e números, imagens, sons, textos inteiros ou tudo isso ao mesmo tempo. É o caso das aplicações de CAD (projeto auxiliado por computador), CASE (engenharia de software auxiliado por computador), OIS (sistema de informações de escritório), aplicações multimídia e hipertextos. Isto obrigou o desenvolvimento de novos conceitos de modelagem de dados, que pudessem atender às necessidades dessas aplicações [Silber,99-p250]. Conseqüentemente, os bancos de dados replicados e distribuídos também tiveram que se adequar a tais necessidades.

1.1– Motivação deste trabalho

Parte da área de gerenciamento de bancos de dados distribuídos é dedicada à utilização da técnica da replicação, voltada principalmente para os casos em que as bases de dados devem ficar desconectadas entre si durante determinados períodos de tempo. Esse tempo de desconexão pode ser de milésimos de segundos até chegar a horas ou mesmo dias. O foco principal da presente dissertação são aplicações que devem ficar desconectadas entre si por longos períodos de tempo. De fato, existem muitas aplicações que necessitam desse tipo de abordagem. Alguns exemplos significativos são listados a seguir:

- Automação de força de vendas: vendedores ou representantes que viajam ou transitam entre diversos clientes poderão processar pedidos e prestar informações *off-line*.
- Redes varejistas: empresas varejistas, com muitas lojas espalhadas em grandes áreas geográficas, não podem depender de um processamento centralizado, principalmente no atendimento ao público (frente de loja).
- Trabalhos em campo: distante de antenas, centros urbanos e sem comunicação com satélites, qualquer aplicação terá que ser processada *off-line.*
- Aplicações embarcadas: aviões, navios e caminhões poderão ficar em muitos momentos sem comunicação de dados com uma central. Nesses momentos devem continuar processando seus dados "normalmente", até haver um posterior contato com a central e uma reconciliação de dados.

Na elaboração desse trabalho, verificou-se que uma replicação eficiente pode ainda auxiliar na migração de aplicativos e base de dados. Em um período de transição, bases de dados de diferentes gerenciadores podem coexistir, enquanto os programas aplicativos vão sendo substituídos pouco a pouco.

Saito [Saito,2005-p43] generaliza como vantagem o uso da técnica de replicação, quando se refere a aplicações que utilizam a Internet como um meio de comunicação, para aplicações em computação móvel sujeitas a conectividade intermitente e compartilhamento de dados em algumas atividades humanas, tais como grupos de engenharia e desenvolvimento de *software,* que freqüentemente trabalham em relativo isolamento.

1.2– Objetivos do trabalho

Assim, esse trabalho tem por objetivo propor, implementar e testar um novo modelo para fazer a replicação de bases de dados, apresentando as seguintes características:

a- Ser aplicável a qualquer tipo de banco de dados, incluindo ambientes com bases heterogêneas.
b- Utilizar replicação de forma assíncrona.
c- Utilizar um protocolo de comunicação com serviços específicos, definido para o processo de replicação.
d- Ser independente de conceitos específicos das linguagens de programação, podendo ser implementado em qualquer linguagem, desde que orientada a objeto.
e- Apresentar soluções que exigem pouca ou nenhuma alteração nos aplicativos usuários dos sistemas de banco de dados.
f- Ser modularizado com divisões claras de responsabilidades, o que permite a inserção de algoritmos especializados em qualquer ponto do modelo.

1.3– Contexto da pesquisa

Este trabalho pode ser incluído na área de Sistemas de Banco de Dados Distribuídos com Replicação Otimista, conforme mostrado no esquema taxionômico abaixo.

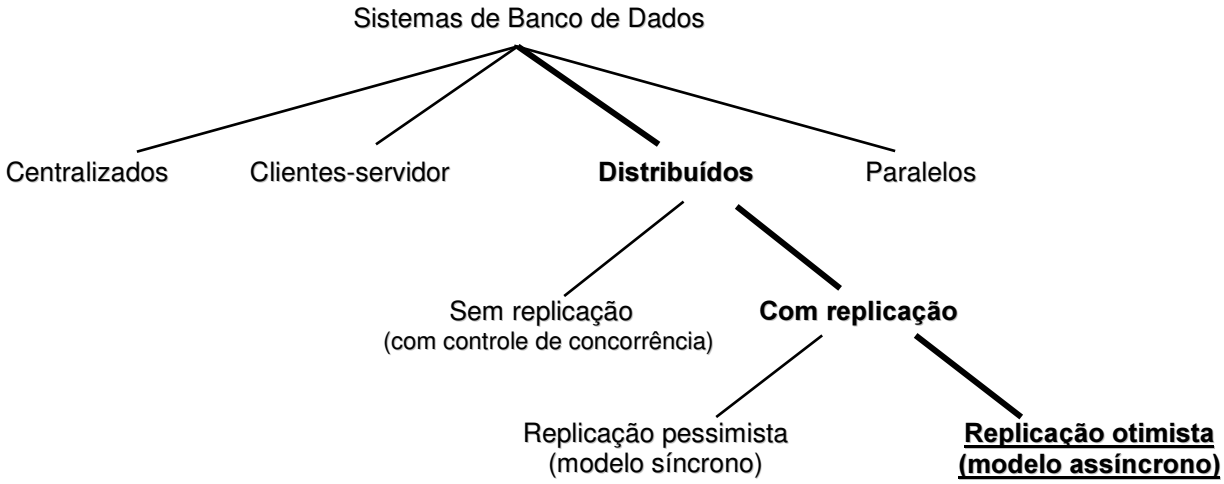


Figura 1.1 – Replicação otimista no contexto da área de sistemas de banco de dados

1.4– Organização do documento

Esse documento está organizado da seguinte forma: no capítulo 2 são tratados os conceitos que dão o suporte teórico necessário para o desenvolvimento da atual proposta. Por se tratar de um assunto pouco sistematizado em livros didáticos, nesse capítulo dedicou-se um espaço maior ao levantamento bibliográfico realizado em vários artigos científicos sobre assuntos relacionados à replicação de dados e também a um resumo dos trabalhos e aplicações relacionados com a atual proposta. No capítulo 3 são descritos os fundamentos do modelo proposto e propostas para extensões do modelo. O capítulo 4 é dedicado aos principais aspectos da implementação desse modelo. Finalmente, apresentam-se as conclusões do trabalho.

Capítulo 2

Principais aspectos teóricos relacionados à replicação

O maior desafio na replicação de banco de dados está em garantir que todas as cópias permaneçam idênticas, independente das transações a que são submetidas. O procedimento que irá garantir isto se chama *conciliação* ou *reconciliação*. O termo *reconciliação* reforça a idéia de que a *conciliação* será repetida inúmeras vezes. Já o termo *conciliação* está mais relacionado às *regras de conciliação,* ou seja, critérios utilizados para resolver divergências entre informações replicadas.

Praticamente, todos os conceitos e soluções utilizados em banco de dados replicados giram em torno do fato de que, em algum momento, haverá reconciliação. Tudo é feito para garantir o sucesso desse procedimento. Esse, por sua vez, garante a segurança, a confiabilidade e eficácia das aplicações que utilizam o banco de dados replicado.

Nas seções 2.1 a 2.3 são revistos alguns aspectos relacionados a sistemas de bancos de dados. A seção 2.4 relaciona conceitos de replicação com os conceitos ligados às bases distribuídas. Em 2.5 a 2.7 expõe-se conceitos de replicação relacionados aos dados: tipos de distribuição de dados, localização e propagação de dados. Já em 2.8 e 2.9 são expostos os conceitos de sincronização e controle de divergências entre bases de dados. Na seção 2.10 são descritas algumas aplicações encontradas na literatura e citados os principais trabalhos relacionados. Finalmente, em 2.11 são feitos alguns comentários sobre os aspectos apresentados no capítulo.

É importante mencionar que não se encontraram fontes consolidadas sobre os aspectos teóricos relacionados à replicação. Os artigos estudados durante a fase de elaboração dessa dissertação, assim como as referências neles mencionadas, em sua maioria, abordam questões específicas ligadas à replicação. A documentação sobre os bancos de dados que possuem o recurso de replicação inclui na realidade um manual de procedimentos, oferecendo uma teoria básica apenas para orientar o usuário dos sistemas. Este capítulo representa, por isto, um esforço em reunir os diversos aspectos teóricos da replicação a partir de diversas fontes.

2.1 – Sistemas de Banco de Dados

Resumidamente, "Sistemas de Banco de Dados" constitui-se de um conjunto de dados, normalmente chamado de "Banco de Dados" (BD), associados a um conjunto de programas, denominados "Sistemas Gerenciadores de Banco de Dados" (SGBD), que permitem acesso de forma controlada aos dados. Os projetos de SGBD visam manipular um grande volume de informações e garantir segurança, eficiência e disponibilidade. Não faz parte do escopo deste

trabalho aprofundar-se nas questões relativas aos SGBD. De todos os aspectos que configuram o tema, apenas as questões relativas às arquiteturas dos SGBD são as mais relevantes. Maiores detalhes sobre uma visão geral sobre SGBDs pode-se encontrar em [Silber,99-p1] e sobre arquiteturas em [Silber,99-545].

Os sistemas de banco de dados podem ser classificados, de acordo com suas arquiteturas, como:

- Centralizados – quando o SGBD reside em um único sistema computacional e não interage com outros sistemas, podendo ser *monousuário* ou *multiusuário*.
- Cliente-servidor – quando o SGBD reside em um sistema computacional, chamado *servidor*, e interage com outros sistemas computacionais, que são os *clientes*. As funcionalidades são divididas entre o *front-end* (cliente) e o *back-end* (servidor).
- Paralelos – quando utiliza em paralelo diversas *CPUs* e discos, com objetivo de ganhar velocidade no processamento e alta escalabilidade. As conexões dos elementos compartilhados são realizadas através de barramentos ou redes de altíssima velocidade.
- Distribuídos – quando o SGBD reside em diversos sistemas computacionais, podendo estes variar de porte e capacidade. Os SGBD distribuídos (SGBDD) não compartilham memória nem discos como os paralelos e podem ser conectados através de redes de alta ou baixa velocidade.
- Móveis – quando o SGBD e o próprio BD são aplicados em ambientes de computação móveis, que utilizam *notebooks, palm-tops,* celulares e outros equipamentos análogos conectados via redes de computação sem fio (*wireless*). Tais sistemas possuem em geral uma topologia dinâmica e muitas vezes os elementos móveis têm que continuar operacionais mesmo quando desconectados da rede.

A parte que mais interessa no contexto da replicação e sincronização são os conceitos relativos a bancos de dados distribuídos. A próxima seção 2.2 é dedicada à sistematização destes conceitos.

2.2 – Banco de dados distribuído

O banco de dados distribuído tem seus dados armazenados em diversos computadores interconectados, chamados também de *sites* ou *nós*. Os computadores podem variar de tamanho ou de função e possuir uma administração separada. A distribuição física depende de alguns fatores, tais como a arquitetura da rede, se local ou de longa distância, ou se as estações são fixas ou móveis. Resumidamente, o banco de dados distribuído distribui dados.

Esse tipo de banco de dados difere profundamente dos *sistemas centralizados,* porque cada *site* ou *nó* possui um SGBD independente, independência esta extensível até ao tipo de *software* utilizado para implementar o SGBD. Difere também dos sistemas *cliente-servidor* porque não tem a figura do *front-end* e *back-end* (esses elementos podem existir, mas restritos às aplicações processadas no *site*). Todos os *sites* podem ser clientes e servidores ao mesmo tempo e implementam apenas as operações que dizem respeito ao gerenciamento dos dados distribuídos, que são: os controles de transações, consultas e atualizações do banco de dados. Resumidamente, sistemas *cliente-servidor* distribuem funções. Um banco de dados distribuído herda, da arquitetura cliente-servidor, os *gerenciadores de dados* e os *gerenciadores de transações,* que passam a controlar as *transações locais,* restritas ao *site,* e *transações globais,* quando necessitam estar sincronizadas com um ou mais *sites*.

Bancos de dados distribuídos diferem dos *sistemas paralelos* porque não compartilham memória, processadores ou discos. Mesmo quando as arquiteturas paralelas implementam a ausência de compartilhamento, o *software* gerenciador continua sendo único e exige uma rede de alta velocidade para interconectar as diferentes máquinas [Silber,99-p556]. Resumidamente, sistemas *paralelos* distribuem controle. Em sistemas distribuídos, os SGBDs são distintos e a intercomunicação entre máquinas é bem menor.

O que caracteriza um BDD é a existência de dois ou mais *sites*, interconectados de forma permanente ou não, sendo que cada *site* deve possuir um SGBD que compartilha um esquema global comum, embora algumas relações não precisem ser idênticas em todos os *sites*[1]. Em geral, utilizam-se nos *sites* um SGBD projetado para sistemas distribuídos e que, ao trabalhar de forma coordenada entre os *sites*, forma o que se chama de SGBDD.

Os principais motivos para utilizar um BDD são: *compartilhar dados* naturalmente distribuídos, como ocorre, por exemplo, na Internet; proporcionar mais *autonomia*, permitindo aos *sites* gerenciar e manter os dados locais e ainda, se necessário, definir graus de autonomia em cada *site*; *melhorar o desempenho*, já que permitindo um aumento na velocidade de acesso aos dados pode-se aumentar o grau de concorrência; aumentar a *disponibilidade* do sistema distribuído, pois, caso um *site* fique indisponível, os demais podem continuar operando normalmente e, havendo dados replicados, aumenta-se a chance das transações serem bem sucedidas.

As maiores dificuldades no uso desses sistemas estão na complexidade dos programas de gerenciamento e o conseqüente aumento do custo; na maior probabilidade de ocorrer erros (alguns muitos sutis) durante a recuperação de falhas e processos de conciliação; no aumento do tempo de processamento, devido à troca de mensagens usadas na coordenação entre *sites*.

2.3 – Sistema de Banco de Dados Múltiplos

Caso os *sites* participantes do BDD possuam SGBDs diferentes, tais sistemas são chamados de *sistemas de bancos de dados múltiplos (multidatabase)* ou *sistemas de bancos de dados distribuídos heterogêneos.* Para que formem um SGBDD, é necessário o uso de um *software* adicional que fará a interface entre os sistemas heterogêneos e fornecerá uma visão única do BDD. Esta camada de *software* é chamada de *sistema de banco de dados múltiplo,* cuja principal função é "criar uma ilusão de integração lógica do banco de dados sem exigir uma integração física correspondente" [Silber,99 p.624].

Um exemplo de estratégia para construir um banco de dados múltiplos é o FDBS – *federated database system* (sistema de bancos de dados federados). Trata-se de uma coleção integrada das bases de dados autônomas, na qual o administrador mantém um controle sobre todos os sistemas locais, mas coopera com a "federação" suportando operações globais [Shete,1990]. Dividem-se em dois tipos: os fortemente acoplados (*Tightly Coupled*) e os fracamente acoplados (*Loosely Coupled*). Maiores detalhes podem ser encontrados na referência citada acima.

---

[1] *Esquema* e *relação* são conceitos do modelo relacional. *Esquema* está relacionado à estrutura dos arquivos e *relação* ao conteúdo do arquivo, ou seja, suas instâncias. Ver [Silber,99-p63].

O principal incentivo ao uso dos sistemas de banco de dados múltiplos é a integração de sistemas existentes. Qualquer solução de sistemas informatizados, que exija uma integração dos bancos de dados, pode não ser possível se, como pré-condição, exigir a conversão de todos os bancos em uso para um único tipo de banco, deixando assim o BDD homogêneo. Silberschatz [Silber,99 p.624] apresenta dois motivos:

- **Dificuldades técnicas:** alto investimento já feito nos bancos existentes e custos proibitivos para a conversão.
- **Dificuldades organizacionais:** se a solução envolver diferentes corporações, empresas e organizações, estas deveriam converter suas bases preexistentes, o que *politicamente* pode ser impossível.

Por essas razões, os sistemas de BD múltiplos apresentam vantagens significativas que superam seus custos diretos e indiretos.

2.3.1 – Visão Unificada de Dados

É necessário utilizar um modelo de banco de dados comum para criar a ilusão de um único sistema de banco de dados integrado. A escolha natural é o *modelo relacional* tendo em comum a linguagem SQL. Entretanto, isto só não basta: é necessário também ter um modelo conceitual comum, ou seja, as entidades, relacionamentos e atributos devem ter o mesmo significado para todos. No nível semântico, questões como conversões de unidades (polegadas x milímetros, por exemplo), grafia de termos (Brazil x Brasil e outras), codificações locais (um cliente, por exemplo, pode ser identificado por um código arbitrário e não pelo CNPJ), são aspectos que o modelo comum deve levar em conta. Em mais baixo nível, existem questões de representações internas, tais como uso dos códigos ASCII x EBCDIC, representação de pontos flutuantes, dentre outros que também se devem considerar.

O esquema conceitual comum, também chamado aqui de modelo global, deve registrar todas as funções de tradução, em todos os níveis, para que os SGBD locais possam interagir com o modelo global. As consultas e operações globais, portanto, levam em conta que as devidas traduções ocorrem localmente e os resultados são apresentados dentro do formato do modelo global.

2.3.2 – Gerenciamento de Transações e Serialização

Existem diversas propostas para gerenciar transações em bancos de dados múltiplos. Em linhas gerais, em um banco de dados múltiplos, normalmente, as *transações locais* e sua *serialização* ocorrem sem o devido controle do sistema global, mesmo quando algumas destas transações fazem parte de uma transação global. Já as *transações globais* e sua *serialização* são controladas pelo sistema global, ignorando a ordenação que cada sistema local irá adotar. Para que globalmente seja possível gerenciar as transações, utiliza-se um protocolo de dois níveis (2LSR), que estabelece regras para serialização, leitura e escrita para os níveis local e global.

Assim, pode-se perceber que sistemas heterogêneos são muito complexos e foge ao escopo deste trabalho aprofundar-se nesta questão. O principal motivo é que o modelo proposto neste trabalho não é o de um gerenciador de sistemas de banco de dados múltiplos, apesar de

utilizar alguns de seus conceitos. Seu foco é a replicação de dados e o protocolo proposto atende apenas às questões de replicação.

2.4 – A replicação *otimista* em bases de dados distribuídas

Anteriormente tratou-se de aspectos gerais de um SGBDD. Quando não existe a replicação, cada nó ou *site* é responsável por criar e manter a sua própria base de dados. Dessa forma, os *sites* possuem uma grande autonomia e seu papel no sistema distribuído é atender as transações oriundas de outros *sites,* além de permitir que seus usuários acessem informações de outros *sites*. Muitas vezes essa integração é tão transparente ao usuário, que ele tem a impressão de ter acesso a uma "grande" base de dados. O melhor exemplo disso é a *Internet*. Em tese, pode-se ter acesso a qualquer documento que esteja armazenado em qualquer lugar do mundo, dando a sensação ao usuário de ter à sua disposição uma enorme base de dados.

Existindo a replicação, porém, os sistemas adquirem características bem particulares. Nessa seção são tratadas as questões relacionadas a essa situação e direcionadas a um caso ainda mais específico: a *replicação otimista.*

2.4.1 – Replicação otimista vs. replicação pessimista

A principal linha divisória entra a replicação otimista e a pessimista está no *controle de concorrência*. A abordagem pessimista faz o controle de concorrência enquanto a otimista não faz. Os *algoritmos pessimistas,* utilizando quaisquer protocolos para controle de concorrência[2], coordenam de forma *síncrona* as atualizações das bases de dados replicadas e bloqueiam os usuários durante o processo de atualização. Dessa forma mantém o sincronismo entre as réplicas. Já os *algoritmos otimistas*, coordenam essas atualizações em *background,* sem bloquear o acesso aos dados pelos usuários, fazendo *reconciliação* quando ocorre uma situação de conflito. Essa abordagem pressupõe que problemas raramente acontecerão e, caso aconteçam, poderão ser resolvidos em sua maioria com um esforço maior de processamento.

É fácil perceber as conseqüências de cada abordagem na arquitetura do SGBDD. Se existe controle de concorrência *síncrono*, é porque as conexões estarão sempre disponíveis e provavelmente serão rápidas e confiáveis (como em uma LAN). No caso de controle *assíncrono,* processado em *background*, provavelmente as conexões não tem garantias de rapidez (como nas WANs). O bloqueio a usuários fica restrito às operações locais e entra em cena os paradigmas da *reconciliação.*

A *replicação otimista* é definida assim por Saito: "O que é Replicação Otimista? É um grupo de técnicas para um compartilhamento eficiente de dados em áreas de grande abrangência ou em ambientes móveis. O recurso chave que separa os algoritmos de replicação otimista de sua contraparte pessimista é sua abordagem de controle de concorrência. Algoritmos pessimistas coordenam de forma síncrona as réplicas durante o acesso e bloqueiam os demais usuários durante a atualização. Algoritmos otimistas permitem que o dado seja acessado sem ter como condição prévia a sincronização e baseiam-se na hipótese otimista de que problemas somente ocorrerão raramente, se ocorrer. Atualizações são propagados em segundo plano e conflitos ocasionais só serão acertados depois de acontecer" [Saito,2005-p43].

O fato das atualizações de dados ocorrerem em segundo plano, de forma assíncrona, traz o conceito de *operação*. Esse termo é usado por Saito para significar um evento de atualização de dados ocorrido em qualquer *site* e suas conseqüências: "Operações diferem das atualizações tradicionais das bases de dados (transação) porque são propagadas e aplicadas em segundo plano, muitas vezes bem depois de ter sido submetida pelo usuário. Conceitualmente, uma operação pode

[2] Os gerenciadores de transação administram *transações locais* e *globais* e os controles de concorrência e recuperação provavelmente utilizarão protocolos que irão garantir as propriedades ACID (atomicidade, consistência, isolamento e durabilidade), tais como o *2PC* ou *3PC*.

ser vista como uma *precondição* para detectar conflitos, combinado com a prescrição para atualizar o objeto" [Saito,2005-p45].

2.4.2 – Replicação otimista como base para um SGBDD

Após o estudo de diversos artigos e livros, com destaque aos trabalhos de [Saito,2005], [Cunha, 2003], [Saito,2002] e [Silber,99], foi possível identificar os principais conceitos relacionados à replicação de dados e quais procedimentos tornam-se mais relevantes:

- Controle de divergência: determina critérios que vão melhorar a confiabilidade da réplica e assinalar os momentos de iniciar a propagação.
- Propagação: define as formas de propagar os dados e as operações, conforme a topologia de comunicação e a freqüência das atualizações.
- Localização dos dados: define o *endereçamento* dos dados, garantindo a sua perfeita identificação e localização.
- Distribuição dos dados: define os critérios que determinam quais dados serão distribuídos (visões dos dados) e como serão distribuídos (*fragmentados*, *replicados* ou ambos) no BDD.
- Sincronização: define critérios de como as transações são manipuladas. Consiste basicamente de três fases: *escalonamento* (*scheduling*), *conciliação (resolução de conflitos)* e *efetivação* (*commitment*).

Além dos procedimentos citados, foram identificados como relevantes os seguintes elementos estruturais relacionados à replicação de dados:

- Réplica de um BD ou fragmento replicado de um BD: é o conjunto de dados submetidos ao procedimento de replicação que podem existir em um ou mais *sites*.
- Domínio da replicação: é o conjunto de *sites* onde os dados replicados coexistem. Muitas vezes chamado também de ambiente de replicação.
- Escala das operações (*schedule*): é um registro das operações locais e globais submetidas pelas *aplicações locais* que são processadas em cada *site* participante do domínio da replicação. Utilizada principalmente pelos procedimentos de propagação, sincronização e controle de divergência.

Os procedimentos relacionados à replicação relacionam-se em uma seqüência lógica. Para dar uma visão geral do relacionamento entre esses procedimentos, elaborou-se a figura 2.1 apresentada a seguir:

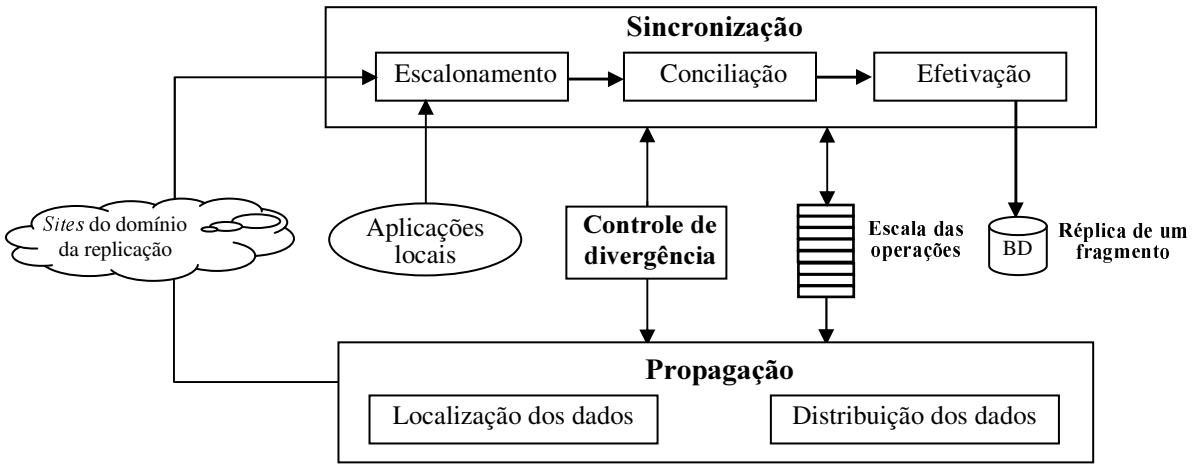


Figura 2.1 – Seqüência de procedimentos que ocorrem em uma base de dados distribuída com replicação

Na figura 2.1, o acesso das aplicações locais ao BD replicado são monitorados pela *sincronização*, que utiliza os critérios do *controle de divergência* e é responsável pelo *escalonamento*, *conciliação* e *efetivação* dos dados. Tudo que ocorre no BD fica registrado na *escala de operações.* Havendo necessidade, o *controle de divergência* também pode forçar a *propagação* de dados. A *propagação*, por sua vez, precisa de informações sobre a *localização dos dados* e *distribuição dos dados*, para se conectar com os *sites* pertencentes ao domínio da replicação e repassar aos demais as operações indicadas pela *escala das operações*. As operações recebidas pelo *site* passam pelo processo de *sincronização* e, ao final, podem ser efetivadas ou não. Nas próximas seções, cada um dos procedimentos citados são um pouco mais detalhados.

2.5 – Distribuição dos dados

Nessa seção são tratados os aspectos relacionados à distribuição dos dados ao longo dos *sites* participantes de um domínio de BDD. Os principais enfoques são a *fragmentação* (horizontal, vertical e mista) e como os modelos de *replicação* afetam a distribuição dos dados.

2.5.1 – Tipos de fragmentação de dados

A *fragmentação*, orientada pelo conteúdo dos dados, define quais dados serão distribuídos e a maneira como isso é feito. Ela divide o banco de dados em várias partes e distribui essas partes entre os *sites* participantes do BDD. A fragmentação pode ser *horizontal, vertical,* ou *mista*, conforme mostrado na figura 2.2.

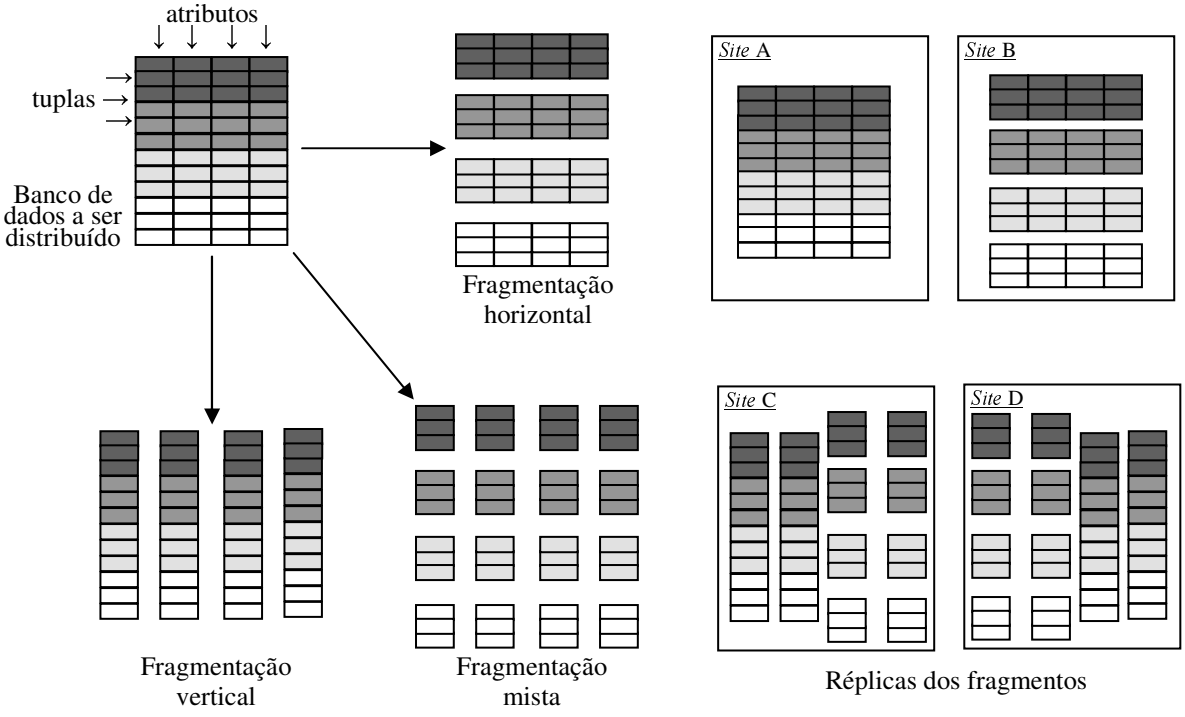


Figura 2.2 – Tipos de fragmentação e replicação de um BDD.

Na fragmentação *horizontal*, o BDD é divido em conjuntos de tuplas (ou registros), cabendo a cada site participante um desses conjuntos que formam um BD. Na fragmentação *vertical* ocorre o mesmo, porém, em relação a um conjunto de atributos (ou campos). Usando as duas simultaneamente, tem-se a fragmentação *mista.*

Quando um mesmo conjunto de tuplas ou atributos aparece repetido em dois ou mais BDs, é porque houve a *replicação do dado*. A réplica do fragmento, pode também ser fragmentada e replicada, e assim sucessivamente. Caso o BDD não seja fragmentado, mas apenas replicado, tem-se então a *replicação total*[3]. [Silber,99-p590].

2.5.2 – Transparência de rede

Transparência de rede é a capacidade que o sistema tem de "esconder" do usuário detalhes da distribuição. Nenhum usuário do sistema ou desenvolvedor de aplicações deve se preocupar com questões relativas à forma como os dados foram fragmentados e replicados, como são identificados, nem onde estão localizados. O sistema deve ser devidamente configurado para fornecer todos os serviços de consulta e atualizações, alcançando os dados onde esses estiverem.

Em sistemas com desconexão programada, alguns dos objetivos citados relativos à transparência de rede são muito difíceis de se conseguir, se não for impossível, dependendo da situação. Um deles se refere à coerência entre consulta e atualizações. Por exemplo, um usuário desconectado pode fazer uma consulta em sua base de dados, enquanto outro *site* atualiza a réplica dessa base. Quando houver a conciliação, o resultado da mesma consulta seria outro e o usuário pode ter ficado com uma informação ultrapassada. Dependendo da situação, isso pode não ser tão grave, levando em conta que mesmo em um ambiente "conectado", o usuário poderia fazer a consulta um pouco antes de outro *site* alterar o dado e, assim, ficaria com a informação desatualizada da mesma forma. Em casos críticos, o ideal é ter mecanismos que avisem ao usuário que a informação consultada mudou.

Outro objetivo, que pode não ser atingido em sistemas com desconexão programada, refere-se às consultas globais. Em ambientes conectados, o sistema submete a consulta a todos os *sites*, faz a união dos resultados e os apresenta ao usuário. Em ambientes desconectados isso é impossível de ser feito. A única alternativa, nem sempre viável, é ter disponível uma réplica completa das bases de dados. Mais uma vez, em casos críticos, o ideal é dispor de mecanismos que avisem ao usuário que a informação pode estar incompleta ou listar os *sites* não consultados.

2.5.3 – Modelos de replicação

Existem três modelos básicos que implementam as funcionalidades da replicação: *mestre-escravo*, *cliente-servidor* e *peer-to-peer* [Ratner,98-p18]. Como é visto a seguir, todos esses modelos afetam diretamente os aspectos ligados à propagação de dados.

---

[3] Deve-se lembrar que cópias *backup* estão fora desse contexto, pois em geral não são consideradas cópias para uso operacional. Fica de fora, também, a técnica de espelhamento de discos rígidos (*mirror*), porque essencialmente são cópias em tempo real, cujo objetivo é aproximar a disponibilidade de servidores a quase 100% do tempo.

No modelo *mestre-escravo*, a replicação nos escravos é sempre idêntica à base de dados do mestre. O modelo é muito simples, de fácil implementação, porém, de uso limitado. Os escravos são de leitura somente e as atualizações são feitas apenas no mestre.

No modelo *cliente-servidor,* o servidor atende a múltiplos clientes. Entretanto, a capacidade dos clientes pode implementar muitas funcionalidades de atualização, que serão depois sincronizadas com o servidor. A maior limitação desse modelo é que os clientes não podem comunicar-se uns com os outros e sincronizar suas respectivas bases de dados. Caso algum cliente precise passar uma informação a outro, é necessário que o cliente origem reporte-se ao servidor e, depois, o cliente destino recupere essa informação do servidor, mesmo que fisicamente estejam um ao lado do outro.

No modelo *peer-to-peer*, não há hierarquias como nos modelos anteriores. Todas as réplicas possuem o mesmo nível. Qualquer par de *sites* pode se comunicar e sincronizar suas bases de dados. A grande vantagem desse modelo é a flexibilidade. A grande desvantagem está na escalabilidade, pois, com o aumento do número de *sites*, aumentam as dificuldades para manter todas as réplicas sincronizadas.

O modelo WARD (*Wide Área Replication Domain*) é um híbrido dos modelos *cliente-servidor* e *peer-to-peer* [Ratner,98-p41]. Basicamente, cada domínio – um conjunto de *sites* próximos geograficamente – usa o modelo *peer-to-peer* para replicação de dados. Um dos *sites* é então eleito para ser o *master* (responsabilidade de servidor) e representar o domínio perante os outros para fins de sincronismo. Essa eleição pode ser dinâmica, alterando o *site* mestre conforme a necessidade. Não se deve confundir esse modelo com a técnica *two-tier replication*, descrita na seção 2.6.2, onde se conectam os servidores de vários conjuntos clientes-servidor.

2.6 – Localização dos dados

Uma vez fragmentado, replicado e distribuído, o sistema gerenciador do banco de dados deve possuir um mecanismo para nomear e localizar os dados de forma *transparente* às aplicações. Duas técnicas são descritas em [Silber,99-p595]: *servidor de nomes* central e *prefixos identificadores* para cada *site*.

Em um *servidor de nomes,* os nomes são registrados com sua respectiva localização em um *site* central. A grande desvantagem desse esquema é que o servidor de nomes pode se transformar em um ponto de estrangulamento e, se ficar inoperante, compromete todo o sistema.

Com *prefixos identificadores,* cada *site* recebe um nome único no sistema. Para que isto fique transparente às aplicações, tais nomes são tratados como *aliases* (nome alternativo), que o sistema gerenciador converte para o nome completo. Por exemplo, para o item de dado *Conta corrente,* o *alias* é [Conta_Corrente] e o nome completo é [site17.Conta_Corrente.f3.r2], significando que Conta_Corrente está no *site17*, *fragmento 3, réplica 2*.

Em ambos os casos, o sistema gerenciador tem que ter meios de saber a qual fragmento ou réplica o usuário está se referindo. Se ele pede informações, por exemplo, sobre uma *Conta_Corrente,* todos os *sites* devem ter tabelas indicando que tal [Conta_Corrente] (ou a faixa em que esse dado está inserido) pode ser encontrado no item [site17.Conta_Corrente.f3.r2].

As atualizações de informações seguem os mesmos princípios. Transparente para o usuário, o sistema cuida de localizar o fragmento correto, atualizá-lo e propagar a atualização para todas as réplicas.

Para garantir um mínimo de *overhead,* o projeto de fragmentação e replicação deve levar a informação para onde existe a maior probabilidade dela ser usada. Mesmo assim, o esquema de transparência adotado tem que permitir consultas globais. Por exemplo, caso uma aplicação solicite ao SGBDD uma "relação de *todos os clientes* cadastrados no ano de 2006", o gerenciador tem que enviar uma mensagem de consulta a todos os *sites* que possuem *fragmentos* do cadastro de clientes, fazer depois uma junção das relações retornadas e encaminhar à aplicação o resultado final da solicitação.

No caso específico de *sites* desconectados por longos períodos de tempo, a transparência pode continuar existindo, desde que os processos de reconciliação ocorram sem interferência do usuário. Já as consultas globais devem ser avaliadas do ponto de vista das aplicações. Sendo imprescindíveis, a única alternativa é replicar em cada *site* todo o conteúdo das bases de dados, ou, pelo menos, todos os fragmentos necessários às aplicações. Neste caso, a *localização dos dados* para fins de consultas e atualizações fica muito mais simples, pois a base, que é replicada, fica no próprio site.

2.7 – Propagação de dados e operações

A propagação é uma das questões mais críticas nos SGBDD. As informações geradas em cada *site* devem ser propagadas aos demais *sites,* para que todo o BDD permaneça íntegro, tanto do ponto de vista da fragmentação (cada tupla no seu devido lugar) como da replicação (todas as cópias permanecerem idênticas). Tais informações podem ser *dados* (tuplas ou parte delas), *operações* (transações efetuadas no BD) ou ambos. A propagação é uma questão estratégica para o sucesso de qualquer projeto de replicação e depende de diversos fatores. Esses fatores são apresentados a seguir.

2.7.1 – Topologia de comunicação

A primeira questão importante para a propagação está relacionada à *topologia de comunicação*. Os tipos de topologia mais usuais são apresentados na figura 2.3. A simples visão das topologias já dá uma idéia das responsabilidades que cabem aos *sites* relativas aos mecanismos de propagação. Deve-se lembrar que a situação mais exigente é aquela em que todos os *sites* possuem a mesma réplica de uma base de dados. Assim, em (1) e (2), o *site A* é o principal responsável pela propagação; se ele falhar, muitos *sites* ficarão com suas réplicas desatualizadas. Em (5), a responsabilidade maior fica dividido entre *A* e *B*. Em (3) e (4), todos os *sites* são responsáveis pela propagação. A diferença é que em (3), em virtude da topologia em anel, um determinado *site* precisa da participação dos outros na propagação e recepção de informações; em (4), cada *site* pode ser o único responsável pela propagação de suas informações e pode ficar aberto à recepção de qualquer outro *site*.

Dá-se o nome *de propagação epidêmica* ao mecanismo através do qual um *site* qualquer propaga para todos os demais *sites* as próprias informações, além daquelas que recebeu de outros *sites* [Cunha, 2003-p47]. Esse mecanismo é uma analogia à propagação de um vírus biológico, onde todas as pessoas são potencialmente transmissoras e receptoras. Para implementar esse mecanismo, evidentemente a topologia mais adequada é a (4).

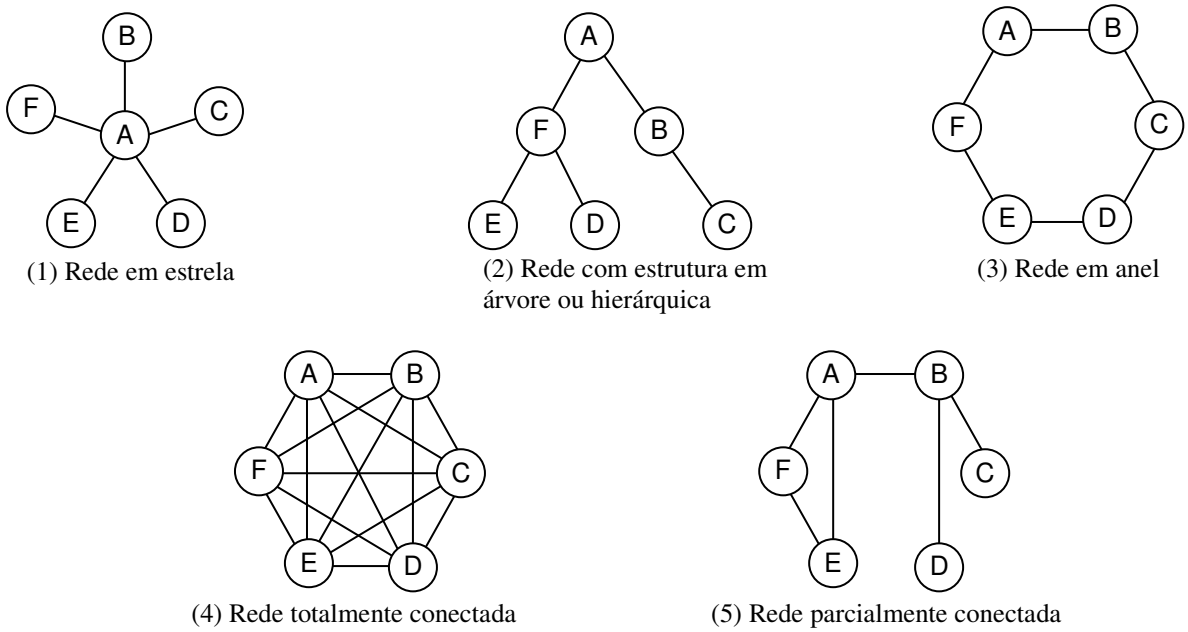


Figura 2.3 – Topologias de comunicação mais usuais

2.7.2 – Número de escritores: mestre e multimestres

Quando apenas um *site* é responsável pelas atualizações, diz-se que este é um sistema *mestre* e os *sites* que recebem as réplicas do mestre, são os *escravos*. Em outras palavras, esse arranjo caracteriza um sistema *mestre-escravo ou* sistema *caching.* Quando mais de um *site* pode fazer atualizações, diz-se que é um sistema *multimestre*.

Não há dúvidas de que sistemas *multimestre* proporcionam maior disponibilidade dos dados, porém, são bem mais complexos. Os problemas de escalonamento e resolução de conflitos só existem nesses sistemas, que também potencialmente apresentam sérios problemas de escalabilidade. O maior objetivo a atingir nos sistemas *multimestre* é a *convergência* dos dados, ou seja, conseguir que todas as réplicas fiquem idênticas. Em sistemas replicados, o termo *convergência* é usado, porque em um determinado instante de tempo as réplicas podem apresentar diferenças, porém, com o *passar do tempo*, elas irão convergir e ficar iguais.

No que diz respeito à escalabilidade, quanto maior o número de escritores, maior a possibilidade de ocorrer conflitos, caso todos os *sites* atualizem e troquem informações relativas a uma mesma operação de forma constante e uniforme. Entretanto, na prática isso não acontece. Estudos mostram que atualizações simultâneas são pouco freqüentes na maioria das aplicações [Saito,2005-p69]. Além disso, a escolha adequada da topologia de comunicação e um controle mais pró-ativo das informações (*proactively controlling*) podem aumentar a velocidade da propagação e reduzir conflitos.

Uma técnica conhecida como *clearinghouse (centro de conciliação)*[4] adota um *site mestre* para fazer uma conciliação geral das atualizações que ocorreram em outros *sites*. Com isso, garante-se a *convergência* dos dados e os conflitos são reportados aos *sites* respectivos [Saito,2005-p69].

---

[4] Por se tratar de um termo muito comum, não confundir com nomes de projetos envolvendo bancos de dados e até mesmo com o uso corrente do termo, que também significa *câmara de compensação* (bancária).

Outra técnica é dividir o sistema replicado em duas camadas (*two-tier replication)*. Uma, sempre conectada, apresenta topologia em anel ou totalmente conectada e congrega os chamados *sites base*. A outra, eventualmente conectada, inclui os *sites móveis* [Gray, 1996]. Os *sites base* trabalham com algoritmos pessimistas, o que garante a integridade de todas as réplicas. Cada *site móvel* comunica-se apenas com *um site base,* o qual recebe atualizações de todos os *sites móveis* sob sua responsabilidade e realiza os procedimentos de conciliação. Cada *site base* então retorna para todos *sites móveis* sob sua tutela as efetivações definitivas ou as rejeições. Percebe-se que essa técnica é uma extensão da topologia estrela, onde os *sites centrais* de cada estrela formam a camada dos *sites base*. Essa técnica melhora a escalabilidade, mas sacrifica a flexibilidade da comunicação. A figura 2.4 mostra as duas camadas mencionadas anteriormente: a dos *sites base* e a dos *sites móveis*, de acordo com o arranjo da técnica *two-tier replication.*

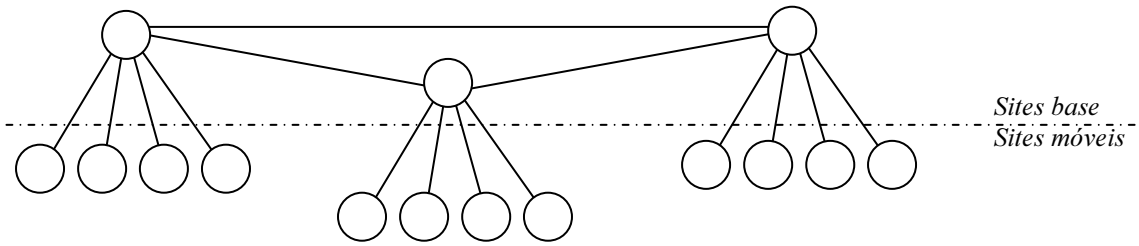


Figura 2.4 – *Two-tier replication*

Diversas soluções podem ser identificadas combinando topologia e escritores. Redes móveis *ad-hoc* (*MANET- Mobile Ad-Hoc Networks*), por exemplo, podem replicar dados apenas naqueles nós que probabilisticamente estão mais sujeitos à desconexão [Gianuzzi,2004]. Outro exemplo é alterar dinamicamente o esquema de objetos replicados, com base em algoritmos que avaliam os padrões de leitura e escrita a que os objetos estão sujeitos, padrões estes que podem mudar continuamente [Wolfson,1997]. Resumindo: o que se procura é combinar da melhor forma possível *velocidade de propagação*, *balanço de carga* e *disponibilidade dos dados*.

2.7.3 – Técnicas de propagação

Existem três tipos de técnicas para iniciar uma propagação: *pull-based, push-based* e *híbrida* [Cunha, 2003-p53].

O *pull-based* é a técnica mais simples. Ela funciona por demanda. A qualquer momento e por qualquer motivo, um *site* toma a iniciativa de entrar em contato com outros para efetivar a sincronização. Esse mecanismo é típico em soluções, onde a iniciativa de estabelecer uma nova conexão é determinada pelo usuário ou quando se utiliza um *polling* ocasional.

O *push-based* já é uma técnica mais dinâmica. Nesse esquema, um *site* toma a iniciativa de comunicar-se imediatamente com os demais, logo quando surge uma atualização. Essa comunicação pode ser individual para cada *site* ou em modo *broadcasting*. Essa atitude mais ativa reduz o tempo de propagação e elimina a necessidade e o custo do *polling*. Pode-se concluir que, em ambientes com características de trabalho *off-line*, essa técnica pode não ser adequada, porque haverá a possibilidade dos *sites* não receberem as mensagens de difusão no momento em que estiverem desconectados.

A técnica *híbrida* utiliza os dois tipos anteriores. Em muitos ambientes móveis, o canal de conexão é utilizado tanto para difusão de mensagens como para requisições. A mesma advertência feita no caso de *push-based,* para ambientes com características de trabalho *off-line,* também vale para a técnica híbrida.

Existem diversas técnicas relacionadas ao processo *push-based* [Saito,2005-p70]. Uma delas é o *blind flooding* (que pode ser traduzido como *inundação cega*), onde o *site* propaga e recebe atualizações de todos os seus vizinhos (chamada também de *propagação epidêmica* – ver seção 2.6.1). A grande desvantagem dessa técnica é que poderá haver um grande número de transmissões desnecessárias. Esse problema pode ser minimizado, quando há um controle permitindo ao *site* saber se determinada operação já foi anteriormente transmitida para outro *site*. Uma das formas de realizar esse controle, detalhada a seguir, é monitorando os enlaces entre os *sites*.

Para evitar o envio desnecessário de operações no caso do *blind flooding* explicado anteriormente, pode-se utilizar a técnica de monitorar o estado dos enlaces (*link-state monitoring*). Isso pode ser feito de duas maneiras. Em uma delas, o *site* monitora o número de duplicatas recebidas e transmitidas, parando de transmitir quando chega a um determinado valor. A outra é monitorando o par origem-destino (*path*) das operações, não enviando mais aquelas que já passaram pelo *site*. Ambas as formas utilizam heurísticas, o que pode ocasionar, por um lado, um longo tempo com propagações desnecessárias, ou, se os parâmetros não forem adequados, não atingir 100% dos *sites*.

Outra técnica *push-based* prevê a utilização de um protocolo *multicast*. O maior problema nesse enfoque é garantir que todos os *sites* recebam as operações, pois eventualmente, podem existir *hosts* inalcançáveis. Algumas soluções adotam o *multicast* em um primeiro momento e depois o *flooding* tradicional, configurado para trabalhar de forma mais lenta, porém garantindo que todos os *sites* serão atingidos.

A técnica *push-based* mais eficiente e, por outro lado, mais complexa, é a que utiliza *matrizes de timestamp*. Como se mostra adiante, essas matrizes são utilizadas para controlar o escalonamento (*scheduling*) das operações. A propagação pode aproveitar esse controle já existente, para determinar quais *sites* ainda não receberam determinadas operações e, com isso, otimizar bastante o mecanismo *push-based*.

A propagação em si, independente da técnica, também pode ser aperfeiçoada através de algoritmos que procuram desempenho e escalabilidade. Um exemplo é a ferramenta *nsync*, que através de parâmetros que caracterizam o ambiente de replicação, faz um planejamento da sincronização entre os *sites* e depois executa este mesmo plano [Schütt,2003].

Nas próximas seções, os conceitos de *timestamp* (*registro do tempo*) e *clock* (*relógio*) são bastante utilizados. Ambos os conceitos são definidos como contadores lógicos, que registram uma sucessão de eventos, e aqui usados indistintamente. Quando algum deles se referir ao tempo real, isso será devidamente advertido.

2.7.4 – Tipos de propagação

Existem dois tipos de propagação de dados: por *transferência de operação* e por *transferência de estado*. Numa propagação por *transferência de operação* são transferidas apenas as operações (ou transações) efetuadas sobre o objeto. Já a *transferência de estado* propaga todo o objeto ou parte dele.
Quando se propaga apenas os dados, os algoritmos de reconciliação obrigatoriamente utilizam apenas *dados* e assim não podem, na maioria das vezes, reconstituir os eventos ocorridos com os dados. Já quando se propaga operações, os algoritmos de reconciliação podem analisar o que foi aplicado sobre os dados e, se for o caso, modificar as operações antes de efetivá-las.

Exemplo: seja um *Site A* que possui um atributo com valor *Va=70* e um Site B com *Vb=40.* No momento da reconciliação, os algoritmos de *transferência de estado* só têm como informação objetiva que o estado de *V* foi alterado. Esses algoritmos, portanto, terão que decidir qual dos dois prevalecerá, com base em critérios não diretamente relacionados com os dados. Na *transferência de operação* é possível saber por que mudaram, como por exemplo, [*Va = 50+20 = 70*] e [*Vb = 0,8 x 50 =40*]. Observe que, neste caso, tem-se duas soluções possíveis [V = 0,8 x (50+20) = 56] ou [V = (0,8 x 50) + 20 = 60] (que poderiam ser definidos por critério de corretude), ambas diferentes do que a *transferência por estado* concluiria. Observa-se, então, que a transferência por operação oferece aos algoritmos de sincronização uma quantidade maior de informações objetivas sobre os dados.

É evidente que *transferência de estado* é bem mais simples do que transferir operações, porém, essa primeira abordagem pode se tornar ineficiente no caso de objetos grandes, além de não possibilitar muitas alternativas para reconciliar informações.
Na próxima seção, apresenta-se uma forma de controlar a propagação de operações e, em seguida, algumas técnicas para otimizar a propagação por transferência de estado.

2.7.4.1 – Propagação com transferência de operação usando *vector clocks*

Na técnica de propagação com *vector clocks,* cada *site* mantém um vetor, onde cada elemento representa o número de operações efetuadas em cada *site* da base distribuída. Por exemplo, para uma base com *M* réplicas, um determinado *site i* terá um vetor $VC_i[1..M]$ com o número de operações efetivadas em cada *site*. O *site k* terá um vetor $VC_k[1..M]$ e assim por diante. A cada operação em *i*, $VC_i[i]=VC_i[i]+1$ e em *k,* $VC_k[k]=VC_k[k]+1$. Quando chega o momento da reconciliação, os *sites* trocam os vetores entre si, de forma que é possível, através de comparações, cada *site* descobrir quais operações devem ser propagadas e para quais *sites*. No site *i,* por exemplo, se $VC_i[i] > VC_k[i]$ (onde VC_k é o vetor recebido do *site k*), então deve-se propagar as operações maiores que $VC_k[i]$ para o *site k*. Por outro lado, se no site *k,* $VC_k[k]=VC_i[k]$, então não se deve propagar nada para o site *i.* Uma alternativa ao contador, é utilizar o relógio em tempo real.

O *vector clocks* é utilizado também no escalonamento de operações, como será visto na seção 2.8.1.

2.7.4.2 – Propagação com transferência de estado

Na propagação com transferência de estado, essencialmente, quando o objeto é alterado, ele é propagado por inteiro a todas as réplicas. Cada site mantém um *timestamp* dos seus objetos. No momento da reconciliação, é transmitido o *timestamp.* Caso algum *site* verifique que o *timestamp* de sua réplica é anterior ao que recebeu, solicita uma transferência do objeto. A forma mais comum para realizar esse procedimento é usando a regra de escrita de Thomas (*Thomas write rule*) [Saito,2005-p61]. Essa regra simplesmente considera como correta a

última versão do objeto com base no *timestamp* e abstém-se de qualquer mecanismo de conciliação.

O algoritmo *Two-Timestamp* é um aperfeiçoamento do descrito anteriormente. Cada *site* mantém dois *timestamps* para cada objeto: um indica quando houve a última reconciliação e o outro é incrementado a cada atualização local. Quando ocorre uma reconciliação, são comparados os dois *timestamps* locais com os dois remotos. Isto possibilita identificar se houve ou não alterações simultâneas em ambos os *sites.* Não havendo, identifica-se quem deve ser atualizado. Havendo, identifica-se uma situação de conflito.

Evidentemente, essa é uma estratégia ineficiente no caso de objetos muito grandes. Existem outras técnicas, descritas a seguir, para melhorar a eficiência da propagação.

Transferência híbrida estado-operação: essa solução mantém um histórico de todas as operações efetuadas no objeto com os respectivos *timestamps* desde a última sincronização, ou seja, mantém informações apenas sobre as diferenças. Quando chega o momento da reconciliação, apenas essas diferenças em relação ao outro *site* são propagadas. Caso a réplica seja muito antiga ou não exista *timestamp*, o objeto completo é enviado.

Algoritmo do bit-modificado: é uma simplificação da técnica anterior, onde apenas assinala-se que o objeto foi modificado e, por conseqüência, transmitido quando houver a reconciliação.

Divisão hierárquica do objeto e comparação: é uma solução quase natural para *sistemas de arquivos* replicados. O objeto é dividido em sub-objetos em uma estrutura de árvore. Cada nó da árvore mantém um *timestamp* dos filhos imediatos. Quando, durante a reconciliação, o *timestamp* difere, é feito o sincronismo usando a regra de escrita de *Thomas* e a alteração é passada para os filhos e assim sucessivamente.

Collision-Resistant Hash Function: é uma solução ideal para objetos que não tem uma estrutura definida, tal como grandes objetos binários. Assim, o objeto é dividido em pequenos blocos fixos e transmite-se para outro *site* apenas o valor de uma função *hash* de cada bloco. O *site* receptor verifica quais blocos de sua réplica não produzem o mesmo valor da função *hash* e solicita ao transmissor apenas esse blocos. Quando não há mudança no tamanho do objeto (por exemplo, uma imagem extensão *bmp*) essa solução funciona bem, mas torna-se ineficiente quando ocorre inclusão ou deleção de bits (como numa imagem compactada tipo extensão *jpg*). Para resolver isto se utiliza o *algoritmo rsync* [Suel,2003], esquematizado na figura 2.5, retirada da mesma fonte.

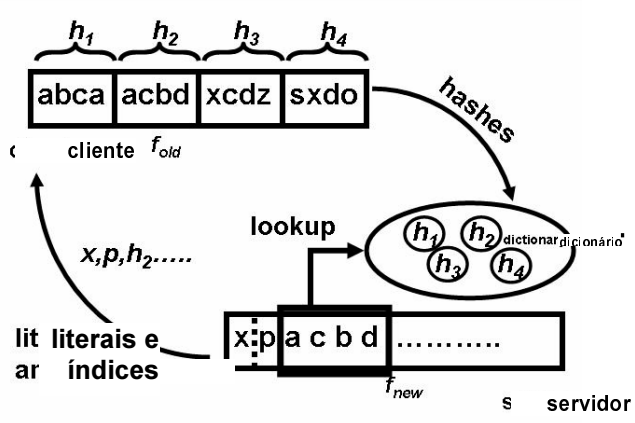


Figura 2.5 – Algoritmo *rsync*

Esse algoritmo também prevê a divisão das duas cópias em blocos de tamanho fixo. O *site* emissor (*cliente*) transmite então os resultados das duas funções *hash* (*dicionário: h1, h2, h3...*) por cada bloco. O receptor (*servidor*) compara esses resultados com os calculados localmente (*lookup*). Caso os resultados de baixo e alto custo coincidirem, o algoritmo considera que os dados comparados são idênticos ($f_{new}=f_{old}$) e prossegue com a verificação do próximo bloco. Esse algoritmo, em caso de não haver diferenças entre as réplicas, prova ser bastante eficiente na verificação, chegando rapidamente ao final dos objetos. Caso os resultados da função *hash* não coincidirem (f_{new} *xpac* $\neq h_1,h_2...$), então o receptor das funções *hash* passa a utilizar um mecanismo de "janela deslizante", no qual ele desliza *byte* a *byte* o bloco do receptor (*xpac*, *pacb*...), até encontrar uma janela que produza resultados coincidentes (*acbd*). A seqüência de *bytes* "deslizados" que provocou as diferenças (*x,p*), acrescidos dos resultados corretos da função *hash* (*h2*...) são transmitidos para o emissor, que assim consegue ajustar a sua réplica. Esse algoritmo impede que uma seqüência de bytes inserida ou removida de uma cópia provoque o envio de todo o objeto.

Set-Reconcilliation Approach: esse procedimento é usado em casos muito específicos, quando o tipo de objeto permite que ele seja transformado em um conjunto de bits de tamanho fixo usando, por exemplo, funções *hash*. São aplicadas funções polinomiais nesse conjunto de bits, transmitindo-se apenas o resultado da função. O receptor resolve então a equação aplicada pela função polinomial para chegar ao valor original associado ao conjunto de bits e verifica as diferenças em relação ao conjunto de bits representativo de sua réplica.

Um caso particular deve ser tratado quando se *elimina um objeto*. Como o princípio básico da transferência de estado é propagar o objeto, eliminar um objeto cria a situação contraditória de se propagar um "objeto inexistente". A solução para isso é usar o chamado "atestado de óbito" (*death certification*) ou a "lápide" (*tombstones*), que propaga o objeto vazio, porém acompanhado do controle que o identifica (um *timestamp,* por exemplo).

Finalmente, os algoritmos que comparam o estado atual de um objeto com o estado anterior do mesmo objeto, com o objetivo de fazer a reconciliação, são nomeados genericamente de *two-way merging algorithm*. Em muitos casos, porém, é possível, e às vezes necessário, comparar o objeto anterior com duas novas versões do mesmo objeto. Tais algoritmos são nomeados de *three-way merging algorithm* e, de fato, apresentam um grau maior de flexibilidade e contam com mais recursos para garantir a corretude da reconciliação. A proposta de Lindholm [Lindholm,2003] é um exemplo de trabalho que se baseia neste princípio, utilizando o XML como um meio para criar um protocolo genérico de reconciliação em ambientes de computação móvel.

A figura 2.6 mostra um resumo das abordagens descritas nessa seção e permite avaliar a importância de alguns critérios que caracterizam a replicação de dados.

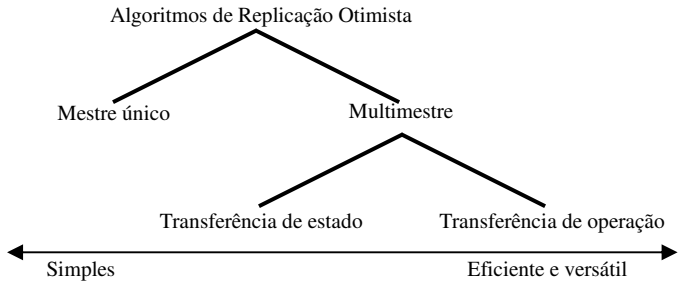


Figura 2.6 – Taxionomia da replicação otimista [Saito,2002-p9]

2.8 – Sincronização

Sincronização é um processo que transforma em cópias idênticas as cópias divergentes de qualquer objeto. Além disso, uma sincronização correta obriga que a convergência das cópias resulte em um estado coerente com as propriedades predefinidas do objeto [Molli,2003-p212]. Em outras palavras, após a sincronização, as cópias devem estar idênticas e corretas.

Manter réplicas idênticas e corretas é um objetivo nem sempre fácil de atingir. Em muitas situações, quando os algoritmos não resolvem determinados conflitos, o sistema precisa reportar-se ao usuário, solicitando que ele resolva o conflito. Não sendo possível reportar-se ao usuário, só resta ao sistema manter as réplicas divergentes e registrar a ocorrência desse evento, até que o administrador ou usuário possam solucionar o conflito.

Quando a absoluta correção simultânea não é um requisito fundamental, muitos sistemas utilizam critérios mais flexíveis, que proporcionam maior rapidez de processamento e são menos complexos. Parte-se do princípio de que existem meios simples e fáceis de corrigir eventuais divergências ou, simplesmente, que elas deixarão de existir na medida em que novas operações ocorrerem. Isso será visto nas próximas seções.

Foi elaborado, para esse trabalho, um esquema do processo de sincronização, apresentado na figura 2.7, baseado nos artigos de Saito sobre o assunto [Saito,2005][Saito,2002]. Esse esquema representa um "cenário" existente em cada *site* no momento da sincronização. Nesse cenário o *site* possui:

- Um conjunto de objetos que são réplicas de outros sites
- Um conjunto de operações controladas localmente, sendo que algumas já foram aplicadas aos objetos, outras estão em processo de propagação e as restantes já foram propagadas.
- Um conjunto de operações recebidas de outros *sites* e que irão passar pelo processo de sincronização.

Entenda-se por *operação* um conjunto formado pelas operações propriamente ditas e pelos controles anexados às operações tais como *timestamp* ou *vector clocks*.

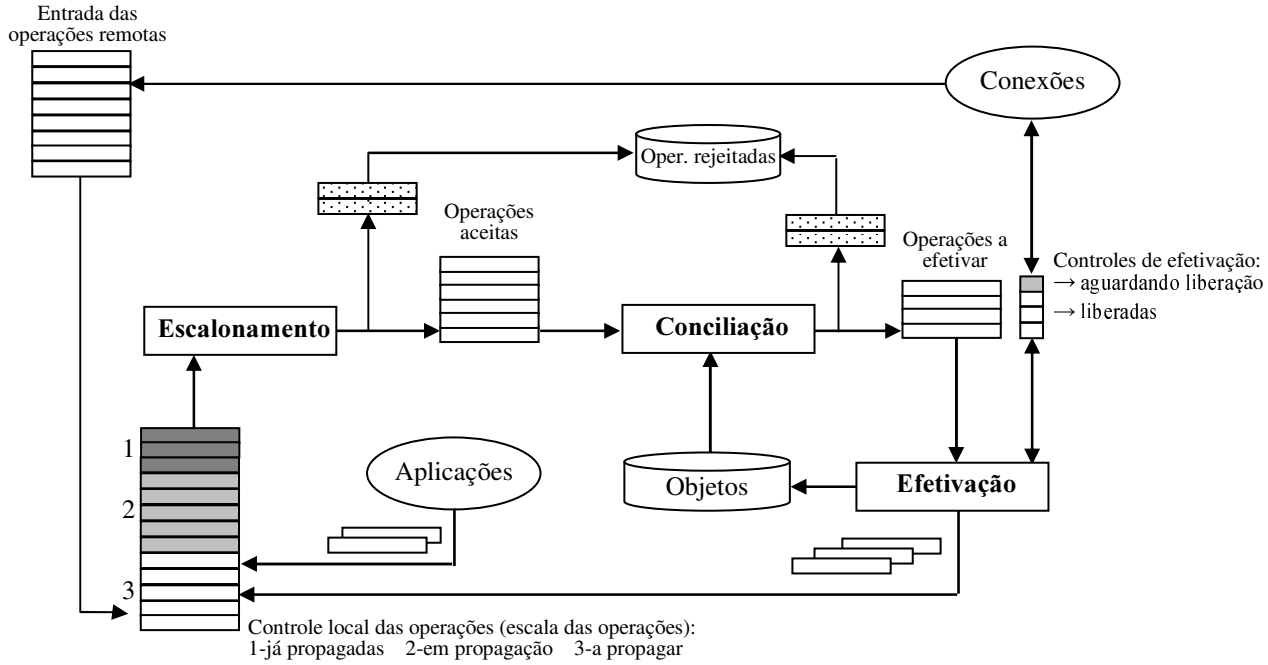


Figura 2.7 – Esquema funcional da sincronização

A figura 2.7 dá uma noção dos aspectos funcionais de um processo de sincronismo. O ***escalonamento,*** a partir da entrada das operações remotas e locais, descarta as operações já aplicadas ou inconsistentes (neste último caso, necessário depois utilizar as técnicas de resolução de conflitos); assinala no controle local das operações aceitas o novo estado que cada uma adquire após a análise das informações que recebeu dos demais *sites*; refaz a "fila" de operações com base em critérios de escalonamento semântico e passa o controle para a *conciliação*. Já a ***conciliação*** mais uma vez refaz a "fila", consultando os objetos locais se necessário e descarta (ou reserva para uso posterior) operações conflitantes. Finalmente, a ***efetivação*** irá atualizar os objetos e pode, dependendo da solução, combinar com os demais *sites* o momento da efetivação. As operações aplicadas aos objetos pela efetivação ou pelas aplicações, são assinaladas no controle das operações locais. O destino das operações rejeitadas dependerá da aplicação.

Saito e outros autores não definem *sincronização* como sendo um processo formado pelas etapas de *escalonamento, conciliação* e *efetivação.* Verifica-se, porém, que isso está implícito quando se estuda os aspectos relacionados à sincronização de objetos. Quando Saito, por exemplo, apresenta os "*elementos da replicação otimista*" cita, nessa ordem, esses elementos: *submissão de operações*, *propagação*, *escalonamento*, *resolução de conflitos* e *efetivação* [Saito,2005-p44]. Com base na definição de Molli, apresentada no início dessa seção, deduz-se facilmente que os três últimos elementos citados por Saito referem-se ao *sincronismo*.

2.8.1 – Escalonamento de operações (*scheduling*)

Por causa da propagação em segundo plano, as operações nem sempre chegam a todos *sites* na mesma ordem. Cada *site* reconstrói então a ordem apropriada que produzirá um resultado provavelmente igual ao que foi produzido nos demais *sites.* Além disso, o resultado tem que coincidir com a expectativa, às vezes intuitiva, do usuário. Dá-se o nome de *escalonamento* a esse processo de ordenar operações de forma coerente [Saito,2005-p46].

O *escalonamento* pode ser *sintático* ou *semântico*. O escalonamento *sintático* ordena as operações com base apenas na informação de "quando", "onde" e "quem" executou a operação. Já o escalonamento *semântico* leva em consideração o significado das operações, podendo explorar as propriedades comutativas das mesmas ou alterar o conteúdo das operações se necessário [Saito,2005-p51].

Para atender aos algoritmos de ambos os tipos de escalonamento, é necessário ordenar as operações na seqüência real em que ocorreram. Se isso não for possível, seja por falta de informação ou porque foi detectada uma *concorrência,* deve-se fazer uma aproximação. Todos os eventos ocorridos durante o processo de ordenamento podem servir como parâmetros para o processo de escalonamento. É importante relembrar a diferença que existe nos ambientes de replicação entre transação e operação, conforme exposto na seção 2.4.1: transação refere-se a atualizações tradicionais em bancos de dados e operações referem-se a atualizações que são propagadas e aplicadas em segundo plano nas bases replicadas.

2.8.1.1 – Ordenando e detectando concorrência

Em uma base de dados distribuída e replicada, qualquer operação submetida em qualquer *site* é, na realidade, de "propriedade" de toda a base de dados. Sejam, por exemplo, duas operações α e β aplicadas nessa ordem, sem levar em consideração, a princípio, quais *sites* possam tê-las submetidas. A garantia de que todos os *sites* irão reconhecer essa ordem ocorre quando:

- α e β foram submetidas pelo mesmo *site* e, por conseqüência, propagadas nessa ordem.
- β foi submetida por um *site* após receber α e, por conseqüência, são propagadas nessa ordem.

Quando não é possível garantir uma determinada ordem, detecta-se uma *concorrência*. Por exemplo, se um *site* submeter β e depois receber α, nada pode garantir a ordem em que essas operações realmente ocorreram. Se o *site* acatar a operação α depois de já ter submetido β, então ele deverá propagar a seqüência por ele aplicada, ou seja, $\beta \rightarrow \alpha$.

Uma solução que resolve em parte esse problema é a *representação explícita* da seqüência de operações. Ao propagar uma operação, o *site* anexa a operação anterior. Dessa forma, se um *site* receber primeiro β e depois o par αβ saberá que α ocorreu antes de β e poderá reconstituir a ordem correta.

Um algoritmo mais preciso é o *vector clocks,* já explicado na seção 2.6.4.1, que é baseado em *timestamp*. Para atender às necessidades de ordenação e concorrência, o *vector clocks* existente no momento da operação é anexado à operação e propagado. Por exemplo, realizando esse procedimento com as operações α e β, os *vector clocks*, agora chamados de VC_α e VC_β, serão propagados. Um *site* qualquer que receber VC_α e VC_β irá compará-los, elemento a elemento. Se *todos os elementos* de VC_α forem menores ou iguais aos respectivos elementos de VC_β , então VC_β *predomina sobre* VC_α. Caso contrário haverá *concorrência*. Quando VC_β predomina sobre VC_α significa que α aconteceu antes de β [Saito,2005-p54]. O *vector clocks* registra o que ocorreu com as operações de α e β em todos os *sites*, oferecendo com isso mais informações para os algoritmos de escalonamento e conciliação.

Um algoritmo mais simples de ordenação das operações é o dos *relógios lógicos (logical clocks).* Segundo esse algoritmo, cada *site* controla seu *timestamp* e propaga a operação com um valor de *timestamp* anexado. O *site* que recebe a operação atualiza o seu *timestamp* com base no maior valor entre o seu e o recebido. Com este procedimento é possível identificar a seqüência das operações do sistema. Por exemplo, sejam C_α e C_β os *timestamps* de duas operações. Caso $C_\alpha < C_\beta$ dá-se a entender que α aconteceu antes de β e o *site* que enviou α ainda não recebeu β. Entretanto, como os *timestamps* C_α e C_β foram atribuídos em *sites* distintos, essa técnica pode identificar a seqüência de operações, mas não identifica *concorrência* porque, em termos de tempo real, não implica que de fato α aconteceu antes de β [Saito,2005-p55] .

Um outro algoritmo simples utiliza os *relógios em tempo real (real-time clocks)*. Nesse algoritmo, cada *site* propaga a operação com a data e hora anexados. Com isso, é possível qualquer *site* ordenar as operações em função do tempo. Caso um *site* receba uma operação com horário anterior à sua última operação, detecta-se uma *concorrência*. O maior inconveniente dessa técnica é garantir que os relógios de todos os *sites* estejam sincronizados, o que exige um certo esforço computacional [Saito,2005-p55].

2.8.1.2 – Escalonamento sintático

No caso do escalonamento *sintático*, o mínimo que deve ser obedecido é a ordem das operações. Nesse caso, os melhores ordenamentos são aqueles baseados em *timestamp*. Esse mecanismo não faz nada quanto às *concorrências* detectadas, ordenando as operações concorrentes por ordem de chegada. Por ser baseado apenas em uma ordem pré-determinada, o *escalonamento sintático* pode ser usado em qualquer um dos tipos de propagação, tanto na transferência de estado como na de operação.

2.8.1.3 – Escalonamento semântico

O *escalonamento semântico* tem uma forte dependência do tipo de aplicação envolvida e utiliza como base as operações submetidas aos objetos. O *escalonamento semântico* não pode, por isso, ser utilizado na propagação por transferência de estado, pois a própria natureza desse tipo de propagação não permite o conhecimento das operações executadas nos objetos.

Diversas abordagens auxiliam na compreensão do funcionamento dessa técnica, a saber: *comutatividade*, *ordenamento canônico*, *transformação operacional e otimização restritiva (optimization approach)* [Saito,2005-p56].

A técnica da *comutatividade* aplica-se quando, por qualquer motivo, é possível trocar a ordem natural das operações. Em geral, esses motivos estão relacionados a determinadas características da aplicação, cujo conhecimento implica em tratamentos diferenciados no que diz respeito à comutatividade. Por exemplo, quando duas ou mais operações podem ser comutadas sem alterar o resultado final, o algoritmo de escalonamento desconsidera os problemas de *concorrência,* que eventualmente possam ter ocorrido com as operações comutáveis. Identificar operações comutáveis também reduz as quantidades de "*roll-backs*" e "*redos*", nas tentativas de procurar a melhor ordem para as operações. Essas tentativas devem, sempre que possível, ficar restritas às operações não comutáveis.

A técnica de *ordenamento canônico* é mais adequada, quando se pode estabelecer uma ordem formal das operações conforme a semântica da aplicação. Um exemplo disto pode ser encontrado em *sistemas de arquivos,* onde só se pode criar um subdiretório se for criado antes um diretório, ou seja, existe implícita uma ordem natural das operações [Saito,2002-p22]. As regras de integridade aplicadas em bancos de dados podem fornecer muitos subsídios para se formar um bom conjunto de regras de *ordenamento canônico*.

A *transformação operacional* é utilizada quando as operações não são comutativas. O objetivo é reescrever as operações para garantir o resultado esperado. Por exemplo, considere um texto "`abc`" replicado, onde um *site* insere "`x`" na primeira posição, resultando no texto "`xabc`", enquanto outro *site* remove o primeiro *byte* resultando em "`bc`". Ao conciliar, obedecendo à ordem das operações, se terá o texto "`abc`" no primeiro site e o texto "`xbc`" no outro. Para evitar isso, o primeiro *site* transforma a operação remove(1) para remove(2) e ambas as réplicas ficam idênticas.

A *otimização restritiva (optimization approach)* utiliza um conceito de *restrições entre operações*. Uma *restrição* pode ser entendida como uma pré-condição imposta às operações e pode ter origem em diversas fontes: no usuário, na aplicação, no tipo de dado ou nos sistemas em uso. Existem restrições de *dependência* (α só executa se β ocorreu), *implicação* (se α ocorreu, então execute β), *escolhas* (execute α ou β) e *temporais* (execute α, se β não ocorrer em um determinado período de tempo). Todas essas restrições podem ser combinadas, formando regras bem complexas e, dependendo da aplicação, outras restrições podem ser criadas. Essas restrições podem ser usadas na conciliação ou até antes, diretamente nas aplicações, para evitar que conflitos ocorram posteriormente.

2.8.1.4 – Critérios de corretude dos escalonamentos

Para que as réplicas de um banco de dados fiquem em um estado convergente, é necessário que a serialização das escalas aplicadas em cada réplica sejam equivalentes [Graham,1995]. Considera-se que a questão da serializabilidade nos *sites* locais já é tratado pelos SGBDs locais assim como a garantia das propriedades ACID [Anderson,1998-p485] [Breitbart,1997-p174]. Na replicação, porém, propaga-se um conjunto de operações originalmente serializadas em cada *site* local. A união de todas as operações propagadas irá compor a escala de operações (*schedule*) que existe em cada *site* do ambiente de replicação. Para garantir que a efetivação dessas operações leve cada réplica a um estado consistente e garanta a correta convergência, é necessário adotar estratégias similares àquelas adotadas pelos SGBDs para garantir a equivalência da escalas de operações. Independente da abordagem adotada pelos algoritmos de serialização, como por exemplo, a equivalência por conflito ou equivalência por visão serializável (descrita com mais detalhes em [Silber,99-p.453]), alguns aspectos devem ser considerados:

– Se as operações de escritas se referirem a objetos distintos não há conflitos. Neste caso, as operações podem ser efetivadas pela ordem de chegada e não existe nenhum problema quanto à comutatividade.

– Sendo as operações relativas ao mesmo objeto, é conveniente mantê-las na ordem de execução conforme a ordem nas transações originais, sendo estas ordenadas conforme os seus respectivos *timestamps*. Neste caso, devem-se analisar três situações que servirão de orientação para a construção dos algoritmos de escalonamento e conciliação, conforme exemplificado na tabela 3.1 e descrito a seguir:

- Caso 1: a operação de escrita não depende do valor lido. Neste caso, sugere-se que se efetive a última escrita com base no *timestamp*. Como exemplificado na tabela, independente da ordem de chegada das operações, a última escrita $W_B(Q)$ é a que prevalece sobre as demais.
  Exemplo: a descrição de um produto em um controle de estoque. Se o nome mudou para *"disco rígido"*, não importa o fato de antes ser descrito como *"hard disk"*.

- Caso 2: a operação de escrita depende do valor lido, as operações são comutativas e não existem pré-condições. Neste caso, sugere-se que todas as operações de leitura e escrita sejam aplicadas na ordem de chegada das transações. Na tabela, exemplifica-se uma das possibilidades para a ordem de chegada das operações.
  Exemplo: ao entrar produtos em um estoque, o saldo de um produto depende do saldo anterior mais as quantidades entradas. Independente da ordem das entradas, o saldo final será o mesmo.

- Caso 3: a operação de escrita depende do valor lido, as operações são comutativas e existem pré-condições, ou as operações não são comutativas. Neste caso sugere-se que todos os *sites* efetivem as operações na mesma seqüência de leituras e escritas, nem que para isto seja necessário retornar o objeto quantas vezes forem necessárias (*rollbacks*) a uma situação anterior ao *timestamp* da primeira operação a ser efetivada conforme a seqüência. Na tabela, exemplifica-se a única ordem de chegada possível para as operações.

Exemplo: ao sair produtos em um estoque, o saldo de um produto depende do saldo anterior menos as quantidades saídas, se a quantidade para sair for menor ou igual ao saldo anterior. Se a quantidade de saída for maior que o saldo, a operação não pode ser efetivada.

Site A	*Site* B	*Site* C	*Site* D		Caso 1	Caso 2	Caso 3
Read(Q)	...	...	...		$W_B(Q)$	$R_C(Q)$	$R_A(Q)$
...	...	Read(Q)	...			$W_C(Q)$	$W_A(Q)$
...	...	...	Read(Q)			$R_A(Q)$	$R_C(Q)$
Write(Q)	...	...	...	Escalas		$W_A(Q)$	$W_C(Q)$
...	...	...	Write(Q)	=>		$R_B(Q)$	$R_D(Q)$
...	Read(Q)	...	...			$W_B(Q)$	$W_D(Q)$
...	...	Write(Q)	...			$R_D(Q)$	$R_B(Q)$
...	Write(Q)	...	...			$W_D(Q)$	$W_B(Q)$

Tabela 2.1 – Seqüência de operações (*schedule*) de possíveis escalonamentos referentes aos três casos descritos anteriormente.

2.8.2 – Conciliação

O procedimento de *conciliação,* como acontece também no escalonamento, é muito dependente do tipo de aplicação. Esse procedimento é composto por duas partes: *detecção do conflito* e *resolução do conflito*. Evidente que não se detectando conflito, o objeto já pode ser efetivado.

Não se deve confundir conflito com atualizações simultâneas sobre um mesmo objeto ocorridas em *sites* diferentes. Pode ocorrer simultaneidade entre operações sem haver conflitos e conflitos entre operações sem haver simultaneidade. Por exemplo, duas operações ocorridas em *sites* diferentes de reserva aérea para um mesmo vôo e assento: podem não ser simultâneas, mas são conflitantes; enquanto uma operação de retirada de um produto em estoque (levando em conta que há saldo suficiente) e outra de reposição de estoque do mesmo produto, ocorridas de forma simultânea em *sites* diferentes, não são conflitantes.

Muitas vezes a *detecção do conflito* utiliza as mesmas informações utilizadas pela *propagação* e o *escalonamento,* o que pode causar certa confusão. As informações são as mesmas, as finalidades, porém, são diferentes. Um exemplo é o *vector clocks*: a propagação utiliza a informação "tempo" para concluir se deve ou não propagar as operações, o escalonamento utiliza o "tempo" para estabelecer uma ordem nas mesmas e a conciliação utiliza o "tempo" para saber se a operação pode ser corretamente aplicada aos objetos.
Não se deve também confundir *resolução de conflitos* com as regras do escalonamento semântico que impedem, em parte, que ocorram conflitos. Por exemplo, se existir a regra "a operação de mudança de preço só é válida se existir antes uma operação de autorização para mudar o preço", o escalonamento semântico pode ordenar as duas operações, *autorização→mudança*, e assim evitar um conflito.

Um conflito ocorre quando uma pré-condição para uma regra ser aplicada não é satisfeita. Por exemplo, não se pode alterar um objeto que foi eliminado, não se pode incluir um objeto já existente, não se pode fazer uma reserva de um objeto já reservado.

Uma vez que existe um conflito, deve-se seguir com a *resolução do conflito*. Uma das formas mais simples é a resolução manual, ou seja, solicita-se ao usuário que resolva o conflito. Por outro lado, resoluções automáticas já dependem muito do tipo de aplicação.

Muitas soluções simplesmente *não fazem conciliação*. Aplicam as operações na ordem do escalonamento, sem verificar se existe conflito entre as operações. A conseqüência dessa prática, entretanto, é perder a última atualização (*lost update*), o que nem sempre é desejável.

Outras aplicações procuram evitar os conflitos adotando técnicas de limitação de inconsistência. Uma técnica é fazer uma rápida e freqüente propagação ou fragmentar os objetos, para diminuir a hipótese de ocorrer conflito. Uma outra técnica é fazer uma *desconexão planejada* [Holliday,2002-p393], onde se mudam os atributos de escrita e leitura dos objetos conforme o *site* esteja conectado ou não. Um banco de dados, por exemplo, pode ficar *read-only* parcial ou totalmente, quando o *site* se desconectar e voltar a ser *read-write* quando a conexão for restabelecida.

2.8.2.1 – Conciliação sintática

A *conciliação sintática* considera que operações concorrentes podem ser conflitantes, quando o escalonamento sintático detectou a concorrência. Quando se utiliza a propagação por *transferência de operação,* o escalonamento sintático oferece poucos recursos para detectar e resolver conflitos. Por causa disto, a conciliação sintática é freqüentemente usada, quando a propagação for por *transferência de estado*. No caso de transferência de estado, tem-se disponível para fins de conciliação, apenas duas versões do objeto: a local e a primeira disponível na fila após o escalonamento. Somente após conciliar as duas versões, passa-se à próxima da fila.

Detecção de conflito: na *transferência de estado*, para detectar um conflito utiliza-se dos controles usados na propagação: *timestamp*, *bit modificado, vector clocks ou death-certification.* O princípio básico é verificar, a partir dos controles disponíveis, se o objeto que chegou é exatamente a próxima versão com relação ao objeto local. Se for, não há conflito. Por exemplo, se objeto local *ts*=8 e o objeto recebido *ts*=9, então não há conflito. Com a técnica *bit-modificado,* não há conflito quando o objeto local não foi modificado em relação ao que chegou.

Resolução de conflito: ocorrendo um conflito, uma das soluções mais simples é pedir a intervenção do usuário que irá decidir qual das versões é válida. Outra opção é permitir que o usuário faça uma mescla das duas versões, criando uma nova. É possível criar regras para automatizar a resolução de conflitos. Tais regras podem basear-se na origem do objeto, no tipo de objeto (que pode permitir regras para criar um objeto novo a partir dos dois em conflito) e outros critérios que são altamente dependentes da aplicação.

2.8.2.2 – Conciliação semântica

Usada quando se utiliza a propagação por *transferência de operação*, pois esse tipo de propagação oferece as informações necessárias para a elaboração de algoritmos mais complexos. A conciliação semântica é ainda mais dependente da aplicação do que a conciliação sintática, sendo difícil fazer generalizações. Os controles utilizados na propagação (*timestamp*, *bit modificado, vector clocks ou death-certification*), consultas a objetos locais, regras pré-estabelecidas e outros elementos fazem parte dos algoritmos de conciliação

semântica. O modelo mais conhecido que utiliza a conciliação semântica é o *Bayou* [Saito,2002-p23], usado em *sistemas de bases de dados móveis*. Com base nesse modelo, são feitas as considerações a seguir:

Detecção de conflito: como requisito mínimo, todas as pré-condições das operações tem que ser explicitamente informadas. Em seguida, faz-se uma *verificação de dependência* (*dependency check*) entre a operação e os objetos, ou seja, verifica-se se o estado dos objetos está de acordo com a pré-condição explicitada. Se a operação atender à pré-condição, é então efetivada.

Resolução de conflito: acontecendo um conflito, faz-se uma *combinação de procedimentos (merge procedure),* que irá alterar ou gerar novas operações para serem aplicadas. Esse procedimento pode repetir-se quantas vezes forem necessárias. Além disso, se necessário, pode-se ainda desfazer e refazer as operações, até se chegar a um resultado considerado bom.

Por exemplo, dois usuários em réplicas diferentes agendam uma reunião para o mesmo horário. Quando ocorrer a reconciliação, o conflito será detectado. Então, o usuário com o maior *timestamp* terá seu horário remarcado para o próximo horário livre. Apesar da simplicidade desse exemplo, a experiência prática com *Bayou* vem demonstrando ser extremamente difícil de ser utilizada [Saito,2002-p23].

A figura 2.8, mostra um esquema taxionômico partindo da hipótese de que conflitos podem ocorrer, como evitá-los e, caso ocorram, como detectá-los e resolvê-los.

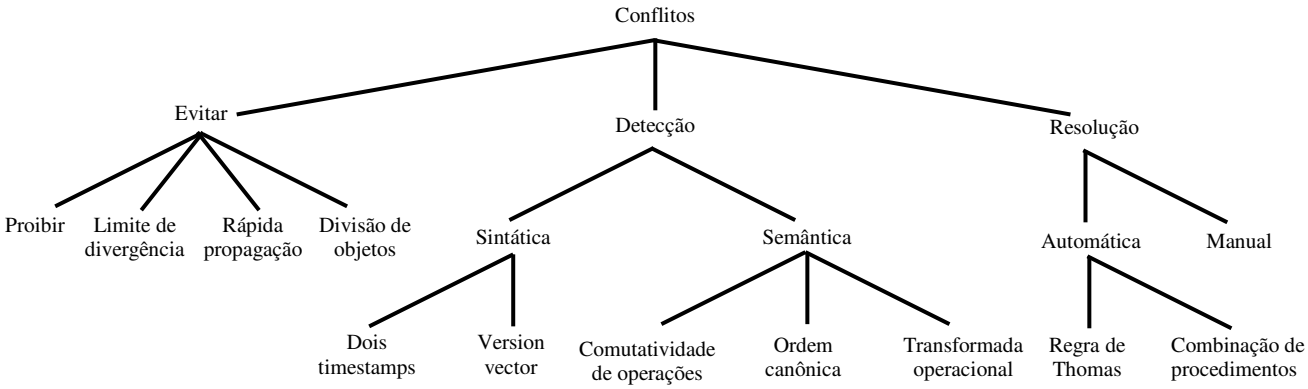


Figura 2.8 – Esquema taxionômico da sincronização [Saito,2002-p14]

2.8.3 – Efetivação (*commitment*)

A efetivação tem três propósitos: primeiro, garantir que todos os *sites* acatem o escalonamento ou resolução de conflitos adotados por um determinado *site*; segundo, informar ao usuário que determinada operação já foi efetivada em todos os *sites* e, por isso, encontra-se estável e seus efeitos não mais estarão sujeitos a *roll-back*; terceiro, prover um mecanismo para controle de espaço, pois garante que as operações estáveis já podem ser retiradas do *site.* [Saito,2005-p59][Saito,2002-p23]. Os mecanismos para se conseguir isso são descritos a seguir.

Observa-se que para realizar os três propósitos citados, é necessário trocar informações entre os *sites*. A essa prática, dá-se o nome de *efetivação explícita*.

Efetivação implícita por entendimento comum: nesse mecanismo os sistemas não utilizam a efetivação explícita, porque o escalonamento já é determinístico e a resolução de conflito não existe, como é o caso do mecanismo da regra de escrita de *Thomas.*

Acatamento em "background": cada *site* utiliza recursos temporais (*vector clocks*, *timestamp,* relógio em tempo real) para manter um controle sobre o progresso das operações nos outros *sites.* Uma técnica para conseguir esse controle é usar um vetor de reconhecimento (*ack vectors*), com base no *vector clocks.* Outra técnica é construir *matrizes de timestamp,* com dimensão *N x M,* onde *N* é o número de *sites* e *M* o número de mestres e onde as linhas das matrizes são os *vector clocks*.

Efetivação por consenso: utiliza de protocolos que indicam as operações que podem ser efetivadas ou canceladas. A *efetivação com base primária (primary-based commitment)* delega a um *site* primário a responsabilidade de indicar que operações podem ser efetivadas e em qual ordem. Na *efetivação por quorom,* cada *site* "vota" a favor da efetivação de cada operação até que, ao atingir uma maioria, a mesma será efetivada.

Estas técnicas podem ter problemas com escalabilidade. Neste caso, faz-se necessário utilizar técnicas auxiliares. Uma delas, por exemplo, é o *Multiview Access Protocols* [Xiangning,1998]. Nesta abordagem, os *sites* são agrupados em *clusters* organizados de forma hierárquica, e cada *cluster* possui uma cópia primária. O protocolo atua, em um primeiro momento, apenas no nível dos *clusters* para consolidar as efetivações e, depois, cada *cluster*, utilizando a sua cópia primária como referência, efetiva as operações nos demais *sites* do *cluster*.

2.9 – Controle de divergências

O objetivo do controle de divergência é garantir uma qualidade mínima para réplicas que implementaram a *consistência eventual (eventual consistency).* Informalmente, o conceito de *consistência eventual* significa que todas as réplicas chegarão ao mesmo valor final, se o usuário parar de submeter operações. Na definição formal, é dito que o objeto possui *consistência eventual* quando todas as réplicas partem de um mesmo estado inicial e o objeto satisfaz às seguintes condições [Saito,2005-p56] [Saito,2002-p21]:

- A qualquer momento, para cada réplica, há um prefixo nas filas de operações (*schedule*) que são equivalentes. Esse estado é chamado *prefixo efetivado* (*committed prefix*) para a réplica.
- A efetivação do prefixo para cada réplica cresce monotonicamente com o tempo.
- Todas as operações aceitas com o *prefixo efetivado* satisfazem a suas pré-condições.
- Qualquer operação, executada ou não, será incluída na fila com *prefixo efetivado*.

Cumprindo-se esse objetivo, as réplicas irão convergir. Mas, para que o controle de divergência possa garantir uma boa qualidade no estado das réplicas, Saito sugere três tipos de mecanismos: *forçar a ordem de leitura e escrita*, *impor limites à divergência* e *usar técnicas probabilísticas* [Saito,2005-p72]. A seguir, detalha-se um pouco mais esses tipos:

Forçar a ordem de leitura e escrita: o objetivo dessa técnica é evitar que, após uma escrita em uma réplica, o usuário leia o valor anterior em outra réplica. Isso não é difícil conseguir durante o escalonamento, desde que se estabeleçam regras para as operações de leitura, que tenham dependência direta com determinados conjuntos de operações de escrita, de forma que

tais operações de leitura ocorram sempre depois das operações de escrita. Quanto à propagação, sugere-se atrasar a operação de leitura, até que o *vector clocks* assinale a conclusão das operações de escrita. Essas técnicas também são chamadas de *dependências explícitas* [Saito,2005-p72].

O maior inconveniente da abordagem anterior é a necessidade de especificar as dependências para cada tipo de operação de leitura. Para contornar isso, pode-se usar um mecanismo chamado de *garantias de sessão (session guarantees),* que generaliza as operações de leitura e escrita em um conjunto de quatro elementos:

- *RYW - Read your writes*: garante que o mesmo usuário sempre vai ler o que ele escreveu previamente.
- *MR - Monotonic reads:* leituras sucessivas do mesmo usuário retornarão sempre o conteúdo das atualizações que estiverem ocorrendo.
- *WFR- Writes follow reads:* aceita uma nova operação de escrita, somente após as escritas observadas por leituras anteriores, pelo mesmo usuário, já estiverem incorporadas à mesma réplica.
- *MW - Monotonic writes:* garante que qualquer operação de escrita só ocorrerá, depois que todas as escritas que a antecederam já tenham sido incorporadas à réplica.

A técnica de *impor limites à divergência* das réplicas procura limitar as inconsistências através de medidas de tempo ou eventos. Um réplica pode ficar "vencida" exigindo uma atualização; pode ser atualizada através de *polling* periódicos; pode estabelecer um número máximo de operações pendentes, recusando novas operações quando atingir o limite e, finalmente, pode estabelecer limites numéricos com base em "quotas", limitando o número de operações permitidas a cada réplica, antes de propagar as operações para outras réplicas.

Finalmente, as *técnicas probabilísticas* baseiam-se no estudo e análise estatística da carga de trabalho das aplicações. Com base no perfil da freqüência de atualização dos objetos replicados, sugere modelos que poderão reduzir o custo da comunicação e melhorar a qualidade das réplicas.

2.10 – Aplicações e trabalhos relacionados

Existem muitos trabalhos e aplicações relacionados à replicação de dados. A abordagem entre os autores varia muito, mas conceitualmente tratam das mesmas questões descritas no capítulo 2. Algumas vezes, referem-se aos mesmos conceitos, porém, com nomes distintos. Por exemplo, Phatak denominou *transferência de estado* por *conciliação centrada em dados* e *transferência de operação* por *conciliação centrada em transação* [Phatak,2004] .

A grande maioria dos trabalhos é voltada para computação móvel. Por outro lado, as necessidades de replicação na Internet também incentivaram o desenvolvimento de muitas aplicações, que passaram a usar conceitos relacionados à replicação de dados. É o caso, por exemplo, do *Active Directory* da Microsoft, *DNS* (*Domain Name Server*), *NIS (Name Information Service*), *CVS (Control Version Software*), Usenet e os "sites espelhos" (*mirror*). Observa-se, porém, que tais aplicações foram desenvolvidas para usos muito específicos. Outras aplicações menos conhecidas também utilizam replicação.

Na próxima seção faz-se uma breve descrição de algumas dessas aplicações, detalha-se um pouco mais o Xmiddle, pelo fato dessa aplicação apresentar algumas características

semelhantes ao modelo proposto e, finalmente, avalia-se a conveniência de especificação de um novo modelo.

2.10.1 – Características principais das aplicações relacionadas

Conforme encontrado em [Saito,2005-p48,49], apresenta-se a seguir uma breve descrição das aplicações mais conhecidas que usam replicação de dados, classificando-as conforme os conceitos abordados no decorrer desse capítulo.

DNS (Domain Name Service): mestre único com transferência de estado.

Usenet (Wide-Area Information Exchange): multimestre com propagação epidêmica.

CVS (Control Version Software): sistema de controle de versão, que permite edição de arquivos por grupos de trabalho, controlando versões e permitindo a pesquisa de versões antigas quando solicitado pelo usuário. É um sistema de transferência de operação multimestre, que centraliza a comunicação através de único repositório em uma topologia estrela.

Bayou: projeto desenvolvido pela Xerox Palo Alto Reasearch Center, para compartilhar dados entre usuários móveis. É do tipo cliente-servidor, sendo que ambos podem ser co-residentes no mesmo *host.* A aplicação cliente interage com o servidor através da *Bayou API* [Musolesi,2002]. *Bayou* é um sistema de transferência de operação multimestre, que usa a propagação epidêmica sobre qualquer tipo de topologia, inclusive topologias dinâmicas.

PDA (Personal Digital Assistants): computador de mão que pode funcionar como uma agenda pessoal e com outras aplicações. Eventualmente, o usuário sincroniza o PDA com seu PC, com troca de informações bidirecional. O PDA é um sistema de transferência de estado, multimestre, tendo o registro como unidade de replicação da base de dados, atualizações e resolução de conflitos.

As tabelas 2.1 e 2.2 apresentam um resumo das principais características dessas aplicações.

	Objeto	**Tipos de propagação**	**M**	**Topologia**	**Técnica de propagação**	**Liberação de espaço**
Active Directory	Par nome-valor	Estado		Qualquer uma	*Pull-based*	Expiração
Bayou	BD	Operação		Qualquer uma	*Timestamp Vector* e Manual	Efetivação cópia primária
CVS	Arquivo	Operação		Estrela	Manual	Manual
DNS	Todo o BD	Estado	1	Estrela	*Push* e *pull-based*	Manual
NIS	Todo o BD	Estado	1	Estrela	*Push-based*	Manual
Palm (PDA)	Registro de BD	Estado		Estrela	Manual	—
Usenet	Artigos	Estado		Qualquer uma	*blind push*	Expiração
Web/file mirror	Arquivo	Estado	1	Árvore	*Pull-based*	Manual

Tabela 2.2 – Aspectos representativos da comunicação para aplicações conhecidas. *M* é o número de mestres.

	Ordenamento	Detecção de conflito	Resolução de conflito	Efetivação	Consistência (controle de divergência)
Active Directory	*Logical clock*	Nenhum	Thomas write rule	Nenhuma	Eventual
Bayou	Ordem recepção e cópia primaria	Predicado (pré-condições)	Definido pelo usuário	Cópia primária	Eventual
CVS	Cópia primária	*Two timestamps*	Exclusão	Cópia primária	Eventual
DNS	Mestre único	—	—	—	Tempo
NIS	Mestre único	—	—	—	Eventual
Palm (PDA)	Por recepção e cópia primária	Bit modificado	Determinado pela aplicação/Manual	Cópia primária	Eventual
Usenet	*Real-time clock*	Nenhum	Thomas write rule	Nenhuma	Eventual
Web/file mirror	Mestre único	—	—	—	Eventual / tempo

Tabela 2.2 – Aspectos representativos da concorrência.

Os produtos comerciais SQLServer [Microsoft,2007], Oracle [Oracle,2007], Sun StorEdge [Sun,2007] também disponibilizam recursos de replicação, mas as implementações ficam restritas às soluções disponibilizadas pela própria empresa, não existindo, portanto, nenhum padrão comum. Foge ao escopo desse trabalho analisar cada um desses produtos. As empresas responsáveis disponibilizam na Internet uma boa documentação a respeito.

Encontrou-se ainda, através da Internet, produtos comerciais que utilizam replicação, a exemplo do Sydeco AFV – Automação de Força de Vendas [Sydeco,2007]. Entretanto, a empresa não disponibiliza informações adicionais sobre as técnicas utilizadas. Outro produto é o *Pervasive Software* [Pervasive,2007], um banco de dados herdeiro do antigo Btrieve da Novell, mas que apresenta os mesmos problemas apontados para os bancos de dados SQLServer, Oracle e Sun StorEdge.

A Universidade da Califórnia, através de contratos com a DARPA (*Defense Advanced Research Projects Agency*), pesquisou e desenvolveu alguns trabalhos voltados à replicação de dados [UCLA,2007], chegando aos produtos: *Ficus (File Sharing Under Fícus), Truffles (Secure Flexible File-Sharing Over Wide-Area Networks), Roam (Replication System for Mobile Environments) e Rumor (Replication for Mobile Computers).* Esses produtos replicam arquivos e diretórios e utilizam transferências de estado.

Destacam-se ainda o *Coda File System* [Coda,2007], que replica sistemas de arquivos, e o *IceCube* [Kermarrec,2001] [IceCube,2007], que apresenta um conjunto de algoritmos e ferramentas para conciliação. Para fazer a replicação e conciliação, esses aplicativos baseiam-se no *log* das aplicações.

Observou-se, no estudo das diversas publicações, que existem muitos projetos de replicação voltados para sistemas de arquivos. No caso de banco de dados, a replicação ocorre apenas com o mesmo tipo de banco de dados, ou seja, as aplicações pesquisadas não suportam ambientes com sistemas de banco de dados heterogêneos. De todas as abordagens estudadas, a que mais se aproxima do modelo proposto no presente trabalho é o *Xmiddle*. Por este motivo, na próxima seção, o *Xmiddle* será um pouco mais detalhado.

2.10.2 – XMiddle

O XMiddle é um sistema de compartilhamento de dados para redes independentes, *open-source,* resultado de um esforço do *Departamento de Ciências da Computação da University College London* e teve como base inicial a tese de Musolesi [Musolesi,2002].

XMiddle é definido da seguinte forma pelo autor: "XMiddle é um *middleware* para computação móvel, que permite a *hosts* móveis (por exemplo, PDAs, telefones móveis, *laptops* e outros equipamentos sem fio) compartilhar informações uns com os outros, sem precisar da existência de nenhuma infraestrutura fixa de rede" [Musolesi,2002-p63]

A arquitetura do XMiddle é "tipica dos sistemas *middleware*" [Musolesi,2002-p65] e é apresentada na figura 3.1, tendo como referência o modelo ISO/OSI:

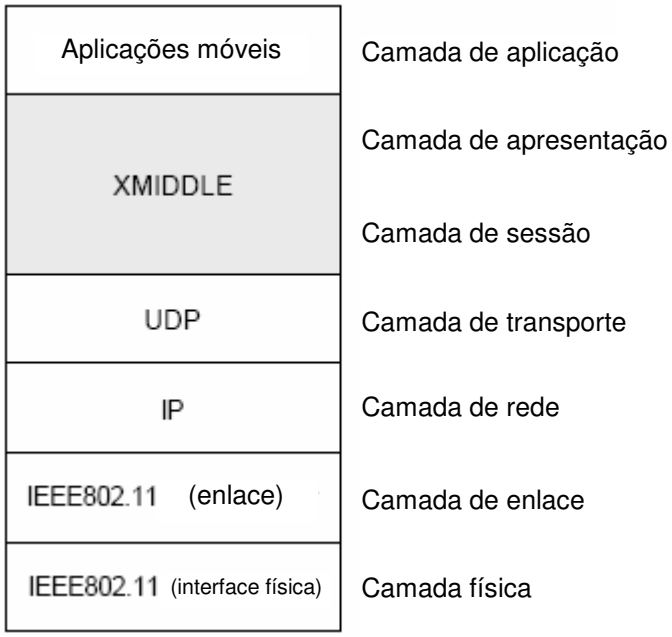


Figura 2.9 – Pilha de protocolos ISO/OSI para ambientes móveis utilizado XMiddle.

O XMiddle, conforme mostra a figura 3.1, implementa funcionalidades das camadas de sessão
e apresentação, sobre um protocolo padrão tipo UDP. A base de dados utilizada pelo XMiddle utiliza a tecnologia XML. No que diz respeito ao nível de apresentação, o XMiddle mapeia documentos XML para uma árvore DOM[5] e disponibiliza, para as aplicações móveis, as primitivas para manipular e compartilhar os dados. O nível de sessão implementa o gerenciamento das conexões e desconexões entre equipamentos móveis.

O XMiddle foi inteiramente implementado em Java e utiliza, portanto, a Máquina Virtual Java. O protótipo utilizou o protocolo UDP sobre IP.

O XMiddle não coincide com a proposta desse trabalho, apesar da sua aparente afinidade, porque está limitado a manipular apenas bases de dados XML e as aplicações usuárias têm

[5] *DOM (Document Object Model),* uma interface de programação padronizada pela W3C para manipular arquivos XML [W3C,2007]

que utilizar obrigatoriamente a interface DOM. Entretanto, por ser *open-source* e feito em Java, dois aspectos que fazem parte dessa proposta, serve de uma boa fonte de referência para o presente trabalho.

2.11 – Comentários

A grande diversidade das abordagens expostas demonstra o quanto os sistemas de replicação podem variar de soluções simples às mais complexas. Observa-se também que muito dos modelos, técnicas e mecanismos expostos ou sugeridos, são muito voltados à certos tipos de aplicação. Utilizar sistemas com replicação, exige um alto nível de especificidade no momento de escolher a abordagem que servirá de orientação ao projeto.

Apesar das grandes vantagens que a replicação de dados proporciona, dos diversos trabalhos e aplicações desenvolvidas, ainda não existe uma solução que seja amplamente aceita pelo mercado, pelas universidades e pelos centros de pesquisa. O estudo realizado no escopo desse trabalho não permitiu descobrir de forma fundamentada as razões dessa constatação, mas é possível perceber algumas causas. Primeiro, muitas soluções são voltadas para aplicações específicas, como é o caso do *DNS, CVS* e do *Active Directory*. Segundo, outras soluções possuem restrições intrínsecas em termos de aplicabilidade, como é o caso do Palm e do XMiddle. Terceiro, por causa ignorada, muitas soluções não tiveram continuidade de suporte, como é o caso do projeto *Bayou*, encerrado em dezembro de 1997, conforme consta em seu próprio *site* na Internet. Quarto, o fato de não existirem padrões no tratamento da "replicação e conciliação" dificulta o desenvolvimento de soluções, trazendo por conseqüência o aumento do custo de aquisição e aprendizado sobre o assunto. Finalmente, em razão da ausência de uma grande demanda por replicação de dados, fica a atual demanda restrita a situações muito específicas, como é o caso da computação móvel. Por outro lado, como o uso da computação móvel tem crescido muito nos últimos anos, percebe-se também um aumento gradativo no interesse por soluções capazes de prover serviços de replicação de dados.

Dessa forma, na lacuna existente nessa área, de um modelo genérico de um ambiente aberto, que dê suporte ao desenvolvimento de aplicações baseadas na replicação de dados, justifica-se a proposta apresentada no próximo capítulo. A replicação, conforme proposta aqui, pode ser uma boa alternativa também aos sistemas múltiplos de banco de dados, pois o efeito final – a ilusão de uma base única, conforme descrito em 2.3 – também ocorrerá apesar da utilização de caminhos diferentes.

Capítulo 3

Modelo assíncrono para replicação, propagação e conciliação de bases de dados distribuídas

Neste capítulo é apresentado o modelo proposto para replicar, propagar e conciliar bases de dados distribuídas em ambientes multimestres. Na seção 3.1 apresenta-se uma visão geral do modelo e o contexto onde ele se aplica. Os detalhes do modelo são apresentados na seção 3.2. A seção 3.3 mostra soluções selecionadas do modelo para situações eventuais, que fogem à rotina normal dos processos previstos pelo modelo. As limitações do modelo são descritos na seção 3.4 e as possibilidades de desenvolvimentos futuros em 3.5. Finalmente, a seção 3.6 é dedicada a comentários gerais.

3.1 – Visão geral

Este trabalho propõe um modelo para ***R**eplicação, **P**ropagação e **C**onciliação de **D**ados* e, por isto, todas as referências ao modelo utilizam o acrônimo **RPCD**. Para atingir os objetivos descritos no item 1.2, o modelo utiliza conceitos de diferentes áreas:

a- Aplicável a qualquer tipo de banco de dados, incluindo ambientes com bases heterogêneas.
   O modelo utiliza o conceito de modelo global e funções de tradução, tratados em *sistemas de bancos de dados múltiplos* (2.3).

b- Planejado para utilizar replicação de forma assíncrona.
   Basicamente utiliza a grande maioria dos conceitos tratados no capítulo 2.

c- Define um protocolo de comunicação com serviços específicos para o processo de replicar dados.
   Utiliza conceitos já bastante conhecidos na área de redes para implementar os serviços voltados às questões de replicação. Utiliza XML na definição das PDUs e modelo de serviços orientado a objeto.

d- Não utiliza conceitos específicos de nenhuma linguagem de programação, podendo ser implementada em qualquer linguagem desde que orientada a objeto.
   Utiliza alguns conceitos de Engenharia de Software e, notadamente, orientação a objeto e UML.

e- Apresenta soluções que exigem pouca ou nenhuma alteração nos aplicativos usuários dos sistemas de banco de dados.
   Utiliza recursos do protocolo TCP/IP, arquivos *pipe* e conceitos de *driver* de acesso aos BDs.

f- Modularizado com divisões claras de responsabilidades, o que permite a inserção de algoritmos especializados em qualquer ponto do modelo.
   Utiliza o conceito de interface, conforme utilizado em muitas linguagens de programação e pela UML. Nada impede que se utilize opcionalmente portas TCP, desde que se crie regras de protocolo para transferir as solicitações entre programas.

Como técnica de modularização pode-se usar pacotes, subsistemas e interfaces. O importante é que cada tipo de problema e a respectiva solução sejam tratados apenas em um pacote, subsistema ou interface específica, o que permite um maior ou menor investimento de recursos, dependendo do grau de sofisticação que se deseje atingir.

Em termos dos conceitos de replicação, o modelo RPCD é fundamentado na *replicação otimista* (seção 2.4) e permite o uso de *bancos de dados múltiplos* (seção 2.3). Este modelo segue os padrões usuais para SGBDD múltiplos: banco de dados relacionais e linguagem SQL nas operações [Silber,625] e utiliza modelos globais. Os outros principais fundamentos do modelo proposto são:

- Distribuição dos dados *cliente-servidor,* permitindo f*ragmentação mista.*
- Localização de dados utilizando apenas a identificação do *site* participante do domínio de replicação.
- A propagação utiliza *transferência de operação* em um ambiente *multimestre* e no modo *pull-based.* Cada base de dados é replicada na *topologia estrela.*

O modelo foi planejado para situar-se na camada intermediária entre o TCP/IP e os aplicativos que acessam as bases de dados. O modelo RPCD, por sua vez, divide-se em duas subcamadas: aplicação e protocolo. A subcamada de aplicação é a responsável por todo o processamento que ocorre em nível local: acesso às bases de dados, acesso aos aplicativos, procedimentos relativos à replicação, conciliação e propagação. A subcamada de protocolo é responsável pelas associações entre *sites,* pelos serviços do protocolo de comunicação e processamento das PDUs. A figura 3.1 ilustra esta arquitetura.

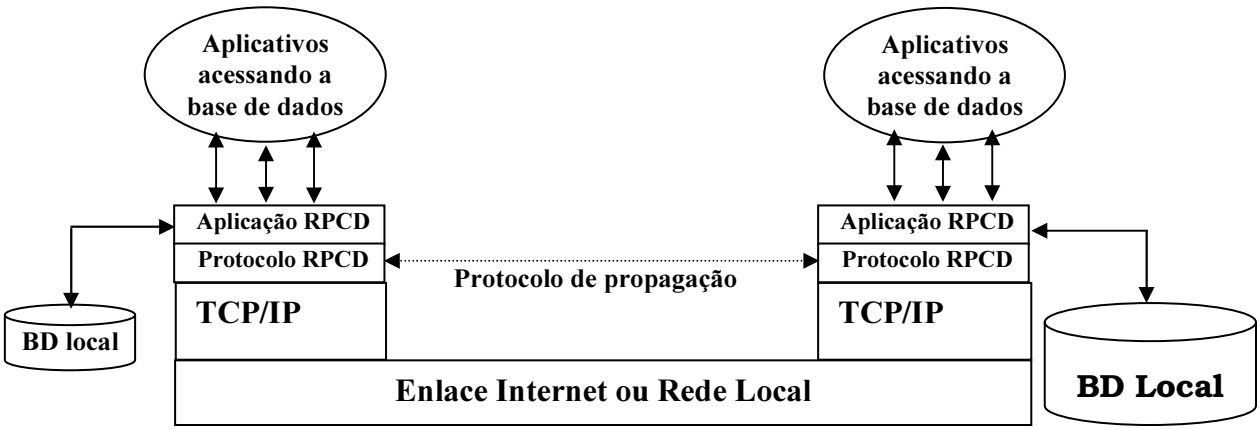


Figura 3.1 – Modelo RPCD no contexto do modelo Internet

Finalmente, o modelo prevê um grau de autonomia muito grande para cada *site* participante do domínio da replicação, no que diz respeito aos dados locais. Em outras palavras, o que ocorre no *site* com os dados locais é de exclusiva responsabilidade do próprio *site*.

Na próxima seção, este modelo será exposto e detalhado. Serão apresentadas também suas limitações e possibilidades de trabalhos futuros.

3.2 – O modelo RPCD

As principais responsabilidades do processo de replicação previstas no modelo RPCD são divididas em 3 módulos: monitoração dos dados; replicação e sincronização dos dados; comunicação (protocolo de propagação). A figura 3.2 ilustra como eles se relacionam.

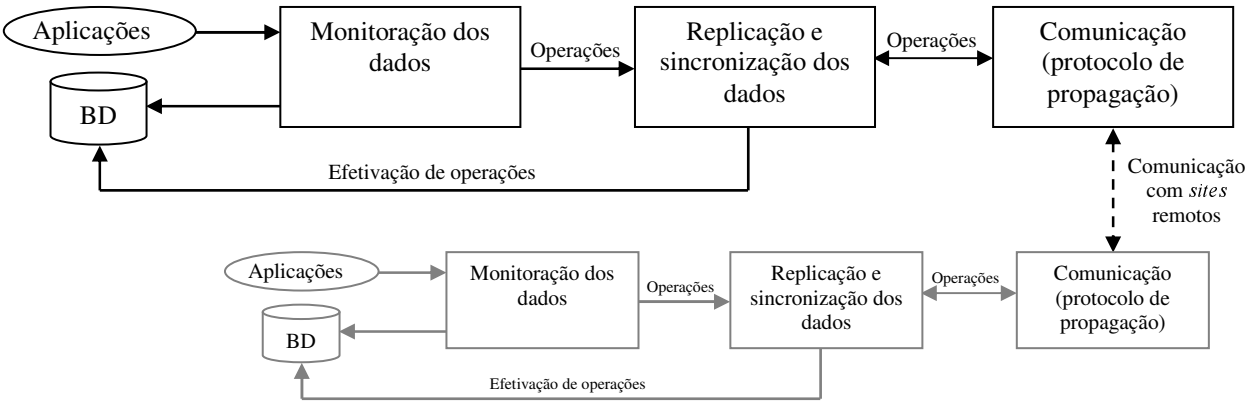


Figura 3.2 – Principais módulos do modelo RPCD

A Monitoração dos Dados tem a finalidade de capturar as operações aplicadas à base de dados e para isto interage com as aplicações e o BD. As operações capturadas são repassadas ao módulo de Replicação e Sincronização dos Dados, onde se encontram todos os algoritmos relativos à replicação e sincronização**.** As operações são distribuídas a todas as bases replicadas localmente e efetivadas, se for o caso. São também armazenadas e propagadas assincronamente através do módulo de Comunicação. Este módulo também recebe operações remotas, que são repassadas ao módulo de Replicação e Sincronização dos Dados, que por sua vez, efetiva ou não as operações nas bases de dados. Nas próximas seções, todos estes módulos são detalhados.

3.2.1 – Monitoração das bases de dados

A única responsabilidade deste módulo é capturar as operações efetivadas na base de dados e repassá-las ao módulo de Replicação e Sincronização dos Dados. Esse repasse é feito através da utilização de uma interface criada para este fim, implementada pelo módulo de Replicação e Sincronização dos Dados.

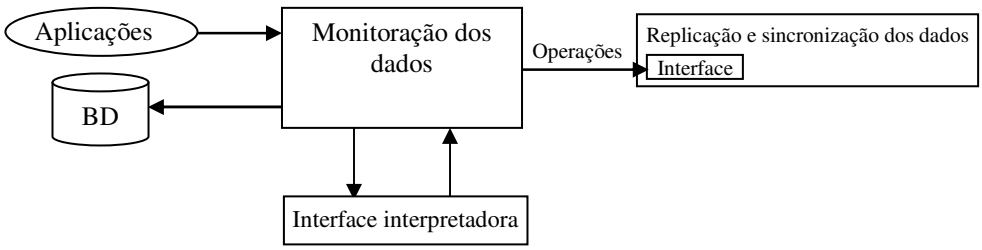


Figura 3.3 – O subsistema de Monitoração de Dados

A monitoração da base de dados é que deve implementar a maior transparência possível aos aplicativos que a utilizam. As técnicas estudadas e testadas para monitorar a base de dados foram três:

a- Interceptação da porta cliente-servidor do banco de dados;
b- Substituição do *driver* de acesso ao banco de dados, seja na forma dedicado (*standalone*) ou como servidor;
c- Leitura de arquivos *pipe* em sistemas antigos que não utilizam SQL.

É evidente que pode ser utilizada qualquer técnica que possibilite o acesso às operações aplicadas às bases de dados. Um exemplo seria o acesso aos *logs* dos bancos de dados, nos casos em que o sistema gerenciador possui este recurso.

Independente de qual técnica utilizada, as operações interceptadas somente são repassadas com uma identificação que seja única em todo o domínio replicado, incluindo a identificação da base de origem. Isto é fundamental no controle das operações de forma geral, porque possibilita a identificação de duplicatas, evita retransmissões desnecessárias durante a propagação, evita aplicar em duplicidade operações na base de dados, não assinala falsos conflitos durante a sincronização e muitas outras utilizações. No caso da aplicação utilizar *trigger* ou *stored procedure* nas expressões SQL, deve-se repassá-las da mesma forma que as demais. O módulo replicador é que irá tratá-las adequadamente. Se o banco de dados já possuir registrados *trigger* ou *stored procedure*, todos os bancos de dados do ambiente de replicação também devem possuí-los.

Devido à sua interferência no fluxo natural de informações entre cliente e servidor, o módulo torna-se crítico em termos de desempenho e fica sendo mais dependente de terceiros (fornecedores de *software* e aplicativos usuários da base de dados), o que se justifica a seguir.

3.2.1.1 – Interceptação da porta cliente-servidor do banco de dados

A grande vantagem no uso da técnica de interceptação da porta cliente-servidor do banco de dados [6] é que, se as configurações dos aplicativos usuários do banco de dados permitirem, este procedimento de replicação fica totalmente transparente para esses usuários. A idéia é simples: todo banco de dados cliente-servidor atende em uma porta específica. Interceptando esta porta, monitoram-se as operações efetivadas na base de dados. Isto pode ser feito de duas formas:

a- Implementando o módulo no ambiente servidor: o módulo atende a porta padrão do banco de dados e redireciona as solicitações para uma outra porta que o banco de dados passa a utilizar. Neste caso, o banco de dados tem que permitir que se altere a porta padrão (o que a grande maioria dos SGBD permite). A vantagem desta forma é que não precisa alterar as configurações cliente e garante-se que todos os aplicativos serão monitorados.
b- Implementando o módulo no ambiente cliente: neste caso configuram-se os aplicativos para acessarem o banco de dados em *"localhost".* O módulo redirecionará as solicitações ao servidor de bando de dados, que estará atendendo na porta padrão. Esta técnica é muito útil na fase de testes, pois permite que o responsável pela implementação deste modelo possa testar as configurações dos módulos RPCD, sem interferir no uso normal do banco de dados.

---

[6] A crítica mais comum a esta técnica é que ela estaria mais para uma "ação *hacker*" do que para uma solução formal. Como exposto ao longo da seção, conhecendo o protocolo *front-end / back-end,* a solução pode ser formalizada. Já as alternativas que substituem a falta do conhecimento do protocolo devem ser vistas como soluções temporárias e usadas com cautela em sistemas comerciais. Tendo acesso aos programas fontes das aplicações, recomenda-se o uso da técnica exposta na seção 3.2.1.2, pois esta é bem mais simples.

Uma vez interceptadas as mensagens, é possível interpretá-las, extrair as operações que estão sendo encaminhadas ao banco de dados e verificar se a operação foi realizada com sucesso.

Para que as mensagens sejam corretamente interpretadas, é fundamental conhecer o protocolo *front-end e back-end* do banco de dados em questão. Infelizmente, este protocolo nem sempre é conhecido ou divulgado pelos fornecedores de *software*. Mesmo os *software opensource* carecem deste tipo de documentação. Até o presente momento, o único banco de dados estudado que fornece este tipo de documentação é o *Postgresql.* De outros bancos *opensource* estudados, como o *Firebird* e *Mysql*, só foi possível conhecer o protocolo estudando os programas fontes dos seus *drives* de acesso. Quando não se tem acesso nem aos programas fonte, só é possível realizar um levantamento mínimo das características dos protocolos de comunicação utilizados, fazendo um "rastreamento" das mensagens à procura de padrões que permitam decodificá-las.

A grande vantagem de se contar com a documentação é que se espera do fornecedor que a mantenha atualizada. Assim, lançadas novas versões, fica fácil alterar o módulo monitor do RCPD. Na ausência da documentação, é necessário repetir todo o trabalho anterior: pesquisar programas fonte ou rastrear mensagens à procura de modificações.

Pode-se fazer a interpretação das mensagens de protocolo, utilizando módulos externos e desenvolvidos por terceiros. Essa é uma abertura, para o caso de algum fornecedor de SGBD prestar o serviço de interpretação sem abrir os fontes ou algoritmos do *driver.* Para isto, o módulo monitor pode exigir uma interface de interpretação externa para utilizar durante o processo de interceptação de mensagens.

Quanto à questão de desempenho, deve-se procurar o menor atraso possível no acesso ao banco de dados, pois do contrário, o tempo de resposta das aplicações será prejudicado. Em termos de implementação, consegue-se isto utilizando *threads* e canais eficientes de *sockets,* conforme se mostra no próximo capítulo.

3.2.1.2 – Substituição do *driver* de acesso ao banco de dados

Muitas vezes não é possível utilizar o ambiente cliente-servidor, pois nem sempre é conveniente instalar um servidor de banco de dados em um *desktop, notebook* ou em equipamentos móveis. Outro motivo é que a interpretação das mensagens, conforme descrito no item anterior, pode ser muito difícil. Substituir o *driver* de acesso é uma boa alternativa para ambos os casos.

Não se trata, obviamente, de reescrever o *driver*. Continua-se utilizando o mesmo *driver* que as aplicações normalmente utilizam, respeitando as licenças e outras questões legais. O que esse trabalho propõe é escrever um programa, em geral na mesma linguagem da aplicação, que encapsula o *driver* de acesso ao BD. A aplicação utiliza este "*driver-shell*", que irá interceptar as operações submetidas ao BD na sua forma original e depois encaminha a solicitação ao *driver* verdadeiro. A resposta também pode ser interceptada e depois encaminhada à aplicação.

Quanto ao aplicativo, talvez seja necessário apenas alterar as suas configurações para que o *driver* interceptador seja utilizado. Caso o sistema não apresente esta flexibilidade (em geral quase todos apresentam), é necessário alterar os programas que utilizam o *driver* ou solicitar ao fornecedor que o faça. Normalmente, tais alterações são muito simples.

Alguns bancos de dados permitem a sua utilização de forma dedicada (*standalone*), fornecendo um "motor" (*engine*) que é incorporado às aplicações. O banco *opensource Hsqldb* é um exemplo. Neste caso, a única solução possível para interceptar as operações é substituir o *driver.* O *Hsqldb* permite também que qualquer aplicação inicie o modo servidor, que passa a atender simultaneamente outras aplicações. A substituição do *driver,* neste caso, ainda é a solução mais simples.

3.2.1.3 – Leitura de arquivos *pipe* em sistemas antigos que não utilizam SQL

É a solução mais trabalhosa, pois exige que as aplicações legadas gravem as alterações feitas nas bases de dados em um arquivo tipo *pipe* ou similar. Para isto, devem-se alterar os programas. A conversão para SQL das alterações efetivadas pelos programas legados nas bases de dados pode ser feita no programa *produtor* das mensagens ou no programa *consumidor* das mensagens. O importante é obter a "versão SQL" da alteração. Se isto não for possível, a única alternativa é acessar diretamente a base de dados e utilizar o paradigma da *transferência de estado.* Isto, por enquanto, não está previsto no modelo RPCD.

A efetivação das operações que irão manter a base legada consistente e replicada pode ser feita pelo mesmo caminho. Mas em geral, existe disponível no mercado programas comerciais e *opensource* que acessam bases antigas, tipo *xBase* e *Paradox,* utilizando SQL. Neste caso, é mais simples utilizar tais programas.

3.2.2 – Replicação e sincronização dos dados

Neste módulo implementam-se as interfaces que recebem as operações do monitor da base de dados e as requisições da conexão remota. O módulo controla também a propagação, a replicação e a sincronização. Todas estas responsabilidades são divididas em quatro subsistemas: *Boundary*, que acomoda as interfaces com os módulos de monitoração de dados e de comunicação; *DatabaseSite,* que realiza o controle da base de dados local do *site*; *RemoteSite*, que implementa o atendimento e controle das bases remotas; e *GlobalModel*, responsável pelo atendimento das exigências do modelo global, que nada mais é do que um modelo conceitual de BD compartilhado por todos os *sites* como um modelo de referência. Opcionalmente, devido à sua importância, este módulo pode dar início às atividades do RPCD. A figura 3.4 abaixo ilustra tais subsistemas. Nas seções seguintes eles são detalhados.

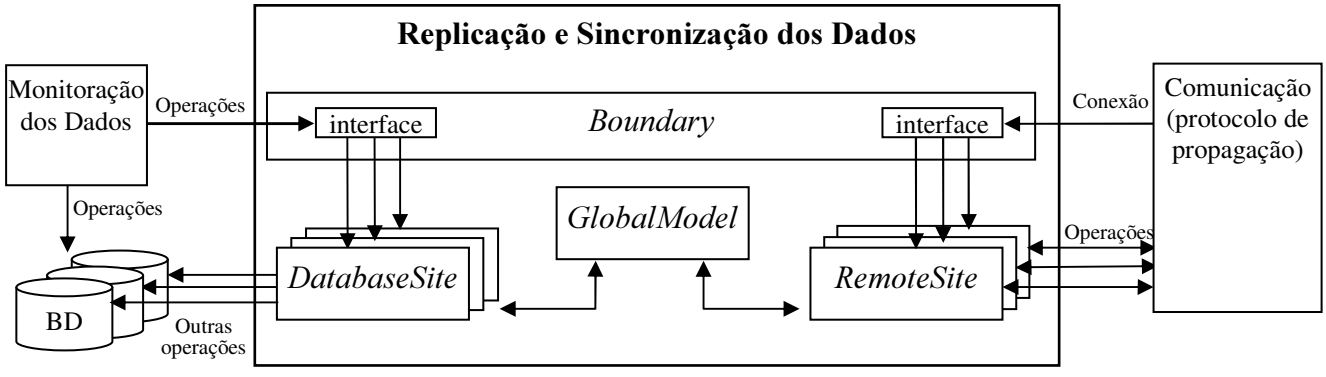


Figura 3.4 – Subsistemas do módulo de Replicação e Sincronização dos Dados

O núcleo da replicação e sincronização é formado pelo *GlobalModel*, os *DatabaseSites* e os *RemoteSites.* Cada *DatabaseSite* é responsável por uma e única base de dados local. Cada *RemoteSite* é responsável por um e único *site* remoto, onde a base é replicada. O *GlobalModel* define o modelo global de dados e cumpre a função de divulgar as operações para todos os *DatabaseSites* e *RemoteSites*.

O subsistema *GlobalModel* cumpre um importante papel na administração do *site*: como baseia-se em um modelo conceitual comum compartilhado por todos os *sites*, isola o que ocorre em termos locais (*DatabaseSite*) do que ocorre nos ambiente remotos (*RemoteSite*). Desta forma, o administrador pode escolher entre um ou mais tipos de SGBD, fazer réplicas internas e ainda utilizar nomes particulares para banco de dados, tabelas, campos e outros elementos do SGBD. A única condição é que se utilize o *GlobalModel* sempre que for replicar uma base de dados, local ou remotamente.

3.2.2.1 – Subsistema de fronteira (*Boundary*)

Este subsistema existe mais por uma questão de *engenharia de software*, que recomenda que todos os serviços disponibilizados por um sistema ao "mundo externo" fiquem em classes específicas para isto[7]. Sua função é basicamente implementar as interfaces exigidas pelos demais módulos.

A interface para o módulo de Monitoração de dados é apenas uma ponte para transferir as operações interceptadas aos *DatabaseSites*. Já a interface exigida pelo subsistema de Comunicação define a transferência de um pedido de conexão, para qualquer programa que procure qual *RemoteSite* pode validar aquele pedido. Tal programa, encontrando o *RemoteSite*, transfere a conexão para ele e encerra sua função. Caso não encontre o *RemoteSite* responsável, após seguir as regras do protocolo, a interface encerra a conexão.

Este subsistema pode oferecer outros serviços e sugere-se que não se desvirtue o seu papel: apenas serviços que interessem aos usuários do RPCD, tais como administradores de banco de dados, gerente de redes, implementadores e outros.

3.2.2.2 – Subsistema *DatabaseSite*

Cada *DatabaseSite* faz a ligação entre uma base de dados local e a sua versão equivalente representada em um modelo global (como se mostra em 3.2.2.3, pode existir mais de um modelo global). Esta ligação significa: receber as operações aplicadas à base de dados e mapeá-las para o modelo global; receber do modelo global operações aplicadas em outros *DatabaseSites* ou *RemoteSites*, mapeá-las para a base de dados e efetivar a operação, se for o caso. Em ambos os casos, o *DatabaseSite* deve implementar o escalonador, ou seja, controlar a escala de

[7] De fato, isto evita que classes que desempenham papéis críticos em uma aplicação tenham que criar mecanismos (como interfaces, por exemplo) para atender aos serviços externos e ao mesmo tempo esconder serviços que só interessam ao sistema em si. As classes de um pacote *boundary sabem* que serviços o mundo externo precisa e *sabem* como usar as classes exclusivas para atender a tais serviços. Este procedimento inclusive é padronizado na UML [Pender,2004-pg.537]

operações aplicadas à base de dados, cuidando da sincronização e conciliação. Seguem-se as justificativas desta proposição.

Todos os paradigmas da replicação são válidos para cada cópia existente no domínio da base replicada, não importando se as cópias são locais ou remotas. Problemas com a sincronização podem acontecer em qualquer cópia e as soluções devem ser aplicadas na cópia afetada, mesmo quando existem algoritmos que consultam outras cópias. Devido a esta especificidade, o modelo aponta como melhor solução apenas um objeto *DatabaseSite* para tratar as questões relativas de uma e única base de dados. Por conseqüência, a escala de operação da cópia é controlada também pelo *DatabaseSite*. Por isto, o *DatabaseSite* é também o responsável por aplicar os algoritmos de sincronização.

Quanto ao modelo global da base de dados, que é apenas conceitual, o *DatabaseSite* faz o mapeamento da semântica local das operações para a semântica global e encaminha ao modelo global a operação mapeada. Recebe também, do modelo global, as operações aplicadas em outras réplicas. Neste caso, ele realiza o mapeamento inverso, antes de efetivar as operações na base local. Este mapeamento, em sua maior parte, deve apresentar uma relação de 1 para 1 (um campo local tem uma e somente uma equivalência com um campo global e vice-versa). Entretanto, também é necessário prever situações de exceção, onde existe equivalência de campos com constantes e funções que estabelecem equivalências mais complexas entre campos e conversão de valores. Exemplos não faltam: um tipo "pessoa Física/Jurídica" no modelo global pode ser "J" ou "F" e na base local "0" ou "1". Deve-se também dar especial atenção aos mapeamentos para expressões SQL com *trigger* ou *stored procedure.* Além da possibilidade de serem bastante complexas, ainda existe a questão da compatibilidade das linguagens aceitas pelos bancos. Finalmente, se a estrutura lógica da base local for muito diferente da estrutura lógica do modelo global, ou se a base local não suportar recursos que outra base suporta, a única solução é o mapeamento conseguir lidar com tais complexidades ou, caso contrário, a replicação não será possível.

As operações são armazenadas na escala de operações e os algoritmos de sincronização decidem entre efetivá-las ou não e, se efetivar, em qual ordem isso é feito. Para evitar muita complexidade – pois as tarefas enunciadas não são muito simples – o modelo indica que a melhor solução é que cada *DatabaseSite* ligue-se apenas a um único modelo global. Caso contrário, para *n DatabaseSites* seria necessário fazer *n* mapeamentos e *n* repasses da mesma operação, o que aumenta a necessidade de controles paralelos. Além disto, modelos globais tendem a ser não muito numerosos.

Finalmente, deve-se utilizar a semântica global ao se armazenar operações. Isto porque muitos algoritmos de sincronização consultam outras filas de operações para tomar decisões. Isto ficará muito mais simples se todas as escalas estiverem obedecendo a um mesmo padrão.

3.2.2.3 – **Subsistema *GlobalModel***

Em termos objetivos, o modelo global é apenas um "contrato", ou seja, todas as réplicas de um domínio de bases replicadas assumem o compromisso de obedecer a um padrão comum. Este padrão determina como são feitos os mapeamentos, como são tratadas as precedências das tabelas, que campos são replicados e outros detalhes ligados aos paradigmas de sistemas de banco de dados. Neste modelo, o subsistema *GlobalModel,* pelo fato de estar ligado a todos os *DatabaseSites* e *RemoteSites,* é o responsável em divulgar para todas as réplicas as operações ocorridas em cada uma das réplicas.

Outra utilização do *GlobalModel* é registrar formalmente a estrutura do modelo global. Sugere-se fazer este registro utilizando expressões formais de SQL (*create table*...e outros). Os dois principais motivos para isto são: em primeiro lugar, muitos algoritmos de sincronização precisam, às vezes, de um banco de dados de caráter temporário com a estrutura típica de uma réplica. O outro motivo é que *sites,* que ainda não possuem banco de dados, podem criar suas estruturas de SGBD com a mesma estrutura do modelo global. Em ambos os casos, armazenar as estruturas do modelo global em expressões SQL facilitaria tais tarefas.

3.2.2.4 – Subsistema *RemoteSite*

Como o nome indica, este subsistema é o responsável pelas réplicas remotas. O *RemoteSite* faz a ligação entre o modelo global do *site* ao qual pertence, com um e único *site* remoto que também utiliza o mesmo modelo global. Esta relação de 1 para 1 justifica-se pelo seguinte motivo: por questões de sincronismo, ambos os lados devem manter um controle da fila de operações intercambiadas, fila esta que sempre tende à igualdade. Na hipótese do *RemoteSite* conectar-se a mais de um *site*, ele deveria manter um controle de fila para cada *site*, além de tratar as questões de segurança particulares de cada *site*. Se, por outro lado, disponibilizasse mais de um modelo global, deveria haver um controle de fila para cada modelo. Em resumo, seria necessário um total de N x M controles para *sites* e modelos globais. O aumento de complexidade do modelo não compensaria eventuais ganhos em configurações ou recursos de máquina.

Exceto pela conexão inicial, que é responsabilidade do subsistema *Boundary*, todos os demais serviços do protocolo de comunicação são implementados ou utilizados pelo *RemoteSite*. Como servidor, deve validar o cliente que solicita a conexão. Como cliente, deve ter a iniciativa de conectar-se a um servidor. O papel assumido pelo *RemoteSite* define-se durante o planejamento do domínio replicado. Apesar da previsão de se manter longo tempo desconectado, havendo disponibilidade, o par de *RemoteSites* pode ficar permanentemente conectado, trocando informações somente quando houver um novo conteúdo para transmissão.

Todas as operações recebidas do *site* remoto são repassadas ao modelo global, assim como as operações recebidas do modelo global são transmitidas ao *site* remoto. Como as operações controladas pelo *RemoteSite* não são aplicadas a nenhum banco de dados, não existe a necessidade de algoritmos de sincronização. Deve-se apenas controlar uma fila de operações para fins de transmissão e recepção.

Uma importante responsabilidade do *RemoteSite* é com a distribuição de tabelas do modelo global e a respectiva **fragmentação de dados** horizontal**.** O *RemoteSite* é que aplica os critérios de fragmentação horizontal, determinando assim quais informações serão replicadas remotamente. Obviamente, o *site* remoto deve estar configurado para que todas as informações selecionadas pela fragmentação possam retornar ao *site* de origem, pois do contrário a replicação ficará comprometida. A fragmentação pode se tornar uma tarefa complexa, principalmente no caso de inserções, precedência de arquivos (tabelas com relacionamento 1-n em cascata), tabelas com muitas chaves estrangeiras e outras características inerentes a sistemas mais complexos. Mesmo assim, o *RemoteSite* é o candidato natural para absorver tais complexidades.

Torna-se importante demonstrar a relação entre a forma de o *RemoteSite* administrar a propagação e o uso da topologia estrela, para a qual o modelo foi planejado. Pelo exposto anteriormente, o *RemoteSite* conecta-se somente a um outro e único *RemoteSite*, configurando assim uma topologia *peer-to-peer*. Mas seja, por exemplo, um determinado banco de dados representado por um *DatabaseSite* e devidamente mapeado para um modelo global. Podem existir diversos *sites* remotos que replicam este banco de dados, desde que configurados pelos respectivos pares de *RemoteSites* (local e remoto), sempre numa base um para um entre o *GlobalModel* e cada *RemoteSite*. Essa forma de replicação configura então uma topologia estrela. A figura 3.5 ilustra este aspecto.

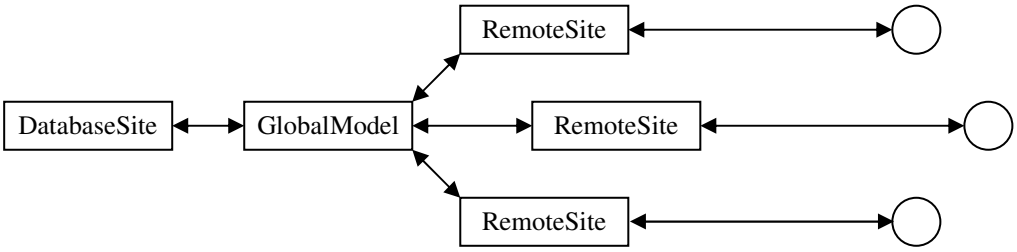


Figura 3.5 – Topologia estrela do ponto de vista do *DatabaseSite*

3.2.2.5 – Coordenação entre os módulos

Mecanismo de propagação: quando uma operação em um determinado BD é capturada pelo módulo *Monitoração dos Dados* (ver figura 3.4), esta é repassada para o módulo *DatabaseSite*, que representa o BD no ambiente de replicação . *DatabaseSite*, por sua vez, faz o mapeamento da operação para o modelo global, registra a operação na escala de operações do BD e repassa a operação para o módulo *GlobalModel*. Este módulo repassa a todos os demais módulos *DatabaseSites* e *RemoteSites* a operação recebida. Cada *DatabaseSite* faz então o mapeamento inverso e aplica a operação no BD pelo qual é responsável. Cada *RemoteSite,* assincronamente, repassa a operação para o seu par localizado em um algum *site* remoto. O mesmo *RemoteSite*, por sua vez, recebe do *site* remoto as operações efetivadas remotamente e as repassa ao módulo *GlobalModel* que, por sua vez, repassa aos demais módulos *DatabaseSites* e *RemoteSites*, fechando assim o ciclo de propagação.

Processo de conciliação e sincronização: quando o *DatabaseSite* recebe uma operação de *GlobalModel*, após realizar o mapeamento da representação global para local, inicia-se o processo de sincronismo do BD. Primeiro, a operação é inserida na escala de operações, utilizando-se as regras estabelecidas para o escalonamento. Depois, a escala é analisada em função das regras, que procuram detectar as possibilidades de conflito com a operação. Não havendo conflitos, a operação será aplicada. Havendo conflitos, aplicam-se então as regras de resolução de conflitos. Essas regras podem ser aplicadas de forma automática ou então os conflitos podem ser reportados ao usuário para que os resolva. Conforme tratado na seção 2.8.1.4, havendo um conhecimento semântico das operações, a serialização das operações na escala pode admitir comutatividade (casos 1 e 2 da tabela 2.1), o que simplifica muito a lógica da sincronização. Não existindo o conhecimento semântico, é conveniente não adotar a comutatividade (caso 3 da tabela 2.1) porque, mesmo apresentando uma lógica mais complexa, o procedimento indicado aumenta as garantias de convergência do BD replicado.

Consistência e convergência: a uniformidade das regras e a precisão dos algoritmos utilizados no processo de conciliação e sincronização procuram garantir que a escala de operações seja idêntica em todos os *DatabaseSites*, garantindo assim a consistência de cada BD replicado. O módulo *GlobalModel* possui acesso a todas as escalas aplicadas em cada BD replicado, seja localmente através dos *DatabaseSites* ou aquelas controladas pelos *RemoteSites*. Desta forma consegue controlar e fornecer garantias quanto à convergência das bases replicadas a partir da análise de todas as escalas existentes.

É evidente que os algoritmos responsáveis pelo sincronismo, consistência e convergência podem atingir altos níveis de sofisticação. O que foi descrito anteriormente, entretanto, simplesmente assinala onde e em que momento no modelo, tais algoritmos podem ser aplicados e quais cuidados levar em conta.

3.2.3 – Comunicação (protocolo de propagação)

Este módulo é o responsável pela propagação propriamente dita e para isto utiliza um protocolo próprio. Este protocolo é chamado de "Protocolo RPCD". Como as exigências dos algoritmos de sincronização podem ser numerosas e complexas, criou-se um protocolo bastante flexível e versátil para adaptar-se a tais exigências. Por esta razão, utilizou-se XML (*eXtended Markup Language*)[8] para especificar as PDUs e objetos como parâmetros das primitivas. Como o modelo é planejado de forma independente de linguagens ou sistema operacional, soluções como o RMI (Java) e DCOM (MS) não foram consideradas. Como o objetivo é utilizar a Internet como meio de comunicação, a escolha natural é utilizar o protocolo TCP/IP para estabelecer as conexões.

As primitivas são divididas em dois grupos: as que cuidam do estabelecimento e término da conexão e as específicas para a atividade de replicação. Os papéis de *cliente* e de *servidor*

[8] O XML, por definição e padronização, oferece duas importantes garantias muito úteis quando utilizadas em protocolos: uma é ser "bem formado", significando que estruturalmente está correto. A outra é ser "válido", ou seja, o conteúdo previsto é validado em termos de valores numéricos, caracteres, texto e outros parâmetros, tais como repetições permitidas, combinações de informações e outros. Para isto, utiliza-se *XML Schema* ou *DTD (Data Type Definition).* Para maiores informações, pode-se consultar [Holzner,2001], capítulos 2 a 5.

ficam bem caracterizados no primeiro grupo: a iniciativa do cliente em iniciar a conexão, enquanto o servidor escuta permanentemente uma porta TCP e realiza a autenticação do usuário. Já no caso do segundo grupo, que se refere às atividades de replicação em um ambiente *multimaster,* fica mais caracterizado uma comunicação *peer-to-peer*. Neste caso, o protocolo não faz distinção alguma entre os participantes da comunicação. Havendo disponibilidade de conexão, a iniciativa de solicitar serviços pode ser periodicamente permutada.

3.2.3.1 – Serviços de conexão

Os serviços básicos de conexão do protocolo RPCD visam estabelecer e terminar de forma ordenada ou abrupta uma conexão entre dois *sites*. Esses serviços incluem ainda a autenticação do cliente junto do servidor. Nas primitivas de serviços adotou-se o prefixo *C* para designar uma primitiva do cliente e o prefixo *S* para designar uma primitiva do lado do servidor. A ausência desses prefixos indica que a primitiva pode ser enviada por ambos.

Estabelecimento da associação – serviço confirmado, incluindo as seguintes primitivas:

C- AssociationRequest:	Requisição de associação Parâmetros: *url* e porta do servidor
S- AssociationIndication:	Indicação de associação solicitada Parâmetro: objeto *Socket* da associação
S- AssociationResponse:	Resposta à associação solicitada Parâmetro: verdadeiro/falso
C- AssociationConfirm:	Confirmação da associação aceita Parâmetro: verdadeiro/falso

Autenticação de usuário – serviço confirmado, incluindo as seguintes primitivas:

C- LoginRequest:	Requisição de *login* Parâmetro: nome do *login* e senha
S- LoginIndication:	Indicação do *login* solicitado Parâmetro: nome do *login* e senha
S- LoginResponse:	Resposta ao login solicitada Parâmetro: verdadeiro/falso
C- LoginConfirm:	Confirmação do login solicitado Parâmetro: verdadeiro/falso

Término abrupto de associação – serviço não-confirmado, incluindo as seguintes primitivas:

AbortRequest:	Requisição de término de qualquer processo Parâmetro: motivo (informativo)
AbortIndication:	Indicação de término de qualquer processo

Parâmetro: motivo (informativo)

Término ordenado de associação – serviço não-confirmado, incluindo as seguintes primitivas:

DisconnectRequest: Requisição de término da conexão

DisconnectIndication: Indicação de término de qualquer processo

Primitivas de uso local:

C- SuspendAssociationTry: Suspender tentativas de associação e desativar protocolo

ErrorIndication: Indicação de erro na solicitação de serviços
Parâmetro: descrição do erro

AttemptIndication: Indicação da tentativa de associação
Parâmetro: número da tentativa.

Estas duas últimas primitivas são específicas da forma como o protocolo foi definido. Para não exigir que o usuário "se lembre" de realizar a sincronização dos dados, o protocolo fica procurando periodicamente pelo servidor se a associação ainda não tiver sido estabelecida. Se por algum motivo isto não for desejável, utiliza-se a primitiva *SuspendAssociationTry,* que desativa as tentativas e o protocolo RPCD. Depois disto, o protocolo tem que ser reativado, manual ou automaticamente, por algum programa. Já a primitiva *ErrorIndication* foi definido para indicar a tentativa de chamar algum serviço quando o estado do protocolo não o admite. Esse é um serviço conveniente, já que a grande maioria dos serviços é assíncrona e por isto aumenta a possibilidade de ações inadequadas por parte das aplicações como, por exemplo, chamar um *SuspendAssociationTry* se a conexão já existir.

3.2.3.2 – Serviços para replicação e sincronismo

Estes serviços são especificados para fins gerais da replicação e sincronismo. São mais voltados ao *comportamento dos sites* do que *ao que devem fazer* (procedimentos). Os parâmetros das primitivas, que são objetos complexos, indicam o que deve ser feito e com quais valores. Enquanto as primitivas de serviço são processadas pelo protocolo RPCD, os objetos são criados e mantidos pelas aplicações usuárias do protocolo. O único compromisso das aplicações é utilizar a primitiva adequada ao tipo de procedimento solicitado à outra parte. A seguir essas primitivas são descritas, bem como se apresentam alguns exemplos de objetos importantes para o protocolo, sugerindo-se situações para sua utilização. Como explicado anteriormente, não há distinção (cliente-servidor) entre as partes.

Serviço de requisição de informações – serviço confirmado, que inclui as seguintes primitivas:

InquiryRequest: Requisição de informações
Parâmetros: texto descritivo e objeto portador do procedimento.

Este serviço aguarda uma resposta de confirmação. Qualquer serviço solicitado por esta primitiva visa recuperar uma simples informação, não devendo alterar nada no *site* para onde

foi encaminhado o procedimento. Na execução desse serviço não é necessário manter informações a respeito do estado do procedimento, tais como: "já foi feito isto", "não está autorizado" e outros. Seguem algumas classes importantes:

Echo: envia um texto qualquer e aguarda recebê-lo de volta.
Esta não é uma versão do "ping" para testar se a associação está ativa. O objetivo é verificar se o conjunto de caracteres encaminhado retorna íntegro. Como existem muitos tipos de "páginas de código de caracteres" utilizados pelos sistemas operacionais, linguagens de programação e SGBD, caso o conjunto não volte íntegro, significa que há problemas com os códigos (ASCII x EBCDIC x ISO e outros). Um texto típico encaminhado pode ser <áéíóúâêîôûàèìòùãõü> e a versão em maiúsculas. Normalmente estes são os caracteres mais sensíveis a problemas.

Time: solicita o horário do *site* remoto.
Algoritmos que utilizam o horário precisam em geral desta informação

TimeWait: solicita o tempo de espera que o *site* remoto pode aguardar por uma resposta.
Para fins de controle próprio, é importante saber quanto tempo o *site* remoto pode esperar por respostas.

InquiryIndication: Indica informações requisitadas
Parâmetros: texto descritivo e objeto portador do procedimento.

Os parâmetros dessa primitiva são semelhantes aos da primitiva *InquiryRequest*, tendo sido recuperados da PDU correspondente transportados entre os *sites.*

InquiryResponse: Resposta à solicitação
Parâmetros: resposta ou objeto resposta

O *site* receptor providencia as informações solicitadas e as retorna ao *site* solicitante. Essa primitiva segue as mesmas orientações expostas em *InquiryRequest.* Seguem algumas classes importantes:

Echo: envia de volta o texto recebido.
Como explicado anteriormente, o objetivo é testar a "página de código de caracteres". Para uma boa qualidade do teste, os caracteres recebidos devem passar por algum tipo de manipulação, como por exemplo, gravar em algum lugar e depois ler de volta e outras ações semelhantes.

Time: retorna o horário do *site*.
Detalhes como fuso horário, tipo de hora utilizada e outros são levados em conta. Estes detalhes devem ser possíveis de se configurar no objeto.

TimeWait: retorna o tempo de espera que se está utilizando.
É o tempo de espera que o foi configurado para utilização.

InquiryConfirm: Confirmação da solicitação
Parâmetros: resposta ou objeto resposta

Os parâmetros dessa primitiva são semelhantes aos da primitiva *InquiryResponse*, tendo sido recuperados da PDU correspondente transportados entre os *sites.*

Serviço de configuração – serviço confirmado, incluindo as seguintes primitivas:

ConfigRequest: Requisição de alteração de configurações
Parâmetros: comentário e objeto portador do procedimento.

Este serviço aguarda uma resposta de confirmação verdadeiro/falso, que indica se a configuração foi acatada. Qualquer serviço solicitado por esta primitiva implica em uma mudança nos parâmetros do protocolo ou da replicação e sincronismo. Como afeta o funcionamento de todo o sistema, seu uso deve cercar-se de cuidados em ambos os lados. Exceto pela sugestão que se segue, provavelmente esse serviço será pouco utilizado.

TimeWait: altera o tempo de espera dos procedimentos.
Caso o *site* julgue que haverá demora em algum procedimento, pode solicitar a mudança para maior do tempo de espera do *site* remoto. Boas conexões entre os *sites* podem levar a diminuir este tempo.

ConfigIndication: Indica alterações na configuração
Parâmetros: comentário e objeto portador do procedimento

O *site* receptor deve avaliar a configuração antes de implementá-la.

ConfigResponse: Resposta à requisição de mudança de configuração
Parâmetro: verdadeiro/falso

O *site* receptor retorna verdadeiro se implementou a configuração e falso, caso contrário.

.

ConfigConfirm: Confirmação da alteração de configuração
Parâmetro: verdadeiro/falso

Serviço de transferência de dados – serviço não-confirmado com as seguintes primitivas:

DataRequest: Requisita o envio de dados
Parâmetros: comentário (informativo) e objeto dado.

O envio de operações e outras indicações relacionadas aos dados são feitas através dessa primitiva. Parte-se do princípio que os dados serão sempre aceitos pelo *site* remoto. Segue a classe mais importante:

Lista de Op: uma lista de instâncias da classe Op, que representa uma operação.
Cada instância de *Op* contém todas as informações relativas à operação, tais como identificação, tabela, tipo de operação, *timestamp*, *site* de origem e quaisquer outras informações necessárias aos algoritmos de replicação e sincronização.

DataIndication: Indica o recebimento de dados
Parâmetros: comentário (informativo) e objeto dado.

O *site* receptor deve aceitar incondicionalmente os dados (objetos *Op*, como descrito anteriormente). Como o protocolo garante um mínimo de integridade às informações (ver início da seção 3.2.3), outras consistências ficam a cargo do módulo de Replicação e Sincronismo dos Dados.

Serviço de troca de papéis – serviço não-confirmado, com as seguintes primitivas:

ServiceRequest: Requisita a inversão dos papéis
Parâmetros: não há.

Durante o intercâmbio de informações, sempre um dos *sites* tem a iniciativa de requisitar serviços. Quando não há mais nada a fazer – o que pode ser medido por um tempo pré-determinado – ele requisita a inversão dos papéis, permitindo que o outro *site* passe a requisitar qualquer serviço.

ServiceIndication: Indica a inversão dos papéis
Parâmetros: não há.

Como explicado anteriormente, essa primitiva indica que a iniciativa de solicitar serviços passou ao *site* receptor desta primitiva. Quando não houver mais nada a fazer, o *site* pode requisitar uma nova inversão de papéis. Este procedimento pode durar indefinidamente, enquanto houver disponibilidade de conexão.

3.2.3.3 – Formato das PDUs

As Unidades de Dados do Protocolo RPCD (PDUs), que são geradas em resposta às primitivas descritas na seção anterior, são especificadas em XML, conforme mostrado a seguir. Os comentários sobre sua utilização encontram-se no próprio XML.

```
<!-- toda PDU do RPCD possui a tag rpcd como raiz e um atributo nseq que contém uma
seqüência de inteiros -->
<rpcd nseq="xxxxx">

    <!-- todas as tags que se seguem devem ser únicas em uma PDU. Aqui estão
        agrupadas apenas para facilitar a leitura -->
    <!-- Não existe PDU para associationRequest, uma vez que não há necessidade de
        passagem de parâmetros iniciais de configuração na associação entre os
        processos de aplicação -->
    <associationResponse confirmed="yes/no"/>

    <loginRequest login="xxxx" passwd="xxxx"/>
    <loginResponse confirmed="yes/no"/>
    <abortRequest>mensagem</abortRequest>
    <disconnectRequest/>

    <inquiryRequest>
        <!-- o aplicativo deve devolver o texto sem alterações -->
        <echo>texto</echo>
        <!-- o aplicativo pede a data e hora remota -->
        <time/>
        <!-- o aplicativo pede o tempo de espera utilizado pelo cliente remoto-->
        <timewait/>
```

```
    </inquiryRequest>

    <inquiryResponse>
        <!-- o aplicativo devolve o texto sem alterações -->
        <echo>texto</echo>
        <!-- o aplicativo devolve a data e hora -->
        <time/>
        <!-- o aplicativo devolve o tempo de espera que utiliza -->
        <timewait/>
    </inquiryResponse>

    <configRequest>
        <!-- o host local solicita ao host remoto mudança no tempo de espera -->
        <timewait time="3000"/>
        <!--a tag que se segue é um exemplo da liberdade que se tem em criar novos objetos-->
        <!-- indicações de arquivos para download utilizado em configurações e programas -->
        <download from="localhost:/ppgee/v001.jpg" to="ppgee/v001.jpg"/>
    </configRequest>

    <!-- o host local confirma ou não que atendeu à alteração solicitada -->
    <configResponse confirmed="yes/no"/>

    <dataRequest>
        <!-- transferências de operações -->
        <op id="xxx" table="xxxx" type="U/I/D" originSite="xxx" timeStamp="xxx">
            expressão
        </op>
        <!--as tags que se seguem são exemplos da liberdade para se criar novos objetos -->
        <!-- indicações de arquivos para pastas replicadas -->
        <download folder="produtos" file="v001.jpg"/>
        <!-- transferências de operações capturadas internamente pelo RpcdDriver-->
        <opdriver dbType="paradox" dbName="/P/Sist/x"  dbPort="8029"
         table="produto" timeNow="123456">
            expressão SQL pronta
        </opdriver>
    </dataRequest>

    <!-- o host local solicita a inversão de papéis -->
    <serviceRequest/>

</rpcd>
```

3.2.3.4 – Operação do protocolo

A figura 3.6 ilustra a operação do protocolo. Utilizou-se a notação de Diagrama de Seqüências da UML para representar graficamente a operação. As chaves explicativas não fazem parte da UML, sendo utilizadas apenas para uma melhor compreensão do diagrama. O diagrama também não está rigorosamente dentro dos padrões UML: para deixar a diagramação mais clara, omitiram-se os diagramas de interação operacional, tais como operações condicionais, *loops* e outros.

A primeira seqüência é relativa aos serviços de associação (indicado pela chave à direita). Inicia-se com o cliente solicitando o estabelecimento da associação (indicado pela chave à esquerda) através do envio da primitiva de serviço *AssociationRequest.* O módulo *ClientAssociation* fica tentando estabelecer uma associação com o módulo *ServerAssociation* até conseguir ou,desiste da tentativa, se o cliente suspender esse procedimento, utilizando o serviço *SuspendAssociationTry*. Obtendo sucesso, o módulo *ClientAssociation* envia um *AssociationConfirm* ao cliente e passa para o próximo passo que é a autenticação do usuário. A autenticação inicia-se com o cliente utilizando o serviço *LoginRequest* e tendo como retorno o *LoginConfirm.* Observe que a qualquer momento pode haver um término abrupto de conexão, ou com o cliente usando o serviço *AbortRequest,* ou por causa de *time-out* ou por iniciativa do servidor. Concluída com sucesso a primeira seqüência, passa-se à segunda que é dedicada à propagação e sincronismo.

Os três principais grupos de serviços – configuração, requisição de informações e transferência de dados – podem ocorrer em qualquer ordem e a qualquer momento. A iniciativa parte do *site* que estiver no estado *WaitingServices* (assinalado na figura 3.5). Inicia-se com o lado servidor (módulos *RemoteSite* e *ServerAssociation*), que a qualquer momento pode inverter os papéis utilizando a primitiva de serviço *ServiceRequest*. Desta forma, a iniciativa passa para o lado cliente que, a qualquer momento, pode solicitar uma inversão de papéis. Esta inversão pode continuar indefinidamente. A qualquer momento, o *site* que estiver com a iniciativa pode também terminar a associação de forma abrupta, utilizando o serviço *AbortRequest*, ou de forma ordenada, utilizando o serviço *DisconnectRequest*. Existe ainda o serviço local *ErrorIndication*, explicado na seção anterior, que pode ocorrer a qualquer momento e em qualquer uma das seqüências.

Serviços de gerenciamento do protocolo não constam do modelo, pois estão mais ligados à implementação. Em sua grande maioria, esses são serviços locais tais como: iniciar e parar o módulo de comunicação, iniciar interfaces para capturar eventos durante o processamento do módulo, interfaces que permitam usuários ou administradores interferirem na resolução de conflitos e outros serviços.

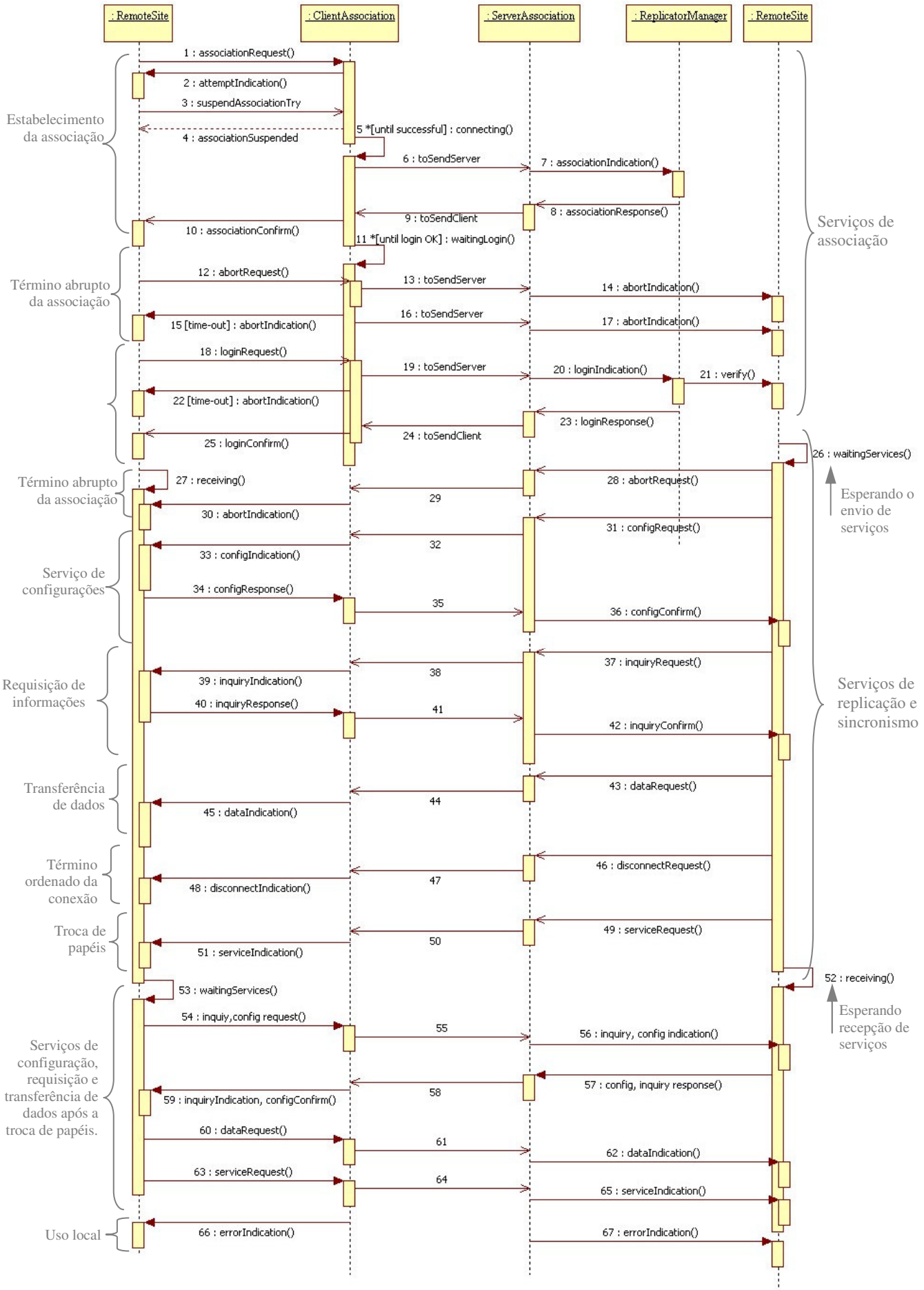


Figura 3.6 – Operação do protocolo RPCD

3.3 – Soluções do modelo RPCD para situações de transição

Entenda-se por transição as situações que sempre ocorrem fora da rotina normal do processo. Nesse trabalho são consideradas cinco situações bastante comuns: início de configuração, realização da primeira réplica, alterações de estrutura de tabelas, abandono de uma réplica irrecuperável e desativação de uma réplica. Parte-se do princípio que o modelo global é administrado por um só responsável (ou um administrador que coordene todos os administradores do ambiente de replicação) e que todos os *sites* irão obedecer a este modelo.

Início de configuração: a primeira exigência é configurar o modelo global. Esta é a configuração mais importante de todo o processo de inicialização e que define todo o futuro da replicação. Implica conhecer bem os bancos de dados envolvidos na replicação, assim como os aplicativos que os acessarão e as características dos *sites* que possuirão réplicas. As configurações do modelo global ficam registrados no sub-sistema *GlobalModel*. Após configurar o modelo global, cada *site* torna-se autônomo o suficiente para estabelecer quais bases locais serão replicadas e como serão mapeadas para o modelo global[9]. Isto é feito configurando-se os *DatabaseSite.* A etapa seguinte é definir quais *sites* remotos irão possuir réplicas e com qual fragmentação. Isto é feito configurando-se os *RemoteSite*. Os *sites* remotos, por sua vez, devem ser configurados com a contrapartida dos *RemoteSite,* que nada mais são que outro objeto *RemoteSite* configurado "ao inverso", ou seja, mapeado do modelo global para uma base local e replicando esta base com o *site* da fonte original dos dados. Evidentemente, o mesmo administrador pode se responsabilizar por todos esses aspectos.

Realização da primeira réplica: após a etapa anterior, deve-se carregar a primeira réplica no site remoto. Logo que iniciar a replicação, a escala de operações começa a funcionar nos respectivos *RemoteSite*. Remotamente, o *DatabaseSite* já deve existir e a base real também, a qual deverá estar a princípio vazia. Caso não existam, o módulo de replicação pode criar a base real com o banco de dados que o administrador determinar (seguindo o *lay-out* do modelo global) e facilmente construir o respectivo *DatabaseSite.* Para manter a coerência com a propagação por *transferência de operação*, a base real será preenchida com um conjunto de comandos *"inserts"* que o *site* principal envia. Após esta etapa, os *sites* iniciam o modo normal de funcionamento. Interrupções ao longo deste processo têm que ser tratadas, para evitar que a réplica já fique comprometida logo de início. No caso de grandes bases de dados, pode-se utilizar a flexibilidade do modelo em criar réplicas locais ou mesmo em uma estação da rede local. Como é assíncrono, finalizado o processo, a replicação é interrompida localmente (é como se o site houvesse se "desconectado" da rede). Nesse momento o administrador deve alterar as configurações de endereçamento, enviar todos os dados ao cliente real e assim que ele se conectar, a replicação assíncrona continua normalmente.

Alteração na estrutura de tabelas: entenda-se por alteração qualquer inclusão, exclusão ou alteração de campos ou tabelas em alguma base local. Quando isto ocorrer, tem que se avaliar a conseqüência dessa alteração. Se nada for feito, ou seja, se o modelo global desconhecer esta alteração, então a alteração ficará restrita à base local e não se refletirá nas respectivas réplicas. O administrador tem a responsabilidade de avaliar esta questão, para não comprometer a qualidade das réplicas. Se a alteração de campos ou tabelas tiver que ser replicada, deve-se alterar o modelo global. O comando de alteração entra na escala de

[9] É importante observar a conseqüência deste princípio. Como cada *site* pode definir o que será replicado a partir da sua base local, ao fazer isto assume uma "autoridade implícita" sobre a réplica da base local. Esta informação pode ser utilizada em diversas situações: como mais um critério na solução de conflitos, como informação complementar sobre qual *site* é a fonte original dos dados e quaisquer outras situações onde esta informação pode ser relevante.

operações, mas o módulo de Replicação e Sincronismo dos Dados deve tratá-la de forma diferenciada. Para isto utiliza as primitivas *InquiryRequest* e *ConfigRequest* para propagar corretamente a alteração das configurações de tabela. Remotamente, cada *site* também deve cercar-se de cuidados, antes de acatar as alterações. Qualquer problema que inviabilize a alteração tem que ser encaminhado ao administrador do modelo global, através de qualquer mecanismo de mensagens de erro.

Abandono uma réplica: em situações onde a qualidade da réplica, por qualquer motivo, esteja comprometida, a única solução é, às vezes, descartá-la. Neste caso, todas as informações da réplica e da escala de operações são eliminadas e deve-se solicitar um novo conjunto de *"inserts"*. Seguem-se, então, os mesmos procedimentos descritos em *"Início de configuração"*. Entre outras alternativas, este processo pode ser feito utilizando as primitivas *InquiryRequest* e *ConfigRequest.*

Desativação de uma réplica: para desativar uma réplica deve-se basicamente retirar os arquivos de configuração e reconfigurar aplicativos e o banco de dados aos parâmetros usuais, o que exige a interferência do administrador local. Uma questão que se coloca é quando o administrador do *site* principal não quer deixar réplicas no *site* remoto que será desativado. O administrador pode então utilizar a primitiva *ConfigRequest* para comandar a eliminação dos dados remotamente e desativar a replicação desabilitando os respectivos *RemoteSite*. Evidentemente, nesse caso, as aplicações remotas começarão a apresentar erros ou não apresentarão as informações esperadas, o que torna inevitável a interferência do administrador local.

3.4 – Limitações do modelo RPCD

Como este modelo baseia-se em alguns paradigmas utilizados em "Sistemas de Banco de Dados Múltiplos" (ver seção 2.3), é natural que possua as limitações inerentes a tais sistemas. As principais limitações devem ocorrer no mapeamento entre as bases locais e os modelos globais. Dependendo do quanto houver diferença entre as estruturas lógicas dos bancos replicados, ou entre os recursos dos bancos e outros fatores como os expostos na seção 2.3.1, o mapeamento poderá ser muito complexo ou mesmo impossível. Neste caso, a única solução é rever os modelos globais e locais.

Não ocorrendo a situação exposta anteriormente, a principal limitação do modelo RPCD é que ele não é adequado para realizar replicações por transferência de estado (ver seção 2.7.4.2). Ele foi planejado para trabalhar de forma assíncrona e com transferência de operações (ver seção 2.7.4.1). Por causa disto, o modelo estabelece um alto grau de autonomia ao *site* na solução de problemas relativos à replicação e sincronização dos dados. A comunicação entre *sites* e o protocolo de comunicação enfatizam apenas a troca de informações. Uma vez intercambiadas as informações, cada *site* decide isoladamente como irá tratá-las. Na replicação por estado, muitos algoritmos precisam fazer exaustivas comparações entre arquivos replicados, exigindo uma comunicação com característica síncrona. Além disto, na forma como o modelo foi desenhado – isolando os arquivos locais do modelo global – a comparação "byte a byte" de informações é totalmente inadequada. Pode-se concluir que para utilizar transferência de estado, seria necessária uma outra modelagem.

Outra limitação do modelo, de caráter mais administrativo do que técnico, ocorrerá com o crescimento do número de bases replicadas, dos modelos globais e dos *sites* pertencentes ao domínio. O modelo adotado prevê relações de 1 para 1, entre cada base real e cada modelo global para cada *site* e com as suas respectivas configurações. Seja, por exemplo, um ambiente de replicação com três *sites*, um modelo global e cada *site* com apenas uma base real: serão necessários seis arquivos de configuração (1 modelo global + 1 base-modelo para cada um dos três *sites*). Acrescentando-se mais um modelo global, já será um total de doze arquivos de configuração (2 modelos + 2 base-modelos para três *sites*). Se houver mais uma base real, já serão dezoito arquivos (2 modelos + 4 bases-modelos para três 3 *sites*). O crescimento dos arquivos de configuração aumenta a possibilidade de redundâncias ou parâmetros inconsistentes. Neste caso, deveria existir um módulo para fazer as devidas consistências em todo o domínio da replicação. Este módulo deveria levar em conta a característica assíncrona do domínio e, provavelmente, exigiria a existência de um *site* principal para utilizar como referência básica. Isto, por enquanto, não está previsto no modelo.

3.5 – Possibilidades e desenvolvimentos futuros

Até o momento, existem duas possibilidades para estudos ou desenvolvimentos futuros com relação ao modelo proposto: utilizar o modelo em topologias em malha e utilizá-lo na criação de tabelas em uma modelagem multidimensional.

Topologias em malha: no item 3.2.2.4 foi realçado o aspecto de uso da topologia estrela do ponto de vista de uma base replicada. Entretanto, o modelo permite uma autonomia razoável para cada *site*, desde que o mesmo obedeça ao modelo global. Isto permite que uma *base A* seja replicada para o *site B*, que por sua vez pode replicar esta base ao *site C* e assim sucessivamente. Nada impede que, em determinada configuração, o *site X* queira replicar a

base A para o *site A* que é, na realidade, o dono original da base. Se isto fosse permitido, estaria configurada uma rede em malha do ponto de vista da *base A* e o "dono original" ficaria descaracterizado. Apesar de possível, esta situação tem que ser analisada mais profundamente em termos de convergência de dados, bem como suas conseqüências na confiabilidade e no sincronismo das réplicas. Para evitar esta situação ainda não estudada, o modelo exige que se mantenha o conceito de *site* "dono original" da base de dados (utilizada em várias partes da descrição do modelo), de forma a não aceitar réplicas de bases que sejam originalmente do próprio *site*. Com isto, a topologia resultante continuará sendo estrela, com possíveis extensões para uma topologia hierárquica.

Tabelas na modelagem multidimensional: esta possibilidade é muito mais teórica do que a anterior, mas analisando o modelo verifica-se que é uma possibilidade. Sem aprofundar-se no conceito de modelagem multidimensional[10], pode-se imaginar uma "tabela fato" como sendo uma tabela do modelo global. Cada tabela representativa de uma dimensão replica seus dados com base neste modelo global. Todas estas tabelas redirecionam a réplica para o mesmo *site* central. Existiria então um *DatabaseSite* que manteria o modelo global em termos de base real e seria a tabela utilizada pelas aplicações do modelo de negócios. Além da relação 1 a 1 entre campos locais e globais, o modelo prevê o uso de funções de conversão nos casos mais complexos. É neste momento que entra a possibilidade de utilizar o modelo na modelagem multidimensional. As funções passariam a ser regra e não exceção. Os algoritmos de sincronismo deveriam remontar as operações do *DatabaseSite,* em função das filas existentes em todos os *RemoteSite* (cada um representaria um dimensão). Após a operação estar completa, ela seria aplicada ao *DatabaseSite* que representa a "tabela fato". Assim, de forma assíncrona, as informações do modelo de negócio seriam sempre atualizadas, toda vez que os *sites* participantes do domínio se conectassem ao *site* central. A figura 3.7 reproduz a figura 3.5, mas apresenta a mesma situação do ponto de vista da modelagem multidimensional.

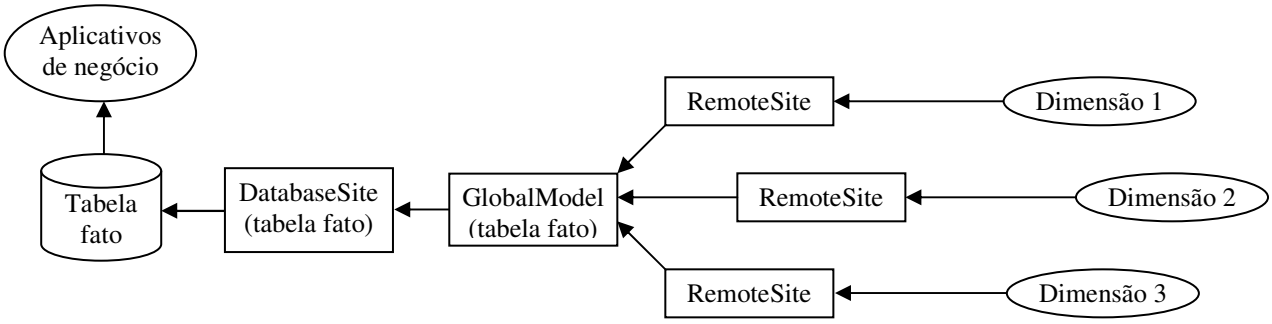


Figura 3.7 – O modelo RPCD utilizado em modelagem multidimensional

As mudanças mais significativas mostradas na figura 3.7 se referem à terminologia usada – significando com isto que os algoritmos envolvidos não se preocupam apenas com a replicação – e no fluxo de informação que é unidirecional, pois, nesse caso, não existe a replicação inversa, no sentido "tabela fato" para as respectivas dimensões.

[10] Modelagem multidimensional em termos práticos é uma modelagem relacional no esquema estrela e a "tabela fato" é uma tabela na segunda forma normal que ocupa o centro da estrela. Possui chaves estrangeiras que representam cada uma das tabelas da estrela, onde cada tabela é uma dimensão. Esta "tabela fato" consolida em um só lugar dados coletados de diversas tabelas.

Esta possibilidade, como dito anteriormente, é ainda teórica. O modelo ainda tem que ser testado e avaliado muitas vezes para o fim que se propõe, que é a replicação de dados em um ambiente assíncrono e *multimestre.* Somente depois pode ser revisto, com o objetivo de utilizá-lo em ambientes de modelagem multidimensional.

Métodos e técnicas para medidas de desempenho e escalabilidade: o foco principal da dissertação foi abordar um modelo para replicação e não questões de desempenho e escalabilidade. Apesar disto, verificou-se que todos os testes tratados no próximo capítulo, mesmo sem medidas formais, desenvolveram-se dentro de expectativas aceitáveis, tanto em redes locais como utilizando a Internet. O modelo RPCD, porém, possui um módulo centralizador, que é o *GlobalSite,* e um módulo com muitas atribuições que é o *DatabaseSite.* Estudos futuros podem desenvolver métodos e técnicas para medir o desempenho e a escalabilidade, principalmente dos módulos citados, em situações extremas, tais como um grande aumento de bases replicadas, de número de *sites* e quantidade de operações. Outra medida a ser estudada seria a capacidade do modelo de fazer a conciliação de informações, pois ele é assíncrono e prevê a possibilidade de longos tempos de desconexão (horas e até dias) ou desconexões freqüentes.

3.6 – Comentários

A replicação envolve procedimentos complexos, principalmente se houver muitos escritores. São muitos os aspectos envolvidos e cada aspecto pode ser aprofundado e aperfeiçoado. Os fundamentos teóricos e os princípios básicos da replicação, tratados no capítulo 2, trazem orientações sobre "o que deve ser feito" e poucas idéias sobre o "como fazer", exceto quando se trata de algoritmos ou situações particulares. Entretanto, a replicação é muito abrangente e qualquer solução tratada de forma isolada contribui muito pouco para uma implementação real. Provavelmente, muitos fornecedores que disponibilizam replicação em seus bancos de dados não expõem a maneira de como o fizeram, porque os detalhes não interessam ao usuário final ou por questões estratégicas. O modelo aqui apresentado enfoca principalmente o aspecto de "como fazer", independente de banco ou linguagem, procurando dividir responsabilidades e assinalando onde os aspectos teóricos da replicação podem ser tratados em cada uma das partes do modelo. O desenvolvimento e a implementação do modelo, expostos no próximo capítulo, permitiram que se aperfeiçoasse o modelo proposto e contribuíram muito para que o modelo ficasse mais factível.

Capítulo 4

Desenvolvimento de uma ferramenta para replicação baseada no Modelo RPCD

Esse capítulo apresenta o desenvolvimento de uma ferramenta baseada no modelo RPCD, com vistas a verificar as premissas descritas anteriormente. A seção 4.1 apresenta, de forma resumida, algumas considerações de caráter geral que antecedem o desenvolvimento da ferramenta.. Na seção 4.2 citam-se as metodologias utilizadas. A especificação formal é exposta na seção 4.3 e a implementação na seção 4.4. Testes e resultados são apresentados na seção 4.5. A seção 4.6 é reservada aos comentários finais.

4.1 – Considerações iniciais

A implementação da ferramenta se baseou na arquitetura descrita na seção 3.1, onde se considera a sua organização em duas subcamadas: "*Application Level*", que equivale à subcamada "Aplicação RPCD" mencionada na seção 3.1 e "*Communication Level*" que equivale à subcamada do "Protocolo RPCD". Para fins de teste dos módulos implementados, desenvolveu-se também quatro aplicações para interagir com a ferramenta. Duas aplicações, sendo uma gráfica e outra do tipo console, para administrar a subcamada *Application Level*. Duas aplicações de apoio, sendo que uma delas testa e monitora a subcamada *Application Level*, além de auxiliar nas configurações necessárias, e a outra testa a subcamada *Communication Level*. A figura 4.1 ilustra este esquema.

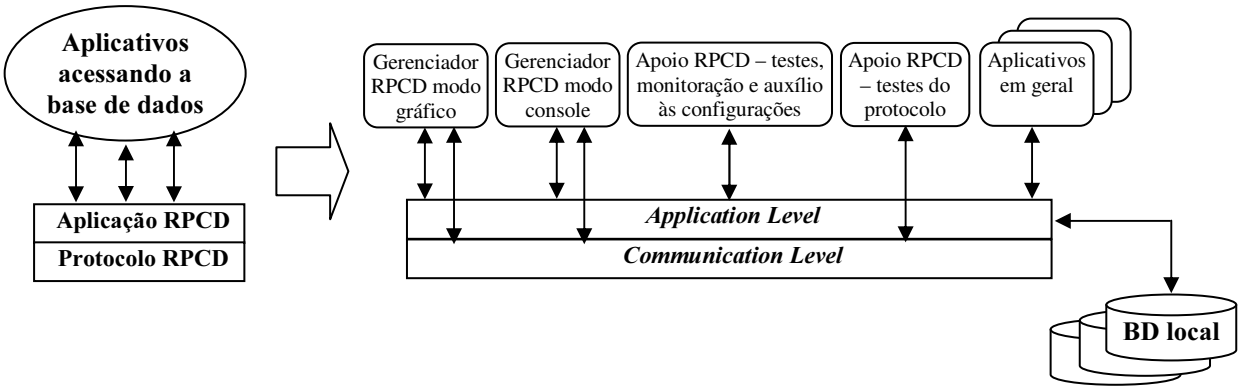


Figura 4.1 – Implementação do modelo RPCD

Pode-se observar na figura 4.1 que as aplicações de apoio interagem diretamente com a subcamada *Communication Level*. Isto é uma característica exclusiva destas aplicações. No uso normal do RPCD, apenas a subcamada *Application Level* é quem utiliza os serviços da subcamada *Communication Level.* O acesso à ferramenta, por parte dos aplicativos em geral, é feito exclusivamente através dos serviços disponibilizados pela subcamada *Application Level.*

Escolheu-se um ambiente real para fins de análise, teste e validação da implementação da ferramenta. O esquema da figura 4.2 mostra, de forma resumida, as principais características de um ambiente de BDDs, onde os *sites* podem permanecer desconectados durante longos períodos de tempo. Este ambiente possui características semelhantes àqueles que implementam a Arquitetura de Objetos Distribuídos apresentada em [Gao,2001-p455].

No ambiente considerado, existe um banco de dados primário que contém toda a base de informação da empresa, fragmentada de forma excludente em função dos dados que dizem respeito à cada *site* remoto. Numa aplicação de Automação da Força de Vendas, por exemplo, o banco de dados primário conteria a base de informação de vendas da empresa e os *sites* remotos a base utilizada por vendedores ou representantes da empresa. O sistema de arquivos contém objetos de apoio à aplicação, tais como imagens, textos explicativos e outros objetos genéricos que não participam do banco de dados.

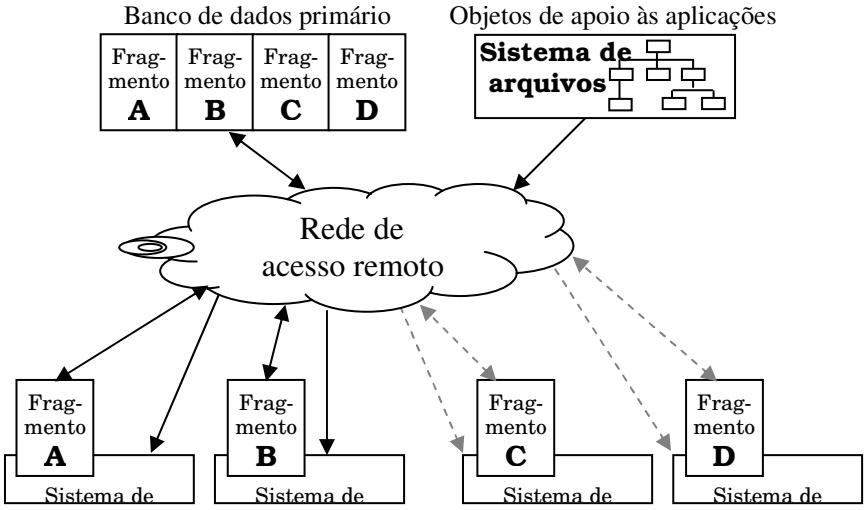


Figura 4.2 – Ambiente de um banco de dados e sistema de arquivos replicados e distribuídos

Na mesma figura, os *sites* com as bases A e B em computadores distintos já estão idênticos ou em processo de reconciliação em relação à base central. Os *sites* com as bases C e D, se alteradas, só ficarão devidamente reconciliadas após a próxima conexão. O *sistema de arquivos* é apenas distribuído, não estando sujeito a alterações.

Informalmente, no exemplo de aplicação mencionada, pode-se imaginar que a proposta atenderia a vendedores do tipo "caixeiro viajante": usando um *notebook,* ele consulta e emite pedidos, fornece preços, mostra imagem de produtos, tudo isso operando *off-line* e podendo ficar dias, durante as viagens, sem ter a oportunidade de conectar-se à matriz. Quando tiver a oportunidade de fazer a conexão, haverá, então, uma reconciliação automática das bases de dados.

Finalmente, deve-se ressaltar que o principal critério que fundamentou as escolhas descritas nas próximas seções foi, sempre que possível, priorizar ferramentas e soluções que não fossem comerciais.

4.2 – Metodologias utilizadas

A metodologia de desenvolvimento adotada foi o RUP (*Rational Unified Process*), mas apenas nos seus aspectos de mais alto nível, conforme sugerido por [Pender,2004-p399]. Isto porque o modelo RPCD não é um sistema de grande porte, como costuma ocorrer quando se trata de modelos de negócios, ou quando se deseja controlar muitos processos, atores e outras características típicas de grandes sistemas. O modelo RPCD pode ser considerado como um sistema de apenas dois processos: manipular operações (nível de aplicação) e propagá-las (nível de comunicação).

4.2.1 – Metodologia de especificação

Utilizou-se *UML (Unified Modeling Language)*[11] como técnica formal de especificação. A escolha se deve ao fato dessa linguagem possuir recursos que permitem uma modelagem abrangente para representação de sistemas. Como ferramenta *case*, utilizou-se o *StarUML*[12] baseado na UML 2.0.

4.2.2 – Metodologia de implementação

Para a implementação utilizou-se a linguagem de programação Java[13]. Escolheu-se essa linguagem porque ela incorpora, de forma bastante precisa, todos os conceitos previstos na orientação a objeto[14]. É também, por enquanto, a única linguagem de amplo uso realmente multiplataforma. A existência da arquitetura J2EE (para aplicações distribuídas em ambientes corporativos) e do JME (Java para computação móvel) são um motivo a mais que podem permitir extensões futuras dessa proposta. O protocolo também foi implementado em Java, utilizando *XML (Extensible Markup Language)*[15] como linguagem de especificação das mensagens. Os arquivos de configuração também são descritos em *XML*. A implementação foi desenvolvida utilizando a plataforma *NetBeans*[16]. O banco de dados escolhido para uso interno da ferramenta foi o *HSQLDB*[17]. Trata-se de um banco de dados com recursos suficientes para atender à ferramenta e trabalhar como servidor remoto, servidor local ou como *engine* (incorporado à aplicação).

---

[11] A *UML (Unified Modelling Language)* define uma linguagem amplamente esquemática para capturar requisitos de sistema e expressar projeto de sistema em termos orientados a objetos. Os diagramas UML são normalmente usados para ilustrar relacionamentos de classe e interações de objeto. A UML foi criada por meio de uma união de metodologias de OO anteriores, incluindo Booch, Jacobsem e OMT [Bond,03-p927]. A UML é gerenciada pelo *Object Management Group (OMG)* – [UML,2007]

[12] *StarUML™ – The Open Source UML/MDA Platform* – [StarUML,2007]

[13] *Java Platform, Standard Edition* – linguagem licenciada *free* pela *Sun Microsystems*. [Java,2007]

[14] Exceto a herança múltipla, da forma como previsto na modelagem orientada a objeto e utilizado na linguagem C++. Apresenta, porém, a *Interface,* recurso que supre de forma bastante abrangente as necessidades da herança múltipla.

[15] Ver observação da seção 3.2.3 sobre XML.

[16] *NetBeans IDE* é um ambiente integrado de desenvolvimento de *software*, livre e *open-source*, apoiado pela *Sun Microsystems*. [NetBeans,2007]

[17] *HSQLDB (Hypersonic SQL Database Engine)* é um banco de dados relacional *open-source,* mantido pelo *HSQL Development Group*. [Hsqldb,2007]

Os bancos de dados utilizados para fins de replicação e que formaram o ambiente de implementação foram *PostgreSQL*[18], *Hxtt*[19] (para tabelas *Paradox* e *xBase*), *MySql* [20], *Firebird* [21] e *Hsqldb*.

4.3 – Especificação formal da ferramenta

Uma das principais características da UML é utilizar a mesma técnica formal de especificação para as fases de levantamento de requisitos, análise, projeto e implementação[22]. Por causa disto, muitas vezes, é difícil ter uma distinção clara entre as fases de um desenvolvimento, pois dependem muito do nível de profundidade que o desenvolvedor adotou em cada fase. Neste desenvolvimento, adotou-se a seguinte abordagem baseada na metodologia RUP: na seção 4.3.1 apresentam-se os casos de uso relacionados aos requisitos do modelo RPCD; na seção 4.3.2 agrupam-se os casos de uso em pacotes e expõem-se as principais classes do sistema; em 4.3.3 se apresenta o contexto onde o sistema deve atuar e os principais relacionamentos com elementos externos ao sistema; finalmente, ao longo da seção 4.4, mostra-se a implementação da ferramenta, que já representa as estruturas básicas do código dos programas gerados.

4.3.1 – Modelo de casos de uso

Os principais atores de modelo são: *Database administrator*, ator humano, que é o administrador das bases de dados; *Site user*, ator humano, que representa todos os usuários finais de um *site* qualquer; *Any User Application*, ator não humano, que representa qualquer aplicativo que utiliza bases replicáveis e por conseqüência deve ser monitorado; *Database*, ator não humano, que representa as bases de dados que serão sincronizadas.

Na figura 4.3, os casos de uso (1), (2) e (3) são implementados pelos Gerenciadores do RPCD. O *Database administrator* configura, inicia e desativa os serviços RPCD. Já *Site user* pode apenas desativá-lo. Importante observar que o ator *Any User Application* pode iniciar os serviços RPCD. Isto significa que um aplicativo cujas bases de dados são replicadas ou qualquer *script* que inicie o aplicativo também pode iniciar os serviços RPCD. Desta forma, garante-se que os serviços RPCD sejam sempre ativados, não exigindo do usuário final que se lembre de iniciar esse procedimento. Entretanto, se houver algum problema, o usuário final pode desativá-lo.

Os casos de uso (4), (5) e (6) representam de fato os objetivos do modelo. O ator *Any User Application* espera que a ferramenta capture e propague as operações que ele efetivou no banco de dados**.** Já o ator *Database* espera que seus dados estejam sendo sincronizados com

---

[18] *PostgreSQL* é um banco de dados relacional, *open-source*, mantido pelo *PostgreSQL Global Development Group*. [Postgre,2007]

[19] *Hxtt (Hongxin Technology & Trade Ltd. of Xiangtan City) Paradox e DBF* são *drives* proprietários para Java que permitem acesso às tabelas relacionais *Paradox* e *xBase* utilizando SQL. [Hxtt,2007]

[20] *MySql* é um banco de dados relacional, *open-source*, mantido pela *MySQL AB (Aktiebolag)*. Na Internet: [Mysql,2007]

[21] *Firebird* é um banco de dados relacional, *open-source,* compatível com o banco proprietário *Interbase* da *Borland.* [Firebird,2007]

[22] A uniformidade entre as fases de análise, projeto e implementação ocorre principalmente através do diagrama de classes. Existem ferramentas que transformam diagramas de classes em código de programas e vice-versa. Para exemplificar esta coesão, basta citar que a OMG (*Object Management Group*), responsável pela padronização da UML, entre outros projetos, quer chegar ao UML executável através da especificação da semântica de ações (*Action Semantics*) [Pender,2004-p7].

suas réplicas. Os casos de uso (4) e (5), por sua vez, sempre precisarão do caso de uso (6) para trocar informações com as bases remotas. Os objetivos envolvidos nesses casos de uso são melhor descritos nos diagramas de classe e seqüência apresentados nas próximas seções.

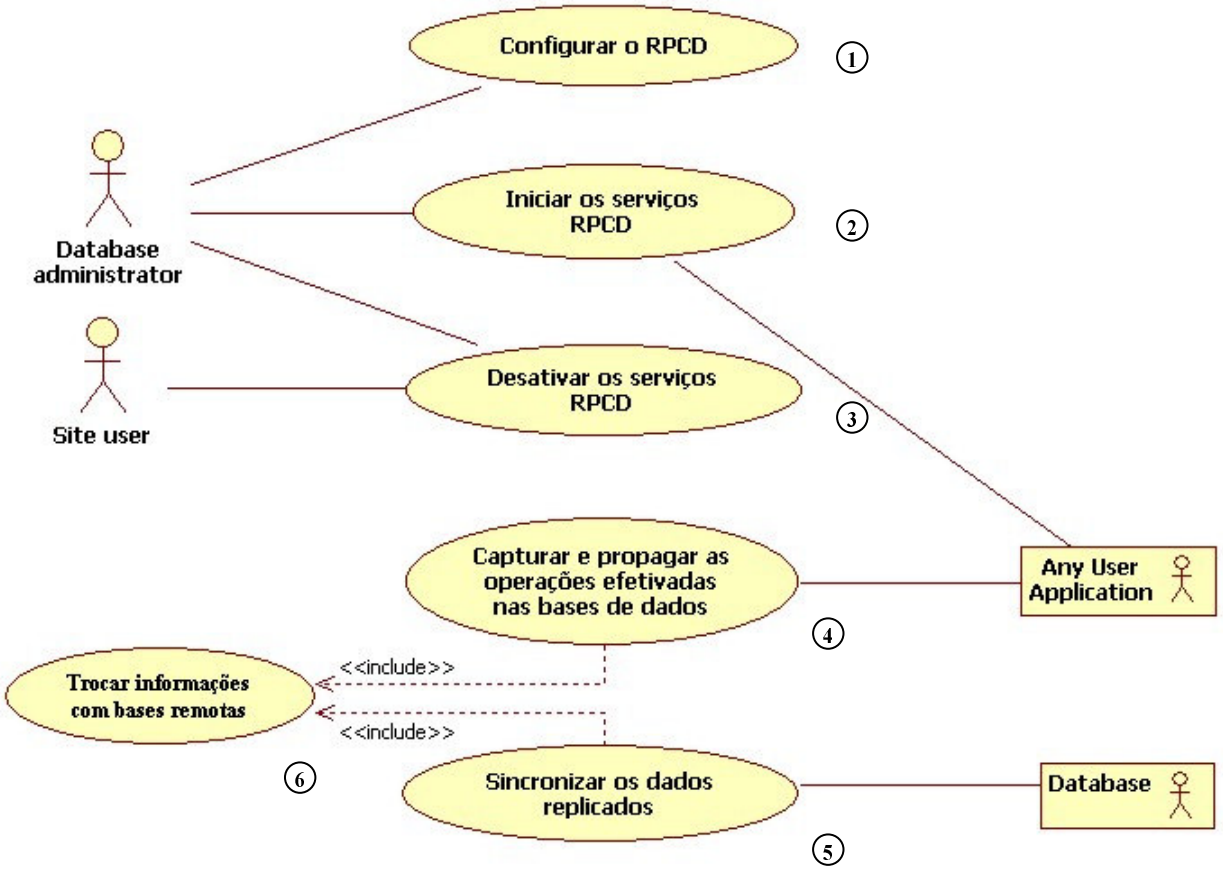


Figura 4.3 – Casos de uso do modelo RPCD

4.3.2 – Modelo de análise

Conforme a orientação indicada pelo método RUP [Pender,2004-p400], após a definição dos casos de uso, deve-se agrupá-los em pacotes. Sendo assim, os casos de uso foram agrupados em três pacotes, como mostra a figura 4.4.

A ferramenta é implementada nos pacotes *Application Level* e *Comunication Level*. Nas próximas seções estes pacotes são detalhados. O pacote *Administração RPCD* foi desenvolvido, mas não é detalhado nessa dissertação, uma vez que os programas que constituem este pacote utilizam o modelo RPCD como qualquer outro aplicativo, ou seja, através de interfaces e arquivos de configuração, sendo assim caracterizado mais como um exemplo de interface usuário-máquina do que como um modelo de administração específico do RPCD.

A figura 4.5 apresenta as principais classes relacionadas aos pacotes *Application Level* e *Comunication Level.*

No pacote *Application Level* a captura de operações efetivadas no banco de dados ocorre nas classes *DatabaseMonitor* e *TableMonitor*. Tarefas comuns, como a utilização da interface

IOperation, são implementadas na classe genérica *Monitor*. Isto atende ao especificado nas seções 3.2.1.1 e 3.2.1.3. Já a classe *RpcdDriver* atende à especificação da seção 3.2.1.2, que recomenda a substituição do *driver* do banco de dados e também utiliza a interface *IOperation* para repassar as operações.

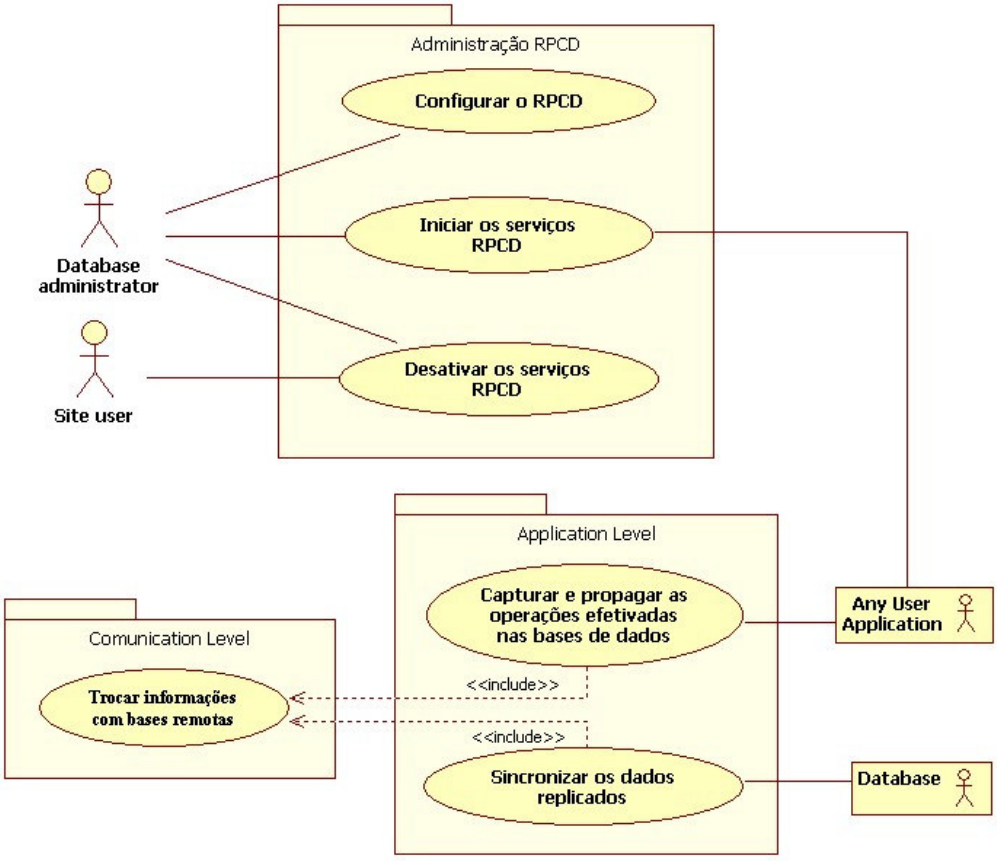


Figura 4.4 – Modelo de pacotes dos casos de uso

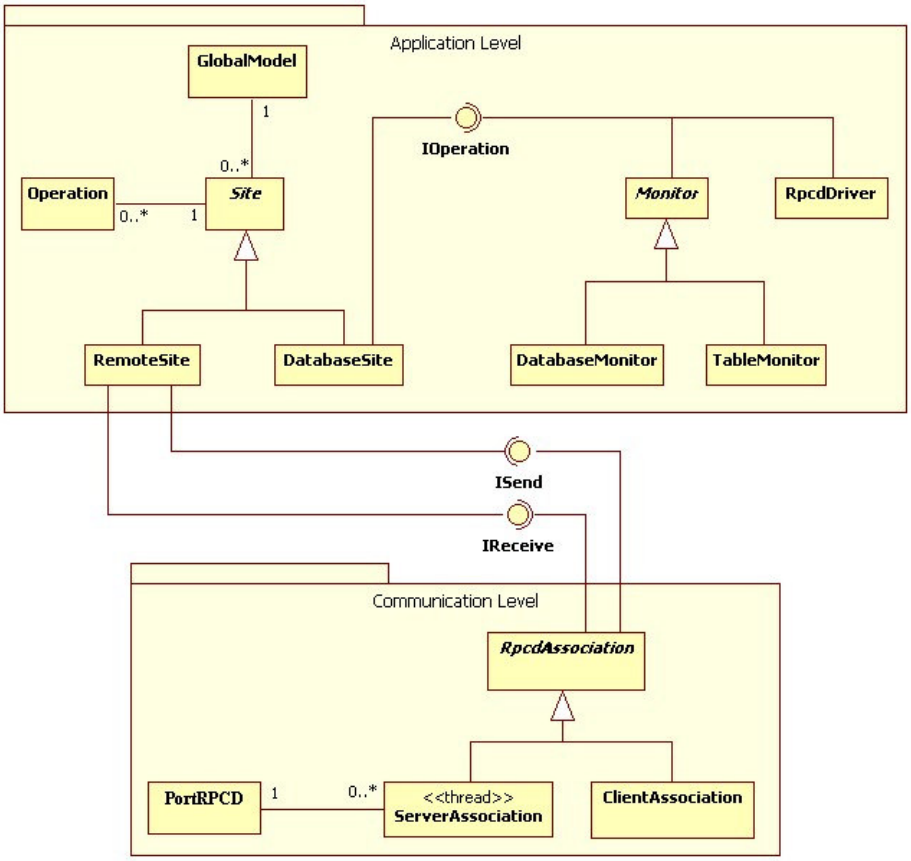


Figura 4.5 – Diagrama de classes

Cada uma das instâncias da classe *DatabaseSite* representa uma base de dados local e recebe as operações efetivadas na base de dados, através da implementação da interface *IOperation*. Estas operações são armazenadas em instâncias da classe *Operation*. O modelo global é definido na classe *GlobalModel* e o mapeamento entre a base local e o modelo global é implementado em *DatabaseSite. GlobalModel* pode estar associado a nenhuma ou diversas instâncias de *DatabaseSite*, mas cada *DatabaseSite* está associado apenas a um *GlobalModel*. Este mecanismo está especificado nas seções 3.2.2.2 e 3.2.2.3.

Os *sites* remotos que podem replicar as bases de dados são implementados nas instâncias da classe *RemoteSite*. Pode-se observar a existência da classe abstrata *Site,* que generaliza *DatabaseSite* e *RemoteSite* e mantém a base de instâncias de *Operation*. Isso significa que tanto *DatabaseSite* e *RemoteSite* mantém as operações aplicadas às bases de dados, tanto as transmitidas quanto as recebidas através um mecanismo comum e tratam o modelo global da mesma forma. Através das interfaces *ISend* e *IReceive,* a classe *RemoteSite* utiliza e implementa respectivamente as primitivas do protocolo RPCD. Esta classe cumpre as especificações da seção 3.2.2.4.

O diagrama de seqüência na figura 4.6 esquematiza quase todos os processos descritos anteriormente. Estão representadas na figura apenas as seqüências relativas ao modo de funcionamento normal da ferramenta e com os *sites* local e remoto já associados.

As seqüências de passos de (1) a (5) representam a captura das operações através da monitoração da base de dados. Através do passo (3), a operação aplicada às bases de dados é encaminhada para o processo de replicação.

As seqüências de passos de (6) a (14) representam todo o processo de replicação. A operação aplicada é armazenada através dos passos de (6) a (8); ela é publicada, ou seja, propagada às

demais bases através dos passos de (9) a (12). Se a réplica for local, a operação é aplicada através do passo (13). Se for remota, é incluída para propagação no passo (11) e, estando associada, será transmitida através do passo (14). Deve-se lembrar que *Site* pode representar tanto a base local (*DatabaseSite*) como a remota (*RemoteSite*).

A propagação remota para o *site* emissor termina no passo (14), mas para o *site* receptor começa no passo (17). Recebendo uma operação remota, o *site* inclui a operação e a propaga às demais bases locais através dos passos (18) a (20) e, depois, segue as seqüências a partir do passo (9) como descrito anteriormente.

As seqüências de passos (21) a (25) mostram a alternativa *RpcdDriver,* que pode ser utilizada no lugar da monitoração das portas dos bancos de dados, indicada através das seqüências de (1) a (5). Pode-se observar que a partir dos passos (3) e (23), as seqüências são idênticas.

Apresenta-se a seguir a relação entre os casos de uso e o diagrama de seqüência para melhor compreensão do modelo de análise:

- Capturar e propagar as operações efetivadas nas bases de dados:
  - Capturar: (1), (2), (4) e (5) ou (21), (22), (24) e (25), ou ambas simultaneamente.
  - Propagar: (3), (6) a (12), (23)
- Sincronizar os dados replicados: (13)
- Trocar informações com bases remotas: (14) a (20)

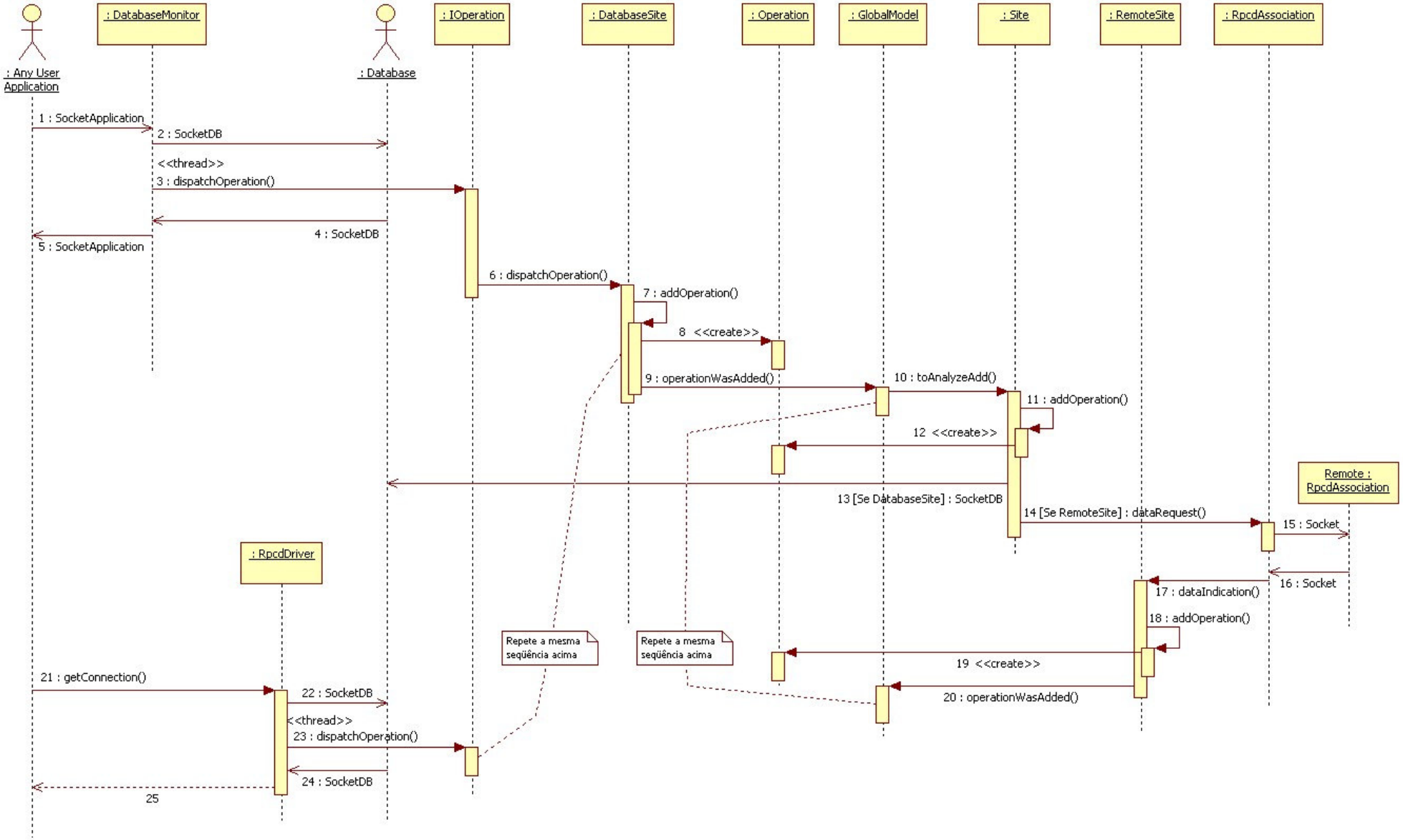


Figura 4.6 – Diagrama de seqüência dos principais casos de uso

O subsistema de fronteira (*Boundary*) não está representado neste esquema, pois como explicado na seção 3.2.2.1, disponibilizar serviços ao "mundo externo" está mais relacionado com o projeto ou implementação. Neste nível apresentam-se apenas as classes essenciais que modelam a análise.

No nível de comunicação (*Communicaion Level*), conforme mostrado na figura 4.5, a classe *RpcdAssociation* implementa e utiliza respectivamente as interfaces *ISend* e *IReceive*, que definem as primitivas do protocolo. Além disto, nesse nível é feita a conversão de objetos de PDU em XML e vice-versa. As classes herdadas, *ServerAssociation* e *ClientAssociation* implementam as diferenças entre o papel de servidor e cliente. Pode-se observar que o estereótipo *<<thread>>* em *ServerAssociation* já indica que o *host* que estiver no papel de servidor permitirá múltiplas conexões. A classe *PortRPCD* é a responsável por atender as solicitações de conexão e criar as *threads* servidoras. Estas classes atendem ao especificado na seção 3.2.3 e o diagrama de seqüência deste nível é exatamente o diagrama de operação do protocolo RPCD, apresentado na figura 3.6 daquela seção.

4.3.3 – Modelo de projeto

Ainda de acordo com a orientação do método RUP, o modelo de projeto aprofunda-se ainda mais nas definições das classes já tendo em vista a implementação. Neste tópico mostra-se como foram projetados os principais módulos do modelo RPCD e como estes se relacionam com os elementos externos ao modelo.

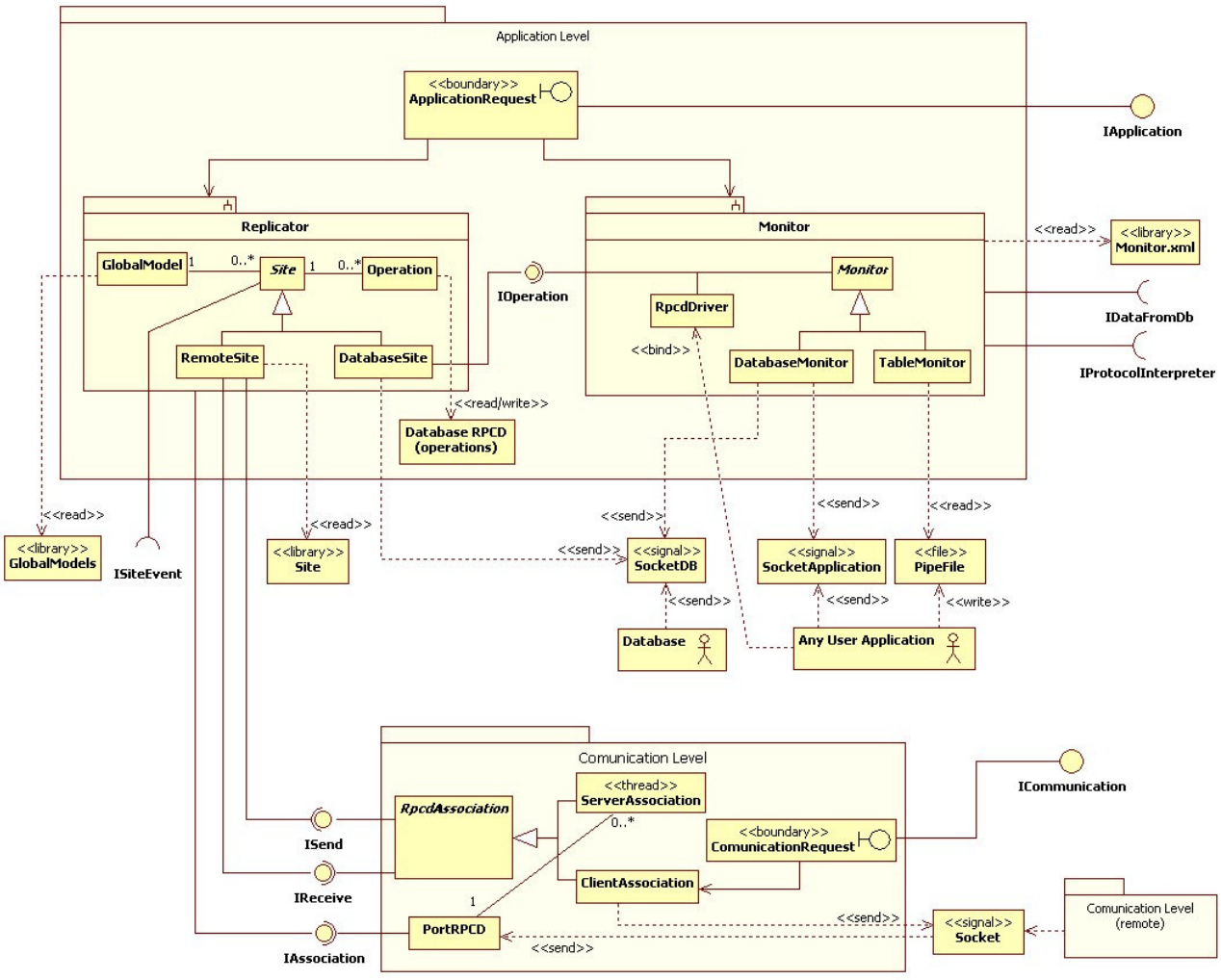


Figura 4.7 – Principais pacotes e subsistemas do projeto do modelo RPCD

A figura 4.7 mostra mais uma vez a divisão do modelo em dois pacotes e detalhando mais os seus elementos constituintes. O pacote *Application Level* implementa a parte de "Monitoração de Dados" no subsistema *Monitor* e a parte de "Replicação e Sincronização dos Dados" no subsistema *Replicator*. O pacote *Comunication Level* implementa a parte relativa à conexão, troca de mensagens e processamento das *PDUs* do protocolo RPCD.

O subsistema *Monitor* repassa as informações capturadas nas bases de dados para todas as classes que implementam *IOperation*. A classe *DatabaseSite*, do subsistema *Replicator,* implementa a interface *IOperation* e, com as informações recebidas, faz a replicação e sincronismo. Ambos os subsistemas dependem de elementos externos para fins de configuração, operação e persistência. O arquivo *Monitor.xml* contém informações para configurar o subsistema *Monitor.* Este, quando ativo, recebe informações das aplicações através do arquivo *PipeFile* e do *SocketApplication*, comunicando-se com os banco de dados através do *SocketDB.* Já o subsistema *Replicator* obtém as configurações necessárias nos arquivos *Site* e *GlobalModels*. Ele também atualiza os bancos de dados utilizando instâncias de *SocketDB*. As operações efetivadas nas bases de dados e gerenciadas pelo *Replicator* são persistidas em *Database RPCD.*

Quanto às interfaces, as que atendem especificações do modelo são a *IProtocolInterpreter*, responsável por interpretar o protocolo *front-end / back-end* entre a aplicação e a base de dados, e aquelas que intermediam os processos entre *Replicator* e *Comunication Level*: *IAssociation*, *ISend* e *IReceive*. As demais existem apenas para uso da implementação: *IApplication*, implementada pela classe *ApplicationRequest*, permite o gerenciamento dos subsistemas *Monitor* e *Replicator* (casos de uso (1),(2) e (3) – ver seção 4.3); a interface *ISiteEvent* possibilita divulgar os eventos ocorridos nos *sites* quando o RPCD estiver ativo; *IDataFromDb* permite o acesso às mensagens de retorno do banco de dados, sendo que isto só é de fato útil durante a fase de testes. Pode-se observar que as interfaces *ISiteEvent* e *IDataFromDb* dependem de uma implementação externa e a sua utilização não é obrigatória. Já a interface *IProtocolInterpreter* tem que ser implementada, externa ou internamente, pois do contrário as mensagens não serão interpretadas e serão repassadas no formato original.

O pacote *Comunication Level* cuida da parte da comunicação e os únicos vínculos com o ambiente externo são os *sites* remotos, o que ocorre com a utilização de *sockets* e da implementação da interface *IComunication,* que permite gerenciar a comunicação (basicamente iniciar e parar). A classe *RemoteSite*, do subsistema *Replicator,* implementa as interfaces *IAssociation* e *IReceive*, que é o mecanismo para receber as primitivas do protocolo. O pacote *Comunication Level* implementa a interface *ISend,* que permite receber as primitivas solicitadas pelo subsistema *Replicator*.

As classes *ApplicationRequest* e *ComunicationRequest* implementam a fronteira (*boundary*) do modelo. *ApplicationRequest* implementa a interface *IApplication,* que é a porta de entrada para gerenciar os subsistemas *Monitor* e *Replicator*. Já a classe *ComunicationRequest* implementa a interface *ICommunication,* que permite gerenciar o pacote *Comunication Level*.

4.4 – Implementação da ferramenta

Nesta seção apresentam-se todas as classes implementadas no seu formato final. Não são apresentadas as classes e interfaces que pertencem à própria linguagem Java. Para fins de clareza, nos diagramas apresentam-se apenas as informações essenciais de cada classe, omitindo-se atributos e métodos privados, com raras exceções. A codificação seguiu a convenção *get, set* e *is* para acesso aos atributos da classe, mas, nos diagramas, os atributos públicos foram mantidos como tal, também para fins de clareza.

4.4.1 – Subsistema *Monitor*

A finalidade deste subsistema é capturar todas as operações efetivadas nas bases de dados. A figura 4.8 mostra o diagrama de classes da implementação. A sua aparente complexidade se deve ao fato desse subsistema implementar em detalhes todas as técnicas sugeridas na seção 3.2.1.

4.4.1.1 – Classes auxiliares do *Monitor*

MonitorManager, OutOperation e Configuration formam um conjunto de classes auxiliares que dão apoio ao funcionamento do subsistema *Monitor*.

MonitorManager é a classe responsável pela criação das instâncias de todos os monitores disponibilizados no arquivo de configuração *Monitor.xml*. Prepara também uma instância de *OutOperation* para disponibilizá-la aos objetos monitor.

OutOperation mantém um conjunto de referências aos objetos, quaisquer que sejam, que implementem a interface *IOperation.* Utilizando esta interface, o subsistema monitor encaminha as mensagens interceptadas pelos monitores a tais objetos.

Configuration é responsável por ler e interpretar as configurações mantidas no arquivo *Monitor.xml* e disponibilizá-las ao objeto *MonitorManager.* É previsto a configuração de três tipos de monitores: interpretadores de mensagens (*ConfigInterpreter*), leitores de arquivo *pipe* (*ConfigPipe*) e os responsáveis pela substituição do *driver* para um processo ambientado (*ConfigRpcdDriver*).

4.4.1.2 – Classes principais do *Monitor*

Monitor, DatabaseMonitor, TableMonitor, RpcdDriverMonitor e *RpcdDriver* são as classes que implementam as especificações de monitoração do modelo RPCD, conforme descrito na seção 3.2.1.

Monitor é a generalização das classes derivadas *<name>Monitor*. Possui um objeto *OutOperation* e garante que apenas uma instância do subsistema *Monitor* seja processada no mesmo *host*.

DatabaseMonitor é a classe que monitora as portas dos bancos de dados conforme descrito na seção 3.2.1.1. Ao ser ativada, quando recebe uma solicitação de um aplicativo, cria duas *threads* com instâncias das classes *ListenerUserApplication* e

AccessUserApplicationDatabase. A primeira atende ao aplicativo e encaminha as mensagens ao banco de dados e a segunda recebe as respostas do banco de dados e as encaminha ao aplicativo. Ambas as classes disponibilizam as mensagens para as interfaces *IOperation* e *IDataFromDB,* através da instância da classe auxiliar *OutOperation*. Conforme previsto, a interpretação do protocolo *front-end / back-end* pode ser implementada externamente ao sistema: para esta implementação foi desenvolvido a *GenericProtocolInterpreter*, classe que implementa um conjunto de subclasses para interpretação do protocolo de alguns bancos de dados.

TableMonitor monitora os arquivos *pipe* conforme descrito na seção 3.2.1.3. *ParadoxTableMonitor* e *ClipperTableMonitor* são classes que implementam o método abstrato *convertOperationForSQL.* Como o próprio nome indica, este método faz conversão da semântica não SQL dos sistemas legados *Paradox* e *Clipper* para a semântica do SQL.

RpcdDriver é a classe que permite processar o RPCD em modo ambientado, substituindo nas aplicações os *drivers* convencionais, conforme descrito na seção 3.2.1.2. Cada uma das classes herdadas representa o *Driver* propriamente dito que está sendo encapsulado. O encapsulamento ocorre somente em *RpcdDriver* através a implementação das interfaces do *java.sql*.

RpcdDriverMonitor é uma classe que se verificou necessária durante os testes: por razões técnicas e de desempenho, é interessante que seja processada apenas uma instância do módulo replicador. Para isto ser possível, o primeiro aplicativo que utilizar o *RpcdDriver* inicia o módulo replicador. Os demais aplicativos não iniciam o módulo replicador, mas enviam as mensagens ao primeiro que foi iniciado. Para possibilitar esta forma de uso criou-se a classe *RpcdDriverMonitor.*

4.4.2 – Subsistema *Replicator*

A finalidade deste subsistema é implementar os requisitos necessários à replicação. A figura 4.9 mostra o diagrama de classes da implementação.

ReplicatorManager é a única classe auxiliar do subsistema. Sua finalidade é criar as instâncias das classes do subsistema a partir dos arquivos de configuração e implementar as interfaces *IOperation* e *IAssociation*. Implementando a interface *IOperation,* simplifica a criação dos objetos *DatabaseSite* e os deixam mais seguros. Durante o processo de replicação, essa interface só faz repassar as mensagens para *DatabaseSite*. Já a interface *IAssociation* é necessária para atender ao nível de comunicação antes da identificação do *site* remoto que tenta comunicar-se. Após a identificação correta, a responsabilidade da comunicação fica com *RemoteSite*.

A classe abstrata *Site* implementa as atividade comuns a *DatabaseSite* e *RemoteSite*. Uma das atividades é fazer a persistência dos objetos que representam as operações interceptadas. Outra atividade é associar-se a um modelo global (instâncias da classe *GlobalModel*) e às tabelas de um modelo global. A utilização das associações depende das classes herdeiras, como se verá a seguir.

DatabaseSite recebe as operações efetivadas na base local. Através de classe *TableMapping* faz o mapeamento entre a tabela local e a tabela global equivalente. Com isto, a operação é persistida no seu formato global. Já os objetos de *RemoteSite* recebem as operações efetivadas em bases remotas através da interface *IReceive* (que define parte das primitivas do protocolo

RPCD). Como a operação já vem no formato global, ela é persistida imediatamente. Aqui se descreveu a captura das operações e sua persistência. A seguir descreve-se o inverso: como as operações são propagadas e aplicadas às bases locais.

Quando ocorre de um objeto *Operation* ser adicionado em *Site,* o objeto informa o fato à instância de *GlobalModel* à qual ele está vinculado. A instância de *GlobalModel,* por sua vez, divulga a operação adicionada a todas instâncias de *Site* a que ele está relacionado (exceto à instância de origem). Cada uma destas instâncias avalia o que deve ser feito com a operação: se for um *DatabaseSite*, decide se aplica ou não à base local, fazendo o mapeamento da tabela global para a local; se for um *RemoteSite* decide se propaga ou não ao *site* remoto, fazendo a fragmentação pertinente ao *site* remoto.

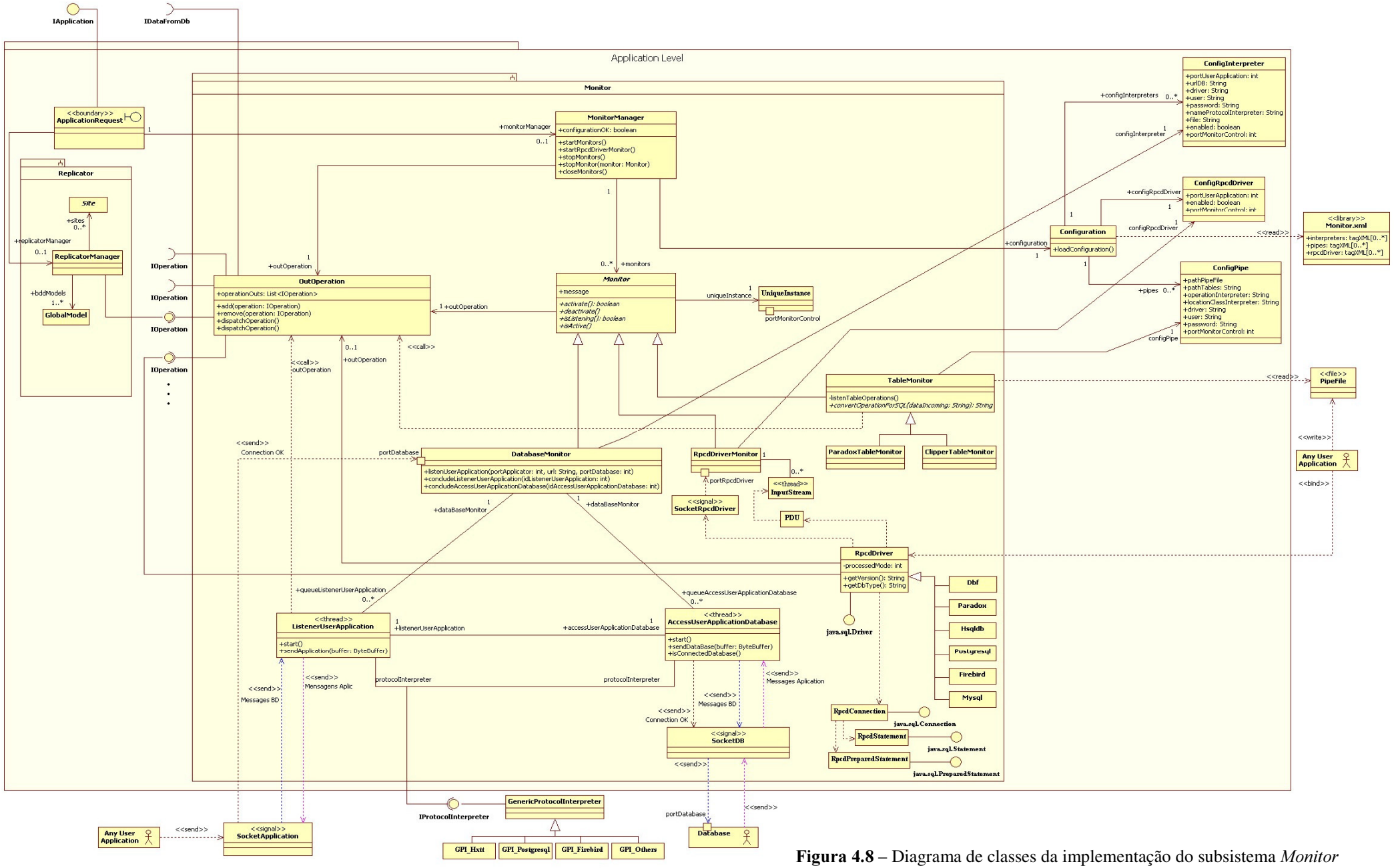


Figura 4.8 – Diagrama de classes da implementação do subsistema *Monitor*

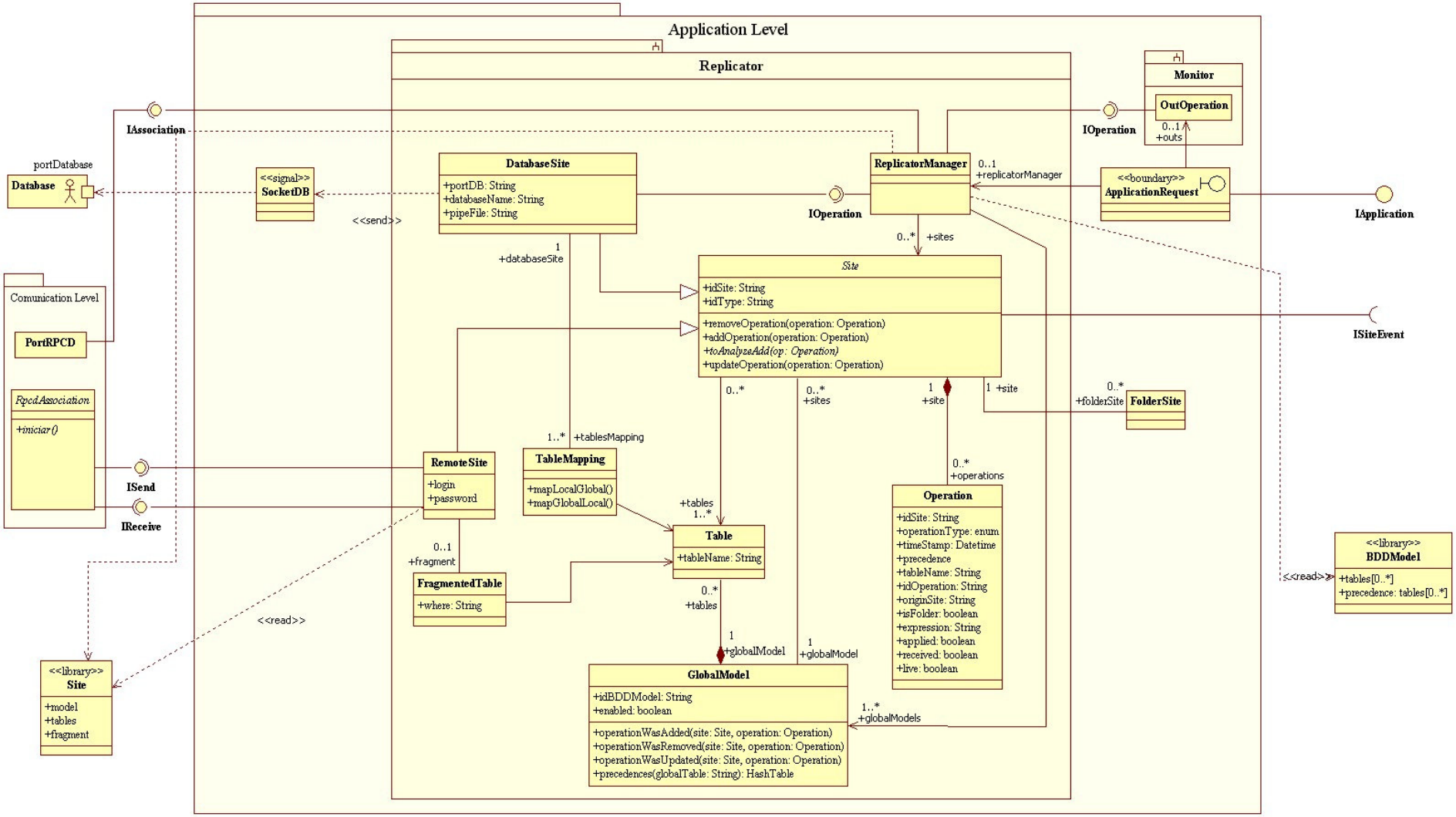


Figura 4.9 – Diagrama de classes da implementação do subsistema *Replicator*

4.4.3 – Pacote *Comunication Level*

Comunication Level, ou simplesmente *Comunication,* é chamado de pacote apenas por questões formais, pois como é menos complexo que o pacote *Application Level,* não justificava dividi-lo em subsistemas. A figura 4.10 mostra o diagrama de classes da sua implementação.

A classe auxiliar *ComunicationRequest* permite iniciar o processo de comunicação. Quando se encontra no papel de servidor, inicia com uma instância da classe *PortRPCD*. Quando no papel cliente, inicia com uma instância da classe *ClientAssociation*.

PortRPCD atende em uma porta TCP, por padrão 8000, mas esse valor pode ser configurado pelo administrador. Recebendo um pedido de conexão de uma instância remota de *ClientAssociation*, cria uma *thread* com um objeto *ServerAssociation,* o qual faz a validação da associação utilizando uma implementação de *IAssociation* e, sendo válida, passa a trocar mensagens com o cliente remoto.

ClientAssociation, quando ativada, tenta conectar-se ao servidor RPCD. Enquanto não conseguir, fica tentando a cada intervalo de tempo, de valor padrão 30s, também configurável. Só para de tentar quando obtém sucesso na conexão ou se for suspensa explicitamente.

Todas as demais primitivas do protocolo são implementados em *RpcdAssociation*. A classe *PDU* implementa as conversões das PDUs expressas em XML do protocolo em objetos PDUs utilizados nas primitivas. *RpcdAssociation* faz uma implementação padrão de *IReceive,* útil em testes, que encaminha as mensagens recebidas para uma saída padrão.

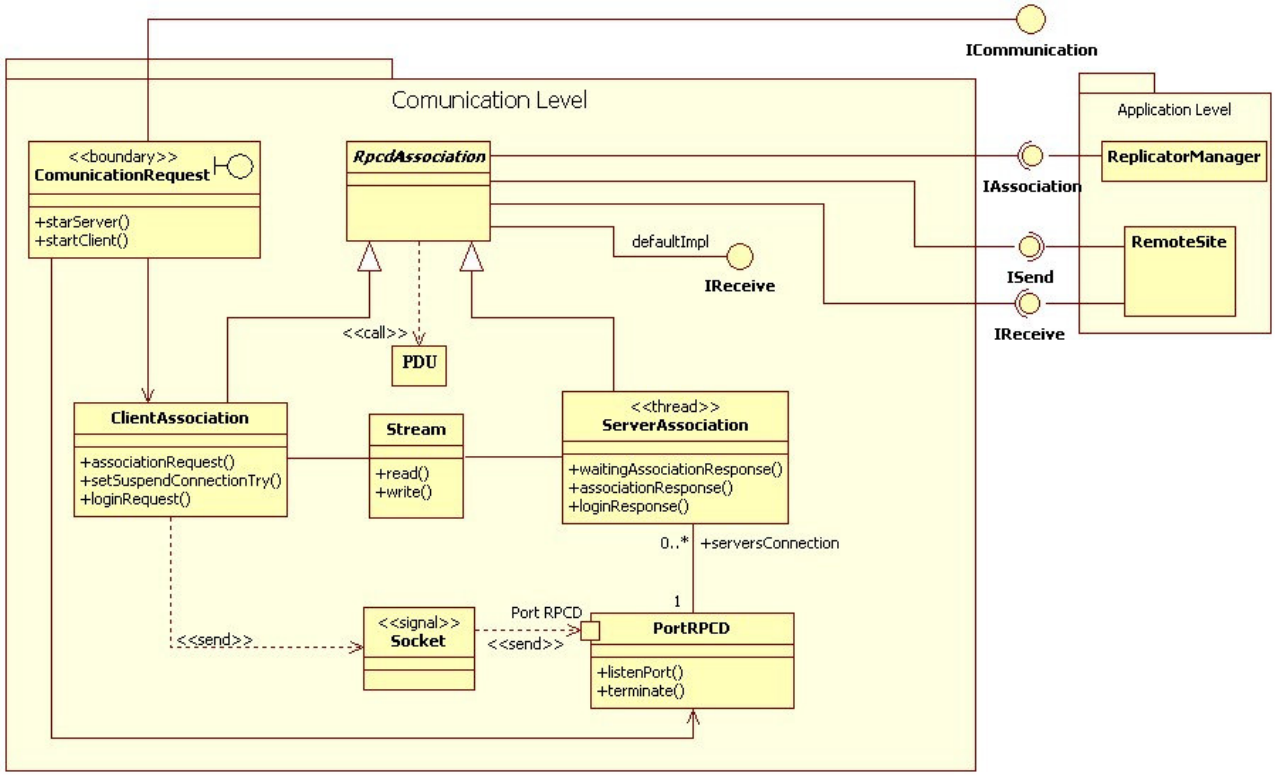


Figura 4.10 – Diagrama de classes da implementação do pacote *Comunication Level*

A figura 4.11 mostra o diagrama de estados da classe *PortRCPD*, responsável por atender a porta RPCD. Sua função é muito simples: atende a porta TCP designada e repassa a referência do *socket* para a classe *ServerAssociation,* através da ação *waitingResponseAssociation*. Pode-se observar que um término normal do serviço de conexão através do gatilho *finalizar*, faz com que a instância da classe pare imediatamente de atender a porta RPCD. Fica no estado *finalizando* enquanto houver conexões ativas e torna-se inativa apenas quando todas as conexões forem encerradas e, só então, se pode eliminar a instância da classe.
Na hipótese de alguma conexão não fechar, pode-se cancelar incondicionalmente a instância.

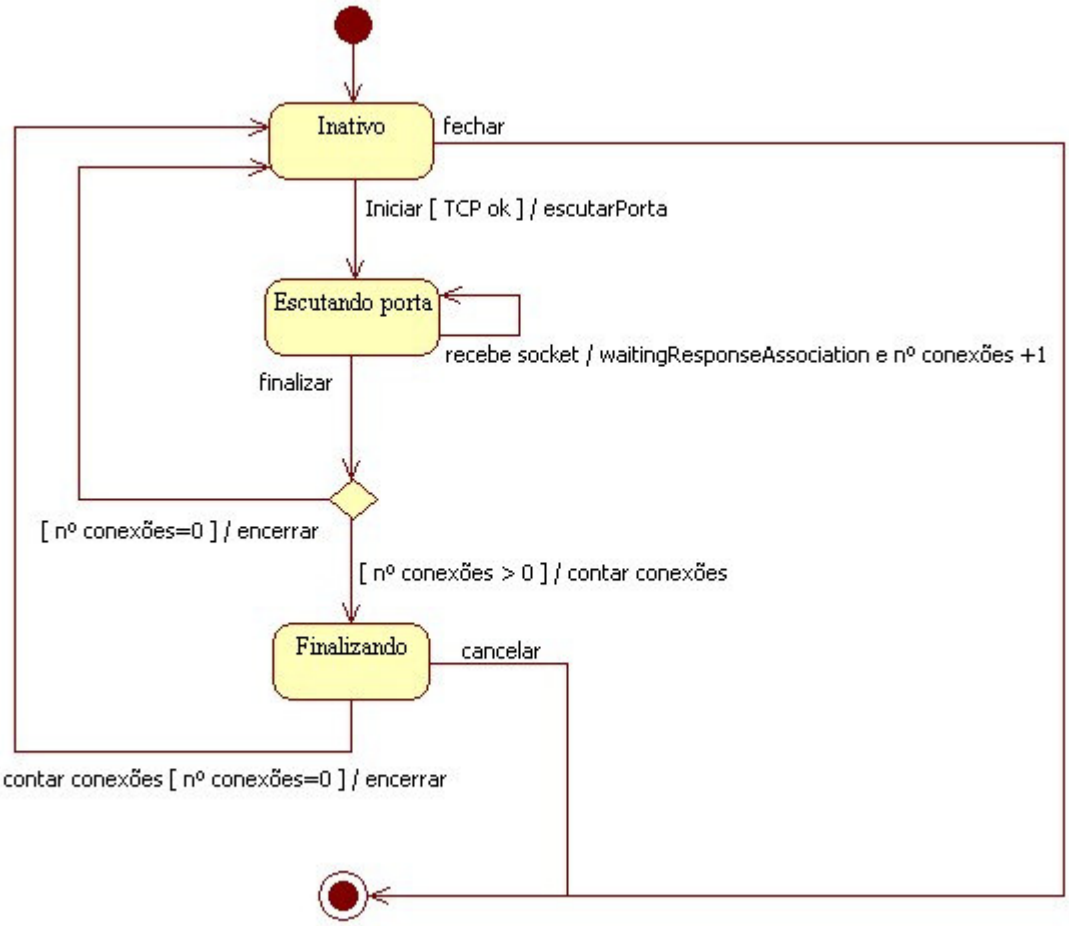


Figura 4.11 – Diagrama de estados da classe *PortRPCD*

As figuras 4.12 e 4.13 mostram os diagramas de estados de *ClientAssociation* e *ServerAssociation*. Exceto pelos estados iniciais, o comportamento de ambas as classes são muito semelhantes. Uma característica comum é que todas as mudanças de estado ocorrem de forma assíncrona, tanto quando se espera uma resposta remota ou quando se aguarda uma resposta através de uma primitiva. Por causa desta característica, na implementação da classe faz-se um teste para saber se a primitiva ou requisição remota pode ser atendida, com a instância no estado em que se encontra. Não atendendo, a primitiva *errorIndication* é chamada e a instância não muda de estado. Isto não está esquematizado no diagrama, pois, como é comum a todos os estados, pouco contribuiria para a clareza do diagrama.

A figura 4.12 mostra o diagrama de estados da classe *ClientAssociation,* responsável pelo papel de cliente previsto no protocolo. A primeira ação é tentar uma associação com o servidor até obter sucesso, exceto se forem suspensas as tentativas. Após a associação, aguarda-se a identificação fornecida pela subcamada *ApplicationLevel*. Recebendo a identificação, ela é encaminhada para o servidor e a instância da classe fica no estado *Authentication* até receber a resposta do servidor. Uma vez autorizada, fica aguardando as requisições remotas no estado de *Receiving*. A partir deste ponto, os papéis de cliente e de servidor se equivalem. Caso a instância da classe seja iniciada no papel de servidor, após um intervalo de tempo ou por requisição direta, ela assume o papel de cliente ficando no estado *Waiting_Services*, ou seja, aguardando as chamadas da subcamada *ApplicationLevel* para encaminhá-las ao *site* remoto.

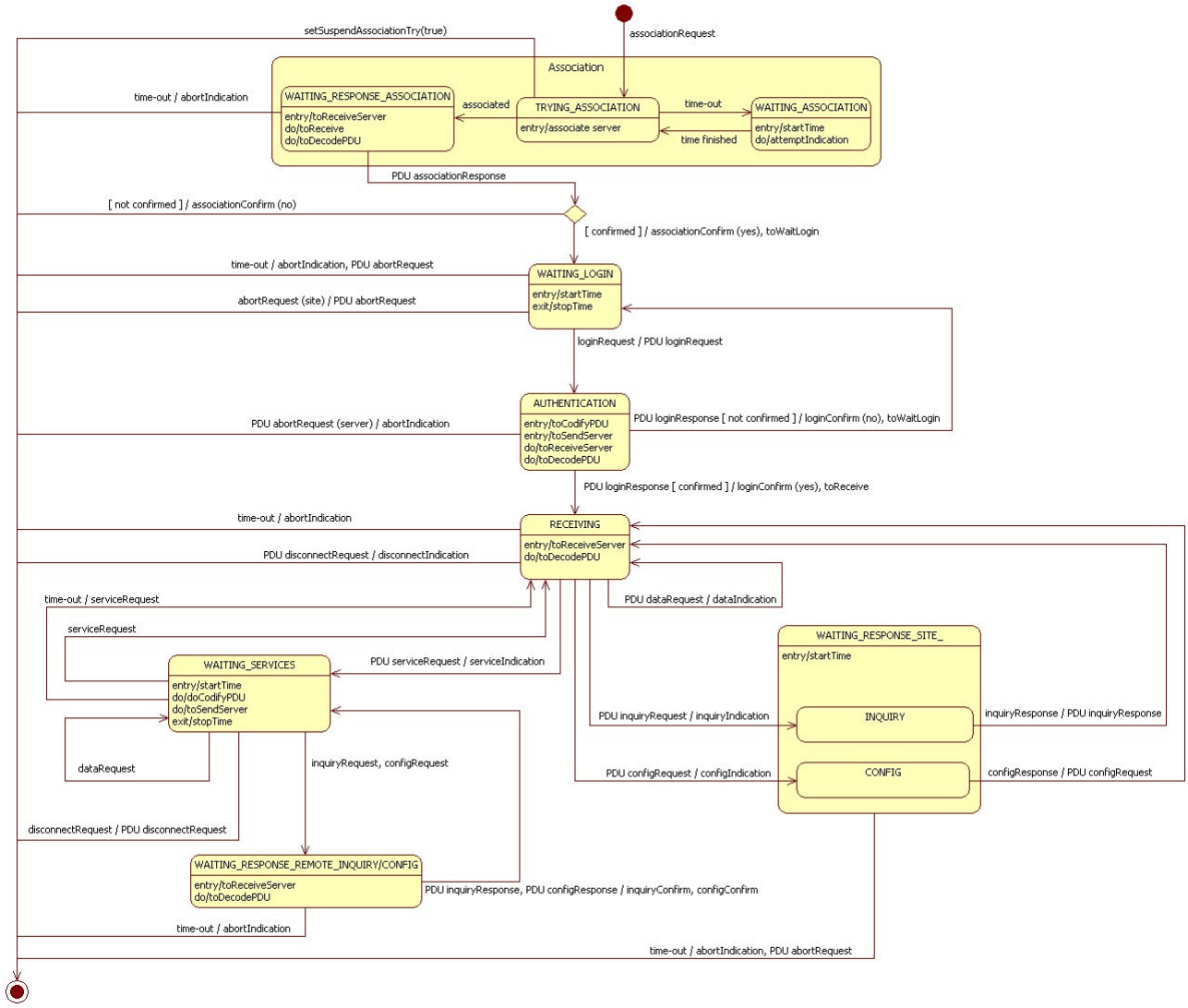


Figura 4.12 – Diagrama de estado da classe *ClientAssociation*

A figura 4.13 mostra o diagrama de estados da classe *ServerAssociation,* responsável pelo papel de servidor previsto no protocolo. Ela se inicia a partir da chamada feita pela instância da classe *PortRPCD*. Logo de início fica aguardando uma autorização do *ApplicationLevel* para prosseguir. Uma vez autorizada, a instância da classe fica aguardando a identificação e senha do cliente remoto. Recebendo essa identificação e senha, encaminha as mesmas para reconhecimento em *ApplicationLevel* e mais uma vez fica aguardando uma resposta, admitindo no máximo três tentativas sem sucesso na identificação. Confirmada a identificação, a instância dessa classe fica no papel de cliente, aguardando a chamada das primitivas, para encaminhar as requisições ao *site* remoto. A partir deste ponto, os papéis de cliente e servidor se equivalem. Caso essa instância seja iniciada no papel de cliente, após um intervalo de tempo sem chamadas ou por requisição direta, assume o papel de servidor, ficando no estado *Receiving*, ou seja, aguardando as requisições do *site* remoto.

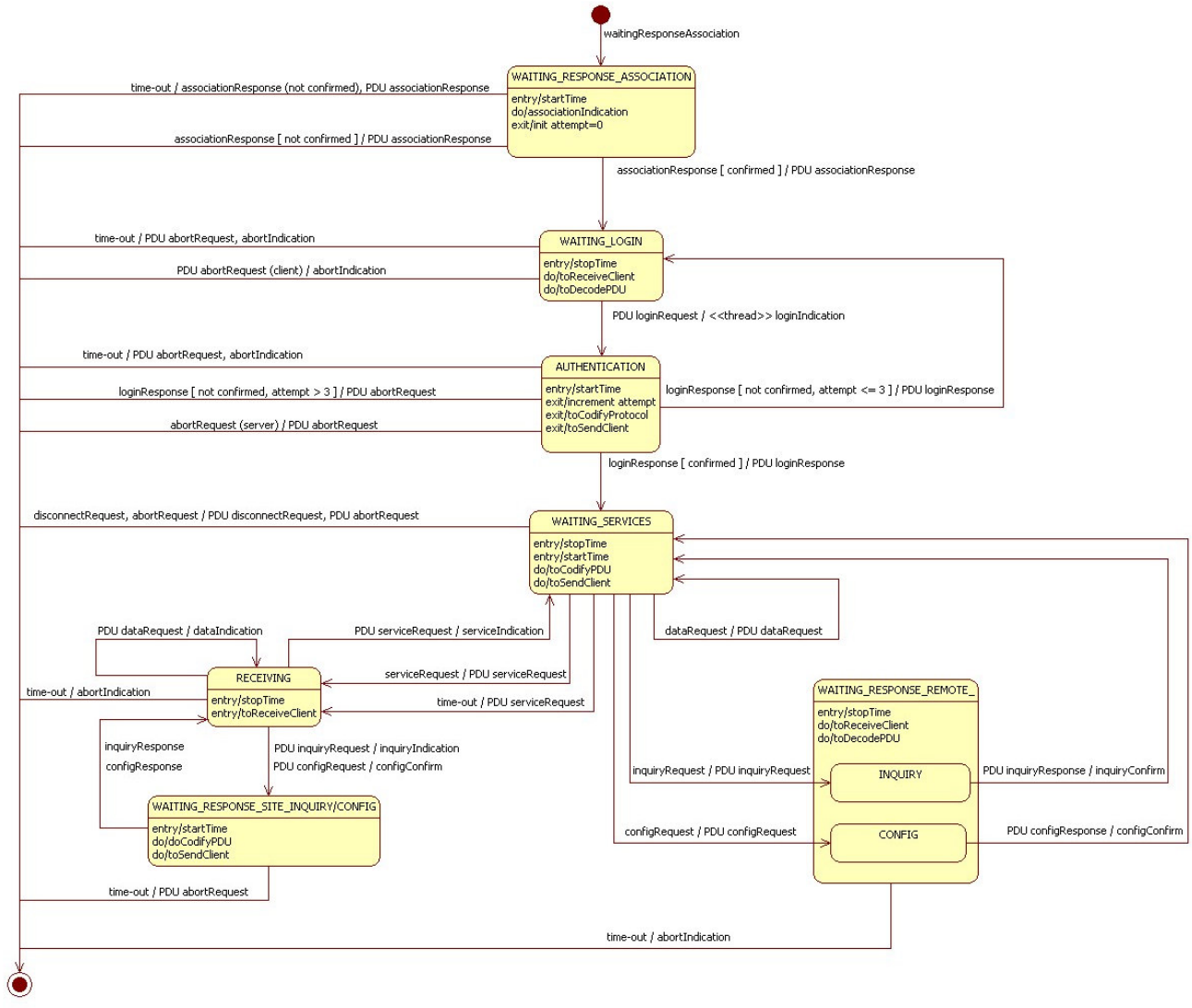


Figura 4.13 – Diagrama de estado da classe *ServerAssociation*

4.5 – Testes e resultados

Para fins de teste adotou-se um modelo simplificado de Automação de Força de Vendas. O principal motivo para esta escolha em detrimento de um modelo real, se deve ao fato que esse último conteria numerosos detalhes referentes ao modelo de negócio da empresa, detalhes estes que pouco acrescentariam aos testes realizados, além de fugir do escopo desta dissertação.

O modelo simplificado contém quatro classes representando um cadastro de clientes, através da classe *Cliente,* um cadastro de produtos, através da classe *Produto*, informações de pedidos, através da classe *Pedido* com sua respectiva classe *Item de produto.* O esquema da figura 4.14 ilustra os relacionamentos entre essas classes. Os atributos que caracterizam cada classe são os mínimos possíveis para fins de clareza nos testes realizados.

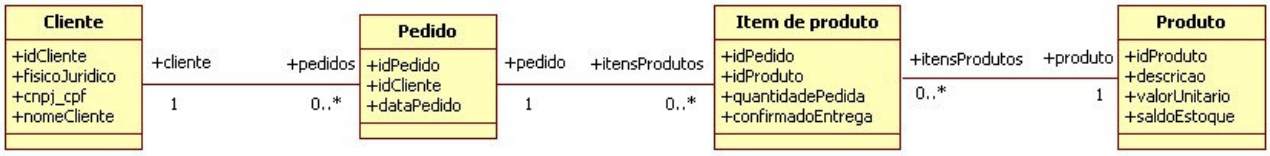


Figura 4.14 – Diagrama de classes do modelo de teste

4.5.1 – Principais características do ambiente de replicação

Deve-se lembrar que, em um ambiente implementado com replicação, existem *sites* que tem a possibilidade de desconectar-se. Por conseqüência, todas as aplicações necessariamente acessam os dados através de *localhost*. Para o *site* funcionar em modo desconectado, o banco de dados replicado tem que ser local, mas isto não é obrigatório. Nada impede que o banco de dados localize-se em qualquer *host,* desde que a conexão esteja sempre disponível. Por causa disto, nos testes realizados, quando houver a indicação de que um determinado *site* utiliza um determinado banco de dados, significa que o *site* tem acesso permanente ao banco de dados, não importando a localização real do mesmo.

As seguintes condições compõem o cenário de testes:

- Todas as tabelas (*Cliente, Produto, Pedido* e *Item do Produto*) são replicadas, sendo que as de *Cliente e Produto* são replicadas integralmente. Já as tabelas *Pedido* e *Item do Produto* podem ser fragmentadas horizontal e verticalmente, dependendo do *site*.

- Três *sites* participam do ambiente de replicação, todos com direito à escrita:

  – *Site Matriz*: Contém as informações integrais de todas as tabelas. É um *site* permanentemente conectado, pois tem funções de servidor. Utiliza banco de dados *PostgreSQL,* é monitorado através da porta 8000 do SGBD e o aplicativo utilizado é a ferramenta administrativa *pgAdmin PostgreSQL Tools,* representando sistemas de terceiros.

– *Site Alecrim*: Contém réplicas integrais das tabelas *Cliente* e *Produto*. Já as tabelas de *Pedido* e *Item de produto* são fragmentadas de forma a disponibilizar informações relativas apenas ao próprio *site.* Pode desconectar-se a qualquer momento. Utiliza como base de dados *DBaseIII plus* e é monitorado através de um arquivo *pipe*. Foi desenvolvido um aplicativo na linguagem Clipper 5.2 para o sistema operacional *DOS,* representando com isto as aplicações legadas

– *Site Tirol*: Contém réplicas integrais das tabelas *Cliente* e *Produto*. Já as tabelas de *Pedido* e *Item de produto* são fragmentadas de forma a disponibilizar informações relativas apenas ao próprio *site.* Este *site* pode desconectar-se a qualquer momento. Utiliza banco de dados *Hsqldb*, ambientado na aplicação e utilizando monitoração ambientada. Foi desenvolvido um aplicativo na linguagem *Java 1.6*, multiplataforma, representando as aplicações atuais e com possibilidade de ser embarcado. Pode acessar qualquer uma das bases citadas, mas no teste acessa apenas o *Hsqldb*.

A escolha desses três *sites*, com as características descritas, teve como objetivo principal permitir testes das características assíncronas previstas no modelo RPCD, da capacidade de realizar replicação entre bancos de dados heterogêneos e dos mecanismos destinados a realizar a fragmentação horizontal da base de dados, entre outros.

Para caracterizar ainda melhor as aplicações legadas, o *layout* do cadastro de clientes foi diferenciado no *Site Alecrim.* Os demais *layouts* serão idênticos em todos os *sites.* A seguir apresenta-se a descrição do *layout* do cadastro de cliente:

– *Site Matriz e Tirol*: Cliente(idCliente:String, fisicoJuridico:String=F|J, cnpj_cpf:String, nomeCliente:String)
– *Site Alecrim*: Cliente(codigo:String, tipo:integer=0|1, cnpj:String, nome:String, telefone:String)

A diversidade adotada nesse ambiente para o cadastro de clientes objetivou principalmente possibilitar a realização de testes dos mapeamentos realizados entre nome de campos e seus valores.

As seguintes regras são adotadas nos vários cenários de testes realizados:

- Qualquer *site* pode incluir, alterar e excluir itens do cadastro de *produto* e *cliente.*
- Não se pode incluir mais de um cliente com o mesmo *cnpj_cpf.*
- Não se pode excluir *cliente* com *pedido* ou *produto* existente em *item de produto.*
- Não se pode excluir *item de produto* se *confirmadoEntrega* for verdadeiro.
- Não se pode excluir *pedido* se existir *item de produto.*
- A quantidade de saldo em estoque (*saldoEstoque*) não pode ficar inferior a zero.

A adoção dessas regras teve como principal objetivo permitir o teste de sincronismo nas operações de escalonamento, conciliação e efetivação de transações.

4.5.2 – Configurações do ambiente de replicação

O primeiro passo diz respeito à definição do modelo global e sua configuração. O modelo global adotado corresponde na preparação do ambiente de testes ao esquema apresentado na figura 4.14. De modo a facilitar esse procedimento inicial, configurou-se o modelo global no banco de dados *PostgreSQL,* utilizando a ferramenta administrativa *pgAdmin,* e depois copiou-se as expressões para o arquivo de configuração do RPCD, adaptando-as à semântica do *Hsqldb*. Detalhes da configuração do ambiente estão apresentados na listagem em XML do Apêndice B. No arquivo de configuração do modelo global, criam-se as tabelas de cliente, produto, pedido e item de produto nos formatos especificados e se determina a ordem de precedência dessas tabelas para fins de escalonamento.

O próximo passo diz respeito às configurações locais de cada *site*, configurações estas que foram implementadas nos *sites* respectivos. Detalhes da configuração em XML dos *sites* locais são apresentados no Apêndice B. A configuração inclui a definição do modelo global com o qual o *site* está vinculado, a localização da base de dados no *site* local e a especificação das tabelas da base local que irão participar do ambiente de replicação e os respectivos mapeamentos entre a tabela local e a tabela global a nível de nome, campos e valores.

O passo seguinte diz respeito às configurações dos *RemoteSite*. Basicamente, para cada par de *sites* onde existe replicação, existe também um par de configurações praticamente idênticas, diferenciando-se apenas pelos critérios de fragmentação e endereços de *hosts*. Detalhes da configuração em XML dos *sites* remotos são apresentados no Apêndice B. A configuração inclui o modelo global com o qual o *site* remoto está vinculado, a especificação das tabelas que participam do ambiente de replicação e as respectivas fragmentações a que cada tabela está submetida.

Finalmente, configurou-se o esquema usado para a monitoração das bases de dados. Esse esquema é necessário para o funcionamento do modelo e seu arquivo de configuração em XML encontra-se no Apêndice B. Os principais aspectos configurados nesse arquivo se referem às portas TCP que serão monitoradas e o nome e localização dos arquivos *pipe* dos sistemas legados. Na monitoração das portas TCP informa-se a URL e a porta da base monitorada e a classe que interpreta o protocolo *front-end/back-end.*

A figura 4.15 ilustra o contexto de todas estas configurações.

4.5.3 – Teste de replicação de bases heterogêneas com os *sites* conectados

O primeiro teste iniciou-se com as bases sincronizadas e já com alguma informação armazenada. A figura 4.16 mostra os aplicativos com seus respectivos cadastros de clientes. Pode-se observar as diferenças entre os registros que compõe cada um desses três cadastros.

Para testar a replicação, os seguintes procedimentos foram realizados:

Site Alecrim – Inclui um cliente número 01004
Site Tirol – Altera o nome do cliente 01001
Site Matriz – Exclui o cliente 01002

Após a propagação e efetivação destas operações, observou-se que os dados foram replicados, conforme mostra a figura 4.17. O teste do mapeamento entre bases heterogêneas fica evidente

observando o *Site Alecrim*, onde o nome dos campos da base local são diferentes dos nomes nas demais bases e o valor do campo *fisicoJuridico* que foi convertido corretamente (*0=F*).

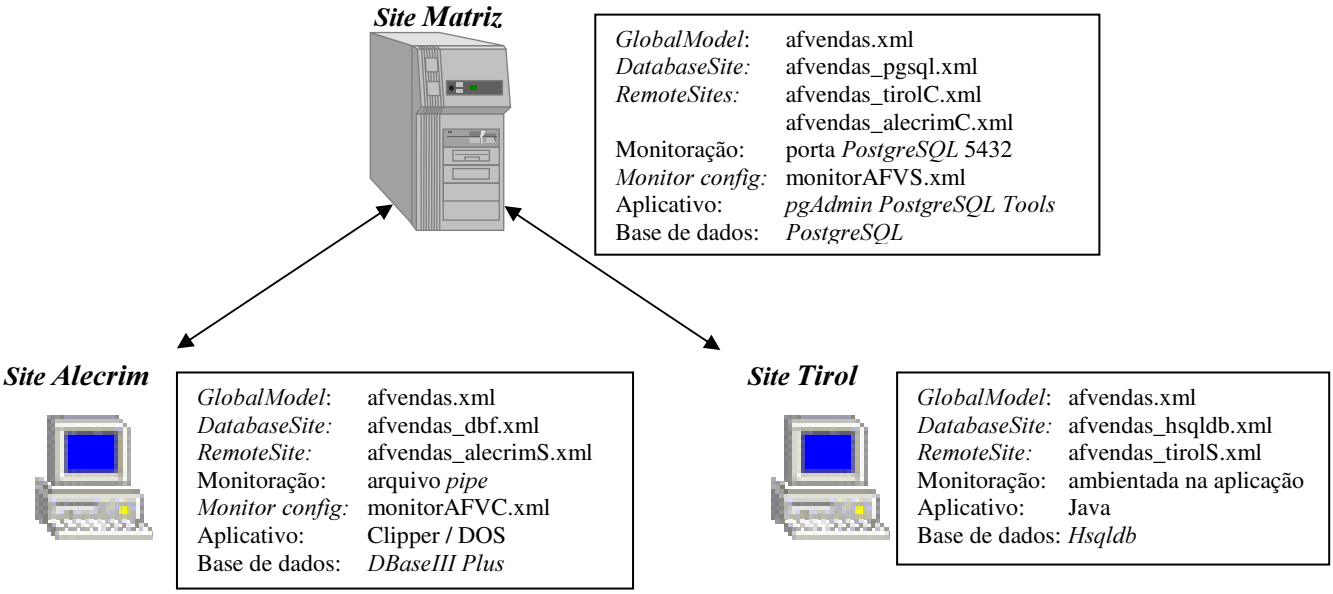
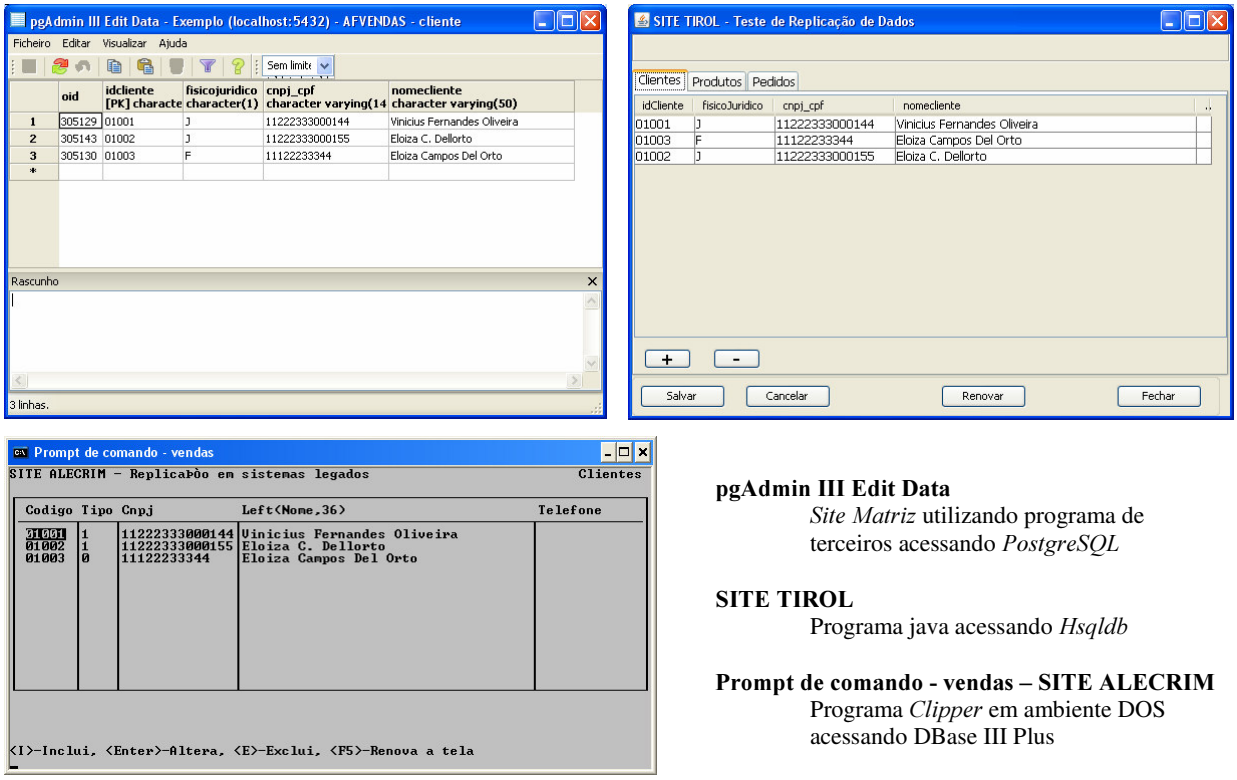


Figura 4.15 – Configuração do ambiente de testes



pgAdmin III Edit Data
Site Matriz utilizando programa de terceiros acessando *PostgreSQL*

SITE TIROL
Programa java acessando *Hsqldb*

Prompt de comando - vendas – SITE ALECRIM
Programa *Clipper* em ambiente DOS acessando DBase III Plus

Figura 4.16 – Cadastro de clientes nos três sites antes das alterações.

4.5.4 – Teste de replicação com *sites* inicialmente desconectados

O objetivo deste teste foi demonstrar de forma clara a característica assíncrona do modelo. Para isto, partindo-se da situação final do teste anterior, mostrado na figura 4.17, desconectaram-se todos os *sites* e os seguintes procedimentos foram realizados:

Site Alecrim – Inclui um cliente número 01005
Site Tirol – Altera o *cnpj_cpf* do cliente 01003
Site Matriz – Exclui o cliente 01004

A figura 4.18 mostra como ficaram os cadastros após as alterações realizadas e antes dos *sites* se conectarem. Após cada um se conectar, os cadastros foram ficando sincronizados e a figura 4.19 mostra os cadastros após se completar toda a sincronização. Este teste demonstra que a implementação permite um modo de funcionamento assíncrono, conforme preconizado no modelo RPCD, onde as operações são armazenadas e depois propagadas quando se estabelece conexão.

4.5.5 – Teste do escalonamento das operações

A base onde são armazenadas as operações aplicadas aos bancos de dados já é organizada de tal forma que já realiza um escalonamento automático das operações. Esta técnica utiliza o *ordenamento canônico*, descrito na seção 2.8.1.3. São ordenadas no primeiro nível pela cláusula *insert,* depois pela cláusula *update* e por fim, pela cláusula *delete.* Dentro de cada um destes tipos de operação, o segundo nível de ordenamento é feito pela *precedência*: primeiro os cadastros de mais alto nível (no teste, *Cliente* e *Produto*), depois as tabelas que possuem relações com outras tabelas, sendo que, quanto mais relações em cascata, mais a tabela é deixada por último (no teste, primeiro considera-se a tabela *Pedido* e depois *ItemProduto*). Finalmente, o último critério é o *timestamp*, prevalecendo sempre a última operação. No ambiente de replicação otimista, estes critérios resolvem grande parte dos problemas relativos ao escalonamento.

Para testar o escalonamento de operações, no ambiente desconectado, inclui-se um *pedido* com seus respectivos *itens* no *Site Alecrim,* sem cadastrar o cliente nem o produto nesse *site*. Foi escolhido o *Site Alecrim* para iniciar este teste, porque a ausência de controle nativo de integridade por parte do *Dbase*, tipo de base de dados utilizada pelo *site*, permite este tipo de procedimento. O *cliente* e o *produto* foram cadastrados no *Site Tirol*. Após os sites se conectarem, o *cliente*, o *produto*, o *pedido* e os *itens* apareceram corretamente em todos os *sites.* Apesar de, na prática, não acontecer este tipo de procedimento, a sua realização permitiu provar que o escalonamento funcionou corretamente. A figura 4.20 ilustra esta situação: a coluna da esquerda mostra os cadastros antes da sincronização e a coluna direita após a sincronização.

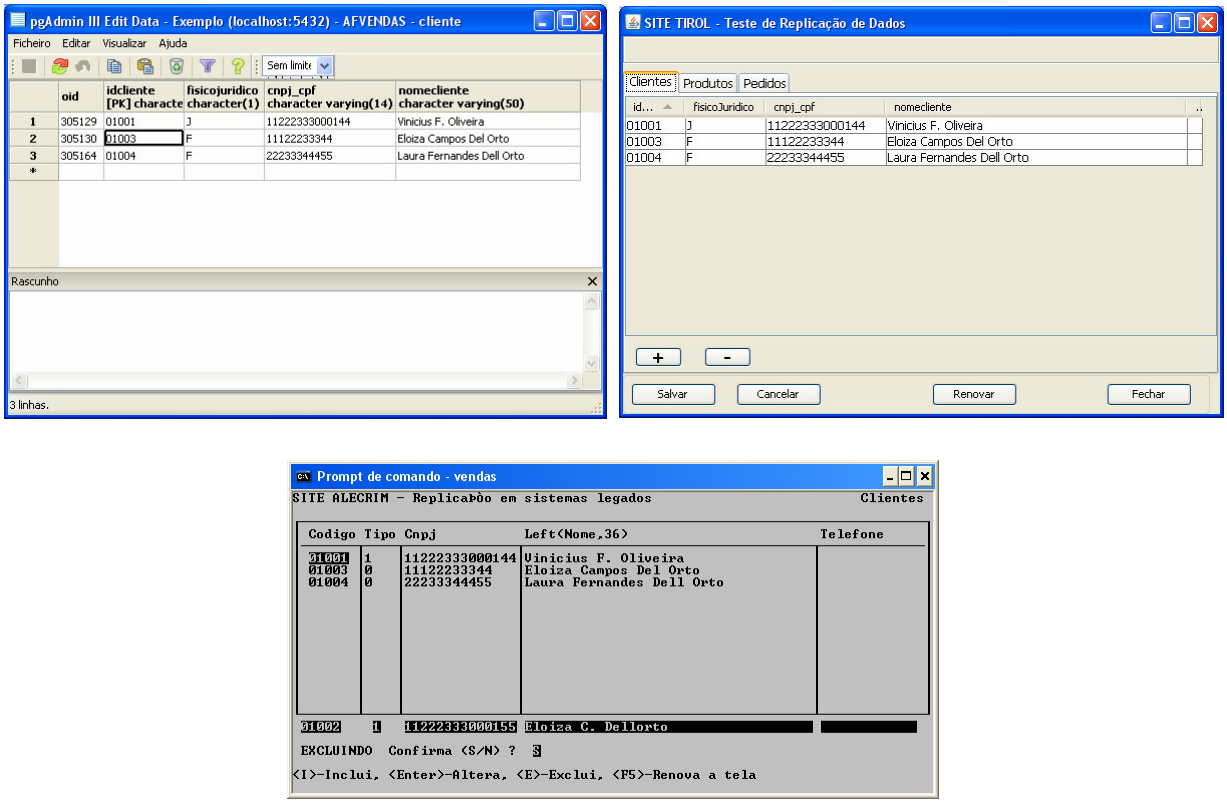


Figura 4.17 – Cadastro de clientes nos três sites depois da replicação.

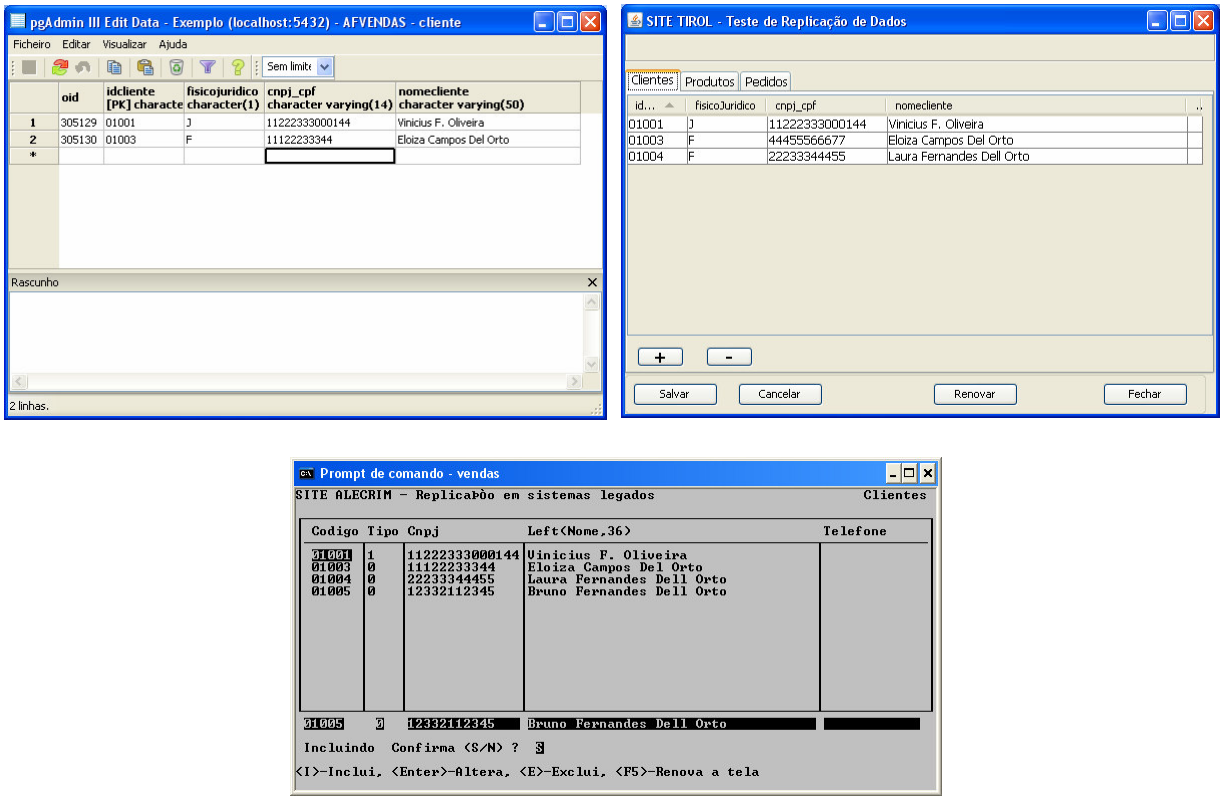


Figura 4.18 – Cadastro de clientes com alterações e com os sites *desconectados*

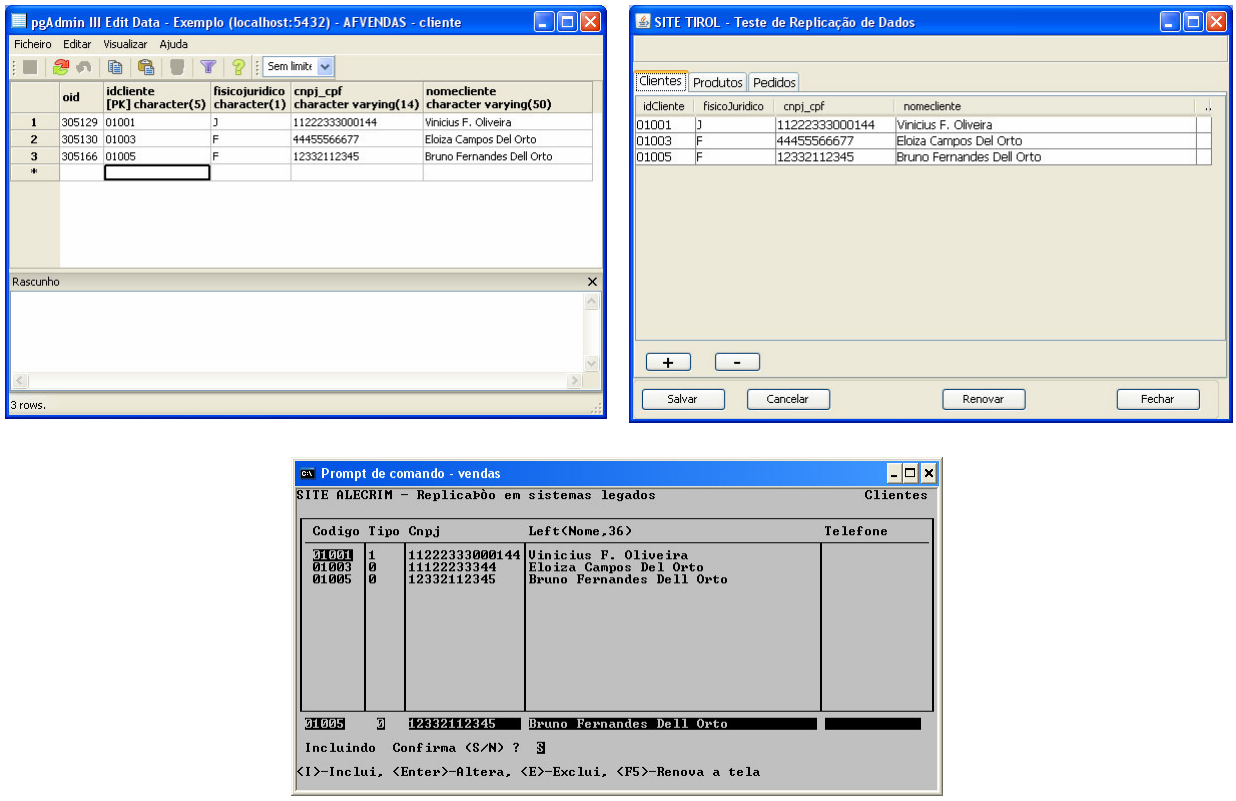


Figura 4.19 – Aplicativos teste mostrando que o cadastro de clientes foi sincronizado

4.5.6 – Teste de replicação em situações de conflito

Os testes anteriores não apresentaram possibilidades de conflito. Porém, apesar de o modelo ser baseado na replicação otimista, situações de conflito podem ocorrer. O custo da correção é alto, porém, espera-se que ocorram poucos problemas. A ferramenta armazena as mensagens de erro provenientes das bases de dados e aquelas geradas por ela própria. A ferramenta permite que o administrador e o usuário possam informar-se sobre erros ocorridos durante a replicação e que exigem algum tipo de medida corretiva. Seguem dois testes que demonstram o custo de uma correção.

No teste realizado, com os sites desconectados, o *Site Tirol* incluiu um *pedido* para um *cliente* e o *Site Matriz* eliminou o *cliente*. Ao ocorrer a sincronização, no *Site Matriz* ocorreu um erro na inclusão de um *pedido* sem *cliente,* no *Site Tirol,* o *cliente* e o *pedido* não foram eliminados porque a integridade referencial do banco de dados não permitiu. No *Site Alecrim,* terminou com um pedido sem o *cliente* respectivo, pois a base de dados não implementa integridade referencial. O problema foi comunicado ao administrador, para que este tome as devidas providências. A figura 4.21 mostra os avisos relativos ao problema. A solução para o caso foi solicitar ao *Site Matriz* incluir novamente o cliente eliminado. Após uma nova sincronização, o *Site Matriz* incluiu o *pedido*, o *Site Tirol* corretamente rejeitou a inclusão do *cliente,* pois o cliente já existia, e o mesmo foi incluído corretamente no *Site Alecrim.*

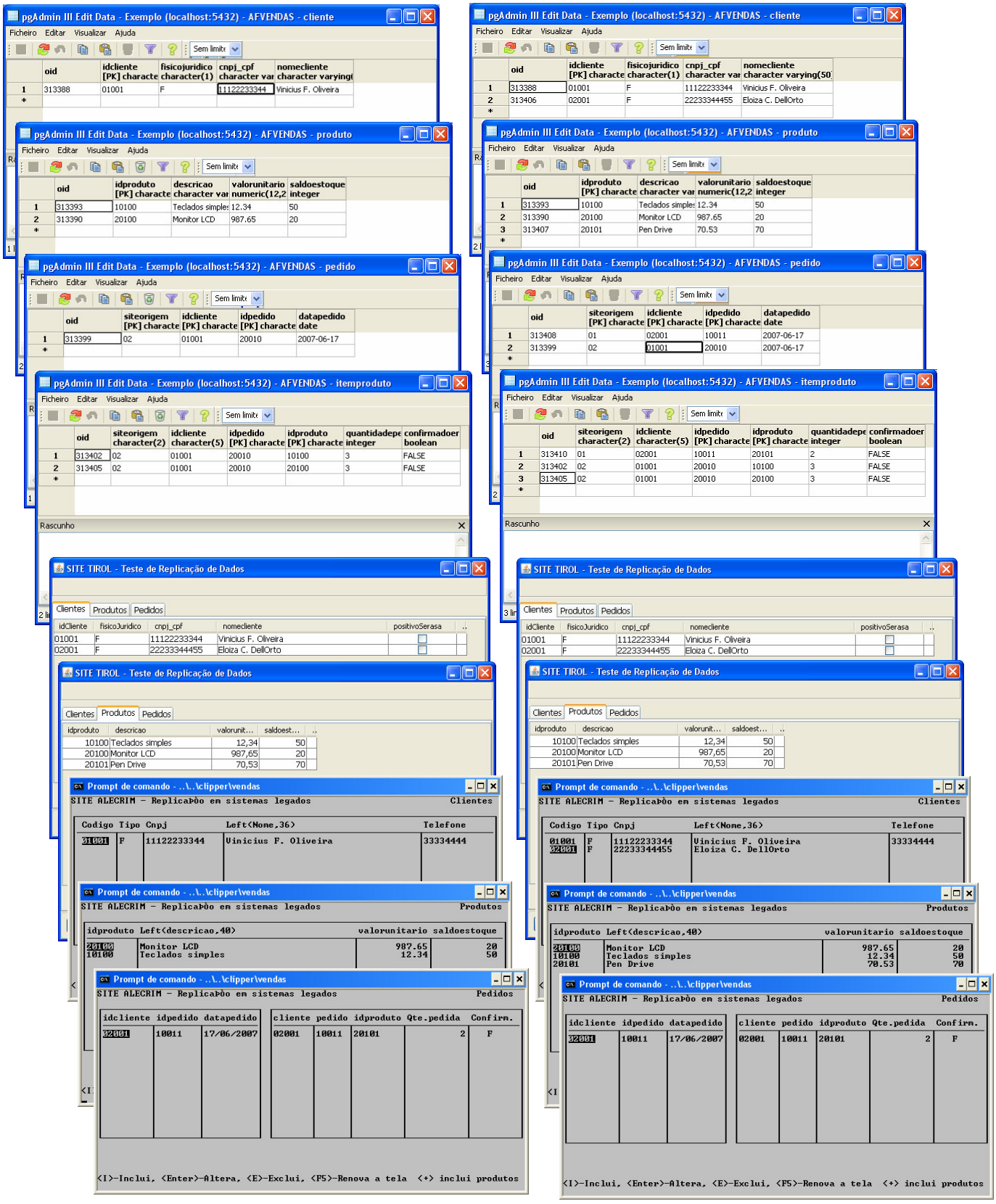


Figura 4.20 – Teste de escalonamento

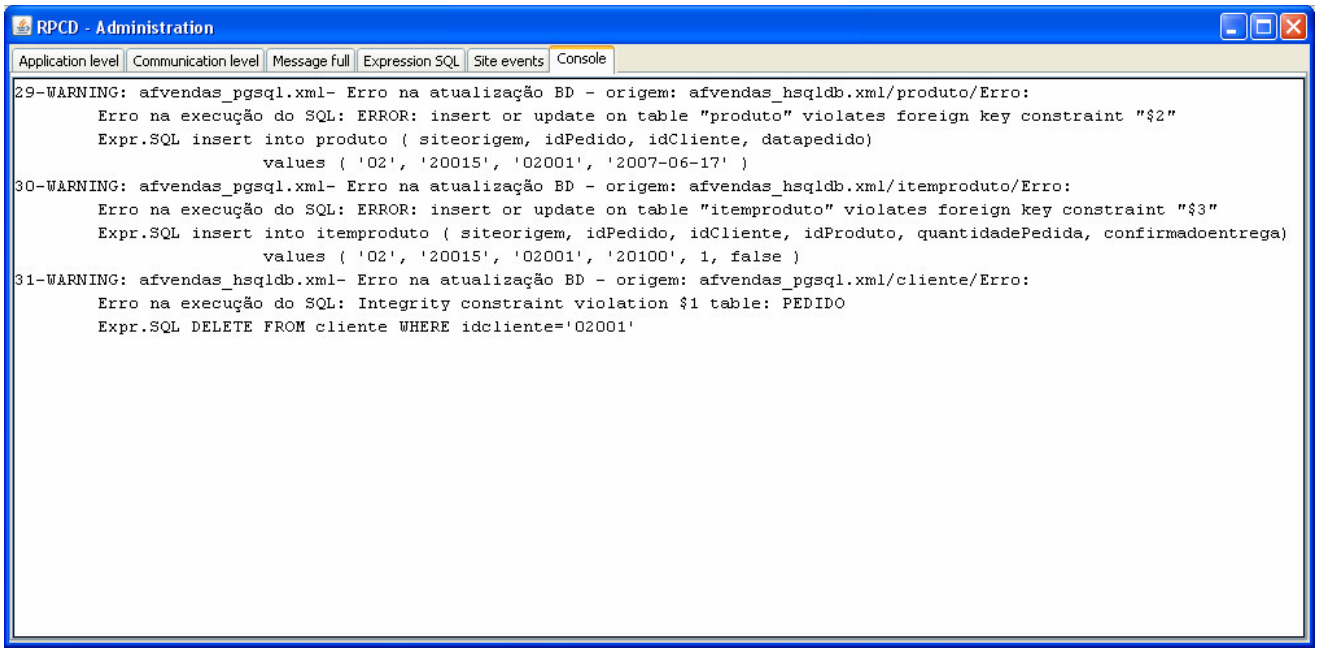


Figura 4.21 – Erros durante a conciliação.

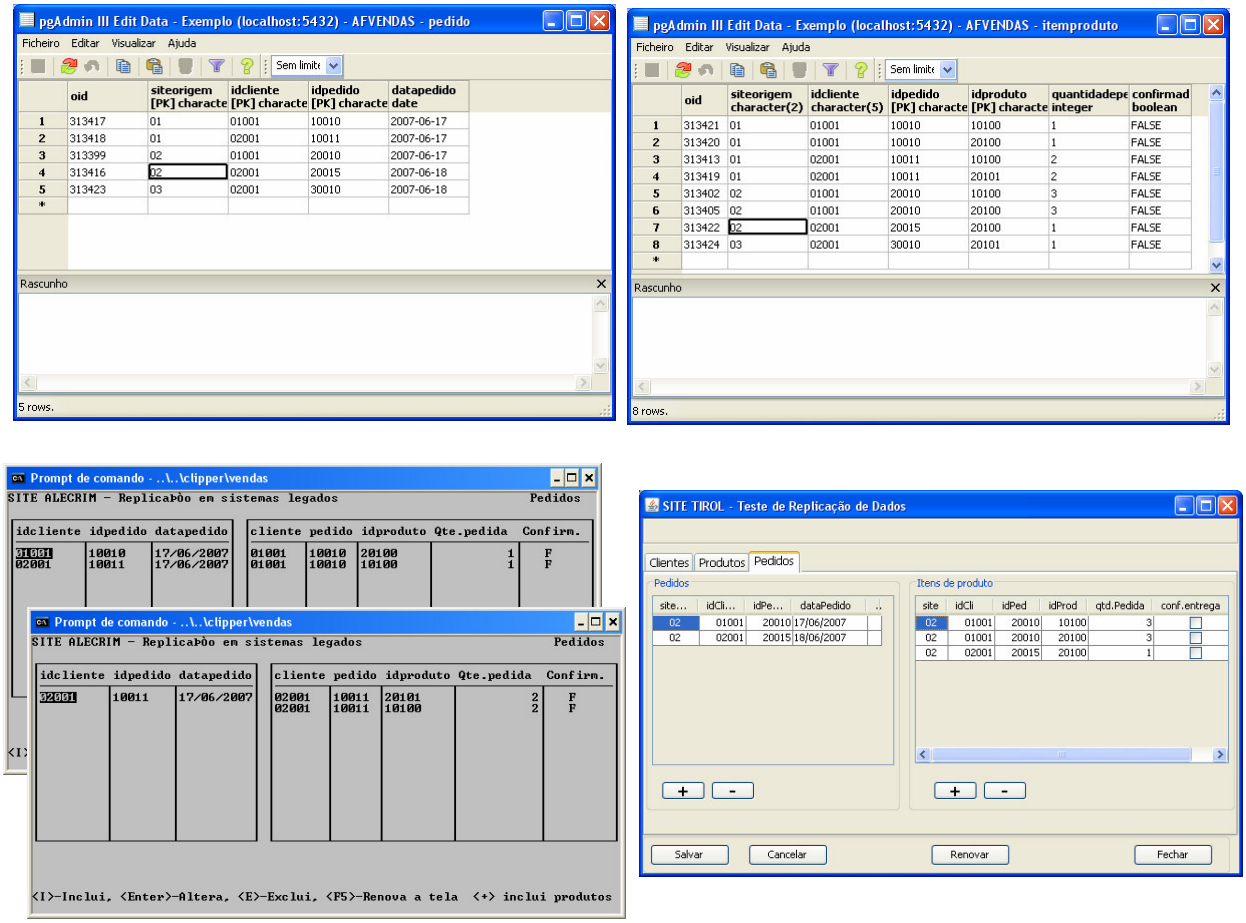


Figura 4.22 – Fragmentação dos dados

4.5.7 – Teste de fragmentação

Como previsto nas regras apresentadas no início da seção 4.5, os pedidos são disponibilizados apenas para seus respectivos *sites,* exceto o *Site Matriz* que centraliza todas as informações. Neste teste, incluíram-se um pedido no *Site Alecrim,* outro pedido no *Site Tirol* e outro pedido no *Site Matriz.* A figura 4.22 mostra que o *Site Matriz* de fato centralizou todas as informações, enquanto os outros dois *sites* continuaram a ver apenas seus próprios pedidos. Como teste complementar, incluiu-se no *Site Matriz* um pedido com o código do *Site Tirol* e este pedido ficou de fato disponível a este *site*.

4.5.8 – Operação da ferramenta

A figura 4.23 mostra como fica organizada no sistema de arquivos uma instalação típica desta ferramenta. A subpasta *afvendas* e o arquivo *ConexaoBD.properties* não são partes integrantes da ferramenta. Na subpasta fica armazenado o banco de dados do aplicativo teste e no arquivo ficam as configurações para acesso ao banco de dados do aplicativo. Foi por simples conveniência que tais arquivos foram colocados junto com a pasta da ferramenta. Quanto à ferramenta, na subpasta *bd* fica o banco de dados das operações aplicadas às bases de dados. Nas subpastas *databasesites, globalModels* e *remotesites* ficam os arquivos de configuração utilizados pelas respectivas classes para instanciar os objetos. O arquivo *monitorAFV.xml* é a configuração do monitor de acesso às bases de dados e *rpcdemo.jar* é o *bytecode* da ferramenta. No caso específico da figura 4.24, *rpcdemo.jar* é o aplicativo que contém de forma *ambientada* a ferramenta.

O modelo RPCD propõe que todas as atividades relativas à replicação sejam as mais transparentes possíveis ao usuário. A única informação que opcionalmente pode ser solicitado ao usuário é o *login* e *senha*. Já pela parte do administrador, o modelo exige bem mais: modelar, configurar o ambiente de replicação e mantê-lo íntegro exige muitas ferramentas auxiliares. Nas próximas seções, são detalhados os principais aspectos relativos a operação da ferramenta, tanto por parte do usuário quanto por parte do administrador..

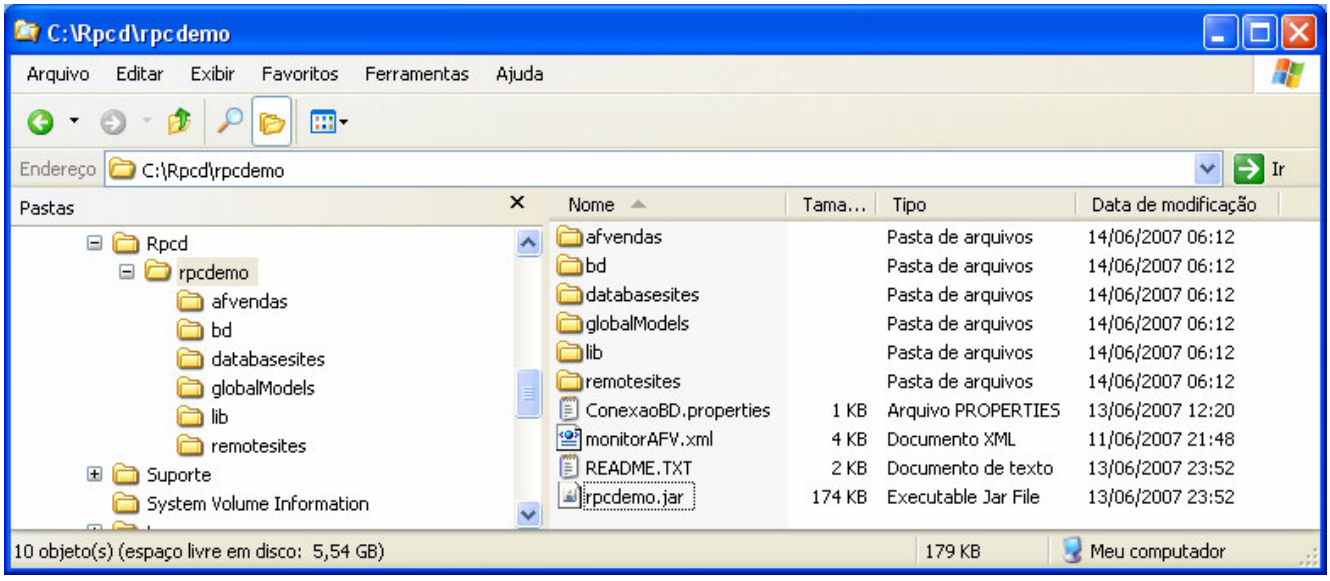


Figura 4.23 – Configuração das pastas em uma instalação típica

4.5.8.1 – Operação da ferramenta por parte do usuário

No caso desta implementação, o *login* e *senha* foram informados nos arquivos de configuração, dispensando a necessidade de solicitá-los ao usuário. Entretanto, caso seja necessário, a implementação disponibiliza determinados recursos através de uma interface, que tornam simples o desenvolvimento de uma tela solicitando o *login* e *senha* do usuário.

No caso da ferramenta ser ambientada no aplicativo, o usuário não pode desativá-la separadamente e, portanto, para desativá-la tem que fechar o aplicativo. No caso das aplicações legadas, em situações normais de uso, a ferramenta só será desativada quando o *host* for desativado. Contudo, é possível o usuário interromper os serviços de replicação e comunicação, utilizando comandos no *prompt* do console, ou se o aplicativo tiver uma opção para interagir com a ferramenta.

Nesta implementação, no *Site Tirol* a ferramenta está ambientada no aplicativo desenvolvido para este *site*: ao iniciá-lo, as partes de replicação e comunicação são ativadas e permanecem assim até o momento em que o aplicativo é fechado.

No caso do *Site Alecrim*, que é uma aplicação legada, a replicação ocorre através da monitoração de um arquivo *pipe*. No *script* de inicialização, que nada mais é do que um arquivo de lote com extensão *bat* do DOS, o monitor é ativado no modo console e todas as mensagens, caso hajam, são gravadas em arquivos *log*. Se por algum motivo o monitor não for ativado, isto não gera problemas, pois o aplicativo legado foi alterado para gravar as alterações, como previsto no modelo. No momento em que o monitor for ativado, as informações gravadas pelo aplicativo legado são lidas e transmitidas ao *site* servidor. As atualizações que chegam são atualizadas automaticamente na base de dados.

No *Site Matriz,* o administrador é quem prepara a ativação da ferramenta, que neste teste é feito no modo gráfico. Na hipótese de a ferramenta não ser ativada, os aplicativos que acessam a base *PostgreSQL* não terão acesso à base, pois estes estão configurados para serem atendidos pela ferramenta do modelo RPCD e não diretamente pelo banco de dados. Sendo assim, não se corre o risco de haver perda de informação.

4.5.8.2 – Operação da ferramenta por parte do administrador

No caso desta implementação, existem duas categorias de interface com o usuário: uma operacional e outra de apoio a testes, auditoria e configurações.

Interfaces operacionais:

As interfaces operacionais foram desenvolvidas no modo gráfico e no modo console. Devido a facilidade de recursos, a interface gráfica possui um pouco mais de funcionalidades que o modo console, mas nada que comprometa a utilização da ferramenta no modo console. Em ambos os modos, o administrador pode iniciar e parar os procedimentos de monitoração das bases de dados, de replicação e de comunicação, seja no modo servidor ou no modo cliente. A figura 4.24 mostra estas interfaces. As abas indicam a função implementada: *Application Level* inicia a replicação e monitoração, podendo iniciar cada um separadamente; *Comunication level* inicia a comunicação entre *sites* e permite que se consulte, a qualquer momento, os detalhes da comunicação, a saber: se foi bem sucedida a associação, em qual estado o *site* encontra-se, se efetivamente está havendo troca de informações, entre outras; da aba console saem todas as mensagens de processo (o *popmenu* visível à direita permite

escolher o nível de detalhamento das mensagens); por último, a interface console que oferece também diversas opções de inicialização, sendo que as mensagens do processo são gravadas em arquivos *log*. As abas *Message Full, Expression SQL* e *Site events* também são áreas de saída de mensagens (não mostrado na figura): as duas primeiras são relacionadas à monitoração, permitindo observar o tráfego de mensagens, e a terceira indica os eventos ocorridos em cada *site* participante do ambiente de replicação.

O administrador pode interagir diretamente com os *sites* conectados, se selecionar a opção *Detalhes* na aba *Comunication Level*. Dependendo do modo em que estiver operando (servidor ou cliente) verá uma ou outra das telas mostradas na figura 4.25. No modo servidor, o número em cada aba indica a porta TCP onde existe um cliente conectado, e, escolhendo a aba, pode interagir diretamente com o cliente. No modo cliente existe apenas uma aba. As opções do menu indicam as primitivas disponíveis para solicitar serviços ao protocolo RPCD. Na figura, por exemplo, o servidor solicitou a primitiva *InquiryRequest/echo.* Com este recurso, o administrador pode assumir total controle sobre o *host* conectado, permitindo ações mais críticas, tais como: alteração na estrutura das tabelas, análise dos arquivos *log*, alteração do *timestamp* e outras tarefas de caráter administrativo.

Interfaces de apoio:

Como todas as configurações da ferramenta são realizadas através de arquivos XML, não foi desenvolvida nenhuma ferramenta para tal, uma vez que editar as configurações não é uma tarefa complexa e existem disponíveis na I*nternet* e comercialmente bons editores de XML. As configurações desta ferramenta, por sua vez, também não são trabalhosas, com apenas duas exceções: os *mapeamentos* entre as bases local e global e os comandos *create table* do modelo global, o que exigiu o desenvolvimento de uma ferramenta auxiliar.

A figura 4.26 mostra as principais interfaces de apoio implementadas. A aba *Drives* carrega todos os *drives* disponíveis e é o ponto de partida para as demais opções. A aba *Conexão* permite acesso, com lista de catálogo, a todos os bancos para os quais a ferramenta foi desenvolvida, permitindo escolher os caminhos e ainda iniciar a monitoração das bases (botão no alto à direita), incluindo-se na interceptação e tendo acesso ao tráfego de mensagens (não mostrado na figura). A aba *SQL* permite enviar quaisquer comandos SQL para a base conectada, além de persistir todas as expressões já solicitadas (o administrador pode excluí-las quando quiser). A aba *Statements* facilita a edição de um registro selecionado na aba *SQL,* monta seis tipos de expressões SQL para atualização do banco de dados e gera a expressão *create table* compatível com a tabela selecionada. A principal função desta aba, além da geração da expressão *create*, é possibilitar testes diversos com as bases de dados. A aba *Global x Local* auxilia no mapeamento entre bases locais e o modelo global: a partir de uma tabela local selecionada na aba *SQL* e escolhendo um modelo global, faz-se o mapeamento desejado e, no final, pode-se gerar o arquivo XML representativo deste mapeamento. Na janela flutuante mostrada na figura, pode-se ainda editar o documento gerado e depois copiá-lo para qualquer editor XML através da área de transferência. A aba *Anotações* apenas persiste qualquer informação editada em sua área de texto.

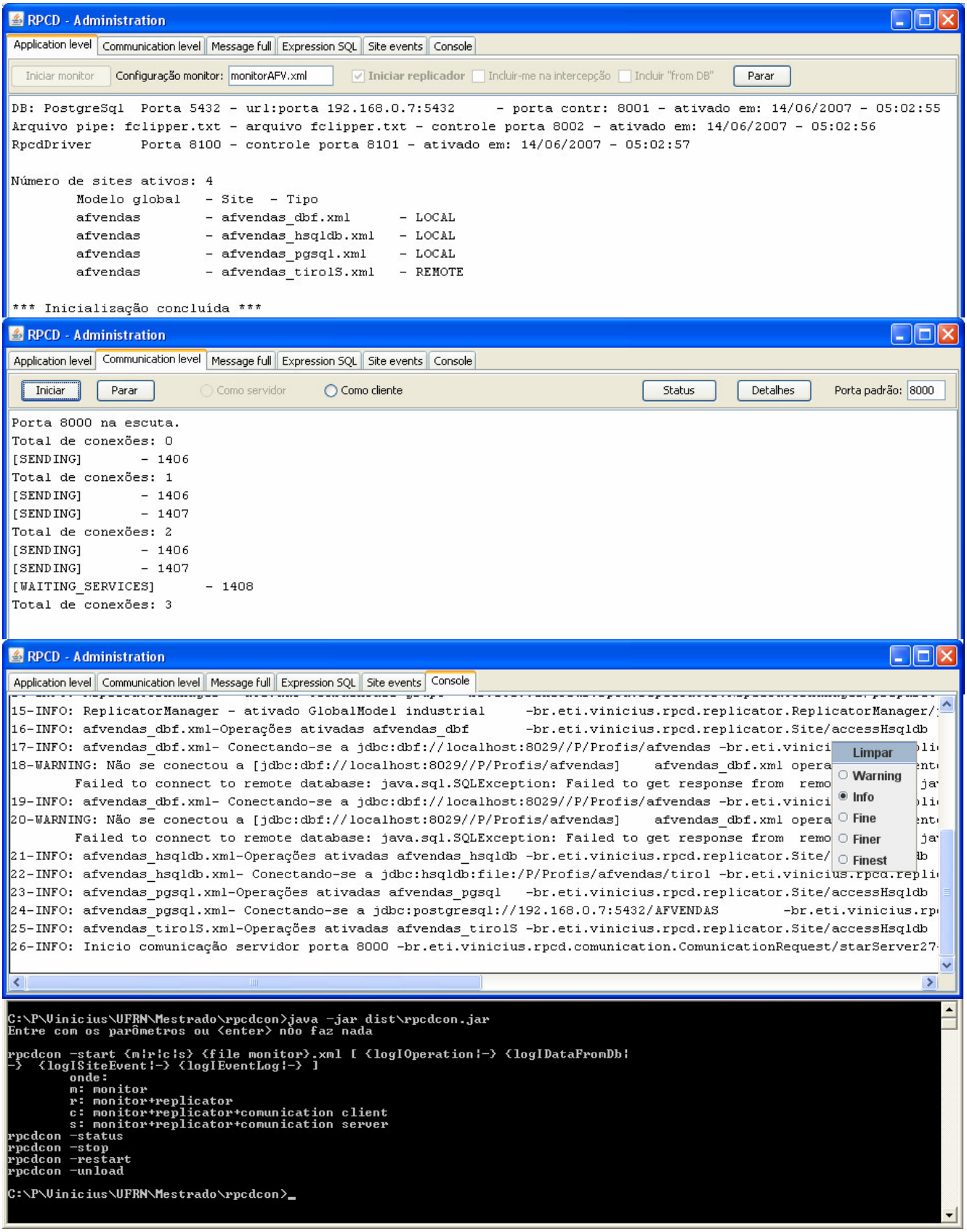


Figura 4.24 – Interfaces de inicialização da ferramenta

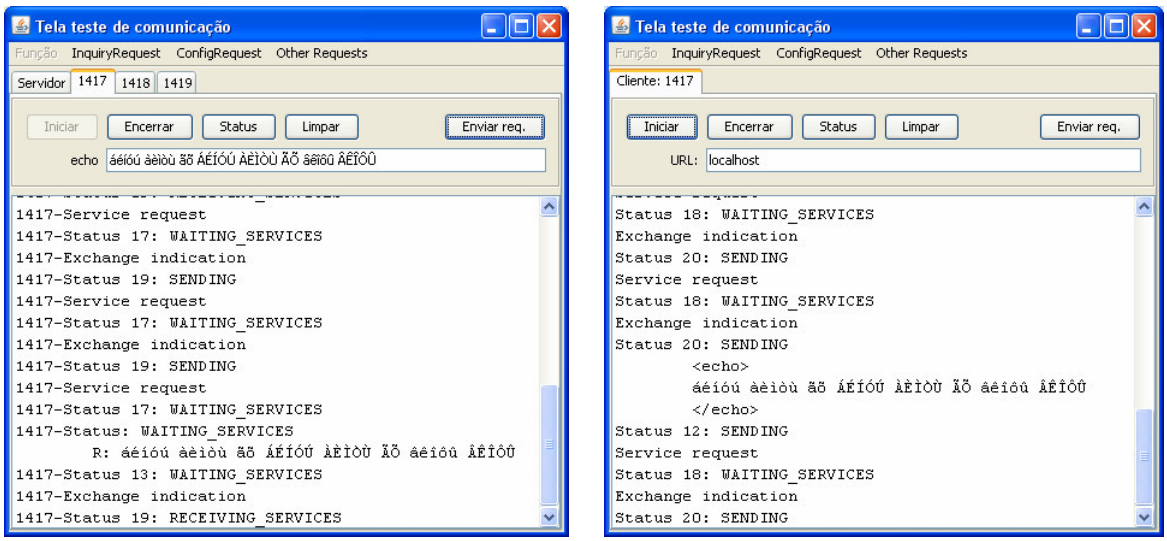


Figura 4.25 – Opção *Detalhes* da aba *Comunication level*

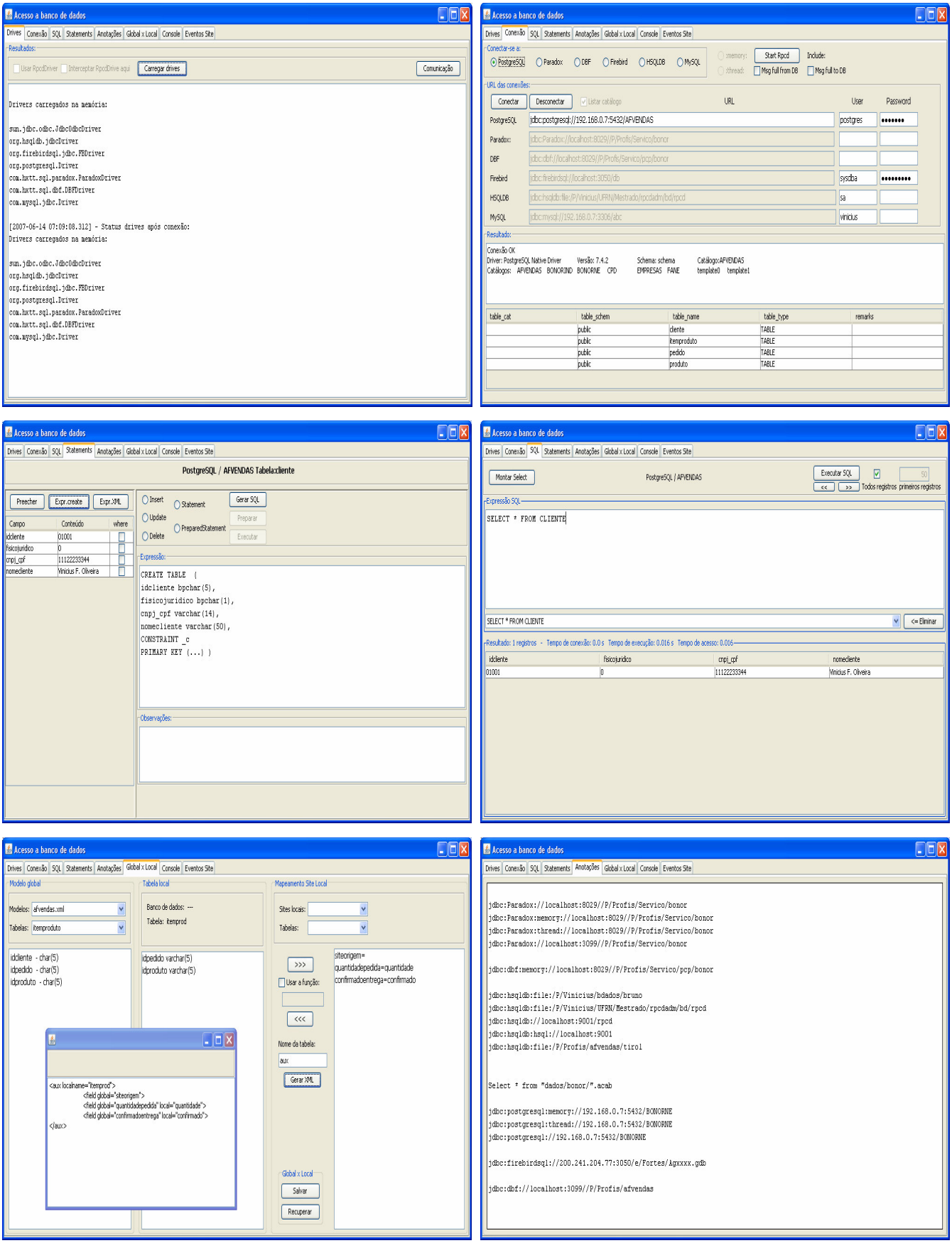


Figura 4.26 – Interfaces de apoio ao administrador

4.6 – Comentários

Esta implementação cumpriu todas as fases de um projeto de *software*, desde a análise dos requisitos até os testes finais. Muito poderia se comentar sobre cada fase, mas ficaria muito extenso e fugiria do objetivo desta dissertação, que é apresentar um modelo para replicação de bases heterogêneas de forma assíncrona. Os maiores desafios, e que tem ainda a possibilidade de melhorar, foram os mapeamentos e a interceptação das operações aplicadas às bases de dados.

As dificuldades na interceptação se devem ao fato de não existir um mecanismo formal para acesso às operações aplicadas às bases de dados, que é um ponto fundamental para o sucesso das replicações de bases de dados, e obrigou a se fazer uso das estratégias apresentadas ao longo deste capítulo.

As dificuldades do mapeamento surgem na interpretação das expressões SQL. A linguagem SQL é poderosa e versátil, acarretando uma enorme variedade de expressões possíveis. Para que o mapeamento seja de fato seguro, seria necessário analisar as expressões SQL utilizando técnicas similares àquelas utilizadas pelas linguagens de programação, durante a fase de interpretação dos códigos. Tais técnicas são complexas e não foram utilizadas nesta implementação. Foi usada, para esta implementação, uma técnica mais simples, que poderia ser chamada de "análise por assinatura". Utiliza-se, para esta técnica, o fato de se conhecer o nome dos campos que podem participar das expressões e, por conseqüência, é possível conhecer a sua "assinatura" numa expressão de tipo: *<campo><delimitador>[<valor>]*, onde o *<delimitador>* pode ser uma operação lógica, vírgula ou espaço e o *<valor>* nem sempre é obrigatório. Para a assinatura ser ainda mais facilmente reconhecida, toda expressão SQL é formatada de forma a garantir que qualquer campo fique expresso na forma *<espaço><campo><delimitador>[<valor><espaço>]* e com formato *case insensitive.* Para preservar o conteúdo dos valores de texto (tipo *string*), procura-se também a sua assinatura: *[<espaço>|<espaço><campo>]<delimitador><aspas><conteúdo><aspas><delimitador><espaço>* Durante todos os testes realizados esta técnica apresentou bons resultados.

Observou-se que a convergência das bases de dados replicadas possui uma forte dependência das configurações locais dos sistemas gerenciadores de banco de dados. Por exemplo, pode acontecer de um *site* configurar o SGBD para tratar a integridade referencial em *cascata*[23] e outro *site* não configurar da mesma forma. Nesse caso, em situações de conflito, muito provavelmente, após a sincronização, as bases terminarão divergentes.

Outro ponto que exigiu bastante cuidado e precisão para implementar foi o protocolo, devido à necessidade de se programar utilizando *multithreading*[24], pois, quase todas as primitivas e serviços, por definição, são assíncronos e precisam utilizar *thread.*

Quanto à linguagem Java, esta não apresentou nenhum tipo de restrição. Todos os recursos necessários estavam disponíveis, desde a utilização de *SocketChannel* (uma forma de implementar *sockets*, disponível a partir do Java 1.5 e que apresenta ótimo desempenho),

[23] Significa, por exemplo, que ao eliminar um registro de uma tabela, todos os demais registros de outras tabelas que possuam uma relação 1-n serão eliminados

[24] Uma tradução sugerida pode ser "múltiplos fluxos de execução", mas os termos *thread* e *multithread* já são consagrados pelo uso.

passando pela elegante forma de tratar o *multithreading,* até ao tratamento de textos, com a utilização de expressões regulares (*regular expression*)[25].

Mesmo não havendo medidas de desempenho e escalabilidade, os testes desenvolveram-se dentro de expectativas aceitáveis, tanto em redes locais como utilizando a Internet. Muitos dos módulos implementados podem incluir facilmente contadores de volume e tempo e disponibilizar tais informações para estatísticas. Isto, porém, está planejado para ser implementado somente após a consolidação da ferramenta ou quando, durante sua utilização efetiva, ocorrerem situações em que a medida de desempenho e escalabilidade torna-se uma necessidade crítica.

[25] "Expressões regulares" é uma semântica para tratar textos e dados. Utilizado pelas mais populares linguagens (*Perl, Java, VB.NET and C#, .NET Framework, PHP, Python, Ruby, Tcl, MySQL,* awk*, and Emacs)* tem-se tornado um padrão e possibilitado criar poderosas ferramentas para manipular textos e dados.

Capítulo 5

Conclusões

São muitas as dificuldades inerentes ao processo de replicação de base de dados. Mas, por outro lado, os benefícios que o processo de replicação pode proporcionar são muito relevantes, o que justifica os esforços na procura de soluções e paradigmas com o objetivo de resolver, ou pelo menos contornar, tais dificuldades.

Se a replicação de uma base de dados homogênea já apresenta seus próprios desafios, replicar bases heterogêneas acrescenta mais um conjunto de novos desafios. Além dos cuidados que se deve ter no que diz respeito à replicação, bases heterogêneas exigem cuidados adicionais na forma como a informação é processada e representada. O que mais incentiva, porém, na procura de boas soluções para tudo isto é que no mundo real, quase todas as empresas são obrigadas a conviver com mais de um tipo de base de dados e muitas vezes exigindo troca de informações entre elas.

O estudo teórico revelou uma grande diversidade de propostas, conceitos e técnicas utilizados nos ambientes de replicação de dados. O modelo RPCD reúne em um só projeto muitos dos elementos abordados neste estudo, separando-os em módulos, definindo responsabilidades das partes que compõe o modelo e sugere algumas técnicas para atingir os seus objetivos. A maior vantagem disto é que cada parte do modelo pode ser administrada separadamente e, por conseqüência, facilitar a alocação de recursos e melhorar a avaliação de resultados.

A implementação de uma ferramenta, baseada no modelo RPCD, proporcionou uma visão mais profunda do próprio modelo, aproximou-lhe dos problemas de ordem prática e conferiu-lhe um pouco mais de maturidade. Vale a pena destacar que durante o desenvolvimento, partes da ferramenta se mostraram muito úteis em outros tipos de aplicação. A monitoração dos acessos a bancos de dados ajudou a depurar diversos programas, assinalando acessos desnecessários ao banco e conexões mal utilizadas. Outro exemplo foi a utilização de canais de *socket* da linguagem Java, que se mostrou um mecanismo muito eficiente para conectar redes distintas, algo que pode ser útil no desenvolvimento de *gateways* ou *firewalls*.

A ferramenta implementada também permitiu que se validassem as premissas consideradas para o modelo, uma vez que os testes realizados verificaram a real possibilidade de bases heterogêneas coexistirem em um mesmo ambiente. Da mesma forma, foi possível constatar que a replicação ocorreu de forma assíncrona e que o protocolo de comunicação de fato transportou as mensagens desejadas entre as bases de dados. A operação da ferramenta se mostrou transparente às aplicações de "terceiros" em uma de suas modalidades de operação, quando utiliza a técnica de interceptação da porta TCP, e com poucas alterações para os aplicativos de usuário, quando utiliza sua outra modalidade de operação, usando a forma ambientada.

Como trabalhos futuros com relação à ampliação do escopo do modelo, existem duas possibilidades que já são possíveis atualmente, mas que necessitariam de uma análise mais detalhada: utilizá-lo em redes com a topologia em malha e como um método alternativo para a manutenção de tabelas na modelagem multidimensional. Maiores detalhes sobre tais possibilidades encontram-se descritos na seção 3.5. Com relação ao desenvolvimento da

ferramenta, pode-se também identificar possibilidades de ampliação de seu escopo, considerando o atendimento a outros gerenciadores de bases de dados. Seria ainda possível o aperfeiçoamento dos algoritmos utilizados para os procedimentos de mapeamento e sincronização das operações aplicadas às bases de dados, tornando-os mais precisos e eficientes. Finalmente, uma outra possibilidade é desenvolver módulos auxiliares para medir o desempenho e a escalabilidade da ferramenta.

Todo o trabalho desenvolvido em torno do modelo RCPD, provou – ou pelo menos assinalou – que, se houver um mínimo de colaboração por parte dos fornecedores de bancos de dados, é possível integrar de forma sistematizada bases de dados de diferentes tipos, algo que é de grande interesse de toda a comunidade vinculada à tecnologia de informação.

Referências bibliográficas

[Anderson,1998] ANDERSON, TODD; BREITBART, YURI; KORTH, HENRY F.; WOOL, AVISHAI; Replication, Consistency and Practicality: Are These Mutually Exclusive ? SIGMOD'98 Seattle, WA, USA ACM; 0-89791-995-5/98/006.

[Bauer,05] Bauer, Christian; King, Gavin; Hibernate em Ação Rio de Janeiro: Editora Ciência Moderna Ltda., 2005

[Bond,03] Bond,Martin; Haywood,Dan; Law,Debbie; Longshaw,Andy; Roxburgh,Peter, Aprenda J2EE em 21 dias São Paulo: Pearson Education do Brasil, 2003

[Breitbart,1997] BREITBART, YURI; KORTH, HENRY F.; Replication and Consistency: Being Lazy Helps Sometimes; PODS'97 Tucson Arizona USA; ACM 0-89791-910-6/97/05.

[Coda,2007] CARNEGIE MELLON UNIVERSITY – SCHOOL OF COMPUTER SCIENCE; The Coda Distributed File System
http://www.coda.cs.cmu.edu
http://www.coda.cs.cmu.edu/ljpaper/lj.html

[Cunha, 2003] CUNHA, DANIEL PEZZI Um estudo das estratégias de replicação e reconciliação de banco de dados móveis em um ambiente wireless. Florianópolis: UFSC,2003. Dissertação (Mestre em Ciência da Computação) Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina.

[Firebird,2007] FIREBIRD FOUNDATION INCORPORATED; Firebird
http://www.firebirdsql.org

[Gao,2003] GAO, LEI; DAHLIN,MIKE; NAYATE,AMOL; ZHENG,JIANDA; IYENGAR,ARUN; Application Specific Data Replication for Edge Services; WWW2003; ACM 1-58113-680-3/03/005; May 20-24,2003, Budapest, Hungary.

[Gianuzzi,2004] GIANUZZI, VITTORIA; Data Replication Effectiveness in Mobile Ad=Hoc Networks. PE-WASUN'04; ACM 1-58113-959-4/04/0010, October 7,2004, Venezia, Italy.

[Graham,1995] GRAHAM,PETER; BARKER, KEN; REZA-HADAEGH, AHMAD; Disconnected Objects: Reconciliation in a Nested Objects Transaction Environment.

[Gray, 1996] GRAY, JIM; HELLAND,PAT; O'NEIL,PATRICK; SHASHA, DENNIS; The Dangers of Replication and a Solution. SIGMOD '96 6/96 Montreal, Canadá

[Kermarrec,2001] KERMARREC, ANNE-MARIE; ROWSTRON, ANTONY; SHAPHIRO,MARC;DRUSCHEL, PETER; The IceCube approach to the reconciliation of divergent replicas; Twentieth ACM Symposium on Principles of Distributed Computing (PODC 2001), 26-29 August 2001, Newport, Rhode Island (USA)

[Hsqldb,2007] HSQL DEVELOPMENT GROUP; Hsql Database Engine
http://hsqldb.org

[Holliday,2002] HOLLIDAY, JOANNE; AGRAWAL, DIVYAKANT; ABBADI, AMREL; Disconnection Modes for Mobile Databases. Wireless Networks 8, 391-402, 2002

[Holzner,2001] Holzner, Steven; Desvendando XML Rio de Janeiro: Campus, 2001

[Hxtt,2007] HONGXIN TECHNOLOGY & TRADE LTD.; JDBC Databases
http://www.hxtt.com

[IceCube,2007] CAMBRIDGE MICROSOFT RESEARCH CENTRE;
http://research.microsoft.com/camdis/icecube.htm

[Java,2007] SUN MICROSYSTEMS, INC.; The Source for Java Developers
http://java.sun.com

[Lindholm,2003] XML Three-way Merge as a Reconciliation Engine for Mobile Data; MobiDE'03; ACM-1-58113-767-2/03/0009; September 19, 2003, San Diego, California, USA.

[Microsoft,2007] MICROSOFT CORPORATION; SQL Server 2005 Books Online - SQL Server Replication
http://msdn2.microsoft.com/en-us/library/ms151198.aspx

[Molli,2003] MOLLI, PASCAL; SKAF-MOLLI, HALA; OSTER, GERALD; IMINE, ABDESSAMAD; Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. ACM, Group´03, November 9-12,2003

[Musolesi,2002] MUSOLESI, MIRCO A Middleware for Data-sharing in Ad Hoc Networks. Bologna: UB,2002. Dissertação (Laurea in Ingegneria Elettronica) Facoltá di Ingegneria, University of Bologna

[Mysql,2007] MYSQL AB; MySQL, The world's most popular open source database
http://www.mysql.org

[NetBeans,2007] NETBEANS CA; NetBeans IDE
http://www.netbeans.org

[Oracle,2007] ORACLE CORPORATION; Oracle7 - Server Distributed Systems, Volume II: Replicated Data
http://www.lsbu.ac.uk/oracle/oracle7/server/doc/SD273/toc.htm

[Phatak,2004] PHATAK, SHIRISH HEMANT; NATH, BADRI; Transaction-Centric Reconciliation in Disconnected Client-Server Databases. Mobile Networks and Applications 9, 459-471, 2004

[Pender,2004] PENDER, TOM; UML a Bíblia. Rio de Janeiro: Elsevier, 2004, Editora Campus

[Pervasive,2007] PERVASIVE SOFTWARE INC; Pervasive DataExchange - Data Replication Technology
http://www.pervasive.com/dataexchange/tech.asp

[Postgre,2007] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP; PostgreSQL
http://www.postgresql.org

[Ratner,98] RATNER, DAVID HOWARD Roam: A Scalable Replication System for Mobile and Distributed Computing. Los Angeles: UCLA, 1998. Dissertação (Doctor of Philosophy in Computer Science) Computer Science Department, University of California

[Saito,2002] SAITO, YASUSHI; SHAPIRO, MARC; Replication: Optimistic Approach HP Laboratories Palo Alto, HPL-2002-33 February, 2002

[Saito,2005] SAITO, YASUSHI; SHAPIRO, MARC; Optimistic Replication. ACM Computing Surveys, v.37,Nº1, March 2005, pp 42-81

[Schütt,2003] SCHÜTT,THORSTEN; SCHINTKE,FLORIAN; REINEFELD,ALEXANDER; Efficient Synchronization of Replicated Data in Distributed Systems; Zuse Institute Berlin (ZIB); EU GridLab; 2003.

[Sheth 1990] SHETH A.; J. LARSON; Federated database systems for managing distributed, heterogeneous and autonomous databases. ACM Computing Surveys, 22, 3, pp. 183-236, 1990.

[Silber,99] Silberchatz, Abraham; Korth, Henry F. e Sudarshan, S. Sistema de Banco de Dados São Paulo: Pearson Makron Books, 1999

[StarUML,2007] The Open Source UML/MDA Platform
http://www.staruml.com

[Suel,2003] SUEL,TORSTEN; NOEL, PATRICK; TRENDAFILOV, DIMITRE; Improved File Synchronization Techniques for Maintaining Large Replicated Collections over Slow Networks. CIS Department Polytechnic University Brooklyn, NY 11201 / NSF CCR-0093400

[Sun,2007] SUN MICROSYSTEMS, INC.; Sun StorEdge Network Data Replicator
http://www.sun.com/products-n-solutions/hardware/docs/Software/Storage_Software/Sun_StorEdge_Network_Data_Replicator/sun_storedge_network_data_replicator_3_0_1/index.html

[Sydeco,2007] SYDECO SYSTEM DEVELOPEMENT COMPANY; Sydeco AVF
http://www.sydeco.com.br

[Taylor,2003] Taylor, Art; Buege, Brian e Randy, Layman Segurança contra Hackers J2EE e JAVA São Paulo: Futura, 2003

[UCLA,2007] UNIVERSITY OF CALIFORNIA – COMPUTER SCIENCE DEPARTMENT – LABORATORY FOR ADVANCED SYSTEM RESEARCH;
http://www.lasr.cs.ucla.edu
http://www.lasr.cs.ucla.edu/ficus/ficus_summary.html
http://www.lasr.cs.ucla.edu/travler/truffles_summary.html
http://www.lasr.cs.ucla.edu/roam98/NewRoamWelcome.html
http://www.lasr.cs.ucla.edu/rumor98/replication.html

[UML,2007] OBJECT MANAGEMENT GROUP - Unified Modeling Language
http://www.uml.org

[W3C,2007] WORLD WIDE WEB CONSORTIUM;
http://www.w3.org/DOM/#what
[Wolfson,1997] WOLFSON,OURI; JAJODIA,SUSHIL; HUANG,YIXIU; An Adaptative Data Replication Algorithm; ACM Transaction on Database Systems Vol.22, No2, June 1997, Pages 255-314.

[Xiangning,1998] XIANGNING,LIU; ABDELSALAM, HELAL; WEIMIN,DU; Multiview Access Protocols for Large Scale Replication; ACM Transactions on Database System, Vol.23, No 2, June 1998, Pages 158-198.

Apêndice A

Exemplo prático de uma replicação

Este exemplo enfoca os principais aspectos ligados à replicação. Observa-se que muitos paradigmas surgem naturalmente na solução dos problemas advindos dos processos de replicação.

Sejam duas réplicas mantidas em *sites* separados (Rc e Ra) com a informação de um saldo bancário(S) de um cliente nº 1 (C_1), assim simbolizados: Rc $\{C_1(S=500)\}$ e Ra$\{C_1(S=500)\}$ onde Rc é a cópia que fica na base de dados central e Ra a cópia que fica em uma agência.

Será explicitada aqui apenas a informação mais volátil (saldo) referente ao cliente C_1, pois outras informações mais estáveis como nome e conta corrente, por exemplo, não estão sujeitas a mudanças e por isso não são significativas para os procedimentos de conciliação. As movimentações de débito e crédito não são disponibilizadas, apenas o saldo é acessível.

Somente essa proposição, considerado inicialmente no exemplo, já traz implícita muitos conceitos. Como existe uma *base de dados central* e cópias nas agências e considerando que cada agência só receberá a cópia dos clientes que lhe pertencem, significa que o banco de dados passou por um processo de *fragmentação horizontal* (seção 2.5.1). Por outro lado, se os analistas do banco deduziram que apenas nome, conta corrente e saldo seriam suficientes para replicar o banco de dados, isso resultou em uma *fragmentação vertical* da base de dados. Ainda nesse exemplo, a base de dados central é a *cópia primária* dos dados replicados. Mas, como ambos os *sites* podem realizar atualizações nos dados, tem-se uma configuração do tipo *multimestre* (seção 2.7.2), ou seja, existe mais de um "mestre" com direito a atualizações e nenhum "escravo" (se existissem, possuiriam apenas uma cópia *read-only*).

A aplicação de interesse desse trabalho supõe que o BD central e o BD da agência permaneçam desconectados durante a maior parte do tempo, sendo esse uma parte intrínsica da solução. Assim, haverá uma *desconexão planejada* (seção 2.8.2) quando todas as *transações globais* (seção 2.2) são encerradas, admitindo-se apenas *transações locais*. Da mesma forma, em algum momento, tanto a agência quanto a central poderão tomar a iniciativa restabelecer a conexão, iniciando então o procedimento de *propagação de dados* (seção 2.7). Como existe a probabilidade dos dados terem sido alterados, pode haver a necessidade de se fazer a *sincronização* das bases de dados (seção 2.9). Como qualquer um dos *sites* pode tomar a iniciativa de estabelecer a conexão, a técnica de propagação dos dados será do tipo *híbrida* (seção 2.7.3).

Antes da primeira conciliação, pressupõe-se que ocorreu uma cópia do banco de dados, seguida de uma desconexão prolongada e um reconexão dos dois *sites*. A tabela A.1 mostra esses passos do ponto de vista de Rc. A célula cinza indica que Rc não tem (e nem terá) conhecimento dos eventos ocorridos em Ra.

Evento:	Rc	Ra
Cópia + desconexão	C_1(S=500)	C_1(S=500)
Atualizações	C_1(S=700) ---	C_1(S=300) C_1(S=500)
Reconexão + reconciliação	C_1(S=700)	C_1(S=500)
Regra de conciliação	O valor do servidor central está sempre correto	
Dado conciliado	C_1(S=700)	C_1(S=700)

Tabela A.1 – Dados replicados e conciliados com base no valor existente no servidor central.

Percebe-se claramente que no momento da reconexão não se pode tirar nenhum tipo de conclusão a respeito do valor do dado final (célula em destaque). Qual valor será o correto, 700 ou 500 ? O valor 500 é o mesmo enviado anteriormente ou existiram alterações ? A regra estabelecida deixa claro que Ra serve, então, apenas para consultas. O servidor central é o único que pode alterar valores, logo em Ra jamais deverá ocorrer eventos de atualização e, caso aconteça, é porque ocorreram erros. Na seção 3 existem exemplos bem conhecidos de bancos de dados que utilizam essa estratégia.

Entretanto, antes de se executar a conciliação propriamente dita, deve-se proceder à *identificação* do que precisa ser reconciliado. No exemplo anterior isto foi feito pela comparação direta entre valores do item de dado. Para que isto seja possível, toda a base de dados tem que ser transferida entre os *sites*, configurando assim um sincronismo por *transferência de estado* (seção 2.7.4.2). Acontece que, em casos de bases de dados maiores, essa é notadamente uma estratégia ineficiente. Para aumentar a eficiência pode-se usar algoritmos otimizados de comparação, como o *algoritimo rsync* (seção 2.7.4.2). Outra opção, que evita a comparação dado a dado, é transferir apenas as informações inseridas, eliminadas ou alteradas. Para isso pode-se usar o *algoritmo de bit modificado* (seção 2.7.4.2), uma solução simples que inclui um indicador ligado a um conjunto de dados, assinalando-os quando esses forem alterados. Na tabela A.2, a questão da conciliação é retomada, ilustrando-se a aplicação da regra de conciliação do *bit modificado*, onde b=1 indica que a informação foi atualizada.

Evento:	Rc	Ra
Cópia + desconexão	... C_1(S=500; b=0) ... C_2 ... ... C_3(S=800; b=0) ... C_4 ... ... C_7(S=300; b=0) ... C_n ...	C_1(S=500; b=0) ... C_2 ... C_3(S=800; b=0) ... C_4 ... C_7(S=300; b=0) ... C_n ...
Atualizações	... C_1(S=350; **b=1**) ... C_2 ... ... C_3(S=800; b=0) ... C_4 ... ... C_7(S=200; **b=1**) ... C_n ...	C_1(S=500; b=0) ... C_2 ... C_3(S=950; **b=1**) ... C_4 ... C_7(S=500; **b=1**) ... C_n ...
Reconexão + reconciliação	C_1(S=350; **b=1**) C_3(S=700; b=0) C_7(S=200; **b=1**)	C_1(S=500; b=0) C_3(S=950; **b=1**) C_7(S=500; **b=1**)
Regra de conciliação	Considerar a réplica alterada como a correta. Se houver alteração em ambas, aplicar a regra comutativa: ($V_{Rc} + V_{Ra} - V_{Rc\ anterior}$).	
Dado conciliado	C_1(S=350; b=0) C_3(S=950; b=0) C_7(S=400; b=0)	C_1(S=350; b=0) C_3(S=950; b=0) C_7(S=400; b=0)

Tabela A.2 – Uso do *bit modificado* e a regra de conciliação comutativa.

Na tabela A.3, a novidade fica por conta da "regra comutativa". Ela só funciona em situações muito específicas e envolvendo dados numéricos. Esse não é o caso geral, pois "conciliação" aplica-se a uma grande diversidade de informações, tais como: arquivos tipo texto, arquivos

html, diretórios, gráficos, imagens e outros. Por isso, as regras de conciliação são muito dependentes do tipo de aplicação.

A simplicidade do último exemplo esconde problemas sérios. Caso seja debitada uma taxa de 1% do saldo quando esse ficar negativo, a seqüência a seguir pode não ser a correta (para simplificação, é mostrada apenas uma "conta" sem o bit indicador):

Evento:	**Rc**	**Ra**
Cópia + desconexão	C_1(S=200)	C_1(S=200)
Atualizações	C_1(S=*–300*) C1(S=*–303*) (taxa de 1%)	C_1(S=400) C_1(S=900)
Reonexão + reconciliação	C_1(S=*–303*)	C_1(S=900)
Regra de conciliação	Considerar a réplica alterada como a correta. Se houver alteração em ambas, aplicar a regra comutativa: ($V_{Rc} + V_{Ra} – V_{Rc\ anterior}$).	
Dado conciliado	C_1(S=397)	C_1(S=397)

Tabela A.3 – Falsos resultados com a aplicação da regra comutativa, devido ao desconhecimento da ordem das operações.

Como é fácil perceber, devido ao crescimento do saldo em Ra, qual é a garantia de que o saldo em Rc de fato ficou negativo ? Ou seja, é necessário que a ordem das atualizações seja explícita, conforme o momento real em que elas aconteceram. Fazer isso não é difícil usando o conceito de *timestamp* (seção 2.6.3) ou *vetor clocks* (seção 2.7.4.1). A tabela A.4 ilustra o uso de *timestamp* com tempo real na tabela anterior.

Evento:	**Rc**	**Ra**
Cópia + desconexão	C_1(S=200,ts=0:00)	C_1(S=200,ts=0:00)
Atualizações	C_1(S=*–300*,ts=1:50) C_1(S=*–303*,ts=1:51)	C_1(S=400,ts=1:30) C_1(S=900,ts=2:20)
Reconexão + reconciliação	C_1(S=200,ts=0:00) C_1(S=*–300*,ts=1:50) C_1(S=*–303*,ts=1:51) C_1(S=*–303*,ts=1:51)	C_1(S=400,ts=1:30) C_1(S=400,ts=1:30) C_1(S=400,ts=1:30) C_1(S=900,ts=2:20)
Regra de conciliação	Considerar a réplica alterada como a correta. Se houver alteração em ambas, aplicar a regra comutativa: ($V_{Rc} + V_{Ra} – V_{Rc\ anterior}$).	
Dado conciliado	C_1(S=400,ts=1:30) C_1(S=*–100*,ts=1:50) C_1(S=*–103*,ts=1:51) C_1(S=397,ts=2:20)	C_1(S=400,ts=1:30) C_1(S=*–100*,ts=1:50) C_1(S=*–103*,ts=1:51) C_1(S=397,ts=2:20)

Tabela A.4 – Uso da técnica do *timestamp* com resultado final falso devido a cálculo com regra comutativa (as setas pontilhadas assinalam um dos cálculos com regra comutativa: -300 + 400 – 200 = –100).

Observa-se que o erro continua existindo, pois a taxa de 1% não pode ser tratada de forma comutativa. Em ts=1:51, o cálculo correto da taxa deixaria o saldo em S= *–101*. A solução para isto poderia ser acrescentar às regras algo do tipo: "Calcular taxa somente após a conciliação". Mas isso só contornaria o problema, pois em outras situações pode não se chegar ao resultado esperado. Conclui-se que apenas registrar o *timestamp* é pouco. Em casos como esse se exige um bom *controle de divergências* (seção 2.9) e de *propagação de dados* (seção 2.7).

Todas essas questões mostram o quanto a abordagem adotada no exemplo é ineficaz em muitas situações. A causa disso se deve ao fato da conciliação estar sendo feita com o uso de uma técnica conhecida como *transferência de estado* (seção 2.7.4.2), ou seja, compara-se o

"estado" das informações permitindo detectar as diferenças, mas não o "porquê" das diferenças. Por conseqüência, a única informação disponível para elaborar os algoritmos de conciliação são os próprios dados e nada mais. Mesmo sofisticando as regras de conciliação, aumenta-se a complexidade do processamento das informações, sem garantias de sucesso, pois sempre estará faltando a informação do que aconteceu de fato com o dado. Por esse motivo, utiliza-se a técnica de disponibilizar todas as *operações*[26] realizadas nos dados, o que dá origem à dois tipos de conciliação: a *sintática* e a *semântica* (seção 2.8.2).

Na tabela A.4 onde, com base na tabela A.4, introduz-se a conciliação *sintática* utilizando as transações (T_n). A regra de conciliação usada é a do *escalonamento sintático* (seção 2.8.1.2), que é uma técnica que explora a ordem cronológica das operações tentando reproduzir os "acontecimentos", enquanto as bases estavam desconectadas.

Evento:	**Rc**	**Ra**
Cópia + desconexão	C_1(S=200) ($T_{0c=}$update, ts=0:00)	C_1(S=200) ($T_{0a=}$update, ts=0:00)
Atualizações	C_1(S=*–300*), T_{1c}(S=S-500,ts=1:50) C_1(S=*–303*), T_{2c}(S=S+0.01*S,ts=1:51) $_{S<0}$	C_1(S=400), T_{1a}(S=S+200,ts=1:30) C_1(S=900), T_{2a}(S=S+500,ts=2:20)
Reconexão + reconciliação	C_1(S=200) (ts=0:00) T_{1a}(S=S+200, ts=1:30) ← T_{1c}(S=S – 500, ts=1:50) → T_{2c}(S=S+0.01*S, ts=1:51) $_{S<0}$ → T_{2a}(S=S+500, ts=2:20) ←	C_1(S=200) (ts=0:00)] T_{1a}(S=S+200, ts=1:30) T_{1c}(S=S – 500, ts=1:50) T_{2c}(S=S+0.01*S, ts=1:51) $_{S<0}$ T_{2a}(S=S+500, ts=2:20)
Regra de conciliação	Aplicar as operações utilizando o escalonamento sintático	
Dado conciliado	C_1(S=400) ($T_{c=}$200+200,up; ts=0:00) C_1(S=*–100*) ($T_{c=}$400-500,up; ts=0:00) C_1(S=*–101*) ($T_{c=}$-100*1.01,up; ts=0:00) C_1(S=399) ($T_{c=}$-101+500,up; ts=0:00)	C_1(S=400) ($T_{a=}$200+200,up; ts=0:00) C_1(S=*–100*) ($T_{a=}$400-500,up; ts=0:00) C_1(S=*–101*) ($T_{a=}$-100*1.01,up; ts=0:00) C_1(S=399) ($T_{a=}$-101+500,up; ts=0:00)

Tabela A.5 – Conciliação *centrada na transação* com regra baseada no *escalonamento sintático*.

Como a técnica do *escalonamento sintático* pode gerar conflitos desnecessários, na tabela A.6 enfoca-se o *escalonamento semântico* (seção 2.8.1.3), um critério mais "inteligente" que o anteriormente citado.

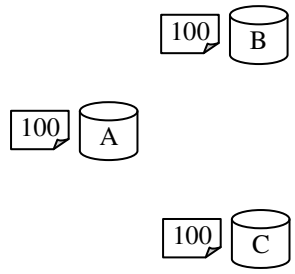
Evento:	**Rc**	**Ra**
Cópia + desconexão	C_1(S=200) ($T_{0c=}$update, ts=0:00)	C_1(S=200) ($T_{0a=}$update, ts=0:00)
Atualizações	C_1(S=*–300*), T_{1c}(S=S-500,ts=1:50) C_1(S=*–303*), T_{2c}(S=S+0.01*S,ts=1:51) $_{S<0}$	C_1(S=400), T_{1a}(S=S+200,ts=1:30) C_1(S=900), T_{2a}(S=S+500,ts=2:20)
Reconexão + reconciliação	C_1(S=200) (ts=0:00) T_{1a}(S=S+200,ts=1:30) ← T_{2a}(S=S+500,ts=2:20) ← T_{1c}(S=S-500,ts=1:50) → T_{2c}(S=S+0.01*S) $_{S<0}$ →	C_1(S=200) (ts=0:00) T_{1a}(S=S+200,ts=1:30) T_{2a}(S=S+500,ts=2:20) T_{1c}(S=S-500,ts=1:50) T_{2c}(S=S+0.01*S) $_{S<0}$
Regra de conciliação	Aplicar primeiro as operações de crédito, depois as de débito e as condicionais por último após fechar o período de 24:00	
Dado conciliado	C_1(S=400) ($T_{c=}$200+200,up; ts=0:00) C_1(S=900) ($T_{c=}$400+500,up; ts=0:00) C_1(S=400) ($T_{c=}$900-500,up; ts=0:00) C_1(S=400) $_{S>0}$	C_1(S=400) ($T_{a=}$200+200,up; ts=0:00) C_1(S=900) ($T_{a=}$400+500,up; ts=0:00) C_1(S=400) ($T_{a=}$900-500,up; ts=0:00) C_1(S=400) $_{S>0}$

Tabela A.6 – Conciliação com regra *semântica* baseada no *escalonamento semântico*.

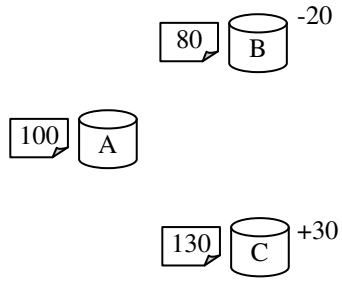
[26] A respeito dos termos *operação* e *transação*: muitos autores utilizam o conceito de *transação* sem entrar no mérito de que as atualizações nos bancos de dados podem ocorrer de forma síncrona ou assíncrona. Já Saito [Saito,2005-p45] é mais específico: quando as atualizações ocorrem de forma *assíncrona*, ele denomina todos os procedimentos envolvidos nesse processo de *operação* (ver seção 2.4.1).

Existem vários tipos de *escalonamento semântico.* Um dos tipos impõe certas restrições para evitar um conflito antes que ele ocorra. Por exemplo: supor que o "saldo" *jamais* deverá ficar negativo e, levando em conta que a distribuição do dado ocorre em dois *sites*, faz-se a uma regra de aplicação onde "qualquer saque não pode ultrapassar a metade do saldo inicial disponível, enquanto a base de dados não for conciliada". Caso existam saques em ambas as bases enquanto elas estiverem desconectadas, a regra garante que não haverá conflitos posteriores.

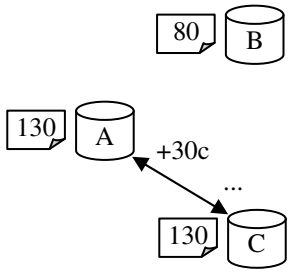
Para finalizar a ilustração dos conceitos envolvidos na replicação de banco de dados, no próximo exemplo é acrescentado um terceiro *site*. Assim, sejam *A*, *B* e *C sites* com direito a atualizações e com o mesmo conjunto de dados replicado. Apresenta-se a seguir uma seqüência de atualizações de um determinado valor:



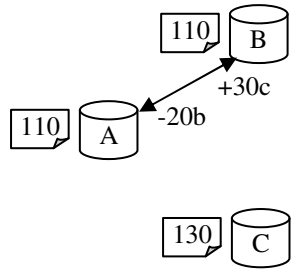
(1) *A*,*B* e *C* estão sincronizados



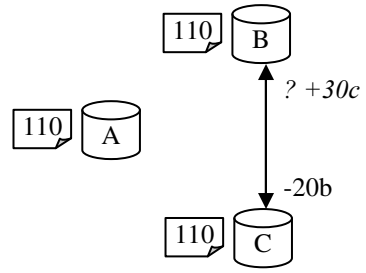
(2) *A*,*B* e *C* divergiram



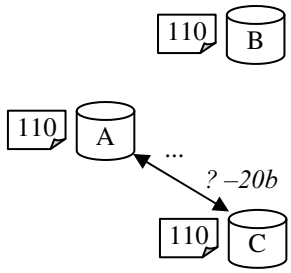
(3) *A* e *C* se sincronizam.



(4) *A* e *B* se sincronizam. *A* passou para *B* a operação que recebeu de *C*. Outra estratégia seria deixar que *C* o fizesse.



(5) *B* e *C* se sincronizam. *B* reconhece que já recebeu a operação +30 ocorrida em *C* e a desconsidera



(6) *A* envia para *C* a operação que recebeu de *B*. *C* reconhece que já recebeu a operação –20 ocorrida em *B* e a desconsidera. Agora os bancos *convergiram* e estão sincronizados

Figura I.1 – Seqüência de eventos em uma base de dados distribuída que levam à convergência das informações

Nesse exemplo pode-se perceber vários detalhes que devem ser levados em conta quando se trata de sincronizar bases replicadas. As operações, por exemplo, tem que ser identificadas; cada *site* tem que controlar as operações que devem ser aplicadas (outra alternativa seria apenas um *site* atualizar e repassar o resultado aos demais); quanto ao controle de propagação das atualizações, isso pode ser feito por um *site* central ou acontecer de forma *epidêmica* (seção 2.6.1); não se entrou no mérito de *quando* deve ser feito a conciliação, que é outro aspecto que influencia muito as características de qualquer solução.

A exposição dos dois exemplos mostra que são muitos os fatores que podem simplificar ou tornar complexa uma base de dados replicada e distribuída. Há fortes dependências entre a solução, o tipo de aplicação e a arquitetura do sistema distribuído. Nas próximas seções são mais detalhados os principais conceitos mencionados superficialmente nesses exemplos.

Apêndice B

Configurações do ambiente de testes do modelo RPCD

Configuração do modelo global[27]:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Modelo global afvendas - Estas informações serão utilizadas pela classe GlobalModel -->
<root>

    <!-- nome global do banco de dados - enabled="true" significa habilitado para replicação -->
    <model id="afvendas" enabled="true">

        <!-- tabelas do modelo global -->
        <tables>

            <!-- tabela cliente - configurado com a declaração CREATE TABLE do SQL -->
            <cliente>
                CREATE TABLE cliente (
                idcliente char(5),
                fisicojuridico char(1),
                cnpj_cpf varchar(14),
                nomecliente varchar(50),
                CONSTRAINT cliente_pkey PRIMARY KEY (idcliente)
                )
            </cliente>

            <!-- tabela produto - Aconselhável que esta tabela e a de cliente sejam declaradas antes das
                 que se seguem por causa das chaves estrangeiras. -->
            <produto>
                CREATE TABLE produto (
                idproduto char(5),
                descricao varchar(50),
                valorunitario numeric(12,2),
                saldoestoque integer,
                CONSTRAINT produto_pkey PRIMARY KEY (idproduto)
                )
            </produto>

            <!-- tabela pedido -->
            <pedido>
                CREATE TABLE pedido (
                siteorigem char(2),
                idcliente char(5),
                idpedido char(5),
                datapedido date,
                CONSTRAINT pedido_pkey PRIMARY KEY (siteorigem, idcliente, idpedido),
                CONSTRAINT "$1" FOREIGN KEY (idcliente) REFERENCES cliente (idcliente)
                ON UPDATE RESTRICT ON DELETE RESTRICT
                )
            </pedido>

            <!-- tabela itemproduto -->
            <itemproduto>
                CREATE TABLE itemproduto (
                siteorigem char(2),
                idcliente char(5),
                idpedido char(5),
                idproduto char(5),
                quantidadepedida integer,
                confirmadoentrega boolean,
                CONSTRAINT itemproduto_pkey PRIMARY KEY (idpedido, idproduto),
                CONSTRAINT "$2" FOREIGN KEY (idproduto) REFERENCES produto (idproduto)
                    ON UPDATE RESTRICT ON DELETE RESTRICT,
                CONSTRAINT "$3" FOREIGN KEY (siteorigem,idcliente,idpedido)
                    REFERENCES pedido (siteorigem,idcliente,idpedido)
                    ON UPDATE RESTRICT ON DELETE RESTRICT )
            </itemproduto>
        </tables>

        <!-- indica as precedências das tabelas - utilizado no escalonamento -->
        <precedence>
            <produto/>
            <cliente>
                <pedido>
                    <itemproduto/>
                </pedido>
            </cliente>
        </precedence>
    </model>
</root>
```

[27] O tipo ***negrito e itálico*** não faz parte da sintaxe do *Xml*. Foi utilizado apenas para dar maior clareza.

Configuração da base local - Site Alecrim

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Alecrim - base de dados DbaseIII Plus - atualizado pelo módulo de replicação utilizando drives Hxtt --
>
<root>
    <!-- modelo global com o qual se relaciona - enabled=true significa a base de dados está habilitada -->
    <model id="afvendas" enabled="true"/>

    <!-- localização do Database no site local - é acessado pelo Hxtt e monitorado pelo arquivo pipe
fclipper.txt -->
    <db dataBaseName="//P/Profis/afvendas" port="8029" pipe="fclipper.txt">
        <!-- driver utilizado pelo RPCD/Hxtt para acesso direto ao banco de dados -->
        <driver name="com.hxtt.sql.dbf.DBFDriver" user="" password=""/>
        <!-- Acesso com java para à base replicada pelo módulo de replicação -->
        <url name="jdbc:dbf://localhost:8029" otherParameters=""/>
    </db>

    <!-- tabelas do banco de dados existentes neste site -->
    <tables>
        <!-- tabela cliente -->
        <cliente localname='cliente' prefix='"alecrim/".'>
            <!-- Mapeamento dos campos globais X campos locais -->
            <field global="idcliente" local="Codigo"/>
            <!-- Mapeamento não é direto. É realizado com uma função que converte os códigos {F|J} para
{0|1} -->
            <field global="fisicojuridico" local="Tipo"
function="br.eti.vinicius.rpcdemo.alecrim$ClienteTipo"/>
            <field global="cnpj_cpf" local="Cnpj"/>
            <field global="nomecliente" local="Nome"/>
            <!-- Necessário declarar os campos locais que não são mapeados - este recurso também pode ser
                 usado para fragmentação vertical -->
            <field local="telefone"/>
        </cliente>

        <!-- tabela pedido - pode-se declarar no mapeamento apenas os campos que configuram exceções -->
        <pedido localname='pedido' prefix='"alecrim/".'>
            <!-- Declara-se o campo global que não existe localmente e qual valor deve assumir no modelo
global -->
            <field global="siteorigem" value="01"/>
        </pedido>

        <!-- tabela itemproduto - como é igual ao modelo global, não precisa ser declarado os mapeamentos -
->
        <itemproduto localname='itemprod' prefix='"alecrim/".'/>

        <!-- tabela produto - como é igual ao modelo global, não precisa ser declarado os mapeamentos -->
        <produto localname='produto' prefix='"alecrim/".'/>
    </tables>
</root>
```

Configuração da base local - Site Tirol

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Tirol - base de dados Hsqldb -->
<root>
    <!-- modelo global com o qual se relaciona  - enabled=true significa a base de dados está habilitada --
>
    <model id="afvendas" enabled="true"/>

    <!-- localização do Database no site local - monitorado e atualizado de forma ambientada -->
    <db dataBaseName="/P/Profis/afvendas/tirol" port="9001">
        <!-- driver utilizado pelo RPCD para acesso direto ao banco de dados -->
        <driver name="org.hsqldb.jdbcDriver" user="sa" password=""/>
        <!-- Observe que a base replicada será acessada de forma ambientada (engine) -
             otherParameters="file:memory:" indica a modalidade da ambientação: file é uma característica
             do Hsqldb e memory uma característica da implementação do RPCD -->
        <url name="jdbc:hsqldb:" otherParameters="file:memory:"/>
    </db>


    <!-- tabelas do banco de dados existentes neste site -->
    <tables>
        <!—tabela cliente - parâmetro create="yes" significa que se não existir a tabela, pode ser criada
no site -->
        <cliente create="yes">
            <!-- Declaração dos campos locais que não são mapeados - não serão criados caso não exista a
tabela -->
            <field local="positivoSerasa"/>
        </cliente>
        <!-- tabela produto -->
        <produto create="yes"/>

        <!-- tabela produto -->
        <pedido create="yes">
            <!-- Declara-se o campo global que não existe localmente e qual valor deve assumir -->
            <field global="siteorigem" value="02"/>
        </pedido>

        <!-- tabela itemproduto -->
        <itemproduto create="yes"/>
    </tables>
</root>
```

Configuração da base local - Site Matriz

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Matriz - base de dados PostgreSQL -->
<root>
    <!-- modelo global com o qual se relaciona  - enabled=true significa a base de dados está habilitada --
>
    <model id="afvendas" enabled="true"/>

    <!-- localização do Database no site local - monitorado pela porta 5432 e atualizado pelo módulo
replicador -->
    <db dataBaseName="/AFVENDAS" port="5432">
        <!-- driver utilizado pelo RPCD para acesso direto ao banco de dados -->
        <driver name="org.postgresql.Driver" user="postgres" password="jupiter"/>
        <!— A base replicada pode estar em qualquer host e não somente em localhost, mas o aplicativo deve
            acessar localhost (jdbc:postgresql://localhost:5432) que o módulo de replicação redireciona Se
            ambos estiverem em localhost, deve-se diferenciar pelas portas (atender na 5431, por exemplo, e
            redirecionar para 5432 -->
        <url name="jdbc:postgresql://192.168.0.7:5432" otherParameters=""/>
    </db>
    <!-- tabelas do banco de dados neste site - como são iguais ao modelo global, não necessitam de
mapeamentos -->
    <tables>
        <cliente/>
        <pedido/>
        <itemproduto/>
        <produto/>
    </tables>
</root>
```

Configuração da base remota - Site Tirol (lado cliente)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Site Tirol - configuração no lado do cliente -->
<root>
    <!-- Parâmetros: enabled="true" significa habilitado para comunicação; login= e passwd= são
         transmitidas pelo cliente e conferidas pelo servidor; server= e port= URL/porta ou IP/porta do
         servidor que o cliente deve acessar -->
    <site enabled="true" login="tirol" passwd="123456" server="192.168.0.2" port="8000"/>

    <!-- modelo global com o qual se relaciona -->
    <tables model="afvendas">

        <!-- tabela cliente replicada sem fragmentação -->
        <cliente/>

        <!-- tabela pedido replicada com fragmentação horizontal - utiliza-se a sintaxe do SQL na tag where
-->
        <pedido>
            <where>
                siteorigem='01'
            </where>
        </pedido>

        <!-- tabela itemproduto replicada com fragmentação horizontal -->
        <itemproduto>
            <!-- Para fragmentar tabelas com relação 1-n utilizou-se o parâmetro <campo> in (<relação de
                 valores>) da cláusula where do SQL - na configuração, "/pedido/where" indica a tag onde se
                 encontra a expressão where do Select que será utilizado -->
            <where>
                idpedido in (Select idpedido from pedido where /pedido/where)
            </where>
        </itemproduto>
        <!-- tabela produto replicada sem fragmentação -->
        <produto/>
    </tables>
</root>
```

Configuração da base remota -Site Tirol (lado servidor)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Site Tirol - configuração no lado do servidor -->
<root>
    <!- Parâmetros: enabled="true" significa habilitado para comunicação, login= e passwd= são transmitidas
         pelo cliente e conferidas pelo servidor, client= indica ao servidor qual host pode ter acesso. Se o
         valor for "any" qualquer host pode acessar. -->
    <site enabled="true" login="tirol" passwd="123456" client="192.168.0.2"/>
                                                    ...
                          (todo o restante é idêntico ao Site Tirol lado cliente)
```

Configuração da base remota -Site Alecrim (lado cliente)

Poderia ser igual à configuração do *Site Tirol (lado cliente)*, exceto pelo *login* que deve ser diferente. Mas para fins de teste e exemplo, adotou-se uma configuração simplificada que produz o mesmo efeito final:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Site Alecrim - configuração no lado do cliente -->
<root>
    <!- A única diferença obrigatória entre configurações de sites é no parâmetro login=, que deve ser
         único para cada site que participa do mesmo ambiente de replicação -->
    <site enabled="false" login="alecrim" passwd="123456" server="localhost" port="8000"/>
    <!-- modelo global com o qual se relaciona -->
    <tables model="afvendas">
        <!-- Tabelas replicadas sem fragmentação - como é o lado cliente, pode transmitir todo o seu
             conteúdo ao servidor sem restrições -->
        <cliente/>
        <pedido/>
        <itemproduto/>
        <produto/>
    </tables>
</root>
```

Configuração da base remota - Site Alecrim (lado servidor)

Idêntico ao *Site Tirol (lado servidor)*, exceto pelos parâmetros *login= e client=:*

```
. . .
<site enabled="true" login="alecrim" passwd="123456" client="192.168.0.3"/>
. . .
```

Monitoração da base de dados

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Monitoração das base de dados -->
<root>

    <!-- Porta inicial para controle gerencial das classes monitoras - para cada instância é atribuída uma
         porta diferente -->
    <monitor initialPort="8001">

        <!-- interpretadores de protocolos de banco de dados -->
        <interpreters>

            <!-- porta para atender os aplicativos que utilizam PostgreSQL -->
            <portUser id="5432">
                <!-- URL da base de dados monitorada -->
                <urlDB url="192.168.0.7"/>
                <!-- Porta da base de dados monitorada -->
                <portDB port="5432"/>
                <!-- parâmetro apenas informativo -->
                <dbname name="PostgreSql"/>
                <!-- classe que interpreta o protocolo front-end/back-end -->
                <protocolInterpreter
name="br.eti.vinicius.rpcd.monitor.GenericProtocolInterpreter$GPI_Postgresql"/>
                <!-- habilitado ou não para ser monitorado -->
                <enabled value="true"/>
                <!-- se ativo e que hora ficou ativo - é informado em tempo de execução quando da ativação
                     do objeto que representa esta configuração -->
                <activatedIn dh="0" dhf=" "/>
            </portUser>

            <!-- porta para atender os aplicativos que utilizam Hxtt -->
            <portUser id="8029">
                <urlDB url="localhost"/>
                <portDB port="8029"/>
                <dbname name="Hxtt"/>
                <protocolInterpreter
name="br.eti.vinicius.rpcd.monitor.GenericProtocolInterpreter$GPI_Hxtt"/>
                <enabled value="false"/>
                <activatedIn dh="0" dhf=" "/>
            </portUser>
        </interpreters>

        <!-- monitores de tabelas relacionais -->
        <pipes>

            <!-- arquivo comum entre às aplicações legadas e a monitoração - para cada pasta contendo
                 tabelas relacionais, utiliza-se um arquivo pipe -->
            <pathPipeFile id="fclipper.txt">
                <!-- classe que converte a semântica da aplicação legada à semântica do SQL -->
                <operationInterpreter name="br.eti.vinicius.rpcd.monitor.ClipperTableMonitor"/>
                <!-- habilitado ou não para ser monitorado -->
                <enabled value="true"/>
                <!-- pasta onde encontram-se as tebelas relacionais - utilizada pelo replicador para
                     atualizar estas tabelas - indica também o caminho absoluto do arquivo pipe:
                     path+"/"+pathPipeFile -->
                <pathTables path="/P/Profis/afvendas/alecrim"/>
                 <!-- se ativo e que hora ficou ativo -->
                <activatedIn dh="0" dhf=" "/>
            </pathPipeFile>
        </pipes>

        <!-- Para uso do RpcdDriver quando atende a diversas aplicações com replicação ambientada no mesmo
host -->
        <rpcdDriver>
            <!-- porta que utiliza para atender as aplicações com o replicador ambientado -->
            <portUser id="8100">
                <!-- url onde atende - normalmente é localhost, eventualmente pode estar em outro host -->
                <urlmonitor url="localhost"/>
                <!-- habilitado ou não para atender -->
                <enabled value="true"/>
                <!-- porta de controle atribuída à instância desta classe -->
                <portMonitorControl value="0"/>
                <!-- se ativo e que hora ficou ativo -->
                <activatedIn dh="0" dhf=" "/>
            </portUser>
        </rpcdDriver>
    </monitor>
</root>
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo